2003 pingwinaria

Polska Grupa Użytkowników Linuxa PLUG
www.linux.org.pl

Komitet Organizacyjny:
Andrzej Boczek
Agnieszka Bohosiewicz
Krzysztof Leszczyński
Ewa Marczyńska
Kondrad Wawruch

Redakcja referatów:
Maja Królikowska
Krzysztof Leszczyński

Grafika:
Tomasz Drążek (okładka i pingwinki)
Antoni Goldstein (mapka)

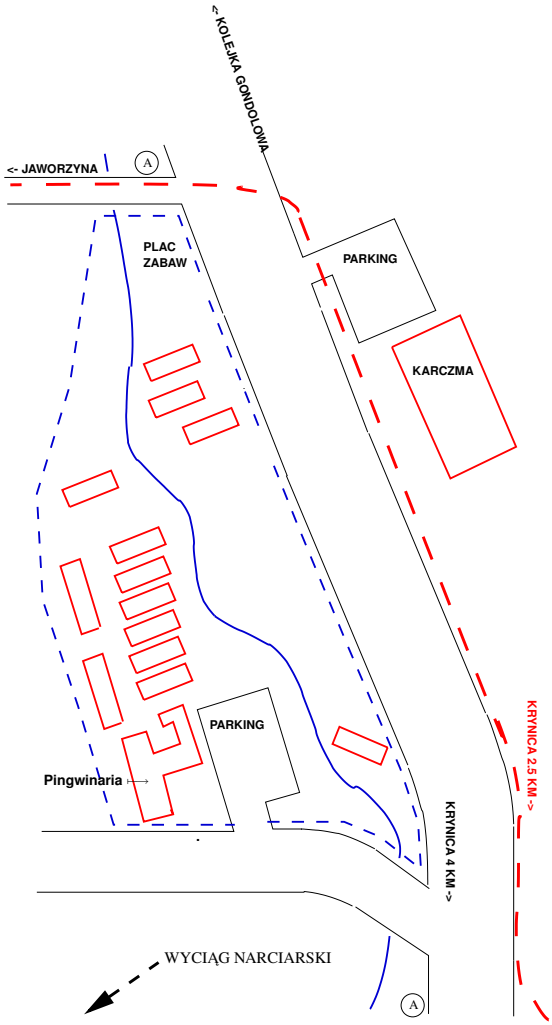
Projekt ligatury Q-wiośnie:
Andrzej Tomaszewski

KaOwcy:
Dawid Kuroczko
Szymon Błaszczyk

Sponsorzy:
Borland Software Corporation
7bulls.com sp. z o. o.

Patron medialny:
Chip

Bardzo nam pomogły następujące osoby (kolejność alfabetyczna):
Hugh Blemings,
Piotr Bolek,
Tomasz Drążek,
Antoni Goldstein,
Bogusław Jackowski,
Paweł Kot,
Piotr Strzelczyk,
Jan Sabak



Mapka okolic hotelu „Czarny Potok”

Kolofon

Materiały zostały przygotowane z wykorzystaniem wyłącznie Wolnego Oprogramowania.

Wszystkie prace związane z przygotowaniem publikacji do druku zostały wykonane wyłącznie w systemie Linux. Redakcja, w zależności od płci używała dystrybucji Slackware lub RedHat.

Teksty otrzymane od autorów opracowaliśmy używając edytora `XEmacs`. Skład został wykonany w systemie TEX z wykorzystaniem opracowanego na potrzeby konferencji, środowiska redakcyjnego, opartego na formacie ConTEXt opracowanego przez Hansa Hagena i jego kolegów z firmy Pragma ADE, Hasselt, Holandia.

Mapkę okolic hotelu narysował Antoni Goldstein. Mapka została następnie dostosowana do celów wydawniczych programem MetaPost.

Teksty referatów złożono krojem QuasiPalladio, opracowanym przez Bogusława Jackowskiego, Janusza Nowackiego i Piotra Strzelczyka na podstawie fontu Palladio, opracowanego przez Hermanna Zapfa i udostępnionego przez firmę URW++. Tytuły referatów złożono krojem Antykwa Toruńska, opracowanym przez Janusza Nowackiego na podstawie rysunków Zygfryda Gardzielewskiego.

„Myślę sobie, że ta zima kiedyś musi minąć
zazieleni się, urośnie kilka drzew . . . ”

Po długiej, ciężkiej zimie idzie wiosna. Musi iść, tak jak na jesień, nie ma na nią rady. Kiedyś pewnie do nas dojdzie. Nie musimy jednak pozostawać bierni, więc poderwijmy się z krzeseł, foteli, kanap i podłogi i ruszmy. Q wiośnie!

Najbardziej trywialna z prawd głosi, że ludzie są różni i że te różnice podobno sprawiają, że nasz świat wygląda tak, jak wygląda. Niektórzy cieszą się z wiosny zdejmując czapki i kożuchy, wystawiając twarze do słońca, szukając oznak kwitnienia niektórych roślin, czy przyglądając się rozwijającym się listkom. Gdzieś w świecie istnieją także barbarzyńskie ludy, które czczą nadejście tej pory roku topiąc lub paląc kukłę, którą nazywają Marzanną. My jednak postanowiliśmy przywitać wiosnę po swojemu, organizując (tak, to już czwarte) Pingwinaria, czyli zlot miłośników, użytkowników i twórców Wolnego Oprogramowana, w szczególności tego, którego symbolem jest pingwin Tux.

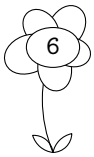
Autorem ligatury Q-wiośnie jest Andrzej Tomaszewski, typograf. Znak składa się z łacińskiej litery „*Q*” i chińskiego ideogramu oznaczającego wiosnę. Chińczycy są jednym z ostatnich ludów uprawiających staranną kaligrafię. Zwyczaj wypracowywania nowych, okolicznościowych form ideogramów jest w chińskiej tradycji mocno zakorzeniony. Z pewnością powitanie wiosny jest dobrą okazją.

W metaforycznym marszu Q wiośnie będziemy mieli okazję wysłuchać ponad dwudziestu referatów o tematyce związanej z systemem Linux, zakresem obejmujących budowę systemu, aplikacje narzędziowe oraz ogólnego przeznaczenia jak i nieodłącznie związane z Wolnym Oprogramowaniem zagadnienia prawne. Będzie także można uczestniczyć w zajęciach towarzyszących, tematycznych jak i zupełnie nie związanych z tematem naszego zlotu. W tym roku organizacją życia towarzyskiego, w czasie naszego czterodniowego *Rejsu*, zajmie się aż dwóch KaOwców. Niech nam żyje nowa tradycja. Skoro jesteśmy przy tradycji. . .

Kolejne Pingwinaria organizowane są w tym samym miejscu – w ośrodku „Czarny Potok”. Jest to spowodowane tym, że poprzednio warunki bytowe oceniliśmy jako bardzo dobre. Nie wiemy jeszcze czy miejsce to pozostanie stałym miejscem zlotów, czy Pingwinaria będą raczej pielgrzymować po kraju. Chwilowo osiadły w Krynicy. Następne mogą też być tutaj lub 100 metrów od twojego domu, jeśli zechcesz nam pomóc. Wszystko może zależeć od Ciebie.

Program Pingwinariów – środa, 26 marca 2003

18^{00} – ∞^{∞} PINGWINARIA TEAM
Prepingwinaria 2003



Program Pingwinariów – czwartek, 27 marca 2003

9^{00} – 10^{00} **– śniadanie–**

10^{00} – 11^{00} Łukasz Jachowicz
RWO-ISOC

11^{00} – 12^{00} Krzysztof Leszczyński
YAsiP – Yet Another /sbin/init Project 0B

12^{00} – 13^{00} Dawid Kuroczko
Quality of Service. Sterowanie przepływem danych w Linuksie

13^{00} – 14^{00} **– obiad–**

15^{45} – 16^{00} organizatorzy
Otwarcie oficjalne

16^{00} – 17^{00} Michał Melewski
Systemy wykrywania włamań (IDS) 14

17^{00} – 18^{00} Krzysztof Zaraska
Prelude Hybrid IDS: current state and development A8

18^{00} – 19^{00} Grzegorz B. Prokopski
SELinux 2F

19^{00} – 20^{00} **– kolacja–**

20^{30} – ∞^{∞} KaOwcy
Powinna tu być jakaś impreza

Program Pingwinariów – piątek, 28 marca 2003

Program Pingwinariów – sobota, 29 marca 2003

9^{00} – 10^{00} **– śniadanie–**

10^{00} – 11^{00} Grzegorz Borowiak, Wojciech Piechowski
Zastosowanie Linuksa do przetwarzania dźwięku w czasie rzeczywistym 75

11^{00} – 12^{00} Marcin Kwadrans
Kompresja video pod Linuksem 7D

12^{00} – 13^{00} Paweł Kaczor, Łukasz Dyś
Sprzętowy odtwarzacz multimedialny z rupieci

13^{00} – 15^{00} **– obiad–**

15^{00} – 16^{00} Jacek Prucia
Testowanie aplikacji www programem flood 83

16^{00} – 17^{00} Łukasz Rżanek
Wizualne technologie tworzenia aplikacji internetowych B8

17^{00} – 18^{00} Krzysztof Krawczyk
IEEE 1394, czyli praktyczne wykorzystanie urządzeń FireWire w Linuksie na przykładzie kamery cyfrowej i dysku twardego 90

18^{00} – 19^{00} Hugh Blemings, Paweł Kot
Gnokii now and then 9F

19^{00} – 20^{00} **– kolacja–**

20^{30} – ∞^{∞} KaOwcy
impreza luźna

Program Pingwinariów – niedziela, 30 marca 2003

9^{00} – 10^{00} **– śniadanie–**

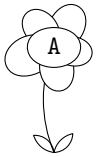
10^{00} – 11^{00} Daniel Koć
Linux na terminalach A8

11^{00} – 12^{00} Hugh Blemings
Linux and Open Source at IBM

12^{00} – 12^{30} Grzegorz B. Prokopski
Debian GNU/Hurd – GNU

12^{30} – 13^{00} Adam Przybyła
Moja przygoda z Pythonem, czyli jak coś zrobić łatwo, szybko i przyjemnie ..... AE

13^{00} – 15^{00} **– obiad–**



YAsiP – Yet Another `/sbin/init` Project

Krzysztof Leszczyński <chris@camk.edu.pl>

STRESZCZENIE: `/sbin/init` jest jednym z najważniejszych procesów w systemach uniksowych, jego praktycznie jedynym zadaniem jest uruchamianie innych programów. Wiele narzędzi dubluje pracę tego programu, z drugiej strony sam `init` mógłby być wyposażony w funkcjonalność, która jest niedostępna procesom, których `pid` nie jest równy 1.

Przyjemnie jest opisywać oprogramowanie, którego albo jeszcze w ogóle nie ma, albo jest w tak wczesnej fazie rozwoju, że na pewno nie można, z czystym sercem, polecać jego wykorzystania. Taka opowieść ma w sobie coś z fantastyki. Mogę z czystym sumieniem twierdzić, że przy pomocy naszego oprogramowania czytający te słowa będzie mógł polecieć na Marsa i naprawdę taką cechę projektować. Życie pewnie, jak zwykle, zweryfikuje te zapędy. Niestety, a może na szczęście, tej akurat cechy nie planujemy, co tylko dowodzi jak bardzo ograniczonymi ludźmi jesteśmy.

URUCHAMIANIE PROCESÓW W SYSTEMACH UNIKSOWYCH

Niewiele tylko kłamiąc można powiedzieć, że w zasadzie każdy działający system uniksowy składa się jądra systemu oraz procesów. Z premedytacją pominę teraz wszelkie inne cechy systemu, jak urządzenia i ich sterowniki, cały bogaty podsystem sieciowy i wiele innych cech. Jądro zaraz po starcie i podstawowej konfiguracji uruchamia tylko jeden proces, w przypadku systemu Linux na ogół `/sbin/init`, taki proces dostaje numer 1. Ten proces pracuje przez cały czas. System kończy pracę wraz z nim.

Ten proces musi być z definicji najbardziej stabilnym procesem w systemie, a zatem trudno się dziwić, że niemal jedyną jego funkcją jest uruchamianie innych procesów.

Dzisiaj istnieje bardzo wiele narzędzi służących do uruchamiania procesów przy zaistnieniu pewnych warunków. Dokonajmy ich pobieżnego przeglądu i zapoznajmy z ich cechami, pozytywnymi i negatywnymi.

Klasyczny `/sbin/init`

`init` jest jednym z najbardziej tradycyjnych programów. Nie zmienił się chyba, w istotny sposób, od czasu powstania systemu Unix. Po uruchomieniu czyta plik `/etc/inittab` i uruchamia zbiór programów. Program działa w jednym z trybów zwanych po angielsku *runlevels*, w którym niektóre programy są uruchamiane i restartowane po zakończeniu, inne nie. Plik `/etc/inittab` oraz tryby (*runlevels*) są jedynymi sposobami konfiguracji inita.

Ta prostota jest wymuszona faktem, że `init` musi być najbezpieczniejszym, a zatem i najlepiej przetestowanym programem w systemie. Pomimo swojej prostoty, w dokumentacji programu są pewne, istotne moim zdaniem, luki. Nie jest opisane co program robi z potokami 0, 1 i 2, czyli standardowym wejściem, standardowym wyjściem i standardowym wyjściem dla błędów. Ze źródeł programu wynika, że wszystkie są przywiązane do konsoli, ale nie wiadomo czy powinno to być standardowe zachowanie.

`init`, a konkretnie proces o numerze 1, jest jedynym procesem, który może nadawać *capability* zabrane innym procesom, na przykład prawo do zrobienia `reboot()`, z tego co wiem, żadna popularna implementacja `/sbin/init` tego nie wykorzystuje. Oczywiście `init` nie jest jedynym programem służącym do uruchamiania innych programów.

Narzędzia `crond`, `atd` oraz `batch`

Chronos (Χρονος), niekiedy mylnie utożsamiany z Kronosem, ojcem Zeusa, był wraz z Eterem i Chaosem stwórcą całego, znanego śmiertelnikom świata i personifikacją czasu. Hory były jego przybranymi córkami, były to: Acte, Anatole, Auge, Carpo, Dike, Dysis, Eirene, Elete, Eunomia, Euporia, Hesperis i Musika.

Rządziły kolejnymi godzinami, co ciekawe Musika rządziła tylko tymi, w których się akurat coś grało, grać można było na lutni lub cytrze albowiem w zamierzchłych czasach Quake'a ani niczego innego nie było jeszcze na rynku. Po

upadku starożytnej kultury, mniej poetycznie nastawieni ludzie po prostu ponumerowali godziny, trwające po 60 minut, niezależnie od tego czy ktoś akurat grał czy nie. Ów podział na minuty i tylko na minuty bez bardziej delikatnego podziału jest podstawą działania omawianego demona. `crond` uruchamia wybrane przez roota lub użytkownika procesy, o określonej godzinie i minucie, określonego dnia. Standardowo strumienie wyjściowe są przesyłane emailem do właściciela procesu. `atd` uruchamia procesy raz, albo o określonej godzinie, jeśli program był zgłoszony do uruchomienia za pomocą polecenia `at`, albo w przypadku zgłoszenia za pomocą `batch` w momencie kiedy obciążenie systemu spadnie poniżej pewnej wartości. Nie jest jasne dlaczego `crond` oraz `atd` to różne programy, w zasadzie robią to samo.

Nie wiadomo też, co się stanie, jeśli pewien program zapisany do wykonania poleceniem `crontab` nie zdąży się zakończyć przed czasem następnego uruchomienia. Weźmy pod uwagę następujący crontab:

```
* * * * * sleep 3600
```

innymi słowy, co minutę chcemy uruchomić polecenie `sleep 3600`. W zależności od użytej wersji programu, `crond` będzie albo czekał na zakończenie zadania albo uruchomi następne. Wersja `vixie-cron` Paula Vixie'ego, używana w Red Hat nie czeka. Staroslackware'owa wersja czekała na wykonanie poprzedniego zadania. Przecież to nie jest wszystko jedno! Takie zachowanie powinno być starannie udokumentowane.

Kolejnym mankamentem jest niemożliwość zobaczenia listy właśnie wykonywanych procesów przez cron oraz listy procesów do wykonania w najbliższym czasie. Taka lista jest możliwa do uzyskania, dla procesów zgłoszonych przez `at` i `batch`, programem `atq`, jeśli tylko zgodzimy się, że jedyną informacją, która możemy uzyskać jest numer zadania przypisany przez `atd`.

Pakiet Daemontools

Profesor D. J. Bernstein stworzył wspaniały zbiór programów, które w zasadzie, po niewielkich przeróbkach, mogłyby zastąpić wysłużonego `init`-a. Są one wyjątkowo starannie napisane i udokumentowane, konia z rzędem temu, kto znajdzie w nich błąd. Nie wdając się szczegóły, na co nie ma w tym konkretnym referacie miejsca, możemy powiedzieć, że jest to system, który uruchamia programy opisane podkatalogami katalogu `/service`. Sygnały wysyłane do procesów oraz przypisanie potoków jest dokładnie opisane. Użytkownik ma do dyspozycji bogaty zbiór programów filtrujących wyniki uruchamianych procesów. Niestety są dwie cechy, które mogą sprawiać problemy autorom

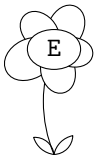
dystrybucji: po pierwsze, licencja na daemontools jest, zdaniem „hipokrytów z Red Hat" zbyt restrykcyjna.[1] Autor jawnie zakazuje wielu zmian, które autorzy dystrybucji mogliby chcieć wprowadzić. Na przykład, nie pozwala na umieszczanie plików z tego pakietu w katalogu `/usr/sbin`, wymagając aby były one w `/usr/local/bin`, co się stało przyczyną niewłączenia pakietu do znakomitej większości dystrybucji. „Hipokryci z PLD"[2] natomiast po prostu przerobili nieco pakiet, tak by pasował do dystrybucji. O ile byłoby prościej, gdyby licencja po prostu na to pozwalała.

Drugim problemem jest niezgadzanie się, i to jawne, autora `daemontools` na nazwanie katalogu z opisem procesów inaczej niż `/service` posadowiony w korzeniu systemu plików; wydaje się, że `/var/lib/service` lub `/etc/service` byłby lepszą propozycją. Prawdopodobnie, w tym wypadku jedynym rozsądnym wyjściem byłaby reimplementacja `daemontools` pod inną nazwą oraz z inną licencją, GPL lub BSD.

Częścią pakietu jest program `svscan`, który skanuje katalog `/service`, lub inny podany mu z wiersza poleceń, co 5 sekund wyszukując nowe zadania do uruchomienia, bądź do zabicia. O dziwo okazuje się, że pracowitość tego programu wcale nie obciąża w istotny sposób całego systemu. Niemniej nie da się przekonać `svscan`, żeby skanował `/service` co minutę lub 5 razy na sekundę, takie zastosowanie też mogłyby się znaleźć, oczywiście najlepiej byłoby gdyby była możliwość wykorzystania pakietu `fam` SGI.

Programy `inetd` oraz `xinetd`

Demona `inetd` nikt rozsądny już dzisiaj nie używa, zajmijmy się zatem `xinetd`. Jest to demon, który słucha sieci TCP/IP – protokołów IPv4 oraz IPv6 i reaguje na próby połączenia TCP, UDP oraz polecenia RPC. Poza kilkoma serwisami, które ma zaimplementowane wewnętrznie, `xinetd` uruchamia programy przypisując im skonfigurowane łącze. Dba też o limitowanie czasu połączenia, czasu w którym połączenie jest dozwolone, komputery, które mogą się podłączać do konkretnych serwisów i kilka innych rzeczy.



Wiadomo dokładnie co `xinetd` robi z potokami 0 i 1 – przypisuje je do otwartego gniazda TCP/IP, nie za bardzo wiadomo co robi z potokiem 2. Tu znowu trzeba spojrzeć do źródeł.

Drugą dziwną cechą, która powinna dać się jednak szybko załatać, jest brak możliwości obsługi gniazd lokalnych `PF_LOCAL`.

[1] Na stronie `http://cr.yp.to/distributors.html`, prof. D. J. Bernstein opisuje dlaczego nazywa Red Hat hipokrytami. Ja tej opinii nie podzielam.

[2] Zakładam, że skoro RH zostali nazwani hipokrytami, to Ci z PLD też by zostali `:-)`

Szczególne prawa init-a

Procesy xinetd, crond i atd nie mają numeru 1 – czyli nie są init-em, a zatem jeśli wywoływany przez nie proces wykona

```
if(fork()==0) exit(0);
```

to program traci kontrolę nad procesem, rodzicem takiego procesu staje się znowu init. Gdyby rodzicem był od razu init to taka operacja nie miała by znaczenia, z punktu widzenia procesu o numerze 1 nie istnieje w ogóle pojęcie programu w tle w podanym wyżej sensie, wszystkie programy są dziećmi, wnukami, bądź (pra)+wnukami tego procesu.

Yet Another /sbin/init Project

Od mniej więcej października 2002 wypłakiwałem się na różnych listach na obecną implementację init-a. W grudniu spotkałem, niestety na razie tylko wirtualnie, miłego człowieka Claude'a Février, zamieszkującego gdzieś pod Grenoble. Postanowiliśmy spróbować reimplementować tak program /sbin/init, aby stał się bardziej otwarty, jednocześnie mając na uwadze newralgiczność tego programu, błąd popełniony w programie init skutkuje zawieszeniem się całego systemu. Podstawowe zadania „teamu" to:

- Zrobienie możliwie prostego, ale nie prostszego, programu, który by umiał zarządzać uruchomionymi procesami i jednocześnie był możliwie odporny na błędy. Program nawet nie musi umieć czytać /etc/inittab
- Zdefiniowanie jednolitego protokołu komunikacji init-a z procesami zarządzającymi
- Reimplementacja niektórych programów, na przykład crond, atd tak, by mogły pracować.
- Zdefiniowanie prostego mechanizmu wtyczek (ang. *plugins*), który by wyposażał program init w nową funkcjonalność.

Nieco o procesach

W systemach uniksowych nie ma pojęcia *uruchamiania programu*. Każdy proces może wykonać jedną z dwóch operacji – fork(), rozwidlającą proces na dwa procesy potomne, jeden o tym samym numerze procesu i drugi o nowym.

- numer procesu i numer procesu rodzicielskiego;
- numer sesji i numer grupy procesów;
- numer rzeczywistego użytkownika, numer grupy głównej i grup dodatkowych;
- alarm, o ile jest ustawiony;
- bieżący katalog roboczy oraz katalog „/”;
- maska tworzenia plików (umask);
- maska sygnałów;
- kolejka sygnałów do przyjęcia;
- zakumulowany zużyty czas procesora;
- *capabilities*.

Rys 1 Cechy procesów w systemie Linux

Drugą z tych operacji jest zastąpienie właśnie wykonywanego procesu innym programem, za tę operację odpowiada funkcja `execve()`.

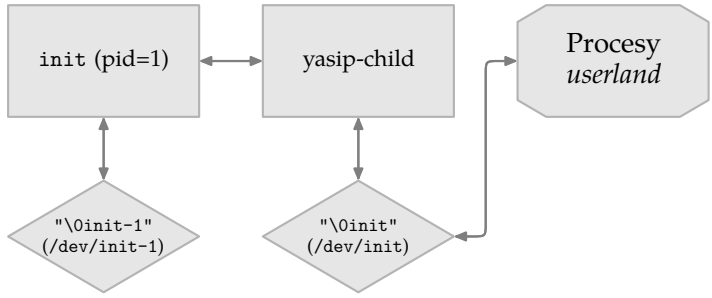
Każdy proces posiada cechy opisane na rysunku 1.

Większość z tych cech nie jest dla nas istotna, ale kilka tak. Uruchamiając po prostu program, np. z shella albo jako podproces innego programu, zazwyczaj nie zmieniamy numeru sesji lub grupy procesów, przeciwnie – zależy nam na tym, aby wszystkie potomne programy były w jednej grupie. Programy takie jak `init` lub `crond` powinny jednak bardzo starannie przypisywać numery sesji oraz grupy procesów. Niekiedy programy, np. takie jak `pppd`, na koniec swojego życia lubią zabijać wszystkie programy ze swojej grupy, o ile nie dostaną własnej, mogą spowodować katastrofę w systemie.

Anatomia programu `/sbin/init`

Jak już napisałem wcześniej, w chwili pisania tego referatu, na przełomie stycznia i lutego 2003, program jest daleki od stanu α, dlatego sporo z przedstawionych poglądów może ulec zmianom. Główny proces, ten o `pid=1`, musi być jak najprostszy. Jego jedynym zadaniem jest trzymanie listy zadań, wraz z ich statusami oraz wykonywanie operacji na *capabilities*. Żadnych innych zadań proces nie wykonuje.

Zaraz po wystartowaniu jako `init`, `yasip` uruchamia proces `yasip-child`, który komunikuje się z initem poprzez anonimowe gniazdo oraz z resztą świata poprzez gniazda `"\0init"` lub `/dev/init`, w zależności od tego, czy system obsługuje lokalne gniazda abstrakcyjne, czy nie. Linux w wersjach 2.2 i 2.4 ma je zaimplementowane. W odróżnieniu od obecnej implementacji `init`-a (`/dev/initctl`), `"\0init"` (lub `/dev/init`) jest gniazdem a nie potokiem. To umożliwia dwustronną komunikację z programami. Sam proces 1 nie komunikuje się w ogóle z innymi procesami poza śledzeniem ich losów i reagowaniem



Rys 2 Komunikacja procesu „1” i jego pochodnych z resztą procesów w systemie.

na SIGCHLD. Istnieje co prawda "\0init1" (lub /dev/init-1), który jest słuchany przez proces 1, ale w zasadzie nie powinien on być wykorzystywany poza testowaniem. Ponieważ prawa /dev/init1 to 0600, jest on i tak niedostępny dla procesów niemających prawa roota. W obecnej implementacji Linuxa nie można nałożyć takich ograniczeń, ale yasip i tak nie pozwoli procesom nierootowym komunikować się z procesem nr 1.

Zauważmy, że jeśli system nie udostępna gniazd abstrakcyjnych, to yasip może otworzyć gniazdo lokalne dopiero po tym jak system, w którym znajduje się /dev zostanie przemontowany w trybie do zapisu, czyli być może nigdy. Dlatego raczej to gniazdo jest używane wyłącznie do celów testowych oraz w systemach, które gniazd abstrakcyjnych nie udostępniają wcale.

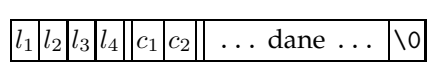
Zadania procesu init

Podstawowym zadaniem procesu jest przeżyć. To jest najbardziej istotne zadanie. init musi istnieć przez całe życie systemu i dlatego musi być możliwie najprostszym, najkrótszym i najdokładniej przetestowanym programem.

Poza przetrwaniem, init musi zarządzać *capabilities*, utrzymywać listę zadań aktualnie wykonywanych, powiadamiać proces potomny yasip-child o zdarzeniach związanych z procesami oraz, na życzenie programu yasip-child, uruchamiać nowe zadania. Szczerze mówiąc, nowe procesy mógłby uruchamiać yasip-child, a nie init jako taki, ale takie działanie byłoby dużo bardziej kosztowne, wymagałoby wykonania podwójnego fork(), zwanego w polskim żargonie *potforkiem*, tak aby proces potomny był dzieckiem init-a. Dlatego yasip-child zleca procesowi „1” uruchomienie odpowiedniego procesu.

Jak już wspomniano, jedną z pierwszych czynności, jaką init robi zaraz po uruchomieniu jest uruchomienie procesu potomnego – `yasip-child`, z którym się komunikuje poprzez anonimowe gniazdo. To gniazdo jest używane do powiadamiania o zdarzeniach. Pomimo, że gniazdo jest typu TCP, łączność jest „pakietowa".

Każdy pakiet przesłany między programami ma ustalony format. Zaczyna się czterema bajtami określającymi długość pakietu, następnie mamy 2 bajty kodu polecenia bądź wyniku. Po kodzie polecenia następuje blok danych zakończony bajtem 0. W bloku danych mogą występować również bajty 0. Dodatkowe zero na końcu służy po prostu sprawdzeniu poprawności całego łańcucha. Na przykład polecenie nr 0 – oznaczające „podaj swoją wersję" zapisujemy jako $|0|0|0|7|0|0|0|$. Polecenie ma 7 bajtów. Prawdę mówiąc nie wierzymy, żeby aż 4 bajty długości były potrzebne, prawdopodobnie wystarczą 2, niemniej mogą się teoretycznie zadania wymagające więcej niż 64kB danych. Na pewno nie przewidujemy pakietów powyżej 4GB, to nie są zadania dla takich małych programów jak `init`.



Lista zadań `init`-a wygląda następująco

`00` podaj swoją wersję.	`12` podaj listę zadań	`F0` wykonaj re-exec
`10` uruchom zadanie	`13` podaj szczegóły konkretnego zadania	`F1` zakończ `init`-a.
`11` zarejestruj zadanie		

Jak widać nie jest ona zbyt imponująca. Zawiera w zasadzie 3 polecenia użytkowe, gdyż polecenie `00` jest tylko na wszelki wypadek, zaś polecenia `F0`, `F1` należą raczej do tych samobójczych. No a gdzie polecenia związane z *capabilities*? Nie ma ich, będą w kolejnej wersji. Prawdopodobnie odpowiednie polecenia będą miały numery zaczynające się od `20`.

```
typedef struct {
   char *job_id;
   pid_t pid, psid, pgrp;
   struct rusage rusage;
   struct *prop_hash;
} job_t;
```

`init` trzyma informację o zadaniach tablicy. Każde zadanie ma swoją, niekoniecznie unikalną nazwę – `job_id`. inita nazwa zadania nie obchodzi, po prostu podaje ją w trakcie podawania listy zadań oraz szczegółów zadania. Pola `psid` i `pgrp` określające odpowiednio numer sesji i numer grupy procesów odpowiadają wartością jakie `init` nadał w trakcie początku wykonywania programu. Oczywiście program ma prawo je zmienić w trakcie swojego życia, `init` ani nie jest w stanie temu zapobiec ani nie stara się śledzić tych zmian. Wszelkie inne dane są przechowywane w postaci

słownika, struktury podobnej do perlowego hasha. W tej strukturze są przechowywane wszelkie zmienne cechy programu, na przykład wiersz poleceń. Obecnie wrzuciliśmy tam pole `dontwait`, jego obecność powoduje, że nie jest bardzo istotne dla użytkownika jak tablica dokładnie wygląda, istotne jest, że jest prosta. Sądzimy, że żadna implementacja zbytnio jej nie rozdmucha.

Program `yasip-child`

Skoro musimy chronić program `init`, przed skomplikowaniem, a co za tym idzie przed nieuchronnymi błędami, które mogą być fatalne dla systemu, możemy wrzucić całą funkcjonalność w proces potomny. Zauważmy, że tych procesów może być więcej niż jeden. `init` po prostu patrzy na otwarte gniazda i uruchamia nowy `yasip-child` w momencie kiedy stwierdzi, że nie ma z kim rozmawiać na temat procesów, czyli w momencie kiedy wszystkie procesy `yasip-child` umarły bądź przynajmniej pozamykały swoje gniazda.

Aby móc zastąpić starego `init`-a naszym (przynajmniej testowo), musimy przynajmniej zaimplementować program interpretujący `/etc/inittab`. Przy okazji, należy podkreślić, że proces „1" w naszej implementacji kompletnie nie wie nic o *runlevels.* I nie musi wiedzieć, musi tylko sprawnie reagować na polecenia swoich procesów potomnych. `yasip-child`, prawdę mówiąc też o nich nic nie wie. Jesteśmy dosyć głęboko przekonani, że prowadzenie, dawno temu w systemie UNIX, pojęcia *runlevel* nie było najszczęśliwszym z pomysłów. Dlatego zaimplementujmy tę funkcjonalność jako jedną z wtyczek (ang. *plugin*).

Wtyczki

Wtyczką nazywamy obiekt (SHARED OBJECT), czyli bibliotekę dynamiczną, którą `yasip-child` może otworzyć za pomocą `dlopen`. Biblioteka może zawierać funkcje obsługi każdego zdarzeń, na przykład `handler_process_exit`. Dokładna lista procedur typu `handler_*` znajduje się w pliku `yasip.h`. Biblioteka też może uruchomić nowy wątek w obrębie procesu `yasip-child` lub nawet uruchomić nowy proces. W większości wypadków, mamy nadzieję w ogóle nie będzie powodu uruchamiać wątków.

Zakończenie

Bardzo luźne, testowe wersje programu można już albo będzie można wkrótce ściągnąć z `ftp://ftp.yasip.org/`, raczej nie należy jeszcze uruchamiać tego jako główny `init`.

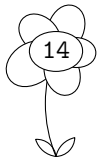
Systemy wykrywania włamań (IDS)

Michał 'carstein' Melewski <carstein@7thguard.net>

STRESZCZENIE: Systemy wykrywania włamań należą do klasy półinteligentnych programów zabezpieczających. Coraz częściej stosuje się je w celu uzupełnienia ochrony realizowanej przez zapory ogniowe. Systemy te można podzielić na dwa podstawowe typy: działające w modelu pojedynczej maszyny oraz działające w modelu sieciowym (NIDS – Network Intrusion Detection System). Do pierwszych zaliczamy np: LIDS, tripwire oraz AIDE, a do drugich np: Snort i Cisco IDS. NIDS-y mogą z kolei działać w dwóch trybach: analizy anomalii oraz wyszukiwania wzorców.

IDS – WPROWADZENIE

Systemy wykrywania włamań (IDS) należą do stosunkowo nowych narzędzi zabezpieczających. Stosuje się je obecnie do uzupełniania ochrony realizowanej przez zapory ogniowe. Pierwsze wyobrażenie takich systemów stworzył w swych książkach William Gibson. Jego wizja *cyberprzestrzeni*, w której kowboje klawiatury walczyli z inteligentnymi systemami obronnymi była tak sugestywna, że jej odpryski widzimy dziś nawet w nazewnictwie pewnych systemów. W tamtych czasach nie istniała ani potrzeba, ani możliwości techniczne, aby te pomysły wcielić w życie. Jeszcze kilka lat temu za wystarczające zabezpiecznie sieci uważano dobrze skonfigurowaną ścianę ogniową, czasami również serwer proxy. Ostatnie kilka lat pokazało jednak, jak mylne było takie podejście. Coraz to nowe techniki przełamywania zabezpieczeń, coraz większa częstotliwość ataków, coraz lepsze techniki maskowania – wszystko to spowodowało konieczność stworzenia nowej klasy narzędzi zabezpieczających. Pierwszymi tego typu systemami były narzędzia chroniące tylko i wyłącznie maszynę, na której były zainstalowane. Szybko jednak stwierdzono, że czasami to nie wystarczy. Ewolucją tego pomysłu były kolejno systemy obejmujące swym działaniem całą sieć



i monitorujące ruch. Ostatnio coraz większą popularność zdobywają systemy hybrydowe, które separują sensory wykrywające ataki od serwera gromadzącego i analizującego dane.

Podstawy działania

Wśród współczesnych systemów wykrywających włamania wyróżniamy zasadniczo dwa modele działania:

- systemy działające w modelu pojedynczej maszyny (HBIDS – Host Based Instrusion Detection System)
- systemy działające w modelu sieciowym (NIDS – Network Instrusion Detection System)

Oczywiście nie ma sensu ich porównywać, gdyż spełniają one zupełnie inne role i świetnie się nawzajem uzupełniają. Scharakteryzuję krótko ich działanie.

HBIDS jest zazwyczaj lokalizowany na strategicznych, z punktu widzenia bezpieczeństwa, komputerach, czyli na maszynach bastionowych i serwerach dostępowych. Jego podstawowym celem jest ochrona, zapewnienie integralności i niezawodności wybranej maszyny. Aby móc spełniać te zadania jest on zazwyczaj wyposażony w różne funkcje, takie jak przechowywanie danych na temat integralności plików w systemie, zdolność do wykrywania koni trojańskich oraz tzw. *rootkitów*, monitorowanie logów oraz wykrywanie prób skanowania systemu. Wyspecjalizowane HBIDS-y aplikowane w postaci łat na jądro systemu potrafią nawet zatroszczyć się o ochronę najbardziej kluczowych i przyziemnych elementów systemu jak ochrona stosu i pamięci programów.

Jeśli chcemy ochronić całą sieć, to nasze zadanie troszkę się komplikuje. NIDS umieszczamy na komputerze, którego interfejs sieciowy pracuje w trybie 'promiscuous', czyli innymi słowy przechwytuje do dalszej analizy wszystkie pakiety, a nie tylko te, które są do niego bezpośrednio adresowane. Jeśli chodzi o analizowanie treści pakietu, to istnieją zasadniczo dwa podejścia; wyszukiwanie wzorców i analiza anomalii.

Jeśli wziąć pod uwagę pierwszy model to jest on dość prosty i klarowny. Zawartość pakietu oraz jego nagłówek porównywane są z bazą sygnatur i jeśli któraś z nich pasuje do wcześniej zdefiniowanej reguły, to podejmowana jest przypisana do tej reguły akcja. Jeśli np. w pakiecie wykryjemy ciąg symboli, o którym wiemy, że jest częścią niebezpiecznego ataku (np. atak przepełniający bufor) i może zagrozić bezpieczeństwu, to NIDS zerwie takie połączenie

i może nawet przeprowadzić kontratak. Wadą tego rozwiązania jest to, że musimy na bieżąco uaktualniać bazę ataków, a mimo to zawsze możemy zostać zaskoczeni przez jakiś nowy, nieznany rodzaj ataku. Zaletą jest mała liczba generowanych fałszywych alarmów.

Drugie podejście jest nieco bardziej skomplikowane. Próbuje się tutaj wyłapywać nowe ataki poprzez wyłapywanie anomalii w ruchu sieciowym oraz poddanie pakietów analizie statystycznej. Oczywiście najpierw, przez pewien czas, sieć musi być *uczona*, jak wygląda normalny ruch. W przykładzie wygląda to tak, że jeśli znacznie (np. o rząd wielkości) wzrośnie liczba zdefragmentowanych pakietów w sieci, to wszczęty zostanie alarm. Niewątpliwą zaletą takiego podejścia jest możliwość wykrycia nowych ataków, a wadą duża liczba fałszywych alarmów.

Obecnie systemy NIDS nie opierają się wyłącznie na jednej metodzie, ale korzystają z kombinacji dwóch powyższych.

HBIDS - opis istniejących rozwiązań.

LIDS i openwall

Są to przykłady dwóch niskopoziomowych systemów aplikowanych w postaci łat na jądro systemu operacyjnego (co niestety *ukręca łeb* wszelkim próbom przenoszenia tych rozwiązań na inne platformy), którym w tym wypadku jest Linux.

LIDS (Linux Intrusion Detection System) to projekt, którego celem było zapewnienie kilku dodatkowych zabezpieczeń na poziomie jądra. Jego podstawową cechą jest możliwość zabezpieczania plików i procesów na poziomie jądra, nawet przed superużytkownikiem. Użyteczność tego rozwiązania jest wyraźnie widoczna, jeśli jakiś cracker przełamał naszą wymyślną linię obrony, zdobył prawa superużytkownika i chce podmienić np. naszą stronę WWW lub załadować dodatkowy (i raczej niezbyt pożądany) moduł do naszego jądra. Realizowane jest to poprzez zdefiniowanie w jądrze list kontroli dostępu zarówno dla plików (chroni przed podmianą lub zmazaniem danych), jak i programów (uniemożliwia postawienie na naszej maszynie koni trojańskich). Kolejną ciekawą cechą jest wykrywacz skanowania portów zaimplementowany na poziomie jądra.

Openwall autorstwa słynnego Solar Designera jest bardzo podobny do LIDS-a, aczkolwiek jego możliwości są nieco większe. Do ważniejszych można zaliczyć implementację niewykonywalnego stosu, ograniczenie tworzenia odnośników i potoków w /tmp oraz niszczenie nieużywanych segmentów pamięci. Rozwinięciem działania openwalla dla jąder serii 2.4.x jest grsecurity, który

wprowadza dodatkowe funkcje, takie jak listy kontroli dostępu, losowe wartości PID dla procesów, limitowane użycie funkcji `chroot()` i wiele innych.

AIDE i tripwire

Kolejnym rodzajem systemów HBIDS są narzędzia służące do kontroli integralności plików. Służą one do ochrony systemu przed podmianą kluczowych dla bezpieczeństwa systemu plików.

Pierwszym z historycznego punktu widzenia i używanym do dziś tego typu narzędziem jest tripwire. Pełni on swą funkcję przechowując wygenerowane sumy kontrolne dla wybranych plików i na żądanie porównując je ze stanem obecnym. W razie jakiejkolwiek różnicy wszczynany jest alarm. Dzięki temu możemy być niemal pewni, że wszelkie programy służące do zabezpieczania naszego serwera nie zostaną podmienione na wersje zawierające konia trojańskiego lub kukułcze jajo.

AIDE (Advanced Intrusion Detection Enviroment) to narzędzie będące rozwinięciem pomysłu tripwire. Różnice praktycznie sprowadzają się do używanych algorytmów generowania sum kontrolnych (AIDE posiada ich pokaźną liczbę i bez problemu można dodawać kolejne).

Ważną wskazówką jest tutaj trzymanie bazy danych sum kontrolnych na zewnętrznym, niezapisywalnym nośniku. Najlepiej, żeby była to zabezpieczona przed zapisem dyskietka lub uprzednio nagrana płyta CD umieszczona w czytniku (a nie nagrywarce). Jeśli będziemy trzymać bazę na twardym dysku, to po włamaniu napastnik po prostu wygeneruje sobie bazę na nowo.

NIDS na przykładzie Snorta.

Do jednych z najbardziej znanych i najczęściej używanych NIDS-ów z pewnością należy Snort stworzony przez Marty'ego Roescha. Jego główne zalety to otwartość kodu, duża liczba dostępnych sygnatur ataków oraz łatwość tworzenia nowych, a także dobra integracja z zewnętrznymi programami (SPADE, Check Point FireWall-1, iptables itp). Postaram się omówić w jaki sposób zabezpieczyć swoją sieć za jego pomocą.

Miejsce w sieci

Na samym początku wchodzimy na dość grząski grunt, obszar wielu świętych wojen, a mianowicie 'gdzie umieścić nasz IDS?'. Istnieją dwa główne punkty widzenia; możemy go (czyli sensor) umieścić przed albo za zaporą ogniową.

Istnieją przeciwnicy i zwolennicy każdego z tych podejść, ale zasadniczo cała sprawa sprowadza się do nazewnictwa. Jeśli umieścimy go za zaporą ogniową, to będzie to 'system wykrywania włamań', a jeśli przed, to otrzymamy 'system wykrywania ataków'. Moim zdaniem na początku powinniśmy umieścić go przed zaporą ogniową w celu stwierdzenia, jakim atakom nasza sieć będzie w przyszłości poddawana, a po zgromadzeniu odpowiedniej ilości danych i dopracowaniu zapory przeniesienie go do wewnątrz. Najlepszym wyjściem dla istniejącej już sieci jest umieszczenie sensora w tzw. *strefie zdemilitaryzowanej*, a jeśli sieć jej nie posiada, to w sieci wewnętrznej, ale na pewno przed wszelkimi serwerami zawierającymi ważne dla nas dane.

INSTALACJA

Instalacja w systemie Linux jest w zasadzie klasyczna. Sprowadza się do wydania trzech nieśmiertelnych poleceń:

```
./configure
make
make install
```

Do ważniejszych opcji skryptu configure należy zaliczyć:

```
--with-mysql
--with-postgresql
--with-odbc
--with-oracle
```

Co pozwoli nam na logowanie wyników działania Snorta do rozmaitych baz danych.

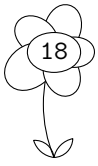
```
--with-snmp
```

Ta opcja z kolei umożliwia nam logowanie rezultatów via SNMP. oraz

```
--enable-flexresp
```

Co pozwoli nam na zrywanie połączeń, jeśli zostanie wykryty atak.

Po skompilowaniu i zainstalowaniu otrzymujemy prawie gotowy do działania sieciowy system wykrywania włamań. Należy jeszcze tylko upewnić się, że skopiowaliśmy wszystkie wymagane pliki konfiguracyjne do `/etc/snort` oraz odpowiednio podpięliśmy skrypty startowe (dostępne wraz ze źródłami w katalogu contrib) i możemy startować.



URUCHOMIENIE

Snorta można uruchomić w trzech trybach działania; jako sniffer, packet logger oraz system wykrywania włamań.

Jeśli np. chcemy wyświetlać tylko nagłówki i dane z przychodzących pakietów TCP/IP (tryb podsłuchiwania) to uruchamiamy Snorta poleceniem

```
./snort -vd
```

Gdybyśmy chcieli przejść w tryb logowania pakietów (ang. *packet logger*) aby przechwycone pakiety zapisać na dysk musielibyśmy uruchomić Snorta w następujący sposób

```
./snort -vd -l ./log
```

co spowoduje, że pakiety trafiać będą do katalogu `log`.

Najważniejszy tryb pracy ma wiele dodatkowych opcji, takich jak podanie sieci macierzystej, sposobu logowania itp., ale zasadniczo pełną funkcjonalność osiąga się poprzez komendę

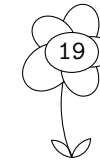
```
/usr/local/bin/snort -d -h 192.168.1.0/24 -l /var/log/snort \
-c /etc/snort/snort.conf -s -D
```

Po dalsze szczegóły opcji odsyłam do podręcznika systemowego.

Jak już wcześniej wspomniałem, w katalogu contrib w źródłach Snorta można znaleźć całkiem dobry skrypt bash-a służący do uruchamiania go przy starcie i zatrzymywania przy zakończeniu pracy systemu. Wystarczy tylko skopiować go do `/etc/init.d` a następnie ustawić symboliczne dowiązania. W moim przypadku (debian) wygląda to tak jak poniżej. Dla innych dystrybucji, z innym systemem inicjowania będzie zapewne inaczej.

```
ln -s /etc/init.d/snort /etc/rc2.d/S95snort
ln -s /etc/init.d/snort /etc/rc6.d/K15snort
ln -s /etc/init.d/snort /etc/rc0.d/K15snort
```

(oczywiście każdy dobierze sobie parametry w pliku, takie, jakie mu odpowiadają, np. zmieni miejsce położenie pliku konfiguracyjnego, numer sieci macierzystej itp.).



PREPROCESOR

Czas zająć się samym działaniem Snorta, pamiętać bowiem należy, że system z pudełka, nieprzystosowany do naszej sieci/naszych potrzeb, bardziej nam zaszkodzi (np. zmęczy liczbą fałszywych alarmów) niż pomoże.

Analizując drogę przechwyconego pakietu można z łatwo wywnioskować, że zanim zostanie on porównany z jakąkolwiek sygnaturą ataku trafia najpierw do preprocesora.

Preprocesorami są również zewnętrzne programy, które właśnie w takie sposób współpracują ze Snortem.

Aby wstawić do Snorta nowy preprocesor lub skonfigurować już istniejący musimy do pliku konfiguracyjnego dodać następującą dyrektywę

```
preprocessor <name>: <options>
```

lub, ewentualnie, odnaleźć już istniejącą.

Ważniejsze preprocesory:

HTTP decode

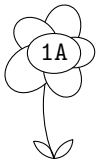
```
preprocessor http_decode: 80 8080 -unicode -cginull
```

Odpowiada on za przetwarzanie ciągów znaków HTTP URI i konwersję ich do ciągów ASCII. Zapobiega to próbom ominięcia NIDS-a poprzez kodowanie pewnych wartości przekazywanych za pomocą zapytań do serwera WWW. Kolejną rzeczą jest wyrzucanie pustych bajtów przekazywanych do CGI.

Portscan detector

```
preprocessor portscan: $HOME_NET P T /var/log/portscan.log
```

Funkcja tego preprocesora jest dość oczywista. Wykrywa on sytuacje, w których zostaje przekroczony próg liczby połączeń na pewną liczbę różnych portów reprezentowanych tu przez symbol 'P', w ciągu liczby sekund określonej przez T. Wykrywa on też wszelkie próby skanowania typu NULL, FIN, SYNFIN, XMAS itd. W planach jest dodanie wykrywania skanowania typu wiele↦jeden i wiele↦wiele.



Frag2

```
preprocessor frag2: 4194304 60
```

Frag2 zajmuje się składaniem w całość zdefragmentowanych pakietów. Pakiety takie są potencjalnie niebezpieczne, ponieważ analizując poszczególne fragmenty osobno możemy dojść do wniosku, że pakiety nie stanowią zagrożenia,

natomiast pakiet złożony przez aplikację po dotarciu na miejsce może się okazać chytrze zamaskowanym, lecz groźnym atakiem. Argumentem tego polecenia jest ilość pamięci jaką można poświęcić na składanie pakietów i czas jaki będziemy oczekiwać na fragmenty.

Stream4

```
preprocessor stream4: [noinspect], [keepstats], [timeout <seconds>] \
[memcap <bytes>], [detect_scans], [detect state_problems]
preprocessor stream4_reassemble: [clientonly], [serveronly], [noalerts] \
[ports <portlist>]
```

Jest to chyba jeden z najpotężniejszych preprocesorów dostępnych w Snorcie. Jego najważniejsze funkcje to składanie strumieni TCP, inspekcja stanu połączenia oraz śledzenie połączeń. Dość obszerny opis dostępny jest w dokumentacji.

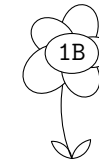
SPADE (Statistical Packet Anomaly Detection Engine) Jest to narzędzie służące do przeprowadzania analiz statystycznych na przechwyconych pakietach, co umożliwia wykrywanie wcześniej nie znanych ataków. W przyszłości ma to być podstawowy model detekcji włamań.

Zestawy reguł

Do podstawowego wykrywania włamań służą Snortowi reguły, która napisane są w specjalnie do tego celu stworzonym języku. Język ten odznacza się elastycznością, zwięzłością i łatwością zrozumienia. Ponieważ najłatwiej uczyć się na przykładach przeanalizujmy przykładową regułę

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80
(msg:"WEB-IIS cmd.exe acces"; flags: A+;
content:"cmd.exe"; nocase;
classtype:web-application-attack; sid:1002;)
```

Jak łatwo zauważyć, jest to sygnatura typowego ataku na serwer stron WWW pewnej znanej firmy. Zajmijmy się więc analizą takiej regułki. Pierwsze co widzimy, to wyróżniający się podział reguły na dwie główne części: część nagłówkową i część opcji. Jeśli chodzi o nagłówek, to widzimy tam taką konstrukcję:




```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80
```

Pierwszym polem jest akcja związana z daną regułą. Standardowo dostępne są akcje: `alert`, `log`, `pass`, `activate` oraz `dynamic`, ale w razie potrzeby możemy tworzyć własne. Kolejne pole określa protokół, w tym wypadku tcp. Obecnie

obsługiwane są protokoły TCP, UDP, ICMP oraz IP. Następne dwa pola określają adres i port maszyny źródłowej (nadawcy). Oczywiście nie musimy się ograniczać do jednego adresu, możemy od razu podać całą podsieć (`$EXTERNAL_NET` to oczywiście zmienna zdeklarowana w pliku `snort.conf`). W naszym przypadku uznajemy, że port nadawcy może być dowolny, stąd słówko kluczowe `any`. Znaczek `->` wskazuje kierunek połączenia i raczej nie należy go zmieniać. Do dalszej części odnosi się opis dwóch poprzednich pól, z tym, że tutaj podajemy adres odbiorcy.

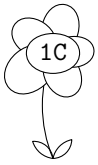
Jeśli chodzi o drugą część:

```
(msg:"WEB-IIS cmd.exe acces"; flags: A+;
content:"cmd.exe"; nocase;
classtype:web-application-attack; sid:1002;)
```

to z powodu liczby dostępnych opcji ograniczę się tylko do analizy flag występujących w przykładzie. Aby w pełni wykorzystać możliwości Snorta należy zapoznać się z dokumentacją. Zaczynając więc po kolei:

- `msg:"Web-IIS cmd.exe acces"` – opis alarmu
- `A+` – nakazuje sprawdzenie wszystkich flag w pakiecie
- `content: "cmd.exe"` – ten wzór musi być obecny w *ładunku* pakietu
- `nocase` – w czasie porównywania nie uwzględnia się wielkości liter
- `classtype: web-application attack` – klasyfikacja alarmu `sid:1002` – Identyfikator reguły Snorta

LOGOWANIE ZDARZEŃ



Skoro mamy już reguły, to zastanówmy się gdzie będą trafiać nasze dane. Snort wspiera naprawdę wiele metod gromadzenia danych o wszelkich nieprawidłowościach w sieci. Moduły zapisu są zaimplementowane podobnie jak moduły preprocesora, czyli włącza się je dyrektywą w pliku `snort.conf`. Dyrektywa ta wygląda następująco

```
output <name>: <options>
```

Wspomnieć też należy, że możemy używać wielu modułów logowania jednocześnie.

Omawiając dostępne moduły należy zacząć od najpowszechniej używanego, czyli od `alert_syslog`, który przekazuje wszystkie wygenerowane alarmy do sysloga, który na podstawie otrzymanych opcji decyduje, w którym pliku dziennika zapisać otrzymane informacje. Przykładowy wpis może wyglądać tak:

```
output alert_syslog: LOG_AUTH LOG_ALERT
```

Jeśli chcemy uzyskać pełny zapisu ataku to na pewno należy wspomnieć o module `log_tcpdump`, który umożliwia zapis pakietów w formacie programu tcpdump, co umożliwia nam wgląd zarówno w nagłówek, jak i treść przechwyconego pakietu.

Podążając za najnowszymi trendami Snort umożliwia oczywiście zapis w formacie XML zarówno na maszynie lokalnej jak i przesyłanie raportu do maszyny zdalnej. Moduł ten pozwala budować rozległe sieci systemów wykrywania włamań z wieloma sensorami i centralnym serwerem gromadzenia informacji.

Moduł `database` umożliwia nam tworzenie dzienników systemowych bezpośrednio w bazach danych. Skrypty języka SQL tworzące struktury potrzebne do przechowywania danych o atakach znajdują się w katalogu contrib w źródłach Snorta.

Istnieje jeszcze wiele innych metod i wszelkich informacje o nich mogą być z łatwością znalezione w dokumentacji programu.

REAKCJA NA INCYDENTY

To, że wszelkie próby ataku zostaną zapisane w dziennikach systemowych (lub w innym, wybranym miejscu) czyni nas może szczęśliwszymi, ale niespecjalnie podnosi (mówimy o krótkim horyzoncie czasowym) bezpieczeństwo naszego systemu. Dobry NIDS powinien mieć możliwość reagowania na incydenty w czasie rzeczywistym. Snort oczywiście takie możliwości posiada, a wykorzystanie ich nie powinno sprawić zbyt wiele problemu.

Pierwszym sposobem jest wykorzystanie zewnętrznego programu o nazwie Guardian. Program ten monitoruje logi i reaguje na próby ataku budując w czasie rzeczywistym reguły zapory ogniowej. Program ten niestety ma pewne wady; po pierwsze jak na razie współpracuje jedynie z ipchains, po drugie wymaga nieco pracy przy integracji, aby nie powodować zbyt dużej liczby fałszywych alarmów.

Kolejną metodą jest użycie w części opcji reguł specjalnej klauzuli - Resp. Przy jej pomocy można łatwo bądź to zresetować połączenie, bądź odesłać komunikat ICMP o niedostępności maszyny. Z użyciem tej opcji wiążą się jednak

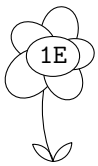
pewne niebezpieczeństwa. Dość łatwo można wprowadzić Snorta w pętlę, bądź skonstruować reguły wzajemnie się wykluczające.

Istnieje także klauzula React, ale ze względu na jej niepełną implementację nie będę się nią zajmował.

PODSUMOWANIE

Mam nadzieję, że przekonałem przynajmniej większość z Was do bliższego zapoznania się z tym ciekawym zagadnieniem, jakim są systemy wykrywania włamań. Praktyka pokazuje, że coraz więcej firm i instytucji zaczyna wdrażać w swoich sieciach właśnie tego typu oprogramowanie. Schodząc jednak na Ziemię należy wymienić kilka wad tego typu systemów. Po pierwsze jest to ograniczona możliwości analizy spowodowane ilością ruchu. Snort na przykład znakomicie sprawdza się monitorując sieci o przepustowości 10 Mbps i 100 Mbps, ale zaczyna mieć poważne trudności kiedy umieścimy go w sieci 1 Gbps. Rozwiązaniem tego problemu mogą być nie programowe systemy wykrywania włamań, ale specjalizowane urządzenia, jak na przykład Cisco IDS. Kolejnym problemem jest, z całą pewnością, konieczność częstej aktualizacji sygnatur ataku, gdyż metody ataku są pernamentnie doskonalone, a nie ma nic gorszego niż *stary* system dający złudne poczucie bezpieczeństwa. Niestety metoda zwana analizą anomalii jest wciąż w tzw. *wieku niemowlęcym* i generuje wiele fałszywych alarmów. Obecnie widać dwie drogi rozwoju tego typu systemów; wykorzystanie sieci neuronowych lub wnioskowania rozmytego, ale to temat na zupełnie inny referat.

ODNOŚNIKI



- `http://www.snort.org` – strona domowa Snorta
- `http://www.cs.tut.fi/~rammer/aide.html` – strona projektu AIDE
- `http://www.lids.org` – strona projektu LIDS
- `http://www.grsecurity.org` – strona projektu grsecurity
- `http://www.securityfocus.com/ids` – bardzo dużo wszelakich informacji o IDS-ach
- `http://www.prelude-ids.org` – strona całkiem niezłego Hybrid IDS-a.

Prelude IDS: current state and development perspectives

Krzysztof Zaraska

ABSTRACT: Despite the existence of many high-quality open source network security tools, there are not many open source products that could be used for enterprise level intrusion detection systems (IDS). The Prelude IDS project aims to fill this gap by creating a free (GPL), robust, fast, and modular hybrid IDS, comparable with existing commercial solutions. Prelude's architecture allows building of centralized, multi-tier systems spanning many hosts and networks and combining both host and network based detection techniques, as well as easy integration of third-party applications as detection modules. This paper focuses on the current stable version (0.8), presenting its architecture, practical deployment considerations as well as overview of current development efforts.

1. INTRODUCTION

Networked computer systems can be subject to various kinds of attacks, originating both from inside and outside of an organization running the system. Therefore, it was necessary to create automated tools that allowed monitoring of the system in order to detect a potentially malicious activity and generate alerts once such activity is detected. These tools are referred to as *Intrusion Detection Systems* (IDS).

Intrusion detection systems can be divided into two primary groups: *host-based* and *network-based*. Host-based IDS analyze the events on one host, such as calls to system functions, file alterations, access control list violation etc. Network-based systems analyze network traffic in order to detect potential intrusions. Host-based systems generally allow better control over the system supervised, while network-based systems can supervise more machines (e.g. it's a typical solutions to use one network IDS per network segment). Both kinds of the systems are complementary, since certain kinds of attacks are easier to detect with one kind than the other, or may be even impossible to detect by the other kind (e.g. a host-based IDS may be unable to detect a DoS attack

against the implementation of TCP/IP stack of the host it is running on). An IDS combining both host and network based detection is called a *hybrid* IDS.

Another possible classification of IDS is into signature-based and non signature-based: the former compare events (e.g. network packets or system log entries) against a database of known manifestations of attacks, called *signatures*. For example finding a `cmd.exe` string in a packet containing an HTTP GET request can denote a potential threat (why would a client want the Web server to run a command interpreter?) and can be used as a signature. The latter class of systems is usually based on statistical analysis (i.e. detecting unusual events) or protocol analysis, for example detecting abnormal message format (e.g. an abnormally long username, indicating a possible overflow attempt) or improper protocol state transitions (e.g. a file transfer immediately following a failed login).

It should be noted that IDS products often combine the techniques, for example a NIDS product may employ both signature-based and statistical detection.

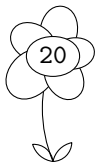
A comprehensive list of currently available intrusion detection systems (both open source and commercial) has been compiled by A. Talisker [1].

2. The IDMEF standard

The diversity of various available intrusion detection systems, performing various types of analysis has created the need for a common data format allowing interoperatibility between various solutions.

Such a format, called Intrusion Detection Message Exchange Format (IDMEF) has been proposed by D. Curry and H. Debar. The specification of the format is currently available in the form of an Internet Draft [2]. The proposed standard describes a data model for describing events that can be reported by intrusion detection systems along with XML DTD intended for representation of IDMEF messages in XML format.

IDMEF uses an object-oriented approach and defines two types of messages:



Heartbeat message, which is sent periodically by the sensor (the entity performing actual intrusion detection) in order to inform the entity it reports to that it is operating correctly.

Alert message, which is generated whenever a sensor detects an event that should be reported (an intrusion attempt).

Both alert and heartbeat messages carry information about the entity that sent the alert (called an *analyzer* in IDMEF specification and a *sensor* in Prelude) and timestamp information.

In an alert message, the event being reported is described using following objects:

ANALYZER the entity which emmitted the alert

CLASSIFICATION what attack has been detected

SOURCE describes the source of the attack. The source may be any combination of multiple objects describing a network node (e.g. IP address), an user, a process or service (i.e. TCP port). An alert can have multiple source objects.

TARGET the target of the attack. The target may be any combination of multiple objects describing a network node, an user, a process, a service or a file. An alert can have multiple target objects.

ASSESSMENT describes the severity of the attack and confidence of the analyzer about the validity of the alert

The data not covered by the model can be transferred as name-value pairs using AdditionalData objects.

The design of IDMEF format provides a number of advantages, such as:

- ability to carry messages generated by both network and host based sensors
- alerts can be aggregated (e.g. a portscan reported initially by a sensor generating one alert per packet can later be aggregated into a single alert containing the information about all the source and target hosts)
- the format is easy to extend, either by using AdditionalData objects or by defining new object types.

It must be noted that IDMEF also has some severe drawbacks, such as:

- complexity
- object-oriented design creates numerous difficulties when creating an equivalent representation for use with relational (SQL) databases
- low performance of XML processing

3. STRUCTURE OF PRELUDE IDS

Prelude IDS is a hybrid, modular and IDMEF-based intrusion detection system, available freely under GPL license.

Historically, Prelude was a network IDS (NIDS). Its unusual feature was that its functionality has been split between two daemons: one performing the detection and one performing reporting operations. Further evolution included embracing of IDMEF standard, adding of host-based analysis capabilities and has led to the current shape of the system.

Prelude was implemented with these goals:

MODULARITY. The functionality is split among different modules. Various modules perform different tasks and it is possible to extend or limit the functionality by simply adding or removing the appropriate module.

IDMEF SUPPORT. IDMEF data model is used for all internal processing of the event data. Because operating on XML introduces a large overhead, Prelude uses its own implementation of the data model as C structures. IDMEF messages between system components are sent in binary format, corresponding to these internal structures (to avoid time-consuming convertions). In order to provide interoperatibility with other products, the internal representation can be converted to standard-compliant XML.

HYBRID DETECTION MODEL. Modularity along with IDMEF-based messaging protocol allows integration of both host-based and network-based detection techniques.

RELIABILITY. Prelude implements certain mechanisms that ensure that IDMEF messages once introduced into the system will not be lost in case of problems.

EXTENSIBILITY. Existing or new detection applications can be easily modified in order to become full-featured sensors thanks to use of libprelude library (see section 4.1).

As mentioned above, a Prelude IDS system is built of several components performing different functions:

SENSORS. A sensor performs host or network-based analysis and sends IDMEF messages to a given manager. Secure communication between sensor and manager is provided by libprelude library (see section 4.1).

MANAGERS. A manager receives, logs and forwards messages from sensors (a single manager can receive alerts from multiple sensors, allowing data centralisation). Note that the manager may be located on a different machine than the sensor reporting to it. This allows moving the overhead related to logging (binary-to-text translation etc.) to another host, thus reducing the

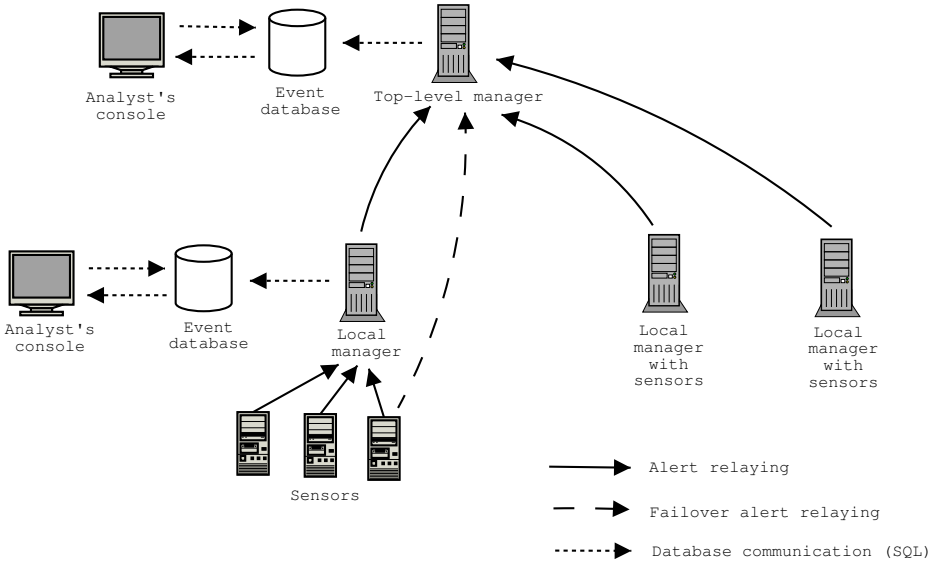


Figure 1 Sensors and managers

CPU requirements of the sensor itself. Also, a manager can be configured to pass received messages to another (higher level) manager. Note that a manager with sensors reporting to it can be viewed from the higher level manager as a "super sensor". An example of system with relaying managers can be seen in figure 1. In this system, a local analyst has access to alert data concerning her own network, while a higher level analyst has access to alert data from all lower level networks.

FRONTENDS. A frontend application allows the user to view and analyze alerts logged by a given manager. An usual setup uses a manager logging to an SQL database and a frontend application reading data from database and presenting them to the user.

4. COMPONENTS OVERVIEW

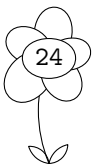
4.1. LIBPRELUDE, THE PRELUDE LIBRARY

libprelude is a shared library providing functions common to all Prelude sensors. In particular, the features include:

- Definitions and helper functions for building IDMEF data structures.
- Handling of communication with Prelude Manager. The library provides functions allowing connection to the Prelude Manager sensor server and sending it IDMEF messages. For performance reasons, IDMEF data are sent in binary form with endianness convertion, thus allowing interoperability between various hardware platforms. libprelude can communicate with Prelude Manager over an UNIX socket (within the same host), encrypted TCP connection (using OpenSSL, preferred for communication with remote host) and unencrypted TCP connection. It should be noted that the most appropriate protocol to be used is chosen automatically (UNIX socket if manager is running on the same host, TCP with SSL otherwise and plaintext TCP if OpenSSL is not available) and the sensor code is independent of the protocol being used. In other words, the library abstracts the communication protocol being used. It also provides convertion between little endian and big endian format as needed.

  In cases when a connection to a manager is lost, the library provides a transparent mechanism which makes periodic reconnection attempts (using exponential backoff). All IDMEF messages which the sensor attempts to send to the manager while the connection is down are stored locally in a spool file and sent immediately after the connection is established again. Alternatively, sensor can be configured to send messages to an alternative manager if communication with the default one fails. One sensor shown in figure 1 is configured this way: if it can't communicate with a local manager, it sends alerts to a top-level manager instead. Sensors may also be configured to send alerts to two or more managers at the time, although it's not a recommended approach for performance reasons.

  The communication protocol also provides sensor authentication mechanisms (via username and password pair for UNIX and plaintext connections or SSL certificates for SSL-protected connections).
- A generic plug-in interface, which allows easy use of dynamically loadable plug-in modules.
- Generic configuration interface, allowing reading of option values from a configuration files and command line arguments.
- Interface to synchronous and asynchronous timers, allowing an user function to be called after a given interval. This feature is commonly used for sending IDMEF *Heartbeat* messages.
- Generic asynchronous queuing interface.



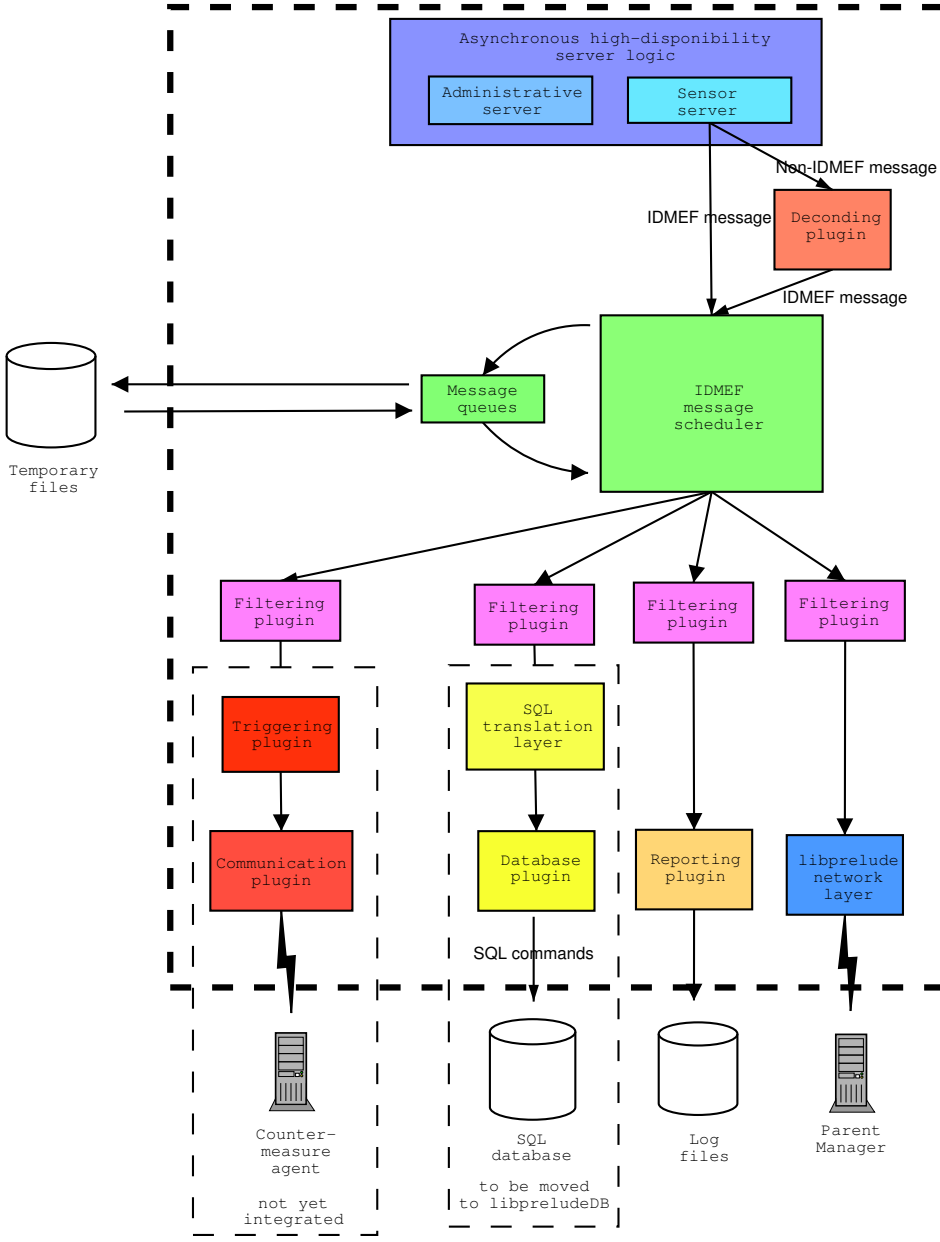


Figure 2 Structure of Prelude Manager

4.2. Prelude Manager

Prelude Manager is the component responsible for processing alerts received from sensors. Its structure is shown in figure 2.

On startup, every sensor connects to a multithreaded server in the Prelude Manager (via plaintext TCP, SSL or UNIX connection) and passes it IDMEF messages (for both *Alert* and *Heartbeat* events). When received, the message is being picked up by the scheduler and placed in one of three queues depending on its priority. The scheduler is capable of queuing lower-priority messages on disk in order to avoid memory exhaustion when dealing with a large message flow.

It must be noted that the sensor does not necessarily need to send IDMEF messages to Manager. A sensor may also use its own data format, providing that the manager is equipped with a corresponding *decoding plugin*, which will translate received data into IDMEF format. The constructed IDMEF message is then passed to scheduler and further processed normally. Such a solution is used in communication between Prelude NIDS and Prelude Manager in order to avoid time consuming IDMEF message generation from within a time critical NIDS application.

The message is subsequently passed to one or more output plugins, which perform alert logging. As of now the following plugins are available:

- output plugins producing human-readable text file and IDMEF XML file
- database plugins (for MySQL and PostgreSQL) storing the messages in SQL database.

The last option allows the use of frontend application (see section 4.6) to view or analyze gathered data.

A manager can also be configured to relay received alerts to one or more managers, called *parent managers*. This capability allows creation of multi-tier systems with different managers responsible for different segments of the supervised system.

Since all IDMEF messages are passed to all output facilities (i.e. all active reporting plugins, a database plugin and parent manager), a *filter plugin* may be placed before a given output facility in order to restrict the set of alerts a facility receives. A message is first passed to a corresponding filter plugin, which performs its evaluation and decides if it should be passed to a given output facility or not. An example use would be a manager configured to log all alerts locally and pass only high-severity alerts to a parent manager.

Prelude Manager is also equipped with an administrative server, allowing connections from an administrative console. Upcoming implementation of the console itself, as well as extensions to the protocol used for communications with sensors will allow on-the-fly reconfiguration of sensors based on user commands sent from the console.

In future versions, Prelude Manager will be equipped also with countermeasure *triggering* and *communication plugins*, forming a part of the active response subsystem. A more detailed description is given in section 5.1.

4.3. Prelude NIDS, the network-based sensor

The Prelude NIDS component implements network-based detection functionality.

The core element of the Prelude NIDS is a packet sniffer that captures layer 2 frames from the interface it is configured to listen on. Each frame is then analyzed in order to detect potential malicious activity.

Depending on the type of the layer 3 and 4 protocol used, IP and TCP reassembly operations can be performed, to reconstruct the data stream. Also, various checks are performed in order to determine if the packet will be considered valid by receiver's TCP/IP stack. These operations aim to defeat the techniques an attacker could use in order to avoid detection of the malicious activity. The topic is covered in detail in [4], while a detailed description of how each of these attacks is handled by Prelude NIDS is given in [3].

After this stage, data is passed to protocol-specific plugins, which perform further analysis. Currently, following protocol plugins are available:

HTTP plugin This plugin looks for a HTTP request in the data stream, normalizes it, and decodes escaped characters (standard HTTP escaping, UTF-8 and Unicode escaping) in order to defeat various HTTP-specific evasions (see [5] for more information). The Unicode decoder attemps to mimic the behavior of the Microsoft Internet Information Server in order to detect attacks such as incorrect overlong UTF-8 sequences, and UTF-8 sequences hiding ASCII characters (see [6]).

FTP and Telnet plugin decodes escaped characters in Telnet and FTP connections.

RPC plugin analyses RPC header within the packet and creates a *(function, version, program)* set for further analysis by signature engine.

Next, normalized data is passed to a signature engine. The engine is based on the concept of decision trees, what allows minimizing of the number of test performed. A detailed description of the engine is given in [7]. The engine itself is independent of the signature format being used; reading of signature set is performed by specialized plugins. Currently, a SnortRules plugin is available, which allows use of rulesets from Snort [15] version 1.9.

Additionally, Prelude NIDS is equipped with following detection plugins:

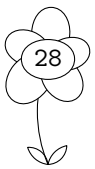
ARPSPOOF PLUGIN allows detection of attacks using ARP protocol. The plugin will emmit an alert if it detects:

- ▷ an ARP request sent to an unicast address instead of broadcast address
- ▷ an ARP request where the Ethernet source address is different from the address in the ARP message
- ▷ an ARP request where the Ethernet destination address is different from the address in the ARP message

Additionally, the plugin allows specifying the IP/MAC address mappings through the *arpwatch* utility and detects ARP messages conflicting with the arpwatch database. [3]

SCANDETECT PLUGIN is responsible for detecting various kinds of portscans. It generates alerts whenever a number of connection attempts within a given time exceeds a given threshold. The values can be set separately for low (below 1024) and high port range.

SHELLCODE PLUGIN, performing polymorphic shellcode detection. In many buffer overflow exploits a long string of NOP opcodes is placed before the actual exploit code. Therefore, a NIDS application is capable of detecting exploitation attempts by looking for long strings of NOP opcodes (for example 0x90 on IA32 or 0x4E71 on M680x0 family) in analyzed traffic. However, as demonstrated by ADMutate tool [8], NOP-equivalent instructions can also be used to achieve the same goal avoiding detection by an IDS looking for strings of NOP opcodes. The shellcode plugin can detect a polymorphic attack using a technique of counting potential NOP-equivalent instructions in a single string, originally described in [9] and implemented in *NIDSfindshellcode* utility [10].



4.4. Prelude LML, the log monitor

Currently the host-based detection in Prelude is achieved by the Prelude Log Monitoring Lackey (LML) program. LML is an UNIX daemon responsible for real-time analysis of system log files. A detailed description of implementation of LML is given in [11].

LML is capable of monitoring local files, as well as running as a server on the standard syslog port (*514/udp*). When monitoring modifications to local files, LML can take advantage of the capabilities of the FAM daemon [12] (if available) in order to shorten the time frame between data being logged to the file by the syslog daemon and processing it (i.e. the time frame during which the attacker which has already gained root privileges could remove or alter contents of the log file to avoid detection). Additionally, the daemon is equipped with a mechanism monitoring file size permitting to detect file deletion or modification.

It should be noted that because of the issue above, it may prefferable to run LML in network server mode along with a standard syslog server logging to files and forwarding log entries to LML daemon. Such configuration also allows many machines to send logs over the network to a single LML daemon on a single host, although could allow the attacker to flood LML server with UDP syslog packets.

After obtaining a log entry (either from a local file or a network socket) LML passes it to its plugins, which perform needed analysis (i.e. matching against a signature set) and generate alerts if necessary. Currently, there are two plugins available: a specialized plugin for interpreting PaX [13] logs, and a general purpose plugin performing matching against a set of signatures constructed using Perl-compatible regular expressions (PCRE). At the time of writing, available signature sets allowed analysis of logs from the following sources:

- Cisco routers
- Exim daemon
- FreeBSD IPFW firewalls
- Linux kernels with grsecurity [14] patch
- Linux Netfilter firewalls
- OpenSSH daemon

```
regex=ipfw: (\d+) Deny (TCP|UDP) ([\d\.]+):(\d+) ([\d\.]+):([\d]+) in via (\w+); \
       class.name=Packet dropped by firewall; \
       impact.completion=failed; \
       impact.type=other; \
       impact.severity=medium; \
       impact.description=Denied incoming packet (rule #$1) $2 $3:$4 -> $5:$6 on $7; \
       source.node.address.address=$3; \
       source.service.port=$4; \
       source.service.protocol=$2; \
       target.node.address.address=$5; \
       target.service.port=$6; \
       target.service.protocol=$2; \
       source.interface=$7;
```

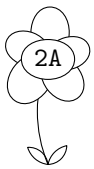
Figure 3 Example LML rule

- ▷ Qpopper daemon
- ▷ ProFTPD daemon
- ▷ vpopmail
- ▷ ZyWall firewalls
- ▷ ZyXEL modems

An example LML rule has been shown in figure 3. It can be seen that the rule is composed from a PCRE signature and specification of values for different fields in IDMEF alert.

4.5. Applications modified to work as Prelude sensors

As mentioned in section 4.1, libprelude's interface allows relatively easy enabling of third-party applications to report to Prelude Manager. At the time of writing, the following software was known to be successfully ported:



libsafe [16] is a shared library which aims towards defeating stack-smashing attacks by substituting its own version of potentially dangerous standard C library calls, such as *strcpy()*. Prelude support, which allows sending alert to Prelude Manager when an exploitation attempt is performed, is included in versions from 2.0-10 on.

systrace [17] is a syscall policy enforcement tool developed by Niels Provos. It has been integrated into OpenBSD, but is also available for NetBSD,

GNU/Linux, FreeBSD and Mac OS X operating systems. The patch allowing reporting policy violations to Prelude Manager is available from `http://www.rstack.org/oudot/prelude/systrace/` .

HONEYD [18] developed by Niels Provos, is a tool allowing creating of virtual hosts on the network, in order to observe the activity of potential attackers aimed at these virtual hosts, thus implementing the concept of *virtual honeypots*. The patch reporting honeypot activity to Prelude Manager is available from `http://www.rstack.org/oudot/prelude/honeypots/`.

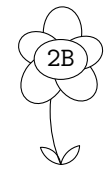
BRO [19] is an IDS developed by Vern Paxson. The patch allowing it to send alerts to Prelude Manager is available from `http://manux.rstack.org/prelude_bro/`.

SNORT [15] is a widely used network intrusion detection system. Despite the fact that Prelude has its own NIDS module (Prelude-NIDS), an experimental patch allowing using Snort as a Prelude sensor is available from `http://www.prelude-ids.org/download/releases/snort-prelude-reporting-patch-0.1.0.tar.gz`

4.6. User frontends

The purpose of the frontend is to allow the user to view logged alerts. Frontends display event information stored in an SQL database by Prelude Manager, and allow performing statistical analysis. Currently, the most popular Prelude frontends are implemented as scripts run by a HTTP server (Apache), and accessible with a Web browser. There are currently two Web frontends available: one written in PHP and one written in Perl. Frontends that work as standalone applications (i.e. do not require running an HTTP server) are under development.

5. Future



5.1. Active Response capabilities

By definition, an intrusion detection system performs detection of potentially dangerous events. It is however natural to think about equipping it with an ability to automatically react to the attack detected.

Such solutions focus on sending spoofed TCP RST or ICMP Unreachable messages in order to tear down an attacker's network connection or performing

a firewall update in order to block communication with attacker's machine. They however suffer from a number of problems, such as attacker's capability to drop RST packet sent by the IDS, or the potential to trick the system into blocking communication with innocent third parties (self-inflicted DoS). A good overview of these issues is presented in [22].

In 2002, an experimental implementation of the active response mechanism was done in Prelude [21]. The subsystem consists of following components:

TRIGGERING PLUGINS in Prelude Manager which analyze received alerts and decide what action should be performed, if any.

COMMUNICATION PLUGINS in Prelude Manager, which are run when the triggering plugin decides it is necessary to perform an action. These plugins are responsible for communication with the device (e.g. a hardware firewall) or the agent software performing the requested action.

AGENTS which perform the requested action (e.g. firewall update).

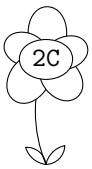
More information about the implementation of active response subsystem in Prelude can be found in [21], [20]. The subsystem is expected to be merged into the main source tree before the next (0.9) major release.

5.2. LIBPRELUDEDB, THE IDMEF DATABASE LIBRARY

As mentioned in section 4.6, both the Prelude Manager and the user frontend communicate directly with a SQL database holding alert data. However this leads to code duplication and dependence of many components on a database structure, resulting in problems when database structure needs to be changed.

A proposed solution is introducing a new shared library, *libpreludeDB*, implementing an abstraction layer between data storage (such as, but not limited to, an SQL database) and any Prelude component wishing to access the event data using a common interface based on IDMEF data model. This will eliminate code duplication as well as allow the use of various database formats and decrease the effort needed to develop the frontend.

For more information on libpreludeDB, see project's homepage [23].



5.3. CORRELATION SUPPORT

An important feature of a modern intrusion detection system is the ability to perform correlation analysis, i.e. find relations between alerts recorded, and

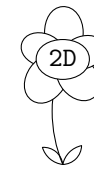
between alerts and network configuration. Currently, the tools are available [25] which correlate alerts recorded in database with results of network assessment obtained using Nessus network scanner [24]. These tools allow to determine whether the attacked network service was vulnerable to the kind of attack used. The tools for performing correlation between alerts are also being developed.

6. Acknowledgements

The author would like to thank Hugh Blemmings and Yoann Vandoorselaere for reviewing and correcting this paper.

7. Bibliography

1. http://www.networkintrusion.co.uk/
2. D. Curry, H. Debar "Intrusion Detection Message Exchange Format Data Model and Extensible Markup Language (XML) Document Type Definition" http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-10.txt
3. Y. Vandoorselaere, L. Oudot "Prelude-IDS, un Système de Détection d'Intrusion hybride opensource", MISC issue 3, July 2002. English version available at http://www.prelude-ids.org/article.php3?id_article=10
4. T. Ptacek, T .Newsham "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection" http://secinf.net/info/ids/idspaper/idspaper.html
5. Rain Forest Puppy, "A look at whisker's anti-IDS tactics", http://www.wiretrip.net/rfp/pages/whitepapers/whiskerids.html
6. Rain Forest Puppy, "IIS %c1%1c bug", BUGTRAQ post, http://www.wiretrip.net/rfp/p/doc.asp/i5/d57.htm
7. J. Brebec "Detèction d'Intrusion – Prelude: Dètection & Signatures" ENSEIRB Bordeaux 2001
8. K2, ADMutate http://www.ktwo.ca/readme.html
9. NGSEC, "Polymorphic Shellcodes vs. Application IDSs", http://www.ngsec.com/docs/whitepapers/ ... ... /polymorphic_shellcodes_vs_app_IDSs.PDF



10. http://www.ngsec.com/downloads/misc/NIDSfindshellcode.tgz

11. A. Launay, P-J. Turpeau "Log Centralization for Prelude NIDS: Prelude Log Monitoring Lackey", ENSEIRB Bordeaux 2002, http://www.rstack.org/oudot/20012002/14b/report.pdf

12. http://oss.sgi.com/projects/fam/

13. http://pageexec.virtualave.net/

14. http://grsecurity.net/

15. http://www.snort.org/

16. http://www.research.avayalabs.com/project/libsafe/

17. http://www.citi.umich.edu/u/provos/systrace/

18. http://www.citi.umich.edu/u/provos/honeyd/

19. http://www.icir.org/vern/bro-info.html

20. K. Zaraska, "IDS Active Response Mechanisms: Countermeasure Subsytem for Prelude IDS", {http://www.prelude-ids.org/download/misc/ ... ... /lsm/2002/slides/krzysztof/lsm.pdf

21. V. Glaume, B. Malguy "Rèassemblage TCP & Contre Mesure au sein de l'IDS Prelude", ENSEIRB Bordeaux 2002

22. J. Larsen, J. Haile "Understanding IDS Active Response Mechanisms" http://online.securityfocus.com/infocus/1540

23. http://mops.uci.agh.edu.pl/~kzaraska/libpreludedb/

24. http://www.nessus.org/

25. http://www.rstack.org/oudot/prelude/correlation/

SELinux w służbie bezpieczeństwu

Grzegorz B. Prokopski

STRESZCZENIE: Security Enhanced Linux jest produktem stworzonym na zamówienie amerykańskiej agencji bezpieczeństwa (NSA). Rozszerza on mechanizmy kontroli dostępu i uprawnień użytkowników w elastyczny i prosty sposób, zwiększając efektywne bezpieczeństwo systemów linuksowych. Podstawowym mechanizmem kontroli dostępu wykorzystywanym przez SELinuksa jest Role Based Access Controls (RBAC), który znacząco różni się od standardowych praw uniksowych, czy ACL-i. Instalacja tego systemu nie jest zbyt skomplikowana, a praktyczne użytkowanie nie zwiększa czasu potrzebnego na administrowanie systemem. Z cała pewnością jest to bardzo ciekawy projekt, który stoi u progu swojej popularności.

DLACZEGO SELINUX?

Dla systemu Linux w ostatnich latach powstało wiele oprogramowania, którego zadaniem jest podnoszenie bezpieczeństwa systemu. Czy zatem istnieje potrzeba tworzenia kolejnego rozwiązania? Na czym polega odmienność i siła SELinuksa? Skąd pochodzi, kto za nim stoi i na jakich zasadach jest dostępny? Jak działa, kto może i kto powinien go używać? Na te i na inne pytania postaram się odpowiedzieć w poniższym artykule traktującym o dość rewolucyjnej, choć czerpiącej ze standardów naturze Security Enhanced Linuksa, czyli Linuksa o wzbogaconym bezpieczeństwie.

CZYM JEST I SKĄD POCHODZI?

Najkrócej mówiąc, SELinux jest systemem z MAC (ang. Mandatory Access Control), czyli obowiązkową kontrolą dostępu, który realizuje politykę RBAC (ang. *Role Based Access Control*), czyli kontroli dostępu opartej na rolach, za pomocą DTAC (ang. Dynamically Typed Access Control) tłumaczonego jako

domenowy system kontroli dostępu. Definicja wydaje się dość skomplikowana; jej dokładniejszym wyjaśnieniem zajmę się w dalszej części artykułu.

Znacznie bardziej zrozumiała jest historia SELinuksa. Prace nad nim są sponsorowane przez amerykańską NSA (ang. *National Security Agency*), a pieniądze te trafiają do pracującej nad projektem SELinuksa Secure Computing Corp., która to firma jest również właścicielem wykorzystywanych w nim patentów. Jednak idea i implementacja, których wynik znamy dziś pod nazwą SELinux nie są wcale młode. Od 1992 roku, w ramach projektu Distributed Trusted Mach (DTMach), rozwijano rozwiązania, które następnie zostały przejęte przez system operacyjny Fluke, dla którego w ramach projektu Flux rozwijano architekturę Flask. Jakkolwiek brzmi to skomplikowanie, to prawdopodobnie najistotniejszym faktem jest, iż to właśnie architekturę Flask zintegrowano z jądrem Linuksa, a powstały projekt ochrzczono imieniem SELinux.

Co kryje się pod maską?

Można wymienić trzy główne elementy składowe SELinuksa:

Jądro – SELinux używa obecnie (nie zawsze tak było) infrastruktury jądra Linuksa znanej pod nazwą LSM (ang. Linux Security Modules – moduły bezpieczeństwa Linuksa), która dostarcza interfejsów pozwalających kontrolować dostęp do obiektów systemu podczas wykonywania działań przez użytkowników (otwarcie pliku, utworzenie katalogu, bindowanie do portu itd.). SELinux podłącza się do tego interfejsu i wymusza w systemie własną politykę bezpieczeństwa. Z punktu widzenia administratora instalującego SELinuksa stanowi on po prostu łatę na jądro.

Zmodyfikowane kluczowe programy – SELinux w większości przypadków pozwala, aby programy działające w systemie nie musiały rozumieć, że system ten nie jest zwykłym Linuksem. Jednak pewne kluczowe dla bezpieczeństwa i w ogóle działania systemu programy muszą zostać rozszerzone o obsługę SELinuksa. Do tych programów należą między innymi: ssh, ls, ps, xdm, login. Administrator systemu musi podczas instalacji albo pobrać zmodyfikowane już wersje używanych programów, albo zaaplikować odpowiednie łaty na źródła używanych programów.

Zasady, czyli Policy – określają prawa dostępu, prawa wykonywania działań w systemie, zachowanie się systemu. Jest to w praktyce najważniejsza część systemu SELinux, gdyż to właśnie te Zasady decydują o skutecznym działaniu całego systemu. W niniejszym artykule poświęcę część miejsca na wyjaśnienie

jak konstrowane są elementy Policy. Administrator systemu generalnie nie powinien być zmuszony do pisania własnych zasad w Policy (chyba, że istnieją pewne niestandardowe wymagania co do kształtu polityki bezpieczeństwa), ale powinien umieć je czasem zmodyfikować do swoich potrzeb, a na pewno powinien rozumieć zasady już istniejące.

Najważniejsze cechy (reklama)

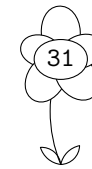
Można powiedzieć, że SELinux jest rozwiązaniem:

- spójnym i kompletnym
  Nie skupia się, jak część innych rozwiązań (np. łat na jądro), na usunięciu pewnych wybranych błędów w polityce bezpieczeństwa, czy problemów wynikającej z przyjętej konstrukcji systemu, np. norm POSIX odnoszących się do własności systemów uniksowych. Zamiast tego SELinux proponuje całościowe rozwiązanie.

- rozszerzalnym i elastycznym
  Administrator systemu może tworzyć w Policy reguły dotyczące nowych pakietów oprogramowania, może też modyfikować reguły istniejące. Dzięki temu można dostosować kształt polityki bezpieczeństwa do istniejących w danym przypadku potrzeb.

- standaryzowanym
  Norma POSIX.6 zawiera opis mechanizmu MAC, który korzysta z etykietowania (ang. labelling) obiektów. SELinux implementuje MAC korzystając właśnie z etykietowania. Ponadto należy bardzo mocno podkreślić, że SELinux, mimo swojej nieco rewolucyjnej natury, świetnie wpisuje się w istniejące standardy systemów uniksowych.

- aktywnie rozwijanym
  Rozwojem SELinuksa zajmuje się opłacana z rządowych funduszy firma komercyjna. Jest to jednak produkt Open Source, a NSA (świadczą o tym jej działania) zależy na jak najszerszym propagowaniu SELinuksa i na jego rzeczywiście otwartym rozwoju. Ponadto opłacani deweloperzy wraz z zapaleńcami, ochotnikami, którzy już przyłączyli się do tego projektu tworzą grupę chętnie pomagającą w problemach związanych z instalacją i użytkowaniem SELinuksa, a to bardzo istotne, szczególnie przy tak nowatorskim projekcie.



Kilka słów o MAC (i ACL)

Ponieważ SELinux jest systemem realizującym MAC, to warto wiedzieć co to dokładnie oznacza. Główna zasada w systemach z MAC stanowi, że użytkownik NIE decyduje o zabezpieczeniach i prawach dostępu do obiektów. Prawa i zabezpieczenia, o których mowa, są definiowane odgórnie, zgodnie z polityką bezpieczeństwa i w przypadku SELinuksa znajdują swoje odbicie w zapisach Policy.

Ponieważ SELinux bywa porównywany z systemami z ACL, czy też (całkowicie błędnie) do systemu z ACL (ang. Access Control Lists), to warto wiedzieć, że:

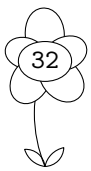
- ACL NIE realizują MAC
- ACL są w sumie bardziej skomplikowane i pracochłonne w zakresie konfiguracji i kontroli poprawności

Można też powiedzieć, iż ACL zajmują się jednostkowymi przypadkami, natomiast, jak się wkrótce okaże, SELinux realizuje MAC przez określanie ogólnych zasad.

ACL-e nie znalazły się w jądrach z serii 2.4 także dlatego, że deweloperzy jądra nie uważają ich za dobry sposób kontrolowania uprawnień użytkowników w systemie. W praktyce znajdują one zastosowanie najczęściej w połączeniu z serwerem Samba, gdyż pozwalają wtedy ustalać uprawnienia do plików z poziomu klientów windowsowych.

Co to jest DTAC?

Skrót DTAC pochodzi od angielskiego Dynamically Typed Access Control, co bywa tłumaczone jako domenowy system kontroli dostępu. Najważniejsze cechy DTAC są następujące:



- każdy obiekt (np. plik, port, urządzenie sieciowe) posiada pojedynczy typ,
- typ obiektu jest wymuszany odgórnymi regułami (nie zależy od decyzji użytkownika, ale od administratora komponującego Policy),
- w systemie można tworzyć niemal dowolną liczbę reguł (rzędu 100 tysięcy) – system jest na to przygotowany,

- dla każdego typu tworzone są zestawy reguł, decydujące o zachowaniu się systemu przy wykonywaniu akcji na obiektach danego typu (więcej na ten temat będzie za chwilę),
- kontrola i testowanie poprawności reguł jest łatwiejsze – IBM opracowuje narzędzie, które pozwala na automatyczną kontrolę spójności i poprawności polityki bezpieczeństwa DTAC.

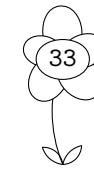
Co to jest RBAC?

Skrót RBAC pochodzi od angielskiego Role Based Access Control, co po polsku oznacza system kontroli dostępu bazujący na rolach. Chociaż pojęcie to nie jest zbyt często używane, to warto wiedzieć, że np. standardowy system uniksowy jest systemem z RBAC. Rzeczywiście, gdy przyjrzymy się bliżej, to widzimy, że mamy tzw. użytkowników systemowych, z których każdy ma prawo wykonywać pewien zakres działań (tzn. realizować pewną rolę w systemie). Drugim elementem kwalifikującym standardowy Unix do systemów z RBAC jest istnienie SUID-ów, za pomocą których można zmienić rolę w ramach której dane działanie jest wykonywane. A jak to się ma do SELinuksa?

- SELinux jest systemem z RBAC, którego zasady są realizowane za pomocą DTAC
- SELinux rozszerza filozofię SUID-ów
- SELinux rozszerza rozumienie ról użytkowników

Ogólnie rzecz biorąc SELinux pozwala w sposób znacznie bardziej elastyczny, dokładniejszy i bardziej dostosowany do konkrentych potrzeb określać i realizować role w ramach systemu.

SELinux – system w systemie



Ponieważ znane już są podstawowe pojęcia, spróbuję lepiej określić w jaki sposób działa SELinux. Przede wszystkim jeden użytkownik w systemie może posiadać wiele ról, przy czym jedna z nich jest rolą domyślną. Każda rola użytkownika posiada zestaw domen dostępu, a jedna z nich jest domyślną dla danej roli. W danej chwili użytkownik może wykonywać jedną rolę w jednej domenie. Mogą następować przełączenia między rolami i domenami. Dla wybranych skrzyżowań domeny i każdego typu definiowane są reguły zachowania się (prawa i inne). Efektywne uprawnienia procesu w systemie zależą więc od

domeny w której w danej chwili działa. SELinux potrafi też współpracować ze standardowym mechanizmem 'capabilities', pozwalając ustalać zestaw 'capabilities' startującego programu zależnie od domeny procesu, który próbuje go uruchomić.

Trzeba jeszcze rozumieć jedną istotną rzecz, czyli to, w jaki sposób reguły SELinuksa współgrają ze standardowym modelem uprawnień UGO (ang. User, Group, Others – Użytkownik, Grupa, Inni) systemu uniksowego. Generalna zasada stanowi, iż SELinux ogranicza uprawnienia do wykonywania zadań w stosunku do systemu bez SELinuksa, natomiast nie nadaje dodatkowych uprawnień, których użytkownik nie miał w niezmodyfikowanym systemie. Jest to podejście praktycznie bardzo poprawne, gdyż w przypadku błędów w polityce bezpieczeństwa SELinuksa, w najgorszym wypadku, gdyby nawet Policy SELinuksa pozwalało wszystkim na wszystko, otrzymalibyśmy zwykły system uniksowy, którego zabezpieczenia wcale nie są takie złe (przecież używamy ich na codzień!).

Security Context procesu i obiektu

Konstrukcję SC, czyli kontekstu bezpieczeństwa procesu, najlepiej wyjaśnić na przykładzie. Po zalogowaniu się do systemu z SELinuksem przez użytkownika greg, jego proces powłoki (ang. shell) będzie działał z następującym kontekstem bezpieczeństwa: `greg:user_r:user_t`, gdzie:

- `greg` – identyfikator użytkownika uniksowego,
- `user_r` – rola domyślna użytkowników po zalogowaniu się,
- `user_t` – domena domyślna roli `user_r`.

Tak naprawdę obiekty w systemie także posiadają pełny SC, np. można zdefiniować SC portu TCP jako: `system_u:object_r:http_port_t`, przy czym znacząca jest jedynie część ostatnia: `http_port_t`, która w przypadku obiektów nazywana jest typem, a nie domeną jak to ma miejsce w przypadku procesów. Należy też mieć jasność co do tego, że z danej roli może korzystać (zgodnie z zapisami w Policy) wielu użytkowników uniksowych, tak jak do jednej domeny może mieć dostęp wiele ról.

Główne rodzaje reguł SELinuksa

Reguły SELinuksa są definiowane dla par domena i typ. Można wyróżnić najczęściej używane rodzaje reguł:

- definiowanie operacji dozwolonych, takich jak czytanie, pisanie, tworzenie, wykonywanie, bindowanie (do portu) lub innych,
- dozwolona zmiana roli w ramach użytkownika (i w ogóle lista ról dozwolonych dla użytkownika),
- dozwolona zmiana domeny w ramach roli (i w ogóle lista domen dozwolonych dla roli),
- automatyczna (wymuszona, jak przy SUID-zie) zmiana roli bądź domeny przy wykonywaniu programu, lub przy wykonywaniu operacji na obiekcie (np. automatyczna zmiana typu przy tworzeniu pliku).

W chwili obecnej w standardowej Policy zlikwidowano wszystkie wymuszenia zmiany roli, zastępując je wymuszeniami zmianami domeny.

Przykład klienta IRC i przeglądarki internetowej

Patrząc na historię błędów w klientach IRC pozwalających przejąć atakującemu kontrolę nad maszyną, na której uruchomiono klienta (przynajmniej w ramach uprawnień użytkownika), nie powinien dziwić fakt, iż w standardowej Policy znajdują się zapisy traktujące te programy w sposób specjalny:

- wprowadzono domenę `user_irc_t`,
- przy wykonaniu programu klienta IRC, którego plik wykonywalny także posiada specjalny typ: `irc_exec_t`, następuje wymuszona zmiana domeny w Security Context procesu
- `user:user_r:user_t => user:user_r:user_irc_t`,
- proces działający w domenie `user_irc_t` ma bardzo mocno ograniczone prawa – praktycznie rzecz biorąc ma możliwość pisania tylko do logów i własnych plików konfiguracyjnych,
- należy zauważyć, iż użytkownik i rola nie podlegają modyfikacji (to nie jest SUID).

Podobne niebezpieczeństwa niesie ze sobą używanie przeglądarek internetowych. Są to zwykle duże i skomplikowane programy, co niestety przekłada się na liczbę błędów i wynikających z nich zagrożeń. Dlatego dla nich, w sposób analogiczny, wprowadzono typ plików wykonywalnych `netscape_exec_t`,

a podczas `exec()` pliku o powyższym typie następuje przełączenie procesu do domeny `user_netscape_t`, która ma następujące, ograniczone prawa:

- może tworzyć pliki o typie `user_netscape_t` i `user_netscape_rw_t` (ten typ służy także do oznaczania plików konfiguracyjnych),
- może wykonywać pobrane programy, ale tylko w ramach bieżącej domeny,
- może drukować (ma prawo wykonywania programu `lpr`).

Powyższe ograniczone uprawnienia pozwalają znacząco zmniejszyć rozmiar szkód spowodowanych ewentualnym włamaniem wykorzystującym np. przepełnienie bufora w przeglądarce internetowej.

Role i domeny w służbie demonów

Użycie separacji uprawnień w stosunku do programów uruchamianych przez użytkownika to jedno pole działania SELinuksa. Jednak ze względu na fakt, iż Linux wciąż jest najczęściej stosowany jako system dla serwerów, a także ponieważ to zwykle systemy serwerowe miewają najwyższe wymagania co do jakości polityki bezpieczeństwa, wydaje się, że użycie SELinuksa dla rozdzielenia uprawnień pomiędzy różnorodne serwisy, demony itd. jest drugim i znacznie szerszym polem zastosowań. Bazując na poprzednim przykładzie można już domniemywać, (i tak jest w istocie), iż role i domeny dla procesów serwerowych separują poszczególne demony i, w miarę potrzeb, także same funkcje demonów. Robią to lepiej, niż z użyciem SUID-ów i użytkowników systemowych, gdyż Policy SELinuksa daje większą elastyczność w doborze uprawnień, nie trzeba tworzyć większej liczby użytkowników systemowych by rozdzielić funkcje. Dodatkowo, jeśli proces serwera potrzebuje cały czas do czegoś prawa roota, to tworzenie użytkowników nie ma sensu (i niejednokrotnie autorzy oprogramowania uciekają się wtedy do jeszcze bardziej skomplikowanych rozwiązań, jak chrooty i inne).

Niestety, ze względu na specyfikę rozwiązań wewnątrz różnych pakietów oprogramowania serwerowego, elementy Policy SELinuksa muszą być definiowane specjalnie dla każdego oprogramowania lub rozszerzane w ramach klasy oprogramowania o wpisy potrzebne do poprawnego działania jej konkretnej implementacji.

Serwer Bind jest przykładową implementacją dla klasy oprogramowania Serwery DNS. Potrzebuje on praw roota co najmniej do startu, a częstokroć także do działania (jeśli musi bindować się do podnoszonych podczas jego działania dynamicznych interfejsów). W tym wypadku, niezależnie od tego, czy

działa on z prawami roota, czy też zwykłego użytkownika, będziemy chcieli ograniczyć jego uprawnienia do następujących:

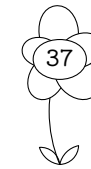
- bindowanie do portu 53 TCP i UDP,
- odczyt plików konfiguracyjnych i plików domen,
- zapis tylko do plików cache.

Dzięki temu, nawet w wypadku włamania do tego demona, gdy działa jako root, praktycznie nie będzie możliwości zniszczenia żadnych danych w systemie (włamywacz będzie mógł co najwyżej odsyłać nieprawdziwe odpowiedzi na zapytania innych serwerów DNS).

Definicje Type Enforcement w SELinux Policy

Definicje TE w Policy dotyczą deklaracji typów, wymuszonej zmiany typu obiektu (np. pliku na dysku), dozwolonej zmiany typu (np. pliku urządzenia konsoli), czy tzw. wektorów dostępu do wszelkiego typu obiektów. Przykłady tych definicji znajdują się na następującym listingu:

```
Listing 1. Definicje Type Enforcement w SELinux Policy
  * deklaracje typów
     type sshd_t, domain, privuser, privrole, privlog, privowner;
         type sshd_exec_t, file_type, exec_type, sysadmfile;
         type sshd_tmp_t, file_type, sysadmfile, tmpfile;
         type sshd_var_run_t, file_type, sysadmfile, pidfile;
  * definicje wymuszonych przejść (makra)
domain_auto_trans(initrc_t, sshd_exec_t, sshd_t)
file_type_auto_trans(sshd_t, tmp_t, sshd_tmp_t)
domain_auto_trans(sshd_t, shell_exec_t, user_t)
  * definicje dozwolonych przejść (bez makr i z makrem)
type_change user_t tty_device_t:chr_file user_tty_device_t;
type_change sysadm_t tty_device_t:chr_file sysadm_tty_device_t;
type_change user_t sshd_devpts_t:chr_file user_devpts_t;
type_change sysadm_t sshd_devpts_t:chr_file sysadm_devpts_t;
domain_trans(sshd_t, shell_exec_t, sysadm_t)
  * definicje wektorów dostępu
allow sshd_t sshd_exec_t:file { read execute entrypoint };
allow sshd_t sshd_tmp_t:file { create read write getattr setattr link unlink rename };
allow sshd_t user_t:process transition;
```



Typy, które będą używane przez inne reguły w Policy, muszą zostać najpierw zdefiniowane. W pierwszej części listingu widać definicje typów: domeny `sshd_t`

(główna domena używana przez programy z pakietu SSH) i typy plików używane przez programy działające w tej domenie. Dzięki tym dodatkowym typom można będzie rozdzielić np. uprawnienia do plików tymczasowych SSH (typ: `sshd_tmp_t`) od innych plików tymczasowych (zwykle mających typ `tmp_t`).

W drugiej części listingu widać definicje wymuszonych przejść. W tym wypadku definicja jest bardziej czytelna, dzięki zastosowaniu makr, np. `file_type_auto_trans`. Mechanizm makr w SELinux Policy w znakomity sposób ułatwia proces jej tworzenia i modyfikacji. W tym wypadku `file_type_auto_trans` mówi, że gdy proces w domenie `sshd_t` otworzy (np. jednocześnie go tworząc) plik, który normalnie miałby domenę `tmp_t`, to nastąpi automatyczna zmiana domeny pliku na `sshd_tmp_t`. Z kolei pierwsze makro `domain_auto_trans` mówi, iż jeżeli proces działający w domenie `initrc_t` (a w takiej domenie są uruchamiane wszystkie skrypty w `/etc/init.d/`) uruchomi program z pliku o typie `sshd_exec_t`, to nowouruchomiony program będzie działał w domenie `sshd_t`. Taka deklaracja (i specjalny typ `*_exec_t`) jest konieczna praktycznie dla każdego pakietu oprogramowania uruchamianego w `/etc/init.d/`.

W trzeciej części listingu umieszczone są zezwolenia na zmianę typu obiektu z jednego na inny dokonywaną przez działający w pewnej domenie program. Na przykład program działający w domenie `user_t` będzie miał prawo zmienić typ pliku urządzenia znakowego z `tty_device_t` na `user_tty_device_t`. Widać tutaj, że w szczególny sposób potraktowane są konsole, na których ma działać użytkownik w domenie `sysadm_t` – będzie on mógł zmienić typ pliku urządzenia znakowego (konsoli, z której ma zamiar korzystać) na `sysadm_tty_device_t`. Ma to uniemożliwić ewentualne podsłuchiwanie lub modyfikowanie komunikacji dokonywanej z użyciem tego urządzenia.

W ostatniej części listingu zawarte są szczegółowe informacje o tym, co proces działający w danej domenie może zrobić z obiektem danego typu.

Definicje Role Based Access Control w SELinux Policy

Drugim rodzajem elmentów w Policy SELinuksa są wpisy odnoszące się do mechanizmu RBAC. Ich przykłady zostały umieszczone na poniższym listingu:

Listing 2. Definicje Role Based Access Control w SELinux Policy

```
* definicje ról
   role system_r types { kernel_t initrc_t getty_t klogd_t };
       role user_r types { user_t user_netscape_t user_irc_t };
       role sysadm_r types { sysadm_t run_init_t };
* definicje dozwolonych przejść
```

```
allow system_r { user_r sysadm_r };
allow user_r sysadm_r;
allow sysadm_r system_r;
  * definicje rozpoznawanych użytkowników
user system_u roles system_r;
user root roles { user_r sysadm_r };
user jdoe roles user_r;
```

Ponieważ, jak nazwa głosi, jest to system oparty na rolach, to nie może zabraknąć definicji ról. W tym wypadku definiujemy również to, z jakich domen kiedykolwiek dana rola może korzystać. Na przykład, dla roli `user_r`, dostępna jest podstawowa, domyślna domena `user_t`, a także pomocnicze domeny `user_netscape_t` i `user_irc_t`, które zostały omówione wcześniej.

W pewnych okolicznościach przydatna jest możliwość przełączania się między rolami na życzenie (służy do tego program newrole). Przykłady dozwolonych przejść miedzy rolami zawarte są w drugiej części listingu.

Generalnie rzecz biorąc system RBAC jest mocno niezależny od kont użytkowników uniksowych. Jednak w wielu przypadkach pożądane jest powiązanie konkretnych użytkowników z zestawami ról, które są dla nich dozwolone. Na załączonym przykładzie widać np., że użytkownik root domyślnie występuje w roli `user_r`, chociaż ma również dostęp do roli `sysadm_r`.

Security Context obiektów sieciowych

Ciekawą możliwością jest definiowanie Security Context dla obiektów systemu związanych z siecią. Ponadto w jasny sposób pokazuje to, że w rzeczywistości obiekty w systemie posiadają pełny SC pomimo, że używany jest tylko typ. W pierwszej części poniższego listingu widać, iż zdefiniowano typ `http_port_t`, który będą posiadały porty TCP o numerach 80 i 8080. Dzięki temu w innych elemenach Policy będzie można zdefiniować, że dostęp do tych portów, do ich typu (np. możliwość bindowania się do nich) mają tylko procesy z wybranych domen. Będzie to np. domena `httpd_t`, w której będzie działał serwer WWW uruchomiony z binarium o typie `httpd_exec_t`, dla którego będzie zdefiniowane wymuszone przejście podczas uruchomienia, do domeny `httpd_t` właśnie. W podobny sposób można regulować prawa do interfejsów sieciowych.

Listing 3. Definicje kontekstów bezpieczeństwa (SC) obiektów systemu

```
  * definicje kontekstów sieciowych
       + porty
          portcon tcp 80 system_u:object_r:http_port_t
              portcon tcp 8080 system_u:object_r:http_port_t
```

```
      + interfejsy sieciowe
         netifcon eth0 system_u:object_r:netif_eth0_t system_u:object_r:netmsg_eth0_t
             netifcon eth1 system_u:object_r:netif_eth1_t system_u:object_r:netmsg_eth1_t

  * definicje domyślnych kontekstów plików

/home system_u:object_r:home_root_t
    /home/[^/]+ -d system_u:object_r:user_home_dir_t
    /home/[^/]+/.+ system_u:object_r:user_home_t
    /home/[^/]+/.ssh(/.*)? system_u:object_r:user_home_ssh_t
```

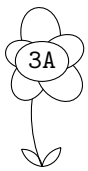
Domyślne Security Contexts plików

Wiadomo, iż każdy plik w systemie plików, będąc obiektem, ma nadany SC, a w nim typ. Przyjrzyjmy się tej kwestii nieco bliżej.

SC plików (a przede wszystkim ich typy) nie są nadawane w sposób automatyczny w momencie uruchomienia systemu z jądrem SELinuksa. Muszą one być nadane w procesie etykietowania (ang. *labelling*), który musi zostać przeprowadzony wcześniej, zanim zaczniemy używać systemu z SELinuksem. Pytanie jakie należy sobie zadać, jest następujące: skąd wiadomo, jaki typ powinien mieć dany plik?

Gdybyśmy zainstalowali źródła SELinuksa, odpowiedź możnaby znaleźć w pliku o nazwie `file_contexts`, stanowiącym składową Policy. Zawiera on definicje, których przykład znajduje się w drugiej części listingu 3.

Informują one program przeprowadzający etykietowanie jakie powinien nadać typy (a dokładniej konteksty bezpieczeństwa) poszczególnym plikom w systemie. W podanym przykładzie widać, że katalog `/home` otrzyma typ `home_root_t`, katalogi użytkowników - `user_home_dir_t`, a znajdujące się w nich pliki i katalogi – typ `user_home_t`. W sposób szczególny potraktowane zostaną pliki używane przez SSH, a znajdujące się w katalogach domowych użytkowników. Otrzymają one własny typ `user_home_ssh_t`. Podobne definicje istnieją dla katalogów używanych przez np. klientów IRC, gdzie trzymane są ich logi i konfiguracja.



Instalacja SELinuksa

Mając już podstawową wiedzę na temat SELinuksa – najwyższy czas przystąpić do jej praktycznego wdrożenia! Aby móc używać SELinuksa trzeba zgromadzić potrzebne elementy:

- łaty na jądro z LSM i SELinuksem,
- łaty na pewne istotne programy w systemie (login, ssh, ls, ps, xdm,...),
- zestaw programów narzędziowych specyficznych dla SELinuksa (chcon i inne).

Zamiast kłopotać się z łatami można ze strony NSA pobrać pełne źródła zmodyfikowanego jądra i zmodyfikowanych programów systemowych. Jeżeli korzystamy z dystrybucji Debiana (stabilnej, bądź niestabilnej), to są dostępne źródła APT zawierające wszystko, co jest potrzebne do instalacji i użytkowania SELinuksa.

W załatanym jądrze należy włączyć następujące opcje:

```
* Networking Options
     + [Y] Network Packet Filtering
* Security Options
     + [Y] Enable different security models
     + [Y] Capabilities Support
     + [Y] NSA SELinux Support
     + [Y] NSA SELinux Development Support
```

Ponieważ i tak musimy kompilować jądro samodzielnie najlepiej nie korzystać z obrazu initrd. W przeciwnym wypadku, po każdej zmianie Policy, nową będzie trzeba umieścić również w obrazie initrd, a to jest zawsze kłopotliwe i czasochłonne.

Po instalacji jądra i instalacji programów użytkowych zmodyfikowanych do potrzeb SELinuksa potrzeba jeszcze wykonać operację etykietowania. Jednak przed jej przeprowadzeniem należy sprawdzić, czy ścieżki podane w pliki `file_contexts` są odpowiednie dla naszej dystrybucji. Jeżeli korzystamy ze źródeł pobranych ze strony NSA, to domyślnie wpisy są odpowiednie dla dystrybucji RedHat. Jeżeli korzystamy z Debiana i pobraliśmy pakiety z odpowiedniego źródła APT, to są one już odpowiednio poprawione. Użytkownicy pozostałych dystrybucji powinni koniecznie sprawdzić zawartość tego pliku. Potem można już wykonać operację etykietowania.

W dystrybucji SELinuksa znajduje się plik Makefile, który zawiera następujące cele:

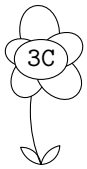
- `install` – kompilacja i instalacja Policy,
- `load` – kompilacja, instalacja i załadowanie Policy,
- `reload` – kompilacja, instalacja i załadowanie lub przeładowanie Policy,
- `relabel` – reetykietowanie systemu plików (FCs),
- `policy` – lokalna kompilacja Policy do testów.

Operacja etykietowania może trwać dość długo, ponieważ wymaga operacji na każdym z plików w lokalnych systemach plików. Można zauważyć, że w katalogu głównym każdej partycji powstał katalog o nazwie `...security` zawierający kilka plików binarnych z dodatkowymi informacjami o każdym z plików na partycji. Po zakończeniu całej operacji, mając zainstalowane jądro z SELinuksem i programy do niego dostosowane, można przystąpić do testów praktycznych.

Pierwszy start

Nawet jeśli wszystko zostało wykonane z należyta starannością podczas pierwszego startu z pewnością pojawi się wiele komunikatów o błędach zgłaszanych przez rozszerzenie SELinuksa. Mimo to system powinien w całości wystartować (dlaczego – to za chwilę). Przykładowy (jeden!) komunikat o błędzie:

```
avc: denied { write } for pid=1112 exe=/bin/login path=/dev/log dev=03:01 ino=15
    scontext=system_u:system_r:local_login_t tcontext=system_u:object_r:device_t
    tclass=sock_file
```



Oznacza on, że proces `/bin/login` posiadający SC `system_u:system_r:local_login_t` próbował zapisywać do gniazda `/dev/log` o typie `device_t`. Komunikat ten wystąpił w rzeczywistej sytuacji i wynikał z niedostosowania Policy do używanego przeze mnie demona syslog-ng. Po dodaniu do Policy dwóch linijek (podanych mi przez Russella Cokera, dewelopera Debiana i SELinuksa) problem ustąpił.

Dzięki zaznaczeniu podczas konfiguracji jądra opcji NSA SELinux Development Support SELinux startuje w trybie permissive, w którym wszystkie błędy wynikające z zabronionego dostępu są zgłaszane, ale sam dostęp do obiektów nie jest zabraniany. System działa wtedy tak, jak normalny Linux, ale zgłasza

błędy. Istnieje kilka metod na przełączenie do trybu enforcing, kiedy to SELinux rzeczywiście realizuje zasady zawarte w Policy:

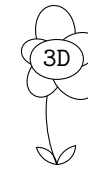
- wyłączenie NSA SELinux Development Support,
- przełączenie poleceniem `avc_toggle` (jeśli zezwala na to Policy – można np. nie pozwolić na powrót do trybu permissive),
- podanie jądru parametru `permissive=1`.

System można startować w trybie enforcing dopiero po upewnieniu się, że zasady Policy są odpowiednio dostosowane i że system będzie w stanie działać tym trybie. Nie należy próbować elemininować wszystkich zgłaszanych błędów, gdyż nie wszystkie one są krytyczne i czasami wynikają z w pełni zamierzonej polityki bezpieczeństwa.

Należy w szczególności zwrócić uwagę na procesy działające (lub próbujące działać) w domenie `initrc_t` (`ps -e --context` pokazuje konteksty procesów). Oznacza to, że najprawdopodobniej dla pakietu oprogramowania do którego należy dany program brakuje odpowiednich wpisów w Policy i podczas startu ze skryptów `/etc/init.d/` SC nie jest zmieniany na właściwy dla danego pakietu oprogramowania. Należy skontrolować `file_contexts` i sprawdzić, czy przypadkiem ścieżki nie są nieprawidłowe. Jeżeli dany pakiet oprogramowania w ogóle nie jest objęty wpisami w Policy, najlepiej zwrócić się o pomoc w ich napisaniu do deweloperów SELinuksa. Pewną pomocą może też być skrypt `scripts/newrules.pl`, który stara się generować nowe zasady allow na podstawie zgłaszanych błędów.

Po starcie systemu będzie można się zalogować. Jeżeli spróbujemy zalogować się jako root z lokalnej konsoli, to będziemy mieli bezpośredni dostęp do roli `sysadm_r` (która w odróżnieniu od domyślnej `user_r` daje rzeczywiste możliwości administrowania systemem). Fragment ekranu logowania użytkownka root wygląda tak:

```
Your default context is root:user_r:user_t
    Do you want to choose a different one? [n]y
    [1] root:user_r:user_t
    [2] root:sysadm_r:sysadm_t
    Enter number of choice: 2
```



Rolę można też zmienić później (w ramach dozwolonych przez Policy) za pomocą polecenia `newrole -r sysadm_r`. Dostęp do roli `sysadm_r` jest zwykle również zabezpieczony drugim hasłem, innym niż hasło roota.

Przydatne informacje

Jeżeli korzystamy z któregoś z graficznych menedżerów logowania, jak `xdm`, `gdm` czy `kdm`, to musi on być załatany aby działał pod SELinuksem. Jeżeli nie chcemy go łatać, to zawsze można użyć startx, startując konsolę graficzną z konsoli tekstowej (login już wykonał pracę zmiany domeny, którą inaczej musiałby wykonać xdm).

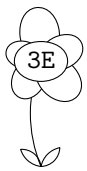
Dobrze jest wiedzieć, czy działamy w trybie `enforcing`, czy `permissive`. Można to sprawdzić poleceniem `avc_enforcing`.

Demony, które w trybie `permissive` startują w domenie `initrc_t` prawie na pewno nie będą poprawnie działać w trybie `enforcing`. Trzeba poprawić Policy, aby działały we własnych domenach.

Przydatne polecenia, to `ps -e --context` i `ls --context`. Pierwsze z nich pozwala sprawdzić z jakimi SC działają procesy, a drugie z nich jakie typy mają pliki. Do celów testowych można zmieniać SC plików za pomocą polecenia chcon. Należy pamiętać, że jeśli nie zostaną dodane odpowiednie wpisy do `file_contexts` zmiana ta zostanie zniwelowana przy kolejnym etykietowaniu.

Jeżeli przełączamy się między jądrem skompilowanym z SELinuksem, a jądrem bez niego, to przed ponownym włączeniem jądra z SELinuksem należy ponownie przeprowadzić etykietowanie, ponieważ w międzyczasie mogły powstać pliki nie posiadające nadanego SC. Chociaż nic nie stoi na przeszkodzie, aby cały czas używać jądra z wkompilowanym SELinuksem, ale w trybie permissive.

Policy jest komponowane z wielu elementów, z których każdy odpowiada za obsługę zazwyczaj jednego pakietu oprogramowania, np. serwera DNS. Jednak w większości przypadków w naszym systemie nie ma zainstalowanych wszystkich pakietów oprogramowania, dla których są dostępne elementy Policy. Dlatego np. podczas instalacji pakietów dla Debiana zostaniemy zapytani, których elementów Policy chcemy rzeczywiście używać. Używając wszystkich dostępnych będziemy mieli około 200 tysięcy zasad. Być może warto tę liczbę zmniejszyć do kilkudziesięciu tysięcy rzeczywiście potrzebnych.



Berlin, patenty i ogólnodostępny root

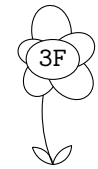
Muszę w tym miejscu zmartwić fanów konsoli graficznej. Niestety SELinux nie jest w stanie w znaczący sposób poprawić problemów z bezpieczeństwem wynikających z obecnego standardu X protokołu, a co za tym idzie – X window. Problem polega na tym, iż nawet jeśli nasz xterm, klient IRC czy przeglądarka internetowa działają w osobnych domenach, to nadal wszystkie te programy

mają możliwość podglądania zawartości okien innych programów i podglądania klawiatury. Dlatego też błąd w kliencie IRC nadal wystawia nasz system na niebezpieczeństwo. Gdybyśmy np. chcieli z xterminala zalogować się gdzieś, atakujący będzie mógł poznać nasze hasło. Być może w przyszłych wersjach X protokołu zostaną wprowadzone np. dwa poziomy zaufania do programów – niższy, bez wyżej wymienionych uprawnień i wyższy, taki jak dotychczasowy. Jednak nacisk na takie zmiany wydaje się być niewielki. Projekt Berlin, alternatywnego protokołu i serwera, który zawierał między innymi elementy bardziej zaawansowanej polityki bezpieczeństwa niestety nie porwał za sobą rzesz fanów, czy deweloperów i nie zyskał popularności, a co za tym idzie, praktycznie nie jest używany.

W sprawie patentów sytuacja wygląda znacznie bardziej pozytywnie. W tej chwili NSA finansuje rozwój SELinuksa, ale patenty na niego posiada komercyjna firma. Może się rodzić podejrzenie, czy firma ta nie zechce w pewnej chwili pieniędzy za użytkowanie oprogramowania korzystającego z opatetnowanych przez nią rozwiązań. Na całe szczęście wydaje się to w tej chwili mało prawdopodobne. NSA wcale nie ma zamiaru np. wykupować tych patentów (co stworzyło by niemiły precedens), ani też Secure Computing Corp. nie bardzo chce walczyć z potężną agencją rządową (bo to ona byłaby pierwszym celem). W przypadku SELinuksa NSA zdaje się całkowicie popierać idee Open Source i Free Software, możemy więc spać spokojnie.

Ciekawy pokaz możliwości SELinuksa przeprowadził Russell Coker, deweloper Debiana, a także jeden z nielicznych nieopłacanych przez nikogo deweloperów SELinuksa. Udostępnił on w Internecie maszynę z publicznie znanym hasłem roota. Okazało się, że będąc rootem można było np. wykonywać chroot, ale praktycznie niczego w systemie nie dało się popsuć. Zastosowane wpisy w Policy nie były wcale bardziej restrykcyjne od standardowych, zostały wręcz w pewnych miejscach złagodzone, np. tak, by każdy mógł odczytać logi systemowe i wiedzieć, dlaczego nie mógł wykonać danego działania. Myślę, że dobrze świadczy to o sile pomysłu SELinuksa.

I co dalej? Używać!



Russell Coker, na pytanie o zakres zastosowania SELinuksa odpowiedział, że stosuje go wszędzie; zarówno na swoich stacjach roboczych, laptopie, jak i na serwerach. Ja jednak myślę, że SELinux przede wszystkim powinien być i będzie szczególnie często używany w systemach serwerowych, gdzie wymagania dotyczące bezpieczeństwa są niejednokrotnie bardzo wysokie.

W chwili obecnej SELinux wychodzi z ukrycia. Rysują się następujące możliwości rozwoju i upowszechnienia tej technologii:

- obecność w oficjalnym jądrze 2.6 – LSM już są obecne w gałęzi 2.5, a SELinux jest (objętościowo, w porównaniu do LSM) jedynie niewielim dodatkiem, który NSA chce koniecznie zintegrować,
- rozbudowanie listy pakietów oprogramowania obsługiwanych przez standardową Policy (tak, aby administrator nie musiał pisać reguł),
- integracja z dystrybucjami – najlepiej tutaj radzi sobie Debian, w którym dzięki wysiłkom Russella Cokera część programów już potrafi radzić sobie z SELinuksem, a w przyszłości SELinux ma stać się standardową (choć opcjonalną) częścią dystrybucji Debiana.

Nie należy jednak ulegać złudzeniu, że technologia ta dopiero będzie zasługiwała na zainteresowanie. Fakty są takie, iż już w chwili obecnej SELinux jest w pełni sprawnym, działającym, przetestowanym, elastycznym i potężnym narzędziem podwyższania bezpieczeństwa systemów linuksowych. Posiada sprawne zaplecze aktywnych deweloperów, wkrótce będzie zintegrowany ze standardowym jądrem i popularnymi dystrybucjami. Z pewnością dużym błędem byłoby zwlekać z poznaniem tak fascynującej i użytecznej technologii!

Słowniczek

- sc RBAC – system kontroli dostępu oparty na rolach
- sc DTAC – mechanizm (niskopoziomowy) definiowania zasad dostępu używany przez SELinuksa do realizacji RBAC
- sc typ – atrybut każdego obiektu dostępu używany jako jedyny do określenia praw dostępu do niego z danej domeny
- sc użytkownik – odpowiada użytkownikowi w sensie UNIXowym, jednak w SELinuksie jeden użytkownik może mieć przypisanych wiele ról
- sc rola – jedna z funkcji wykonywanych przez użytkowników, do jej skutecznego wykonania może być potrzeba praw do jednej lub wielu domen
- sc domena – definiuje zestaw praw do obiektów o wskazanych typach

- sc policy – zestaw reguł w systemie SELinux
- sc security context – kontekst bezpieczeństwa, dot. zarówno obiektów (dla nich znaczenie ma tylko typ), jak i procesów

Epoll – rozwiązanie problemu wielu tysięcy jednoczesnych połączeń TCP/IP

Marcin Gozdalik <gozdal@gozdal.eu.org>

STRESZCZENIE: Łatka `epoll` (autorstwa Davide Libenzi) dostępna jest dla jąder serii 2.4 i jest zintegrowana z rozwojową wersją 2.5. Stanowi ona rozwiązanie problemu znanego w literaturze angielskojęzycznej jako problem C10K, czyli obsługi co najmniej 10000 połączeń TCP/IP w tej samej chwili. Mechanizm oferowany przez tę łatkę posłużył do zbudowania bardzo wydajnego serwera proxy zdolnego obsłużyć co najmniej 20 razy więcej połączeń niż wymagane 10000. Dodatkowo chciałbym pokazać jak `epoll` wypada na tle mechanizmów wbudowanych w Windows 2000 Server.

W dalszej części opisałem dokładniej problem C10K, jego dostępne rozwiązania w różnych systemach operacyjnych wraz z ich wadami. Następnie zawarłem postulaty jakie powinno spełniać rozwiązanie C10K aby uniknąć wad poprzedników oraz opis łatek z rodziny `epoll`: `/dev/epoll` oraz `sys_epoll` – ich historii, pomysłu za nimi stojącego, sposobu obsługi z punktu widzenia użytkownika oraz krótki opis implementacji. Na końcu zamieściłem opis konkretnego zastosowania, w którym `epoll` spisuje się znakomicie, kilkukrotnie przewyższając (w sensie liczby jednocześnie obsługiwanych połączeń TCP/IP) rozwiązania zawarte w Windows 2000 Server.

OPIS PROBLEMU

Coraz więcej ludzi ma dostęp do Internetu; odwiedzają oni strony internetowe, sprawdzają pocztę, czy rozmawiają przez programy typu IM (ang. *Instant Messenger*). Jak obsłużyć ciągle rosnącą rzeszę użytkowników? Można kupować coraz więcej komputerów (co ma w wielu przypadkach przyjemną cechę zwiększania niezawodności całego systemu, ale jednocześnie dosyć mocno bije po kieszeni) lub zmienić oprogramowanie serwera tak, aby mogło obsłużyć więcej klientów.

Najjaskrawszym przykładem potrzeby obsługiwania wielu tysięcy połączeń TCP/IP jest oprogramowanie serwera dla usługi typu IM. Nie jest tutaj potrzebna ani wielka moc obliczeniowa (serwer jedynie odbiera wiadomości i przesyła je dalej) ani szybkie dyski (znakomita większość wiadomości jest przesyłana od razu i żyje tylko krótką chwilę w RAM-ie). Najbardziej potrzebna jest dobra implementacja TCP/IP: skalowalna, stabilna, zdolna obsłużyć wiele tysięcy połączeń, oszczędnie gospodarująca pamięcią operacyjną, zgodna ze standardami i oferująca użytkownikowi wygodny i nieograniczający interfejs.

Nazwa C10K jest skrótem od Connections 10K, czyli 10 tysięcy połączeń. Nie wiem kiedy ukuty został ów skrót (niewykluczone że stworzył go Dan Kegel), ale zapewne 10000 połączeń było wtedy astronomiczną i nieosiągalną liczbą. Trzy tendencyjnie wybrane cytaty z września 1999:

Arjan van de Ven[1]

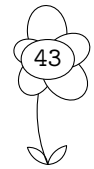
> *SIGIO would be a nice way of knowing which connection has some data, but at 15000 connections, memory requirements might lead to the conclusion that you need some sort of "private tcp/ip stack just for this cause" doing everything at interrupt-level. This, of course, would not be nothing for Linux, and the result probably isn't "Linux" as we know it, but some (hardware) vendors do such souped-up stuff for their server-boxes*

Alan Cox[2]:

> *For 15000 connections you will need about 20K a connection in kernel space too – so thats another 300Mb.*

Doug Royer[3]:

> *I just left Sun Microsystems where I was the architect for the Sun calendar server. I spend several months working on those kinds of issues.*
>
> *Solaris 2.6 and above can handle 100,000 connections if the server has plenty of memory and is properly configured.*
>
> *The test was done by having clients connect, the server simply echoed any characters back the same connection while at the same time the server accepted new connections. When we hit 100,000 simultaneous connections we stopped testing.*
>
> *Often the application running has more of an effect on the system performance than a Solaris file descriptor limitation.*



[1] http://www.uwsg.iu.edu/hypermail/linux/kernel/9909.1/0271.html
[2] http://www.uwsg.iu.edu/hypermail/linux/kernel/9909.1/0287.html
[3] http://lists.fsck.com/lists/impp/9909/msg00040.html

Do not try this on Solaris versions pre-2.6. It will seem to work, however you might trash the OS memory if the application is running as root as pre-2.6 versions assumed that 1024 was a hard limit. It can work, it often does not on pre-2.6.

Dostępne rozwiązania

tradycyjny `poll/select`

Za pomocą standardowego wywołania `read/recv/recfrom` proces może czytać tylko z jednego gniazda (ang. *socket*). Aby serwer mógł obsłużyć więcej niż jednego klienta na raz w *niksach wprowadzono wywołania systemowe `poll` oraz `select`. Pozwalają one procesowi przekazać do jądra zbiór deskryptorów plików, które znajdują się w polu zainteresowania procesu wraz z operacjami jakie proces chciałby wykonać na danym deskryptorze (read i/lub write). Wykonanie procesu zostaje wtedy zawieszone aż do czasu gdy którykolwiek z deskryptorów (a dokładniej: plik związany z deskryptorem) jest w stanie pozwalającym na wykonanie zadanej operacji bez blokowania (jeżeli istnieje co najmniej jeden taki deskryptor w momencie wykonywania `poll/select` proces oczywiście nie jest zawieszany). Proces jest wznawiany gdy pojawi się co najmniej jeden taki deskryptor a informacja o wszystkich deskryptorach na których można wykonać żądane operacje jest przekazywana z powrotem procesowi. `poll` różni się od `select` jedynie sposobem przekazywania parametrów (w Linuksie implementacja obydwu tych funkcji jest wspólna), a co za tym idzie ograniczeniami na liczbę obsługiwanych na raz deskryptorów: dla `select` wynosi ona `FD_SET_SIZE`, dla `poll` jest nieograniczona.

Należy zaznaczyć, że zwrócenie przez jądro informacji o możliwości wykonania operacji odczytu lub zapisu na deskryptorze (bez zawieszenia wykonywania procesu) jest jedynie wskazówką dla procesu. Pomiędzy sprawdzeniem aktualnego stanu przez jądro a wykonaniem operacji przez proces mija trochę czasu i wiele w tym czasie może się wydarzyć (np. połączenie sieciowe może zostać zerwane). Dlatego proces nie może nic zakładać na temat stanu deskryptora i musi wykonywać wszystkie operacje w trybie nieblokującym, będąc przygotowanym na obsługę dowolnego błędu.

Funkcje `poll` oraz `select`, jeżeli zasygnalizowały obecność danych do odebrania z deskryptora, a użytkownik nie odebrał ich wszystkich (np. do połączenia sieciowego nadeszło n bajtów, a użytkownik przy pomocy funkcji `read/recv/recvfrom` odebrał m, gdzie $m < n$), to przy następnym wywołaniu `poll/select` dla tego samego deskryptora znowu zostanie zasygnalizowana obecność danych. Taka semantyka nazywana jest po angielsku *level-triggered*,

czyli wyzwalania poziomem. Dopóki dany warunek zachodzi (tutaj: obecność danych do odebrania), dopóty będzie on sygnalizowany. Inną możliwością jest semantyka wyzwalania zboczem (ang. *edge-triggered*), gdzie warunek jest sygnalizowany tylko raz – w momencie jego wystąpienia. Takie rozwiązanie kładzie większą odpowiedzialność na programiście, gdyż nie może on „przegapić" żadnej sygnalizacji, bo może doprowadzić to do straty danych.

Podstawową wadą tego mechanizmu jest to, że złożoność jednego wywołania funkcji `poll/select` to O(n), gdzie n to w przypadku `poll` liczba wszystkich deskryptorów, którymi proces jest zainteresowany, a dla `select` – najwyższy numer deskryptora. Skąd bierze się taka wysoka złożoność (najlepszą złożonością byłoby O(m) gdzie m to liczba deskryptorów, które wymagają obsłużenia)? Po pierwsze program musi przekazać do jądra całą listę deskryptorów, co wprowadza spory narzut na kopiowanie danych pomiędzy pamięcią jądra a pamięcią użytkownika. Przy tysiącach deskryptorów i tysiącu wywołań `poll/select` na sekundę może to mieć ujemny wpływ na wydajność aplikacji. Po drugie program użytkowy musi przejrzeć całą tablicę zwróconą przez funkcję systemową – nie ma możliwości otrzymania jedynie listy deskryptorów, które należy obsłużyć. Dodatkowo specyfikacja `poll/select` nie mówi nic na temat złożoności implemetacji `poll/select` w jądrze – w szczególności jądro może po kolei sprawdzać wszystkie pliki wskazywane przez deskryptory, co również jest przyczyną niskiej wydajności takiego mechanizmu. Widać, że nawet nieaktywne połączenia zużywają czas procesora, co ułatwia atak DoS na taki serwer – wystarczy stworzyć wiele połączeń z serwerem aby obciążenie procesora sięgnęło 100%.

Opisane powyżej problemy sa przyczyną, dla której wiele serwerów sieciowych (np. Apache) tworzy kilka procesów, z których każdy obsługuje część połączeń. Rozwiązanie takie zwiększa niezawodność systemu (błąd w jednym z procesów powoduje odłączenie jedynie części klientów), ale oczywiście zwiększa narzut ze strony systemu operacyjnego na przełączanie pomiędzy zadaniami (zarówno na sam *scheduler* jak i na przełączanie kontekstu). Aby jak najlepiej wykorzystać mechanizm `poll/select` należałoby zapewnić, że zestaw deskryptorów zwracany przez jądro będzie „gęsty", tzn. wiele deskryptorów rzeczywiście będzie wymagało obsługi. Można liczyć, że zgodnie z regułą 20/80 20% połączeń będzie tworzyło 80% ruchu. Dlatego, dodając możliwość „migracji" połączeń pomiędzy procesami, można stworzyć dwie grupy procesów: obsługujące połączenia bardziej i mniej aktywne. Dzięki temu procesy obsługujące mniej aktywne połączenia będą wykonywane rzadziej nie zabierając cennego czasu procesora, a procesy z drugiej grupy nie będą traciły czasu na przeglądanie rzadkiej tablicy deskryptorów.

INNE ROZWIĄZANIA WYZWALANE POZIOMEM – `kqueue`, `/dev/poll`
Aby rozwiązać problemy związane z tradycyjnym `poll/select` zaproponowano rozwiązania zachowujące semantykę wyzwalania poziomem. Dwoma mechanizmami udostępniającymi tę semantykę są `kqueue` we FreeBSD (udostępniają one także semantykę wyzwalania zboczem) oraz `/dev/poll` w Solarisie. Zmuszają one jądro do sprawdzania, czy dany warunek ciągle zachodzi i wydaje się, że narzut z tym związany spowodował przejście raczej na semantykę wyzwalania zboczem.

ROZWIĄZANIA WYZWALANE ZBOCZEM – REAL-TIME SIGNALS (SIGNAL-PER-FD), `kqueue`, `epoll`
Sygnały czasu rzeczywistego (ang. *real-time signals*) są rozszerzeniem standardowego mechanizmu sygnałów dostępnego w *nixach. Zdefiniowane przez POSIX pozwalają na kolejkowanie sygnałów oraz dostarczanie poza samym numerem sygnału wiadomości z nim związanej. Dzięki temu można zażądać od jądra, aby przy każdym zdarzeniu wysyłało do procesu sygnał czasu rzeczywistego wraz z informacją o deskryptorze oraz typie zdarzenia. Podobnie działa mechanizm `kqueue` we FreeBSD (chociaż nie jest oparty na sygnałach).

Takie podejście nie jest wolne od wad. Przy dużej liczbie deskryptorów liczba wygenerowanych zdarzeń może być bardzo duża, a kolejka przechowująca nieodebrane sygnały ma ograniczoną długość. Aby nie stracić żadnej informacji Linux wysyła wtedy do procesu tradycyjny sygnał SIGIO, który informuje o przepełnieniu kolejki. Proces po odebraniu tego sygnału musi obsłużyć wszystkie zdarzenia za pomocą standardowego `poll/select` i dopiero wtedy może wrócić do odbierania informacji o zdarzeniach za pomocą sygnałów czasu rzeczywistego. Aby poprawić tę wadę zaproponowano łatę signal-per-fd, która zapewnia że w kolejce będzie jedynie jeden sygnał dla jednego deskryptora. Dzięki temu można poprawić wydajność (nie ma niepotrzebnego wstawiania do kolejki dublujących się sygnałów) oraz zapanować nad liczbą sygnałów obecnych na raz w kolejce. Eksperymentalny serwer X15 korzystający z mechanizmu sygnałów czasu rzeczywistego w połączeniu z łatą signal-per-fd wedle autorów X15 jest szybszy nawet od Tuxa (czyli serwera HTTP wbudowanego w samo jądro). Nie rozwiązano jednak problemu (obecnego także w I/O Completion Ports opisanego niżej) zbyt częstego wywoływania jądra przy odbieraniu informacji z kolejki. Jedno wywołanie daje jedynie informacje o jednym deskryptorze, co przy obsługiwaniu tysięcy deskryptorów na sekundę powoduje niepotrzebny narzut na ciągłe przełączenia między jądrem a procesem użytkownika.

STANDARDY – AIO WEDŁUG POSIX, I/O COMPLETION PORTS MICROSOFTU PORTS

Większość powyższych rozwiązań była tworzona *ad hoc*, jako szczególne rozwiązania pewnego ogólniejszego problemu. Skupiają się one bowiem na obsłudze jedynie deskryptorów powiązanych z gniazdami. Tymczasem asynchroniczne we/wy ma także duże znaczenie dla dysków, np. dla SZBD, operujących na bazach danych zawartych na wielu dyskach. Jednolity interfejs otrzymywania informacji o asynchronicznych zdarzeniach na różnych obiektach zaproponował m.in. POSIX – AIO (Asynchronous I/O) oraz Microsoft – I/O Completion Ports. Poczynając od Windows NT 4.0 wszystkie operacje we/wy są wykonywanie asynchronicznie, więc IOCP są jedynie interfejsem z poziomu użytkownika do pewnego mechanizmu systemowego. Dodatkowo umożliwiają one równe rozłożenie obciążenia na wszystkie wątki korzystające z jednego portu (jeżeli wątków jest tyle ile procesorów to wszystkie procesory są równomiernie obciążone).

Tymczasem Linux w wersji 2.4 wewnętrznie wykonuje operacje we/wy synchronicznie i implementacja AIO na jądrach z tej linii wymaga tworzenia wątków których jedynym zadaniem jest zlecanie żądań we/wy, czekanie na ich wynik i zwracanie ich wyniku swemu rodzicowi. Jest to rozwiązanie tyleż eleganckie z punktu widzenia użytkownika (nie widzi on tworzenia dodatkowych wątków), co nieefektywne. Na szczęście Linux 2.5 zmierza ku pełnej asynchroniczności operacji we/wy, co pozwoli na efektywną implementację AIO w niedalekiej przyszłości. Łata AIO (opracowywana przez Bena LaHaise) nie obsługuje jednak na razie gniazd. Do jądra włączono więc `epoll`, aby programy pisane pod Linuksa już teraz mogły korzystać z dobrodziejstw asynchronicznego we/wy w przypadku operacji sieciowych.

Czego należy oczekiwać od dobrego rozwiązania?

Głos w tej kwestii zabierało wiele osób. Ostatecznie przemówił Linus i opisał interfejs, który według niego byłby właściwy. Musi on spełniać następujące warunki:

a) nieużywanie struktur danych o rozmiarze zależnym od liczby zdarzeń (jak sygnały czasu rzeczywistego), a jedynie od liczby deskryptorów,

b) uniknięcie bezproduktywnego przeglądania tablic niezawierających użytecznych informacji – jądro powinno zwracać informację tylko o deskryptorach wymagających uwagi,

c) nieprzekazywanie do jądra za każdym razem całego zestawu informacji o deskryptorach; zamiast tego rejestrowanie deskryptorów krok po kroku,

d) zwracanie informacji o wielu deskryptorach na raz, a nie jednym na wywołanie.

/dev/epoll

Spełnieniem postulatów z poprzedniego punktu jest interfejs `/dev/epoll`. Jak pisze Davide Libenzi na swojej stronie, szkic pierwszej wersji powstał 1 lipca 2001, a pierwsza znaleziona przeze mnie publiczna wzmianka o projekcie to wiadomość na liście linux-kernel z 10 lipca 2001. `/dev/epoll` powstała jako łata do jąder serii 2.4, a następnie została dostosowana do jąder serii 2.5.

`/dev/epoll` ma semantykę wyzwalania zboczem i korzysta z niezbyt ładnego interfejsu: specjalnego urządzenia znakowego, którego zawartość musi zostać za pomocą funkcji systemowej `mmap()` zmapowana do przestrzeni adresowej procesu. Proces chcący korzystać z `/dev/epoll` musi najpierw otworzyć urządzenie znakowe o numerze głównym 10 i numerze podrzędnym 124. Następnie musi zażądać przydzielenia w jądrze pamięci dla przewidywanej maksymalnej liczby deskryptorów (jest ona ograniczona przez stałą `MAX_FDS_IN_EVENTPOLL` w pliku include/linux/eventpoll.h – standardowo jest to 128*1024). Po przydzieleniu pamięci trzeba uzyskać do niej dostęp za pomocą wywołania `mmap()`. Powyższy opis ilustruje kod:

```
if ((kdpfd = open("/dev/epoll", O_RDWR)) == -1) { }
if (ioctl(kdpfd, EP_ALLOC, maxfds)) { }
if ((map = (char *) mmap(NULL, EP_MAP_SIZE(maxfds),
                PROT_READ, MAP_PRIVATE,
                kdpfd, 0)) == (char *) -1) { }
```

Aby dodać deskryptor do listy obsługiwanych deskryptorów należy zapisać odpowiednią strukturę do otwartego `/dev/epoll` (jest to sama stuktura, która jest używana przy korzystaniu ze zwykłego `poll`):


```
struct pollfd {
       int fd;           /* deskryptor pliku */
       short events;     /* zdarzenia, o których żądamy
                                    powiadomienia */
       short revents;    /* zdarzenia, które faktycznie
                                    zaszły */
};
```

```
struct pollfd pfd;

pfd.fd = fd;
pfd.events = POLLIN | POLLOUT | POLLERR | POLLHUP;
pfd.revents = 0;
if (write(kdpfd, &pfd, sizeof(pfd)) != sizeof(pfd)) { }
```

Usunięcie odbywa się identycznie, tylko w `pfd.events` należy wstawić flagę POLLREMOVE:

```
pfd.events = POLLREMOVE;
```

Zauważmy, że nie ma potrzeby rejestrowania deskryptora bez flagi POLLOUT, a następnie zmieniania rejestracji (przez ustawienie flagi POLLOUT), gdy proces dostanie błąd EAGAIN przy próbie zapisu do tego deskryptora. Ponieważ zdarzenia są wyzwalane zboczem, to dopóki proces nie będzie pisał do deskryptora POLLOUT, dla tego deskryptora nie będzie zwracane w `ioctl(..., EP_POLL, ...)` (patrz niżej). Dopiero gdy proces otrzyma EAGAIN przy próbie zapisu do deskryptora możliwe jest pojawienie się zdarzenia POLLOUT.

Aby dowiedzieć się, jakie deskryptory wymagają obsługi należy najpierw wykonać `ioctl()` a następnie przejrzeć zwróconą tablicę:

```
struct evpoll {
        int ep_timeout;
        unsigned long ep_resoff;
};

struct pollfd *pfds;
struct evpoll evp;

for (;;) {
        evp.ep_timeout = STD_SCHED_TIMEOUT;
        evp.ep_resoff = 0;

        nfds = ioctl(kdpfd, EP_POLL, &evp);
        pfds = (struct pollfd *) (map + evp.ep_resoff);
        for (ii = 0; ii < nfds; ii++, pfds++) {
                ...
        }
}
```

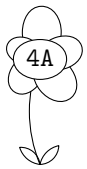
W strukturze evpoll w polu `ep_resoff` zawarty jest początek (względem wcześniej otrzymanego adresu z funkcji `mmap()`) tablicy deskryptorów gotowych na operacje I/O. Tablica zawiera `nfds` elementów typu `struct pollfd`.

Interfejs ten ma kilka wad. Po pierwsze, standardowe ograniczenie na 131072 połączeń nie jest dobrze udokumentowane, co może powodować pewne problemy przy konieczności obsługi tak dużej ich liczby. Po drugie, wraz z rejestrowaniem deskryptora, nie da się przypisać mu jakiejkolwiek informacji użytecznej dla procesu i zwracanej przy `ioctl(..., EP_POLL, ...)`. Powoduje to konieczność trzymania w procesie tablicy indeksowanej numerami deskryptorów, co jest marnotrawstem pamięci i do pewnego stopnia zdublowaniem tablicy `files->fd` w `task_struct` procesu. Trzecią wadą jest ohyda interfejsu zmuszającego programistę do wywoływania `mmap()` i `ioctl()` z pewnymi magicznymi argumentami. Interfejs taki zapewnia oczywiście znakomitą wydajność (brak kopiowania pomiędzy przestrzenią adresową użytkownika a przestrzenią adresową jądra oraz tylko jedno wywołanie jądra przy sprawdzaniu gotowości wielu deskryptorów), ale jego brzydota nie pozwoliła `/dev/epoll` na wejście do oficjalnego jądra 2.5 i zmusiła autora łaty do przeprojektowania całego interfejsu.

`sys_epoll`

Linus odmówił włączenia `/dev/epoll` do jąder 2.5 ze względu na niezbyt piękny interfejs. Dzięki wątkowi na linux-kernel zapoczątkowanemu przez Hannę Linder w październiku 2002 Davide Libenzi napisał łatę `sys_epoll`, która ostatecznie została zaakceptowana przez Linusa i 30. października oficjalnie stała się częścią jądra 2.5.45. Łata ta następnie została przepisana na jądro 2.4. Aktualna jej wersja to 0.60 (od 2.5.51 oraz dostępna do ściągnięcia ze strony domowej Davida Libenzi dla 2.4.20).

`sys_epoll`, jak sama nazwa wskazuje, jest interfejsem do tego samego mechanizmu co opisany wyżej `/dev/epoll` (mechanizmu rozumianego jako abstrakcyjna idea, gdyż sama implementacja zmieniła się dosyć znacznie między `/dev/epoll` a `sys_epoll`), lecz korzysta on jedynie z nowych wywołań systemowych (nie ma potrzeby korzystania z `mmap()` ani `ioctl()`). Dodatkowo zniesiony został sztywny limit na maksymalną liczbę obsługiwanych deskryptorów.



Obsługa z poziomu użytkownika zmieniła się znacząco. Trzymając się starej *nixowej zasady, że wszystko jest plikiem, nowa funkcja systemowa `epoll_create` zwraca deskryptor pliku:

```
int epoll_create(int size)
```

Deskryptor ten wskazuje na specjalny plik, z którego, za pomocą `epoll_wait`, można odczytywać powiadomienia. Parametr size jest wskazówką dla jądra mówiącą jaką maksymalną liczbę deskryptorów przewiduje użytkownik. Nie jest to jednak sztywny limit: dodanie większej liczby deskryptorów spowoduje powiększenie odpowiedniej struktury w jądrze.

Dodanie, usunięcie lub zmodyfikowanie deskryptora pliku dokonuje się za pomocą `epoll_ctl`:

```
typedef union epoll_data {
        void *ptr;
        int fd;
        __uint32_t u32;
        __uint64_t u64;
} epoll_data_t;

struct epoll_event {
        __uint32_t events;  /* zdarzenia, o których
                               żądamy powiadomienia */
        epoll_data_t data;  /* do dyspozycji użytkownika */
};

int epoll_ctl(int epfd, int op, int fd,
                        struct epoll_event *event)
```

Parametr `epfd` to deskryptor zwrócony przez `epoll_create`, `op` to jedna z trzech wartości:

- `EPOLL_CTL_ADD` – dodaj nowy deskryptor do listy obserwowanych
- `EPOLL_CTL_MOD` – zmodyfikuj strukturę `epoll_event` związaną z deskryptorem
- `EPOLL_CTL_DEL` – usuń podany deskryptor z listy obserwowanych

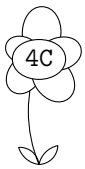
Parametr `fd` to deskryptor, którego dotyczy operacja a w `event` zapisane są zdarzenia o których żądamy powiadomienia oraz dowolna wartość wybrana przez użytkownika. Aby uzyskać od jądra informacje jakie zdarzenia zaszły na obserwowanych deskryptorach używamy `epoll_wait`:

```
int epoll_wait(int epfd, struct epoll_event *events,
                        int maxevents, int timeout)
```

Parametr `epfd` to deskryptor zwrócony przez `epoll_create`, `events` to bufor przygotowany przez użytkownika, `maxevents` to rozmiar tego bufora a `timeout` to liczba milisekund. Jeżeli jądro nie ma informacji o żadnych zdarzeniach interesujących użytkownika i przez `timeout` milisekund nie zajdą żadne takie zdarzenia, `epoll_wait` zwraca 0. Jeżeli jądro ma informacje o zdarzeniach, to zwraca liczbę wypełnionych elementów tablicy `events`, czyli liczbę deskryptorów wymagających obsługi. Zwrócenie -1 oznacza oczywiście błąd. Przykładowy kod korzystający z `sys_epoll`:

```
for(;;) {
    nfds = epoll_wait(kdpfd, &pfds, maxevents, -1);

    for(n = 0; n < nfds; ++n) {
        if(pfds[n].fd == s) {
            client = accept(listener, (struct sockaddr*)&local,
                                      &addrlen);
            if(client < 0){
                perror("accept");
                continue;
            }
            setnonblocking(client);
            if (epoll_ctl(kdpfd, EPOLL_CTL_ADD, client,
                                  EPOLLIN | EPOLLOUT) < 0) {
            fprintf(stderr,
                    "epoll set insertion error: fd=%d0,
                    client);
            return -1;
        }
    }
    else
        do_use_fd(pfds[n].ptr);
    }
}
```



Interfejs taki jest na pewno bardziej elegancki, ale wydaje się że została stracona podstawowa zaleta `/dev/epoll`: wydajność. Informacje o deskryptorach zwracane przez `epoll_wait` muszą być bowiem kopiowane pomiędzy przestrzenią adresową jądra i użytkownika. Okazuje się jednak, że testy przeprowadzone przez Linux Scalability Effort nie wykazały żadnej przewagi `/dev/epoll` nad

`sys_epoll`. Dlatego zalecane jest w przyszłych aplikacjach wykorzystywanie interfejsu `sys_epoll`, który wydaje się być ustablizowany (zachodziły w interfejsie drobne zmiany aż do jądra 2.5.51, lecz ostatnio interfejs nie był już zmieniany).

Sposób implementacji

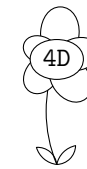
`/dev/epoll` oraz wczesne wersje `sys_epoll` używały bardzo prostych (przez to efektywnych) struktur danych: dwóch tablic, w których zapisywane były informacje o deskryptorach wymagających uwagi. Jedna tablica służyła do zapisywania aktualnie generowanych zdarzeń, a adres drugiej przekazywany był użytkownikowi w wyniku wywołania `ioctl(..., EP_POLL, ...)`. W momencie kolejnego wywołania `EP_POLL` wystarczyło przekazać użytkownikowi adres tablicy aktualnej, a nowe zdarzenia zacząć zapisywać w tablicy nieużywanej przez użytkownika.

Aby otrzymywać informacje o zdarzeniach dodano do struktury pliku możliwość definiowania dowolnych funkcji (tzw. callbacków) które będą wywoływane w momencie zajścia zdarzeń:

```
struct file {
        ...
        /* file callback list */
        rwlock_t f_cblock;
        struct list_head f_cblist;
        ...
}
```

Dodanie deskryptora pliku do eventpoll powoduje zarejestrowanie funkcji `notify_proc` na liście `f_cblist` odpowiedniej struktury `file` (wskazywanej przez dany deskryptor). `notify_proc` uaktualnia odpowiednią tablicę zapisując informację jakie zdarzenia wymagają obsługi na deskryptorze. Aby zajście zdarzeń faktycznie wykonywało wszystkie funkcje zarejestrowane na `f_cblist` potrzebne było dodanie odpowiednich wywołań w kodzie odpowiedzialnym za obsługę TCP/IP oraz łącz (ang. *pipe*). Dlatego w wersji `/dev/epoll` dla jąder 2.4 tylko takie typy deskryptorów można obsługiwać za pomocą `epoll`.

`sys_epoll` dla jąder serii 2.5 (mam na myśli implementację, która pojawiła się w jądrze 2.5.52) korzysta z nowej funkcjonalności jądra, tzn. zasypiania w kolejce (ang. *waitqueue*) z wyspecyfikowaniem funkcji, która ma być wykonana w momencie obudzenia kolejki. Standardowo funkcja ta to:



```
kernel/sched.c:
int default_wake_function(wait_queue_t *curr, unsigned mode,
                                                int sync)
{
        task_t *p = curr->task;
        return try_to_wake_up(p, mode, sync);
}
```

która czyni dokładnie to co obudzenie w jądrach 2.4 (wywołanie `try_to_wake_up`). Można jednak podać inną funkcję przy zasypianiu na kolejce – dzięki temu nie trzeba dodawać nowego mechanizmu (jak `f_cblist` w jądrach 2.4), tylko skorzystać z już istniejących. Funkcją wywoływaną po zajściu zdarzenia jest `ep_poll_callback`:

```
fs/eventpoll.c:
static void ep_ptable_queue_proc(struct file *file,
                                 wait_queue_head_t *whead,
                                 poll_table *pt):
        ...
        init_waitqueue_func_entry(&pwq->wait, ep_poll_callback);
        ...
```

Zmienił się także sposób zapamiętywania zdarzeń przekazywanych później użytkownikowi. Jeżeli deskryptor nie jest jeszcze na liście deskryptorów gotowych, zajście zdarzenia powoduje dodanie nowego elementu do listy `rdllist` we wnętrzu funkcji `ep_poll_callback`. Jeżeli deskryptor już jest na liście `rdllist` nie trzeba nic robić poza obudzeniem:

```
fs/eventpoll.c:
static int ep_poll_callback(wait_queue_t *wait, unsigned mode,
                                                  int sync):
        ...
        /* If this file is already in the ready list we exit soon */
        if (EP_IS_LINKED(&epi->rdllink))
                goto is_linked;
        list_add_tail(&epi->rdllink, &ep->rdllist);
is_linked:
```

Następnie trzeba obudzić ewentualnych czekających:

```
        /*
         * Wake up ( if active ) both the eventpoll wait list
           and the ->poll()
         * wait list.
         */
        if (waitqueue_active(&ep->wq))
                wake_up(&ep->wq);
        if (waitqueue_active(&ep->poll_wait))
                pwake++;
```

Na kolejce `ep->wq` może spać jedynie funkcja

```
fs/eventpoll.c:
static int ep_poll(struct eventpoll *ep,
                   struct epoll_event *events,
                   int maxevents, long timeout)
```

która jest wywoływana z `sys_epoll_wait`. Po obudzeniu `ep_poll` za pomocą `ep_events_transfer` przesyła zdarzenia z `rdllist` do tablicy danej przez użytkownika używając `ep_send_events`:

```
fs/eventpoll.c:
static int ep_send_events(struct eventpoll *ep, struct epitem **aepi,
                          int nepi, struct epoll_event *events):
         ...
   for (i = 0, eventcnt = 0, eventbuf = 0; i < nepi; i++, aepi++) {
       epi = *aepi;

       /* Get the ready file event set */
       revents = epi->file->f_op->poll(epi->file, NULL);

       if (revents & epi->event.events) {
           event[eventbuf] = epi->event;
           event[eventbuf].events &= revents;
           eventbuf++;
           if (eventbuf == EP_MAX_BUF_EVENTS) {
               if (__copy_to_user(&events[eventcnt], event,
                         eventbuf * sizeof(struct epoll_event))) {
                   for (; i < nepi; i++, aepi++)
                       ep_release_epitem(*aepi);
                   return -EFAULT;
```

```
                }
                eventcnt += eventbuf;
                eventbuf = 0;
            }
        }
        ep_release_epitem(epi);
    }
    ...
```

Widać, że faktyczne zdarzenia jakie są gotowe na deskryptorze sprawdzane są tuż przed przesłaniem do użytkownika.

Użycie listy gotowych deskryptorów (`rdllist`) rozwiązuje problem tablic stałej długości i pozwala na obsługę dowolnie dużej liczby plików.

Konkretne zastosowanie

Łatką `epoll` zainteresowałem się w trakcie szukania sposobów obsługi wielu tysięcy połączeń TCP/IP dla celów systemu Gadu-Gadu. Dotychczasowe rozwiązania (oparte na Windows 2000) nie radziły sobie dostatecznie dobrze i konieczne było znalezienie wydajniejszej alternatywy.

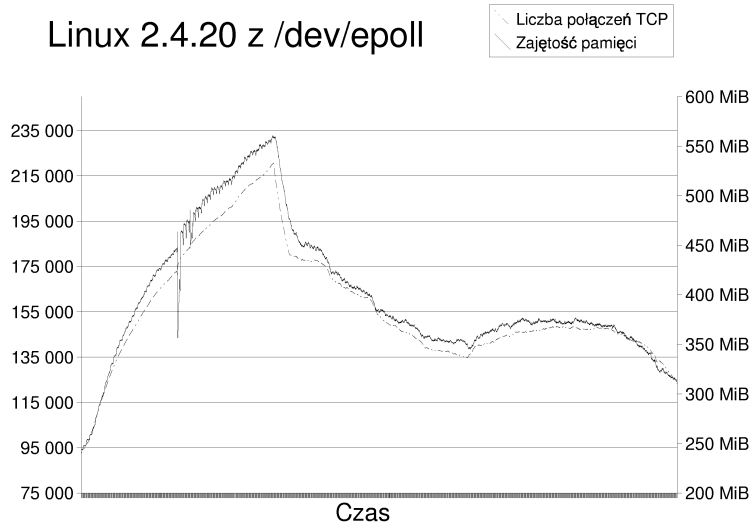
Za pomocą `/dev/epoll` napisałem serwer pośredniczący obsługujący z jednej strony klientów Gadu-Gadu, a z drugiej – centralny serwer przekazujący wiadomości pomiędzy użytkownikami.

Serwery pośredniczące działają na mocno okrojonym systemie Debian Woody, z własnoręcznie skompilowanym jądrem 2.4.20 z łatką `/dev/epoll`. Jeden serwer działa na Pentium IV 2,4GHz z 1GB RAM, a drugi na 2x Pentium III 1 GHz z 2GB RAM.

Pamięć

Wykres 1 ukazuje zajętość pamięci (tylko pamięci zajmowanej przez podsystem TCP/IP w jądrze!) oraz liczbę połączeń. Jedno połączenie zajmuje średnio 2,6KiB, a największa liczba połączeń jaką się udało obsłużyć bez używania pliku wymiany to 220700. Do zużycia pamięci przez jądro dochodzi bowiem zużycie pamięci przez proces serwera, który potrzebuje ok. 2KiB na połączenie na różne bufory.

Widać, że zużycie pamięci jest zależne liniowo od liczby połączeń. Należy się spodziewać, że dla 2GB liczba obsługiwanych połączeń może sięgnąć 400-450 tys. a dla 4GB nawet 900 tys. (biorąc pod uwagę także zajętość pamięci przez



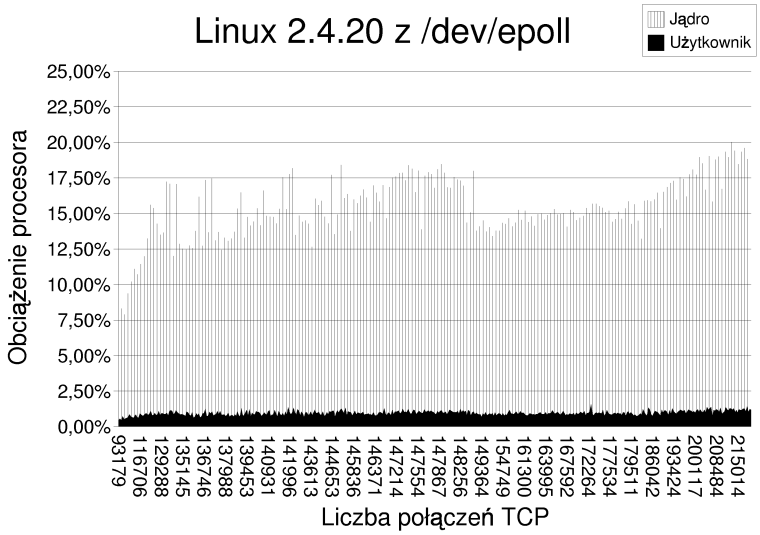
Rys 1

proces serwera; oszczędniejsze gospodarowanie pamięcią w procesie serwera mogłoby tę liczbę zwiększyć).

Obciążenie procesora

Obciążenie (wykres : 2) charakteryzuje się dużą zmiennością w czasie i zapewne bardziej zależy od chwilowej większej liczby nadchodzących pakietów (w szczególności pakietów SYN), niż od liczby połączeń. Ponieważ obciążenie procesora jest na bardzo niskim poziomie, nie podjęto próby dokładnego zbadania od czego ono zależy. Na maszynach wieloprocesorowych implementacja TCP/IP i `/dev/epoll` skalują się bardzo przyjemnie: na maszynie 2xPentium III 1GHz, przy tym samym obciążeniu procesora, obsługuje ok. 30-40% więcej połączeń niż na jednoprocesorowym Pentium IV 2,4 GHz. Wynika z tego, że nie jest tutaj potrzebna wielka moc obliczeniowa, a raczej szybkość obsługi przerwań generowanych przez karty sieciowe.

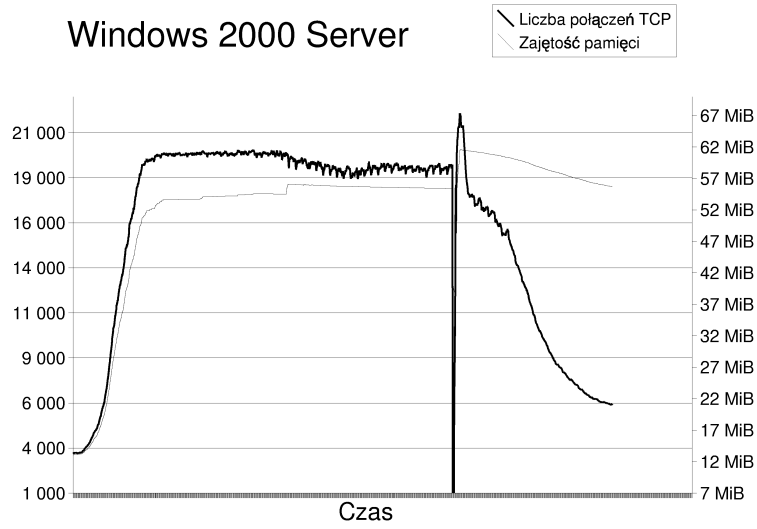
Windows 2000 Server



Rys 2

A jak to wygląda w przypadku Windows 2000 Server? Na wykresie : ?? za pamięć przydzieloną TCP uznałem całą pulę niestronicowaną (ang. *non-paged pool*). Na pewno nie jest to dokładne podejście, ale kształt wykresu pokazuje, że Windows 2000 niezbyt dobrze radzi sobie ze zwalnianiem pamięci w tym obszarze.

Po prawej stronie wykresu, gdy liczba połączeń wyraźnie spada, zajęta pamięć nie jest odzyskiwana. Szacunkowe zaptrzebowanie na pamięć niestronicowaną to około 2,8KiB na połączenie, czyli ok. 10% więcej niż w przypadku Linuksa. Skąd się jednak bierze znaczna przewaga Linuksa na Windows 2000 o której wspomniałem na początku? Otóż podczas prób zwiększenia liczby obsługiwanych połączeń pod Windows 2000 szybko okazało się, że jest granica któej nie da się przeskoczyć. Poniżej znajduje się cytat z artykułu Q126402 z Microsoft Knowledge Base. Sam artykuł nie jest już dostępny, a kopia, którą posiadam, została wydobyta z Google (ale niestety teraz nie ma jej już nawet tam):



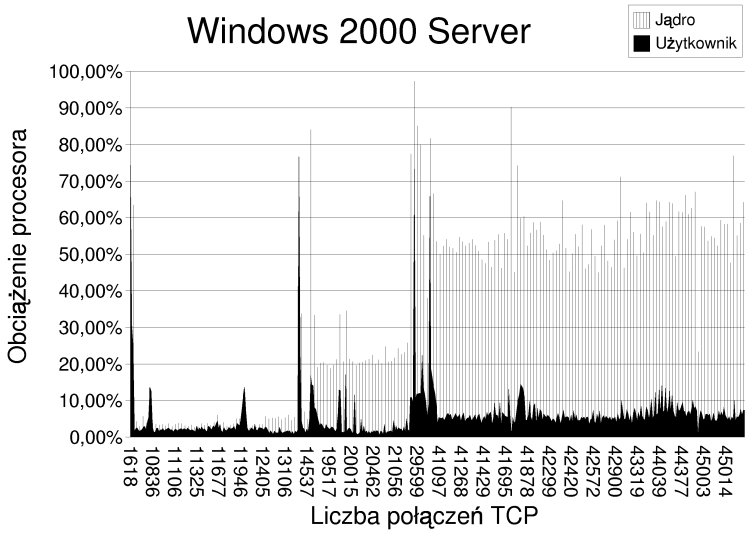
Rys 3

```
The information in this article applies to:
Microsoft Windows 2000 Server
Microsoft Windows 2000 Advanced Server
Microsoft Windows 2000 Professional
Microsoft Windows 2000 Datacenter Server
Microsoft Windows NT Workstation versions 3.5, 3.51, 4.0
Microsoft Windows NT Server versions 3.5, 3.51, 4.0

MaximumNonPagedPoolSize = DefaultMaximumNonPagedPool + ((Physical MB -
 4) * MaxAdditionNonPagedPoolPerMB)

If MaximumNonPagedPoolSize < (NonPagedPoolSize + PAGE_SIZE * 16),
then MaximumNonPagedPoolSize = (NonPagedPoolSize + PAGE_SIZE * 16)

IF MaximumNonPagedPoolSize >= 128 MB MaximumNonPagedPoolSize = 128 MB
```



Rys 4

Pozwala to na optymistyczne szacowanie, że Windows 2000 może obsłużyć maksymalnie ok. 60 tys połączeń, nawet mając do dyspozycji 4 GB RAM-u.

Dodatkową przeszkodą jest stopień wykorzystania procesora, znacznie wyższy niż w przypadku Linuksa (widać to na wykresie 4).

DPC (ang. *Deferred Procedure Call*) jest odpowiednikiem softirq z Linuksa, czyli nie wdając się w szczegóły, jest to z grubsza rzecz biorąc czas poświęcony na obsługę TCP/IP. Należy tutaj podkreślić, że oprogramowanie działające na Windows 2000 nie korzystało z I/O Completion Ports, a jedynie z tzw. *overlapped sockets*, tzn. mechanizmu, w którym system, po zajściu każdego zdarzenia, wywołuje przekazaną przez proces procedurę, przekazując jej informacje o zdarzeniu (w szczególności gniazdo którego dotyczyło). Wywoływanie przez jądro procedur z przestrzeni użytkownika w świecie Windows NT nazywane jest APC (ang. *Asynchronous Procedure Call*). Czas poświęcony przez procesor na obsługę APC nie został jednak ujety na wykresie, gdyż nie przekraczał 1%. Gdzie w takim razie tkwi przyczyna tak słabej wydajności Windows 2000?

Nie przeprowadzałem dokładnych badań nad tą kwestią, ale dobrym kandydatem wydaje się być nieefektywność funkcji, która na podstawie adresu i portu docelowego przychodzącego pakietu określa do jakiego połączenia on należy. Korzysta ona z jakiejś funkcji mieszającej (ang. *hash function*) i być może nieefektywna implementacja owej funkcji mieszającej (np. kładąca duży nacisk na port docelowy, który w tym przypadku jest zawsze taki sam) powoduje często konieczność rozwiązywania konfliktów w tablicy mieszającej. Zgodnie z zaleceniami dokumentacji Microsoftu zwiększono w rejestrze systemowym parametr MaxHashTablesSize na maksymalną wartość (65536), ale nie rozwiązało to problemu. Niewykluczone jest, że nieefektywność tkwi właśnie w wykorzystaniu mechanizmu *overlapped sockets*, lecz nawet redukując stopień wykorzystania procesora nie da się obejść drugiego z ograniczeń, czyli niewielkiej dopuszczalnej wielkości puli pamięci niestronicowanej.

Podsumowanie

`epoll` w połączeniu z istniejącą implementacją TCP/IP jest bardzo wydajnym i stabilnym mechanizmem. Rozwiązanie oparte na Linuksie 2.4 i `/dev/epoll` pracuje od trzech miesięcy obsługując codziennie średnio 130 tys. połączeń (tzn. średnio w każdej chwili serwer obsługuje 130 tys. jednoczesnych połączeń). Dysponując odpowiednio dużą ilością pamięci RAM oraz szerokim łączem sieciowym można obsłużyć dowolnie dużą liczbę połączeń („dowolnie dużą" należy rozumieć w sensie zdrowego rozsądku, nie *stricte* matematycznie `;-)`. Wedle niepotwierdzonych informacji FreeBSD również zdaje się dobrze radzić z wielką ilością połączeń:

```
From: Dan Kegel[4]

Date: Wed Jul 11 2001 - 19:39:23 EST

...
2) A little birdie told me that someone had gotten a freebsd
 box to handle something like half a million connections.
...
```

później w tym samym wątku:

```
...
Here's what the little birdie told me exactly:
```

[4] `http://www.uwsg.iu.edu/hypermail/linux/kernel/0107.1/0575.html`

```
> You'll be happy to know I've achieved over 500,000 connections
> on Pentium hardware with 4G of RAM and 2 1Gbit cards on FreeBSD 4.3.
...
```

Wtedy być może był to rekord świata, lecz ostatnio na Slashdocie pojawiła się informacje o prawdopodobnym aktualnym rekordzie:
`http://bsd.slashdot.org/bsd/03/01/30/1352202.shtml?tid=122`
oryginalna wiadomość tutaj: `http://docs.freebsd.org/cgi/ ...`
`... /getmsg.cgi?fetch=58510+0+archive/2003/freebsd-hackers/`
`...`
`... /20030202.freebsd-hackers`

```
FreeBSD developer Terry Lambert, in a recent posting to
the 'freebsd-hackers' mailing list, mentioned that he'd
tuned a FreeBSD 4.4 box with 4GB of RAM to achieve
1,603,127 simultaneous IP connections, and goes on to
say: 'As far as I know, I hold the single machine
connection record for an x86 box.' This is an impressive
achievement any way you look at it (though it begs the
question of whether or not the box had any resources left
to actually do anything with those connections...), and it
speaks well of both FreeBSD's capabilities and Terry's
skills and knowledge. I'm curious, though, if anyone has
approached, matched, or exceeded that number elsewhere?
```

Podziękowania

Dla Malwiny za wszystko, Czarka Kaliszyka i Bogusia Kluge za przejrzenie i poprawienie błędów oraz oczywiście Majaki za tolerowanie wysyłania wszystkiego w ostatniej chwili :-)

Zastrzeżenia

Odnośniki

- http://www.kegel.com/c10k.html – opis problemu obsługi wielu tysięcy połączeń TCP/IP na jednym komputerze
- http://people.freebsd.org/~jlemon/papers/kqueue.pdf – kqueue we FreeBSD
- http://soldc.sun.com/articles/polling_efficient.html – /dev/poll w Solarisie
- http://www.sysinternals.com/ntw2k/info/comport.shtml – IOCP
- http://www.hpl.hp.com/techreports/2000/HPL-2000-174.html – opis signal-per-fd
- http://www.xmailserver.org/linux-patches/nio-improve.html – strona domowa łat epoll
- http://www.chromium.com/cgi-bin/crosforum/YaBB.pl – serwer X15
- http://www.kvack.org/~blah/aio/ – AIO dla Linuksa
- http://lse.sourceforge.net/io/aionotes.txt – Uwagi na temat AIO
- http://www.uwsg.iu.edu/hypermail/linux/kernel/ ... ... /0010.3/0003.html – Linusa opis dobrego interfejsu
- http://www.uwsg.iu.edu/hypermail/linux/kernel/ ... ... /0107.1/0393.html – pierwsza wzmianka o /dev/epoll
- http://www.uwsg.iu.edu/hypermail/linux/kernel/ ... ... /0210.2/1100.html – początek prac nad sys_epoll
- http://lse.sourceforge.net/epoll/index.html – Linux Scalability Effort o epoll

Udostępnianie serwera baz danych opartego na PostgreSQL

mgr inż. Adam Buraczewski <adamb@polbox.pl>
Samodzielna Pracownia Sztucznej Inteligencji, Politechnika Warszawska

STRESZCZENIE: PostgreSQL jest jednym z najstarszych i najbardziej zaawansowanych serwerów relacyjnych baz danych dostępnych na zasadach Open Source, który pod względem swoich możliwości, elastyczności i wydajności śmiało konkuruje z drogimi, komercyjnymi produktami.

Polem, na którym system ten nie zdobył dotąd zbyt dużej popularności jest publiczne udostępnianie usług serwera bazodanowego. Sytuacja taka ma miejsce u dostawców Internetu, oferujących swoim klientom konta z obsługą PHP i bazą danych, czy też na uniwersytetach, na których pracownicy i studenci korzystają z ogólnodostępnych serwerów. Powodów takiego stanu rzeczy jest wiele, przede wszystkim PostgreSQL nie miał dotąd opinii systemu bezpiecznego, w którym można w pełni kontrolować działania użytkowników.

Referat dotyczy najnowszych wersji systemu, w których przebudowano pod tym kątem mechanizm uprawnień i dodano nowe, ciekawe możliwości. Omówiono w nim metody administracji dużymi instalacjami PostgreSQLa, automatyzacji i optymalizacji jego pracy.

WPROWADZENIE

Termin PostgreSQL jest zapewne znany wszystkim użytkownikom Linuksa i wolnego oprogramowania – jest to nazwa jednego z kilku systemów zarządzania relacyjnymi bazami danych dostępnych na zasadach Open Source. Za jego konkurentów uważa się MySQL, Firebird/Interbase, SAP DB i inne systemy baz danych wykorzystujące język SQL, wśród których PostgreSQL wyróżnia się liberalnością licencji i zaawansowanymi możliwościami. Nic w tym dziwnego, bowiem system ten zaczął powstawać jeszcze w połowie lat osiemdziesiątych,

na Uniwersytecie Kalifornijskim w Berkeley. Wówczas prof. Michael Stonebreaker uzyskał grant, którego celem były eksperymenty z nowymi rozwiązaniami w dziedzinie relacyjnych baz danych i stworzenie silnika bazodanowego, mającego szczególne zastosowanie w nauce i inżynierii. W efekcie powstało oprogramowanie, które nadal jest intensywnie rozwijane, przenoszone na nowe platformy, a jego funkcjonalność pozwala mu śmiało konkurować z drogimi, komercyjnymi systemami. Do jego największych zalet zaliczyć można bogactwo wbudowanego dialektu języka SQL, wydajność, zachowywaną nawet w sytuacji dużego obciążenia, dobrze działające zabezpieczenia przed utratą danych w wyniku awarii, a przede wszystkim niesłychaną wręcz elastyczność, możliwości rozbudowy i programowania. PostgreSQL rozwija się obecnie dzięki wolontariuszom z całego świata oraz wielu firmom, które swoimi pracownikami, zasobami i funduszami wspierają ten projekt.

Wszystkie wymienione wcześniej cechy PostgreSQLa powodują, że jest to system rozpatrywany jako serce wielu projektów informatycznych. Szczególne uznanie zdobył wśród programistów pracujących dotąd z systemami komercyjnymi, nawet takimi jak Oracle, Informix, DB2 oraz MS SQL, i jest chętnie wybierany w mniej krytycznych zastosowaniach jako ich doskonały darmowy odpowiednik. Natomiast niszą, w której system ten nie zdobył jeszcze dużej popularności, są serwery baz danych udostępniane klientom przez dostawców Internetu lub też studentom na uczelniach. W takich przypadkach często wykorzystywany jest młodszy i mniej zaawansowany system MySQL, który dotychczas oferował większe możliwości kontroli poczynań użytkowników niż PostgreSQL i był prostszy w administracji. System PostgreSQL był co prawda także wykorzystywany w takich zastosowaniach, ale stosunkowo rzadko i to jedynie po dokonaniu w jego kodzie szeregu poprawek. Wraz z ukazaniem się pod koniec 2002 roku wersji 7.3 systemu PostgreSQL sytuacja ta zaczęła ulegać zmianie, bowiem najnowsze jego wydanie oferuje znacznie większe możliwości w zakresie kontrolowania użytkowników i egzekwowania praw dostępu. Stało się tak dzięki aktywnej współpracy osób i instytucji, które takiej funkcjonalności oczekiwały – głównym motorem rozwoju systemu PostgreSQL są bowiem potrzeby jego użytkowników.

W niniejszym referacie postarano się omówić różne przypadki zastosowania systemu PostgreSQL i konfiguracje, w których mogłyby pracować bazy danych. Dla każdej z nich przedstawiono zagadnienia związane z tworzeniem i administracją, a także optymalizacją pracy takiego systemu.

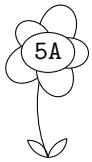
Polityka bezpieczeństwa

Decydując się na udostępnienie serwera bazodanowego w sieci, w szczególności w sieci publicznej, jaką jest Internet, należy przedsięwziąć wszelkie możliwe środki, aby był on odpowiednio zabezpieczony przed potencjalnymi atakami. Wybierając metody obrony należy wziąć pod uwagę wiele czynników, w tym: wartość danych zgromadzonych w bazach, charakter, w jakim serwer jest wykorzystywany, a także liczbę i uprawnienia jego użytkowników. Najważniejsze zasady, którymi powinien kierować się projektant udostępnianego systemu, są następujące:

- Jak najlepiej odseparować serwer bazodanowy od Internetu. Najlepiej byłoby, aby serwer w ogóle nie miał publicznego numeru IP i był dostępny z zewnątrz jedynie za pośrednictwem oprogramowania, które realizuje interfejs użytkownika. Zwykle oznacza to umieszczenie systemu PostgreSQL na oddzielnej maszynie i umożliwienie łączenia się do baz danych jedynie ze wskazanych hostów, na których działa np. serwer WWW lub SSH.

- Jeżeli to nie jest konieczne, nie umożliwiać użytkownikom i aplikacjom wydawania dowolnych poleceń języka SQL. W przypadku wielu projektów można z góry opisać wszystkie możliwe operacje, które będą wykonywane na bazie danych, a następnie stworzyć dodatkowy poziom oprogramowania (tzw. *middleware*), który będzie pośredniczył pomiędzy nieznanym użytkownikiem a serwerem, sprawdzając poprawność przekazywanych danych i uniemożliwiając wykonanie „obcych" poleceń. Takie oprogramowanie chroni bazę danych przed użytkownikami, którzy mając bezpośredni dostęp do serwera bazodanowego mogliby wydać polecenia obciążające lub wręcz blokujące działanie serwera. Dodatkową zaletą tego rozwiązania mogłoby być wprowadzenie własnego protokołu transmisji danych.

- Stosując mechanizmy schematów i list kontroli dostępu ograniczyć uprawnienia poszczególnych użytkowników do obiektów baz danych: tabel, perspektyw, procedur składowanych itp. W szczególności należy zadbać, aby nie powstawały w systemie nadmiarowe konta użytkowników, o zbyt dużych uprawnieniach, warto też wykorzystać mechanizm grup użytkowników do łatwiejszego zarządzania ich prawami. W wielu instalacjach stosuje się nawet ograniczanie dostępu do danych poprzez odebranie zwykłym użytkownikom jakichkolwiek praw do odczytywania i modyfikacji tabel i udostępnianie ich jedynie poprzez perspektywy i specjalne procedury składowane, które



dodatkowo kontrolują uprawnienia użytkowników, choć zwykle takie postępowanie nie jest konieczne.

- Monitorować poczynania użytkowników, zarówno poprzez odpowiedni dziennik odnotowujący połączenia do baz danych, wydane polecenia i czas ich wykonywania, jak i poprzez zbieranie statystyk dotyczących funkcjonowania serwera. Szczególnie należy monitorować liczbę równoczesnych połączeń do bazy danych, tworzenie przez użytkowników nowych obiektów (np. tabel tymczasowych), wykorzystanie buforów, pamięci oraz dysków.

Administrator systemu PostgreSQL powinien też na bieżąco śledzić listy dyskusyjne poświęcone błędom w tym systemie (`pgsql-bugs`, `pgsql-hackers`, `pgsql-patches`) i wykorzystywać pojawiające się tam informacje oraz łaty.

Nawiązywanie połączenia użytkownika z bazą danych

Zabezpieczenia w systemie PostgreSQL można podzielić na dwie grupy: związane z nawiązywaniem połączenia z bazą danych i związane z określaniem uprawnień i możliwości zalogowanych użytkowników. Za pierwszą część odpowiada przede wszystkim plik `pg_hba.conf`, a za drugą – listy kontroli dostępu (Access Control Lists, ACL), skojarzone z poszczególnymi obiektami bazy.

Plik `pg_hba.conf` ma bardzo prostą budowę: określa którzy użytkownicy mogą się łączyć z którymi bazami danych, dodatkowo nakładając ograniczenia na adresy IP, z których następuje połączenie oraz na rodzaj autoryzacji. Dostępne są następujące metody:

- Metoda `trust` powoduje wyłączenie autoryzowania użytkowników – system przyjmuje, że każdy kto się poda za danego użytkownika nim faktycznie jest. Z przyczyn oczywistych nie jest to metoda, którą można polecić dla publicznych serwerów.

- Metoda `ident` wykorzystuje protokół auth/ident do sprawdzenia nazwy użytkowników. Nie jest to metoda wiarygodna, gdy połączenia z bazą danych są realizowane nielokalnie, ale nadaje się do połączeń lokalnych, zdejmując z użytkowników konieczność dodatkowego podawania haseł. Dodatkowy plik, `pg_ident.conf` pozwala na przemapowania nazw użytkowników systemu operacyjnego na użytkowników w bazie danych.

- Metoda `md5` wymaga podania hasła przez klienta bazy danych. Hasło jest łączone z nazwą użytkownika i bazy, szyfrowane algorytmem MD5 i dopiero

tak przesyłane. Dzięki temu utrudniono ewentualne podsłuchanie hasła przez niepowołane osoby (tzw. *sniffing*).

- Metody `password` oraz `crypt` są podobne do `md5`, są jednak bardziej podatne na ewentualne podsłuchanie. Zachowano je ze względu na zgodność z poprzednimi wersjami systemu PostgreSQL, nie powinny być jednak używane.

Z punktu widzenia bezpieczeństwa systemu najlepsze wydają się dwie opcje: albo zezwolenie użytkownikom jedynie na połączenia lokalne, albo z wykorzystaniem autoryzacji `md5`. W tym drugim przypadku zwykłe połączenia nie są szyfrowane, można jednak w tym celu wykorzystać program SSH/OpenSSH. Pisząc np.: `ssh konto@adres.pl -L 5431:127.0.01:5432`

Warto zaznaczyć, że choć PostgreSQL oferuje możliwość korzystania z SSL do połączeń, to jego autorzy polecają rozwiązanie z SSH jako dające większe możliwości. Dzięki temu można też zwiększyć bezpieczeństwo systemu, gdyż użytkownik będzie autoryzowany podwójnie: raz przez system operacyjny i serwer SSH, a drugi – przez PostgreSQLa.

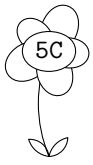
System PostgreSQL oferuje jeszcze jedną możliwość, jaką jest przełączanie kont użytkowników podczas jednej sesji klienta z bazą danych. Jest to dopuszczane jedynie dla kont z uprawnieniami superużytkowników (administratorów). Polecenie:

```
SET SESSION AUTHORIZATION użytkownik;
```

powoduje przełączenie konta bieżącego użytkownika na wskazane, natomiast polecenia:

```
RESET SESSION AUTHORIZATION;
RESET ALL;
```

powodują przywrócenie wszystkich ustawień z początku danej sesji. Mechanizm ten nie nadaje się co prawda do zastosowania, gdy aplikacje użytkownika łączą się bezpośrednio z PostgreSQLem, ale jest doskonały do zastosowania w opisanym wcześniej oprogramowaniu typu *middleware*, które pośredniczy między klientem a serwerem.



Instalacje z dużą liczbą użytkowników

W odróżnieniu od niewielkich instalacji systemu PostgreSQL, przewidzianych dla kilku do kilkunastu użytkowników, sytuacje w których jest wiele baz danych i dziesiątki lub wręcz setki użytkowników są znacznie trudniejsze do

administracji, chociaż PostgreSQL radzi sobie z nimi znakomicie. Aby umożliwić więcej równoczesnych połączeń do serwera należy oczywiście zwiększyć parametr `max_connections` w pliku `postgresql.conf` i odpowiednio skonfigurować inne parametry w tym pliku (niektóre zagadnienia konfiguracji opisano w dalszej części referatu). Istnieje jednak kilka dodatkowych mechanizmów, które wspomagają pracę w takich systemach:

- Mechanizm schematów (ang. *schemes*), który pozwala na wyróżnienie w poszczególnych bazach danych swojego rodzaju przestrzeni nazw. Schematy są głównie przeznaczone do podziału obiektów bazy danych (tabel, perspektyw, procedur składowanych itp.) w zależności od zadań użytkowników. Można nadawać użytkownikom oddzielne uprawnienia, zarówno do całych schematów jak i poszczególnych obiektów.

- Ustawiając opcję `db_user_namespace` w pliku `postgresql.conf` można zezwolić na obsługę użytkowników lokalnych dla baz danych. Nazwy takich użytkowników są przechowywane w systemie w postaci `użytkownik@baza_danych`, dzięki czemu można tworzyć niezależne ich pule dla każdej serwowanej bazy danych.

- Opcja `statement_timeout` pozwala z kolei na ustawienie maksymalnego czasu wykonywania pojedyńczego polecenia SQL. Polecenia wykonywane dłużej są automatycznie przerywane przez serwer, co zapobiega nadmiernemu obciążeniu serwera przez jednego użytkownika. Podobnie, opcja `deadlock_timeout` określa czas, po którym oczekiwanie na zwolnienie blokad kończy się błędem; na bardziej obciążonych serwerach powinna to być wartość większa niż domyślna 1s.

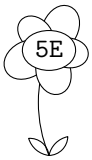
- Stosując polecenie `REVOKE` należy uniemożliwić zwykłym użytkownikom tworzenie nowych obiektów w bazie: tabel, tabel tymczasowych, procedur itp. Oczywiście dotyczy to tylko baz, których struktura nie będzie już zmieniana. Warto zaznaczyć, że dopiero w wersji 7.3 systemu jest możliwe nadawanie szczegółowych uprawnień do różnych obiektów, w tym do procedur składowanych i schematów.

- W wersji 7.3 systemu PostgreSQL wprowadzono nowy atrybut procedur składowanych: `SECURITY DEFINER`, który powoduje, że procedura wykonywana jest nie na prawach użytkownika, który ją uruchomił, lecz na prawach użytkownika, który ją zdefiniował. Dzięki temu można tworzyć systemy,

w których co prawda zwykły użytkownik nie ma możliwości np. modyfikacji zawartości tabel, ale może uruchamiać procedurę, która po skontrolowaniu parametrów będzie mogła wykonywać polecenia `INSERT`, `UPDATE` oraz `DELETE`.

Warto w tym miejscu wspomnieć, że w systemie PostgreSQL istnieje pewna niedoskonałość mechanizmów rządzących listą użytkowników. Jest to jedna z nielicznych rzeczy dzielonych pomiędzy wszystkie bazy danych obsługiwane w ramach jednej instalacji tego systemu. W obecnych wersjach PostgreSQLa, usuwając użytkownika poleceniem `DROP USER`, nie można automatycznie usunąć informacji o prawach z list kontroli dostępu skojarzonych z obiektami w bazach danych. Co więcej, stworzenie nowego użytkownika powoduje nadanie mu pierwszego wolnego identyfikatora (niestety nie są one nadawane z sekwencji), co powoduje, że można stworzyć użytkownika o identyfikatorze, który miał dotąd jakiś inny użytkownik i że nowy użytkownik przejmie wszystkie uprawnienia starego. Jest pewne, że ta niedoskonałość zostanie w przyszłości usunięta, lecz twórcy systemu PostgreSQL nie zdecydowali się jeszcze jak to będzie dokładnie rozwiązane. Na razie lepiej nie kasować użytkowników, można za to blokować ich konta. Opisano to w dalszej części referatu.

Opisany powyżej mechanizm schematów pozwala na bardzo skrupulatne określanie praw użytkowników. Dla zgodności „w dół" z poprzednimi wersjami systemu PostgreSQL, jego autorzy zdecydowali się na stworzenie w każdej nowej bazie danych domyślnego schematu `public`, który, podobnie jak to było w starszych wersjach tego systemu, pozwala na dowolne tworzenie nowych obiektów (w tym tabel i tabel tymczasowych) dowolnym użytkownikom, którzy mogą wykonywać polecenia SQL w danej bazie danych. Zwykle nie jest to pożądane, dlatego w miarę możliwości należy albo skasować schemat `public` i korzystać jedynie z oddzielnie utworzonych schematów, albo ograniczyć uprawnienia do schematu `public`.



Udostępnianie serwera

Współcześnie wiele serwisów WWW jest realizowanych dynamicznie: strony nie są przechowywane w ostatecznej formie, lecz są generowane „w locie", na podstawie żądania wysłanego przez przeglądarkę. Do realizacji tego typu stron wykorzystuje się zwykle skryptowe języki programowania (głównie PHP i Perl) oraz bazy danych. Podstawowe problemy przy oferowaniu kont na takim serwerze dotyczą ograniczenia miejsca przeznaczonego dla poszczególnych baz danych. Nie może to się przecież odbywać z wykorzystaniem mechanizmu

quota systemu operacyjnego, ponieważ jedynie serwer bazodanowy zna ścieżki do plików składających się na poszczególne bazy.

W takich sytuacjach pomocny jest dodatkowy moduł `dbsize`, dostępny wraz ze źródłami systemu PostgreSQL. Pozwala on na szacowanie objętości wskazanej bazy danych (udostępnia funkcję `database_size()`) i daje się łatwo połączyć z zewnętrznymi skryptami. Wykorzystując tę funkcjonalność, administrator takiego systemu powinien napisać program, który będzie:

- monitorował wielkość baz danych poszczególnych użytkowników,
- w przypadku przekroczenia limitu wielkości bazy – podejmował jakąś akcję, np. blokował dostęp do tej bazy.

Program taki może oczywiście korzystać z jakiejś dodatkowej, specjalnej bazy danych, która będzie przechowywać informacje o kontach użytkowników, ilości wykupionego miejsca, wniesionych opłatach itp. Blokowanie dostępu może odbywać się poprzez np. wykonanie na prawach superużytkownika następującego polecenia:

```
ALTER USER użytkownik VALID UNTIL '1970-01-01'
```

W przypadku tego typu systemów, ważne jest także odpowiednie blokowanie dostępu do baz danych, tak żeby do danej bazy danych mógł dostać się jedynie jej właściciel. Służy do tego specjalna opcja, `sameuser`, którą wpisujemy do pliku `pg_hba.conf` jako nazwę bazy danych:

```
host sameuser all 0.0.0.0 0.0.0.0 md5
```

W powyższym wpisie można słowo kluczowe `all` zastąpić wpisem `+grupa`, co ograniczy dostęp jedynie dla użytkowników z podanej grupy. Warto także pomyśleć o zablokowaniu dostępu do baz danych z zewnątrz i zezwolić użytkownikom jedynie na korzystanie z tunelu poprzez SSH, co opisano we wcześniejszych sekcjach referatu.

Ze względu na wykorzystanie takiego serwera przez osoby trzecie, administracja nim może być znacznie utrudniona. Trudno bowiem powiedzieć, do czego serwer będzie wykorzystywany, jak poszczególne bazy danych będą eksploatowane, na ile zapisane w nich dane będą się zmieniać itp. To utrudnia podejmowanie decyzji dotyczących np. częstotliwości wykonywania polecenia `VACUUM` dla poszczególnych baz danych. Pomóc w tym może projekt nazwany Auto Vacuum Daemon, dostępny na `gborg.postgresql.org`. Analizuje on, czy poszczególne tabele baz danych wymagają uruchomienia procesu `VACUUM`

i uruchamia go w razie potrzeby. Inne zalecenia dla administratora takiego systemu są podobne, jak w przypadku wcześniej omówionej wieloużytkownikowej instalacji. Dodatkowo należy zwrócić uwagę na:

- możliwość korzystania przez użytkowników takiego systemu z języków proceduralnych, najlepiej zezwalając co najwyżej na tworzenie procedur składowanych w językach SQL i PL/SQL,
- możliwość stosowania różnych lokalizacji baz danych (polecenie `initlocation`), co pozwala np. rozłożyć bazy danych na różnych dyskach lub partycjach,
- konieczność logowania działań użytkowników (opcje `log_connectios`, `log_pid`, `log_duration`, `log_statement`, oraz `log_timestamp` w pliku `postgresql.conf`).

PostgreSQL a uczelniana pracownia komputerowa

Jak napisano na wstępie, PostgreSQL był początkowo projektem uniwersyteckim i służył głównie celom naukowym. Zdobył też bardzo dużą popularność jako system, na podstawie którego nauczano baz danych na wielu uczelniach. Duże znaczenie miała tutaj otwartość kodu źródłowego, którego jakość i przejrzystość jest oceniana bardzo wysoko, jak również duże możliwości i zgodność tego systemu ze standardami. Trzeba jednak pamiętać, że specyfika zastosowania programu w dydaktyce różni się znacznie od typowej wieloużytkownikowej i wielobazowej instalacji, przedstawionej w poprzedniej sekcji referatu.

Przede wszystkim studenci tworzący własne bazy danych i oprogramowanie do nich powinni mieć więcej swobody w korzystaniu z serwera i rozwijaniu własnego oprogramowania. Powinni mieć dostęp do większej ilości mechanizmów, w tym np. do języków proceduralnych. Z jednej strony dobrze by było, gdyby każdy student miał uprawnienia superużytkownika, mógł samodzielnie tworzyć nowe bazy danych i nowych użytkowników, oraz korzystać z różnych języków proceduralnych i możliwości rozbudowy oferowanej przez PostgreSQL, a z drugiej – taka sytuacja narusza przecież wiele zasad bezpieczeństwa, zagrażając innym bazom danych działającym na tym samym serwerze.

Pewnym rozwiązaniem tego problemu jest umożliwienie uruchamiania przez studentów wielu instalacji systemu PostgreSQL na tym samym komputerze. Jest to jak najbardziej możliwe, wymaga jednak poświęcenia od kilkudziesięciu do kilkuset megabajtów dysku dla każdego studenta. Wystarczy wówczas, że każdy student uruchomi na swoim koncie program `initdb`, tworząc własną, oddzielną

instalację systemu, w terminologii PostgreSQLa zwaną klastrem (ang. *cluster*). Można to zrealizować następującymi poleceniami:

```
mkdir pgsql
initdb -D pgsql -E latin2 -W
pg_ctl -D pgsql -l pgsql/logfile start
```

Żeby powyższe rozwiązanie zadziałało, każdy uruchomiony proces `postmaster` powinien nasłuchiwać połączeń na innym porcie, najlepiej o numerze ustalonym z administratorem komputera i do tego różnym od standardowego 5432. Administrator powinien też uruchomić na tej maszynie skrypt, który będzie automatycznie kończył działanie procesów użytkowników, którzy się już wylogowali – dzięki temu w systemie nie będą pozostawać działające niepotrzebnie demony. Studenci natomiast muszą być pouczeni co do zawartości plików `postgresql.conf` oraz `pg_hba.conf`, która nie będzie zbytnio obciążać systemu i stwarzać gróźb włamania. Najlepiej, gdy plik `postgresql.conf` zostanie pozostawiony bez zmian (poza wspomnianym numerem portu), ponieważ wówczas zasoby systemu nie są intensywnie wykorzystywane, natomiast plik `pg_hba.conf` będzie zawierać jedynie wpisy:

```
local all all md5
host all all 127.0.0.1 255.255.255.255 md5
```

co pozwoli na przyjmowanie samych połączeń lokalnych. Wpisy te mogą być też uzupełnione przez administratora komputera odpowiednimi regułami dla firewalla, które będą blokować dostęp z zewnątrz do danej puli portów.

Przedstawione rozwiązanie ma pewne wady, do których należy znacznie większe wykorzystanie zasobów (mocy obliczeniowej, dysku, pamięci, gniazd TCP/IP) niż to by miało miejsce w przypadku uruchomionego tylko jednego procesu `postmaster`. Studenci otrzymują jednak do swojej dyspozycji własne instalacje baz danych, które mogą lepiej poznawać, administrować nimi itp. Rozwiązanie to szczególnie dobrze sprawuje się w przypadku pracowni składającej się z wielu komputerów z zainstalowanym systemem Linux, wówczas każdy student może uruchomić swój serwer bazodanowy na oddzielnej maszynie. Należy tu zaznaczyć, że PostgreSQL pozwala na przechowywanie swoich plików roboczych (klastra) na wolumenie NFS.

Na koniec warto dodać, że przedstawione w poprzednich sekcjach referatu metody udostępniania baz danych dla wielu użytkowników także mają zastosowanie w dydaktyce, szczególnie na pierwszym etapie nauki języka SQL. Studenci łączą się do przygotowanej wcześniej bazy danych i ćwiczą swoje umiejętności. Warto tu zastosować możliwość korzystania z szablonów baz danych:

prowadzący zajęcia może przygotować bazę wzorcową a następnie dla każdego użytkownika wykonać serię poleceń:

```
DROP DATABASE nazwa;
CREATE DATABASE nazwa
WITH OWNER = nazwa TEMPLATE = baza_wzorcowa;
```

które spowodują usunięcie dotychczasowych baz danych i stworzenie nowych na podstawie podanego szablonu.

Optymalizacja systemu

Najważniejszym problemem związanym z systemem PostgreSQL jest jego odpowiednia konfiguracja i optymalizacja. System ten bowiem jest domyślnie skonfigurowany tak, żeby wykorzystywał minimum zasobów komputera (pamięci, pamięci dzielonej, semaforów), co powoduje że do praktycznie każdych poważnych zastosowań należy go odpowiednio przekonfigurować. Poniżej przedstawiono najważniejsze opcje pliku `postgresql.conf`, które muszą być zmienione:

- `shared_buffers`, standardowo równa 64, powinna być ustawiona na np. 10000 (dokładna wartość zależy od ilości RAM w komputerze i przeznaczenia serwera). Opcja ta odpowiada za ilość pamięci dzielonej wykorzystywanej przez jedną instalację PostgreSQLa (określa liczbę 8 KB bloków). Pamięć ta służy systemowi do różnych celów, w tym jako cache baz danych i zapytań. Bez zwiększenia tej wartości PostgreSQL działa bardzo wolno.

- `sort_mem` odpowiada z kolei za ilość pamięci przeznaczonej do potrzeb sortowania danych. Gdy jej zabraknie, PostgreSQL wykorzystuje pliki tymczasowe. Wartość tego parametru nie może być ani zbyt duża ani zbyt mała, należy ją dobrać eksperymentalnie do konkretnego serwera, już podczas jego typowej pracy.

- `max_fsm_relations` określa, ile tabel i indeksów będzie śledzić podsystem Free Space Manager (FSM). Standardowa wartość może być zbyt mała, a można ją łatwo wyznaczyć następującym zapytaniem SQL (należy zsumować jego wynik dla wszystkich baz danych):

  ```
  SELECT COUNT(*) FROM pg_class WHERE NOT relkind IN ('i','v');
  ```

- `max_fsm_pages` mówi ile 8 KB stron będzie śledzić podsystem FSM – wartość tę można odczytać analizując wynik polecenia `VACUUM VERBOSE`, a dokładniej:

sumując wyświetlone przez to polecenie parametry `Changed:` dla wszystkich tabel i wszystkich baz danych. Przy częstszym wykonywaniu polecenia `VACUUM` wartość tego parametru może być mniejsza. Pomiaru należy dokonywać podczas zwykłej pracy bazy danych – najlepiej wykonać go kilkakrotnie i wynik uśrednić.

- `random_page_cost` określa wydajność, z jaką system operacyjny odczytuje dane z dysku i można ją wyznaczyć dla danego komputera programem `randcost`: `ftp://candle.pha.pa.us/pub/postgresql/randcost`

- pozostałe parametry optymalizatora najlepiej pozostawić do wyznaczenia programowi `pgautotune`, który po prostu uruchamia system z różnymi ustawieniami i wylicza takie ich wartości, dla których wydajność jest największa.

Do oceny wydajności PostgreSQLa można także wykorzystać Statistic Collector, który jest dostępny począwszy od wersji 7.3 tego systemu. Pozwala on na śledzenie wykorzystania poszczególnych tabel i indeksów oraz informuje o wykorzystaniu pamięci cache i buforów. Dzięki temu można się dowiedzieć, na ile aktualne ustawienia serwera są optymalne.

W przypadku wielu serwerów bazodanowych warto jeszcze rozważyć możliwość przeniesienia plików dziennika (WAL, znajdują się w katalogu `pg_xlog`) na inny fizyczny dysk. Można to zrealizować bardzo prosto: wystarczy zatrzymać serwer (komendą `pg_ctl stop`), przenieść wspomniany katalog w całości w inne miejsce, upewnić się, że ma on odpowiednio ustawione prawa dostępu, w starym miejscu zrobić link symboliczny wskazujący na ten katalog i ponownie uruchomić serwer. Taka prosta operacja pozwala na znaczne zwiększenie wydajności serwera, zwłaszcza w przypadku intensywnie modyfikowanych baz danych. W przyszłości autorzy systemu PostgreSQL przewidują stworzenie mechanizmu tzw. *tablespaces*, który będzie pozwalał na dowolne rozmieszczanie baz danych, tabel i indeksów na różnych dyskach.

Na koniec warto wspomnieć, że choć PostgreSQL doskonale działa w systemach wieloprocesorowych, to jak dotąd próby jego uruchamianie na klastrach (np. Mosix) nie dały zbyt dobrych rezultatów. Powodem takiego stanu rzeczy jest głównie intensywne wykorzystanie w tym systemie pamięci dzielonej i semaforów. W przyszłości będą dostępne narzędzia do replikacji typu *multimaster* (już obecnie trwają prace nad mechanizmem Two-Phase Commit, będącego podstawą takiej funkcjonalności), dające faktycznie możliwości zbliżone do klastra.

Podsumowanie

PostgreSQL jest systemem, który bardzo dobrze sprawuje się w środowiskach wieloużytkownikowych. Jego bardzo duże możliwości i elastyczność pozwalają na wykorzystanie go z powodzeniem do różnych projektów specjalistycznych, ale też jako serwera bazodanowego ogólnego stosowania, udostępnianego szerokiemu gronu użytkowników nawet u dostawców Internetu lub na uczelniach. Najważniejszym problemem jest jednak brak zwięzłej dokumentacji, która opisywałaby wszystkie aspekty stosowania i administracji tego systemu, informacje są bowiem rozrzucone pomiędzy dostępnymi podręcznikami, różnymi, często nieoficjalnymi stronami w Internecie, a nawet archiwum list dyskusyjnych.

Zagadnienia przedstawione w referacie być może pomogą w administrowaniu opartymi o PostgreSQL systemami bazodanowymi.

Źródła informacji o PostgreSQL

Poniżej zestawiono najważniejsze adresy (źródła programów i wiedzy) związane z systemem PostgreSQL:

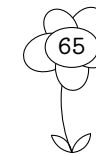
- `http://www.postgresql.org` – strona domowa projektu PostgreSQL,
- `http://techdocs.postgresql.org` – kompendium wiedzy na temat systemu, jego budowy, administracji itp.,
- `http://gborg.postgresql.org` – lista różnych projektów związanych z systemem PostgreSQL, zawiera wiele użytecznych rozszerzeń i narzędzi administracyjnych; znajduje się tam większość omawianych w referacie programów,
- `http://archives.postgresql.org` – archiwum angielskojęzycznych list dyskusyjnych dotyczących systemu PostgreSQL,
- `pl.comp.bazy-danych` – polska grupa dyskusyjna, na której często poruszane są sprawy systemu PostgreSQL,
- `http://www.dbf.pl/faq` – FAQ grupy pl.comp.bazy-danych, obejmujące najczęściej zadawane pytania dotyczące baz danych, zawiera osobny rozdział o Postgresie.

Natywne bazy XML

Michał Sajdak <xterm@linuxnews.pl>

XML? Przecież mam własny standard zapisu danych. Pracowałem nad nim kilka dni. Dzięki niemu mój program działa o pół sekundy szybciej niż z użyciem XML-a! Pomnożcie to przez tysiące operacji – to około godziny zyskanego czasu! Co się jednak stanie, drogi programisto, jeśli ktoś inny będzie zmuszony (bo dobrowolnie tego nie zrobi) zajrzeć do Twego kodu? Jakie znaczenie ma godzina w porównaniu z przebijaniem sie przez programistyczne sztuczki i zarośla kodu? Omawiana sytuacja to tylko jeden przykład gdzie XML może być przydatny.

No dobrze, mamy XML-a – ale jak go przechowywać? Na dysku? W jednym polu bazy relacyjnej? Składać go z pól bazy relacyjnej? Być może tak, choć widać, że jeśli nasz system operuje głównie na dokumentach, żadna z tych możliwości nas nie satysfakcjonuje. Bo przecież nie będziemy przeszukiwać ręcznie plików na dysku, albo grzebać po bazie, której struktura jest koszmarna . . . Nieciekawe wydaje się też rozwiązanie, w którym najpierw rozbijamy XML-a na kawałki tylko po to, by po jakimś czasie złożyć go z tych samych kawałków. Oczywiście wszystko ręcznie, wszystko za każdym razem inaczej.



Naturalnym rozwiązaniem wydaje się być użycie bazy danych zdolnej do przechowywania XML-i. Do dyspozycji mamy 3 ogólne rozwiązania proponowane przez Inicjatywę XML:DB:

- natywne bazy XML (Native XML Databases), które definiują logiczny model dla dokumentów XML, przechowują i udostępniają dokumenty zgodnie z tym modelem. Dokument XML w tym modelu to podstawowa porcja danych – odpowiednik wiersza w relacyjnych bazach danych. Nie jest zdefiniowany żaden sposób przechowywania plików XML; mogą one być trzymane w relacyjnej bazie danych, ale także np. jako zwykłe pliki na dysku;
- bazy z funkcjonalnością XML (XML Enabled Databases), posiadające możliwość odwzorowania przechowywanych danych na XML;
- hybrydowe bazy XML (Hybrid XML Databases) – inne bazy danych wykorzystujące XML-a.

Oczywiście wybór konkretnego rozwiązania, jak prawie zawsze, zależy od naszej aplikacji. Zajmiemy się tylko jednym rodzajem baz: natywnymi bazami XML. Od razu nasuwa się pytanie co zyskujemy używając takiej bazy, a co będzie stanowić problem. Zaletą jest na pewno fakt, że osiągamy pewien cel: mamy bazę danych, do której możemy dodawać XML-e, usuwać je, modyfikować oraz – jak to bywa w bazie danych – mamy możliwość przeszukiwania. Nie musimy sięgać do ekwilibrystyki polegającej np. na wspomniamym wcześniej ręcznym wyciąganiu danych z bazy relacyjnej i odpowiednim ich składaniu. Za takie możliwości musimy zapłacić jednak pewnymi niedogodnościami:

- otrzymujemy stosunkowo niewielką wydajność (choć zazwyczaj bazy umożliwiają indeksowanie);
- możemy mieć problemy z integralnością danych (związane z przechowywaniem plików);
- nie ma dobrej metody zmieniania XML-i w bazie (można wyciągnąć dokument, zmienić go i zapisać, ale nie o to chodzi...);
- cała koncepcja natywnych baz XML jest jeszcze niedojrzała (sam XML solidnie nie zadomowił się jeszcze na rynku, a co dopiero natywne bazy XML; cały czas trwają prace nad standardami).

Jeśli wady te nie godzą w kluczowe założenia naszej aplikacji i inne rozwiązanie nie wydaje sie nam najszczęśliwsze, możemy podjąć decyzję o wykorzystaniu

natywnej bazy XML. Na pewno przyda nam się wiedza związana z językami zapytań (SQL, czyli Structured Query Language, znany z relacyjnych baz danych, jest tu mało przydatny). Podobne jak nauka obcego języka pomaga nam poznać obcy kraj - tak nauka języków zapytań bardzo pomoże nam w poznaniu natywnych baz XML. Powinniśmy rzucić okiem na najbardziej popularne z języków:

- XPath, który został użyty raczej z powodu braku innego języka niż ze względu na jego możliwości;
- XQuery, który jest dość 'mocnym' językiem, wywodzącym się bezpośrednio z języka Quilt, a pośrednio z XQL-a (XML Query Language), SQL-a i OQL-a (Object Query Language). Jest on rozszerzeniem XPath 2.0, umożliwiającym m.in. złączenia, grupowanie. Dostępne są w nim funkcje (wraz z rekurencją), pętle for, itd.;
- XUpdate, który umożliwia dokonywania zmian w plikach XML (tworzenie nowych elementów, ich zmianę, usuwanie). Zapytanie jest plikiem XML (warto zwrócić uwagę na podobieństwo do XSLT).

Na koniec możemy przejść do poszukiwania konkretnej implementacji bazy o najbardziej interesujących nas możliwościach. Do naszej dyspozycji są niekomercyjne bazy OpenSource jak np. Xindice, eXist oraz pokaźna ilość baz komercyjnych jak np.: Tamino czy TextML. Nie pozostaje nic innego tylko instalować i testować.

A co z przyszłością i perspektywami natywnych baz XML? Są one związane z samym XML-em. XML systematycznie, choć powoli, wchodzi na rynek. Natywne bazy XML nie są jakąś rewolucją zmieniającą świat informatyczny z dnia na dzień. Cała idea jest dość młoda i da się zauważyć pewną ospałość towarzyszącą zagadnieniu. Czy natywne bazy XML dojrzeją do tego by używać je powszechnie, jak to ma miejsce w przypadku XML-a? Czas pokaże.

PRZYDATNE ODNOŚNIKI:

- `http://www.xmldb.org/` (Inicjatywa XML:DB, tutaj warto zacząć swą przygodę z natywnymi bazami XML);
- `http://www.xml.com/pub/a/2001/10/31/nativexmldb.html` (krótki artykuł omawiający podstawy natywnych baz XML);
- `http://www.rpbourret.com/xml/XMLDBLinks.htm` (duża liczba odnośników do ciekawych stron związanych z tematem).

Reverse engineering w świetle prawa autorskiego

Roman Bieda <prawnik@prawnik.net.pl>

Jednym z warunków rozwoju wolnego oprogramowania jest zapewnienie jego kompatybilności z oprogramowaniem rozpowszechnianym w oparciu o komercyjne licencje. Tylko w ten sposób można zapewnić konkurencję na rynku oprogramowania oraz zagwarantować użytkownikom realną możliwość wyboru. Problem w tym, że dominujący model dystrybucji oprogramowania zakłada rozpowszechnianie programu wyłącznie w formie kodu wynikowego. W praktyce jedynym sposobem poznania zasad działania programu komputerowego może okazać się wykorzystanie techniki reverse engineering. Powstaje zatem pytanie, w jakim zakresie i na jakich zasadach twórcy wolnego oprogramowania mogą, w świetle polskiego prawa autorskiego, wykorzystywać tę technikę.

Omówienie problemu rozpocząć należy od kilku słów na temat zakresu ochrony autorskoprawnej programu komputerowego.

Przepisy prawa autorskiego, dotyczące ochrony programów komputerowych, stanowią niemal dosłowne tłumaczenie postanowień Dyrektywy Rady Wspólnot Europejskich o ochronie prawnej programów komputerowych (1). Programy komputerowe podlegają ochronie jak utwory literackie, chyba że inaczej postanowiono w rozdziale 7 ustawy. Ochronie autorskoprawnej podlegają wyłącznie programy spełniające przesłanki utworu w rozumieniu art. 1 ust. 1 pr. aut. A zatem przedmiotem prawa autorskiego jest jedynie program komputerowy

stanowiący „przejaw działalności twórczej o indywidualnym charakterze, ustalony w jakiejkolwiek postaci". Stwierdzenie, że program komputerowy podlega ochronie autorskoprawnej nie wyjaśnia jednak, jaki jest w istocie zakres tej ochrony (2). Wyznaczenie granicy między chronionymi i niechronionymi elementami programu jest jednym z najbardziej kontrowersyjnych zagadnień prawa autorskiego. Szersze omówienie tej kwestii przekracza zakres niniejszej pracy, poprzestanę na zasygnalizowaniu problemu.

Ochrona autorskoprawna obejmuje wszystkie „formy wyrażania" programu. Na gruncie ustawy takiej samej ochronie podlega kod źródłowy jak kod wynikowy programu (3). Zgodnie z podstawową zasadą prawa autorskiego ochroną może być objęty wyłącznie sposób wyrażenia, natomiast nie są objęte ochroną „odkrycia, idee, procedury, metody i zasady działania oraz koncepcje matematyczne (art. 1 ust. 2 (1) pr. aut.)" (4).

Zasadę tę w stosunku do programów komputerowych potwierdza art. 74 ust. 2 pr. aut., zgodnie z którym ochronie nie podlegają „idee i zasady będące podstawą jakiegokolwiek elementu programu komputerowego, w tym podstawą łączy". W konsekwencji należy uznać, że nie podlegają ochronie autorskoprawnej „reguły logiczne i matematyczne zastosowane w programie, pomysły, metody i procedury stanowiące podstawę algorytmów i języków programowania, jak również sposoby postępowania (metody działania) i formuły rozwiązania określonych zagadnień (w tym algorytmy matematyczne)" (5).

Interfejsy

Dla twórców wolnego oprogramowania szczególnie istotne jest poznanie metod działania, idei i zasad leżących u podstaw interfejsów. Dostęp do tych informacji jest bowiem nieodzowny dla stworzenia kompatybilnego oprogramowania (6). Prawo autorskie posługuje się terminem „łącza" (art. 74 ust. 2 pr. aut.). Ustawodawca nie przybliża znaczenia tego terminu i, jak się wydaje, zamieszczenie takiej definicji w ustawie nie byłoby właściwym rozwiązaniem. Natomiast preambuła Dyrektywy wyjaśnia, że przez „interfejsy" należy rozumieć części programu umożliwiające połączenie i interakcję pomiędzy poszczególnymi elementami software'u i hardware'u. Ochrona interfejsów oparta jest na takich samych zasadach, jak pozostałych elementów programu komputerowego. Jak już zostało to zasygnalizowane, ochrona nie rozciąga się na idee i zasady leżące u podstaw interfejsów. W konsekwencji nie podlegają ochronie informacje dotyczące współdziałania programu z innymi programami (7). Oczywiście ochrona rozciąga się na „formę wyrażania" idei i zasad zawartych w interfejsach. Jednak

programista, dysponujący informacjami o zasadach działania interfejsów, może implementować je w samodzielnie stworzonym kodzie programu.

Należy zwrócić uwagę, że w praktyce możliwości twórczego działania programisty piszącego kod interfejsu są bardzo ograniczone. Programista projektujący interfejs zdeterminowany jest przede wszystkim funkcją, jaką ma spełniać ten element programu oraz standardami panującymi w tej dziedzinie. W przypadku, gdy kod interfejsu stanowi jedyną możliwość wyrażenia danej reguły, danego rozwiązania, należy uznać, że taki fragment kodu programu pozbawiony jest ochrony autorskoprawnej. Kod interfejsu nie odznacza się bowiem w takim przypadku cechą „indywidualnego charakteru". Usprawiedliwione jest zatem przejęcie takiego fragmentu kodu do nowo tworzonego programu (8).

Reverse engineering

Skoro interesujące twórców wolnego oprogramowania informacje nie podlegają ochronie autorskoprawnej, to dlaczego tyle uwagi prawnicy poświęcają technice reverse engineering?

Zauważmy, że dotarcie do niechronionych odkryć, pomysłów i idei zawartych w utworze nie stwarza zazwyczaj większych trudności. Chcąc poznać ideę, pomysł, na którym oparta została książka lub film, wystarczy przeczytać daną pozycje lub wybrać się do kina i obejrzeć film. Natomiast program komputerowy stanowi utwór szczególnego rodzaju. Poznanie niechronionych elementów programu możliwe jest wyłącznie w drodze analizy jego działania lub dekompilacji (9). Dokonanie tych czynności wiąże się z trwałym lub czasowym zwielokrotnieniem programu komputerowego albo tłumaczeniem jego formy - w rozumieniu art. 74 ust. 4 pr. aut., czyli z wkroczeniem w monopol autorski. A zatem, generalnie czynności te wymagają zgody podmiotu uprawnionego z praw autorskich.

Ustawodawca, mając na uwadze uzasadnione interesy użytkowników, w tym zapewnienie możliwości dotarcia do niechronionych elementów programu, wprowadził postanowienia ograniczające monopol autorski. W konsekwencji prawo autorskie zawiera przepisy pozwalające legalnemu użytkownikowi programu na stosowanie w ograniczonym zakresie technik reverse engineering, bez konieczności uzyskania zgody podmiotu praw autorskich.

Należy podkreślić, że wykorzystanie techniki reverse engineering bez zgody podmiotu praw autorskich dopuszczalne jest wyłącznie w zakresie i na warunkach określonych w ustawie. Prowadzenie reverse engineering w szerszym zakresie lub z naruszeniem zasad określonych w ustawie stanowi naruszenie prawa autorskiego.

W oparciu o przepisy prawa autorskiego możemy mówić o dwóch postaciach techniki reverse engineering, mianowicie o przewidzianej w art. 75 ust. 2 pkt. 2 tzw. reverse analysis oraz, wprowadzonym w art. 75 ust. 2 pkt. 3, zezwoleniu na dekompilację programu. (10).

Reverse analysis

W oparciu o przepis art. 75 ust. 2 pkt. 2 pr. aut. osoba uprawniona do korzystania z egzemplarza programu komputerowego ma prawo do obserwowania, badania i testowania funkcjonowania programu. Czynności te dokonywane mają być w celu poznania idei i zasad leżących u podstaw programu. Analiza działania programu może odbywać się wyłącznie w trakcie „wprowadzania, wyświetlania, stosowania, przekazywania lub przechowywania" programu komputerowego.

Zgodnie z przepisem, czynności polegające na obserwowaniu, badaniu i testowaniu funkcjonowania programu mogą zostać podjęte w celu poznania wszelkich zawartych w nim idei i zasad. Odmiennie niż w przypadku dekompilacji programu, działania użytkownika nie muszą ograniczać się do uzyskania informacji niezbędnych do zapewnienia kompatybilności. Użytkownik dysponuje szeroką swobodą w zakresie wykorzystania uzyskanych w drodze analizy programu informacji o zasadach i ideach leżących u jego podstaw. Informacje te mogą stać się przedmiotem publikacji naukowej, ilustracją wykładu czy posłużyć jako podstawa do pracy nad konkurencyjnym programem.

Należy podkreślić, że zezwolenie na analizę programu, o którym mowa w art. 75 ust. 2 pkt. 2, nie może zostać wyłączone lub ograniczone w umowie (art. 76 pr. aut.).

Dekompilacja

W konkretnym przypadku poprzestanie na reverse analysis może nie pozwolić na poznanie idei i zasad leżących u podstaw programu w zakresie niezbędnym do zapewnienia kompatybilności międzyprogramowej. Prawo autorskie dopuszcza zatem dekompilację programu, obwarowując ją jednak szeregiem warunków oraz ograniczeń w zakresie wykorzystania uzyskanych informacji.

Zgodnie z prawem autorskim nie wymaga zgody uprawnionego „zwielokrotnianie kodu lub tłumaczenie jego formy w rozumieniu art. 74 ust. 4 pkt. 1 i 2, jeżeli jest to niezbędne do uzyskania informacji koniecznych do osiągnięcia współdziałania niezależnie stworzonego programu komputerowego z innymi programami komputerowymi" (art. 75 ust. 2 pkt. 3 pr. aut.).

A zatem podstawową przesłankę dopuszczalności dekompilacji stanowi cel, jakiemu ma służyć. Dekompilacja prowadzona może być wyłącznie w celu uzyskania informacji koniecznych do zapewnienia kompatybilności międzyprogramowej.

W konsekwencji, przepis nie stanowi podstawy do dekompilacji dokonanej przez entuzjastę komputerów zainteresowanego poznaniem budowy programu (11). Nie będzie on także usprawiedliwieniem dla dekompilacji prowadzonej wyłącznie w celach naukowych, szkoleniowych, czy z czystej przyjemności poznawania nowych rozwiązań technicznych (12).

Dekompilacja może zostać legalnie przeprowadzana wyłącznie w sytuacji, gdy jest to „niezbędne" do uzyskania informacji pozwalających na zapewnienie współdziałania programów. Dokonujący dekompilacji musi zatem wykazać, że inne metody uzyskania koniecznych informacji, w szczególności omówiona powyżej analiza programu, okazały się niewystarczające.

Dekompilacja może być prowadzona zarówno w celu uzyskania kompatybilności z programem poddanym dekompilacji, jak również z innym programem niż program dekompilowany. Jak wskazują J. Barta, R. Markiewicz wykładnia językowa i celowościowa przepisu przemawia za „zajęciem stanowiska dopuszczającego tworzenie na podstawie dekompilacji, zrealizowanej zgodnie z art. 75, programów „równoważnych" z programem dekompilowanym i współpracujących z tym samym programem co program dekompilowany" (13). Ponadto dopuszczalność dekompilacji uzależniona jest od łącznego spełnienia kolejnych trzech warunków wskazanych w przepisie (art. 75 ust. 2 pkt. 3 lit. a,b,c).

a) Dekompilacja może być wykonywana wyłącznie przez „licencjobiorcę lub inną osobę uprawnioną do korzystania z egzemplarza programu komputerowego" bądź przez inną osobę działającą na rzecz uprawnionego. A zatem uprawniony nie musi „osobiście" prowadzić dekompilacji, może powierzyć wykonanie tej czynności zatrudnionemu programiście.

b) Kolejnym warunkiem dopuszczalności dekompilacji jest stwierdzenie, że informacje konieczne do zapewnienia kompatybilności programów nie były „uprzednio łatwo dostępne" dla uprawnionego. Uprzednia „łatwa dostępność" takich informacji wyklucza bowiem „niezbędność" dekompilacji. W konsekwencji, należy uznać za niedopuszczalną dekompilację, gdy takie informacje zostały podane do publicznej wiadomości przez producenta oprogramowania (np. w dokumentacji programu, w literaturze naukowej). Udostępnienie kodu źródłowego interfejsów należy uznać za „łatwe udostępnienie" informacji niezbędnych do zapewniania współdziałania programów.

W praktyce mogą powstawać spory, czy udostępnione przez producenta informacje (dokumentacja, kod źródłowy) są wystarczające do tworzenia kompatybilnego oprogramowania. Problemy tego rodzaju mogą powstać szczególnie w sytuacji, gdy producent „nawiązując" do idei wolnego oprogramowania, udostępnia cześć kodu źródłowego programu. Po przystąpieniu do analizy udostępnionego fragmentu kodu może okazać się, że nie zawiera on informacji niezbędnych do zapewnienia kompatybilności.

Bez wątpienia nie będzie można uznać, że informacje są „łatwo dostępne" w przypadku, gdy producent wiążę ich udostępnienie z dodatkową opłatą (14).

Kolejnym bardzo istotnym zagadnieniem jest odpowiedź na pytanie, czy dokonujący dekompilacji zobowiązany jest przed jej rozpoczęciem poinformować o planowanej dekompilacji producenta programu i wystąpić o udostępnianie informacji dotyczących zasad kompatybilności międzyprogramowej. Przepisy prawa autorskiego nie nakładają takiego obowiązku. Moim zdaniem twórca wolnego oprogramowania nie jest zobowiązany do informowania i konsultowania swoich planów z producentem programu mającego być przedmiotem dekompilacji (15). Przed przystąpieniem do dekompilacji programista musi jednak dokonać stosownej analizy co do zakresu publicznie udostępnionych informacji na temat zasad kompatybilności programu. Niewątpliwie producent programu będzie w tym zakresie osobą najbardziej kompetentną (16). Ponadto należy pamiętać, że dekompilacja jest procesem czasochłonnym i absorbującym znaczne nakłady finansowe. W konkretnym przypadku współpraca w zakresie udostępnienia koniecznych informacji między twórcą wolnego programu a producentem programu, co do którego rozważamy podjęcie dekompilacji, może okazać się korzystna dla obu stron.

c) Omawiany przepis ogranicza zakres dozwolonej dekompilacji „do tych części oryginalnego programu komputerowego, które są niezbędne do osiągnięcia współdziałania" (tj. kompatybilności) z innym programem. Jeżeli producent programu oznaczył, które części programu odpowiadają za kompatybilność, to programista powinien rozpocząć dekompilację od wskazanych części programu. Nie wyklucza to jednak prowadzenia dekompilacji w szerszym zakresie, jeżeli oznaczone fragmenty programu faktycznie nie zawierały niezbędnych informacji.

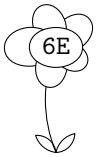
Wydaje się, że powyższe ograniczenie przedmiotu dekompilacji stanowi najsurowszy z warunków wyznaczających ramy legalnej dekompilacji. Dokonującemu dekompilacji bardzo trudno jest wskazać, przed zapoznaniem się z jej wynikami, która część programu zawiera informacje dotyczące zasad kompatybilności. W konsekwencji, wielu programistów nie potrafiąc jednoznacznie wskazać stosownej części programu, zrezygnuje z dekompilacji w obawie przed

naruszeniem prawa autorskiego (17). Wydaje się, że w konkretnym przypadku, wobec wystąpienia z roszczeniem z tytułu naruszenia prawa autorskiego przez zbyt szeroko przeprowadzoną dekompilację, uzasadnionym może być skorzystanie przez sąd z konstrukcji nadużycia prawa podmiotowego (art. 5 k.c.).

Przepisy prawa autorskiego nie tylko określają warunki dopuszczalności dekompilacji, ale również wprowadzają istotne ograniczenia w zakresie wykorzystania uzyskanych w ten sposób informacji. Zgodnie z postanowieniami art. 75 ust. 3 pr. aut. informacje uzyskane w drodze dekompilacji nie mogą zostać:

- wykorzystane do innych celów niż osiągnięcie współdziałania niezależnie stworzonego programu komputerowego,
- przekazane innym osobom, chyba że jest to niezbędne do osiągnięcia współdziałania niezależnie stworzonego programu komputerowego,
- wykorzystane do rozwijania, wytwarzania lub wprowadzania do obrotu programu komputerowego o istotnie podobnej formie wyrażenia lub do innych czynności naruszających prawa autorskie.

Przyjrzyjmy się bliżej powyższym postanowieniom. Zauważmy, że informacje uzyskane dzięki dekompilacji mogą zostać przekazane innemu programiście, odpowiadającemu za stworzenie kompatybilnego programu. W praktyce często wykorzystywana jest technika tzw. „chińskiego muru". Jedna grupa programistów dekompiluje program i sporządza raport o zasadach funkcjonowania interfejsów (o zasadach kompatybilności programu). Natomiast druga grupa programistów, w oparciu o ten raport, pisze własny kod programu. Prawo autorskie dopuszcza tego typu „przepływ" informacji między programistami. Nie jest natomiast dopuszczalne przekazywanie innym osobom informacji w innym celu niż zapewnienie kompatybilności międzyprogramowej. Uzyskane w drodze dekompilacji informacje nie mogą zatem zostać przedmiotem publikacji naukowej lub być wykorzystane jako ilustracja wykładu.



Praktyka orzecznicza stoi przed trudnym zadaniem oceny czy „nowy" program posiada „istotnie podobną formę wyrażania" do programu poddanego dekompilacji. Podkreślić należy, że przepis nie wyklucza tworzenia programów zbliżonych pod względem funkcjonalnym. Jak stwierdził Sąd Apelacyjny w Poznaniu „Podobieństwo w zakresie samej funkcji programów nie może być dostateczną podstawą naruszenia prawa autorskiego. Aby można było mówić o naruszeniu prawa autorskiego, podobieństwo porównywanych utworów musi być innego rodzaju niż podobieństwo wynikające ze sposobu przedstawienia zadania oraz kontynuacji i rozwijania ogólnie znanych danych" (18).

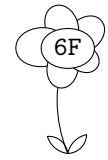
Nasuwa się kolejne praktyczne pytanie, w jakiej fazie prac nad wolnym programem, któremu chcemy zapewnić kompatybilność, możemy podjąć dekompilację wcześniej stworzonego programu. Przepisy ustawy nie precyzują tej kwestii. Przepis art. 75 ust. 2 pkt. 3 pr. aut. stanowi jednak, że dekompilacja ma prowadzić do zapewnienia kompatybilności „niezależnie stworzonego programu komputerowego z innymi programami komputerowymi." Należy zatem przyjąć, że najpierw musi rozpocząć się praca nad „niezależnie stworzonym programem" a dopiero potem twórca może przystąpić do dekompilacji innego programu. Oczywiście, w konkretnym przypadku dekompilacja może mieć miejsce na bardzo wczesnym etapie prac nad programem. Konieczne jest jednak, aby twórca na tyle znał zasady działania własnego programu, by mógł wskazać na pewien związek programu z programem poddanym dekompilacji. Programista musi wykazać, dlaczego niezbędnym okazało się poddanie dekompilacji akurat tego programu.

Należy zwrócić uwagę, że prawo autorskie pomija klauzulę zawartą w art. 6 ust. 3 Dyrektywy. W oparciu o art. 35 pr. aut. należy przyjąć, iż przepisy o dozwolonej dekompilacji nie mogą być interpretowane w sposób, który prowadziłby do naruszenia normalnego korzystania z utworu lub godził w słuszne interesy twórcy.

Należy podkreślić, że podobnie jak w przepadku zezwolenia na analizę programu, również zezwolenie na dekompilację programu nie może zostać wyłączone lub ograniczone w umowie (art. 76 pr. aut.). Umowa licencji może jednak zezwalać na analizę i dekompilację programu komputerowego w szerszym zakresie, niż przewidują to przepisy ustawy.

Zakończenie

Przepisy prawa autorskiego dopuszczają praktykę reverse engineering, w tym zarówno reverse analysing jak dekompilację programu komputerowego. Technika reverse engineering stanowi bardzo często jedyną praktyczną możliwość dotarcia do niechronionych prawem autorskim elementów programu. Zakaz reverse engineering, w szczególności dekompilacji programu, byłby równoznaczny z wprowadzeniem ochrony tajemnicy treści programu przez prawa wyłączne oraz pośrednim przyznaniem praw wyłącznych do elementów programu nie podlegających ochronie autorskoprawnej (19). Sytuacja taka byłaby sprzeczna z jedną z podstawowych zasad prawa autorskiego, zgodnie z którą ochrona z prawa autorskiego nie rozciąga się na pomysły, idee i zasady leżące u podstaw programu.



PRZYPISY:

1. Dyrektywa Rady Wspólnot Europejskich Nr 91/250 z dnia 14 maja 1991 r. o ochronie prawnej programów komputerowych.

2. Problem ten był przedmiotem rozstrzygnięć sądów amerykańskich. W sprawie „Whelan Associates, Inc. v. Jaslow Dental Laboratory, Inc." sąd uznał, że ideą programu „Dentalab" jest „ sprawna organizacja laboratorium dentystycznego". Ideą programu jest w tym ujęciu jego ostateczny cel. Natomiast kod programu oraz jego struktura i organizacja, stanowią chronioną prawem formę wyrażania. Zerwanie z takim rozumieniem idei nastąpiło w orzeczeniu „Computer Associates International, Inc. v. Altai Inc.". Więcej zobacz: J. Barta, R. Markiewicz, Główne problemy prawa komputerowego, Warszawa 1993, str. 31, A. Nowicka, Prawnoautorska i patentowa ochrona programów komputerowych, Warszawa 1995, str. 36, M. Grewal, Copyright Protection of Computer Software, E.P.I.R. 1996 /8, R. Bieda, Reverse engineering - praktyka dekompilacji programów komputerowych w świetle prawa, Linux+ 2002, nr. 64

3. Reżimem ochrony właściwym dla programów komputerowych objęta jest również dokumentacja projektowa w zakresie, w jakim wyrażony w niej został sam program. Ponadto dokumentacja projektowa oraz dokumentacja użytkownika może podlegać ochronie w szerszym zakresie na zasadach ogólnych prawa autorskiego.

4. Zobacz również art. 9 ust. 2 TRIPS

5. A. Nowicka, Prawnoautorska i patentowa . . . j. w., str. 118

6. zobacz np. U. Bath, Access to Information v. Intellectual Property Rights, E.P.I.R., 2002/3, str. 142

7. A. Nowicka, Prawnoautorska i patentowa . . . j. w., str. 121

8. J. Barta, R. Markiewicz, Programy komputerowe, Rzeczpospolita 1994.10.11, A. Nowicka, Prawnoautorska i patentowa . . . j. w., str. 70, str. 121, M. Byrska, Ochrona programu komputerowego w nowym prawie autorskim, Warszawa 1994, str. 79

9. Oczywiście mam na myśli sytuacje, gdy dysponujemy wyłącznie kodem wynikowym programu. W przypadku wolnego oprogramowania problemy tego typu nie powstają.

10. W artykule nie podejmuję problemu dopuszczalnej dekompilacji w ramach wykonywania czynności, do których uprawniony jest użytkownik na gruncie art. 75 ust. 1 pr. aut.

11. W oparciu o przepisy Dyrektywy zwraca na to uwagę: D. S. Karjala, Recent United States and International Development in Software Protection, E.I.P.R., 1994/1, str. 19

12. Cele którym może służyć reverse engineering omawia np. Cem Kaner, The Problem of Reverse Engineering, `http://www.kaner.com/pdfs/reveng.pdf`

13. J. Barta, R. Markiewicz w: J. Barta, M. Czajkowska-Dąbrowska, Z. Ćwiąkalski, R. Markiewicz, E. Traple, Komentarz do ustawy o prawie autorskim i prawach pokrewnych, Warszawa 1995, str. 359

14. Pogląd taki na gruncie Dyrektywy prezentuje np. A. Nowicka, Prawnoautorska i patentowa . . . j. w., str. 90

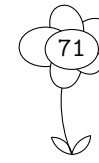
15. Tak. J. Sobczak, Prawo autorskie i prawa pokrewne, Warszawa-Poznań 2000, str. 183 Natomiast M. Byrska dopuszcza nałożenie na licencjobiorcę w drodze zapisów umownych, obowiązku zwrócenia się do licencjodawcy o udostępnienie koniecznych informacji. M. Byrska, Ochrona programu komputerowego. j. w., str. 69

16. E. R. Kroker, The Computer Directive and the Balance of Rights, E.P.I.R., 1997 r/5., str. 250

17. j. w., str. 250

18. Postanowienie S. Apel. w Poznaniu z dnia 4 stycznia 1995 r., I ACr 422/94, zamieszczone w: J. Barta, R. Markiewicz, Prawo autorskie. Przepisy, Orzecznictwo, Umowy międzynarodowe, Warszawa 2002 r, str. 1067

19. J. Barta, R. Markiewicz, Główne problemy . . . j. w., str. 52

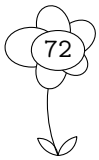


Samba, Windows NT/2000 i relacje zaufania

Marcin Szcześniak

Wraz z coraz głębszym wnikaniem Samby w środowisko systemów Windows NT/2000/XP wzrasta poziom wsparcia funkcjonujących w nich domen. Jednym z istotnych ich elementów są relacje zaufania które pozwalają na dokonywania uwierzytelniania użytkowników w sieci.

Pierwszą chronologicznie i najlepiej opracowaną relacją jest przynależność do domeny Windows NT. Wymaga ona posiadania przez serwer Samba konta na głównym kontrolerze domeny (PDC). Przy wszelkich operacjach wykonywanych przez użytkowników domeny na takim serwerze, Samba rozstrzyga kwestie uwierzytelniania „odpytując" w odpowiedni sposób PDC. Musi więc znać szczegółowe dane konta, którym się może posłużyć, aby taką operację przeprowadzić. Dane takie są organizowane w postaci struktury nazywanej wewnętrznie *machine account password*. Nazwa jest już trochę historyczna, bo nie odzwierciedla rodzaju relacji zaufania. Chodzi o to, że główny kontroler domeny musi posiadać konto „zaufanej maszyny" (ang. *trusted machine*), aby relacja, czyli w tym przypadku przynależność do domeny, mogła funkcjonować. Struktura zaimplementowana w Sambie przechowuje hash hasła NT, oraz czas jego ostatniej modyfikacji. Czas ten jest potrzebny do okresowej zmiany hasła przez maszynę należącą do domeny, czyli w tym przypadku serwer Samba. Domyślnie zmiana taka następuje co 7 dni. Całość jest obecnie przechowywana



w pliku `secrets.tdb`, lecz wkrótce przejdzie do zaplecza bazy kont użytkowników (ang. *passdb backend*) aby umożliwić jej rozpowszechnianie na serwerach BDC poprzez replikację.

Drugą relacją jest przynależność do domeny Active Directory. Nie jest to obsługiwane przez obecną stabilną wersję Samby, lecz wersję deweloperską, która ma być wydaniem 3.0. Posługuje się już nią bardzo wiele instytucji, włącznie z firmami oferującymi tzw. serwery NAS (ang. *Network Appliance Storage*). Tutaj, podobnie jak w domenach NT, Samba również musi użyć odpowiedniego hasła skojarzonego z kontem, żeby podłączyć się do kontrolera domeny i przeprowadzić uwierzytelnianie. Mechanizmy, które leżą u podstaw tego procesu są jednak zupełnie inne niż w poprzednim przypadku. Podstawową jest zamiana uwierzytelniania NTLM, na sposoby jakie oferuje Kerberos V. Kontroler domeny Active Directory pełni jednocześnie rolę Centrum Dystrybucji Kluczy (ang. *KDC – Key Distribution Center*) wydającego tzw. bilety kerberos (ang. *kerberos tickets*). W tej sytuacji domena Windows 2000 jest jednocześnie realmem kerberos. Rolę bazy kont użytkowników i zaufanych maszyn pełniła dotychczas baza danych SAM (ang. *Security Access Manager*), natomiast teraz służy do tego katalog LDAP. Dzięki temu, że Kerberos obsługuje koncepcję „współdzielonego sekretu" (ang. *shared secret*) możliwe jest koncepcyjne odtworzenie mechanizmów zaufania które istniały w domenach Windows NT. W sytuacji gdy Samba jest członkiem domeny Active Directory, jedyne konieczne do zaimplementowania części, to obsługa specyficznego dla serwerów Microsoft Windows uwierzytelniania Kerberos V oraz nowy mechanizm jego negocjowania (SPNEGO) na poziomie połączeń SMB. Z tym wiąże się konieczność użycia innych struktur danych w celu przechowywania hasła skojarzonego z kontem serwera Samba w domenie ADS (Active Directory), jak również konieczność zaimplementowania wsparcia identyfikatorów obiektów (OID) które są wymieniane podczas negocjacji SPNEGO. Podobnie jak w poprzednim przypadku, hasło jest przechowywane w chwili obecnej w pliku secrets.tdb.

Trzecim rodzajem są relacje zaufania między domenami WinNT. Działają one podobnie do zaufania obowiązującego w przypadku przynależności do domeny. Różnica jest taka, że relacje te przekraczają granice domen, więc zobowiązane do ich respektowania są główne kontrolery, a nie poszczególne stacje robocze. Międzydomenowe uwierzytelnienie polega na zweryfikowaniu poprawności podanych informacji (użytkownik/hasło) w imieniu konkretnego użytkownika. Weryfikacji takiej dokonuje kontroler domeny obcej na kontrolerze domeny macierzystej. Aby możliwe było dopuszczenie lub zabronienie takiego procesu, istnieją relacje zaufania między domenami. Są one w rzeczywistości specyficznymi kontami w bazie SAM (podobnymi do kont zaufanych

maszyn) i komplementarnymi dla nich hasłami przechowywanymi w tej bazie, lecz „po drugiej stronie". Dzięki temu relacje te nie są dwukierunkowe, tzn. zaufanie w jedną stronę nie implikuje zaufania w drugą. Wykorzystując takie konta, kontrolery domen podłączają się do siebie aby móc przeprowadzić zdalne uwierzytelnienie użytkownika który korzysta z zasobów domeny obcej. Zarówno obsługa kont zaufanej domeny, jak i haseł „w drugą stronę" jest zaimplementowana w deweloperskiej wersji Samby (wzajemne uwierzytelnianie jest na etapie testów). Te pierwsze są przechowywane w zapleczu kont (ang. *passdb backend*) z odpowiednią flagą, która oznacza że są to konta zaufanych domen (ang. *interdomain trusted account*). W ten sposób Samba może być kontrolerem domeny zaufanej. Z kolei hasła znajdują się w pliku `secrets.tdb` w postaci struktur o nazwie 'trusted domain password'. Zawierają m.in. nazwę domeny zaufanej, hasło do konta i datę ostatniej zmiany.

Czwartym typem są relacje zaufania między domenami Active Directory. Bazują one również na uwierzytelnianiu Kerberos i wykorzystywanym w nim mechanizmie „współdzielonego sekretu". W przeciwieństwie do domen NT jednak, nie są to relacje jednokierunkowe. Do tego, aby obydwie domeny (a raczej ich kontrolery) ufały sobie wzajemnie, wystarczy nawiązanie jednej relacji zaufania, a nie, jak w przypadku NT, dwóch. Są one przechowywane w katalogu LDAP jako odpowiedni rodzaj konta i umożliwiają w ten sposób łączenie ze sobą drzew domen tworząc las domen active directory (ang. *forest*). W chwili obecnej, Samba może pracować tylko jako członek domeny ADS (Active Directory), więc opieka nad relacjami zaufania spada całkowicie na kontroler ADS. Trwają jednak prace nad implementacją trybu kontrolera domeny i wówczas konieczny do pracy będzie również serwer LDAP (najczęściej OpenLDAP). Do tego czasu może również zostać wydana nowa wersja serwera Kerberos V obsługująca algorytm szyfrowanie wprowadzony przez firmę Microsoft. Dzięki czemu jego implementacja wewnątrz Samby nie będzie już potrzebna. Naturalne jest także, że wówczas opieka nad wszystkimi biletami Kerberos które będą potrzebne do komunikacji ze środowiskiem ADS, przejdzie do pliku keytab który jest standardowym elementem systemu Kerberos.

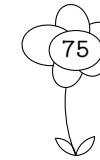
W związku z wieloma podobieństwami w opisanych relacjach trwają obecnie prace nad uogólnieniem sposobu przechowywania haseł do zaufanych kont, niezależnie od tego, jakiego rodzaju jest to konto. W ten sposób, takie same struktury mogą być umieszczane tymi samymi funkcjami najpierw w pliku secrets.tdb, a potem bez większych kłopotów w bazie kont użytkowników w celu umożliwienia ich replikowania.

Zastosowanie systemu Linux do przetwarzania dźwięku w czasie rzeczywistym

Grzegorz Borowiak <grzes@gnu.univ.gda.pl>
Wojciech Piechowski <wojt@gnu.univ.gda.pl>

STRESZCZENIE: Chociaż Linux nie jest systemem operacyjnym czasu rzeczywistego, w większości przypadków z powodzeniem nadaje się do obróbki dźwięku w czasie rzeczywistym. W takim przetwarzaniu najistotniejsze jest opóźnienie (ang. *latency*), tzn. czas, który upływa od pojawienia się próbki na wejściu do pojawienia się jej na wyjściu. Czas ten musi być stały, ponieważ próbki pojawiają się na wejściu w stałym tempie i w takim muszą się pojawiać na wyjściu. Dlatego trzeba założyć pewien limit opóźnienia taki, że system na pewno go nie przekroczy. Praca przedstawia różne techniki, jakimi można się posłużyć, by zminimalizować ten limit. Oczywiście zakładamy, że wątek (wątki) zajmujące się przetwarzaniem dźwięku są jedynymi aktywnymi wątkami w systemie, tzn. na komputerze przetwarzającym dźwięk nie chodzi w tle łamanie haseł, ani szukanie kosmitów, ani nikt nie gra na nim w Quake'a. W takim wypadku ów założony limit musiałby być bardzo duży. Zajmować się będziemy realizacją przetwarzania przy użyciu OSS (Open Sound System).

TRZY RÓŻNE PODEJŚCIA DO CYFROWEGO PRZETWARZANIA DŹWIĘKU



PODEJŚCIE `read/write`

Jest to najbardziej podstawowe podejście dla sprzętu typu PC i działa z każdą kartą dźwiękową obsługującą full-duplex. Polega na tym, że karta dźwiękowa odczytuje pewną ustaloną liczbę próbek z wejścia (tzw. fragment), następnie cały taki fragment jest przetwarzany, a na koniec wysyłany na wyjście karty dźwiękowej. Wszystko to oczywiście dzieje się w pętli.

Karta dźwiękowa jest tak skonstruowana, że wysyła przerwanie po odsłuchaniu lub odegraniu każdego fragmentu. Te przerwania są obsługiwane przez sterowniki OSS, które z kolei przetwarzają je na zdarzenia związane z urządzeniem `/dev/dsp` w taki sposób, że mogą budzić wątki oczekujące na te zdarzenia, za pośrednictwem wywołań systemowych `read()`, `write()` i `select()`. Na początku za pomocą wywołań `ioctl()` definiujemy rozmiar pojedynczego fragmentu oraz liczbę fragmentów, które mogą zostać nadane nieblokująco. Jeżeli, na przykład, ustalimy liczbę fragmentów na 2, a rozmiar fragmentu na 64 próbki, to jeżeli bufor wyjściowy sterownika jest pusty, to nieblokująco można zapisać (czyli zlecić do odegrania) 128 próbek. Zapisanie 129. próbki będzie już blokujące – po przyjęciu 128 próbek urządzenie `/dev/dsp` przestaje być gotowe do zapisu. Zaczyna być gotowe dopiero po odegraniu pierwszego fragmentu, tzn. 64 próbek.

Dzięki temu mechanizmowi mamy załatwiony tzw. *double buffering* – możemy spokojnie nadać kolejny, $(n+1)$-szy fragment do odegrania, a w tym czasie karta odgrywa n-ty, przed chwilą skończywszy odgrywać $(n-1)$-szy. To częściowo zapobiega zjawisku zwanemu *underrun* – sytuacji, gdy karta nie dostaje we właściwym czasie danych do odegrania. Jakiś niezerowy czas bowiem na pewno upłynąłby od końca odgrywania fragmentu, poprzez wygenerowanie przerwania przez kartę dźwiękową, jego obsługę przez jądro, obudzenie właściwego wątku do nadania przez ten wątek kolejnych danych.

Pseudokod przetwarzania dźwięku przy takim podejściu wygląda następująco:

```
sample i_buf[fragment_size];
sample o_buf[fragment_size];
while (1) {
       read(/dev/dsp, i_buf, fragment_size);
       process(o_buf, i_buf, fragment_size);
       write(/dev/dsp, o_buf, fragment_size);
}
```


Nie będziemy tutaj szczegółowo omawiać API OSS, tzn. jakie dokładnie powinny być wywołania systemowe do obsługi karty i jakie ich parametry. Można o tym poczytać w „OSS Programmer's Guide" [1] i w innych dokumentach.

Wadą takiego podejścia jest stosunkowo duże opóźnienie. Bierze się to z zależności czasowych pomiędzy fazami odczytu, przetwarzania i zapisu. Ani jedna próbka z danego fragmentu nie może zostać przetworzona dopóki cały fragment nie zostanie odczytany. Z kolei zanim fragment zostanie nadany na wyjście, musi zostać w całości przetworzony. W związku z tym przetworzona

próbka pojawi się na wyjściu nie wcześniej niż suma długości czasu trwania pojedynczego fragmentu oraz czasu przetwarzania tego fragmentu.
W dalszych rozważaniach przyjmiemy następujące oznaczenia:

- F długość fragmentu, wyrażona jako czas (czyli liczba próbek we fragmencie podzielona przez częstotliwość próbkowania).
- $p(F)$ najdłuższy możliwy czas maszynowy przetwarzania fragmentu długości F. Czas maszynowy – tzn. taki, który upłynąłby, gdyby procesor nie zajmował się niczym innym, nawet przerwania byłyby zablokowane. Oczywiście, w typowym przypadku $p(F) < F$.
- $q(F)$ najdłuższy możliwy czas rzeczywisty przetwarzania fragmentu długości F. Czas rzeczywisty – tzn. taki, jaki rzeczywiście może upłynąć od rozpoczęcia przetwarzania fragmentu do zakończenia jego przetwarzania.
- $d(F)$ czas rzeczywisty jaki upłynie od pojawienia się próbki na wejściu karty dźwiękowej do pojawienia się jej przetworzonej na wyjściu, przy zastosowania fragmentu długości F.
- T okres wewnętrznego zegara schedulera. Typowo wynosi on 10ms, ale można go zmienić – o tym w dalszej części dokumentu.

Długość pojedynczego fragmentu nie może być krótsza niż okres schedulera, czyli $F \geq T$. W przypadku próby użycia krótszego fragmentu występuje *underrun* – objawia się to „rwaniem" się sygnału.

Z kolei $q(F) = p(F) + T$. Dzieje się tak dlatego, że nawet jeżeli nasz wątek ma bardzo wysoki priorytet, w środowisku wielozadaniowym ogólnego zastosowania takim jak Linux, zawsze istnieje możwliość przyznania kwantu czasu procesowi o niższym priorytecie. Jeżeli nastąpi to podczas przetwarzania fragmentu, ten kwant czasu doda się do czasu maszynowego jego przetwarzania.

Z tego wszystkiego płynie wniosek, że:

$$d(F) \geq F + q(F) = F + p(F) + T \geq p(F) + 2T.$$

To jest dość sporo, zwłaszcza jeżeli mamy zamiar robić dość procesorochłonne przetwarzanie. Jeżeli $p(F)$ jest bliskie F, to $d(F)$ jest bliskie $3T$.

Podejście MMAP

Takie podejście polega na bezpośrednim odczytywaniu próbek z bufora DMA odczytu i zapisywaniu do bufora DMA zapisu. W odróżnieniu od na przykład ALSA, OSS pozwala na taki dostęp. Polega to na tym, że jednocześnie w pętli

wykonują się zapis i odczyt. Karta dźwiękowa odbiera próbkę po próbce z wejścia i zapisuje je kolejno do bufora wejściowego. Wątek przetwarzający aktywnie czeka na kolejne pojawiające się w buforze wejściowym próbki (*polling*) i przetworzywszy je, zapisuje na kolejnych pozycjach w buforze wyjściowym, który jest przeznaczony do wyemitowania przez kartę dźwiękową. Zapis i odczyt do i z buforów odbywa się w „samopowtarzalnej" pętli. Nie trzeba za każdym razem wznawiać odczytu/zapisu, tym zajmuje się sterownik OSS. W tym przypadku pseudokod wyglądałby następująco:

```
sample i_buf[buffer_size]={ UNREACHABLE, UNREACHABLE, ... };
sample o_buf[buffer_size]={ 0, 0, ... };
int seek = 0;
read_in_loop(/dev/dsp, i_buf, buffer_size);
write_in_loop(/dev/dsp, o_buf, buffer_size);
while (1) {
       while (i_buf[seek] == UNREACHABLE);
       o_buf[(seek+LATENCY*sample_rate)%buffer_size] =
                 process(i_buf[seek]);
       seek++;
       if (seek==buffer_size) seek=0;
}
```

gdzie `UNREACHABLE` jest wartością, która nigdy nie powinna pojawić się na wejściu karty dźwiękowej. Jeżeli próbki są 16-bitowe ze znakiem, to `UNREACHABLE` może być równe `0x7fff`, co odpowiada przesterowaniu sygnału wejściowego, a to w dobrze skonfigurowanym systemie nigdy nie zajdzie.

Proces działający według przedstawionego wyżej algorytmu byłby użyteczny, gdyby nie to, że w takiej postaci zużywa się cały dostępny czas procesora. Z jednej strony proces musi mieć wysoki priorytet, aby nie zostawiać procesora na zbyt długo, z drugiej strony przy wysokim priorytecie zagłodziłby pozostałe wątki.

Rozwiązaniem tego problemu jest drobna przeróbka, polegająca na tym, że wątek ten przekazuje procesor następnemu wątkowi w kolejce, ale pozostaje aktywny. Służy do tego systemowe wywołanie `sched_yield()`. Przerobiony pseudokod wygląda następująco:

```
sample i_buf[buffer_size]={ UNREACHABLE, UNREACHABLE, ... };
sample o_buf[buffer_size]={ 0, 0, ... };
int seek = 0;
read_in_loop(/dev/dsp, i_buf, buffer_size);
write_in_loop(/dev/dsp, o_buf, buffer_size);
```

```
while (1) {
       while (i_buf[seek] == UNREACHABLE) sched_yield();
       o_buf[(seek+latency*sample_rate)%buffer_size] =
                 process(i_buf[seek]);
       seek++;
       if (seek==buffer_size) seek=0;
}
```

Wartość `latency` w powyższym pseudokodzie jest to całkowity czas opóźnienia. Ten czas z oczywistych względów nie może być krótszy niż okres schedulera – na tyle bowiem wątek może rozstać się z procesorem i przez cały ten czas karta dźwiękowa musi mieć w buforze wyjściowe dane do emitowania. I jest to właściwie jedyne ograniczenie – czas przetwarzania pojedynczej próbki jest pomijalnie mały.

Istnieje możliwość uzyskania jeszcze krótszego opóźnienia. Możemy „oddać" procesor na mniej niż okres schedulera, na przykład na 25% tego okresu, nie wywołując od razu `sched_yield()`, lecz dopiero po czasie równym 75% okresu schedulera. To oczywiście pociągnie za sobą zwiększone zużycie procesora, ale nie zagłodzenie systemu. Może to jednak znacznie opóźnić reakcje systemu.

Pseudokod procesu wykorzystującego tę metodę mógłby wyglądać następująco:

```
sample i_buf[buffer_size]={ UNREACHABLE, UNREACHABLE, ... };
sample o_buf[buffer_size]={ 0, 0, ... };
int seek = 0;
int samples_to_wait = sample_rate * (sched_timeslice - spare_time);
if (samples_to_wait <= 0) samples_per_cycle = 1;
read_in_loop(/dev/dsp, i_buf, buffer_size);
write_in_loop(/dev/dsp, o_buf, buffer_size);
while (1) {
       int samples_ready=0;
       while (i_buf[seek] != UNREACHABLE) {
              samples_ready++;
              seek++;
              if (seek==buffer_size) seek=0;
       }
       for (int i=0; i<samples_ready+samples_to_wait; i++) {
              while (i_buf[seek] == UNREACHABLE);
              o_buf[(seek+latency*sample_rate)%buffer_size] =
                        process(i_buf[seek]);
```

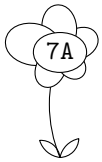
```
                seek++;
                if (seek==buffer_size) seek=0;
        }
        sched_yield();
}
```

Przeanalizujmy działanie powyższego algorytmu. Po uzyskaniu dostępu do procesora proces zlicza próbki gotowe do przetwarzania, czekające w buforze. Następnie proces rozpoczyna przetwarzanie kolejnych próbek. Najpierw przetwarzane są te, które czekały w buforze – jest ich `samples_ready`. Następnie przetwarzane jest jeszcze `samples_to_wait` próbek, których nie było w buforze w chwili zliczania i na które proces czeka aktywnie (część z nich oczywiście mogła się w międzyczasie pojawić w buforze). Po przetworzeniu ostatniej z tych próbek proces oddaje procesor.

Zauważmy, że proces oddaje procesor na czas co najwyżej `spare_time`. Między kolejnymi początkami przebiegu pętli mija zawsze czas równy kwantowi schedulera (tutaj reprezentowanemu przez zmienną `sched_timeslice`). Pojedynczy przebieg pętli trwa co najmniej tyle, ile trwa dostarczenie przez kartę dźwiękową `samples_to_wait` próbek. Wartość `samples_to_wait` jest akurat tak dobrana, żeby czas dostarczania tylu próbek trwał `sched_timeslice` – `spare_time`, i tyle czasu procesora z każdego okresu schedulera zajmować będzie proces przetwarzający dźwięk. Pozostałe `spare_time` czasu procesora zostanie oddane pozostałym procesom. Oczywiście, w tym przypadku latency musi wynosić co najmniej `spare_time`.

Nie tylko my doceniamy użyteczność metody MMAP. Docenili ją również twórcy OSS, w ogóle udostępniając API do robienia takich rzeczy. Metoda ta stosowana jest również w grach – na przykład w Quake. Niestety, nie doceniła tego podejścia ekipa tworząca system ALSA – tam nie ma rodzimego API do tej metody, a jest ona dostępna jedynie poprzez emulację OSS. No cóż, dobre i to.

Podejście przerwaniowe



To podejście w zasadzie nie nadaje się do wykorzystania na sprzęcie typu PC, ponieważ karty dźwiękowe nie są odpowiednio do tego celu skonstruowane – nie można w dowolnej chwili odczytać stanu na wejściu, ani ustalić stanu na wyjściu karty. Podejście przerwaniowe pasuje raczej do systemów wbudowanych, dedykowanych, a wspominamy o nim tylko dla kompletności opracowania.

Przy zastosowaniu takiego podejścia kod przetwarzający pojedynczą próbkę wywołujemy bezpośrednio z procedury obsługi przerwania zegarowego, przy czym zegar tyka z częstotliwością próbkowania. Procedura obsługi przerwania

odczytuje bieżący stan na wejściu audio do systemu (poprzez chodzący non-stop przetwornik D/A), przetwarza ją i podaje na wyjście audio (poprzez chodzący non-stop przetwornik A/D).

Pseudokod procedury obsługi przerwania wygląda następująco:

```
clock_intr()
{
       static sample last_output;
       sample current_input;
       outport(ADC_port, last_output);
       current_input = inport(DAC_port);
       last_output = process(current_input);
}
```

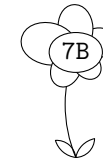
Z uwagi na niemożność wykorzystania tej metody przy wykorzystaniu PC ze standardowymi kartami dźwiękowymi nie będziemy rozwijali tego tematu.

Przystosowanie jądra systemu do przetwarzania dźwięku

Jak już zostało wykazane, bardzo istotną z punktu widzenia przetwarzania dźwięku cechą jest ziarnistość schedulera. Standardowo scheduler systemu Linux ma okres 10 ms. W przypadku podejścia pierwszego dawałoby to czas opóźnienia około 30 ms. W niektórych zastosowaniach jest to nie do zaakceptowania.

Pomocna jednak okazuje się łata Roberta Love'a. Umożliwia ona sterowanie schedulerem. Steruje się go wpisując różne liczby do poszczególnych plików katalogu `/proc/sys/sched`. Są tam następujące pliki:

- `child_penalty`
- `interactive_delta`
- `max_sleep_avg`
- `max_timeslice`
- `min_timeslice`
- `parent_penalty`
- `prio_bonus_ratio`
- `starvation_limit`
- `thread_penalty`
- `user_penalty`



Nie będziemy tutaj szczegółowo omawiać znaczenia wszystkich tych wartości. Można o tym poczytać na przykład w `/usr/src/linux/Documentation/filesystems/proc.txt` [2] (Uwaga! Informacje o nowym schedulerze pojawią się tam dopiero po zainstalowaniu łaty z tym schedulerem!).

Z punktu widzenia naszego zagadnienia najbardziej istotne są `max_timeslice` i `min_timeslice`. Są to wielkości kwantów czasu przydziela-

nych przez scheduler wątkom o odpowiednio najwyższym i najniższym priorytecie, obydwie mierzone w milisekundach. Dla przetwarzania dźwięku w czasie rzeczywistym najlepiej jest, gdy obydwie te wartości są ustawione na 1ms.

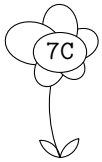
Aby jednak takie ustawienia zadziałały, konieczna jest zmiana taktowania jądra z 100Hz (domyślnie) na 1000Hz. Tego niestety nie da się zrobić w czasie pracy – konieczna jest rekompilacja jądra, i to niestety całego. Jest tak dlatego, że scheduler nie może działać z rozdzielczością większą niż zegar jądra. Ale na szczęście taka rekompilacja jest jednorazowa i niczym w zasadzie nie grozi – trochę więcej czasu procesor będzie zużywał na obsługę przerwania zegarowego, ale jeżeli zapragniemy przywrócić dłuższe kwanty czasu, możemy to zrobić bez problemu wpisując odpowiednie wartości do `/proc/sys/sched`.

Domyślna częstotliwość 100Hz jest utrzymywana głównie z powodów historycznych. Kiedyś komputery były powolniejsze i czas przełączania kontekstu mógł trwać nawet milisekundę. Obecnie ten czas jest znacznie krótszy i można bez żadnego problemu stosować częstotliwość 1000Hz.

Bibliografia

[1] *Open Sound System Programmer's Guide*, Jeff Tander, `http://www.opensound.com/pguide/oss.pdf`

[2] `/usr/src/linux/Documentation/filesystems/proc.txt`

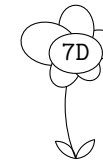


Kompresja wideo pod Linuksem

Marcin Kwadrans <quar@3miasto.net>

STRESZCZENIE: Dla systemu Linux, wbrew przekonaniu niektórych, istnieje całkiem spora liczba narzędzi umożliwiających kompresję strumieni wideo. Skompresowane dane `audio/wideo` mogą być przechowywane w ustandaryzowanych plikach typu MPEG (`ffmpeg`, `fame`). Możliwe jest też „upychanie" danych multimedialnych, zapewniające ich prawidłową synchronizację, a nie narzucające im konkretnego formatu, tzw. kontenery (`avi`, `ogm`). Wśród kodeków króluje `divx` z wieloma dostępnymi implementacjami (DIVX 4/5, OpenDivx, XVid, kodek zawarty w bibliotece `libavcodec`). W samym procesie kompresji można wykorzystać wiele różnych narzędzi (`x2divx`, `ffmpeg`, `mencoder`). Istnieją również kombajny posiadające przejrzysty interfejs graficzny (`avidemux`).

Sekwencje wideo w postaci nieskompresowanej zajmują relatywnie dużo miejsca na dysku. Dla przykładu – aby przechować minutę animacji w rozdzielczości 352x288 przy 25 klatkach na sekundę potrzeba ok. 420MB przestrzeni dyskowej. Oczywistym staje się fakt, że kompresja staje się niezbędna. Użycie takich samych algorytmów, jak przy kompresji zwykłych danych (np. algorytmu Huffmana) pozwalało na zaoszczędzenie zaledwie kilku procent zajętego miejsca przez animację. Była to kompresja bezstratna. By ograniczyć rozmiar pliku, trzeba było zrezygnować częściowo z jakości. Pewnym rozwiązaniem okazało się zastosowanie technik używanych w formacie JPEG. Nowy format uzyskał nazwę MJPEG (Motion JPEG). Następnym krokiem było wzięcie pod uwagę faktu, że animacje składają się z szeregu poruszających się fragmentów obrazu, można więc spróbować ten ruch opisać pewnymi wzorami. Dzięki połączeniu możliwości MJPEG i kompensacji ruchu powstał format MPEG (w trzech wersjach), a w końcu bardzo popularny DIVX. Oczywiście historia ta jest bardzo skrótowa i pominąłem tutaj wiele technik. Szczególnie dużo nowych rozwiązań i oprogramowania było dostępnych wyłącznie dla systemu Microsoft Windows, ewentualnie na platformę Apple Macintosh. Jak sytuacja wygląda pod Linuksem? Jeśli chodzi o kodeki używające MJPEG i standardu MPEG 1/2, to tutaj



niemal od początku *niksy, w tym Linux, nie odstawały od konkurentów. Formaty te są nieźle opisane i łatwo było stworzyć narzędzia odtwarzające, a nawet dokonujące kompresji (np. `mpegencode` – program stworzony na Uniwersytecie w Berkeley). W nowszych rozwiązaniach często nie jest to łatwe i nawet do tej pory możliwe czasem tylko przez emulację środowiska Windows (np. Intel Indeo, pierwsze wersje DIVX).

MPEG

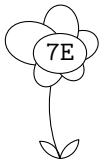
Istnieje kilka wersji MPEG:

`MPEG1` umożliwia kompresję obrazu w niskich rozdzielczościach przy stosunkowo niskiej wartości bitrate (np. 352X288 jakość podobną do VHS otrzymuje się przy bitrate ok 1.5Mbit na sekundę);

`MPEG2` używane do nieco wyższych rozdzielczości (np. 720X480 przy jakości CCIR-601 uzyskuje się przy bitrate od 15 Mbit na sekundę);

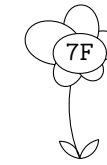
`MPEG4` umożliwia obniżenie bitrate i dalsze zwiększenie rozdzielczości.

Pliki MPEG z reguły mają rozszerzenie `.mpg`, `.mpeg`, `.m1v`, `.m2v` lub `.dat`, w zależności od wersji i użytego programu do kompresji. Istnieje dość szeroki wachlarz kompresorów MPEG: `fame`, `ffmpeg`, `mpeg2enc` z pakietu `mjpegtools`, transcode, mencoder. Część z nich wykorzystuje zewnętrzne biblioteki do przeprowadzenia kompresji. Różnią się one między sobą dość znacznie, nie tylko pod względem prędkości i jakości (w tym względzie moim faworytem jest `ffmpeg`), ale również mnogością formatów wejściowych (tutaj chyba najlepszy okazuje się `mencoder`). Tylko część z nich umożliwia jednoczesną kompresję dźwęku (`ffmpeg`, `transcode`). Format MPEG 1/2 wymusza przechowywanie dźwięku w postaci MP2. Jest to stosunkowe stare i mało popularne rozwiązanie. Teoretycznie nie ma większych przeciwskazań by użyć bardziej popularnego MP3, niestety w praktyce tylko nieliczne programy są w stanie odtworzyć takie pliki (`mplayer`). Istnieje również możliwość kompresji do formatów `VCD/SVCD`, które umożliwią późniejsze nagranie na CD (`ffmpeg`, `mpeg2enc`, `transcode`). Taką płytę można odtworzyć w większości domowych odtwarzaczy. Osobiście do kompresji do MPEG używam `transcode`, który umożliwia kompresję poprzez wiele różnych bibliotek, ma obsługę dźwięku i akceptuje wiele formatów wejściowych.



KODEKI, KODEKI...

Nazwa „kodek" nie jest jednoznaczna. Używa się jej do określenia programów realizujących pewien algorytm kompresji (audio lub wideo) lub jako nazwę tego „algorytmu". Tutaj będę używał tego słowa w tym drugim znaczeniu. Implementacje niektórych kodeków są dostępne wyłącznie pod systemem Windows, a kompresja przy ich użyciu może być możliwa tylko przez emulację tego środowiska (używany jest fragment kodu `wine`). Jedynym znanym mi programem (i biblioteką) to umożliwiającą jest `avifile`. Inne narzędzia często pozwalają na kompresję przy jej użyciu. Przykładami mogą być `transcode` i `x2divx`. Transcode przy użyciu avifile umożliwia kompresję do DIVX3 oraz Intel Indeo 5. Niemniej jednak nie ma czego żałować, bo istnieje kilka sprawdzonych rozwiązań, które są dostępne pod Linuksem (MJPEG, DIVX). `ffmpeg` umożliwia kompresję do stosunkowo dużej liczby różnych kodeków (DIVX 3, DIVX 4/5, WMV7, H263, MJPEG). Wielu autorów oprogramowania (np. `mencoder` czy `transcode`), którzy nie stworzyli własnych implementacji, korzysta z biblioteki zawartej w `ffmpeg`. DIVX 3 powstał poprzez dostosowanie MPEG4 firmy Microsoft do obsługi plików AVI. Z powodu pewnych kontrowersji związanych ze sprawą licencjonowania DIVX, kodek ten nie wydostał się poza grono hobbystów. Kompresję przy jego użyciu umożliwiają `ffmpeg` i `x2divx`. `ffmpeg` jest znacznie szybszy. W chwili obecnej DIVX3 jest wypierany przez nowsze wersje. DIVX 4/5 jest w pełni kompatybilny z standardem MPEG 4. Podobnie jak jego poprzednik, umożliwia przechowywanie skompresowanych danych wideo w plikach AVI. W chwili obecnej pod Linuksem istnieje kilka różnych implementacji tego kodeka: `OpenDivx`, `libavcodec`, `Xvid` oraz `divx5`. `OpenDivx` nie jest już rozwijany, ale jego kod stał się podstawą do pisania innych implementacji. Moim zdaniem najlepszą jakość zapewnia `libavcodec`. Z drugiej strony zauważyłem, że `libavcodec` ma problemy z kompresją szybko zmieniających się scen (tzw. *High Motion*). Dodatkową zaletą przemawiającą na korzyść `libavcodec` jest dobra obsługa kompresji dwuprzebiegowej. `Divx5` jest oprogramowaniem z zamkniętym kodem źródłowym i w związku z tym prawdopodobnie mniej zoptymalizowanym pod kątem obsługi konkretnych procesorów, poza tym wynikowa jakość obrazu jest tu zauważalnie niższa niż w przypadku dwóch pozostałych. Być może dobrym kompromisem staje się więc `Xvid`.

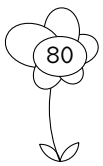


KONTENERY

Kodeki umożliwiają kompresję wideo lub audio, lecz skompresowane dane nie są przystosowane do przechowywania w pliku. Konieczne staje się więc zapakowanie tych danych do pewnego pojemnika. Rolę takiego pojemnika pełni plik

zwany kontenerem. Kontener sam w sobie nie definiuje w żaden sposób formatu w nim przechowywanego. Dodatkowym jego zadaniem jest przechowywanie informacji dotyczących synchronizacji dźwięku i obrazu. Do kontenerów należy zaliczyć ASF, AVI oraz najnowszy OGM. ASF jest mało popularny pod Linuksem ze względu na ograniczenia patentowe. Jedynym znanym mi narzędziem, które umożliwia zapis do pliku ASF jest `ffmpeg`. Najczęściej używane jest AVI. AVI nie było projektowane pod kątem szerokiego wsparcia dla różnych kodeków. Początkowo było używane wyłącznie do przechowywania nieskompresowanych danych. Pliki AVI charakteryzują się dużym narzutem miejsca związanego z synchronizacją i formatowaniem. OGM powstało niejako przy okazji projektowania formatu OGG Vorbis. OGG okazał się na tyle elastyczny, by umożliwić umieszczenie w nim nie tylko dźwięku, ale również danych wideo. Dźwięk nie musi być koniecznie przechowywany w OGG Vorbis, możliwe jest użycie innych formatów np. AC3 czy MP3. Pliki OGM, oprócz mniejszego narzutu na formatowanie, cechują się bardziej precyzyjnym przewijaniem oraz lepszą obsługą kodeków dźwięku ze zmiennym *bitrate*. OGM jest również bardziej tolerancyjny na błędy, np. uszkodzone fragmenty pliku mogą być pominięte bez konieczności przerywania odtwarzania. Dodatkowo OGM potrafi przechowywać dodatkowy strumień napisów. W przypadku plików AVI jest to możliwe wyłącznie przez wczytanie napisów z osobnego pliku. Dla niektórych dużą zaletą może być brak powiązań OGM z firmą Microsoft. Niestety w chwili obecnej nie spotkałem żadnego programu, który potrafiłby kompresować dane bezpośrednio umieszczając je w tym kontenerze. Przeważnie niezbędne jest zapisanie obrazu w pliku AVI, dźwięku w formacie PCM, AC3, MP3 lub OGG, a dopiero później, przy pomocy narzędzi zawartych w pakiecie ogmtools, umieszczenie danych w kontenerze OGM. Gotowy plik można odtworzyć w większości popularnych odtwarzarek, osobiście, z dobrym skutkiem, używam do tego celu programu `mplayer`.

Nie lubię konsoli



Wszystkie opisane narzędzia pracują w trybie tekstowym i wymagają wprowadzenia dużej liczby przełączników z linii poleceń, dla niektórych użytkowników może być to wadą. Możliwe jest co prawda pewne zautomatyzowanie pracy przy pomocy skryptów, lecz i w tym przypadku praca dla osób przyzwyczajonych do trybu graficznego nie jest najprzyjemniejsza. Istnieje kilka programów oferujących graficzne GUI. Najbardziej znane to `avidemux` oraz Cinelerra. Cinelerra jest bardzo ciężka do skompilowania, a prekompilowane binaria nie zawsze chcą działać. Jest również trudna w obsłudze. Umożliwia eksport do wielu różnych formatów, m.in. MPEG, MJPEG, DIVX, a nawet QuickTime. Ma bardzo

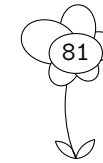
duże możliwości edycyjne. Wymaga bardzo silnego sprzętu. Cinelerra raczej nie jest przeznaczona dla zwykłych użytkowników, lecz dla osób, które chcą uczynić ze swojego komputera (komputerów) farmę do obróbki wideo. Avidemux umożliwia kompresję do formatu DIVX 4/5 (dostępne jest wsparcie dla trzech implementacji) oraz do formatu MPEG. Audio jest kompresowane do AC3, MP2 lub MP3. Umożliwia również kopiowanie strumieni bez ich rekompresji oraz cięcie fragmentów animacji. Wadami programu jest skromna ilość formatów wejściowych i mała stabilność programu. Jego możliwości nie są tak duże jak programów VirtualDub czy Nandub, ale rozwój posuwa się w prawidłowym kierunku.

Podsumowanie

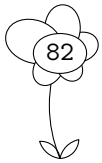
Kompresja wideo pod Linuksem nie jest może jeszcze popularna, ale ma szansę się taką stać w przyszłości. Zbiór kodeków i obsługiwanych formatów jest wystarczający do podstawowych zastosowań. Standard MPEG oraz popularny DIVX są dobrze obsługiwane. Możliwe jest zrzucanie sekwencji z kart telewizyjnych i ich kompresja w czasie rzeczywistym. Część programów jest zoptymalizowana pod kątem wsparcia instrukcji multimedialnych (MMX). Dzięki temu oraz możliwości optymalizacji pod kątem naszego procesora, przedstawione tu narzędzia nierzadko są szybsze niż ich odpowiedniki z systemu Windows. Pewną wadą są niewielkie możliwości pracy w trybie graficznym i dość skomplikowana obsługa.

Odnośniki

- Opis specyfikacji MPEG i encoder mpegencode: `http://bmrc.berkeley.edu/frame/research/mpeg/`
- Porównanie możliwości różnych kodeków MPEG4: `http://www.mplayerhq.hu/~michael/codec-features.htm`
- W czym OGM jest lepszy od AVI? `http://www.newlifeanime.com/nlaxvidogg.html`
- Fame: `http://fame.sourceforge.net/`
- Mjpegtools (z encoderem mpeg2enc): `http://mjpeg.sourceforge.net/`
- Ffmpeg (zawiera `libavcodec`): http://ffmpeg.sourceforge.net/



- Transcode: http://www.theorie.physik.uni-goettingen.de/~ostreich/transcode/
- Mencoder: http://www.mplayerhq.hu/
- Xvid: http://www.xvid.org/
- Divx5: http://www.divx.com/
- Avifile: http://avifile.sourceforge.net/
- x2divx: http://www.emulinks.de/divx/
- Ogmtools: http://www.bunkus.org/videotools/ogmtools/
- Cinelerra: http://www.heroinewarrior.com/cinelerra.php3
- Avidemux: http://fixounet.free.fr/avidemux/



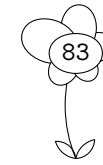
Testowanie aplikacji WWW programem flood

Jacek Prucia <jacek.prucia@acn.waw.pl>

Klasyczne serwisy internetowe to przykłady produktów, które nigdy nie osiągają ostatecznego kształtu. Bezustannie się do nich coś dodaje, poprawia czy też optymalizuje. Przy tego typu pracach bardzo pomocny jest tester aplikacji, który pozwoli ustalić, czy ostatnie poprawki coś popsuły lub o ile (jeżeli w ogóle) poprawiły wydajność. Flood jest właśnie programem tego typu, rozwijanym w ramach Apache Software Foundation.

Trochę historii

Oczywiście nie od razu Rzym zbudowano. Początki testów aplikacji webowych to proste, niewielkie programy pozwalające na wystosowanie jednego lub znacznie większej liczby zapytań, ale zwykle w oparciu o jeden URL. Czym zatem różniły się od konstrukcji w stylu:




```
echo -ne "GET / HTTP/1.1\nHost: www.x.com\n\n" | nc www.x.com 80
```

Przede wszystkim dosyć szczegółowym pomiarem czasu. Nie chodzi tutaj bynajmniej o precyzję, tylko o rozbicie mierzonego czasu na kolejne etapy wysyłania zapytania - utworzenie gniazda, przygotowanie samego zapytania, jego

wysłanie i odbiór danych z serwera. Oczywiście dla administratora najbardziej istotny był czas pomiędzy wysłaniem danych, a otrzymaniem odpowiedzi z serwera, jest to bowiem czas jaki (z grubsza) zajęło serwerowi przetworzenie zapytania. Czas ten, w połączeniu z całkowitą liczbą zapytań, pozwalał na wyznaczenie najbardziej podstawowej cechy charakteryzującej wydajność serwera, czyli współczynnika *requests per second* (zapytania na sekundę). Najpopularniejsi przedstawiciele tego typu programów to niewielki ApacheBench stanowiący część każdej dystrybucji serwera Apache oraz opracowany przez Hewlett-Packard bardzo ciekawy program `httperf`.

Wspomniane programy nieźle radziły sobie z testowaniem wydajności serwera, ale niespecjalnie nadawały się do testów aplikacji internetowych. Możliwość odwołania się do tylko jednego adresu URL była niewątpliwie bardzo ograniczającym czynnikiem. Oczywiście nic nie stało na przeszkodzie, aby napisać skrypt (bash/perl/python/cokolwiek), który czytałby adresy URL z dowolnego źródła (`stdin`, plik) i odpowiednio uruchamiał właściwe programy z odpowiednimi opcjami. Problem pojawiał się na etapie analizy danych, gdyż w rezultacie otrzymywało się kilka logicznie odseparowanych raportów. Także i w tym przypadku można było sobie poradzić skryptami, ale daleko było takim rozwiązaniom do wygody i sprawności.

Na radykalną zmianę sytuacji wpłynęła era, której zmierzch właśnie obserwujemy, a mianowicie fala dotcom-ów. Na temat firm identyfikowanych z tym właśnie schematem napisano wiele rozpraw, ale niestety w większości przypadków skupiono się na aspektach ekonomicznych, całkowicie pomijając kwestie informatyczne. Tymczasem istnienie takiego okresu miało swój pozytywny wpływ na informatykę. Nie chodzi tutaj oczywiście o zbyt wybujałe uposażenia kadry informatycznej, ale o przeróżne ciekawe projekty, które obecnie są uważane za *nieuzasadnione z ekonomicznego punktu widzenia*. Problemy w funkcjonowaniu firm *nowej gospodarki* paradoksalnie przyczyniły się do rozwoju zagadnienia testowania aplikacji internetowych. Aplikacje te stanowiły bowiem nierzadko swojego rodzaju trzon istnienia firmy i podstawę jej funkcjonowania. Zabezpieczenie płynności finansowej wiązało się zatem w bezpośredni sposób z zapewnieniem ciągłości funkcjonowania aplikacji internetowej. Na takie zapotrzebowanie jako pierwsze odpowiedziały różne firmy, oferując rozwiązania różnej maści. Oczywiście pojawiły się także propozycje ze strony środowiska Open Source, które jednak dopiero niedawno dorównały poziomem oprogramowaniu komercyjnemu.

Po co w ogóle robić testy? Możliwość identyfikacji wąskich gardeł w aplikacjach internetowych i ich późniejsza likwidacja, to nie tylko przejaw solidności programistycznej, ale także twarda waluta. Wydajna aplikacja będzie pochłaniała znacznie mniej zasobów sprzętowych, co oczywiście przekłada się na realne

oszczędności. Ponadto szybkość jako taka wpływa na komfort pracy z aplikacją, a to, dla nadmiernie rozpieszczonych użytkowników sieci, bardzo istotna cecha.

Trzy oblicza testowania

Szeroko rozumiane testy aplikacji internetowych znajdują głównie zastosowanie podczas przeprowadzania następujących operacji:

Analiza wydajności serwisu (ang. *application performance analysis*).
To najbardziej podstawowa forma testu, która sprowadza się do następujących po sobie kolejnych wywołań poszczególnych elementów serwisu. Wynik takiego testu to po prostu kilka liczb określających czas, jaki serwer poświęcił na wygenerowanie odpowiedzi. Wynik w odniesieniu do pewnych założeń i/lub oczekiwań pozwala na wskazanie elementów serwisu, które funkcjonują zbyt wolno. Fragmenty takie można potem starannie wyprofilować i określić lokalizację *wąskiego gardła*, które praktycznie zawsze leży u podstaw problemów z wydajnością.

Testy regresyjne (ang. *application regression tests*).
W przypadku aplikacji internetowych testy regresyjne mają dokładnie taki sam cel jak w przypadku zwykłych programów – zachowanie kompatybilności z poprzednimi wersjami. Wprowadzenie zmiany w krytycznym fragmencie serwisu może (zgodnie ze złośliwymi prawami Murphego) nieść ze sobą poważny błąd. Nie zawsze da się go wyłowić tuż po wprowadzeniu zmian w kodzie, dlatego też tworzy się lub wykorzystuje istniejące (jeżeli projektant aplikacji je przewidział) tzw. przypadki testowe (użytkownik się zalogował, użytkownik przeczytał artykuł o numerze 123 itd.). Przekładają się one oczywiście na kilka występujących w ścisłej kolejności stron, które trzeba precyzyjnie wywoływać, przesyłając w miarę potrzeby różne dane. Otrzymane od serwera dokumenty HTML można sprawdzać na okoliczność pożądanych lub niepożądanych ciągów znaków i w ten sposób określać, czy dane zapytanie się powiodło, czy też nie. Czas odpowiedzi serwera jest w tym teście praktycznie nieistotny.

Planowanie wydajności serwisu (ang. *web capacity planning*).
Zastosowanie szybkiego serwera w połączeniu z dobrze opracowanym kodem aplikacji to solidna podstawa, ale w żadnym wypadku nie gwarancja poprawnie działającego serwisu. Większość osób odpowiedzialnych za zarządzanie w firmach jest zainteresowana osiągnięciem konkretnego efektu. Stosując bardzo pokrętną przenośnię można powiedzieć, że nie interesuje ich jak szybko może jechać samochód, tylko, czy o godzinie X dojedzie na miejsce przeznaczenia.

Aby otrzymać odpowiedź na takie pytanie muszą oni wstępnie określić owo *miejsce przeznaczenia*, czyli zakładane obciążenie serwisu. Sam test, korzystając z przeróżnych kombinacji przypadków użycia, symuluje realnych użytkowników serwisu – czytane są artykuły, dodawane komentarze itd. Oczywiście liczba zapytań w zadanej jednostce czasu jest na tyle duża, aby ostatecznie uzyskać zakładane natężenie. Wynik rozpatruje się na różne sposoby. Pod uwagę brana jest wydajność serwera (oraz być może bazy danych, która może znajdować się tuż za serwerem WWW), szybkość i poprawność pracy aplikacji. Jak widać ten test jest najbardziej kompleksowy, a interpretacja jego wyników najbardziej *płynna*.

Oczywiście, poza tymi dosyć oczywistymi zastosowaniami, takie programy jak flood można wykorzystać do różnych drobnych zadań. Można logować się do jakiegoś serwisu i automatycznie wykonywać jakieś zupełnie mechaniczne zadanie np.: rejestracja komputera w sieci (tego wymaga Neostrada, Astercity itd.), wysłanie SMS-a poprzez stronę WWW operatora itd. W zasadzie da się zrobić wszystko do czego potrzebna jest przeglądarka.

Instalacja

Wszystko co potrzebne jest do instalacji programu flood znaleźć można na następującej stronie:

```
http://httpd.apache.org/test/flood/
```

Jak sugeruje adres URL, program flood jest częścią nieco większego podprojektu `httpd-test` prowadzonego w ramach projektu `httpd` server, którego najbardziej znanym efektem jest serwer Apache. Prócz programu flood w repozytorium znajduje się pokaźnych rozmiarów test regresyjny serwera Apache (napisany jest w języku Perl) oraz moduł `mod_specweb`, który umożliwia wykorzystanie serwera Apache do testu SPECWeb99 (`http://www.spec.org/`).

Program można pobrać korzystając z gotowego archiwum `.tar.gz` lub bezpośrednio z repozytorium CVS. Ze względów czysto formalnych proces przygotowania nowej wersji programu pochłania sporo czasu. Zasady publikowania nowych wersji programu `flood` są takie same jak serwera Apache (wybór Release Manager w drodze głosowania, procedura testowania itd.). Biorąc pod uwagę fakt, że `flood` jest zdecydowanie mniej popularnym programem, to przez całą procedurę przechodzi się dopiero, gdy rozmiar zmian naprawdę uzasadnia sens całej tej zabawy. Z tego względu, pobierając źródła z repozytorium, na pewno będziemy posiadać znacznie nowszą wersję programu niż ta która zawarta jest

w oficjalnym wydaniu. Nad tym co trafia do repozytorium czuwa kilka wyjątkowo solidnych osób, więc szansa na poprawną kompilację jest bardzo wysoka. Pobranie źródeł z repozytorium sprowadza się do wydania kilku poleceń:

```
cvs -d :pserver:anoncvs@cvs.apache.org:/home/cvspublic login
cvs -d :pserver:anoncvs@cvs.apache.org:/home/cvspublic co httpd-test/flood
cd flood
cvs -d :pserver:anoncvs@cvs.apache.org:/home/cvspublic co apr
cvs -d :pserver:anoncvs@cvs.apache.org:/home/cvspublic co apr-util
```

Jak wynika z poleceń zamieszczonych powyżej – program flood korzysta z bibliotek `apr` oraz `apr-util`. Nie musimy się jednak specjalnie przejmować wersją obydwu bibliotek, gdyż `flood` nie korzysta z tych części, które ulegają dynamicznym zmianom. W tej kwestii prym wiedzie serwer Apache, który w zasadzie steruje rozwojem biblioteki. Kompilacja i instalacja, dzięki tzw. *GNU build system* (autoconf, automake, libtool) jest banalna i sprowadza się do wydania 3 nieśmiertelnych poleceń:

```
./configure
make
make install
```

Po ich wykonaniu program wyląduje w katalogu `/usr/local/flood/`, chyba że zażyczymy sobie inną lokalizację korzystając z opcji `--prefix` skryptu `configure`. Zanim jednak usuniemy po instalacji katalog ze źródłami programu, proponuję najpierw skopiować w uprzednio wybrane miejsce zawartość podkatalogu `examples`. Zawiera on kilka przykładowych konfiguracji oraz skrypty awkowe do analizy wyników. Tych kilka drobiazgów przyda się w późniejszej pracy.

Architektura programu

Architektura programu `flood` jest dosyć złożona. Dzieje się tak dlatego, że za pomocą jednej sesji flooda, bez pomocy żadnych zewnętrznych skryptów, można wystosować od jednego do kilkuset (lub nawet więcej) zapytań. Aby nieco uprościć tę architekturę zastosowano oryginalny schemat nazw, o podłożu rolniczym, chociaż cały schemat zaczyna się niewinnie. Podstawowym elementem leżącym najniżej w hierarchii jest adres URL. Adresy URL grupowane są w większe grupy, które mogą stanowić razem wspomniane już przypadki testowe. Właściwe zapytania wykonuje... farmer. W zależności od różnych warunków owym farmerem jest proces lub wątek. Dla użytkownika ten fakt

nie powinien być jednak istotny i nic nie stoi na przeszkodzie, aby wyobraził on sobie człowieka mozolnie wklepującego adresy do przeglądarki. Na pracę farmera ma wpływ profil, który określa w jaki sposób wystosowywane są zapytania (round-robin, losowe adresy itd.), jak weryfikowane odpowiedzi i zliczane wyniki. Farmerzy mogą być grupowani w (a jakże...) farmy, w ramach których można określić ile razy farmerzy mają wykonać swoją pracę lub jak długo mają ją wykonywać. W architekturze zawarto jeszcze miejsce dla dwóch pozycji – collective i megaconglomerate. Pierwszy element pozwala na grupowanie farm w ramach jednej maszyny, a megaconglomerate grupuje kolektywy porozrzucane po różnych maszynach. Niestety obydwa wymienione elementy nie zostały jeszcze zaimplementowane i na razie zajmują tylko kilka linijek w dokumencie DESIGN.

Od strony programistycznej i użytkowej najciekawszy jest profil. Pozostałe elementy służą albo do przechowywania danych, albo do układania farmerów w logiczną strukturę. Natomiast profil można na różne sposoby modyfikować i uzyskiwać w ten sposób pożądane zachowanie programu. Na profil składa się mnóstwo przeróżnych funkcji, które reprezentują kolejne etapy pracy farmera. Domyślnie każdemu z tych etapów przyporządkowana jest pusta funkcja *generic*, która może wykonywać czynność w najbardziej podstawowym zakresie, lub całkowicie rezygnować z wpływu na dany etap. Jako że flood to dosyć młody program, to sporo etapów jest obsługiwanych tylko przez funkcję generic. Etapy które posiadają alternatywne funkcje, to te, które wymienione są w sekcji profile pliku konfiguracyjnego (m.in. `verify_resp` itd.). Aby przekonać się jakie to pozycje zostały pominięte i jakich etapów pracy programu się tyczą, trzeba zajrzeć do pliku źródłowego `flood_profile.c` i obejrzeć dokładnie tablicę `profile_event_handlers`.

Let's do some testing!

Przygotowanie testu sprowadza się do przygotowania odpowiedniego pliku XML. W chwili obecnej proces ten wymaga uruchomienia ulubionego edytora, aczkolwiek dąży się do tego, aby w przyszłości zautomatyzować go za pomocą GUI. Nie ważne jak bardzo lubimy konsolę – proces konwersji niektórych znaków (np.: `&` na `&` czy też `ą` na `#x0105`) jest na tyle uciążliwy, że o ile edytor nie ułatwi nam życia, to przygotowanie konfiguracji pochłonie trochę czasu i nerwów (że o kawie i papierosach nie wspomnę). A biorąc pod uwagę fakt, że flood korzysta z biblioteki `apr` i `apr-util`, która wprowadza własny parser typu DOM (chodzi oczywiście o sam koncept, a nie o zgodność z API) bazujący na znakomitym i szeroko znanym parserze expat – będzie to konieczne.

Niestety expat obsługuje tylko UTF-8 oraz ISO-8859-1, a w najbliższej przyszłości nie planuje się przesiadki na jakiś inny parser, który by ów problem załatwił (np.: Xerces).

Elementem głównym dla całego pliku konfiguracyjnego jest element `<flood>`. Pierwszy blok jaki w jego ramach zamieścimy, to lista adresów URL do przetestowania. W tym konkretnym przypadku będzie to proste logowanie do serwisu:

```
<urllist>
   <name>logowanie</test>
   <description>logowanie do serwisu</description>
   <url>www.site.com/welcome.html</url>
   <url method="post" payload="username=foo&amp;password=bar">www.site.com/login.php</url>
</urllist>
```

Pierwszy adres URL wywoła klasyczny GET, podczas gdy drugi skorzysta z metody POST i prócz samego zapytania wyśle także obiekt zawierający dokładnie to, co zawiera atrybut `payload`. Fakt, że użytkownik się zalogował można jeszcze sprawdzić, ale o tym trochę dalej. Nie ma potrzeby zbytnio komplikować pierwszego testu, dostatecznie oszpeca go atrybut `payload` drugiego adresu. Nie da się ukryć, że listy adresów z dużą liczbą zapytań typu POST szybko tracą na czytelności.

Zanim podepniemy listę adresów URL pod farmera, musimy dla niego przygotować odpowiedni profil, chociażby taki:

```
<profile>
       <name>Profil typowy</name>
       <description>W kolo Macieju</description>
       <useurllist>logowanie</useurllist>
       <profiletype>round_robin</profiletype>
       <socket>generic</socket>
       <report>relative_times</report>
       <verify_resp>verify_200</verify_resp>
</profile>
```


Jak widać profil powiązany jest z konkretną grupą adresów URL. Adresy URL będą pobierane z listy w kolejności właściwej dla danego profilu. Wybrany powyżej typ profilu, czyli `round robin`, to klasyczne *w koło Macieju*, czyli sekwencyjne powtarzanie adresów w kolejności, w jakiej występują. Element `socket` pozwala na określenie typu gniazda. Do wyboru jest standardowe gniazdo (generic) oraz gniazdo dla połączeń keepalive (`keepalive`). Kolejny element, raport, określa sposób przygotowywania ostatecznych wyników. Dostępne wartości (`easy`, `simple`, `relative_times`) różnią się praktycznie tylko sposobem

reprezentacji czasu (absolutny, mierzony od startu zapytania). Ostatni element, czyli `verify_resp` określa sposób weryfikacji odpowiedzi. Na razie dostępne są tylko dwie wartości: `verify_200` oraz `verify_status_code`, które uznają kody `2xx` oraz `3xx` za `OK`, a wszystkie inne za `FAIL`. Oczywiście taki sposób oceny daleki jest od doskonałości, ale wprowadzenie nieco bardziej sprawiedliwego schematu wymaga sporo pracy i konieczności wprowadzenia innych elementów do programu.

A teraz przydałby się ktoś, kto by całą tą robotę grzecznie i bez strajków wykonał. Trzeba przygotować sekcję dla farmera.

```
<farmer>
       <name>Andrzej L.</name>
       <time>300</time>
       <useprofile>Profil typowy</useprofile>
</farmer>
```

Element `time` wprowadza odgórne ograniczenie czasu pracy, z którego pracownik skwapliwie korzysta. Jeżeli wykracza poza czas potrzebny do jednorazowego wykonania pracy, to farmer wykonuje ją po raz kolejny przerywając w momencie upływu czasu. Zamiast tego można skorzystać z elementu `<count>`, który określi ile razy farmer ma wykonać jednostkową pracę bez względu na to jak dużo czasu mu to zajmie.

Farmera należy umieścić na farmie, nawet jeżeli będzie tam wiódł samotne życie. W miarę potrzeb można farmera wygodnie sklonować korzystając do tego celu z atrybutu `count`.

```
<farm>
       <name>Bingo</name>
       <usefarmer count="2">Andrzej L.</usefarmer>
</farm>
```

Jednym z najbardziej denerwujących błędów w programie jest wymóg, aby farma nazywała się „Bingo”. Całe szczęście błąd jest na tyle trywialny, że niedługo zostanie poprawiony.

Na tym można zakończyć edycję pliku. Teraz wystarczy całość zapisać do pliku np.: `test.xml` i uruchomić program flood w następujący sposób:

```
% flood /path/to/test.xml
```

Po chwili pracy programu powinniśmy otrzymać mniej więcej takie wyniki:

```
1044230201595559 214868 214918 422937 422981 OK  16386 http://www.site.com/welcome.html
1044230201596922 218240 218252 430603 430633 OK  32771 http://www.site.com/welcome.html
```

```
1044230203015523 209276 209308 662837 662872 OK  16386 http://www.site.com/login.php
1044230203025536 211308 211323 663398 663431 OK  32771 http://www.site.com/login.php
1044230204675545 212363 212394 421951 421986 OK  16386 http://www.site.com/welcome.html
1044230204685527 218894 218912 427765 427796 OK  32771 http://www.site.com/welcome.html
1044230206105500 233273 233288 672214 672249 OK  32771 http://www.site.com/login.php
1044230206095537 238362 238397 682597 682620 OK  16386 http://www.site.com/login.php
```

Jak je interpretować? To zależy od rodzaju raportu jaki wybraliśmy, ale generalnie to jest to raczej robota dla skryptu. Z grubsza jednak kolejne kolumny mają następujące znaczenie:

1. start zapytania, wynik funkcji `time()` (absolutny)
2. czas potrzebny do nawiązania połączenia (relatywny)
3. czas potrzebny do wysłania zapytania (relatywny)
4. czas jaki upłynął do otrzymania odpowiedzi (relatywny)
5. czas potrzebny na zamknięcie połączenia (relatywny)
6. rezultat zapytania (2 wartości: OK lub FAIL)
7. identyfikator procesu/wątku (grupowanie wyników dla farmera)
8. adres URL wykorzystany w zapytaniu (grupowanie wyników dla strony)

Obecnie wykorzystywany jest czysto tekstowy format, aczkolwiek w przyszłości rozważa się migrację do XML-a, bowiem wynik działania programu flood w założeniach miał być przeznaczony dla skryptów/programów analizujących. Narazie dostępne są tylko dwa skrypty: `analize-relative` oraz `analize-relative-ramp`. Obydwa zostały napisane w awk, gdyż na samym początku prac nad programem takie rozwiązanie wydawało się najprostsze i najszybsze do uzyskania. Wynik działania pierwszego z nich znajduje się poniżej:

```
Slowest pages on average (worst 5):
   Average times (sec)
connect        write   read    close   hits    URL
0.2221 0.2221  0.4447  0.4448  4       http://www.site.com/welcome
0.2105 0.2105  0.6703  0.6703  4       http://www.site.com/login.php
Requests: 8 Time: 2.23 Req/Sec: 3.59
```

Jak widać, w nieco bardziej przejrzystej formie podaje on do wiadomości wyniki razem z upragnioną liczbą zapytań na sekundę (`Req/Sec`). Natomiast skrypt `analize-relative-ramp` służy do analizy zapytań w których wykorzystano opóźnienia.

Na tym można zakończyć pierwszy test. Pora przypatrzeć się temu, co flood naprawdę potrafi.

Drobne czary-mary

Podczas formowania testowej listy adresów URL wspomniałem, że warto byłoby sprawdzić, czy użytkownik naprawdę się zalogował. Jak to zrobić? Wystarczy przypatrzeć się wynikowemu HTML-owi w poszukiwaniu charakterystycznej frazy. Powiedzmy, że będzie to coś takiego: `"witaj user!"`. Teraz tylko wystarczy na tę okoliczność napisać wyrażenie regularne, powiedzmy takie: `"witaj (.*)!"`. Potem wystarczy już tylko przygotować odpowiedni element url.

```
<url method="post" payload="username=foo&amp;password=bar"
    responsename="user"
    responsetemplate="witaj (.*)!">www.site.com/login.php</url>
```

Jeżeli wyrażenie regularne nic nie znajdzie w wynikowym HTML-u, to program zgłosi błąd i zakończy pracę. Jeżeli jednak coś uda się dopasować, to wynik trafi do zmiennej `$user`, którą można później wykorzystywać w kolejnych zapytaniach, chociażby tak:

```
<url requestparamcount="1"
     requesttemplate="http://www.site.com/showuser.php?user=${user}" />
```

Opracowywanie kilku wyrażeń regularnych na sporej wielkości stronę jest oczywiście dosyć uciążliwe. Dlatego też rozważane są inne możliwości pobierania zmiennych z wynikowej strony. Przede wszystkim jest to konkretny nagłówek HTTP (np.: `X-Flood`) interpretowany wedle ściśle ustalonych zasad, np.: `zmienna1=wartosc1;zmienna2=wartosc2` itd. Rozwiązanie takie nie nadaje się jednak specjalnie do gotowych już aplikacji. Dlatego też zmienna będzie mogła być poszukiwana wedle bardzo luźnych zasad przy wykorzystaniu biblioteki `el-kabong` HTML (`http://ekhtml.sf.net`) lub nieco precyzyjniej przy wykorzystaniu parsera typu DOM.

Kolejny, bardzo przydatny drobiazg, to sekwencje. Pozwalają one na wykonanie tego samego zapytania kilka razy pod rząd, ale za każdym razem z nieco zmodyfikowanym adresem URL. Prosty przykład powinien wyjaśnić nieco zawiłą definicję:


```
<urllist>
  <sequence sequencename="myseq" sequencelist="bob, jane, joe">
      <url requesttemplate="http://www.site.com/~${myseq}/" />
  </sequence>
</urllist>
```

Taka lista przełoży się ostatecznie na następujące adresy URL:

```
http://www.site.com/~bob/
http://www.site.com/~jane/
http://www.site.com/~joe/
```

Niestety sekwencje (przynajmniej narazie) są statyczne. Należy je przygotować ręcznie, aczkolwiek są pomysły, aby sekwencja była budowana z wartości kolejnych zmiennych.

Wszystkie serwisy wspierane przez bazę danych bardzo często korzystają z identyfikatorów obiektów w bazie danych. Do częstych należą zapytania w postaci:

```
http://www.site.com/showprod.php?id=345
```

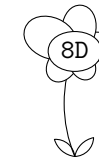
Jeżeli nie chcemy wydobywać konkretnych identyfikatorów z listy zawartej na stronie HTML, to można skorzystać z generatora liczb losowych. Wystarczy zastosować następujący element URL:

```
<url requestparamcount="1"
     requesttemplate="http://www.site.com/showprod.php?id=${=foo}" />
```

Dzięki takiej konstrukcji program wygeneruje liczbę losową i przypisze jej wartość na zmienną foo. Ze zmiennej tej można korzystać potem w kolejnych zapytaniach, np:

```
http://www.site.com/showprod.php?id=${=foo}
http://www.site.com/buyprod.php?id=${foo}
http://www.site.com/removeprod.php?id=${foo}
```

Niestety, poważnym ograniczeniem jest brak możliwości określania przedziału liczb z jakiego ma być wybierana liczba losowa. Domyślny zakres dla funkcji rand() określany przez stałą `RAND_MAX` (`stdlib.h`) to (na moim systemie) 2147483647, czyli dosyć sporo. Nie każda baza może się poszczycić tabelą z tyloma rekordami.



Jeżeli program flood wykorzystywany jest do regresyjnych testów aplikacji, to dobrze jest odseparować strukturę poszczególnych elementów serwisu od adresu, na którym aktualnie aplikacja jest osadzona. Można tego dokonać korzystając z elementu `baseurl`.

```
<urllist>
   <baseurl>http://www.site.com</baseurl>
   <url>/welcome.html<url>
   <url>/login.php</url>
</urllist>
```

Tak przygotowana lista może być przystosowana do nowej lokalizacji tylko przez dokonanie kosmetycznej zmiany.

Na tym oczywiście nie kończą się fajerwerki jakie oferuje flood, ale z różnych względów większa część z nich obecnie jest w stanie implementacji. Trochę czasu potrwa, zanim ujrzą one światło dzienne, gdyż program flood czeka kilka głębokich strukturalnych zmian. Chociaż dla rozwoju programu będą one zbawienne, to jednak od strony funkcjonalności programu wydawać się będą po prostu zastojem w pracach.

Widoki na przyszłość

Priorytet numer jeden to przepisanie programu flood z wykorzystaniem biblioteki `apr-serf`. Podstawową korzyścią tej zmiany ma być możliwość realizacji niektórych aspektów programu flood jako filtrów. Dlaczego jest to aż takie istotne? Weźmy na przykład weryfikację odpowiedzi. W chwili obecnej sprawdzany jest tylko kod HTTP. Z drugiej strony dobrze byłoby sprawdzać tekst na okoliczność pożądanych ciągów znaków (operacja zakończona powodzeniem, produkt dodany, itd.) lub też takich, które nie powinny w nim wystąpić (błędy PHP, JSP exception stack trace itp.). Jeżeli wszystkie testy realizowane byłyby jako osobne funkcje, to należy pamiętać, że element `verify_resp` akceptuje tylko nazwę jednej funkcji. Łączenie wszystkich testów w jedną duża funkcję nie wydaje się być dobrym rozwiązaniem. Stąd też pomysł zaimportowania znanego z serwera Apache mechanizmu filtrów.

Druga sprawa to dobra dokumentacja. Ciężko pracować z programem, którego dokumentacja to komentarze w przykładowych plikach konfiguracyjnych i logi doczepiane do poszczególnych commitów. Problem stanowi jednak standard dokumentacji w ramach projektu HTTP Server Project. Pewne podwaliny pod sensowną dokumentację zostały już stworzone, a prace trwają.

Na samym końcu warto wspomnieć o opracowaniu przejrzystego GUI. Format XML jest wyjątkowo nieprzyjemny w tym konkretnym zastosowaniu, toteż opracowanie nieco bardziej przyjaznej formy opracowywania testów także widnieje na liście TODO. Samo zagadnienie jest zresztą bardzo szerokie, bo wymaga także opracowania sposobu gromadzenia adresów URL. Wszystko jest w porządku, gdy mamy do przygotowania listę zawierającą 5, może 10 adresów, ale

lista która ma 50, 100 czy też 200 adresów to już zadanie dosyć uciążliwe nawet dla bardzo wytrwałego *pianisty*. Rozwiązaniem tego problemu może być analiza wyniku działania programu do analizy pakietów, proste proxy lub nawet wtyczka (plugin) do przeglądarki.

Oczywiście po drodze jest cała masa różnych drobnych usprawnień i poprawek błędów, których flood ma tyle co dobra gleba pędraków. Mimo tych niedociągnięć flood to program który do zagadnienia testów podchodzi bardzo kompleksowo i zapewne w momencie osiągnięcia pełnej stabilności będzie bardzo silną konkurencją dla oprogramowania komercyjnego.

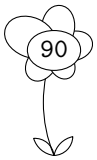
IEEE 1394, czyli praktyczne wykorzystanie urządzeń FireWire w Linuksie

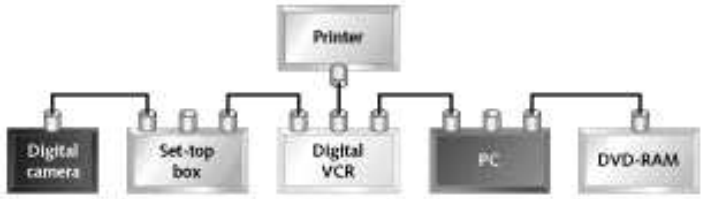
Krzysztof Krawczyk <krzysiek@linux.com.pl>
LiNUX+

STRESZCZENIE: Część współczesnych komputerów jest już wyposażona we wbudowany interfejs FireWire, ale do reszty należy dokupić oddzielną kartę PCI lub PCMCIA. Najczęstszym zastosowaniem tego rozwiązania jest podłączanie do komputera kamery cyfrowej, ale można je także wykorzystać do podłączaniu zewnętrznych nośników pamięci masowych lub łączenia komputerów w niewielką sieć lokalną. Jądro Linuksa posiada w tej chwili zadowalające wsparcie dla tego standardu, a dostępne oprogramowanie pozwala na znaczące wykorzystanie jego możliwości. W czasie prelekcji zostanie omówiony pokrótce sam standard, a także jego wykorzystanie w Linuksie na przykładzie podłączania zewnętrznego dysku twardego oraz kamery cyfrowej. Ponadto zostanie przybliżone oprogramowanie służące do pobierania i obróbki danych wideo z kamery cyfrowej, czyli DVgrab, Kino oraz Cinelerra.

WSTĘP

Na naszych komputerach często gromadzimy bardzo duże ilości danych, wśród których palmę pierwszeństwa dzierżą pliki multimedialne. Wymuszają one na nas stosowanie coraz to szybszych metod ich przesyłania pomiędzy komputerami lub innymi urządzeniami elektronicznymi. Wśród istniejących rozwiązań, które służą właśnie do przesyłania dużych ilości danych (przede wszystkim multimedialnych), warto wymienić szybką magistralę szeregową o nazwie FireWire. W 1986 roku została ona zaprezentowana przez firmę Apple Computer jako alternatywa dla interfejsu SCSI. W 1995 roku, w wyniku przeprowadzonych prac certyfikacyjnych, stała się ona oficjalnym przemysłowym standardem, zaakceptowanym przez organizację IEEE-SA (Institute of Electrical and Electronics Engineers Standards Association – http://standards.ieee.org/) i oznaczonym





Rys 1

jako IEEE 1394-1995. W marcu 2000 roku, po kilku latach ustalania szczegółów, specyfikacja standardu została uaktualniona i nazwana `IEEE 1394a-2000`. Zanim to nastąpiło, rozpoczęte zostały także prace nad nową wersją magistrali, która oferowałaby znacznie większe możliwości. Ostatecznie w kwietniu 2002 roku przyjęta została specyfikacja o nazwie `IEEE 1394b`. Godne uwagi jest to, że firma Sony wprowadziła własną nazwę na określenie tej magistrali – `i.Link`.

Aktualnie popularyzacją standardu `IEEE 1394` i urządzeń elektronicznych, które go wykorzystują, zajmuje się organizacja 1394 Trade Association (`http://www.1394ta.org/`).

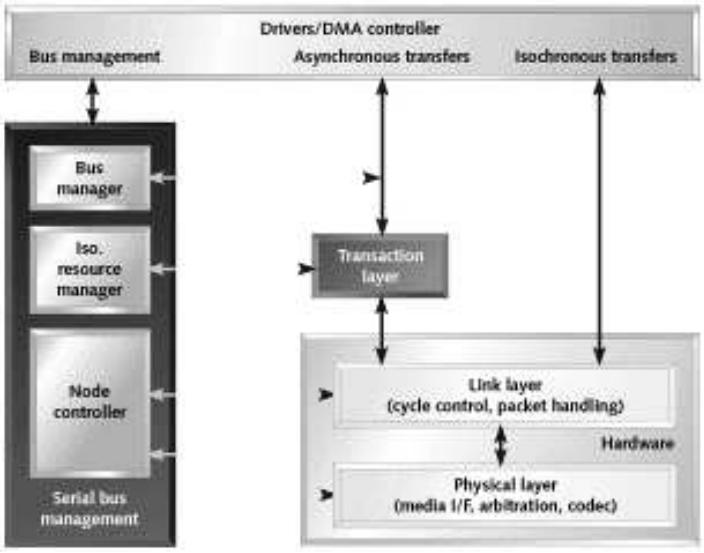
Cechy magistrali IEEE 1394

Magistrala IEEE 1394 wykorzystuje technologię sieci P2P (ang. *peer-to-peer*), a to oznacza, że urządzenia (węzły), które zostały do niej dołączone, mogą ze sobą współpracować bez pośrednictwa wyróżnionego elementu, np. komputera klasy PC jako kontrolera sterującego pracą magistrali. Każde z urządzeń może posiadać kilka portów (w praktyce nie więcej niż trzy), które działają jak przekaźniki, tzn. przesyłają dalej pakiety otrzymane przez inne porty danego węzła. Ilustruje to Rysunek 1, na którym kamera cyfrowa może być źródłem danych dla drukarki lub nagrywarki DVD.

Konfiguracja magistrali uaktualniana jest automatycznie przy podłączaniu lub odłączaniu dowolnego urządzenia. Proces ten rozpoczyna się hierarchicznie od urządzeń, które podłączone są tylko do jednego urządzenia, a kończy na tych, które „rozdzielają" inne urządzenia (zazwyczaj jedno z nich staje się głównym urządzeniem na magistrali).

W przypadku pracy z komputerem możliwe jest wykorzystywanie tzw. mechanizmu hot-plug, czyli bezpiecznego podłączania lub odłączania urządzeń podczas normalnej pracy komputera.

Do jednej magistrali mogą być podłączone 64 urządzenia (w topologii drzewa lub gwiazdy), a ponadto możliwe jest zmostkowanie do 1024 magistral. Jedynym



Rys 2

warunkiem, który musi zostać spełniony podczas łączenia ze sobą urządzeń, jest to, aby nie tworzyły one pętli zamkniętej.

W ramach magistrali IEEE 1394 dostępne są dwa tryby przesyłania danych: izochroniczny oraz asynchroniczny. Pierwszy z nich nie zapewnia sprawdzania błędów ani retransmisji danych, ale pozwala na wykorzystanie do 80% dostępnego pasma magistrali – jest to najlepsze rozwiązanie, gdy kluczowe jest zagwarantowanie dostępności danych, np. strumienia wideo lub audio, bez żadnych opóźnień (nadawanie w równych odstępach czasu), ale z tolerancją na pewne błędy. Drugi z nich charakteryzuje się pełną kontrolą nad przesyłanymi danymi (oczekiwanie na potwierdzenie poprawnego odebrania danych) – jest to najlepsze rozwiązanie, gdy niezbędne jest zachowanie pełnej integralności przesyłanych danych, np. przy ich zapisie na dysk twardy.

Prędkości przesyłania danych, oferowane przez magistralę IEEE 1394, zdefiniowane są w postaci trzech trybów: S100 (98304 Mb/s), S200 (196608 Mb/s) oraz S400 (393216 Mb/s). Otrzymane wartości zaokrąglane są odpowiednio do 100 Mb/s, 200 Mb/s oraz 400 Mb/s.

Specyfikacja magistrali IEEE 1394 definiuje cztery warstwy wykorzystywanego protokołu: fizyczną, łącza, transakcji oraz zarządzania magistralą szeregową. Ilustruje to Rysunek 2, na którym przedstawione są zależności pomiędzy nimi.

Warstwa fizyczna definiuje okablowanie, złącza, sygnały elektryczne, mechanizm arbitrażu oraz szeregowe kodowanie i dekodowanie wysyłanych lub odbieranych danych. Jako okablowanie wykorzystywany jest ekranowany kabel sześciożyłowy, w którego skład wchodzą dwie pary oddzielnie ekranowanych kanałów danych oraz dwa przewody zasilające (napięcie zasilające 8–40 V i pobór prądu do 1,5A). Istnieje także czterożyłowa wersja tego kabla, która jest pozbawiona przewodów zasilających (jest ona stosowana w urządzeniach, które z założenia posiadają własne źródło zasilania, np. kamery cyfrowe). Ograniczenie długości kabla w pierwotnej wersji specyfikacji wynosiło 4,5 metra, ale ostatecznie zostało ono zwiększone do 100 metrów. Stosowane złącza mogą być cztero- lub sześciopinowe (w przypadku podłączania kamery cyfrowej do komputera, z jednej strony stosuje się pierwszy rodzaj złącza, a z drugiej – drugi).

Warstwa łącza jest interfejsem pomiędzy warstwą fizyczną a warstwą transakcji. Jest ona odpowiedzialna za sprawdzanie sum CRC w otrzymanych pakietach oraz wyliczanie i dołączanie sum CRC do wysyłanych pakietów. Ponadto, ponieważ do przesyłania danych w sposób izochroniczny nie jest wykorzystywana warstwa transakcji, jej rolę przejmuje warstwa łącza. Analiza nagłówków pakietów oraz detekcja rodzaju transakcji jest również zadaniem tej warstwy – uzyskane informacje przesyłane są do warstwy transakcji.

Warstwa transakcji jest wykorzystywana do przesyłania danych w sposób asynchroniczny. Dostępnych jest kilka typów transakcji: proste odczytania słowa (czterech bajtów), proste zapisanie słowa, odczytania danych o zmiennej długości, zapisania danych o zmiennej długości oraz blokowanie.

Z zarządzaniem magistralą związane jest przydzielenie węzłom roli wzorca cykli (zostaje nim automatycznie główny węzeł magistrali, który co 125μs rozpoczyna nowy cykl), zarządcy zasobów izochronicznych oraz zarządcy magistrali (do jego zadań należy publikacja map topologii i szybkości, sterowanie zasilaniem oraz optymalizowanie ruchu na magistrali).

Taka definicja standardu magistrali IEEE 1394 pozwalała na dowolną jej implementację, co utrudniało wdrażanie tej technologii. W związku z tym organizacja 1394 Trade Association przygotowała specyfikację OHCI (Open Host Controller Interface), aby nowe układy cyfrowe oraz wykorzystujące je urządzenia mogły być w prosty sposób wspierane przez wszystkie systemy operacyjne. Od tej pory producenci sprzętu zapewniają kompatybilność ich urządzeń z tą specyfikacją.

IEEE 1394 a USB

Podczas omawiania magistrali IEEE 1394 należy wspomnieć także o magistrali USB (Universal Serial Bus), która jest podobnym rozwiązaniem, ale przezna-

czonym do innych zastosowań. O ile magistrala IEEE 1394 stworzona został do zapewnienia komunikacji pomiędzy urządzeniami audiowizualnymi, o tyle magistrala USB powstała jako standard dla komputerowych urządzeń peryferyjnych.

Najważniejszym ograniczeniem pierwotnej wersji specyfikacji USB (była nią 1.1) była niewystarczająca prędkość przesyłania danych (do 12 Mb/s). Eliminowało ją to automatycznie z zastosowań typowych dla IEEE 1394, czyli transmisji strumienia danych wideo i audio w czasie rzeczywistym. W momencie przygotowania specyfikacji USB 2.0 maksymalna przepustowość tej magistrali została zwiększona do 480 Mb/s. Chociaż wersja IEEE 1394a-2000 zapewniała mniejsze możliwości (do 400 Mb/s), to jednak wersja IEEE 1394b ponownie podniosła poprzeczkę i wprowadziła nowe tryby: S800, S1600 oraz przyszłościowy S3200 (w praktyce zatem prędkości do1600 Mb/s).

Oprócz tak podstawowej różnicy w obu magistralach, istnieją jeszcze inne, takie jak: okablowanie i złącza (USB 1.1/2.0: dwa przewody sygnałowe oraz dwa przewody zasilające; IEEE 1394: dwie pary przewodów sygnałowych oraz opcjonalnie dwa przewody zasilające), maksymalna długość przewodu łączącego dwa węzły (USB 1.1/2.0: 5m; IEEE 1394a-2000: 4.5m; IEEE 1394b: 100m), maksymalna odległość pomiędzy najbardziej oddalonymi węzłami (USB 1.1/2.0: 35m; IEEE 1394: 72m) czy zarządzanie magistralą (USB 1.1/2.0: niezbędny działający komputer i nadrzędny kontroler; IEEE 1394: praca bez centralnego nadzoru oraz brak konieczności obecności komputera).

Ich cechy wspólne wynikają z takich samych założeń projektowych – magistrala szeregowa z automatyczną konfiguracją przy podłączaniu i odłączaniu urządzeń, automatycznym przydziałem adresów, automatycznym terminowaniem fizycznych zakończeń oraz przesyłaniem danych w postaci pakietów.

LINUX1394

Wsparcie dla magistrali IEEE 1394 w Linuksie, na które składa się wydzielony podsystem w jądrze oraz biblioteka programistyczna libraw1394, zapewniająca bezpośredni dostęp do magistrali, pojawiło w jądrze od wersji 2.3.40, a aktualnie można z niego korzystać w jądrach serii 2.2, 2.4 oraz 2.5. Głównym źródłem informacji na temat wykorzystania tej technologii w Linuksie jest serwis IEEE 1394 for Linux (`http://www.linux1394.org/`) i to tam znajdują się najnowsze sterowniki dla magistrali IEEE 1394.

Wśród obsługiwanych chipsetów można wyróżnić przede wszystkim PCILynx/PCILynx2 firmy Texas Instruments, a także wszystkie zgodne ze specyfikacją OHCI (od różnych producentów). Nie są wspierane własne układy firmy

Sony, wykorzystywane w niektórych komputerach VAIO, a także układ Adaptec AIC-5800. Prowadzone są prace, aby wspierać karty, które bazują na innych układach firmy Adaptec, ale udało się osiągnąć pewien sukces tylko w kilku przypadkach.

Aby rozpocząć pracę z magistralą IEEE 1394 pod Linuksem, należy posiadać przynajmniej jedno urządzenie tego typu, które w przypadku komputerów z procesorami rodziny x86 powinno być zrealizowane w postaci karty PCI lub PCMCIA. Istnieje także szansa, że odpowiedni port jest już wbudowany na płycie głównej. Do testów warto także posiadać jakieś urządzenia peryferyjne, które wykorzystuje ten standard, np. zewnętrzny dysk twardy lub kamerę cyfrową.

W dalszej części tego opracowania, dla zwrócenia uwagi, będzie wykorzystywana karta PCMCIA z zestawu AFW-1430 FireConnect for Notebooks firmy Adaptec, kamera cyfrowa MV 400 firmy Canon oraz dysk twardy EIDE/ATAPI firmy Western Digital w zewnętrznej obudowie firmy DataFab z portem IEEE 1394.

Wszystkie wykorzystane przykłady będą specyficzne dla dystrybucji Linuksa Aurox 8.0, która została zainstalowana na komputerze przenośnym IBM Thinkpad T21.

Instalacja

Dystrybucja Linuksa Aurox 8.0 poprzez swoje standardowe jądro w wersji 2.4.18 udostępnia od razu wszystkie moduły jądra związane z magistralą IEEE 1394, a zatem rekompilacja jądra w tym przypadku nie jest konieczna – można ten proces pominąć.

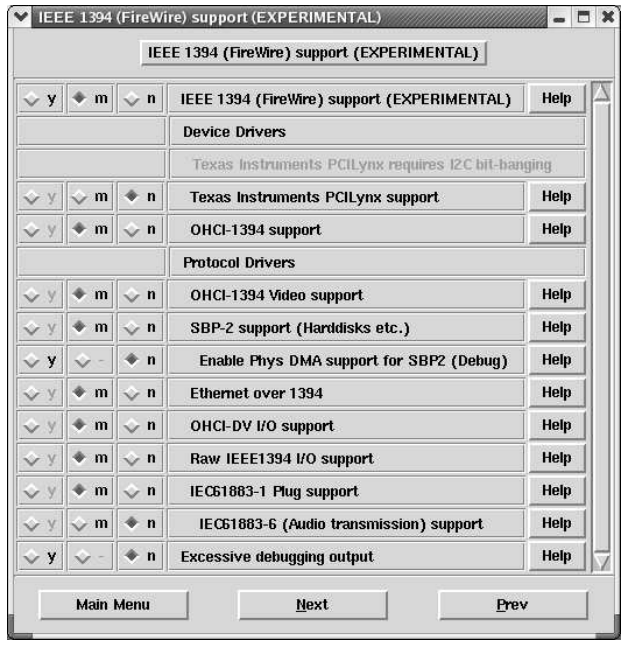
Jeżeli w systemie wykorzystywane jest jądro Linux w wersji niższej niż 2.4.18 lub jądro Linux z serii 2.2, należy wcześniej pobrać niezbędną łatkę (ang. *patch*) i zaaplikować ją na źródłach jądra:

2.2: `http://download.sourceforge.net/linux1394/ ... ... /ieee1394-2.2.19-20010527.gz`,

2.4.0/2.4.1: `http://download.sourceforge.net/linux1394/ ... ... /ieee1394-2.4.0-20010228.gz`,

2.4.2+: `http://download.sourceforge.net/linux1394/ ... ... /ieee1394-2.4.4-20010527.gz`)

Nawet, gdy posiadamy najnowsze stabilne jądro Linux (np. 2.4.20), warto pobrać aktualną wersję sterowników dla magistrali IEEE 1394 (przechowywane są one w repozytorium narzędzia Subversion –



Rys 3

`http://www.linux1394.org/viewcvs/branches/linux-2.4/ ...`
`... /linux-2.4.tar.gz?tarball=1&rev=773&root=Linux+IEEE-1394`), a następnie podmienić zawartość katalogu `drivers/ieee1394` w źródłach jądra.

Jądro należy skompilować i zainstalować w sposób typowy dla Linuksa lub wykorzystywanej dystrybucji Linuksa. Podczas jego konfiguracji w części Code maturity level options trzeba zaznaczyć opcję Prompt for development and/or incomplete code/drivers, a w części IEEE 1394 (FireWire) support (EXPERIMENTAL) opcje: IEEE 1394 (FireWire) support, OHCI-1394 (Open Host Controller Interface) support oraz Raw IEEE 1394 I/O support, albo jako moduły (jest to zalecane – ieee1394.o, ohci1394.o oraz raw1394.o), albo jako wkompilowane elementy jądra. Ponadto, jeżeli zamierzamy korzystać z przesyłania danych wideo w trybie izochronicznym, warto zaznaczyć opcję OHCI-1394 Video support (np. jako moduł: video1394.o), jeżeli z urządzeń pamięci masowej – opcję SBP-2 support (Harddisks etc.) (tylko jako moduł: sbp2.o), a jeżeli z interfejsu Ethernet – opcję Ethernet over 1394 (np. jako moduł: `eth1394.o`). Gdy posiadamy urządzenie z chipsetem TI PCILynx, należy, zamiast OHCI-1394, zaznaczyć opcję odpowiednią dla niego. Ilustruje to Rysunek 3.

Kolejnym krokiem jest zainstalowanie biblioteki libraw1394, która zapewnia programom bezpośredni dostęp do magistrali poprzez udostępniany przez jądro interfejs raw1394. Można tego dokonać korzystając ze źródeł (`http://download.sourceforge.net/libraw1394/libraw1394_0.9.0.tar.gz` – w standardowy sposób) albo z pakietu binarnego, np. rpm (`http://download.sourceforge.net/libraw1394/libraw1394-0.9.0-1.i386.rpm`) lub deb (`http://download.sourceforge.net/libraw1394/ ... ... /libraw1394-5_0.9.0-1_i386.deb`). W przypadku źródeł niezbędne jest także stworzenie odpowiedniego pliku urządzenia (`/dev/raw1394`).

Po instalacji, jeżeli było zmieniane jądro, można już przeładować system. Po ponownym jego uruchomieniu można załadować do pamięci odpowiednie moduły jądra, czyli przede wszystkim wykonać: modprobe ohci1394 oraz modprobe raw1394. W rezultacie w systemie powinny być dostępne wszystkie kluczowe elementy konieczne dla wykorzystania magistrali IEEE 1394.

W komputerach przenośnych, w których wykorzystywane są karty PCMCIA, ładowaniem odpowiednich modułów w dużej części zajmuje się podsystem wsparcia dla tych właśnie kart (wyjątkiem jest moduł raw1394.o oraz video1394.o).

Testowanie

Prawidłowe wykrycie przez jądro podłączonych urządzeń IEEE 1394 można zweryfikować wydając polecenie dmesg. Pojawienie się linii zaczynającej się od

```
ieee1394: Host added: Node[....
```

świadczy o dostępności danego węzła na magistrali. Po włożeniu karty kontrolera IEEE 1394 do gniazda PCMCIA komputera można zaobserwować np. takie wpisy:

```
cs: cb_alloc(bus 2): vendor 0x1033, device 0x00cd
PCI: Enabling device 02:00.0 (0000 -> 0002)
ohci1394: $Revision: 1.101 $ Ben Collins <bcollins@debian.org>
ohci1394_0: OHCI-1394 1.0 (PCI): IRQ=[11]  MMIO=[10400000-10401000]  Max Packet=[1024]
ieee1394: Host added: Node[00:1023]  GUID[0030952410002d13]  [Linux OHCI-1394]
po podłączeniu zewnętrznego dysku np. takie:
ieee1394: Device added: Node[01:1023]  GUID[0030ffa7e0000da7]  [Oxford Semiconductor Ltd.   ]
SCSI subsystem driver Revision: 1.00
ieee1394: sbp2: Logged into SBP-2 device
spurious 8259A interrupt: IRQ7.
ieee1394: sbp2: Node[01:1023]: Max speed [S400] - Max payload [1024]
scsi0 : IEEE-1394 SBP-2 protocol driver (host: ohci1394)
SBP-2 module load options:
```

```
- Max speed supported: S400
- Max sectors per I/O supported: 255
- Max outstanding commands supported: 8
- Max outstanding commands per lun supported: 1
- Serialized I/O (debug): no
  Vendor: WDC AC21  Model: 600H             Rev:
  Type:   Direct-Access                     ANSI SCSI revision: 06
```

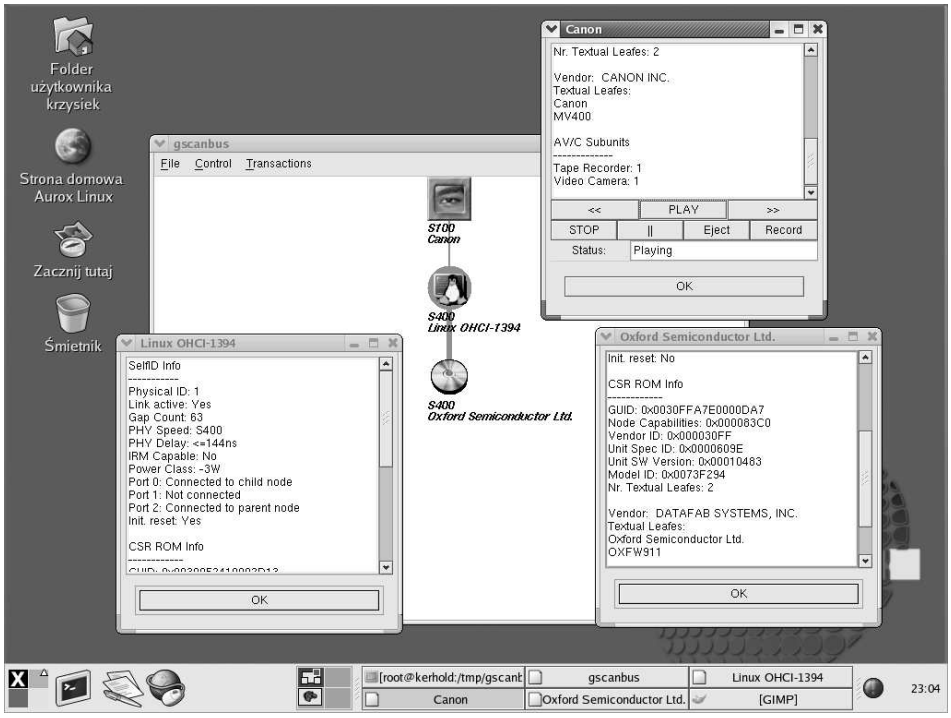
a po dołączeniu jeszcze kamery cyfrowej np. takie:

```
ieee1394: Node 01:1023 changed to 00:1023
ieee1394: sbp2: Reconnected to SBP-2 device
ieee1394: sbp2: Node[00:1023]: Max speed [S400] - Max payload [1024]
ieee1394: Node 00:1023 changed to 01:1023
ieee1394: Device added: Node[02:1023]  GUID[00008500003bae06]  [Canon]
```

Uruchomienie narzędzia testlibraw z pakietu biblioteki libraw1394 spowoduje wyświetlenie dodatkowych informacji. Przy dostępności trzech urządzeń mają one taką postać:

```
successfully got handle
current generation number: 3
1 card(s) found
  nodes on bus:  3, card name: ohci1394
using first card found: 3 nodes on bus, local ID is 1, IRM is 2
doing transactions with custom tag handler
trying to send read request to node 0... failed with error: Invalid argument
trying to send read request to node 1... completed with value 0xf0368acc
trying to send read request to node 2... completed with value 0xd2398acc
using standard tag handler and synchronous calls
trying to read from node 0... trying to read from node 1... completed with value 0x87c49dcc
trying to read from node 2... completed with value 0x93c89dcc
testing FCP monitoring on local node
got fcp command from node 1 of 8 bytes: 01 23 45 67 89 ab cd ef
got fcp response from node 1 of 8 bytes: 01 23 45 67 89 ab cd ef
polling for leftover messages
```


Zadaniem tego narzędzia jest odczytanie danych ze wszystkich urządzeń podłączonych do magistrali IEEE 1394. Sprawdza ono, czy biblioteka libraw1394 jest zainstalowana poprawnie, podsystem IEEE 1394 w odpowiedniej wersji jest załadowany do jądra, biblioteka libraw1394 jest w stanie komunikować się z jądrem poprzez urządzenie `/dev/raw1394`, a podłączone węzły są osiągalne i odpowiadają na operację odczytu danych.



Rys 4

Przydatną aplikacją, która pozwala na graficzną wizualizację topologii magistrali, jest Gscanbus (`http://download.berlios.de/gscanbus/ ... ... /gscanbus-0.7.1.tgz`). Po jej zainstalowaniu i uruchomieniu można dowiedzieć się szczegółowych informacji o wszystkich węzłach magistrali poprzez kliknięcie myszką reprezentujących je ikonek. Ilustruje to Rysunek 4.

Aby zaobserwować proces automatycznej konfiguracji magistrali, warto dokonać dalszych eksperymentów podczas pracy komputera poprzez odłączanie i przyłączanie do magistrali innych urządzeń peryferyjnych.

DYSK TWARDY

Po podłączeniu dysku twardego do magistrali IEEE 1394 można uzyskać do niego dostęp wykorzystując protokół SBP-2 (Serial Bus Protocol) – zapewnia on wsparcie dla urządzeń pamięci masowej (dyski twarde, napędy CD/DVD, nagrywarki CD/DVD) przeznaczonych dla tej magistrali.

W Linuksie obsługę standardu SBP-2 zapewnia moduł `sbp2.o`, który jest sterownikiem wysokiego poziomu w podsystemie IEEE 1394, a także niskiego poziomu w podsystemie SCSI – pomiędzy nimi następuje tunelowanie komend. Oznacza to, że do prawidłowego działania omawianych urządzeń pamięci masowej niezbędne jest posiadanie podstawowych sterowników SCSI wkompilowanych w jądro lub dostępnych w postaci modułów. Moduł `sbp2.o` powinien zostać załadowany dopiero, gdy będą już załadowane sterowniki SCSI oraz IEEE 1394. W komputerach przenośnych, w których jako kontroler magistrali IEEE 1394 jest wykorzystywana karta PCMCIA, ta operacja nie będzie konieczna – podsystem obsługi tych kart sam się wszystkim zajmie.

Aktualnie sterownik SBP-2 podłącza wykryte przez siebie urządzenia pamięci masowej do podsystemu SCSI tylko podczas ładowania go do pamięci lub po resecie magistrali IEEE 1394. Aby wykryć późniejsze dodanie lub usunięcie urządzenia, trzeba skorzystać ze specjalnie przygotowanego skryptu (`http://www.garloff.de/kurt/linux/rescan-scsi-bus.sh`), który dokonuje ponownego przeskanowania magistrali SCSI, albo prostych operacji w ramach systemu plików procfs: `add-single-device` oraz `remove-single-device`. W pierwszym przypadku wystarczy tylko uruchomić skrypt, a w drugim wykonać polecenie:

```
echo "scsi add-single-device 0 0 0 0" > /proc/scsi/scsi
```

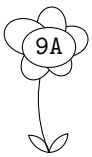
lub

```
echo "scsi remove-single-device 0 0 0 0" > /proc/scsi/scsi
```

gdzie odpowiednie parametry (liczby) to numer kontrolera, numer magistrali, numer urządzenia oraz jego identyfikator. Istnieje także możliwość sprawdzenia, jakie urządzenia są aktualnie dostępne w systemie:

```
cat /proc/scsi/scsi
```

Jeszcze innym rozwiązaniem jest przeładowanie samego sterownika SBP-2.



Po załadowaniu sterownika SBP-2 podłączony dysk twardy jest widoczny jako dysk twardy SCSI, a to oznacza, że dostęp do niego jest możliwy poprzez urządzenia SCSI (w tym przypadku /dev/sda). Zamontowanie pierwszej partycji takiego dysku w systemie plików jest wykonywane zatem za pomocą standardowego polecenia:

```
mount /dev/sda1 /mnt
```

Możliwe jest także wykonywanie na takim dysku wszystkich typowych operacji dla dysków twardych, takich jak podział na partycje czy zakładanie nowego systemu plików.

Szybkość tak podłączonego dysku nie jest ograniczana przez magistralę IEEE1394, która zapewnia mu pasmo przenoszenia S400 (400 Mbit/s), lecz przez jego możliwości transferu. W celu zwiększenia wydajności takiego rozwiązania preferowanym systemem plików jest ext2 zamiast np. fat/fat32.

Kamera cyfrowa

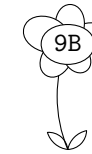
Wykorzystanie kamery cyfrowej podłączonej do magistrali IEEE 1394 jest możliwe za pomocą dwóch modułów (interfejsów): raw1394 oraz video1394. Różnica pomiędzy nimi polega głównie na wydajności, gdyż video1394 używa kanałów DMA do przeprowadzania transmisji w trybie izochronicznym, a to jest szybsze i w mniejszym stopniu obciążające procesor.

Od jądra 2.4.19 zostały zmienione główne i podrzędne numery urządzenia /dev/video1394 z 172/0 na 171/16. Ponadto w związku z wprowadzeniem wsparcia dla devfs plik urządzenia `/dev/video1394` został dostosowany do nowego schematu nazewnictwa: `/dev/video1394/0` (171/16), `/dev/video1394/1` (171/17) itd. Jeżeli w systemie nie jest jednak wykorzystywany devfs, plik urządzenia można zrobić ręcznie:

```
mknod -m 666 /dev/video1394 c 171 16
```

Karta sieciowa

Wykorzystanie magistrali IEEE 1394 w celu zbudowania sieci lokalnej do wymiany danych jest możliwe za pomocą modułu eth1394. Implementuje on protokół przesyłania datagramów IPv4 poprzez magistralę szeregową IEEE 1394, który został zdefiniowany w RFC 2734. Aktualnie przeznaczony on jest jednak tylko do testów, gdyż nie zapewnia pełnej funkcjonalności oraz jest bardzo niestabilny.



Programy

DVgrab

Najprostszym narzędziem, które można wykorzystać podczas pracy z kamerą cyfrową, jest DVgrab (`http://kino.schirmacher.de/download/dvgrab-1.1b2.tar.gz`). Jego jedynym zadaniem jest pobranie danych wideo i audio standardu DV z urządzenia oraz zapisanie ich w pliku, np.:

```
dvgrab -frames 1000 dvgrab
```

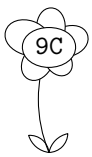
spowoduje zapisanie 250 sekund danych wideo (dla standardu PAL) w formacie avi (kodek `dv1`) do pliku `dvgrab001.avi`. Tak otrzymane dane można potem przetwarzać w innych programach do edycji wideo.

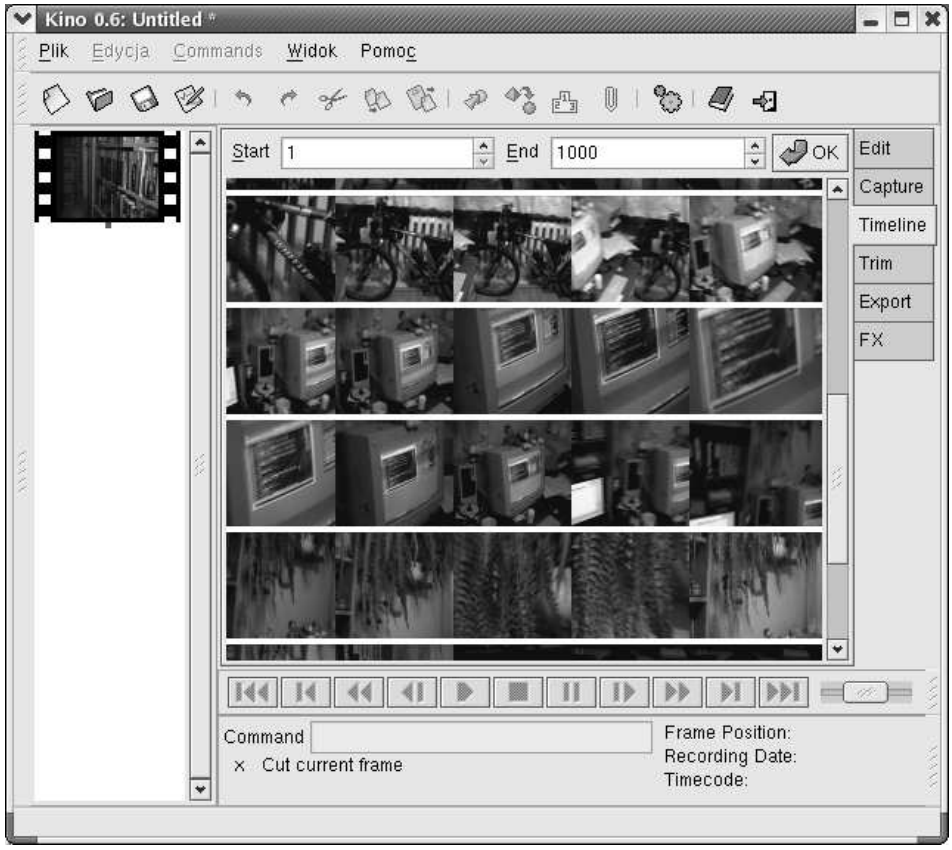
Narzędzie DVgrab jest standardowo dostępne w większości popularnych dystrybucji.

KINO

Kino (`http://kino.schirmacher.de/`) jest nieliniowym edytorem danych w standardzie DV, ściśle zintegrowanym z podsystemem IEEE 1394. Pozwala on na sterowanie kamerą cyfrową, w tym przechwytywanie danych z niej oraz przesyłanie ich z powrotem. Dane zapisuje do plików w formacie RawDV lub Avi (kodeki dv1 oraz dv2).

Instalację pakietu można przeprowadzić albo ze źródeł, albo z pakietów binarnych. W przypadku Auroksa 8.0 przygotowane zostało wygodne repozytorium dla apt-rpm (`http://kino.schirmacher.de/article/view/44/1/11/`), więc po odpowiedniej jego konfiguracji wystarczy wydać komendę apt-get install kino – zainstaluje ona Kino oraz przydatne narzędzia (mjpegtools) oraz biblioteki (libdv, libavc1394).

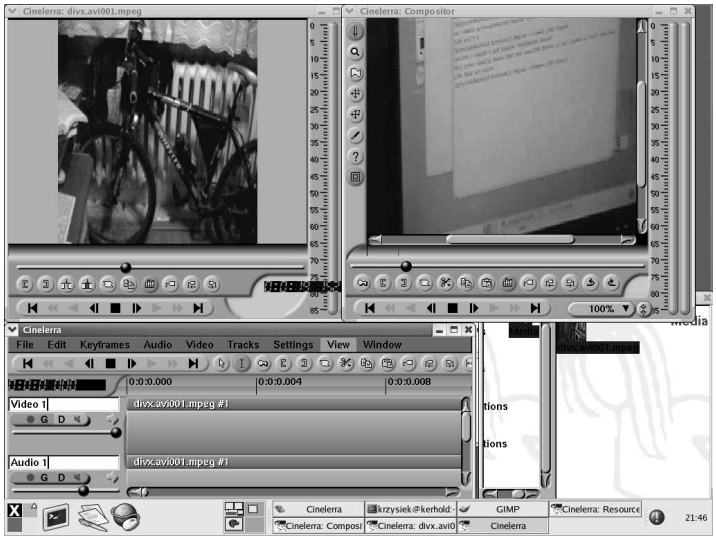




Rys 5

Wykorzystując Kino można załadować wiele plików wideo, wycinać i wklejać fragmenty danych wideo/audio oraz zapisywać wyniki (projekt) w postaci listy decyzji edycyjnych (format SMIL XML). Ilustruje to Rysunek 5.

Za pomocą zewnętrznych narzędzie możliwe jest ponadto eksportowanie danych w formatach: pojedynczych klatek (biblioteka Imlib1: ppm, gif, tif, jpg, png itd.), mp3 (lame), ogg (oggenc) oraz MPEG-1, MPEG-2 i DivX (mjpegtools).



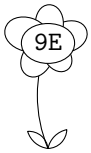
Rys 6

CINELERRA

Cinelerra jest zaawansowanym pakietem do obróbki danych wideo. Posiada on wbudowane wsparcie dla pobierania danych w formacie DV z kamery cyfrowej. Ilustruje to Rysunek 6.

ZAKOŃCZENIE

Aktualnie trwają intensywne prace nad zwiększeniem funkcjonalności podsystemu IEEE 1394 w Linuksie w ramach rozwoju jąder serii 2.5. Tylko niektóre z nowych rozwiązań jest przenoszone do serii jąder 2.4, więc wszystkie osoby, które są zainteresowane najnowszymi możliwościami wykorzystania magistrali IEEE 1394 w Linuksie, powinny sięgnąć po najnowszą rozwojową wersję jądra oraz pobrać nowe sterowniki z systemu Subversion projektu Linux1394. Dla przykładu w jądrach serii 2.5 trwają prace nad nowym API o nazwie rawiso, bazującym na raw1394, które wyeliminuje konieczność istnienia modułów video1394, dv1394 oraz amdtp.



Przyszłość samej magistrali IEEE 1394 przedstawia się także interesująco, np. firma Philips Semiconductor przedstawiła niedawno możliwość wykorzystania magistrali IEEE 1394 za pomocą sieci bezprzewodowych.

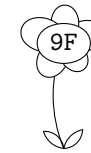
GNOKII
Then and Now

Hugh Blemings
Paweł Kot

The early days—gnokii then

To their credit, Nokia was one of the first mobile phone manufacturers to make a suite of software available to interface their mobiles to a computer. Their suite, "Nokia Cellular Data Suite" (NCDS) is Windows based and provides a complete set of tools for manipulating phonebook entries, sending SMS messages etc.

The version available in 1998 also allowed the mobile to be used as an AT compatible modem. Unlike modern models, this was something the handsets were unable to do standalone. Essentially the software provided a virtual serial driver that sat between the standard Windows serial layer and the phone. The driver translated between Hayes AT style modem commands and the proprietary protocol used by the phones (the so called FBUS or MBUS protocols).



The catalyst for what became gnokii was basically that Hugh had a Nokia 3810 which he used with a laptop running dual boot Linux/Windows. Not being able to use the 3810 under Linux meant no mobile 'net access without reboot, clearly an untenable situation :)

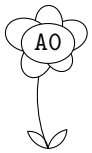
The project initially supported the then current 3810/3110/8110 model series—another project had started for the then new 6110 series but not generated any code. After a short dialogue we combined the two efforts figuring (correctly as it happened) that much of the higher level code would be common anyway. Pavel Janík based in Brno, Czech Republic became co-author having contributed the majority of the 6110 series code. Discussions in mid October 1998 to "formalise" the project lead to `gnokii-0.0.1` being released on January 26th, 1999.

During the early stages there was some dialogue back and forth with Nokia, eventually leading to contact with the gentleman responsible for the NCDS project itself. Regrettably after a promising start the discussions stalled in June 1999 on the matter of an open source release. Nokia's concern it seemed revolved around potential liability if someone used the information they provided to defeat SIM locks and the like. Ironically this was worked out by people outside the gnokii project anyway...

A quick introduction to early FBUS protocols

By the time we reached the impasse with Nokia, there had actually been several releases of gnokii and we were well underway in understanding the protocols in use. To understand what we had to work out, a little explanation of how the FBUS protocol and GSM phones operate will assist. Note that what follows is what we now know, how we came to know this is covered below.

Fax and data calls on a GSM network actually send binary data over the air interface using the Radio Link Protocol (RLP). Some munging of the data stream is done depending on the other end of the link. If the other end is an ISDN connection the data stays in the digital domain the whole time. If the other end is an analogue modem or fax then a device at the far end of the network modulates/demodulates the RLP data into the baseband signals required by the far end's analogue modem.



What follows relates to the 3110/3810/8110 series, the 6110 series uses a completely different and rather more complex variant of FBUS.

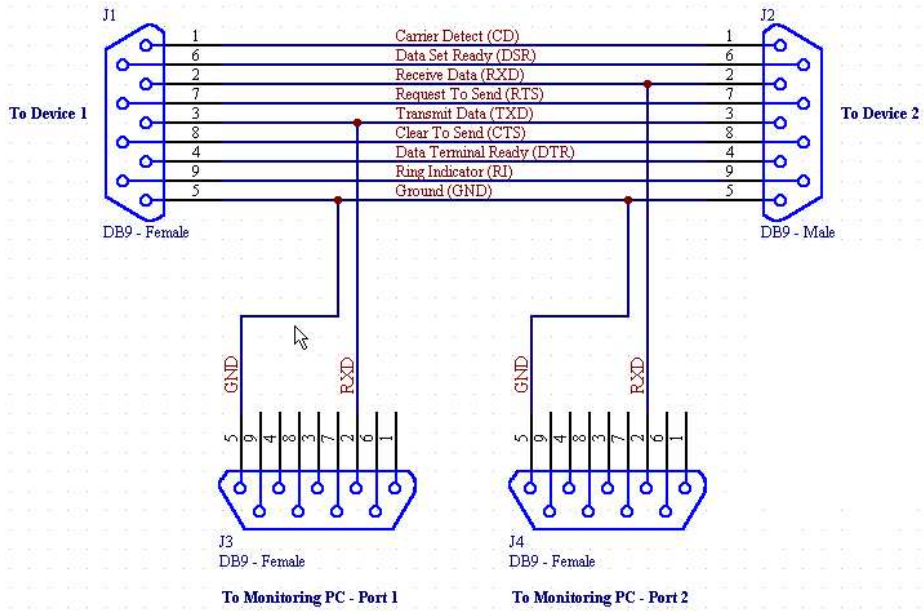
Most of the actions over FBUS followed a command/response sort of model. The phone would also send some messages asynchronously in response to events occurring on the network such as incoming calls, SMS messages arriving etc. When run in data or fax mode (we never supported the latter on the 3110 series), the phone streams raw RLP frames off the GSM network down the interface and similarly expects RLP frames to be sent to it. To get data calls going we had to write our own RLP implementation from the relevant

ETSI specifications. No small task but the Open Source Development model prevailed.

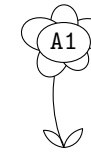
Working it out ourselves

The FBUS protocol documentation wasn't available to us so we set about working it out ourselves. Different members of the team used slightly different setups but the basic idea was to employ a second PC and a "sniffer" cable to monitor data going between the phone and the PC running NCDS. Contrary to popular belief for a number of reasons we never took the approach of disassembling the NCDS binary.

For the curious, an example serial sniffer cable is shown below



At the time we started working on gnokii there weren't any suitable open source tools for sniffing serial protocols. Some suitable C code was cobbled together for the purpose. This would output data flowing in one direction enclosed in square brackets, data in the other in braces with the hexadecimal byte and ASCII character (if valid) in between them.



```
{01 }{02 }{4aJ}{16 }{5f_}[04 ][02 ][4aJ][0e ][42B][04 ][05 ][4bK][15 ][01 ][03 ]
[02 ][5f_]{01 }{02 }{4bK}{1d }{55U}{01 }{02 }{3f?}{17 }{2b+}[04 ][02 ][3f?][0f ]
[366][04 ][1b ][41A][16 ][02 ][00 ][00 ][00 ][00 ][00 ][00 ][a7 ][01 ][01 ][00 ]
[0c ][2b+][366][311][344][311][311][399][399][300][300][300][300][00 ][f9 ]{01 }
```

The basic code had a number of limitations, most notable was that it didn't really address time alignment of the received data—thus it wasn't clear from the output from the sniffer whether the PC or phone sent data first. This was a bit of a nuisance initially but it quickly became apparent what was going on.

This tool was used to capture data corresponding to the "idle" state of the phone to PC connection. The idle "heartbeat" provided sufficient information to guess the basic protocol. Further captures were done while various actions were performed such as reading/writing phone book entries etc. A capture was done at startup to determine the initial handshaking bytes.

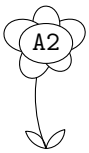
There was now enough information to write a simple state machine based parser. This was used to look at incoming bytes and decode them into the appropriate fields. When an unknown message was received it was dumped in the following fashion.

```
To: MT43 SN11 CS51 [02 ][04 ]

Fr: MT46 SN17 CS79 [04 ][48H][6fo][6dm][65e][0c ][2b+][366][311][322][366]
                   [322][366][322][366][300][311][366]
```

The capture above shows Hugh's old home phone number being retrieved from the phone by NCDS. The message to the phone is Message Type `0x43` with data fields `0x02` specifying SIM memory and location `0x04`. In the response—Message Type `0x46`—the `0x04` byte is the length of the name field and 0x0c is the length of the number field. Rest is pretty self evident...

The early releases



The result of this early work was the gnokii project, then lead by Pavel Janík and Hugh Blemings. gnokii was developed as a free software project, released under the GNU General Public Licence.

Initially, gnokii was just a command line test tool with support for the phones mentioned above. Over time, the test tool became more and more powerful and the support for the phones matured.

As the project progressed, other tools were created. Besides the `gnokii` command line there are three main tools: `gnokiid`, `xgnokii` and `smsd`.

Gnokiid is the virtual modem application provided for Nokia the 6110 family phones. These mobiles don't have AT command support but they are able to make data calls. Gnokiid provides a software modem emulator that passes the data to the phone using the FBUS protocol. Gnokiid creates a `/dev/gnokii` device for communication. It is only used with the 6110, 6130, 6150, 5110 and 5130 phones.

Xgnokii, as the name suggests is a X based application that provides a graphical interface to the phone. Amongst other things, xgnokii allows you to read and write phonebook entries, read, write and send SMS messages and monitor battery and received signal strength.

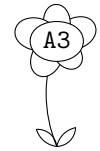
The SMSD (SMS daemon) program is intended for receiving and sending SMS's. The program is designed to use modules (plugins) to work with an SQL server. Currently supported SQL engines are PostgreSQL, MySQL and a special module '`file`' which is designed to work without an SQL database.

Changes in the team

Unfortunately the gnokii development process has not always been peaceful. Around 3 years ago gnokii passed though something of a leadership crisis. Marcin Wiącek, one of the gnokii contributors, became frustrated that his patches sent to the gnokii list were dropped by the gnokii team leaders and started publically complaining about gnokii maintainership. For both Hugh and Pavel external pressures had simply got to the point where they could no longer devote the time required.

As a result, Chris Kemp and Paweł Kot took over the leadership of the project. For various reasons it was decided not to provide Marcin with direct CVS access. While the new core team tried to find consensus with Marcin it eluded all concerned and he forked his own project called mygnokii based on `gnokii-0.3.3-pre8`.

Since then there were efforts to merge the projects but none succeeded. Marcin also dropped the mygnokii project and decided to write a new version from scratch under the name "mygnokii2" The project was later renamed to gammu. Unfortunately there's no official communication between the two projects at present.



Gnokii now

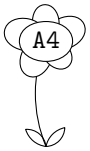
As discussed in the preceding text, many things have changed since the project began. Many users and developers have contributed to the project, too many

to mention them all here. The CREDITS file in the gnokii sources contains a mostly complete list of contributors. The project evolved and began to support more and more different phones. Currently most popular Nokia phones are supported as well as the AT capable models. The table below shows driver, its current support and the phones supported by the driver.

DRIVER	PHONES	SUPPORT
nk2110	Nokia 2110, 2140, 6080	outdated
nk3110	Nokia 3110, 3810, 8110, 8110i	rising again :-)
nk6100	Nokia 6110, 6130, 6150, 6190, 5110, 5130, 5190, 3210, 3310, 3330, 3360, 3410, 8210, 8290	full
nk6160	Nokia 6160, 5120	partial
nk7110	Nokia 6210, 6250, 7110	mostly complete
nk6510	Nokia 6310, 6510, 8310	mostly complete
atgen	Nokia 6210/7110/8210/6310/6510, Motorola Timeport P7389i (L series), Siemens S25/SL45i, ...	improving

The current gnokii development team is much larger than it used to be, in alphabetical order:

- Borbely Zoltan <bozo@andrews.hu>
  Bozo takes care of: build system, gnokiid, statemachine, link layer, nk6110 and nk6210 drivers.
- Chris Kemp <ck231@cam.ac.uk>
  Chris takes care of statemachine, ringtones support and link layer.
- Jan Derfinak <ja@mail.upjs.sk>
  Jan is the creator and maintainer of xgnokii and smsd.
- Ladislav Michl <ladis@linux-mips.org>
  Ladis joined quite recently. He looks after link layer and AT driver (mainly duncall and cbus support).
- Manfred Jonsson <manfred.jonsson@gmx.de>
  Manfred takes care of device layer and AT driver.
- Markus Plail <plail@web.de>
  Markus takes care of the nk6210 and nk6510 drivers. All credits for the latter driver go to Markus. Markus also maintains xgnokii.
- Pavel Machek <pavel@suse.cz>
  Pavel takes care of: libsms, bitmaps, ringtones and nk2110 and AT drivers.
- Paweł Kot <pkot@linuxnews.pl>
  Paweł is current project leader. He authored the updated libsms based on the previous implementation, GSM specifications and dif-



ferent extensions. Paweł maintains: gnokii, libgnokii, libsms, build system, documentation (yes, it sucks), bitmap support, nk3110, nk6110, nk6210, nk6510 and AT drivers.

Hugh Blemings and Pavel Janík are still with us but are not as active as they used to be. They still chip in the occasional hint and help out with some organisational issues.

There are some other people involved in the gnokii development. For example, thanks to Marcel Holtmann, one of the Linux Bluetooth stack maintainers, we have reliable Bluetooth support.

Current gnokii design

Gnokii has evolved over nearly five years. We don't claim that it is a perfect design, but it is getting better with the every update.

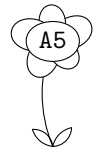
Gnokii is currently supported on Linux, win32, MacOS X, FreeBSD and Solaris operating systems. The most active development is of course being done on Linux, but win32 and MacOS X people are doing a good job as well. FreeBSD and Solaris ports get synced from time to time.

The major difference between the current version of gnokii and the previous ones is that all base functionality is provided by libgnokii. This is a library with a well defined API that can be linked to an external application dynamically or statically. With previous versions one had to grab the whole gnokii sources and link with the object files built during gnokii compilation.

At some stage we stopped using threads in the internal structures. Using them didn't add much and caused portability and debugging problems. This means that libgnokii needs to work in synchronous mode, but this isn't proving to be much of a disadvantage so far as all supported devices work this way.

libgnokii is internally split into various layers. The lowest layer is the device layer. Currently we support serial ports on all operating systems as well as IrDA and Bluetooth (BlueZ stack) on Linux.

Above the device layer is the link layer. This layer provides FBUS and MBUS as the main protocols. It also provides a layer for AT capable phones and other, less popular protocols.



Above all these is the phone driver. It contains the phone series specific code.

At the highest level of abstraction there are miscellaneous sublibs that are responsible for handling various kinds of the functionality: sms support, calendar support, bitmaps support, ringtone support, etc. With the earlier gnokii

releases this layer did not exist and it is still not present in some functional areas but it is proving to be a step in the right direction.

In the middle, across all these layers is a state machine, data structures and functions that allow us to use a stateful connection.

Gnokii usage

gnokii is the command line tool, primarily developed to test the implemented functionality, but with time it grew to the most powerful tool in the gnokii package.

There are few main areas of gnokii functionality: sms support, phonebook handling, calendar handling, security functions and other.

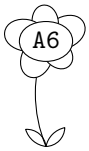
SMS support is an area where a lot of things left to do. We can send and receive many types of the messages: plain text, unicode text, bitmaps (operator logos, caller icons, startup logos), ringtones, picture messages, concatenated messages (ie. messages containing more then 160 characters from the default alphabet). Moreover you can save a SMS message to a mailbox and then read it with Your Favourite Mail Client (tm). It is worth pointing out that gnokii provides the possibility to send a flash SMS, eg. SMS that is immediately shown on the phone display instead of saving it to the SIM card or the phone memory.

Newer Nokia phones have more SMS capabilities in regard to storing them. Gnokii supports all folders features that Nokia provides.

Phonebook and calendar support offer full read-write access to phone/SIM card data. vCard and vCalendar formats are supported.

More interesting features are the "security" options. You can get the state of the phone, eg. get the info that the phone is waiting for you to input the PUK number. Then you can enter this number and even receive the code from the phone. These options are not enabled by default, you need to configure gnokii with `--enable-security`.

Gnokii now provides a total of 66 different command line switches and almost every switch has multiple options available.



All this functionality is also provided by xgnokii. Some of the features are of course better accessible with GUI, so you probably want to try out xgnokii as well.

Future goals

Our next goal is to achieve multiple phone support. We have done much work in this direction and in theory, the framework should support it, but no testing has been done so far.

Much work has been done last year in the internal gnokii design area. We've put much effort into making gnokii more flexible and more extensible.

The "SMS industry" also goes forward. There are new SMS types: EMS, MMS, which are to be supported by gnokii in the future.

The main area of the further gnokii development is the user interface for the functionality provided by libgnokii. As of 0.5.0 version gnokii project provides libgnokii with stable API to be used in the external applications. All user applications from the gnokii project: gnokii, gnokiid, smsd and xgnokii, make use of libgnokii.

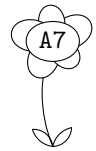
More applications are planned. Recently two projects began to establish: new GUI designed for Windows in C++ and new GUI written in gtk+ 2.0.

New cellular phones provided by mobile vendors permanently surprises us with their functionality. Is seems that there will always be something to add for gnokii.

The other direction of gnokii development is the integration with the third party application. Such application can be OpenOBEX, that we want to use for the Bluetooth file transfer to/from the phone, or misc PIMs that we want to synchronise data with.

The history of gnokii has been a lot of fun but not without the occasional difficulty. We're looking forward to improving it with every new release and thank those that have helped bring us this far.

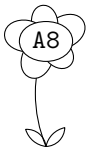
February 2003
Hugh Blemings
Paweł Kot



Linux na terminalach

Daniel Koć <kocio@linuxnews.pl>

Jednym z problemów przy wdrażaniu Linuksa do instytucji jest bariera sprzętowa: współczesne oprogramowanie na biurko znane jest ze swojej zasobożerności, a zakup odpowiedniej klasy maszyn to niebagatelny wydatek, który może ostudzić zapał nawet największych entuzjastów. Tymczasem istnieje ekonomiczne rozwiązanie, oparte na lokalnej sieci – Linux Terminal Server Project. Potrzebne programy są wówczas uruchamiane po stronie serwera, co pozwala efektywnie wykorzystać nawet te stare pecety, które do niczego innego już by się nie nadawały. Zdaniem twórców projektu dobrze wyposażony serwer jest w stanie obsłużyć do 40 takich stacji.



Pożytek ze smoka

Synonimem komputera dla większości z nas jest komputer osobisty – maszyna, z której korzystamy na co dzień w domu, pracy czy szkole, ale faktem jest już także powszechna obecność sieci informatycznych, zarówno Internetu jak i sieci lokalnych (LAN), gdzie korzystamy z usług oferowanych przez różnego rodzaju serwery. Burzliwy wzrost zasięgu, prędkości i niezawodności tych sieci sprawia, że powoli urzeczywistnia się nienowe już hasło: „dopiero sieć to komputer”. Sprawna komunikacja między maszynami powoduje, że zacierają

się granice między zasobami zdalnymi i lokalnymi, komputer przed którym siedzi użytkownik staje się tylko oknem na świat, a naczelnym problemem stworzenie poręcznego interfejsu do poruszania się po tym świecie.

W praktyce jednak różnice w prędkości przesyłu dzielą środowisko pracy na trzy części: wydajną maszynę lokalną, przewidywalną sieć firmową oraz praktycznie niekontrolowane obszary sieci zewnętrznych. Często rolę serwera sieci lokalnej w małych i średnich firmach, szkołach i innych podobnych organizacjach pełni Linux. Jego typowe zadania w tej roli ograniczają się do udostępniania przestrzeni dyskowej i bezpiecznego wyjścia do Internetu, względnie witryny WWW i anonimowego FTP, oraz obsługi wewnętrznej poczty. O ile nie jest to stare 486 tylko standardowy pecet, to przez większość czasu nie wykorzystuje nawet ułamka swojej mocy, a przecież taki smok aż się prosi o dociążenie!

Tymczasem na biurkach użytkowników dzieje się nieustający dramat: szybko starzejące się moralnie stacje robocze, odstające parametrami od gwałtownie rosnących wymagań współczesnego oprogramowania, są w stanie przyprawić o zapaść budżet każdej, nawet zasobnej instytucji. One zresztą także nie są wykorzystywane optymalnie, ponieważ tylko w wyjątkowych przypadkach komputer nie spędza dnia głównie w oczekiwaniu na reakcję człowieka, a maksymalne obciążenie trwa każdorazowo tylko chwilę.

Ziarnko do ziarnka

Odpowiedzią na bolączki takiej – dość powszechnej – sytuacji, jest nieco inny niż tradycyjnie podział zadań między serwerem a klientami. Wykorzystamy tu fakt, że sieć lokalna ma zwykle duży zapas przepustowości, oraz elastyczność środowiska X Window, które pozwala na przeźroczyste uruchamianie aplikacji na zdalnym komputerze tak, jakby były odpalane lokalnie.

Są to tylko najbardziej podstawowe założenia – w rzeczywistości sprawa jest znacznie bardziej skomplikowana i wymaga zaprzęgnięcia do pracy jeszcze kilku dodatkowych mechanizmów. Dlatego zamiast ponownie wymyślać koło, lepiej skorzystać z gotowego rozwiązania jakim jest Linux Terminal Server Project i jedynie dostosować je do potrzeb danej instytucji.

Zasadniczy pomysł polega na przeniesieniu ciężaru wykonywania programów na serwer, w rękach operatora pozostawiając tylko wprowadzanie i odczytywanie danych przez uproszczony terminal, a więc jest odkurzeniem technologii z czasów pierwszych komputerów. Podobne są wprawdzie motywacje zestawienia takiego scentralizowanego układu (oszczędność), jednak obecnie wystawienie nawet dosyć potężnego serwera nie przekroczy zapewne kosztu kilku standardowych maszyn. Współczesne serwery są także dużo mniejsze od

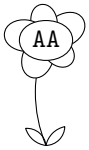
swoich protoplastów, a miejsce kart perforowanych i wydruków zajmuje normalne stanowisko z klawiaturą i myszką oraz graficznym interfejsem na monitorze, choć niekoniecznie najnowsze i niekoniecznie w ogóle z twardym dyskiem...

Chwilowe wzrosty aktywności procesów uruchamianych przez różne osoby teoretycznie nie powinny na siebie zbyt często nachodzić, stąd efekt jakby każda z nich dysponowała całym serwerem tylko dla siebie. Tajemniczy skokowy przyrost wydajności przy stosunkowo niewielkim nakładzie środków wynika po prostu z bardziej sprawiedliwego wykorzystania tych marnujących się zazwyczaj zasobów, podobnie jak dzieje się to na przykład podczas współdzielenia w sieci drukarek, gdy zakup dużo droższego modelu może się okazać tańszy w eksploatacji, oferując jednocześnie prędkość i funkcje niedostępne w urządzeniach przeznaczonych dla pojedynczych stanowisk.

Per aspera...

Ponieważ nieco zawiły proces instalacji jest szczegółowo udokumentowany na stronie `http://ltsp.org/instructions-3.0.html`, ograniczę się do krótkiego objaśnienia jego kolejnych kroków. Polecam także lekturę czytelnej instrukcji dla Mandrake 9.0, opublikowaną w końcowej części prezentacji leżącej pod adresem `http://ltsp.org/contrib/tim-ltsp.pdf`. W dalszym opisie będę się posiłkował podanym tam przykładem.

Pierwszy etap polega na uruchomieniu wyżej wspomnianych dodatkowych usług na serwerze. Są to: NFS (udostępnianie przestrzeni dyskowej serwera), DHCP (centralne nadawanie stacji roboczej numeru IP) oraz TFTP (uproszczony sposób transmisji plików po sieci). Następnie pobieramy i instalujemy pakiety LTSP dla swojej dystrybucji w odpowiedniej wersji: `ltsp_kernel` (jądro Linuksa dla terminali), `ltsp_core` (pakiet podstawowy, umożliwia zdalną pracę w trybie tekstowym), `ltsp_x_core` (środowisko graficzne XFree86 4.x) oraz `ltsp_x_fonts` (zestaw czcionek dla trybu graficznego). W przypadku posiadania karty graficznej nie obsługiwanej przez `XFree86` z pakietu `ltsp_x_core` (np. popularnej S3) należy zainstalować odpowiedni pakiet z nazwą rozpoczynającą się od `ltsp_x336` (zawiera serwer XFree86 3.3.6 dla danej karty).



Dodatkowo można także doinstalować pakiety: `ltsp_local_apps` (obsługa aplikacji uruchamianych lokalnie na terminalach), `ltsp_local_netscape` (Netscape przygotowany do pracy w trybie lokalnym), `ltspwebcam` (obsługa kamery), `ltspsound` (obsługa karty dźwiękowej), `lts_scanner` (obsługa skanera) czy `ltsp_initrd_kit` (zestaw do tworzenia obrazu ściąganego wraz z jądrem) i `ltsp_util_src` (źródła narzędzi napisanych dla LTSP), oraz inne dostępne, wedle potrzeb.

Jeżeli serwer należy dopiero postawić i nie mamy nic przeciwko Red Hatowi, to warto zainteresować się specjalizowaną dystrybucją K12 Linux Terminal Server Project, która składa się z gotowych płytek z Red Hat Linux zawierających ponadto najświeższe łaty bezpieczeństwa oraz potrzebne pakiety LTSP, co skróci czas przygotowania rekwizytów i uniezależni nas od obecności i stanu sieci zewnętrznej. Adresy tej i podobnych do niej dystrybucji Czytelnik znajdzie w odnośnikach na końcu artykułu. Gotowe pakiety można pobrać także dla różnych wersji standardowych dystrybucji Red Hat, Mandrake, SuSE, Debian, Caldera eDesktop i eServer oraz Conectiva.

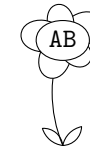
Konfiguracja serwera terminali odbywa się po przejściu do katalogu `/opt/ltsp/templates` i uruchomieniu z prawami administratora skryptu `./ltsp_initialize`. Teraz wykonujemy dowiązanie symboliczne:

```
ln -s /tftpboot /var/lib/.
```

i przystępujemy do konfiguracji stacji roboczych w plikach: `/etc/dhcpd.conf` (przypisanie numerów IP do numerów MAC, czyli do fizycznych kart sieciowych), /etc/hosts (przypisanie nazwy do tych numerów IP), `/opt/ltsp/i386/etc/lts.conf` (ustalenie opcji pracy terminali).

Przedostatnią czynnością instalacyjną jest umożliwienie startu terminala. Niektóre karty sieciowe są do tego przygotowane, jednak w typowym przypadku wystarczy odpowiednio spreparowana dyskietka startowa (potem należy jedynie przestawić odpowiednią opcję w BIOS-ie). Najprościej pobrać jej obraz z serwisu `ROM-o-matic.net`, gdzie po wybraniu modelu karty sieciowej dla konkretnych stacji oraz formatu `.lzdsk` dostajemy niewielki pliczek do zapisania na dyskietce przy pomocy dd (Linux)/rawrite (DOS). Uwaga: trzeba wybrać serię stabilną (5.0.x), gdyż seria rozwojowa (5.1.x) nie współpracuje z LTSP! Teraz tylko uruchamiamy na serwerze usługi NFS, DHCP oraz TFTP i wszystko jest już przygotowane do próby generalnej.

...AD ASTRA



Skołowanemu Czytelnikowi należy się wyjaśnienie co tak właściwie zrobił przechodząc powyższą ścieżkę zdrowia. Najlepiej przyjrzyjmy się procesowi uruchamiania terminala: podczas startu dyskietka (lub sieciówka) wczytuje kod rozruchowy który każe maszynie rozpytać się o serwer DHCP. Serwer identyfikuje numer MAC karty sieciowej nadawcy i na podstawie wpisu w pliku `/etc/dhcpd.conf` przyznaje mu numer IP, z którym – poprzez wiersz w `/etc/hosts` – związana jest także nazwa sieciowa (to również ważne). W tym momencie stacja jest pełnoprawnym członkiem sieci, gotowym do pobrania

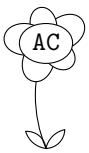
z serwera pliku jądra przez TFTP, które zaraz po przybyciu na terminal przejmuje stery w maszynie. Jeśli i to się powiodło, to w dalszej kolejności zostanie między innymi zamontowany udział dyskowy udostępniany z serwera przy pomocy NFS. Reszta należy do samego LTSP, które w razie problemów z grafiką możemy wywołać w trybie tekstowym. Aby uruchamiać aplikacje lokalnie potrzebne będzie jeszcze uruchomienie usługi NIS.

Instalacja i konfiguracja na serwerze graficznego menedżera logowania (np. KDM, GDM, XDM itp.) wieńczy dzieło: na ekranie terminala powinien się pojawić formularz wejścia do systemu. Odtąd zarejestrowani w systemie użytkownicy mogą korzystać ze swojego konta z dowolnej tak przygotowanej stacji roboczej.

Decyzja należy do Ciebie

O ile techniczne aspekty (zwłaszcza dostosowywanie plików konfiguracyjnych) mogą się zdawać złożone, to prawdziwym wyzwaniem ma szansę się okazać dopiero przekonanie użytkowników do zmiany dotychczasowych przyzwyczajeń. Poza typowymi problemami dotyczącymi przejścia na Linuksa mogą się zdarzyć także specyficzne kłopoty, jak na przykład błąd tkwiący swego czasu we wtyczce odtwarzacza animacji Flash, który ujawniał się tylko w wypadku uruchamiania przeglądarki na zdalnej maszynie, czyli między innymi właśnie na LTSP. Z drugiej strony centralnie zarządzany linuksowy serwer jest marzeniem niejednego administratora, zmniejsza bowiem ilość faktycznie zainstalowanego oprogramowania i związanego z tym nieodłącznie bałaganu.

Ciekawym zastosowaniem LTSP mogą być kafejki internetowe (poniżej odnośnik do zestawu odpowiednich narzędzi), ale równie dobrze może się sprawować w szkolnej pracowni lub biurze, o ile użytkownicy nie mają nadzwyczajnych wymagań co do wydajności. Zdecydowanie niewskazane jest natomiast wdrażanie LTSP tam, gdzie występują duże ilości szybko zmieniającej się na ekranie grafiki, a więc na przykład w grach 3D. Warto również pamiętać, że rozwiązanie takie nadaje tylko się do sieci lokalnej, ponieważ tylko ta jest zwykle w stanie zapewnić odpowiednią prędkość reakcji systemu, ale i ona może się przecież w końcu zapchać.



Projekt zasługuje na bliższe poznanie przez administratorów małych sieci i podsieci, a w przypadku posiadania pewnych ilości starego, ale nadal sprawnego sprzętu, warto zacząć od próby reanimacji w ramach LTSP zanim wyrzuci się go na złom. Bo ostatecznie to nic nie kosztuje, a znany slogan nie na darmo głosi: *Linux—because 486 is a horrible thing to waste!* ;-)

ODNOŚNIKI

- http://ltsp.org/ – główna strona LTSP
- http://www.k12ltsp.org/ – K-12, dystrybucja oparta na Red Hat Linux
- http://www.abuledu.org/ – AbulEdu, francuska dystrybucja oparta na Mandrake Linux
- http://marl.linuxfreunde.de/kmLinuxTSE – kmLinuxTSE, niemiecka dystrybucja oparta na kmLinux
- http://termserv.berlios.de/ – inna niemiecka dystrybucja
- http://termserv.berlios.de/ltsp-module/ – moduł Webmina do konfiguracji LTSP w przeglądarkach WWW
- http://akinimod.sourceforge.net/icafe.html – zestaw narzędzi do obsługi kafejki internetowej na LTSP
- http://www.rom-o-matic.net/ – generator dyskietek startowych dla stacji roboczych (dla LTSP tylko seria stabilna!)

Moja Przygoda z Pythonem czyli jak coś zrobić łatwo, szybko i przyjemnie

Adam Przybyła <adam@ertel.com.pl>

STRESZCZENIE: Python jest językiem, który bardzo dobrze nadaje się do rozwiązywania wielu prostych zadań, o czym przekonałem się osobiście w swojej pracy zawodowej.

Te standardowe zadania obejmują szeroki wachlarz programów, poczynając od skryptów generujących strony WWW, poprzez używanie WebServices z zastosowaniem XML-RPC i SOAP, pracę z wyszukiwarką napisaną w Pythonie – Ultraseek, obliczenia numeryczne i małe symulacje, sterowanie urządzeniami elektronicznymi, a na współpracy z ruterami CISCO kończąc.

Oprócz gotowych rozwiązań programistycznych warto zwrócić uwagę na powiązania, jakie Python ma z językami funkcyjnymi, a także na jego aktualne tendencje rozwojowe i to, jak można te elementy będzie wykorzystać w swojej pracy.

Każda wielka podróż zaczyna się od jednego małego kroczku
Chińskie przysłowie

Przyznam, że na codzień Python nie jest podstawą moich działań, używam go do czynności pomocniczych, ale nie wyobrażam sobie tego co robię bez tego przyjemnego i łatwego języka. Przedstawienie tu Pythona traktuję jako możliwość zainteresowania szerszego grona ludzi rozwiązaniem, które może nie są tak konkretne jak postawienie serwera czy instalacja Samby, ale o ile ktoś włoży w to troche wysiłku, to uzyska całkiem interesujące rezultaty.

Sam moją styczność, „przygodę" z Pythonem zacząłem od pracy przy wyszukiwarce internetowej Ultraseek. Ktoś poszukiwał osoby znającej się na języku Python i spotkał się ze mną, świeżo upieczonym adeptem tej tajemnej sztuki. I tu spotkało mnie pierwsze zaskoczenie, po tygodniu czytania dokumentacji byłem osobą biegle poruszającą się w temacie. To było tak łatwe i przyjemne,

a zajęło mi tak mało czasu. Znajomość tego języka przydała mi się wiele razy, jak na nakłady włożone w naukę, było to bardzo opłacalne.

To, co stanowi o potędze tego języka i jego przydatności, to potężna biblioteka funkcji i zaimplementowanych narzędzi. W Polsce, do generowania dynamicznych stron WWW stosuje się powszechnie PHP, a półki w księgarniach uginają się od wielu książek na ten temat. Mało kto wie, że do często stosowanego Apache istnieje moduł `mod_python` pozwalający na stosowanie w tym celu Pythona. Po prostej instalacji uzyskujemy pełne możliwości jakie daje nam Python i nie musimy wcale uczyć się nowego języka. Prosta konfiguracja `mod_python` z Apache:

```
<Directory /var/www/python>
        AddHandler python-program .py
        PythonHandler myscript.py
</Directory>
```

Umieszczamy tę sekwencję w konfiguracji Apache, przeładowujemy serwer i wszystko działa.

Zasada korzystania z `mod_python` jest jednak troche inna niż w przypadku PHP. Moduł posiada kilka trybów pracy na samym początku wystarczy jednak tylko ten, który podałem. Później warto zainteresować się trybem `publisher` lub `zpublisher`. Dla podanego przykładu skrypt `myscript.py` obsługuje wszystkie wywołania w danym katalogu. Zajmuje się także obróbką parametrów.

Jeśli zależy nam na zasymulowaniu zachowania PHP, skrypt mógłby wyglądać następująco:

```
from mod_python import apache
import sys,os,string

def handler(req):
       req.content_type = "text/html"
       req.send_http_header()
       os.chdir("/var/www/python")
       plik=os.path.basename(str(req.uri))
       if plik in os.listdir("."):
               sys.stdout=req
               execfile(plik)
       return apache.OK

def authenhandler(req):
        pw=req.get_basic_auth_pw()
```

```
        user=req.connection.user
        if pw == "secret" and user in ["juzer"]:
                return apache.OK
        else:
                return apache.HTTP_UNAUTHORIZED
```

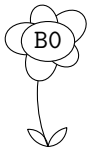
Procedura `handler` będzie wywoływana dla każdego URL i to ona wywołuje poszczególne pliki w naszym przykładzie. Procedura `authenhandler` zajmuje się autoryzacją naszego dostępu (wymagane uwzględnienie tego w konfiguracji Apache). Nic nie stoi na przeszkodzie, aby w wywołaniu funkcji `handler` zawrzeć odwołania do dowolnej biblioteki i przekształcić URL w to co chcemy.

SAX, XSL i DOM stoją do dyspozycji a są o niebo lepiej zrobione niż te dostępne w PHP. W naszym przypadku ograniczymy się do najprostszej wersji programu:

```
print """
<HTML>
 <HEAD>
  <META HTTP-EQUIV="Content-type"
        CONTENT="text/html; charset=iso-8859-2">
  <TITLE>Powitanie</TITLE>
 </HEAD>
 <BODY>
  Witaj Świecie!
 </BODY>
</HTML>
"""
```

Efektu wywołania nie muszę chyba wypisywać.

Aktualnie szybkość `mod_python`-a nie jest zbyt duża ale w miejscach, gdzie zależy nam na działaniu w języku, który znamy i który ma większe możliwości niż PHP, jest on całkiem sensowną alternatywą.



Python wykazuje dużą dynamikę jeśli chodzi o rozwój. Aktualnie podobne strony generowane są dwukrotnie wolniej w `mod_python` niż w PHP, ale w najbliższej perspektywie, gdzie są duże szanse na pojawienie się kompilatorow JIT, ten stosunek ulegnie na pewno zmianie. Przy wyborze rozwiązania warto kierować się także ilością dostępnych rozszerzeń. W tej konkurencji Python jest nie do pobicia.

Popularnym tematem przewijającym się na pewno przez wszelkiego rodzaju media są kwestie rozproszonych usług opartych o `.NET`. Czy jednak nie można zrobić tego trochę inaczej? WebServices są także dostępne w języku o którym

mówimy. Dla wielu rozwiązań to prostsza alternatywa w zupełności wystarczy. Jeśli kilka modułów służących do realizacji zdalnego wywoływania funkcji: XML-RPC i SOAP. Dla prostoty przedstawie tylko rozwiązanie oparte tylko o XML-RPC.

W XML-RPC rozróżniamy klienta i serwer usług. Serwer udostępnia określone procedury lub całe, kompletne obiekty. Parametry wywołania zostają przekazane poprzez sieć za pomocą przekształcenia wywołania w XML:

```
<methodCall>
       <methodName>nazwafunkcji</methodName>
       <params>
               <param><value><int>5</int></value></param>
               <param><value><int>3</int></value></param>
       </params>
</methodCall>
```

Odpowiedź serwera także jest w postaci XML. Zapewnia to zgodność między różnymi językami, które komunikują się poprzez ten protokół, a także różnymi architekturami. Zupełnie nie ma problemu z kompatybilnością. Oto prosty przykład serwera XML-RPC udostępniającego dwie funkcje:

```
#!/usr/local/bin/python
import sys
import SimpleXMLRPCServer

def test():
       return 1

def hello():
        return "Hello World"

server =SimpleXMLRPCServer.SimpleXMLRPCServer(("twoj_host.pl", 8000))
server.register_function(test)
server.register_function(hello)
server.serve_forever()
```

Klient, który będzie z tego korzystał, jest jeszcze prostszy:

```
#!/usr/local/bin/python
import xmlrpclib,string
s=xmlrpclib.Server("http://twoj_host.pl:8000")
```

```
print s.test()
print s.hello()
```

Jest to o tyle ciekawe, że po obydwu stronach można stosować dowolny język: PHP i Python, C i Python, co nam jest w danej chwili potrzebne i co jest dostępne. I wszystkie te języki dogadają się poprzez XML-RPC. Udostępniać można także całe obiekty, co widać na przykładzie tego serwera:

```
class matematyczne:
    def div(self, x, y) : return div(x,y)

server = SimpleXMLRPCServer("twoj_host.pl:8000")
server.register_instance(matematyczne())
server.serve_forever()
```

Istnieje wiele technologii o większych możliwościach niż XML-RPC: CORBA z obsługującym ją w Pythonie modułem ORBit , czy `.NET` z Python.NET, ale w większości przypadków zupełnie spokojnie wystarczy do realizacji projektu te proste rozwiązanie – XML-RPC, czy trochę bardziej skomplikowany SOAP. Oba są wspierane przez Pythona.

Nie odchodząc od tematów związanych z sieciami, warto wspomnieć o jednym z największych projektów jaki powstał w Pythonie, a jest nim na pewno wyszukiwarka Ultraseek. Całość ma co prawda zamknięte źródła, ale o ile znamy Pythona jesteśmy w stanie zmienić zachowanie całego programu. Zmianom mogą podlegać strony jakie wyświetla wbudowany w wyszukiwarkę serwer WWW, jak i same metody obróbki pobieranych i indeksowanych dokumentów. Dobre wsparcie dla XML jest tu bardzo (w tym drugim przypadku) pomocne. Standardowo serwer WWW rezyduje na porcie 8765, ale można go zmienić na dowolny inny. Podobnie jak w przypadku apache określamy katalog w którym znajdują się pliki które generują zawartość naszej strony. Pliki te przypominają standardowy HTML ale maja wstawki w Pythonie.

Nie jest to może tak przejrzyste jak PHP, ale daje się łatwo używać. Jest to jednak rozwiązanie specyficzne tylko dla tego programu.

```
<table width='100%'><tr><td><p>
<--$write(rand.choice(config.tips));-->
</td><tr></table>

&$nazwazmiennej_pythona;
```

```
<--$
imie="Adam"
-->
<b>Autor ma na imie &$imie;.</b>
```

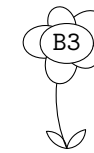
Powyższe przykłady nie wymagają chyba komentarza. Obiekt config, dostępny w skryptach, można odszukać w konfiguracji Ultraseeka, choć nigdzie nie jest on dokładnie udokumentowany.

W katalogu serwera WWW znajdują się skrypty `index.html` i `query.html` (rozszerzenia są mylące), które będa wykonywane przy obsłudze zapytania, od nas zależy jaki będzie wygląd generowanych stron. Stosunkowo niedawno wyszukiwarkę tę można było ściagnąć za darmo do testów, możliwe że aktualny właściciel, Yahoo, powróci do tej opcji, aktualnie nie jest to jednak, niestety, możliwe. Wyszukiwarka ta jest także dobrym przykładem na to, że Python nadaje się idealnie na język skryptów pozwalających na dopasowanie dużego programu do naszych indywidualnych potrzeb. Pokazuje, że na bazie OpenSource można budować duże rozwiązania komercyjne.

Na mniejszą skalę często stajemy wobec prozaicznych problemów związanych z połączeniem naszych urządzeń elektronicznych z komputerem. Wydawać by się mogło, że potrzeba do tego specjalnie pisanych sterowników działających na poziomie sprzętu, nic bardziej błędnego. Istnieją rozwiązania dla naprawdę drogiego sprzętu, gdzie sterowanie odbywa się poprzez język skryptowy – jest to tanie, łatwo wprowadzić zmiany i trudno się pomylić, a prędkość jaką uzyskujemy na porcie szeregowym w zupełności wystarcza.

Język Python wydaje się być idealny do tej funkcji. Co prawda, nie ma tych funkcji w standardzie, ale krótka sesja z Google skieruje nas na stronę `http://pyserial.sourceforge.net`, gdzie znajdziemy najlepszy aktualnie moduł do obsługi portu szeregowego. Instalacja nie powinna sprawić dużych problemów, zwłaszcza jeśli posiadamy Distutils:

```
# tar zxvf pyserial-1.17.tar.gz
# cd pyserial-1.17
# python setup.py build
# python setup.py install
```



Takie podejście ma wiele zalet, ale na pewno jedną z najważniejszych jest możliwość przenoszenia kodu między różnymi platformami. Ten sam kod będzie działał na Linuksie i na Windows, ma to duże znaczenie jeśli chcemy podłączać nasze urządzenie w różnych miejscach. Kilka przykładów użycia tego modułu, które będą działać na różnych platformach:

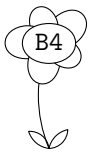
```
# Otwarcie portu 0 z parametrami 9600,8,N,1, "no timeout"
import serial
ser = serial.Serial(0)  #otwarcie 0 portu
print ser.portstr       #wypisanie jaki port został otwarty
ser.write("hello")      #wysyłamy ciąg znaków przez port
ser.close()             #zamykamy port

# Otwieramy port przez nazwe z parametrami 19200,8,N,1, "1s timeout"
ser = serial.Serial('/dev/ttyS1', 19200, timeout=1)
x = ser.read()          #czytamy byte
s = ser.read(10)        #czytamy do 10 bajtów (z timeout)
line = ser.readline()   #czytamy linie zakończoną "\n"
ser.close()
```

Jest to rozwiązanie bardzo przydatne dla wszelkich prac przy modemach, sterownikach podłączonych do portu szeregowego, czy komunikacji między różnymi systemami gdzie zależy nam na przenośności kodu. Przydaje się to również w przypadku projektów, gdzie mamy mało czasu na realizację – ale to dotyczy wszystkich projektów w Pythonie, nie tylko z użyciem modułu pyserial. W Pythonie pisze się szybko i bezbłędnie. Duże znaczenie ma tu składnia wspierająca estetykę programów.

Często, gdy pracujemy przy urządzeniach/komputerach dedykowanych określonym funkcjom, potrzebujemy wykonywać pewne czynności w sposób okresowy, jesteśmy w stanie zalgorytmizować te czynności, ale niestety urządzenia te zwykle nie posiadają zbyt dużych możliwości ich oprogramowywania. Przykładem niech będzie ruter Cisco, dopracowany w wielu kwestiach, posiadający obsługę wielu protokołów, ale czegoś tak prozaicznego jak cron tam nie uświadczymy. Z pomocą może nam tu przyjść Python i telnetlib, dzięki którym mamy pełne możliwości, jakie daje nam rozwiązanie podobne do expecta i pełne możliwości, jakie daje nam język programowania Python.

Prosty przykład zastosowania:




```
import getpass
import sys
import telnetlib

HOST = "localhost"
user = raw_input("Enter your remote account: ")
password = getpass.getpass()
```

```
tn = telnetlib.Telnet(HOST)

tn.read_until("login: ")
tn.write(user + "\n")
if password:
    tn.read_until("Password: ")
    tn.write(password + "\n")

tn.write("ls\n")
tn.write("exit\n")

print tn.read_all()
```

Prawdziwe zalety ukazują się jednak dopiero przy pracy ze wspomnianymi na początku rozwiązaniami zamkniętymi:

```
#! /usr/local/bin/python
import os
import telnetlib

cisco=telnetlib.Telnet("c5000-cisco",23)
cisco.write("juzer\r\n")
cisco.read_until("Password:")
cisco.write("haslo1\r\n")
cisco.read_until("c5000>")
cisco.write("en\r\n")
cisco.read_until("Password:")
cisco.write("haslonaenable\r\n")
cisco.read_until("c5000#")
cisco.write("ter l 0\r\n")
cisco.read_until("c5000#")
cisco.write("sh u\r\n")
users=cisco.read_until("c5000#")
cisco.write("q\r\n")
print users
```

Pobrane z rutera informacje możemy zupełnie spokojnie zapisać do bazy SQL czy zrobić z nimi wiele jeszcze innych rzeczy – wszystko zależy od naszej inwencji, nie jesteśmy już ograniczeni możliwościami rutera.

Aby zakończyć ten przegląd różnych zastosowań Pythona warto tu wspomnieć jeszcze o pakiecie próbującym konkurować z MATLABem. SciPy udostępnia szeroką gamę procedur i ma duże szanse pomóc niektórym w wiekszości prostych problemów, nie wymagając od dużych nakładów finansowych. Wiele dużych projektów próbuje wykorzystać zalety darmowości tego rozwiązania, warto więc zastanowić się nad tym i w swoim małym projekcie.

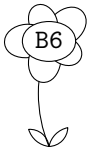
Python jest określany często jako język bardzo wysokiego poziomu, do wielu problemów pozwala podejść bez używania specjalizowanych programów, starałem się to wykazać w przedstawionych przykładach. Pełne spektrum możliwości jest jest jednak większe, wystarczy tylko użyć wyszukiwarki, by znaleźć namiary na interesujący nas moduł, czy poświęcić trochę czasu i zrobić go samemu.

Open Source to nie tylko duże projekty i pełne rozwiązania, warto także zainteresować się nim także w miejscach, gdzie tylko uzupełniają nasz cały projekt. To są małe kroki, ale ważne w przypadku lansowania Open Source.

Python ma jednak pewne możliwości, o których można zapomnieć skupiając się tylko na kwestii wiązania różnych protokołów i modułów. Sam w sobie jest ciekawym językiem, w którym z jednej strony występuje możliwość programowania obiektowego, a z drugiej strony funkcyjnego. Jak na tak mały język jest to zdumiewające (kto próbował zgłębiąć tajemną wiedze CommonLispa, ten to rozumie). O ile podejście obiektowe jest dosyć dobrze rozpropagowane i zrozumiałe dla większości, o tyle konstrukcje funkcyjne są używane z rzadka.

Prosty przykład:

```
>>> lista2=[1,2,3,4,5]
>>> for i in range(len(lista2)):
.  lista2[i]=lista2[i]*lista2[i]
.
>>> lista2
[1, 4, 9, 16, 25]
>>> lista2=[1,2,3,4,5]
>>> map(lambda i: i*i, lista2)
[1, 4, 9, 16, 25]
>>>
```



Jak widać drugie rozwiązanie jest bardziej przejrzyste, deklarujemy tylko co chcemy aby było robione a nie zajmujemy się każdym krokiem algorytmu. Funkcja map wykonuje zdefiniowaną przez nas funkcję nienazwaną dla wszystkich elementów listy. W tym przypadku funkcją tą jest podnoszenie do kwadratu.

Podobnie jak zaznaczenie że dana funkcja ma się wykonywać na całej liście możemy zaznaczyć, że określone elementy mają w tej liście zostać, zbudowanie takiego filtra jest bardzo proste:

```
>>> def zero(i):
.  return i==0
.
>>> def mniejnizzero(i):
.  return i<0
.
>>> lista=[1,-7,2,3,-1,3,0,0]
>>> f=zero
>>> filter(f,lista)
[0, 0]
>>> f=mniejnizzero
>>> filter(f,lista)
[-7, -1]
>>>
```

W tym przypadku nie wykorzystano już konstrukcji z lambda ale można to zrobić bez problemu. Te same konstrukcje zapisane imperatywnie byłyby o wiele bardziej skomplikowane. Na zakończenie troche inaczej zapisane poprzednie przykłady, może taka składnia łatwiej się przyjmie:

```
>>> [i*i for i in [1, 4, 9, 16, 25]]
[1, 16, 81, 256, 625]
>>> [i*i for i in [1, 2, 3, 4, 5]]
[1, 4, 9, 16, 25]
>>> [x for x in [1,-7,2,3,-1,3,0,0] if x==0]
[0, 0]
>>> [x for x in [1,-7,2,3,-1,3,0,0] if x<0]
[-7, -1]
>>># Zagadka a to co robi? ;-)
>>>n=5
>>>reduce(lambda i,k: i*k,range(1,n))
24
```

Mam nadzieję, że po tych wszystkich informacjach będzie można docenić Pythona jako syperklej łaczący różne elementy ale także wszyscy będa w nim widzieć całkiem ciekawy język do tworzenia własnych programów.

Wstęp do IntraWeb

Chad Hower, Łukasz Rżanek

STRESZCZENIE: Artykuł ten ma ma na celu wprowadzenie w środowisko IntraWeb – pierwszego w pełni wizualnego, zintegrowanego narzędzia do tworzenia stron internetowych w ramach środowisk programistycznych firmy Borland. Pominięte zostaną szczegóły techniczne, omówione podczas prelekcji, na którą serdecznie zapraszam. No, może jednak *odrobinę* technicznie będzie...

INTRAWEB JEST NARZĘDZIEM do tworzenia aplikacji WWW, dostarczającym sprawnych rozwiązań do szybkiego tworzenia aplikacji internetowych, intranetowych i ekstranetowych, łatwych w utrzymaniu i konserwacji. Inne techniki tworzenia aplikacji WWW wymagają zwykle rozmaitych dodatkowych operacji czy narzędzi, jak np. śledzenia stanu, technik skryptowych CGI czy też skomplikowanych konfiguracji klienckich. W przeciwieństwie do tego IntraWeb pozwala na zbudowanie aplikacji za pomocą dostępnych w środowisku komponentów, a następnie na zainstalowanie jej na serwerze. Po wykonaniu tych dwóch nieskomplikowanych czynności, dowolny klient dysponujący przeglądarką wspierającą standard HTML 4 uzyskuje dostęp do aplikacji. W porównaniu z innymi narzędziami do tworzenia stron WWW noszącymi miano RAD (Rapid Application Development), IntraWeb zasługuje na to miano podwójnie. Cała informacja wprowadzona za pośrednictwem formularzy obsługiwana jest przez IntraWeb w sposób transparentny – nie są wymagane żadne komponenty-adaptery, żadne przetwarzanie danych wejściowych, wystarczający jest czysty kod w Delphi. Dzięki temu nie jest konieczne tworzenie dokumentów HTML ani skryptów JavaScript czy CGI. Ponadto, dla większej wygody użytkownika, stworzona aplikacja może być również uruchamiana w sposób lokalny, jako niezależny plik wykonywalny.

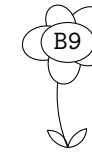
IntraWeb jest jak Kylix

Tworzenie aplikacji IntraWeb nie różni się zbytnio od tworzenia typowej aplikacji w środowisku Kylix. Większość użytkowników, nawet tych nie mających żadnego doświadczenia w tworzeniu aplikacji WWW, zdolnych jest stworzyć za pomocą IntraWeb prostą funkcjonującą aplikację WWW („weblikację") w ciągu niespełna pół godziny. Nie jest już konieczne spędzanie długiego czasu na studiowaniu szczegółów współpracy stron HTML z protokołem HTTP, ani też uczenie się JavaScript po to, by stworzyć dla klienta odpowiedni interfejs użytkownika. Wystarczą wyłącznie umiejętności związane z rodzimymi mechanizmami Delphi. Istnieją co prawda inne narzędzia do tworzenia aplikacji WWW, wykorzystujące Delphi jako języka programowania, lecz IntraWeb, jako Internetowy Kylix, ingeruje we wszystkie mechanizmy Kyliksa, z projektantem formularzy na czele.

HTML nie jest już niezbędny

Podobnie jak Delphi uwalnia programistę od nużących szczegółów API systemowego, IntraWeb izoluje go od funkcji Web API związanych z językiem HTML, JavaScript, CGI i HTTP. Dostarcza on mianowicie kontrolek skrywających w sobie funkcjonalność tych mechanizmów w taki sam sposób, jak np. kontrolki VCL (Visual Component Library – hierarchia klas przyspieszająca tworzenie aplikacji i zmniejszająca ilość koniecznego do napisania kodu) skrywają funkcjonalność Windows API. Umieszczając te kontrolki na formularzu można zbudować funkcjonalną weblikację, bez napisania chociażby jednej linii w HTML-u czy JavaScript, ani troszczenia się o szczegóły protokołu HTTP.

To, że IntraWeb jest narzędziem wysokiego poziomu nie oznacza bynajmniej, że jest on narzędziem mało elastycznym. Dostarcza on bowiem wielu opcji konfiguracyjnych i sterujących, nie odcina też programisty od rodzimego języka HTML – podobnie jak CLX (Component Library for Cross Platform – biblioteka komponentów firmy Borland, bazująca na VCL, pozwalająca na szybkie tworzenie przenośnych między Windows a Linuksem programów), nie odcina go od bezpośredniego operowania funkcjami API.



IntraWeb – dla weblikacji i sieci WWW

Wyobraźmy sobie przez chwilę, iż sieć WWW składa się z dużej liczby gwoździ (to strony WWW) oraz śrub (to aplikacje WWW – „weblikacje"). Z tego punktu widzenia typowe narzędzie do tworzenia aplikacji WWW kwalifikuje się jako młotek – idealny dla gwoździ, bezużyteczny dla śrubek, bądź jako śrubokręt –

z oczywistym odwróceniem przydatności. W tym ujęciu IntraWeb jest pierwszym narzędziem dostarczającym właściwych narzędzi, zarówno dla gwoździ, jak i śrub.

Czym jest weblikacja?

Weblikacja (Weblication) to zbitka słowna określająca aplikację WWW – Web Application. W chwili obecnej jedynym specjalizowanym narzędziem ukierunkowanym na tworzenie weblikacji jest IntraWeb. Ogólnie rzecz biorąc, funkcjonowanie sieci WWW opiera się na dwóch głównych fundamentach:

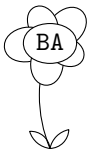
- Weblikacje – aplikacje pracujące w kontekście sieci WWW. W uproszczeniu weblikacja jest odmianą "zwykłej" aplikacji, istniejącej w postaci pliku wykonywalnego. Wykorzystuje jednak mechanizmy internetowe i komunikuje się z użytkownikiem za pomocą przeglądarek WWW.
- Strony WWW – interaktywne i dynamiczne strony internetowe, łączone ze stronami statycznymi, formujące cały system WWW lub jego część.

Weblikacje czy strony WWW?

Aplikacja internetowa przyjmuje niekiedy formę pośrednią pomiędzy przedstawionymi wyżej kategoriami, w większości przypadków można ją jednak zaklasyfikować do jednej z nich. Poniższe zestawienie najważniejszych cech owych kategorii z pewnością ułatwi taką klasyfikację:

IntraWeb przeznaczony jest głównie do tworzenia i wdrażania weblikacji, można go jednak także użyć w celu rozszerzenia zawartości stron WWW o elementy interaktywne – służy do tego tryb strony (page mode).

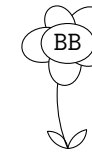
Tryb aplikacyjny



Tryb aplikacyjny (application mode) jest podstawowym trybem zastosowania IntraWeb, w tym bowiem trybie dokonuje się tworzenia weblikacji. Tryb ten dostarcza pełne środowisko RAD, pozwalające tworzyć weblikacje równie prosto, jak tworzy się typowe aplikacje w Kyliksie. Nie jest przy tym wymagane żadne doświadczenie w tworzeniu aplikacji internetowych.

Magia trybu aplikacyjnego polega na pewnym ograniczeniu elastyczności; nie oznacza to bynajmniej, iż praca w trybie aplikacyjnym jest mało elastyczna, jak można by zrazu domniemywać. Proces budowania aplikacji jest w dużym stopniu konfigurowalny, dzięki możliwości emitowania "surowych"

Weblikacja	Strona WWW
Przeznaczona dla intranetu, ekstranetu lub określonej grupy społeczności internetowej.	Zazwyczaj przeznaczona na publiczny użytek, dla anonimowego internauty, chociaż stosowane także w intranetach i ekstranetach.
Większość stron jest w wysokim stopniu interaktywna. Strony statyczne, bez interakcji, to zazwyczaj raporty generowane na podstawie danych użytkownika.	Wiele stron statycznych lub stron z niewielką ilością elementów dynamicznych.
Większość stron jest wzajemnie ze sobą powiązanych i wzajemnie zależnych od przekazywanych danych.	Strony są w większości samodzielne lub uzależnione jedynie wymienianymi danymi.
Utrzymywanie złożonej informacji o stanie, często nawet tak złożonej, że praktycznie nie dającej się zapisywać do strumienia.	Utrzymywanie co najwyżej minimalnej informacji o stanie.
Użytkownik rozpoczyna pracę z aplikacją i kontynuuje ją przez pewien czas.	Użytkownik może przemieszczać się pomiędzy różnymi stronami WWW, zatrzymując się na każdej z nich dowolnie długo.
Użytkownik świadomie uruchamia konkretną weblikację dla wykonania określonego zadania. Ze względów bezpieczeństwa użytkownicy poddawani są procedurze logowania przy dostępie do informacji chronionej.	Użytkownik ogląda konkretną stronę, pochodzącą z konkretnej sieci WWW; strona ta może być (choć nie musi) wyposażona w elementy dynamiczne lub możliwość wprowadzania danych.

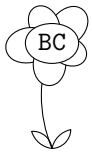


Rys 1 Weblikacja a strona WWW

Weblikacja	Strona WWW
Kompletna aplikacja obsługi wysyłki, umożliwiająca kontrolowanie kosztów spedycji, obsługę kont, zarządzanie środkami transportu, drukowanie zamówień i etykiet itp	Strona zawierająca wyszczególnienie wysłanych pakietów wraz z ich numerami przewozowymi
Złożony system śledzenia błędów do zarządzania aplikacjami przez programistów	Strona określająca status konkretnego błędu, lub związany z nim raport dla użytkowników
Rozproszony system finansowo-księgowy	Strona umożliwiająca przeliczanie walut
Kompletny system rejestrowania i realizacji zamówień dla działu sprzedaży	
Systemy bankowości online	

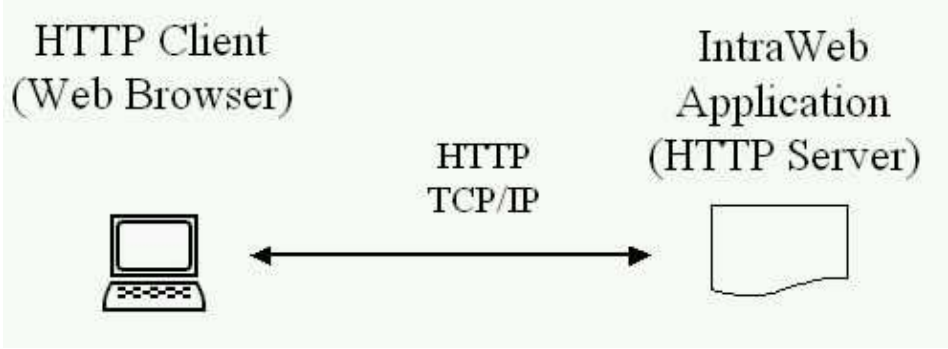
Rys 2 Przykłady

stron HTML, wykorzystania arkuszy stylów, zastosowania szablonów, a nawet tworzenia własnych komponentów do wielokrotnego wykorzystania. Szablony umożliwiają programistom zestandaryzowane tworzenie aplikacji, zaś artyści od Internetu mogą dzięki nim kontrolować wygląd i funkcjonowanie interfejsu użytkownika. Szablony oddzielają warstwę prezentacji od warstwy implementacyjnej – i dostarczają jeszcze wielu innych możliwości konfiguracyjnych. Tryb aplikacyjny zbudowany został na bazie trybu strony. Uwalnia on użytkownika od wielu skomplikowanych szczegółów związanych z tworzeniem aplikacji internetowych.



W trybie tym nasza aplikacja, zwykły plik wykonywalny, uruchamia się w ramach serwera WWW IntraWeb, specjalnie przystosowanego do obsługi tego typu programów. Wszystkie zapytania będą obsługiwane bezpośrednio przez naszą aplikację, z pominięciem jakichkolwiek innych warstw (patrz rysunek 3).

Tryb aplikacyjny każdego klienta (nie przeglądarkę!) obsługuje w osobnym wątku i dla każdego przydziela osobne zarządzanie pamięcią. Dzięki temu możliwe staje się wykorzystanie takich zaawansowanych funkcji serwera, jak automatyczne zarządzanie sesjami użytkowników wraz z tworzeniem dla nich



Rys 3 Diagram ilustrujący sposób pracy trybu aplikacyjnego

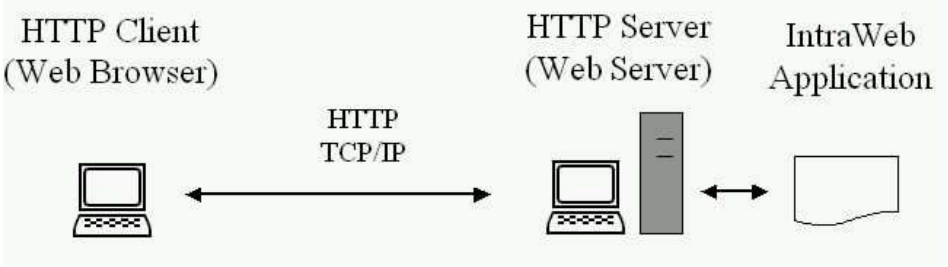
dowolnych zmiennych globalnych i lokalnych, oraz referncyjna pamięć podręczna danych i stron. Niesie to jednak za sobą ograniczenia na ilość obsługiwanych współbieżnie klientów, gdyż wymagania takiego serwera są bardzo duże.

Tryb webserwera (animowany aplikacyjny)

Tryb ten w zasadzie nie różni się od standardowego trybu aplikacyjnego w samej swej idei. Nadal mamy wszystkie jego zalety, jak możliwość wykorzystania szablonów czy arkuszy stylów, budowy aplikacji w oparciu o wizualne komponenty, etc. Jednak sama aplikacja nie jest już standardowym plikiem ELF z zaszytym w sobie serwerem WWW. Tym razem będzie to biblioteka współdzielona DSO (Apache Dynamic Shared Object), uruchamiana w ramach standardowego serwera Apache.

W trybie tym wszystkie odpowiedzi na zapytania klienta opracowywane są w odpowiedzi na jawne wywołanie poprzez działający serwer WWW, a dokładniej poprzez wykonanie referencji przekazanej jako URI (Uniform Resource Identifier) przez przeglądarkę obsługującą zapytanie. Następnym krokiem po otrzymaniu takowego zapytania będzie przekazanie podstawowych informacji dotyczących wywołania do aplikacji IntraWeb, która na tej podstawie generuje w ramach swojego wątku odpowiedź i przekazuje ją ponownie do serwera w celu ew. dalszej obróbki i, ostatecznie, wysłania do przeglądarki klienta. Ilustruje to diagram 4

W trybie tym każda sesja klienta będzie obsługiwana przez pojedyńczy proces serwera WWW zawierającego w sobie aplikację IntraWeb. Dzięki temu nadal możemy korzystać z automatycznego zarządzania sesją oraz pamięci podręcznej danych tak, jak to miało miejsce w trybie aplikacyjnym. Wątki takie



Rys 4 Diagram ilustrujący sposób pracy trybu aplikacyjnego obsługiwanego przez serwer WWW

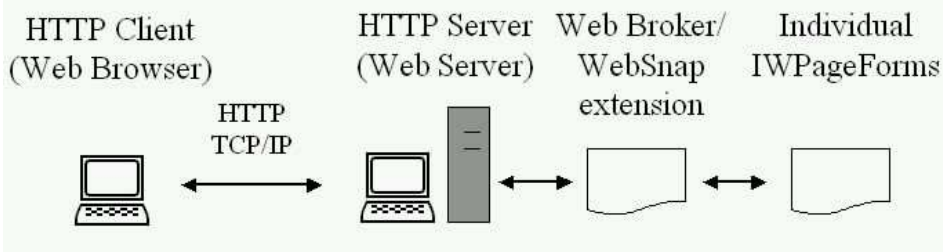
mają jednak swoją określona żywotność, która określa czas przez jaki klient będzie rozpoznawalny przez swój wątek serwera.

Tryb strony

IntraWeb zawiera także tryb strony (page mode), przeznaczony dla tworzenia pojedynczych, interaktywnych stron WWW na potrzeby większej sieci. Tryb ten nie jest jednakże idealnie przystosowany do wszelkich aspektów związanych z zarządzaniem sieciami WWW – niektóre języki w rodzaju ASP czy PHP nadają się do tego lepiej, mogą być jednak używane w połączeniu z trybem strony IntraWeb. Tryb strony nadaje się za to idealnie do tworzenia wysoce interaktywnych i skomplikowanych raportów, formularzy wprowadzania danych, wykresów, diagramów itp. Jeżeli dana strona WWW posiada wysoki stopień interakcji i/lub dynamiki, jeżeli jest to zasadniczy przejaw jej funkcjonowania i jeżeli nie jest ona częścią weblikacji, tryb strony jest idealnym narzędziem do jej tworzenia i edycji. Tryb strony jest bardziej elastyczny w użyciu od trybu aplikacyjnego, lecz wymaga od programisty nieco większego wysiłku.

IntraWeb pracujący w trybie strony nie jest już podstawowym narzędziem do obróbki stron WWW, lecz wartościowym uzupełnieniem całego warsztatu – umożliwia on bowiem pełną integrację z WebBrokerem, WebSnapem i innymi narzędziami tego typu. Programiści tworzący dotychczas aplikacje z wykorzystaniem w/w technologii mogą z pożytkiem wykorzystać nabyte umiejętności, zyskując dodatkowo kolejne użyteczne narzędzie.

W trybie tym wszystkie zapytania ponownie kierowane są do standardowego serwera WWW, np. Apache. Zapytanie takie, poprzez referencję URI, kierowane jest do aplikacji typu CGI czy DSO (stworzonej w ramach dowolnej technologii dostępnej w środowisku Kylix), która rozdziela odpowiednie zadania, przekierowując je do odpowiednich modułów. Jeśli moduł taki został stworzony



Rys 5 Diagram ilustrujący sposób pracy trybu strony

w IntraWeb, zostanie on zainicjowany i uruchomiony w celu przetworzenia żądania klienta i wygenerowania odpowiedzi zgodnej z jego życzeniami. Odpowiedz następnie zostanie przetworzona przez mechanizm aplikacji i przekazana do serwera WWW celem przesłania do klienta.

W tym trybie pracy IntraWeb traci większość swoich magicznych własności. Ponieważ każde zadanie obsługiwane jest przez osobno inicjowany do tego celu moduł, tracimy automatyczne zarządzanie sesjami użytkowników oraz podręczną pamięć danych.

Dlaczego dwa różne tryby pracy?

IntraWeb oferuje dwa różne tryby pracy w celu zaspokojenia dwóch różnych typów potrzeb programistycznych. Tryb aplikacyjny dedykowany jest zasadniczo do tworzenia i wdrażaniu weblikacji, głównym przeznaczeniem trybu strony jest natomiast obróbka stron WWW, zazwyczaj w celu nadania im większej dynamiki lub wyposażenia w funkcje interaktywne. Dotychczasowe narzędzia do programowania internetowego organicznie łączą w sobie obydwie te funkcje, przez co budowanie weblikacji za ich pomocą jest trudniejsze nią przy użyciu IntraWeb.

Tryb aplikacyjny raz jeszcze – zalety

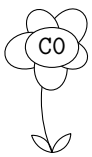
Należy tutaj jeszcze raz przyjrzeć się przyjętemu przez IntraWeb modelowi pracy, gdyż chcąc stworzyć w pełni elastyczne, stanowe środowisko pracy aplikacji, poczyniono kilka podstawowych założeń co do architektury generowanej ostatecznie aplikacji.

Po pierwsze... założono, że każde żądanie klienta powinno być identyfikowalne i śledzone. Aplikacje internetowe z zasady nie są stanowe, tzn. nie

trzymają informacji o swoim stanie w sposób naturalny. W tym przypadku to na programistę spada śledzenie poczynań użytkownika (np. poprzez cookies) oraz reagowanie na jego poczynania. Jest to pracochłonne, kłopotliwe, a przede wszystkim trudne, przez co jest to częsta lokalizacja błędów istotnych dla działania aplikacji. Utrudnia to ponadto tworzenie i utrzymywanie właściwych dla danego klienta zbiorów danych czy zmiennych, ponieważ wszystkie takie obiekty muszą być zapisane albo na jego komputerze, albo na serwerze, przy utrzymaniu referencji do sesji użytkownika. W większości przypadków twórcy takich programów rezygnują po prostu z pełnego zarządzania stanem aplikacji dla danego klienta na rzecz udawanej stanowości, tzn. przechowywania danych szczątkowych.

Warto tutaj zaznaczyć, że jako klienta ja osobiście rozumiem nie pojedyńczy komputer w sieci, ale jedną sesje przeglądarki internetowej. Jest to dość brzemienna w skutki różnica – wszystkie ciasteczka na komputerze przechowywane są jako obiekty globalne dla wszystkich sesji przeglądarki. Zatem zadanie stworzenia w pełni stanowej aplikacji staje się jeszcze większym wyzwaniem.

IntraWeb działa tutaj w dość innowacyjny sposób. Skorzystano z podstawowej cechy wszystkich serwerów WWW, tzw. animowanej wielowątkowości. Otóż praktycznie każdy serwer WWW dla osobnych klientów (w rozumieniu sesji przeglądarki) generuje osobny wątek swojego głównego procesu. W wątku takim zawierają się wszystkie informacje dotyczące samego serwera, ale również wszystkie aplikacje zbudowane jako biblioteki współdzielone. Dzięki temu, a także dzięki chronionym trybie obsługi pamięci, możliwym stało się identyfikowanie pojedynczej sesji klienta oraz przywiązanie do niej odpowiedniego wątka serwera wraz ze wszystkimi zmiennymi aplikacji IntraWeb uruchomionej w jego podwątku. żegnajcie ciasteczka – nigdy więcej nie trzeba będzie się martwić prostymi obiektami danych niezbędnymi na czas działania aplikacji. Teraz cookies mogą służyć tylko w jednym celu – do długoterminowego przechowywania identyfikatora klienta.



PO DRUGIE... założono, że każdy programista znający środowiska firmy Borland powinien być w stanie stworzyć zaawansowaną aplikację internetową w ramach IntraWeb. Założenie to jest o tyle niebezpieczne, że nie każdy przeciętny użytkownik Delphi, C++ Buildera czy Kyliksa musi znać specyfikę tworzenia aplikacji internetowych oraz składnię i kruczki HTML.

IntraWeb został zatem oparty o standardowy look and feel tychże środowisk. Mamy zatem identyczne formatki, identyczne (a raczej podobne) komponenty, zdarzenia, etc. Jedyna różnica polega na tym, że po skompilowaniu całości to wszystko zamieni się w... czysty w swej postaci HTML. Twórcy IntraWeb poszli krok dalej – dołożyli podobne w swym działaniu komponenty obsługi danych

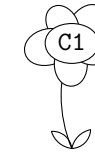
oraz ich prezentacji do tych, jakie są używane w ramach bibliotek CLX czy VCL. Te podobieństwa są tak duże, że gdyby nie specyficzne nazewnictwo, to wielu nie zauważyłoby być może różnicy pomiędzy aplikacjami IntraWeb, a aplikacjami okienkowymi GUI.

PO TRZECIE... założono, że nie każdy potrzebuje tworzyć aplikacje Intranetowe, gdzie prędkość działania czy generowane obciążanie serwera nie mają znaczenia, i pozostawiono wiele furtek i możliwości. Dzięki temu każdy z omawianych trybów pracy można zaadoptować tak, aby był możliwie bardzo dostosowany do naszych potrzeb. Nie dość tego – założono, że nie każdy programista będzie korzystał z języka Delphi czy C++, w których to powstawały pierwsze wersje IntraWeb. Stąd też powstały kolejne wersje tego środowiska, na platformy Java i .Net, dające pełną wolność wyboru.

JAKI TRYB WYKORZYSTUJĄ INNE NARZĘDZIA?

Nie jest łatwo scharakteryzować pracę innych narzędzi internetowych w kategoriach dwóch trybów pracy IntraWeb; jeżeli jednak przyjrzeć się bliżej funkcjonowaniu tych narzędzi, to jest ono zdecydowanie bliższe trybowi strony – jego istotą jest przecież „obróbka" pojedynczych stron. Nie oznacza to oczywiście, iż narzędzia te nie nadają się do tworzenia weblikacji, jednak tworzenie to przypominać może niekiedy próbę wkręcania śrub za pomocą młotka. Proces budowania aplikacji za pomocą tychże narzędzi polega głównie na opracowywaniu oddzielnych stron WWW, połączonych ze sobą i przekazujących sobie informacje za pomocą parametrów. Wiele narzędzi umożliwia także zarządzanie informacją o stanie serwera, co nie zmienia faktu iż same strony WWW są w dalszym ciągu bezstanowe i muszą pobierać tą informację ze specjalnego obiektu serwera. Niektóre z narzędzi pozwalają także na wbudowywanie informacji o stanie bezpośrednio w strony HTML, zwiększa to jednak obciążenie sieci i stwarza potencjalne zagrożenia dla bezpieczeństwa.

Aby zdać sobie sprawę z tego, w jaki sposób istniejące narzędzia podchodzą do problemu konstrukcji aplikacji WWW, wyobraźmy sobie aplikację złożoną z 10 różnych formularzy, z których każdy znajduje się w oddzielnym pliku wykonywalnym. Aby uzyskać dostęp do danego formularza, należy uruchomić właściwy plik, przekazując mu niezbędną informację na przykład za pomocą parametrów lub pliku tekstowego. Jeżeli nawet wygląda to kuriozalnie, to z pewnością trafnie oddaje istotę filozofii wspomnianych narzędzi – w ten właśnie sposób funkcjonuje większość obecnych aplikacji internetowych. Tryb aplikacyjny IntraWeb jest raczej słabo przystosowany do ich obróbki: weblikacje, na potrzeby których został stworzony, są bowiem czymś zupełnie innym.



Ideą IntraWeb w tym wypadku jest odejście od takiego trybu pracy, w którym wszelkie zmienne muszą być przekazywane przez mechanizm zewnętrzny czy też wskaźnik do określonego obiektu po stronie serwera czy klienta. W tym wypadku zamiast stosować tak skomplikowane techniki odwołujemy się do... lokalnych zmiennych. Wystarczy zadeklarować zmienną dowolnego typu globalnie dla całej weblikacji, aby mieć dostęp do niej z każdego miejsca pisanego kodu. To naprawdę proste!

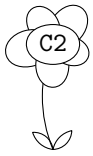
Wyjaśnienia skrótów i terminów

VCL – Visual Component Library

To biblioteka komponentów, popularna w systemie Windows, we wszystkich produktach typu RAD firmy Borland na tą platformę. Biblioteki te mogą być tworzone zarówno w języku Delphi jak i C++, przy czym nie ma to znaczenia przy ostatecznym importowaniu do środowiska. Biblioteki VCL mogą zawierać dowolną ilość komponentów do wykorzystywania przy procesie tworzenia aplikacji. Komponenty takie to zamknięte zestawy modularnych bloków kodów źródłowych implementujących podstawowe, powtarzalne funkcje lub obiekty aplikacji, np. widgety komponentów graficznych lub obsługę protokołów sieciowych.

CLX – Component Library for Cross Platforms

To biblioteka komponentów zaprojektowana wraz ze środowiskiem Kylix. Podstawowa idea przyświecająca temu przedsięwzięciu to pełna przenośność między systemami. Biblioteka taka, podobnie jak VCL, może zawierać dowolną ilość komponentów do modularnego wykorzystania w ramach aplikacji, z tą różnicą, iż mogą one przy jednakowym interfejsie obsługiwać dwa lub więcej systemów operacyjnych.



DSO – Apache Dynamic Shared Object

Dynamic Shared Object to biblioteki klasy Shared Object (SO) lub Dynamic Link Library (DLL) ładowane przez serwer Apache jako integralna część serwera. Zwykle biblioteki te obsługują dodatkową funkcjonalność serwera (np. `mod_ssl` obsługuje szyfrowanie SSL), jednak istnieje także możliwość ich wykorzystywania jako źródło skryptowe stron internetowych. Zaletą takiego rozwiązania jest duży zysk wydajności i mniejsze obciążenie systemu.

KYLIX

Generalnie jest to pierwsze pełne środowisko RAD dla systemu operacyjnego Linux, które powstało na bazie środowisk Delphi oraz C++ Builder. Zaprojektowane jako pełne środowisko pracy dla programisty zawiera w sobie natywny, wydajny kompilator dla języka Delphi, a od wersji trzeciej także dla języka C++. Zaawansowane funkcje debuggowania aplikacji, code-insight (podpowiadanie składni) oraz wiele innych dodatków stworzone zostały aby skrócić i uprościć proces tworzenia aplikacji.

Więcej informacji:

- strona BSC Polska Sp. z o.o. (`http://www.borland.pl/kylix/`)
- Borland Software Corporation (`http://www.borland.com/kylix/`).