Rozdział 14

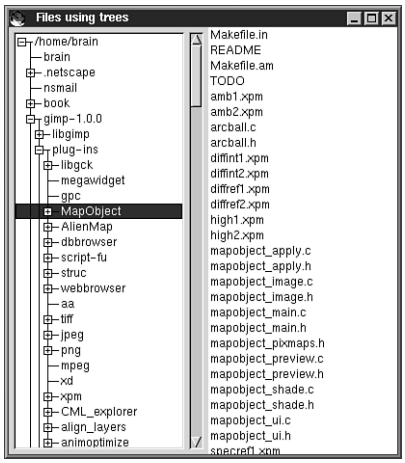
Drzewa, c-listy i zakładki

Drzewa (GtkTree), c-listy (GtkCList) i zakładki (GtkNotebook) są bardziej interesujące, niż poprzednio omawiane proste kontrolki. Ilustrują wielkie możliwości drzemiące w GTK+. Udostępniając obiektowy interfejs, znacznie ułatwiają tworzenie aplikacji. Omawiane w tym rozdziale trzy kontrolki są jednymi z najbardziej przydatnych, choć GTK+ oferuje znacznie więcej kontrolek.

Kontrolka drzewa

Kontrolka drzewa (GtkTree) wyświetla dane w postaci hierarchicznej; może to być drzewo katalogów albo drzewo genealogiczne. Drzewa umożliwiają oglądanie całej hierarchii elementów, ale pozwalają też na zwinięcie poszczególnych gałęzi drzewa.

Rysunek 14.1 przedstawia przykład kontrolki GtkTree umieszczonej obok pola listy, co umożliwia przeglądanie katalogów i plików. W drzewie, po lewej stronie, wyświetlane są tylko katalogi. Podkatalogi wszystkich umieszczonych w drzewie katalogów są wyświetlane jako liście drzewa i można je zwinąć. Pole listy wyświetla pliki, znajdujące się w zaznaczonym katalogu. Napiszemy ten program w trakcie zapoznawania się z kontrolką drzewa.



Rysunek 14.1. Kontrolka drzewa w działaniu.

Tworzenie drzewa

Do tworzenia kontrolki GtkTree służy funkcja gtk_tree_new. Utworzone drzewo jest początkowo puste i trzeba je wypełnić. Aby utworzyć liść drzewa, należy użyć funkcji gtk_tree_item_new_with_label i przekazać do niej nazwę liścia. Po utworzeniu liścia należy przypisać go do węzła macierzystego. Węzłem tym może być kontrolka GtkTree (dla węzła najwyższego poziomu) albo inny liść. Poniższy kod tworzy kontrolkę GtkTree i dodaje do niej węzeł najwyższego poziomu.

```
/* --- Tworzymy drzewo --- */
drzewo = gtk_tree_new();

/* --- Tworzymy liść --- */
lisc = gtk_tree_item_new_with_label ("Pierwszy liść");

/* --- Dodajemy liść do drzewa --- */
gtk_tree_append (GTK_TREE (drzewo), lisc);

/* --- uwidaczniamy drzewo i liść --- */
gtk_widget_show (lisc);
gtk_widget_show (drzewo);
```

Teraz możemy zacząć dodawać kolejne liście, ale dodawanie danych tworzących drzewo odbywa się w kilku krokach. Najpierw tworzymy nowe drzewo GtkTree przy pomocy funkcji gtk_tree_new. Następnie za-

znaczamy je jako poddrzewo i przypisujemy mu drzewo macierzyste, przy pomocy funkcji gtk_tree_item_set_subtree. Możemy następnie dodawać elementy do poddrzewa za pomocą tych samych funkcji, które dodają elementy do drzewa.

```
/* --- Tworzymy nowe drzewo --- */
poddrzewo = gtk_tree_new();

/* --- Liść będzie rodzicem poddrzewa --- */
gtk_tree_item_set_subtree (GTK_TREE_ITEM (lisc), poddrzewo);

/* --- Tworzymy nowy liść --- */
nowy_lisc = gtk_tree_item_new_with_label ("licœæ");

/* --- Dodajemy nowy liść do poddrzewa --- */
gtk_tree_append (GTK_TREE (poddrzewo), nowy_lisc);
```

Funkcja gtk_tree_append nie jest jedyną funkcją, która dodaje elementy do drzewa. Możemy wykorzystać także funkcję gtk_tree_prepend, która wstawia elementy na początku drzewa, oraz funkcję gtk_tree_insert, która wstawia elementy w określonym punkcie drzewa.

```
/* --- Dodajemy element na początku drzewa --- */
gtk_tree_prepend (GTK_TREE (drzewo), lisc);

/* --- Wstawiamy element na początek przy pomocy "insert" --- */
gtk_tree_insert (GTK_TREE (drzewo), lisc, 0);

/* --- Wstawiamy element na koniec przy pomocy "insert" --- */
gtk_tree_insert (GTK_TREE (drzewo), lisc, -1);

/* --- Wstawiamy element za drugim elementem --- */
gtk_tree_insert (GTK_TREE (drzewo), lisc, 2);
```

Aby usunąć element drzewa, możemy skorzystać z funkcji gtk_tree_remove_item. Kiedy element zostanie usunięty z drzewa, usuwane są także jego wszystkie liście.

```
/* --- Usuwamy liść z drzewa --- */
gtk_tree_remove_item (drzewo, lisc);
```

Aby oczyścić całe drzewo z liści, możemy użyć funkcji gtk_tree_clear_items. Elementy, które mają zostać usunięte, określamy przy pomocy początkowej i końcowej pozycji. Wszystkie liście potomne można usunąć za pomocą polecenia:

```
/* --- Usuwamy wszystkie liście drzewa --- */
```

```
gtk_tree_clear_items (drzewo, 0, -1);
```

Sygnały drzewa

Kontrolka GtkTree dysponuje pewną liczbą własnych sygnałów. Sygnały selection_changed, select_child i unselect_child są wysyłane do kontrolki drzewa w celu poinformowania o stanie zaznaczonych elementów. Elementy drzewa mogą otrzymywać sygnały collapse_tree i expand_tree.

Tworzenie przeglądarki plików

Wyposażeni w podane wyżej informacje możemy zacząć pisanie programu przeglądarki plików. Przeglądarka składa się z kontrolki GtkTree oraz pola listy, umieszczonych obok siebie wewnątrz okna aplikacji. Najpierw przedstawimy główną funkcję, która tworzy okno programu:

```
/*
 * main
 *
 * Program zaczyna się tutaj
 */
int main (int argc, char *argv[])
{
   GtkWidget *okno;

   /* --- Inicjacja GTK --- */
   gtk_init (&argc, &argv);

   /* --- Tworzymy okno najwyższego poziomu --- */
   okno = gtk_window_new (GTK_WINDOW_TOPLEVEL);

   /* --- Nadajemy mu tytuł --- */
   gtk_window_set_title (GTK_WINDOW (okno), "Pliki w drzewie");

   /* --- Ustawiamy rozmiar okna. --- */
   gtk_widget_set_usize (okno, 250, 250);

   /* --- Należy zawsze podłączyć sygnał delete_event
    *     do głównego okna.
    */
   gtk_signal_connect (GTK_OBJECT (okno), "delete_event",
                       GTK_SIGNAL_FUNC (delete_event), NULL);
```

```
    gtk_widget_show (okno);

    /* --- Tworzymy drzewo  --- */
    UtworzDrzewo (okno);

    gtk_main ();

    exit (0);
}
```

Funkcja UtworzDrzewo tworzy poziome pole pakujące, w którym umieścimy obok siebie kontrolkę drzewa i pole listy. Kontrolka drzewa będzie umieszczona w przewijanym oknie; dzięki temu, jeśli elementy drzewa zostaną rozwinięte i drzewo przekroczy rozmiar okna, użytkownik będzie mógł skorzystać z pasków przewijania i zobaczyć resztę drzewa. Funkcja getcwd zwraca bieżący katalog roboczy, który zostanie wyświetlony w korzeniu kontrolki drzewa. Element ten pokaże użytkownikowi pełną ścieżkę do katalogu, w którym będzie umieszczony program. Informacje o podkatalogach są przekazywane do funkcji UtworzPoddrzewo, aby wypełnić kontrolkę drzewa wszystkimi katalogami niższego rzędu.

```
/*
 * UtworzDrzewo
 *
 * Tworzy drzewo pokazujące strukturę plików.
 *
 * okno - okno macierzyste
 */
static void UtworzDrzewo (GtkWidget *okno)
{
    char bufor[MAKS_SCIEZKA];
    GtkWidget *pole1;
    GtkWidget *pole2;
    GtkWidget *okno_przew;
    GtkWidget *drzewo;
    GtkWidget *lisc;

    /* --- Poziome pole pakujące --- */
    pole1 = gtk_hbox_new (FALSE, 0);
    gtk_container_add (GTK_CONTAINER (okno), pole1);
    gtk_widget_show (pole1);

    /* --- Pole pakujące na drzewo --- */
    pole2 = gtk_vbox_new(FALSE, 0);
```

```
    gtk_box_pack_start(GTK_BOX(pole1), pole2, TRUE, TRUE, 0);
    gtk_container_border_width(GTK_CONTAINER(pole2), 5);
    gtk_widget_show(pole2);

    /* --- Tworzymy przewijane okno na drzewo --- */
    okno_przew = gtk_scrolled_window_new (NULL, NULL);
    gtk_scrolled_window_set_policy                (GTK_SCROLLED_WINDOW
(okno_przew),
                    GTK_POLICY_AUTOMATIC, GTK_POLICY_AUTOMATIC);
    gtk_box_pack_start (GTK_BOX (pole2), okno_przew, TRUE, TRUE, 0);
    gtk_widget_set_usize (okno_przew, 250, 250);
    gtk_widget_show (okno_przew);

    /*
     * --- Tworzymy korzeń drzewa
     */
    drzewo = gtk_tree_new();

    /*
     * --- Tworzymy pole listy
     */
    polelisty = gtk_list_new ();
    gtk_widget_set_usize (polelisty, 250, 250);
    gtk_box_pack_start (GTK_BOX (pole1), polelisty, TRUE, TRUE, 0);
    gtk_widget_show (polelisty);

    /* --- Dodajemy drzewo do przewijanego okna --- */
    gtk_scrolled_window_add_with_viewport (
                        GTK_SCROLLED_WINDOW(okno_przew),
                        drzewo);

    /* --- Uwidaczniamy drzewo. --- */
    gtk_widget_show (drzewo);

    /*
     * --- Tworzymy kontrolkę dla podstawowego elementu
     *     (bieżącego katalogu
     */
    lisc = gtk_tree_item_new_with_label (
                getcwd (bufor, sizeof (bufor)));

    /* --- Dodajemy element do drzewa --- */
    gtk_tree_append (GTK_TREE (drzewo), lisc);
```

```
    /* --- Uwidaczniamy element --- */
    gtk_widget_show (lisc);

    /* --- Tworzymy poddrzewo tego elementy --- */
    UtworzPoddrzewo (getcwd (bufor, sizeof (bufor)),
                getcwd (bufor, sizeof (bufor)), lisc);

    /* --- Uwidaczniamy okno --- */

    gtk_widget_show (okno);
}
```

Funkcja UtworzPoddrzewo pobiera katalogi, znajdujące się na ścieżce. Katalogi te stają się liśćmi węzła, który reprezentuje ścieżkę. Na przykład katalog ksiazka może mieć podkatalogi rozdz1, rozdz2 i rozdz3. W takim przypadku węzeł drzewa ksiazka będzie miał trzy węzły potomne— rozdz1, rozdz2 i rozdz3. Funkcja opendir pozwala na przejrzenie wszystkich plików i podkatalogów w dowolnym katalogu. Następnie wykorzystujemy funkcję Katalog, która sprawdza, czy dana pozycja katalogu jest zwykłym plikiem, czy też katalogiem. Jeśli element jest katalogiem, wówczas rekurencyjnie przekazujemy go do funkcji UtworzPoddrzewo, aby znaleźć jego węzły potomne. Każdy węzeł posiada funkcję zwrotną dla sygnału select, więc kiedy użytkownik kliknie na nazwie katalogu, możemy wyświetlić w polu listy wszystkie zawarte w katalogu pliki.

```
/*
 * UtworzPoddrzewo
 *
 * Dodaje katalogi do drzewa i wiąże je z procedurą
 * obsługi sygnału - kiedy element zostanie wybrany,
 * pole listy zostanie wypełnione plikami znajdującymi
 * się w katalogu.
 *
 * szSciezka - Ścieżka do dodawanych plików.
 */
static void UtworzPoddrzewo (char *szSciezka, char *szKatalog,
                    GtkWidget* element)
{
   DIR *katalog;
   struct dirent *wpisKatalogowy;
   GtkWidget* poddrzewo_elementu = NULL;
   GtkWidget* nowy_element;
   char bufor[MAKS_SCIEZKA];
```

```
/* --- Odczytujemy bieżący katalog --- */
katalog = opendir (szSciezka);

/* --- Odczytując zawartość katalogu...  --- */
while (wpisKatalogowy = readdir (katalog)) {

   /* --- ...nie bierzemy pod uwagę tych wpisów --- */
   if (!strcmp (wpisKatalogowy->d_name, "..") ||
      !strcmp (wpisKatalogowy->d_name, ".")) {

      /* --- Ignorujemy katalogi "." i ".." --- */
   } else {

      /* --- Tworzymy pełną ścieżkę --- */
      sprintf (bufor, "%s/%s", szSciezka,
            wpisKatalogowy->d_name);

      /* --- Jeśli jest to katalog --- */
      if (Katalog (bufor)) {

         if (poddrzewo_elementu == NULL) {

            /* --- Tworzymy nowe poddrzewo --- */
            poddrzewo_elementu = gtk_tree_new ();

            /* --- Dodajemy poddrzewo do drzewa --- */
            gtk_tree_item_set_subtree (GTK_TREE_ITEM (
                  element), poddrzewo_elementu);
         }

         /* --- Tworzymy nowy element dla pliku --- */
         nowy_element = gtk_tree_item_new_with_label (
                     wpisKatalogowy->d_name);

         /* --- Dodajemy element do drzewa --- */
         gtk_tree_append (GTK_TREE (poddrzewo_elementu),
               nowy_element);

         /* --- Dodajemy wszystkie elementy podrzędne --- */
         UtworzPoddrzewo (bufor, wpisKatalogowy->d_name,
               nowy_element);

         /* --- Uwidaczniamy element --- */
         gtk_widget_show (nowy_element);
```

```
            /* --- Informujemy o wybraniu elementu --- */
            gtk_signal_connect (GTK_OBJECT (nowy_element),
                  "select",
                  GTK_SIGNAL_FUNC (wybrano_element),
                  g_strdup (bufor));

         }
      }
   }
   /* --- Gotowe --- */
   closedir (katalog);

   gtk_widget_show (element);
}
```

Poniżej zamieszczamy funkcję Katalog, która przyjmuje nazwę elementu katalogu i używa funkcji stat, aby sprawdzić, czy element ten jest plikiem, czy katalogiem.

```
/*
 * Katalog
 *
 * Sprawdza, czy plik jest katalogiem, czy zwykłym plikiem.
 *
 * bufor - pełna ścieżka wraz z nazwą pliku.
 *
 * zwraca TRUE, jeśli plik jest katalogiem
 */
int Katalog (char *bufor)
{

   struct stat buf;
   if (stat (bufor, &buf) < 0) {

      /* --- Błąd - ignorujemy. --- */
      return (FALSE);
   }

   /* --- Sprawdzamy, czy plik jest katalogiem --- */
   return (S_ISDIR (buf.st_mode));
}
```

Funkcja wybrano_element jest wywoływana za każdym razem, kiedy użytkownik kliknie element kontrolki drzewa. Funkcja wyświetla krótki

komunikat (w celach ilustracyjnych) i wywołuje funkcję WypelnijPoleListy, która wyświetla listę plików w katalogu.

```
void wybrano_element (GtkWidget *kontrolka, gpointer dane)
{
   g_print ("wybrano element %s\n", (char *) dane);

   WypelnijPoleListy ((char *) dane);
}
```

Kod funkcji WypelnijPoleListy właściwie nie wymaga wyjaśnień. Otrzymawszy ścieżkę, funkcja przegląda wszystkie pliki/katalogi na ścieżce i wypełnia pole listy nazwami wszystkich plików, które znajdują się w katalogu.

```
/*
 * WypelnijPoleListy
 *
 * Dodaje pliki, znajdujące się w katalogu do pola
 * listy. Uwzględnia tylko zwykłe pliki.
 *
 * szSciezka - ścieżka do dodawanych plików.
 */
static void WypelnijPoleListy (char *szSciezka)
{
   DIR *katalog;
   struct dirent *wpisKatalogowy;
   char bufor[MAKS_SCIEZKA];

   /* --- Czyścimy pole listy. --- */
   gtk_list_clear_items (GTK_LIST (polelisty), 0,
            g_list_length (GTK_LIST (polelisty)->children));

   /* --- Odczytujemy bieżący katalog --- */
   katalog = opendir (szSciezka);

   /* --- Odczytując bieżący katalog...  --- */
   while (wpisKatalogowy = readdir (katalog)) {

      /* --- ...nie bierzemy pod uwagę tych wpisów --- */
      if (!strcmp (wpisKatalogowy->d_name, "..") ||
         !strcmp (wpisKatalogowy->d_name, ".")) {

         /* --- Ignorujemy katalogi "." i ".." --- */
      } else {
```

```
            /* --- Tworzymy pełną ścieżkę --- */
            sprintf (bufor, "%s/%s", szSciezka,
                  wpisKatalogowy->d_name);

            /* --- Jeśli to nie jest katalog --- */
            if (!Katalog (bufor)) {

               /* --- Dodajemy plik do pola listy --- */
               DodajElementListy (polelisty, wpisKatalogowy->d_name);
            }
         }
      }

      /* --- Gotowe --- */
      closedir (katalog);
   }
```

Funkcja DodajElementListy jest po prostu skrótem, który dodaje łańcuch do pola listy.

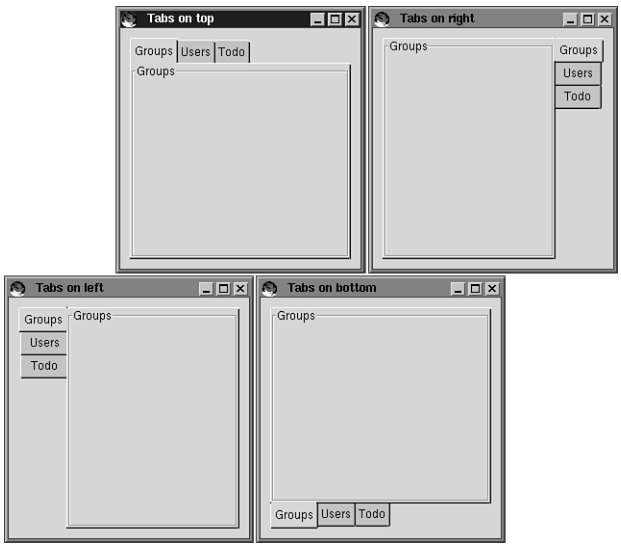
```
   /*
    * DodajElementListy
    *
    * Dodaje element do pola listy.
    *
    * polelisty - lista, do której należy dodać element.
    * sTekst - tekst, który należy wyświetlić w polu listy.
    */
   void DodajElementListy (GtkWidget *polelisty, char *sTekst)
   {
      GtkWidget *element;

      /* --- Tworzymy element listy na podstawie danych --- */
      element = gtk_list_item_new_with_label (sTekst);

      /* --- Dodajemy element do pola listy --- */
      gtk_container_add (GTK_CONTAINER (polelisty), element);

      /* --- Uwidaczniamy element --- */
      gtk_widget_show (element);
   }
```

W rezultacie otrzymujemy aplikację przeglądarki plików, która używa bieżącego katalogu jako punktu startowego, odczytuje strukturę systemu

plików (nie sprawdzając żadnych katalogów położonych wyżej od bieżącego) i wyświetla drzewo systemu plików poniżej katalogu bieżącego. Kliknięcie na nazwie katalogu spowoduje wyświetlenie znajdujących się w nim plików. Poszczególne katalogi można rozwijać lub zwijać, klikając znak plusa albo minusa umieszczony obok nazwy katalogu.

Kontrolka notatnika

Kontrolka notatnika (GtkNotebook) umożliwia wyświetlanie informacji na kilku stronach. Każdą ze stron można przesunąć na wierzch, klikając na reprezentującej ją zakładce. Zakładki są opisane, dzięki czemu można zidentyfikować poszczególne strony. Zakładki mogą znajdować się na górze strony, na dole strony, albo na jej prawej lub lewej krawędzi.



Rysunek 14.2. Kontrolka notatnika.

Kontrolkę GtkNotebook tworzymy przy pomocy funkcji gtk_notebook_new. Zakładki dodajemy przy pomocy funkcji gtk_notebook_set_tab_pos, przekazując jedno z możliwych położeń (GTK_POS_TOP, GTK_POS_BOTTOM, GTK_POS_LEFT albi GTK_POS_RIGHT).

```
/* --- Tworzymy notatnik --- */
notatnik = gtk_notebook_new();

/* --- Umieszczamy zakładki na górze notatnika --- */
```

```
gtk_notebook_set_tab_pos (GTK_NOTEBOOK (notatnik), GTK_POS_TOP);
```

Dodawanie i usuwanie stron

Kontrolkę GtkNotebook można używać dopiero wtedy, kiedy dodamy do niej kilka stron. Czynimy to przy pomocy funkcji gtk_notebook_append_page, która dodaje nowe strony na końcu listy stron. Funkcja gtk_notebook_prepend_page dodaje strony na początku listy, a gtk_notebook_insert_page umożliwia wstawienie strony w dowolnym miejscu listy. Kontrolki przekazywane do tych funkcji muszą zostać utworzone przed próbą utworzenia strony. Zazwyczaj tworzy się etykietę dla zakładki oraz jakiś typ pojemnika, w którym będzie można umieścić inne kontrolki. Aby usunąć stronę, można użyć funkcji gtk_notebook_remove_page.

```
/* --- Tworzymy etykietę i pojemnik dla nowej strony --- */
etykieta = gtk_label_new ("Pierwsza zakładka");
pojemnik = gtk_frame_new ("Informacje jawne");

/* --- Dodajemy stronę na końcu notatnika --- */
gtk_notebook_append_page (notatnik, pojemnik, etykieta);

/* --- Tworzymy etykietę i pojemnik dla nowej strony --- */
etykieta = gtk_label_new ("Ostatnia zakładka");
pojemnik = gtk_frame_new ("Informacje ściśle tajne");

/* --- Dodajemy stronę na końcu notatnika --- */
gtk_notebook_append_page (notatnik, pojemnik, etykieta);

/* --- Tworzymy etykietę i pojemnik dla nowej strony --- */
etykieta = gtk_label_new ("Środkowa zakładka");
pojemnik = gtk_frame_new ("Informacje niejawne");

/* --- Wstawiamy stronę w środek notatnika --- */
***gtk_notebook_append_page (notatnik, pojemnik, etykieta);
```

Manipulowanie stronami

Można użyć funkcji gtk_notebook_current_page, aby pobrać bieżącą stronę oraz funkcji gtk_notebook_set_page, aby ustawić bieżącą stronę. Funkcje gtk_notebook_next_page i gtk_notebook_prev_page umożliwiają poruszanie się w górę i w dół na liście stron. Funkcja gtk_notebook_set_show_tabs pozwala wyświetlić albo ukryć zakładki notatnika.

```
/* --- Pobieramy bieżącą stronę --- */
nStrona = gtk_notebook_current_page (notatnik);

/* --- Przechodzimy do następnej strony --- */
gtk_notebook_set_page (notatnik, nStrona + 1);

/* --- Przechodzimy do poprzedniej strony --- */
gtk_notebook_prev_page (notatnik);

/* --- Przechodzimy do następnej strony (znowu) --- */
gtk_notebook_next_page (notatnik);

/* --- Ukrywamy zakładki przed użytkownikiem --- */
gtk_notebook_set_show_tabs (notatnik, false);
```

Ukrywanie zakładek może początkowo wydać się dziwnym pomysłem, ponieważ użytkownik nie będzie mógł wówczas kliknąć na zakładce, aby przejść do innej strony notatnika, ale czasem aktualnie wyświetlana strona powinna być ustawiana przez aplikację. Przypomina to mechanizm znany z kreatorów w aplikacjach firmy Microsoft, które prowadzą użytkownika przez skomplikowaną procedurę. Procedura może na przykład wymagać przejścia przez dziesięć kroków, a zazwyczaj łatwiej jest wyświetlać jednocześnie tylko jeden krok. Kiedy użytkownik zakończy czynności w danym kroku, aplikacja może przejść do następnej strony i wykonać następny krok.

Program znajdujący się na końcu tego rozdziału zawiera w swoim oknie kilka stron, które ilustrują wykorzystanie kontrolki GtkNotebook w aplikacji.

Kontrolka GtkCList

Kontrolka GtkCList pozwala na wyświetlenie tabelarycznych danych wewnątrz pojedynczej kontrolki. Może zawierać dowolną liczbę kolumn i rzędów i pozwala na ustawienie szerokości kolumn na żądaną wartość. Kontrolkę tę często wykorzystuje się do przeglądania informacji z bazy danych; wówczas każdy rząd w kontrolce GtkCList odpowiada rzędowi w bazie danych, a każda kolumna – polu w bazie danych.

Podczas tworzenia kontrolki GtkCList określamy stałą liczbę kolumn, której nie można już później zmienić. Jeśli zmiana liczby kolumn okaże się konieczna, trzeba usunąć i ponownie utworzyć kontrolkę. Kontrolkę GtkCList tworzymy przy pomocy funkcji gtk_clist_new, której przekazujemy żądaną liczbę kolumn, albo przy pomocy funkcji gtk_clist_new_with_titles, której przekazujemy dodatkowo tablicę nagłówków kolumn. Aby

ustawić albo zmodyfikować nagłówki, możemy skorzystać z funkcji gtk_clist_set_column_title.

```
/* --- Tworzymy tabelę o trzech kolumnach --- */
clista = gtk_clist_new (3);

/* --- Nadajemy tytuły kolumnom --- */
gtk_clist_set_column_title (GTK_CLIST (clista), 0, "ID")
gtk_clist_set_column_title (GTK_CLIST (clista), 1, "Nazwisko")
gtk_clist_set_column_title (GTK_CLIST (clista), 2, "Adres")
```

Poniżej zamieszczamy krótszą wersję tego samego kodu:

```
/* --- Definiujemy nagłówki --- */
char *szNaglowki[] = { "ID", "Nazwisko", "Adres");

/* --- Tworzymy c-listę z nagłówkami pojedynczą instrukcją --- */
clista = gtk_clist_new_with_titles (3, szNaglowki);
```

Dodawanie danych do GtkCList

Można dołączyć dane do kontrolki GtkCList przy pomocy funkcji gtk_clist_append. Przyjmuje ona parametry w postaci c-listy oraz tablicy łańcuchów. Tablica powinna zawierać tyle elementów, ile kolumn ma kontrolka GtkCList.

```
/* --- Dodajemy rząd statycznych danych --- */
char *szDane = {"0123", "Eryk", ul. Główna 123"};

/* --- Dodajemy rząd danych do c-listy --- */
gtk_clist_append (GTK_CLIST (clista), szDane);
```

Oprócz dołączania danych na końcu c-listy, możemy wstawić je w dowolnym punkcie przy pomocy funkcji gtk_clist_insert, przekazując jej indeks, w którym należy wstawić rząd.

```
/* --- Wstawiamy rząd statycznych danych --- */
char *szDane = {"0123", "Eryk", ul. Główna 123"};

/* --- Wstawiamy dane za 10 rzędem --- */
gtk_clist_insert (GTK_CLIST (clista), 10, szDane);
```

Po dodaniu tekstu do c-listy możemy modyfikować go przy pomocy funkcji gtk_clist_set_text. Funkcja gtk_clist_get_text przyjmuje wskaźnik do łańcucha (char **) i wypełnia go danymi z określonego rzędu i kolumny.

Jeśli wskaźnik ma wartość NULL, oznacza to, że kontrolka GtkCList nie jest wypełniona. Funkcja zwraca niezerową wartość, jeśli uda jej się ustawić wskaźnik. Tekst nie jest kopią i nie należy go bezpośrednio modyfikować.

```
char *dane;

/* --- zmieniamy dane w rzędzie/kolumnie --- */
gtk_clist_set_text (GTK_CLIST (clista), 3, 1, "Jan");

/* --- Pobieramy tekst --- */
gtk_clist_get_text (GTK_CLIST (clista), 3, 1, &data);
```

Usuwanie rzędów

Można usunąć rzędy c-listy przy pomocy funkcji gtk_clist_remove, przekazując jej indeks usuwanego rzędu. Jeśli jednak chcemy usunąć wszystkie rzędy, szybciej jest wywołać funkcję gtk_clist_clear.

```
/* --- Usuwamy pierwszy rząd --- */
gtk_clist_remove (GTK_CLIST (clista), 0);

/* --- Usuwamy wszystkie elementy c-listy --- */
gtk_clist_clear (GTK_CLIST (clista));
```

Przyspieszanie wstawiania i usuwania

Podobnie jak kontrolka tekstu, c-lista może zawierać duże zbiory danych, zwłaszcza w przypadku wyświetlania tabel baz danych. Aby przyspieszyć proces wstawiania i usuwania informacji w GtkCList, kontrolkę można „zamrozić", podobnie jak kontrolkę tekstową, aby zapobiec uaktualnianiu danych do czasu zakończenia modyfikacji danych. Funkcja gtk_clist_freeze zapobiega uaktualnianiu kontrolki aż do czasu wywołania funkcji gtk_clist_thaw.

```
/* --- Zamrażamy kontrolkę --- */
gtk_clist_freeze (GTK_CLIST (clista));

/* --- przetwarzamy dane --- */

/* --- Odmrażamy listę --- */
gtk_clist_thaw (GTK_CLIST (clista));
```

Cechy nagłówków

Pasek nagłówków posiada kilka właściwości, które aplikacja może zmodyfikować. Możemy zdecydować, czy w ogóle chcemy pokazywać nagłówki. Funkcja gtk_clist_column_titles_hide ukrywa pasek nagłówków; funkcja gtk_clist_column_titles_show go pokazuje. Zamiast tytułu możemy wyświetlić kontrolkę – funkcja gtk_clist_set_column_widget pozwala umieścić dowolną kontrolkę w pasku nagłówków, dzięki czemu możemy umieścić w nim na przykład rysunki.

```
/* --- Ukrywamy pasek nagłówków --- */
gtk_clist_column_titles_hide (GTK_CLIST (clista));

/* --- Uwidaczniamy nagłówki --- */
gtk_clist_column_titles_show (GTK_CLIST (clista));

/* --- Umieszczamy w nagłówku uprzednio utworzoną piksmapę --- */
gtk_clist_set_column_widget (GTK_CLIST (clista), 1, piksmapa);
```

Parametry rzędów i kolumn

Podczas tworzenia kontrolki GtkCList trzeba samodzielnie dostosować szerokość kolumn. Kontrolka nie wie, jak szerokie powinny być kolumny, i opiera domyślną szerokość kolumny na szerokości nagłówka. Aby ustawić szerokość dowolnej kolumny w c-liście, możemy skorzystać z funkcji gtk_clist_set_column_width. Możemy także ustalić wysokość rzędu przy pomocy funkcji gtk_clist_set_row_height, ale jeśli nie zmieniamy czcionki kontrolki albo nie dodajemy obszernych grafik, ustawianie wysokości rzędu nie powinno być potrzebne.

```
/* --- Ustawiamy szerokość kolumny na 100 pikseli --- */
gtk_clist_set_column_width (GTK_CLIST (clista), 0, 100);

/* --- Zmieniamy wysokość rzędu --- */
gtk_clist_set_row_height (GTK_CLIST (clista), 25);
```

Dane w kolumnach można również wyświetlać z wyrównaniem do środka albo do prawej lub lewej strony. Funkcja gtk_clist_set_column_justification przyjmuje kontrolkę, kolumnę i parametr typu GtkJustification. Dozwolonymi wartościami dla GtkJustification są GTK_JUSTIFY_LEFT, GTK_JUSTIFY_RIGHT, GTK_JUSTIFY_CENTER i GTK_JUSTIFY_FILL.

Grafika w kontrolce GtkCList

Do tej pory omawialiśmy tylko wyświetlanie informacji tekstowych. Patrzenie na długie kolumny tekstu i liczb może być zniechęcające, więc clista pozwala na umieszczanie w rzędach także piksmap. Piksmapę wstawiamy przy pomocy funkcji gtk_clist_set_pixmap; możemy także ją sprawdzić przy pomocy funkcji gtk_clist_get_pixmap. Ważna uwaga: kiedy dodajemy do listy informacje tekstowe (przy pomocy funkcji wstawiających albo dołączających), kolumny, które wyświetlają piksmapy, powinny mieć tekst ustawiony na NULL. W przeciwnym przypadku piksmapy nie zostaną wyświetlone.

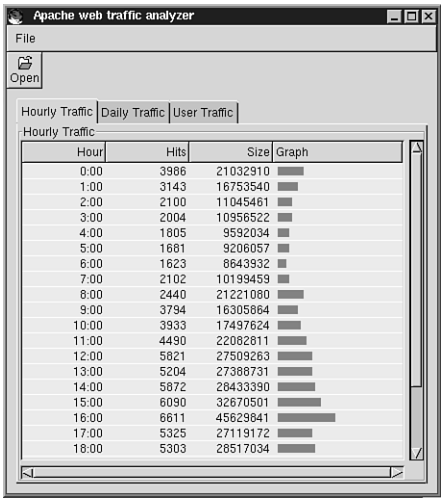
```
/* --- Dodajemy piksmapę --- */
gtk_clist_set_pixmap (GTK_CLIST (clista), rzad, kol, piks, maska);

/* --- Sprawdzamy istniejącą piksmapę --- */
gtk_clist_get_pixmap (GTK_CLIST (clista), rzad, kol, &piks, &maska);
```

Przykładowa aplikacja

Poniższa aplikacja wykorzystuje kontrolki GtkNotebook i GtkCList, aby wyświetlić statystyki z dziennika serwera WWW Apache. Poszczególne statystyki znajdują się na trzech stronach notatnika. Pierwsza wyświetla ruch w sieci według godzin (patrz rysunek 14.3). Informacje te pozwalają stwierdzić, czy w godzinach szczytu dysponujemy odpowiednią przepustowością. Druga strona wyświetla ruch według dni, zaczynając od początku pliku dziennika, co pozwala zauważyć pewne trendy – być może reklama zamieszczona w jednym z serwisów spowodowała gwałtowny wzrost ruchu, czego zapewne wcześniej nie zauważyliśmy.

Trzecia strona pokazuje ruch skierowany do stron użytkowników. Aplikacja traktuje stronę domową serwisu jako odrębnego użytkownika (*ROOT).

Każda lista GtkCList będzie zawierać piksmapy, które będą graficznie obrazować ruch, czy to dzienny, czy godzinowy, czy według użytkownika. Dane te pozwolą administratorowi szybko zauważyć problemy – na przykład serwis dla dorosłych, prowadzony na domowej stronie któregoś z użytkowników. Dużo szybciej jest rzucić okiem na wykres, niż przeglądać rzędy liczb w poszukiwaniu informacji.



Rysunek 14.3. Ruch godzinowy.

Wiele plików wchodzących w skład projektu napisaliśmy wcześniej – ponowne użycie tego samego kodu jest jedną z idei przewodnich książki. Pliki rozne.c, postep.c i wybpliku.c omówiliśmy w poprzednich rozdziałach. Plik interfejs.c został nieco zmodyfikowany, aby uwzględnić zmieniony interfejs aplikacji. Najważniejsze pliki projektu to moduł analizujący dziennik (analiza.c), moduł wyświetlający notatnik i c-listy (notatnik.c) oraz generator piksmap (bitmapy.c).

Piksmapy wyświetlane w tabeli mają szczególną właściwość: są generowane w locie. Zamiast używać statycznych danych dla piksmap, używamy generującej je procedury, która przydziela miejsce na piksmapę i tworzy ją na podstawie rozmiaru poziomego paska, który chcemy wyświetlić. Generowanie poziomego, graficznego paska jest nieskomplikowane, a zarazem elastyczniejsze, niż tworzenie statycznych danych dla piksmapy. Możemy łatwo zmienić rozmiar i rozdzielczość wykresu, modyfikując procedurę, która tworzy dane piksmapy. Gdybyśmy jednak zdecydowali się użycie statycznych rysunków, spędzilibyśmy mnóstwo czasu nad edycją wszystkich możliwych piksmap, z których tworzone są paski wykresu.

typydziennika.h

Plik typydziennika.h definiuje struktury danych, używane w aplikacji. Każda struktura przechowuje konkretny zbiór danych, które musimy uwzględnić w aplikacji.

Struktura typZnacznikCzasowy przechowuje datę i czas, w którym nastąpiło „trafienie" w stronę WWW.

```
typedef struct {

  int rok;
  int miesiac;
  int dzien;
  int godziny;
  int minuty;
  int sekundy;

} typZnacznikCzasowy;
```

Struktura typTrafienie przechowuje wszystkie informacje o trafieniu w stronę WWW.

```
typedef struct {

  char *sIp;
  char *sUzytk;
  char *sData;
  char *sPol;
  char *sURL;
  int nRezultat;
  int nRozmiar;
  typZnacznikCzasowy data;

} typTrafienie;
```

Struktura typStat przechowuje liczbę trafień w URL i rozmiar przesłanych danych (w bajtach).

```
typedef struct {

  char *sURL;
  long nTrafienia;
  long nRozmiar;

} typStat;
```

Struktura typData przechowuje datę i służy do analizowania ruchu według daty.

```
typedef struct {

  int rok;
```

```
    int miesiac;
    int dzien;

} typData;
```

Struktura typDaneDaty przechowuje liczbę trafień i rozmiar przesłanych danych (w bajtach) w jednym dniu..

```
typedef struct {

    long nTrafienia;
    long nRozmiar;
    typData *data;

} typDaneDaty;
```

analiza.c

Aby można było wyświetlić informacje, trzeba je odczytać i podsumować. Dane są przechowywane w drzewie GTree, dzięki czemu sprawdzanie i pobieranie informacji przebiega szybko. Dziennik serwera zawiera wiele rzędów danych, a my potrzebujemy szybkiego dostępu do aktualizowanej informacji. Każdy rekord z dziennika jest wczytywany, analizowany i zachowywany jako trafienie. Trafienie jest następnie przekazywane do różnych funkcji, które tworzą podsumowanie na podstawie danych zawartych w trafieniu. Jeśli na przykład pobrano stronę konkretnego użytkownika w konkretnym dniu, trafienie zostanie dodane do sumarycznych danych użytkownika oraz do sumarycznych danych dnia. Kiedy analiza pliku dziennika dobiegnie końca, wówczas można wyświetlić go w c-liście na podstawie sumarycznych danych.

```
/*
 * Plik: analog.c
 * Auth: Eric Harlow
 *
 * Analizuje plik dziennika serwera Apache
 * i podsumowuje zawarte w nim informacje.
 */

#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
```

```
#include <time.h>
#include <gtk/gtk.h>
#include "typydziennika.h"

void Inicjacja ();
void SledzUzytkownikow (typTrafienie *trafienie);
void SledzDaty (typTrafienie *trafienie);
void SledzPliki (typTrafienie *trafienie);
void ZacznijPostep ();
void UaktualnijPostep (long poz, long dlug);
void ZakonczPostep ();
void UaktualnijStatystyki (typTrafienie *trafienie);

/*
 * Tablice do przechowywania danych godzinowych
 */
long trafieniaOGodzinie[24];
long rozmiarOGodzinie[24];

/*
 * Drzewa przechowujące różne typy danych
 */
GTree *drzewoPlikow = NULL;
GTree *drzewoDat = NULL;
GTree *drzewoUzytk = NULL;

void AnalizujLinie (char *bufor);

#define MIESIACE 12
char *sPoprawneMiesiace[] = { "Jan", "Feb", "Mar", "Apr", "May", "Jun",
                              "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" };

/*
 * DataNaLancuch
 *
 * Zamienia datę na łańcuch
 * data - data do przekształcenia
 * bufor - char* wystarczająco duży, aby przechować datę/czas.
 */
void DataNaLancuch (typZnacznikCzasowy *data, char *bufor)
{
   sprintf (bufor, "%d/%d/%d %d:%d:%d ",
                           data->miesiac,
```

```
                    data->dzien,
                    data->rok,
                    data->godziny,
                    data->minuty,
                    data->sekundy);
}

/*
 * PrzeksztalcDate
 *
 * Przekształca datę z formatu 30/Feb/1999:11:42:23,
 * w którym serwer Apache zapisuje daty w swoich dziennikach.
 */
void PrzeksztalcDate (char *sData, typZnacznikCzasowy *data)
{
   char sMiesiac[33];
   int i;

   /* --- Break down the data into its components --- */
   sscanf (sData, "%d/%3c/%d:%d:%d:%d", &data->dzien,
                                 sMiesiac,
                                 &data->rok,
                                 &data->godziny,
                                 &data->minuty,
                                 &data->sekundy);

   /* --- Zamieniamy datę tekstową na liczbową --- */
   for (i = 0; i < MIESIACE; i++) {

      /* --- Czy to miesiąc? --- */
      if (!strncasecmp (sPoprawneMiesiace[i], sMiesiac, strlen
➥(sPoprawneMiesiace[i]))) {
         data->miesiac = i + 1;
         break;
      }
   }
}

/*
 * AnalizujDziennik
 *
 * Analizuje plik dziennika i organizuje zawarte w nim dane
```

```
 * w formacie, który będziemy mogli poddać interpretacji.
 *
 * sPlik - Plik do odczytania
 */
void AnalizujDziennik (char *sPlik)
{
   FILE *fp;
   char bufor[350];
   long nDlugoscPliku = 0;
   struct stat statusPliku;

   /* --- Przydzielamy drzewa i inicjujemy dane --- */
   Inicjacja ();

   /* --- Pobieramy nazwę pliku --- */
   stat (sPlik, &statusPliku);
   nDlugoscPliku = statusPliku.st_size;

   /* --- Otwieramy le plik <--- to po francusku. --- */
   fp = fopen (sPlik, "r");

   /* --- Upewniamy się, że *coś* otworzyliśmy --- */
   if (fp) {

      /* --- Pokazujemy pasek postępów --- */
      ZacznijPostep ();

      /* --- Dopóki mamy dane --- */
      while (!feof (fp)) {

         /* --- Uaktualniamy pasek postępów --- */
         UaktualnijPostep (ftell (fp), nDlugoscPliku);

         /* --- Odczytujemy linię pliku --- */
         fgets (bufor, sizeof (bufor), fp);

         /* --- Jeśli to nie jest koniec pliku --- */
         if (!feof (fp)) {

            /* --- Analizujemy dane --- */
            AnalizujLinie (bufor);
         }
      }
```

```
        /* --- Gotowe - pasek postępów jest już niepotrzebny --- */
        ZakonczPostep ();

        /* --- Zamykamy plik --- */
        fclose (fp);
    }
}

/*
 * AnalizujLinie
 *
 * Analizujemy linię odczytaną z pliku dziennika, rozbijamy
 * ją na poszczególne składniki i uaktualniamy przechowywane
 * statystyki.
 */
void AnalizujLinie (char *bufor)
{

    char *sNieznany;
    char *sNull;
    char *sDane;
    char rezultat[88];
    typTrafienie trafienie;

    /* --- Pobieramy podstawowe informacje --- */
    trafienie.sIp = strtok (bufor, " ");
    sNieznany = strtok (NULL, " ");
    trafienie.sUzytk = strtok (NULL, " ");

    // --- Pobieramy dane --- */
    sDane = strtok (NULL, "]");
    sDane++;

    /* --- Przekształcamy łańcuch daty --- */
    PrzeksztalcDate (sDane, &trafienie.data);

    /* --- Pobieramy plik, rozmiar, status --- */
    sNull = strtok (NULL, "\"");
    trafienie.sPol = strtok (NULL, "\"");
    trafienie.nRezultat = atoi (strtok (NULL, " "));
    trafienie.nRozmiar = atoi (strtok (NULL, " "));

    /* --- Wysyłanie strony zakończyło się błędem --- */
```

```
    if (trafienie.nRezultat == 404) return;

    /* --- Zamieniamy z powrotem na łańcuch --- */
    DataNaLancuch (&trafienie.data, rezultat);

    sNieznany = strtok (trafienie.sPol, " ");
    trafienie.sURL = strtok (NULL, " ");

    /* --- Poprawny URL--- */
    if (trafienie.sURL) {

        /* --- Uaktualniamy statystyki --- */
        UaktualnijStatystyki (&trafienie);
    }
}

/*
 * UaktualnijStatystyki
 *
 * Uaktualnia informacje o "trafieniu".
 * Obecnie przechowywanymi statystykami są:
 *   Czas, w którym nastapiło trafienie
 *   Ruch w poszczególnych dniach
 *   Ruch związany z poszczególnymi komputerami w sieci
 */
void UaktualnijStatystyki (typTrafienie *trafienie)
{
    /* -- Uaktualniamy statystyki godzinowe --- */
    trafieniaOGodzinie[trafienie->data.godziny]++;
    rozmiarOGodzinie[trafienie->data.godziny] += trafienie->nRozmiar;

    /* --- Uaktualniamy pozostałe informacje --- */
    SledzPliki (trafienie);
    SledzDaty (trafienie);
    SledzUzytkownikow (trafienie);
}

/*
 * SledzUzytkownikow
 *
 * Śledzi ruch według stron użytkowników
 */
void SledzUzytkownikow (typTrafienie *trafienie)
```

```
{
   typStat *stat;
   char *sUzytkownik;

   /* --- Złe dane --- */
   if (trafienie->sURL == NULL || strlen (trafienie->sURL) == 0)
return;

   /* --- Pobieramy pierwszą część ścieżki --- */
   sUzytkownik = strtok (&trafienie->sURL[1], "/");

   if (sUzytkownik == NULL) return;

   /* --- Jeśli jest to strona domowa użytkownika --- */
   if (sUzytkownik[0] == '~') {

      /* --- Pobieramy nazwę --- */
      sUzytkownik = &sUzytkownik[1];

   /* --- Jeśli ~ było zakodowane jako %7E --- */
   } else if (!strcmp (sUzytkownik, "%7E")) {

      /* --- Pobieramy nazwę --- */
      sUzytkownik = &sUzytkownik[3];
   } else {

      /* --- Nie jest to strona użytkownika --- */
      sUzytkownik = "*ROOT";
   }

   /* --- Sprawdzamy, czy użytkownik już istnieje w drzewie --- */
   stat = g_tree_lookup (drzewoUzytk, sUzytkownik);

   /* --- Jeśli użytkownik nie istnieje --- */
   if (stat == NULL) {

      /* --- Tworzymy miejsce na użytkownika --- */
      stat = g_malloc (sizeof (typStat));

      /* --- Wypełniamy pola --- */
      stat->sURL = strdup (sUzytkownik);
      stat->nTrafienia = 0;
      stat->nRozmiar = 0;

      /* --- Wstawiamy użytkownika do drzewa --- */
```

```
        g_tree_insert (drzewoUzytk, stat->sURL, stat);
    }

    /* --- Uaktualniamy liczbę trafień dla użytkownika --- */
    stat->nTrafienia ++;
    stat->nRozmiar += trafienie->nRozmiar;
}

/*
 * SledzPliki
 *
 * Śledzi liczbę trafień w poszczególne pliki
 */
void SledzPliki (typTrafienie *trafienie)
{
    typStat *stat;

    /* --- Wyszukujemy URL w drzewie --- */
    stat = g_tree_lookup (drzewoPlikow, trafienie->sURL);

    /* --- Jeśli nie znajdziemy URL-a --- */
    if (stat == NULL) {

        /* --- Tworzymy węzeł dla URL-a --- */
        stat = g_malloc (sizeof (typStat));

        /* --- Wypełniamy węzeł --- */
        stat->sURL = strdup (trafienie->sURL);
        stat->nTrafienia = 0;
        stat->nRozmiar = 0;

        /* --- Dodajemy węzeł do drzewa --- */
        g_tree_insert (drzewoPlikow, stat->sURL, stat);
    }

    /* --- Uaktualniamy liczbę trafień w plik --- */
    stat->nTrafienia ++;
    stat->nRozmiar = trafienie->nRozmiar;
}

/*
 * SledzDaty
 *
 * Śledzi ruch w poszczególnych dniach
```

```
 */
void SledzDaty (typTrafienie *trafienie)
{
   typData data;
   typData *nowadata;
   typDaneDaty *info;

   /* --- Pobieramy datę trafienia --- */
   data.rok = trafienie->data.rok;
   data.miesiac = trafienie->data.miesiac;
   data.dzien = trafienie->data.dzien;

   /* --- Wyszukujemy dane --- */
   info = g_tree_lookup (drzewoDat, &data);

   /* --- Nie ma danych dla daty? --- */
   if (info == NULL) {

      /* --- Tworzymy pole przechowujące dane daty --- */
      info = g_malloc (sizeof (typDaneDaty));
      nowadata = g_malloc (sizeof (typData));

      /* --- Wypełniamy pola --- */
      *nowadata = data;

      info->nTrafienia = 0;
      info->nRozmiar = 0;
      info->data = nowadata;

      /* --- Dodajemy dane daty do drzewa --- */
      g_tree_insert (drzewoDat, nowadata, info);
   }

   /* --- Uaktualniamy liczbę trafień dla daty --- */
   info->nTrafienia ++;
   info->nRozmiar += trafienie->nRozmiar;
}

/*
 * PorownajLancuchy
 *
 * Funkcja do porównywania dwóch łańcuchów, wykorzystywana
 * jako funkcja zwrotna dla drzewa. Moglibyśmy umieścić funkcję
 * strcmp bezpośrednio jako funkcję zwrotną drzewa, ale
```

```
 * w przyszłości możemy zechcieć zmienić sposób porównywania.
*/
gint PorownajLancuchy (gpointer g1, gpointer g2)
{
   return (strcmp ((char *) g1, (char *) g2));
}

/*
 * PorownajDaty
 *
 * Porównuje dwie daty i zwraca wartość ujemną, jeśli pierwsza
 * data jest mniejsza od drugiej, dodatnią, jeśli pierwsza
 * data jest większa od drugiej, i zero, jeśli daty są równe.
 */
gint PorownajDaty (gpointer g1, gpointer g2)
{
   typData *d1 = (typData *) g1;
   typData *d2 = (typData *) g2;

   /* --- Rok ma najwyższe pierwszeństwo --- */
   if (d1->rok == d2->rok) {

      /* --- Lata te same, a miesiące? --- */
      if (d1->miesiac == d2->miesiac) {

         /* --- Zwracamy różnicę pomiędzy dniami --- */
         return (d1->dzien - d2->dzien);
      } else {

         /* --- Miesiące są różne - obliczamy przyrost --- */
         return (d1->miesiac - d2->miesiac);
      }
   } else {

      /* --- Lata są różne - obliczamy przyrost --- */
      return (d1->rok == d2->rok);
   }
}

/*
 * PobierzTrafieniaOGodzinie
 *
 * Funkcja zwraca liczbę trafień dla danego czasu dnia
```

```
 */
void PobierzTrafieniaOGodzinie (int nGodziny, long *trafienia, long *rozmiar)
{
   /* --- Jeśli zegar jest poza zakresem --- */
   if (nGgodziny < 0 || nGodziny > 23) {

      *trafienia = 0;
      *rozmiar = 0;

   } else {
      *trafienia = trafieniaOGodzinie[nGodziny];
      *rozmiar = rozmiarOGodzinie[nGodziny];
   }
}

/*
 * Inicjacja
 *
 * Inicjuje dane, aby można było wczytać plik dziennika.
 * Tworzy drzewa, przechowujące informacje z dziennika.
 */
void Inicjacja ()
{
   int i;

   /* --- Tworzymy drzewa do przechowywania informacji --- */
   drzewoPlikow = g_tree_new (PorownajLancuchy);
   drzewoDat = g_tree_new (PorownajDaty);
   drzewoUzytk = g_tree_new (PorownajLancuchy);

   /* --- Oczyszczamy tablicę z trafieniami według godzin --- */
   for (i = 0; i < 24; i++) {
      trafieniaOGodzinie[i] = 0;
      rozmiarOGodzinie[i] = 0;
   }
}

/*
 * ZwolnijURLe
 *
 * Zwalniamy pamięć przydzieloną na URL-e.
 * Jest to funkcja zwrotna obchodu drzewa.
 */
```

```
gint ZwolnijURLe (gpointer klucz, gpointer wartosc, gpointer dane)
{
   typStat *info;

   info = (typStat *) wartosc;
   free (info->sURL);
   g_free (info);
   return (0);
}

/*
 * ZwolnijDaty
 *
 * Zwalniamy pamięć przydzieloną na śledzenie ruchu
 * według dat. Jest to funkcja zwrotna obchodu drzewa.
 */
gint ZwolnijDaty (gpointer klucz, gpointer wartosc, gpointer dane)
{
   typDaneDaty *info;

   info = (typDaneDaty *) wartosc;
   g_free (info->data);
   g_free (info);
   return (0);
}

/*
 * ZwolnijZasoby
 *
 * Zwalnia pamięć używaną przez węzły drzewa, a następnie
 * usuwa drzewa.
 */
void ZwolnijZasoby ()
{
   /* --- Zwalniamy dane przechowywane w drzewie --- */
   g_tree_traverse (drzewoUzytk, ZwolnijURLe, G_IN_ORDER, NULL);
   g_tree_traverse (drzewoDat, ZwolnijDaty, G_IN_ORDER, NULL);
   g_tree_traverse (drzewoPlikow, ZwolnijURLe, G_IN_ORDER, NULL);

   /* --- Usuwamy drzewa --- */
   g_tree_destroy (drzewoUzytk);
   g_tree_destroy (drzewoDat);
```

```
    g_tree_destroy (drzewoPlikow);

    /* --- Zerujemy wskaźniki --- */
    drzewoUzytk = NULL;
    drzewoDat = NULL;
    drzewoPlikow = NULL;
}
```

bitmapy.c

Kontrolka GtkClist wyświetla wykres, który ilustruje ruch w serwisie, według dnia albo godziny. Zamiast tworzyć kilka plików .xpm z piksmapami, lepiej użyć kodu, który dynamicznie wygeneruje dane xpm.

Dzięki temu łatwo jest zmienić szerokość wykresu. Zamiast przerysowywać poszczególne paski, możemy po prostu określić, jak szeroki powinien być wykres. Zauważmy, że wykres jest poziomy, dzięki czemu dane piksmapy są takie same dla każdego rzędu. Możemy wykorzystać to spostrzeżenie i użyć tego samego łańcucha we wszystkich rzędach danych piksmapy.

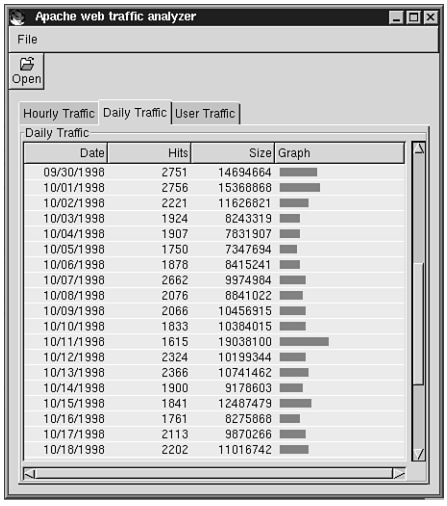
```
#include <gtk/gtk.h>
#include <strings.h>

/*
 * UtworzBitmapePaska
 *
 * Tworzy piksmapę o zadanych charakterystykach
 *
 * wysok - wysokość tworzonej piksmapy
 * szerok - szerokość tworzonej piksmapy
 * rozmiar - jak długi powinien być pasek
 * sKolor - kolor wypełnienia paska
 */
char **UtworzBitmapePaska (int wysok, int szerok, int rozmiar, char *sKolor)
{
    char **sBitmapa;
    char *nowybufor;
    char bufor[88];
    int i;

    /* --- Przydzielamy miejsce na dane --- */
    sBitmapa = g_malloc ((wysok + 1 + 2) * sizeof (gchar *));
```

```
    /* --- Tworzymy nagłówek piksmapy - wysokość/szerokość/
           kolory/liczba znaków na kolor --- */
    sprintf (bufor, "%d %d 2 1", szerok, wysok);
    sBitmapa[0] = g_strdup (bufor);

    /* --- Definiujemy kolor przezroczysty ("none") --- */
    sBitmapa[1] = g_strdup ("  c None");

    /* --- Definiujemy kolor wypełnienia --- */
    sprintf (bufor, "X c %s", sKolor);
    sBitmapa[2] = g_strdup (bufor);

    /* --- Wypełniamy bufor na podstawie rozmiaru --- */
    strcpy (bufor, " ");
    for (i = 0; i < rozmiar; i++) {

       strcat (bufor, "X");
    }

    /* --- Reszta zostaje niewypełniona --- */
    while (i < szerok) {
       strcat (bufor, " ");
       i++;
    }

    /* --- Dodajemy spację --- */
    strcat (bufor, " ");

    /* --- Kopiujemy łańcuch --- */
    nowybufor = g_strdup (bufor);

    /* --- Takie same dane dla wszystkich rzędów piksmapy --- */
    for (i = 3; i < wysok+3; i++) {
       sBitmapa[i] = nowybufor;
    }

    /* --- Zwracamy utworzoną piksmapę --- */
    return (sBitmapa);
}

/*
 * ZwolnijBitmapePaska
 *
 * Zwalniamy pamięć, przydzieloną na dane bitmapy.
```

```
 */
void ZwolnijBitmapePaska (char **bitmapa)
{
   g_free (bitmapa[0]);
   g_free (bitmapa[1]);
   g_free (bitmapa[2]);
   g_free (bitmapa[3]);
   g_free (bitmapa);
}
```

notatnik.c

Po załadowaniu danych do drzew wywołujemy zamieszczone niżej procedury, które wyświetlają dane w poszczególnych kontrolkach GtkCList. Kontrolki tworzone są na oddzielnych stronach, a użytkownik może zmieniać strony, aby obejrzeć zawarte na nich informacje. Dane dla kontrolek GtkClist tworzone są poprzez obchód drzewa. W trakcie obchodu pobieramy i wyświetlamy informacje tekstowe (data, użytkownik, trafienie itd.) oraz obliczmy maksymalny ruch. Po ustaleniu maksymalnego ruchu obchodzimy drzewo ponownie, aby wyświetlić wykresy. Obliczenie maksymalnego natężenia ruchu jest niezbędne, ponieważ wykresy są rysowane względem największej wartości i nie mogą zostać wyświetlone, dopóki jej nie ustalimy.



Rysunek 14.4. Ruch dzienny.

```
/*
 * Autor: Eric Harlow
 * Plik: notatnik.c
 *
 * Aplikacja korzystająca z kontrolki notatnika.
 */

#include <gtk/gtk.h>
#include "typydziennika.h"

extern GTree *drzewoDat;
extern GTree *drzewoUzytk;

GtkWidget *stronaGodziny = NULL;
GtkWidget *stronaDni = NULL;
GtkWidget *stronaUzytk = NULL;
GtkWidget *clistaGodziny = NULL;
GtkWidget *clistaDni = NULL;
GtkWidget *clistaUzytk = NULL;

typedef struct {
   GtkWidget *kontrolka;
   long nMaksRozmiar;
   long rzad;
} typDaneWykresu;

/*
 * Tytuły wyświetlane w C-listach na różnych stronach
 * Titles displayed on the clist for the various pages
 */
char *szTytulyGodziny[] = {"Godzina", "Trafienia", "Rozmiar", "Wykres"};
char *szTytulyDni[] = {"Data", "Trafienia", "Rozmiar", "Wykres"};
char *szTytulyUzytk[] = {"Użytkownik", "Trafienia", "Rozmiar", "Wykres"};

#define LICZBA_WYKRESOW 21
GdkPixmap *piksmapyWykresow [LICZBA_WYKRESOW];
GdkBitmap *maski[LICZBA_WYKRESOW];

char **UtworzBitmapePaska (int wysok, int szerok, int rozmiar, char *sKolor);
void ZwolnijZasoby ();
void WypelnijUzytk ();
void WypelnijDni ();
void WypelnijGodziny ();
```

```
void PobierzTrafieniaWGodzinie (int nGodziny, long *trafienia, long *rozmiar);
void ZwolnijBitmapePaska (char **bitmapa);

/*
 * GenerujPiksmapy
 *
 * Generuje piksmapy dla poziomych pasków wykresu o wszystkich
 * rozmiarach, które obsługuje aplikacja.
 */
void GenerujPiksmapy (GtkWidget *kontrolka)
{
   int i;
   gchar **piksmapa_d;

   /* --- Dla każdego możliwego wykresu --- */
   for (i = 0; i < LICZBA_WYKRESOW; i++) {

      /* --- Pobieramy dane dla wykresu --- */
      piksmapa_d = UtworzBitmapePaska (9, 65, i * 3, "#ff0000");

      /* --- Tworzymy piksmapę --- */
      piksmapyWykresow[i] = gdk_pixmap_create_from_xpm_d (
                  kontrolka->window,
                  &maski[i], NULL,
                  (gpointer) piksmapa_d);

      /* --- Zwalniamy dane --- */
      ZwolnijBitmapePaska (piksmapa_d);
   }
}

/*
 * ZmianaStrony
 *
 * Zdarzenie to występuje, kiedy ognisko przesuwa
 * się na inną stronę.
 */
static void ZmianaStrony (GtkWidget *kontrolka,
                  GtkNotebookPage *strona,
                  gint numer_strony)
{

}
```

```
/*
 * DodajStrone
 *
 * Dodaje stronę do notatnika
 *
 * notatnik - istniejący notatnik
 * szNazwa - nazwa dodawanej strony
 */
GtkWidget *DodajStrone (GtkWidget *notatnik, char *szNazwa)
{
   GtkWidget *etykieta;
   GtkWidget *ramka;

   /* --- Tworzymy etykietę na podstawie nazwy --- */
   etykieta = gtk_label_new (szNazwa);
   gtk_widget_show (etykieta);

   /* --- Tworzymy ramkę na stronie --- */
   ramka = gtk_frame_new (szNazwa);
   gtk_widget_show (ramka);

   /* --- Dodajemy stronę z ramką i etykietą --- */
   gtk_notebook_append_page (GTK_NOTEBOOK (notatnik), ramka,
etykieta);

   return (ramka);
}

/*
 * UtworzNotatnik
 *
 * Tworzy nowy notatnik i dodaje do niego strony.
 *
 * okno - okno, w którym znajdzie się notatnik.
 */
void UtworzNotatnik (GtkWidget *okno)
{
   GtkWidget *notatnik;

   /* --- tworzymy notatnik --- */
   notatnik = gtk_notebook_new ();

   /* --- Sprawdzamy wystąpienie zdarzenia switch_page --- */
```

```
    gtk_signal_connect (GTK_OBJECT (notatnik), "switch_page",
                GTK_SIGNAL_FUNC (ZmianaStrony), NULL);

    /* --- Zakładki mają znajdować się na górze --- */
    gtk_notebook_set_tab_pos (GTK_NOTEBOOK (notatnik),
GTK_POS_TOP);

    /* --- Dodajemy notatnik to okna --- */
    gtk_box_pack_start (GTK_BOX (okno), notatnik, TRUE, TRUE, 0);

    /* --- Ustawiamy obramowanie notatnika --- */
    gtk_container_border_width (GTK_CONTAINER (notatnik), 10);

    /* --- Dodajemy strony do notatnika --- */
    stronaGodziny = DodajStrone (notatnik, "Ruch godzinowy");
    stronaDni = DodajStrone (notatnik, "Ruch dzienny");
    stronaUzytk = DodajStrone (notatnik, "Ruch wg. użytkownika");

    /* --- Pokazujemy wszystkie kontrolki. --- */
    gtk_widget_show_all (okno);
}

/*
 * WypelnijStrony
 *
 * Wypełnia strony notatnika informacjami z dziennika.
 * Wypełnia stronę godzinową, dzienną i wg. użytkownika.
 *
 * Zwania dane, wykorzystane do wygenerowania stron.
 */
void WypelnijStrony ()
{
    /* --- Zwalniamy dane C-list, jeśli listy były
           już używane --- */
    if (clistaUzytk) {
        gtk_clist_clear (GTK_CLIST (clistaUzytk));
    }
    if (clistaGodziny) {
        gtk_clist_clear (GTK_CLIST (clistaGodziny));
    }
    if (clistaDni) {
        gtk_clist_clear (GTK_CLIST (clistaDni));
    }
```

```
    /* --- Wypełniamy wszystkie strony --- */
    WypelnijGodziny ();
    WypelnijDni ();
    WypelnijUzytk ();

    /* --- Zwalniamy zasoby, przydzielone podczas
           analizy dziennika --- */
    ZwolnijZasoby ();
}

/*
 * WypelnijGodziny
 *
 * Wypełnia C-listę danymi o ruchu godzinowym.
 * Zakładamy, że drzewa są wypełnione gotowymi do
 * użycia danymi.
 */
void WypelnijGodziny ()
{
    gchar *strWartosc[4];
    int i;
    int ix;
    long trafienia;
    long rozmiar;
    gchar bufor0[88];
    gchar bufor1[88];
    gchar bufor2[88];
    long nMaksRozmiar = 0;

    /* --- Tablica, używana do wstawiania danych
           do C-listy --- */
    strWartosc[0] = bufor0;
    strWartosc[1] = bufor1;
    strWartosc[2] = bufor2;

    /* --- Tutaj mamy NULL, ponieważ jest to piksmapa --- */
    strWartosc[3] = NULL;

    /* --- Jeśli c-lista nie jest jeszcze utworzona... --- */
    if (clistaGodziny == NULL) {

        /* --- Tworzymy c-listę o czterech kolumnach --- */
        clistaGodziny = gtk_clist_new_with_titles (4, szTytulyGodziny);
```

```
    /* --- Uwidaczniamy nagłówki kolumn --- */
    gtk_clist_column_titles_show (GTK_CLIST (clistaGodziny));

    /* --- Ustawiamy szerokości kolumn --- */
    gtk_clist_set_column_width (GTK_CLIST (clistaGodziny), 0, 80);
    gtk_clist_set_column_width (GTK_CLIST (clistaGodziny), 1, 80);
    gtk_clist_set_column_width (GTK_CLIST (clistaGodziny), 2, 80);
    gtk_clist_set_column_width (GTK_CLIST (clistaGodziny), 3, 40);

    /* --- Ustawiamy justowanie każdej z kolumn --- */
    gtk_clist_set_column_justification (GTK_CLIST (clistaGodziny),
                                        0, GTK_JUSTIFY_RIGHT);
    gtk_clist_set_column_justification (GTK_CLIST (clistaGodziny),
                                        1, GTK_JUSTIFY_RIGHT);
    gtk_clist_set_column_justification (GTK_CLIST (clistaGodziny),
                                        2, GTK_JUSTIFY_RIGHT);

    /* --- Dodajemy c-listę do właściwej strony --- */
    gtk_container_add (GTK_CONTAINER (stronaGodziny), clistaGodziny);
}

/* --- Generujemy rząd dla każdej z 24 godzin dnia --- */
for (i = 0; i < 24; i++) {

    /* --- Pokazujemy czas - np. 3:00 --- */
    sprintf (strWartosc[0], "%d:00", i);

    /* --- Pobieramy liczbę trafień w tej godzinie --- */
    PobierzTrafieniaOGodzinie (i, &trafienia, &rozmiar);

    /* --- Wyświetlamy liczbę trafień i przesłanych bajtów --- */
    sprintf (strWartosc[1], "%ld", trafienia);
    sprintf (strWartosc[2], "%ld", rozmiar);

    /* --- Dodajemy dane do c-listy --- */
    gtk_clist_append (GTK_CLIST (clistaGodziny), strWartosc);

    /* --- Zapamiętujemy największy blok przesłanych danych --- */
    if (rozmiar > nMaksRozmiar) {
       nMaksRozmiar = rozmiar;
    }
}

/*
```

```
     * Po wygenerowaniu c-listy musimy dodać do niej poziome
     * wykresy. Nie mogliśmy zrobić tego wcześniej, ponieważ nie
     * wiedzieliśmy, jaka jest maksymalna wartość.
     */

    /* --- Dla każdej godziny dnia --- */
    for (i = 0; i < 24; i++) {

        /* --- Pobieramy trafienia w tej godzinie --- */
        PobierzTrafieniaOGodzinie (i, &trafienia, &rozmiar);

        /* --- Obliczamy długość wykresu --- */
        ix = (rozmiar * LICZBA_WYKRESOW-1) / nMaksRozmiar;

        /* --- Wyświetlamy wykres w c-liście --- */
        gtk_clist_set_pixmap (GTK_CLIST (clistaGgodziny),
            i, 3, (GdkPixmap *) piksmapyWykresow[ix], maski[ix]);
    }

    /* --- Pokazujemy c-listę --- */
    gtk_widget_show_all (GTK_WIDGET (clistaGodziny));
}

/*
 * PokazDaneDnia
 *
 * Pokazuje informacje o ruchu w danym dniu.
 * Umieszcza je na c-liście, która wyświetla wykres
 * ruchu dziennego.
 *
 * Wywoływana przez funkcję zwrotną obchodu drzewa!
 */
gint PokazDaneDnia (gpointer klucz, gpointer wartosc, gpointer dane)
{
    char *strWartosc[4];
    typDaneDaty *daneDnia;
    long *pnMaks;
    char bufor0[88];
    char bufor1[88];
    char bufor2[88];

    /* --- Pobieramy przekazane dane --- */
    daneDnia = (typDaneDaty *) wartosc;
```

```
    pnMaks = (long *) dane;

    /* --- Ustawiamy struktury, wypełniające c-listę --- */
    strWartosc[0] = bufor0;
    strWartosc[1] = bufor1;
    strWartosc[2] = bufor2;
    strWartosc[3] = NULL;

    /* --- Umieszczamy datę w pierwszej kolumnie --- */
    sprintf (strWartosc[0], "%02d/%02d/%4d", daneDnia->data->miesiac,
                                    daneDnia->data->dzien,
                                    daneDnia->data->rok);

    /* --- Wpisujemy trafienia i ilość przesłanych danych --- */
    sprintf (strWartosc[1], "%ld", daneDnia->nTrafienia);
    sprintf (strWartosc[2], "%ld", daneDnia->nRozmiar);

    /* --- Dodajemy dane do c-listy --- */
    gtk_clist_append (GTK_CLIST (clistaDni), strWartosc);

    /* --- Zapamiętujemy maksymalną wartość --- */
    if (*pnMaks < daneDnia->nRozmiar) {

       *pnMaks = daneDnia->nRozmiar;
    }

    /* --- 0 => kontynuujemy obchód --- */
    return (0);
}

/*
 * PokazDaneUzytk
 *
 * Pokazuje informacje o użytkowniku (bez wykresów), ale
 * zachowuje maksymalną liczbę bajtów, aby można było
 * później wygenerować wykres.
 *
 * Wywoływana przez funkcję zwrotną obchodu drzewa!
 */
gint PokazDaneUzytk (gpointer klucz, gpointer wartosc, gpointer dane)
{
    char *strWartosc[4];
    typStat *info;
```

```
    long *pnMaks;
    char bufor0[88];
    char bufor1[88];
    char bufor2[88];

    /* --- Pobieramy przekazane dane --- */
    info = (typStat *) wartosc;
    pnMaks = (long *) dane;

    /* --- Bufory do dołączania danych --- */
    strWartosc[0] = bufor0;
    strWartosc[1] = bufor1;
    strWartosc[2] = bufor2;
    strWartosc[3] = NULL;

    /* --- Aktualizujemy URL w pierwszej kolumnie --- */
    sprintf (strWartosc[0], "%s", info->sURL);

    /* --- Aktualizujemy trafienia i rozmiar w bajtach --- */
    sprintf (strWartosc[1], "%ld", info->nTrafienia);
    sprintf (strWartosc[2], "%ld", info->nRozmiar);

    /* --- Dodajemy dane do c-listy --- */
    gtk_clist_append (GTK_CLIST (clistaUzytk), strWartosc);

    /* --- Zapamiętujemy maksymalny rozmiar --- */
    if (info->nRozmiar > *pnMaks) {
        *pnMaks = info->nRozmiar;
    }

    return (0);
}

/*
 * PokazWykres
 *
 * Wyświetla dzienny wykres.
 *
 * Wywoływana jako funkcja zwrotna obchodu drzewa
 */
gint PokazWykres (gpointer klucz, gpointer wartosc, gpointer dane)
{
    int ix;
```

```
    typDaneWykresu *daneWykresu = (typDaneWykresu *) dane;
    typDaneDaty *daneDnia = (typDaneDaty *) wartosc;

    /* --- Określamy, który wykres należy wyświetlić --- */
    ix = (daneDnia->nRozmiar * LICZBA_WYKRESOW-1) /
                ➥daneWykresu->nMaksRozmiar;

    /* --- Ustawiamy piksmapę w c-liście --- */
    gtk_clist_set_pixmap (GTK_CLIST (daneWykresu->kontrolka),
                    daneWykresu->rzad, 3, piksmapyWykresow[ix],
                    maski[ix]);

    /* --- Następny rząd do wyświetlenia --- */
    daneWykresu->rzad++;

    /* --- Kontynuujemy... --- */
    return (0);
}

/*
 * WypelnijDni
 *
 * Wypełnia c-listę danymi z drzewa, Zakłada, że
 * drzewo jest całkowicie wypełnione danymi.
 */
void WypelnijDni ()
{
    gchar *strWartosc[4];
    long nMaksDzien;
    gchar bufor0[88];
    gchar bufor1[88];
    gchar bufor2[88];
    typDaneWykresu daneWykresu;

    /* --- tworzymy tabelę --- */
    strWartosc[0] = bufor0;
    strWartosc[1] = bufor1;
    strWartosc[2] = bufor2;

    /* --- NULL - tu będzie umieszczona grafika. --- */
    strWartosc[3] = NULL;

    /* --- Jeśli c-lista jeszcze nie została utworzona... --- */
```

```
    if (clistaDni == NULL) {

      /* --- Tworzymy c-listę --- */
      clistaDni = gtk_clist_new_with_titles (4, szTytulyDni);

      /* --- Wyświetlamy nagłówki kolumn --- */
      gtk_clist_column_titles_show (GTK_CLIST (clistaDni));

      /* --- Ustawiamy szerokości kolumn --- */
      gtk_clist_set_column_width (GTK_CLIST (clistaDni), 0, 80);
      gtk_clist_set_column_width (GTK_CLIST (clistaDni), 1, 80);
      gtk_clist_set_column_width (GTK_CLIST (clistaDni), 2, 80);

      /* --- Ustawiamy justowanie kolumn --- */
      gtk_clist_set_column_justification (GTK_CLIST (clistaDni),
                                          0, GTK_JUSTIFY_RIGHT);
      gtk_clist_set_column_justification (GTK_CLIST (clistaDni),
                                          1, GTK_JUSTIFY_RIGHT);
      gtk_clist_set_column_justification (GTK_CLIST (clistaDni),
                                          2, GTK_JUSTIFY_RIGHT);

      /* --- Dodajemy c-listę do strony notatnika --- */
      gtk_container_add (GTK_CONTAINER (stronaDni), clistaDni);
    }

    /* --- Ustawiamy maksymalny rozmiar na zero --- */
    nMaksDzien = 0;

    /*
     * --- Obchodzimy drzewo i wyświetlamy informacje tekstowe,
     *     zapamiętując maksymalną wartość, aby można było
     *     wyświetlić wykres.
     */
    g_tree_traverse (drzewoDat, PokazDaneDnia, G_IN_ORDER,
&nMaksDzien);

    /* --- Dane potrzebne do wyświetlenia wykresu --- */
    daneWykresu.nMaksRozmiar = nMaksDzien;
    daneWykresu.kontrolka = clistaDni;
    daneWykresu.rzad = 0;

    /* --- Ponownie obchodzimy drzewo i wyświetlamy wykresy --- */
    g_tree_traverse (drzewoDat, PokazWykres, G_IN_ORDER,
                     &daneWykresu);
```

```
    /* --- Teraz pokazujemy c-listę --- */
    gtk_widget_show_all (GTK_WIDGET (clistaDni));
}

/*
 * PokazWykresUzytk
 *
 * Wyświetla wykres dla każdego użytkownika.
 * Wywoływana podczas obchodu drzewa - jest to funkcja zwrotna,
 * operująca na danych z drzewa.
 *
 * wartosc - zawiera dane o działaniach tego użytkownika
 * dane - zawiera informacje o wykresie, w tym o kontrolce
 *        i wartości maksymalnej.
 */
gint PokazWykresUzytk (gpointer klucz, gpointer wartosc, gpointer dane)
{
    int ix;
    typDaneWykresu *daneWykresu = (typDaneWykresu *) dane;
    typStat *daneStat = (typStat *) wartosc;

    /* --- Jak duży powinien być wykres? --- */
    ix = (long) (((double) daneStat->nRozmiar * LICZBA_WYKRESOW-1) /
                    daneWykresu->nMaksRozmiar);

    /* --- Wybieramy piksmapę o odpowiednim rozmiarze --- */
    gtk_clist_set_pixmap (GTK_CLIST (daneWykresu->kontrolka),
                    daneWykresu->rzad, 3, piksmapyWykresow[ix],
                    maski[ix]);

    /* --- Przechodzimy do następnego rzędu. --- */
    daneWykresu->rzad++;

    return (0);
}

/*
 * WypelnijUzytk
 *
 * Wypełnia stronę notatnika danymi o ruchu według użytkowników.
 * Dokonujemy tego w dwóch krokach - najpierw wyświetlamy dane
 * tekstowe i obliczamy maksymalną wartość, a następnie rysujemy
 * wykres na podstawie maksymalnej wartości.
```

```
 */
void WypelnijUzytk ()
{
   gchar *strWartosc[4];
   gchar bufor0[88];
   gchar bufor1[88];
   gchar bufor2[88];
   long nMaks;
   typDaneWykresu daneWykresu;

   /* --- Buforowane wartości --- */
   strWartosc[0] = bufor0;
   strWartosc[1] = bufor1;
   strWartosc[2] = bufor2;
   strWartosc[3] = NULL;

   /* --- Jeśli nie ma jeszcze c-listy użytkowników... --- */
   if (clistaUzytk == NULL) {

      /* --- Tworzymy c-listę z nagłówkami --- */
      clistaUzytk = gtk_clist_new_with_titles (4, szTytulyUzytk);

      /* --- Pokazujemy nagłówki --- */
      gtk_clist_column_titles_show (GTK_CLIST (clistaUzytk));

      /* --- Ustawiamy szerokość kolumn. --- */
      gtk_clist_set_column_width (GTK_CLIST (clistaUzytk), 0, 80);
      gtk_clist_set_column_width (GTK_CLIST (clistaUzytk), 1, 80);
      gtk_clist_set_column_width (GTK_CLIST (clistaUzytk), 2, 80);

      /* --- Justujemy kolumny --- */
      gtk_clist_set_column_justification (GTK_CLIST (clistaUzytk),
                                          0, GTK_JUSTIFY_LEFT);
      gtk_clist_set_column_justification (GTK_CLIST (clistaUzytk),
                                          1, GTK_JUSTIFY_RIGHT);
      gtk_clist_set_column_justification (GTK_CLIST (clistaUzytk),
                                          2, GTK_JUSTIFY_RIGHT);

      /* --- Dodajemy c-listę do strony notatnika. --- */
      gtk_container_add (GTK_CONTAINER (stronaUzytk), clistaUzytk);
   }

   /* --- Obchodzimy drzewo, aby wyświetlić wartości tekstowe
```

```
        i znaleźć wartość maksymalną --- */
    nMaks = 0;
    g_tree_traverse (drzewoUzytk, PokazDaneUzytk, G_IN_ORDER, &nMaks);

    /* -- Wypełniamy strukturę dla "graficznego" obchodu drzewa --- */
    daneWykresu.nMaksRozmiar = nMaks;
    daneWykresu.kontrolka = clistaUzytk;
    daneWykresu.rzad = 0;

    /* --- Wyświetlamy wykresy --- */
    g_tree_traverse (drzewoUzytk, PokazWykresUzytk, G_IN_ORDER,
&daneWykresu);

    gtk_widget_show_all (GTK_WIDGET (clistaUzytk));
}
```

Podsumowanie

Teraz możemy korzystać z bardziej skomplikowanych kontrolek, aby tworzyć bardziej interesujące aplikacje. Kontrolka GtkTree wyświetla informacje w postaci przypominającej drzewo. Można oglądać wszystkie gałęzie drzewa, albo rozwinąć tylko te, które zawierają żądane informacje. Kontrolka GtkNotebook przydaje się do wyświetlania wielu stron, które mogą być przełączane przez użytkownika albo samą aplikację. Możemy ukryć zakładki notatnika i sterować wyświetlaniem stron z wnętrza programu. Kontrolka GtkClist znakomicie nadaje się do wyświetlania wielu kolumn danych, zazwyczaj pochodzących z bazy danych. W kontrolce można mieszać grafikę i tekst, co uprzyjemnia przeglądanie informacji.