Rozdział 6

Dalsze kontrolki: ramki, tekst, okna dialogowe, okno wyboru pliku, pasek postępów

Proste kontrolki to dopiero początek GTK+. Pakiet ten udostępnia o wiele więcej kontrolek, pozwalających programistom na tworzenie kompletnych aplikacji. Niniejszy rozdział omawia niektóre spośród nich, w tym kontrolkę ramki, okna dialogowego, paska postępów, oraz dysponującą ogromnymi możliwościami kontrolkę tekstu.

Ramki

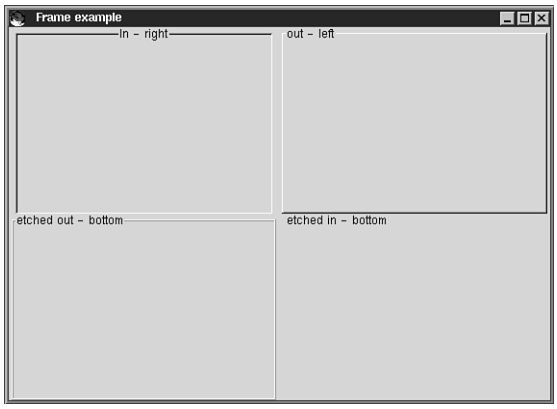
Ramki są pojemnikami, zazwyczaj otoczonymi obramowaniem i zaopatrzonymi w tytuł, które służą do grupowania kontrolek. Ramki pomagają użytkownikom w rozróżnieniu odrębnych pól na ekranie. Ramkę tworzy się przy pomocy funkcji gtk_frame_new, która tworzy ramkę domyślną. Typowa sekwencja tworzenia ramki wygląda następująco:

```
/* --- tworzymy ramkę --- */
ramka = gtk_frame_new ("Tytuł");

/* --- tutaj ustawiamy jej atrybuty --- */

/* --- uwidaczniamy ramkę --- */
gtk_widget_show (ramka);

/* --- dodajemy ją do pojemnika albo pola --- */
gtk_container_add (okno, ramka);
```

Po stworzeniu ramki możemy wykorzystać funkcję gtk_frame_set_label, aby zmienić etykietę ramki:

```
gtk_frame_set_label (ramka, "Hej, nowy tytuł!");
```

Właściwości brzegu ramki można ustawić przy pomocy funkcji gtk_frame_set_shadow. Właściwości ramki określają, w jaki sposób ramka będzie wyglądała na ekranie. Ustawienie ramki na GTK_SHADOW_IN sprawia wrażenie, że cały obszar ramki jest wklęsły, natomiast GTK_SHADOW_OUT sprawia wrażenie, że ramka jest wyniesiona ponad inne kontrolki. Efekt ten osiąga się przy pomocy odpowiedniego doboru kolorów. Częściej używane wartości GTK_SHADOW_ETCHED_OUT i GTK_SHADOW_ETCHED_IN rysują tylko brzeg ramki, a GTK_SHADOW_ ETCHED_NONE w ogóle nie rysuje brzegu. Różne typy ramek przedstawia rysunek 6.1.



Rysunek 6.1. Ramki.

Tytuł może pojawić się w ramce w środku, po lewej albo po prawej stronie. Do zmiany justowania służy funkcja gtk_frame_set_label_align. Parametry, których używaliśmy do wyrównywania tekstu etykiet, mają zastosowanie także tutaj. Wyrównanie do lewej określamy przez 0, do środka przez .5, a do prawej przez 1. Poniższe przykłady ilustrują użycie parametrów:

```
/* --- ustawiamy etykietę ramki po lewej stronie --- */
gtk_frame_set_label_align (GTK_FRAME (ramka), 0, 0);

/* --- ustawiamy etykietę ramki w środku --- */
gtk_frame_set_label_align (GTK_FRAME (ramka), .5, 0);

/* --- ustawiamy etykietę ramki po prawej stronie --- */
gtk_frame_set_label_align (GTK_FRAME (ramka), 1, 0);
```

Po utworzeniu ramki można dodawać do niej kontrolki. Ramkę należy traktować jako zwykła kontrolkę pojemnika, choć zaopatrzoną w obramowanie i tytuł. Jeśli chcemy dodać do niej więcej niż jedną kontrolkę, musimy użyć pola albo tabeli pakującej.

Kontrolka tekstu

Kontrolka tekstu (patrz rysunek 6.2) przypomina kontrolkę wpisu (patrz rozdział 4), ale umożliwia wprowadzenie wielu linii tekstu, co predestynuje ją do wykorzystania w edytorze; zajmiemy się tym zresztą w dalszym rozdziale. Kontrolka tekstu jest znacznie bardziej skomplikowana, niż mniejsze kontrolki, ponieważ posiada dużo więcej ustawialnych atrybutów.

Kontrolka tekstu obsługuje pewną liczbę skrótów klawiszowych, w tym standardowe skróty służące do wycinania, kopiowania i wklejania tekstu. Skróty te można stosować również w opisanej wcześniej kontrolce wpisu, ponieważ obie wywodzą się od GtkEditable. Oto niektóre skróty klawiszowe:

- CTRL+A - Początek linii
- CTRL+E - Koniec linii
- CTRL+N - Następna linia
- CTRL+P - Poprzednia linia
- CTRL+B - Jeden znak do tyłu
- CTRL+F - Jeden znak do przodu
- ALT+B - Jedno słowo do tyłu
- ALT+F - Jedno słowo do przodu

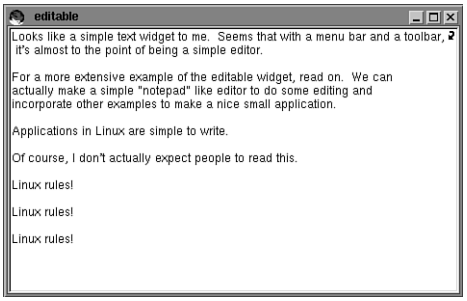
Skróty edycyjne:

- CTRL+H - Usuwa znak przed kursorem (Backspace)
- CTRL+D - Usuwa znak za kursorem (Delete)
- CTRL+W - Usuwa słowo przed kursorem
- ALT+D - Usuwa słowo za kursorem
- CTRL+K - Usuwa znaki do końca linii
- CTRL+U - Usuwa linię

Skróty operujące na zaznaczeniach:

- CTRL+X - Wycina tekst i umieszcza go w schowku
- CTRL+C - Kopiuje tekst do schowka
- CTRL+V - Wkleja tekst ze schowka

Kontrolka tekstu jest niemal gotowym edytorem. Stworzenie przy jej pomocy pełnej aplikacji edytora nie wymaga wiele pracy, o czym można przekonać się w rozdziale 8, „Tworzenie prostego edytora tekstu".



Rysunek 6.2. Kontrolka tekstu.

Tworzenie kontrolki tekstu

Kontrolkę tekstu tworzymy przy pomocy funkcji gtk_text_new, przekazując do niej wartości typu GtkAdjustment, która określają parametry poziomej i pionowej regulacji. Zazwyczaj można tu użyć NULL, co spowoduje zastosowanie domyślnej regulacji. Kontrolkę tekstu tworzymy zatem w następujący sposób:

```
tekst = gtk_text_new (NULL, NULL);
```

Kontrolkę tekstu można ustawić w tryb „tylko do odczytu" przy pomocy funkcji gtk_text_set_editable. Funkcja ta ustawia pole na „tylko do odczytu", jeśli przekażemy jej parametr FALSE; w innym przypadku zawartość kontrolki będzie można edytować. Domyślnie edycja jest dozwolona.

```
/* --- pozwalamy na edytowanie kontrolki tekstu --- */
gtk_text_set_editable (GTK_TEXT (tekst), TRUE);

/* --- przełączamy kontrolkę w tryb "tylko do odczytu" --- */
gtk_text_set_editable (GTK_TEXT (tekst), FALSE);
```

Można ustawić pozycję kursora przy pomocy funkcji gtk_text_set_point. Funkcja ta przyjmuje parametr w postaci indeksu, który określa, gdzie

ma znaleźć się kursor. Indeks oznacza numer znaku w tekście, na przykład:

```
/* --- ustawiamy kursor za dziesiątym znakiem --- */
gtk_text_set_point (GTK_TEXT (tekst), 10);
```

Można także sprawdzić pozycję kursora przy pomocy funkcji gtk_text_get_point. Zwraca ona indeks, który określa przesunięcie kursora względem początku tekstu.

```
/* --- sprawdzamy, gdzie jest kursor --- */
gtk_text_get_point (GTK_TEXT (tekst));
```

Funkcja gtk_text_get_length służy do sprawdzania, ile znaków znajduje się w kontrolce tekstu, natomiast funkcja gtk_editable_get_chars służy do pobierania tekstu, znajdującego się w kontrolce. Na przykład, aby pobrać wszystkie znaki w kontrolce tekstu, można użyć następującego kodu:

```
/* --- pobieramy liczbę znaków w kontrolce --- */
nDlugosc = gtk_text_get_length (GTK_TEXT (tekst);

/* --- pobieramy tekst z kontrolki --- */
bufor = gtk_editable_get_chars (GTK_EDITABLE (tekst),
                        (gint) 0,
                        (gint) nDlugosc);
```

Zamiast obliczać długość tekstu w kontrolce, można skorzystać z wartości -1, która wskazuje na ostatni znak.

```
/* --- pobieramy tekst z kontrolki --- */
bufor = gtk_editable_get_chars (GTK_EDITABLE (tekst),
                        (gint) 0,
                        (gint) -1);
```

Bufor zwracany przez funkcję gtk_editable_get_chars jest nowym blokiem pamięci, przydzielanym specjalnie dla wartości powrotnej. Zajmowanie tej pamięci aż do momentu zakończenia aplikacji sprzeciwia się regułom sztuki programowania (ktoś mógłby wówczas pomyśleć, że jest to aplikacja Microsoftu!). Należy więc użyć funkcji g_free, kiedy pamięć nie jest już potrzebna.

```
/* --- pamięć jest już niepotrzebna, zwalniamy ją --- */
g_free (bufor);
```

Można pobrać pojedynczy znak z kontrolki tekstu przy pomocy makra GTK_TEXT_INDEX. Przekazujemy mu indeks i otrzymujemy znak z określonej pozycji.

```
/* --- pobieramy sto pierwszy znak z kontrolki --- */
znak = GTK_TEXT_INDEX (GTK_TEXT (tekst), 101);
```

Można także wstawić tekst do kontrolki przy pomocy funkcji gtk_text_insert_text. Wstawia ona znaki w bieżącej pozycji kursora. Jeśli tekst ma być wstawiony przy użyciu określonej czcionki lub koloru, należy użyć funkcji gtk_text_insert wraz z informacjami o kolorze i czcionce. Przekazanie NULL spowoduje użycie bieżących parametrów tekstu.

```
/* --- dodajemy tekst bez określania czcionki/koloru --- */
gtk_text_insert (GTK_TEXT (tekst), NULL, NULL, NULL,
        bufor, strlen (bufor));

/* --- dodajemy tekst z określoną czcionką/kolorem --- */
gtk_text_insert (GTK_TEXT (tekst), czcionka,
        kolor_tekstu, kolor_tla,
        bufor, strlen (bufor));

/* --- dodajemy tekst z bieżącą czcionką/kolorem --- */
gtk_text_insert_text (GTK_TEXT (tekst), bufor,
        strlen (bufor), &nPozycja);
```

Usuwanie tekstu odbywa się względem bieżącej pozycji kursora. Usuwamy tekst zaczynając od bieżącej pozycji kursora, a kończąc na określonym punkcie przed lub za kursorem. Funkcja gtk_text_forward_delete usuwa znaki znajdujące się za kursorem, a funkcja gtk_text_backward_delete przed kursorem.

```
/* --- usuwamy dziesięć znaków za kursorem --- */
bSukces = gtk_text_forward_delete (GTK_TEXT (tekst), (guint) 10);

/* --- usuwamy dziesięć znaków przed kursorem --- */
bSukces = gtk_text_backward_delete (GTK_TEXT (tekst), (guint) 10);
```

Zwracaną wartością jest TRUE, jeśli usunięcie znaków się powiodło, a FALSE w przeciwnym przypadku.

Wstawianie i usuwanie tekstu

Wstawianie i usuwanie tekstu może być wolne, jeśli operujemy na dużych ilościach tekstu. Kontrolka jest przerysowywana po każdym wstawieniu lub usunięciu tekstu, co nie zawsze jest pożądane, zwłaszcza jeśli naraz wstawiamy lub usuwamy kilka łańcuchów. Czasem lepszym rozwiązaniem jest wstrzymanie rysowania kontrolki na czas operacji wsta-

wiania i usuwania tekstu. Można wstrzymać automatyczne uaktualnianie obrazu, wywołując funkcję gtk_text_freeze, i ponownie je uruchomić funkcją gtk_text_thaw. Funkcję gtk_text_freeze wywołujemy przed modyfikacją tekstu w kontrolce. Kiedy dokonamy żądanych modyfikacji, używamy funkcji gtk_text_thaw aby „odmrozić" kontrolkę. Spowoduje to jej przerysowanie, o ile dokonano zmian, kiedy była zamrożona. Odpowiedni kod wygląda następująco:

```
/* --- Zamrażamy kontrolkę --- */
gtk_text_freeze (GTK_TEXT (tekst));

/* --- ...tutaj modyfikujemy tekst kontrolki... --- */

/* --- Odmrażamy kontrolkę --- */
gtk_text_thaw (GTK_TEXT (tekst));
```

Jeśli nie użyjemy funkcji gtk_text_freeze i gtk_text_thaw podczas wstawiania lub usuwania dużej ilości tekstu, kontrolka będzie przerysowywana po każdej operacji. Zamiast sekwencji [wstaw tekst][wstaw tekst][wstaw tekst][przerysuj] aplikacja wykona więc sekwencję [wstaw tekst][przerysuj] [wstaw tekst][przerysuj][wstaw tekst][przerysuj]. Dwa dodatkowe przerysowania są niepotrzebne, ponieważ kontrolka i tak zostanie poddana dalszym modyfikacjom, które będą wymagały jej uaktualnienia. Lepiej jest więc poinformować kontrolkę, aby zaniechała aktualizacji, zanim nie dobiegną końca wszystkie modyfikacje. Sytuacja taka może mieć miejsce na przykład podczas operacji szukania i zamiany w edytorze. Korzystniej jest zamrozić ekran podczas dokonywania zamian i wyświetlić dopiero ostateczny rezultat, niż spowalniać całą operację przez wielokrotne odświeżanie kontrolki po każdej zmianie tekstu.

Kontrolki tekstu używane w różnego rodzaju edytorach muszą obsługiwać opcje wycinania, kopiowania i wklejania. Kontrolka tekstu automatycznie obsługuje wycinanie (CTRL+X), kopiowanie (CTRL+C) i wklejanie (CTRL+V). Wiele osób lubi jednak podczas kopiowania czy wklejania korzystać z myszy; możemy zaimplementować te operacje przy pomocy odpowiednich funkcji.

Funkcja gtk_editable_copy_clipboard kopiuje zaznaczony tekst i umieszcza go w schowku. Tekst pozostaje tam, dopóki nie zostanie przywołany lub nadpisany.

```
/* --- kopiujemy zaznaczony tekst i umieszczamy go w schowku --- */
gtk_editable_copy_clipboard (GTK_EDITABLE (tekst),
GDK_CURRENT_TIME);
```

Funkcja gtk_editable_cut_clipboard usuwa zaznaczony tekst z kontrolki i przenosi go do schowka.

```
/* --- usuwamy zaznaczony tekst i umieszczamy go w schowku --- */
gtk_editable_cut_clipboard (GTK_EDITABLE (tekst),
GDK_CURRENT_TIME);
```

Funkcja gtk_editable_paste_clipboard pobiera tekst ze schowka i wstawia go w bieżącej pozycji kursora.

```
/* --- kopiujemy tekst ze schowka i wstawiamy go do kontrolki --- */
/* --- w bieżącej pozycji kursora --- */
gtk_editable_paste_clipboard (GTK_EDITABLE (tekst),
GDK_CURRENT_TIME);
```

Paski przewijania

Paski przewijania są istotnym elementem kontrolki tekstu. Przydają się w sytuacji, kiedy użytkownik wprowadzi więcej tekstu, niż może zmieścić się na ekranie, co - o dziwo - zdarza się bardzo często. Kontrolka może pomieścić więcej tekstu, niż jest w stanie wyświetlić. Użytkownik może zechcieć przewinąć dane przy pomocy myszy, ale bez pasków przewijania raczej nie będzie to możliwe. Kontrolka tekstu będzie znacznie użyteczniejsza, jeśli wyposażymy ją w paski przewijania, które pozwolą użytkownikowi przeglądać tekst przy pomocy myszy.

Tworzenie pasków przewijania i dołączanie ich do kontrolki tekstu jest bardzo łatwe. Poniższy kod tworzy kontrolkę tekstu wewnątrz tabeli i dodaje paski przewijania do pola edycyjnego. Tworzymy tabelę pakującą 2 x 2, aby umieścić w niej kontrolkę tekstu z paskiem przewijania pionowego po jej prawej stronie, a paskiem przewijania poziomego na spodzie. Kontrolka tekstu znajduje się w lewym górnym rogu (0-1, 0-1) tabeli pakującej, pasek pionowy w prawym górnym rogu (1-2, 0-1), a pasek poziomy w lewym dolnym rogu (0-1, 1-2).

```
/* --- tworzymy kontrolkę --- */
tabela = gtk_table_new (2, 2, FALSE);

/* --- uwidaczniamy ją --- */
gtk_widget_show (tabela);

/* --- tworzymy kontrolkę --- */
tekst = gtk_text_new (NULL, NULL);
/* --- dodajemy kontrolkę do tabeli pakującej --- */
gtk_table_attach (GTK_TABLE (tabela), tekst, 0, 1, 0, 1,
```

```
        GTK_EXPAND | GTK_SHRINK | GTK_FILL,
        GTK_EXPAND | GTK_SHRINK | GTK_FILL, 0, 0);

    /* --- uwidaczniamy ją --- */
    gtk_widget_show (tekst);
```

Po umieszczeniu kontrolki tekstu w tabeli pakującej możemy dodać do niej paski przewijania. Powinny one zostać dodane do struktury, opisującej kontrolkę tekstu. W strukturze tej znajdują się pola hadj i vadj, które są wskaźnikami typu GtkAdjustment. Aby utworzyć pasek przewijania i związać z nim wskaźnik GtkAdjustment kontrolki tekstu, należy po prostu przekazać do niego ten wskaźnik w momencie tworzenia.

```
    /* ------------------------------------------------- */
    /* --- tworzymy poziomy pasek przewijania --- */
    /* ------------------------------------------------- */
    xpasek = gtk_hscrollbar_new{xe " gtk_hscrollbar_new, funkcja"}{xe "funkcje:
    gtk_hscrollbar_new "} (GTK_TEXT (tekst)->hadj);

    /* --- umieszczamy poziomy pasek pod kontrolką tekstu --- */
    gtk_table_attach (GTK_TABLE (tabela), xpasek, 0, 1, 1, 2,
          GTK_EXPAND | GTK_SHRINK | GTK_FILL, GTK_FILL, 0, 0);

    /* --- Wyświetlamy kontrolkę --- */
    gtk_widget_show (xpasek);

    /* ------------------------------------------------- */
    /* --- tworzymy pionowy pasek przewijania --- */
    /* ------------------------------------------------- */
    ypasek = gtk_vscrollbar_new{xe " gtk_vscrollbar_new, funkcja"}{xe "funkcje:
    gtk_vscrollbar_new "} (GTK_TEXT (tekst)->vadj);

    /* --- umieszczamy pionowy pasek prawej stronie tekstu --- */
    gtk_table_attach (GTK_TABLE (tabela), ypasek, 1, 2, 0, 1,
          GTK_FILL, GTK_EXPAND | GTK_SHRINK | GTK_FILL, 0, 0);

    /* --- Wyświetlamy kontrolkę --- */
    gtk_widget_show (ypasek);
```

Jeśli poziomy pasek przewijania nie jest potrzebny, możemy po prostu utworzyć pionowe pole pakujące i umieścić pionowy pasek przewijania obok kontrolki tekstu. Ilustruje to następujący kod:

```
    /* --- tworzymy kontrolkę --- */
    tekst = gtk_text_new (NULL, NULL);
```

```
/* --- dodajemy kontrolkę tekstową do pola pakującego --- */
gtk_box_pack_start (GTK_BOX (ypole), tekst, FALSE, FALSE, 0);
/* --- tworzymy pionowy pasek przewijania --- */
ypasek = gtk_vscrollbar_new (GTK_TEXT (tekst)->vadj);

/* --- dodajemy pasek po prawej stronie kontrolki tekstu --- */
gtk_box_pack_start (GTK_BOX (ypole), ypasek, FALSE, FALSE, 0);

/* --- uwidaczniamy pasek --- */
gtk_widget_show (ypasek);
```

Okna dialogowe

Okna dialogowe służą do wyświetlania informacji („Twój dysk jest bliski zapełnienia"), zadawania pytań użytkownikowi („Czy na pewno chcesz to zrobić?") albo uzyskiwania innych informacji od użytkownika („Pod jaką nazwą chcesz zapisać plik?").

Okna dialogowe tworzy się przy użyciu funkcji gtk_dialog_new, która tworzy puste okno dialogowe, bez przycisków i tekstu. Utworzone okno dialogowe posiada dwa pola pakujące (vbox, action_area), do których można dodawać inne kontrolki. Pole vbox służy do umieszczania etykiet albo innych informacji, a przyciski umieszcza się w action_area. Puste okno dialogowe nie przyda się do niczego, dopóki nie umieścimy w nim kontrolki tekstu i jednego lub dwóch przycisków. Musi też wyświetlać jakąś informację dla użytkownika. Najłatwiejszym sposobem uczynienia okna użytecznym jest wyświetlenie jakiejś informacji i przycisku „OK". Możemy utworzyć etykietę i dodać ją do pionowego pola pakującego vbox w następujący sposób:

```
/* --- tworzymy etykietę --- */
etykieta = gtk_label_new (szKomunikat);

/* --- dajemy jej trochę przestrzeni życiowej --- */
gtk_misc_set_padding (GTK_MISC (etykieta), 10, 10);

/* --- dodajemy ją to pionowego pola pakującego --- */
gtk_box_pack_start (GTK_BOX (GTK_DIALOG (okno_dialogowe)->vbox),
                    etykieta, TRUE, TRUE, 0);

/* --- uwidaczniamy etykietę --- */
gtk_widget_show (etykieta);
```

Okno wygląda miło, ale nadal nie dostarcza żadnych informacji od użytkownika. Musimy więc dodać przycisk do obszaru action_area. Kiedy dodajemy przycisk do okna dialogowego, zazwyczaj dobrze jest ustawić jego znacznik GTK_CAN_DEFAULT{xe "GTK_CAN_DEFAULT"}. Znacznik ten określa, że przycisk może być domyślną kontrolką w oknie dialogowym, to znaczy użytkownik może nacisnąć ENTER zamiast klikać na przycisku. Funkcja gtk_widget_ grab_default umożliwia ustawianie domyślnej kontrolki. W naszym prostym przykładzie mamy tylko jeden przycisk; możemy więc go uczynić domyślnym przyciskiem dla okna dialogowego, aby użytkownik mógł wybrać go przy pomocy klawisza ENTER. Domyślny przycisk posiada szerokie obramowanie. Dodanie przycisku „OK" jest nadzwyczaj proste:

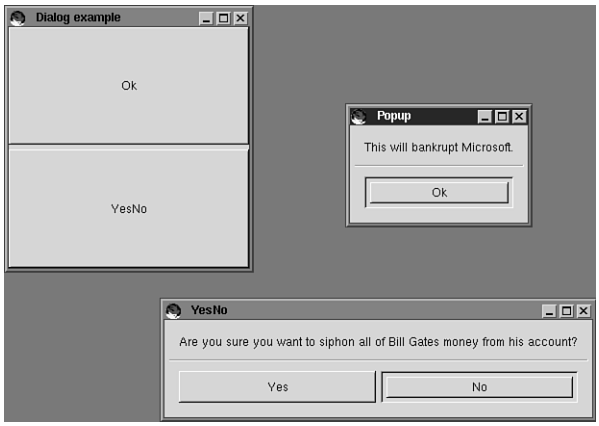
```
/* --- tworzymy przycisk --- */
przycisk = gtk_button_new_with_label ("OK");

/* --- pozwalamy na uczynienie przycisku domyślnym --- */
GTK_WIDGET_SET_FLAGS (przycisk, GTK_CAN_DEFAULT);

/* --- dodajemy przycisk do obszaru action_area --- */
gtk_box_pack_start (GTK_BOX(GTK_DIALOG (okno_dialogowe)->action_area),
        przycisk, TRUE, TRUE, 0);

/* --- przesuwamy ognisko na przycisk --- */
gtk_widget_grab_default (przycisk);

/* --- uwidaczniamy go --- */
gtk_widget_show (przycisk);

/* --- oczekujemy na kliknięcie przycisku --- */
gtk_signal_connect (GTK_OBJECT (przycisk), "clicked",
        GTK_SIGNAL_FUNC (kliknietoOK),
        &wartosc_zwrotna);
```

Linux obsługuje kilka typów okien dialogowych. Niektóre z nich są oknami modalnymi, inne niemodalnymi. Okna modalne ogniskują sterowanie w aplikacji na oknie dialogowym tak, że użytkownik nie może używać innych okien, dopóki nie zamknie okna dialogowego. Program może przetwarzać tylko te zdarzenia, które pochodzą z okna dialogowego. Aplikacja w Linuksie może wyświetlić modalne okno dialogowe aby upewnić się, że użytkownik zapoznał się z jakąś informacją, bądź też wtedy, kiedy informacje uzyskiwane z okna są niezbędne do dalszej pracy programu. Często modalne jest okno wyboru pliku - ponieważ użytkownik chce wczytać plik, nie ma sensu robić niczego innego, dopóki

użytkownik nie wybierze pliku i nie naciśnie „OK", albo nie zrezygnuje z wczytywania pliku, naciskając „Anuluj". Kilka okien dialogowych w działaniu przedstawia rysunek 6.3.

Niemodalne okna dialogowe nie posiadają takich właściwości. Można na przykład wyświetlić w oknie dialogowym tablicę kolorów, która pozwala użytkownikowi na wybór koloru. W tym samym czasie mogą być aktywne inne okna, bez potrzeby ograniczania ogniska do okna wyboru koloru.

Zamieszczona poniżej funkcja OknoDialogowe tworzy proste okno dialogowe i czeka, aż użytkownik wciśnie przycisk „OK". Tworzymy okno dialogowe, etykietę dla wyświetlanego komunikatu (umieszczając ją w oknie dialogowym), wreszcie tworzymy przycisk „OK" i również umieszczamy go w oknie. Przycisk posiada przypisaną funkcję obsługi zdarzenia, która zamknie okno po naciśnięciu przycisku. Okno to jest modalne, ponieważ wywołuje funkcję gtk_grab_add, która zapewnia, że zdarzenia mogą zachodzić tylko w tym oknie.



Rysunek 6.3. Okna dialogowe.

```
/*
 * OknoDialogowe
 *
 * Wyświetla okno dialogowe z komunikatem i przyciskiem "OK"
 */
void OknoDialogowe (char *szKomunikat)
{
   static GtkWidget *etykieta;
   GtkWidget *przycisk;
```

```
    GtkWidget *okno_dialogowe;

    /* --- tworzymy okno dialogowe --- */
    okno_dialogowe = gtk_dialog_new ();

    /* --- przechwytujemy sygnał zamykania okna, --- */
    /* --- aby zwolnić ognisko --- */
    gtk_signal_connect (GTK_OBJECT (okno_dialogowe), "destroy",
                        GTK_SIGNAL_FUNC (ZamkniecieOkna),
                        &okno_dialogowe);

    /* --- dodajemy tytuł do okna --- */
    gtk_window_set_title (GTK_WINDOW (okno_dialogowe), "Okno
dialogowe");

    /* --- tworzymy niewielkie obramowanie --- */
    gtk_container_border_width (GTK_CONTAINER (ono_dialogowe), 5);

    /* ------------------------------- */
    /* --- tworzymy komunikat --- */
    /* ------------------------------- */

    /* --- umieszczamy komunikat w etykiecie --- */
    etykieta = gtk_label_new (szKomunikat);

    /* --- zostawiamy trochę miejsca wokół etykiety --- */
    gtk_misc_set_padding (GTK_MISC (etykieta), 10, 10);

    /* --- dodajemy etykietę do okna dialogowego --- */
    gtk_box_pack_start (GTK_BOX (GTK_DIALOG (okno_dialogowe)->vbox),
                        etykieta, TRUE, TRUE, 0);

    /* --- uwidaczniamy etykietę --- */
    gtk_widget_show (etykieta);

    /* ----------------------- */
    /* --- przycisk "OK" --- */
    /* ----------------------- */

    /* --- tworzymy przycisk "OK" --- */
    przycisk = gtk_button_new_with_label ("OK");

    /* --- musimy zamknąć okno po naciśnięciu "OK" --- */
    gtk_signal_connect (GTK_OBJECT (przycisk), "clicked",
                        GTK_SIGNAL_FUNC (ZamknijDialog),
```

```
                        okno_dialogowe);

    /* --- pozwalamy, aby przycisk mógł być domyślnym --- */
    GTK_WIDGET_SET_FLAGS (przycisk, GTK_CAN_DEFAULT);

    /* --- dodajemy przycisk do okna dialogowego --- */
    gtk_box_pack_start (GTK_BOX (
                        GTK_DIALOG (okno_dialogowe)->action_area),
                        przycisk, TRUE, TRUE, 0);

    /* --- czynimy przycisk domyślnym --- */
    gtk_widget_grab_default (przycisk);

    /* --- uwidaczniamy przycisk --- */
    gtk_widget_show (przycisk);

    /* --- uwidaczniamy okno dialogowe --- */
    gtk_widget_show (okno_dialogowe);

    gtk_grab_add  (okno_dialogowe);
}
```

Funkcja ZamknijDialog, przypisana do przycisku OK, wywołuje funkcję gtk_grab_remove, aby zakończyć tryb modalny okna dialogowego, a następnie zamyka okno. Do funkcji przekazywany jest w polu danych wskaźnik GtkWidget okna dialogowego, ponieważ funkcja zwrotna nie wie, z którego okna dialogowego pochodzi sygnał (wie tylko, że sygnał pochodzi od jakiejś kontrolki). Przekazywanie okna dialogowego do funkcji zwrotnej ustawiliśmy w wywołaniu gtk_signal_connect; dzięki temu będziemy mogli zamknąć okno.

```
/*
 * ZamknijOkno
 *
 * Zamyka okno dialogowe. Otrzymuje jako parametr uchwyt do okna.
 */
void ZamknijOkno (GtkWidget *kontrolka, gpointer dane)
{
    /* --- zamykamy okno --- */
    gtk_widget_destroy (GTK_WIDGET (dane));
}
```

Podczas tworzenia okna dialogowego użyliśmy funkcji gtk_grab_add, aby uczynić je modalnym. Zamykając okno należy wywołać funkcję gtk_grab_

remove, aby usunąć modalność okna. Sygnał "destroy" nadaje się do tego idealnie, ponieważ wskazuje, że okno jest zamykane.

```
/*
 * ZamkniecieOkna
 *
 * Wywoływana przed zamknięciem okna. Zwraca FALSE, aby pozwolić
 * na jego zamknięcie. Zwalnia przechwycenie sygnałów, które uczyniło
 * okno modalnym.
 */
void ZamkniecieOkna (GtkWidget *kontrolka, gpointer dane)
{
   gtk_grab_remove (GTK_WIDGET (kontrolka));
{
```

Przykłady okien dialogowych „Tak/Nie" albo „OK/Anuluj" są nieco bardziej skomplikowane. Wyświetlają one komunikat i umożliwiają użytkownikowi dokonanie wyboru. Zadają pytania typu „Czy na pewno chcesz to zrobić?", a wybór któregoś z przycisków powoduje odmienne reakcje programu. Możemy tworzyć okna dialogowe tego typu rozszerzając poprzedni przykład, poprzez przypisanie każdemu przyciskowi w oknie dialogowym innej procedury obsługi.

Zauważmy, że w kolejnym przykładzie zarówno dla przycisku „Tak", jak i „Nie" ustawiony jest znacznik GTK_CAN_DEFAULT. Dzięki temu obydwa przyciski mogą stać się domyślnymi. Jeśli klikniemy na dowolnym z nich i przytrzymamy przycisk myszy, wówczas zostanie on otoczony szerokim obramowaniem. Naciśnięcie w tym momencie klawisza ENTER jest równoznaczne z naciśnięciem tego przycisku. Okno dialogowe jest początkowo wyświetlane z domyślnym przyciskiem „Nie", co określamy w wywołaniu funkcji gtk_widget_grab_default. Zazwyczaj jako domyślne powinno się ustawiać działanie niedestrukcyjne; na przykład program, który formatuje dysk twardy i zadaje pytanie „Czy na pewno chcesz to zrobić?", powinien prawdopodobnie ustawić przycisk „Nie" jako domyślny.

Funkcja TakNie przyjmuje komunikat („Czy na pewno chcesz to zrobić?") i dwie funkcje. Funkcja FunkcjaTak powinna być wywoływana wtedy, kiedy użytkownik naciśnie przycisk „Tak", a FunkcjaNie w przeciwnym przypadku.

```
/*
 * TakNie
 *
 * Funkcja wyświetlająca okno Tak/Nie
```

```
 */
void TakNie (char *szKomunikat, void (*FunkcjaTak)(), void (*FunkcjaNie)())
{
   GtkWidget *etykieta;
   GtkWidget *przycisk;
   GtkWidget *okno_dialogowe;

   /* --- tworzymy okno dialogowe --- */
   okno_dialogowe = gtk_dialog_new ();

   /* --- Przechwytujemy sygnał "destroy" aby zwolnić blokadę --- */
   gtk_signal_connect (GTK_OBJECT (okno_dialogowe), "destroy",
                       GTK_SIGNAL_FUNC (ZamkniecieOkna),
                       &okno_dialogowe);

   /* --- ustawiamy tytuł --- */
   gtk_window_set_title (GTK_WINDOW (okno_dialogowe), "TakNie");

   /* --- dodajemy niewielkie obramowanie --- */
   gtk_container_border_width (GTK_CONTAINER (okno_dialogowe), 5);

   /* ---------------------------------- */
   /* --- tworzymy komunikat --- */
   /* ---------------------------------- */

   /* --- tworzymy etykietę z komunikatem --- */
   etykieta = gtk_label_new (szKomunikat);

   /* --- trochę miejsca na etykietę --- */
   gtk_misc_set_padding (GTK_MISC (etykieta), 10, 10);

   /* --- dodajemy etykietę do okna dialogowego --- */
   gtk_box_pack_start (GTK_BOX (GTK_DIALOG (okno_dialogowe)->vbox),
                       etykieta, TRUE, TRUE, 0);

   /* --- wyświetlamy etykietę --- */
   gtk_widget_show (etykieta);

   /* ----------------------- */
   /* --- Przycisk Tak --- */
   /* ----------------------- */

    /* --- tworzymy przycisk "tak" --- */
   przycisk = gtk_button_new_with_label ("Tak");
```

```
    gtk_signal_connect (GTK_OBJECT (przycisk), "clicked",
                        GTK_SIGNAL_FUNC (FunkcjaTak),
                        okno_dialogowe);

    GTK_WIDGET_SET_FLAGS (przycisk, GTK_CAN_DEFAULT);

    /* --- dodajemy przycisk do okna dialogowego --- */
    gtk_box_pack_start (GTK_BOX (
                         GTK_DIALOG (okno_dialogowe)->action_area),
                         przycisk, TRUE, TRUE, 0);

    /* --- uwidaczniamy przycisk --- */
    gtk_widget_show (przycisk);

    /* ---------------------- */
    /* --- Przycisk Nie --- */
    /* ---------------------- */

    /* --- tworzymy przycisk "nie" --- */
    przycisk = gtk_button_new_with_label ("Nie");

    gtk_signal_connect (GTK_OBJECT (przycisk), "clicked",
                        GTK_SIGNAL_FUNC (FunkcjaNie),
                        okno_dialogowe);

    /* --- Pozwalamy, aby przycisk "nie" był domyślnym --- */
    GTK_WIDGET_SET_FLAGS (przycisk, GTK_CAN_DEFAULT);

    /* --- dodajemy przycisk "nie" do okna dialogowego --- */
    gtk_box_pack_start (GTK_BOX (
                         GTK_DIALOG (okno_dialogowe)->action_area),
                         przycisk, TRUE, TRUE, 0);

    /* --- Czynimy przycisk "nie" domyślnym --- */
    gtk_widget_grab_default (przycisk);

    /* --- Uwidaczniamy przycisk --- */
    gtk_widget_show (przycisk);

    /* --- Wyświetlamy okno dialogowe --- */
    gtk_widget_show (okno_dialogowe);

    /* --- Teraz można używać tylko tego okna --- */
    gtk_grab_add (okno_dialogowe);
}
```

Funkcję TakNie można teraz wykorzystać w aplikacjach do tworzenia zapierających dech w piersiach okien dialogowych. Wywołanie funkcji wygląda następująco:

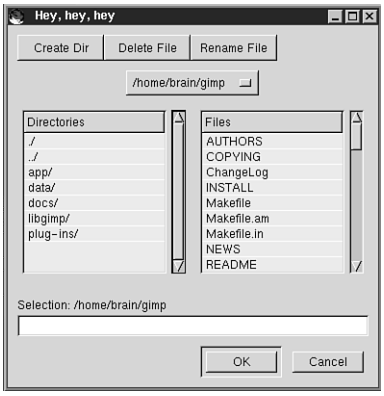
```
TakNie ("Czy na pewno chcesz pobrać "
    "wszystkie pieniądze z konta Billa Gatesa?",
    WyswietlTak, WyswietlNie);
```

Funkcje WyswietlTak i WyswietlNie powinny usuwać okno dialogowe, kiedy zakończą przetwarzanie. Funkcja TakNie przekazuje wskaźnik GtkPointer do okna dialogowego jako parametr gpointer funkcji zwrotnej dla sygnału "clicked". Funkcje WyswietlTak i WyswietlNie są tutaj niewielkie, ale nic nie stoi na przeszkodzie, aby były większe i wykonywały dowolne operacje:

```
/*
 * WyswietlTak
 *
 * Pokazuje, że wciśnięto przycisk "Tak"
 */
void WyswietlTak (GtkWidget *kontrolka, gpointer dane)
{
   /* --- Wyświetlamy komunikat --- */
   g_print ("Wyciągam $60 000 000 000,00 z konta\n");

/* --- zamykamy okno --- */
   gtk_widget_destroy (GTK_WIDGET (dane));
}

/*
 * WyswietlNie
 *
 * Pokazuje, że wciśnięto przycisk "Nie"
 */
void WyswietlNie (GtkWidget *kontrolka, gpointer dane)
{
   /* --- Wyświetlamy komunikat --- */
   g_print ("Pieniądze i tak nie dają szczęścia\n");

   /* --- zamykamy okno --- */
   gtk_widget_destroy (GTK_WIDGET (dane));
}
```

Okno wyboru pliku

Okno wyboru pliku jest wyspecjalizowanym oknem dialogowym, udostępniającym standardowy sposób wyboru pliku i podobnym do uniwersalnego okna operacji na pliku znanego z Windows (*Common Dialog*). Dzięki niemu aplikacje GTK+ mogą posiadać wspólny wygląd i styl. Okno to pozwala na wybranie pliku przez użytkownika i przeprowadzenie pewnej czynności na wybranym pliku (patrz rysunek 6.4).



Rysunek 6.4. Okno wyboru pliku.

Okno wyboru pliku w GTK+ tworzy się przy pomocy funkcji gtk_file_selection_new, do której należy przekazać tytuł okna. Dobrze jest ustawić tytuł na „Zapisz jako" albo „Otwórz", na wypadek, gdyby użytkownik zapomniał, co właściwie chce zrobić, albo musiał wstać od komputera akurat wtedy, kiedy na ekranie pojawia się okno.

```
/* --- tworzymy okno dialogowe z tytułem --- */
plikw = gtk_file_selection_new ("Zapisz jako");
```

Jeśli okno dialogowe zostało wyświetlone dlatego, że użytkownik wybrał z menu opcję „Zapisz jako", wówczas pole nazwy pliku w oknie dialogowym powinno zawierać bieżącą nazwę pliku (mogą też istnieć inne powody ustawienia domyślnej wartości tego pola). Można ustawić domyślną nazwę pliku przy pomocy funkcji gtk_file_selection_set_filename.

```
/* --- ustawiamy domyślną nazwę pliku --- */
gtk_file_selection_set_filename (GTK_FILE_SELECTION (plikw),
                                 "problemy.txt");
```

Okno wyboru pliku posiada zbiór kontrolek, do których można uzyskać dostęp przez strukturę GtkFileSelection, zdefiniowaną w pliku gtkfilesel.h w następujący sposób:

```
struct _GtkFileSelection
{
  GtkWindow window;

  GtkWidget  *dir_list;
  GtkWidget  *file_list;
  GtkWidget  *selection_entry;
  GtkWidget  *selection_text;
  GtkWidget  *main_vbox;
  GtkWidget  *ok_button;
  GtkWidget  *cancel_button;
  GtkWidget  *help_button;
  GtkWidget  *history_pulldown;
  GtkWidget  *history_menu;
  GList         *history_list;
  GtkWidget  *fileop_dialog;
  GtkWidget  *fileop_entry;
  gchar         *fileop_file;
  gpointer       cmpl_state;

  GtkWidget   *fileop_c_dir;
  GtkWidget   *fileop_del_file;
  GtkWidget   *fileop_ren_file;

  GtkWidget *button_area;
  GtkWidget *action_area;

  };
```

Najczęściej używanymi kontrolkami są przyciski ok_button, cancel_button i help_button. Dla większości pozostałych kontrolek zdefiniowane są domyślne reakcje, których zazwyczaj nie trzeba zmieniać. Przyciski są częścią okna wyboru pliku i można zarejestrować dla nich swoje własne zdarzenia. Jeśli chcielibyśmy na przykład zostać poinformowani, że użytkownik nacisnął klawisz „Cancel", musielibyśmy umieścić w programie fragment kodu, który ustawiałby procedurę obsługi zdarzenia dla kontrolki cancel_button w strukturze GtkFileSelection.

```
gtk_signal_connect_object (
    GTK_OBJECT (GTK_FILE_SELECTION (plikw)->cancel_button),
```

```
"clicked",
(GtkSignalFunc) gtk_widget_destroy,
GTK_OBJECT (plikw));
```

Funkcja gtk_signal_connect_object działa podobnie do gtk_signal_connect - ustawia funkcje zwrotne dla sygnałów, ale wywoływana funkcja zwrotna otrzymuje tylko czwarty parametr funkcji gtk_signal_connect_object. Powoduje to, że funkcja zwrotna nie jest w stanie stwierdzić, która kontrolka ją wywołała. Funkcję zwrotną należy zdefiniować następująco:

```
gint FunkcjaZwrotna (GtkObject *kontrolka)
{
}
```

Obiekt, przesłany do funkcji zwrotnej, byłby w naszym przykładzie oknem wyboru pliku, co skonfigurowaliśmy w wywołaniu funkcji gtk_signal_connect_object. Odniesienie do obiektu, który spowodował wywołanie funkcji (w tym przypadku jest to przycisk „Cancel") zostanie usunięte, ale w tym przypadku nie ma to znaczenia.

Przycisk „Ok" można skonfigurować tak, aby przeprowadzał jakąś operację na pliku, przy pomocy podobnej procedury obsługi zdarzenia. Jeśli zaś aplikacja ma być przyjazna dla użytkownika, można zaopatrzyć w odpowiednią procedurę przycisk „Help" (można też usunąć go z okna, aby użytkownik nie nabrał fałszywego przekonania, że aplikacja zamierza mu w czymś pomóc, ale - naturalnie - nie jest to działanie przyjazne dla użytkownika).

Pasek postępów

Pasek postępów (GtkProgressBar) jest niewielką, elegancką kontrolką, używaną często w celu zobrazowania bieżącego stanu funkcji, których wykonanie zabiera dużo czasu. Wyświetlenie postępów jest z pewnością lepsze, niż pozostawienie użytkownika w niepewności co do czasu zakończenia operacji. Twórca programu posiada pełną kontrolę nad paskiem postępów, ponieważ do uaktualniania stanu kontrolki trzeba napisać odpowiedni kod.

GtkProgressBar dziedziczy wiele funkcji z kontrolki GtkProgress. Obie te kontrolki posiadają sporo możliwości, których nie będziemy tu opisywać. Stworzymy podstawową wersję paska postępów, której będziemy używać w dalszych przykładach podczas wykonywania długotrwałych operacji (patrz rysunek 6.5).

Kontrolkę paska postępów tworzymy przy pomocy funkcji gtk_progress_ bar_new albo gtk_progress_bar_new_with_adjustment. Po stworzeniu paska możemy uaktualnić go przy pomocy funkcji gtk_progress_ set_percent, do której przekazujemy procentową wartość ukończenia operacji, gdzie 0 oznacza ścisły początek, a 1 zakończenie operacji.

Kontrolka będzie korzystać ze struktury, przechowującej informacje na temat paska postępów. Potrzebne dane to używane okno dialogowe, kontrolka paska postępów, znacznik, który wskazuje, czy dozwolone jest zamknięcie okna dialogowego, oraz informacja o procencie ukończenia operacji, wyświetlona ostatnio na pasku postępów.

```
typedef struct {
   GtkWidget *pasekpostepow;
   GtkWidget *okno;
   int bPostep;
   int nOstatniProcent;
} typDanePostepu;
```

Rysunek 6.5. Pasek postępów.

Będziemy sterować naszym paskiem postępów przy pomocy trzech funkcji. Pierwsza to ZacznijPostep. Utworzy ona pasek postępów w oknie dialogowym.

```
/*
 * ZacznijPostep
 *
 * Tworzy okno dla paska postępów
 */

void ZacznijPostep ()
{
   GtkWidget *etykieta;
   GtkWidget *tabela;
   GtkWidget *okno;
   GtkAdjustment *reg;

   pdane = g_malloc (sizeof (typDanePostepu));
```

```
    pdane->nOstatniProcent = -1;
    pdane->bPostep = TRUE;
    /*
     * --- tworzymy okno najwyższego poziomu
     */
    okno = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    pdane->okno = okno;

    /* --- Podłączamy sygnał "delete_event" --- */
    gtk_signal_connect (GTK_OBJECT (okno), "delete_event",
                        GTK_SIGNAL_FUNC          (CzyMoznaZamknacOkno),
pdane);

    gtk_container_border_width (GTK_CONTAINER (okno), 10);

    /* --- tworzymy tabelę --- */
    tabela = gtk_table_new (3, 2, TRUE);
    gtk_container_add (GTK_CONTAINER (okno), tabela);

    /* --- dodajemy etykietę do tabeli --- */
    etykieta = gtk_label_new ("Otwieram plik...");
    gtk_table_attach_defaults (GTK_TABLE (tabela), etykieta, 0,2,0,1);
    gtk_widget_show (etykieta);

    /* --- dodajemy pasek postępów do tabeli --- */
    reg = (GtkAdjustment *) gtk_adjustment_new (0, 0, 400, 0, 0, 0);
    pdane->pasekpostepu = gtk_progress_bar_new_with_adjustment (reg);
    gtk_table_attach_defaults (GTK_TABLE (tabela),
                        pdane->pasekpostepu, 0,2,1,2);
    gtk_widget_show (pdane->pasekpostepu);

    /* --- uwidaczniamy wszystkie kontrolki --- */
    gtk_widget_show (tabela);
    gtk_widget_show (okno);
}
```

Funkcja UaktualnijPostep będzie obrazowała na pasku postępów bieżący stan długotrwałej operacji. Będzie przyjmować dwie wartości liczbowe - bieżącą wartość stanu i ostateczną wartość stanu. Kiedy bieżąca wartość stanu będzie równa ostatecznej, będzie to oznaczać, że operacja została zakończona. Moglibyśmy użyć tej funkcji podczas wczytywania dużego pliku; ostateczną wartością stanu byłby wówczas rozmiar pliku, a bieżącą wartością - liczba wczytanych do tej pory bajtów.

```
/*
 * UaktualnijPostep
 *
 * Uaktualnia pasek postępów, aby odzwierciedlić
 * postępy we wczytywaniu pliku.
 *
 * poz - jaka część pliku została wczytana.
 * dlug - długość pliku
 * (poz / dlug) = % wczytania pliku
 */
void UaktualnijPostep (long poz, long dlug)
{
   gfloat pwartosc;
   int procent;

    /* --- zapobiegamy dzieleniu przez zero --- */
   if (dlug > 0) {

      /* --- obliczamy procent --- */
      pwartosc = (gfloat) poz / (gfloat) dlug;

         procent = pwartosc * 100;

      if (pdane->nOstatniProcent != procent) {

         /* --- Uaktualniamy wyświetlaną wartość --- */
         gtk_progress_set_percentage (
               ➥GTK_PROGRESS (pdane->pasekpostepu), pwartosc);

         /* --- Odświeżamy okna - także okno postępów --- */
         while (gtk_events_pending ()) {
            gtk_main_iteration ();
         }
         pdane->nOstatniProcent = procent;
      }
   }
}
```

Funkcja ZakonczPostep zamyka okno dialogowe z paskiem postępów. Powinna zostać wywołana po zakończeniu operacji w celu zamknięcia okna.

```
/*
```

```
 * ZakonczPostep
 *
 * Zamyka okno z paskiem postępów
 */
void ZakonczPostep ()
{
   /* --- Pozwalamy na zamknięcie okna --- */
   pdane->bPostep = FALSE;

   /* --- usuwamy okno --- */
   gtk_widget_destroy (pdane->okno);

   /* --- zwalniamy przydzieloną pamięć --- */
   g_free (pdane);

   pdane = NULL;
}
```

Funkcja CzyMoznaZamknacOkno uniemożliwia użytkownikowi zamknięcie okna, zanim operacja nie dobiegnie końca. Jest wywoływana przez sygnał "delete_event", wysyłany, kiedy użytkownik spróbuje zamknąć okno. Wartość powrotna określa, czy można zamknąć okno.

```
/*
 * CzyMoznaZamknacOkno
 *
 * Funkcja ta sprawdza, czy można zamknąć okno dialogowe.
 */
gint CzyMoznaZamknacOkno (GtkWidget *kontrolka)
{
   /* --- TRUE => nie można zamknąć --- */
   /* --- FALSE => można zamknąć --- */
   return (pdane->bPostep);
}
```

Poniżej zamieszczamy przykład, ilustrujący działanie paska postępów. Tworzy on okno, zawierające przycisk. Po naciśnięciu przycisku tworzone jest (przy pomocy funkcji ZacznijPostep) okno dialogowe z paskiem postępów, który jest uaktualniany co 100 milisekund przy pomocy zegara.

Korzystanie z zegara

Zegar tworzy się przy pomocy funkcji gtk_timeout_add. Funkcja ta przyjmuje częstotliwość zegara, inną funkcję i parametr danych. Funkcja, przekazana jako parametr do gtk_timeout_add będzie wywoływana z zadaną częstotliwością, przy czym będzie do niej przekazywany parametr danych. Na przykład następujące wywołanie funkcji:

```
pzegar = gtk_timeout_add (100, UaktualnijZegarPaska, dane);
```

spowoduje wywoływanie co 100 milisekund funkcji UaktualnijZegarPaska z parametrem dane. Wartością zwrotną funkcji gtk_timeout_add jest identyfikator zegara, który jest potrzebny do zakończenia jego pracy. Aby zaprzestać wywoływania funkcji UaktualnijZegarPaska, należy skorzystać z funkcji gtk_timeout_remove. Przekazujemy do niej identyfikator, otrzymany z wywołania gtk_timeout_add.

```
gtk_timeout_remove (pzegar);
```

Test paska postępów

Oto program, służący do przetestowania działania paska postępów:

```
/*
 * Tutaj zaczyna się kod aplikacji
 */

#include <gtk/gtk.h>

int pzegar;
int nWartosc;

/*
 * UaktualnijZegarPaska
 *
 * Jest to funkcja zwrotna zegara. Uaktualnia pasek postępów
 * i zamyka jego okno, kiedy pasek osiągnie końcową wartość.
 */
UaktualnijZegarPaska (gpointer dane)
{

   /* --- wartość przyrostu --- */
   nWartosc += 1;

   /* --- uaktualniamy pasek postępów  --- */
   UaktualnijPostep (nWartosc, 100);
```

```
    /* --- Zamykamy go, jeśli osiągnął końcową wartość --- */
    if (nWartosc == 100) {
        ZakonczPostep ();
        gtk_timeout_remove (pzegar);
    }
}

/*
 * KliknietoPrzycisk
 *
 * Wywoływana po kliknięciu przycisku w celu utworzenia zegara
 * i paska postępów.
 */
gint KliknietoPrzycisk (GtkWidget *kontrolka, gpointer dane)
{
    /* --- Inicjacja --- */
    nWartosc = 0;
    ZacznijPostep ();

    /* --- Wywołujemy zegar --- */
    pzegar = gtk_timeout_add (100, UaktualnijZegarPaska, dane);
}

/*
 * ZamknijOknoAplikacji
 *
 * Zamyka aplikację
 */
gint ZamknijOknoAplikacji ()
{
    gtk_main_quit ();
    return (FALSE);
}

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

```
    GtkWidget *przycisk;
    gtk_init (&argc, &argv);

    /* --- tworzymy główne okno i nadajemy mu tytuł --- */
    okno = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (okno), "Pasek postêpów");

    /* --- kończymy aplikację po wykryciu sygnału delete_event --- */
    gtk_signal_connect (GTK_OBJECT (okno), "delete_event",
                        GTK_SIGNAL_FUNC (ZamknijOknoAplikacji), NULL);

    /* --- tworzymy przycisk, który wyświetli pasek postępów ...--- */
    przycisk = gtk_button_new_with_label ("Pasek postêpów");
    gtk_widget_show (przycisk);

    /* --- ...a tutaj ustawiamy jego procedurę obsługi --- */
    gtk_signal_connect (GTK_OBJECT (przycisk), "clicked",
                        GTK_SIGNAL_FUNC (KliknietoPrzycisk), NULL);

    gtk_container_add (GTK_CONTAINER (okno), przycisk);
    gtk_widget_show (okno);

    /* --- Oddajemy sterowanie do GTK --- */
    gtk_main ();
    exit (0);
}
```

Podsumowanie

Kontrolki opisane w tym rozdziale są bardziej skomplikowane, niż proste kontrolki, którymi zajmowaliśmy się wcześniej. Okna dialogowe są używane w GTK+ bardzo często, a przykładem ich zastosowania jest okno wyboru pliku. Okno wyboru pliku jest standardowym oknem dialogowym, pozwalającym użytkownikowi na wybieranie plików. Kontrolka tekstu jest niemal gotowym edytorem. Pasek postępów może zobrazować postępy w wykonaniu długotrwałej operacji. Wszystkie te kontrolki znacznie ułatwiają pisanie aplikacji dla Linuksa.