Header signal.h

Header	Opis zawartych funkcji	Funkcje
signal.h	Funkcje obsługujące sygnał	signal(), raise()

#include<signal.h>

Sygnał to przerwanie generowane przez oprogramowanie, które to przerwanie jest wysyłane do procesu przez system operacyjny.

A więc sygnały służą jako środek komunikacji pomiędzy systemem operacyjnym a działającym programem (procesem). Są to przerwania generowane automatycznie, wysyłane do uruchomonego programu przez system operacyjny, gdy np. naciśniesz Ctrl+C lub gdy inny proces (błąd komunikacji międzyprocesowej) wyśle sygnał do naszego programu. Sygnał może także zgłaszać nietypowe zachowanie w programie, takie jak dzielenie przez zero lub innej niewykonalnej operacji matematycznej, brak dostępu do określonych komórek pamięci lub awarię sprzętu.

W dalszym procesie możemy skierować działanie programu w odpowiednim kierunku w zależności od charakteru zdarzenia. Można bowiem określić procedurę obsługi sygnału dla każdego z sygnałów. Sygnały są obsługiwane zgodnie z domyślnym zachowaniem, jednak język C oferuje również możliwość zarządzania nimi zgodnie z naszymi wymaganiami.

Dwie funkcje, które zajmują się sygnałem, to ***signal()***, która konfiguruje procedurę obsługi dla warunku błędu, oraz ***raise()***, która uruchamia warunek sygnału. Reszta wyszczególnionych tu poniżej elementów z **signal.h** służy tym dwóm funkcjom.

- **signal (** nazwa sygnału, funkcja obsługi sygnału **)** – ustawia obsługę sygnału
- **raise (** nazwa sygnału **)** – uruchamia obsługę sygnału

Skopiowane z pliku nagłówkowego **signal.h** dla Turbo C++ (zawarte też w Dev C++):

Jedna z trzech poniższych nazw może być deklarowana jako '**funkcja obsługi sygnału**' w funkcji **signal()**.

```
#define SIG_DFL ((_CatcherPTR)0)       /* Default action */
#define SIG_IGN ((_CatcherPTR)1)       /* Ignore action */
#define SIG_ERR ((_CatcherPTR)-1)      /* Error return */

#define SIGINT 2       <-- te sześć wartości to właśnie 'nazwy sygnału' w definicji signal() i raise()
#define SIGILL 4      /* Illegal instruction */
#define SIGFPE 8      /* Floating point trap */
#define SIGSEGV 11    /* Memory access violation */
#define SIGTERM 15
#define SIGABRT 22

#ifdef __cplusplus
void _Cdecl (* _Cdecl signal(int __sig, void _Cdecl (* __func)(int))) (int);
#else
void _Cdecl (* _Cdecl signal(int __sig, void _Cdecl (* func)())) (int);
#endif
```

```
int _Cdecl raise(int __sig);
```

Tak więc **__sig** to nasza 'nazwa sygnału': **SIGINT**, **SIGILL**, **SIGFPE**, **SIGSEGV**, **SIGTERM** lub **SIGABRT**. **func** to 'funkcja obsługi sygnału'; **__func** sugeruje deklarowane w pliku nagłówkowym **signal.h**: **SIG_DFL**, **SIG_IGN** lub **SIG_ERR**.

Teraz najważniejszą rzeczą jest bardziej usystematyzować powyższe zapisy a następnie podać przykłady, które ugruntują zrozumienie ich zastosowania.

Zamast uproszczonego zapisu z pliku nagłówkowego:

```
void (* signal(int sig, void (*func)(int)) )(int)
```

napiszmy to tak:

```
void (*signal(int nazwa sygnału, void (*funkcja obsługi sygnału)(int)))(int)
```

Ten zapis ustawia akcję podejmowaną po odebraniu sygnału ***nazwa sygnału*** (sig) przez program. Jeśli wartość ***funkcja obsługi sygnału*** (func) to SIG_DFL, wtedy zostanie zastosowana domyślna obsługa tego sygnału. Jeśli wartość ***funkcja obsługi sygnału*** (func) to SIG_IGN, sygnał zostanie zignorowany. W przeciwnym razie ***funkcja obsługi sygnału*** (func) wskazuje na funkcję obsługi sygnału, która ma zostać wywołana po wystąpieniu sygnału.

Tak węc funkcja signal() ustawia akcję (***funkcja obsługi sygnału***), która ma zostać wykonana w momencie, gdy program otrzyma sygnał ***nazwa sygnału***. Inaczej mówiąc jest to zarejestrowanie funkcji ***funkcja obsługi sygnału**, która bezpośrednio po wysłaniu sygnału ma być wywołana z argumentem ***nazwa sygnału*.**
funkcja obsługi sygnału - Może to być funkcja zdefiniowana przez programistę lub jedna z następujących predefiniowanych funkcji: SIG_DFL albo SIG_IGN (patrz poniżej).

```
int raise(int nazwa sygnału)
```

nazwa sygnału	Numer sygnału	Powód generowania sygnału
SIGINT	**2**	Przerwanie, np. z klawiatury, takie jak naciśnięcie [Ctrl]-[C]. Zazwyczaj taka akcja jest generowana przez użytkownika.
SIGILL	**4**	Niedozwolona instrukcja lub anomalie w trakcie wykonywania programu, które są zwykle spowodowane problemami pamięci.
SIGFPE	**8**	Wystąpienie błędu operacji zmiennoprzecinkowej lub innego matematycznego błędu, np. dzielenie przez zero.
SIGSEGV	**11**	Podjęto próbę uzyskania dostępu do pamięci, na którą system nie pozwala.
SIGTERM	**15**	Wymóg zakończenia programu (prawdopodobnie spowodowany zamknięciem systemu operacyjnego).
SIGABRT	**22**	Wymuszone, niekontrolowane zakończenie wykonywania programu, być może wywołane działaniem **abort()**.

Sygnały te są generowane automatycznie. Można je również wygenerować, wywołując funkcję **raise**() jak to przedstawiono w poniższym przykładzie. (Na podstawie: Peter Hipson - Advanced C, str 512.)

```
/* Generowanie sygnału funkcją raise() */

#include <stdio.h>
#include <signal.h>                 // ten temat
#include <locale.h>                 // dla 'setlocale()'
```

```c
void obsluga_sygnalu( int sygnal ); // prototyp funkcji
int numer_sygnalu;

int main(void)
{  setlocale(LC_CTYPE, "Polish");   // polskie znaki

   printf("Początkowy numer sygnału: %d\n", SIG_DFL);  // wartość domyślna

   signal(SIGINT, obsluga_sygnalu);
   printf("\nWysłany sygnał: %d\n", SIGINT);
   raise(SIGINT);
   printf("Aktualny numer sygnału: %d\n", numer_sygnalu);

   signal(SIGILL, obsluga_sygnalu);
   printf("\nWysłany sygnał: %d\n", SIGILL);
   raise(SIGILL);
   printf("Aktualny numer sygnału: %d\n", numer_sygnalu);

   signal(SIGFPE, obsluga_sygnalu);
   printf("\nWysłany sygnał: %d\n", SIGFPE);
   raise(SIGFPE);
   printf("Aktualny numer sygnału: %d\n", numer_sygnalu);

   signal(SIGSEGV, obsluga_sygnalu);
   printf("\nWysłany sygnał: %d\n", SIGSEGV);
   raise(SIGSEGV);
   printf("Aktualny numer sygnału: %d\n", numer_sygnalu);

   signal(SIGTERM, obsluga_sygnalu);
   printf("\nWysłany sygnał: %d\n", SIGTERM);
   raise(SIGTERM);
   printf("Aktualny numer sygnału: %d\n", numer_sygnalu);

   signal(SIGABRT, obsluga_sygnalu);
   printf("\nWysłany sygnał: %d\n", SIGABRT);
   raise(SIGABRT);
   printf("Aktualny numer sygnału: %d\n", numer_sygnalu);
}

void obsluga_sygnalu( int sygnal )
{
   numer_sygnalu = sygnal;
}
```

Wynik:

```
Początkowy numer sygnału: 0

Wysłany sygnał: 2
Aktualny numer sygnału: 2
```

```
Wysłany sygnał: 4
Aktualny numer sygnału: 4

Wysłany sygnał: 8
Aktualny numer sygnału: 8

Wysłany sygnał: 11
Aktualny numer sygnału: 11

Wysłany sygnał: 15
Aktualny numer sygnału: 15

Wysłany sygnał: 22
Aktualny numer sygnału: 22

--------------------------------
Process exited after 4.415 seconds with return value 0
Press any key to continue . . . _
```

Dopóki nie zmienimy akcji przez funkcję ***raise()*** to będzie się wykonywać ostatnia ustawiona akcja. Aby to sprawdzić, znajdź np. linię

`raise(SIGILL);` i ją zablokuj: `// raise(SIGILL);`

Uruchom program ponownie zwracając uwagę na zmianę wyniku.

Informacje dodatkowe:

- Tuż przed wywołaniem (***funkcja obsługi sygnału***)(***nazwa sygnału***) jest niejawnie wywoływana funkcja **`signal`**`(`***nazwa sygnału***`,` `SIG_DFL`). Powoduje to ustanowienie domniemanej obsługi sygnału ***nazwa sygnału***. W kodzie powyżej to "Początkowy numer sygnału".
- Poza obsługą domniemaną można zażądać zignorowania sygnału. W tym celu można posłużyć się drugim argumentem funkcji **`signal()`** wyrażonym za pomocą symbolu `SIG_IGN`.

(Na podstawie: Jan Bielecki, Encyklopedia Języka C dla IBM PC, tom 2 - Biblioteki, strona 223)

Działanie:

`signal` kojarzy ***funkcję obsługi sygnału*** z sygnałem o numerze ***nazwa sygnału***. Sposób, w jaki sygnał numeruje ***nazwę sygnału*** zależy od wartości ***funkcja obsługi sygnału***:
(Poniższe makra będą używane zarówno w funkcji ***signal()*** jak i ***raise()***. Makra `SIG_` służą do definiowania funkcji sygnałowych.)

wartość ***funkcja obsługi sygnału***	obsługa sygnału
`SIG_DFL`	obsługa domyślna
`SIG_ERR`	wskazuje błąd funkcji **signal()**
`SIG_IGN`	sygnał jest zignorowany
wskaźnik do funkcji *func*	wywoływana jest ***funkcja obsługi sygnału***.

Tak więc zapis:

```
void (*signal( int nazwa sygnału, void (*funkcja obsługi sygnału) (int))) (int);
```

ustawia `procedurę obsługi` **`błędów`** dla *signal* ***`nazwa sygnału`***. `Procedurę obsługi` **`sygnału`** można ustawić w taki sposób, aby wystąpiła domyślna obsługa (`SIG_DFL`), sygnał był ignorowany (`SIG_IGN`) lub wywoływana była funkcja zdefiniowana przez użytkownika (inna ***funkcja obsługi sygnału***).

- ✓ Jeżeli ***signal()*** wykona się z sukcesem, rezultatem jest dana typu **`int`** reprezentowana przez drugi argument ostatniego wywołania funkcji ***signal()***, czyli ***funkcja obsługi sygnału*** z pierwszym argumentem (***nazwa sygnału***), albo
- ✓ dana reprezentowana przez symbol `SIG_ERR` w przypadku niepomyślnego wywołania funkcji ***signal()***.

Takie terminy jak:
- `SIGINT`, `SIGILL`, `SIGFPE`, `SIGSEGV`, `SIGTERM`, `SIGABRT`,
- `SIG_DFL`, `SIG_ERR`, `SIG_IGN`

określa się mianem nazw symbolicznych (*symbolic names*). Sygnały są identyfikowane przez przypisane do nich liczby całkowite.

Z poniższego, najbardziej prostego przykładu wynika, że:

- Wykrycie błędu lub sekwencji [Ctrl]-[C] wcale nie musi przerywać programu, w przeciwieństwie do ***assert()***. Zamiast `exit(1);` może to być kontynuacja działania programu. W praktyce można rozważyć następujące scenariusze:
  - zignoruj sygnał,
  - zakończ proces,
  - zablokuj proces,
  - odblokuj proces.

- Ustawienie obsługi sygnału - funkcja '**`signal()`**' - nie musi być w miejscu wystąpienia zagrożenia dla działania programu, tak jak ojciec, pilnujący dziecka na placu zabaw, nie musi z nim wchodzic do piaskownicy. Wystarczy, że zasygnalizuje swoją obecność w programie (czytaj: placu zabaw).

```c
/* Przykład użycia 'signal' */

#include<stdio.h>
#include<stdlib.h>                  // dla exit(1)
#include<signal.h>                  // ten temat
#include <locale.h>                 // dla 'setlocale()'

void PrzerwanieProgramu(int);

int main()
{  setlocale(LC_CTYPE, "Polish");  // polskie znaki
   char s[100];

   if ( signal(SIGINT, PrzerwanieProgramu) == SIG_ERR)
   {   printf("Niemożliwe ustawienie obsługi sygnałów");
       exit(0);
   }

   while (0 == 0)                  // to samo co: while ( 1 ) czyli zawsze prawda
   {   puts("Tu wpisz zdanie albo naciśnij [Ctrl]+[C] aby wyjść z programu:");
       gets(s);
       printf("Wpisałeś: %s\n\n", s);
   }
```

```
   return 0;
}

void PrzerwanieProgramu(int)
{
   puts("\n\nWykryłem kombinację klawiszy [Ctrl]+[C] !");
   puts("Wyjście z programu poprzez exit(1)");
   getc(stdin);
   exit(1);
}
```

Być może [Enter] będziesz musiał przycisnąć dwa razy. Bierze się to z technicznej realizacji obsługi sygnałów.

Przykładowy wynik:

```
Tu wpisz zdanie albo naciśnij [Ctrl]+[C] aby wyjść z programu:
Nie mam czasu bo jestem wzywany na boisko. Opuszczam program...
Wpisałeś: Nie mam czasu bo jestem wzywany na boisko. Opuszczam program...

Tu wpisz zdanie albo naciśnij [Ctrl]+[C] aby wyjść z programu:
Wpisałeś: Nie mam czasu bo jestem wzywany na boisko. Opuszczam program...

Tu wpisz zdanie albo naciśnij [Ctrl]+[C] aby wyjść z programu:


Wykryłem kombinację klawiszy [Ctrl]+[C] !
Wyjście z programu poprzez exit(1)
_
```