

# Headers: 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.

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** ( nazwy sygnału, funkcja obsługi sygnału ) – ustawia obsługę sygnału
- **raise** ( nazwy 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()
#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 wybrnąć z tego gąszczu nazw, uprościć i usystematyzować powyższe zapisy. Następnie podać przykłady, które to ugruntują.

ANSI C używa 'sygnału', aby poinformować program o wystąpieniu błędu. Plik nagłówkowy **signal.h** definiuje warunki, które są przechwytywane i opcjonalnie przekazywane do programu.

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

Funkcja 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)
```

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

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()'

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 signal )
{
    numer_sygnalu = signal;
}

```

## 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 <i>funkcja obsługi sygnału</i>	obsługa sygnału
-----	-----
<b>SIG_DFL</b>	obsługa domyślna
<b>SIG_ERR</b>	wskazuje błąd funkcji <b>signal()</b>
<b>SIG_IGN</b>	sygnał jest zignorowany
wskaźnik do funkcji <i>f</i>	wywoływana jest <i>funkcja obsługi sygnału</i> .

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 (*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:
  - o zignoruj sygnał,
  - o zakończ proces,
  - o zablokuj proces,
  - o 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 wchodzić do piaskownicy. Wystarczy, że zasygnalizuje swoją obecność w programie (czytaj: placu zabaw).

```
/* 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 )
  { 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)
_
```