Sygnały

Sygnały to swego rodzaju programowe przerwania, to znaczy zdarzenia, które mogą nastąpić w dowolnym momencie działania procesu, niezależnie od tego, co dany proces aktualnie robi.

Sygnały są *asynchroniczne*. Umożliwiają komunikację między procesami.

Procesy mogą otrzymywać sygnały

- z jądra systemu,
- od samego siebie
- od innych procesów
- od użytkownika.

Sygnały są wysyłane:

- za pomocą funkcji systemowej kill
- za pomocą polecenia kill
- za pomocą klawiatury - tylko wybrane sygnały
- przez pewne sytuacje wyjątkowe wykrywane przez oprogramowanie systemowe
- przez pewne sytuacje wyjątkowe wykrywane przez sprzęt

Istnieją pewne ograniczenia - proces może wysyłać je tylko do procesów mających tego samego właściciela oraz należących do tej samej grupy (te same identyfikatory uid i gid). Bez ograniczeń może to czynić jedynie jądro i administrator. Ponadto jedynym procesem, który nie odbiera sygnałów jest init (PID = 1).

Sygnał jest **dostarczony** (ang. *delivered*) do procesu, gdy proces podejmuje akcję obsługi sygnału.

Proces może zareagować na otrzymany sygnał na różne sposoby:

- Zezwolenie na domyślną obsługę sygnału – najczęściej jest to zakończenie procesu i ewentualny zrzut zawartości segmentów pamięci na dysk (plik core).
- Zignorowanie sygnału – można zignorować wszystkie sygnały poza SIGKILL oraz SIGSTOP, co zapewnia, że proces zawsze można zakończyć.
- Przechwycenie sygnału – podjęcie akcji zdefiniowanej przez użytkownika
- Maskowanie – blokowanie sygnału tak, aby nie był dostarczany.

Działanie domyślne – czynności podejmowane przez jądro, gdy pojawi się sygnał. Są to:

- zakończenie (ang. *termination*)
- ignorowanie (ang. *ignoring*)
- zrzut pamięci (ang. *core dump*)
- zatrzymanie(ang. *stopped*)

W czasie pomiędzy wygenerowaniem sygnału a jego dostarczeniem, sygnał jest w stanie **oczekiwania** (*ang. pending)* na dostarczenie.

Sygnał zablokowany (ang. *blocked*) jest to sygnał, który nie może być dostarczony. Pozostaje w stanie oczekiwania.
Maska sygnałów (ang. *signal mask*) jest to zbiór sygnałów, które są dla procesu zablokowane

Sygnałom przypisane są nazwy, rozpoczynające się od liter "SIG", oraz numery. O ile proces nie został zakończony w wyniku przerwania, kontynuuje on swoje działanie od miejsca przerwania.

Dwa sygnały - SIGKILL i SIGSTOP - nie mogą zostać przechwycone ani zignorowane. Po otrzymaniu któregoś z nich proces musi wykonać akcję domyślną. Daje nam to możliwość bezwarunkowego zatrzymania/zakończenia dowolnego procesu, jeżeli zajdzie taka potrzeba. Nie powinno się ignorować również niektórych sygnałów związanych z błędami sprzętowymi. Jeśli chcemy przechwytywać jakiś sygnał, musimy zdefiniować funkcję która będzie wykonywana po otrzymaniu takiego sygnału, oraz powiadomić jądro za pomocą funkcji signal(), która to funkcja. Dzięki temu możemy np. sprawić że po naciśnięciu przez użytkownika ctrl+c, nasz program zdąży jeszcze zamknąć połączenia sieciowe czy pliki, usunąć pliki tymczasowe itd.

Rodzaje sygnałów

Nazwy sygnałów są zdefiniowane w `signal.h`. Pełna lista znajduje się np tutaj lub w manualu polecenia kill.

Nazwa	Numer	Znaczenie	Czynność domyślna
SIGHUP	1	Przerwanie łączności z terminalem. Służy do zakończenia pracy wszystkich procesów w momencie zakończenia sesji w danym terminalu.	Zakończenie
SIGINT	2	Terminalowe przerwanie (Ctrl+C)	Zakończenie
SIGQUIT	3	Terminalowe zakończenie (Ctrl+\)	Zrzut pamięci i zakończenie
SIGILL	4	Nielegalna instrukcja sprzętowa	Zrzut pamięci i zakończenie
SIGABRT	6	Przerwanie procesu. Wysyłany przez funkcję abort()	Zrzut pamięci i zakończenie
SIGFPE	8	Wyjątek arytmetyczny (np. dzielenie przez 0)	Zrzut pamięci i zakończenie
SIGKILL	**9**	**Unicestwienie (nie da się przechwycić ani zignorować)**	**Zakończenie**
SIGSEGV	11	Niepoprawne wskazanie pamięci	Zrzut pamięci i zakończenie
SIGPIPE	13	Zapis do potoku zamkniętego z jednej strony (nikt nie czyta)	Ignorowany
SIGALRM	14	Pobudka (upłynął czas ustawiony funkcją alarm() lub setitimer())	Ignorowany
SIGTERM	15	Zakończenie programowe (domyślny sygnał polecenia kill)	Zakończenie
SIGCHLD	17	Zakończenie procesu potomnego	Ignorowany
SIGSTOP	**19**	**Zatrzymanie (nie da się przechwycić ani zignorować)**	**Zatrzymanie**
SIGCONT	18	Kontynuacja wstrzymanego procesu	Ignorowany
SIGTSTP	20	Terminalowe zatrzymanie (Ctrl+Z lub Ctrl+Y)	Zatrzymanie
SIGTTIN	21	Czytanie z terminala przez proces drugoplanowy	Zatrzymanie
SIFTTOU	22	Pisanie do terminala przez proces drugoplanowy	Zatrzymanie
SIGUSR1	10	Sygnał zdefiniowany przez użytkownika	
SIGUSR1	12	Sygnał zdefiniowany przez użytkownika	

Sygnały czasu rzeczywistego

`SIGRTMIN`, **`SIGRTMIN+n`**, **`SIGRTMAX`**

Sygnały czasu rzeczywistego to rozszerzenie mechanizmu sygnałów. Systemy UNIX-owe wspierają 32 sygnały czasu rzeczywistego. Mają one numery od SIGRTMIN do SIGRTMAX. Do kolejnych sygnałów odwołujemy się za pomocą notacji SIGRTMIN+n, ponieważ ich numery różnią się w różnych systemach UNIXowych, choć zawsze są w tej samej kolejności.

Sygnały te nie posiadają predefiniowanego znaczenia; można je wykorzystać do celów określonych w danej aplikacji. Domyślnią akcją sygnału czasu rzeczywistego jest przerwanie procesu. Sygnały czasu rzeczywistego, w przeciwieństwie do standardowych sygnałów, są kolejkowane; przechowywane są w kolejce FIFO. Ponato mają priorytety, tzn. im mniejszy numer sygnału tym wcześniej zostanie dostarczony dany sygnał.

Polecenia Unixa

`kill` - wysyła do procesu lub grupy procesów określony sygnał. Można mu przekazać zarówno numer sygnału, jak i jego nazwę.

`trap` - polecenie, służące do określenia reakcji procesu na dany sygnał.

Wysyłanie sygnałów

```
kill(int pid, int SIGNAL);
```

Funkcja `kill` służy do przesyłania sygnału do wskazanego procesu w systemie. Wymaga dołączenia nagłówków **sys/types.h** and **signal.h**.

Argument pid określa identyfikator procesu-odbiorcy sygnału, natomiast sig jest numerem wysyłanego sygnału.

Jeśli:

pid > 0	sygnał jest wysyłany do procesu o identyfikatorze pid
pid = 0	sygnał jest wysyłany do wszystkich procesów w grupie procesu wysyłającego sygnał
pid = -1	sygnał jest wysyłany do wszystkich procesów w systemie, z wyjątkiem procesów specjalnych, na przykład procesu init; nadal obowiązują ograniczenia związane z prawami
pid < -1	sygnał jest wysyłany do wszystkich procesów we wskazanej grupie -pid

Funkcja kill zwraca 0, jeśli sygnał został pomyślnie wysłany, w przeciwnym wypadku zwraca -1 i ustawia kod błędu w zmiennej `errno`.

`int raise( int signal);`

Wysyła sygnał do bieżącego procesu; do tego celu w rzeczywistości wykorzystuje funkcję `kill()`. O ile nie zostanie przechwycenie lub zignorowanie sygnału, proces zostanie zakończony. Wywołanie `raise()` jest równoważne z wywołaniem `kill(getpid(), sig);`

`int sigqueue(pid_t pid, int sig, const union sigval value)`

Funkcja ta wysyła sygnał sig do procesu o danym pid. Jeśli przekazany pid jest równy 0 sygnał nie zostanie wysłany, natomiast nastąpi sprawdzenie ewentualnych błędów, które mogłyby nastąpić przy wysyłaniu.

Argument *sigval* może zawierać dodatkową wartość wysłaną wraz z sygnałem. Typ sigval zdefiniowany jest następująco:

```
union sigval {
  int sival_int;
  void *sival_ptr;
}
```

Odbieranie sygnałów – signal

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

Funkcja ta służy do ustawienia przechwytywania danego sygnału.

Pierwszy argument to numer przechwytywanego sygnału, drugi - wskaźnik na funkcję, która wykona się w przypadku przechwycenia tego sygnału. Funkcja obsługi sygnału musi przyjmować odkładnie jeden argument (numer sygnału) i nic nie zwracać. Funkcja ta zwraca poprzedni, lub SIG ERR jeśli wystąpił błąd.

Zamiast wskaźnika na funkcję *func* można również przekazać jedną z dwóch zmiennych: SIG IGN, oznaczającą ignorowanie sygnału lub SIG DEF, oznaczającą domyślną reakcję na sygnał.

```
void (*signal (int signo, void (*func)(int)));
```

gdzie:

- signo - określa numer sygnału, dla którego definiowana jest obsługa.
- handler - nazwa funkcji obsługi sygnału zdefiniowanej przez użytkownika. Może on również przyjmować jedną z wartości:
  - - SIG_IGN - oznacza, że sygnał będzie ignorowany
  - - SIG_DFL - sygnał będzie obsługiwany w sposób domyślny, zdefiniowany w systemie.

Funkcja zwraca SIG_ERR jeśli wystąpił błąd, lub adres poprzedniej funkcji obsługi sygnału.

Przykład :

```
#include <stdio.h>
#include <signal.h>


/* procedura obslugi sygnalu SIGINT */
void obslugaINT(int signum){
  printf("Obsluga sygnalu SIGINT\n";
}


main() {
  /* zarejestrowanie obslugi sygnalu SIGINT */
  signal(SIGINT, obslugaINT)
  /* nieskonczona petla */
  while(1)
  sleep(100);
}
```

Niezawodna obsługa sygnałów (ang. reliable signals)

Funkcje niezawodnej obsługi sygnałów muszą się charakteryzować:

- stałą (ang. persistent) obsługą sygnałów;
- blokowaniem tego samego sygnału podczas jego obsługi;
- jednokrotnym dostarczeniem sygnału do procesu po odblokowaniu
- możliwością maskowania (blokowania) sygnałów

Funkcją, która ma zapewniać niezawodną obsługę sygnałów jest sigaction.

Odbieranie sygnałów – sigaction

```
int sigaction(int sig_no, const struct sigaction *act, struct sigaction
*old_act);
```

Jest to rozszerzenie funkcji `signal` - służy zatem do zmiany dyspozycji sygnału.

Wykorzystywana tu struktura `sigaction` wygląda następująco:

```
struct sigaction{
  void (*sa_handler)();/* Wskaźnik do funkcji obsługi sygnału*/
  sigset_t sa_mask;      /* Maska sygnałów – czyli zbiór sygnałów blokowanych podczas obsługi
                            bieżącego sygnału, sygnał przetwarzany jest blokowany domyślnie */
  int sa_flags;          /* Nadzoruje obsługę sygnału przez jądro*/
};
```

sa_mask zawiera zbiór sygnałów, które mają być zablokowane na czas wykonania tej funkcji. W ten sposób możemy się zabezpieczyć przed odebraniem jakiegoś sygnału (i w konsekwencji wykonaniem jego funkcji) w czasie, kiedy jeszcze wykonuje się funkcja obsługująca inny sygnał. W szczególności drugie obsłużenie tego samego sygnału podczas obsługiwania pierwszego jego egzemplarza jest zawsze blokowane.

Przykładowe użycie:

```
void au(int sig_no) {
  printf("Otrzymale signal %d.\n", sig_no);
}

int main() {
 struct sigaction act;
 act.sa_handler = au;
 sigemptyset(&act.sa_mask);
 act.sa_flags = 0;
 sigaction(SIGINT, &act, NULL);

 while(1) {
   printf("Witaj.\n");
   sleep(3);
 }
 return 0;
}
```

Czekanie na sygnały

```
void pause();
```

Zawiesza wywołujący proces aż do chwili otrzymania dowolnego sygnału. Jeśli sygnał jest ignorowany przez proces, to funkcja pause też go ignoruje. Najczęściej sygnałem, którego oczekuje pause jest sygnał pobudki SIGALARM.

```
unisigned int sleep(unsigned int seconds);
```

Usypia wywołujący ją proces na określoną w argumencie liczbę sekund. Funkcja zwraca 0 lub liczbę sekund, pozostających do zakończenia drzemki. Sprawia, że proces wywołujący ją jest zawieszany, dopóki nie zostanie wyzerowany licznik czasu określający czas pozostający do końca drzemki lub proces przechwyci sygnał, a procedura jego obsługi po zakończeniu pracy wykona return. . Funkcja zdefiniowana w bibliotece unistd.h.

```
unsigned int alarm (unsigned int sec);
```

Ustala czas, po jakim zostanie wygenerowany jednorazowo sygnał SIGALARM. Jeśli nie ignorujemy lub nie przechwytujemy tego sygnału, to domyślną akcją jest zakończenie procesu. Funkcja zdefiniowana w bibliotece unistd.h.

Zbiory sygnałów

Zbiory (zestawy) sygnałów definiuje się, aby pogrupować różne sygnały. Ma to umożliwić wykonywanie pewnych działań na całej grupie sygnałów. Zestaw sygnałów definiowany jest przez typ `sigset_t`. W tej strukturze każdemu sygnałowi przypisany jest jeden bit (prawda/fałsz), mówiący, czy dany sygnał należy do danego zestawu czy nie.

```
int sigemptyset ( sigset_t* signal_set );
```

Inicjalizacja pustego zbioru sygnałów.

```
int sigfillset ( sigset_t* signal_set );
```

Inicjalizacja zestawu zawierającego wszystkie sygnały istniejące w systemie.

```
int sigaddset ( sigset_t* signal_set, int sig_no );
```

Dodawanie pojedynczego sygnału do zbioru.

```
int sigdelset ( sigset_t* signal_set, int sig_no );
```

Usunięcie pojedynczego sygnału z zestawu.

```
int sigismember ( sigset_t *signal_set, int sig_no );
```

Sprawdzenie, czy w zestawie znajduje się dany sygnał.

Przykład

```
/* utworzenie zbioru dwóch sygnałów SIGINT i SIGQUIT */
sigset_t twosigs;
sigemptyset(&twosigs);
sigaddset(&twosigs, SIGINT);
sigaddset(&twosigs, SIGQUIT);
```

Maskowanie – blokowanie sygnałów

Można poinformować jądro o tym iż nie chcemy, aby przekazywano sygnały bezpośrednio do danego procesu. Do tego celu wykorzystuje się zbiory sygnałów zwane maskami. Kiedy jądro usiłuje przekazać do procesu sygnał, który aktualnie jest blokowany, to zostaje on przechowany do momentu jego odblokowania lub ustawienia ignorowania tego sygnału przez proces.

```
int sigprocmask(int how, const sigset_t *new_set, sigset_t *old_set);
```

Funkcja ustawiająca maskę dla aktualnego procesu.

Parametr *how* definiuje sposób uaktualnienia maski sygnałów. Może przyjmować następujące wartości:

- SIG_BLOCK - nowa maska to połączenie maski starej i nowej (new_set - zbiór sygnałów, które chcemy blokować).
- SIG_UNBLOCK - maska podana jako argument to zbiór sygnałów, które chcemy odblokować.
- SIG_SETMASK - nadpisujemy starą maskę nową.

Do parametru old_set zostanie zapisana poprzednia maska.

Przykłady wywołania

```
#include <signal.h>

sigset_t newmask;/* sygnały do blokowania */
sigset_t oldmask; /* aktualna maska sygnałów */
sigemptyset(&newmask); /* wyczyść zbiór blokowanych sygnałów */
sigaddset(&newmask, SIGINT); /* dodaj SIGINT do zbioru */
/* Dodaj do zbioru sygnałów zablokowanych */
if (sigprocmask(SIG_BLOCK, &newmask, &oldmask) < 0)
    perror("Nie udało się zablokować sygnału");
/* tutaj chroniony kod */
if (sigprocmask(SIG_SETMASK, &newmask, NULL) < 0)
    perror("Nie udało się przywrócić maski sygnałów")};
```

```
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>

sigset_t oldmask, blockmask;
pid_t mychild;
sigfillset(&blockmask);
if (sigprocmask(SIG_SETMASK, &blockmask, &oldmask) == -1) {
    perror("Nie udało się zablokować wszystkich sygnałów");
    exit(1);
}
if ((mychild = fork()) == -1) {
    perror("Nie powołano procesu potomnego");
    exit(1);}
else if (mychild == 0) {
        if (sigprocmask(SIG_SETMASK, &oldmask, NULL) == -1){
            perror("Proces potomny nie odtworzył maski sygnałów");
            exit(1);
        }
    /* .....kod procesu potomnego ..... */
    } else {
        if (sigprocmask(SIG_SETMASK, &oldmask, NULL) == -1){
            perror("Proces macierzysty nie odtworzył maski sygnałów ");
            exit(1);
        }
        /* ..... kod procesu macierzystego..... */ }
```

Czekanie na określony sygnał

```
int sigpending(sigset_t *set);
```

Służy do odczytania listy sygnałów, które oczekuję na odblokowanie w danym procesie ((ang. *pending signals*)). Do zmiennej set zapisywany jest zestaw oczekujących sygnałów.

```
int sigsuspend(cost sigset_t *set);
```

Służy do odebrania sygnału oczekującego

Tymczasowo zastępuje procesową maskę sygnałów na tę wskazaną parametrem set, a także wstrzymuje działanie procesu do momentu, kiedy nadejdzie odblokowany sygnał. Po obsłudze sygnału ponownie jest ustawiana maska sprzed wywołania `sigsuspend`.

Zwraca -1 jeśli otrzymany sygnał nie powoduje zakończenia procesu.

Dziedziczenie

- Po wykonaniu funkcji fork proces potomny dziedziczy po swoim przodku wartości maski sygnałów i ustalenia dotyczące obsługi sygnałów.
- Nieobsłużone sygnały procesu macierzystego są czyszczone.
- Po wykonaniu funkcji exec maska obsługi sygnałów i nieobsłużone sygnały są takie same jak w procesie, w którym wywołano funkcję exec.