Sygnał - Forma komunikacji między procesami w systemie Unix. Jest to pewien komunikat wysyłany do procesu, oznaczający że, proces ma chwilowo przerwać wykonywanie. Pojawia się on asynchronicznie, tzn. w dowolnym momencie w czasie pracy procesu, niezależnie od tego co dany proces akurat robi. Otrzymanie sygnału zazwyczaj znaczy, że nastąpiło jakieś zdarzenie wyjątkowe, które wymaga od procesu natychmiastowej reakcji. Sygnały są zazwyczaj używane do takich zadań jak kończenie działania procesów, czy informowanie demonów, że mają ponownie odczytać pliki konfiguracyjne. Procesy otrzymują sygnały od jądra, innego procesu lub użytkownika.

Możliwe reakcje procesu na otrzymany sygnał:

1. Wykonanie akcji domyślnej. Najczęściej zakończenie procesu i ewentualny zrzut zawartości segmentów pamięci na dysk.
2. Zignorowanie sygnału.
3. Podjęcie akcji zdefiniowanej przez użytkownika, tzw. przechwycenie sygnału.
4. Maskowanie- blokowanie sygnału tak, aby nie był dostarczany.

Każdy sygnał ma przypisany numer oraz nazwę zaczynającą się od "SIG". Nazwy te są zdefiniowane w pliku "signal.h". Po obsłużeniu sygnału przerwany proces kontynuuje działanie od miejsca przerwania (o ile nie został zakończony). 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.

Definiowanie własnych zestawów sygnałów

Definiowanie zestawów(zbiorów) sygnałów jest jedną z podstawowych operacji podczas programowania użyciem sygnałów. Zestawy definiujemy, aby pogrupować pewną ilość różnych sygnałów, co umożliwi wykonywanie pewnych działań na grupie. Typem definiującym zestaw sygnałów jest **sigset_t**. Wewnątrz struktury **sigset_t** każdemu sygnałowi przypisany jest jeden bit (prawda lub fałsz).

Definicja typu sigset_t z pliku asm/signal.h:

```
typedef unsigned long sigset_t;          /* co najmniej 32 bity */
```

Typ sigset_t służy do trzymania zbioru sygnałów (każdemu sygnałowi odpowiada dokładnie jeden bit). Jak widać jest to liczba co najmniej 32-bitowa, stąd stała ilość sygnałów definiowana jest w następujący sposób:

```
#define _NSIG       32
#define NSIG  _NSIG
```

Funkcje wykorzystywane podczas działań na zestawach sygnałów:

1. int sigemptyset (sigset_t *zbiór_sygnałów);

Funkcja dokonuje inicjalizacji pustego zestawu. Zawsze zwraca 0.

2. int sigfillset (sigset_t *zbiór_sygnałów);

Funkcja dokonuje inicjalizacji zestawu i wpisuje do niego wszystkie sygnały używane w systemie. Zwracana wartość to 0.

3. int sigaddset (sigset_t *zbiór_sygnałów, int nr_sygnału);

Funkcja dodaje sygnał *nr_*sygnału do zbioru sygnałów *zbiór_sygnałów.* (Dodawanie pojedynczych sygnałów do zestawu.) Nie blokuje ani odblokowuję żadnych sygnałów.

The return value is 0 on success and -1 on failure (w przypadku niepowodzenia). The following *errno* error condition is defined for this function:
EINVAL

Parametr *nr_sygnału* nie określa prawidłowego sygnału.

4. int sigdelset (sigset_t *zbiór_sygnałów, int nr_sygnału);

Funkcja usuwa sygnał *nr_sygnału* ze zbioru sygnałów *zbiór_sygnałów.* (Usuwanie pojedynczych sygnałów z zestawu.) Nie blokuje ani odblokowuję żadnych sygnałów.

The return value is 0 on success and -1 on failure. The following *errno* error condition is defined for this function:
EINVAL

Parametr *nr_sygnału* nie określa prawidłowego sygnału.

5. int sigismember (sigset_t *zbiór_sygnałów, int nr_sygnału);

Funkcja sprawdza, czy w zestawie istnieje dany sygnał.

It returns 1 if the signal is in the set, 0 if not, and -1 if there is an error.

The following *errno* error condition is defined for this function:

EINVAL

Parametr *nr_sygnału* nie określa prawidłowego sygnału.

Aby wyświetlić listę wszystkich sygnałów, które obsługuje Twój system linux musisz wywołać komendę kill z flaga "-l" w ten sposób:

maar@computer:~ $ kill -l

Funkcje systemowe

1. 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.

2. int kill(int pid, int sig)

- służy do wysyłania sygnału.

DEFINICJA: int kill(int pid, int sig)
gdzie:
pid - pid procesu do którego wysyłam sygnał
sig - nr. sygnalu
WYNIK: 0 w przypadku sukcesu
-1 gdy nastąpił błąd; wtedy errno przybiera następujące wartości:
EINVAL błędny parametr (nr sygnału)
ESRCH nie znaleziony proces do zabicia
EPERM proces nie posiada praw do wykonania tej funkcji

Drugim argumentem funkcji jest numer sygnału, który ma być wysłany do procesu lub grupy procesów opisanych przez pierwszy argument. W zależności od wartości argumentu pid zmienia sie adresat:

- gdy pid= 0, wysyłany jest sygnał do grupy aktualnego procesu
- gdy pid=-1, wysyłany jest sygnał do wszystkich procesów prócz własnego (próby wysyłania)
- gdy pid<-1, wysyłany jest sygnał do grupy -pid
- gdy pid> 0, wysyłany jest sygnał do procesu o numerze pid

Niezależnie od rodzaju parametrów funkcja korzysta z funkcji jadra *send_sig* z parametrem perm równym 0. Stad błąd EPERM wystąpi tylko wtedy, gdy wystąpiłby w przypadku wywołania funkcji *send_sig*.

3. int sigprocmask(int how, sigset_t *set , sigset_t *oset)

- służy do ustawiania maski blokowania sygnałów.

DEFINICJA: int sigprocmask(int how, sigset_t *set , sigset_t *oset)
gdzie:
how - sposób zmiany maski,
set - wskaźnik na nowa maskę blokowania sygnałów,
oset - zbiór sygnałów przeznaczony do zapamiętania starej maski.
WYNIK: 0 w przypadku sukcesu
-1 gdy wystąpił błąd; ustawia ERRNO na:
EINVAL gdy how miało wartość nieznanej opcji;

Pierwszym argumentem funkcji jest numer sposobu ustawiania maski. Może przybrać wartość jednej ze stałych zdefiniowanych w asm/signal.h:

#define SIG_BLOCK 0 - nowa maska będzie połączeniem starej maski i maski podanej w set (maska podana w set zawiera zbiór sygnałów, które chcemy zablokować)
#define SIG_UNBLOCK 1 - maska podana w set zawiera zbiór sygnałów, które chcemy odblokować
#define SIG_SETMASK 2 - stara maska jest nadpisywana maską zawartą w set

Gdy wskaźnik oset jest niezerowy to jest tam wpisywana stara maska blokowania sygnałów sprzed wywołania tej funkcji.

Kiedy jądro próbuje przekazać do procesu sygnał, który aktualnie jest zablokowany(nie dotyczy SIGKILL i SIGSTOP), to sygnał jest przechowywany do czasu, aż proces ustawi ignorowanie dla tego sygnału lub odblokuje ten sygnał.

Z wnętrza procesu możemy odczytać listę sygnałów, które oczekują na odblokowanie przez niego. Służy do tego funkcja sigpending():

4. int sigpending(sigset_t *set)

- pobiera zbiór wstrzymywanych (przez maskę blokowania) sygnałów .

DEFINICJA: int sigpending(sigset_t *set)
WYNIK: 0 w przypadku sukcesu
-1 gdy nastąpił błąd zapisu pod adres set

Zbiór blokowanych sygnałów jest wstawiany pod set.

5. int sigaction(int signum , const struct sigaction *action , struct sigaction *oldaction)

- ustawia nowa akcje związana z danym sygnałem. Nie tylko zmienia obsługę sygnału na stałe (do następnej świadomej zmiany), ale również ustawia jego blokowanie na czas wykonywania funkcji obsługi. Pozwala ponadto zablokować w tym czasie inne sygnały.

DEFINICJA: int sigaction(int signum , const struct sigaction *action , struct sigaction *oldaction)
WYNIK: 0 w przypadku sukcesu
-1 gdy nastąpił błąd (errno=EINVAL)
gdzie:
signum - numer sygnału,
action - struktura typu sigaction opisująca nowy sposób obsługi,
oldaction - struktura typu sigaction przeznaczona do zapamiętania starego sposobu obsługi.

Struktura sigaction wygląda następująco:

```
struct sigaction{

void (*sa_handler)();
sigset_t sa_mask;
int sa_flags;
};
```

Jest to rozszerzona wersja funkcji signal, służąca do tego samego czyli do zmiany dyspozycji sygnału. Zmienna sa_handler zawiera wskaźnik do funkcji obsługi sygnału. 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. Funkcja **sigaction**() blokuje wybrane sygnały jedynie na czas obsługi nadesłanego sygnału. Istnieje możliwość ustawienia maski blokowanych sygnałów na dowolnie długi okres aż do ponownej zmiany przy pomocy funkcji: **int sigprocmask(**int how, sigset_t *set , sigset_t *oset**).**

http://student.agh.edu.pl/~okarmus/referat.html

http://www.gnu.org/software/libc/manual/html_node/Signal-Sets.html

http://students.mimuw.edu.pl/SO/Linux/Temat01/temat0152.html

http://home.elka.pw.edu.pl/~edobkows/USUX/lab10_fun_sys/LE2.pdf