

3 Sygnały

3.1 Opis sygnałów

Najprostszą metodą komunikacji międzyprocesowej w systemie UNIX są **sygnały**. Umożliwiają one *asynchroniczne przerwanie* działania procesu przez inny proces lub jądro, aby przerwany proces mógł zareagować na określone zdarzenie. Można je traktować jako software'owe wersje przerwania sprzętowych. Sygnały mogą pochodzić z różnych źródeł:

- **Od sprzętu** – np. gdy jakiś proces próbuje uzyskać dostęp do adresów spoza własnej przestrzeni adresowej lub kiedy zostanie w nim wykonane dzielenie przez zero.
- **Od jądra** – są to sygnały powiadamiające proces o dostępności urządzeń wejścia-wyjścia, na które proces czekał, np. o dostępności terminala.
- **Od innych procesów** – proces potomny powiadamia swego przodka o tym, że się zakończył.
- **Od użytkowników** – użytkownicy mogą generować sygnały zakończenia, przerywania lub stopu za pomocą klawiatury (sekwencje klawiszy generujące sygnały można sprawdzić komendą `stty -a`).

Sygnały mają przypisane nazwy zaczynające się od sekwencji SIG i są odpowiednio ponumerowane – szczegółowy opis dla danego systemu można znaleźć w: `man 7 signal`. Definicje odpowiednich stałych znajdują się w pliku nagłówkowym `<signal.h>` (lub plikach włączanych w nim). Listę ważniejszych sygnałów przedstawia Tablica 1. Proces, który otrzymał sygnał może zareagować na trzy sposoby:

1. **Wykonać operację domyślną.** Dla większości sygnałów domyślną reakcją jest zakończenie działania procesu, po uprzednim powiadomieniu o tym procesu macierzystego. Czasem generowany jest plik rzutu (ang. *core*), tzn. obraz pamięci zajmowanej przez proces.
2. **Zignorować sygnał.** Proces może to zrobić w reakcji na wszystkie sygnały z wyjątkiem dwóch: SIGSTOP (zatrzymanie procesu) i SIGKILL (bezw warunkowe zakończenie procesu). Dzięki niemożności ignorowania tych dwóch sygnałów system operacyjny jest zawsze w stanie usunąć niepożądane procesy.
3. **Przechwycić sygnał.** Przechwycenie sygnału oznacza wykonanie przez proces specjalnej procedury obsługi – po jej wykonaniu proces może powrócić do swego zasadniczego działania (o ile jest to właściwe w danej sytuacji). Podobnie jak ignorować, przechwytywać można wszystkie sygnały z wyjątkiem: SIGSTOP i SIGKILL.

Proces potomny dziedziczy po swoim przodku mechanizmy reagowania na wybrane sygnały. Jeżeli jednak potomek uruchomi nowy program przy pomocy funkcji `exec`, to przywrócone zostają domyślne procedury obsługi sygnałów.

Sygnal	Opis	Domyślna akcja
SIGABRT	Wysyłany przez funkcję <code>abort</code>	Zakończenie, zrzut
SIGALRM	Minał czas ustawiony przez funkcję <code>alarm</code>	Zakończenie
SIGBUS	Błąd sprzętowy (szyny)	Zakończenie, zrzut
SIGCHLD	Zakończenie procesu potomnego	Ignorowanie
SIGCONT	Uruchomienie po wstrzymaniu	Ignorowanie
SIGHUP	Zakończenie procesu sterującego terminalem	Zakończenie
SIGFPE	Wyjątek arytmetyczny	Zakończenie, zrzut
SIGILL	Nielegalna instrukcja	Zakończenie, zrzut
SIGINT	Przerwanie z klawiatury [Ctrl-C]	Zakończenie
SIGKILL	Bezwarunkowe zakończenie procesu (nie może być zignorowany ani przechwycony)	Zakończenie
SIGQUIT	Sekwencja wyjścia z klawiatury [Ctrl-\\]	Zakończenie, zrzut
SIGPIPE	Proces pisze do potoku, z którego nikt nie czyta	Zakończenie
SIGSEGV	Naruszenie ograniczeń pamięci	Zakończenie, zrzut
SIGSTOP	Zatrzymanie procesu – bez zakończenia (nie może być zignorowany ani przechwycony)	Zatrzymanie procesu
SIGTERM	Żądanie zakończenia	Zakończenie
SIGTSTP	Sekwencja zatrzymania z klawiatury [Ctrl-Z]	Zatrzymanie procesu
SIGTTIN	Proces w tle czyta z terminala sterującego	Zatrzymanie procesu
SIGTTOU	Proces w tle pisze na terminal sterujący	Zatrzymanie procesu
SIGUSR1	Sygnal użytkownika nr 1	Zakończenie
SIGUSR2	Sygnal użytkownika nr 2	Zakończenie

Tablica 1: *Lista ważniejszych sygnałów.*

3.2 Wysyłanie sygnałów

Do wysyłania sygnałów do procesów i ich grup służy funkcja systemowa `kill`. Parametr `pid`

Pliki włączane	<sys/types.h>, <signal.h>		
Prototyp	<code>int kill(pid_t pid, int sig);</code>		
Zwracana wartość	Sukces	Porażka	Czy zmienia <code>errno</code>
	0	-1	Tak

określa proces lub grupę procesów, do których zostanie wysłany sygnał – jego znaczenie objaśnia poniższa tabelka.

Wartość <code>pid</code>	Jakie procesy odbierają sygnał
<code>> 0</code>	Proces o <code>PID = pid</code>
<code>= 0</code>	Procesy należące do samej grupy co proces wysyłający sygnał
<code>< -1</code>	Procesy należące do grupy o <code>PGID = -pid</code>

Parametr `sig` oznacza numer wysyłanego sygnału (można używać nazw symbolicznych). Jeżeli `sig = 0`, to funkcja `kill` nie wysyła sygnału, ale wykonuje test błędów.

Z poziomu powłoki sygnały można wysyłać za pomocą polecenia:

`kill -sig pid`

Znaczenie parametrów `sig` i `pid` jest takie jak powyżej. Listę nazw symbolicznych sygnałów dla polecenia `kill` można uzyskać wykonując: `kill -l` (nazwy te różnią się od opisanych powyżej nazw sygnałów tym, że pominięto w nich człon `SIG`).

Sygnał `SIGALRM` można wysłać posługując się funkcją systemową `alarm`. Funkcja ta generuje sygnał kiedy minie ustalona liczba sekund przekazana przez parametr `sec`. Jeżeli `sec = 0`, to czasomierz zostanie wyzerowany.

Pliki włączane	<unistd.h>		
Prototyp	unsigned alarm(unsigned sec);		
Zwracana wartość	Sukces	Porażka	Czy zmienia <code>errno</code>
	Liczba pozostałych sekund		Nie

3.3 Obsługa sygnałów

Pliki włączane	<signal.h>		
Prototyp	void (*signal(int sig, void (*handler)(int)))(int);		
Zwracana wartość	Sukces	Porażka	Czy zmienia <code>errno</code>
	Poprzednia dyspozycja sygnału	<code>SIG_ERR (-1)</code>	Tak

Do modyfikowania sposobu, w jaki proces zareaguje na sygnał można użyć funkcji `signal`. Prototyp tej funkcji na pierwszy rzut oka wywołuje często niemałą konsternację. Łatwiej go zrozumieć posługując się pomocniczą definicją typu `sighandler_t` będącego wskaźnikiem do funkcji:

```
typedef void (*sighandler_t)(int);
sighandler_t signal(int sig, sighandler_t handler);
```

Pierwszym parametrem funkcji `signal` jest numer sygnału, który ma być obsługiwany – za wyjątkiem `SIGKILL` i `SIGSTOP`. Drugim parametrem natomiast jest wskaźnik do funkcji, która ma być wywołana w chwili przybycia sygnału. Funkcja ta może być określona stałymi `SIG_DFL`, `SIG_IGN` lub zdefiniowana przez użytkownika. Stała `SIG_DFL` oznacza domyślną obsługę sygnału, natomiast `SIG_IGN` ignorowanie sygnału. Funkcja do obsługi sygnału ma jeden parametr typu `int`, do którego zostanie automatycznie wstawiony numer sygnału.

Aby spowodować oczekiwanie procesu na pojawienie się sygnału można posłużyć się funkcją biblioteczną `pause`. Funkcja ta zawiesza proces do czasu odebrania sygnału, który nie został zignorowany. Funkcja `pause` wraca tylko w przypadku przechwycenia sygnału i powrotu funkcji obsługi sygnału; zwraca wtedy wartość `-1` i ustawia zmienną `errno` na `EINTR`.

Przykładowe wywołania funkcji `signal` powodujące ignorowanie sygnału `SIGQUIT` oraz włączenie własnej obsługi sygnału `SIGINT` przy pomocy funkcji `my_sighandler` mogą wyglądać następująco:

Pliki włączane	<unistd.h>		
Prototyp	int pause(void);		
Zwracana wartość	Sukces	Porażka	Czy zmienia <code>errno</code>
	−1, jeśli sygnał nie powoduje zakończenia procesu	Nie zwraca nic	Tak

```

void my_sighandler(int);
...
if (signal(SIGQUIT,SIG_IGN) == SIG_ERR){
    perror("Funkcja signal ma problem z SIGQUIT");
    exit(EXIT_FAILURE);
}
...
if (signal(SIGINT,my_sighandler) == SIG_ERR){
    perror("Funkcja signal ma problem z SIGINT");
    exit(EXIT_FAILURE);
}
...

```

Oczywiście, funkcja `my_sighandler` musi zostać zdefiniowana przez użytkownika, zgodnie z jego życzeniem obsługi sygnału `SIGINT`.

Funkcja `signal` występuje we wszystkich wersjach systemu UNIX, ale niestety nie jest niezawodna (może nie obsłużyć poprawnie wielu sygnałów, które następują w krótkim czasie po sobie). Dlatego w standardzie POSIX wprowadzono dodatkową, niezawodną funkcję do obsługi sygnałów – o nazwie `sigaction`, ale jest ona niestety bardziej skomplikowana w użyciu od funkcji `signal` (szczegóły można znaleźć w podręczniku `man`).

ĆWICZENIE 3: WYSYŁANIE I OBSŁUGA SYGNAŁÓW

Napisać program do obsługi sygnałów z możliwościami: (1) wykonania operacji domyślnej, (2) ignorowania oraz (3) przechwycenia i własnej obsługi sygnału (np. numer sygnału oraz opcję obsługi przekazywać za pomocą argumentów wywołania programu).

- Uruchomić program i wysyłać do niego sygnały przy pomocy sekwencji klawiszy oraz przy pomocy polecenia `kill`.
- Uruchomić powyższy program poprzez funkcję `exec` w procesie potomnym innego procesu i wysyłać do niego sygnały poprzez funkcję systemową `kill` z procesu macierzystego.
- Uruchomić grupę kilku procesów i wysyłać sygnały do całej grupy procesów.