Sygnały

Joanna Reda i Tomasz Maciejak
Wydział Matematyki Informatyki i Mechaniki Uniwersystetu Warszawskiego

10 stycznia 2001

Spis treści

1 Wstęp

1.1 Ogólnie

Sygnały służą do informowania procesu, że zaszło jakieś zdarzenie. Proces nie wie kiedy pojawi się jakiś sygnał (są one wysyłane asynchronicznie), nie wie kto jest nadawcą, nie wie również ile sygnałów danego typu zostało do niego wysłanych. Nadawcą sygnału może być jądro lub inny proces.

Po odebraniu sygnału proces może:

- zignorować sygnał
- zakończyć się
- dokonać zrzutu pamięci (*ang. dump core*) i zakończyć się
- zatrzymać się
- przechwycić sygnał tj. dostarczyć funkcję (*ang. signal handler*), która będzie wywołana po odebranu danego synału

Oczywiście nie wszystkie sygnały mogą być przechwytywane lub ignorowane, gdyż adminstrator musi mieć możliwość zakończenia dowolnego procesu.

1.2 Tablica sygnałów

Nazwa procesu	Wartość	Czynność domyślna	Znaczenie
SIGHUP	1	zakończenie procesu	zawieszenie
SIGINT	2	zakończenie procesu	przerwanie z klawiatury
SIGQUIT	3	zakończenie procesu i zrzut pamięci	znak zakończenia
SIGILL	4	zakończenie procesu i zrzut pamięci	niedozwolona instrukcja
SIGTRAP	5	zakończenie procesu i zrzut pamięci	pułapka śledzenia
SIGABRT	6	zakończenie procesu i zrzut pamięci	błąd sprzętowy IOT
SIGIOT	6	zakończenie procesu i zrzut pamięci	błąd sprzętowy IOT
SIGBUS	7	zakończenie procesu i zrzut pamięci	błąd szyny
SIGFPE	8	zakończenie procesu i zrzut pamięci	błąd sprzętowy FPE
SIGKILL	9	zakończenie procesu	zakończenie procesu
SIGUSR1	10	zakończenie procesu	zdef. przez użytkownika
SIGSEGV	11	zakończenie procesu i zrzut pamięci	naruszenie segmentacji
SIGUSR2	12	zakończenie procesu	zdef. przez użytkownika
SIGPIPE	13	zakończenie procesu	dane nie są odbierane z łącza kom.
SIGALRM	14	zakończenie procesu	budzik
SIGTERM	15	zakończenie procesu	programowe zakończenie procesu
SIGSTKFLT	16	zakończenie procesu	błąd stosu na koprocesorze
SIGCHLD	17	ignorowanie	zakończenie procesu potomnego
SIGCONT	18		wznowienie zatrzymanego procesu
SIGSTOP	19	zatrzymanie procesu	zatrzymanie
SIGTSTP	20	zatrzymanie procesu	wprowadzenie znaku zatrzymania
SIGTTIN	21	zatrzymanie procesu	czytanie w tle z terminala sterującego
SIGTTOU	22	zatrzymanie procesu	pisanie w tle na terminalu sterującym
SIGURG	23	ignorowanie	dane wysokopriorytetowe w gnieźnie
SIGXCPU	24	zakończenie procesu i zrzut pamięci	przekroczenie czasu procesora
SIGXFSZ	25	zakończenie procesu i zrzut pamięci	przekroczenie rozmiaru pliku
SIGVTALRM	26	zakończenie procesu	budzik zegara wirtualnego
SIGPROF	27	zakończenie procesu	budzik profilometra
SIGWINCH	28	ignorowanie	zmiana rozmiaru okna
SIGIO	29	zakończenie procesu	we-wy dozwolone dla deskr. pliku
SIGPOLL	29	zakończenie procesu	zdarzenie dotyczące urządzenia strum.
SIGPWR	30	zakończenie procesu	brak zasilania
SIGUNUSED	31	zakończenie procesu	sygnał nieużywany

Dodatkowe informacje:

Sygnałow SIGKILL oraz SIGSTOP nie można przechwycić ani zignorować.

2 Struktury danych

2.1 Struktura `sigaction`

```
struct sigaction {
__sighandler_t sa_handler;
unsigned long sa_flags;
void (*sa_restorer)(void);
sigset_t sa_mask;
};
```

- `void (*sa_handler)(int num)` - opisuje akcję związaną z sygnałem o numerze `num` i może być równa SIG_DFL - domyślna akcja, SIG_IGN - ignorowanie sygnału lub wskaźnikowi do funkcji obsługi sygnału
- `sigset_t sa_mask` - maska sygnałów blokowanych w trakcie wykonywania funkcji obsługi sygnału (`handlera`), dodatkowo sygnał, który spowodował wywołanie funkcji zostanie zablokowany, chyba że są ustawione w `sa_flags` flagi SA_NODEFER lub SA_NOMASK
- `int sa_flags` - flagi opisujące specyficzne zachowania podczas obsługi przerwania

  Najważniejsze z tych flag to:

  – SA_ONESHOT lub SA_RESETHAND - restartuj obsługę do domyślnej po pierwszym obsłużeniu z pomocą handlera
  – SA_RESTART - ustaw zgodność z semantyką BSD tzn. jeśli funkcja systemowa została przerwana to jest ponawiana
  – SA_NOMASK lub SA_NODEFER - nie zapobiegaj otrzymywaniu sygnału wewnątrz własnego handlera
  – SA_NOCLDSTOP - jeśli numer sygnału to SIGCHLD, to nie otrzymuj potwierdzenia, gdy dziecko zmienia stan na STOPPED (po otrzymaniu sygnału SIGSTOP, SIGTSTP, SIGTTIN lub SIGTTOU)

2.2 Struktura `task_struct` (pola dotyczące sygnałów)

- `int sigpending` - równe 0 jeśli brak sygnałów nieblokowanych, otrzymanych i nie obsłużonych
- `int exit_signal` - numer sygnału wysyłanego, gdy proces się kończy (domyślnie SIGCHLD)
- `int pdeath_signal` - numer sygnału wysyłanego, gdy rodzic umiera - potrzebny aby jądro mogło podczepić dzieci do procesu `init`
- `struct signal_struct *sig` - tablica opisująca zachowanie procesu po otrzymaniu sygnału
- `sigset_t signal` - poszczególne bity są ustawione, gdy odpowiadające im sygnały przyszły do procesu
- `sigset_t blocked` - poszczególne bity są ustawione, gdy odpowiadające im sygnały są blokowane
- `signal_queue *sigqueue, **sigqueue_tail` - wskaźniki odpowiednio do kolejki sygnałów i jej ogona

3 Wysyłanie sygnałów

3.1 Funkcja `send_sig_info`

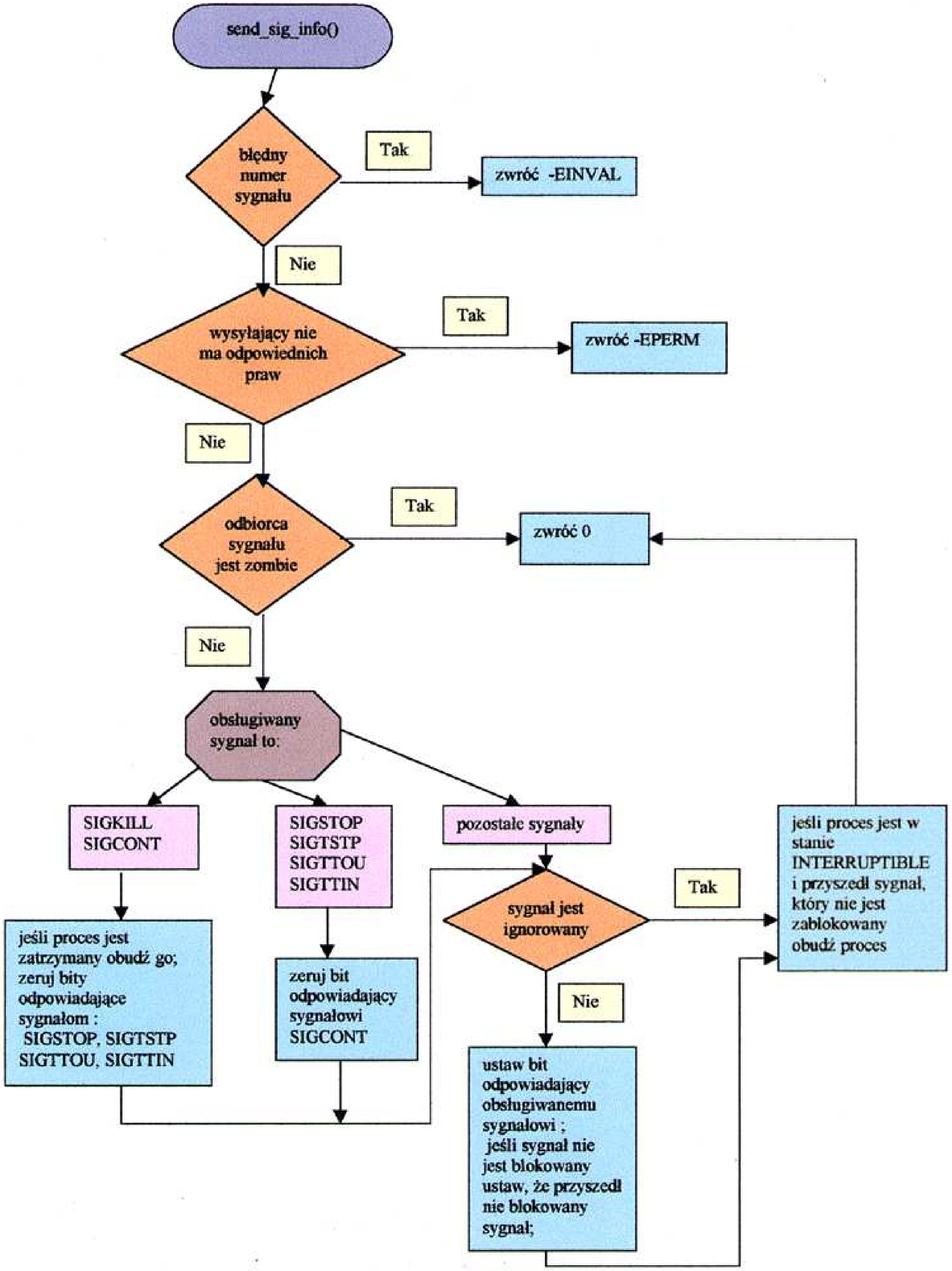
```
int send_sig_info(int sig, struct siginfo *info, struct struct_task *t)
```

Funkcja służy do wysłania sygnału.
Parametry:

- sig - numer wysyłanego sygnału
- info - informacje o wysyłającym //sygnał jest wysyłany przez bieżący proces (`current`)
- t - proces do którego jest wysyłany sygnał

Treść funkcji:

```
if (błędny numer sygnału) return -EINVAL;
if (wysyłający nie ma praw do wysłania sygnału) return -EPERM;
if (!sig lub odbiorca sygnału jest zombie) return 0;
switch(sig)
  case SIGKILL: case SIGCONT:
    if (proces t jest zatrzymany ) obudź proces t;
    zeruj bity odpowiadające sygnałom: SIGSTOP, SIGTSTP, SIGTTOU, SIGTTIN;
    break;
  case SIGSTOP: case SIGTSTP; case SIGTTOU; case SIGTTIN
    zeruj bit odpowiadający sygnałowi SIGCONT;
    break;
//koniec switch
if (t ignoruje sygnał sig)
if (t jest w stanie TASK_INTERRUPTIBLE oraz przyszedł sygnał, który nie jest blokowany) obudź proces t i kończy się wykonywanie funkcji;
  ustaw bit odpowiadający sygnałowi sig u procesu t;
if (sygnał sig nie zablokowany)
  ustaw, że przyszedł sygnał, który nie jest blokowany (sigpending);
if (t jest w stanie TASK_INTERRUPTIBLE )
  obudź proces t;
return 0;
```


send_sig_info()
błędny numer sygnału
Tak
zwróć -EINVAL
Nie
wysyłający nie ma odpowiednich praw
Tak
zwróć -EPERM
Nie
odbiorca sygnału jest zombie
Tak
zwróć 0
Nie
obsługiwany sygnał to:
SIGKILL SIGCONT
SIGSTOP SIGTSTP SIGTTOU SIGTTIN
pozostałe sygnały
jeśli proces jest zatrzymany obudź go; zeruj bity odpowiadające sygnałom : SIGSTOP, SIGTSTP SIGTTOU, SIGTTIN
zeruj bit odpowiadający sygnałowi SIGCONT
sygnał jest ignorowany
Tak
Nie
ustaw bit odpowiadający obsługiwanemu sygnałowi ; jeśli sygnał nie jest blokowany ustaw, że przyszedł nie blokowany sygnał;
jeśli proces jest w stanie INTERRUPTIBLE i przyszedł sygnał, który nie jest zablokowany obudź proces

3.2 Funkcja `force_sig_info`

`int force_sig_info(int sig,struct siginfo *info,struct struct_task *t)`

Funkcja służy do bezwarunkowego wysłania sygnału

Parametry jak w funkcji `send_sig_info`

1. if (proces t jest zombie) return -ESRCH;
2. if (proces t ma ustawione ignorowanie sygnału sig) ustaw na obsługę domyślną;
3. if (sygnał sig jest blokowany) odblokuj;
4. wywołaj funkcję `send_sig_info`;

3.3 Funkcje `kill_pg_info`, `kill_sl_info`, `kill_proc_info`

Kolejne trzy funkcje sprawdzają poprawność parametrów i wysyłają sygnał (przy pomocy `send_sig_info` do odpowiedniego procesu/procesów.

- `int kill_pg_info(int sig,struct siginfo *info,pid_t pgrp)` - wysyła sygnał do procesów z grupy pgrp
- `int kill_sl_info(int sig,struct siginfo *info,pid_t sess)` - wysyła sygnał do lidera grupy procesów
- `int kill_proc_info(int sig, struct siginfo *info,pid_t pid)` - wysyła sygnał do procesu o pidzie pid

3.4 Funkcja `kill_something_info`

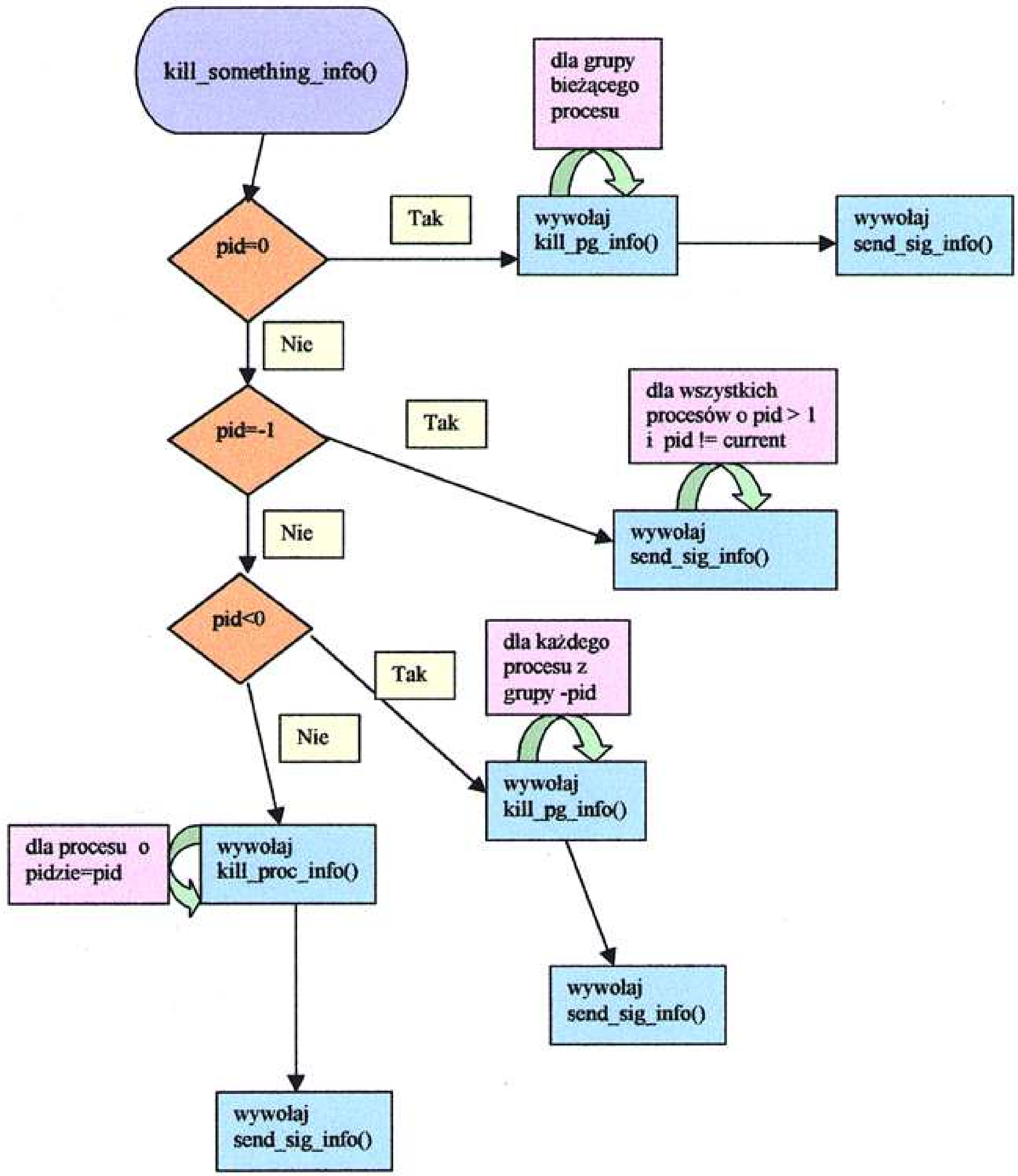
`int kill_something_info(int sig,struct siginfo,int pid)`

Wysyła sygnał do procesu lub grupy procesów przy pomocy funkcji (w zależności od parametru pid): `kill_pg_info`, `kill_proc_info`, `send_sig_info`.

Zwracana wartość i błędy:

Jeśli wszystko się powiodło funkcja zwraca 0, w przeciwnym przypadku:

- EINVAL - podano błędny numer sygnału
- ESRCH - nie istnieje proces o podanym numerze lub o podanym numerze grupy
- EPERM - proces wysyłający nie ma prawa do wysyłania sygnału


kill_something_info()
pid=0
Tak
wywołaj
kill_pg_info()
dla grupy
bieżącego
procesu
wywołaj
send_sig_info()
Nie
pid=-1
Tak
dla wszystkich
procesów o pid > 1
i pid != current
wywołaj
send_sig_info()
Nie
pid<0
Tak
dla każdego
procesu z
grupy -pid
wywołaj
kill_pg_info()
wywołaj
send_sig_info()
Nie
dla procesu o
pidzie=pid
wywołaj
kill_proc_info()
wywołaj
send_sig_info()

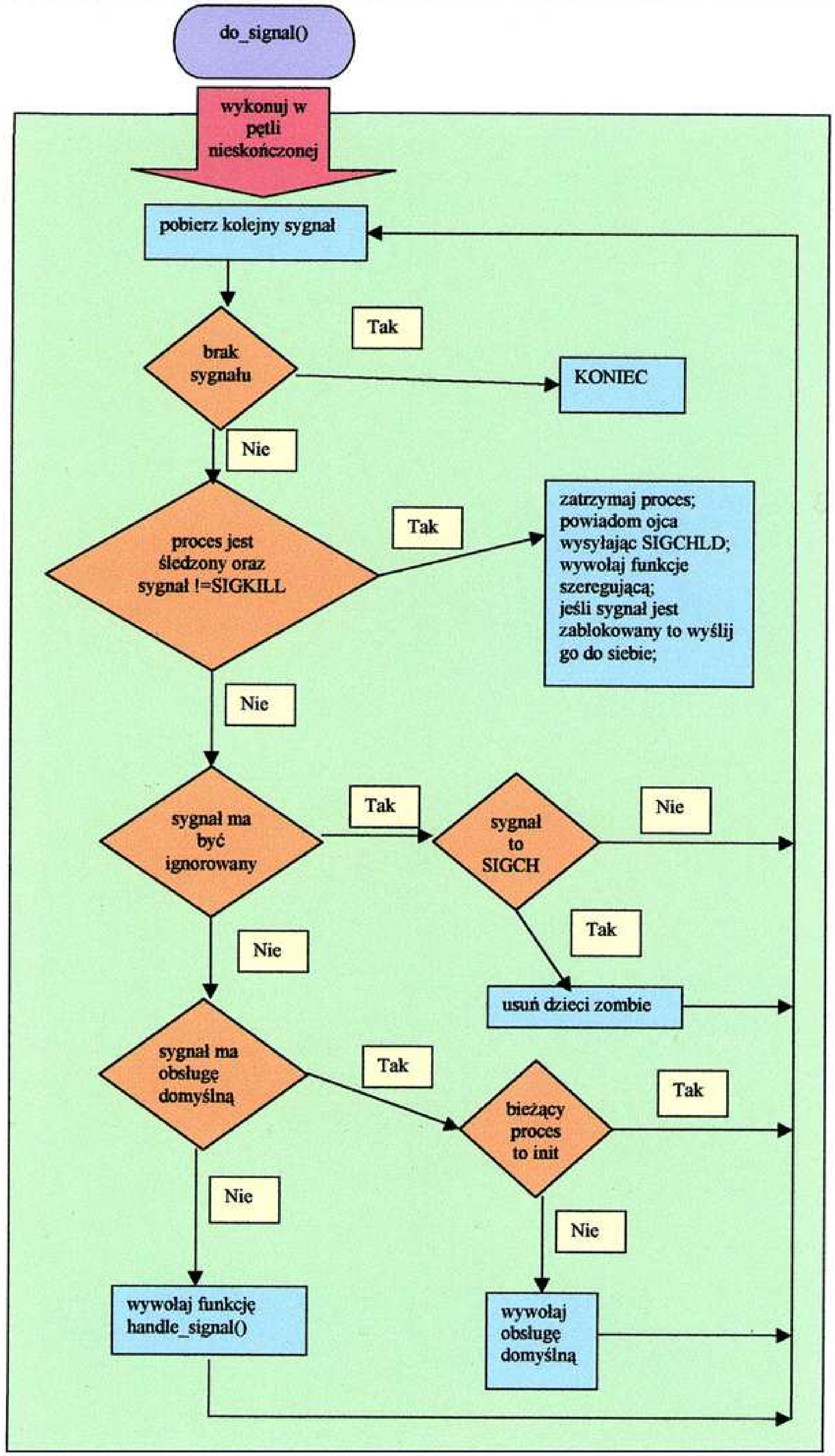
4 Odbieranie sygnałów

4.1 Funkcja `do_signal`

Funkcja `do_signal` służy do obsługi sygnałów, które zostały wysłane do procesu, ale jeszcze nie zostały obsłużone. Jest ona wywoływana przy powrocie procesu z trybu jądra do trybu użytkownika (podczas powrotów z funkcji systemowych lub przy powrocie z przerwania).

```
int do_signal(struct ps_regs *regs, sigset_t *oldset)
```

```
if (!oldset) oldset=stara maska sygnałów;
for ( ; ; )
  signr = kolejny sygnał z kolejki sygnałów;
  if (brak sygnałów) break;
  if (proces jest śledzony) && (signr!= SIGKILL)
    patrz sekcja dotycząca debugera
  if (signr ma być ignorowany)
    if (signr != SIGCHLD) continue;
    usuń wszystkie dzieci zombie; //signr==SIGCHLD
    continue;
  if (signr ma mieć obsługę domyślną)
    kod_wyjścia = signr;
    if (bieżący proces to init) continue;//init nie obsługuje sygnałów
    switch (signr)
      case SIGCONT: case SIGCHLD: case SIGWINCH continue;
      case SIGTSTP: case SIGTTIN: case SIGTTOU :
        if (jest procesem osieroconym) continue;
      case SIGSTOP:
        zatrzymaj proces;
        kod_wyjścia = signr;
        if (ojciec chce, żeby go zawiadamiać o zatrzymaniu dziecka) zawiadom ojca;
        wywołaj funkcję szeregującą;
      case SIGQUIT; case SIGILL; case SIGTRAP;case SIGABRT; case SIGFPE; case SIGSEGV
        dokonaj zrzutu pamięci i zakończ proces;
    //koniec switch
  //koniec if (obsługa domyślna)
  wywołaj funkcję handle_signal;
```


do_signal()
wykonuj w pętli nieskończonej
pobierz kolejny sygnał
brak sygnału
Tak
KONIEC
Nie
proces jest śledzony oraz sygnał !=SIGKILL
Tak
zatrzymaj proces; powiadom ojca wysyłając SIGCHLD; wywołaj funkcje szeregującą; jeśli sygnał jest zablokowany to wyślij go do siebie;
Nie
sygnał ma być ignorowany
Tak
sygnał to SIGCH
Nie
Tak
usuń dzieci zombie
Nie
sygnał ma obsługę domyślną
Tak
bieżący proces to init
Tak
Nie
Nie
wywołaj funkcję handle_signal()
wywołaj obsługę domyślną

4.2 Funkcja `handle_signal`

`void handle_signal(unsigned long sig, struct k_sigaction *ka, siginfo_t *info, sigset_t *oldset, struct pt_regs *regs)`

1. włóż na stos : adres instukcji, która właśnie miała być wywołana, a następnie adres funkcji sys_sigreturn
2. ustaw wskaźnik instrukcji procesu na pierwszą instrukcję procedury obsługi - czyli najpierw zostanie wywołana procedura obsługi, a dopiero potem funkcja systemowa sys_sigreturn
3. if (flaga sa_flags = SA_ONESHOT) przywróć obsługę domyślną;
4. if (flaga sa_flags != SA_NODEFER) blokuj obsługiwany sygnał;

5 Najważniesze funkcje systemowe w skrócie

5.1 Funkcja `sys_sigprocmask`

`int sigprocmask(int how, old_sigset_t *set, old_sigset_t *oset)`
Zmiana obecnie zablokowanych sygnałów. Argument `how` może przyjmować następujące wartości:

- SIG_BLOCK - suma logiczna (binarna) z poprzednią maską
- SIG_UNBLOCK - różnica (binarna) z poprzednią maską
- SIG_SETMASK - przypisanie czyli zmiana maski na `set`

5.2 Funkcja `sys_sigpending`

`int sigpending(old_sigset_t *set)` pozwala sprawdzić, które z zablokowanych sygnałów przyszły (przecięcie pól `signal` i `blocked` z `task_struct`.

5.3 Funkcja `do_sigaction`

`int do_sigaction(int sig, const struct k_sigaction *act, struct k_sigaction *oact)`
Używana do zmiany sposobu obsługi sygnału.
Schemat działania:

- funkcja pobiera 3 parametry:
  - numer sygnału
  - opis nowej akcji (`struktura k_sigaction (= sigaction)`)
  - opis starej akcji (`struktura sigaction`)
- Jeśli jest to sygnał spoza zakresu (1.._NSIG) lub jest to próba zablokowania sygnałów SIGKILL lub SIGSTOP to zwróć -EINVAL
- Usuń z maski blokowanie SIGKILL lub SIGSTOP (patrz opis struktury sigaction)
- Jeśli sygnał ma obsługę domyślną i jest to sygnał SIGCONT lub SIGCHLD lub SIGWINCH lub jest to dowolny sygnał z obsługą SIG_IGN (ignoruj) to usuń go z tablicy sygnałów (ignoruj wstecz) przelicz od nowa sygnały, które mają przyszły (*ang. pending signals*)

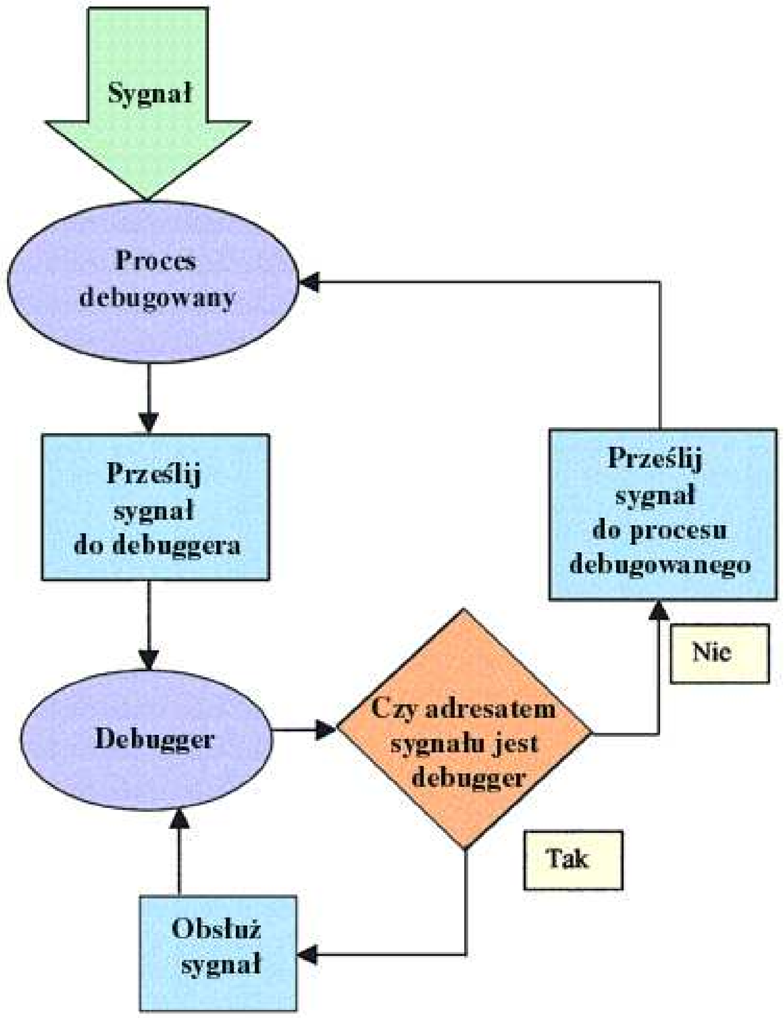
Definicja POSIX dla SIG_IGN:
Nadchodzący sygnał(`pending signal`), który jest ignorowany powinien być odrzucony (*ang. discarded, discard tzn. pozbyć się*)
I wyjątek:
Sygnał SIGCHLD, którego domyślną obsługą jest SIG_IGN i powoduje automatyczne zbieranie/zgarnianie (*ang. reaping*) dzieci pomimo, że POSIX mówi, że powinno być ignorowane.

5.4 Inne funkcje

`int recalc_sigpending(task_struct current)` - odświeża wartość pola `sigpending`
`int notify_parent(task_struct current, int przyczyna)` - poinformowanie rodzica o zmianie stanu dziecka zazwyczaj na ZOMBIE lub STOPPED

6 Jak to robi debugger

Jeśli program jest debugowany, to po otrzymaniu sygnału nie obsługuje go sam. Zamiast tego powiadamia rodzica (funkcją `notify_parent`), czyli debugger o tym, że otrzymał sygnał i zapisuje jego kod w `exit_code` w swoim `task_struct`. Wywołuje funkcje `schedule`, aby debugger mógł zdecydować czy nasz proces ma obsłużyć ten sygnał, czy nie.



7 Przykładowe zadania

1. Pokaż zachowanie procesu po otrzymaniu sygnału SIGUSR1 jeśli jest to sygnał:
   (a) blokowany
   (b) ignorowany
   (c) ma ustawioną obsługę domyślną
2. Pokaż zachowanie się procesu po otrzymaniu sygnału SIGCHLD, gdy:
   (a) jest obsługa domyślna
   (b) jest to sygnał ignorowany
3. Dlaczego obsługa sygnału jest realizowana w trybie użytkownika ?
4. Co się stanie, gdy proces otrzyma dwa razy w krótkich odstępach czasowych sygnał SIGQUIT, jeśli:
   (a) ten sygnał jest ignorowany i jest ustawiona flaga SA_RESTART
   (b) ten sygnał jest ignorowany i jest ustawiona flaga SA_NODEFER
   (c) ten sygnał jest ignorowany i jest ustawiona flaga SA_ONESHOT
   (d) ten sygnał jest blokowany i jest ustawiona flaga SA_ONESHOT

8 Odpowiedzi do zadań

1. (a) Proces odznaczy przyjście sygnału ustawiając odpowiedni bit w zmiennej `signal` w swoim `task_struct`. Poza tym zachowa się tak, jakby sygnał był ignorowany.
   (b) Proces zachowa się tak, jakby nie otrzymał sygnału, ale aby stwierdzić, że sygnał jest ignorowany *musi odzyskać procesor*.
   (c) Proces zakończy swoje działanie.
2. (a) Proces zacznie posprzątać dzieci zombie.
   (b) Domyślne zachowanie, to ignorowanie (patrz punkt wyżej).
3. Po pierwsze nie ma powodu, aby wykonywać to w trybie jądra. Po drugie istnałoby niebezpieczeństwo manipulacji danymi systemowymi w trakcie realizacji funkcji obsługi sygnału (np. zmiana właściciela własnego procesu na root-a -> bramka dla wirusów)
4. Tak jak przy ignorowaniu sygnału (flagi nie mają wpływu na zachowanie się obsługi sygnału jeśli sygnał jest ignorowany lub blokowany i wtedy gdy obsługa jest domyślna). Wartość tych flag jest sprawdzana dopiero w funkcji `handle_signal` czyli jest dokonywana w momencie "wykonywania"funkcji obsługi (nie tej domyślnej). "Wykonywania", bo wtedy jest tylko modyfikowany stos wywołań procesu, a nie funkcja jest realizowana.

9 Bibliografia

9.1 Struktury

/include/asm-i386/signal.h - `sigaction`
/include/linux/sched.h - `task_struct`

9.2 Funkcje

/kernel/signal.c - `kill_pg_info, kill_sl_info, kill_proc_info, kill_something_info, notify_parent, do_sigaction, sys_sigprocmask, send_sig_info, force_sig_info, sys_sigpending`
/arch/i386/kernel/signal.c - `do_signal, handle_signal`
/include/linux/sched.h - `recalc_sigpending`