Systemy operacyjne

Laboratorium 4

Mateusz Małek
31 marca 2017

Laboratorium 4

Sygnały

Sygnały

- Rodzaj przerwania programowego generowanego przez system operacyjny lub użytkownika (ręcznie z terminala lub programowo z innego procesu)
- Asynchroniczny sposób komunikacji
  - Proces wykonuje się, “przychodzi” sygnał, proces wstrzymuje wykonywanie swoich normalnych zadań i przełącza się do procedury obsługi sygnału
- Sygnał może nieść ze sobą jakąś dodatkową informację (liczba typu int)
- Nadejście sygnału może przerwać blokujące wywołanie systemowe (nie zawsze natychmiast - “uninterruptible sleep”); w zależności od parametrów sposobu obsługi sygnału (np. flaga SA_RESTART w sigaction) i rodzaju syscalla, po zakończeniu obsługi sygnału wywołanie systemowe jest automatycznie ponawiane lub jest zwracany błąd, a errno zostaje ustawione na EINTR (“Interrupted system call”)

Sygnały

- Można obsłużyć na kilka sposobów (ustawianych dla poszczególnych sygnałów otrzymywanych w ramach danego procesu)
  - Domyślny (zwykle zakończenie procesu, czasami połączone z core dumpem)
  - Zignorowanie (sygnał w żaden sposób nie wpływa na proces)
  - Uruchomienie w procesie określonej przez siebie funkcji obsługi sygnału (handler)
  - *Zamaskowanie* (opóźnienie domyślnego sposobu obsługi lub wykonania handlera do momentu aż sygnał zostanie przez nas odmaskowany)
  - Jedynym możliwym sposobem obsługi sygnałów SIGKILL i SIGSTOP, jest ich domyślna, systemowa procedura obsługi (nie możemy ich przechwycić w programie)
- Procesy potomne dziedziczą po rodzicu sposób obsługi sygnałów
- Wywołanie execve (w jakikolwiek jawny/niejawny sposób) resetuje wszystkie niestandardowe handlery na domyślny sposób obsługi sygnału

Sygnały

- “Zwykłe” (mają swoje numery i skojarzone z nimi umowne nazwy)

  1 - SIGHUP, 2 - SIGINT, 3 - SIGQUIT, 6 - SIGABRT, **9 - SIGKILL**, 11 - SIGSEGV, 15 - SIGTERM, 18 - SIGCONT, **19 - SIGSTOP**, …
  10 - SIGUSR1, 12 - SIGUSR2 - jedyne dwa które explicite są wskazane jako “do wykorzystania przez użytkownika”
  - Jeśli ich źródłem był system operacyjny, to miały dokładnie określony powód
    - I jest nim jakiś problem - programowy (np. odwołanie się do NULL pointera - SIGSEGV, błędna instrukcja - SIGILL) lub sprzętowy (typowo SIGBUS)
  - Jeśli sygnał był zamaskowany, a do momentu jego odblokowania proces otrzymał go wielokrotnie - zostanie dostarczony tylko raz
  - Jeśli zamaskowanych było wiele sygnałów, a do momentu ich odblokowania otrzymano różne - nie ma gwarancji w jakiej kolejności zostaną dostarczone
- Czasu rzeczywistego (numery od SIGRTMIN do SIGRTMAX)

  pierwszy - SIGRTMIN, drugi - SIGRTMIN + 1, …, ostatni - SIGRTMAX
  - Jeśli są w danym momencie zamaskowane, to są kolejkowane (do pewnej granicy…)
  - Po odblokowaniu dostarczane są według rosnącego numeru, a dalej - w kolejności wysłania
    - *Pamiętaj że sygnał może przenosić ze sobą dodatkową informację (liczba typu int)*

Wysyłanie sygnałów z terminala

- Polecenie kill
  - kill -l (l jak Leokadia) - wyświetla listę dostępnych, znanych sygnałów
  - kill -s SYGNAŁ PID, kill -SYGNAŁ PID - wysyła podany sygnał do procesu PID
    - SYGNAŁ można podać jako liczbą, nazwę lub nazwę z SIG na początku (SIGnazwa)
    - Wszystkie poniższe wywołania skutkują wysłaniem sygnału SIGKILL (nr 9) do procesu o numerze PID 1234:
      kill -s 9 1234, kill -9 1234, kill -s KILL 1234, kill -s KILL 1234, kill -s SIGKILL 1234, kill -SIGKILL 1234
- Kombinacje klawiszy
  - Ctrl+C - wysyła sygnał SIGINT, który standardowo przerywa proces
  - Ctrl+Z - wysyła sygnał SIGTSTP, który nieobsłużony przekształca się w SIGSTOP i wstrzymuje działanie programu
    - Wznowienia programu (SIGCONT) można dokonać funkcjami powłoki (np. fg lub bg)
  - Ctrl+\ - wysyła sygnał SIGQUIT, który standardowo przerywa proces z jednoczesnym wygenerowaniem jego core dumpa

Funkcje do wysyłania sygnałów

- int kill(pid_t pid, int sig) - man 2 kill
  - Wysyła sygnał o numerze sig do procesu o PID równym pid (pid > 0), grupy procesów o numerze -pid (pid < 0), bieżącej grupy procesów (pid == 0) lub do wszystkich możliwych procesów za wyjątkiem init (pid == -1)
  - Jeśli jako numer sygnału podamy 0, sygnał nie zostanie wysłany, ale zostanie sprawdzone istnienie procesu o podanym numerze
- int raise(int sig) - man 3 raise
  - Równoważne kill(getpid(), sig) (chyba że program jest wielowątkowy)
- int sigqueue(pid_t pid, int sig, const union sigval value) - man 2 sigqueue
  - Działanie analogiczne do kill, ale możemy dodatkowo przekazać z sygnałem pewną wartość (za pośrednictwem unii value)

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

Funkcje do ustawiania obsługi sygnałów

- sighandler_t signal(int signum, sighandler_t handler) - man 2 signal
  - Ustawia sposób obsługi sygnału signum - zignorowanie (SIG_IGN), domyślny (SIG_DFL) lub wywołanie handlera (wskaźnik do funkcji)
  - Jeśli ustawiono handler, to przy jego najbliższym wywołaniu dla danego sygnału zostaje przywrócony domyślny sposób obsługi (zwykle zakończenie procesu); trzeba go ustawić ponownie (np. na końcu handlera), jeśli chcemy użyć go przy kolejnym nadejściu sygnału
- typedef void (*sighandler_t)(int)
  - Sygnatura najprostszego handlera do obsługi sygnałów - przyjmuje jako argument numer dostarczonego sygnału (jeden handler może obsługiwać kilka sygnałów na raz)
  - W handlerach sygnałów powinno się unikać korzystania ze złożonych funkcji bibliotecznych i wykonywać jedynie proste operacje - np. zmianę globalnych flag lub wypisywanie komunikatów funkcjami systemowymi (w man 7 signal kompletna lista “bezpiecznych” funkcji)
    - Np. write(STDERR_FILENO, text, strlen(text)), gdzie text to char *

Funkcje do ustawiania obsługi sygnałów

- int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact) - man 2 sigaction
  - Zmienia procedurę obsługi sygnału signum na opisaną w strukturze wskazanej przez act
    - Jeśli act jest NULLem - nie zmienia procedury
  - Zapisuje informacje o dotychczasowej procedurze obsługi sygnału do struktury wskazywanej przez oldact
    - Jeśli oldact jest NULLem - nie zapisuje informacji
  - Jeśli act i oldact są NULLem - to nie ma sensu ;)
  - Struktura sigaction pozwala dokładnie skonfigurować sposób obsługi sygnałów
    - Można w polu sa_handler podać wskaźnik do funkcji, która jako swój argument przyjmie numer odebranego sygnału (jak w signal)
    - Można ustawić flagę SA_SIGACTION i **zamiast** sa_handler ustawić w polu sa_sigaction wskaźnik do funkcji która przyjmie numer sygnału, wskaźnik do siginfo_t ze szczegółami na temat sygnału oraz wskaźnik do void z kontekstem (typowo handlery nie wykorzystują informacji o kontekście do czegokolwiek)

Struktura sigaction

```
struct sigaction {
    void     (*sa_handler)(int);
    void     (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t   sa_mask;
    int        sa_flags;
    void     (*sa_restorer)(void);
};
```

- W zadaniu przyda się szczególnie wspomniana flaga SA_SIGINFO
- Funkcja signal może być zrealizowana jako wywołanie sigaction z flagą SA_RESETHAND (która powoduje przywrócenie domyślnego handlera po obsłużeniu sygnału)
- sa_mask pozwala uchronić się przed dostarczeniem nowego sygnału zanim skończymy obsługiwać poprzedni

Struktura siginfo_t

```
siginfo_t {
  int      si_signo;     /* Signal number */
  int      si_errno;     /* An errno value */
  int      si_code;      /* Signal code */
  int      si_trapno;    /* Trap number that caused
                            hardware-generated signal
                            (unused on most architectures) */
  pid_t    si_pid;       /* Sending process ID */
  uid_t    si_uid;       /* Real user ID of sending process */
  int      si_status;    /* Exit value or signal */
  clock_t  si_utime;     /* User time consumed */
  clock_t  si_stime;     /* System time consumed */
  sigval_t si_value;     /* Signal value */
  int      si_int;       /* POSIX.1b signal */
  void    *si_ptr;       /* POSIX.1b signal */
  int      si_overrun;   /* Timer overrun count;
                            POSIX.1b timers */
  int      si_timerid;   /* Timer ID; POSIX.1b timers */
  oid     *si_addr;      /* Memory location which caused fault */
  long     si_band;      /* Band event (was int in
                            glibc 2.3.2 and earlier) */
  int      si_fd;        /* File descriptor */
  short    si_addr_lsb;  /* Least significant bit of address
                            (since Linux 2.6.32) */
  void    *si_lower;     /* Lower bound when address violation
                            occurred (since Linux 3.19) */
  void    *si_upper;     /* Upper bound when address violation
                            occurred (since Linux 3.19) */
  int      si_pkey;      /* Protection key on PTE that caused
                            fault (since Linux 4.6) */
  void    *si_call_addr; /* Address of system call instruction
                            (since Linux 3.5) */
  int      si_syscall;   /* Number of attempted system call
                            (since Linux 3.5) */
  unsigned int si_arch;  /* Architecture of attempted system call
                            (since Linux 3.5) */
}
```

- si_pid przyda się w zadaniu do odczytania od którego procesu otrzymano dany sygnał
- si_value to wartość przekazana przy użyciu trzeciego argumentu do siqueue

Funkcje związane z oczekiwaniem...

- unsigned int alarm(unsigned int seconds) - int 2 alarm
  - Żąda dostarczenia SIGALRM do procesu po podanej ilości sekund, kasując przy tym wcześniej ustawioną dyspozycję (dla seconds = 0 usuwa ustawiony alarm)
  - Zwraca ilość sekund które pozostały do dostarczenia poprzednio ustawionego alarmu
- unsigned int sleep(unsigned int seconds) - int 3 sleep
  - Usypia proces na podaną ilość sekund **lub do momentu otrzymania sygnału**
  - Może być zaimplementowane w oparciu o SIGALRM, dlatego nie należy w kodzie programu mieszać sleep z alarm (zgodnie z dokumentacją: “bad idea”)
- int pause(void) - int 2 pause
  - Wstrzymuje działanie procesu do momentu dostarczenia sygnału który nie jest ignorowany (ma ustawiony handler lub akcję domyślną powodującą zakończenie procesu)

Obsługa zbiorów sygnałów (sigset_t)

- int sigemptyset(sigset_t *set)
  - Inicjalizuje strukturę pustym zestawem sygnałów (tak żeby w np. dynamicznie zaalokowanej nie znajdowały się śmieci)
- int sigfillset(sigset_t *set)
  - Inicjalizuje strukturę pełnym (wszystkimi) zestawem sygnałów
- int sigaddset(sigset_t *set, int signum)
  - Dodaje podany sygnał do zestawu sygnałów
- int sigdelset(sigset_t *set, int signum)
  - Usuwa podany sygnał z zestawu sygnałów
- int sigismember(const sigset_t *set, int signum)
  - Sprawdza czy podany sygnał znajduje się w zestawie sygnałów
- Wszystkie powyższe opisane w man 3 sigismember

Maskowanie sygnałów (wstrzymywanie dostarczenia)

- int sigprocmask(int how, const sigset_t *set, sigset_t *oldset) - man 2 sigprocmask
  - Zmienia maskę sygnałów przez zablokowanie (SIG_BLOCK)/odblokowanie (SIG_UNBLOCK) lub przez ustawienie maski na dokładnie taką jak w set (SIG_SETMASK)
  - Zwraca przez oldset maskę sygnałów która była ustawiona do tej pory (o ile oldset != NULL)
  - Sygnał zamaskowany oczekuje na odblokowanie (chyba że jest i tak ignorowany) i dopiero wtedy uruchamiane są jego handlery
- int sigsuspend(const sigset_t *mask) - man 2 sigsuspend
  - Tymczasowo zmienia maskę sygnałów na mask i usypia proces do momentu otrzymania niezamaskowanego, nieignorowanego sygnału
- Typowo używane wspólnie ze sobą
  - Przed wejściem do “sekcji krytycznej” maskujemy sygnały przez sigprocmask
  - Po zakończeniu wykonywania kodu z “sekcji krytycznej” wywołujemy sigsuspend(oldset), gdzie oldset zostało ustawione w wyniku wywołania sigprocmask
  - W zadaniu będą używane kiedy będziemy oczekiwali na sygnały od rodzica/potomków

Maskowanie sygnałów (wstrzymywanie dostarczenia)

- int sigpending(sigset_t *set) - man 2 sigpending
  - W podanym zbiorze sygnałów umieszcza te, które nie zostały jeszcze dostarczone do obsłużenia przez proces z powodu ustawionej w danym momencie maski sygnałów

Errata do materiałów pomocniczych z UPeL

- init (PID 1) **odbiera** sygnały - ale tylko te, dla których ma ustawione handlery
  - *“The only signals that can be sent to process ID 1, the init process, are those for which init has explicitly installed signal handlers”*
- Nie ma sygnału **SIGALARM**, jest **SIGALRM**
- Nie ma sygnału **SIGABORT**, jest **SIGABRT**
- Sygnatura funkcji-handlera dla sygnału ma postać **void (*func)(int)** (przyjmuje jako argument numer sygnału)

Dziękuję za uwagę