Systemy operacyjne czasu rzeczywistego

Sygnały. Alarmy. Timery.

Paweł Strączyński
pstraczynski@tu.kielce.pl
Katedra Urządzeń Elektrycznych i Automatyki

Sygnały

Procesy w Linux'ie komunikują się pomiędzy sobą między innymi za pomocą sygnałów tzw. przerwań programowych które informują o tym że zaszło jakieś zdarzenie.

Sygnały mogą być generowane przez użytkownika, jądro bądź procesy które mogą się komunikować między sobą za pomocą sygnałów.

Sygnały mogą być generowane bezpośrednio przez użytkownika (np. funkcja `kill()`), może wysyłać je jądro lub wzajemnie procesy miedzy sobą.

Sygnały są mechanizmem asynchronicznym.

Sygnały

Pewne znaki z terminala również powodują wygenerowanie sygnałów.

Znak przerwania (Ctrl-C lub Delete) służy do zakończenia bieżącego procesu poprzez wygenerowanie sygnału SIGINT.

Znak zakończenia (zazwyczaj Ctrl-\) powoduje wysłanie sygnału SIGQUIT który to spowoduje zakończenie wykonywania bieżącego procesu z zapisaniem obrazu pamięci.

Sygnały

SIGTERM- Sygnał programowe zakończenia procesu
Sygnał ten jest standardowym sygnałem zakończenia procesu. Jest on generowany po wysłaniu polecenbia. Sygnału tego jest także używany podczas wyłączania systemu w celu zakończenia wszystkich aktywnych procesów.
Programy powinny przyjmować domyślną obsługę tego sygnału, lub jak najszybciej zrobić po sobie porządek i wywoływać funkcję exit().
Domyślnie sygnał ten powoduje zakończenie procesu.

SIGKILL- Sygnał zakończenie procesu
Jest jedynym całkowicie pewny sposób zakończenia wykonywania procesu. Żaden proces który odbiera ten sygnał nie może go zignorować, ani przechwycić poprzez dostarczenie własnej funkcji do obsługi tego sygnału. Sygnału tego powinno się stosowany tylko w przypadkach wyjątkowych. W zwykłaych przypadkach zalecany jest sygnał SIGTERM

Sygnały

SIGINT - Znak przerwania (*ang. interrupt*)
Sygnał ten jest zwykle wysyłany do każdego procesu powiązanego z danym terminalem po wciśnięci przycisku przerwania. Można deaktywować działanie klawisza przerwania oraz zmienić sam klawisz, wywołując funkcję systemową ioctl().
Domyślnie powoduje zakończenie procesu.

SIGSEGV - Naruszenie segmentacji
Sygnał ten generowany jest najczęściej po odwołaniu się przez proces do takiego adresu w pamięci, do którego nie ma dostępu.
Domyślnie powoduje zakończenie procesu z zapisem obrazu pamięci.

Sygnały

```c
void int_handler(int signum)
{
        fprintf(stderr, "-- Sygnal %i przechwycony\n", signum);
        sleep(1);
}
…
int i;
struct sigaction sa;
sa.sa_handler = int_handler;
sigfillset(&(sa.sa_mask)); //inicjalizacja i ustawienie sygnału
sa.sa_flags = 0;
sigaction(SIGINT, &sa, 0); //zmiana akcji dla sygnału

for(i = 0; i < 20; ++i) {
        printf("Praca procesu (%i)\n", i);
        sleep(1);
}
…
```

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

Sygnały

Po wciśnięci kombinacji klawiszy CTRL+C sygnał zostaje przechwycony oraz zostaje wykonana funkcja jego obsługi.

```
Praca procesu (0)
Praca procesu (1)
Praca procesu (2)
^C-- Sygnal 2 przechwycony
Praca procesu (3)
^C-- Sygnal 2 przechwycony
Praca procesu (4)
^C-- Sygnal 2 przechwycony
Praca procesu (5)
```

Sygnały

int	**sigemptyset** (sigset_t *set) Initialize and empty a signal set.
int	**sigfillset** (sigset_t *set) Initialize and fill a signal set.
int	**sigaddset** (sigset_t *set, int sig) Add a signal to a signal set.
int	**sigdelset** (sigset_t *set, int sig) Delete a signal from a signal set.
int	**sigismember** (const sigset_t *set, int sig) Test for a signal in a signal set.
int	**sigaction** (int sig, const struct sigaction *act, struct sigaction *oact) Examine and change a signal action.
int	**pthread_kill** (pthread_t thread, int sig) Send a signal to a thread.
int	**pthread_sigqueue_np** (pthread_t thread, int sig, union sigval value) Queue a signal to a thread.
int	**sigpending** (sigset_t *set) Examine pending signals.
int	**pthread_sigmask** (int how, const sigset_t *set, sigset_t *oset) Examine and change the set of signals blocked by a thread.
int	**sigwait** (const sigset_t *set, int *sig) Wait for signals.
int	**sigwaitinfo** (const sigset_t *__restrict__ set, siginfo_t *__restrict__ info) Wait for signals.
int	**sigtimedwait** (const sigset_t *__restrict__ set, siginfo_t *__restrict__ info, const struct timespec *__restrict__ timeout) Wait during a bounded time for signals.

Alarmy

Alarmy są to układy czasowe które pozwalają na generowanie cyklicznych zdarzeń z zadanym interwałem czasowym.

int	**rt_alarm_create** (RT_ALARM *alarm, const char *name, rt_alarm_t handler, void *cookie) Create an alarm object from kernel space. More...
int	**rt_alarm_delete** (RT_ALARM *alarm) Delete an alarm. More...
int	**rt_alarm_start** (RT_ALARM *alarm, RTIME value, RTIME interval) Start an alarm. More...
int	**rt_alarm_stop** (RT_ALARM *alarm) Stop an alarm. More...
int	**rt_alarm_inquire** (RT_ALARM *alarm, RT_ALARM_INFO *info) Inquire about an alarm. More...

Alarmy

Jeżeli funkcja `rt_alarm_wait()` zwróci 0, wówczas wykonywany jest kod obsługi alarmu, w tym przypadku drukowanie na standardowym wyjściu.

```
#include <native/alarm.h>
...
RT_ALARM al;
...
rt_alarm_create(&al,"AL");
rt_alarm_start(&al, alarm_v, period);
...
void alarm_service(void *arg){
     for(;;){
        ret=rt_alarm_wait(&al);
        if(!ret){
           rt_printf("ALARM\n");
        }
     }
}
```

Alarmy

Jeżeli funkcja `rt_alarm_wait()` zwróci 0, wówczas wykonywany jest kod obsługi alarmu, w tym przypadku drukowanie na standardowym wyjściu.

```
#include <native/alarm.h>
...
RT_ALARM al;
...
rt_alarm_create(&al,"AL");
rt_alarm_start(&al, alarm_v, period);
...
void alarm_service(void *arg){
     for(;;){
        ret=rt_alarm_wait(&al);
        if(!ret){
           rt_printf("ALARM\n");
        }
     }
}
```

Alarmy

Jeżeli funkcja `rt_alarm_wait()` zwróci 0, wówczas wykonywany jest kod obsługi alarmu, w tym przypadku drukowanie na standardowym wyjściu.

```
ALARM
ALARM
ALARM
ALARM
ALARM
ALARM
```

Alarmy

Jeżeli funkcja `rt_alarm_wait()` zwróci 0, wówczas wykonywany jest kod obsługi alarmu, w tym przypadku drukowanie na standardowym wyjściu.

```
ALARM
ALARM
ALARM
ALARM
ALARM
ALARM
```

Timery

Przykład pobierania czasu w standardzie POSIX. Czas systemu zwracany jest w sekundach od 1 stycznia 1970r 00:00:00 GTM

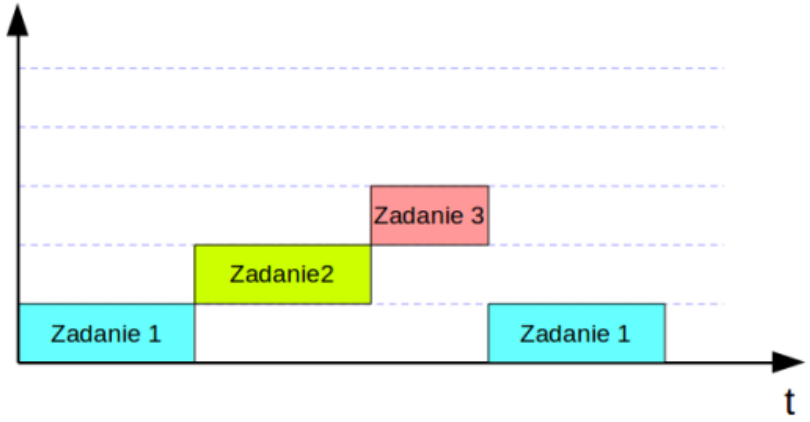
```
struct timespec ts;
// wywołanie time()
printf("Zwrocona wartosc wywolania %d sekund\n", time(NULL));
// pobranie czasu
clock_gettime(CLOCK_REALTIME, &ts);
printf("clock_gettime zwrocilo:\n");
printf("%d sekund i %ld nanosekund\n", ts.tv_sec, ts.tv_nsec);
```

Funkcja `time()` zwraca czas w sekundach, natomiast `clock_gettime()` zwraca czas w sekundach oraz nanosekundach.

Timery

SRTIME	**rt_timer_ns2tsc** (SRTIME ns) Convert nanoseconds to local CPU clock ticks. More...
SRTIME	**rt_timer_tsc2ns** (SRTIME ticks) Convert local CPU clock ticks to nanoseconds. More...
RTIME	**rt_timer_tsc** (void) Return the current TSC value. More...
RTIME	**rt_timer_read** (void) Return the current system time. More...
SRTIME	**rt_timer_ns2ticks** (SRTIME ns) Convert nanoseconds to internal clock ticks. More...
SRTIME	**rt_timer_ticks2ns** (SRTIME ticks) Convert internal clock ticks to nanoseconds. More...
int	**rt_timer_inquire** (**RT_TIMER_INFO** *info) Inquire about the timer. More...
void	**rt_timer_spin** (RTIME ns) Busy wait burning CPU cycles. More...
int	**rt_timer_set_mode** (RTIME nstick) Set the system clock rate. More...

freeRTOS



Algorytm szeregujący z wywłaszczeniami

xTaskCreate() - funkcja służąca do utworzenia nowego zadania. Jako argumenty pobiera m.in. nazwę funkcji obsługującej zadanie, nazwę pod jaką zadanie jest identyfikowane w systemie, rozmiar stosu, priorytet zadania czy uchwyt pozwalający na dokonywanie dalszych operacji na konkretnym zadaniu.

vTaskDelete() - funkcja dokonująca usunięcia zadania. Jako argument przyjmuje uchwyt do zadania przypisany do niego w trakcie utworzenia.

vTaskStartScheduler() - funkcja dokonuje uruchomienia algorytmu szeregowania zadań.

vTaskDelay() - funkcja blokująca zadanie na czas określony liczbą taktów zegara.

vTaskDelayUntil() - funkcja blokująca na ściśle określony czasu. Poza liczbą taktów pobiera jako argument liczbę taktów jaka upłynęła od czasu uruchomienia algorytmu szeregującego.

vTaskPrioritySet() - funkcja służąca do ustawienia priorytetu dla zadania. Jako argumenty przujmuje: uchwyt do zadania oraz priorytet do ustawienia.

vTaskPriorityGet() - funkcja zwraca priorytet zadania do którego uchwyt podano jako argument funkcji.

vTaskSuspend() - funkcja służąca do wstrzymania zadania (tryb Suspended). Jako argument funkcja przyjmuje uchwyt do wstrzymanego zadania. Jeżeli wstrzymane ma zostać zadanie wewnątrz którego funkcja jest wywołana argumentem funkcji może być wartość NULL.

vTaskResume() - funkcja służąca do wznowienia działania zawieszonego działania. Argumentem funkcji jest uchwyt do zadania.