Zwielokrotnianie wejścia–wyjścia

Bartłomiej Świercz

Katedra Mikroelektroniki i Technik Informatycznych

Łódź, 21 marca 2006

Wstęp

Zwielokrotnianie wejścia–wyjścia — informowanie przez jądro procesu użytkownika o spełnieniu jednego lub więcej warunku wejścia–wyjścia dotyczącego np. gotowości deskryptora do odczytu danych z bufora systemu.

Ze zwielokrotnienia wejścia–wyjścia korzysta się najczęściej w przypadku:

- gdy program obsługuje więcej niż jeden deskryptor lub większą liczbę gniazd,
- w sytuacji gdy program stosuje jednocześnie gniazdo nasłuchowe i połączeniowe,
- kiedy serwer udostępnia naraz wiele usług.

Modele wejścia–wyjścia

- Model wejścia–wejścia blokującego.
- Model wejścia–wejścia nieblokującego.
- Model wejścia–wejścia zwielokrotnionego.
- Model wejścia–wejścia sterowanego sygnałami.
- Model wejścia–wejścia asynchronicznego.

Model wejścia–wyjścia blokującego

Jest to najbardziej rozpowszechniony model. Domyślnie każdy deskryptor i gniazdo jest blokujące. Oznacza to że podczas wywołania funkcji `read` proces zostaje zablokowany przez jądro systemu na czas odczytania bufora lub przekazania informacji o błędzie.

Uwaga: Należy pamiętać że w modelu wejścia–wyjścia blokującego funkcje służące do operacji na deskryptorach mogą być przerwane przez sygnał!

Model wejścia–wyjścia nieblokującego

Model nieblokujący wymaga utworzenia deskryptora w trybie nieblokującym (flaga `O_NONBLOCK`). Wywołując funkcję systemową na takim deskryptorze (np. `read`) jądro napotykając sytuację, w której musi uśpić proces (np. nie ma danych w buforze) nie usypia go lecz zwraca informację o błędzie (`EAGAIN`).

Model wejścia–wyjścia zwielokrotnionego

W modelu wejścia–wyjścia zwielokrotnionego korzystamy z wywołania funkcji systemowych `select` lub `poll`. Proces nie blokuje się na funkcjach dostępu do bufora (`read` lub `write`), lecz na funkcjach `select` i `poll`.

Funkcje służące do zwielokrotniania wejścia–wyjścia zostaną omówione w dalszej części prezentacji.

Model wejścia–wyjścia sterowanego sygnałami

W modelu tym jądro generuje sygnał `SIGIO` za każdym razem kiedy deskryptor gotowy jest do czytania.

Model wejścia–wyjścia asynchronicznego

Jest to nowy model wprowadzony przez standard POSIX.1g jako rozszerzenie dla systemów czasu rzeczywistego. Do obsługi wejścia–wyjścia asynchronicznego służą funkcje `aio_read` i `aio_write`.

Główna różnica pomiędzy modelem sterowanym sygnałami, a modelem asynchronicznym wejścia–wyjścia polega na tym, że w modelu asynchronicznym dostajemy informację o zakończeniu operacji, a w modelu sterowanym sygnałami informację o możliwości rozpoczęcia operacji.

Funkcja select

Funkcja `select` nakazuje jądru informowanie procesu użytkownika o zmianach stanu podanych jako argumenty deskryptorów.

```
#include <sys/select.h>

int select(int n, fd_set *readfds, fd_set *writefds, fd_set
    *exceptfds, struct timeval *timeout);

FD_CLR(int fd, fd_set *set);
FD_ISSET(int fd, fd_set *set);
FD_SET(int fd, fd_set *set);
FD_ZERO(fd_set *set);
```

Określenie czasu

```
#include <sys/time.h>

struct timeval
{
    long    tv_sec;         /* seconds */
    long    tv_usec;        /* microseconds */
};
```

Możemy przekazać argument dotyczący czasu na trzy rożne sposoby:

- Przekazać wartość `NULL` co oznacza, że funkcja `select` ma czekać w nieskończoność.
- Możemy określić szczegółowy czas i polecić, aby funkcja `select` nie czekała dłużej niż czas podany jako argument.
- Możemy ustawić czas na zero, co spowoduje, że funkcja `select` będzie odpytywała system o stan deskryptorów (tryb polling).

Funkcja poll

Funkcja `poll` ma podobne zastosowanie co funkcja `select` lecz w porównaniu do funkcji `select` dostarcza więcej informacji dotyczących urządzeń strumieniowych.

```
#include <sys/poll.h>

int poll(struct pollfd *ufds, nfds_t nfds, int timeout);

struct pollfd
{
    int fd;             /* file descriptor */
    short events;       /* requested events */
    short revents;      /* returned events */
};
```

Określenie czasu dla funkcji poll

Funkcja `poll` ze względu na parametr `timeout`:

- `INFTIM` — oczekuje aż do skutku (wystąpienia zdarzenia na deskryptorze).
- 0 — powraca natychmiast (polling).
- > 0 — czeka na zdażenie nie dłużej niż czas podany w milisekundach.

Porównanie funkcji poll i select

- Funkcja `poll` jest bardziej skalowalna.
- Funkcja `poll` przekazuje dodatkowe informacje o połączeniach strumieniowych.
- W jądrze Linux 2.6 funkcja `select` jest zaimplementowana za pomocą funkcji `poll`.
- Funkcja `select` jest znacznie częściej spotykana.