Linux — Podsystem Wejścia–Wyjścia

Piotr Sawicki

4 stycznia 1999

1 Urządzenia ogólnie — urządzenia znakowe

Autor: Tien Nguyen Hoang

1.1 WSTĘP

System operacyjny nie posiadający systemu wejścia-wyjścia pozostaje czarną skrzynką bez jakiegokolwiek dostępu z zewnątrz. Dzięki systemowi wejścia-wyjścia możliwa jest obsługa urządzeń zewnętrznych— umożliwiających różnorodną komunikację użytkownika z systemem. W dzisiejszych systemach niezbędnymi urządzeniami są szeroko pojęte terminale (np. X-TERMy— klawiatury i monitory podpięte do PC-ta lub też wirtualne terminale obsługiwane przez sieć)— pamięci masowe (dyski— streamery— CD-ROMy— etc.)— drukarki— modemy— karty sieciowe— karty graficzne i dzwiękowe— etc. Oprócz fizycznych urządzeń niektóre wirtualne mechanizmy systemu operacyjnego mogą być obsługiwane przez system jako urządzenia wejścia- wyjścia(np. sterownik pamięci czy też mechanizm komunikacji korzystający z wirtualnych terminali "pty").

Wbrew tytułowi pracy głównym tematem będzie podprogram obsługi urządzeń— ponieważ jest to część systemu operacyjnego służąca do sterowania urządzeń.

Normalnie Linux udostępnia użytkownikom swoje usługi poprzez mechanizm funkcji systemowych— ale dostęp do podprogramów obsługi urządzeń realizowany jest przez system plików

1.2 PLIKI SPECJALNE

Każde urządzenie w Unixie jest zainstalowane w systemie jako osobny plik. Jest to tzw. plik specjalny. Taki sposób dostępu do urządzeń zapewnia między innymi:

- możliwość rozmieszczania 'urządzeń' w różnych miejscach systemu plików
- identyczny sposób dostępu do urządzeń co do plików (open— close— read— write ...)

- łatwe zarządzanie dostępem (identyczne możliwości dotyczące praw dostępu jak dla zwykłych plików)

Przy operacjach na zwykłych plikach po wywołaniu określonej funkcji na zadanym pliku system operacyjny szuka właściwego odpowiednika dla danego systemu plików (np. open dla ext2 o ile dany system plików jest typu ext2). Dla pliku specjalnego odpowiadającego pewnemu urządzeniu system szuka funkcji właściwych dla danego urządzenia. Dostarczaniem takich funkcji zajmują się sterowniki urządzeń.

Żeby mieć dostęp do urządzenia trzeba otwierać taki plik im odpowiadający w celu odnalezienia podprogram obsługi. Jeśli nie ma podprogramu to zwracany jest błąd— w przeciwnym przypadku wywołana jest funkcja open tego podprogramu. Jeśli wykonanie funkcji open kończy się sukcesem to dalsze operacje będą wykonane przez ten podprogram.

Podprogramy obsługi zawierają funkcje pozwalające wykonać operacje wejścia-wyjścia na urządzeniach— które nie muszą być urządzeniami fizycznymi.

Jedno urządzenie może mieć dwa pliki - jeden dla dostępu blokowego— a drugi dla dostępu znakowego. Otrzymujemy wtedy dwa niezależne dostępy do jednego urządzenia. Wada takiego rozwiazania jest to— że nie jesteśmy w stanie uzyskać jednoznacznej kontroli dostępu do danego urządzenia. W szczególności trudno jest uzyskać wyłączny dostęp do urządzenia (tzn. żeby dane urządzenie bylo naraz otwarte tylko przez jeden proces).

1.3 DOSTĘP DO URZĄDZEŃ

Żeby mieć dostęp do urządzeń trzeba stworzyć plik specjalny zawierający podprogram obsługi urządzenia. Plik specjalny tworzy się przez polecenie mknod a nie poleceniem open z flagą O_CREAT. Tworzenie plików specjalnych nie wymaga istnienia odpowiadających im podprogramów obsługi urządzeń.

1. Tworzenie urządzenia — mknod urządzenie c 100 101

   Polecenie to tworzy plik specjalny o nazwie urządzenie i zapisuje— że jego podprogram obsługi urządzenia jest identyfikowany przez parę ('c'— 100)— a liczba 101 będzie przekazywana podprogramowi obsługi przy wszystkich operacjach na tym pliku.

   Aby utworzyć plik specjalny trzeba mieć uprawnień administratora— ponieważ Linux nie sprawdza uprawnienia do tworzenia urządzenia— a kontrola uprawnień odbywa sie wyłącznie na poziomie sytemu plików. Więc ktoś może stworzyć plik specjalny odpowiadający twardemu dyskowi i odczytać z niego dowolne informacje.

2. Czytanie z urządzenia cat urządzenie Polecenie to kończy się błędem(w momencie wywołania funkcji systemowej open) gdy nie istnieje podprogram obsługi tego urządzenia. Jeśli podprogram obsługi istnieje to on decyduje czy pozwolić na otwarcie urządzenia oraz które operacje uznać za poprawne. Czasami operacje na urządzeniach fizycznie nie podłączone wcale nie zwracają błędu.

3. Kasowanie urządzenia — rm urządzenie

   Usuwanie pliku specjalnego nie ma nic związanego z podprogramem obsługi— w szczególności ten podprogram nie musi istnieć.

1.4 URZĄDZENIA ZNAKOWE I BLOKOWE

Komunikacja z urządzeniami sprowadza sie do przesyłania znaków. Dostęp do urządzeń zewnętrznych możemy podzielić na dostęp blokowy i znakowy. Dostęp znakowy jest to rodzaj dostępu w którym przy odczycie zadania możemy podzielić na poszczególne znaki. W dostępie blokowym najmniejszą jednostką na jaką możemy podzielić zadanie jest blok - pewien pakiet znaków o określonej długości. Teoretycznie wszystkie urządzenia powinny udostępniać dostęp znakowy— a tylko niektóre dostęp blokowy. W praktyce niektóre sterowniki urządzeń dają nam jedynie dostęp blokowy— uznając dostęp znakowy za zbędny. Przykładem urządzenia blokowego może być dysk twardy— stacje dyskietek— ZIP drive-y— etc. Urządzenia znakowe to np. terminale i myszy. Pewne fizyczne parametry urządzeń wymuszają użycie danego trybu dostępu (jako jedynego logicznego sposobu dostępu). Trudno sobie wyobrazić dostęp do myszy w trybie blokowym - musielibyśmy wtedy czekać na reakcje komputera dopiero po wykonaniu odpowiedniej liczby ruchów... (mysz przesyłałaby dopiero wtedy dane— gdy dopełnilibyśmy bufor myszy do pełnego bloku). Dostęp do dysku w trybie znakowym powodowałby znaczne spowolnienie komunikacji— ponieważjuż na poziomie hardware'owego mechanizmu obsługi dysku czytany jest od razu cały sektor. Gdybyśmy chcieli odczytać cały sektor w trybie znakowym— byłby on czytany tyle razy— ile zawiera znaków. Dostęp do urządzeń blokowych sprowadza sie do zapisu i odczytu bloków o stałej wielkości.

Dla wszystkich urządzeń blokowych Linux dostarcza mechanizm pamięci buforowej— dzięki któremu możemy uzyskać znaczne przyśpieszenie komunikacji. O ile system otrzyma zadanie odczytu bloku— sprawdza najpierw czy w pamięci buforowej nie znajduje się aktualna kopia tego bloku (wtedy nie trzeba go czytać z urządzenia).

Podprogram obsługi urządzenia jest identyfikowany parą (litera— liczba). Jeśli litera jest 'b' to mamy do czynienia z urządzeniem blokowym— jeśli 'c' to znakowym. Właśnie ta litera wskazuje w której z dwóch tablic rozdzielczych znajduje urządzenia. Liczba jest jednobajtowa identyfikująca pozycję w tablicy rozdzielczej.

1.5 TABLICE ROZDZIELCZE

Tablice rozdzielcze urządzeń blokowych i znakowych służą do przechowywania informacji o zainstalowanych urządzeniach w systemie oraz o funkcjach dostępu do tych urządzeń. Wywołanie funkcji systemowej (np. open— close) dla pliku specjalnego przechodzi przez tablicę rozdzielczą odpowiadającą typowi danego pliku.

Istnieje dwie tablice rozdzielcze urządzeń: znakowa i blokowa.Każdemu podprogramowi obsługi odpowiada conajmniej jedna pozycja w tablicy. Elementy w tablicach to są struktury:

```
struct device_struct {
   const char* name
     /*wskaźnik do nazwy podprogramu obsługi*/
   struct file_operations *fops
     /*dostęp do podprogramu obsługi spoza
       jądra przechodzi przez te funkcje--- których
wskaźniki znajdują się w tej strukturze */
static struct device_struct chrdevs [MAX_CHRDEV]
  = {{ NULL--- NULL }---};
static struct device_struct blkdevs [MAX_BLKDEV]
  = {{ NULL--- NULL }---};
```

Stale MAX_CHRDEV oraz MAX_BLKDEV sa zdefiniowane w pliku major.h a ich wartosc jest równa 32.

Podprogram obsługi urządzenia jest uważany za zainstalowany w systemie tylko wtedy gdy zajmuje jakąś pozycję w tablicy rozdzielczej. Każde urządzenie ma standardowy numer identyfikujący. Można go znaleźć w pliku linux/Documentation/devices.txt.

Numer urządzenia=((numer główny«8)|numer drugorzędny) /*ang.major«8|minor*/

Numer główny i drugorzędny są liczbami ośmiobitowymi. Pierwsza identyfikuje pozycję w tablicy urządzeń— drugi jest to liczba przekazywana podprogramowi podczas wszelkich operacji na urządzeniu. Łącznie z flagą określającą typ urządzenia ('b' lub 'c') stanowią trzema składnikami i-węzła opisującego plik specjalny.

Często zdarza się— że do systemu mamy podłączone wiele urządzeń tego samego typu (np. identyczne dyski).Wtedy ich obsługa będzie realizowana przez jeden podprogram. Numer główny określa nam grupę urządzeń do której należy dane urządzenie. Wszystkie urządzenia z danym numerem głównym są obsługiwane identycznie. Numer drugorzędny może decydować o tym— o które z wielu takich samych urządzeń nam chodzi zapewniając nam unikatowość w danej grupie numeru głównego.

Oprócz swojego numeru głównego i drugorzędnego (MAJOR i MINOR) urządzenie posiada w Linuxie swój unikalny numer w polu i_dev w i-węzle. Zmienna jest typu dev_t (zdefiniowanego jako short). Przykład:

```
major=3 minor=0                dysk 0
              i w [1..63]      partycja i dysku 0
        minor=64               dysk 1
              i w [65..127]    partycja i-64 dysku 1

major=1 minor=1         pamięć fizyczna
      2         pamięć wirtualna jądra
```

```
    3        urządzenie /dev/null
    4        dostęp do portu wejścia-wyjścia
            5        generator zer (/dev/zero)
            8  generator liczb losowych.
```

Jak widać urządzenia spełniają różnorodne funkcje— ale są obsługiwane przez ten sam podprogram obsługi

Funkcje operujące na tablicach rozdzielczych urządzeń to:

- int get_device_list (char * page)

  Na zmienną page zwraca informację o zainstalowanych urządzeniach— wypisuje kolejno (dla urządzeń znakowych i blokowych) numer i nazwę urządzenia. - struct file_operations * get_blkfops (unsigned int major) Dla zadanego wirtualnego urządzenia blokowego (parametr major) zwraca strukture file operations z tablicy rozdzielczej.

- int register_blkdev (unsigned int major— const char * name— struct file operations * fops)

  Instaluje w systemie nowe urządzenie - wpisuje do tablicy rozdzielczej nowe urządzenie blokowe o podanych parametrach. Jako liczbę major można podać 0— wtedy szuka pierwszego wolnego numeru. Możliwe błedy: EBUSY (gdy major = 0 i nie ma wolnego miejsca— albo podano juz zajęty major)— EINVAL (gdy major > MAX_BLKDEV).

- int unregister_blkdev (unsigned int major— const char * name)

  Wyinstalowuje urządzenie - ustawia jego nazwę i wskaźnik do operacji na NULL. Błąd: EINVAL (gdy zły numer major lub dla podanego numeru nie zgadza sie nazwa).

- int check_disk_change (dev_t dev)

  Sprawdza czy zadane urządzenie o wewnętrznym numerze dev nie zmieniło nośnika (np. wyjęcie dyskietki— zmiana płyty CD itp.). Jeżeli tak to aktualizuje dane. Dla każdego urządzenia jest inna funkcja sprawdzająca zmianę nośnika - pole check_media_change w strukturze fops.

Tablice rozdzielcze są inicjowane podczas ładowania systemu. Odpowiadają za to funkcje chr_dev_init() (w mem.c) i blk_dev_init() (w ll_rw_blk.c). Przykładowo:

```
chr_dev_init();
  - register_chr_dev(MEM_MAJOR--- mem"--- &memory_fops);
  - tty_init();
  - lp_init();
  - mouse_init();
  - itp.
```

1.6 DWA RODZAJE PODPROGRAMÓW OBSŁUGI URZĄDZEŃ

Jeżeli podprogram obsługi urządzenia otrzymuje żądanie wykonania jakiejś operacji na tym urządzeniu i przekaże to żądanie do urządzenia— to musi w jakiś sposób zorientować się— że zostało ono wykonane (albo że wystąpił błąd). Zwróćmy uwagę— że dotyczy to zarówno niemal wszystkich podprogramów obsługi urządzeń zewnętrznych— jak również niektórych urządzeń wirtualnych— które wykonują np. read w sposób blokujący (jak /dev/random). Urządzenia wirtualne mogą wieszać się po prostu na jakiejś kolejce (jak w random_read)— czekając na zajście pewnego wydarzenia w systemie. Inaczej jest z obsługą urządzeń zewnętrznych.Podprogram obsługi takiego urządzenia może:

- co pewien czas sprawdzać— czy jego żądanie zostało już wykonane— czyli ódpytywać" urządzenie. Taki podprogram obsługi urządzenia nazywamy odpytującym (ang. polling device driver).

  Sterownik odpytujący nie korzysta z przerwań— wykonuje na raz jedno zadanie i to w sposób ciagły (nie przerywa pracy na wykonanie innego zadania). Najważniejsza jednak cecha (jak rownież najwieksza wada) jest blokowanie systemu podczas odczytu z urządzenia (aktywne czekanie na możliwość zapisu i odczytu). Zaleta jest prosta budowa sterownika— niewielki nakład pracy systemu na obsługę wejścia/wyjścia (nie licząc aktywnego czekania) oraz nie zajmowanie przerwań. Ostatnia cecha jest o tyle ważna—że często dysponujemy niewielką liczbą wolnych przerwań. Sterowniki odpytujące można stosować do urządzeń— dla których rzadko występuje sytuacja wielu zleceń w jednym czasie albo też są one równie szybkie jak procesor.

```
             proces1 proces2
 ________________|________|_________
|     __________       ____________ |
|    |pobór     |<-|  | __________ ||
|   _|urządzenia|  |__||          |||
|  | |__________|     ||urządzenie|||
|  |  __________  |-->||          |||
|  ->|sterownik | |   ||__________|||
|   _|odpytujący| |   |            ||
|  | |__________| |   |            ||
|  |  _________   |   |            ||
|  | |zwrot    |  |   |            ||
|  ->|wyniku   |__|   |    SO      ||
|    |_________|      |____________||
|___________________________________|
```

W zamieszczonym przykładzie zadanie Procesu 1 jest realizowane— a Proces 2 musi czekać na zwolnienie urządzenia. Oczekiwanie to jest realizowane przez aktywne czekanie - co jakiś czas Proces 2 sprawdza— czy urządzenie jest wolne. Dodatkowo warto zauważyć— że Proces 1 aktywnie czeka na otrzymanie wyników.

- czekać na przerwanie od urządzenia— które w ten sposób zasygnalizuje wykonanie operacji. Taki podprogram obsługi urządzenia nazywamy przerwaniowym (ang. interrupt-driven device drivers). Zwykle— ze względu na możliwość awarii lub zgubienia się przerwania— podprogram obsługi ustawia sobie jakiś maksymalny czas oczekiwania (ang. timeout)— po upływie którego próbuje np. retransmisji lub innego działania naprawczego.

W systemie Linux większość podprogramów obsługi działa przerwaniowo— ale niektóre— np. drukarka— mogą działać w trybie odpytywania. Jeżeli podprogram obsługi ma działać przerwaniowo— to musi umieć zdobyć sobie linię IRQ i zainstalować procedurę obsługi przerwania.

Sterownik przerwaniowy dzięki przerwaniom jest w stanie uniknąć aktywnego czekania. W przypadku urządzeń blokowych dodatkowo może być wykorzystana pamięć buforowa:

```
            procesy
--------------------------|-----
|      pobranie żadań__________ |
|      _________    | pamięć   ||
|  ___/______   |___| buforowa ||<---|
| |urządzenie|      |__________||    |
| |blokowe   |      |  ______  ||  __|____
| |__________|      |  ______  || |wyniki |
|     |------------>|  ______  || |_______|
|     przerwanie    |   SO     ||____|
|                   |__________||
|_______________________________|
```

Zadanie odczytu i zapisu jest kierowane najpierw do pamięci buforowej. W przypadku zadania odczytu— gdy nie ma w pamięci buforowej odpowiednich bloków— zadanie jest odkładane na pewną kolejkę— z której zadania są pobierane przez urządzenie (niekoniecznie po jednym). W oczekiwaniu na dane— 1system operacyjny może wykonywać inne zadania. Urządzenie po wykonaniu pobranych zadań wysyła PRZERWANIE— które budzi sterownik urządzenia— który zajmuje się odesłaniem danych do pamięci buforowej— skąd mogą być one pobrane przez proces.

W przypadku zadania zapisu— system może zadecydować o odłożeniu na kolejkę zadania zapisu— albo też może przetrzymać dane zadanie na pewien okres (opóźniony zapis).

W Linuxie niektóre urządzenia mogą być sterowane na obydwa sposoby. Np. sterownik drukarki standardowo działa jako sterownik odpytujący— ale lokalizując kartę I/O dostarczającą przerwanie dla drukarki— sterownik może przejść w tryb przerwaniowy.

1.7 ALGORYTMY OPEN— CLOSE— READ— WRITE— IOCTL— LSEEK— SELECT

W Linuxie— tak jak w innych Unixach— dostęp do urządzeń zewnętrznych odbywa się poprzez system plików. Wobec tego operacje wykonywane na urządzeniach zewnętrznych są takie same jak na plikach - są to: open— close— read— write— lseek lecz także ioctl— select— check_media_change i revalidate.

Jeśli plik— na którym któraś z tych operacji jest wykonywana jest plikiem specjalnym(tzn. plikiem urządzenia)— to wówczas przypisana mu w jego i-węzle tablica rozdzielcza file_operations wskazuje na tablicę danego urządzenia i jest wykonywana funkcja specyficzna dla tego urządzenia.

Tablica rozdzielcza zdefiniowana jest za pomocą następującej struktury (zawartej w pliku include/linux/fs.h):

```
struct file_operations
int (*lseek) (struct inode *— struct file *— off_t— int);
int (*read) (struct inode *— struct file *— char *— int);
int (*write) (struct inode *— struct file *— const char *— int);
int (*readdir) (struct inode *— struct file *— void *— filldir_t);
int (*select) (struct inode *— struct file *— int— select_table *);
int (*ioctl) (struct inode *— struct file *— unsigned int— unsigned long);
int (*mmap) (struct inode *— struct file *— struct vm_area_struct *);
int (*open) (struct inode *— struct file *);
void (*release) (struct inode *— struct file *);
int (*fsync) (struct inode *— struct file *);
int (*fasync) (struct inode *— struct file *— int);
int (*check_media_change) (kdev_t dev);
int (*revalidate) (kdev_t dev);
;
```

Jak widać— wszystkie funkcje dostają wskaźnik na i-węzeł oraz na plik— lub zakodowany numer główny i drugorzędny urządzenia. W pierwszym przypadku poznają drugorzędny numer urządzenia (potrzebny do jego zidentyfikowania) ze struktury reprezentujacej i-wezeł (pole rdev).

Należy jednak zaznaczyć— że znaczną wiekszość rzeczy— które system musi zrobić— wykonuje sama funkcja systemowa— wobec czego charakterystyczna dla urządzenia funkcja może być bardzo krótka— lub wręcz nie robi nic (tak jest czesto np. w przypadku lseek).

Wobec tego— proces wykonywania funkcji systemowych dla plików specjalnych można podzielić na dwie części:

1. Wykonanie funkcji systemowej— praktycznie identyczne jak dla pliku zwyklego.

2. Wykonanie funkcji specyficznej dla urządzenia— wskazywanej przez odpowiedni element tablicy rozdzielczej.

Funkcje specyficzne dla urządzeń są zdefiniowane w programach obsługi urządzeń znajdujących sie w katalogu drivers/char lub drivers/block (dla urządzeń znakowych i blokowych— odpowiednio). Są one inne dla każdego rodzaju urządzenia (tzn. dla każdego urządzenia o innym numerze głównym).

Dlatego też dużo ciekawszymi funkcjami są specyficzne dla plików specjalnych funkcje select— ioctl— check_media_change i revalidate.

1. ***lseek***

   DEFINICJA: asmlinkage long sys_lseek(unsigned int fd— off_t offset— unsigned int origin)

   WYNIK: nowa pozycja lub

   EBADF (fd jest błędnym deskryptorem)—

   EINVAL (origin jest błędną wartością).

   Funkcja lseek służy do zmiany miejsca— w którym zostanie wykonana następna operacja odczytu lub zapisu. Realizuje ją funkcja systemowa sys_lseek. Offset oznacza miejsce (prawdopodobnie liczbe bajtów od początku)— od którego ma się rozpocząć najbliższa operacja odczytu/zapisu— a origin ma nastepujące znaczenie:

   0 = liczymy offset od samego poczatku;

   1 = liczymy offset od aktualnej pozycji;

   2 = liczymy offset od końca.

   Jeśli w tablicy rozdzielczej urządzenia wskaźnik lseek jest ustawiony na NULL (tak jest w przypadku wiekszości urządzeń)— wykonywana jest standardowa operacja— polegająca na odpowiednim zmodyfikowaniu wskaźnika file->f_pos. Funkcja lseek zwraca pozycję w pliku.

   Jeśli zdefiniowana jest lseek w programie obsługi urzadzenia— jest ona wykonywana (tasma np. może zostać przewinięta w odpowiednie miejsce).

2. ***open/close (release)***

   DEFINICJA: asmlinkage int sys_open(const char * filename—int flags—int mode)

   WYNIK: nowy deskryptor lub

   ENFILE (przekroczona max liczba otwartych deskryptorow procesu)—

   EMFILE (przekroczona max liczba otwartych deskryptorow pliku)—

   ENOTDIR— ENOENT— ENAMETOOLONG— EFAULT— EACCES— ETXTBSY— EISDIR.

   DEFINICJA: asmlinkage int sys_close(unsigned int fd)

   WYNIK: 0 (sukces) lub

   EBADF (fd nie jest poprawnym— otwartym deskryptorem).

   Funkcja open służy do otwierania pliku specjalnego (rozpoczęcie pracy z urządzeniem)— a funkcja close - do zamknięcia pliku specjalnego. Funkcje systemowe sys_open i sys_close dzialają identycznie jak w przypadku plików zwykłych.

Podczas otwierania pliku system przydziela miejsce w pamięci na i-wezeł— zwieksza jego licznik odwołań— przydziela pozycję w tablicy plików i deskryptor. Warto pamiętać— że w i-wezle pliku specjalnego— w polu rdev zakodowany jest numer główny i drugorzędny urządzenia— którego tenże plik dotyczy.

Funkcja sys_open wywołuje funkcje *_open specyficzną dla urządzenia (jeśli jest ona zdefiniowana w tablicy rozdzielczej). Funkcja otwarcia urządzenia musi utworzyć wszystkie potrzebne sobie struktury danych— ew. uruchomić urządzenie.

Funkcja sys_close wywoluje funkcje *_release tylko wtedy— gdy już żaden proces nie używa i-wezła.

Funkcje sys_open i sys_close są zdefiniowane w pliku fs/open.c

Funkcja *_open zdefiniowana w programie obsługi bardzo często liczy liczbę odwołań do urządzenia (nie można do tego celu wykorzystać liczby odwołań do i-wezła— ponieważ jedno urządzenie może mieć kilka i-wezłów; można utworzyć kilka plików specjalnych reprezentujących ten sam numer główny i drugorzędny). Ten licznik jest wykorzystywany poźniej— przy *_release. Jeśli żaden proces nie używa urządzenia— można je (w razie potrzeby) np. wyłączyć. W przypadku urządzeń blokowych— funkcja *_release musi jeszcze pamietać o ew. zapisaniu wszystkich buforów znajdujących się wciąż w pamięci (robi to np. wykonując block_fsync).

3. ***read/write***

   DEFINICJA: asmlinkage int sys_read(unsigned int fd—char * buf—int count)

   WYNIK: liczba przeczytanych bajtów lub

   EFAULT (buf poza przestrzenią adresową)

   EINVAL (nie można czytać z urządzenia)

   EBADF (błędny deskryptor)— EISDIR (katalog)

   EAGAIN (operacja nieblokująca nie dała rezultatów)

   EINTR (przerwanie przez sygnał)

   DEFINICJA: asmlinkage int sys_write(unsigned int fd—char * buf— unsigned int count)

   WYNIK: liczba zapisanych bajtów lub

   EFAULT— EINVAL— EBADF— EAGAIN— EINT (jak wyzej)— EPIPE.

   Argumenty:

   buf - wskaźnik na początek bufora;

   count - liczba bajtów do odczytania/zapisania;

Wynik: ilość odczytanych/zapisanych bajtów.

Te funkcje służą do czytania i pisania z/na urządzenie. Są one obsługiwane przez funkcje systemowe sys_read i sys_write zdefiniowane w pliku fs/read_write.c.

Funkcje systemowe uruchamiają odpowiednie funkcje zdefiniowane w programie obsługi urządzenia. W przypadku urządzeń znakowych są to po prostu funkcje *_read i *_write— które odpowiednio czytają/zapisują na urządzenie. W przypadku urządzeń blokowych sprawa jest bardziej skomplikowana— ponieważ odczyt i zapis jest buforowany i odbywa się poprzez tzw. procedury strategii. Najczęściej wówczas wskaźniki w tablicy rozdzielczej wskazują na funkcje (odpowiednio) block_read i block_write zdefiniowane w pliku fs/block_dev.c— które wywołują nastepnie procedury strategii. (Dokladnie tak jest w przypadku dysku twardego; w przypadku stacji dysków elastycznych istnieją funkcje floppy_read i floppy_write— których jedyna funkcja oprócz uruchamiania block_read/block_write jest jedynie sprawdzanie— czy nie została zmieniona dyskietka).

4. ***Procedury strategii***

   Procedury strategii służą do czytania/zapisu z urządzeń blokowych za pomocą buforów.

   Jeśli program żąda odczytania/zapisania danych— system określa blok— w którym te dane się znajdują. Jeśli ten blok jest w buforze (w pamięci)— pobiera go stamtąd. Jeśli nie— bufor tworzy zadanie (request)— które ma wykonać operację dyskową.

   Wobec tego— programy obsługi urządzeń blokowych nie posiadają funkcji *_read i *_write— tylko procedury obsługi żądań do_*_request.

   Działa to następująco (omawiamy czytanie; zapis jest analogiczny):

   * określamy— które bloki dyskowe mamy przeczytać (włącznie z ew. czytaniem z wyprzedzeniem - read ahead);

   * pobieramy algorytmem getblk nagłówki dla każdego bloku - może okazać się— że blok jest już w pamięci.

   * wywołujemy ll_rw_block;

   - ll_rw_block tworzy żądania.

   - Dodajemy żądanie do kolejki żądań (all_requests) za pomocą funkcji make_request;

   - Uwaga: żądania zapisu mogą zająć co najwyzej 2/3 tej kolejki. Żądania są wsortowywane za pomocą algorytmu windowego (ang. elevator algorithm)— który minimalizuje czas ruchu głowicy. Jest to realizowane za pomocą nastepującego porządku:

   1. W starszych wersjach Linuxa (nie w 2.0): żądania odczytu przed żądaniami zapisu;

2. Żądania pochodzące od urządzeń o mniejszych numerach drugorzędnych najpierw;

3. Żądania dotyczące bloków o mniejszych numerach najpierw; Algorytm windowy realizuje makro IN_ORDER() zdefiniowane w include/linux/blk.h:

#define IN_ORDER(s1—s2) ((s1)->rq_dev<(s2)->rq_dev|| (((s1)->rq_dev==(s2)->rq_dev && (s1)->sector<(s2)->sector)))

- lista all_requests mieści NR_REQUEST żądań (standardowo 64). W tym miejscu proces może zostać uśpiony (w stanie TASK_UNINTERRUPTIBLE)— czekając na przydzielenie slotu na żądanie.

* jeśli nasze żądanie jest pierwsze na liście— wywołujemy zdefiniowaną dla każdego urządzenia blokowego funkcje request_fn (jest to właśnie procedura strategii)— która inicjuje realizację żądania (komunikuje się z urządzeniem zewnętrznym i ustawią funkcje— która uruchomi się po nadejściu przerwania).

* po nadejściu przerwania (zakończeniu operacji) uaktualniamy bufor— budzimy ewentualne procesy— które być może czekają na zwolnienie slotu w all_requests i realizujemy następne żądanie z kolejki.

5. ***ioctl***

   DEFINICJA: asmlinkage int sys_ioctl(unsigned int fd— unsigned int cmd— unsigned long arg)

   WYNIK: 0 (sukces) lub

   EBADF (błędny deskryptor)—

   ENOTTY/EINVAL (błędny kod zadania dla danego urządzenia).

   Argumenty:

   cmd - kod polecenia

   arg - argument (zazwyczaj wskaźnik do struktury)

   ioctl jest realizowana przez funkcję systemową sys_ioctl— zawartą w pliku fs/ioctl.c. Jest to ogólny interface dla wszystkich poleceń specyficznych dla urządzeń— których nie da się zrealizować za pomocą pozostałych funkcji. Są to przeważnie funkcje sterownicze— definiujące parametry pracy danego urządzenia.

   Dla każdego urządzenia komendy ioctl są inne. Sama funkcja sys_ioctl obsługuje ich tylko kilka (np. FIOCLEX - ustawia flagę close_on_exit itd.).

   W przypadku stacji dysków elastycznych np. ioctle służa do: wyjmowania nośnika— formatowania dysków— pobierania parametrów nosników i samego napędu itd. W przypadku dysków twardych można za ich pomocą pobrać geometrię dysku— odczytać tablicę partycji itd. Duże znaczenie ma ioctl przy ustawianiu parametrów terminali (patrz Terminale).

6. ***check_media_change/revalidate***

   Te funkcje są zdefiniowane tylko dla urządzen z wymienialnym nośnikiem (np. stacja dysków elastycznych). check_media_change sprawdza— czy nośnik został zmieniony i— ewentualnie— przez revalidate uaktualnia wszystkie informacje— jakie system posiada o urządzeniu (np. tablice i-wezlow). W szczególności— w pliku devices.c jest zdefiniowana funkcja

   int check_disk_change (kdev_t dev)

   Jest ona wywoływana podczas mount i open— a jej działanie bardzo dobrze pokazuje korzystanie z dwóch opisywanych funkcji:

   * wywołuje fops->check_media_change(dev);

   * uaktualnia superblok— i-wezly— bufory;

   * jeśli trzeba (tzn. jesli jest zdefiniowana)— wywołuje fops->revalidate(dev).

7. ***select***

   DEFINICJA: asmlinkage int sys_select(int n— fd_set *inp— fd_set *outp— fd_set *exp— struct timeval *tvp)

   WYNIK: liczba deskryptorów— których stan sie zmienił lub

   EBADF (w jednym ze zbiorów znalazł się błędny deskryptor)—

   EINTR— EINVAL (n<0)— ENOMEM (brak pamięci).

   Argumenty:

   n - największy numer deskryptora użytego w zbiorach inp— outp i exp plus 1.

   inp - zbiór deskryptorów urządzeń— na których mają pojawić się dane;

   outp - zbiór deskryptorów urządzeń— na których ma się pojawić możliwość zapisu;

   exp - zbiór deskryptorów urządzeń— na których mają się pojawić wyjątki;

   tvp - maksymalny czas— który mamy czekać (gdy = 0— czekamy do skutku).

   select służy do odpytywania urządzeń. Stosuje się je w sytuacjach— kiedy nie chcemy tracić czasu procesora na aktywne sprawdzanie— czy na którymś z urządzeń można już coś zapisać lub z niego odczytać. Funkcja służy do ominięcia aktywnego czekania w pętlach z operacją read/write z flagą O_NDELAY. Select kończy się— gdy:

   * na deskryptorze ze zbioru inp pojawiły się jakieś znaki do odczytu;

   * na deskryptorze ze zbioru outp można natychmiast coś zapisać;

   * na deskryptorze ze zbioru exp pojawił się wyjątek;

   * skończył się czas określony przez tvp.

Po zakończeniu działania funkcji zbiory deskryptorów są modyfikowane— aby zaznaczyć— które z nich zmieniły stan (tzn. na których odpowiednie operacje mogą się powieść).

Zbiory deskryptorów (struktura fd_set) obsługuje się za pomocą makr zdefiniowanych w include/asm-i386/posix_types.h:

FD_CLR(int fd— fd_set *set) - usuwa fd ze zbioru set;

FD_ISSET(int fd— fd_set *set) - zwraca 1 jeśli fd należy do set;

FD_SET(int fd— fd_set *set) - dodaje fd do zbioru set.

FD_ZERO(fd_set *set) - czyści zbiór set.

Funkcja select działa następująco:

* zerujemy zbiory wyjściowe res_in— res_out i res_ex;

* zmieniamy stan procesu na TASK_INTERRUPTIBLE;

* for i=0..n-1 (petla po wszystkich deskryptorach) jeśli deskryptor i należy do któregoś ze zbiorów i operacja select specyficzna dla urządzenia zwraca 1 to dodajemy ten deskryptor do odpowiedniego zbioru wyjściowego;

* jeśli (nic nie znaleźliśmy i mamy czas)— to: schedule(); powrót do for;

* zmieniamy stan procesu na TASK_RUNNING. Idea tej funkcji polega na tym— że po pierwszym sprawdzeniu wszystkich deskryptorów proces jest usypiany— po czym może zostać obudzony dopiero po otrzymaniu przerwania od któregoś z odpytywanych urządzeń. Wówczas petla sprawdzająca jest wykonywana ponownie.

2 Urządzenie „mem” i urządzenia blokowe

Autor : Piotr Sawicki

2.1 Urządzenie „mem”

Programy obsługi urządzeń mogą dotyczyć nie tylko fizycznych peryferiów, ale również urządzeń abstrakcyjnych. Dzięki temu można dla wygody użytkownika zaimplementować dowolne potrzebne operacje wewnątrz podsystemu wejścia/wyjścia (co daje wszystkie zalety systemu plików i spojną komunikację programów ze środowiskiem). Przykładem tego jest urządzenie znakowe „mem”. Jego numery podrzędne oznaczają różne funkcje wykonywane przez odrębne podurządzenia :

1. /dev/mem
   Pozwala na dostęp do pamięci.
2. /dev/kmem
   Jak wyżej, jednak dotyczy pamięci jądra systemu operacyjnego.
3. /dev/null
   „Nicość” - wysyłane tam dane giną bezpowrotnie (nie zajmując zasobów komputera). Pozwala to na „wysypywanie do śmieci” niepotrzebnych informacji.
4. /dev/port
   Umożliwia dostęp do portów wejścia/wyjścia (przez które komputer komunikuje sie z urządzeniami zewnętrznymi).
5. /dev/zero
   Nieskończony zasobnik zer. Przydatny np. przy czyszczeniu obszarów dysku.
6. /dev/full
   To podurządzenie istnieje tylko po to, żeby generować błąd braku miejsca przy próbie zapisu.
7. /dev/random
   Generuje liczby pseudolosowe, czekając, aż osiągnie wystarczający poziom losowości (dzięki przypadkowym zdarzeniom w systemie).
8. /dev/urandom
   Podobnie jak wyżej, z tym że działa nieblokująco (ale przez to mniej losowo).

2.2 Porty

Do komunikacji z urządzeniami zewnętrznymi służą tzw. porty wejscia/wyjscia. Można się do nich odwyływać bezpośrednio (funkcje systemowe in* i out*) lub za pomocą urządzenia /dev/port (mimo większej wygody jest to niestety wolniejsze).

Aby programy obsługi urządzeń wiedziały, przez które porty porozumiewać się z właściwym sprzętem, konieczna jest (o ile numery nie zostały z góry ustalone) inicjalizacja. Polega ona na szukaniu odpowiednich portów metodą prób i błędów (licząc na prawidłowe zachowanie podłączonego urządzenia).

Może to jednak przez przypadek doprowadzić do kolizji (omyłkowego rozpoznania tego samego portu przez różne programy obsługi). Można temu zapobiec dzięki mechanizmowi rezerwacji portów. Port zarezerwowany nie podlega dalszym poszukiwaniom. Do zapamiętania, które spójne fragmenty przestrzeni portów są zarezerwowane, służy w systemie tablica połączonych w listę zakresów numerów portów.

2.3 Generator liczb losowych

W wielu zastosowaniach komputerów, mimo że są to urządzenia deterministyczne, przydatne jest wykorzystanie zjawiska losowości - np. w kryptografii, algorytmach randomizowanych czy choćby losowaniu sygnaturek do listów. Dobrym przykładem jest szyfrowanie i uwierzytelnianie komunikatów. Metodę szyfrowania kluczem tajnym, mimo szybkich i wydajnych implementacji, dyskwalifikuje konieczność jawnego przekazania klucza. Z drugiej strony szyfrowanie kluczem publicznym, choć wygodne i bezpieczne, jest bardzo powolne z powodu konieczności użycia skomplikowanych algorytmów. Z tego powodu stosuje się tzw. cyfrowe koperty, czyli dane są szyfrowane losowym kluczem tajnym, który z kolei jest szyfrowany kluczem publicznym (zysk na szybkości i bezpieczenstwie). Wymaga to jednak umiejętności tworzenia liczb (pseudo)losowych. Pewną przypadkowość można uzyskac na podstawie trudnych do przewidzenia zdarzeń w systemie operacyjnym. Właśnie na tym polega implementacja generatora liczb losowych w Linuxie.

Podstawą generatora jest 512-bajtowy pojemnik losowości - jego stan ulega zmianom pod wpływem zdarzeń dotyczących tzw. źródeł losowości. Są to :

- mysz
- klawiatura
- urządzenia blokowe
- linie przerwań opatrzone atrybutem SA_SAMPLE_RANDOM
- generator liczb losowych (jego wykorzystywanie jest losowe!)

Ponieważ zdarzenia takie nie są zupełnie losowe (np. znaki wciskane na klawiaturze są w dużym stopniu ze sobą skorelowane), to dla każdego z nich obliczany jest wnoszony poziom entropii.

Utworzenie liczby pseudolosowej polega na wykonaniu funkcji mieszającej (MD5 lub SHA) na pojemniku losowości (który jest następnie modyfikowany, żeby kolejne wywołanie nie zwróciło tej samej liczby).

Generator liczb losowych może działać w dwóch trybach :

- blokującym (/dev/random) - liczby pseudolosowe tworzone są dopiero, gdy w pojemniku losowości panuje odpowiedni poziom entropii
- nieblokującym (/dev/urandom) - liczby generowane są natychmiast po zażądaniu, ale mogą byc przez to mniej losowe

2.4 Urządzenia blokowe

Mnogość rodzajów urządzeń możliwych do podłączenia do komputera nie oznacza, że każde z nich obsługiwane jest w zupełnie inny sposób. Wszelkiego typu dyski (i peryferia im podobne), mimo różnic technicznych, zachowują się dość podobnie. Ich podstawową funkcją jest zapis i odczyt jednocześnie całych bloków bajtów za pomocą głowicy, której zajmuje dużo czasu wyszukanie żądanych danych. Konieczne są więc algorytmy buforowania wejścia/wyjścia oraz przyspieszania obsługi poprzez odpowiednie szeregowanie poleceń i przewidujące odczytywanie „na zapas". Aby nie implementować tego oddzielnie dla każdego typu urządzenia, wydzielono je w odrębną grupę urządzeń blokowych. W przeciwieństwie do pozostałych (znakowych) kod ich obsługi jest w dużym stopniu wspólny - różnią się one niskopoziomowymi procedurami komunikacji z kontrolerem urządzenia przez porty wejścia/wyjścia (choć możliwe jest również stworzenie specyficznej dla urządzenia strategii obsługiwania żądań).

W systemie pamiętane są następujące dane o urządzeniach blokowych :

- ilość bloków
- wielkość bloku
- wielkość sektora

Dla każdej z tych cech istnieje oddzielna dwuwymiarowa tablica indeksowana po numerach głównych urządzeń (oznaczających programy obsługi urządzeń blokowych - maksymalnie może ich być 128) i numerach podrzędnych urządzeń.

Dane programu obsługi

Systemowe tablice zawierają dla każdego programu obsługi urządzenia blokowego :

- ilość sektorów do przeczytania z wyprzedzeniem
- atrybuty „tylko do odczytu"
- strukturę zawierającą
  - strategię kolejności wykonywania żądań (zwykle polega ona po prostu na sekwencyjnym przetwarzaniu kolejki) - musi ona znać niskopoziomowy sposób obsługi urządzenia, czyli metody komunikacji z kontrolerem urządzenia przy pomocy portów wejścia/wyjścia i przerwań
  - kolejkę żądań wejścia/wyjścia
  - strażnika powyższej kolejki
  - kolejkę zadań (patrz : przerwania) - zawiera ona funkcję usuwającą strażnika z kolejki żądań

2.5 Żądania

Podstawowym pojęciem przy obsłudze wejścia/wyjścia jest żądanie - jest to polecenie pojedynczej operacji na urządzeniu (w przypadku odczytu/zapisu dotyczy ono spójnego fragmentu przestrzeni adresowej). Żądania nie są wykonywane natychmiast po ich zgłoszeniu - wynika to choćby ze specyfiki urządzeń blokowych oraz ich powolności względem procesora. Odrzucane są żądania zapisu na urządzeniu tylko do odczytu.

Po przyjęciu żądania system najpierw sprawdza, czy jego fizyczne wykonanie jest naprawdę konieczne. Jeśli jest to próba odczytu sektorów, których aktualne wersje są już w pamięci, lub też próba zapisu stron, które nie zmieniły się od ostatniego zapisywania (nieustawiony atrybut „dirty"), to żądanie uznaje się za wykonane dzięki przewidującym mechanizmom buforowania i odczytu z wyprzedzeniem.

Kolejną optymalizacją jest usiłowanie połączenia przyjmowanego żądania z jakimś innym, czekającym na wykonanie. Jest to możliwe, gdy dotyczą one tego samego polecenia na tym samym urządzeniu i sąsiadujących fragmentach jego przestrzeni adresowej.

Kiedy powyższe kruczki zawiodą, żądanie uznane za odrębne zostaje zapamiętane : w systemowej tablicy żądań i na liście żądań programu obsługi. Ponieważ jednak tablica jest skończona (64 pozycje), koniecznym może się okazać przeczekanie w kolejce na zwolnienie się miejsca. Dopiero wtedy następuje wsortowanie żądania w listę programu obsługi. Jest ona uporządkowana leksykograficznie : według drugorzędnego numeru urządzenia i początkowego numeru sektora, którego dotyczy żądanie. W trakcie aktualizacji struktur przerwania są maskowane. Pamiętane są następujące dane o żądaniu :

- rodzaj żądanego polecenia
- urządzenie, którego żądanie dotyczy
- bufory w pamięci oraz przestrzeń urządzenia (zakres sektorów) na których ma być wykonane wejście/wyjście
- licznik błędów
- status : aktywne/nieaktywne, oraz dla SCSI: zajęte, wykonane, w trakcie rozłączania

Obsługa żądań następuje :

- dla urządzeń SCSI : w momencie trafienia żądania na listę
- dla pozostałych urządzeń : kolejno w miarę wykonywania żądań, dopóki lista jest niepusta.

Teoretycznie żądania są obsługiwane według specyficznej dla programu obsługi strategii, jednak zazwyczaj (np. dla twardego dysku) stosowane jest po prostu pobieranie kolejnych żądań z listy (co oznacza zgodność z algorytmem C-Look).

Ponieważ żądania odczytu są bardziej krytyczne od zapisu, są faworyzowane. Dzieje się to jednak nie poprzez np. większy priorytet (zmniejszałoby to przepustowość systemu), ale przez zagwarantowanie, że w systemowej tablicy żądań zapisy mogą zajmować tylko co najwyżej 2/3 miejsca.

Wykonanie żądania :

- urządzenie informowane jest poprzez porty wejścia/wyjścia o treści polecenia
- ustawiany jest realizujący polecenie podprogram obsługi przerwania (które nastąpi gdy urządzenie się przygotuje)

W trakcie powyższej procedury blokowane są przerwania od obsługiwanego urządzenia (wyłączenie tej blokady jest możliwe, jednak niebezpieczne i niezalecane). Jeśli żądanie dotyczy odczytu, to czytane są nie tylko sektory zażądane, ale również pewna ustalona (specyficzna dla programu obsługi) liczba następnych sektorów - istnieje duże prawdopodobieństwo, że zostaną one zażądane w przyszłości. W razie wystpienia błędów w trakcie wykonywania żądania : jeśli ich liczba przekroczy z góry ustaloną wartość graniczną, następuje rezygnacja z obsługi żądania.

2.6 Dysk twardy

Obsługą dysku twardego zajmuje się program obsługi o numerze 3. Potrafi on obsugiwać 2 dyski po 64 partycje. Komunikuje się on z dyskiem przez porty wejścia/wyjścia. Z powodu małej szybkości przesuwu głowicy dysku w stosunku do prędkości procesora, wymiana danych odbywa się w trakcie przerwań (linia przerwań 14). Wywoływane są one przez kontroler dysku w momencie ustawienia się głowicy na potrzebnej ścieżce.

2.7 Operacje na dysku

Operacje udostępniane przez program obsługi dysku twardego to :

- odczyt/zapis
- rekalibracja dysku
- odczytanie informacji o partycjach i dysku (w tym geometria dysku, tablica partycji)
- synchroniczne zapisanie buforw partycji
- pobieranie/ustawianie :
    - ilości sektorów do jednoczesnego odczytu/zapisu
    - ilości sektorów czytanych z wyprzedzeniem

  - atrybutu „tylko do odczytu” dla partycji
  - maskowania przerwań od dysku w trakcie obsługi żądań

2.8 Stany dysku

W danej chwili dysk twardy może znajdować się w jednym ze stanów :

- gotowy na przyjęcie żądania
- gotowy do przesyłu danych
- zajęty
- ustawia głowicę
- index (używany do wykrywania starych napędów Seagate)
- błąd, błąd zapisu
- poprawiono błąd

2.9 Inicjowanie obsługi dysku twardego

Aby program obsługi dysku twardego mógł prawidłowo pracować należy go wcześniej zainicjować. Polega to na :

- pobraniu danych o dyskach i tablicach partycji
- ustawieniu obsługi przerwań
- rezerwacji portów wejścia/wyjścia (zwykłe+1 przeznaczony do resetowania)
- rezerwacji buforów (po 1 kB) dla każdej partycji
- rejestracji dysku twardego w tablicy urządzeń blokowych

2.10 Obsługa żądań przez dysk

Strategia obsługi żądań przez dysk twardy polega na wykonywaniu żądań kolejno z listy - dzięki jej posortowaniu przez mechanizm obsługi urządzeń blokowych daje to algorytm C-Look. Wykonanie żądania :

- przeliczenie adresu logicznego na adres fizyczny (korzystając z informacji o geometrii dysku)
- przekazanie żądania do dysku poprzez porty wejścia/wyjścia
- ustawienie obsługi przerwania (które nastąpi, gdy dysk przygotuje się do wykonania żądania)

Dysk po otrzymaniu polecenia ustawia głowicę pod zadanym adresem i zgłasza przerwanie, podczas którego następuje transmisja danych. Jeśli są jeszcze żądania w kolejce, cała operacja się powtarza.

2.11 Błędy dysku

W celu wykrycia błędów stosuje się pomiar czasu : nienadejście przerwania po z góry ustalonym okresie (0,6 s) oznacza sytuację nieprawidłową. Wychwytywane są również błędy zgłoszone przez kontroler dysku twardego. W razie wystąpienia błędu rozpoznawany jest jego rodzaj :

- uszkodzony sektor
- nienaprawialny błąd
- nie znaleziono sektora
- błąd statusu napędu
- nie odnaleziono ścieżki zerowej (możliwe, że dysk nie jest podłączony)
- nie odnaleziono znacznika adresu (możliwe, że dysk nie jest sformatowany)

Początkowo pojawienie się błędów powoduje ponawianie operacji. Po przekroczeniu krytycznej liczby błędów dysk jest rekalibrowany, a jeśli to nieprzyniesie skutku - resetowany. Co ciekawe, zresetowanie następuje nie bezpośrednio po wykryciu błędu, ale dopiero przed wykonaniem następnego żądania.

3 Przerwania

Autor: Elżbieta Filipkowska

System linux umożliwia urządzeniom takim jak zegar systemowy czy urządzenia wejścia-wyjścia asynchroniczne przerwanie pracy jednostki centralnej. Po otrzymaniu przerwania jądro zapamiętuje swój bieżący kontekst, określa przyczynę przerwania i obsługuje je tzn. wywołuje odpowiednią procedurę obsługi przerwania. Po jej zakończeniu odtwarza kontekst z chwili przerwania i kontynuuje pracę tak, jakby nic się nie stało.

Przerwania można podzielić na:

*wyjątki (exceptions) są one generowane przez sam procesor. Dzielą się na:

-niepowodzenia (faults) zgłaszane są przed pełnym wykonaniem instrukcji i umożliwiają powtórne jej wykonanie

-potrzaski (traps) sygnalizowane po wykonaniu instrukcji, jeśli spełnione są wyzwalające je warunki

-załamania (aborts) powstałe w wyniku poważnych i nienaprawialnych błędów

*przerwania sprzętowe (handware interrupts) zgłaszane na żądanie jakiegoś urządzenia przez jego sterownik przerwań

Poszczególnym przerwaniom przydzielone są pewne priorytety. W czasie obsługi przerwania, wszystkie przerwania o niższym priorytecie są zablokowane, obsługiwane będą natomiast przerwania o priorytetach wyższych. Jeśli poziom pracy procesora jest niższy niż poziom zgłoszonego przerwania, to CPU przyjmuje przerwanie przed zdekodowaniem następnej instrukcji i podnosi poziom pracy procesora tak, aby żadne inne z tego lub niższego poziomu nie mogło być zauważone podczas obsługi danego prerwania. Jeśli natomiast pozom pracy procesora jest wyższy od poziomu zgłoszonego przerwania, to nie zostanie ono zauważone przez procesor.

Ogólny opis algorytmu obsługi przerwań

Po odebraniu przerwania, jądro zapamiętuje bieżący kontekst rejestru wykonywanego procesu (odkłada na stos) i tworzy jego nową warstwę.

Następnie określa źródło lub przyczynę przerwania identyfikując jego typ. Kiedy system otrzymuje przerwanie pobiera jego numer. Dzięki temu numerowi procesor może ustanić adres procedury obsługi przerwania i wywołać ją.

Po zakończeniu się procedury obsługi przerwania, jądro dokonuje odtworzenia kontekstu z chwili przerwania. Następnie wznawia wykonanie odtworzonej warstwy kontekstu.

Algorytm Ten wyglada następująco: zachowaj poprzednią warstwę kontekstu (włóż ją na stos); określ źródło przerwania; znajdź wektor przerwania; wywołaj funkcję obsługi przerwania; odtwórz poprzednią warstwę kontekstu (zdejmij ze stosu)

Obsługa przerwań sprzętowych

Urządzenie zgłasza sterownikowi przerwań, że zaszła sytuacja o której powinien dowiedzieć się procesor. Sterownik sprawdza, czy nie jest obsługiwane przerwanie o wyższym lub równym zgłoszonemu priorytecie. Jeśli jest czeka na moment, w którym wszystkie takie przerwania zostaną obsłużone. Jeśli natomiast obsługiwane jest obsługiwane jest przerwanie o priorytecie niższym, to jego obsługa może być przerwana. Gdy sterownik może już zgłosić procesorowi przerwanie, musi jeszcze poczekać na ustawienie flagi IF. Flaga ta odblokowuje przerwania i jest ona automatycznie kasowana przez procesor przed rozpoczęciem wykonywania procedury obsługi przerwania. Następnie sterownik czeka na sygnał zakończenia obsługi przerwania EOI (End Of Interrupt). Dopuki sterownik nie otrzyma od procedury obsługi przerwania tego sygnału uważa przerwanie za obsługiwane.

Ogólny agorytm obsługi przerwań sprzętowych jest następująsy: (zmienna x oznacza numer przerwania) zachowaj rejestry; zamaskuj przerwania(x); wyślij EOI do sterownika; zwiększ intr_count; obsłuż przerwanie(x) //wywołaj procedurę wyższego poziomu odmaskuj przerwanie(x); zmniejsz intr_count;

Procedury obsługi przerwań

-fast_IRQ_interrupt() (procedura śzybka")

Procerura ta jest wywoływana wtedy, gdy jest ustawiona flaga SA_INTERRUPT.

Wykonuje ona kolejno:

1.potwierdza otrzymanie przerwania

2.maskuje przerwania

3.zwiększa licznik przerwań intr_count

4.wywołuje do_fast_IRQ z numerem przerwania joko argument

5.odmaskowuje przerwania

6.zmniejsza licznik intr_count

-IRQ_interrupt() (procedura "wolna")

Procedura ta jest wykonywana, gdy flaga SA_INTERRUPT nie jest ustawiona.

Różnini się od IRG_fast_interrupt tym, ze funkcja obsługi przerwania jest wywoływana z niezamaskowanymi przerwaniami, oraz samą funkcją obsługi

przerwań (zamiast do_fast_IRQ wołana jest do_IRQ).

-bad_IRQ_interrupt()

Ta procedura jest wywoływana dla przerwań nie mających zarejestrowanych procerur obsługi. Potwierdza ona tylko otrzymanie przerwania.

Funkcje do_fast_IRQ() i do_IRQ() są bardzo podobne. Wykonują one po prostu wszystkie zarejestrowane dla danego przerwania procedury obsługi. Zasadnicza różnica polega na tym, że funkcja do_fast_IRQ() jest (w przeciwieństwie do do_IRQ()) wykonywana przy zablokowanych przerwaniach.

Rejestrowanie procedur obsługi przerwań

Aby móc obsługiwać przerwanie, tzn. aby do_IRQ() i do_fast_IRQ() mogły wywołać procedury obsługujące dane przerwanie, należy te procedury wcześniej zarejestrować.

Robi się to przy pomocy funkcji int request_irq(unsigned int irq, void (*handler)(int, void*, struct pt_reqs*) unsigned long flags, const char* device, void* dev_id)

Funkcja ta dołącza do listy nową strukturę irqaction (Jest to struktura, która służy do pamiętania procedur obsługi przerwań pochodzących od urządzeń.

Jądro systemu przechowuje szesnastoelementową tablicę wskaźników do tych struktur - irq_action. Są one dynamicznie alokowane podczas wywołania funkcji request_irq()). Jeśli przerwanie było zablokowane odblokowuje je.

Pierwszym parametrem funkcji request_irq() jest numer przerwania, które chcemy obsługiwać. Kolejne to:

-wskaźnik do procedury obsługi przerwania

-flagi (opis dalej)

-nazwa urządzenia

-ten parametr został dodany do wywołania request_irq() w nowych wersjach Linuxa (od 1.3.70), służy do identyfikowania urządzenia w przypadku, gdy linie przerwań są współdzielone przez kilka urządzeń

Parametry, z którymi wywoływana jest funkcja handler to:

-numer linii przerwania

-wskaźnik do struktury dev_id służący do zidentyfikowania urządzenia, jeśli linia przerwania jest współdzielona przez urządzenia

-struktura opisująca rejestry procesora

Flagi

-SA_INTERRUPT

Jeśli flaga ta jest ustawiona, to obsługa przerwania jest śzybka". Następuje zamaskowanie innych przerwań, a po powrocie z procedury obsługi przerwanie nie ma szeregowania. Jeśli flaga nie jest ustawiona, to funkcja obsługi przerwania wykonuje się z odmaskowanymi przerwaniami, a po powrocie sprawdzane jest ustawienie globalnej flagi need_resched. Jeśli jest ona ustawiona , to wywoływana jest funkcja szeregująca (schedule()).

-SA_SHIRQ

Ponieważ jest tylko 16 linii przerwań, a urządzeń chcących z nich korzystać może byc więcej, ustawienie tej flagi umożliwia współdzielenie tej jednej linii przez wiele urządzeń. Wtedy dla danego numeru linii przerwania mamy listę struktur irq_action.

-SA_SAMPLE_RANDOM

Jeśli flaga ta jest ustawiona, wywoływana jest dodatkowo funkcja add_interrupt_randomness(), która zawiększa entropię generatora liczb losowych. Przerwania (pochodzące zarówno od zegara systemowego, jak i urządzeń zewnętrznych) mogą służyć jako źródło losowości dla generatora (szcególnie przerwania od urządzeń zewnętrznych bo przychodzą niereg

Wyszukiwanie linii przerwań

Aby dowiedzieć się z którą z szesnastu linii przerwań (IRQ - Interrupt Request Lines) związane jest fizyczne urządzenie programy obsługi urzadzeń korzystają z nastepujacych funkcji:

-long_probe_on()

-probe_irq_off()

W celu odszukania linii przerwania procesor, wywołując funkcję long_probe_on(), pobiera maskę z ustawionymi bitami odpowiadającymi nieaktywnym przerwaniom. Następnie zleca urządzeniu zgłoszenie przerwania (za pomocą portów wejścia-wyjścia) i czeka pewien określony czas na to zgłoszenie. Wynik odbiera wywołując funkcję probe_irq_off() podając jej jako parametr pobraną wcześniej maskę. Jeśli funkcja ta zwróci liczbę dodatnią, to jest to numer linii przerwania (tylko w tym przepadku cała ta operacja kończy się powodzeniem). Jeśli wynikiem będzie 0, oznacza to, że żadne przerwanie nie nadeszło w czasie kiedy procesor na nie czekał. Jeśli natomiast Funkcja probe_irq_off() zwróci liczbę ujemną, to nadeszło więcej niż jedno przerwanie. Liczba jest równa -(maksymalny numer przerwania jakie zostało zgłoszone). W dwóch ostatnich przypadkach nie otrzymujemy odpowiedzi na pytanie z którą linią przerwań związane jest dane urządzenie.

"Dolne" i "Górne" połowy

Procedura obsługi przerwwnia wykonuje czasem dość czasochłonne zadania. Jeśli odbywa się to przy zamaskowanych przerwaniach, to blokuje blokuje wiele przerwań z tego i niższych poziomów. Taki stan rzeczy może w efekcie doprowadzic do tego, że pewne przerwania mogą nie zostać wogóle obsłużone.

Wprowadzono podział obsługi przerwania na dwie cześci:

-"górną połowę" (top half)

-"dolną połową" (bottom half)

Pierwsza z tych części (top half) wykonuje się przy zamaskowanych przerwaniach i zajmuje sie najcześciej sprawdzeniem stanu urządzenia oraz kopiowaniem danych z urządzenia do bufora w pamięci.

Druga częsc (bottom half) jest wywoływana po pierwszej (Czesto jądro zleca wykonanie jej poza kontekstem pewnwego perocesu np. podczas szeregowania zadań).Wykonuje się z odmaskowanymi przerwaniami (zazwyczaj dłużej od top half).

Jądro LInuxa przechowuje tablicę 32 wskaźników do funkcji typu "bottom half" - bh_base. Umożliwia ono wykonywanie tych funkcji w tle (np przy wychodzeniu z funkcji systemowej t.j. podczas wykonywania ret_from_sys_call, lub w czasie wykonywania schedule() - szeregowania zadań.

Ze względu na niewielki rozmiar tablicy bh_base wprowadzono kolejki zadań.

Kolejki zadań (Task Queues)

Kolejki zadań to listy struktur tq_struct, które to struktury zawierają wskaźniki do funkcji typu "bottom half", oraz do ich argumentów.

struct tq_struct struct tq_struct *next; int sync; void (*routine) (void*); void* data

Pola tej struktury oznaczają kolejno:

-wskaźnik do następnego elementu listy

-ten parametr służy do tego, aby sprawdzić czy dana struktura już należy do listy (wtedy sync=1, wpp sync=0)

-wskaźnik do funkcji typu "bottom half"

-argumenty dla tej funkcji

Do obsługi kolejek zadań służą funkcje:

-gueue_task

Ta funkcja dołącza nową strukture tq_struct (którą pobiera jako parametr) do kolejki (także przekazywanej jako parametr)

-run_task_queue

Wykonuje wszystkie funkcje z listy podanej jako parametr. Zaczyna od końca tejże listy.Każda zdjęta z listy funkcja w czasie wykonywania się moze powtórnie się na niej umieścić.

4 Terminale

Autor : Tomasz Ostrowski

4.1 Wstęp

Terminale są to urządzenia znakowe (*Character Devices*) przeznaczone do kontaktowania się systemu z ludźmi. Z tego względu muszą być przystosowane do ludzkiego trybu pracy. Głównie do tego, że człowiek jest omylny i zanim napisze to co chciał (tak jak ja w tej chwili) to się tzry<ˆ*W*>tryz<ˆ*W*>trzy razy pomyli. Muszą także mieć możliwość przekształcania tekstu wprowadzanego przez człowieka na dane zrozumiałe dla komputera.

Także terminal, mimo że jest urządzeniem znakowym, nie może przesyłać informacji bajt po bajcie. Właśnie ze względu na ludzkie błędy, oraz przez względy wydajnościowe. Przesyła on dane 'po linijce'.

W Linuxie, dostępnych jest także sporo ułatwień, powodujących że praca na terminalu jest wygodna. O ile *administrator systemu* tego nie zabroni, każdy użytkownik może mieć otwartych kilka sesji i pracować jakby na kilku komputerach na raz. Np. osobiście, jak programuję, bardzo lubię mieć na jednym terminalu uruchomiony edytor z aktualnie pisanym programem, na drugim pisany program kompilować, a na trzecim najczęściej jakąś dokumentację.

Oprócz tego, ponieważ Linux jest systemem zorientowanym sieciowo (jak każdy Unix), ma możliwość zdalnej pracy na maszynie. Czyli można pracować na komputerze, do którego ma się dostęp jedynie przez sieć. Służą do tego tzw. *pseudoterminale*.

W dodatku terminale muszą mieć możliwość wysyłania sygnałów do podporządkowanych im procesów, np. sygnału *SIGINT*, czyli żądania zakończenia pracy (standardowo wysyłany przez kombinację klawiszy *<Ctrl>+C*), albo jej wstrzymania (sygnał *SIGSTOP* — *<Ctrl>+Z*). Muszą też powiadamiać podległe im procesy o swojej śmierci (*SIGHUP*), aby i one mogły się zakończyć lub aby zaadoptował je *init*, czyli proces o numerze 1, przodek wszystkich procesów.

4.2 Terminologia

Z terminalami związanych jest kilka pojęć. Oto niektóre z nich:

- terminal sterujący procesu

  Jest to terminal z którym związany jest dany proces. Nie każdy proces musi być związany z jakimś terminalem. Np. tzw. demony są takimi procesami. Są to procesy, które uruchamiane są najczęściej przy starcie systemu, i oferują usługi, które powinny być dostępne przez cały czas pracy komputera. Kilka z nich to:

  - kswapd zajmujący się obsługą pamięci pomocniczej (*swap*)
  - klogd obsługujący *logi systemowe*
  - crond uruchamiający programy 'co jakiś czas'

- grupa terminalowa

  Zbiór procesów mających ten sam terminal sterujący.

- dyscyplina linii

  Określa się tak wstępne przetwarzanie znaków na drodze terminal–proces. Właśnie dzięki niej można do procesów wysyłać sygnały z klawiatury. Pozwala ona także m. in. na nie wypisywanie echa wprowadzanych znaków, np. przy wpisywaniu hasła.

4.3 Dyscyplina linii

Dyscyplina linii, jak już wspomniałem, jest wstępnym przetwarzaniem znaków między terminalem a komputerem i odwrotnie. Ponieważ na samym początku, tekst z jakim ma do czynienia terminal jest brudny. Tzn. nie jest to tekst, który chcielibyśmy uzyskać, a ciąg naciśnień klawiszy zawierających sam tekst, znaki typu *<Backspace>*, czy żądania zakończenia działania programu *<Ctrl>+C*. Dyscyplina linii, jeśli sobie inaczej nie zażyczymy, zamieni ten tekst na to, co przypuszczalnie chcieliśmy napisać (np. *rm -fr /*), wypisze echo na terminal, podzieli tekst na linijki i wyśle je do procesu, którym steruje.

Oczywiście można zmieniać dyscyplinę linii (za pomocą funkcji systemowej *ioctl*) a nawet ją wyłączyć, przełączając terminal w tryb surowy (w przeciwieństwie do zwykłego — kanonicznego).

Dyscyplina linii pracuje na trzech tzw. *listach znakowych*, pełniących rolę buforów. Są to listy znakowe:

- zawierająca dane wyjściowe terminala
- zawierające nieprzetworzone dane wejściowe, wprowadzane przez użytkownika, tzw. surowa lista znakowa
- zawierająca przetworzone dane wejściowe (kanoniczna lista znakowa)

Algorytm pisania na terminal wygląda mniej więcej tak:

```
while ( nie koniec danych ) {
  while ( jest miejsce na wyjściowej liście znakowej
      i jeszcze nie koniec danych ) {
    dyscyplina linii czyta znaki z przestrzeni adresowej
      użytkownika
    przetwarza je
    zapisuje na wyjściowej liście znakowej
  }
  if ( przepełniona lista ) {
    zatrzymaj proces przesyłający dane
    prześlij do terminala znaki z listy
    wznów proces przesyłający dane
  }
```

```
  else {
    prześlij do terminala znaki z listy
  }
}
```

A algorytm czytania z terminala:

```
if ( pusta kanoniczna lista znakowa ) {
  if ( pusta surowa lista znakowa) {
    if ( terminal otwarty w trybie nieblokującym )
      return;
    czekaj aż surowa lista znakowa nie będzie pusta
  }
  if ( terminal w trybie surowym )
    skopiuj dane z listy surowej na kanoniczną
  else {
    skopiuj dane przechodząc przez dyscyplinę linii do
      pierwszego znaku końca wiersza
  }
}
kopiuj znaki z listy kanonicznej do przestrzeni adresowej
  użytkownika
```

4.4 Konsola

Konsola to jeden z trzech używanych w Linuxie typów terminali. To ten terminal, z którym mamy do czynienia, jeżeli pracujemy bezpośrednio na komputerze, w trybie tekstowym. Mówi się wtedy, że się pracuje na konsoli. Jak już wspomniałem w Linuxie wymyślono bardzo duże ułatwienie pracy, czyli *wirtualne konsole*. Jest ich kilka (zazwyczaj sześć) i można na nich niezależnie pracować, a przełącza się między nimi klawiszami *<Alt>+<F1>–<Alt>+<F6>* (inaczej *<Meta>+<F1>–<Meta>+<F6>*). Naciśnięcie tych klawiszy jest interpretowane jeszcze przed dyscypliną linii, żeby aktualna konsola nie mogła się zorientować, że nie jest już aktualna.

4.5 Pseudoterminale

Pseudoterminale są drugim typem terminali, z którym użytkownik Linuxa ma najczęściej do czynienia. Są to terminale służące do połączeń zdalnych, przez sieć. Muszą zapewniać 'udawanie' zwykłego terminala, czyli muszą mieć dyscyplinę linii itp., ale przesyłają znaki nie do pamięci kontrolowanego procesu, ale przez łącze sieciowe (po polsku *gniazdo*, po angielsku *socket*).

Przykładem pseudoterminala jest np. *xterm*, terminal dla X Windows. Chociaż niekoniecznie komputer musi być podłączony do sieci, aby móc uruchomić xterma, to obsługa sieci komputerowej musi być wkompilowana w jądro. Dzięki takiemu rozwiązaniu praktycznie nie ma ograniczenia na liczbę uruchomianych xtermów.

Inny typ terminali, to terminal szeregowy, służący do połączeń przez linie szeregowe.