Rozdział 1

Obsługa urządzenia blokowego

Ten rozdział omawia ogólnie zagadnienia związane z urządzeniami blokowymi. Wyjaśnia co to są urządzenia blokowe, w jaki sposób jądro Linuksa je obsługuje.

Autorem tego rozdziału jest Jan Słupski (*jslupski@email.com*).

1.1 Wstęp

Zadaniem systemu operacyjnego jest zapewnienie dostępu do różnych urządzeń zainstalowanych w systemie. Urządzenia mogą być bardzo różne i wymagać różnych sposobów komunikacji[1], jednak system operacyjny powinien udostępniać zunifikowane procedury dostępu do tych urządzeń programom użytkownika, tak żeby różnice sprzętowe nie były z zewnątrz widoczne.

Na poziomie programowym unifikacja zapewnina jest przez **Wirtualny System Plików** (ang. *Virtual File System* **VFS**). Dzięki niemu niezależnie od tego jaki system plików zainstalowany jest na urządzeniu, użytkownik zawsze w taki sam sposób odwołuje się do danych na nim zapisanych.

Odowołania do sprzętu odbywają się poprzez **specjalne pliki urządzeń** (ang. **device special file**) umieszczone w strukturze systemu plików. Standardowo takie pliki umieszcza się w Linuksie w katalogu `/dev`, ale nie jest to obligatoryjne. Komunikacja ze sprzętem z poziomu użytkownika odbywa się poprzez operacje na plikch specjalnych przypisanych odpowiednim urządzeniom. Jednakże użytkownik odwołuje się do tego sprzętu zupełnie tak jakby odwoływał się do zwykłego pliku. Ma do dyspozycji te same funkcje.

Za faktyczne realizowanie tej komunikacji (wymianę danych ze sprzętem) odpowiadają **podprogramy obsługi urządzenia** (ang. *device drivers*). Są to niskopoziomowe, specyficzne dla danego urządzenia, programy przypisane do konkretnych typów urządzeń. Poprzez ustalony interfejs zapewniają standardowym funkcjom dostęp do operacji na urządzeniach.

Urządzenia w Linuksie podzielone są na trzy grupy, zależnie od właściwości sprzętu – możliwych spsobów przekazywania danych. Są to urządzenia **znakowe**, **blokowe** i **sieciowe**.

[1] Tak naprawdę każde urządzenie posiada kontroler (układ elektroniczny z mikroprocesorem), z którym system operacyjny się komunikuje. Dla uproszczenia będę mówił po prostu urządzenie.

1.1.1 Urządzenie blokowe

Urządzenia blokowe, różnią się od znakowych sposobem dostępu do danych. Dostęp do pamięci którą dysponują odbywa się w trybie bezpośrednim. Oznacza to, że przy każdej operacji odczytu i zapisu możemy podać adres fragmentu pamięci o który nam chodzi. I adresy te mogą mieć losową (niesekwencyjną) kolejność. Wszystkie urządzenia, które nie spełniają tego warunku, są urządzeniami znakowymi. Z uwagi na konieczność nieograniczonego dostępu do pamięci tylko urządzenia blokowe mogą obsługiwać zamontowane systemy plików.

Zapis i odczyt z urządzeń blokowych może odbywać się tylko w ustalonych partiach danych. Taka partia danych może składać się z jednego lub więcej bloków o stałym, określonym dla danego urządzenia rozmiarze. Standardowe i najczęściej używane wielkości bloków to 512, 1024, lub 2048 bajtów. W celu lepszej optymalizacji dostępu do danych na urządzeniu blokowym wszystkie transfery są dokonywane przez **pamięć buforową** (ang. *buffer cache*). Używane są standardowe mechanizmy pamięci buforowej (opisane w innym rozdziale), ażeby zminimalizować konieczność wielokrotnych dostępów do fizycznego urządzenia, przy częstych odwołaniach do tych samych danych. W Linuksie przeznacza się dużą część pamięci operacyjnej na buforowanie dostępu do dysków twardych.

1.2 Obsługa urządzenia blokowego

Urządzenia blokowe dostępne są, poprzez wcześnij omówione pliki specjalne. Programy używają standardowych funkcji do otwierania, zamykania, odczytu i zapisu.

Każdy plik specjaly plik urządzenia ma przypisane: **nadrzędny** i **podrzędny numer identyfikacyjny** (ang. *major/minor identification number*).

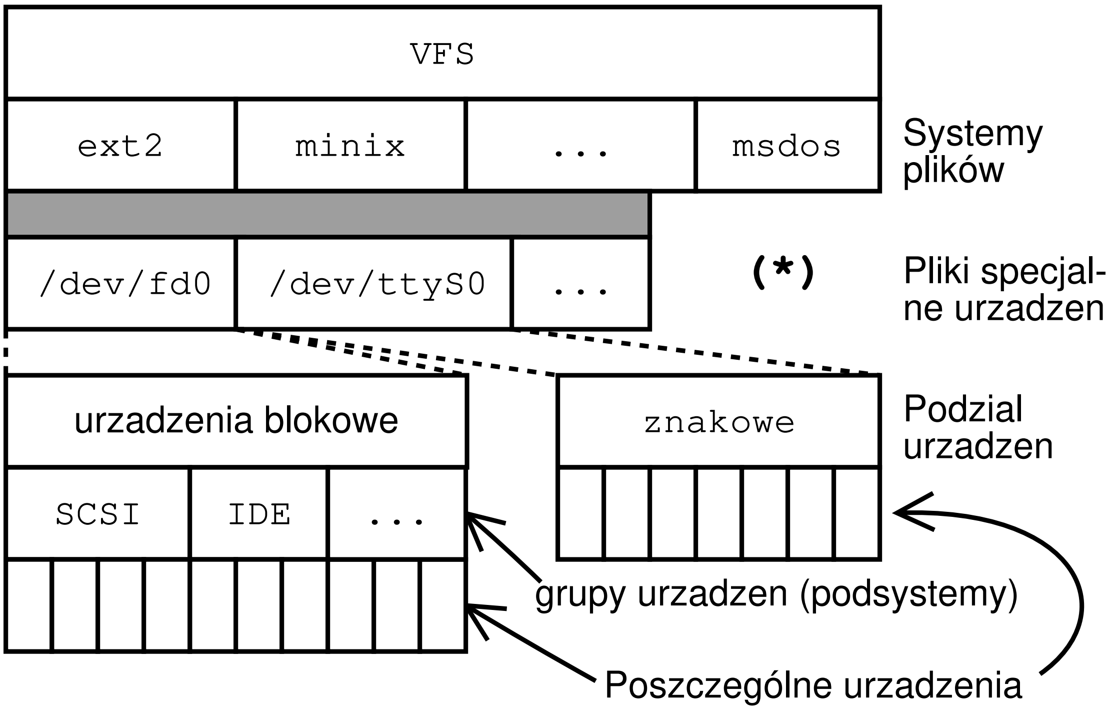
```
#ls -l /dev/hda /dev/hda1 /dev/hda2 /dev/sda
brw-rw----    1 root     disk       3,   0 Aug 24 11:00 /dev/hda
brw-rw----    1 root     disk       3,   1 Aug 24 11:00 /dev/hda1
brw-rw----    1 root     disk       3,   2 Aug 24 11:00 /dev/hda2
brw-rw----    1 root     disk       8,   0 Aug 24 11:00 /dev/sda
```

W tym przykładzie `hda1` jest urządzeniem blokowym o nadrzędnym numerze 3 i podrzędnym 1. Numer nadrzędny służy do identyfikacji urządzenia[2], a podrzędny do wyszczególnienia z jakiej części urządzenia chielibyśmy korzystać (na przykład partycji dysku twardego, lub jednego z terminali), lub przekazania programowi obsługi w jaki sposób byśmy chcieli komunikować się z urządzeniem (np. `/dev/zero` i `/dev/null`).

1.2.1 Rejestracja podprogramu obsługi urządzenia blokowego

Urządzenia blokowe dzielą się na klasy (np. SCSI, IDE, ...). Każda klasa jest **podsystemem**, który obsługuje poszczególne urządzenia w tej klasie. Na przykład urządzenie SCSI0 komunikuje się (poprzez kontroler) ze swoim podprogramem obsługi. Podprogram obsługi (jak i wszystkie inne podprogramy obsługi urządzeń SCSI) komunikują się z podsystemem SCSI, który wymienia dane już bezpośrednio z jądrem systemu.

[2] W zasadzie kontrolera urządzenia – dlatego dwa dyski podpięte do jednego urządzenia mają ten sam numer nadrzędny.



Rysunek 1.1: Podsystemy i urządzenia
(*) Nie wszystkie systemy plików dopuszczają tworzenie plików specjalnych.

Aby zapewnić jednorodność pośród wszystkich podsystemów, każdy podsystem który ma być używany musi zarejestrować się w jądrze. W tym celu uruchmia funkcję `register_blkdev()`, która dokonuje wpisu w **tablicy rozdzielczej urządzeń blokowych** `blkdevs`. Tablica `blkdevs` jest indeksowana nadrzędnymi numerami urządzeń i składa się ze struktur `device_struct` [3].

```
static struct device_struct blkdevs[MAX_BLKDEV];
struct device_struct {
  const char * name;
  struct file_operations * fops;
};
```

Struktura składa się ze wskaźnika na nazwę zarejestrowanego podsystemu, oraz z wskaźnika na strukturę `file_operations`.

Struktura `file_operations` jest zbiorem wskaźników do funkcji operacji na urządzeniu, specyficznych dla każdego podsystemu, z których korzysta kernel.

Oprócz tego każdy program obsługi urządzeń blokowych musi umożliwiać operacje związane z beśporednimi dostępami do urządzenia (z poziomu pamięci podręcznej). W tym celu – w podobny sposób jak w przypadku `blkdevs` – jest wypełniana tablica `blk_dev` [1]. W jej strukturze zapamietywany jest adres funkcji realizującej żądania dostępu do fizycznego urządzenia i zmienne związane z obsługą kolejki takich żądań.

```
extern struct blk_dev_struct blk_dev[MAX_BLKDEV];

struct blk_dev_struct {
  request_fn_proc   *request_fn;
  queue_proc        *queue;   /* queue_proc has to be atomic */
  void              *data;
  struct request    *current_request;
  struct request     plug;
  struct tq_struct   plug_tq;
  elevator_t         elevator;
};
```

1.3 Obsługa żądania I/O

Program użytkownika, gdy potrzebuje wymienić dane z urządzeniem, wywołuje funkcję odczytu, lub zapisu – `read()`, lub `write()`. Te funkcje wywołują swoje odpowiedniki w jądrze – funkcje `sys_read()` i `sys_write()` [4].

Funkcje systemowe na podstawie informacji (numeru nadrzędnego urządzenia i typu urządzenia) z inode'u otwieranego pliku (w tym przypadku pliku specjalnego urządzenia blokowego) odczytują z tablicy rozdzielczej urządzeń blokowych adresy funkcji odczytu i zapisu. W przypadku urządzeń blokowych są to funkcje `block_read()` i `block_write()` udostępnione przez VFS.

Zadaniem tych funkcji jest sprawdzenie czy żądane bloki znajdują się już w pamięci buforowej i zoptymalizowanie żądań w ten sposób, aby wykorzystanie cache'u było maksymalne, a liczba operacji wymagających komunikacji z urządzeniem minimana. Szczegóły działania tych funkcji pisane sa w rozdziałach im poświęconych.

Te żądania, których nie udało się obsłużyć z wykorzystaniem pamięci buforowej, są przekazywane do funkcji `ll_rw_block()` [2]. Jej zadaniem jest wczytanie do pamięci podręcznej żądanych danych. Przed sformułowaniem ostatecznego żądania do urządzenia przeprowadzana jest kolejna optymalizacja. Sprawdzane jest, czy któreś z żądanych danych nie zostały już odświerzone, a pozostałe próbuje się łączyć w większe bloki.

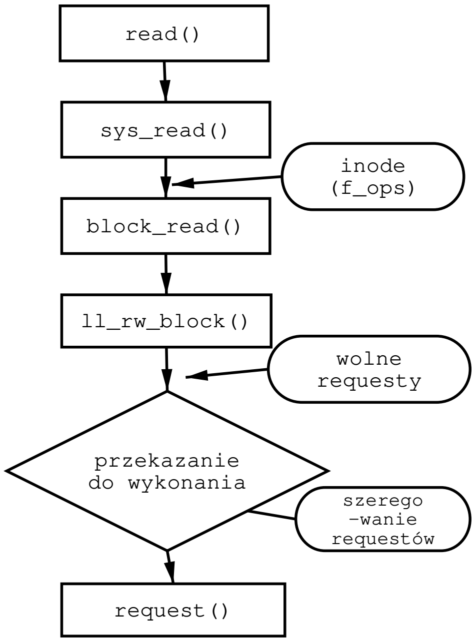
Potem szereguje się żądania za pomocą **algorytmu windowego** (ang. *elevator algorithm*). Jej zadaniem jest takie ułożenie kolejności żądań, aby przyspieszyć szybkość wyszukiwania danych na urządzeniu. Pod uwagę bierze się podrzędny numer urządzenia i numer sektora (adresu danych).

```
#define IN_ORDER(s1,s2) \
((s1)->rq_dev < (s2)->rq_dev || (((s1)->rq_dev == (s2)->rq_dev && \
(s1)->sector < (s2)->sector)))
```

W przypadku dysku twardego oznacza to takie uszeregowanie żądań, aby wykonujac jeden ciąg zleceń głowica nie musiała się cofać. Dokładniejszy opis szeregowania żądań jest w rodziale *Szeregowanie żądań do urządzeń blokowych.*

Sformułowana lista żądań przekazywana jest do wykonania **procedurze strategicznej** (będącej częścią podprogramu obsługi), która realizuje niskopoziomowy dostęp.

W czasie wykonywania zlecenia odczytu proces wywołujący jest usypiany i zawieszany na semaforze. Po wczytaniu danych do jego przestrzeni adresowej, jest budzony.



Rysunek 1.2: Diagram wywołań funkcji `read()` na urządzeniu blokowym (pliku specjalnym) Funkcja `write()` działa analogicznie.

1.3.1 Przekazanie żądań do wykonania

Dla ustalenia uwagi oznaczmy jako `dev` wskaznik na strukturę `blk_dev_struct`, znajdujący się w tablicy `blk_dev[]` dla pewnego urządzenia.

```
struct blk_dev_struct *dev=blk_dev[MAJOR(urzadzenie)];
```

Każde urządzenie ma dwa stany. Jest albo "zatkane" (ang. *plug*) i wtedy
`dev->current_request == &dev->plug`
albo "odetkane" (`current_request` wskazuje na prawdziwą kolejke żądań). Jeśli w kolejce nie ma żądań, urządzenie zostaje "zatkane".

Żądanie otrzymane z `ll_rw_block()` jest poddawane optymalizacji (jak to jest opisane wcześniej) i dołączne funkcją `add_request()` do kolejki `current_request` w strukturze `*dev` (zgodnie z algorytmem windowym). Jeśli jest urządzenie jest "odetkane", to funkcja `add_request()` natychmiast przekazuje je do wykonania. Jeśli jest "zatkane", to jest dołączne do kolejki za sztucznym żądaniem `plug`, po czym urządzenie jest "odtykane".

Nad uruchamianiem `request()` dla kolejnych żądań czuwa niezależna kolejka zadań `tq_disk`.

```
/* The "disk" task queue is used to start the actual requests after a plug */
   DECLARE_TASK_QUEUE(tq_disk);
```

1.3.2 Zakończenie żądania

Podprogramy obsługi urządzeń mogą działać w dwóch trybach, zależnych od sprzętu.

Pierwszy przypadek występuje gdy sprzęt może wysyłać przerwanie (ang. **IRQ** – *interrupt*), np. w przypadku zakończenia odczytu danych. Wtedy po złożeniu zlecenia, podprogram obsługi nie czeka na zakończenie realizacji zlecenia, tylko przechodzi do realizacji następnych zadań.

Drugi pzypadek może być stosowany niezależnie czy sprzęt obsługuje przerwania. Metoda ta nazywana **polling** polega na ciągłym skanowaniu urządzenia i kontroli jego stanu.

Po stwierdzeniu gotowości do przekazania danych, następuje ich zapis w pamięci buforowej i kontrola wraca do programu wywołującego.

1.4 Uwagi końcowe

- Urządzenia nie muszą być koniecznie podłączone do fizycznych urządzeń zewnętrznych. Zamiast tego sterownik (podprogram obsługi) może udawać urządzenie. W ten sposób działają urządzenia takie jak `/dev/ram0` (ramdysk), `/dev/null`.
- Informacje o numerach urządzeń można znaleźć w dokumentacj kernela [5]. Jest tam też informacja w jaki sposób ubiegać się o zarejestrowananie własnego numeru (nadrzędnego lub podrzędnego).
- Informacje o aktualnie zarejestrowanych w systemie urządzeniach i ich numerach można znaleźć w `/proc/devices`.
- Podproramy obsługi urządzeń mogą być na stałe wkompilowane do jądra systemy, lub ładowane jako moduły. Moduły mogą być ładowane ręcznie, lub na prośbę jądra – gdy ktoś próbuje się odwołać do urządzenia do którego nie jest załadowany podprogram obsługi.

1.4.1 Dowodzenie poprawności kodu...

W ten sposób twórcy Linuksa dowodzą poprawność kodu... ;–)

```
/* Uhhuh.. Nasty dead-lock possible here.. */
if (buffer_locked(bh))
  return;
/* Maybe the above fixes it, and maybe it doesn't boot. Life is interesting */
lock_buffer(bh);
```

Źródła kernela – funkcja `make_request()` [2]:

Literatura

[KHG] LDP[3]: *Kernel Hackers' Guide*

[TLK] LDP: *The Linux Kernel* Version 0.8–3

[1] Linux[4] header: *include/linux/blkdev.h*

[2] Linux kernel: *linux/drivers/block/ll_rw_blk.c*

[3] Linux kernel: *linux/fs/devices.c*

[4] Linux kernel: *linux/fs/read_write.c*

[5] Linux kernel documenation: *linux/Documentation/devices.txt*

[LKMPG] LDP: *Linux Kernel Module Programming Guide*

[BACH] Bach J.: *Budowa systemu operacyjnego UNIX*. Warszawa, WNT 1995.

[3]Linux Documentation Project, polskie archiuwm dostępne przez http://www.linux.org.pl/

[4]Źródła jądra systemu w wersji 2.2.17, dostępne z ftp://ftp.kernel.org/