Wykład dwunasty

Urządzenia znakowe i blokowe w Linuksie

Jednym z zastosowań wirtualnego systemu plików opisanego na poprzednim wykładzie jest obsługa urządzeń wejścia – wyjścia. Pojęcie „urządzenie" niekoniecznie musi oznaczać fizyczny układ, może to również być urządzenie wirtualne. W systemach operacyjnych kompatybilnych z Uniksem wyróżnia się trzy podstawowe rodzaje urządzeń – blokowe, znakowe i sieciowe[1]. Zanim przejdziemy do opisu zagadnień związanych ściśle z jądrem Linuksa, przedstawmy typową strukturę sprzętowego urządzenia wejścia – wyjścia biorąc za przykład architekturę x86[2]. Każde urządzenie, które współpracuje z procesorem jest z nim połączone przy pomocy magistrali I/O (*ang. Input – Output*). Ta magistrala jest podzielona na trzy składowe: magistralę danych, adresową i sterowania. Procesory 32-bitowe serii Pentium używają 16 z 32 linii do adresowania urządzeń i 8, 16 lub 32 z 64 linii do przesyłania danych. Szyna wejścia – wyjścia nie jest bezpośrednio połączona z urządzeniem lecz za pośrednictwem struktury sprzętowej, która składa się maksymalnie z trzech komponentów: portów I/O, interfejsu i/lub kontrolera. Porty są specjalnym zestawem adresów, które są przypisane danemu urządzeniu. W komputerach kompatybilnych z IBM PC można wykorzystać do 65536 portów 8 – bitowych, które można łączyć razem w większe jednostki. Procesory Intela i kompatybilne z nimi obsługują porty za pomocą odrębnych rozkazów maszynowych, ale można również odwzorować je w przestrzeni adresowej pamięci operacyjnej[3]. Ten drugi sposób jest chętniej wykorzystywany, ponieważ jest szybszy i umożliwia współpracę z DMA. Porty wejścia – wyjścia są ułożone w zestawy rejestrów umożliwiających komunikację z urządzeniem. Do typowych rejestrów należą: rejestr statusu (inaczej: stanu), sterowania, wejścia i wyjścia. Dosyć często zdarza się, że ten sam rejestr pełni dwie funkcje, np.: jednocześnie jest rejestrem wejściowym i wyjściowym lub rejestrem sterowania i stanu. Interfejsy I/O są układami elektronicznymi, które tłumaczą wartości w portach na polecenia dla urządzenia oraz wykrywają zmiany w stanie urządzenia i uaktualniają odpowiednio rejestr statusu. Dodatkowo są one połączone z kontrolerem przerwań i to one odpowiadają za zgłaszanie przerwania na rzecz urządzenia. Istnieją dwa rodzaje interfejsów: wyspecjalizowane, przeznaczone dla pewnego konkretnego rodzaju urządzenia, jak np.: klawiatura, karta graficzna, dysk, mysz, karta sieciowa i interfejsy ogólnego przeznaczenia, które mogą obsługiwać kilka różnych urządzeń, np.: port równoległy, szeregowy, magistrala USB, interfejsy PCMCIA i SCSI. W przypadku obsługi bardziej skomplikowanych urządzeń potrzebny jest kontroler, który interpretuje wysokopoziomowe polecenia otrzymywane z interfejsu I/O i przekształca je na szereg impulsów elektrycznych (mikrorozkazów) zrozumiałych dla urządzenia lub na podstawie sygnałów otrzymanych z urządzenia I/O modyfikuje zawartość rejestrów z nim związanych.
W systemach kompatybilnych z Uniksem urządzenia są traktowane jak pliki, tzn. są reprezentowane w systemie plików[4] i są obsługiwane przez te same wywołania systemowe co pliki. Pliki reprezentujące urządzenia są nazywane plikami specjalnymi lub po prostu plikami urządzeń. Posiadają one, oprócz nazwy trzy atrybuty: typ – określający, czy dane urządzenie jest blokowe, czy znakowe, główny numer urządzenia (*ang. major device number*) oraz poboczny numer urządzenia (*ang. minor device number*). W jądrach Linuksa serii 2.6 i 3.x i wyższych te dwie ostatnie wartości są zapisywane w jednym 32 – bitowym słowie pamięci, przy czym 12 – bitów przeznaczonych jest na numer główny, a kolejne 20 na numer poboczny. Pisząc swój własny sterownik urządzenia nie należy polegać na tym podziale, gdyż we wcześniejszych wersjach Linuksa wielkość tego słowa była 16 – bitowa, a nie jest wykluczone, że w przyszłych wersjach nie ulegnie ona zmianie, dlatego należy się zawsze posługiwać typem *dev_t* i makrodefinicjami *MAJOR*, *MINOR* i *MKDEV,* które odpowiednio ustalają na podstawie zmiennej typu *dev_t*, wartość numeru głównego, wartość numeru pobocznego oraz łączą te numery w jedną wartość typu *dev_t*. Numer główny identyfikuje sterownik, który obsługuje dane urządzenie lub grupę urządzeń, natomiast numer poboczny służy sterownikowi do ustalenia, które urządzenie z tej grupy w danej chwili wymaga obsługi.
Urządzenia znakowe adresują dane (zazwyczaj) sekwencyjnie i mogą je przesyłać względnie małymi porcjami o różnej wielkości. Są prostsze w obsłudze, więc zostaną opisane jako pierwsze, przed urządzeniami blokowymi.
Każde urządzenie znakowe, które jest obecne w systemie, musi posiadać swój sterownik będący częścią jądra systemu. Może on występować w dwóch postaciach: albo może być wkompilowany na stałe w obraz jądra lub być dołączany w postaci modułu. Pierwszą czynnością, jaką musi taki sterownik wykonać jest uzyskanie jednego lub większej liczby numerów urządzeń. Wykonuje to przy pomocy funkcji:

*int register_chrdev_region(dev_t first, unsigned int count, char *name);*

Parametr *first* oznacza wartość pierwszego numeru z puli jaka ma zostać przydzielona (jakie numery są już zajęte można sprawdzić w dostarczanej z jądrem dokumentacji oraz w pliku */proc/devices* lub w katalogu */sys*)[5]. Argument *count* określa liczbę numerów, a *name* nazwę urządzenia, które zostanie stowarzyszone z tymi numerami. Jeśli operacja przydziału się powiedzie funkcja zwraca wartość „0". Bardziej użyteczną i elastyczną jest inna funkcja pozwalająca na dynamiczne rezerwowanie numerów urządzeń:

*int alloc_chrdev_region(dev_t *dev, unsigned int firstminor, unsigned int count, char *name);*

Parametr *dev* jest parametrem wyjściowym zawierającym (po wywołaniu zakończonym sukcesem) pierwszy numer z puli numerów przydzielonych urządzeniu, drugi parametr określa wartość pierwszego pobocznego numeru i zazwyczaj jest równy zero, pozostałe parametry mają takie samo znaczenie, jak w poprzedniej funkcji. Jeśli numery urządzeń nie będą dłużej potrzebne, należy je zwolnić przy pomocy funkcji:

void unregister_chrdev_region(dev_t first, unsigned int count);

Sterowniki urządzeń znakowych korzystają z trzech struktur związanych z VFS: obiektu pliku, struktury operacji pliku i obiektu i- węzła. Struktury te zostały opisane na poprzednim wykładzie, teraz wyjaśnimy tylko sposób korzystania z nich, jeśli są wykorzystywane do operowania na urządzeniach, a nie na zwykłych plikach. Struktura operacji na pliku powinna oczywiście zawierać wskaźniki do metod służących do obsługi urządzenia. Jej polu *owner* powinna być przypisana wartość makrodefinicji THIS_MODULE, jeśli sterownik jest ładowany jako moduł. Zapobiega to usunięciu modułu, w momencie gdy wykonywana jest jedna z metod. Najczęściej autorzy sterowników urządzeń oprogramowywują cztery metody: *open*(), *release*(), *read*() i *write*(), choć implementowanie ich wszystkich jednocześnie nie jest obowiązkowe. Jeśli zachodzi potrzeba obsługi specyficznych dla danego urządzenia funkcji, które nie mogą być obsłużone przez wymienione wcześniej metody, to implementowana jest jedna lub więcej metoda *ioctl*() lub któraś z jej wersji. Część metod może pozostać niezaimplementowana, ale należy sprawdzić, w jaki sposób jądro obsługuje takie przypadki, gdyż dla każdej metody ta obsługa może być inna. W obiekcie pliku (*struct file*) ważne dla sterownika mogą być: pole *mode*, które zawiera prawa dostępu do urządzenia, pole *f_pos* zawierające wskaźnik bieżącej pozycji pliku, pole *f_flags*, zawierające flagi, pole *f_ops*, będące wskaźnikiem do struktury metod, pole *private_data* i pole *f_dentry* będące wskaźnikiem na obiekt wpisu do katalogu. Pole *mode* może być badane przez metodę *open()*, ale nie jest to wymogiem – jądro samo sprawdza prawa dostępu do urządzenia. Z pola flag, korzysta się głównie po to, by określić, czy operacje dotyczące urządzenia mają być blokujące, czy nieblokujące. Zawartość pola *f_pos* (64 – bity) jest przekazywana przez wartość do metody *llseek()*, która zwraca zmodyfikowaną wartość tego wskaźnika. Inne metody, takie jak *read()* i *write()* obsługują to pole poprzez wskaźnik, który jest im przekazywany jako ostatni argument. Pole *private_data* jest wskaźnikiem bez określonego typu. Można je wykorzystać do przechowywania adresu dynamicznie przydzielonego obszaru pamięci, w którym mogą być przechowywane dane, które powinny odznaczać się trwałością, tzn. nie mogą być niszczone między kolejnymi wywołaniami metod. Przydzielenie pamięci na te dane powinno być przeprowadzane w metodzie *open()* przy jej pierwszym wywołaniu, a zwolnienie w metodzie *release()* po ostatnim wywołaniu *close()*. Programiści piszący sterowniki nie muszą się martwić o inicjację pola *f_dentry*. Jest ono używane, aby uzyskać wskaźnik na obiekt i-węzła odpowiadającego obsługiwanemu urządzeniu. W obiekcie i-węzła (*struct i-node*) możemy użyć pola *i_rdev* zawierającego numer urządzenia. Typ tego pola zmieniał się kilkukrotnie podczas rozwoju jądra, więc obecnie, aby odczytać z obiektu i-węzła główny i poboczny numer urządzenia należy użyć następujących funkcji:

*unsigned int iminor(struct inode *inode);*
*unsigned int imajor(struct inode *inode);*

Innym polem, które należy zainicjować w tym obiekcie jest wskaźnik *i_cdev*, wskazujący na strukturę jądra, która reprezentuje urządzenie znakowe. Taką strukturę można stworzyć i zainicjować dynamicznie, za pomocą funkcji *cdev_alloc()*, lub statycznie i zainicjować za pomocą:

1 Linux wyróżnia również dodatkowe kategorie i podkategorie urządzeń, ale nie są one widoczne dla przestrzeni użytkownika, tak jak te trzy podstawowe. Występują one wyłącznie wewnątrz jądra systemu.
2 Nie jest to przykład idealny, ale jeden z najbardziej popularnych.
3 Inne procesory, jak np.: procesory Motoroli obsługują urządzenia wyłącznie odwzorowując ich porty w pamięci operacyjnej. To pozwala na ujednolicenie obsługi urządzeń peryferyjnych i pamięci.
4 Za wyjątkiem interfejsów sieciowych.
5 Jeśli sterownik ma być włączony na stałe do kodu jądra, to numery główny i poboczny urządzenia nie mogą być dobrane na zasadzie „pierwszy wolny". Muszą one być zarejestrowane przez organizację *Linux assigned name and numbers authority* (www.lanana.org).

*void cdev_init(struct cdev *cdev, struct file_operations *fops);*

W obu przypadkach trzeba zainicjować pole *owner* takiej struktury, które powinno mieć wartość makra THIS_MODULE. Inicjacja za pomocą *cdev_alloc()* wymaga również bezpośredniej inicjacji pola *ops* struktury *cdev*, które powinno wskazywać na strukturę metod obiektu pliku. Po stworzeniu *cdev* należy dodać ją do innych tego typu struktur przechowywanych przez jądro za pomocą funkcji:

*int cdev_add(struct cdev *dev, dev_t num, unsigned int count);*

Komplementarną do niej jest funkcja *void cdev_del(struct cdev *dev).* Każde z urządzeń obsługiwanych przez sterownik musi być opisywane wewnętrznie przez taką strukturę. W starszych wersjach jądra rejestrowanie urządzenia nie wymagało tworzenia struktury *cdev* i odbywało się poprzez funkcję *register_chrdev()*. Usunięcie urządzenia odbywało się z kolei za pośrednictwem *unregister_chrdev().* Ten sposób nie będzie tu dokładniej opisywany. W późniejszych wydaniach jądra serii 2.6 pojawiła się konieczność powiadamiania podsystemu tworzącego tzw. model urządzeń o dodaniu nowego sterownika. To powiadomienie obowiązuje także w najnowszych jądrach i jest realizowane podczas inicjacji sterownika za pomocą makrodefinicji i funkcji. Makrodefinicją jest:

class_create(owner, name);

Tworzy ona strukturę opisującą klasę obsługiwanego przez sterownik urządzenia. Przez pierwszy jej parametr przekazywana jest wartość makra THIS_MODULE, a przez drugi nazwa klasy urządzenia. Funkcją jest:

*struct device *device_create(struct class *class, struct device *parent, dev_t devt, void *drvdata, const char fmt, …);*

Tworzy ona strukturę urządzenia i rejestruje je w systemie plików *sysfs*. Dzięki temu informacje o urządzeniu będą dostępne dla przestrzeni użytkownika. Pierwszy parametr tej funkcji to wskaźnik do struktury klasy, drugi to wskaźnik do nadrzędnej struktury urządzenia (argumentem może być NULL, jeśli ona nie istnieje), trzeci to numer urządzenia, czwarty to wskaźnik do danych dodawanych do struktury i wykorzystywanych przez funkcje realizujące wywołania zwrotne (argument też może być NULL), przez piąty parametr przekazywana jest nazwa urządzania, która może być formatowana tak jak argumenty funkcji *printf()*.
Przed usunięciem sterownika z jądra należy wywołać dwie kolejne funkcje. Pierwsza z nich to:

*void device_destroy(struct class *cls, dev_t devt);*

Usuwa ona strukturę opisującą urządzenie. Jako pierwszy argument przyjmuje ona wskaźnik na zwalnianą strukturę, a jako drugi numer urządzenia związanego z tą strukturą. Druga funkcja to:

*void class_destroy(struct class *cls);*

Jako argument wywołania przyjmuje ona wskaźnik do zwalnianej struktury opisującej klasę urządzenia.
Metody obsługujące urządzenia powinny działać według określonego protokołu. Metoda *open()* powinna wykonywać następujące czynności:

- Zidentyfikować urządzenie, które jest obsługiwane, czyli określić jego numer poboczny.
- Sprawdzić, czy nie wystąpiły specyficzne dla urządzenia błędy.
- Zainicjować urządzenie, jeśli jest to pierwsze jego otwarcie.
- Zaktualizować wskaźnik pozycji pliku, jeśli zachodzi taka konieczność.
- Przydzielić i wypełnić pamięć na dane prywatne, jeśli jest taka potrzeba.

Metoda *release()* powinna działać według następującego scenariusza:

- Zwolnić pamięć na dane prywatne, jeśli była ona przydzielana w metodzie *open*().
- Wyłączyć (ang. *shut down*) urządzenie przy ostatnim wywołaniu *close()*.

Również metody *read*() i *write*() muszą działać według pewnego „standardu". Metoda *read*() powinna zwracać ilość faktycznie przeczytanych informacji z urządzenia lub błędy, takie jak -EINTR (otrzymano sygnał), -EFAULT (błędny adres) lub -EIO (ogólny błąd wejścia-wyjścia). Podobnie powinna zachowywać się metoda *write*().

Z podobnych struktur i operacji korzystają sterowniki urządzeń blokowych, jednak ich obsługa jest bardziej skomplikowana, więc część szczegółów zostanie przedstawiona dopiero na następnym wykładzie. Urządzenia blokowe umożliwiają swobodny dostęp do danych i przesyłają je porcjami nazywanymi blokami (stąd nazwa urządzeń), których wielkość jest parzystą wielokrotnością rozmiaru sektora[6].

Pierwszą czynnością wykonywaną przez sterownik urządzenia blokowego jest pozyskanie numeru głównego, za pomocą wywołania funkcji *register_blkdev()* zadeklarowanej w pliku nagłówkowym <linux/fs.h>:

*int register_blkdev(unsigned int major, const char *name);*

Jeśli w wywołaniu wartość parametru *major* będzie równa zero, to jądro automatycznie przydzieli sterownikowi pierwszy wolny numer główny. Można go zwolnić wywołując funkcję *unregister_blkdev()*, o prototypie:

*void unregister_blkdev(unsigned int major, const char *name);*

Urządzenia blokowe mają własną strukturę metod, odpowiadającą strukturze metod obiektu pliku, zdefiniowaną w pliku nagłówkowym <linux/blkdev.h> i nazwaną *struct block_device_operations.* Zawiera ona pole *owner* oraz między innymi wskaźniki na funkcje *open()*, *release()*, *ioctl(), compat_ioctl*(), *check_events()* i *revalidate_disk()*. Metoda *check_events()* jest wywoływana między innymi wtedy, gdy zmieni się nośnik w urządzeniu wymiennym, a *revalidate_disk()* w odpowiedzi na wywołanie tej wcześniejszej.

Rolę struktury *cdev* dla urządzeń blokowych pełni struktura *struct gendisk* zdefiniowana w pliku nagłówkowym <linux/genhd.h>. Zawiera ona pola *major* (numer główny urządzenia), *first_minor* (pierwszy numer poboczny), *minors* (liczba numerów pobocznych), *disk_name* (nazwa dysku – maksymalnie 32 znaki), *fops* (wskaźnik na strukturę *struct block_device_operations*), *queue* (wskaźnik na kolejkę żądań), *flags* (flagi – rzadko używane, najczęściej w przypadku urządzeń wymiennych i napędów dysków optycznych), oraz *private_data* (dane prywatne sterownika). Struktura *struct gendisk* przechowuje również wielkość (pojemność) nośnika urządzenia, która wyrażona jest jako wielokrotność rozmiaru pojedynczego sektora i jest ustawiana przy pomocy wywołania funkcji *set_capacity().* Pamięć na tę strukturę jest

6 Sektor ma w jądrze Linuksa wielkość 512 bajtów.

przydzielana za pomocą funkcji *alloc_disk(),* a zwalniana, po wyzerowaniu licznika odwołań, za pomocą *put_disk()*:

*struct gendisk *alloc_disk(int minors);*
*void put_disk(struct gendisk *disk);*

Każda taka struktura jest związana z pojedynczym urządzeniem obsługiwanym przez sterownik, np. jedną partycją dysku twardego. Aby takie urządzenie stało się dostępne dla systemu należy przekazać tę strukturę do wywołania funkcji *add_disk*():

*void add_disk(struct gendisk *gd);*

Wyrejestrować strukturę *gendisk* należy za pomocą funkcji *del_gendisk()*:

*void del_gendisk(struct gendisk *gd);*

Najważniejszym polem tej struktury jest pole *queue* będące wskaźnikiem na kolejkę żądań. Pamięć na tę kolejkę jest przydzielana za pomocą funkcji *blk_init_queue()*:

*request_queue_t *blk_init_queue(request_fn_proc *request, spinlock_t *lock);*

Pierwszym argumentem wywołania tej funkcji jest wskaźnik na funkcję *request()*, która odpowiedzialna jest za realizację pojedynczego żądania. Przez drugi parametr przekazywany jest adres lub wskaźnik na rygiel pętlowy, synchronizujący dostęp do tej kolejki. Jeśli sterownik obsługuje urządzenia o rzeczywistym dostępie swobodnym, takie jak np. pamięć flash, to kolejka żądań jest zbędna. W takim przypadku pole *queue* struktury *struct gendisk* jest inicjowane za pomocą wywołania funkcji *blk_alloc_queue()*:

*request_queue_t *blk_alloc_queue(int flags);*

Przy takiej inicjacji pola *queue* sterownik powinien dostarczyć funkcji *make_request()*, która jest odpowiednikiem *request*(). Ta funkcja jest rejestrowana przez sterownik za pomocą wywołania funkcji *blk_queue_make_request()*:

*void blk_queue_make_request(request_queue_t *queue, make_request_fn *func);*

Pamięć przydzielona na kolejkę żądań jest zwalniana przy pomocy wywołania funkcji *blk_cleanup_queue()*:

*void blk_cleanup_queue(struct request_queue *q);*

Szczegóły budowy struktury opisującej pojedyncze żądanie oraz inne zagadnienia związane z obsługą urządzeń blokowych zostaną opisane w następnym wykładzie.