Операційні системи

Лекція 11

Керування введенням-виведенням в ОС Linux, UNIX, Windows

План лекції

- Керування введенням-виведенням в ОС UNIX і Linux
- Керування введенням-виведенням в ОС Windows

Введення-виведення в UNIX і Linux

- Введення-виведення здійснюється через файлову систему
- Кожному драйверу пристрою відповідає один або кілька спеціальних файлів пристроїв
- Файли пристроїв традиційно розміщені у каталозі /dev
- Кожний файл пристрою характеризується чотирма параметрами
  - Ім’я файлу
    - застосовується для доступу до пристрою з прикладних програм
  - Тип пристрою (символьний чи блоковий)
    - фактично вказує на таблицю – одна таблиця для символьних пристроїв, друга – для блокових
  - Major number – номер драйвера у таблиці
    - ціле число, як правило, 1 байт (може 2)
  - Minor number – номер пристрою
    - це число передають драйверу, драйвер може працювати з кількома пристроями, у тому числі різними

Робота з файлами пристроїв

- Файли пристроїв, які і звичайні файли, можна створювати й видаляти
  - На драйвер це ніяк не впливає
  - Команда створення файлу пристрою:

  mknod /dev/mydevice c 150 1
- Звернення до файлу:
  - Спочатку – звернення до файлу за іменем
  - Система звертається до індексного дескриптора
  - Перевіряє права доступу
  - З дескриптора визначає тип пристрою і номер драйвера
  - Звертається до драйвера і передає йому номер пристрою
  - Виконує задану файлову операцію

Операції роботи з пристроями

- Драйвер зобов'язаний підтримувати стандартні файлові операції
  - open(), read(), write(), lseek()
  - Прикладні програми виконують такі операції так, якби вони працювали із звичайними файлами
- Для деяких пристроїв крім стандартних операцій, можливо, визначені й деякі інші операції
  - Для реалізації нестандартних операцій передбачено універсальний виклик ioctl()

ioctl(int d, int request, char *agrp);

d – файловий дескриптор

request – код операції

* agrp – покажчик на довільну пам'ять

  - Наприклад, для приводу оптичного диску так можна визначити операцію EJECT

Структура драйвера

- Код ініціалізації (одна функція init())
  - Код виконується під час завантаження ядра системи або під час завантаження модуля драйвера
  - Цей код реалізує реєстрацію драйвера у системі (вибір номера, реєстрацію оброблювачів переривань)
  - Цей код не може створювати спеціальні файли!
- Реалізацію файлових операцій і ioctl()
  - Для символьних пристроїв: open(), close(), read(), write(), lseek(), select(), mmap()
  - Для блокових пристроїв є особливість: реакцію на операції зчитування і записування викликають не прямо, а після проходження керування через буферний кеш; для цього їх реалізують в одній функції
    - UNIX – strategy()
    - Linux – request()
- Обробники переривань
  - Обробників переривань може і не бути: драйвер може не застосовувати переривання, а виконувати опитування пристроїв
  - Обробники переривань мають верхню і нижню половини

Введення-виведення з розподілом і об’єднанням

- За одну операцію здійснюється зчитування в, або записування з кількох не пов’язаних ділянок пам’яті
  - Введення – scatter – readv()
  - Виведення – gather – writev()

```
ssize_t readv(int fdl, const struct iovec *iov, int count);
```

fdl – дескриптор відкритого файлу;

iov – масив структур, які задають набір ділянок пам’яті для введення і виведення;

count – кількість структур у масиві iov

- Кожний елемент масиву містить два поля:

iov_base – задає базову адресу ділянки пам’яті

iov_len – задає довжину ділянки пам’яті

- Виклики повертають загальну кількість байтів (зчитаних або записаних)

Введення-виведення з повідомленням

- Введення-виведення з повідомленням про стан дескрипторів
  1. Готують структуру даних fdarr з описом усіх дескрипторів, стан яких треба відстежувати
  2. Передають fdarr у системний виклик повідомлення (у POSIX – виклики select() і poll())
  3. Після виходу з виклику fdarr містить інформацію про стан усіх дескрипторів
  4. У циклі обходять усі елементи fdarr і для кожного з них визначають готовність дескриптора
- Введення-виведення з повідомленням про події (у FreeBSD – kqueue, Linux 2.6 – epoll)
  1. Системний виклик (epoll_create()) створює структуру даних у ядрі (прослуховувальний об'єкт)
  2. Для прослуховувального об'єкта формують набір дескрипторів, для кожного з них вказують події, які цікавлять (epoll_ctl())
  3. Виклик повідомлення (epoll_wait()) повертає інформацію лише про ті дескриптори, які змінили свій стан з моменту останнього виклику

Асинхронне введення-виведення

- Стандарт POSIX передбачає такі виклики для асинхронного введення-виведення:
  - aio_read() – зчитування
  - aio_write() – записування
  - aio_suspend() – очікування
  - aio_cancel() – переривання
  - aio_return() – отримання результату
  - aio_error() – отримання статусу операції
- Усі виклики, крім aio_suspend(), приймають параметром покажчик на структуру aiocb з полями:
  - aio_fildes – дескриптор файлу, для якого здійснюють введення-виведення
  - aio_buf – покажчик на буфер, у який зчитає дані aio_read() і з якого запише дані aio_write()
  - aio_nbytes() – розмір буфера
- Формат виклику aio_suspend():

```
int aio_suspend(struct aiocb *list[], int cnt, struct timespec *tout);
```

Послідовність виконання операції введення-виведення

1. Процес користувача готує буфер у своєму адресному просторі
2. Процес користувача виконує системний виклик read() для спеціального файлу пристрою, і передає у виклик адресу буфера
3. Відбувається перехід у режим ядра
   - Переключення контексту не здійснюють
4. На підставі інформації з індексного дескриптора спеціального файлу визначають необхідний драйвер і викликають функцію, яка зареєстрована як реалізація файлової операції read() для відповідного драйвера
5. Функція:
   - Виконує необхідні підготовчі операції
     - Наприклад, розміщує буфер у пам'яті ядра
   - Відсилає контролеру пристрою запит на виконання операції зчитування
   - Переходить у режим очікування
     - Для цього як правило використовують функцію sleep_on()
     - При цьому процес, що викликав операцію, призупиняється

Послідовність виконання операції введення-виведення (2)

6. Контролер здійснює зчитування
   - При цьому він, можливо, використовує буфер, наданий йому функцією драйвера
7. Після завершення зчитування контролер викликає переривання
8. Апаратне забезпечення активізує верхню половину оброблювача переривання
9. Код верхньої половини ставить нижню половину у чергу на виконання
10. Код нижньої половини:
    - Заповнює буфер, якщо він не був заповнений контролером
    - Виконує інші необхідні дії для завершення операції введення
    - Поновлює виконання процесу, що очікує
11. Керування після поновлення виконання процесу повертається у реалізацію функції read() для драйвера – у код, що слідує за викликом sleep_on()
    - Цей код копіює дані з буфера ядра у буфер режиму користувача
12. Керування повертають у процес користувача

Введення-виведення у Windows

- Базовий компонент – менеджер введення-виведення (I/O Manager)
- Операції – асинхронні
  - Синхронні операції реалізують як асинхронні операції + очікування
- Операції введення-виведення відображаються у вигляді структур даних – пакетів запитів введення-виведення (I/O Request Packet)
  - Менеджер введення-виведення створює пакет і передає покажчик на нього потрібному драйверу
  - Драйвер
    - отримує пакет,
    - виконує потрібну операцію,
    - повертає пакет як індикатор виконання операції або для передачі його іншому драйверу
  - Після завершення операції введення-виведення, менеджер вивільняє пам’ять, яку займав пакет

Асинхронне введення-виведення

- На відміну від стандарту POSIX, у Win32 API для асинхронного введення-виведення можна застосовувати стандартні функції файлового в/в – ReadFile() і WriteFile()
  - Для цього у функції передається покажчик на спеціальну структуру OVERLAPPED
  - Файл повинен бути відкритим з дозволом асинхронних операцій FILE_FLAG_OVERLAPPED
- Для очікування завершення введення-виведення використовують універсальну функцію очікування
  - Наприклад, WaitForSingleObject()
- Для отримання результату необхідно використати функцію GetOverlappedResult()
- Для переривання введення-виведення використовують функцію CancelIo()

Порти завершення введення-виведення (*I/O completion port*)

- Спочатку створюють новий об’єкт порту, потім додають у нього файлові дескриптори
  - Застосовують виклик CreateCompletionPort()

  CreateCompletionPort (fdarr[key], ph, key, R_{max});
  - Один з параметрів, який передають у виклик – це максимальна кількість потоків, що можуть виконуватись у системі R_{max}
    - Оптимально – дорівнює кількості процесорних ядер
- Після цього формують *пул робочих потоків* (*thread pool*)
  - Кількість має перевищувати R_{max}
  - Кожний з потоків повинен виконувати один і той самий код

```
For ( : : ) {
    // очікування на об’єкті порту, заданому дескриптором ph
    GetQueuedCompletionStatus (ph, nbytes, &key, &ov, INFINITE);
    // тут потік є активним
    ReadFile (fdarr[key], request, … );      // прочитати запит
    process_request (request);          // обслужити клієнта
}
```

Категорії драйверів

- Типи драйверів згідно *Windows Driver Model*, WDM (для ядра версії 5)
  - Драйвери шини
    - Керують логічною або фізичною шиною, відповідають за виявлення пристроїв
  - Функціональні драйвери
    - Керують пристроєм конкретного типу
  - Драйвери-фільтри
    - Доповнюють або змінюють поведінку інших драйверів
- Категорії драйверів ядра (крім WDM-драйверів)
  - Файлових систем
    - Перетворюють запити введення-виведення, що використовують файли, у запити до низькорівневих драйверів пристроїв
  - Відображення (*Display Drivers*)
    - Перетворюють незалежні від пристрою запити підсистеми GDI в команди графічного адаптера або команди записування у пам’ять
  - Успадковані
    - Розроблені для Windows NT 4

Категорії драйверів (2)

- Категорії драйверів режиму користувача
  - Наприклад, драйвери принтера
    - Перетворюють незалежні від пристрою запити підсистеми GDI в команди конкретного принтера і передають їх WDM драйверу
- Категорії драйверів в залежності від рівня підтримки конкретного пристрою
  - Клас-драйвери
    - Реалізують інтерфейс оброблення запитів введення-виведення, специфічних для конкретного класу пристроїв (диски, CD-ROM)
  - Порт-драйвери
    - Реалізують інтерфейс оброблення запитів введення-виведення, специфічних для певного класу портів (SCSI)
  - Мініпорт-драйвери
    - Керують реальним конкретним пристроєм

Структура драйвера пристрою

- Процедура ініціалізації *Driver Entry*
  - Її виконує менеджер введення-виведення під час завантаження драйвера у систему
  - Здійснює глобальну ініціалізацію структур даних драйвера
- Процедура додавання пристрою *Add-device routine*
  - Для реалізації технології Plug and Play
  - Менеджер Plug and Play викликає цю процедуру, якщо знаходить пристрій, за який відповідає драйвер
- Набір процедур диспетчеризації *Dispatch Routines*
  - Аналогічні функціям файлових операцій в UNIX і Linux
- Процедура оброблення переривання *Interrupt Service Routine*, ISR
  - Аналогічна верхній половині оброблювача переривання
  - Основне завдання – запланувати для виконання нижню половину оброблювача
- Процедура відкладеного оброблення переривання DPC *Routine*
  - Аналогічна нижній половині оброблювача переривання

Послідовність виконання операції введення-виведення

1. Запит в/в перехоплює динамічна бібліотека
   - Наприклад, Win32 перехоплює виклик WriteFile()
2. Динамічна бібліотека викликає внутрішню функцію NTWriteFile(), яка звертається до менеджера в/в
3. Менеджер в/в створює пакет IRP, розміщає його у пам'яті і відправляє посилання на нього драйверу викликом функції IoCallDriver()
4. Драйвер бере дані з IRP і передає їх контролеру пристрою, після чого дає команду розпочати операцію в/в
5. Для синхронного в/в драйвер викликає функцію очікування
   - Поточний потік призупиняють
6. Коли операція завершується, контролер викликає переривання
7. Драйвер викликає функцію IoCompleteRequest() для повідомлення менеджеру в/в про завершення дій з пакетом, після чого виконують код завершення операції

Завершення запиту введення-виведення

- Звичайно завершення зводиться до копіювання даних в адресний простір процесу користувача
- У разі синхронного введення-виведення адресний простір належить до процесу, що робив виклик
  - Дані можуть бути записані у нього безпосередньо
- Якщо запит був асинхронним, активний потік ймовірно належить до іншого процесу
  - Необхідно дочекатися, поки адресний простір потрібного процесу не стане доступний
  - Для цього менеджер в/в планує до виконання спеціальну *APC-процедуру* (від *Asynchronous Procedure Call*)
  - APC-процедура виконується лише у контексті конкретного потоку
    - Отримує керування
    - Копіює потрібні дані в адресний простір процесу
    - Вивільняє пам'ять з-під пакета IRP
    - Переводить файловий дескриптор (або порт завершення введення-виведення) у сигналізований стан