ЛЕКЦІЯ 9. КЕРУВАННЯ ПРИСТРОЯМИ ВВЕДЕННЯ-ВИВЕДЕННЯ

План

15.1. Завдання підсистеми введення-виведення
15.2. Організація підсистеми введення-виведення
15.3. Способи виконання операцій введення-виведення
15.4. Підсистема введення-виведення ядра
15.5. Введення-виведення у режимі користувача
15.6. Таймери і системний час
15.7. Керування введенням-виведенням: Windows XP

15.1. Завдання підсистеми введення-виведення

Основним завданням підсистеми введення-виведення є реалізація доступу до зовнішніх пристроїв із прикладних програм, яка повинна забезпечити:

- *ефективність* (можливість використання ОС всіх засобів оптимізації, які надає апаратне забезпечення), спільне використання і захист зовнішніх пристроїв за умов багатозадачності;
- *універсальність для прикладних програм* (ОС має приховувати від прикладних програм відмінності в інтерфейсі апаратного забезпечення, надаючи стандартний інтерфейс доступу до різних пристроїв), при цьому потрібно завжди залишати можливість прямого доступу до пристрою, оминаючи стандартний інтерфейс;
- *універсальність для розробників* системного програмного забезпечення (драйверів пристроїв), щоб під час розробки драйвера для нового пристрою можна було скористатися наявними напрацюваннями і легко забезпечити інтеграцію цього драйвера у підсистему введення-виведення.

15.1.1. Забезпечення ефективності доступу до пристроїв

Забезпечення ефективності вимагає розв'язання кількох важливих задач.

1. Передусім — це коректна взаємодія процесора із контролерами пристроїв. Відомо, що кожен зовнішній пристрій має контролер, який забезпечує керування пристроєм на найнижчому рівні і є фактично спеціалізованим процесором. Після отримання команди від ОС контролер забезпечує її виконання, при цьому пристрій якийсь час не взаємодіє із процесором комп'ютера, тому той може виконувати інші задачі. Виконавши команду, контролер повідомляє системі про завершення операції введення-виведення, генеруючи відповідну подію. Операційній системі в цьому разі потрібно спланувати процесорний час таким чином, щоб драйвери пристроїв могли ефективно реагувати на події контролера та було забезпечене виконання коду процесів користувача.

2. Керування пам'яттю під час введення-виведення. Оперативна пам'ять є швидшим ресурсом, ніж зовнішні пристрої, тому ОС може

підвищувати ефективність доступу до пристроїв проміжним зберіганням даних у пам'яті.

15.1.2. Забезпечення спільного використання зовнішніх пристроїв

Під час спільного використання зовнішніх пристроїв мають виконуватися певні умови.

1. ОС повинна мати можливість забезпечувати одночасний доступ кількох процесів до зовнішнього пристрою і розв'язувати можливі конфлікти (тобто необхідна підтримка синхронізації доступу до пристроїв). Деякі пристрої (наприклад, модем або сканер) можна використати тільки одним процесом у конкретний момент часу, тоді як жорсткий диск завжди використовують спільно;

2. Слід забезпечити захист пристроїв від несанкціонованого доступу. Такий захист можна організувати або для пристрою як цілого (наприклад, можна відкрити модем для доступу тільки певній групі користувачів), або для деякої підмножини даних пристрою (наприклад, різні файли на жорсткому диску можуть мати різні права доступу).

3. У разі спільного використання пристрою треба розподілити операції введення-виведення різних процесів, для того щоб уникнути «накладок» даних одних процесів на дані інших (наприклад, під час спільного використання принтера важливо відрізняти одні задачі від інших і не переходити до друкування результатів наступної задачі до того, як завершилося виведення попередньої).

15.1.3. Універсальність інтерфейсу прикладного програмування

Оскільки пристрої введення-виведення доволі різноманітні, дуже важливо уніфікувати доступ до них із прикладних програм. Для реалізації цієї ідеї підсистема введення-виведення має використовувати набір базових абстракцій, під час застосування яких можна надати доступ до різних зовнішніх пристроїв узагальненим способом. Для більшості сучасних ОС такою абстракцією є абстракція файла, що відображається як набір байтів, з яким можна працювати за допомогою спеціальних операцій файлового введення-виведення. Файл, що відповідає пристрою (його називають файлом пристрою), не відповідає набору даних на диску, а є засобом організації універсального доступу різних компонентів ОС і прикладних програм до деякого пристрою введення-виведення.

15.1.4. Універсальність інтерфейсу драйверів пристроїв

Драйвер пристрою - це програмний модуль, що керує взаємодією ОС із конкретним зовнішнім пристроєм.

Драйвери практично завжди виконують у режимі ядра. Вони можуть бути завантажені у пам'ять і вивантажені з неї під час виконання.

Набір драйверів, доступних для операційної системи, визначає набір апаратного забезпечення, із яким ця система може працювати. Якщо драйвер для потрібного пристрою відсутній, користувачу залишається

лише чекати, поки він не буде створений (альтернативою є самостійна розробка). Часто в цій ситуації користувач змушений перейти до використання іншої ОС. Для того щоб знизити ймовірність такого небажаного розвитку подій, підсистемі введення-виведення потрібно забезпечити зручний універсальний і добре документований інтерфейс між драйверами й іншими компонентами операційної системи. Наявність такого інтерфейсу і зручного середовища розробки драйверів дає змогу спростити їхнє створення.

Зручність середовища розробки драйверів визначає набір функцій і шаблонів, наданих програмістові. Часто набір засобів, призначений для розробки драйверів під конкретну операційну систему, постачають розробники цієї ОС у вигляді окремого продукту, який називають *DDK* (Driver Development Kit). У нього входять заголовні файли, бібліотеки, можливо, спеціальні версії компіляторів і налагоджувачів, а також документація.

15.2. Організація підсистеми введення-виведення

Розглянемо особливості організації такої підсистеми.

Структура підсистеми розділена на рівні, що дає змогу забезпечити, з одного боку, підтримку всього спектра зовнішніх пристроїв (на нижньому рівні, де розташовані апаратні драйвери), а з іншого — уніфікацію доступу до різних пристроїв (на верхніх рівнях). Зазначимо, що досягти повної ідентичності доступу до всіх можливих пристроїв майже неможливо, тому, крім горизонтальних рівнів, виділяються вертикальні групи пристроїв, взаємодія з якими ґрунтується на загальних принципах.

Деякі функції цієї підсистеми не можуть бути реалізовані у драйверах. Це, насамперед, засоби, що забезпечують координацію роботи всіх модулів підтримки введення-виведення у системі, які об'єднуються у *менеджер введення-виведення.* Він, з одного боку, забезпечує доступ прикладних програм до відповідних засобів введення-виведення (наприклад, через інтерфейс файлової системи, реалізований як набір системних викликів), з іншого — середовище функціонування драйверів пристроїв, реалізує загальні служби, такі як буферизація, обмін даними між драйверами тощо.

4 Драйвери пристроїв теж розглядають на кількох рівнях. Поряд із низькорівневими апаратними драйверами для керування конкретними пристроями є драйвери вищого рівня. Вони взаємодіють із апаратними драйверами і на підставі отриманих від них даних виконують складніші завдання.

15.2.1. Символьні, блокові та мережні драйвери пристроїв

Ще одна важлива класифікація драйверів уперше з'явилася в UNIX і відтоді її широко використовують, оскільки вона добре відображає специфіку різних пристроїв. Пристрої та драйвери відповідно до цієї класифікації розділяються на три категорії: *блокові* або *блок-орієнтовані*

(block-oriented, block), *символьні* або *байт-орієнтовані* (character-oriented, character) і *мережні* (network).

Для блокових пристроїв дані зберігають блоками однакового розміру, при цьому кожен блок має свою адресу, і за допомогою відповідного драйвера до нього можна отримати прямий доступ. Основним блоковим пристроєм є диск.

Символьні пристрої розглядають дані як потік байтів, при цьому окремий байт адресований бути не може. Прикладами таких пристроїв є модем, клавіатура, миша, принтер тощо. Базовими системними викликами для символьних пристроїв є виклики читання і записування одного байта.

Окремою категорією є мережні пристрої, які надаються прикладним програмам у вигляді *мережних інтерфейсів* зі своїм набором допустимих операцій, які відображають специфіку мережного введення-виведення (наприклад, ненадійність зв'язку).

15.3. Способи виконання операцій введення-виведення

Зовнішній пристрій взаємодіє із комп'ютерною системою через точку зв'язку, яку називають *портом* (port). Якщо кілька пристроїв з'єднані між собою і можуть обмінюватися повідомленнями відповідно до заздалегідь визначеного протоколу, то кажуть, що вони використовують *шину* (bus).

Як відомо, пристрої зв'язуються із комп'ютером через контролери. Є два базові способи зв'язку із контролером: через *порт введення-виведення* (I/O port) і *відображувану пам'ять* (memory-mapped I/O). У першому випадку дані пересилають за допомогою спеціальних інструкцій, у другому — робота із певною ділянкою пам'яті спричиняє взаємодію із контролером.

Деякі пристрої застосовують обидві технології відразу. Наприклад, графічний контролер використовує набір портів для організації керування і регіон відображуваної пам'яті для зберігання вмісту екрана.

Спілкування із контролером через порт звичайно зводиться до

15.3.1. Опитування пристроїв

Припустимо, що контролер може повідомити, що він зайнятий, увімкнувши біт busy регістра статусу. Застосування повідомляє про команду записування вмиканням біта write командного регістра, а про те, що є команда — за допомогою біта c ready того самого регістра.

На першому етапі застосування займається *опитуванням пристрою* (polling), фактично воно перебуває в циклі активного очікування. Одноразове опитування здійснюється дуже швидко, для нього досить трьох інструкцій процесора — читання регістра, виділення біта статусу і переходу за умовою, пов'язаною з цим бітом. Проблеми з'являються, коли опитування потрібно повторювати багаторазово, у цьому разі буде зайвим завантаження процесора. Для деяких пристроїв прийнятним є опитування через фіксований інтервал часу.

У більшості інших випадків потрібно організовувати введення-виведення, кероване перериваннями.

15.3.2. Введення-виведення, кероване перериваннями

Базовий механізм переривань дає змогу процесору відповідати на асинхронні події. Така подія може бути згенерована контролером після закінчення введення-виведення або у разі помилки, після чого процесор зберігає стан і переходить до виконання оброблювача переривання, встановленого ОС. Цим знімають необхідність опитування пристрою — система може продовжувати звичайне виконання після початку операції введення-виведення.

15.4. Підсистема введення-виведення ядра

15.4.1. Планування операцій введення-виведення

Планування введення-виведення звичайно реалізоване як середньотермінове планування. Як відомо, з кожним пристроєм пов'язують чергу очікування, під час виконання блокувального виклику потік поміщають у чергу для відповідного пристрою, з якої його звичайно вивільняє оброблювач переривання, різним пристроям можуть присвоювати різні пріоритети.

Ще одним прикладом планування введення-виведення є дискове планування, розглянуте в розділі 12 [1].

15.4.2. Буферизація

Найважливішою технологією підвищення ефективності обміну даними між пристроєм і застосуванням або між двома пристроями є буферизація. Для неї виділяють спеціальну ділянку пам'яті, яка зберігає дані під час цього обміну і є буфером. Залежно від того, скільки буферів використовують і де вони перебувають, розрізняють кілька підходів до організації буферизації [1].

Способи реалізації буферизації

Буфер, у який копіюються дані від пристрою, можна організувати в адресному просторі процесу користувача. Хоча такий підхід і дає виграш у продуктивності, його не можна вважати прийнятним, оскільки сторінка із таким буфером може у будь-який момент бути заміщена у пам'яті та скинута на диск.

Першим підходом, який реально використовують на практиці, є *одинарна буферизація в ядрі*. У цьому разі у ядрі створюють буфер, куди копіюють дані в міру їхнього надходження від пристрою. Коли цей буфер заповнюється, весь його вміст за одну операцію копіюють у буфер, що перебуває у просторі користувача. Аналогічно під час виведення даних їх спочатку копіюють у буфер ядра, після чого вже ядро відповідатиме за їхнє виведення на пристрій. Це дає змогу реалізувати семантику копіювання, оскільки після копіювання даних у буфер ядра інформація із буфера користувача у підсистему введення-виведення гарантовано більше не

потрапить — процес може продовжувати свою роботу і використовувати цей буфер для своїх потреб.

Узагальнення цієї схеми на *n* буферів називають *циклічною буферизацією,* її можна використовувати тоді, коли час збереження буфера перевищує час його заповнення.

Буферизація і кешування

Буферизацію варто відрізняти від кешування. Основна відмінність між ними полягає в тому, що буфер може містити єдину наявну копію даних, тоді як кеш за визначенням зберігає у більш швидкій пам'яті копію даних з іншого місця.

З іншого боку, ділянку пам'яті в деяких випадках можна використати і як буфер, і як кеш. Наприклад, якщо після виконання операції введення із використанням буфера надійде запит на таку саму операцію, дані можуть бути отримані із буфера, який при цьому буде частиною кеша. Сукупність таких буферів називають буферним кешем (buffer cache).

Використання буферного кеша дає можливість накопичувати дані для збереження їх на диску великими обсягами за одну операцію, що сприяє підвищенню ефективності роботи підсистеми введення-виведення.

15.4.4. Спулінг

Спулінг (spooling) — технологія виведення даних із використанням буфера, що працює за принципом FIFO. Такий буфер називають *спулом* (spool) або ділянкою спула (spool area).

Спулінг використовують тоді, коли виведення даних має виконуватися неподільними порціями *(роботами,* jobs). Неподільність робіт полягає в тому, що їхній вміст під час виведення не перемішується Прикладом такого виведення є робота із розподілюваним принтером, коли запити на друкування документів приходять від багатьох процесів у довільному порядку, але друкуватися документи можуть тільки по одному (тут роботою є документ). Іншим прикладом є відсилання електронної пошти (роботами є повідомлення).

Роботи надходять у спул і в ньому вишиковуються у FIFO-чергу (нові роботи додаються у її хвіст). Як тільки пристрій вивільняється, роботу із голови черги передають пристрою для виведення. Зазначимо, що загалом спулінг не можна назвати технологією керування введенням-виведенням режиму ядра — його часто реалізують процеси користувача (такі як демон друкування Ірг для UNIX).

15.5. Введення-виведення у режимі користувача

Тут зупинимося на взаємодії підсистеми введення-виведення із процесами режиму користувача та на різних методах організації введення-виведення з режиму користувача.

15.5.1. Синхронне введення-виведення

У більшості випадків введення-виведення на рівні апаратного забезпечення кероване перериваннями, а отже є асинхронним. Однак використати асинхронну обробку даних завжди складніше, ніж синхронну, тому найчастіше введення-виведення в ОС реалізоване у вигляді набору *блокувальних* або *синхронних* системних викликів, подібних до read(), write() або. Під час виконання такого виклику поточний потік призупиняють, переміщуючи в чергу очікування для цього пристрою. Після завершення операції введення-виведення і отримання всіх даних від пристрою потік переходить у стан готовності та може продовжити своє виконання.

Однак синхронне введення-виведення підходить не для всіх застосувань. Зокрема, воно не підходить для таких категорій програм:

♦ серверів, що обслуговують багатьох клієнтів (отримавши з'єднання від одного клієнта, потрібно мати можливість відразу обслуговувати й інших);

♦ застосувань, що працюють із журналом (після виклику функції записування в журнал потрібно продовжити виконання негайно, не очікуючи завершення виведення);

♦ мультимедійних застосувань (відіславши запит на читання одного кадру, потрібно одночасно показувати інші).

Для вирішення цієї проблеми запропоновано кілька підходів, про які йтиметься нижче.

15.5.2. Багатопотокова організація введення-виведення

Принципи, що лежать в основі першого із можливих підходів до розв'язання проблем синхронного введення-виведення, розглянуто в розділі 3 [1]. Цей підхід полягає в тому, що за необхідності виконання асинхронного введення-виведення у застосуванні створюють новий потік, у якому виконуватиметься звичайне, синхронне введення-виведення. При блокуванні цього потоку вихідний потік продовжуватиме своє виконання.

15.5.3. Введення-виведення із повідомленням

Першою технологією, яку можна використати для організації введення-виведення без блокування і яка не вимагає організації багатопотоковості, є ***введення-виведення із повідомленням***. Ця технологія має й інші назви, наприклад мультиплексування введення-виведення (I/O multiplexing).

Загальні принципи введення-виведення із повідомленням

Якщо потрібно в циклі виконати блокувальний виклик (наприклад, read ()) для кількох файлових дескрипторів, може трапитися так, що один із викликів заблокує поточний потік у той момент, коли на дескрипторі, який використовується в іншому виклику, з'являться дані.

У цьому разі виконання введення-виведення поділяють на кілька етапів.

1. Спеціальний системний виклик (виклик повідомлення) визначає, чи можна виконати синхронне введення-виведення хоча б для одного дескриптора із заданого набору без блокування потоку.

2. Як тільки хоча б один дескриптор із набору стає готовий до введення-виведення без блокування, виклик повідомлення повертає керування; при цьому поточний потік може визначити, для яких саме дескрипторів може бути виконане введення-виведення.

3. Потік, що викликає, може тепер у циклі обійти всі дескриптори, визначені внаслідок повідомлення на етапі 2, і виконати введення-виведення для кожного з них, блокування поточного потоку ця операція в загальному випадку не спричинить.

15.5.4. Асинхронне введення-виведення

Асинхронне введення-виведення реалізоване у деяких UNIX-системах. Одна з найповніших реалізацій цієї технології доступна також в системах лінії Windows XP, де її називають *введенням-виведенням із перекриттям* (overlapped I/O). Основна ідея тут полягає в тому, що потік, який почав виконувати введення-виведення, не блокують до його завершення. Асинхронне введення-виведення зводиться до виконання таких дій.

Потік виконує системний виклик асинхронного введення або виведення, який ставить операцію введення-виведення в чергу і негайно повертає керування.

♦ Потік продовжує виконання паралельно з операцією введення-виведення.

♦ Коли операція введення-виведення завершується, потік отримує про це повідомлення.

Операція може бути перервана до свого завершення.

Основним підходом до отримання повідомлення про завершення асинхронного введення-виведення є виконання операції очікування завершення введення-виведення. При цьому потік призупиняють до завершення асинхронної операції

15.5.5. Порти завершення введення-виведення

Остання із розглянутих нами технологій — *порт завершення введення-виведення* (I/O completion port) — поєднує багатопотоковість із асинхронним введенням-виведенням для вирішення проблем розробки серверів, що обслуговують велику кількість одночасних запитів.

З іншого боку, запропоновані дотепер альтернативні однопотокові підходи (введення-виведення із повідомленням і асинхронне введення-виведення) не можуть використати переваги багатопроцесорних архітектур. Хотілося б досягти деякого компромісу між великою кількістю потоків і необхідністю використовувати багатопроцесорність.

Одним із варіантів такого компромісу є організація *пула потоків* (thread pool). При цьому в момент запуску серверного застосування

заздалегідь створюють набір потоків (пул), кожен із яких готовий обслуговувати запити. Коли приходить запит, перевіряють, чи є в пулі вільні потоки, якщо є, з нього вибирають потік, який починає обслуговувати запит. Після виконання запиту потік повертають у пул. Коли з появою нового запиту вільних потоків у пулі немає, запит поміщають у чергу, і він там очікує, поки не вивільниться потік, що може його обслужити. При цьому можна керувати розміром пула, досягаючи того, щоб кількість активних потоків збігалася із кількістю процесорів у системі.

Для підтримки організації такого пула потоків і було розроблено технологію портів завершення введення-виведення. Такі порти дотепер реалізовані лише у системах лінії Windows XP. Розглянемо особливості їхнього використання [1].

15.6. Таймери і системний час

Таймери керують пристроями, які передають у систему інформацію про час. Вони відстежують поточний час доби, здійснюють облік витрат процесорного часу, повідомляють процеси про події, що відбуваються через певний проміжок часу тощо. Робота із такими пристроями відрізняється від традиційної моделі введення-виведення, для них використовують окремий набір системних викликів.

15.6.1. Керування системним часом

Апаратний таймер - це пристрій, що генерує переривання таймера через певний проміжок часу. Розглянемо, як такий пристрій можна використати для відстеження поточного системного часу.

Таке завдання розв'язують просто: створюють лічильник, який збільшують для кожного переривання таймера. Основною проблемою є розмір цього лічильника, а саме:

♦ 32-бітне значення не може зберігати достатньо великий проміжок часу (переповнення такого лічильника за частоти переривання таймера 60 Гц настане упродовж двох років);

♦ 64-бітне значення на 32-бітному процесорі (наприклад, в архітектурі IA-32) оброблятиметься неефективно.

Для реалізації 32-бітного лічильника звичайно використовують такі підходи.

♦ Зберігають лише інформацію про секунди, а про долі поточної секунди (мілі-секунди, мікросекунди) — окремо. У цьому разі лічильника секунд вистачить для зберігання інформації про 2^{32} с (більш як на 135 років).

4 Зберігають інформацію про кількість переривань із моменту останнього завантаження системи, а час останнього завантаження зберігають окремо (як 64-бітне значення). У разі запиту поточного часу значення лічильника і збережений час завантаження додають.

Якщо поряд із таймером у системі є годинник, значення цього лічильника може час від часу звірятися із показаннями годинника.

15.7. Керування введенням-виведенням: Windows XP

15.7.1. Основні компоненти підсистеми введення-виведення

Базовим компонентом підсистеми введення-виведення Windows XP є менеджер введення-виведення (I/O Manager).

Передавання даних між рівнями підсистеми

Обмін даними між рівнями підсистеми введення-виведення є асинхронним. Більшу частину таких даних подано у вигляді пакетів, які передають від одного компонента підсистеми до іншого, можливо, змінюючи на ходу. Кажуть, що ця підсистема Windows XP є *керованою пакетами* (packet driven). Такі пакети називають *пакетами запитів введення-виведення* (I/O Request Packet, IRP), для стислості називатимемо їх *пакетами IRP*.

Менеджер введення-виведення створює пакет IRP, що відображає операцію введення-виведення, передає покажчик на нього потрібному драйверу і вивільняє пам'ять з-під нього після завершення операції. Драйвер, у свою чергу, отримує такий пакет, виконує визначену в ньому операцію і повертає його назад менеджерові введення-виведення як індикатор завершення операції або для передавання іншому драйверу для подальшої обробки.

Категорії драйверів пристроїв

Windows XP дає змогу використати кілька категорій драйверів режиму ядра.

Найбільше поширення останнім часом набули *WDM-драйвери*. Такі драйвери мають відповідати вимогам стандарту, який називають *Windows Driver Model* (WDM). Його розроблено для драйверів, використовуваних у лінії Windows XP та останніх версіях Consumer Windows (Windows 98/Me). Звичайно для переносу таких драйверів між системами достатньо їх перекомпілювати, а деякі з них сумісні на рівні двійкового коду. Розрізняють три типи WDM-драйверів.

Драйвери шини (bus drivers) керують логічною або фізичною шиною (наприклад, PCI, USB, ISA). Такий драйвер відповідає за виявлення пристроїв, з'єднаних із певною шиною.

Функціональні драйвери (function drivers) керують пристроєм конкретного типу. Драйвери шини надають пристрої функціональним драйверам. Звичайно тільки функціональний драйвер працює з апаратним забезпеченням пристрою, саме він дає змогу системі використати пристрій *Драйвери-фільтри* (filter drivers) доповнюють або змінюють поведінку інших драйверів.

Насправді жоден драйвер, відповідно до стандарту WDM, не може цілковито відповідати за керування пристроєм, усі вони доповнюють один одного.

Крім WDM-драйверів, у Windows XP підтримують такі категорії драйверів ядра: *файлових систем,* відповідальні за перетворення запитів введення-виведення, що використовують файли, у запити до низькорівневих драйверів пристроїв (наприклад, драйвера жорсткого диска); *відображення* (display drivers) підсистеми Win32, які перетворюють незалежні від пристрою запити GDI-підсистеми в команди графічного адаптера або у прямі операції записування у відеопам'ять; *успадковані,* розроблені для Windows NT.

На доповнення до драйверів ядра Windows XP підтримує драйвери режиму користувача. До них, зокрема, належать драйвери принтерів, які перетворюють незалежні від пристрою запити GDI-підсистеми в команди відповідного принтера і передають ці команди WDM-драйверу (наприклад, драйверу паралельного порту або універсальному драйверу USB-принтера).

Підтримка конкретного пристрою може бути розділена між кількома драйверами. Залежно від рівня цієї підтримки виділяються додаткові категорії драйверів.

♦ *Клас-драйвери* (class drivers). Реалізують інтерфейс обробки запитів введення-виведення, специфічних для конкретного класу пристроїв, наприклад драйвери дисків або пристроїв CD-ROM.

♦ *Порт-драйвери* (port drivers). Реалізують інтерфейс обробки запитів введення-виведення, специфічних для певного класу портів введення-виведення; зокрема до цієї категорії належить драйвер підтримки SCSI.

♦ *Мініпорт-драйвери* (miniport drivers). Керують реальними пристроями (наприклад, SCSI-адаптерами конкретного типу) і реалізують інтерфейс, наданий клас-драйверами і порт-драйверами.

Структура драйвера пристрою

Розглянемо структуру драйвера пристрою [1]. Можна виділити основні процедури драйвера.

♦ *Процедура ініціалізації.* Звичайно називається DriverEntry, її виконує менеджер введення-виведення під час завантаження драйвера у систему, і зазвичай вона здійснює глобальну ініціалізацію структур даних драйвера.

♦ *Процедура додавання пристрою* (add-device routine). Вона має бути реалізована будь-яким драйвером, що підтримує специфікацію Plug and Play. Менеджер Plug and Play викликає цю процедуру, якщо знаходить пристрій, за який відповідає драйвер. У ній звичайно створюють структуру даних, відображувану пристроєм (об'єкт пристрою).

♦ Набір *процедур диспетчеризації* (dispatch routines). Ці процедури реалізують дії, допустимі для пристрою (відкриття, закриття, читання, записування тощо). Саме їх викликає менеджер введення-виведення під час виконання запиту.

♦ *Процедура обробки переривання* аналогічна коду верхньої половини оброблювача переривання для Linux. Вона **є** оброблювачем переривання від пристрою, виконується із високим пріоритетом; основне її завдання - запланувати для виконання нижню половину оброблювача (DPC-процедуру).

♦ *Процедура відкладеної обробки* ***переривання,*** DPC-процедура (DPC routine), відповідає коду нижньої половини оброблювача переривання в Linux. Вона виконує більшу частину роботи, пов'язаної з обробкою переривання, після чого сигналізує про необхідність переходу до коду завершення введення-виведення.

Висновки

✓ Однією із найважливіших функцій ОС є керування пристроями введення-виведення. Під час його реалізації насамперед важливо безпосередньо реалізувати виконання операцій введення-виведення. Найпоширенішими підходами до розв'язання цього завдання є опитування пристроїв введення-виведення на основі переривань і використання контролерів доступу до пам'яті (DMA).

✓ Другим важливим завданням є реалізація операцій з організації виконання введення-виведення у ядрі. Основними підходами тут є планування операцій введення-виведення, буферизація і спулінг.

✓ Третім завданням є організація різних засобів введення-виведення для використання в режимі користувача. Сучасні ОС надають різні високоефективні підходи до реалізації таких засобів: синхронне й асинхронне введення-виведення, введення-виведення із повідомленням, порти завершення введення-виведення. Більшість цих засобів розраховані на використання у поєднанні з багатопотоковістю.

✓ Для реалізації всіх цих можливостей ОС повинна мати драйвери пристроїв, які реалізують базовий набір операцій доступу до пристроїв і надають для використання цих операцій простий у застосуванні універсальний інтерфейс.

Контрольні запитання та завдання

1. Коли краще використовувати опитування завершення введення-виведення, а коли — введення-виведення, кероване перериваннями?

2. Для яких із приведених пристроїв є сенс використовувати буферизацію, для яких — спулінг, а для яких — кешування:

а) миша;

б) накопичувач на магнітній стрічці (стример);

в) накопичувач на жорстких магнітних дисках;

г) графічний адаптер?

3. У сучасних ОС введення-виведення з повідомленням часто дає змогу досягти підтримки більшого числа клієнтів порівняно з іншими підходами. Поясніть, у яких випадках це відбувається і чому.

4. Які характеристики повинна мати підсистема введення-виведення?

5. Яким чином забезпечують універсальність інтерфейсу драйверів пристроїв?

6. Які є типи драйверів пристроїв? Яка різниця між блоковими та символьними драйверами?

7. Які методи використовуються в підсистемі введення-виведення на рівні ядра?

8. Які є способи буферизації?

9. Яка різниця між синхронним та асинхронним введенням-виведенням?

10. З яких компонентів складається підсистема введення-виведення Windows XP?