Міністерство освіти і науки України
Відокремлений структурний підрозділ
«Ковельський промислово-економічний фаховий коледж
Луцького національного технічного університету»

«Операційні системи»

Методичні вказівки до самостійної роботи
для здобувачів освітньо-професійного ступеня
фаховий молодший бакалавр IV курсу
спеціальності 122 Комп'ютерні науки
денної форми навчання

Ковель, 2022

Самостійна робота №1

Тема: Операційна система як розподілювач ресурсів

Мета: Дослідити, як операційна система розподіляє ресурсами

План

1. **Що таке ресурси?**
2. **Види розподілення ресурсів.**

Теоретичні відомості

1.Що таке ресурси?

Операційна система має ефективно розподіляти ресурси. Під ресурсами розуміють процесорний час, дисковий простір, пам'ять, засоби доступу до зовнішніх пристроїв. Операційна система виступає в ролі менеджера цих ресурсів і надає їх прикладним програмам на вимогу.

2.Види розподілення ресурсів.

Розрізняють два основні види розподілу ресурсів. У разі просторового розподілу ресурс доступний декільком споживачам одночасно, при цьому кожен із них може користуватися частиною ресурсу (так розподіляється пам'ять). У разі часового розподілу система ставить споживачів у чергу і згідно з нею надає їм змогу користуватися всім ресурсом обмежений час (так розподіляється процесор в од- нопроцесорних системах).

При розподілі ресурсів ОС розв'язує можливі конфлікти, запобігає несан-кціонованому доступу програм до тих ресурсів, на які вони не мають прав, забезпечує ефективну роботу комп'ютерної системи.

Питання для самоконтролю

1. Наведіть кілька прикладів просторового і часового розподілу ресурсів комп'ютера.
2. Від чого залежить вибір того чи іншого методу розподілу?

Теми для рефератів

1. Сучасні архітектури розподілених систем.
2. Класифікація сучасних операційних систем.

Список рекомендованих джерел

1. Шеховцов В.А. Операційні системи. – К.:Видавнича група BHV, 2005. – 576с.:іл.
2. Рихтер Д.Windows для профессионалов: Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows. – М.: Русская редакция, 2001. – 752с.
3. Харт Дж.В. Системное программирование в бреде Win32 – М.: Вильямс, 2001 – 464с.
4. Рихтер Джеффри. Windows для проффесионалов: Програмирование для Windows 95 иWindowsNT 4.0. на базе Win32/API. – М.: Изд.отдел «Русская редакция», 1997. – 712с.
5. Румянцев П.В. Азбука программирования в Win32 API. – М.: Радио и связь, 1998. – 272с.

Самостійна робота №2

Тема: Особливості архітектуриUnix.

Мета: Вивчити особливості ОС Unix та базові механізми роботи.

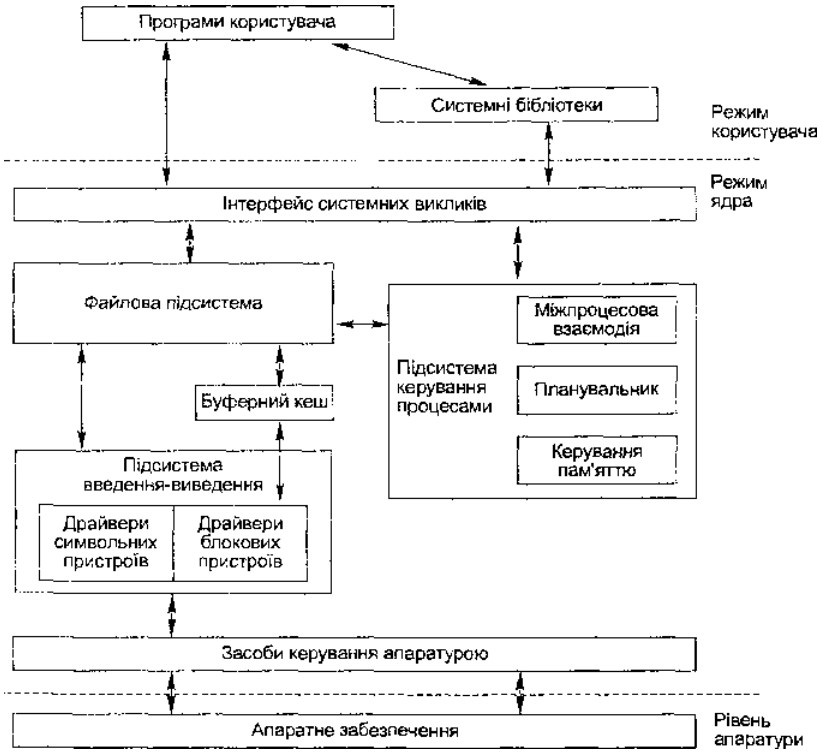
План

1. **Базова архітектура Unix.**
2. **Архітектура ОСUnix:**
   - **2.1.Підсистема керування процесами**
   - **2.2.Файлова підсистема**
   - **2.3.Підсистема введення-виведення.**
3. **Відмінності архітектури Unix.**

Теоретичні відомості

1.Базова архітектура Unix

UNIX є прикладом досить простої архітектури ОС. Більша частина функціональності цієї системи міститься в ядрі, ядро спілкується із прикладними програмами за допомогою системних викликів. Базова структура класичного ядра UNIX зображена на мал.



2.Архітектура ОС Unix:

2.1 Підсистема керування процесами

Система складається із трьох основних компонентів: підсистеми керування процесами, файлової підсистеми та підсистеми введення-виведення.

Підсистема керування процесами контролює створення та вилучення процесів, розподілення системних ресурсів між ними, міжпроцесову взаємодію, керування пам’яттю.

2.2 Файлова підсистема

Файлова підсистема забезпечує єдиний інтерфейс доступу до даних, розташованих на дискових накопичувачах, і до периферійних пристроїв. Такий інтерфейс є однією з найважливіших особливостей Unix. Одні й ті самі системні виклики використовують як для обміну даними із диском, так і для виведення на

термінал або принтер (програма працює із принтером так само, як із файлом). При цьому файлова система переадресовує запити відповідним модулям підсистеми введення-виведення, а ті — безпосередньо периферійним пристроям. Крім того, файлова підсистема контролює права доступу до файлів, які значною мірою визначають привілеї користувача в системі.

2.3Підсистема введення-виведення

Підсистема введення-виведення виконує запити файлової підсистеми, взаємодіючи з драйверами пристроїв. В Unix розрізняють два типи пристроїв: символьні (наприклад, принтер) і блокові (наприклад, жорсткий диск). Основна відмінність між ними полягає втому, що блоковий пристрій допускає прямий доступ. Для підвищення продуктивності роботи із блоковими пристроями використовують буферний кеш — ділянку пам'яті, у якій зберігаються дані, зчитані з диска останніми. Під час наступних звертань до цих даних вони можуть бути отримані з кеша.

3.Відмінності архітектури Unix

Сучасні Unix -системи дещо відрізняються за своєю архітектурою.

♦ У них виділено окремий менеджер пам'яті, відповідальний за підтримку віртуальної пам'яті.

♦ Стандартом для реалізації інтерфейсу файлової системи є віртуальна файлова система, що абстрагує цей інтерфейс і дає змогу організувати підтримку різних типів файлових систем.

♦ У цих системах підтримується багатопроцесорна обробка, а також багатопото- ковість.

Базові архітектурні рішення, такі як доступ до всіх пристроїв введення-виведення через інтерфейс файлової системи або організація системних викликів, залишаються незмінними в усіх реалізаціях Unix.

Питання для самоконтролю

1. Чи може процесор переходити у привілейований режим під час виконання програми користувача?
2. Чи може така програма виконуватися виключно в привілейованому режимі?

Теми для рефератів

1. Реалізація керування основною пам`яттю Unix.
2. Сторінкова адресація в Unix.
3. Керування введенням- виведенням Unix.

Список рекомендованих джерел

1. Шеховцов В.А. Операційні системи. – К.:Видавнича група BHV, 2005. – 576с.:іл.
2. Рихтер Д.Windows для профессионалов: Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows. – М.: Русская редакция, 2001. – 752с.
3. Харт Дж.В. Системное программирование в бреде Win32 – М.: Вильямс, 2001 – 464с.
4. Рихтер Джеффри. Windows для проффесионалов: Програмирование для Windows 95 и WindowsNT 4.0. на базе Win32/API. – М.: Изд.отдел «Русская редакция», 1997. – 712с.

Самостійна робота №3

Тема: Особливості архітектуриLinux.

Мета: Охарактеризувати технологічний процес, зміст, основні положення в Linux.

План

1. **Основні складові частини Linux.**
2. **Призначення ядра Linux і його особливості**
3. **Модулі ядра**
4. **Особливості системних бібліотек**
5. **Застосування користувача**

Теоретичні відомості

1.Основні складові частини Linux.

В ОС Linux можна виділити три основні частини:

♦ ядро, яке реалізує основні функції ОС (керування процесами, пам'яттю, введенням-виведенням тощо);

♦ системні бібліотеки, що визначають стандартний набір функцій для використання у застосуваннях (виконання таких функцій не потребує переходу в привілейований режим);

♦ системні утиліти (прикладні програми, які виконують спеціалізовані задачі).

2.Призначення ядра Linux і його особливості

Linux реалізує технологію монолітного ядра. Весь код і структури даних ядра пе-ребувають в одному адресному просторі. У ядрі можна виділити кілька функціональних компонентів

♦ Планувальник процесів — відповідає за реалізацію багатозадачності в системі (обробка переривань, робота з таймером, створення і завершення процесів, перемикання контексту).

♦ Менеджер пам'яті — виділяє окремий адресний простір для кожного процесу і реалізує підтримку віртуальної пам'яті.

♦ Віртуальна файлова система — надає універсальний інтерфейс взаємодії з різними файловими системами та пристроями введення-виведення.

♦ Драйвери пристроїв — забезпечують безпосередню роботу з периферійними пристроями. Доступ до них здійснюється через інтерфейс віртуальної файлової системи.

♦ Мережний інтерфейс — забезпечує доступ до реалізації мережних протоколів і драйверів мережних пристроїв.

♦ Підсистема міжпроцесової взаємодії — пропонує механізми, які дають змогу різним процесам у системі обмінюватися даними між собою.

Деякі із цих підсистем є логічними компонентами системи, вони завантажуються у пам'ять разом із ядром і залишаються там постійно. Компоненти інших підсистем (наприклад, драйвери пристроїв) вигідно реалізовувати так, щоб їхній код міг завантажуватися у пам'ять на вимогу. Для розв'язання цього завдання Linux підтримує концепцію модулів ядра.

3. Модулі ядра

Ядро Linux дає можливість на вимогу завантажувати у пам’ять і вивантажувати з неї окремі секції коду. Такі секції називають модулями ядра (kernel modules) і виконують у привілейованому режимі.

Модулі ядра надають низку переваг.

Код модулів може завантажуватися в пам’ять у процесі роботи системи, що спрощує налагодження компонентів ядра, насамперед драйверів.

♦ З’являється можливість змінювати набір компонентів ядра під час виконання: ті з них, які в цей момент не використовуються, можна не завантажувати у пам’ять.

♦ Модулі є винятком із правила, за яким код, що розширює функції ядра, відповідно до ліцензії Linux має бути відкритим. Це дає змогу виробникам апаратного забезпечення розробляти драйвери під Linux, навіть якщо не заплановано надавати доступ до їхнього вихідного коду.

Підтримка модулів у Linux складається із трьох компонентів.

♦ Засоби керування модулями дають можливість завантажувати модулі у пам’ять і здійснювати обмін даними між модулями та іншою частиною ядра.

♦ Засоби реєстрації драйверів дозволяють модулям повідомляти іншу частину ядра про те, що новий драйвер став доступним.

♦ Засоби розв’язання конфліктів дають змогу драйверам пристроїв резервувати апаратні ресурси і захищати їх від випадкового використання іншими драйверами.

Один модуль може зареєструвати кілька драйверів, якщо це потрібно (наприклад, для двох різних механізмів доступу до пристрою).

Модулі можуть бути завантажені заздалегідь — під час старту системи (завантажувальні модулі) або у процесі виконання програми, яка викликає їхні функції. Після завантаження код модуля перебуває в тому ж самому адресному просторі, що й інший код ядра. Помилка в модулі є критичною для системи.

4. Особливості системних бібліотек

Системні бібліотеки Linux є динамічними бібліотеками, котрі завантажуються у пам’ять тільки тоді, коли у них виникає потреба.

Вони виконують ряд функцій:

♦ реалізацію пакувальників системних викликів;

♦ розширення функціональності системних викликів (до таких бібліотек належить бібліотека введення-виведення мови С, яка реалізує на основі системних викликів такі функції, як printfO);

♦ реалізацію службових функцій режиму користувача (сортування, функції обробки рядків тощо).

5. Застосування користувача

Застосування користувача в Linux використовують функції із системних бібліотек і через них взаємодіють із ядром за допомогою системних викликів.

Питання для самоконтролю

1. Автор Linux Лінус Торвальдс стверджує, що мобільність Linux має поширюватися на системи з «прийнятною» (reasonable) архітектурою. Які апаратні засоби повинна підтримувати така архітектура?
2. Наведіть переваги і недоліки реалізації взаємодії прикладної програми з операційною системою в Linux.

Теми для рефератів

1. Архітектура мережної підтримки Linux.
2. Керування введенням-виведенням Linux.
3. Виконувані файли в Linux.

Список рекомендованих джерел

1. Шеховцов В.А. Операційні системи. – К.:Видавнича група BHV, 2005. – 576с.:іл.
2. Рихтер Д.Windows для профессионалов: Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows. – М.: Русская редакция, 2001. – 752с.
3. Харт Дж.В. Системное программирование в бреде Win32 – М.: Вильямс, 2001 – 464с.
4. Рихтер Джеффри. Windows для проффесионалов: Програмирование для Windows 95 и WindowsNT 4.0. на базе Win32/API. – М.: Изд.отдел «Русская редакция», 1997. – 712с.
5. Румянцев П.В. Азбука программирования в Win32 API. – М.: Радио и связь, 1998. – 272с.

Самостійна робота №5

Тема: Керування процесамиі потоками в UNIX і Linux

Мета: Вивчити основні етапи керування процесами в UNIX і Linux.

План

1. **Образ процесу**
2. **Ідентифікаційна інформація та атрибути безпеки процесу**
3. **Керуючий блок процесу**
4. **Завершення процесу**
5. **Запуск програми**

5.1.Запуск виконуваних файлів у прикладних програмах

5.2. Реалізація технології fork+exec

6. Очікування завершення процесу

6.1. Системний виклик waitpid()

6.2. Синхронне виконання процесів у прикладних програмах

Теоретичні відомості

1. Образ процесу

В UNIX-системах образ процесу містить такі компоненти:

♦ керуючий блок процесу;

♦ код програми, яку виконує процес;

♦ стек процесу, де зберігаються тимчасові дані (параметри процедур, повернені значення, локальні змінні тощо);

♦ глобальні дані, спільні для всього процесу.

Для кожного компонента образу процесу виділяють окрему ділянку пам’яті. У розділі 9 буде докладно розглянуто структуру різних ділянок пам’яті процесу та особливості їхнього використання.

2. Ідентифікаційна інформація та атрибути безпеки процесу

Із кожним процесом у системі пов’язана ідентифікаційна інформація.

Ідентифікатор процесу (pid) є унікальним у межах усієї системи, і його використовують для доступу до цього процесу. Ідентифікатор процесу і nitзавжди дорівнює одиниці.

Ідентифікатор процесу-предка (ppid) задають під час його створення. Будь-який процес має мати доступ до цього ідентифікатора. Так в UNIX-системах обов’язково підтримується зв’язок «предок-нащадок». Якщо предок процесу *P*завершує виконання, предком цього процесу автоматично стає і nit, тому ppidдля *P* дорівнюватиме одиниці.

Із процесом також пов’язаний набір атрибутів безпеки.

♦Реальні ідентифікатори користувача і групи процесу (uid, gid) відповідають користувачеві, що запустив програму, внаслідок чого в системі з’явився відповідний процес.

♦ Ефективні ідентифікатори користувача і групи процесу (euid, egid) використовують у спеціальному режимі виконання процесу — виконанні з правами власника.

3.**Керуючий блок процесу**

Розглянемо структуру керуючого блоку процесу в Linuxі відмінності його реалізації у традиційних UNIX-системах [58, 95].

Керуючий блок процесу в Linuxвідображається структурою даних taskstruct. До найважливіших полів цієї структури належать поля, що містять таку інформацію:

♦ ідентифікаційні дані (зокрема pid— ідентифікатор процесу);

♦ стан процесу (виконання, очікування тощо);

♦ покажчики на структури предка і нащадків;

♦ час створення процесу та загальний час виконання (так звані таймери процесу);

♦ стан процесора (вміст регістрів і лічильник інструкцій);

♦ атрибути безпеки процесу (uid, gid, euid, egid).

Зазначимо, що в ядрі Linuxвідсутня окрема структура даних для потоку, тому інформація про стан процесора міститься в керуючому блоці процесу.

Крім перелічених вище, task_structмає кілька полів спеціалізованого призначення, необхідних для різних підсистем Linux:

♦ відомості для обробки сигналів;

♦ інформація для планування процесів;

♦ інформація про файли і каталоги, пов’язані із процесом;

♦ структури даних для підсистеми керування пам’яттю.

Дані полів taskstructможуть спільно використовувати декілька процесів спеціалізованого призначення, у цьому випадку такі процеси фактично є потоками. Докладніше дане питання буде розглянуте під час вивчення реалізації багатопотоковості в Linux.

Керуючі блоки процесу зберігаються в ядрі у спеціальній структурі даних. До появи ядра версії 2.

♦ таку структуру називали таблицею процесів системи; це був масив, максимальна довжина якого не могла перевищувати

♦ Кбайт. У ядрі версії 2

.♦ таблиця процесів була замінена двома динамічними структурами даних, що не мають такого обмеження на довжину:

♦ хеш-таблицею (де в ролі хеша виступає pidпроцесу), ця структура дає змогу швидко знаходити процес за його ідентифікатором;

♦ кільцевим двозв’язним списком, ця структура забезпечує виконання дій у циклі для всіх процесів системи.

Тепер обмеження на максимальну кількість процесів перевіряється тільки всередині реалізації функції forkOі залежить від обсягу доступної пам’яті (наприклад, є відомості що у системі з 512 Мбайт пам’яті можна створити близько

Для нащадка forkOповертає нуль, а для предка — ідентифікатор (pid) створеного процесу-нащадка. Коли з якоїсь причини нащадок не був створений, fork() поверне -1. Тому звичайний код роботи з fork() має такий вигляд:

```
pid_tpid:
   if ((pid - forkO) — -1) { /* помилка, аварійне завершення */ } if (pid
— 0) {
        // це нащадок
```

```
}
else{
        // це предок
        printf("нащадок запущений з кодом Xd\n". pid):
}
```

Після створення процесу він може дістати доступ до ідентифікаційної інформації за допомогою системного виклику getpidO, який повертає ідентифікатор поточного процесу, і getppidO, що повертає ідентифікатор процесу-предка:

```
    #include<unistd.h> pid_t mypid.
parent_pid; mypid = getpidO: parent_pid =
getppidO;
```

4.Завершенняпроцесу

Для завершення процесу в UNIX-системах використовують системний викликexitO. Розглянемо реалізацію цього системного виклику в Linux. Під час його виконання відбуваються такі дії.

1. Стан процесу стає TASKZOMBIE (зомбі, про цей стан див.нижче).
2. Процес повідомляє предків і нащадківпро те, що він завершився (за допомогою спеціальних сигналів).
3. Звільняються ресурси, виділені під час виклику forkO.
4. Планувальника повідомляють про те, що контекст можна перемикати.

У прикладних програмах для завершення процесу можна використати безпосередньо системний виклик _exi t() або його пакувальник — бібліотечну функцію exit(). Ця функція закриває всі потоки процесу, коректно вивільняє всі ресурси і викликає exitO для фактичного його завершення:

```
#include<unistd.h>
void _exit(int status): // status задає код повернення #include<stdlib.h>
void exit(int status): // status задає код повернення exit (128):    //        вихід з кодом повернення 128
```

Зазначимо, що краще не викликати exit() тоді, коли процес може використо¬вувати ресурси разом з іншими процесами (наприклад, він є процесом-нащадком, який має предка, причому нащадок успадкував у предка дескриптори ресурсів). Причина полягає втому, що в цьому разі спроба звільнити ресурси в нащадку призведе до того, що вони виявляться звільненими й у предка. Для завершення таких процесів потрібно використати exitO.

5.Запуск програми

Запуск нових програм в UNIXвідокремлений від створення процесів і реалізований за допомогою системних викликів сімейства execO. На практиці звичайно реалізують один виклик (у Linuxце — execveO).

У Linuxпідтримують концепцію оброблювачів двійкових форматів(binaryformathandlers). Формати виконуваних файлів заздалегідь реєструють в ядрі, при цьому їхній набір може динамічно змінюватися. З кожним форматом пов’язаний оброблювач — спеціальна структура даних 1 і nux_binfmt, що містить покажчики на процедури завантаження виконуваного файла (1 oad_binary) і динамічної бібліотеки (1oad_shlib) для цього формату.

Тепер зупинимося на послідовності кроків виконання execveOy Linux. Вхід-

ними параметрами цього виклику є рядок з ім'ям виконуваного файла програми, масив аргументів командного рядка (до яких у коді процесу можна буде дістати доступ за допомогою масиву argv) і масив змінних оточення (що є парами *ім'я=значення).* Виклик execve() відбувається так.

1. Відкривають виконуваний файл програми.
2. Використовуючи вміст цього файла, ініціалізують спеціальну структуру даних bprm, що містить інформацію, необхідну для виконання файла, зокрема відомості про його двійковий формат.
3. Виконують дії для забезпечення безпеки виконання коду.
4. На базі параметрів виклику формують командний рядок і набір змінних оточення програми (це спеціальні поля структури bprm).
5. Для кожного зареєстрованого оброблювача бінарного формату викликають процедуру завантаження виконуваного файла через покажчик і oadbinaryi перевіряють результат цього виклику:

а) уразі коректного завантаження оброблювач починає виконувати код нової програми;

б) якщо жоден з оброблювачів не зміг коректно завантажити код, повертають помилку.

Детальному розгляду процесу завантаження програмного коду в пам'ять і форматів *виконуваних файлів буде присвячено розділ 14.*

Запуск виконуваних файлів у прикладних програмах

Розглянемо приклад запуску виконуваного файла за допомогою системного виклику execveO. Синтаксис цього виклику такий:

```
#include<unistd.h>
int execve(constchar *filename.  //      повнеім'я файлу
           constchar *argv[], //          масив     аргументів командного рядка
           constchar *envp[]);    //      масивзмінних оточення
```

Останнім елементом argvi envpмає бути нульовий покажчик.

Повернення з виклику буде тільки тоді, коли програму не вдалося завантажити (наприклад, файл не знайдено). Тоді буде повернено -1.

Наведемо приклад формування параметрів і виклику (зазначимо, що першим елементом argvзадане ім'я виконуваної програми):

```
char *prog = "./child";
char *args[] = { "./child", "argl", NULL }:
char *env[] = { NULL }:
if(execve (prog, args, env) == -1) {
printf("помилка під час виконання програми: Xs\n". prog); exit
(-1);
}
```

Реалізація технології fork+exec

Наведемоприкладкоду, щореалізуєтехнологіюfork+exec: if ((pld = forkO) < 0) exit (-1);

if (pid == 0) { // нащадок завантажує замість себе іншу програму *II* … завдання prog, argsi envif(execve(prog, args, env) == -1) {

printfCпомилка під час запуску нащадка\п");

```
exit(-1);
 }
 }
 else{
 // предок продовжує роботу (наприклад, очікує закінчення нащадка)
```

6.Очікування завершення процесу

Коли процес завершується, його керуючий блок не вилучається зі списку й хеша процесів негайно, а залишається там доти, поки інший процес (предок) не видалить його звідти. Якщо процес насправді всистемі відсутній (він завершений), а є тільки його керуючий блок, то такий процес називають *процесом-зомбі* (zombieprocess).

Системний виклик waitpid()

Для вилучення із системи інформації про процес в Linuxможна використати описаний вище системний виклик waitO, але частіше застосовують його більш універсальний варіант — waitpidO. Цей виклик перевіряє, чи є керуючий блок відповідного процесу в системі. Якщо він є, а процес не перебуває у стані зомбі (тобто ще виконується), то процес у разі виклику wa1tpid() переходить у стан очікування. Коли ж процес-нащадок завершується, предок виходить зі стану очікування і вилучає керуючий блок нащадка. Якщо предок не викличе waitpidOдля нащадка, той може залишитися у стані зомбі надовго.

Синхронне виконання процесів у прикладних програмах

Розглянемо реалізацію синхронного виконання процесів на базі waitpidO. Відповідно до POSIXцей системний виклик визначається так:

```
#i nciude<sys/wait.h>
pid_twaitpid(pid_tpid. // pldпроцесу, який очікуємо
        int *status, // інформація про статус завершення нащадка
    intoptions): // задаватимемо як 0
```

Параметр pidможна задавати як 0, що означатиме очікування процесу із тієї ж групи, до якої належить предок, або як -1, що означатиме очікування будь-якого нащадка. Наведемо приклад реалізації синхронного виконання з очікуванням:

```
pid_tpid;
   if ((pid = forkO) == -1) exit(-l): if (pid == 0)
{
    // нащадок - виклик execO
}
else {
    // предок - чекати нащадка int Status;
    waitpid (pid. Sstatus. 0);
    // продовжувати виконання
}
```

Зі значення statusможна отримати додаткову інформацію про процес-наща- док, що завершився. Для цього є низка макропідстановок з <sys/wait.h>:

♦ WIFEXITED(status) — ненульове значення, коли нащадок завершився нормально; 4 WEXITSTATUS(status) — код повернення нащадка (тільки коли WIFEXITEDO!=0). Код повернення нащадка отриманий таким чином:

```
    waitpid (pid, Sstatus. 0): if
(WIFEXITED(status))
        printf("нащадок завершився з кодом M\n". WEXITSTATUS(status));
```

Асинхронне виконання процесів може бути реалізоване на основі сигналів і буде розглянуте в наступному розділі.

Питання для самоконтролю

1. У яких ситуаціях під час розробки програмного забезпечення доцільніше використовувати модель процесів, а не модель потоків?
2. Для трьох станів потоків — виконання, готовності й очікування — перелічіть усі можливі переходи зі стану в стан і назвіть причини таких переходів. Скажіть також, які переходи неможливі, та поясніть чому. У якому зі станів потоки перебувають найдовше?

Теми для рефератів

1. Робота із системним журналом UNIX
2. Підтримка шифрувальних файлових систем у Linux
3. Підтримка багатопроцесорності в Linux

Список рекомендованих джерел

1. Шеховцов В.А. Операційні системи. – К.:Видавнича група BHV, 2005. – 576с.:іл.
2. Рихтер Д.Windows для профессионалов: Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows. – М.: Русская редакция, 2001. – 752с.
3. Харт Дж.В. Системное программирование в бреде Win32 – М.: Вильямс, 2001 – 464с.
4. Рихтер Джеффри. Windows для проффесионалов: Програмирование для Windows 95 и WindowsNT 4.0. на базе Win32/API. – М.: Изд.отдел «Русская редакция», 1997. – 712с.
5. Румянцев П.В. Азбука программирования в Win32 API. – М.: Радио и связь, 1998. – 272с.

Самостійна робота №6

Тема:Планування процесів і потоків

Мета: Ознайомитись з принципами планування процесів та потоків

План

1. Загальніпринципи, видитастратегіїпланування

1.1 Механізми і політика планування

1.2Застосовність принципів планування

2. Алгоритмипланування

Теоретичні відомості

1. Загальні принципи планування

З погляду планування виконання потоку можна зобразити як цикл чергування періодів обчислень (використання процесора) і періодів очікування введення-виведення. Інтервал часу, упродовж якого потік виконує тільки інструкції процесора, називають інтервалом використання процесора (CPUburst), інтервал часу, коли потік очікує введення-виведення, — інтервалом введення-виведення (I/Oburst). Найчастіше ці інтервали мають довжину від 2 до 8 мс.

Потоки, які більше часу витрачають на обчислення і менше — на введення-виведення, називають *обмеженими можливостями процесора*(CPUbound). Вони активно використовують процесор. Основною їхньою характеристикою є час, витрачений на обчислення, інтервали використання процесора для них довші. Потоки, які більшу частину часу перебувають в очікуванні введення-виведення, називають обмеженими можливостями введення-виведення (I/Obound). Такі потоки завантажують процесор значно менше, а середня довжина інтервалу використання процесора для них невелика. Що вища тактова частота процесора, то більше потоків можна віднести до другої категорії.

1.1. Механізми і політика планування

Слід розрізняти *механізми* і *політику* планування. До механізмів планування належать засоби перемикання контексту, засоби синхронізації потоків тощо, до політики планування - засоби визначення моменту часу, коли необхідно перемкнути контекст. Ту частину системи, яка відповідає за політику планування, називають *планувальником*(scheduler), а алгоритм, що використовують при цьому, - *алгоритмом планування*(schedulingalgorithm).

Є різні критерії оцінки політики планування, одні з них застосовні для всіх систем, інші — лише для пакетних систем або лише для інтерактивних.

Сьогодні найчастіше використовують три критерії оцінки досягнення мети.

4" *Мінімальний час відгуку.* Це найважливіший критерій для інтерактивних систем. Під часом відгуку розуміють час між запуском потоку (або введенням користувачем інтерактивної команди) і отриманням першої відповіді. Для сучасних систем прийнятним часом відгуку вважають 50-150 мс.

Максимальна пропускна здатність. Це кількість задач, які система може виконувати за одиницю часу (наприклад, за секунду). Такий критерій доцільно застосовувати у пакетних системах; в інтерактивних системах він може бути використаний для фонових задач. Щоб підвищити пропускну здатність, необхідно:

♦ скорочувати час даремного навантаження (наприклад, час, необхідний для перемикання контексту);

♦ ефективніше використати ресурси (для того, щоб ані процесор, ані пристрої введення-виведення не простоювали).

♦ Третім критерієм є *справедливість,* яка полягає в тому, що процесорний час потокам виділяють відповідно до їхньої важливості. Справедливість забезпечує такий розподіл процесорного часу, що всі потоки просуваються у своєму виконанні, і жоден не простоює. Відзначимо, що реалізація справедливої політики планування не завжди призводить до зменшення середнього часу відгуку. Іноді для цього потрібно зробити систему менш справедливою.

1.2. Застосовність принципів планування

Принципи планування потоків застосовні насамперед до багатопотокових систем із реалізацією схеми 1:1 (тут плануються винятково потоки ядра), а також до систем з реалізацією моделі процесів. В останньому випадку замість терміна «потік» можна вживати термін «процес», а інформацію, необхідну для планування, зберігати в структурах даних процесів. Складніші принципи планування використовують у багатопотокових системах, для яких кількість потоків користувача не збігається з кількістю потоків ядра (схеми 1:Мі *M:N).*Для них потрібні два планувальники: один для роботи на рівні ядра, інший — у режимі користувача.

2.Алгоритми планування

Як ми вже знаємо, алгоритм планування дає змогу короткотерміновому планувальникові вибирати з готових до виконання потоків той, котрий потрібно виконувати наступним. Можна сказати, що алгоритми планування реалізують політику планування.

Залежно від стратегії планування, яку реалізують алгоритми, їх поділяють на витісняльні та невитісняльні. Витісняльні алгоритми переривають потоки під час їхнього виконання, невитісняльні — не переривають. Деякі алгоритми відповідають лише одній із цих стратегій, інші можуть мати як витісняльний, так і невитісняльний варіанти реалізації.

Питання для самоконтролю

1. Що таке максимальна пропускна здатність?
2. Що таке обмежені можливості процесора (CPUbound)?
3. Назвіть критерії оцінки політики планування.

Теми для рефератів

1. Види планування.
2.Програмний інтерфейс планування
3. Механізми і політика планування.

Список рекомендованих джерел

1. Шеховцов В.А. Операційні системи. – К.:Видавнича група BHV, 2005. – 576с.:іл.
2. Рихтер Д. Windows для профессионалов: Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows. – М.: Русская редакция, 2001. – 752с.
3. Харт Дж.В. Системное программирование в бреде Win32 – М.: Вильямс, 2001 – 464с.
4. Рихтер Джеффри. Windows для проффесионалов: Програмирование для Windows 95 и WindowsNT 4.0. на базе Win32/API. – М.: Изд.отдел «Русская редакция», 1997. – 712с.
5. Румянцев П.В. Азбука программирования в Win32 API. – М.: Радио и связь, 1998. – 272с.

Самостійна робота №7

Тема: Взаємодія потоків. Міжпроцесова взаємодія.

Мета: Охарактеризувати процес взаємодії потоків та міжпроцесової взаємодії

План

1. **Основні принципи взаємодії потоків.**
2. **Міжпроцесова взаємодія.**
   - **2.1. Види міжпроцесової взаємодії.**
     - **2.1.1. Методи розподілювальної пам`яті.**
     - **2.1.2. Методи передавання повідомлень**
     - **2.1.3. Технологія відображуваної пам`яті.**
     - **2.1.4. Особливості міжпроцесової взаємодії.**

Теоретичні відомості

1. Основні принципи взаємодії потоків

Потоки, які виконуються в рамках процесу паралельно, можуть бути незалежними або взаємодіяти між собою.

Потік є незалежним, якщо він не впливає на виконання інших потоків процесу, не зазнає впливу з їхнього боку, та не має з ними жодних спільних даних. Його виконання однозначно залежить від вхідних даних і називається детермінованим.

Усі інші потоки є такими, що взаємодіють. Ці потоки мають дані, спільні з іншими потоками (вони перебувають в адресному просторі їхнього процесу). їх виконання залежить не тільки від вхідних даних, але й від виконання інших потоків, тобто вони є *недетермінованими* (далі розглянемо докладно приклади такої недетермінованості).

Результати виконання незалежного потоку завжди можна повторити, чого не можна сказати про потоки, що взаємодіють.

Дані, які є загальними для кількох потоків, називають спільно використовуваними даними (shareddata). Це — найважливіша концепція багатопотокового програмування. Усякий потік може в будь-який момент часу змінити такі дані. Механізми забезпечення коректного доступу до спільно використовуваних даних називають механізмами синхронізації потоків.

Працювати із незалежними потоками простіше, ніж із тими, що взаємодіють. Програміст може не враховувати того, що одночасно з таким потоком виконуються інші, а також не звертати уваги на стан спільно використовуваних даних, з якими працює потік.

Проте обійтися безреалізації взаємодії потоків неможливо з кількох причин. Необхідно організовувати спільне використання інформації під час роботи з потоками. Наприклад, користувачі бази даних або веб-сервера можуть захотіти одночасно виконати запити на отримання однієї й тієї самої інформації, і система має забезпечити її паралельне отримання потоками, що обслуговують цих користувачів.

♦ Коректна реалізація такої взаємодії та використання відповідних алгоритмів можуть значно прискорити обчислювальний процес на багатопроцесорних системах. При цьому задачі розділяють на підзадачі, які

виконують паралельно на різних процесорах, а потім їхні результати збирають разом для отримання остаточного розв'язання. Таку технологію називають *технологією паралельних обчислень.*

♦ У задачах, що вимагають паралельного виконання обчислень та операцій введення-виведення, потоки, що виконують введення-виведення, повинні мати можливість подавати сигнали іншим потокам із завершенням своїх операцій. Подібна організація дає змогу розбивати задачі на окремі виконувані модулі, оформлені як окремі потоки, при цьому вихід одного модуля може бути входом для іншого, а також підвищується гнучкість системи, оскільки окремі модулі можна міняти, не чіпаючи інших.

Необхідність організації паралельного виконання потоків, що взаємодіють, потребує наявності механізмів обміну даними між ними і забезпечення їхньої синхронізації.

2. Міжпроцесова взаємодія

Дотепер ми розглядали взаємодію потоків одного процесу. Головною особливістю цієї взаємодії є простота технічної реалізації обміну даними між ними — усі потоки одного процесу використовують один адресний простір, а отже, можуть вільно отримувати доступ до спільно використовуваних даних, ніби вони є їх власними. Оскільки технічних труднощів із реалізацією обміну даними тут немає, основною проблемою, яку потрібно вирішувати в цьому випадку, є синхронізація потоків.

З іншого боку, кожен потік виконується в рамках адресного простору деякого процесу, тому часто постає задача організації взаємодії між потоками різних процесів. Ідеться власне про міжпроцесову взаємодію (interprocesscommunication, IPC). Ця технологія з'явилася задовго до поширення багатопотоковості.

Для потоків різних процесів питання забезпечення синхронізації теж є актуальними, але вони в більшості випадків не грунтуються на понятті спільно використовуваних даних (такі дані за замовчуванням для процесів відсутні). Крім того, додається досить складна задача забезпечення обміну даними між захищеними адресними просторами. Підходи до її розв'язання визначають різні види міжпроцесової взаємодії.

2.1. Види міжпроцесової взаємодії

Реалізація міжпроцесової взаємодії здійснюється трьома основними методами: *передавання повідомлень, розподілюваної пам'яті* та *відображуваної пам'яті.*

Сигнали були розглянуті раніше, тому що їхнє використання не зводиться тільки до організації IPC (синхронні сигнали є засобом оповіщення процесу про виняткову ситуацію); без них складно пояснити ряд базових понять керування процесами (наприклад, очікування завершення процесу).

2.1.1. Методи розподілюваної пам’яті

Методи розподілюваної пам’яті (sharedmemory) дають змогу двом процесам обмінюватися даними через загальний буфер пам’яті. Перед обміном даними кожний із тих процесів має приєднати цей буфер до свого адресного простору з використанням спеціальних системних викликів (перед цим перевіряють права). Буфер доступний у системі за допомогою процедури іменування, термін його існування звичайно обмежений часом роботи всієї системи. Дані в ньому фактично є спільно використовуваними, як і для потоків. Жодних засобів синхронізації доступу до цих даних розподілювана пам’ять не забезпечує, програміст, так само, як і при розробці багатопотокових застосувань, має організовувати її сам.

2.1.2. Методи передавання повідомлень

В основі методів передавання повідомлень (message passing) лежать різні технології, що дають змогу потокам різних процесів (які, можливо, виконуються на різних комп’ютерах) обмінюватися інформацією у вигляді фрагментів даних фіксованої чи змінної довжини, котрі називають повідомленнями (messages). Процеси можуть приймати і відсилати повідомлення, при цьому автоматично забезпечується їхнє пересилання між адресними просторами процесів одного комп’ютера або через мережу. Важливою особливістю технологій передавання повідомлень є те, шо вони не спираються на спільно використовувані дані — процеси можуть обмінюватися повідомленнями, навіть не знаючи один про одного.

2.1.3. Технологія відображуваної пам’яті

Ще однією категорією засобів міжпроцесової взаємодії є відображувана пам’ять (mapped memory). У ряді ОС відображувана пам’ять є базовим системним механізмом, на якому грунтуються інші види міжпроцесової взаємодії та системні вирішення. Звичайно відображувану пам’ять використовують у поєднанні з інтерфейсом файлової системи, в такому разі говорять про файли, відображувані у пам’ять (memory-mapped files).

Ця технологія зводиться до того, що за допомогою спеціального системного виклику (зазвичай це mmapO) певну частину адресного простору процесу одно-значно пов’язують із вмістом файла. Після цього будь-яка операція записування в таку пам’ять спричиняє зміну вмісту відображеного файла, яка відразу стає доступною усім застосуванням, що мають доступ до цього файла. Інші застосування теж можуть відобразити той самий файл у свій адресний простір і обмінюватися через нього даними один з одним.

2.1.4. Особливості міжпроцесової взаємодії

Тепер можна порівняти характеристики міжпроцесової взаємодії із характеристиками взаємодії потоків одного процесу.

♦ Проблема організації обміну даними є актуальною тільки для міжпроцесової взаємодії, оскільки потоки обмінюються даними через загальний адресний простір. Обмін даними між потоками схожий на використання розподілюваної пам’яті, але не потребує підготовчих дій.

♦ Проблема синхронізації доступу до спільно використовуваних даних є актуальною для взаємодії потоків і для міжпроцесової взаємодії із використанням розподілюваної пам’яті. Використання механізму передавання повідомлень не грунтується на спільно використовуваних даних.

Питання для самоконтролю

1. Які дані називають спільно використовуваними(shareddata)?
2. Які основні методи реалізації міжпроцесової взаємодії?

Теми для рефератів

1. Критичні секції та блокування.
2. Міжпроцесова взаємодія на базі спільної пам`яті.
3. Основи передавання повідомлень.

Список рекомендованих джерел

1. Шеховцов В.А. Операційні системи. – К.:Видавнича група BHV, 2005. – 576с.:іл.
2. Рихтер Д. Windows для профессионалов: Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows. – М.: Русская редакция, 2001. – 752с.
3. Харт Дж.В. Системное программирование в бреде Win32 – М.: Вильямс, 2001 – 464с.
4. Рихтер Джеффри. Windows для проффесионалов: Програмирование для Windows 95 и WindowsNT 4.0. на базе Win32/API. – М.: Изд.отдел «Русская редакция», 1997. – 712с.
5. Румянцев П.В. Азбука программирования в Win32 API. – М.: Радио и связь, 1998. – 272с.

Самостійна робота №8

Тема: Проблеми багатопотокових застосувань.
Мета: Познайомитись з проблемами багатопотокових застосувань.

План

1. **Інверсія пріоритету**
2. **Ступінь деталізації блокувань**
3. **Відмови в обслуговуванні**

Теоретичні відомості

Під час розробки багатопотокових застосувань необхідно враховувати, крім описаних раніше, інші важливі джерела можливих проблем. У цьому розділі розглянемо три з них: інверсію пріоритету, ступінь деталізації блокувань і відмову в обслуговуванні.

1. Інверсія пріоритету

Припустимо, що в системі є потоки із різним пріоритетом, які виконуються паралельно і мають спільно використовувати ресурси. Синхронізацію доступу до ресурсів забезпечують блокування. Розглянемо три потоки: 7# — з високим пріоритетом, TL — із низьким і Тм, пріоритет якого менший, ніж у Тн, але більший, ніж у Tjr. Припустимо, що Тн і 7} спільно використовують деякий ресурс. Можлива така ситуація

1. TL заблоковує ресурс.

2.ТЦ виходить зі стану очікування, переходить у стан готовності до виконання і намагається отримати доступ до ресурсу. Оскільки ресурс зайнятий потоком TL, Tff не може отримати керування і переходить у стан очікування.

3.До того, як Ті звільнить ресурс, J\і переходить у стан готовності і як процес із вищим пріоритетом починає виконуватися, витісняючи TL.

4.Тепер, оскільки TL не виконується, він не може звільнити ресурс, а отже, процес Тн продовжує залишатися у стані очікування доти, поки процес Тм (із нижчим пріоритетом) продовжуватиме своє виконання.

Таку ситуацію називають інверсією пріоритету (priority inversion) . Потік Тн продовжуватиме очікування доти, поки 7} не звільнить ресурс, тому, якщо за час виконання Тм у стан готовності перейдуть нові процеси проміжного пріоритету, вони не дозволять TL повернути керування, і таке очікування триватиме ще довше.

Інверсія пріоритету, ж і стан змагання, є прикладом помилки, що проявля¬ється тільки в певних умовах, котрі практично неможливо наперед відтворити. Класичним прикладом інверсії пріоритету можна вважати документовану помилку ОС реального часу автоматичного апарата Pathfinder, запущеного для дослідження поверхні Марса в 1997 році. Внаслідок цієї помилки головний потік керування апаратом призупинявся на недопустимий час і ОС апаратно перевантажувалася прямо на Марсі. Проблема полягала у відсиланні головним потоком інформації низькопріоритетній задачі обробки наукових даних через канал. Виходило так, що в той момент, коли ця задача утримувала блокування на канал (призупиняючи тим самим головний потік), її витісняли задачі проміжного пріоритету.

Для розв’язання проблеми інверсії пріоритету можна тимчасово підвищувати пріоритет Ті до пріоритету Тн на той час, поки він утримує блокування на ресурс, потрібний Тн (спадкування пріоритету). У цьому разі до розблокування витиснення потоку TL потоком Тм буде неможливе. Для реалізації цього підходу потрібно пов’язати із кожним ресурсом значення Ргаах — максимальний пріоритет потоку', якому потрібний цей ресурс, і для кожного потоку, що блокуватиме цей ресурс, підвищувати пріоритет до значення Ртах на час, поки він утримує це блокування.

2.Ступінь деталізації блокувань

Ступінь деталізації блокування (lockgranularity) визначає обсяг даних, захищених блокуваннями; що вище цей ступінь, то менше даних захищає кожне блокування. Помилки у визначенні ступеня деталізації блокувань у багатопотокових програмах можуть призвести до зниження продуктивності, появи взаємних блокувань та інших проблем.

Що менше детальними є блокування, то менший ступінь паралелізму припустимий у застосуванні (кількість потоків, які можуть одночасно працювати з даними). Це може негативно впливати на продуктивність застосування. Граничним випадком є стратегія «одного великого блокування», коли весь код застосування або модуля виконують разом з утриманням одного м’ютекса, тобто роботу фактично виконують в однопотоковому режимі.

Якщо в нас, навпаки, ступінь деталізації блокувань високий (є багато дрібних блокувань), ступінь паралелізму теж виявиться високим. При цьому потоки рідше блокують один одного, але застосування стає складнішим, додаються витрати на утримування блокувань і зростає ризик взаємного блокування. Якщо блокування «дроблять», зупинитися потрібно тоді, коли черги потоків, що очікують на м’ютексах, більшу частину часу будуть порожніми і додавання нових блокувань нічого не дасть.

Є різні адаптивні схеми. Наприклад, можна починати із дрібних блокувань, які, коли захопити їх багато, перетворюються на одне велике (ескалація блокувань). Можна, навпаки, використати великі блокування, а під час виникнення проблем із паралелізмом розбивати їх на дрібні.

3.Відмови в обслуговуванні

Відмовавобслуговуванні (denialofservice) — ситуація, заякоїпотікувійдевне-скінченнийциклусерединікритичноїсекції,
ітакимчиномназавждизаблокуєвсііншіпотокинавходівцюкритичнусекцію,
зупинивши, зрештою, застосуванняуцілому.

Простого засобу розв’язання цієї проблеми немає, як і простої відповіді на низку запитань аналогічного плану, пов’язаних, наприклад, із крахом програми всередині критичної секції тощо. Рекомендовано ретельно розробляти код критичних секцій, щоб таких помилок не допускати. Тут можуть бути корисними

неблокувальні виклики типу mutex_trylock() і стратегії, подібні до двофазного блокування і сторожового потоку'.

Питання для самоконтролю

1. Назвіть адаптивні схеми деталізації блокувань
2. В чому полягає відмова в обслуговуванні.

Теми для рефератів

1. Використання потоків для організації паралельних обчислень
2. Реалізація моделювання динамічних систем

Список рекомендованих джерел

1. Шеховцов В.А. Операційні системи. – К.:Видавнича група BHV, 2005. – 576с.:іл.
2. Рихтер Д. Windows для профессионалов: Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows. – М.: Русская редакция, 2001. – 752с.
3. Харт Дж.В. Системное программирование в бреде Win32 – М.: Вильямс, 2001 – 464с.
4. Рихтер Джеффри. Windows для проффесионалов: Програмирование для Windows 95 и WindowsNT 4.0. на базе Win32/API. – М.: Изд.отдел «Русская редакция», 1997. – 712с.
5. Румянцев П.В. Азбука программирования в Win32 API. – М.: Радио и связь, 1998. – 272с.

Самостійна робота №9

Тема: Реалізація керування основною пам`яттю Linux

Мета: Перевірити ступінь засвоєння знань керування основною пам`яттю Linux

План

1. **Використання сегментації в Linux.Формування логічнихадрес.**
2. **Сторінкова адресація в Linux.**
3. **Розташування ядра у фізичній пам'яті.**
4. **Особливості адресації процесів і ядра.**
5. **Використання асоціативної пам'яті.**

Теоретичні відомості

1. Використання сегментації в Linux.Формування логічних адрес

Як уже зазначалося, необхідність підтримки сегментації призводить до того, що програми стають складнішими, оскільки задача виділення сегментів і формування коректних логічних адрес лягає на програміста. Цю проблему в Linux вирішують доволі просто — ядро практично не використовує засобів підтримки сегментації архітектури IA-32. У системі підтримують мінімальну кількість сегментів, без яких неможлива коректна адресація пам'яті процесором (сегменти коду і даних ядра та режиму користувача). Код ядра і режиму користувача спільно використовує ці сегменти.

Сегменти коду використовують під час формування логічних адрес коду (для виклику процедур тощо); такі сегменти позначають як доступні для читання і виконання. Сегменти даних призначені для формування логічних адрес даних (глобальних змінних, стека тощо) і позначаються як доступні для читання і записування. Сегменти режиму користувача доступні з режиму користувача, сегменти ядра — тільки з режиму ядра.

Усі сегменти, які використовуються у Linux, визначають межу зсуву, що дає змогу створити в рамках кожного з них 4 Гбайт логічних адрес. Де означає, що Linuxфактично передає всю роботу з керування пам'яттю на рівень перетворення між лінійними і фізичними адресами (оскільки кожна логічна адреса відповідає лінійній).

Далі в цьому розділі вважатимемо логічні адреси вже сформованими (на базі відповідного сегмента) і перетвореними на лінійні адреси.

2. Сторінкова адресація в Linux

У ядрі Linuxверсії 2.4 використовують трирівневу організацію таблиць сторінок. Підтримуються три типи таблиць сторінок: *глобальний*(PageGlobalDirectory, PGD); *проміжний каталог сторінок*(PageMiddleDirectory, PMD); *таблиця сторінок* (PageTable).

Кожний глобальний каталог містить адреси одного або кількох проміжних каталогів сторінок, а ті, своєю чергою, — адреси таблиць сторінок. Елементи таблиць сторінок (PTE) вказують на фрейми фізичної пам'яті.

Кожний процес має свій глобальний каталог сторінок і набір таблиць сторінок. Під час перемикання контексту Linuxзберігає значення регістра сг3 у керуючому блоці процесу, що передає керування, і завантажує в цей регістр значення з керуючого блоку процесу, що починає виконуватися. Отже, коли процес починає

виконуватися, пристрій сторінкової підтримки вже посилається на коректний набір таблиць сторінок.

Тепер розглянемо роботу цієї трирівневої організації для архітектури IA-32, яка дає можливість мати тільки два рівні таблиць. Насправді ситуація досить проста — проміжний каталог таблиць оголошують порожнім, водночас його місце в ланцюжку покажчиків зберігають для того, щоб той самий код міг працювати для різних архітектур. У цьому разі PGDвідповідає каталогу сторінок IA-32 (його елементи містять адреси таблиць сторінок), а під час роботи із покажчиком на PMDнасправді працюють із покажчиком на відповідний йому елемент PGD, відразу отримуючи доступ до таблиці сторінок. Між таблицями сторінок Linuxі таблицями сторінок IA-32 завжди дотримується однозначна відповідність.

Для платформо-незалежного визначення розміру сторінки в Linuxвикористовують системний виклик getpagesizeO:

finclude<un1std.h>

printf("Po3Mip сторінки: M\n“. getpagesizeO):

3. Розташування ядра у фізичній пам’яті

Ядро Linux завантажують у набір зарезервованих фреймів пам’яті, які заборонено вивантажувати на диск або передавати процесу користувача, що захищає код і дані ядра від випадкового або навмисного ушкодження.

Завантаження ядра починається із другого мегабайта пам’яті (перший мегабайт пропускають, тому що в ньому є ділянки, які не можуть бути використані, наприклад відеопам’ять текстового режиму, код B108 тощо). Із ядра завжди можна визначити фізичні адреси початку та кінця його коду і даних.

4. Особливості адресації процесів і ядра

Лінійний адресний простір кожного процесу поділяють на дві частини: перші 3 Гбайт адрес використовують у режимі ядра та користувача; вони відображають захищений адресний простір процесу; решту 1 Гбайт адрес використовують тільки в режимі ядра.

Елементи глобального каталогу процесу, що визначають адреси до 3 Гбайт, можуть бути задані самим процесом, інші елементи мають бути однаковими для всіх процесів і задаватися ядром.

Потоки ядране використовують елементів глобального каталогу першого діапазону. На практиці, коли відбувається передавання керування потоку ядра, не змінюється значення регістра сг3, тобто потік ядра використовує таблиці сторінок процесу користувача, що виконувався останнім (оскільки йому потрібні тільки елементи, доступні в режимі ядра, а вони в усіх процесах користувача однакові).

Адресний простір ядра починається із четвертого гігабайта лінійної пам’яті. Для прямого відображення на фізичні адреси доступні перші 896 Мбайт цього простору (128 Мбайт, що залишилися, використовується переважно для динамічного розподілу пам’яті ядром).

5. Використання асоціативної пам’яті

Під час роботи з асоціативною пам’яттю основне завдання ядра полягає у зменшенні потреби її очищення. Для цього вживають таких заходів.

♦ Під час планування невелику перевагу має процес, який використовує той

самий набір таблиць сторінок, що й процес, який повертає керування (під час перемикання між такими процесами очищення кеша трансляції не відбувається).

♦ Реалізація потоків ядра, котрі використовують таблиці сторінок останнього процесу, теж призводить до того, що під час перемикання між процесом і потоком ядра очищення не відбувається.

Питання для самоконтролю

1. Перелічіть відмінності в реалізації та використанні сегментів даних і сегментів коду.
2. Охарактеризуйте сегменти, без яких неможлива коректна адресація пам’яті процессором.

Теми для рефератів

1. Структура адресного простору процесів і ядра
2. Асоціативна пам’ять

Список рекомендованих джерел

1. Шеховцов В.А. Операційні системи. – К.:Видавнича група BHV, 2005. – 576с.:іл.
2. Рихтер Д. Windows для профессионалов: Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows. – М.: Русская редакция, 2001. – 752с.
3. Харт Дж.В. Системное программирование в бреде Win32 – М.: Вильямс, 2001 – 464с.
4. Рихтер Джеффри. Windows для проффесионалов: Програмирование для Windows 95 и WindowsNT 4.0. на базе Win32/API. – М.: Изд.отдел «Русская редакция», 1997. – 712с.
5. Румянцев П.В. Азбука программирования в Win32 API. – М.: Радио и связь, 1998. – 272с.

Самостійна робота №10

Тема: Реалізація керування основною пам`яттю Windows

Мета: Перевірити ступінь засвоєння знань керування основною пам`яттю Windows

План

1. Сегментація у WindowsXP
2. Сторінкова адресація у WindowsXP
3. Особливості адресації процесів і ядра
4. Структура адресного простору процесів і ядра

Теоретичні відомості

1.Сегментація у WindowsXP

Система WindowsXP використовує загальні сегменти пам’яті подібно до того, як це робиться в Linux. Для всіх сегментів у програмі задають однакові значення бази і межі, тому роботу з керування пам’яттю аналогічним чином передають на рівень лінійних адрес (які є зсувом у цих загальних сегментах).

2.Сторінкова адресація у WindowsXP

Під час роботи з лінійними адресами у WindowsXP використовують дворівневі таблиці сторінок, повністю відповідні архітектурі IA-32 (див. Розділ 8.3.4). У кожного процесу є свій каталог сторінок, кожен елемент якого вказує на таблицю сторінок. Таблиці сторінок усіх рівнів містять по 1024 елементи таблиці сторінок, кожний такий елемент вказує на фрейм фізичної пам’яті. Фізичну адресу каталогу сторінок зберігають у блоці KPROCESS.

Розмір лінійної адреси, з якою працює система, становить 32 біти. З них 10 біт відповідають адресі в каталозі сторінок, ще 10 – це індекс елемента в таблиці, останні 12 біт адресують конкретний байт сторінки (і є зсувом).

Розмір елемента таблиці сторінок теж становить 32 біти. Перші 20 біт адресують конкретний фрейм (і використовуються разом із останніми 12 біт лінійної адреси), а інші 12 біт описують атрибути сторінки (захист, стан сторінки в пам’яті, який файл підкачування використовує). Якщо сторінка не перебуває у пам’яті, то в перших 20 біт зберігають зсув у файлі підкачування.

Для платформо-незалежного визначення розміру сторінки у Win32 API використовують універсальну функцію отримання інформації про систему GetSystemlnfoO:

```
SYSTEM_INF0 info; // структура для отримання інформації про
систему GetSystemlnfoC&info):
printf(“Po3Mip сторінки: £d\n”, info.dwPageSize):
```

3.Особливості адресації процесів і ядра

Лінійний адресний простір процесу поділяється на дві частини: перші 2 Гбайт адрес доступні для процесу в режимі користувача і є його захищеним адресним простором; інші 2 Гбайт адрес доступні тільки в режимі ядра і відображають систем-

ний адресний простір.

Зазначимо, що таке співвідношення між адресним простором процесу і ядра відрізняється від прийнятого в Linux(3 Гбайт для процесу, 1 Гбайт для ядра).

Деякі версії \УїІк1оу/5 ХР дають можливість задати співвідношення 3 Гбайт/1 Гбайтпід час завантаження системи.

4.Структура адресного простору процесів і ядра

В адресному просторі процесу можна виділити такі ділянки:

- перші 64 Кбайт (починаючи з нульової адреси) — це спеціальна ділянка, доступ до якої завжди спричиняє помилки;
- усю пам'ять між першими 64 Кбайт і останніми 136 Кбайт (майже 2 Гбайт) може використовувати процес під час свого виконання;
  далі розташовані два блоки по 4 Кбайт: блоки оточення потоку (ТЕВ) і процесу (РЕВ) (див.розділ 3);
- наступні 4 Кбайт - ділянка пам'яті, куди відображаються різні системні дані (системний час, значення лічильника системних годин, номер версії системи), тому для доступу до них процесу не потрібно перемикатися в режим ядра;
- останні 64 Кбайт використовують для запобігання спробам доступу за межі адресного простору процесу (спроба доступу до цієї пам'яті дасть помилку).

Системний адресний простір містить велику кількість різних ділянок. Найважливіші з них наведено нижче.

- Перші 512 Мбайт системного адресного простору використовують для завантаження ядра системи.
- 4 Мбайт пам'яті виділяють під каталог сторінок і таблиці сторінок процесу.
- Спеціальну ділянку пам'яті розміром 4 Мбайт, яку називають гіперпростором (Ііурешрасе), використовують для відображення різних структур даних, специфічних для процесу, на системний адресний простір (наприклад, вона містить список сторінок робочого набору процесу).
  512 Мбайт виділяють під системний кеш.
- У системний адресний простір відображаються спеціальні ділянки пам'яті — вивантажуваний пул і невивантажуваний пул, які розглянемо в розділі 10.
- Приблизно 4 Мбайт у самому кінці системного адресного простору виділяють під структури даних, необхідні для створення аварійного образу пам'яті, а також для структур даних НАЬ.

Питання для самоконтролю

1. У якому блоці зберігають фізичну адресу каталогу сторінок?
2. На які частини поділяється лінійний адресний простір?

Теми для рефератів

1. Реалізація динамічного керування пам'яттю в Windows XP.
2. Особливості кешування у Windows XP.

Список рекомендованих джерел

1. Шеховцов В.А. Операційні системи. – К.:Видавнича група BHV, 2005. – 576с.:іл.
2. Рихтер Д. Windows для профессионалов: Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows. – М.: Русская редакция, 2001. – 752с.
3. Харт Дж.В. Системное программирование в бреде Win32 – М.: Вильямс, 2001 – 464с.
4. Рихтер Джеффри. Windows для проффесионалов: Програмирование для Windows 95 и WindowsNT 4.0. на базе Win32/API. – М.: Изд.отдел «Русская редакция», 1997. – 712с.
5. Румянцев П.В. Азбука программирования в Win32 API. – М.: Радио и связь, 1998. – 272с.

Самостійна робота №11

Тема: Поняття підкачування. Проблеми реалізації підкачування сторінок.
Мета: Виробити вміння та навички про підкачування та реалізацію підкачування сторінок

План

1.Поняття підкачування

2.Завантаження сторінок на вимогу. Особливості підкачуваннясторінок

2.1.Апаратна підтримка підкачування сторінок

2.2.Поняття сторінкового переривання

2.3.Продуктивність завантаження на вимогу

Теоретична частина

1. Поняття підкачування

Описана технологія повного завантаження і вивантаження процесів традиційно називалася підкачуванням або простіш підкачуванням, але тут вживатимемо цей термін у ширшому значенні. У сучасних ОС під підкачуванням (swapping) розуміють увесь набір технологій, які здійснюють взаємодію із диском під час реалізації віртуальної пам’яті, щоб дати можливість кожному процесу звертатися до великого діапазону логічних адрес за рахунок використання дискового простору.Розглянемо загальні принципи підкачування. Як відомо, зняття вимоги непе-рервності фізичного простору, куди відображається адресний простір процесу, і можливість переміщення процесу в пам'яті під час його виконання дає змогу не тримати одночасно в основній пам’яті всі блоки пам’яті (сторінки або сегменти), які утворюють адресний простір цього процесу. Під час завантаження процесу в основну пам’ять у ній розміщають лише кілька його блоків, які потрібні для по¬чатку роботи. Частину адресного простору процесу, що у конкретний момент часу відображається на основну пам’ять, називають резидентною множиною процесу (resident set). Поки процес звертається тільки до пам’яті резидентно!' множини, виконання процесу не переривають. Як тільки здійснюється посилання на блок, що перебуває за межами резидентно! множини (тобто відображений на диск), відбувається апаратне переривання. Оброблювач цього переривання призупиняє процес і запускає дискову операцію читання потрібного блоку із диска в основну пам’ять. Коли блок зчитаний, операційну систему сповіщають про це, після чого процес переводять у стан готовності й, зрештою, поновлюють, після чого він про-довжує свою роботу, ніби нічого й не сталося; на момент його поновлення потрібний блок уже перебуває в основній пам’яті, де процес і розраховував його знайти.Реалізація підкачування використовує правило «дев’яносто до десяти». Ідеальною реалізацією керування пам’яттю є надання кожному процесові пам’яті, за розміром порівнянної із жорстким диском, а за швидкістю доступу — з основною пам’яттю. Оскільки за правилом «дев’яносто до десяти» на 10 % адресного простору припадає 90 % посилань на пам’ять, як деяке наближення до ідеальної реа¬лізації можна розглядати такий підхід: зберігати ці 10

% в основній пам’яті, а інший адресний простір відображати на диск. Як показано на рис. 9.1, частіше викори-стовують сторінки 0,3,4,6, тому їхній вміст зберігають в основній пам’яті, а сторінки 1, 2, 5, 7 використовують рідше, тому їхній вміст зберігають на диск.Головною проблемою залишається ухвалення рішення про те, які із блоків па-м’ятімають в конкретний момент відображатися на основну пам’ять, а які — на диск.

Внаслідок використання технології підкачування кількість викокзшаних процесів збільшується (для кожного з них в основній пам’яті перебуватиме тільки частина блоків). Підкачування дає також змогу виконувати процеси, які за розміром більші, ніж основна пам’ять (для таких процесів у різні моменти часу в основну пам’ять відображатимуться різні блоки).

Ця технологія діє у припущенні, що не всі сторінки процесу мають завантажуватися у пам’ять негайно. Завантажуються тільки ті, що необхідні для початку його роботи, інші — коли стають потрібні. Процес переносу сторінки із диска у пам’ять називають завантаженням із диска(swappingin), процес переносу сторінки із пам’яті на диск — вивантаженням на диск(swappingout).

2. Завантаження сторінок на вимогу. Особливості підкачування сторінок

Базовий підхід, який використовують під час реалізації підкачування сторінок із диска, називають технологією завантаження сторінок на вимогу (demand paging).

Ця технологія діє у припущенні, що не всі сторінки процесу мають завантажу-ватися у пам’ять негайно. Завантажуються тільки ті, що необхідні для початку його роботи, інші — коли стають потрібні. Процес переносу сторінки із диска у пам’ять називають завантаженням із диска (swapping in), процес переносу сто¬рінки із пам’яті на диск — вивантаженням на диск (swapping out).

2.1. Апаратна підтримка підкачування сторінок

Для організації апаратної підтримки підкачування кожний елемент таблиці сторінок має містити спеціальний біт — біт присутності сторінки у пам’яті P. Коли він дорівнює одиниці, то це означає, що відповідна сторінка завантажена в основну пам’ять, і їй відповідає деякий фрейм. Якщо біт присутності сторінки дорівнює нулю, це означає, що дана сторінка перебуває на диску і має бути завантажена в основну пам’ять перед використанням.

Для сторінки може бути заданий біт модифікації M. Його спочатку (під час завантаження сторінки із диска) покладають рівним нулю, потім — одиниці, якщо сторінку модифікують, коли вона завантажена у фрейм основної пам’яті. Наявність такого біта дає змогу оптимізувати операції вивантаження сторінок на диск. Коли намагаються вивантажити на диск вміст сторінки, для якої біт M дорівнює нулю, це означає, що записування на диск можна проігнорувати, бо вміст сторін¬ки не змінився від часу завантаження у пам’ять. У розділі 9.5 побачимо, як цей біт можна використати для реалізації заміщення сторінок (визначення сторінки, яку потрібно вивантажити на диск у разі нестачі фреймів фізичної пам’яті). У таблиці сторінок архітектури IA-32 даний біт називають Dirty.

Якщо процес працює тільки зі сторінками, для яких біт P дорівнює одиниці (його резидентна множина не змінюється), складається враження, що всі його

сторінки є у пам’яті, хоча насправді це може бути і не так (просто сторінок, яких немає у пам’яті, у цьому разі взагалі не використовують).

2.2. Поняття сторінкового переривання

Коли процес робить спробу доступу до сторінки, для якої біт Р дорівнює нулю, то, як ми вже бачили під час вивчення загальної стратегії підкачування, відбувається апаратне переривання. Його називають сторінковим перериванням або сторінковою відмовою (page fault). ОС має обробити це переривання.

♦ Сторінку перевіряють на доступність для цього процесу (діапазон доступної пам’яті зберігається у структурі даних процесу).

♦ Якщо сторінка недоступна, процес переривають, якщо доступна, знаходять вільний фрейм фізичної пам’яті та ставлять у чергу дискову операцію читан¬ня потрібної сторінки в цей фрейм.

♦ Після читання модифікують відповідний запис таблиці сторінок (біт присутності покладають рівним одиниці).

♦ Перезапускають інструкцію, що викликала апаратне переривання.

Тепер процес може отримати доступ до сторінки, ніби вона завжди була в пам’яті. Процес поновлюють у тому самому стані, в якому він був перед перериванням.

Чисте завантаження сторінок на вимогу зводиться до того, що жодну сторінку не завантажують у пам’ять завчасно. При цьому після запуску процесу жодна з його сторінок не перебуватиме в пам’яті: усі вони будуть довантажуватися внас¬лідок сторінкових переривань за потребою. Є альтернативні підходи (наприклад, попереднє завантаження сторінок), які будуть описані далі.

Зазначимо, що в загальному випадку сторінкове переривання не обов’язково спричиняє підкачування сторінки з диска: воно може бути результатом звертан¬ня до сторінки, що не належить до адресного простору процесу. У цьому разі має бути згенерована помилка. З іншого боку, причиною сторінкового переривання під час чистого завантаження на вимогу може бути звертання до зовсім нової сторінки пам’яті, яка жодного разу не була використана процесом. Оброблювач може зарезервувати новий фрейм, проініціалізувати його (наприклад, заповнити нулями) і поставити йому у відповідність сторінку

2.3. Продуктивність завантаження на вимогу

Реалізація завантаження на вимогу із підкачуванням сторінок із диска може істотно впливати на продуктивність ОС. Зробимо найпростіший розрахунок такого впливу.

Спрощену характеристику, яку використовують для оцінки продуктивності системи із завантаженням сторінок на вимогу, називають ефективніш часом доступу і розраховують за такою формулою:

Тea = (1 - Ppf) x Тта + Pp/ X Tpf,

де Ppf — імовірність сторінкового переривання; Тта — середній час доступу до пам’яті; Tpf — середній час обробки сторінкового переривання.

У сучасних системах час доступу до пам’яті Тта становить від 20 до 200 не. Розрахуємо час обробки сторінкового переривання

Коли відбувається сторінкове переривання, система виконує багато різноманітних дій (генерацію переривання, виклик оброблювача, збереження регістрів

процесора, перевірку коректності доступу до пам'яті, початок операції читання із диска, перемикання контексту на інший процес, обробку переривання від диска про завершення операції читання, корекцію таблиці сторінок тощо). У результаті кількість інструкцій процесора, необхідних для обробки сторінкового переривання, виявляється дуже великою (десятки тисяч), і середній час обробки сягає кількох мілісекунд (одна мілісекунда відповідає мільйону наносекунд).

Припустимо, що величина Тта становить 100 не, Трj— 10 мс, а ймовірність ви-никнення сторінкового переривання — 0,001 (одне переривання на 1000 звертань до пам'яті, цю величину називають рівнем сторінкових переривань — page fault rate). Тоді ефективний час доступу буде рівний:

Теа " (1 - 0,001) X 100 + 0,001 X 10 000 000 = 10 099,9 не.

Як бачимо, внаслідок сторінкових переривань, що виникають з імовірністю одна тисячна, ефективний час доступу виявився в 100 разів більшим, ніж під час роботи з основною пам'яттю, тобто продуктивність системи знизилася в 100 разів. Щоб здобути зниження продуктивності на прийнятну величину, наприклад на 10 %, необхідно, щоб імовірність сторінкового переривання була приблизно одна мільйонна:

Ppf = (0,1 х Тпа) / (Tpf - Tma) - 10/9 999 900 - 1 / 999 990.

Для досягнення прийнятної продуктивності системи із завантаженням сторінок на вимогу рівень сторінкових переривань має бути надзвичайно низьким, тому будь-які поліпшення в алгоритмах керування пам'яттю, що знижують цей рівень, завжди виправдані. По суті, будь-які витрати часу на виконання найскладніших алгоритмів окупатимуться, якщо зменшується кількість сторінкових переривань, оскільки одне таке переривання обробляється повільніше, ніж виконується алгоритм майже будь-якої складності.

Питання для самоконтролю

1. За якою формулою розраховується ефективніший час доступу?
2. Як ОС обробляє переривання?

Теми для рефератів

1. Проблеми реалізації підкачування сторінок
2. Заміщення сторінок

Список рекомендованих джерел

1. Шеховцов В.А. Операційні системи. – К.:Видавнича група BHV, 2005. – 576с.:іл.
2. Рихтер Д. Windows для профессионалов: Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows. – М.: Русская редакция, 2001. – 752с.
3. Харт Дж.В. Системное программирование в бреде Win32 – М.: Вильямс, 2001 – 464с.
4. Рихтер Джеффри. Windows для проффесионалов: Програмирование для Windows 95 и WindowsNT 4.0. на базе Win32/API. – М.: Изд.отдел «Русская редакция», 1997. – 712с.
5. Румянцев П.В. Азбука программирования в Win32 API. – М.: Радио и связь, 1998. – 272с.

Самостійна робота №12

Тема: Пробуксування і керування резидентною множиною.
Мета:Виробити вміння та навички про пробуксування і керування резидентною множиною.

План

1. Поняття пробуксовування
2. Локальність посилань
3. Поняття робочого набору. Модель робочого набору
4. **Аспекти боротьби з пробуксовуванням**

Теоретична частина

1. Поняття пробуксовування

Пробуксовуванням (thrashing) називаютьстанпроцесу, количерезсторінковіпереривиннявінвитрачаєбільшечасунапідкачуваннясторінок, аніжвласненавиконання. У такому стані процес фактично непрацездатний

Пробуксовування виникає тоді, коли процес часто вивантажує із пам'яті сторінки, які йому незабаром знову будуть потрібні. Урезультаті більшу частину часу такі процеси перебувають у призупиненому стані, очікуючи завершення операції введення-виведення для читання сторінки із диска. Отож, на додачу до пам'яті, за розміром порівнянної із диском, отримують пам'ять, порівнянну із диском і за часом доступу.

Назвемо деякі причини пробуксовування.

Процес не використовує пам'ять повторно (для нього не працює правило «дев'яносто до десяти»),

Процес використовує пам'ять повторно, але він надто великий за обсягом, тому його резидентна множина не поміщається у фізичній пам'яті.

Запущено надто багато процесів, тому їхня сумарна резидентна множина не поміщається у фізичній пам'яті.

У перших двох випадках ситуація майже непіддається виправленню, найкраща порада, яку тут можна дати, — це збільшити обсяг фізичної пам'яті. У третьому випадку ОС може почати такі дії: з'ясувати, скільки пам'яті необхідно кожному процесові, і змінити пріоритети планування так, щоб процеси ставилися на вико¬нання групами, вимоги яких до пам'яті можуть бути задоволені; заборонити або обмежити запуск нових процесів.

2. Локальність посилань

Можна створити таку програму, яка постійно звертатиметься до різних сторінок, розкиданих великим адресним простором, генеруючи багато сторінкових переривань. Насправді реальні застосування працюють не так: вони зберігають локальність посилань (locality of reference), коли на різних етапах виконання процес посилається тільки на деяку невелику підмножину своїх сторінок, що є одним із наслідків відомого правила «дев'яносто до десяти».

Набір сторінок, які активно використовуються разом, називають локальніс

(locality). Кажуть, що процес під час свого виконання переміщується від локальності до локальності. Так, у разі виклику функція визначає нову локальність, де будуть посилання на інструкції, локальні змінні та підмножину глобальних змінних. Після виходу із функції її локальність більше не використовують (хоча можна це зробити в майбутньому, якщо функція буде викликана знову). Отже, локальності визначає структура програми та її даних.

Якщо виділено достатньо пам'яті для всіх сторінок поточної локальності, сторінкові переривання до переходу до наступної локальності не генеруватимуться. Якщо пам'яті недостатньо, система перебуватиме у стані пробуксовування.

Завданням керування резидентною множиною є така його динамічна корекція, щоб у будь-який момент часу ця множина давала змоіу розміщувати всі сторінки поточної локальності.

3. Поняття робочого набору. Модель робочого набору

Для того щоб коректно керувати резидентною множиною процесу, необхідно знати той набір сторінок, який йому знадобиться під час виконання. Звичайно, у повному обсязі таке знання реалізоване бути не може, бо це вимагає вміння вгадувати майбутнє. Спроби оцінити потрібний процесу набір сторінок, ґрунтуючись на особливостях використання сторінок у недалекому минулому, привели до концепціїробочого набору (working set).

Найважливішою характеристикою цього підходу є інтервал часу довжиною T, який відлічують від поточного моменту часу назад. Якщо T вимірюють у секундах, то цей інтервал визначає останні T секунд виконання процесу і задає вікно робочого набору. Ширину вікна завжди залишають постійною, а межі його зсуваються із часом; верхньою межею завжди є поточний момент часу, а нижньою поточний момент мінус T.

Під робочим набором процесу розуміють усі сторінки, до яких він звертався у вікні робочого набору. У кожний новий момент часу робочий набір буде різним, оскільки вікно робочого процесу переміщується, сторінки, до яких не було доступу за час T, із набору вилучають, щойно використані сторінки додають у набір Розглянемо, як можна використати концепцію робочого набору:

- Виділяючи пам'ять, можна дати кожному процесові стільки сторінок, щоб у них помістився його робочий набір.
- У разі заміщення сторінок можна вибирати для заміщення переважно ті сторінки, що не належать до робочого набору.
- Під час планування процесів можна не ставити процес на виконання доти, поки всі сторінки його робочого набору не опиняться у пам'яті.

Концепцію робочого набору часто використовують у поєднанні із локальнимзаміщенням і буферизацією сторінок. Наприклад, можна постійно коригувати локальний кеш сторінок процесів для того щоб він приблизно відповідав за розміром його робочому набору. Цим підвищують гнучкість локального заміщення (є можливість дозволити процесам використати через певний час вільні фрейми інших процесів, зберігши при цьому ізольованість одних процесів від інших).

Головним недоліком моделі робочого процесу є складність реалізації. Вона пов'язана насамперед із тим, що вікно робочого процесу постійно переміщується, нові сторінки постійно додають у нього, старі — вилучають. Іншим недоліком є складність вибору величини T (звичайно її підбирають емпірично) і складність організації робочого набору при спільному використанні сторінками різних процесів.

Наведемо один із варіантів реалізації моделювання робочого набору. Із кожним фреймом пов'язують спеціальну характеристику — час простою (idle time). За перериванням від таймера перевіряють, чи встановлено біт R, якщо так, час простою і біт R обнулюють (сторінку залишають у робочому наборі), якщо ні. проміжок часу між скануваннями сторінок додають до часу простою. Коли час простою перевищує T, сторінку виключають із робочого набору. У реальних системах значення T перебуває в межах хвилини.

Зазвичай найбільше сторінкових переривань відбувається під час завантаження процесу у пам'ять, тому багато систем намагаються відстежувати робочі набори.

4. Практичні аспекти боротьби з пробуксовуванням

Метод робочого набору відіграє важливу роль у дослідженнях поведінки процесів у системі. Однак у сучасних ОС його практичне застосування незначне через складність реалізації, а також внаслідок того, що сьогодні пробуксовування в системах трапляється все рідше і програмні методи боротьби з ним втрачають своє значення.

У більшості випадків зниження ймовірності пробуксовування пов'язане із прогресом в апаратному забезпеченні.

♦ У комп'ютерах зараз встановлюють більше основної пам'яті, тому потреба у підкачуванні знижується, а сторінкові переривання трапляються рідше.

♦ Оскільки процесори стають швидшими, процеси виконуються із більшою швидкістю, раніше вивільняючи зайняту пам'ять.

♦ Хоча в системі завжди є процеси, які можуть використати всю доступну пам'ять, кількість процесів, яким достатньо виділеної пам'яті, у середньому збільшується. Фактично, найдієвіший спосіб боротьби із пробуксовуванням полягає у придбанні та встановленні додаткової основної пам'яті.

Питання для самоконтролю

1. Назвіть практичні аспекти боротьби з пробоксуванням.
2. Як можна використовувати концепцію робочого набору.

Теми для рефератів

1. Керування адресним простором процесу
2. Організація заміщення сторінок

Список рекомендованих джерел

1. Шеховцов В.А. Операційні системи. – К.:Видавнича група BHV, 2005.– 576с.:іл.
2. Рихтер Д. Windows для профессионалов: Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows. – М.: Русская редакция, 2001. – 752с.

3. Харт Дж.В. Системное программирование в бреде Win32 – М.: Вильямс, 2001 – 464с.
4. Рихтер Джеффри. Windows для проффесионалов: Програмирование для Windows 95 и Windows NT 4.0. на базе Win32/API. – М.: Изд.отдел «Русская редакция», 1997. – 712с.
5. Румянцев П.В. Азбука программирования в Win32 API. – М.: Радио и связь, 1998. – 272с.

Самостійна робота №13

Тема: Реалізація віртуальної пам'яті в Linux.
Мета:Пояснити процес реалізаціїї віртуальної пам'яті в Linux.

План

1. **Керування адреснимпростором процесу**
2. **Організація заміщення сторінок**

Теоретична частина

1. Керування адресним простором процесу

Адресний простір процесу складається з усіх лінійних адрес, які йому дозволено використовувати. Ядро може динамічно змінювати адресний простір процесу шляхом додавання або вилучення інтервалів лінійних адрес.

Інтервали лінійних адрес зображуються спеціальними структурами даних — *регіонами пам'яті*(memoryregions). Кожний регіон описують *дескриптором регіону*(vm_area_struct), що містить його початкову лінійну адресу, першу адресу після його кінцевої, прапорці прав доступу (читання, запис, виконання, заборона вивантаження на диск тощо). Розмір регіону має бути кратним 4 Кбайт, щоб його дані заповнювали всі призначені фрейми пам'яті. Регіони пам'яті процесу ніколи не перекриваються, ядро намагається з'єднувати сусідні регіони у більший регіон.

Усю інформацію про адресний простір процесу описують спеціальною структурою даних — *дескриптором пам'яті*(memorydescriptor, mmstruct). Покажчик на дескриптор пам'яті процесу зберігають у керуючому блоці процесу (task_struct). Уцьомудескрипторізберігаютьтакуінформацію, якількістьрегіонівпам'яті, покажчикнаглобальнийкаталогсторінок, адресирізнихділянокпам'яті (коду, даних, динамічноїділянки, стека).

Крім того, у дескрипторі пам'яті процесу є покажчики на дві структури даних, призначених *для* забезпечення доступу до регіонів його пам'яті. Перша з них —однозв'язний список усіх регіонів процесу, який використовують для прискорення сканування всього адресного простору, друга — спеціальне бінарне дерево пошуку (що теж об'єднує всі регіони процесу), використовуване для прискорення пошуку конкретної адреси пам'яті.

Для виділення інтервалу вдаються до такого.

1. Відшукують вільний інтервал потрібної довжини (із використанням бінарного дерева).
2. Визначають регіон, що передує цьому інтервалу, і регіон, що йде за ним (коли при цьому отримують регіон, який вже використовує цей інтервал, усе починають спочатку).
3. Роблять спробу об'єднати попередній і наступний регіони із цим інтервалом. Якщо це не вдається, у пам'яті розміщують дескриптор нового регіону. Його ініціалізують адресами початку і кінця інтервалу і додають у список і дерево регіонів процесу.
4. Повертають початкову лінійну адресу нового або об'єднаного регіону.

Уразі вилучення інтервалу лінійних адрес із адресного простору процесу важливо враховувати, що такий інтервал звичайно не збігається із регіоном

пам’яті, він може бути частиною регіону або охоплювати кілька регіонів. Виконують такі дії.

1. Сканують список усіх регіонів пам’яті, які належать процесу, та вилучають із цього списку всі регіони, які перетинаються із заданим інтервалом.
2. Вилучені регіони скорочують або розбивають на два (залежно від характеру перетинання регіону з інтервалом).
3. Вивільняють фізичну пам’ять і змінюють таблиці сторінок процесу для вилученого інтервалу; скорочені регіони повертають назад у список і дерево регіонів процесу.

Із використанням описаних алгоритмів виділення і вилучення інтервалів лінійних адрес реалізовано засоби роботи з адресним простором процесу з режиму користувача. До цих засобів належать системний викликр ( ); його головним призначенням є реалізація відображуваної пам’яті, а також можливість організувати виділення регіонів пам’яті заданого розміру; функції та системні виклики керування динамічною пам’яттю (mallocO та інші).

Уразі спроби доступу до логічної адреси пам’яті, якій у конкретний момент не відповідає фізична адреса, виникає сторінкове переривання. Це може бути результатом помилки програміста, частиною механізму завантаження сторінок на вимогу або технології копіювання під час записування.

Коли переривання відбулося в режимі ядра, поточний процес негайно завершують (причиною переривання в цьому разі може бути передавання невірного параметра в системний виклик або помилка в коді ядра).

Коли переривання відбулося в режимі користувача, визначають, якому із регіонів пам’яті процесу відповідає лінійна адреса, що спричинила це переривання. За відсутності такого регіону процес завершують (відбулося звертання за невірною адресою). Крім того, процес завершують, якщо з’ясовується, що переривання викликане спробою записування в регіон, відкритий для читання. Завершення процесу здійснюють відсиланням йому сигналу SIGSEGV.

2. Організація заміщення сторінок

У цьому розділі розглянемо організацію заміщення сторінок у пам’яті. Відразу ж зазначимо, що в Linuxреалізоване фонове заміщення сторінок, за яке відповідає потік ядра kswapd. Цей потік у колишніх версіях ядра запускався за перериванням від таймера кілька разів за секунду, у ядрі версії 2.6 необхідність його запуску визначає наявність вільних сторінок пам’яті у системі.

Організація заміщення сторінок у Linuxґрунтується на їх буферизації. Організовують два списки сторінок: список активних (active_list) — містить сторінки, які використовують процеси і визначає робочий набір процесів; список неактивних (inactive_list) — містить сторінки, які не так важливі для процесів (не використовуються в цей момент часу). Модифікована сторінка перебуває в списку неактивних якийсь час, перш ніж її збережуть на диску.

Нові сторінки додають у початок списку неактивних сторінок. За нестачі пам’яті частину сторінок переміщують з кінця списку активних сторінок у початок списку неактивних, а потім починають вивільнення сторінок із кінця списку неактивних сторінок

Для визначення робочого набору процесу застосовують прапорець використання сторінки (PG_referenced). Нові сторінки створюють із PG_referenced= 0. Коли процесор зчитує сторінку вперше, прапорець покладають рівним одиниці. Якщо потрібно задати PGreferencedрівним одиниці для сторінки, що перебуває у списку неактивних сторінок, його обнулюють, і сторінку переміщують у список активних (так формують робочий набір процесу). З іншого боку, неактивну сторінку можна використати для розміщення нових даних незалежно від значення цього прапорця. Якщо ж сторінка, для якої PG_referenced= 1, має бути перемішена у список неактивних сторінок, система дає змогу це зробити тільки з другої спроби обходу (сторінці дають другий шанс для її використання).

Під час обходу списку неактивних сторінок для вивільнення пам’яті може виникнутиситуація, коли цього відразу зробити не можна.

♦ Сторінка може бути відображена в адресний простір процесу або кількох процесів. Щоб вивільнити сторінку, відображення спочатку потрібно вилучити для кожного такого процесу.

♦ Сторінка може бути заблокована у пам’яті (наприклад, у ній виділено буфер, який використовують для введення даних із пристрою). Під час обходу таку сторінку пропускають і роблять спробу знайти іншу сторінку для вивільнення. Ця сторінка може бути знову перевірена у такому проході.

♦ Сторінка була модифікована, тому ЇЇ спочатку треба записати на диск. Як тільки не відображену в адресний простір процесу модифіковану сторінку пе¬реміщують із початку в кінець списку неактивних сторінок, система скидає її вміст на диск.

Дляорганізаціїблокуваннясторінокупам’ятівLinuxвикористовуютьсистемнийвикликmlock()протопипякоговизначенийвзаголовномуфайлісysmman.h. СистемнийВИКЛИКmlock() блокуєупам’ятінабірсторінок, munlockOзнімаєцеблокування. Обидваційвикликиприймаютьдвапараметри: адресу, зякоїпочинаєтьсяблокованийнабірсторінок, ірозмірпам’яті, якупотрібнозаблокуватиаборозблокувати.

Наведемоприкладблокуванняоднієїсторінкиізвикористаннямцихвикликів.

```
#include <sys/man.h>
#include <std’ib.h> // для mallocO size_t pagesize = getpagesizeO;
// виділенняпам'ятіпідсторінку char* page = (char *)maT!oc(pagesize):
// сторінка заблокована у пам'яті й не може бути вивантажена на диск пїіоек (page, pagesize):
!! ...
// розблокуваннясторінкиmun1ock(page. pagesize);
```

Системний виклик mlockalU) блокує у пам'яті всі сторінки поточного процесу. Користуватися ним потрібно з обережністю, оскільки заблоковану пам’ять неможна використовувати під час розподілу пам’яті доти, поки вона не буде розблокована (або поки процес не завершиться).

Питання для самоконтролю

1. Що називають дескриптором пам’яті?
2. За допомогою якого системного виклику можна отримати кількість переривань?

Теми для рефератів

1. Принципи функціонування розподілювачів пам’яті
2. Організація списків вільних блоків

Список рекомендованих джерел

1. Шеховцов В.А. Операційні системи. – К.:Видавнича група BHV, 2005. – 576с.:іл.
2. Рихтер Д. Windows для профессионалов: Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows. – М.: Русская редакция, 2001. – 752с.
3. Харт Дж.В. Системное программирование в бреде Win32 – М.: Вильямс, 2001 – 464с.
4. Рихтер Джеффри. Windows для проффесионалов: Програмирование для Windows 95 и Windows NT 4.0. на базе Win32/API. – М.: Изд.отдел «Русская редакция», 1997. – 712с.
5. Румянцев П.В. Азбука программирования в Win32 API. – М.: Радио и связь, 1998. – 272с.

Самостійна робота №14

Тема: Реалізація віртуальної пам’яті у Windows.
Мета:Пояснити процес реалізаціїї віртуальної пам’яті уWindows.

План

1. **Віртуальний адресний простір процесу**
2. **Організація заміщення сторінок**

Теоретична частина

1. Віртуальний адресний простір процесу

Сторінки адресного простору процесу можуть бути вільні (free), зарезервовані (reserved) і підтверджені (committed).

Вільні сторінки не можна використати процесом прямо, їх потрібно спочатку зарезервувати. Це можливо зробити будь-яким процесом системи. Після цього інші процеси резервувати ту саму сторінку не можуть.

У свою чергу, процес, що зарезервував сторінку, не може цю сторінку використати до її підтвердження, тому що зв’язок із конкретними даними або програмами для неї не визначений.

Підтверджені сторінки безпосередньо пов’язані із простором підтримки на диску. Такі сторінки можуть бути двох типів.

♦ Дані для сторінок першого типу перебувають у звичайних файлах на диску. До таких сторінок належать сторінки коду (їм відповідають виконувані файли) і сторінки, що відповідають файлам даних, відображеним у пам’ять (див. розділ 11). Для таких сторінок простором підтримки будуть відповідні файли. Один і той самий файл може підтримувати блоки адресного простору різних процесів.

♦ Сторінки другого типу не пов’язані прямо із файлами на диску. Це може бути, наприклад, сторінка, що містить глобальні змінні програми. Для таких сторінок простором підтримки є спеціальний файл підкачування (swap file). У ньому не резервують простір доти, поки не з’явиться необхідність вивантаження сторінок на диск. Сторінки, зарезервовані у файлі підкачування, називають ще тіньовими сторінками (shadow pages).

Коли для процесу виділяють адресний простір, більшу йото частину становлять вільні сторінки, які не можуть бути використані негайно. Для того щоб використати блоки цього простору, процес спочатку має зарезервувати в ньому відповідні регіони пам’яті.

Регіон пам’яті відображає неперервний блок логічного адресного простору процесу, який може використати застосування. Регіони характеризуються початковою адресою і довжиною. Початкова адреса регіону повинна бути кратною 64 Кбайт, а його розмір — кратним розміру сторінки (4 Кбайт).

Для резервування регіону пам’яті використовують функцію VirtualAllocO (одним із її параметрів має бути прапорець MEM_RESERVE). Після того як регіон був зарезервований, інші запити виділення пам’яті не можуть його повторно резервувати. Ось приклад виклику VirtualAllocO для резервування регіону розміру size:

PVOID addr = VirtualAlloc(NULL. Size. MEM_RESERVE. PAGE_READWRITE);

Першим параметром VirtualAllocO є адреса пам’яті, за якою роблять виділення, якщо вона дорівнює NULL, пам’ять виділяють у довільній ділянці адресного простору процесу. Останнім параметром є режим резервування пам’яті, серед можливих значень цього параметра можна виділити PAGE_READONLY — резервування тільки для читання; PAGE_READWRITE — резервування для читання і записування.

Однак і після резервування регіону будь-яка спроба доступу до відповідної пам’яті спричинятиме помилку. Щоб такою пам’яттю можна було користуватися, резервування регіону має бути підтверджене (commit). Підтвердження зводиться до виділення місця у файлі підкачування сторінок (для сторінок регіону створюютьвідповідні тіньові сторінки). Для підтвердження резервування регіону теж вико-ристовують функцію Vtrtua 1АПос(), але їй потрібно передати прапорець MEM_COMMIT: VirtualAlloc(addr. size. MEM_COMMIT. PAGE_READWRITE);

Звичайною стратегією для застосування є резервування максимально великого регіону пам’яті, що м-оже знадобитися для його роботи, а потім підтвердження цього резервування для невеликих блоків всередині регіону в разі необхідності. Це підвищує продуктивність, тому що операція резервування не потребує досту¬пу до диска і виконується швидше, ніж операція підтвердження.

Можна зарезервувати і підтвердити регіон пам’яті протягом одного виклику функції

VirtualAllocCaddr, size, MEM_RESERVE | MEM_COMMIT, PAGE_READWRITE);

Після того, як використання регіону завершене, його резервування потрібно скасувати. Для цього використовують функцію VirtualFreeCaddr, 0. MEM_RELEASE);

Зазначимо, що цій функції передають не розмір пам’яті, яка звільняється, а нуль (ОС сама визначає розмір регіону).

Наведемо приклад використання резервування і підтвердження пам’яті. При-пустимо, що необхідно працювати із масивом у пам’яті, але в конкретний момент часу буде потрібний тільки один із його елементів. У цьому разі доцільно зарезервувати пам’ять для всього масиву, після чого підтверджувати пам’ять для окремих елементів.

// резервування масиву зі 100 елементів int *array = (int *) VirtualAlloc(NULL.
100 * sizeof(int). MEM_RESERVE. PAGE_READWRITE);
// підтвердження елемента масиву з індексом 10
VirtualAlloc(&array[10], sizeof(int). MEM_C0MMIT, PAGE_READWRITE);
// робота із пам'яттю array[10] = 200;
printfCflaHi у пам'яті: Xd\n", аггау[10]);
// вивільнення масиву VirtualFree(array. 0. MEM_RELEASE):

Назвемо причини виникнення сторінкових переривань у Windows XP.

♦ Звертання до сторінки, що не була підтверджена.

♦ Звертання до сторінки із недостатніми правами. Ці два випадки є фатальними помилками і виправленню не підлягають.

♦ Звертання для записування до сторінки, спільно використовуваної процесами. У цьому разі можна скористатися технологією копіювання під час записування.

♦ Необхідність розширення стека процесу. У цьому разі оброблювач переривання має виділити новий фрейм і заповнити його нулями.

♦ Звертання до сторінки, що була підтверджена, але в конкретний момент не завантажена у фізичну пам'ять. Під час обробки такої ситуації використовують локальність посилань: із диска завантажують не лише безпосередньо потрібну сторінку, але й кілька прилеглих до неї, тому наступного разу їх уже не доведеться заново підкачувати. Цим зменшують загальну кількість сторінкових переривань.

Поточну кількість сторінкових переривань для процесу можна отримати за допомогою функції GetProcessMemorylnfoO:

#inc1ude <psap1.h>

PROCESS_MEM0RY_C0UNTERS info:

GetProcessMemoryInfo(GetCurrentProcess(). &info, sizeof (info)): printf("Усього сторінкових збоїв: ld\n", info.PageFaultCount);

2. Організація заміщення сторінок

Базовий принцип реалізації заміщення сторінок у Windows XP — підтримка деякої мінімальної кількості вільних сторінок у пам'яті. Для цього використовують кілька концепцій: робочі набори, буферизацію, старіння, фонове заміщення і зворотне відображення сторінок.

Поняття робочого набору є центральним для заміщення сторінок у Windows XP. У цій ОС під робочим набором розуміють множину підтверджених сторінок процесу, завантажених в основну пам'ять. Під час звертання до таких сторінок не виникатиме сторінкових переривань. Кожний набір описують двома параметрами: його нижньою і верхньою межами. Ці межі не є фіксованими, за певних умов процес може за них виходити; крім того, вони пізніше можуть мінятися. Початкове значення меж однакове для всіх процесів (нижня межа має бути в діапазоні 20-50, верхня - 45-345 сторінок).

Менеджер пам'яті постійно контролює сторінкові переривання для процесу, коригуючи його робочий набір. Якщо під час обробки сторінкового переривання виявляють, що розмір робочого набору процесу менший за мінімальне значення, до цього набору додають сторінку, якщо ж більший за максимальне значення — із набору вилучають сторінку. Оскільки всі ці дії стосуються робочого набору того процесу, що викликав сторінкове переривання, базова стратегія заміщення сторінок є локальною. Втім, локальність заміщення є відносною: у деяких ситуаціях система може коригувати робочий набір одного процесу за рахунок інших (наприклад, якщо для одного процесу бракує фізичної пам'яті, а для інших її достатньо).

Крім локального коригування робочого набору в оброблювачах сторінкових переривань, у системі також реалізовано глобальне фонове заміщення сторінок. Спеціальний потік ядра, який називають менеджером балансової множини (balance set manager), виконується за таймером раз за секунду і перевіряє, чи не опустилася кількість вільних сторінок у системі нижче за допустиму межу. Якщо так, потік

запускає інший потік ядра — менеджер робочих наборів (working set manager), що забирає додаткові сторінки у процесів, коригуючи їхні робочі набори.

Під час вибору сторінок для вилучення із робочого набору застосовують модифікацію годинникового алгоритму із використанням концепції старіння сторінок. Із кожною сторінкою пов'язаний цілочисловий лічильник ступеня старіння. Усі сторінки набору обходять по черзі. Якщо біт R для сторінки дорівнює 0, лічильник збільшують на одиницю, якщо R дорівнює 1, лічильник обнуляється. Внаслідок обходу заміщуються сторінки із найбільшим значенням лічильника. Заміщені сторінки не зберігають негайно на диску, а буферизують.

Питання для самоконтролю

1. Що таке базовий принцип реалізації заміщення сторінок у Windows XP?
2. Назвіть причини виникнення сторінкових переривань у Windows XP?

Теми для рефератів

1. Динамічний розподіл пам'яті в Linux та Windows.
2. Структури файлів і файлових систем.

Список рекомендованих джерел

1. Шеховцов В.А. Операційні системи. – К.:Видавнича група BHV, 2005. – 576с.:іл.
2. Рихтер Д. Windows для профессионалов: Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows. – М.: Русская редакция, 2001. – 752с.
3. Харт Дж.В. Системное программирование в бреде Win32 – М.: Вильямс, 2001 – 464с.
4. Рихтер Джеффри. Windows для проффесионалов: Програмирование для Windows 95 и Windows NT 4.0. на базе Win32/API. – М.: Изд.отдел «Русская редакция», 1997. – 712с.
5. Румянцев П.В. Азбука программирования в Win32 API. – М.: Радио и связь, 1998. – 272с.