В.П. Майданюк, Г.Б. Ракитянська, В.А. Каплун

СИСТЕМНЕ ПРОГРАМУВАННЯ І ОПЕРАЦІЙНІ СИСТЕМИ
ОСНОВИ ТЕОРІЇ ОПЕРАЦІЙНИХ СИСТЕМ

Міністерство освіти і науки України
Вінницький державний технічний університет

В.П. Майданюк, Г.Б. Ракитянська, В.А. Каплун

СИСТЕМНЕ ПРОГРАМУВАННЯ І ОПЕРАЦІЙНІ СИСТЕМИ
ОСНОВИ ТЕОРІЇ ОПЕРАЦІЙНИХ СИСТЕМ

Затверджено Ученою радою Вінницького державного технічного університету як навчальний посібник для студентів бакалаврського напрямку 6.0804 – “Комп'ютерні науки” спеціальності 7.080403 – “Програмне забезпечення автоматизованих систем” денної та заочної форм навчання. Протокол № 5 від 12 грудня 2002 р.

Вінниця ВДТУ 2003

УДК 681.325.5
М 52

Р е ц е н з е н т и:

В.П. Кожем'яко, доктор технічних наук, професор
Р.Н. Квєтний, доктор технічних наук, професор
О.О. Коваленко, кандидат технічних наук, доцент

Рекомендовано до видання Ученою радою Вінницького державного технічного університету Міністерства освіти і науки України

Майданюк В.П., Ракитянська Г.Б., Каплун В.А.
М52 **Системне програмування і операційні системи. Основи теорії операційних систем.** Навчальний посібник. - Вінниця: ВДТУ, 2003. – 92 с.

У посібнику розглядаються питання побудови і функціонування операційних систем. Посібник розроблений у відповідності з планом кафедри програмного забезпечення та програмою дисципліни "Системне програмування та операційні системи".

УДК 681.325.5

Міністерство освіти і науки України
Вінницький державний технічний університет

В.П. Майданюк, Г.Б. Ракитянська, В.А. Каплун

СИСТЕМНЕ ПРОГРАМУВАННЯ І ОПЕРАЦІЙНІ СИСТЕМИ
ОСНОВИ ТЕОРІЇ ОПЕРАЦІЙНИХ СИСТЕМ

Усі цитати, цифровий, фактичний матеріал та бібліографічні відомості перевірені, написання одиниць відповідає стандартам.
Зауваження рецензентів враховані.

Автори: _____ В.П.Майданюк
(підпис)
_____ Г.Б.Ракитянська
(підпис)
_____ В.А.Каплун
(підпис)

Вимогам, які висуваються до навчальної літератури, відповідає.
До друку і в світ дозволяю на підставі § 2 п.15 "Єдиних правил..."

Проректор з навчальної та науково-методичної роботи
В.О.Леонтьєв

Затверджено
на засіданні кафедри ПЗ
Протокол № від 3.12.2002р.
Зав. кафедрою
_____ А.М.Пєтух
(підпис)

Вінниця ВДТУ 2003

Навчальне видання

Володимир Павлович Майданюк,
Ганна Борисівна Ракитянська,
Валентина Аполінаріївна Каплун

СИСТЕМНЕ ПРОГРАМУВАННЯ І ОПЕРАЦІЙНІ СИСТЕМИ
ОСНОВИ ТЕОРІЇ ОПЕРАЦІЙНИХ СИСТЕМ

Оригінал-макет підготовлено авторами

Редактор С.А. Малішевська

Навчально-методичний відділ ВДТУ
Свідоцтво Держкомінформу України
серія ДК № 746 від 25.12.2001
21021, м.Вінниця, Хмельницьке шосе, 95, ВДТУ

Підписано до друку Гарнітура Times New Roman
Формат 29,7х42 1/4 Папір офсетний
Друк різографічний Ум. друк. арк.
Тираж прим.
Зам. №

Віддруковано в комп'ютерному інформаційно-видавничому центрі
Вінницького державного технічного університету
Свідоцтво Держкомінформу України
серія ДК № 746 від 25.12.2001
21021, м. Вінниця, Хмельницьке шосе, 95

В.П. Майданюк, Г.Б. Ракитянська, В.А. Каплун

СИСТЕМНЕ ПРОГРАМУВАННЯ І ОПЕРАЦІЙНІ СИСТЕМИ
ОСНОВИ ТЕОРІЇ ОПЕРАЦІЙНИХ СИСТЕМ

КЛЮЧОВІ СЛОВА

Керування процесами, керування пам’яттю, організація і керування віртуальною пам’яттю, керування процесорами, керування пристроями і зовнішньою пам’яттю, захист інформації в операційних системах, оцінювання продуктивності операційних систем

3.12.02

_____ В.П. Майданюк
(підпис)

_____ Г.Б. Ракитянська
(підпис)

_____ В.А. Каплун
(підпис)

Зміст

Вступ

Даний навчальний посібник призначений для студентів спеціальності 7.080403 “Програмне забезпечення автоматизованих систем” для самостійної підготовки з дисциплін “Системне програмування і операційні системи”. Він містить відомості про основи теорії операційних систем і є одним з серії навчальних посібників з даної дисципліни. Поза межами даного навчального посібника залишились питання проектування інтерфейсів користувача, оскільки ці відомості наведені в [1], який можна розглядати як один з розділів задуманої серії, а також питання пов’язані з будовою операційних систем для мереж, які розглядаються на старших курсах в окремій дисципліні. Оглядові матеріали з питань, що розглядаються можна знайти в [2], в якості лабораторного практикуму можуть бути використані матеріали наведені в [3].

У посібнику достатньо повно розглянуті такі важливі питання як керуванням процесами, керуванням оперативною пам’яттю, плануванням завантаження процесорів, керуванням та розподілом дискової пам’яті. Особлива увага приділена організації та керуванню віртуальною пам’яттю, оскільки це тісно пов’язано з надійною роботою багатозадачних операційних систем.

На думку авторів наведені в навчальному посібнику відомості можуть бути використані студентами інших спеціальностей, а також магістрами та аспірантами в їх самостійній роботі.

Автори висловлюють щиру подяку рецензентам, доброзичливе ставлення та поради яких сприяли покращенню змісту та структури навчального посібника.

1 Основні поняття операційних систем

Операційна система (ОС) часто в більшій мірі визначає уявлення користувача про машину, ніж сама апаратура цієї машини. Очевидно, що необхідно мати чітке і адекватне означення операційної системи, знати які функції виконує операційна система і якими ресурсами вона керує.

1.1 Основні терміни і означення

Відповідно до ГОСТ 1981-83 під операційною системою *розуміють систему програм, призначену для забезпечення певного рівня ефективності обчислювальної системи за рахунок автоматизованого керування її роботою і надання користувачам певного роду послуг* [4].

Дейтел вважає, що *операційна система - це набір програм, як звичайних, так і мікропрограм, які забезпечують можливість використання апаратури комп'ютера. При цьому апаратура комп'ютера вважається "сирою" обчислювальною потужністю, а задача ОС - в тому, щоб зробити апаратуру доступною і по можливості зручною для користувача* [5].

Завдяки ОС користувач отримує можливість не замислюватись про фізичні деталі будови обчислювальних машин з якими він працює. Замість цього користувач працює з функціональним еквівалентом комп'ютера, який створюється для нього певною операційною системою. Користувач працює на так званій "розширеній" машині (рис.1.1).

Рисунок 1.1 - Структура "розширеної" машини

Під *програмним забезпеченням* (ПЗ) ЕОМ розуміють сукупність програм, процедур і правил разом з усією зв'язаною з цим документацією, яка дозволяє використовувати обчислювальну машину для розв’язання різних задач.

За задачами і функціями програмне забезпечення (ПЗ) можна поділити на дві групи: загальне та спеціальне.

Загальне ПЗ включає в себе засоби контролю, системи програмування, власне саму операційну систему.

Спеціальне ПЗ - пакети прикладних програм.

Операційна система - це система, яка керує виконанням інших програм. Частина ОС, що постійно знаходяться в основній пам'яті, називається *резидентною частиною* ОС або *ядром* ОС. Програми, що

викликаються в основну пам'ять для виконання певних функцій, але не зберігаються там постійно, називаються *транзитами*. Розділення ОС на ядро і транзити дозволяє зменшити область пам'яті, яку займає ОС і, відповідно, збільшити область програм користувачів. ОС побудована за модульним принципом, який дозволяє користувачеві налагоджувати систему відповідно до конкретної конфігурації технічних засобів.

Процес настроювання ОС на конкретну конфігурацію апаратних засобів і задач користувача, називається *генерацією системи*. Засоби генерації являють собою сукупність програм і правил, необхідних для виконання настроювання ОС [4].

1.2 Функції, ресурси та взаємозв'язки ОС

ОС реалізує багато *функцій*, серед яких є такі:

- визначає інтерфейс користувача;
- забезпечує розподілення апаратних ресурсів між користувачами;
- надає можливість працювати зі спільними даними в режимі колективного доступу;
- планує доступ користувачів до спільних ресурсів;
- забезпечує ефективне виконання операцій введення-виведення;
- відновлює інформацію і обчислювальний процес у випадку помилки.

ОС керує такими основними *ресурсами:*

- процесорами;
- пам'яттю;
- пристроями введення-виведення;
- даними.

ОС *взаємодіє* з:

- операторами ЕОМ;
- прикладними програмістами;
- системними програмістами (супроводжування ОС, налагоджування відповідно до вимог конкретної машини, доробка обслуговування нових типів пристроїв);
- адміністраторами ОС (які встановлюють принципи і порядок роботи, взаємодіють з ОС для забезпечення певного порядку);
- програмами;
- апаратними засобами;
- користувачами.

1.3 Покоління операційних систем

Нульове покоління (40-і роки).

ОС не було. Всі програми писалися на машинній мові і користувач мав доступ до всіх пристроїв.

Перше покоління (50-і роки).

Початок систем *пакетної обробки*. Декілька задач об'єднувались в групи (пакети). Запущена на виконання задача отримувала всі ресурси машини. Після завершення задачі (аварійного чи нормального) управління передавалося ОС, яка "прочищала" машину і забезпечувала введення та запуск нової задачі.

Друге покоління (початок 60-х років).

Характерною особливістю було те, що вони створювались як системи колективного користування з *мультипрограмним режимом роботи* і як перші *ОС мультипроцесорного типу*. Декілька задач знаходилися в пам'яті комп'ютера, а центральний процесор швидко переключався із задачі на задачу. Вони забезпечували незалежність програмування від зовнішніх пристроїв (не було потреби конкретно вказувати фізичний номер потрібного пристою). Були розроблені перші *системи з розподілом часу*, які давали можливість безпосередньої роботи з комп'ютером в *діалоговому (інтерактивному) режимі* за допомогою пультів-терміналів. З'явились перші *системи реального часу*, коли комп'ютери використовувались для управління технологічними процесами. Для систем реального часу характерне те, що вони забезпечують термінову реакцію на передбачені події. Оскільки ці системи повинні бути завжди в стані готовності, то велику частину часу вони простоювали, отже, працювали зі значним недовантаженням і тому коштували досить дорого.

Третє покоління (середина 60-х – середина 70 років).

Початком цього покоління вважається поява в 1964 році сімейства комп'ютерів IBM-360. Це були громіздкі комп'ютери (*машини загального призначення)*, призначені виконувати будь-які задачі з будь-якої області (але не кожному користувачу потрібні всі можливості) і тому дуже коштовні.

ОС для цих комп'ютерів були *багаторежимними системами*. Забезпечували пакетну обробку, режим реального часу, розподіл часу і мультипроцесорний режим. Недоліки: громіздкість і малоефективність, складність мов керування завданнями. Основні принципи, закладені в цих ОС, використовуються дотепер.

Четверте покоління (середина 80-х років і донині).

Для цього покоління характерні:

- Концепція розподіленої обробки даних за рахунок використання персональних комп'ютерів (ПК) і мереж.
- Дружній інтерфейс, орієнтований на непідготовленого користувача [6].

Завдяки появі мікропроцесора були розроблені *персональні комп'ютери.* З'являються системи з управлінням за допомогою меню. Важливу роль стали відігравати *системи баз даних.* Почала широко розповсюджуватись концепція *віртуальних машин*, коли користувач не

замислюється над фізичними деталями будови обчислювальних машин або систем, а має справу з функціональним еквівалентом комп'ютера, створеним для нього операційною системою [5].

1.4 Класифікації операційних систем

Класифікація ОС може бути проведена за декількома критеріями.

1. За кількістю задач, що одночасно виконуються: однопрограмні та багатопрограмні ОС.
2. За кількістю користувачів: з одним користувачем та з декількома.
3. За типом інтерфейсу користувача: ОС безпосереднього доступу (діалогові), непрямого доступу (системи пакетної обробки).
4. За числом процесорів: однопроцесорні і мультипроцесорні.
5. За видом обробки даних: ОС із розподіленою обробкою даних (мережеві ОС) і з нерозподіленою обробкою.
6. За часом реакції системи: операційні системи реального часу та системи, що працюють зі швидкостями, визначеними внутрішніми особливостями самих систем.

Системи реального часу забезпечують виконання запиту протягом інтервалу часу, який відповідає реакції людини, або об'єкта, яким керує система.

Обчислювальна система працює в *однопрограмному режимі* (рис.1.2), якщо вона обслуговує тільки одну програму користувача. Користувач може працювати або в режимі безпосереднього доступу, або непрямого доступу (пакетний режим). Режим пакетної обробки дозволяє зменшити простої процесора, але, оскільки одна програма не може завантажити роботою всі пристрої, то це призводить до їх простою, що не є ефективним.

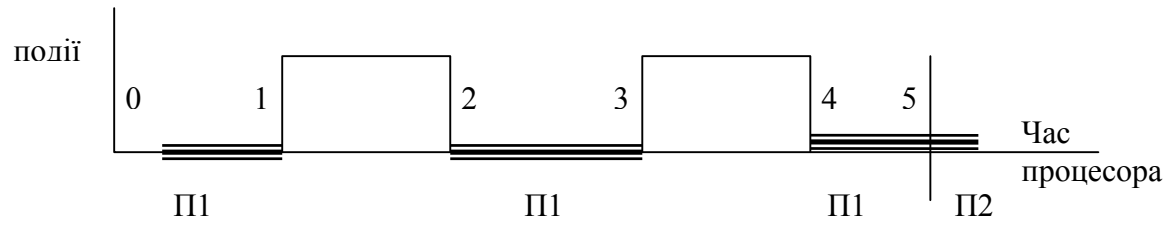


Рисунок 1.2 - Однопрограмний режим. 0-1, 2-3, 4-5 – виконання процесу П1, 5... - виконання П2, 1-2,3-4 - очікування зовнішньої події (процесор простоює).

Обчислювальна система працює в *мультипрограмному режимі*, якщо декілька програм одночасно знаходиться в пам'яті комп'ютера, причому виконання будь-якої з них може бути перервано для виконання іншої програми з наступним поверненням до виконання перерваної програми. Користувач може працювати як в режимі непрямого доступу (пакетна обробка), так і в режимі безпосереднього доступу.

Розрізняють такі типи багатопрограмної роботи:

- класичне мультипрограмування;
- паралельна обробка;
- розподіл часу.

Класичне мультипрограмування характеризується тим, що правила переходу від однієї програми до іншої визначається з міркувань досягнення максимальної ефективності шляхом широкого використання суміщень.

Програма може бути перервана у двох випадках, а саме:

- якщо їй потрібне виконання якої-небудь події (природні переривання);
- якщо вона завершила виконання (рис. 1.3).

Недоліки: одна програма може монополізувати час процесора і заблокувати виконання інших. Мультипрограмування не сумісне з безпосереднім доступом і є різновидом пакетної обробки.

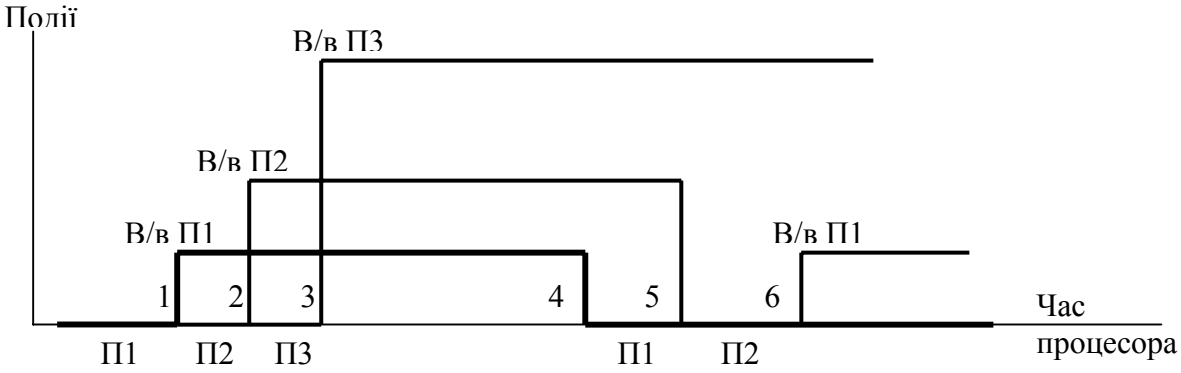


Рисунок 1.3 - Часова діаграма для класичного мультипрограмування
(3-4 - простій процесора)

Під час *паралельної обробки* (рис.1.4) перехід від однієї програми до іншої виконується через досить короткий інтервал часу порівняно з швидкістю роботи машини, щоб створити у людини враження одночасного виконання декількох програм.

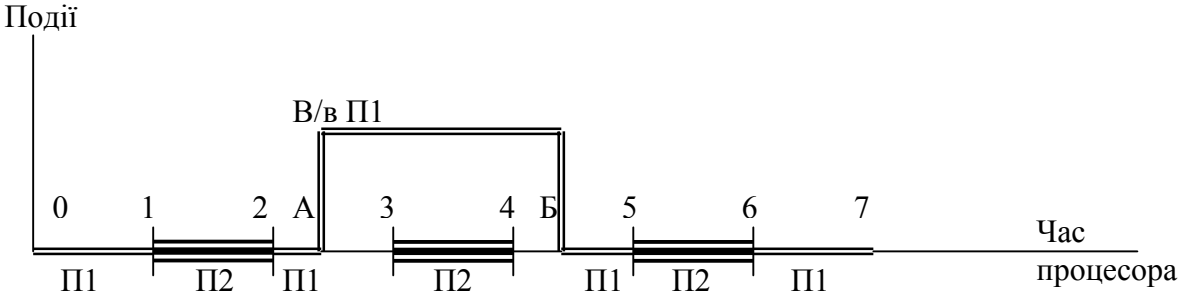


Рисунок 1.4 - Часова діаграма паралельної обробки
(А-3, 4-Б - простій процесора)

Паралельна обробка допускає організацію безпосереднього доступу.

Недоліки: перехід від програми до програми можливий лише через строго визначені інтервали часу через примусові переривання, тому виникають простої процесора. Примусові переривання генеруються системним таймером.

Системи з розподілом часу - це режим багатопрограмної роботи, коли перехід від однієї програми до іншої виконується за рахунок як "природних", так і “примусових” переривань (рис. 1.5).

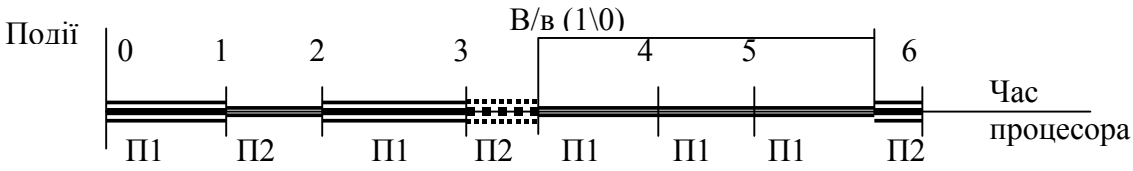


Рисунок 1.5 - Часова діаграма системи з розподілом часу

1.5 Склад і загальна схема функціонування ОС

З точки зору користувача ОС виконує такі основні функції:

- Керування завданнями.
- Керування задачами (процесами).
- Керування даними.
- Керування відновленням.

У відповідності з цим поділяється і керівна програма ОС. Керування завданнями визначає перехід від завдання користувача до задачі, що виконує машина. Підпрограми керування завданнями приймають і планують завдання по мірі їх надходження.

Керування задачами (процесами) слідкує за фактичним виконанням робіт; керування здійснюється відповідно з однією з дисциплін планування (LIFO, FIFO і т.д.).

Керування даними спрощує розміщення, пошук і підтримку всіх даних незалежно від способу їх організації або зберігання.

Керування відновленням дозволяє системі корегувати або виявляти помилку в роботі будь-якого елементу апаратури і замінювати або відключати непрацюючі пристрої.

З точки зору системних програмістів ОС керує пам'яттю, процесорами, зовнішніми пристроями, інформацією. У відповідності з цим ми розглядатимемо такі компоненти ОС:

1. Керування процесами,
2. Керування пам'яттю,
4. Керування процесорами,
5. Керування зовнішньою пам'яттю,
6. Керування пристроями введення-виведення [5].

1.6 Ієрархічна структура ОС

Більшість ранніх ОС складались з однієї великої програми, але цей підхід став призводити до ускладнень реалізації. Вирішенням цієї проблеми стало застосування принципу модульності та ієрархічний підхід до структури ОС. *Ієрархічний підхід полягає в тому, що набори керівних програм рознесені по рівнях (шарах) спадаючих рангів*. Цей підхід запропонував Дейкстра в 1968 році. Структурна схема ОС показана на рис.1.6.

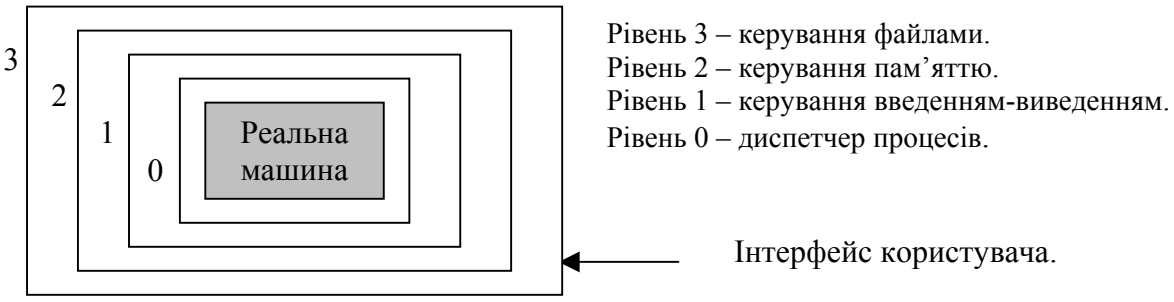


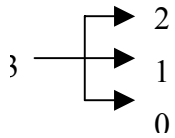
Рисунок 1.6 - Схема операційної системи з ієрархічною структурою

Не існує твердого правила, які модулі слід розміщувати на якому рівні.

Кожен шар може використовувати тільки функції, які йому надають нижчі рівні, так, ніби вони є реальною машиною. ОС може бути створена і налагоджена по шарах, що зменшує складність розробки.

Розрізняють системи із строгою і прозорою ієрархією:

Строга ієрархія: 3 → 2 →1 → 0

Прозора ієрархія: 

В системах з строгою ієрархією функції вищого рангу можуть звертатись до функцій тільки одного шару рангом нижче.

1.7 Основні принципи побудови ОС

1. *Частотний принцип* - для дій, які часто зустрічаються і забезпечуються умови їх швидкого виконання.
2. *Принцип модульності*.
3. *Принцип функціональної вибірковості* - виділяється частина важливих модулів, які завжди повинні бути в пам'яті (виділяється ядро).
4. *Принцип генерованості*.
5. *Принцип функціональної надлишковості* – забезпечується можливість виконання однієї і тієї ж роботи різними засобами.

6. *Принцип "за замовчуванням"* - оснований на зберіганні в системі деяких базових описів структур, процесів, конфігурації обладнання, що забезпечує початкову працездатність системи без її генерації.
7. *Принцип переміщуваності* - побудова модулів, виконання яких не залежить від їх місця в пам'яті.
8. *Принцип захисту* - необхідна розробка засобів, які виключають вплив одних програм на інші.
9. *Принцип незалежності* програм від зовнішніх пристроїв.
10. Принцип відкритої і нарощуваної ОС [4].

Контрольні питання

1. Наведіть означення ОС.
2. Дайте характеристику резидентної і транзитної частин ОС.
3. Наведіть і охарактеризуйте покоління ОС.
4. Критерії класифікації ОС.
5. Охарактеризуйте однопрограмний режим роботи ОС.
6. Дайте класифікацію багатопрограмних режимів ОС.
7. Охарактеризуйте будову ОС з точки зору користувача і системного програміста.
8. Які переваги ієрархічного підходу до будови ОС?
9. Наведіть основні принципи побудови ОС.

2 Керування процесами

Термін *процес* почали використовувати розробники системи MULTICS в 60-х роках. Є різні визначення процесу. Наведемо деякі з них:

1. Процес – це програма в стадії її виконання.
2. Процес – це об'єкт, якому виділяється процесор [5].

Згідно з ГОСТ 19781-83, ***процес*** *- це система дій, яка реалізує певну функцію в обчислювальній системі і оформлена так, що керуюча програма обчислювальної системи може перерозподіляти* ***ресурси*** *цієї системи з метою забезпечення мультипрограмування.*

Ресурс *це засіб обчислювальної системи, який може бути виділений процесу на визначений інтервал часу* [4].

2.1 Стани процесів

За період свого існування процес проходить через різні дискретні стани, а саме:

- *стан виконання* (якщо процесу в даний момент виділено центральний процесор);
- *стан готовності* (якщо процес міг би відразу використати центральний процесор, наданий в його розпорядження);
- *стан блокування* (якщо процес очікує появу деякої події, щоб продовжити виконання);
- *стан призупинений блокований*;
- *стан призупинений готовий*.

В однопроцесорній системі в кожен момент часу може виконуватись тільки один процес, декілька процесів можуть знаходитись в стані готовності, декілька - в стані блокування. ОС створює список готових до виконання і заблокованих процесів. Список готових процесів впорядковується за пріоритетами. Наступним процесом, який отримує центральний процесор, буде перший процес з цього списку. Розблокування процесів виконується в тому порядку, в якому відбуваються очікувані події [4, 5].

2.2 Перехід процесу зі стану в стан

Завдання створює процес. Процес записується до списку готових процесів відповідно до свого пріоритету. Якщо всі процеси мають однаковий пріоритет, то процес записується в кінець списку. По мірі завершення виконання попередніх процесів процес переміщується до головної частини списку. Коли він стає першим в списку готових процесів, то по завершенні попереднього процесу йому виділяється центральний процесор (ЦП) і він переходить із стану готовності в стан виконання. По завершенні кванту часу, виділеного процесу, центральний процесор

передається наступному процесу, а даний процес переходить в стан готовності. Якщо впродовж кванту часу, виділеного даному процесу, йому буде потрібна операція введення-виведення або інші події, необхідні для продовження роботи, він переводиться в стан блокування (рис.2.1).

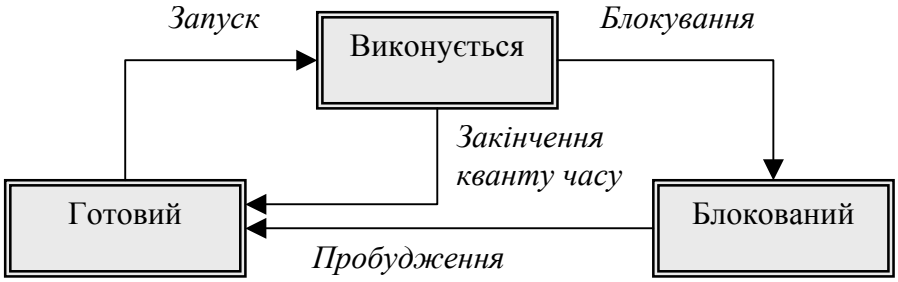


Рисунок 2.1 - Діаграма станів процесу

Зміну станів процесу позначають так:

Запуск: готовий ⟶ виконується;
Пробудження: блокований ⟶ готовий;
Блокування: виконується ⟶ блокований;
Закінчення кванту часу: виконується ⟶ готовий.

Надання ЦП першому процесу зі списку готових називається запуском, або вибором процесу на виконання. Запуск виконується системним диспетчером. Щоб не допустити захоплення ресурсів одним процесом, ОС встановлює в таймері інтервал часу, впродовж якого даному процесу дозволяється займати ЦП. Якщо процес добровільно не звільнить ЦП, то таймер формує переривання, за яким керування буде передане ОС (диспетчеру), який переводить даний процес в стан готовності. Якщо процес до закінчення виділеного йому кванту часу ініціює операцію введення-виведення, то він звільняє ЦП і переходить в стан блокування. Лише блокування ініціюється самим процесом, інші зміни стану ініціюються зовнішніми об'єктами відносно даного процесу.

Блок керування процесом PCB (programm control block)

Представником процесу в ОС є блок керування процесом (PCB). Це структура даних, яка містить інформацію про даний процес, в тому числі поточний стан процесу, пріоритет, унікальний ідентифікатор процесу, по-кажчик виділених ресурсів, пам'яті, область зберігання регістрів. В PCB, наприклад, може бути записана інформація, необхідна для перезапуску процесу. В багатьох ЕОМ передбачена апаратна підтримка операцій з PCB. Вводиться спеціальний регістр, який вказує на PCB поточного процесу; є спеціальні команди процесора, які забезпечують швидкі операції з PCB [4-7].

2.3 Операції над процесами

Системи керування процесами повинні мати можливість виконувати деякі операції над процесами, в тому числі:

- створення процесу;
- знищення процесу;
- призупинення процесу;
- відновлення процесу;
- зміна пріоритету процесу;
- блокування процесу;
- пробудження процесу;
- запуск (вибір) процесу.

Створення процесу складається з операцій присвоєння імені процесу, включення цього імені в список імен процесів, відомих системі, визначення початкового пріоритету процесу, формування блоку керування процесом, виділення процесові початкових ресурсів. Процес може породити новий процес, який називається дочірнім, а той, що породжує – батьківським.

Знищення процесу. Процес видаляється з системи, ресурси віддаються системі, ім'я процесу стирається зі списків імен відомих ОС, а РСВ звільняється. В деяких системах дочірні процеси знищуються при знищенні батьківського, в інших продовжують існування незалежно від батьківських.

Зміна пріоритету означає зміну значення пріоритету в блоці керування даним процесом.

Призупинення процесу бувають короткі і довготривалі. У випадку тривалого призупинення ресурси процесу повинні бути звільнені. Причинами призупинень можуть бути такі явища:

- система працює не цілком надійно, є ознаки можливої відмови. Процес відновлюється після виправлення помилки;
- призупинення, ініціатором яких є користувач, який має сумнів в правильності проміжних результатів;
- призупинення під час короткочасних піків навантаження системи.

Відновлення (активація) - операція підготовки процесу до повторного запуску з тієї ж точки, в якій він був призупинений (рис.2.2).

Процес, що виконується в однопроцесорній системі, може призупинити лише сам себе. В усіх інших станах процеси можуть бути призупинені іншими процесами.

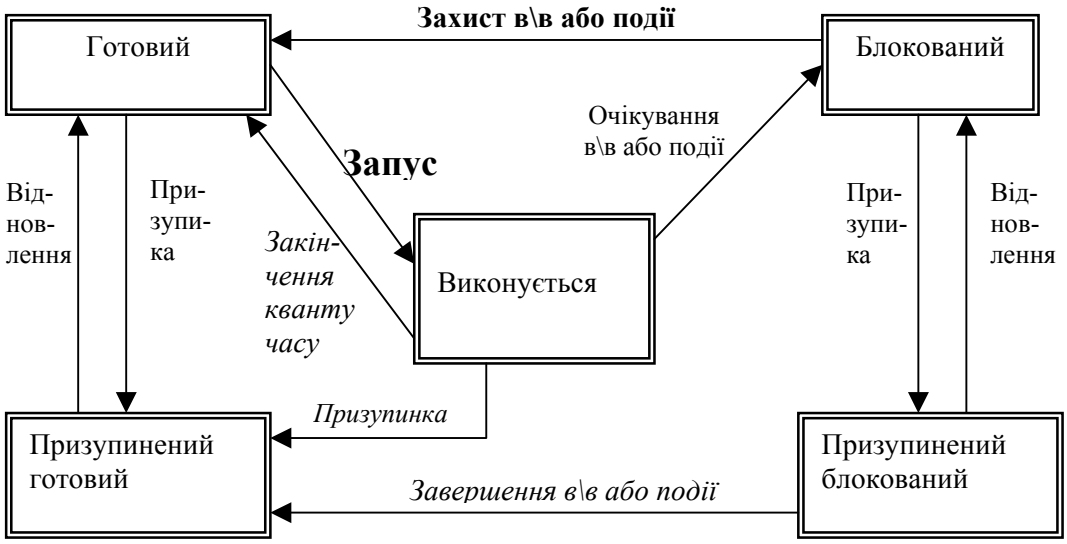


Рисунок 2.2 - Модифікована діаграма станів процесу з урахуванням призупинень і відновлень

2.4 Обробка переривань

Переривання *- це подія, під час якої змінюється нормальна послідовність команд, що виконується процесором*. Під час переривання виконується така послідовність дій:

- керування передається операційній системі;
- операційна система запам'ятовує стан перерваного процесу, інколи PCB перерваного процесу;
- ОС аналізує тип переривання і передає керування відповідній програмі обробки переривань;
- після обробки переривання ОС може відновити виконання перерваного процесу.

Типи переривань:

1. *SVC (Super Visor Call) - переривання*. Це запит, який генерується програмою користувача для отримання конкретної системної послуги (вимога на операцію введення-виведення, збільшення розміру виділеної пам’яті, зв’язок з оператором). Механізм SVC дозволяє захистити ОС від користувача. Ця команда повинна мати параметри для вибору функції ОС (аналог в MS-DOS - INT 21H).
2. *Переривання введення-виведення.* Ці переривання ініціюються апаратурою введення-виведення. Вони сигналізують центральному процесору про зміни стану каналу або пристроїв введення-виведення.
3. *Зовнішні переривання*. Причинами цих переривань можуть бути різні події - закінчення кванту часу, натиснення кнопки

переривання на пульті, сигнал переривання від іншого процесора в мультипроцесорній системі.

4. *Переривання по рестарту (*коли користувач натискає кнопку рестарту на пульті керування або коли інший процесор передає команду рестарту).
5. *Переривання по контролю (помилці) програми* (ділення на нуль, трасування та ін.).
6. *Переривання по контролю (помилці) машини (*апаратні помилки).

Для обробки кожного з цих переривань в складі операційної системи передбачені програми, які називаються обробниками переривань (ІН – Interrupt Handler).

2.5 Ядро ОС

Всі операції, пов'язані з процесами, виконуються під керуванням ядра ОС. Це лише невелика частина ОС, але яка дуже часто використовується, і тому вона резидентно розміщується в основній пам'яті. Інші частини ОС переміщуються в зовнішню пам’ять і назад по мірі необхідності.

Основні функції ядра ОС такі***:***

- обробка переривань;
- створення і знищення процесів;
- переключення процесу із стану в стан;
- синхронізація процесів, організація взаємодії між процесами;
- маніпулювання РСВ;
- підтримка операцій введення-виведення;
- підтримка розподілу і перерозподілу пам'яті;
- підтримка роботи файлової системи;
- підтримка механізмів виклику і повернення під час звертання до процедур;
- підтримка функцій по веденню обліку роботи машини.

Вхід в ядро ОС, як правило, виконується за перериванням. Коли ядро реагує на переривання, то забороняються інші переривання. Ядро виконує мінімально можливу обробку переривання, потім переривання передається відповідному системному процесу, після початку роботи якого переривання дозволяються. Виділення ядра забезпечує кращі швидкісні характеристики, що можливо при реалізації діалогових режимів.

Реалізація ядра. Значна частина функцій ядра реалізується мікро-програмними засобами, що забезпечує більшу швидкість виконання функцій операційних систем .

2.6 Класифікація процесів і види відношень між ними

За часом розвитку процеси поділяються на:

- *процеси реального часу* - гарантовано закінчуються до настання конкретного моменту часу;
- *інтерактивні процеси* - існують не більше інтервалу часу допустимої реакції ЕОМ на запит користувача;
- *пакетні процеси* - процеси, що не ввійшли в попередні класи;
- *еквівалентні процеси* - мають однаковий результат обробки одних і тих же вхідних даних, по одній і тій же або різних програмах, на одному і тому ж або різних процесорах;
- *тотожні процеси* - це еквівалентні процеси, в яких обробка даних відбувається за однією тією ж програмою, але траси не збігаються. (Траса - порядок і тривалість перебування процесу в допустимих станах на інтервалі існування).
- *рівні процеси* - тотожні процеси, у яких траси збігаються;
- *послідовні процеси* - інтервали існування не перетинаються в часі;
- *паралельні процеси* - існують одночасно на певному відрізку часу;
- комбіновані процеси - на деякому інтервалі часу є хоча б одна точка, в якій вони існують одночасно.

За місцем розвитку процеси бувають:

- *внутрішні процеси* - виконуються на центральному процесорі (ЦП) (або процесорах);
- *зовнішні* - виконуються на процесорах, відмінних від ЦП під керуванням ОС;
- *процеси користувача* - процеси виконання програм користувача;
- *системні процеси* - процеси виконання програм ОС.

За способами зв'язків між собою процеси можуть бути взаємозв'язаними, ізольованими, інформаційно-незалежними, взаємодіючими і конкурентними.

Взаємозв'язані процеси - це такі, між якими за допомогою системи керування процесами підтримуються зв'язки: функціональні, просторово-часові, зв'язки з керування, інформаційні.

Ізольовані процеси - це процеси, які не підтримують зв'язки один з одним. Повністю ізольованих процесів в ОС не існує, є процеси зі слабкими зв'язками.

Інформаційно-незалежні процеси - це взаємозв'язані процеси, які спільно використовують ресурси, але інформаційного зв'язку не мають.

Взаємодіючі процеси – це взаємозв'язані процеси, які мають інформаційні зв'язки між собою

Конкурентні процеси – це взаємозв'язані процеси, які мають зв'язок по ресурсах [4].

Взаємозв'язані процеси вступають у відношення між собою.

Види відношень між процесами:

1. *Відношення передування* - для двох процесів це означає, що перший процес повинен переходити в активний стан раніше другого.
2. *Відношення пріоритетності* - процес з пріоритетом Р може бути переведений в активний стан під час виконання двох умов:

- в стані готовності до даного процесора немає процесів з вищим пріоритетом;
- даний процесор або вільний в цей час, або використовується процесом з нижчим пріоритетом.

3. *Відношення взаємовиключення* - декілька процесів використовують спільний ресурс; сукупність дій над спільним ресурсом в складі одного процесу називається критичною ділянкою або областю; критична область одного процесу не повинна виконуватись одночасно з критичною областю іншого.

Відношення, в які вступають процеси, визначають правила синхронізації під час виконання процесів. Реалізація правил синхронізації здійснюється за допомогою механізмів синхронізації. Часто ці механізми мають програмно-апаратну форму реалізації. Особливості кожної конкретної взаємодії між двома або більше процесами визначаються ***задачами синхронізації***. Кількість задач синхронізації необмежена, однак деякі з них є типовими:

- задача взаємовиключення;
- задача "виробники-споживачі";
- задача "письменники-читачі";
- задача "філософи, що обідають".

Спеціальні засоби операційних систем (ОС) встановлюють деякі обмеження на послідовність виконання взаємозв'язаних паралельних процесів, забезпечуючи тим самим їх синхронізацію та спілкування. Невірна реалізація задач синхронізації може призвести до тупиків у системі. Для розв’язання задач синхронізації можуть застосовуватись різні механізми синхронізації. Так, для розв'язання задачі взаємовиключення може застосовуватись алгоритм Деккера або синхронізація за допомогою семафорів. Однак чисто програмна реалізація алгоритмів синхронізації є дуже важким процесом. Ще більш важко довести правильність рішення. Тому в багатьох ЕОМ є механізми апаратної підтримки алгоритмів синхронізації [5].

2.7 Взаємовиключення

Розглянемо наступний приклад. Декілька паралельних процесів звертаються до спільної змінної P1 і додають до неї "1".
Кожний процес має власну копію фрагменту програми:

```
LOAD  P1
ADD   1
STORE P1.
```

Нехай P1=20. Припустимо, що один із процесів виконав команди завантаження LOAD і додавання ADD, після чого значення акумулятора стало рівним 21. Уявимо, що в цей момент закінчився виділений квант часу і перший процес віддає процесор другому процесу. Другий процес виконує всі три команди і встановлює значення змінної P1 рівним 21. Потім керування передається першому процесу, який також присвоює змінній P1 значення 21. Таким чином, через нескоординований доступ до спільної змінної система фактично губить одне звертання до змінної P1. Вірна загальна сума повинна скласти 22.

Цю задачу можна розв'язати, якщо кожному процесу дати монопольне, виключне право доступу до змінної P1. Коли один процес збільшує цю змінну, всі інші процеси, яким також необхідно її змінити в цей самий час, очікують. Це і називається *взаємовиключенням*.

Коли процес звертається до поділюваних даних, то говорять, що він знаходиться на *критичній ділянці*. Якщо будь-який процес знаходиться на критичній ділянці, то інші процеси можуть виконуватись, але без входу на цю критичну ділянку.

2.8 Реалізація взаємовиключення за допомогою алгоритму Деккера

Найпростішим виходом із цієї ситуації було б заборонити переривання на критичній ділянці, але в системах з розподілом часу і в системах реального часу час грає дуже важливу роль. Система може не відреагувати на будь-яку зовнішню подію, або час відповіді збільшиться.

Розглянемо декілька варіантів реалізації взаємовиключення з дотриманням таких обмежень:

- задача повинна бути розв’язана чисто програмними засобами на машині, що не має спеціальних команд взаємовиключення;
- не повинно бути ніяких обмежень щодо швидкості виконання асинхронних паралельних процесів;
- процеси, що знаходяться поза межами своїх критичних ділянок, не заважають іншим процесам входити в їх власні критичні ділянки;
- не повинно бути нескінченого відстрочення моменту входу процесів у критичні ділянки.

На рис. 2.3. наведена перша версія фрагменту програми реалізації взаємовиключення для двох паралельних процесів.

```
Program     Версія1
    var     НомерПроцесу : ціла;
    procedure Процес1;
    begin
        while  true    do
        begin
            while НомерПроцесу=2 do;
            КритичнаДілянка1;
            НомерПроцесу:=2;
            інші  оператори процесу1;
        end;
    end;

    procedure Процес2;
    begin
        while  true    do
        begin
            while НомерПроцесу=1 do;
            КритичнаДілянка2;
            НомерПроцесу:=1;
            інші  оператори процесу 2;
        end;
    end;
    Begin
        НомерПроцесу:=1;
        ParBegin
            Процес1;
            Процес2;
        ParEnd;
    End.
```

Рисунок 2.3 - Програмна реалізація примітивів взаємо виключення (версія 1)

Конструкція ***ParBegin/ParEnd*** дозволяє організувати паралельну роботу процесів “Процес1”, “Процес2”.

Ця програма реалізує взаємовиключення. Однак, після того, як Процес1 увійде на свою критичну ділянку і вийде з неї, а потім знов захоче увійти, а Процес2 ще не настане, то виникне затримка. Такий режим з однією глобальною змінною називається *режимом жорсткої синхронізації*.

Введемо дві змінні: П1 і П2, які мають значення “істина”, якщо Процес1 і Процес2 відповідно знаходяться всередині своїх критичних ділянок (рис.2.4).

Версія 2 не гарантує взаємовиключення. Оскільки Процес1 і Процес2 є паралельними, вони можуть одночасно почати виконувати свої послідовності взаємовиключення. Процес1 може перевірити змінну П2 і якщо П2=FALSE встановить П1=TRUE, а саме в цей час Процес2, виявивши П1=FALSE, зайде в свою критичну ділянку. Тобто, обидва процеси можуть одночасно ввійти в критичну ділянку.

```
Program     Варіант2;
var         П1,П2: логічний;
procedure   Процес1;

begin
        while  true     do

        begin
                while П2=true do;
                П1:=true;
                КритичнаДілянка1;
                П1:=false;
                 інші оператори процесу 1;
        end;

end;

procedure   Процес2;

begin
        while  true     do
        begin
                while П1=true do;
                П2:=true;
                КритичнаДілянка2;
                П2:=false;
                інші оператори процесу 2 ;
        end;
end;

Begin
        П1:=false;
        П2:=false;

        ParBegin
                Процес1;
                Процес2;
        ParEnd;

End.
```

Рисунок 2.4 - Програмна реалізація примітивів взаємовиключення (версія 2)

В наступній версії (рис. 2.5) для запобігання цьому передбачається встановлення кожним процесом свого власного прапорця перед виконанням циклу очікування. Однак, якщо процеси одночасно вийдуть на цикл while, то виникне тупик, який може тривати нескінченно.

```
Program     Версія3;
Var   X1,X2 : логічний;
procedure   Процес1;
begin
      while  true   do
      begin
            X1:=true;
            while X2=true do;
            КритичнаДілянка1;
            X1:=false;
            інші оператори процесу 1;
      end;
end;
procedure   Процес2;
begin
      while  true   do
      begin
            X2:=true;
            while X1=true do;
            КритичнаДілянка2;
            X2:=false;
             інші оператори процесу 2;
      end;
end;
Begin
      X1:=false;
      X2:=false;
      ParBegin
            Процес1;
            Процес2;
      ParEnd;
End.
```

Рисунок 2.5 - Програмна реалізація примітивів взаємовиключення (версія 3)

Тому в програмі версії 4 (рис.2.6) передбачена короткочасна установка фальшивого значення прапорця кожним процесом, що увійшов в цикл. Завдяки чому другий процес отримує можливість вийти з свого циклу очікування, коли встановлений власний прапорець. Хоча це і гарантує взаємовиключення і відсутність тупика, але виникає інша проблема, а саме нескінченне відкладання.

Процеси можуть виконуватись “тандемом”, один за одним у такій послідовності: кожний процес може встановити свій прапорець в “TRUE”; провести перевірку на початку циклу; увійти в тіло циклу; встановити свій прапорець в “FALSE”; знову встановити свій прапорець в “TRUE”; повторити цю послідовність, починаючи з перевірки при вході в цикл. Коли процеси будуть виконувати ці дії, умови перевірки будуть “TRUE”. Такий режим роботи малоймовірний, але все ж можливий. Отже, і ця версія не може бути беззаперечно прийнята.

```
Program     Варіант4;
Var   X1,X2 : логічний;
procedure   Процес1;
begin
      while  true    do
      begin
            X1:=true;
            while X2=true do
            begin
                  X1:=false;
                  Затримка (Випадкова, декілька тактів);
                  X1:=true;
            end;
            КритичнаДілянка1;
            X1:=false;
            ... інші оператори процесу 1 .....;
      end;
end;
procedure   Процес2;
begin
      while  true    do
      begin
            X2:=true;
            while X1=true do
            begin
                  X2:=false;
                  Затримка (Випадкова, декілька тактів);
                  X2:=true;
            end;
            КритичнаДілянка2;
            X2:=false;
            ... інші оператори процесу 2 .....;
      end;
end;
Begin
      X1:=false;
      X2:=false;
      ParBegin
            Процес1;
            Процес2;
      ParEnd;
End.
```

Рисунок 2.6 - Програмна реалізація примітивів взаємовиключення (версія 4)

Вперше запропонував програмну реалізацію механізму взаємовиключення, яка не вимагає спеціальних апаратно реалізованих команд, голландський математик Деккер. Вводиться ще одна змінна “ВибранийПроцес” (рис. 2.7). Процес1 сповіщає про бажання увійти у свою критичну ділянку (X1:= TRUE), потім переходить в цикл, в якому перевіряє, чи не хоче увійти у свою критичну ділянку Процес2.

```
Program     АлгоритмДеккера;
Var   ВибранийПроцес: (1, 2);
      X1,X2 : логічний;
procedure   Процес1;
begin
      while  true   do
      begin
            X1:=true;
            while X2=true do
                  if     ВибранийПроцес=2  then
                  begin
                        X1:=false;
                        While  ВибранийПроцес=2  do;
                        X1:=true;
                  end;
                  КритичнаДілянка1;
                  ВибранийПроцес:=2;
                  X1:=false;
                  ... інші оператори процесу 1 .....;
      end;
end;
procedure   Процес2;
begin  while  true   do
      begin
            X2:=true;
            while X1=true do
                  if     ВибранийПроцес=1  then
                  begin
                        X2:=false;
                        While  ВибранийПроцес=1  do;
                        X2:=true;
                  end;
                  КритичнаДілянка2;
                  ВибранийПроцес:=1;
                  X2:=false;
                  ... інші оператори процесу 2 .....;
      end;
end;
Begin
      X1:=false;
      X2:=false;
      ВибранийПроцес:=1;
      ParBegin
            Процес1;
            Процес2;
      ParEnd;
End.
```

Рисунок 2.7 - Реалізація взаємовиключення відповідно алгоритму Деккера

Якщо X2= FALSE, то Процес1 пропускає тіло циклу очікування і входить у свою критичну ділянку. Якщо X2= TRUE, то Процес1 входить в

тіло свого циклу очікування. Тут аналізує змінну “ВибранийПроцес”, і якщо вибраним процесом є Процес1, то пропускає тіло свого циклу “if” і повторно виконує тіло свого циклу очікування до моменту, поки Процес2 не скине свій прапорець (поки не стане X2=FALSE). Якщо Вибраний-Процес=2, то Процес1 входить в тіло свого циклу “if”, де скидає свій власний прапорець (X1= FALSE), а потім блокується в циклі очікування, допоки ВибранийПроцес=2, тобто надає право Процесу2 увійти у свою критичну ділянку.

2.9 Апаратна реалізація взаємовиключення (команда *testandset)*

Алгоритм Деккера – це програмна реалізація задачі взаємовиключення. В деяких комп’ютерах вводиться спеціальна неподільна апаратна команда testandset (перевірити і встановити) - ***testandset (a, b).***

```
Program     Приклад Testandset;
    Var     Активний : логічний;
    procedure   Процес1;
    var     НеП1: логічний;      // НеП1 - першому процесу входити не можна
    begin
        while  true   do
        begin
            НеП1:=true;
            while  НеП1=true   do      testandset (НеП1, Активний);
            КритичнаДілянка1;
            ВибранийПроцес:=2;
            Активний:=false;
            ... інші оператори процесу 1 .....;
        end;
    end;
    procedure   Процес2;
    var     НеП2: логічний;      // НеП2 - другому процесу входити не можна
    begin
        while  true   do
        begin
            НеП2:=true;
            while  НеП2=true   do      testandset (НеП2, Активний);
            КритичнаДілянка2;
            Активний:=false;
            ... інші оператори процесу 2 .....;
        end;
    end;
    Begin
        Активний:=false;
        ParBegin
            Процес1;
            Процес2;
        ParEnd;
        End.
```

Рисунок 2.8 - Реалізація взаємовиключення за допомогою команди testandset

Команда testandset зчитує значення логічної змінної b, копіює його в a, і присвоює b значення TRUE. Приклад реалізації взаємовиключення за допомогою команди testandset наведений на рис. 2.8. В такому варіанті не виключено нескінченне відкладання, але ймовірність його дуже мала.

2.10 Семафори. Синхронізація за допомогою семафорів

Семафор - захищена змінна, значення якої може зчитуватись і мінятись за допомогою спеціальних операцій P,V та операцій ініціалізації.

Розрізняють двійкові та семафори з лічильником. *Двійкові* семафори можуть приймати значення '0' або '1'. Семафори з лічильником можуть приймати невід'ємні цілі значення. Семафори з лічильником корисні, якщо деякий ресурс виділяється з пула ідентичних ресурсів. Під час ініціалізації такого семафора в його лічильнику вказується кількісний покажчик об'єму ресурсів пула.

Операції над семафорами:

- *операція P(S):* якщо S>0 то S:=S-1
  інакше (очікувати на S);
- *операція V(S):* якщо (один або більше процесів очікують на S)
  то (дозволити одному з них продовжити роботу)
  інакше S:=S+1.

Операції P і V - неподільні, подібно команді testandset. Ділянки взаємовиключення за семафором S обмежуються операціями P(S) та V(S). Семафори та операції над ними можуть бути реалізовані як програмно, так і апаратно. Як правило, вони реалізуються в ядрі ОС, де виконують керування зміною станів процесів. За допомогою семафорів можна реалізувати багато задач синхронізації: взаємовиключення, блокування-відновлення, та інші.

Реалізація взаємодії «блокування-відновлення”

Дано два процеси, один з яких може виявити подію, на яку очікує другий процес, але не може виявити її самостійно (рис. 2.9 а).

Реалізація взаємодії “виробник-споживач”

Виробник - це процес, що генерує інформацію. Споживач - це процес, що використовує цю інформацію. Будемо вважати, що вони взаємодіють один з одним за допомогою змінної БуферЧисла (БЧ), і швидкості цих процесів різні. Якщо не вжити спеціальних заходів синхронізації, то інформація буде губитись (споживач повільний), або дублюватись (споживач швидкий). Змінні: ЧЗ - число занесено, ВД - виключний доступ, БЧ - буфер числа (рис. 2.9 б).

```
Programm
блокування_відновлення;
        var     подія : семафор;
procedure Процес1;
    begin
        інші_оператори;
        P(подія);
        інші_оператори;
    end;
procedure Процес2;
    begin
        інші_оператори;
        V(подія);
        інші_оператори;
    end;
BEGIN
    ініціалізація_семафора(подія,
0);
    ParBegin
        Процес1;
        Процес2;
     ParEnd;
END.
```

а)

```
Programm виробник_споживач;
    var ВД : семафор;
                БЧ : ціла;
                ЧЗ : семафор;
procedure процес_виробник;
    var НР : ціла;
    begin
        обчислення наступного
результату {НР};
        P(ВД);
        БЧ:=НР;
        V(ЧЗ);
    end;
procedure процес_споживач;
    var    НР : ціла;
    begin
        P(ЧЗ);
        НР:=БЧ;
        V(ВД);
        записати наступний
результат {НР};
    end;
BEGIN
    ініціалізація_семафора(ВД, 1);
    ініціалізація_семафора(ЧЗ, 0);
    ParBegin
        процес_виробник;
        процес_споживач;
     ParEnd;
END.
```

б)

Рисунок 2.9 - Синхронізація за допомогою семафорів

Реалізація семафорів і операцій над ними

Операції P і V з циклом активного очікування можуть бути реалізовані за допомогою алгоритму Деккера або команди testandset. Щоб виключити цикли активного очікування, команди операцій над семафорами реалізуються в ядрі ОС. Можлива реалізація неподільності операцій P і V простою забороною переривань (в однопроцесорних системах) [5].

2.11 Моніторoподібні засоби синхронізації

Алгоритм Деккера і семафори мають деякі недоліки. Наприклад, невірне використання цих примітивів може призвести до порушення працездатності системи паралельної обробки. Необхідні механізми, які б:

а) спрощували розв'язання складних задач паралельної обробки;

б) спрощували доведення коректності програм;

в) важко псувались або невірно використовувались.

В значній мірі ці проблеми вирішуються за допомогою моніторів. *Монітор - це засіб забезпечення паралелізму, який містить як дані, так і процедури, необхідні для реалізації конкретного розподілу ресурсу.* Щоб забезпечити виділення необхідного йому ресурсу, процес повинен звернутися до конкретної процедури монітора. В кожний момент часу лише один процес може увійти в монітор. Внутрішні дані монітора доступні лише з середини, що забезпечує розробку надійних програм користувача [4-5].

Вводяться примітиви: WAIT - очікування, SIGNAL - сигналізації. За командою WAIT створюється черга процесів, що очікують на деякий ресурс, якщо він зайнятий. SIGNAL виконується, коли деякий ресурс звільняється і повідомляє процеси, що очікували на цей ресурс, про можливість його зайняття. Процеси можуть знаходитись в режимі очікування за межами монітора з різних причин, тому введене поняття змінної умови, а саме:

WAIT(умова), SIGNAL(умова).

Контрольні питання

1. Дайте означення процесу.
2. Наведіть класифікацію процесів.
3. Охарактеризуйте стани процесу.
4. Що таке блок керування процесом?
5. Які операції над процесами виконує ОС?
6. Наведіть означення та класифікацію переривань в ОС.
7. Ядро ОС та його функції.
8. Наведіть типові задачі синхронізації.
9. Охарактеризуйте механізми синхронізації.
10. Що таке критична ділянка.
11. У чому недоліки реалізації задачі взаємовиключення за допомогою заборони переривань?
12. Дайте аналіз алгоритму Деккера.
13. Охарактеризуйте семафори та операції над ними.
14. Наведіть аналіз апаратної реалізація алгоритмів синхронізації.
15. Як виконується синхронізація процесів за допомогою семафорів?
16. Які переваги реалізації алгоритмів синхронізації за допомогою мікропрограмування?
17. Що таке тупики та причини їх виникнення?
18. Дайте характеристику моніторoподібних засобів синхронізації паралельних процесів.

3 Керування пам’яттю

Організація і керування основною, первинною або фізичною (реальною) пам’яттю обчислювальної машини – один з найважливіших факторів, який визначає побудову операційних систем.

3.1 Організація пам’яті

Пам'ять розділяється на байти, слова (декілька байт). Кожна комірка пам'яті має свою адресу. Множина адрес, доступних програмі називається *адресним простором*. Розглянемо деякі апаратно-реалізовані функції, які можуть бути корисними під час розробки ОС.

1. *Розшарування пам'яті*. Збільшується швидкість доступу до оперативної пам’яті. Сусідні за адресами комірки розміщуються в різних блоках пам'яті, що дає можливість звертатися одночасно до декількох комірок.
2. *Регістр-переміщення.* Забезпечує можливість динамічного переміщення програм в пам'яті.
3. *Захист пам'яті*. Виконується за допомогою граничних регістрів.
4. *Віртуальна пам'ять*. Дає можливість вказувати в програмі адреси, які не обов'язково відповідають фізичним адресам основної пам'яті. Віртуальні адреси під час роботи програми динамічно перетворюються за допомогою апаратних засобів в реальні адреси основної пам'яті.

3.2 Ієрархія пам'яті

Розрізняють такі види пам’яті:

- *зовнішню (вторинну)* пам'ять (ЗП);
- *основну (первинну)* пам'ять (ОП);
- *кеш-пам'ять* (КП).

Зв’язок між різними типами пам’яті показано на рис.3.1.

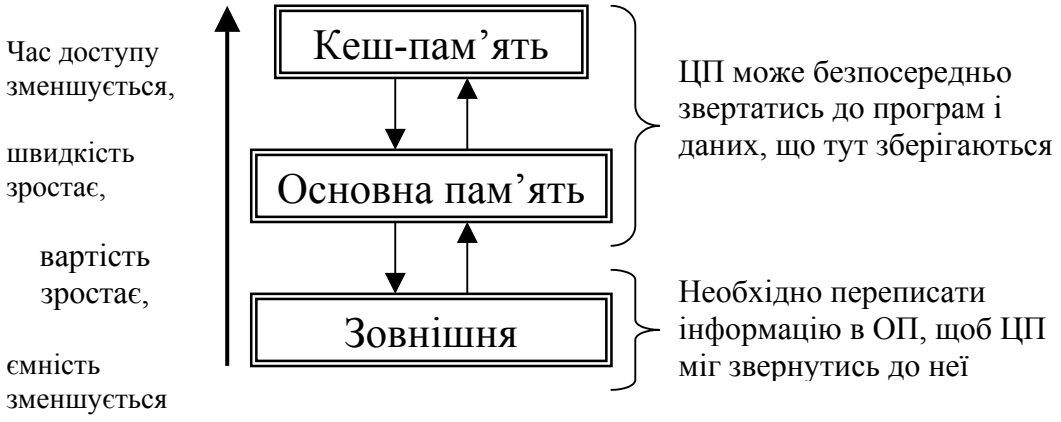


Рисунок 3.1 - Ієрархічна організація пам'яті

Кеш-пам'ять - найшвидкодійніша, але при цьому губиться час на переписування інформації з основної пам'яті до кеш-пам'яті [5].

3.3 Стратегії керування пам'яттю

Стратегії керування пам’яттю спрямовані на те, щоб забезпечити найкращі можливості використання ресурсів основної пам'яті.

Стратегії керування пам’яттю поділяються на декілька категорій:

1. ***Стратегії вибірки*** ставлять за мету визначити, коли необхідно записати черговий блок програми або даних в основну пам'ять. Ця категорія стратегій поділяється на:

- стратегії вибірки за запитом;
- стратегії випереджувальної вибірки.

2. ***Стратегії розміщення*** (рис. 3.2) визначають, в яке місце основної пам'яті необхідно помістити програму або дані. Вони поділяються на стратегії:

- першого придатного,
- найбільш придатного,
- найменш придатного.

Стратегія першого придатного - нові дані розміщуються в першій зустрінутій ділянці, що підходить за розмірами.

Стратегія найбільш придатного – програма розміщується в ділянці пам'яті, що найточніше підходить за розміром.

Стратегія найменш придатного - програма займає максимально вільну ділянку пам'яті.

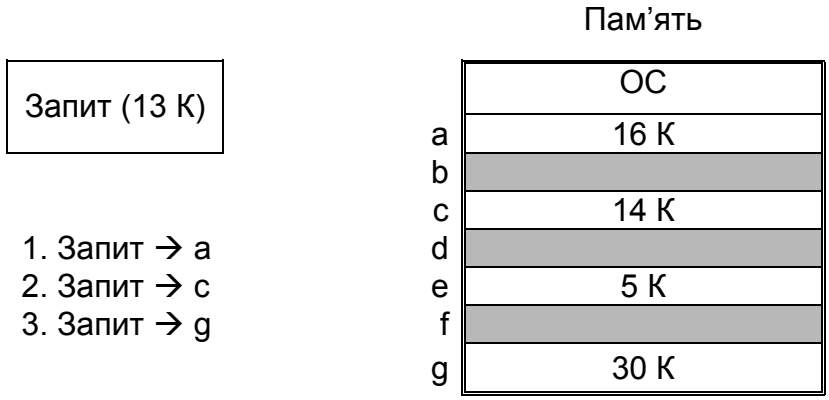


Рисунок 3.2 - Стратегії розміщення інформації в пам'яті (1 - першого придатного, 2 - найбільш придатного, 3 - найменш придатного)

3. ***Стратегії заміщення*** визначають блок програми або даних, який треба вивести з основної пам'яті, щоб звільнити місце для нових даних або програм.

3.4 Зв'язний і незв'язний розподіл пам'яті

При зв'язному розподілі пам'яті кожна програма записується в один суцільний блок пам'яті. При незв'язному розподілі програма розбивається на ряд блоків (сегментів), які можуть розміщуватись в ділянках пам'яті, які не обов'язково сусідні.

Системи віртуальної пам'яті передбачають незв'язний розподіл пам'яті. Переваги незв'язного розподілу пам'яті в тому, що може виконуватись програма, яку неможливо розмістити в одному блоці. Зв'язний розподіл пам'яті для однопрограмної системи наведено на рис.3.3.

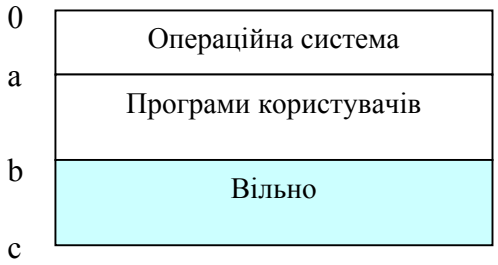


Рисунок 3.3 - Зв'язний розподіл пам'яті в однопрограмних ОС

Якщо розмір програми перевищує розмір пам'яті, то використовують так звані оверлейні сегменти. Ручна організація оверлейного режиму не забезпечує модифікації програми і вимагає складного планування. Проблеми оверлейних сегментів відпали завдяки використанню віртуальної пам'яті.

Захист пам'яті в однопрограмних ОС можна організувати за допомогою одного граничного регістра, значення якого порівнюється з адресами, до яких відбувається звернення з програми користувача. Для звернення до ОС використовується SVC-команди. По іншому до ОС звернутись не можна. Тобто, захищається тільки область пам'яті, в якій знаходяться програми ОС (рис.3.4).

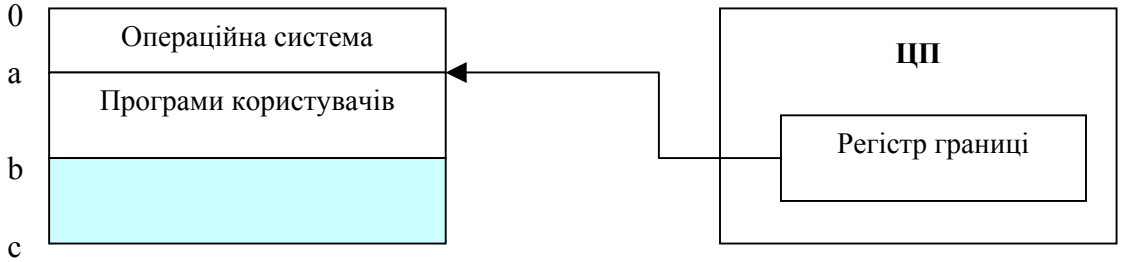


Рисунок 3.4 - Захист пам'яті в однопрограмних ОС

3.5 Мультипрограмування з фіксованими розділами

Трансляція і завантаження в абсолютних адресах

Пам'ять розбивається на розділи фіксованого розміру. В кожному розділі розміщується одне завдання. ЦП швидко переключається з завдання на завдання, створюючи ілюзію одночасного виконання. Розділи

мають фіксований розмір. До кожного розділу утворюється своя черга програм, які можуть виконуватись лише в цьому розділі (рис.3.5 а).

Недоліком такої організації пам’яті є неефективне використання ресурсів, оскільки трансляція відбувається в абсолютних адресах, і програма може виконуватись лише в одному розділі.

Трансляція і завантаження переміщуваних модулів

Існує одна черга завдань, які можуть надходити до будь-якого вільного розділу. Однак, існуючі транслятори та завантажувачі є значно складнішими (рис.3.5.б).

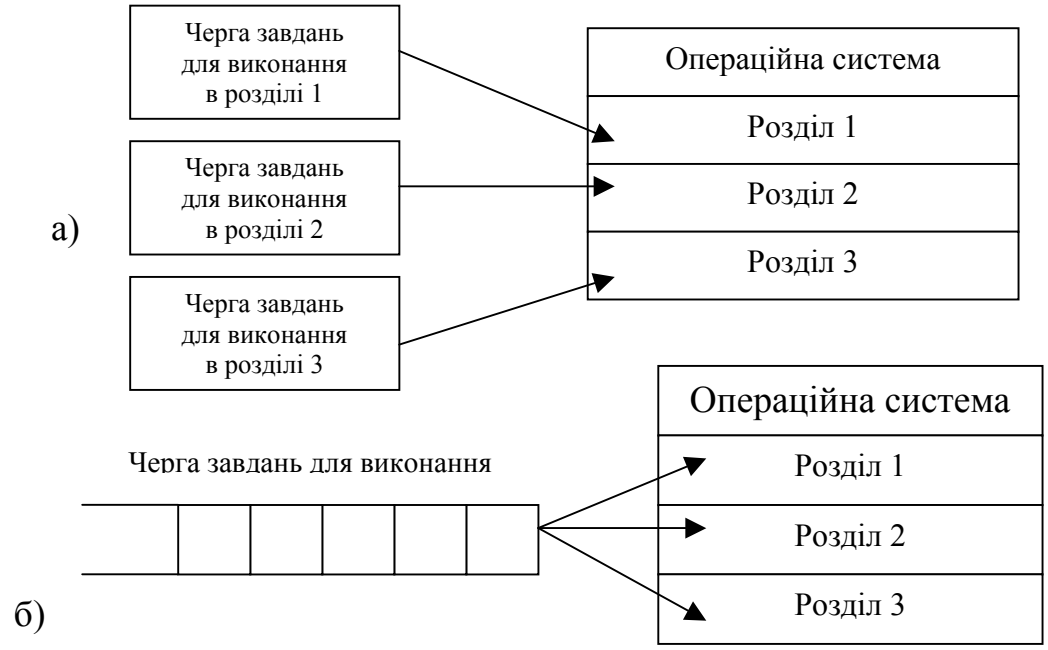


Рисунок 3.5 - Мультипрограмування з фіксованими розділами
(а - трансляція і завантаження програм в абсолютних адресах;
б – трансляція і завантаження модулів, що переміщуються)

Захист пам’яті в мультипрограмних системах

В мультипрограмних системах зі зв'язним розподілом пам'яті захист пам'яті реалізується за допомогою декількох граничних регістрів. Програми, які хочуть звернутись до послуг ОС, використовують SVC-команди.

3.6 Мультипрограмування з змінними розділами при зв'язному розподілі пам'яті

За такого підходу не дотримуються фіксованих границь розділів, а навпаки, кожному завданню виділяється стільки пам'яті, скільки необхідно. Однак, коли завдання завершуються, утворюються дірки (фрагментація пам'яті). Долають це явище такими способами:

- об’єднання вільних сусідніх ділянок пам'яті (злиття);
- ущільнення пам'яті або "збір сміття" (рис. 3.6).

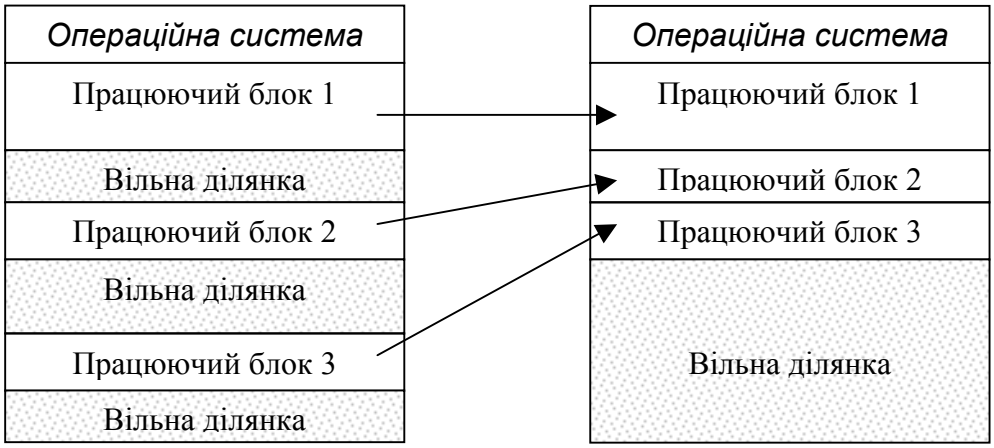


Рисунок 3.6 - Ущільнення пам'яті

До недоліків ущільнення пам'яті слід віднести те, що воно забирає ресурси системи: під час проведення ущільнення збільшується час відповіді системи; необхідно зберігати дані про розміщення програм.

3.7. Мультипрограмування зі свопінгом

Мультипрограмування зі свопінгом - це схема розподілу пам'яті, під час реалізації якої програми користувача не зберігаються в основній пам'яті до їх завершення. Пам'ять на короткий період виділяється одному завданню, потім це завдання виводиться і замість нього вводиться інше завдання. Перші системи з розподілом часу були системами зі свопінгом. На основі систем зі свопінгом були створені системи зі сторінковою організацією. Розрізняють системи з свопінгом, в яких в основній пам'яті розміщується:

- одна програма користувача;
- декілька програм користувача.

Слід зауважити, що фрагментація пам’яті має місце в будь-якій обчислювальній системі незалежно від типу організації пам'яті [4-5].

Контрольні питання

1. Яка апаратна підтримка необхідна для підвищення ефективності використання основної пам’яті?
2. Охарактеризуйте ієрархічну організацію пам’яті.
3. Наведіть стратегії керування пам’яттю.
4. Дайте аналіз зв’язного та незв’язного розподілу пам’яті.
5. Наведіть приклади розподілу пам’яті в однопрограмних ОС.
6. Які методи захисту пам’яті ОС застосовуються в однопрограмних та мультипрограмних системах?
7. Охарактеризуйте мультипрограмування з фіксованими розділами.
8. Охарактеризуйте мультипрограмування з змінними розділами.
9. Які особливості систем з стопінгом?

4 Організація і керування віртуальною пам’яттю

Вперше віртуальна пам’ять була застосована в машині ATLAS в Манчестерському університеті в 1960 р. Існує три способи реалізації віртуальної пам'яті:

- сторінкова;
- сегментна;
- комбінована.

В системах віртуальної пам'яті адреси, що формуються програмами, не обов'язково збігаються з адресами реальної основної пам'яті.

Термін *віртуальна пам’ять* асоціюється з можливістю адресувати простір пам’яті набагато більший, ніж ємність первинної (реальної, фізичної) пам’яті конкретної обчислювальної машини.

4.1 Основні концепції

Сутність концепції віртуальної пам'яті (рис.4.1) полягає в тому, що адреси, які формуються програмами, відокремлюються від адрес реальної пам'яті. *Віртуальні адреси* - це адреси, на які посилається процес, що виконується. Адреси, що існують в реальній основній пам'яті, називаються *реальними адресами*. Діапазон віртуальних адрес називається віртуальним адресним простором. Для встановлення відповідності між реальними і віртуальними адресами застосовуються механізми динамічного перетворення адрес. Суміжні адреси віртуального простору не обов'язково будуть суміжними в реальній пам'яті (штучна суміжність), але користувач звільнений від необхідності враховувати розміщення в реальній пам'яті. Віртуальний адресний простір завжди значно ширший за реальний адресний простір.

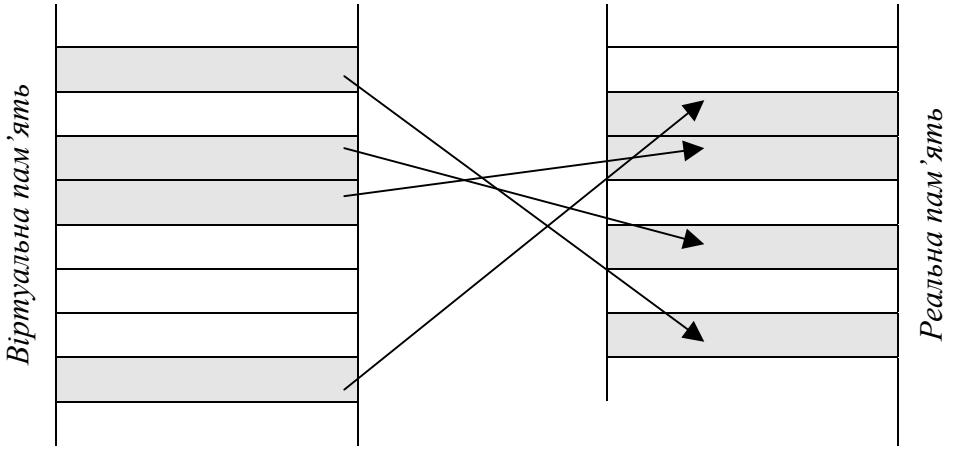


Рисунок 4.1 - Концепція віртуальної пам'яті

4.2 Механізм відображення віртуальних адрес в реальні

Механізм динамічного перетворення адрес повинен вести таблиці, які показували б які комірки віртуальної пам'яті знаходяться в реальній пам'яті і де саме. Якщо б відображення виконувалось побайтно, то ці таблиці займали б більше місця, ніж самі процеси. Тому елементи інформації групуються в блоки і система слідкує, в яких місцях реальної пам'яті знаходяться блоки віртуальної пам'яті. Розміри блоків вибираються з урахуванням декількох вимог, що суперечать одна одній.

Формат віртуальної адреси в системі поблочного відображення приведений на рис. 4.2.

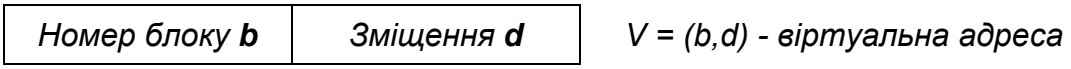


Рисунок 4.2 - Формат віртуальної адреси

Якщо всі блоки мають однаковий розмір, то така організація віртуальної пам'яті називається *сторінковою*, а блоки сторінками. Якщо блоки можуть бути різних розмірів, то така організація пам'яті називаються *сегментною*, а блоки сегментами.

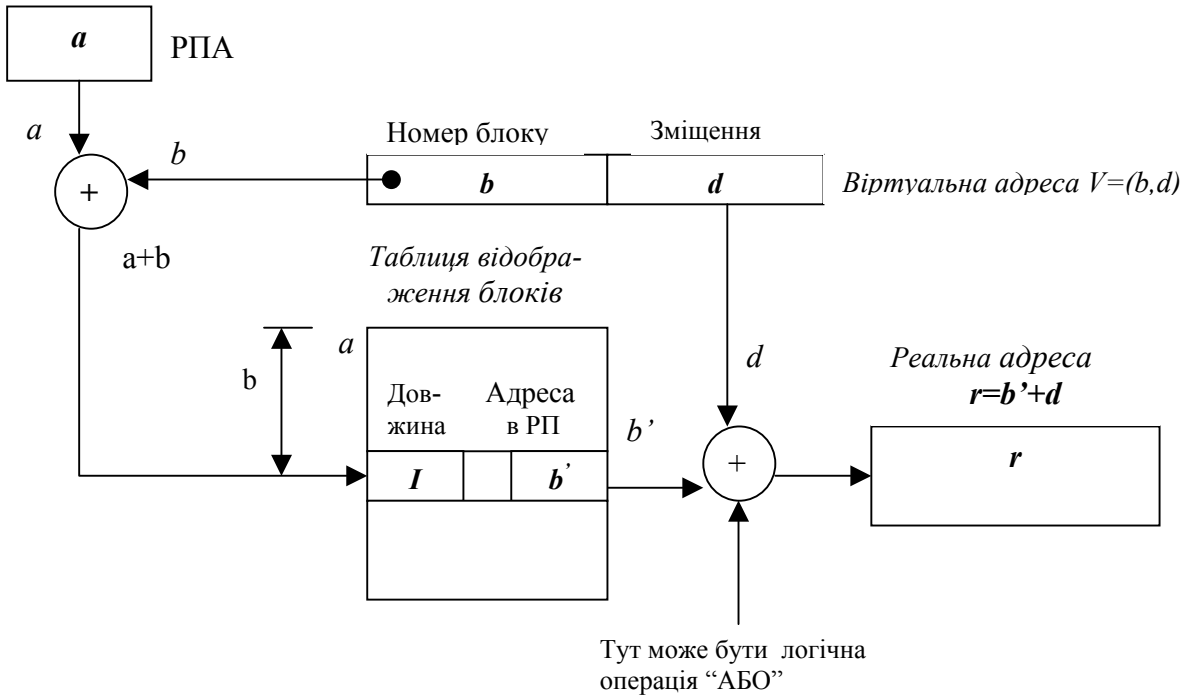


Рисунок 4.3 - Перетворення віртуальної адреси в реальну при поблочному відображенні (РП – реальна пам'ять; b' – початкова адреса блоку в реальній пам'яті; I – довжина блоку).

Кожен процес має свою таблицю відображення адрес, яку система веде в реальній пам'яті. Реальна адреса *a* цієї таблиці завантажується до спеціального регістра ЦП, який називається *регістром початкової адреси* (РПА) таблиці відображення блоків (ТВБ). ТВБ містить по одному

рядку для кожного блоку процесу, причому блоки йдуть в послідовному порядку (рис.4.3).

4.3 Сторінкова організація віртуальної пам'яті

Формат віртуальної адреси в системі з сторінковою організацією пам’яті має такий вигляд:

Номер сторінки ***p***	*Зміщення* ***d***

V = (p,d) - віртуальна адреса

Процес виконується до тих пір, поки його поточна сторінка знаходиться в основній пам'яті. Сторінки переписуються з зовнішньої пам'яті в основну і розміщуються в блоках, які називаються *сторінковими кадрами* і мають розмір сторінки. Сторінка, яка надходить з зовнішньої пам'яті, може бути розміщена в будь-якому вільному сторінковому кадрі. Сторінкові кадри починаються з адрес первинної пам'яті, кратних, як правило, розміру сторінки.

Перетворення адрес сторінок методом прямого відображення

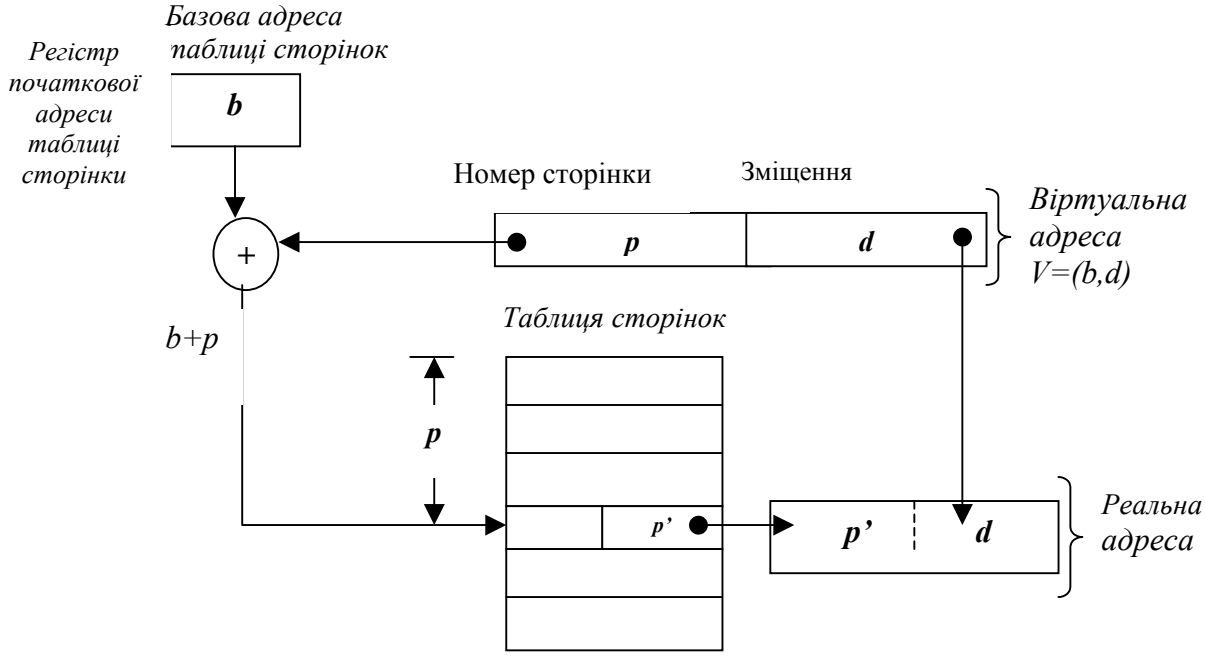


Рисунок 4.4 - Перетворення адресів сторінок методом прямого відображення

Реальна адреса формується методом конкатенації p' і d. Формат рядка таблиці сторінок може мати такий вигляд:

Біт-ознака присутності сторінки	*Адреса зовнішньої пам'яті (якщо сторінки немає в реальній пам'яті)*	*Номер сторінкового кадру (якщо сторінка знаходиться в реальній пам'яті)*
I	S	p’

Якщо I=0, то сторінки немає в реальній пам'яті, а якщо до неї будуть звертання, то формується переривання за відсутністю сторінки і необхідна сторінка завантажується в реальну пам’ять. Якщо I=1, то сторінка знаходиться в реальній пам'яті

Недоліком такого перетворення є те, що швидкість виконання програми знижується майже вдвічі, оскільки необхідно виконувати два цикли зчитування пам'яті.

Перетворення адрес сторінок асоціативним відображенням

Одним з методів прискорення динамічного перетворення адрес є застосування асоціативної пам’яті для розміщення таблиці перетворення сторінок (рис.4.5), оскільки асоціативна пам’ять значно швидша ніж звичайна пам’ять з довільним доступом.

Недоліком асоціативного перетворення є дуже велика вартість.

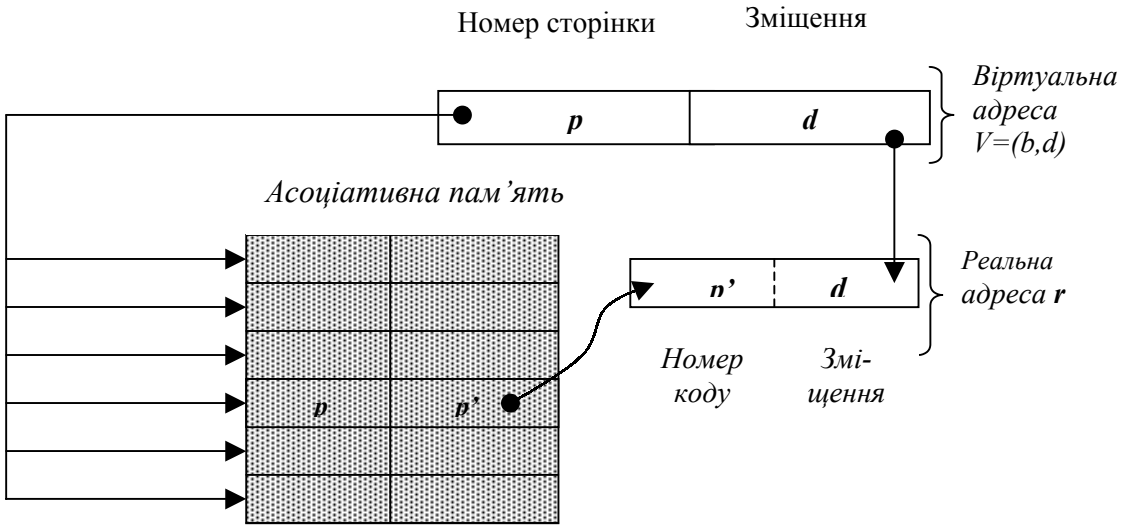


Рисунок 4.5 - Перетворення адрес сторінок асоціативним відображенням

Перетворення адрес сторінок комбінованим відображенням

Цей підхід передбачає використання асоціативної пам’яті для збереження тільки тих рядків повної таблиці відображення сторінок для процесу, до яких були звертання протягом останнього інтервалу часу (рис.4.6). Операція (b+p) виконується лише тоді, коли сторінка не знайдена в асоціативній таблиці. Швидкодія складає 90% від повної асоціативної пам'яті.

Колективне використання програм і даних в системах зі сторінковою організацією

В системах з розподілом часу часто багато користувачів виконують одні й ті ж самі програми. Якщо кожному користувачу виділити індивідуальні копії цих програм, то значна частина пам'яті витрачалась би марно. Рішенням є спільне використання сторінок, які можна розділяти.

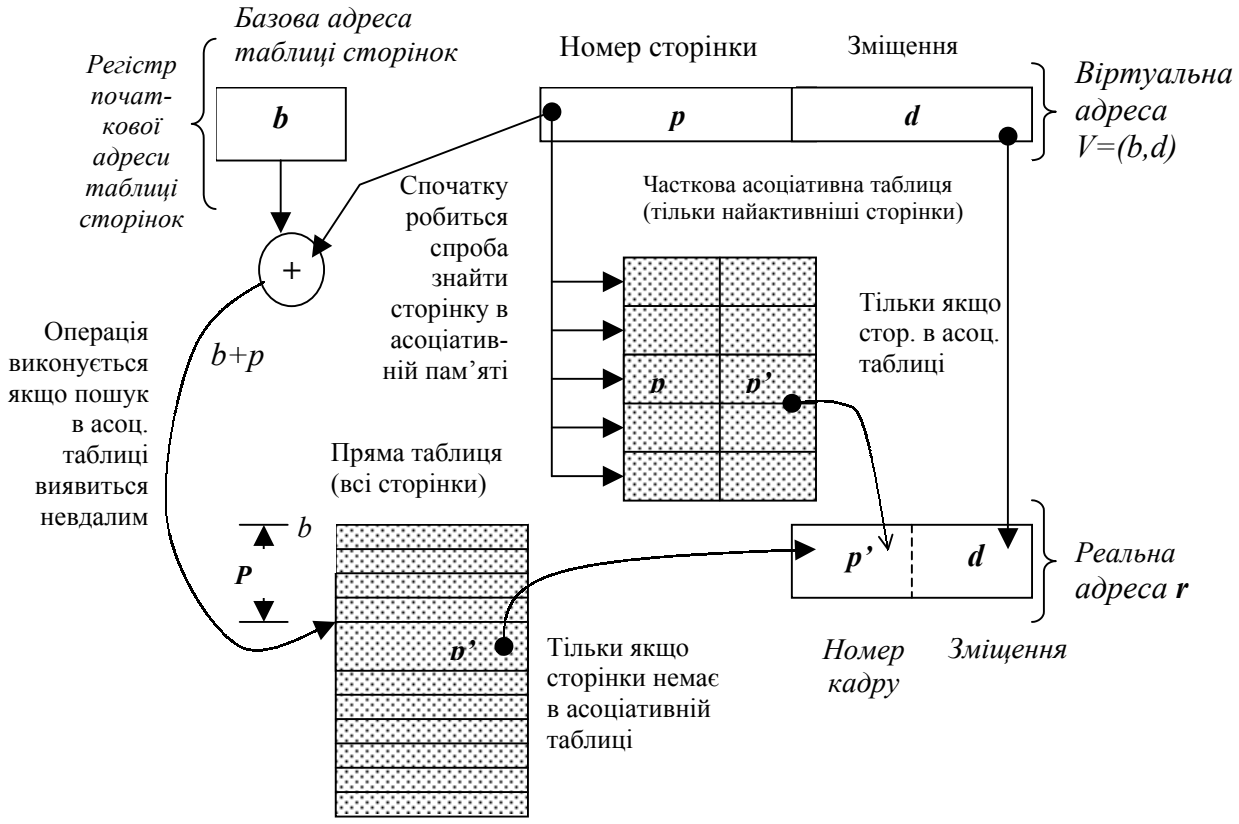


Рисунок 4.6 - Перетворення адрес сторінок комбінованим відображенням

Програми складаються з області процедур і даних. Колективно можна використовувати лише незмінні процедури (реєнтерабельні) і незмінні дані. Кожну сторінку необхідно ідентифікувати як таку, що використовується колективно або ні. Ця ознака може міститися в таблиці сторінок. Спільне використання досягається за рахунок адресації в таблиці сторінок різних процесів на один і той самий сторінковий кадр.

4.4 Сегментна організація віртуальної пам'яті

В системах з сегментною організацією пам'яті програми можуть займати багато блоків основної пам'яті, причому не обов'язково однакового розміру. Блоки можуть розміщуватись один біля одного або ні, перекриватись частково або повністю.

Процес виконується тоді, коли сегмент знаходиться в основній пам'яті. Сегменти передаються до основної пам'яті повністю. В загальному випадку перетворення адрес сегментів може бути прямим, асоціативним, комбінованим. На рис.4.7 подано випадок прямого перетворення віртуальних адрес в реальні.

Стратегії розміщення в сегментній організації віртуальної пам'яті ідентичні стратегіям розміщення в мультипрограмуванні із змінними розділами. Найчастіше використовуються стратегії першого придатного і найбільш придатного.

Формат рядка таблиці сегментів може мати такий вигляд:

Ознака наявності сегмента в пам'яті	*Адреса сегмента в зовнішній пам'яті*	*Довжина сегмента*	*R, W, E, A - біти ознак захисту: R - доступ для читання; W - доступ для запису; E - доступ для виконання; A - доступ для доповнення.*				*Базова адреса сегмента, якщо сегмент знаходиться в ОП*
I	A	L	R	W	E	A	S'

Якщо під час виконанні процесу відбувається звернення до будь-якого сегмента, і біт ***I*** у відповідному рядку таблиці сегментів дорівнює нулю, то формується переривання за відсутністю даного сегмента в основній пам'яті і ОС завантажує його до основної пам'яті. Після завантаження сегмента обробка адреси продовжується. Зміщення ***d*** порівнюється з довжиною сегмента ***L***. Якщо $d>L$, то генерується переривання по виходу за межі сегмента і ОС припиняє виконання даного процесу. Якщо ***d*** в межах сегмента, то виконується контроль по біту захисту, щоб переконатись, що операція доступу дозволена. Якщо це так, то формується реальна адреса $r=S'+d$, яка відповідає віртуальній адресі $V(S, d)$. Якщо доступ не дозволений, то генерується переривання по захисту і ОС припиняє виконання процесу.

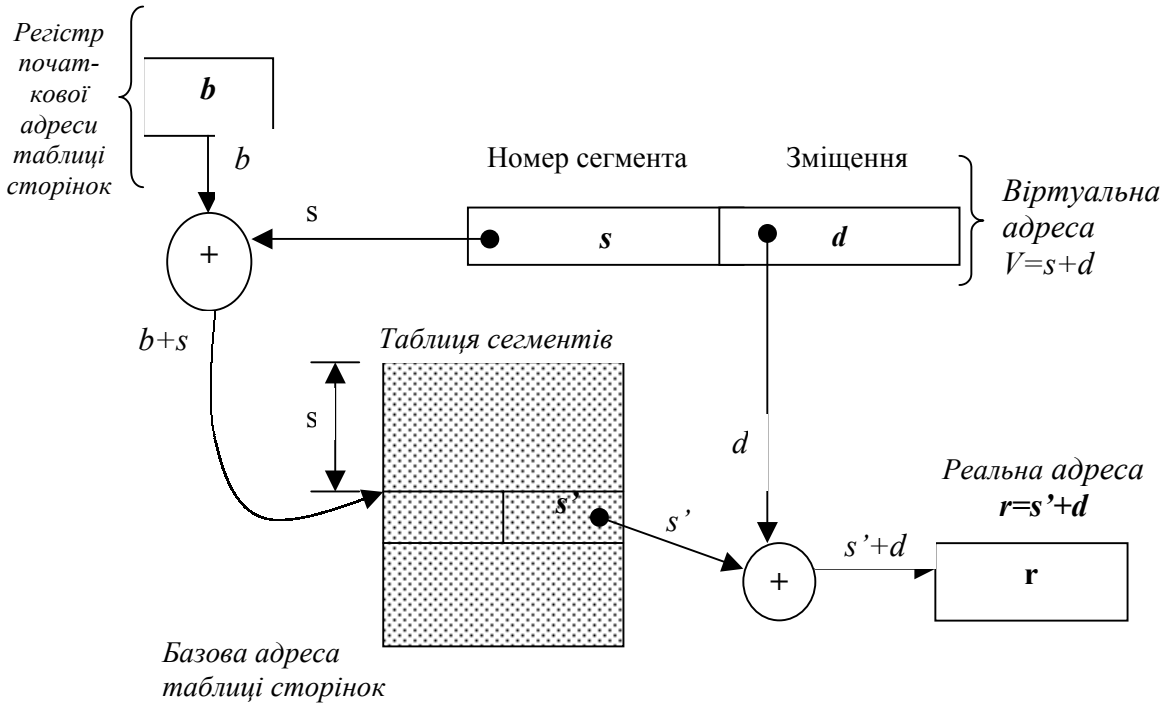


Рисунок 4.7 - Перетворення віртуальної адреси в сегментній системі

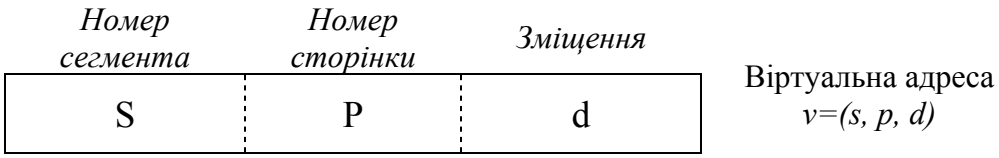
Колективне використання програм і даних в системах із сегментною організацією віртуальної пам'яті

Однією з основних переваг систем з сегментною організацією є те, що значно спрощується колективне використання програм і даних у порівнянні з системами зі сторінковою організацією пам'яті. Це відбувається через зменшення таблиць сегментів порівняно з розміром таблиць сторінок.

Наприклад, якщо масив займає 3,5 сторінки, то для кожної сторінки треба окремо вказувати її колективне використання, тоді як в системах з сегментною організацією можна об'єднати їх в один сегмент і колективний доступ вказувати лише раз.

4.5 Комбінована сторінково-сегментна організація пам'яті

У системах із сторінково-сегментною організацією застосовується трикомпонентна (тривимірна) адресація, тобто віртуальна адреса визначається як функція трьох змінних *V=(s; p; d)*, де *s* - номер сегмента, p - номер сторінки, *d* - зміщення в межах сторінки. Формат віртуальної адреси такий:



Розглянемо динамічне перетворення адрес в системах із сторінково-сегментною організацією (рис.4.8), а саме: візьмемо приклад асоціативно-прямого відображення. Спочатку в асоціативній таблиці сторінок система пробує знайти рядок з параметрами (s, p). Якщо шукана сторінка відсутня в асоціативній таблиці, то перетворення адрес виконується способом повного прямого відображення.

Структура таблиць, які система веде при сторінково-сегментній організації приведена на рис. 4.9. На самому верхньому рівні знаходиться *таблиця процесів* (одна на всю систему), яка містить по рядку для кожного процесу відомого системі, і цей рядок вказує на *таблицю сегментів* цього процесу (по одній на кожен процес). Кожний рядок таблиці сегментів процесу вказує на *таблицю сторінок* відповідного сегмента, а кожний рядок таблиці сторінок вказує або на сторінковий кадр, в якому розміщена дана сторінка, або на адресу зовнішньої пам'яті, де можна знайти цю сторінку.

В сторінково-сегментній системі організації віртуальної пам'яті переваги від колективного використання програм і даних більші, ніж при чисто сегментній організації, оскільки в таблицях сегментів для різних процесів є рядки, які вказують на одну й ту саму таблицю сторінок.

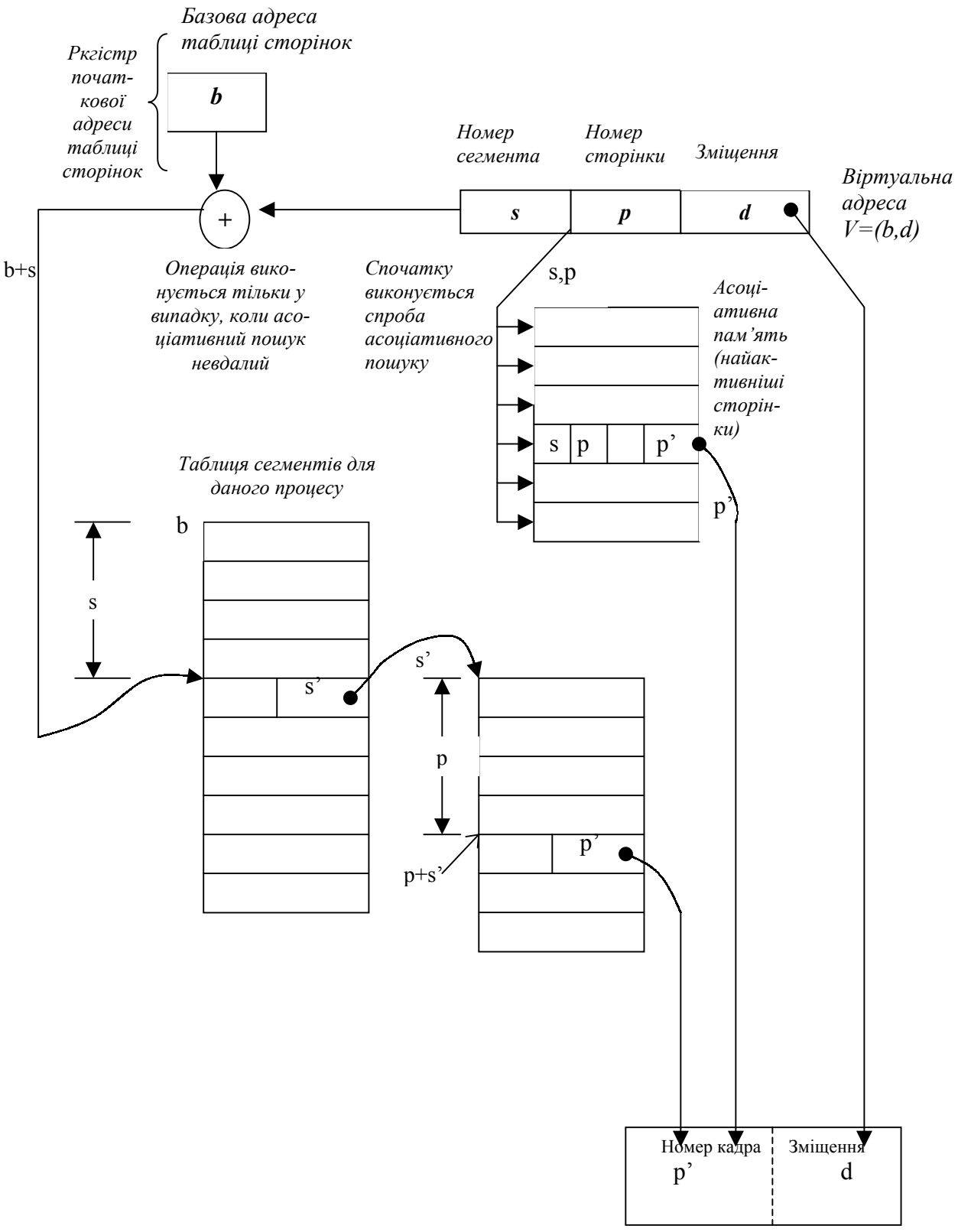


Рисунок 4.8 - Перетворення віртуальної адреси шляхом комбінованого асоціативно-прямого відображення в сторінково-сегментній системі

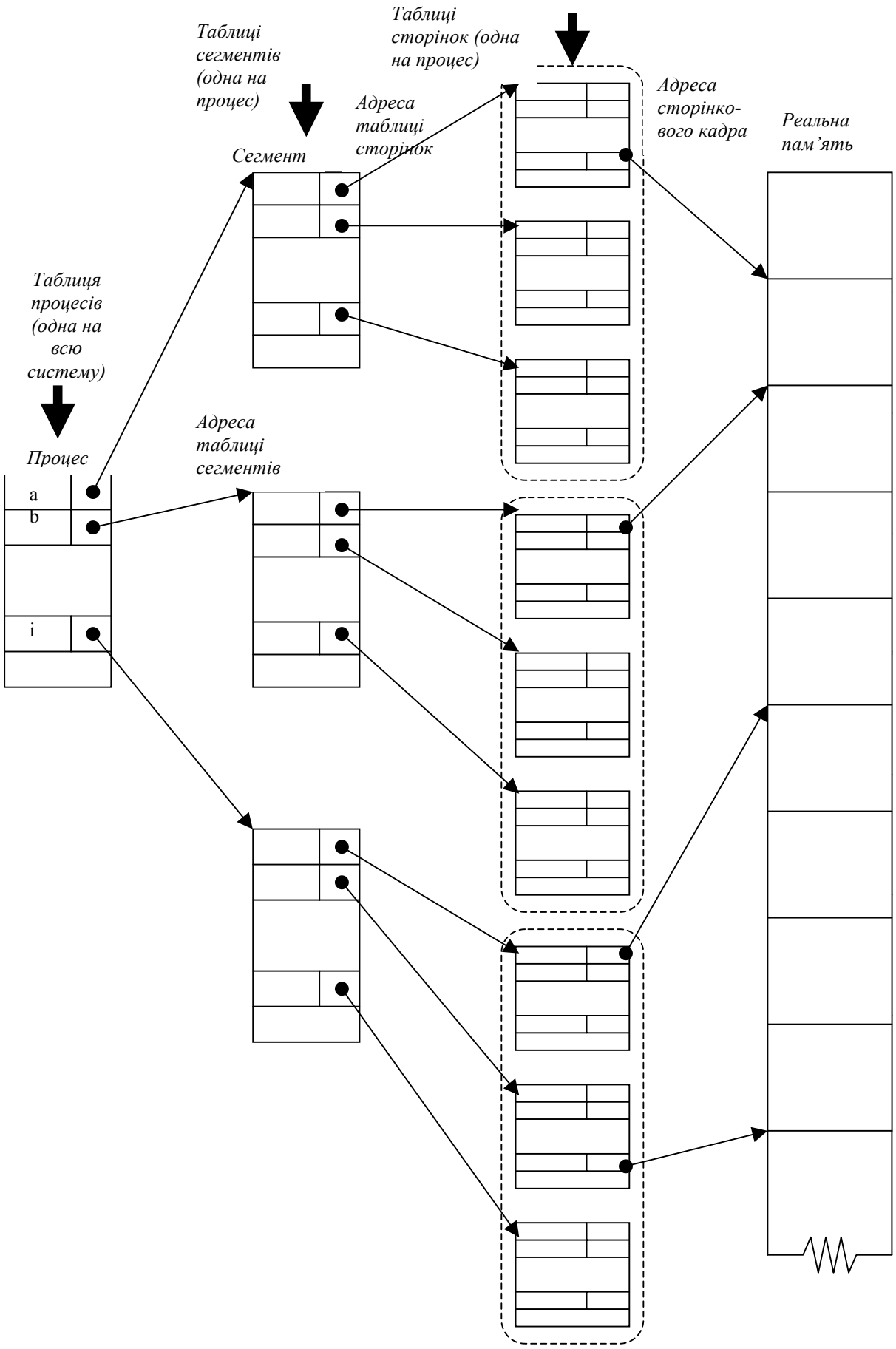


Рисунок 4.9 - Структура таблиць для сторінково-сегментної системи

4.6 Стратегії керування віртуальною пам’яттю

1. ***Стратегії вибірки*** (вштовхування). Їх мета – визначити, в який момент слід переписати сторінку або сегмент із вторинної пам’яті в первинну. *Вибірка за запитом* передбачає, що система очікує посилання на сторінку або сегмент, і тільки дочекавшись цього посилання від працюючого процесу, переписує дану сторінку або сегмент у первинну пам’ять. При *вибірці з випередженням* передбачається, що система заздалегідь намагається визначити, до яких сторінок або сегментів звернеться процес. Якщо вірогідність звернення висока і в первинній пам’яті є в наявності вільне місце, то відповідні сторінки або сегменти будуть переписані в первинну пам’ять ще до того, як до них відбудеться звернення.

2. ***Стратегії розміщення.*** В системах зі сторінковою організацією розміщення вирішуються просто. Будь-яка сторінка, що надходить із зовнішньої пам'яті, розміщується у вільний сторінковий кадр, бо всі сторінки одного розміру. В системах з сегментною організацією потрібні стратегії розміщення, розглянуті вище, в розділі, що стосувався мультипрограмування із змінними розділами.

3. ***Стратегії заміщення*** (виштовхування). Їх основна мета - вирішити, яку сторінку або сегмент необхідно видалити з основної пам'яті, щоб звільнити місце для сегмента або сторінки, що надходять, якщо первинна пам’ять повністю зайнята. Розглянемо такі стратегії заміщення сторінок:

- Принцип оптимальності - необхідно заміняти ту сторінку, до якої в подальшому не буде звертань протягом довгого часу. Недоліком цього принципу є те, що його важко реалізувати, оскільки важко прогнозувати майбутнє.
- Заміщення випадкової сторінки - застосовується рідко, бо можна замінити сторінку, до якої багато звертань.
- Принцип FIFO - "виштовхування" першої сторінки, що надійшла, тобто тієї, яка знаходилась в пам’яті найдовше. Однак, цей принцип може призводити до заміщення активних сторінок. Той факт, що сторінка постійно знаходиться в оперативній пам'яті, може свідчити про те, що вона постійно в роботі.
- Принцип LRU - виштовхування сторінки, яка довше всіх не використовується. Цей принцип використовується рідко. Може статися так, що сторінка, до якої не було звернень, може стати потрібною.
- Принцип LFU - виштовхування сторінки, яка використовується рідше за інші. Враховується інтенсивність використання сторінок, але інтенсивність використання сторінки може змінитись в майбутньому.

- Принцип NUR - виштовхування сторінки, яка не використовувалась останнім часом. Дуже близька стратегія до LRU. Вводяться два апаратних біти:

а) *біт-ознака звернення* = 0, якщо до сторінки не було звернень і
= 1, якщо звернення були;

б) *біт-ознака модифікації* = 0, якщо сторінка не змінювалась і
= 1, якщо сторінка змінювалась.

Біти-ознака звернення періодично скидаються в "нуль" операційною системою. Алгоритм NUR передбачає існування чотирьох груп сторінок:

група 1 - звертань і модифікацій не було;
група 2 - звертань не було, модифікації були;
група 3 - звертання були, модифікацій не було;
група 4 - звертання і модифікації були.

Сторінки груп з меншими номерами виштовхуються в першу чергу.

4.6.1 Локальність

Більшість стратегій керування пам'яттю базуються на концепції локальності. Її сутність в тому, що розподіл запитів процесів на звертання до пам'яті має, як правило, нерівномірний характер з високим ступенем локальної концентрації. Розрізняють часову і просторову локальність.

Часова локальність означає, що до комірок пам'яті, до яких недавно проводилось звертання, з великою ймовірністю будуть звертання в найближчому майбутньому з таких причин:

- наявність програмних циклів;
- наявність підпрограм;
- наявність стеків;
- наявність змінних, що використовуються як лічильники для накопичення сум.

Просторова локальність означає, що якщо були звертання до деякої комірки пам'яті, то з великою ймовірністю можна очікувати звертання до сусідніх комірок. Причини цього такі:

- організація даних у вигляді масивів;
- послідовне виконання коду програми;
- тенденція програмістів розміщувати поруч описи зв'язаних змінних.

Наслідком локалізації звернень до пам’яті є те, що програма може ефективно виконуватись, коли в пам'яті знаходиться множина найбільш популярних сторінок. На основі вивчення властивостей локальності Денінг сформулював *теорію робочих множин:* допоки в основній пам'яті є множина найбільш інтенсивно використовуваних сторінок процесу, частота переривань за відсутності сторінки змінюється мало.

4.6.2 Робочі множини (підмножини)

Робоча множина - це множина сторінок, до яких процес активно звертається протягом деякого відрізку часу свого виконання. Для ефективного виконання процесу необхідно, щоб його робоча множина знаходилась в первинній пам'яті. В іншому випадку виникає режим інтенсивної підкачки сторінок, причому це можуть бути одні і ті самі сторінки. Такий режим називається *пробуксовкою (трешингом)*.

Стратегії керування пам'яттю у відповідності з робочими множинами спрямовані на те, щоб робоча множина була в оперативній пам'яті. Рішення про те, чи необхідно додати новий процес до набору активних, приймається з урахуванням того, чи є достатньо пам'яті для розміщення робочої множини сторінок цього процесу.

Робоча множина сторінок W(t,w) в момент часу t є набір сторінок до яких процес звертався протягом інтервалу [t-w; t]. Змінна w називається розміром вікна робочої множини (рис.4.10 а).

Залежність розміру робочої множини від розміру вікна показано на рис. рис. 4.10 б.

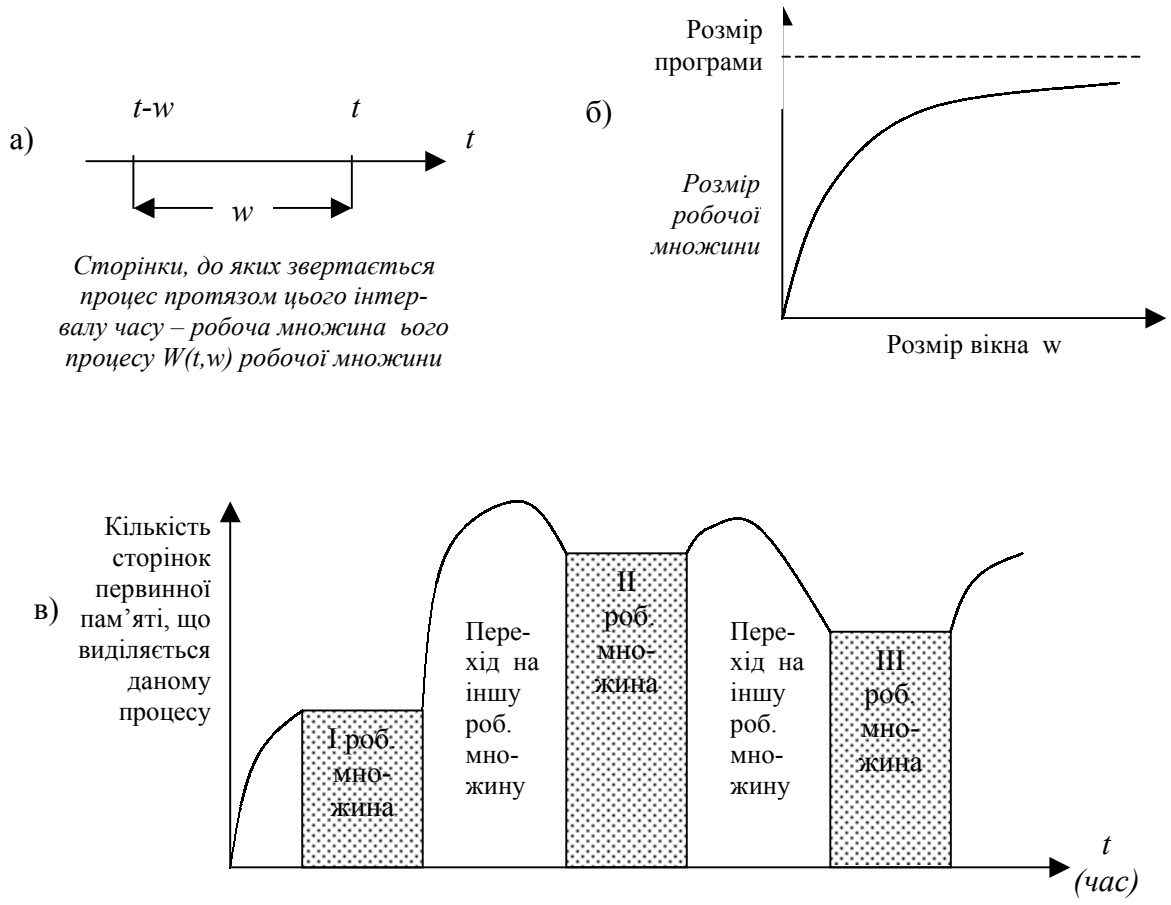


Рисунок 4.10 - Керування пам'яттю на основі робочих множин

Реальна робоча множина процесу – це множина сторінок, які повинні знаходитись в первинній пам’яті, щоб процес міг ефективно виконуватись. Під час роботи процесу його робоча множина динамічно змінюється.

На рис.4.10.в показано, як міг би використати первинну пам’ять процесор керуючи пам’яттю на основі робочих множин. Спочатку, оскільки процес вимагає сторінки своєї робочої множини не всі разом, а по одній, він поступово отримує достатній об’єм пам’яті для своєї поточної робочої множини. Потім на деякий період часу цей об’єм стабілізується, оскільки процес інтенсивно звертається до сторінок своєї першої робочої множини. З часом процес перейде на другу свою робочу множину. Спочатку крива на графіку підніметься вище рівня другої множини, оскільки швидко виконується підкачка сторінок нової робочої множини (при цьому система не може миттєво визначити, розширює свою множину процес чи переходить на іншу). Після того як процес почне стабільно звертатись до сторінок наступної робочої множини, система скорочує об’єм пам’яті виділений процесу до числа сторінок другої робочої множини. Тобто, існують сплески, коли відбувається перехід з однієї робочої множини до іншої, а потім система адаптується до цього переходу.

4.6.3 Підкачка сторінок по запиту

За використання саме такої стратегії підкачки сторінок говорять такі аргументи:

- шлях, який вибере програма, передбачити неможливо;
- в оперативній пам'яті будуть знаходитись лише необхідні сторінки;
- накладні витрати на визначення сторінок, які потрібно передати до оперативної пам'яті, мінімальні.

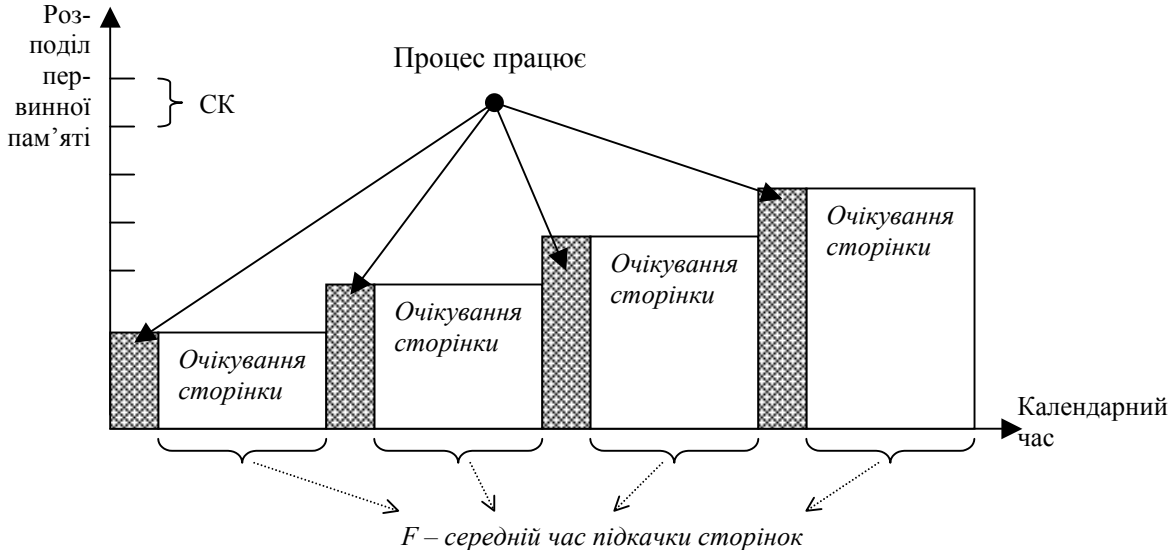


Рисунок 4.11 - Добуток “простір•час” при підкачці сторінок за запитом (СК – сторінковий кадр)

Недоліки підкачки сторінок за запитом такі: великі періоди очікування на підкачку сторінок, тобто добуток “простір•час” (рис.4.11) занадто великий.

Добуток “простір•час” - це комплексний покажчик, що відображає і об’єм, і час використання пам’яті процесом. Зменшення цього добутку за рахунок періоду очікування процесом потрібних йому сторінок є головною метою всіх стратегій керування пам’яттю.

4.6.4 Підкачка сторінок з упередженням

Система намагається заздалегідь передбачити, які сторінки потрібні процесу, а потім, коли в основній пам’яті з’являється вільне місце, завантажує туди підготовлені сторінки. Поки процес працює зі своїми поточними сторінками, система вже переглядає сторінки, готує їх до використання.

Переваги цього методу полягають в тому, що

- якщо правильно вибрати рішення з вибірки сторінки, то час виконання процесу значно зменшиться;
- оскільки апаратура дешевіє, то наслідки неоптимальних рішень стають менш важливими. Можна придбати додаткову пам’ять, щоб забезпечити накопичування додаткових сторінок, які при даному механізмі підкачки будуть передаватись в основну пам’ять [5].

4.6.5 Звільнення сторінок та їх розмір

Сторінки, які більше не потрібні, повинні якимось чином виводитися з робочих множин. Існує деякий період часу, під час якого непотрібні сторінки все ж таки ще залишаються в основній пам’яті. Користувач може за власною ініціативою дати повідомлення про видалення сторінки з пам’яті і звільнити відповідний сторінковий кадр. Це могло б виключити недоцільне використання пам’яті та прискорити виконання програми. Але більшість користувачів взагалі не приймають рішень на системному рівні (частіше вони навіть не замислюються над тим, що таке сторінка). Отже, такий спосіб звільнення сторінок майже нереальний.

Можна сподіватись, що реально ця проблема буде вирішена за допомогою компіляторів та операційних систем, які будуть автоматично виявляти ситуації, що потребують звільнення сторінок, причому робити це набагато ефективніше.

Щодо розміру сторінок, то для правильного підбору необхідно враховувати багато факторів, як то:

- чим менший розмір сторінки, тим більша кількість сторінок та сторінкових кадрів буде в системі і тим більші будуть розміри таблиці сторінок. Якщо таблиці сторінок займають первинну пам’ять, то, з огляду на це, треба збільшувати розміри сторінок;

- при великих розмірах сторінок до первинної пам’яті потрапляє велика кількість інформації, яка не завжди може знадобитися. З цих міркувань треба зменшити розміри сторінок;
- оскільки обмін даними з дисковою пам’яттю займає відносно багато часу, і ми хочемо звести до мінімуму число обмінів, то слід збільшувати розміри сторінок.

Можна навести ще декілька таких доводів. Універсальних розмірів сторінок немає. Експериментально встановлено, що більш необхідні сторінки малих розмірів. Про це говорять дані, приведені в таблиці.

Фірма	Модель	Розмір сторінки	Одиниця виміру
IBM	360/67	1024	32-бітове слово
IBM	370/168	1024 або 512	32-бітове слово
DEC	PDP-10, PDP-20	512	36-бітове слово
DEC	YAX 11/780	128	32-бітове слово
Intel	80386,80486	4096	8 - бітове слово

Контрольні питання

1. Охарактеризуйте концепцію віртуальної пам’яті.
2. Які ви знаєте види організації віртуальної пам’яті?
3. Чим відрізняються сторінки і сегменти?
4. В чому різниця між сторінкою і сторінковим кадром?
5. Охарактеризуйте механізм динамічного перетворення віртуальних адрес в реальні.
6. Які переваги і недоліки асоціативного перетворення віртуальних адрес в реальні?
7. Які таблиці веде система при сторінково-сегментній організації віртуальної пам’яті?
8. Охарактеризуйте стратегії керування віртуальною пам’яттю.
9. Наведіть основні положення керування віртуальною пам’яттю на основі робочих множин.
10. Дайте порівняльний аналіз стратегій підкачки сторінок за запитом і з упередженням.
11. Якого розміру повинні бути сторінки?

5 Керування процесорами

Процеси мають можливість виконувати певну роботу, коли в їх розпорядженні є фізичні процесори. В даній главі розглядаються питання, коли, в яких випадках і яким процесам необхідно виділяти процесор, іншими словами, *питання планування завантаження процесора.*

5.1 Рівні планування в ОС

Процеси можуть виконувати конкретну роботу, якщо отримують в користування фізичні процесори. Розрізняють три основні рівні планування в ОС:

- *планування верхнього рівня*, яке деколи називають плануванням завдань або плануванням допуску. Засоби цього рівня визначають яким завданням буде дозволено конкурувати за захват ресурсів системи. Завдання, що входять у систему, стають процесами або групами процесів (рис.5.1);

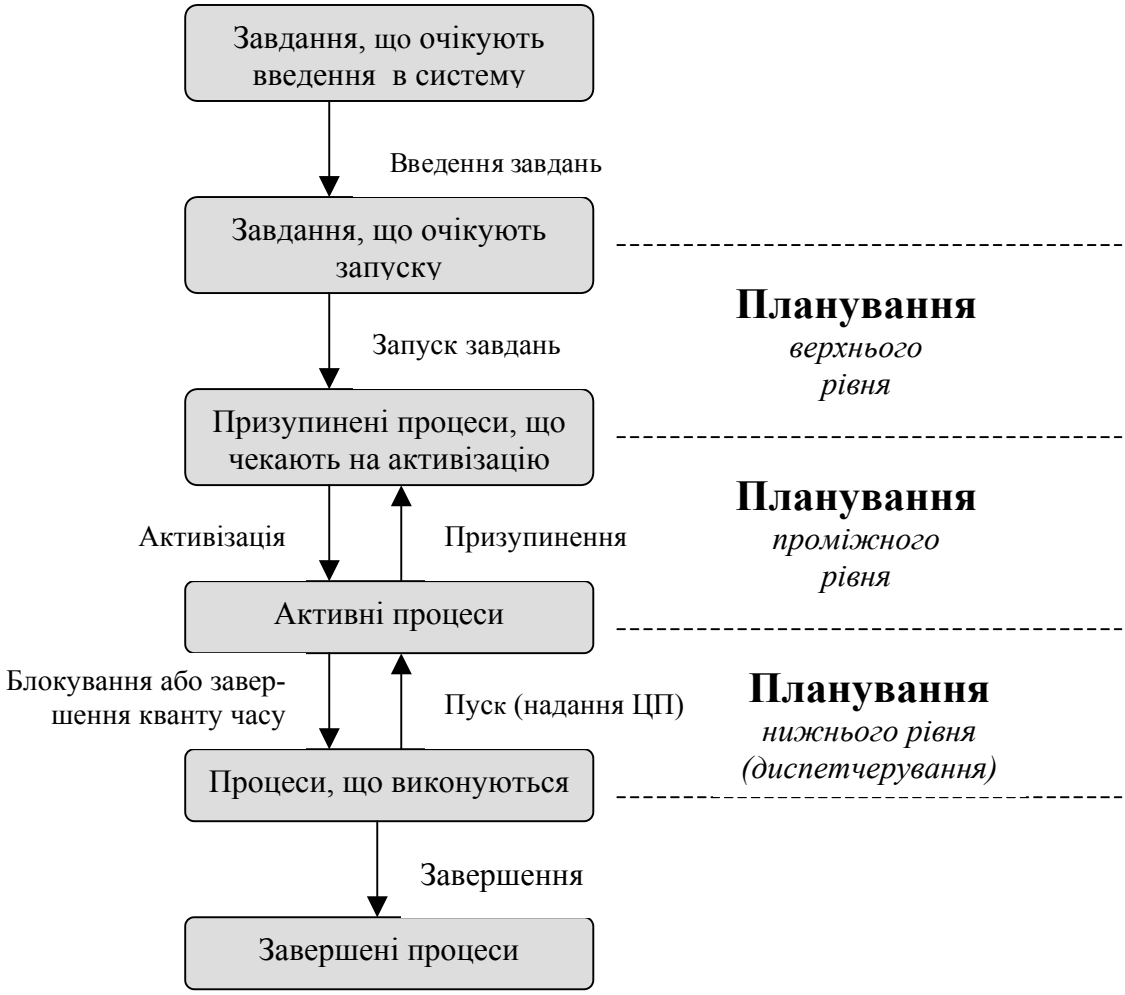


Рисунок 5.1 - Рівні планування

- *планування проміжного рівня*. Засоби цього рівня визначають, яким процесам буде дозволено змагатися за захват центрального процесора (ЦП). Планувальник проміжного рівня оперативно реагує на коливання системного навантаження, короткочасно призупиняє і знову активізує процеси, що забезпечує рівномірну роботу системи і допомагає досягненню глобальних цільових швидкісних характеристик;
- *планування нижнього рівня.* Засоби цього рівня визначають, якому із готових до виконання процесів буде виділений вільний ЦП і фактично виділяють ЦП даному процесу (тобто, виконують диспетчерські функції). Планувальник нижнього рівня називають *диспетчером*. Він працює з великою частотою, багато раз у секунду, і тому завжди повинен бути в основній пам'яті [4-5,7].

5.2 Планування з переключенням і без переключення

Якщо після передачі ЦП деякому процесу, відібрати ЦП не можна до завершення процесу, то говорять про дисципліну *планування без переключення*. Якщо ЦП відібрати можна, то це дисципліна *планування з переключенням*. Планування з переключенням потрібно в системах реального часу або інтерактивних системах, де необхідно гарантувати прийнятний час відповіді.

Обидва підходи мають свої недоліки:

- в схемі з переключенням в основній пам'яті треба розміщувати декілька процесів, щоб швидко перемикались з одного на інший, перемикання збільшує час виконання процесів;
- в схемах без перемикання довго очікують короткі завдання.

Перемикання здійснюється за рахунок інтервального таймера і переривань. За перериванням від таймера керування передається ОС, яка вирішує, якому процесу віддати ЦП.

Часові переривання дозволяють гарантувати прийнятний час відповіді в діалоговому режимі, не дозволяють зависати системі через зациклення програм користувача, дозволяють процесам реагувати на події, що залежать від часу. Якщо процес отримує ЦП, то він зберігає керування над ним, поки добровільно сам не звільнить ЦП, або поки не виникне часове переривання. Розмір кванту часу, на який процесу надається процесор, змінюється від системи до системи. Якщо всі процеси в системі лімітуються центральним процесором (тобто відсутні операції введення-виведення і відповідно інтерактивні користувачі), то перемикання взагалі непотрібні.

5.3 Пріоритети процесів

Пріоритети процесів можуть назначатися ззовні або присвоюватись автоматично, можуть бути зробленими або купленими, статичними або динамічними. Статичні пріоритети не змінюються під час виконання

процесу, механізми статичної пріоритетності легко реалізуються. Динамічні пріоритети змінюються в залежності від змін зовнішньої для даного процесу ситуації, а тому схеми динамічної пріоритетності набагато складніші в реалізації. Куплені пріоритети купуються за гроші користувачами.

5.4 Дисципліни планування

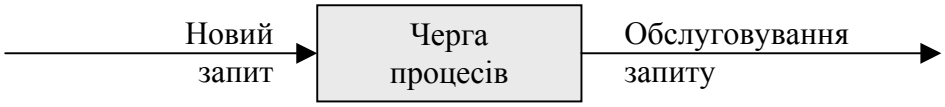
Ідеї мультипрограмування пов'язані з наявністю черг процесів. Черги мають місце при звертанні до каналів, модулів ОС, наборів даних, ЦП. Використання багатьма процесами того чи іншого ресурсу, який в кожен момент часу може обслуговувати тільки один процес, виконується за допомогою дисциплін розподілу ресурсів, основою яких є: дисципліни формування черг на ресурси та дисципліни обслуговування черг.

Дисципліни обслуговування черг - це сукупність правил, які визначають порядок вилучення процесу з черги з подальшим наданням ресурсу цьому процесу для використання.

Дисципліни формування черг - це сукупність правил, які визначають порядок розміщення процесів в черзі.

Для формування і обслуговування черг використовуються одні й ті ж або близькі дисципліни планування.

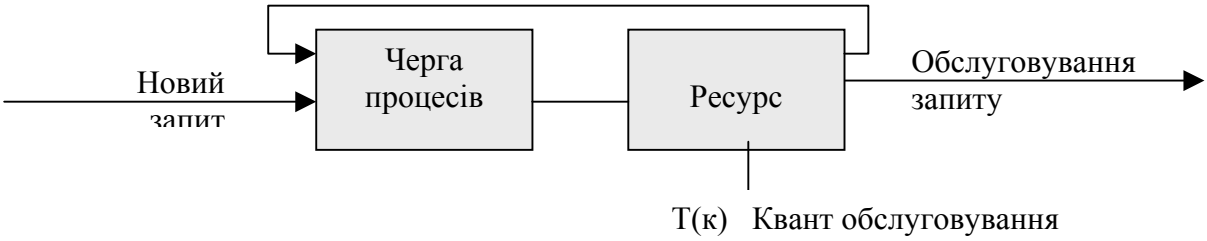
1. ***Планування за терміном завершення.*** Виконання певних завдань повинно закінчуватись раніше певного терміну, наприклад, за додаткову оплату.
2. ***Планування за принципом FIFO (first-in-first-out).*** Найбільш простою дисципліною планування є принцип FIFO: першим прийшов - першим обслуговується.



FIFO - це дисципліна планування без перемикання.
Недоліки: довгі завдання змушують очікувати короткі, неважливі - важливі, не гарантується задовільний час відповіді.

3. ***Планування за принципом LIFO (last-in-first-out).*** Останній прийшов обслуговується першим. Це дисципліна планування без переключення. Проста в реалізації. Недоліки – ті ж, що і в FIFO.
4. ***Циклічне планування (RR – round robin).*** При цьому плануванні диспетчирування виконується за принципом FIFO, однак кожного разу процесу виділяється обмежена кількість часу ЦП, яка називається часовим квантом. Якщо процес не закінчується за виділений квант

часу, то ЦП у нього відбирається і віддається наступному процесу. Процес, у якого перехопили ЦП, переходить у кінець списку готових до виконання процесів.



Ця дисципліна ефективна для роботи з розподілом часу, коли система повинна гарантувати прийнятний час відповіді для всіх інтерактивних користувачів.

5. ***Планування за принципом SJF (shortest-job-first).*** При плануванні SJF найбільш коротке завдання виконується першим. Це дисципліна планування без переключення. Основна проблема, яка виникає при реалізації цього принципу, полягає в необхідності визначення часу, протягом якого буде виконуватись завдання. Цей принцип забезпечує зменшення середнього часу очікування в порівнянні з FIFO, але час очікування коливається в широких межах. Користувачі можуть обманювати систему, задаючи мінімальний оціночний час виконання.
6. ***Планування за принципом SRT (shortest-remaining-time)*** – планування за найменшим часом, що залишився до завершення завдання. Це аналог SJF з переключенням. Короткі завдання виконуються майже відразу. Може використовуватись в системах з розподілом часу.
   Недоліки: велика дисперсія часу очікування.
7. ***Планування за принципом HRN (highest-response-next)*** - планування за найбільшим відносним часом реакції. Це дисципліна планування без переключення, відповідно до якої пріоритет кожного завдання є не тільки функцією часу обслуговування цього завдання, але також і часу, витраченого завданням на очікування обслуговування. Після того, як завдання отримує ЦП, воно виконується до завершення. Динамічно пріоритети обчислюються за формулою:

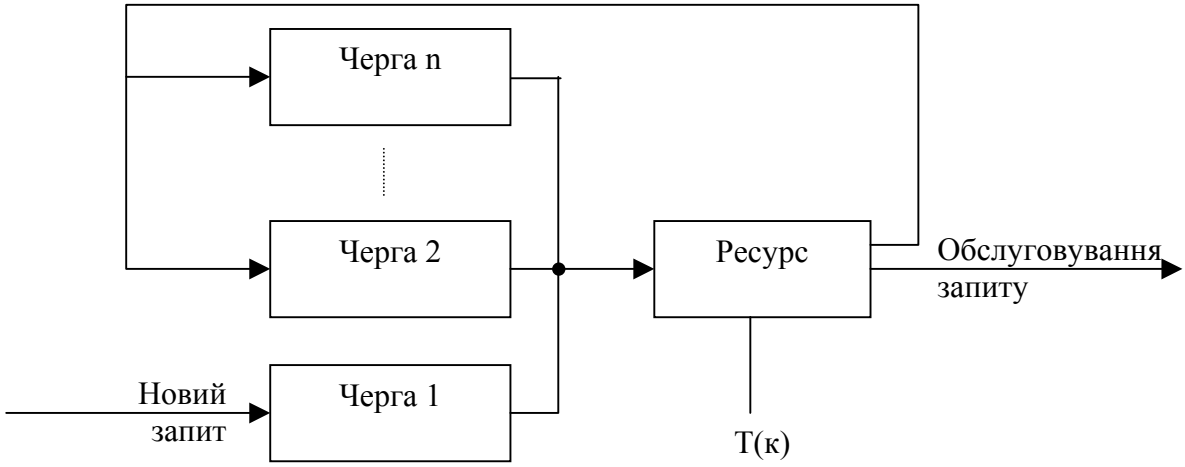
Пріоритет = (Час очікування + Час обслуговування) / Час обслуговування

8. ***Багаторівневі черги.*** Механізм планування повинен:

   - віддавати перевагу коротким завданням;
   - віддавати перевагу завданням, які лімітуються введенням-виведенням;
   - якомога швидше визначати характер завдання і відповідним чином планувати його виконання.

Багаторівневі черги зі зворотними зв’язками значною мірою забезпечують досягнення цих цілей.

Схема багаторівневої черги така:



5.5 Мультипроцесорні системи

З появою дешевих мікропроцесорів та їх комплектів з'явилась можливість створювати недорогі мультипроцесорні системи (МПС). Перевагами МПС є їх надійність і висока продуктивність. Головна мета застосування мультипроцесорних систем збільшення продуктивності за рахунок використання паралелізму під час виконання програм. Однак є багато складностей:

- люди думають послідовними категоріями;
- мови паралельного програмування не отримали достатнього поширення;
- налагоджувати паралельні програми набагато складніше, ніж послідовні.

Реальний прогрес можливий лише тоді, коли ОС і компілятори зможуть автоматично виявляти і виконувати розпаралелення.

5.6 Автоматичне розпаралелювання

Паралелізм в програмах може бути *явним або неявним*. *Явний паралелізм* програміст сам вказує у своїй програмі. Однак, це пов'язано з рядом негативних моментів. Так, програми з явним вказуванням паралелізму стають більш складними для їх модифікації. З другого боку, спроможність явного вказування паралелізму вимагає від програміста високої кваліфікації. Малоймовірно, що ним будуть указані всі можливі ситуації.

Більш реальним є виявлення *неявного паралелізму* за допомогою спеціальних механізмів розпаралелювання, які включаються в

операційні системи, компілятори та апаратні засоби комп'ютерів. Найбільш поширеними способами пошуку неявного паралелізму:

- розщеплення циклу;
- редукція висоти дерева;
- правило "ніколи не чекати".

Розщеплення циклу

У програмному циклі передбачається багатократне виконання деякої послідовності операторів, які називаються тілом циклу, до тих пір, поки не стане істинною відповідна умова завершення. В деяких випадках у тіло циклу входять оператори, які можуть виконуватись паралельно. Розглянемо такий цикл.

```
FOR i=1 TO 4 DO
A(i) = B(i) + C(i);
```

У даному програмному циклі послідовний процесор виконує по черзі такі операції:

```
A(1)=B(1)+C(1);
A(2)=B(2)+C(2);
A(3)=B(3)+C(3);
A(4)=B(4)+C(4).
```

У мультипроцесорній системі ці оператори можна виконувати паралельно за допомогою 4-х процесорів, завдяки чому значно зменшується час виконання всього циклу.

Компілятор, який виконує автоматичне розпаралелення, може перетворити приведений вище цикл у таку форму:

```
COBEGIN;
    A(1)=B(1)+C(1);
    A(2)=B(2)+C(2);
    A(3)=B(3)+C(3);
    A(4)=B(4)+C(4);
COEND;
```

де конструкція COBEGIN/COEND вказує обчислювальній системі на можливість паралельного виконання операторів.

Редукція висоти дерева

Застосовуючи асоціативні, комутативні та дистрибутивні властивості арифметики, компілятори можуть виявляти неявний паралелізм в алгебраїчних виразах і генерувати об'єктний код для мультипроцесорних систем з указанням операцій, які можуть виконуватись одночасно.

Звичайний компілятор діє за такою схемою:

1. Виконуються операції у вкладених дужках, тобто ті операції, які мають більш глибокий рівень вкладеності.
2. Потім виконуються операції в дужках.
3. Виконуються операції піднесення до степіні.

4. Потім виконуються операції множення і ділення.
5. Дальше - операції додавання і віднімання.
6. Якщо операції є рівноцінними, то вони виконуються зліва направо.

Застосування таких правил дає можливість компілятору встановлювати однозначний послідовний порядок дій для обчислення алгебраїчного виразу, а потім переходити до генерації машинного коду, що реалізує його обчислення.

Часто однозначність і послідовність порядку дій під час обчислень не обов'язкова. Наприклад, оскільки операції множення і додавання є комутативними, то компілятор, який генерує код для перемноження p і q, може видати p*q або q*p; результати обчислень будуть однаковими. Аналогічно під час додавання p і q компілятор спрацює однаково: p+q=q+p.

Розглянемо декілька прикладів зменшення висоти дерева.

Зменшення висоти дерева за рахунок асоціативності

Використовуючи асоціативність операції додавання, вираз

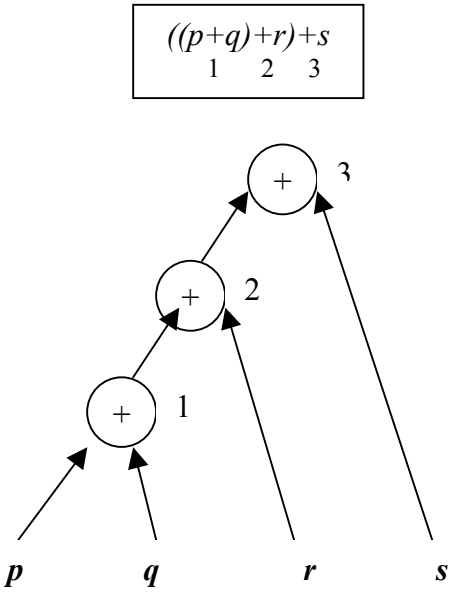
((p+q)+r)+s

можна перетворити у вираз

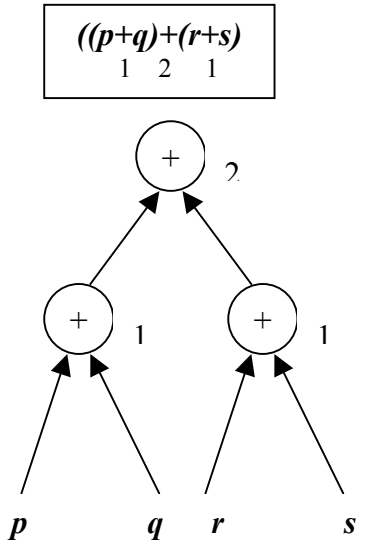
(p+q)+(r+s),

який більш пристосований для паралельних обчислень.

Звичайний компілятор

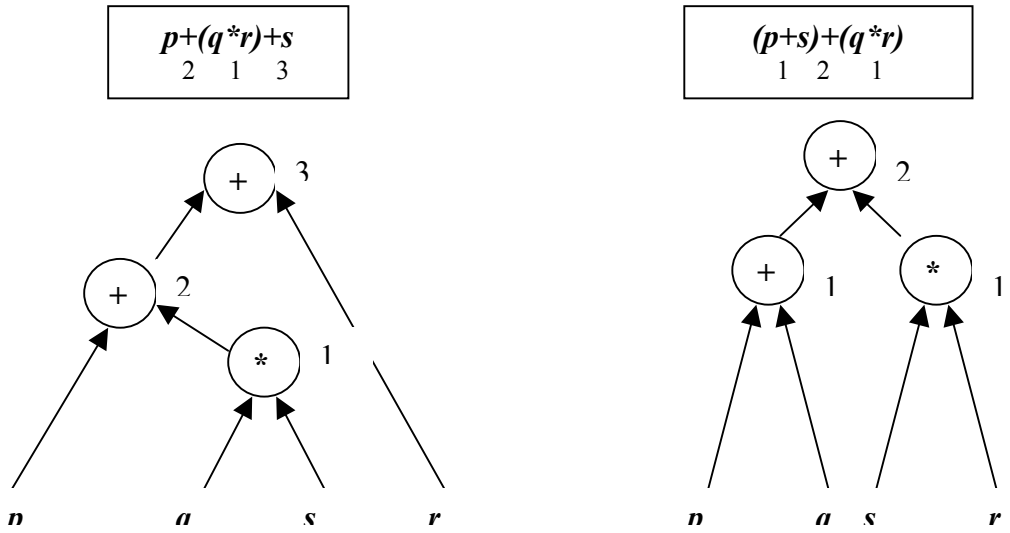


Паралельний компілятор



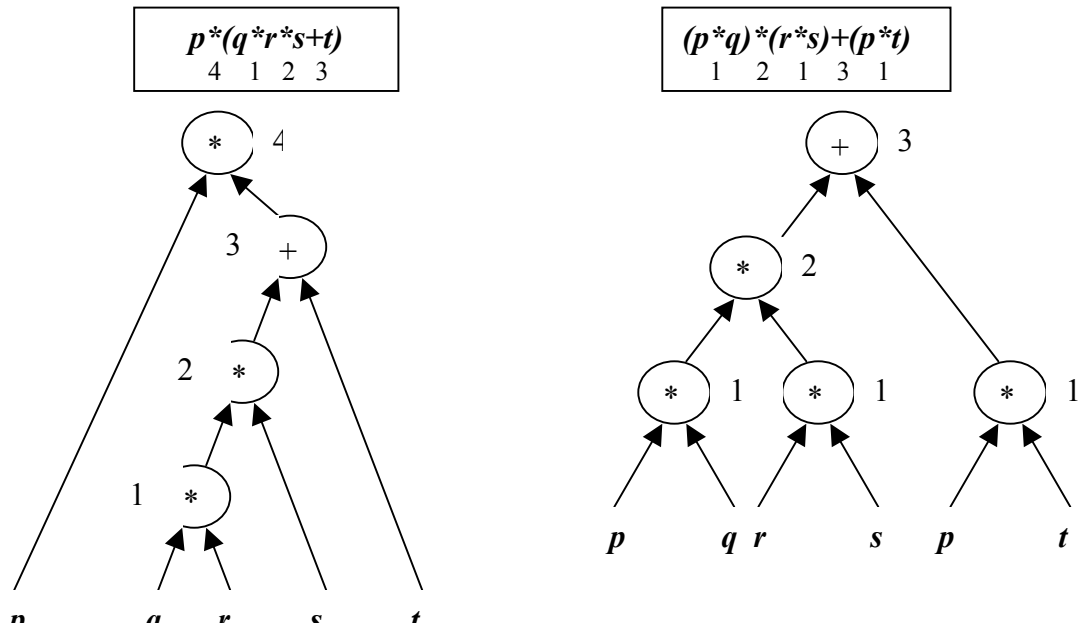
Зменшення висоти дерева за рахунок комутативності

Звичайний компілятор *Паралельний компілятор*



Зменшення висоти дерева за рахунок дистрибутивності

Звичайний компілятор *Паралельний компілятор*

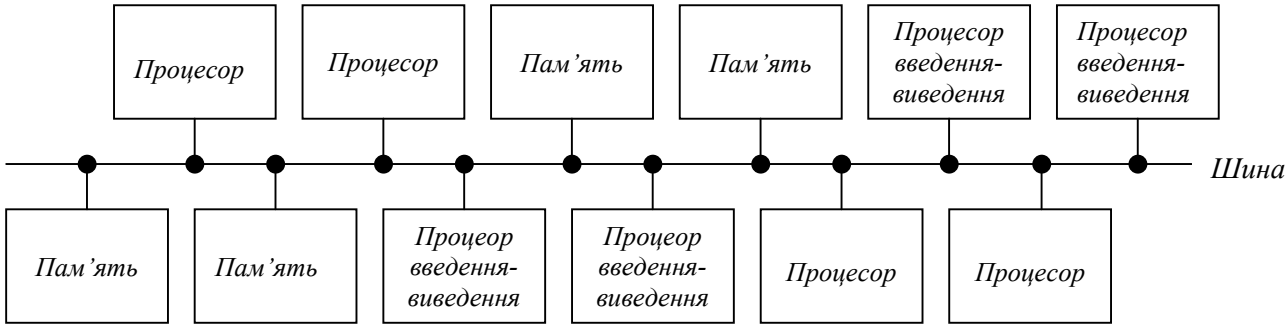


Правило "ніколи не чекати"

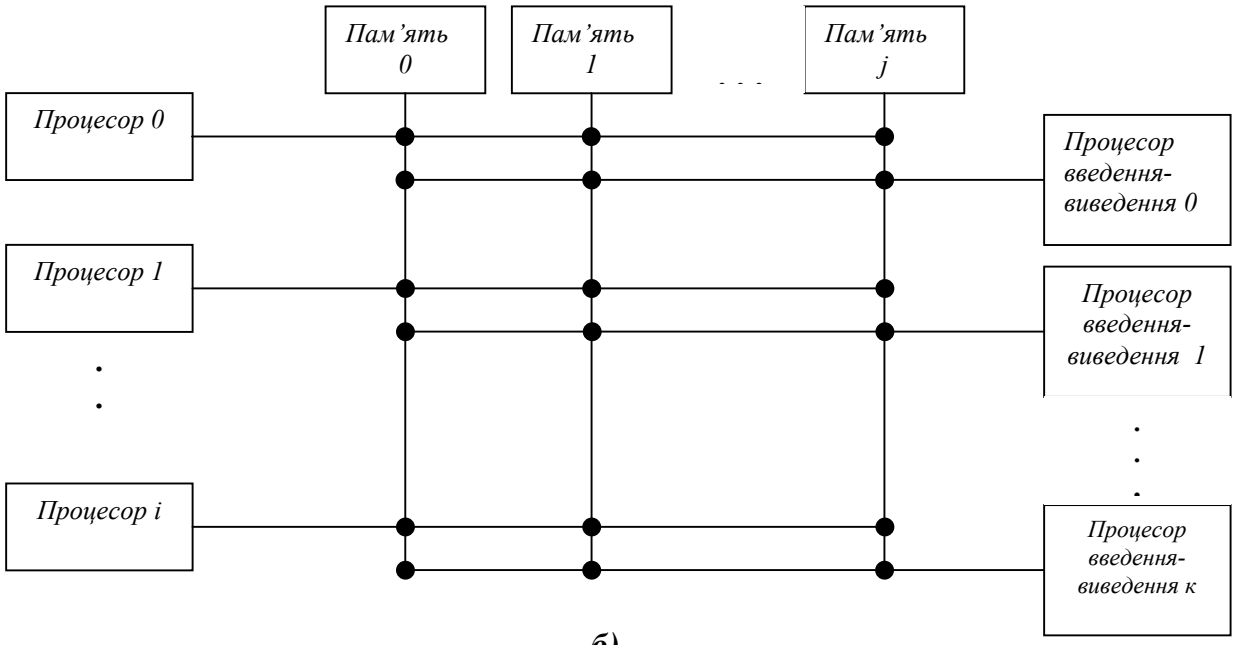
Правило "ніколи не чекати" говорить по те, що краще доручити деякому процесору роботу, яка або буде, або не буде використана в подальшому, ніж залишити цей процесор бездіяльним.

Розглянемо такий приклад:

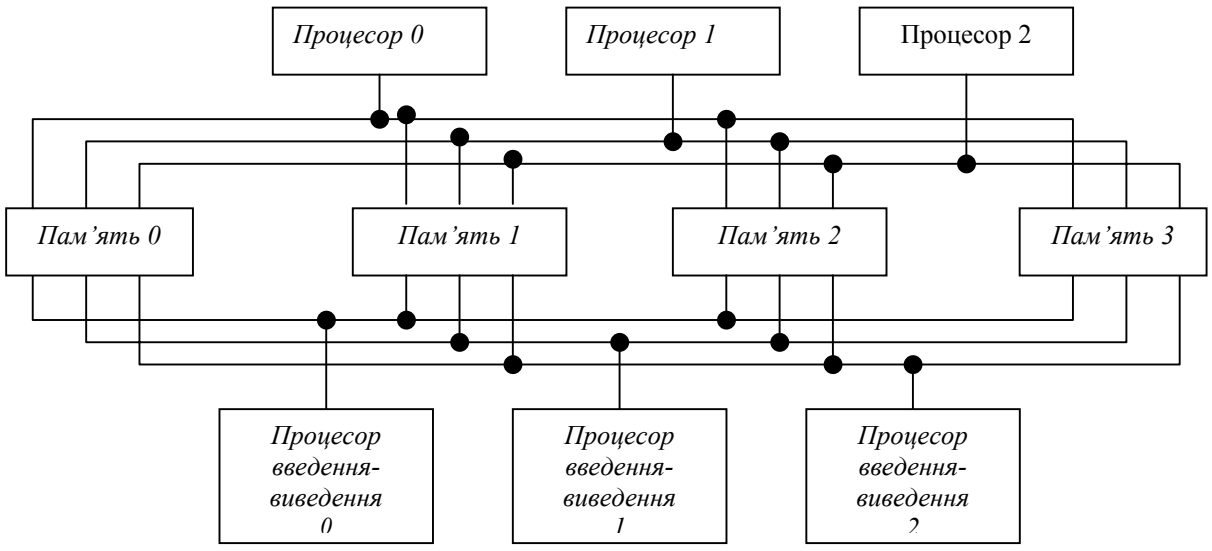
```
A=B*C;
IF A=9 THEN D=10;
E=D*F;
```



а)



б)



в)

Рисунок 5.2 - Організація мультипроцесорної апаратури (а – спільна шина, б – матриця координатної комутації, в – багатопортова пам'ять

У залежності від результату, отриманого під час виконання першого оператора, другий оператор може змінити значення D або залишити його попереднім. Якщо значення D не змінюється, то третій оператор можна було б виконувати паралельно з першим. Якщо D змінюється, то під час виконання третього оператора необхідно використовувати нове значення D.

Стратегія "ніколи не чекати" полягає в тому, щоб завжди виконувався третій оператор паралельно з першим. Якщо значення D не зміниться, то результат виконання третього оператора буде вже готовий і все обчислення завершиться швидше.

5.7 Організація мультипроцесорної апаратури

Дійсно мультипроцесорні комплекси можна охарактеризувати таким чином:

- Мультипроцесорний комплекс містить два і більше процесорів з приблизно однаковою продуктивністю.
- Процесори мають доступ до спільної пам’яті.
- Всі процесори мають колективний доступ до каналів, контролерів і пристроїв введення-виведення.
- Весь комплекс працює під керуванням однієї ОС.

Розрізняють три найбільш розповсюджені види організації мультипроцесорних комплексів:

- з спільною шиною;
- з матрицею координатної комутації;
- з багатопортовою пам'яттю.

Організація з спільною шиною використовує один канал зв’язку між всіма функціональними пристроями (рис. 5.2 а). Архітектура з спільною шиною дозволяє легко вводити нові пристрої, підключаючи їх безпосередньо до шини. Недоліки спільної шини такі:

- несправності в шині виводять з ладу весь комплекс;
- швидкість передачі даних в системі обмежена пропускною здатністю шини.

Незважаючи на економічність і простоту через указані недоліки спільна шина застосовується лише в побудові невеликих мультипроцесорних комплексів.

Матриця координатної комутації.

В цій схемі звертання до різних пристроїв пам’яті можуть виконуватись одночасно (рис.5.2.б), що збільшує швидкодію всієї обчислювальної системи. Однак, апаратні засоби необхідні для побудови координатного комутатора можуть бути доволі складні, оскільки кожний вузол

комутатора містить апаратуру керування, комутації і пріоритетного арбітражу.

Багатопортова пам'ять характеризується тим, що апаратура керування, комутації і пріоритетного арбітражу розміщується в інтерфейсі кожного пристрою пам'яті (рис.5.2.в). За такої організації кожний функціональний пристрій може отримати доступ до кожного пристрою пам'яті, але тільки по конкретному порту, або каналу зв'язку. Портам пам'яті присвоюються постійні пріоритети для розв'язування конфліктних ситуацій між пристроями, що одночасно звертаються до однієї і тієї ж пам'яті.

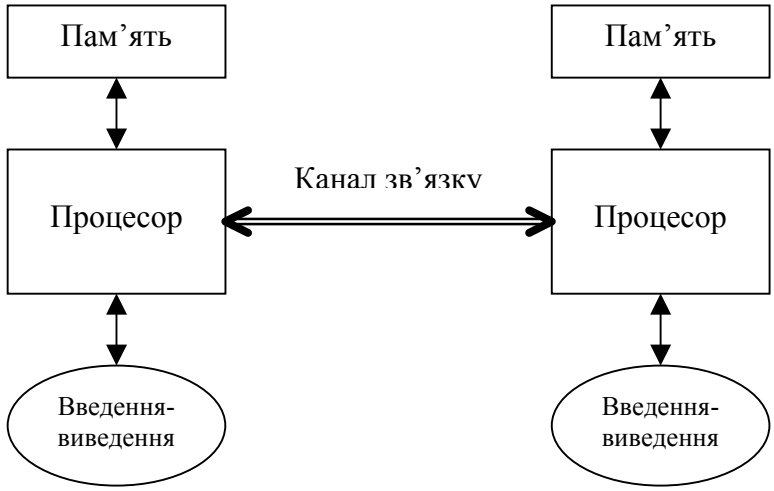
Крім того, багатопортова пам'ять може дозволяти доступ до різних пристроїв пам'яті різним підгрупам центральних процесорів або процесорів введення-виведення, що дуже важливо для систем високої секретності.

Недоліком систем з багатопортовою пам'яттю є великі витрати кабелю на виконання необхідних з'єднань.

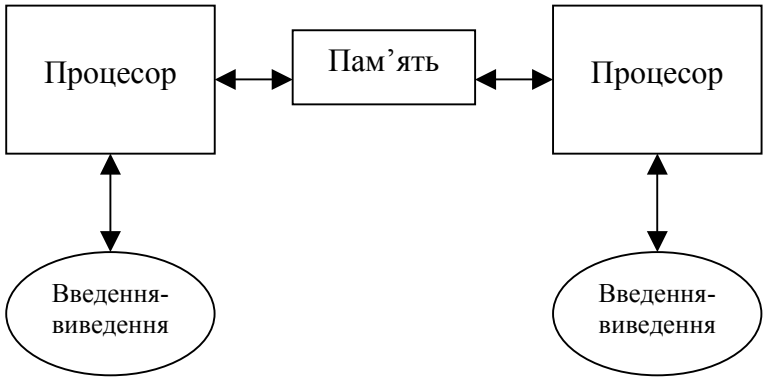
Системи зі слабо та сильно зв'язаними процесорами

Мультипроцесорні системи з слабо зв'язаними процесорами (багатомашинні комплекси) передбачають з'єднання двох або більше обчислювальних машин за допомогою каналу зв'язку. Кожна машина має свою власну операційну систему і пам'ять.

Мультипроцесорна система зі слабо зв'язаними процесорами



Мультипроцесорна система з сильно зв'язаними процесорами



Мультипроцесорні системи з сильно зв'язаними процесорами (багатопроцесорні обчислювальні комплекси) мають єдину пам'ять, яка колективно використовується різними процесорами, і єдину операційну систему, яка керує роботою всіх процесорів та інших апаратних засобів системи.

Організація головний-підлеглий

При такій організації один з процесорів виконує функції головного, а інші є підлеглими. В якості головного використовується процесор загального призначення, який може виконувати як обчислення, так і операції введення-виведення. Підлеглі процесори виконують тільки обчислення.

Основний недолік організації "головний-підлеглий" полягає в асиметричності апаратних засобів, оскільки операції введення-виведення може виконувати тільки головний процесор. Тому більш перспективною вважається симетрична організація.

Симетрична мультипроцесорна система

В такій системі всі процесори функціонально еквівалентні і можуть виконувати як обчислення так і операції введення-виведення.

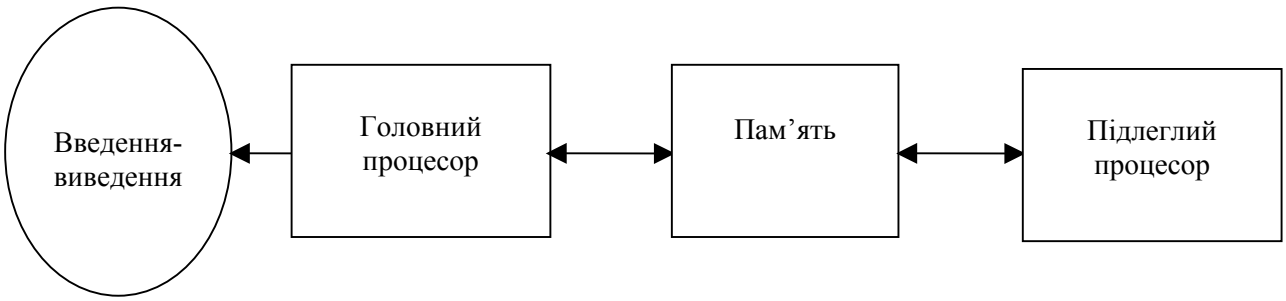
5.8 Мультипроцесорні операційні системи

Існують три варіанти організації операційних систем для мульти-процесорних комплексів:

- Головний - підлеглий.
- Окремий монітор на кожний процесор.
- Симетрична організація (процесори ідентичні).

Організація "головний-підлеглий"

При організації операційної системи "головний-підлеглий" операційна система виконується тільки на головному процесорі. На підлеглих процесорах можуть виконуватись тільки програми користувачів.



Коли процесу, який виконується на підлеглому процесорі потрібна послуга ОС, він генерує сигнал переривання. Якщо підлеглих процесорів багато, то можуть створюватись черги до головного процесора і таким

чином обчислювальна потужність підлеглих процесорів використовується не в повній мірі.

Ще одним недоліком цього типу організації є недостатня надійність, оскільки вихід головного процесора з ладу може призвести до виходу з ладу всієї системи(в деяких системах підлеглий процесор може взяти на себе функції головного).

Організація з роздільними моніторами

При організації з роздільними моніторами кожний процесор має власну операційну систему, яка певним чином реагує на переривання від програм користувачів, що працюють на цьому процесорі. Кожний процесор керує своїми ресурсами, незалежно від інших процесорів, наприклад, файлами або пристроями вводу-виводу. Процес, запланований для виконання на якомусь певному процесорі, виконується на ньому до завершення. Доступ до спільних таблиць виконується з використанням механізму взаємо виключення.

Не передбачається ніякої взаємодії процесорів при виконанні індивідуального процесу. Деякі процесори можуть лишатися вільними, в той самий час як один процесор виконує довготривалий процес.

Симетрична організація

Вдалішою є система з симетричною організацією. Характеристики цієї організації такі:

1. Найбільш складна для реалізації. Всі процесори ідентичні і кожен з них може керувати будь-яким пристроєм введення-виведення або звертатися до пам'яті.
2. ОС переміщується від одного процесора комплексу до іншого. Процесор, який в даний момент відповідає за ведення системних таблиць і виконання системних функцій, називається моніторним процесором. В кожен конкретний момент часу моніторним може бути лише один процесор, що усуває конфлікти під час роботи з системною інформацією.
3. Процес може виконуватись на будь-якому процесорі в різні періоди часу. Процесори можуть кооперуватися для виконання одного завдання.
4. Оскільки програми ОС виконуються на багатьох процесорах, реєнтерабельний код і взаємовиключення обов'язкові для ОС.
5. Необхідні програмні і апаратні засоби для розв'язання конфліктних ситуацій.
6. Навантаження є точніше збалансованим і організація в цілому є найбільш надійною з розглянутих.

5.9 Продуктивність мультипроцесорних систем

Збільшення кількості процесорів в n разів не означає, що продуктивність збільшиться в n разів через:

- додаткові витрати на роботу ОС;
- збільшення конкуренції за системні ресурси;
- затримки в апаратурі комунікації і маршрутизації.

Відомі мультипроцесорні системи, в яких при двох процесорах продуктивність зростає в 1.8 рази, при 3-х - в 2,1 рази, при 4-х - в 3,6 рази.

5.10 Відновлення після помилок

Для відновлення працездатності МПС застосовуються різноманітні прийоми, а саме:

1. Критичні дані для системи і процесів повинні мати декілька копій в різних модулях пам'яті.
2. Система повинна мати засоби знаходження і виправлення апаратних помилок.
3. ОС повинна вміти керувати як мінімальною, так і максимальною конфігурацією апаратних засобів.
4. Не завантажені ресурси процесорів необхідно використовувати для знаходження потенційних помилок.
5. ОС повинна вміти передавати процеси від процесора, що вийшов з ладу, процесорові, що працює.

5.11 Мережні операційні системи і розподілені операційні системи

Ідеї закладені в мультипроцесорних ОС отримали подальший розвиток в операційних системах для мереж. Загальноприйнятим є такий спосіб розділення операційних систем для мереж:

- *мережні операційні системи;*
- *розподілені операційні системи.*

У випадку мережних ОС на кожному комп'ютері діє своя немережна ОС. Функціями мереж керують прикладні програми. Такий підхід дозволяє використовувати в мережах уже існуюче програмне забезпечення. Недоліком є відсутність однорідності мережі.

У випадку розподіленої ОС від індивідуальних операційних систем відмовляються і створюється одна ОС для всієї мережі. Цей підхід вимагає значних зусиль для створення операційної системи, але дозволяє отримати концептуальну єдність, що спрощує розуміння мережі, її обслуговування та модифікацію.

Для мереж, які відрізняються архітектурно і розкидані географічно, звичайно використовують перший підхід. В локальних мережах, як правило, застосовуються розподілені операційні системи [8].

Контрольні питання

1. Рівні планування і різниця між ними.
2. Планування з переключенням і без переключення.
3. Механізми зміни пріоритетів.
4. Статичні пріоритети.
5. Динамічні пріоритети.
6. Куплені пріоритети.
7. Які дисципліни планування, крім наведених, вам відомі?
8. Позитивні і негативні якості:
    - принципу планування FIFO;
    - циклічного планування;
    - планування SJF;
    - планування SRT;
    - планування HRN.
9. Види організації мультипроцесорних комплексів.
10. Системи з слабо і сильно зв'язаними процесорами.
11. Типи операційних систем мультипроцесорних комплексів.
12. Продуктивність мультипроцесорних систем у залежності від кількості процесорів.
13. Цілі мультипроцесорних систем.
14. Види паралелізму в програмах.
15. Методи виявлення неявного паралелізму.

6 Керування пристроями і зовнішньою пам'яттю

Неефективність мультипрограмних обчислювальних систем часто обумовлена неправильним використанням пристроїв, особливо це стосується пристроїв зовнішньої пам'яті. Саме розгляду питань керування пристроями і зовнішньою пам'яттю присвячена дана глава.

6.1 Функції системи керування пристроями

Функції системи керування пристроями такі:

- слідкування за станом пристроїв, вся інформація про пристрій міститься у блоці керування пристроєм (UCB – unit control block);
- прийняття рішень, кому і на який термін виділяється пристрій;
- виділення і звільнення пристроїв.

Модуль, який відслідковує стан пристроїв, називається *регулювальником введення-виведення.*

Обробник введення-виведення відповідає за створення канальної програми для виконання операцій введення-виведення з конкретного пристрою, обробку пов'язаних з введення-виведенням переривань та оптимізацію функціонування пристрою.

Планувальник введення-виведення приймає рішення про те, коли процесор введення-виведення повинен бути приписаний запиту і визначає шлях до пристрою.

Для керування пристроями та їх розподілом використовуються три основних способи:

- спосіб монопольного використання пристроїв;
- спосіб сумісного використання пристроїв (комунальний);
- спосіб віртуального використання пристроїв.

6.2 Блоки керування пристроями, каналами і контролерами

Процесор може звертатись до пристроїв через відповідні канали і контролери (рис. 6.1).

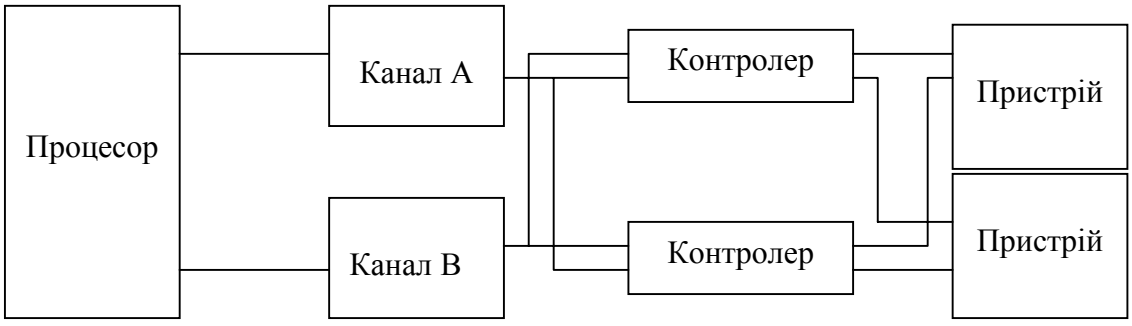


Рисунок 6.1 - Підключення пристроїв до процесора.

Всю інформацію про пристрої, контролери і канали ОС (регулювальник введення-виведення) розміщує в спеціальні структури даних:

1. Блок керування пристроями (UCB).
2. Блок керування контролерами (CLUCB).
3. Блок керування каналами(CCB).

Блок керування пристроями містить таку інформацію:

- ідентифікатор пристрою;
- стан пристрою;
- список контролерів, підключених до даного пристрою;
- список процесорів, які очікують на даний пристрій.

Блок керування каналом містить таку інформацію:

- ідентифікатор каналу;
- стан каналу;
- список контролерів, підключених до даного каналу;
- список процесорів, які очікують на даний канал.

Блок керування контролером містить таку інформацію:

- ідентифікатор контролера;
- стан контролера;
- список каналів, підключених до контролерів;
- список процесорів, які очікують на даний контролер.

Особливо важливим є забезпечення оптимального використання пристроїв зовнішньої пам'яті, серед яких найбільш поширені накопичувачі на магнітних дисках [7].

6.3 Накопичувачі на магнітних дисках

6.3.1 Робота накопичувача на магнітних дисках

Схематично накопичувач на жорстких магнітних дисках зображений на рис. 6.2. Дані записуються на поверхні ряду магнітних дисків [8]. Ці диски жорстко з'єднані з загальним шпинделем, який обертається з дуже високою швидкістю (до 7200 обертів за хвилину). Доступ до даних виконується за допомогою ряду магнітних головок читання-запису, по одній головці на дискову поверхню. Головці доступні тільки ті дані, які знаходяться на ділянці дискової поверхні безпосередньо під (над) нею. Всі головки закріплені на одній каретці або блоці позиціонера. Каретка з головками може переміщуватись по радіусу диска в тому чи іншому напрямку. Якщо каретка не переміщується в даний момент, то кожна магнітна головка описує на дисковій поверхні кругову доріжку. Положення доріжок визначається кроком двигуна, який переміщує каретку і не визначається спеціальним чином. Кожна доріжка може поділятись на сектори. Положення сектора указується при форматуванні диска

спеціальними записами. Група доріжок, які знаходяться під всіма магнітними головками в якомусь конкретному положенні каретки, утворюють вертикальний циліндр.

Процес переміщення каретки з головками на новий циліндр називається *операцією пошуку циліндра* (рис.6.2,б) або підведення. Час, який витрачається на переміщення ділянки поверхні з поточного положення в положення під головкою читання-запису, називається *часом пошуку запису* (час пошуку даних на доріжці), причому

$$t_{пошуку\ циліндра} >> t_{пошуку\ запису.}$$

6.3.2 Основні стратегії оптимізації швидкісних дискової пам'яті

В мультипрограмних ОС одночасно виконується багато процесів, і всі вони можуть генерувати запити на доступ до дисків. Оскільки запити надходять частіше, ніж обслуговуються, то до кожного дискового пристрою формується черга запитів, яку треба планувати.

Відомі два найбільш поширених *види планування*:

- оптимізація за часом пошуку циліндра;
- оптимізація за часом пошуку запису.

Оскільки

$$t_{пошуку\ циліндра} >> t_{пошуку\ запису},$$

то більшість алгоритмів планування ставить за мету мінімізацію часу пошуку циліндруа Мінімізація часу пошуку запису незначно впливає на загальні швидкісні характеристики системи, якщо не враховувати режими дуже високих навантажень [6].

Цільові характеристики принципів планування

Для розподілу принципів планування за категоріями можуть бути використані такі критерії:

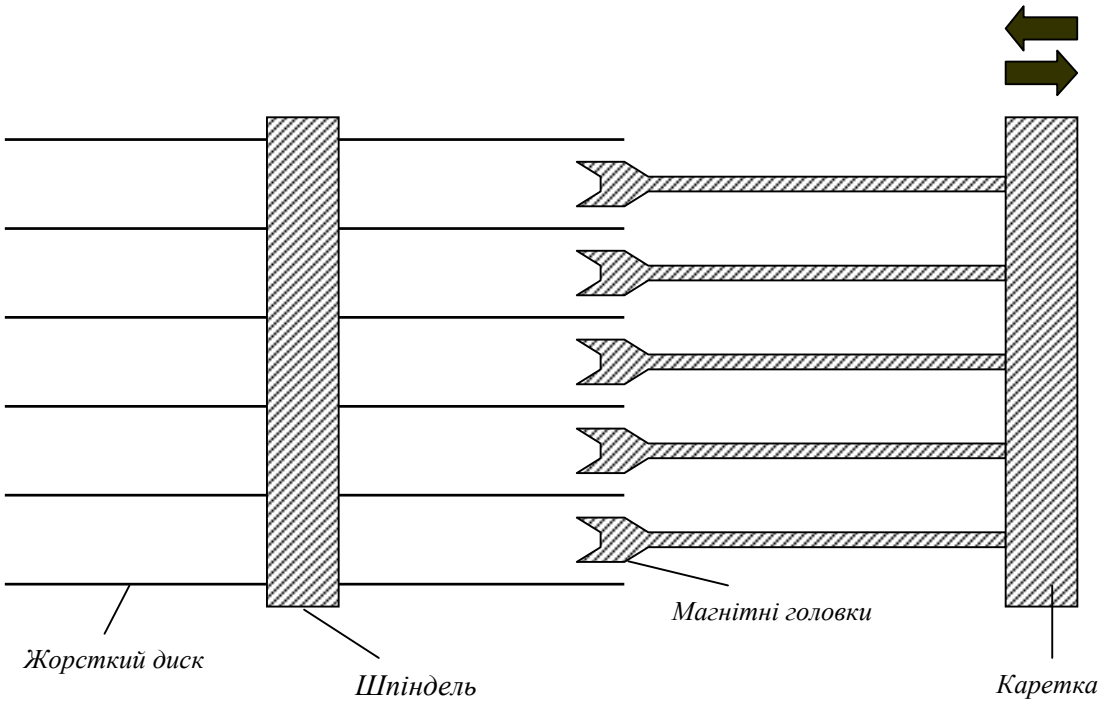
- пропускна здатність;
- середній час відповіді;
- розкид (дисперсія) часу відповіді (передбачуваність).

Очевидно, що кожна стратегія планування повинна бути направлена на збільшення пропускної здатності системи, мінімізацію середнього часу відповіді і дисперсії часу відповіді системи.

Стратегії оптимізації пошуку циліндра

1. Стратегія FCFS («перший прийшов - обслуговується першим»). До недоліків цієї стратегії відносять те, що вона вимагає багато часу на пошук циліндра, бо необхідні тривалі підведення від зовнішніх доріжок до внутрішніх і навпаки. З іншого боку ця стратегія забезпечує невелику дисперсію і є прийнятною, якщо дискова пам'ять працює з невеликим навантаженням.

а)



б)

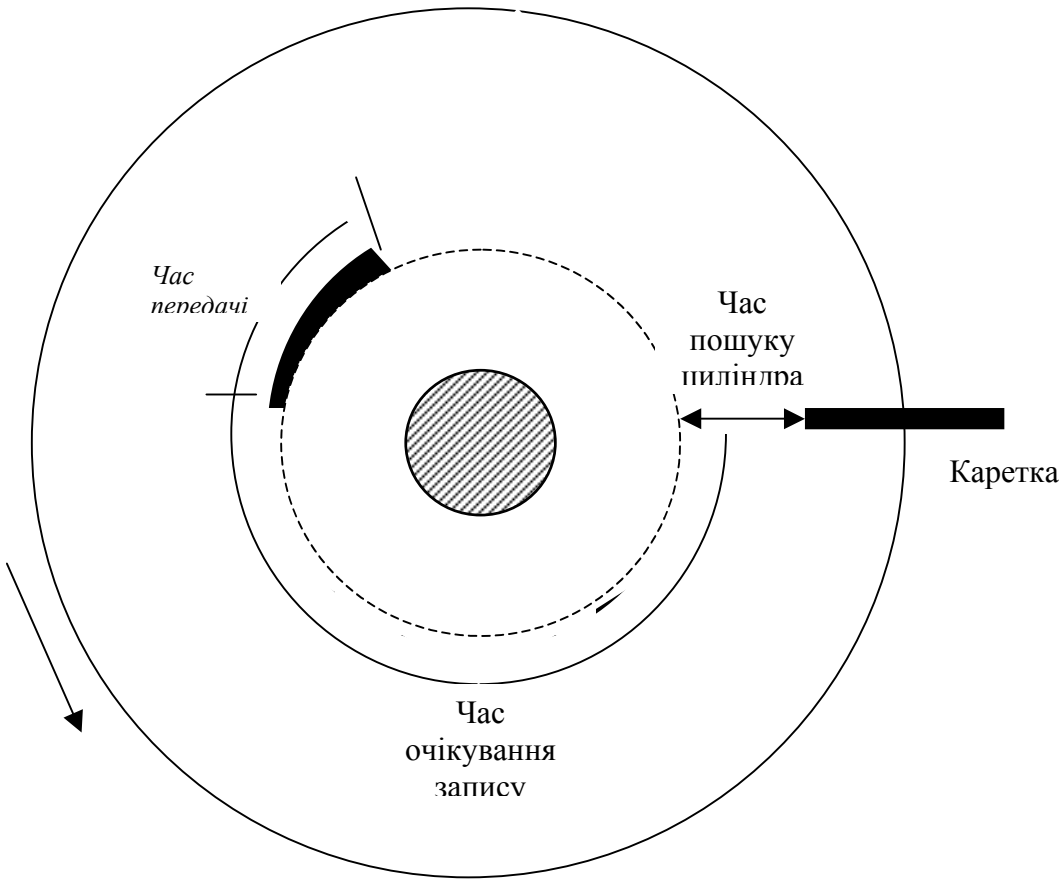


Рисунок 6.2 - Робота накопичувача на магнітних дисках (а – дисковий пристрій з головками, що переміщуються, б – часові характеристики при доступі до інформації на магнітних дисках*)*

2. Стратегія SSTF («з найменшим часом пошуку - першим»). Наступним вибирається запит, для якого необхідні мінімальні переміщення каретки. Недоліком є дискримінація запитів до внутрішніх і зовнішніх доріжок, збільшення часу відповіді і непридатність до інтерактивних дій через збільшення дисперсії. До переваг відносять кращу пропускну здатність, порівняно з FCFS, та кращий середній час відповіді для помірних навантажень.

3. Планування за принципом SCAN («сканування»). Каретка з головками рухається вперед-назад над поверхнею диску, обслуговуючи всі запити, які зустрічаються на шляху. Напрям руху змінюється, якщо в поточному напрямку немає запитів на обслуговування або по досягненні останньої доріжки. Стратегія SCAN дуже подібна на SSTF з точки зору підвищення пропускної здатності і зменшення середнього часу відповіді, однак вона значно зменшує дискримінацію крайніх доріжок, забезпечуючи тим самим значно меншу дисперсію часу відповіді. SCAN є основою для більшості практично реалізованих стратегій планування роботи з дисковою пам'яттю.

4. Планування N-step SCAN ("N-крокове сканування"). Каретка з головками рухається так само, як і в стратегії SCAN, однак всі запити, які надходять під час ходу в одному напрямку, групуються і упорядковуються таким чином, щоб їх можна було ефективно обслужити під час зворотного руху головки. Найважливішою її рисою є мала дисперсія часу відповіді в порівнянні з принципом SSTF або SCAN.

5. C-SCAN («циклічне сканування»). Обслуговуючи запити, каретка з головками рухається лише в напрямку від зовнішніх доріжок до внутрішніх. Якщо попереду більше немає запитів на обслуговування, то каретка стрибком повертається назад, обслуговуючи запит, найближчий до зовнішніх доріжок. Як правило, ця стратегія реалізуються так, що запити, які надходять під час поточного прямого ходу обслуговуються під час наступного прямого ходу. Характеризується циклічне сканування дуже малою дисперсією часу відповіді. Результати моделювання показали, що найефективніша стратегія планування роботи дискової пам'яті могла б мати два режими: SCAN - в режимі малих навантажень, C-SCAN – в режимі великих навантажень.

6. Схема Ешенбаха. Каретка рухається так само, як і при застосуванні стратегії C-SCAN, однак, під час обслуговування кожного циліндра виконується доступ до однієї повної доріжки інформації, незалежно від того, чи є ще запити до цього циліндра. Передбачається переупорядкування запитів для обслуговування в рамках одного циліндра з урахуванням кутового положення записів. Однак, якщо два запити відносяться до секторів одного циліндра, що перекриваються, то тільки один з них обслуговується під час поточного ходу каретки. Така схема

мінімізує час пошуку циліндра і час очікування запису. Однак, пізніші дослідження показали, що стратегія C-SCAN з оптимізацією пошуку запису є більш ефективною ніж схема Ешенбаха.

Оптимізація за часом очікування запису

В умовах великих навантажень ймовірність одночасних звернень до конкретного циліндра зростає, стає доцільним проводити оптимізацію не тільки за часом пошуку циліндра, а й за часом очікування запису (пошуку запису). Аналогом стратегії SSTF, яка застосовується під час оптимізації пошуку циліндра, є стратегія SLTF (з найменшим часом очікування - першим). Ця стратегія близька до теоретично оптимальної, до того ж її легко реалізувати.

Коли доцільно планувати роботу з дисками?

Іноді планування може знизити швидкісні характеристики системи. Вид планування вибирається з урахуванням особливостей кожної ОС. Швидкодія залежить від способу організації файлів, конфігурації дискової підсистеми.

6.4 Файлова система

6.4.1 Операції з файлами та їх елементами

Файл - це пойменована сукупність даних. Звичайно розміщується на пристроях зовнішньої пам'яті (диски, стрічки, барабани).

З файлами можливо виконувати такі операції:

- відкрити файл (*open*) - підготовка файлу до звертань;
- закрити файл (*close*) - забороняються подальші звертання до файлу поки він не буде знову відкритий;
- створити файл (*create*) - забезпечує формування нового файлу;
- знищити файл *(destroy)* - руйнує файл;
- копіювати файл *(copy)* - створює ще один екземпляр існуючого файлу з новим іменем;
- перейменувати файл *(rename)* - змінює ім'я файлу;
- вивести файл *(list)* - забезпечується виведення файлу на екран або на принтер.

В рамках файлу можливо виконувати наступні операції з елементами даних:

- прочитати *(read)* – забезпечує введення елемента даних з файлу в процес;
- записати *(write)* – забезпечує виведення елемента даних з процесу в файл;
- поновити *(update)* – забезпечує модифікацію існуючого елемента даних;

- виключити *(delete)* - забезпечує виключення елемента даних з файлу.

Файлова система - це частина загальної системи керування пам'яттю, яка відповідає за керування файлами, що зберігаються на пристроях зовнішньої пам'яті. Вона також відповідає за контрольований розподіл інформації між багатьма користувачами.

6.4.2 Функції файлової системи

Файлова система реалізує безліч функцій. Деякі функції з них такі:

1. Дозволяє користувачам створювати, знищувати і модифікувати файли.
2. Дозволяє користувачам колективно використовувати файли.
3. Повинна бути передбачена можливість контрольованого доступу до файлів – доступ для читання, доступ для запису, доступ для виконання або різні сполучення цих видів доступу.
4. Користувачам повинна надаватись можливість задавати зручні для прикладної програми структури файлів.
5. Надавати можливість користувачам керувати передачею інформації між файлами.
6. В системі повинні бути передбачені засоби захисту і відновлення інформації.
7. ОС повинна надати можливість користувачам звертатись до файлів за допомогою символічних імен, а не через імена фізичних пристроїв.
8. Файлова система повинна забезпечувати дружній до користувача інтерфейс.

6.4.3 Ієрархія даних

Всі дані, що обробляються обчислювальною машиною складаються з *бітів* – двійкових цифр 0 і 1. Біти можуть приймати значення 0 або 1.

На наступному рівні ієрархії даних знаходяться *байти* (8 біт) або *символи*, які мають фіксовану довжину в бітах.

Групу взаємозв’язаних символів називають *полем*. Наприклад, числове поле містить тільки цифри, алфавітне поле – букви і пропуски, символьне поле додатково містить спеціальні символи (наприклад, $367.19).

Група взаємозв’язаних полів називається *записом*. Наприклад, запис, який відноситься до студента може містити окремі поля, де вказується ідентифікаційний номер, прізвище, адрес, номер телефону і т.д. *Ключ запису* – це поле керування, яке однозначно ідентифікує даний запис. Наприклад, в якості ключа запису може використовуватись ідентифікаційний номер студента.

Група взаємозв’язаних записів – це файл. Наприклад файл студентів може містити по одному запису на кожного студента.

Найвищий рівень ієрархії база даних – це група взаємозв’язаних файлів.

6.4.4 Записи і буферизація

Фізичний запис (блок) - це одиниця інформації, яка реально зчитується з пристрою або записується на нього.

Логічний запис - це сукупність даних, які розглядаються користувачем як єдине ціле. Якщо фізичний запис містить декілька логічних, то файл складається із *зблокованих записів*. Якщо ж кожний фізичний запис містить лише один логічний запис, то кажуть, що файл складається з *незблокованих записів*.

Записи можуть мати *фіксовану* (всі записи мають однакову довжину) або *змінну довжину* (записи можуть змінюватись за розміром).

Буферизація дозволяє проводити операції запису-зчитування дискової пам'яті одночасно з операціями обчислення. В основній пам’яті виділяються певні ділянки пам’яті, де можна розташувати декілька фізичних блоків файлу одночасно. Такі ділянки пам’яті називаються *буфером*. Часто використовують *подвійну буферизацію*. Існує два буфери. Спочатку записи, що їх формує працюючий процес, розміщуються в першому буфері, поки не заповнять його. Після цього ініціюється передача даних з першого буфера в зовнішню пам’ять. Але під час цієї передачі процес продовжує формувати записи, які тепер заповнюють другий буфер. Коли другий буфер буде заповнено і завершиться передача з першого буфера, ініціюється передача даних з другого буфера в зовнішню пам’ять, а перший буфер процес знову заповнює записами.

6.4.5 Види організації файлів

Організація файлів - це метод розташування записів файлу в зовнішній пам’яті. Розрізняють такі види організації файлів:

- *Послідовна організація файлів* – організація, при якій записи розміщуються у фізичному порядку, застосовується, в основному, для зберігання інформації на магнітних стрічках, перфострічках. Дискові файли також можуть мати послідовну організацію.
- *Індексно-послідовна* - спосіб організації, коли записи розміщуються у логічній послідовності відповідно із значенням ключів, які містяться у кожному записі; в системі є таблиця, в якій вказуються фізичні адреси головних записів. Доступ до записів здійснюється шляхом пошуку за системним індексом.
- *Пряма організація* - така організація, при якій доступ до записів виконується безпосередньо за їх фізичними адресами. Програма

може розміщувати записи на зовнішні пристрої в будь-якому зручному для неї порядку.

- *Бібліотечна організація* використовується тоді, коли файли складаються з послідовних підфайлів (елементів файла). Початкова адреса кожного елемента зберігається в директорії файлу. Бібліотечні файли використовуються для зберігання програмних бібліотек.

Ці види організації мають багато різних варіантів, характерних для кожної конкретної системи.

Для позначення носія запису використовується термін *том*. Для магнітних стрічок – це касета з магнітною стрічкою, а для дисків – дисковий пакет.

6.4.6 Характеристики файлів

Для файлів можуть бути вказані такі характеристики:

- *розмір файлу* - визначає кількість інформації, що зберігається у файлі;
- *мінливість файлу* - характеризує частоту внесення нових записів і знищення старих. Якщо ця частота мала, то файл називається *статичним*, коли велика – *динамічним* або *мінливим;*
- *активність файлу* - визначається відсотком записів, що обробляються протягом даного прогону.

6.4.7 Засоби файлової системи

Файлова система, як правило, містить такі засоби:

1. *Засоби доступу до файлів*. Визначають конкретну організацію доступу до даних, що зберігаються в файлах.
2. *Засоби керування файлами* - забезпечують зберігання файлів, звертання до них, їх захист та колективне використання.
3. *Засоби керування зовнішньою пам'яттю* - забезпечують розподіл простору зовнішньої пам'яті для розміщення файлів.
4. *Засоби забезпечення цілісності файлів* - відповідають за те, щоб інформація була записана справді в даний файл і справді з нього зчитана.

Головна функція файлової системи - розподіл простору зовнішньої пам'яті і керування її роботою. При цьому файлова система може бути організована за ієрархічним принципом (рис.6.3). Вона містить кореневий каталог, який розміщується у визначеному місці диска, а директорії користувачів зберігаються як звичайні файли.

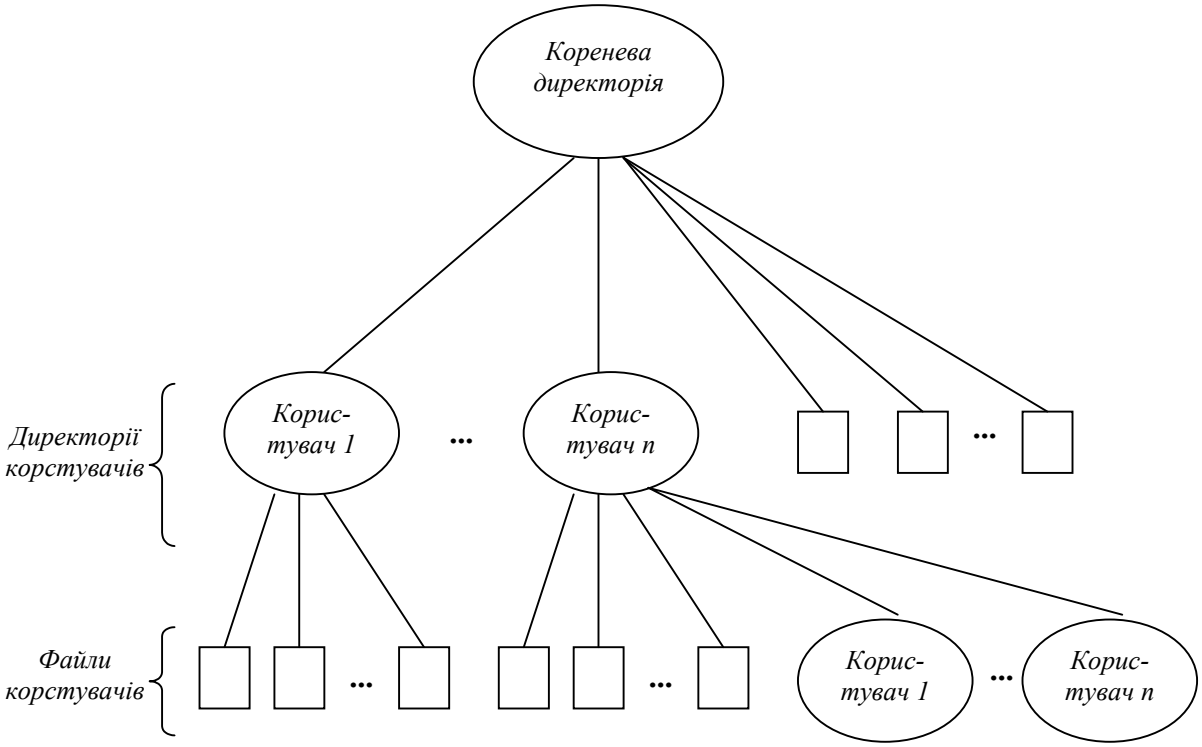


Рисунок 6.3 - Організація файлової системи

6.5 Розподіл зовнішньої пам'яті

Розрізняють наступні види розподілу зовнішньої пам'яті:

- зв'язний розподіл;
- незв'язний.

При ***зв'язному розподілі*** кожному файлу відводиться одна неперервна область зовнішньої пам’яті. Переваги такого розподілу в тому, що збільшується швидкість доступу, оскільки логічні записи розміщені поруч; просто створювати директорії.

До недоліків слід віднести те, що в разі, якщо немає неперервної області достатнього розміру, то файл створити неможливо, а також велику фрагментацію дискової пам’яті.

Незв'язний розподіл – така організація, при якій один файл може займати декілька окремих, не обов’язково сусідніх, блоків зовнішньої пам’яті. Незв’язаний розподіл може бути виконаний:

- за допомогою списків секторів;
- за допомогою поблочного розподілу.

Розподіл за допомогою списку секторів. Пам'ять розглядається як набір секторів; сектори одного файлу містять посилання один на одного. В списку вільного простору містяться всі вільні сектори дискової пам’яті. Недоліком такого розподілу є низька швидкодія через тривалий час

пошуку та через обробку покажчиків, а також зменшення об’єму пам’яті для зберігання даних через велику кількість вказівників.

Поблочний розподіл. Пам'ять розподіляється блоками суміжних секторів (кластерів). Відомі такі реалізації поблочного розподілу:

- за допомогою ланцюжків блоків;
- за допомогою ланцюжків індексних блоків;
- за допомогою таблиць відображення.

В схемі з *ланцюжками блоків* (рис.6.4,а) рядок в директорії користувача вказує на перший блок файлу. Кожний блок, що входить в файл, має фіксовану довжину і містить дві частини: блок даних і вказівник на наступний блок даного файлу.

Недоліком такого розподілу є низька швидкість доступу до блоків.

В схемі з *ланцюгами індексів* вказівники поміщаються в окремі індексні блоки (рис.6.4,б). Кожен індексний блок містить фіксовану кількість елементів, а кожен рядок цього блоку містить ідентифікатор запису і вказівник на цей запис. Якщо для опису файла необхідно декілька індексних блоків, то організується ланцюг цих блоків.

В порівнянні з ланцюгами блоків організація з ланцюгами індексних блоків забезпечує значно більшу швидкість виконання операцій з файлами, оскільки для пошуку необхідного індексного блоку достатньо розглянути лише індексні блоки. Для зменшення часу пошуку індексні блоки можна розміщувати поряд в зовнішній пам’яті.

Головний недолік цієї схеми полягає в тому, що для вставки додаткових записів в файл необхідна повна перебудова структури індексних блоків.

В схемі з *таблицями поблочного відображення* (рис.6.4,в) замість вказівників використовуються номери блоків. Вводиться таблиця відображення файлів, яка містить по одному рядку на кожний блок диска. Рядок в директорії користувача вказує на рядок таблиці відображення, який відповідає першому блоку даного файла. Кожний рядок таблиці відображення містить номер наступного блоку даного файла.

Таким чином, всі блоки файла можна знаходити, послідовно переглядаючи рядки таблиці відображення файлів. В тих рядках таблиці, які відповідають останнім блокам файлів, записується деяке граничне значення (наприклад “EOF”), яке означає що цим блоком закінчується файл. Розглянута схема значно спрощує операції по вставці і видаленню записів файла.

Таблиці відображення блоків використовуються в багатьох популярних файлових системах, зокрема, в операційних системах сімейства Unix.

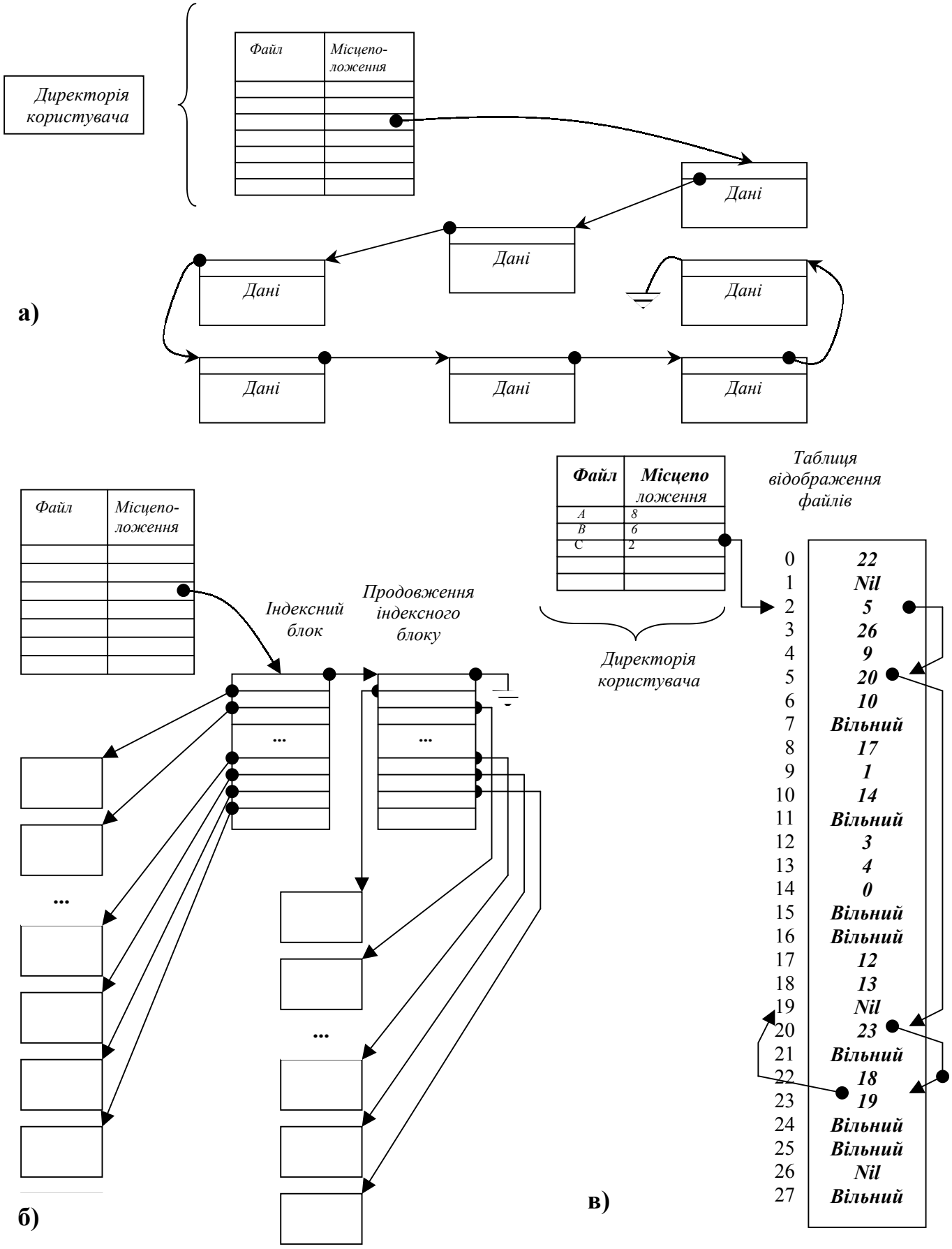


Рисунок 6.4 - Розподіл зовнішньої пам’яті (а – ланцюг блоків, б – ланцюг індексних блоків, в – таблиця поблочного відображення файлів)

6.6 Дескриптор файла

Дескриптор файла або *блок керування файлом* - це блок керування, який містить інформацію, необхідну системі для виконання операцій над файлом. Ця структура в значній мірі залежить від конкретної системи і як правило, містить таку інформацію:

- символічне ім'я файла;
- інформацію про розміщення файла в зовнішній пам'яті;
- тип організації файла (послідовний, індексно-послідовний, прямий, бібліотечний);
- тип пристроїв на яких розміщено файл;
- дані для керуванням доступом до файла;
- тип файла (дані, код програми і т.д.);
- дата і час створення файла;
- дата і час останньої модифікації файла;
- дата і час знищення файла;
- лічильник активності доступу та ін.

Як правило, дескриптори файлів зберігаються у зовнішній пам’яті, і передаються в основну тільки після відкриття відповідного файлу. Користувач не може безпосередньо звертатись до дескриптора файла, оскільки ним управляє файлова система.

6.7. Керування доступом до файлів

Одним з можливих методів керування доступом до файлів полягає у використанні двовимірної матриці керування доступом, в якій указуються всі користувачі і всі файли системи. В ній елементи Aij приймають значення 0, якщо відповідному користувачеві доступ до даного файла заборонений, і 1, якщо доступ дозволений.

Користувач	Файли
	1 2 3 4 5 6 7 . . .
1	1 0 1 1 0 0 0 . . .
2	0 0 1 0 0 0 1 . . .
3	1 0 0 0 0 0 1 . . .
4	0 1 1 0 1 1 0 . . .
...	. . .

В цій матриці замість “1” і “0” можуть використовуватись спеціальні коди для вказання різних прав доступу: “тільки для читання”, “тільки для запису”, “тільки для виконання” і т.д.

В системі з дуже великою кількістю користувачів матриця може бути дуже великою і розрідженою.

Для зменшення розміру матриці використовують списки доступу в залежності від класу користувача. Звичайна класифікація передбачає такі категорії:

- “власник” файла - користувач, який створив даний файл;
- “допущений користувач” – власник аказує, що з його файлом може працювати інша людина;
- “група” або “проект” – всім членам групи надається доступ до файлів зв’язаних з даним проектом;
- загальнодоступний файл [6-8].

Контрольні питання

1. Що таке час пошуку циліндра і час пошуку запису?
2. Коли необхідно планування роботи з магнітними дисками?
3. На оптимізацію чого направлені більшість стратегій планування роботи з магнітними дисками?
4. Охарактеризуйте стратегії оптимізації пошуку циліндра.
5. Охарактеризуйте стратегії оптимізації пошуку запису.
6. Яка різниця між фізичними та логічними записами?
7. Для чого застосовується буферизація під час роботі з магнітними дисками?
8. Що таке файл і які операції з ним можна виконувати?
9. Дайте порівняльну характеристику різних видів організації файлів.
10. Охарактеризуйте ієрархію даних в операційних системах.
11. Що таке файлова система і які її основні функції?
12. Які засоби містить файлова система та її організація?
13. Дайте порівняльний аналіз різних підходів до розподілу зовнішньої пам’яті.
14. Що таке дескриптор файла і його вміст?
15. Охарактеризуйте методи керування доступом до файла.

7 Методи захисту інформації в операційних системах

Широке застосування комп'ютерів та комп'ютерних мереж в різних сферах життєдіяльності людини, ставить питання захисту операційних систем та інформації, що зберігається в комп'ютерах на одне з перших місць.

7.1 Загальні відомості

У більшості операційних систем є механізми ідентифікації користувача, які забезпечують той чи інший рівень захисту інформації. Основні методи захисту інформації в операційних системах наступні:

- захист інформації за допомогою матриці управління доступом та списків управління доступом;
- захист інформації за допомогою "паролів";
- захист інформації за допомогою шифрування-дешифрування (криптографія).

Недоліки двох перших методів полягають у тому, що "ключі" доступу зберігаються в самій системі. Це може призвести до того, що підготовлений недобросовісний користувач може їх розкрити і скористатись секретною інформацією.

При шифруванні інформації ключ кодування не повинен зберігатись у системі. Користувач вводить його тільки тоді, коли зашифровує або розшифровує інформацію.

Питання шифрування-дешифрування інформації є предметом дисципліни під назвою "криптографія". Розроблено ряд стандартів, які забезпечують надійний захист інформації. Найбільш поширеними є дві схеми шифрування – DES (Data Encryption Standard) і RSA (отримав назву по перших буквах прізвищ авторів – Rivest, Shamir, Adleman). DES-схема симетрична, в ній для шифрування і дешифрування використовується один і той же ключ. Схема RSA асиметрична, ключі шифрування і дешифрування в ній різні [8,9].

7.2 Шифрування інформації згідно з стандартом DES

Алгоритм застосовується для шифрування і дешифрування блоків даних довжиною 64 біти під управлінням 64-бітового ключа. До блоку, який підлягає шифруванню, застосовується початкова перестановка (IP), потім складна обчислювальна процедура в залежності від ключа, а в кінці зворотна перестановка (IP**(-1)). Загальна схема процесу шифрування приведена на рис.7.1.

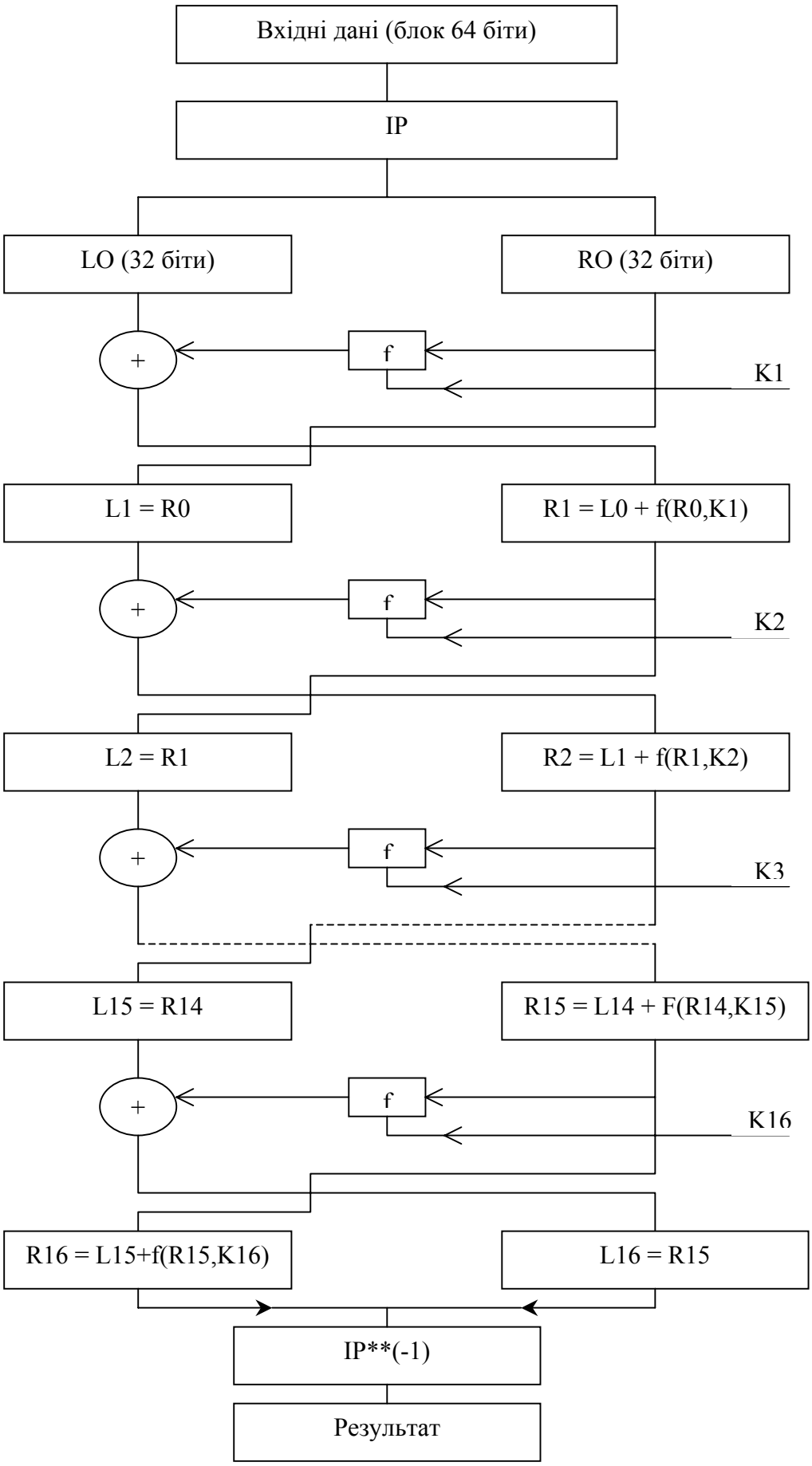


Рисунок 7.1 - Шифрування інформації згідно зі стандартом DES

Початкова перестановка IP визначається такою таблицею:

I P: 58 50 42 34 26 18 10 02
60 52 44 36 28 20 12 04
62 54 46 38 30 22 14 06
64 56 48 40 32 24 16 08
57 49 41 33 25 17 09 01
59 51 43 35 27 19 11 03
61 53 45 37 29 21 13 05
63 55 47 39 31 23 15 07

a(i,j) - номер біта вхідної 64-бітової послідовності

Зворотна перестановка IP**(-1) має такий вигляд:

I P $^{-1}$: 40 08 48 16 56 24 64 32
39 07 47 15 55 23 63 31
38 06 46 14 54 22 62 30
37 05 45 13 53 21 61 29
36 04 44 12 52 20 60 28
35 03 43 11 51 19 59 27
34 02 42 10 50 18 58 26
33 01 41 09 49 17 57 25

L,R - 32-бітні послідовності;
K1...K16 - 48-бітні послідовності, які вибираються з 64-бітного ключа;
"+" - додавання за модулем "2";

Таким чином, процес шифрування описується такими виразами:

$$L(N) = R(N-1)$$
$$R(N) = L(N-1) + f(R(N-1),K(N)) \quad (1)$$
$$K(N) = KS(N, KEY),$$

де KS - функція вибору ключів,
f(R,K) - функція шифрування.

Дешифрування

Оскільки застосовується додавання за модулем "2", то з (1) випливає

$$R(N-1) = L(N)$$
$$L(N-1) = R(N) + f(R(N-1),K(N)) \quad . \quad (2)$$

Тобто для шифрування необхідно застосувати той самий алгоритм, що й для дешифрування, тільки ключі вибираються в зворотному порядку.

Функція шифрування f(R,K)

Загальна схема обчислення f(R,K) приведена на рис. 7.2. На цьому рисунку "E" позначає функцію, для якої вхідним є блок із 32 біт, а вихідним - блок із 48 біт. Вихідні 48 біт, записані у вигляді восьми блоків по 6 біт, отримують шляхом вибору бітів із вхідних даних у відповідності із наступною таблицею.

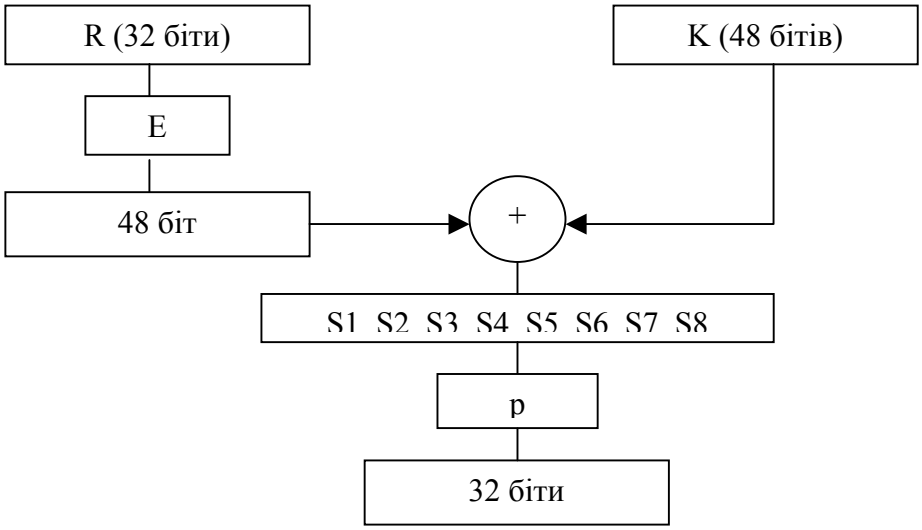


Рисунок 7.2 - Обчислення функції шифрування f(R,K)

Таблиця вибору бітів для "E":

32 01 02 03 04 05
04 05 06 07 08 09
08 09 10 11 12 13
12 13 14 15 16 17
16 17 18 19 20 21
20 21 22 23 24 25
24 25 26 27 28 29
28 29 30 31 32 01

Для функцій S1...S8 - вхідні дані блоки по 6 біт. Перший і останній біти цієї послідовності задають номер рядка, а 4 середніх біта - номер стовпця. Наприклад, для S1:

011011 - вхідний блок;
01(1) - номер рядка;
1101(13) - номер стовпця;
0101(5) - вихідні дані.

Нумерація стовпців і рядків починається з "0". Примітивні функції S1, ... , S8 наступні:

S1: 14 04 13 01 02 15 11 08 03 10 06 12 05 09 00 07
00 15 07 04 14 02 13 01 10 06 12 11 09 05 03 08
04 01 14 08 13 06 02 11 15 12 09 07 03 10 05 00
15 12 08 02 04 09 01 07 05 11 03 14 10 00 06 13

S2: 15 01 08 14 06 11 03 04 09 07 02 13 12 00 05 10
03 13 04 07 15 02 08 14 12 00 01 10 06 09 11 05
00 14 07 11 10 04 13 01 05 08 12 06 09 03 02 15
13 08 10 01 03 15 04 02 11 06 07 12 00 05 14 09

S3: 10 00 09 14 06 03 15 05 01 13 12 07 11 04 02 08
13 07 00 09 03 04 06 10 02 08 05 14 12 11 15 01
13 06 04 09 08 15 03 00 11 01 02 12 05 10 14 07
01 10 13 00 06 09 08 07 04 15 14 03 11 05 02 12

S4: 07 13 14 03 00 06 09 10 01 02 08 05 11 12 04 15
13 08 11 05 06 15 00 03 04 07 02 12 01 10 14 09
10 06 09 00 12 11 07 13 15 01 03 14 05 02 08 04
03 15 00 06 10 01 13 08 09 04 05 11 12 07 02 14

S5: 02 12 04 01 07 10 11 06 08 05 03 15 13 00 14 09
14 11 02 12 04 07 13 01 05 00 15 10 03 09 08 06
04 02 01 11 10 13 07 08 15 09 12 05 06 03 00 14
11 08 12 07 01 14 02 13 06 15 00 09 10 04 05 03

S6: 12 01 10 15 09 02 06 08 00 13 03 04 14 07 05 11
10 15 04 02 07 12 09 05 06 01 13 14 00 11 03 08
09 14 15 05 02 08 12 03 07 00 04 10 01 13 11 06
04 03 02 12 09 05 15 10 11 14 01 07 06 00 08 13

S7: 04 11 02 14 15 00 08 13 03 12 09 07 05 10 06 01
13 00 11 07 04 09 01 10 14 03 05 12 02 15 08 06
01 04 11 13 12 03 07 14 10 15 06 08 00 05 09 02
06 11 13 08 01 04 10 07 09 05 00 15 14 02 03 12

S8: 13 02 08 04 06 15 11 01 10 09 03 14 05 00 12 07
01 15 13 08 10 03 07 04 12 05 06 11 00 14 09 02
07 11 04 01 09 12 14 02 00 06 10 13 15 03 05 08
02 01 14 07 04 10 08 13 15 12 09 00 03 05 06 11

Перестановна функція "P" породжує 32-бітовий результат із 32-бітових вхідних послідовностей. Вона визначається наведеною нижче таблицею.

Перестановка "P":

16 07 20 21
29 12 28 17
01 1523 26
05 1831 10
02 0824 14
32 2703 09
19 1330 06
22 1104 25

Функція вибору ключів KS

Схема вибору 48-бітових блоків K1, ... , K16 з 64-бітового ключа приведена на рис. 7.3.

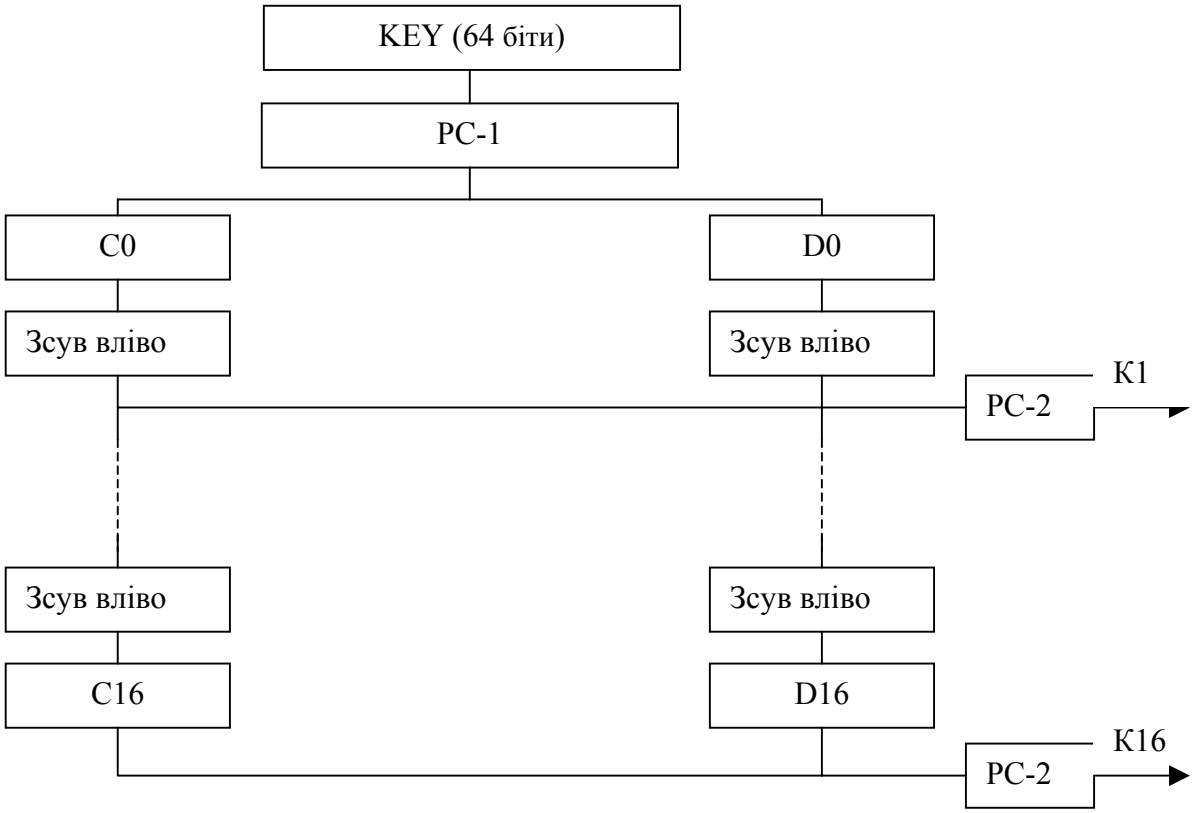


Рисунок 7.3 - Вибір ключів

Способи отримання блоків C0, D0, блоків C(n), D(n) із блоків C(n-1), D(n-1), а також ключів K1, ... , K16 визначаються такими таблицями.

Номер ітерації	Число зсувів вліво
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Перестановка PC-1

C0:

57 49 41 33 25 17 09
01 58 50 42 34 26 18
10 02 59 51 43 35 27
19 11 03 60 52 44 36

D0:

63 55 47 39 31 23 15
07 62 54 46 38 30 22
14 06 61 53 45 37 29
21 13 05 28 20 12 04

Примітка:

Біти 8,16, ... , 64 використовуються для контролю по непарності в кожному байті. Зсув вліво циклічний

Перестановка PC-2: 11 24 01 05
15 06 21 10
12 04 26 08
27 20 13 02
31 37 47 55
51 45 33 48
39 56 34 53
50 36 29 32

7.3 Інші схеми шифрування

Алгоритм RSA

Щоб використовувати алгоритм RSA, необхідно спочатку згенерувати відкритий і секретний ключі, виконавши такі кроки:

1. Виберемо два дуже великі прості числа p і q.
2. Визначимо n = p*q.
3. Виберемо велике випадкове число d, яке є взаємно-простим з результатом множення (p-1)*(q-1).
4. Визначимо таке число e, для якого істинним є співвідношення

$$(e*d) \bmod ((p-1)*(q-1)) = 1.$$

5. Назвемо відкритим ключем числа {e, n}, а секретним ключем числа {d, n}.

Тепер, щоб зашифрувати дані по відкритому ключу {e, n}, необхідно:

1. Розбити текст, що шифрується, на блоки довжиною по n символів і представити кожний символ блоку числом M(i) = 0,1, ..., n-1.
2. Зашифрувати текст як послідовність чисел M(i) за формулою

$$C(i) = (M(i)^{**}e) \bmod n.$$

Щоб розшифрувати ці дані з використанням секретного ключа {d, n}, необхідно виконати такі обчислення:

$$M(i) = (C(i)^{**}d) \bmod n.$$

Тепер тільки необхідно, використовуючи табличні перетворення, за значенням M(i) визначити початковий код символу.

Розроблено також і вітчизняний стандарт шифрування даних - ГОСТ 28147-89. Однак його програмна реалізація дуже складна і практично немає ніякого сенсу через низьку швидкодію.

Генератор псевдовипадкових чисел

У системах, які не вимагають високого ступеня захисту, можуть використовуватись шифри з меншою криптостійкістю. Реалізація шифрування з використанням цих методів набагато простіша, крім того,

вони забезпечують високу швидкодію. Одним із таких методів є шифрування за допомогою датчика псевдовипадкових чисел(ПСЧ). Принцип шифрування полягає в генерації гами шифру за допомогою датчика ПСЧ і накладенні отриманої гами на відкриті дані, так щоб можна було відновити початкові дані (наприклад, за рахунок використання додавання за модулем "2").

Процес дешифрування зводиться до повторної генерації гами шифру за відомим ключем і накладенням її на зашифровані дані.

Найбільш доступним і ефективним є конгруентний лінійний генератор ПСЧ, який формує послідовність псевдовипадкових чисел T(i) відповідно до співвідношення

$$T(i+1) = (A*T(i)+C) \bmod M,$$

де A, C - константи, T(0) - початкове задане число, M = (2**b), b - довжина слова ЕОМ у бітах. Період повторення псевдовипадкових чисел залежить від вибраних значень A і C. Лінійний конгруентний датчик має максимальну довжину M тоді і тільки тоді, коли C-непарне і Amod 4 = 1.

Контрольні питання

1. Які методи захисту інформації застосовують в ОС?
2. Що таке матриця доступу?
3. Поясніть застосування списків доступу?
4. Які недоліки методів захисту інформації за допомогою паролів, матриць доступу та списків доступу?
5. Які ви знаєте типи алгоритмів шифрування інформації?
6. В чому різниця між DES і RSA схемами шифрування?
7. Поясніть сутність алгоритму DES.
8. Поясніть сутність алгоритму RSA.
9. Поясніть принципи шифрування інформації за допомогою генератора ПСЧ.
10. Дайте порівняльний аналіз алгоритмів шифрування інформації.

8 Оцінювання продуктивності операційних систем

Оскільки операційна система – це в першу чергу система керування ресурсами, то для розробника важливо вміти визначати наскільки ефективно операційна система використовує ресурси.

8.1 Методи оцінювання продуктивності операційних систем

Відомі такі методи оцінювання продуктивності операційних систем:

- елементарні годинники;
- суміші команд;
- зразкові програми;
- аналітичні моделі;
- вимірювальні програми;
- синтетичні програми;
- моделювання.

Елементарні годинники використовують для порівняння апаратури за часом виконання декількох основних апаратних операцій. Однак, обчислювальна система може мати набір команд, який містить сотні різних операцій. Тому порівняння процесорів за елементарним часом дає дуже мало інформації.

У методі *суміші команд* використовується зважений середній час виконання декількох різних команд для оцінювання параметрів обчислювальної системи. При цьому коректний вибір суміші команд і їх ваги є дуже складним завданням.

Зразкові програми - це типові програми, які могли б виконуватись на даній машині. Вони можуть бути корисними під час оцінювання компонент програмного забезпечення. Використання зразкових програм потребує застосування великого об’єму ручної роботи.

Вимірювальні програми - це реальні програми, які виконуються на досліджуваній машині. Як вимірювальні, беруть типові для даного класу задач програми, які вирішуються на даній установці. Слабким місцем цього методу є суб'єктивність при виборі вимірювальних програм.

Синтетичні програми поєднують у собі риси зразкових і вимірювальних програм. Це реальні програми, які спеціально складені для оцінювання визначених можливостей машини. Складання таких програм вимагає високої кваліфікації програміста і неабияких витрат часу.

Під час *моделювання* розробляється машинна модель випробовуваної системи. Поведінка моделі за деякий проміжок модельного часу відображає функціонування оцінюваної системи. Моделювання вимагає великих витрат машинного часу, породжує вихідні дані гігантських розмірів [8].

8.2 Аналітичне моделювання комп'ютерних систем

На відміну від інших методів, *аналітичні моделі* дозволяють швидко отримати більш-менш точні оцінки продуктивності обчислювальної системи, хоча застосовувати їх можуть тільки кваліфіковані математики.

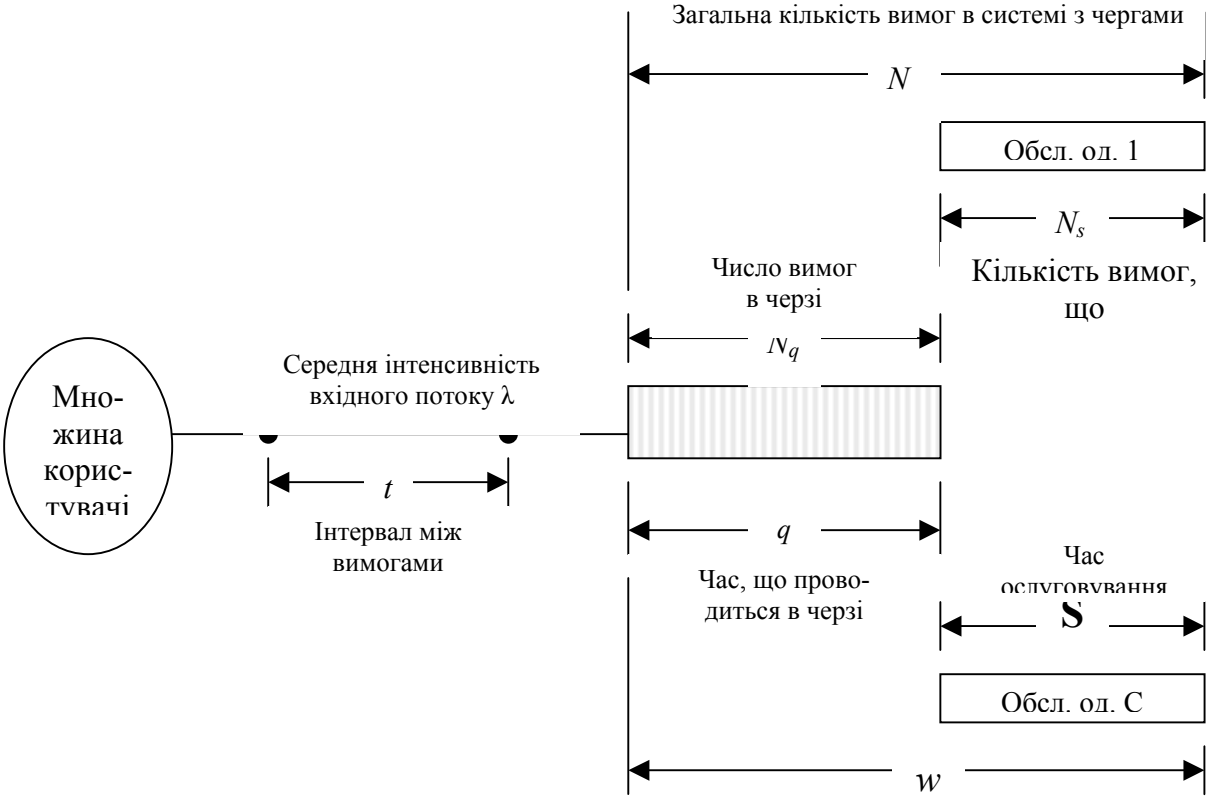


Рисунок 8.1 - Імовірні величини в моделях теорії черг
(w - загальний час знаходження заявки в системі)

Аналітична модель - це математичне представлення обчислювальної системи або її компонент. Знаходять застосування математичні моделі багатьох видів, однак, найбільш "гнучкими" і корисними є моделі, створені на основі теорії черг і марковських процесів.

Деякі ймовірні величини в типовій системі з чергою приведені на рис. 8.1 [8,10]. Заявки можуть надходити в систему відповідно до будь-якого закону розподілу. Однак, в елементарній теорії черг вважають, що заявки, що надходять, утворюють пуасонівський процес. Тому розглядаються тільки M/M/c системи.

8.2.1 Основні математичні формули для M/M/c систем

1. Приведена інтенсивність:

$$u = \frac{E(s)}{E(t)} = \frac{\lambda}{\mu} = \lambda \cdot E(s),$$

λ - середня інтенсивність надходження заявок у систему ($\lambda = \frac{1}{E(t)}$);

μ - середня інтенсивність обслуговування ($\mu = \frac{1}{E(s)}$);

E(s) - очікуваний час обслуговування однієї заявки,
E(t) - очікуваний інтервал між заявками.

2. Навантаження на ресурс:

$$\rho = \frac{u}{c} = \frac{\lambda}{\mu \cdot c},$$

c - число пристроїв, що обслуговують заявки.

3. Імовірність того, що всі пристрої зайняті і нові заявки, що надходять, повинні чекати в черзі:

$$C(c,u) = \frac{\frac{u^c}{c!}}{\frac{u^c}{c!} + (1-\rho) \cdot \sum_{n=0}^{c-1} \frac{u^n}{n!}}.$$

4. Середній час у черзі:

$$W_q = \frac{C(c,u) \cdot E(s)}{c \cdot (1-\rho)}.$$

5. Середній час у системі: $W = W_q + E(s).$

6. Максимальний час у черзі для 90% заявок:

$$\pi_q(90) = \frac{E(s)}{c \cdot *1-\rho)} \cdot \ln(10C(c,u)).$$

7. Очікуване число заявок у системі: $L = \lambda \cdot W.$

Очікуване число заявок у черзі: $L_q = \lambda \cdot W_q.$

8.2.2 Процеси розмноження і загибелі

Часто для описання системи можна використовувати множину дискретних станів S0, S1, S2, ..., Sn , в одному з яких може знаходитись система. Зручною моделлю для описання таких систем є марковські процеси. Поведінка марковського процесу в майбутньому визначається поточним станом системи і перехідними імовірностями.

В моделюванні комп'ютерних систем особливо часто застосовують марківські процеси розмноження та загибелі (рис.8.2). Дослідження цього класу процесів простіше, ніж загальних процесів Маркова [8, 10].

Процеси розмноження і загибелі - це один з видів марківських процесів.

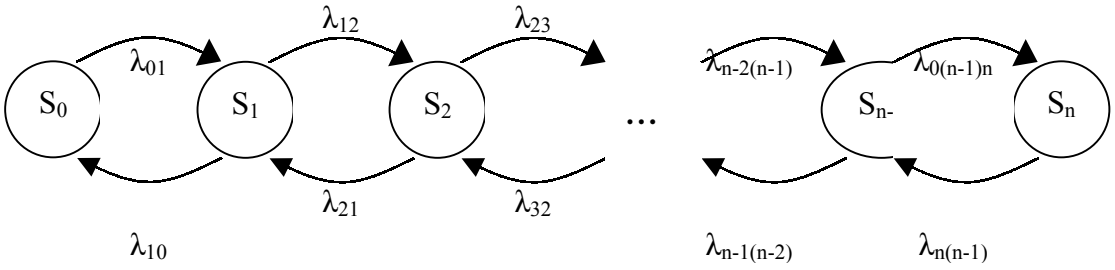


Рисунок 8.2 - Процеси розмноження і загибелі

Неперервний марковський процес має такі властивості:

$$\lambda_{ij} = 0, \text{ якщо } j \neq i+1,\ j \neq i-1,$$

де λ_{ij} – інтенсивність переходів із стану S_i в стан S_j.
Введемо позначення:

$\lambda_{i(i+1)} = b_i$ - середня інтенсивність розмноження в стані S_i;

$\lambda_{i(i-1)} = d_i$ - середня інтенсивність загибелі в стані S_i.

Імовірність того, що процес знаходиться в стані S_i, позначимо P_i. Тоді

$$P_{i+1} = \frac{b_i}{d_{i+1}} \cdot P_i .$$

Середнє число необслужених у системі заявок E_i дорівнює:

$$E_i = P_1 + 2 \cdot P_2 + \ldots + n \cdot P_n .$$

З урахуванням того, що

$$P_1 + P_2 + \ldots + P_n = 1,$$

ми можемо знайти розв'язок неперервного процесу розмноження і загибелі.

Контрольні питання

1. Охарактеризуйте основні методи оцінювання продуктивності ОС.
2. Які ви знаєте програми моделювання?
3. Синтетичні програми.
4. Вузькі місця в ОС. Як їх знайти?
5. Зворотний зв'язок. Як від нього залежить продуктивність систем?
6. Типова система з чергою.
7. Основний закон розподілу в елементарній теорії черг.
8. Сталий і перехідний режими.
9. Результат Літтла.
10. Основні формули для розрахунку M/M/c систем.
11. Визначення марковського процесу.
12. Моделі процесів розмноження і загибелі.
13. Переваги і недоліки аналітичних методів оцінювання продуктивності операційних систем.

Література

1. Пєтух А.М., Майданюк В.П. Інтерфейс “Користувач-комп’ютер”: Навчальний посібник. – Вінниця: ВДТУ, 1999. – 66 с.
2. Системне програмування і операційні системи /Г.О.Лосєв, С.І.Перевозніков, О.В.Сілагін – Вінниця: ВДТУ, 2000. – 66 с.
3. Методичні вказівки до виконання лабораторних робіт з дисципліни “Операційні системи ЕОМ” для студентів бакалаврського напрямку 6.804 – “Комп’ютерні науки”, спеціальності інженерії 7.080403 – “Програмне забезпечення обчислювальної техніки і автоматизованих систем”/ Уклад. В.П. Майданюк, О.Н. Романюк – Вінниця, ВДТУ, 1997. – 41 с.
4. Соловьев Г.Н., Никитин В.Д. Операционные системы ЕОМ. - М.: Высш.шк.,1989.
5. Дейтел Г. Введение в операционные системы: В 2-х т. Т.1: Пер. с англ. - М.: Мир, 1987.
6. Мак-Федрис Пол. Руководство по Windows-98: Пер. с англ. – М.: Бином, 1999. – 400 с.
7. Мэдник С. Операционные системы: Пер. с англ. - М.: Мир, 1980.
8. Дейтел Г. Введение в операционные системы: в 2-х т. Т.2: Пер. с англ. - М.: Мир, 1987.
9. Защита информации в персональних ЭВМ/ Спесивцев А.В., Вегнер В.А., Крутяков А.Ю. и др. - М.: Радио и связь, МП "Веста", 1992.
10. Коваленко И.М., Гнеденко Б.В. Теория вероятностей. - К.: Выща школа, 1990.