УПРАВЛІННЯ ПАМ’ЯТТЮ В ОПЕРАЦІЙНИХ СИСТЕМАХ

Пам'ять в операційних системах - це ресурс комп'ютера, призначений для зберігання програмного коду і даних. Пам'ять зображають як масив машинних слів або байтів з їхніми адресами.

Існують різні види пам'яті, які знаходяться в певній ієрархії. **Жорсткий диск комп'ютера** (допоміжний запам'ятовувальний пристрій – *secondary storage*) є найдешевшим і найповільнішим запам'ятовувальним пристроєм. **Оперативна пам'ять** зберігається в мікросхемах пам'яті, її називають **основною пам'яттю** (*main memory*) – вона є швидшою й дорожчою. Швидшими засобами зберігання даних є **кеші** процесора і обсяг цих кешів обмежений.

Керування пам'яттю в ОС – досить складне завдання. Переважно потрібної за характеристиками пам'яті виявляється недостатньо, і щоб це не заважало роботі користувача, стає необхідним координувати між собою різні види пам'яті.

Управління пам'яттю поряд з управлінням процесами і ресурсами, – одна з найбільш важливих функцій операційної системи.

Будь-яка програма, введена в систему, повинна бути розміщена в пам'яті і оформлена у вигляді процесу для її виконання. Кожна програма при введенні в систему поміщається у вхідну чергу – сукупність процесів на диску, які очікують розміщення в пам'яті для виконання своїх програм. До свого виконання користувацькі програми проходять у системі декілька стадій.

Багатоетапна обробка користувацької програми.

Вихідний код програми (у формі текстового файлу) мовою високого рівня або на асемблері перетворюється компілятором або асемблером в об'єктний модуль, що містить бінарні виконувані машинні команди і таблицю символів, визначених і використаних в даному модулі коду. Розглянута фаза називається часом компіляції.

Однак об'єктний модуль не може безпосередньо виконуватися, оскільки він містить недозволені посилання на зовнішні модулі та їх компоненти. Наступна фаза обробки програми – редагування зв'язків. Редактор зв'язків

(linker) – системна програма, яка отримує на вхід один або кілька об'єктних модулів, а на виході видає завантажувальний модуль – двійковий код, утворений кодом декількох об'єктних модулів, в якому дозволені всі міжмодульні посилання – для кожного символу, зовнішнього для даного об'єктного модуля.

Завантажувальний модуль може бути завантажений в пам'ять для виконання з допомогою ще однієї системної програми – завантажувача (loader), який отримує на вхід завантажувальний модуль і файли з бінарними кодами системних бібліотек, які використовує програма. Завантажувач, об'єднуючи код програми з кодами системних бібліотек, створює бінарний образ програми в пам'яті.

Логічний та фізичний адресний простір.

Концепція логічного адресного простору, пов'язаного з відповідним фізичним простором, є однією з основних для управління пам'яттю.

Логічною адресою називається адреса, що генерована процесором при виконанні машинної команди.

Фізична адреса – це реальна адреса в пам'яті, яку «бачить» і «розуміє» пристрій управління пам'яттю (Memory Management Unit - MMU).

Логічні адреси збігаються з фізичними при зв'язуванні адрес під час компіляції або під час завантаження (тобто до виконання програми). Однак при зв'язуванні адрес під час виконання логічні адреси відрізняються від фізичних.

Пристрій управління пам'яттю.

Пристрій управління пам'яттю (*Memory Management Unit – MMU*) – це один з модулів апаратури, що відповідає за адресацію пам'яті і пов'язаний з процесором та іншими пристроями системною шиною. З точки зору підтримки описаних концепцій адресації, пристрій управління пам'яттю – це апаратура, перетворююча логічну адресу (отримана по загальній шині від процесора) у фізичний (реальна адреса в пам'яті, за якою і відбувається звернення).

Апаратура MMU використовує значення регістру переміщення, що містить адресу початку області пам'яті, виділеної ОС для програми користувача. MMU додає значення регістра переміщення до (логічної) адреси, що згенерована програмою користувача, отримуючи в результаті фізичну адресу.

Програма користувача працює тільки з логічними адресами і не «бачить» фізичних адрес. Схема адресації і перетворення логічної адреси у фізичну з використанням регістра переміщення зображена на рисунку 3.1.

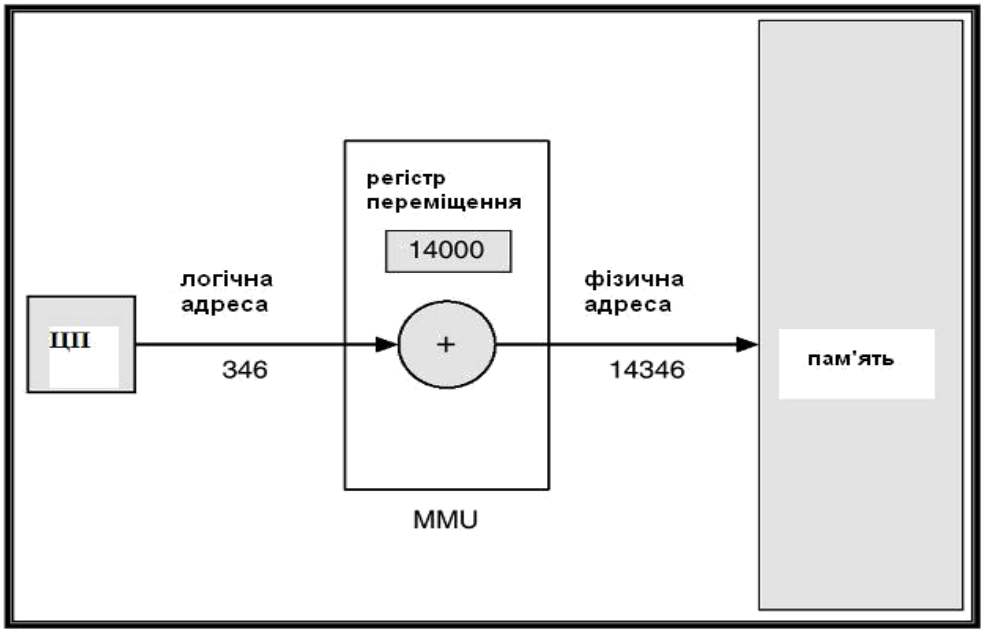

Рисунок 3.1 – Адресація з використанням регістру переміщення

Відкачування і підкачка.

Користувацький процес може перебувати в різних станах під час обробки системою. Зокрема, процес може бути деякий час неактивним, якщо, наприклад, він виконується в режимі поділу часу, і користувач за терміналом обмірковує наступну команду або редагує вихідний код своєї програми.

подібних випадках процес може бути відкачано операційною системою на диск тому, що пам'ять, яку він займає необхідна в даний момент для іншого активного процесу.

Відкачування і підкачка (swapping) – це дії операційної системи з відкачування (запису) образу неактивного процесу на диск або підкачування (зчитування) активного процесу в основну пам'ять. Необхідність виконання подібних дій викликана нестачею основної пам'яті.

Файл відкачування (backing store) – область дискової пам'яті, використовувана операційною системою для зберігання образів відкачаних процесів. Файл відкачування організується максимально ефективно: забезпечується прямий доступ до всіх образів процесів в пам'яті (наприклад, через таблицю за номером процесу).

Популярний різновид стратегії відкачування і підкачки – ***roll out / roll in:*** відкачування і підкачка на базі пріоритетів; більш пріоритетні процеси виконуються, менш пріоритетні – відкачуються на диск.

При відкачуванні найбільше часу витрачається при передачі даних: повний образ процесу може займати велику область пам'яті. Загальний час відкачування пропорційний розміру відкачуваних даних.

поширених ОС – UNIX, Linux, Windows та ін – реалізовані різні стратегії відкачування і підкачки.

Схема відкачування та підкачування зображена на рисунку 3.2.

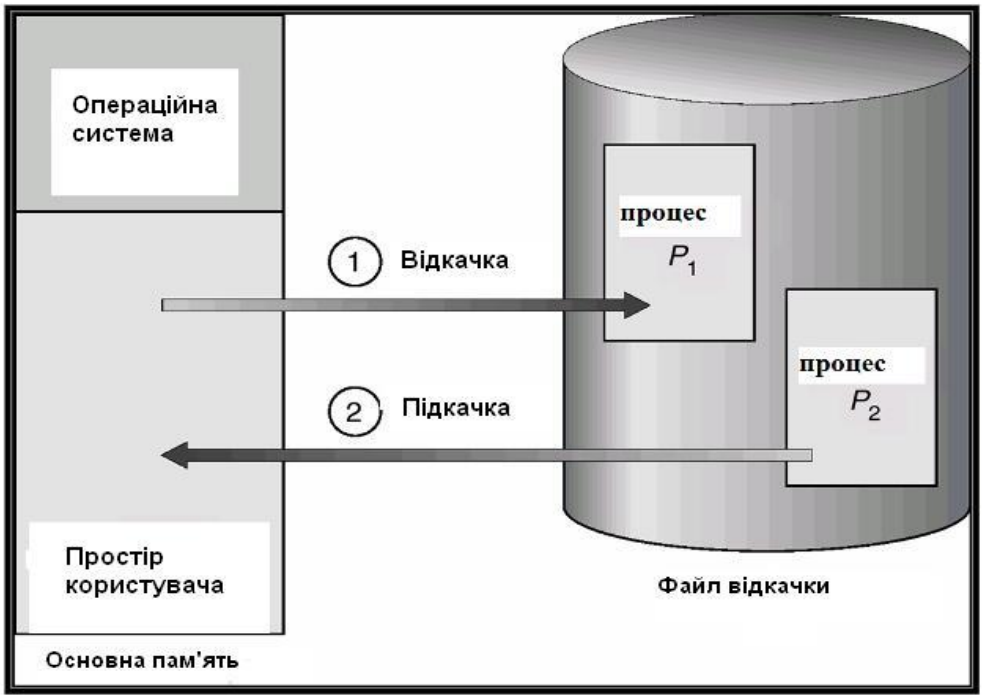

Рисунок 3.2 – Схема підкачки і відкачки

3.1 Технології розподілу пам'яті

Оперативна пам'ять (ОП) – це важливий ресурс ЕОМ. У ранніх ОС управління пам'яттю зводилося до завантаження програми і її даних із зовнішньої пам'яті в ОП.

Функції мультипрограмної ОС по управлінню пам'яттю:

відстеження вільної і зайнятої пам'яті;

виділення пам'яті процесам і звільнення пам'яті по завершенні цих процесів;

захист пам'яті;

настроювання адрес програм;

виділення динамічної пам'яті;

переміщення програм між ОП і зовнішньою пам'яттю. При виконанні останньої функції існує дві стратегії:

1) *свопінг*, або проста підкачка – програми цілком завантажуються в ОП і цілком вивантажуються на диск;

2) програми частково завантажуються в ОП і частково вивантажуються на диск; якщо це прозоро для програміста, то така стратегія називається *віртуальною пам'яттю*, а якщо не прозоро, то *оверлеєм*.

Головна операція по управлінню пам'яттю – це розміщення програми в ОП для її виконання процесором. У сучасних ОС при цьому використовується складна система, що називається *віртуальною пам'яттю*, яка використовує одну чи обидві базові технології – сегменти і сторінки. Але спочатку ми познайомимося з більш простими технологіями.

3.1.1 Фіксований розподіл пам'яті

У цьому випадку пам'ять поділяється на області з фіксованими межами.

Розділи однакового розміру. Будь-який процес, розміри якого не перевищують розміру розділу, може бути завантажений в будь-який доступний розділ. Якщо ж всі розділи зайняті і немає жодного процесу в стані готовності або роботи, то ОС може вивантажити процес з будь-якого розділу і завантажити нього інший процес, забезпечуючи процесор роботою. Прийняття рішення, який процес вивантажити, – це завдання планувальника.

Недолік даного способу – неефективне використання ОП, так як маленька програма цілком займає великий розділ. Це явище називається *внутрішньою фрагментацією.*

Розділи різного розміру. Цей спосіб не усуває повністю недолік, зазначений вище, але зменшує його вплив.

При призначенні розділу процесу існує два підходи:

кожен процес можна розмістити в найменшому розділі, здатному повністю вмістити даний процес; це мінімізує внутрішню фрагментацію, але для кожного розділу потрібна своя черга планувальника;

кожен процес можна розмістити в найменшому доступному розділі, здатному повністю вмістити даний процес; при цьому використовується єдина черга планувальника.

Недоліки методу фіксованого розподілу:

• кількість розділів, що визначена в момент генерації системи, обмежує кількість активних процесів;

• так як розміри розділів встановлюються заздалегідь, це призводить до неефективного використання пам'яті.

Фіксований розподіл пам'яті є не ефективним і тому не застосовується в сучасних ОС.

3.1.2 Динамічний розподіл

цьому випадку в пам'яті формується змінна кількість розділів змінної довжини, а при розміщенні процесу в ОП для нього виділяється строго необхідна кількість ОП. Цьому методу властива *зовнішня фрагментація*, тобто фрагментованою стає пам'ять, зовнішня по відношенню до всіх розділів. Для позбавлення від зовнішньої фрагментації ОС здійснює ущільнення: час від часу процеси переміщаються в пам'яті так, щоб вони займали суміжні області ОП. Для виконання такого ущільнення потрібен додатковий час і здатність програм до динамічного переміщення.

При організації динамічного розподілу використовується три основних алгоритми розміщення процесу в пам'яті:

найкращий підходящий-вибирається блок, розмір якого найбільш близький до необхідного;

перший підходящий – перевіряються всі вільні блоки з початку ОП і вибирається перший з них, достатній за розміром;

наступний підходящий – перевіряються всі вільні блоки, починаючи з того місця, де востаннє був виділений блок, і вибирається перший блок, достатній за розміром.

3.1.3 Система двійників

Як вже було сказано, фіксований розподіл обмежує кількість активних процесів і неефективно використовує ОП при невідповідності розмірів розділів процесів. Динамічний розподіл вимагає більш складної організації і великих накладних витрат на ущільнення пам'яті.

Система двійників є компромісом між цими двома способами. При її використанні ОП розподіляється блоками розміром $2k$, $L \le k \le U$, де $2L$-мінімальний розмір блоку, що виділяється, а $2U$-максимальний блок, що розподіляється (розмір всієї пам'яті, що доступний для розподілу).

Спочатку весь доступний для розподілу простір ОП розглядається як єдиний блок розміру $2U$. При запиті блоку довільного розміру s, якщо $2U-1 \le s \le 2U$, то виділяється весь блок. В іншому випадку цей блок розділяється на два блоки-двійники з розмірами $2U-1$. Якщо тепер $2U-2 \le s \le 2U-1$, то за запитом виділяється один з двох цих двійників, в іншому випадку один з двійників знову ділиться навпіл. Цей процес продовжується до тих пір, поки не буде згенеровано найменший блок, розмір якого не менше s.

При всьому цьому система постійно веде список доступних блоків для кожного розміру $2i$. Доступний блок може бути вилучений зі списку ($2i + 1$) поділом його навпіл і внесенням двох нових доступних блоків розміру $2i$ в список i. Коли ж пара двійників у списку i опиняється звільненою, вони видаляються зі списку і об'єднуються в єдиний блок в списку ($i + 1$).

Модифікована версія системи двійників використовується для розподілу пам'яті ядром ОС Linux.

3.1.4 Проста сторінкова організація

При використанні цього методу ОП розділяється на ряд кадрів рівного розміру.

Кожен процес також розділяється на сторінки рівного розміру, що збігається з розміром кадрів.

Процес завантажується шляхом завантаження всіх його сторінок в доступні, але не обов'язково суміжні кадри. При цьому відсутня зовнішня фрагментація, але є невелика внутрішня фрагментація, що є складовою частиною останньої сторінки процесу.

На рисунку 3.3 показано використання сторінок і кадрів.

№ кадру	Основна пам'ять	№ кадру	Основна пам'ять	№ кадру	Основна пам'ять
0	A.0	0	A.0	0	A.0
1	A.1	1	A.1	1	A.1
2	A.2	2	A.2	2	A.2
3	A.3	3	A.3	3	A.3
4	B.0	4	-	4	Д.0
5	B.1	5	-	5	Д.1
6	B.2	6	-	6	Д.2
7	C.0	7	C.0	7	C.0
8	C.1	8	C.1	8	C.1
9	C.2	9	C.2	9	C.2
10	C.3	10	C.3	10	C.3
11		11		11	Д.3
12		12		12	Д.4
13		13		13	
14		14		14	
а) Завантаження процесів A, B, C.		**б)** Вивантаження процесу B		**с)** Завантаження процесу Д	

Рисунок 3.3 – Завантаження процесів А, В, С, Д

Коли необхідно завантажити процес А (складається із чотирьох сторінок) ОС знаходить вільні кадри і завантажує туди сторінки процесу А.

Далі завантажується процес В (три сторінки) і процес С (чотири сторінки) (рисунок 3.3.-а).

За умовою процес Д має п'ять сторінок, тобто для завантаження йому не вистачає кадрів. Тому, процес В буде призупинено і вивантажено (рисунок 3.3.-б). Процеси блокуються і завантажується процес Д (рисунок 3.3.-с).

Як бачимо, для завантаження процесу при простій сторінковій організації пам'яті не обов'язковим є суміжне розташування кадрів, в які розміщуються сторінки одного процесу.

Для кожного процесу ОС підтримує таблицю сторінок, яка вказує розташування кадрів кожної сторінки процесу (рисунок 3.4). Також формується таблиця вільних кадрів.

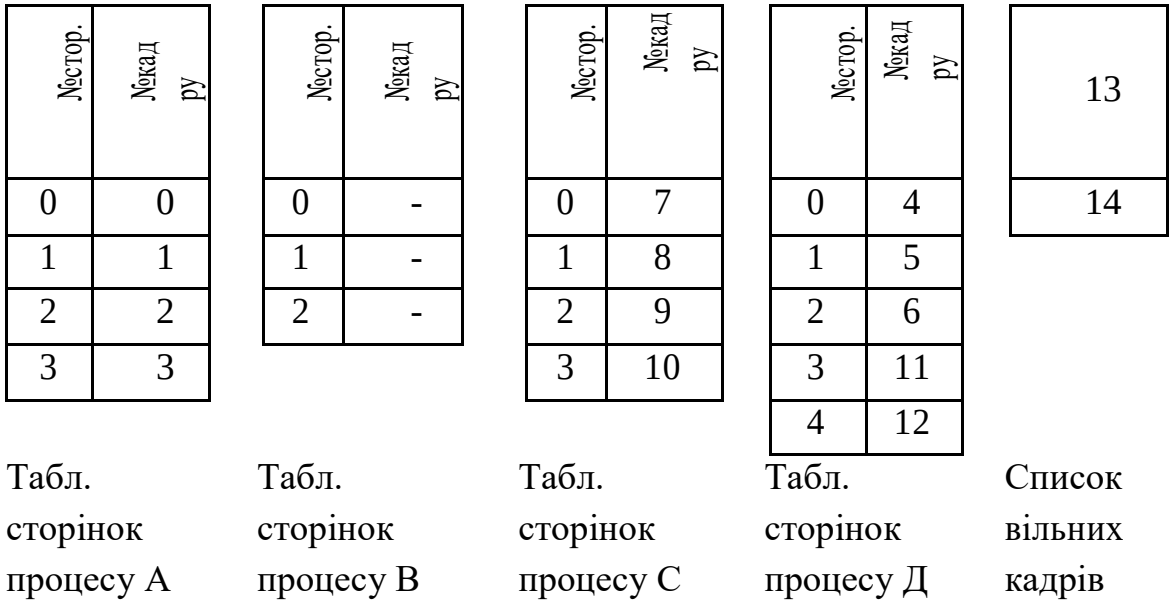

Рисунок 3.4 – Таблиці сторінок процесів

Таблиця сторінок містить по одному запису для кожної сторінки процесу. Це таблиця індексується номером сторінки починаючи з нуля, а кожний запис таблиці містить номер кадру в ОП. Операційна система підтримує єдиний список вільних кадрів.

програмі логічна адреса дорівнює номеру сторінки і зміщення всередині сторінки.

Фізична ж адреса дорівнює номеру кадру і зміщення.

Перетворення логічних адрес у фізичні – це завдання апаратного забезпечення.

На рисунку 3.5 наведено приклад трансляції логічної адреси з використанням простої сторінкової організації пам'яті для 16-бітових адрес.

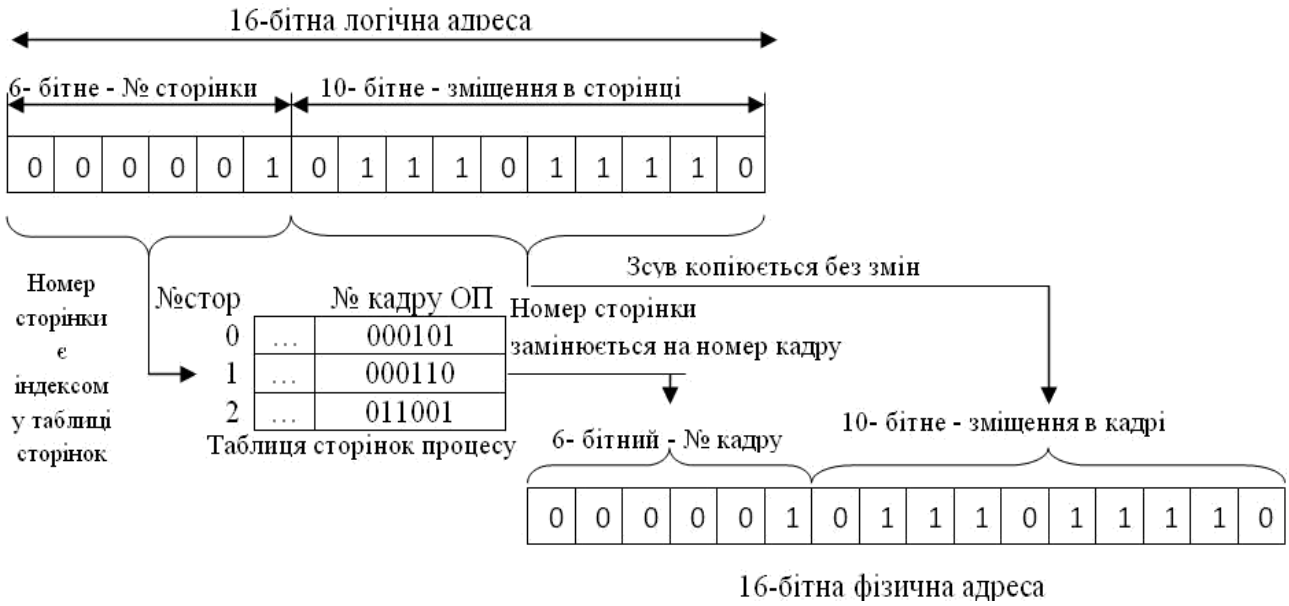

Рисунок 3.5 – Приклад трансляції логічних адрес в фізичну адресу при сторінковій організації пам'яті

Таким чином, проста сторінкова організація подібна фіксованому розподілу. Її відмінність полягає в малому розмірі розділів, які до того ж можуть бути несуміжними. Розмір сторінки повинен бути кратним ступеню числа 2.

3.1.5 Проста сегментна організація

Тут кожен процес поділяється на ряд сегментів і завантажується шляхом завантаження всіх його сегментів в динамічні (не обов'язково суміжні) розділи. Логічна адреса в цьому випадку дорівнює номеру сегмента і зміщення. При цьому, у порівнянні з динамічним розподілом, відсутня внутрішня фрагментація, краще використовується пам'ять і знижені накладні витрати.

Даний спосіб схожий на динамічний розподіл пам'яті. Якщо в ньому не використовуються оверлеї і віртуальна пам'ять, то для виконання програми всі сегменти повинні бути завантажені в ОП. Відмінність же від динамічного розподілу полягає в тому, що сегменти програми можуть займати кілька розділів, до того ж несуміжних.

Цей спосіб усуває внутрішню фрагментацію, але породжує зовнішню фрагментацію. Сегментація при його використанні прозора для програміста і застосовується при розміщенні коду і даних в різних сегментах для організації декількох незалежних адресних просторів. Як і при сторінковій організації, тут

використовується таблиця сегментів для кожного процесу і ведеться список вільних блоків ОП. Кожен запис таблиці сегментів при цьому повинен містити стартову адресу сегмента в ОП і його довжину.

Приклад трансляції адреси з використанням простої сегментної організації пам'яті для 16-бітових адрес показаний на рисунку 3. 6.

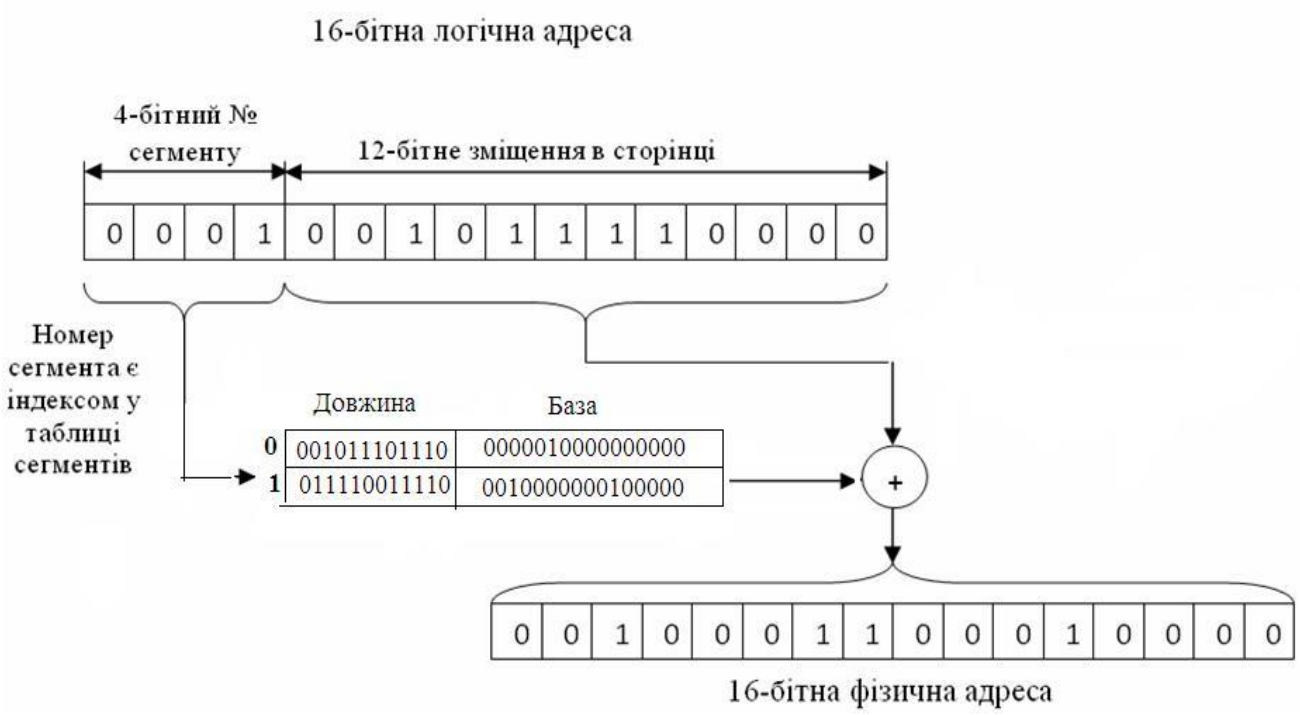

Рисунок 3.6 – Приклад трансляції фізичних адрес в логічну адресу при сегментації

3.2 Віртуальна пам'ять

3.2.1 Передумови для організації віртуальної пам'яті

Оскільки процес виконується тільки в основній пам'яті, ця пам'ять також називається *реальною*. Однак програміст або користувач мають справу з потенційно більшою за обсягом пам'яттю, виділеною на диску. Ця пам'ять називається ***віртуальною***[1].

Вперше сповіщення про віртуальну пам’ять на основі сторінкової організації з’явилося у 1962 році в роботі Kilburn I та ін. "*One-Level Storage System*" і незабаром після цього віртуальна пам’ять стала широко застосовуватись в комерційних системах.

Передумови для організації віртуальної пам'яті.

Всі звернення до пам'яті в рамках процесу являють собою логічні адреси, які динамічно транслюються в фізичні адреси під час виконання. Це означає, що процес може бути вивантажений на диск і знову завантажений в ОП, так що під час роботи він може перебувати в різних місцях ОП.

Процес може бути розбитий на кілька частин (сторінок або сегментів), які не обов'язково повинні розташовуватися в ОП єдиним безперервним блоком. Це забезпечується завдяки використанню динамічної трансляції адрес, таблиць сторінок або сегментів.

При виконанні цих умов одночасна наявність всіх сторінок або сегментів процесу в ОП не є необхідним.

Розглянемо, як це можна здійснити. Припустимо, що настав час завантаження нового процесу в пам'ять. ОС починає таке завантаження з розміщення в пам'яті тільки одного або декількох блоків (тут під блоком ми матимемо на увазі сторінку або сегмент), включаючи блок, що містить початок програми.

Частина процесу, що розташовується в деякий момент часу в ОП, називається *резидентною множиною процесу*. Далі, під час виконання процесу за допомогою таблиці сторінок або сегментів процесор завжди може визначити, чи розташовується блок, до якого потрібно звернення, в ОП. Якщо ні, то процесор генерує переривання, що свідчить про відсутність блоку в пам'яті; тоді ОС переводить перерваний процес в заблокований стан і завантажує в ОП блок, що містить адресу, через яку відбулося переривання.

Після того, як необхідний блок завантажений в ОП, виконується переривання введення/виведення, що передає керування операційній системі, яка, в свою чергу, переводить заблокований процес в стан готовності, і процес продовжує виконання з перерваної команди.

Переваги використання віртуальної пам'яті:

в оперативній пам'яті може підтримуватися більша кількість процесів;

процес може бути більше, ніж обсяг всієї фізичної пам'яті.

таблиці 3.1. порівняно переваги і недоліки всіх видів організації пам'яті.

Таблиця 3.1 – Порівняння різних технологій розподілу пам’яті

Технологія	Опис	Переваги	Недоліки
Фіксований розподіл	Основна пам'ять розділяється на ряд статичних розділів під час генерації системи. Процес може бути завантажений в розділ рівного або більшого розміру	Простота реалізації, малі системні накладні витрати	Неефективне використання пам'яті через внутрішню фрагментацію, фіксована максимальна кількість активних процесів
Динамічний розподіл	Розділи створюються динамічно, кожен процес завантажується в розділ строго необхідного розміру	Відсутня внутрішня фрагментація, більш ефективне використання основної пам'яті	Неефективне використання процесора через необхідність ущільнення для протидії зовнішній фрагментації
Проста сторінкова організація	Основна пам'ять розподілена на ряд кадрів рівного розміру. Кожен процес розподілений на деяку кількість сторінок рівного розміру тієї ж довжини, що і кадри. Процес завантажується шляхом завантаження всіх його сторінок в доступні, але не обов'язково послідовні, кадри	Відсутня зовнішня фрагментація	Є невелика внутрішня фрагментація
Проста сегментація	Кожний процес розподілено на ряд сегментів. Процес завантажується шляхом завантаження всіх своїх сегментів в динамічні (не обов’язково суміжні) розділи	Відсутня внутрішня фрагментація	Покращене використання пам’яті и знижені накладні витрати в порівнянні з динамічним розподіленням
Сторінкова організація віртуальної пам’яті	Все, як при простій сторінковій організації, з тим винятком, що не потрібно одночасно завантажувати всі сторінки процесу. Необхідні нерезиденті сторінки автоматично завантажуються в пам'ять	Немає зовнішньої фрагментації; більш високий ступінь багатозадачності; великий віртуальний адресний простір	Накладні витрати через складність системи управління пам'яттю

Технологія	Опис	Переваги	Недоліки
Сегментація віртуальної пам'яті	Все, як при простій сегментації, з тим винятком, що не потрібно одночасно завантажувати всі сегменти процесу. Необхідні нерезиденті сегменти автоматично завантажуються в пам'ять	Немає внутрішньої фрагментації; більш високий ступінь багатозадачності; великий віртуальний адресний простір; підтримка захисту і спільного використання	Накладні витрати через складність системи управління пам'яттю

Коли використовується віртуальна пам'ять, то віртуальні адреси не передаються напряму шиною пам'яті. Замість цього вони передаються диспетчеру пам'яті (*MMU – Memory Management* Unit), який відображує віртуальні адреси на фізичні адреси пам'яті (рисунок 3.7). Тут диспетчер пам'яті показаний як частина мікросхеми процесора.

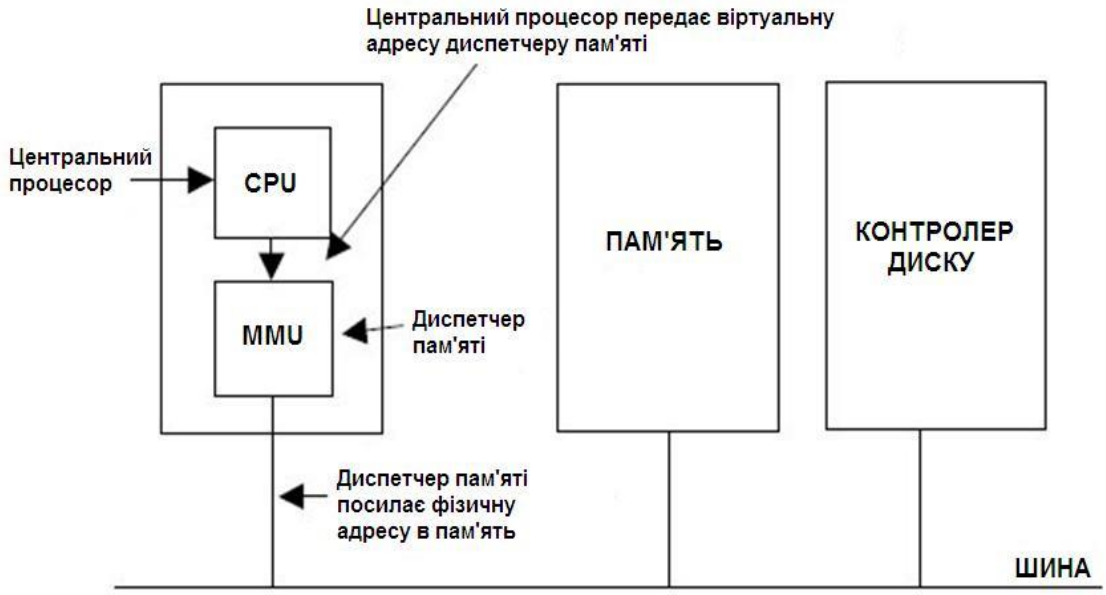

Рисунок 3.7 – Диспетчер пам'яті

теперішній час реалізація віртуальної пам'яті може бути організована різними способами, що відрізняються в основному способом структуризації віртуального адресного простору. Виділяють три методи реалізації віртуальної пам'яті:

1. Сторінкова віртуальна пам'ять – організує переміщення даних між основною пам'яттю і диском сторінками – частинами віртуального адресного простору фіксованого і порівняно невеликого розміру.

Сегментна віртуальна пам'ять – передбачає переміщення даних сегментами - частинами віртуального адресного простору довільного розміру.

Сегментно-сторінкова віртуальна пам'ять – використовує дворівневий поділ, при якому віртуальний адресний простір поділяється на сегменти, а потім сегменти поділяються на сторінки. Одиницею переміщення даних є сторінка.

Для тимчасового зберігання сегментів і сторінок на диску відводиться спеціальна область або спеціальний файл (сторінковий файл або файл підкачки – paging file). Поточний розмір сторінкового файлу є важливим параметром, що впливає на можливості ОС: чим більший сторінковий файл, тим більше додатків може одночасно виконувати ОС (при фіксованому розмірі оперативної пам'яті). Але при збільшенні числа додатків, що одночасно працюють збільшується і розмір сторінкового файлу, а це сповільнює їх роботу, так як значна частина часу витрачається на переміщення даних на диск і в зворотному напрямку.

Розмір сторінкового файлу в сучасних ОС є таким параметром, що налаштовується, і може бути визначений адміністратором системи для досягнення компромісу між рівнем програмування і швидкодією системи.

3.2.2 Сторінкова організація віртуальної пам'яті

Термін «віртуальна пам'ять» зазвичай асоціюється з системами, що використовують сторінкову організацію, хоча використовується і віртуальна пам'ять на основі сегментації.

Структура таблиці сторінок.

При розгляді простої сторінкової організації ми вказували, що кожен процес має свою таблицю сторінок, що створюється операційною системою. Кожен запис в цій таблиці сторінок містить номер кадру відповідної сторінки в пам'яті. Таблиця сторінок розташовується в ОП.

Для організації віртуальної пам'яті кожний запис таблиці сторінок обов'язково має містити *біт присутності P*, що вказує на присутність відповідної сторінки в ОП; якщо ця сторінка розташовується в ОП, то в записі таблиці також міститься номер її кадру.

Іншим керуючим бітом в записі таблиці сторінок є *біт модифікації M*. Він вказує, чи було змінено вміст даної сторінки з часу її останнього завантаження в ОП. Якщо таких змін не було, то при настанні часу заміни сторінки в займаному нею в даний момент кадрі записувати цю сторінку на диск не потрібно, так як на диску вже є її точна копія.

Базовий механізм читання слова з пам'яті включає в себе трансляцію віртуальної, або логічної адреси, що складається з номера сторінки і зміщення у фізичну адресу, яка являє собою номер кадру і зсув, з використанням таблиці сторінок.

принципі, кожна віртуальна адреса викликає звернення до двох фізичних адрес: одне звернення потрібно для вибірки відповідного запису з таблиці сторінок і ще одне - для власне звернення до даних, що адресуються. Отже, проста схема віртуальної пам'яті, по суті, подвоює час звернення до ОП. Для подолання цієї проблеми більшість схем віртуальної пам'яті використовує спеціальний високошвидкісний кеш для записів таблиці сторінок, який зазвичай називається буфером швидкого перетворення адреси, або просто буфером пошуку трансляції (TLB, Translation Lookaside Buffer). Цей кеш функціонує так само, як і звичайний кеш пам'яті.

3.2.3 Сегментація

Сегментація дозволяє програмісту розглядати пам'ять як область, що складається з безлічі адресних просторів, або сегментів, які можуть мати різні (фактично – динамічні) розміри. Звернення до пам'яті використовують адреси, що представляють собою пари: (номер сегмента, зміщення).

Така організація має ряд переваг у порівнянні з несегментованим адресним простором:

спрощується обробка зростаючих структур даних;

спрощується спільне використання коду і даних різними процесами;

покращується захист даних.

Як відомо, при простій сегментній організації кожен процес має власну таблицю сегментів, у кожному записі якої зазначена початкова адреса відповідного сегмента в ОП і його довжина. Та ж таблиця сегментів потрібна і при схемі віртуальної пам'яті, заснованої на сегментації, однак структура записів таблиці сегментів в цьому випадку ускладнюється.

Оскільки в ОП можуть знаходитися не всі сегменти процесу, в кожному записі потрібна наявність *біта присутності*, який вказує на те, чи розташовується даний сегмент в ОП. Якщо сегмент розташований в ОП, то запис також включає його початкову адресу і довжину.

Ще один біт, необхідний в даній схемі, – *біт модифікації*, який вказує, чи було змінено вміст сегменту з часу його останнього завантаження в ОП. Якщо змін не було, то при вивантаженні сегмента немає необхідності в його запису на диск.

Основний механізм читання слова з пам'яті включає в себе перетворення віртуальної, або логічної, адреси, що складається з номера сегмента і зміщення,

фізичну адресу з використанням таблиці сегментів, для зберігання якої використовується ОП. Коли запускається певний процес, в одному з регістрів зберігається стартова адреса його таблиці сегментів. Номер сегмента з віртуальною адресою використовується в якості індексу таблиці, що дозволяє визначити початкову адресу сегменту. Для отримання ж фізичної адреси до початкової адреси сегмента додається зсув з віртуальної адреси.

3.2.4 Комбінація сегментації і сторінкової організації

сторінкова організація, і сегментація мають свої переваги. Сторінкова організація, прозора для програміста, усуває зовнішню фрагментацію і, таким чином, забезпечує ефективне використання ОП. Так як переміщувані в ОП і з неї блоки мають фіксований, однаковий розмір, то полегшується створення ефективних алгоритмів управління пам'яттю. Сегментація ж, є видимою для програміста і має свої переваги.

Деякі обчислювальні системи використовують переваги обох методів.

таких системах адресний простір користувача розбивається на ряд сегментів на розсуд програміста. Кожен сегмент, у свою чергу, розбивається на ряд сторінок фіксованого розміру, відповідного до розміру кадру ОП. Якщо розмір сегмента менше за розмір сторінки, то він займає сторінку цілком. З точки зору програміста, логічна адреса в цьому випадку складається з номера сегмента і зміщення в ньому. З позиції ж ОС зсув в сегменті слід розглядати як номер сторінки певного сегмента і зміщення в ній.

3.2.5 Алгоритми управління віртуальною пам'яттю

Алгоритми (стратегії) операційної системи, пов'язані з управлінням віртуальною пам'яттю, діляться на наступні групи.

Стратегія вибірки. Стратегія вибірки визначає, коли сторінка повинна бути передана в ОП. При цьому можливі два основні варіанти: *на вимогу і попередньо*. При вибірці *за вимогою* сторінка передається в ОП тільки тоді, коли виконується звернення до комірки пам'яті, розташованої на цій сторінці.

разі ж *попередньої вибірки* завантажується не тільки сторінка, що викликала переривання звернення. Якщо сторінки процесу розташовані у вторинній пам'яті послідовно, то набагато більш ефективною буде завантаження в ОП декількох послідовних сторінок за один раз, ніж завантаження цих же сторінок по одній протягом певного проміжку часу.

Стратегія розміщення. Стратегія розміщення визначає, де саме у фізичній пам'яті будуть розташовуватися частини процесу. У разі «чистої» сегментації стратегія розміщення використовує алгоритми першого відповідного, наступного, що підходить та інші. Однак для систем, що використовують тільки сторінкову організацію або сторінкову організацію в поєднанні з сегментацією, стратегія розміщення зазвичай не так важлива, оскільки апаратна трансляція адреси і апаратне звернення до пам'яті однаково результативні при будь-яких поєднаннях «сторінка-кадр».

Стратегія заміщення. Стратегія заміщення відповідає за вибір сторінок

ОП для їх заміщення сторінками, що завантажуються з вторинної пам'яті. Ця стратегія включає в себе ряд взаємопов'язаних питань: 1) яка кількість кадрів має бути виділена кожному активному процесу; 2) чи має безліч сторінок, які потенційно можуть бути заміщені сторінками, що завантажуються обмежуватися одним процесом, або в якості кандидатів на заміщення можуть розглядатися всі кадри ОП; 3) які саме сторінки з розглянутої безлічі слід вибрати для заміщення.

Коли всі кадри ОП зайняті, а нам потрібно розмістити нову сторінку, стратегія заміщення визначає, яка з сторінок, що знаходяться в даний час в ОП повинна бути вивантажена, щоб звільнити кадр для сторінки, що завантажується. Всі варіанти стратегії при цьому спрямовані на те, щоб вивантажити сторінку, звернень до якої в найближчому майбутньому не буде.

відповідності з принципом локалізації, часто спостерігається сильна кореляція між сторінками, до яких останнім часом виконувалися звернення, і сторінками, до яких будуть звернення найближчим часом. Більшість стратегій намагаються визначити майбутню поведінку програми на основі її минулої поведінки. Однак при розгляді різних стратегій слід враховувати, що чим більш досконалий і інтелектуальний алгоритм використовує стратегія, тим вище будуть накладні витрати при його реалізації.

Оптимальний алгоритм полягає у виборі для заміщення тієї сторінки, звернення до якої буде виконуватися через найбільший проміжок часу в порівнянні з усіма іншими сторінками. Цей алгоритм призводить до мінімальної кількості переривань через відсутність сторінок. Зрозуміло, що реалізувати такий алгоритм на практиці неможливо, оскільки для цього системі потрібно знати всі майбутні події. Однак цей алгоритм є стандартом, з яким порівнюються реальні алгоритми.

Стратегія довше за всіх невикористовуваного елементу *(LRU, least Recently Used)* заміщає в пам'яті ту сторінку, звернень до якої не було довше, ніж до інших сторінок: згідно з принципом локалізації, можна очікувати, що ця сторінка не буде використовуватися і в найближчому майбутньому. Ця стратегія близька до оптимальної. Основна ж проблема полягає в складності її реалізації.

Стратегія «першим увійшов – першим вийшов» розглядає кадри сторінок процесу як циклічний буфер з циклічним же видаленням сторінок з нього. Це одна з найпростіших у реалізації стратегій заміщення, яка заміщає сторінку, що знаходиться в ОП довше інших. Однак далеко не завжди ця сторінка використовується дійсно рідко.

Годинникова стратегія є компромісом між стратегією довше за всіх невикористовуваного елемента (близькою до оптимальної, але складною в реалізації) та стратегією «першим увійшов – першим вийшов» (простою в реалізації, але далекою від оптимальної). У найпростішій схемі годинникової стратегії з кожним кадром зв'язується один додатковий біт, що носить назву *«біт використання».* Коли сторінка вперше завантажується в кадр, її біт використання встановлюється рівним 1. При наступних зверненнях до сторінки цей біт також встановлюється рівним 1. При роботі алгоритму заміщення безліч кадрів розглядається як циклічний буфер, з яким пов'язаний покажчик, і при заміщенні сторінки покажчик переміщається до наступного кадру в буфері.

Коли настає час заміщення сторінки, ОС сканує буфер для пошуку кадру, біт використання якого дорівнює 0, і всякий раз, коли в процесі цього пошуку зустрічається кадр з бітом використання, рівним 1, він скидається в 0. Для заміщення вибирається перший, що зустрінеться кадр з нульовим бітом використання. Якщо ж всі кадри мають біт використання, рівний 1, то покажчик робить повний круг і повертається до початкового положення, замінюючи сторінку в цьому кадрі.