Лекція №2.7. Керування оперативною пам’яттю

Головне завдання комп'ютерної системи – *виконувати програми*. Програми разом з даними, до яких вони мають доступ, в процесі виконання повинні (принаймні частково) знаходитися в оперативній пам'яті. Операційній системі доводиться вирішувати задачу розподілу пам'яті між процесами користувача і компонентами ОС. Ця діяльність називається ***керуванням пам'яттю***.

Під пам'яттю розумітимемо ресурс комп'ютера, призначений для зберігання програмного коду і даних. Пам'ять зображають як масив машинних слів або байтів з їхніми адресами. У фон-неймановській архітектурі комп'ютерних систем процесор вибирає інструкції і дані з пам'яті та може зберігати в ній результати виконання операцій.

Таким чином, пам'ять є найважливішим ресурсом, що вимагає ретельного керування.

Частина ОС, яка відповідає за керування пам'яттю, називається ***менеджером пам'яті***.

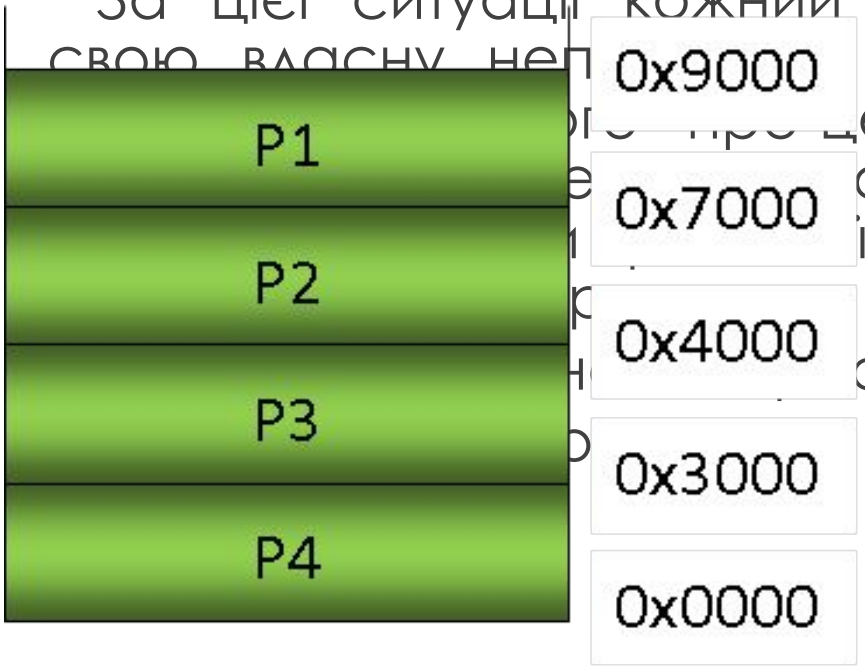
Різні види пам'яті організовані в ієрархію за *ступенем часу доступу*, *вартістю* і *об'єму*.

На нижніх рівнях такої ієрархії перебуває дешевша і повільніша пам'ять більшого обсягу, а в міру просування ієрархією нагору пам'ять стає дорожчою і швидшою (а її обсяг стає меншим). Найдешевшим і найповільнішим запам'ятовувальним пристроєм є жорсткий диск комп'ютера. Його називають також ***допоміжним запам'ятовувальним пристроєм***. Швидшою й дорожчою є оперативна пам'ять, що зберігається в мікросхемах пам'яті, встановлених на комп'ютері, — таку пам'ять називатимемо ***основною чи оперативною пам'яттю***. Ще швидшими засобами зберігання даних є різні кеші процесора, а обсяг цих кешів ще обмеженіший.

2.7.1. Основи технології віртуальної пам'яті

Спочатку розглянемо передумови введення концепції віртуальної пам'яті. Наведемо найпростіший з можливих способів спільного використання фізичної пам'яті кількома процесами (рис. 2.7.1).

За цієї ситуації кожний процес завантажують у свою власну нел ділянку фізичної пам'яті, ого процесу починається відразу е о. На рис. 2.7.1 праворуч ї пам'яті, починаючи з яких р стання о



Якщо проаналізувати особливості розподілу пам'яті на основі цього підходу, можуть виникнути такі запитання.

- *Як виконувати процеси, котрим потрібно більше фізичної пам'яті, ніж встановлено на комп'ютері?*
- *Що відбудеться, коли процес виконає операцію записування за невірною адресою (наприклад, процес P2 — за адресою 0x7500)?*
- *Що робити, коли процесу (наприклад, процесу P1) буде потрібна додаткова пам'ять під час його виконання?*
- *Коли процес отримає інформацію про конкретну адресу фізичної пам'яті, що з неї розпочнеться його виконання, і як мають бути перетворені адреси пам'яті, використані в його коді?*
- *Що робити, коли процесу не потрібна вся пам'ять, виділена для нього?*

Пряме завантаження процесів у фізичну пам'ять не дає змоги дати відповіді на ці запитання. Очевидно, що потрібні деякі засоби трансляції адрес пам'яті (зв'язування), які давали б змогу процесам використовувати набори адрес, котрі відрізняються від адрес фізичної пам'яті.

Програміст у своїй програмі звичайно не використовує адреси пам'яті безпосередньо, замість них вживаються символічні імена (функцій, глобальних змінних тощо). Внаслідок компіляції та компонування ці імена прив'язують до переміщуваних адрес (такі адреси задають у відносних одиницях, наприклад «100 байт від початку модуля»). Такі адреси ще називають ***переміщуваними адресами***. Під час виконання програми переміщувані адреси прив'язують до абсолютних адрес у пам'яті. По суті, кожна прив'язка — це відображення одного набору адрес на інший.

До адрес, використовуваних у програмах, ставляться такі вимоги.

Захист пам'яті. Помилки в адресації, що трапляються в коді процесу, повинні впливати тільки на виконання цього процесу. Коли процес Р2 зробить операцію записування за адресою 0x7500, то він і має бути перерваний за помилкою. Стратегія захисту пам'яті зводиться до того, що для кожного процесу зберігається діапазон коректних адрес, і кожна операція доступу до пам'яті перевіряється на належність адреси цьому діапазону.

Відсутність прив'язування до адрес фізичної пам'яті. Процес має можливості виконуватися незалежно від його місця в пам'яті та від розміру фізичної пам'яті. Адресний простір процесу виділяється як великий статичний набір адрес, при цьому кожна адреса та-кого набору є переміщуваною. Процесор і апарат-не забезпечення повинні мати змогу перетворювати такі адреси у фізичні адреси основної пам'яті (при цьому та сама переміщувана адреса в різний час або для різних процесів може відповідати різним фізичним адресам).

2.7.1.1. Типи адрес

Для ідентифікації команд програми і даних використовуються *адреси*.

Адреси поділяють на такі:

символьні імена. Привласнює програміст (наприклад, мітки);

віртуальні адреси. Формує транслятор. Початкова адреса дорівнює нулю;

фізичні адреси – номери елементів пам'яті, де насправді будуть розміщені команди та дані.

Сукупність віртуальних адрес ***складає віртуальний адресний простір (ВАП)***. Він визначається розрядністю комп'ютера. Для 32-розрядних комп' ютерів – це максимум FFFFFFFF, що становить **4 Гбайт**.

Типи подання віртуальних адрес:

- ***лінійне***, за якого адреса спочатку завжди дорівнює нулю, а адреса – ціле число;
- ***поділ на сегменти***, за якого адреса – це пара чисел (*n*,*m*), де *n* – номер сегмента, *m* – зсув.

Максимально можливий ВАП – визначається розрядністю процесора. Для 32-розрядного Intel Pentium ця величина становить 4 Гбайт.

Призначений ВАП дійсно необхідний процесу для роботи. Його називають також ***образом процесу***. Призначений ВАП може перевищувати фізичну ємність пам'яті. На цьому заснований механізм віртуальної пам'яті.

2.7.1.2. Алгоритми розподілення пам'яті без використання зовнішньої пам'яті

- розподілення пам'яті на фіксовані розділи;
- розподілення пам'яті на динамічні розділи.

У разі використання цього алгоритму пам'ять у початковий момент часу вважається вільною (за винятком пам'яті, відведеної для ОС). Кожному процесу відводиться вся необхідна пам'ять. Якщо її не вистачає, то процес не створюється. У довільний момент часу пам'ять є випадковою послідовністю зайнятих і вільних ділянок.

2.7.1.3. Алгоритми розподілу пам'яті з використанням зовнішньої пам'яті

Для повного завантаження процесора можуть знадобитися інколи сотні інтерактивних завдань. Всі вони мають бути розміщені в пам'яті, велика частина яких перебуває в стані очікування. Логічно було б на час очікування, в разі браку фізичної пам'яті, витісняти їх на диск, а коли необхідно, повертати в пам'ять. Така підміна (віртуалізація) оперативної пам'яті дисковою пам'яттю істотно підвищує рівень мультипрограмування. Важливо, що всі дії з переміщення відбуваються автоматично, без участі програміста.

Для віртуалізації застосовують два основні підходи:

- ***свопінг*** – образ процесу вивантажується на диск і повертається в пам'ять цілком (часто називають підкачуванням);
- ***віртуальна пам'ять*** – образ процесу вивантажується на диск і повертається в пам'ять частинами (сегментами, сторінками...).

2.7.1.4. Поняття віртуальної пам'яті

Віртуальна пам'ять — це технологія, в якій вводиться рівень додаткових перетворень між адресами пам'яті, які використовує процес, і адресами фізичної пам'яті комп'ютера.

Такі перетворення мають забезпечувати захист пам'яті та відсутність прив'язання процесу до адрес фізичної пам'яті.

Завдяки віртуальній пам'яті фізична пам'ять адресного простору процесу може бути фрагментованою, оскільки основний обсяг пам'яті, яку займає процес, більшу частину часу залишається вільним. Адреси можна переміщати так, щоб основній пам'яті відповідали тільки ті розділи адресного простору процесу, які справді використовуються у конкретний момент.

При цьому не використовувані розділи адресного простору можна ставити у відповідність повільнішій пам'яті, наприклад простору на жорсткому диску, а в цей час інші процеси можуть використовувати основну пам'ять, у яку раніше відображалися адреси цих розділів. Коли ж розділ знадобиться, його дані завантажують з диска в основну пам'ять, можливо, замість розділів, які стали непотрібними в конкретний момент (і які, своєю чергою, тепер збережуться на диску). Дані можуть зчитуватися з диска в основну пам'ять під час звертання до них.

У такий спосіб можна значно збільшити розмір адресного простору процесу і забезпечити виконання процесів, що за розміром перевищують основну пам'ять.

2.7.1.5. Проблеми реалізації віртуальної пам'яті. Фрагментація пам'яті

Основна проблема, що виникає у разі використання віртуальної пам'яті, стосується ефективності її реалізації. Оскільки перетворення адрес необхідно робити під час кожного звертання до пам'яті, недбала реалізація цього перетворення може призвести до найгірших наслідків для продуктивності всієї системи. Якщо для більшості звертань до пам'яті система буде змушена насправді звертатися до диска (який у десятки тисяч разів повільніший, ніж основна пам'ять), працювати із такою системою стане практично неможливо.

Ще однією проблемою є фрагментація пам'яті, що виникає за ситуації, коли неможливо використати вільну пам'ять. Розрізняють зовнішню і внутрішню фрагментацію пам'яті (рис. 2.7.2).

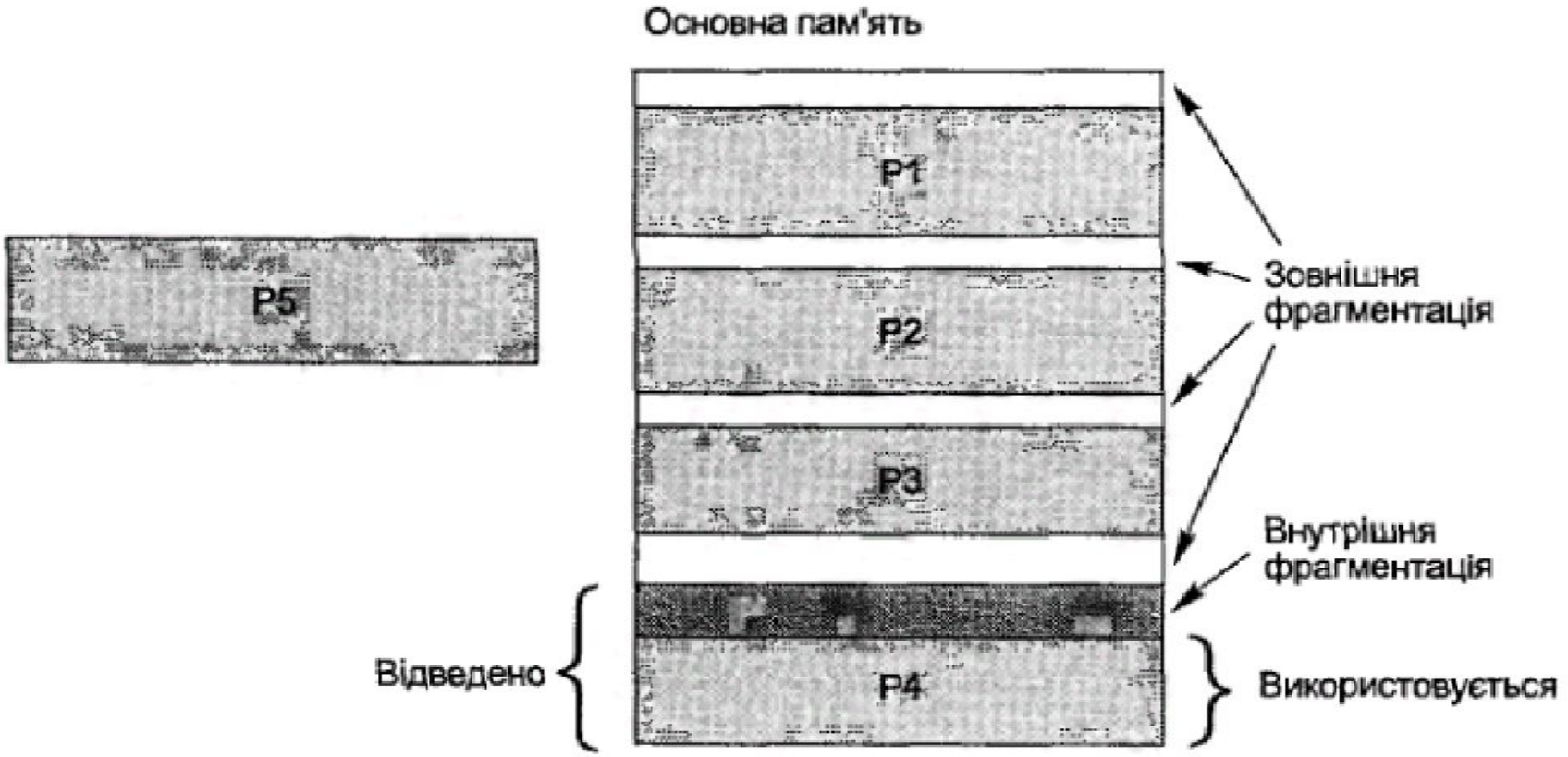


Рис. 2.7.2. Зовнішня і внутрішня фрагментація

Зовнішня фрагментація зводиться до того, що внаслідок виділення і наступного звільнення пам'яті в ній утворюються вільні блоки малого розміру — ***діри*** (holes). Через це може виникнути ситуація, за якої неможливо виділити неперервний блок пам'яті розміру N, оскільки немає жодного неперервного вільного блоку, розмір якого S>N, хоча загалом обсяг вільного простору пам'яті перевищує N.

Внутрішня фрагментація зводиться до того, що за запитом виділяють блоки пам'яті більшого розміру, ніж насправді будуть використовуватися, у результаті всередині виділених блоків залишаються невикористовувані ділянки, які вже не можуть бути призначені для чогось іншого.

2.7.1.6. Логічна і фізична адресація пам'яті

Найважливішими поняттями концепції віртуальної пам'яті є ***логічна*** і ***фізична адресація пам'яті***.

Логічна або віртуальна адреса - адреса, яку генерує програма, запущена на деякому процесорі. Адреси, що використовують інструкції конкретного процесора, є логічними адресами. Сукупність логічних адрес становить логічний адресний простір.

Фізична адреса — адреса, якою оперує мікросхема пам'яті. Прикладна програма в сучасних комп'ютерах ніколи не має справи з фізичними адресами. Спеціальний апаратний пристрій MMU (memory management unit — пристрій керування пам'яттю) відповідає за перетворення логічних адрес у фізичні.

Сукупність усіх доступних фізичних адрес становить ***фізичний адресний простір***.

Отже, якщо в комп'ютері є мікросхеми на 256 Мбайт пам'яті, то саме такий обсяг пам'яті адресують фізично. Логічно зазвичай адресують значно більше пам'яті. Найпростіша схема перетворення адрес зображена на рис. 2.7.3.

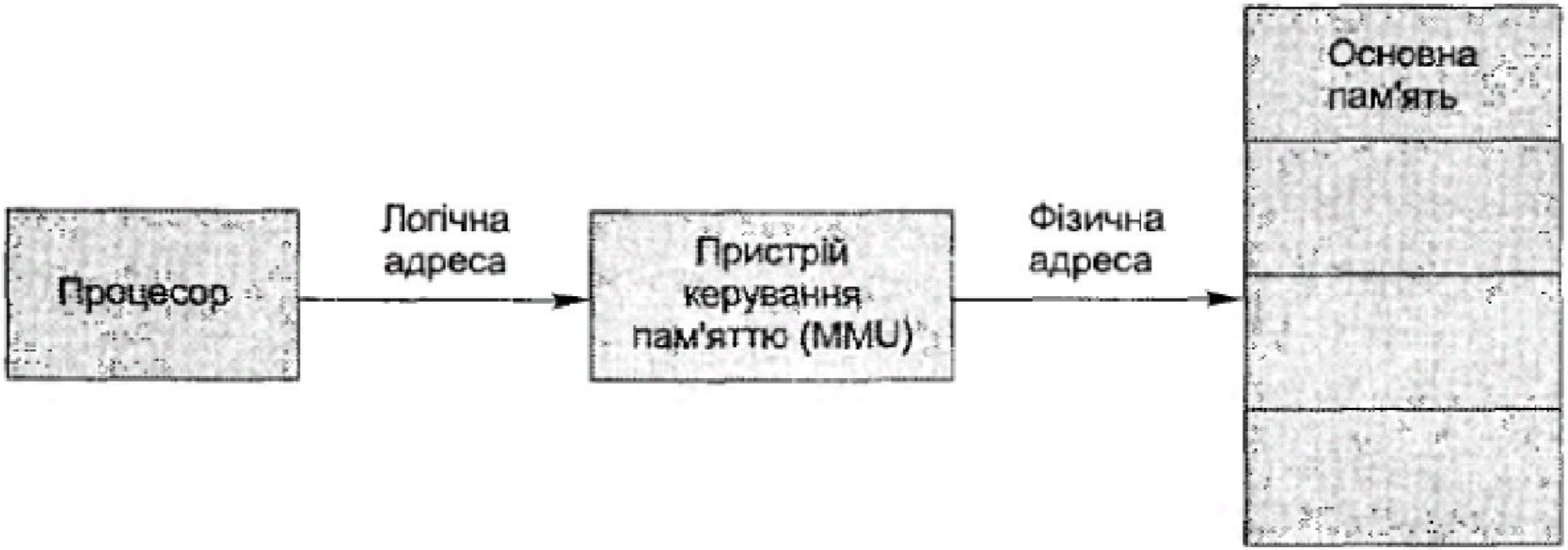


Рис. 2.7.3. Перетворення логічних адрес у фізичні

Специфіку перетворення логічних адрес у фізичні визна-чають різні підходи до керування оперативною пам'яттю, вивчення яких буде основною темою цього розділу.

2.7.1.7. Підхід базового і межового регістрів

Під час реалізації віртуальної пам'яті необхідно забезпе-чити захист пам'яті, переміщення процесів у пам'яті та спільне використання пам'яті кількома процесами.

Одним із найпростіших способів задовольнити ці вимоги є підхід базового і межового регістрів. Для ко-жного процесу в двох регістрах процесора зберіга-ють два значення — **базової адреси (base)** і **межі (bo-unds)**. Кожний доступ до логічної адреси апаратно пе-ретворюється у фізичну адресу шляхом додавання логічної адреси до базової. Якщо отримувана фізична адреса не потрапляє в діапазон **(base, base+bounds)**, вважають, що адреса невірна, і

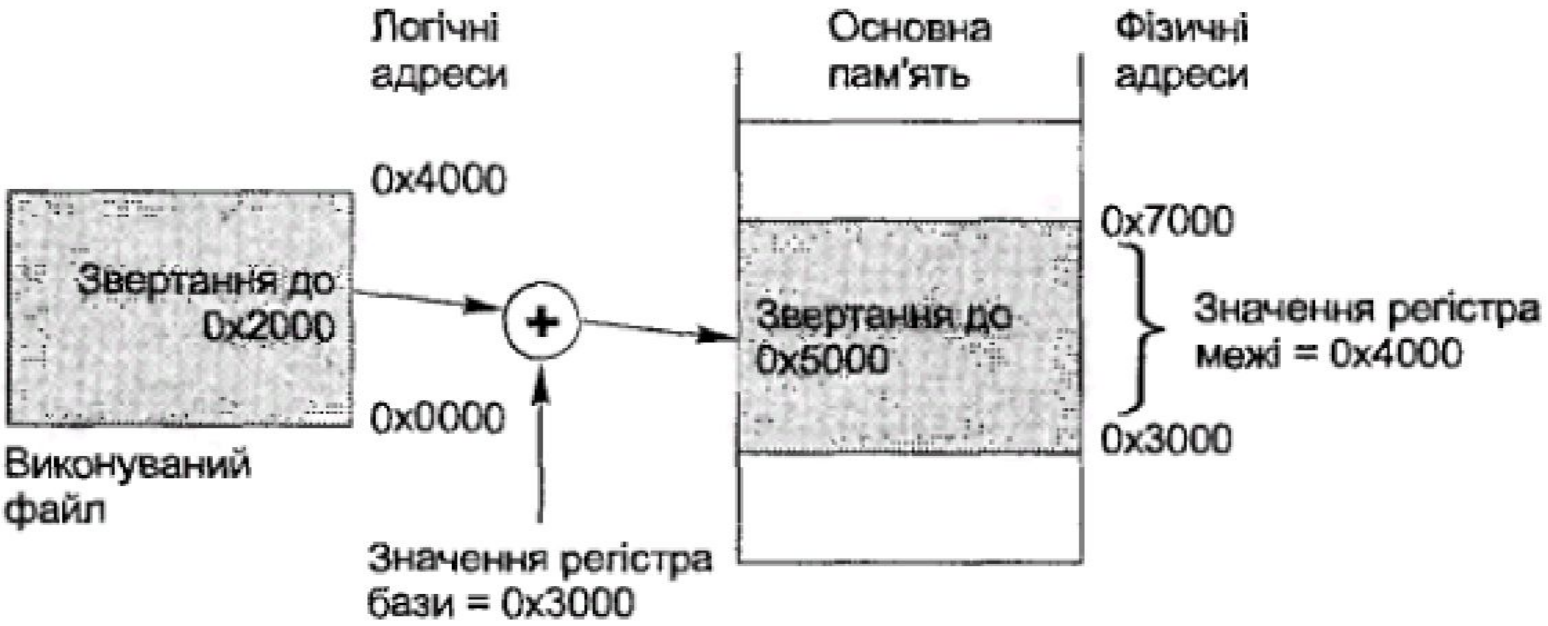


Рис. 2.7.4. Використання базового і межового регістрів

Такий підхід є найпростішим прикладом реалізації дина-мічного переміщення процесів у пам'яті. Усі інші підходи, які буде розглянуто в цьому розділі, є різними варіантами розвитку цієї базової схеми. Наприклад, те, що кожний процес у разі використання цього підходу має свої власні значення базового і межового регістрів, є найпростішою реалізацією концепції адресного простору процесу, яка ґрунтується на тому, що кожний процес має власне відображення пам'яті.

Для організації захисту пам'яті в цій ситуації необхідно, щоб застосування користувача не могли змінювати значення базового і межового регістрів. Для цього достатньо інструкції такої зміни зробити доступними тільки у привілейованому режимі процесора.

До переваг цього підходу належать простота, скромні вимоги до апаратного забезпечення (потрібні тільки два регістри), висока ефективність. Однак сьогодні його практично не використовують через низку недоліків, пов'язаних насамперед з тим, що адресний простір процесу все одно відображається на один неперервний блок фізичної пам'яті: незрозуміло, як динамічно розширювати адресний простір процесу; різні процеси не можуть спільно використовувати пам'ять; немає розподілу коду і даних.

За такого підходу для процесу виділяють тільки одну пару значень «базова адреса-межа». Природним розвитком цієї ідеї стало відображення адресного простору процесу за допомогою кількох діапазонів адрес фізичної пам'яті, кожен з яких задають власною парою значень базової адреси і межі. Так виникла концепція *сегментації пам'яті*.

2.7.2. Сегментація пам'яті

2.7.2.1. Особливості сегментації пам'яті

Сегментація пам'яті дає змогу зображати логічний адресний простір як сукупність незалежних блоків змінної довжини, які називають ***сегментами***. Кожний сегмент звичайно містить дані одного призначення, наприклад в одному може бути стек, в іншому — програмний код і т. д. У кожного сегмента є ім'я і довжина (для зручності реалізації поряд з іменами використовують номери).

Логічна адреса складається з номера сегмента і зсуву всередині сегмента; з такими адресами працює прикладна програма. Компілятори часто створюють окремі сегменти для різних даних програми (сегмент коду, сегмент даних, сегмент стека).

Під час завантаження програми у пам'ять створюють таблицю дескрипторів сегментів процесу, кожний елемент якої відповідає одному сегменту і складається із базової адреси, значення межі та прав доступу.

Під час формування адреси її сегментна частина вказує на відповідний елемент таблиці дескрипторів сегментів процесу. Якщо зсув більший, ніж задане значення межі (або якщо права доступу процесу не відповідають правам, заданим для сегмента), то апаратне забезпечення генерує помилку. Коли ж усе гаразд, сума бази і зсуву в разі чистої сегментації дасть у результаті фізичну адресу в основній пам'яті. Якщо сегмент вивантажений на диск, спроба доступу до нього спричиняє його завантаження з диска в основну пам'ять. У підсумку кожному сегменту відповідає неперервний блок пам'яті такої самої довжини, що перебуває в довільному місці фізичної пам'яті або на диску. Загальний підхід до перетворення адреси у разі сегментації показаний на рис. 2.7.5.

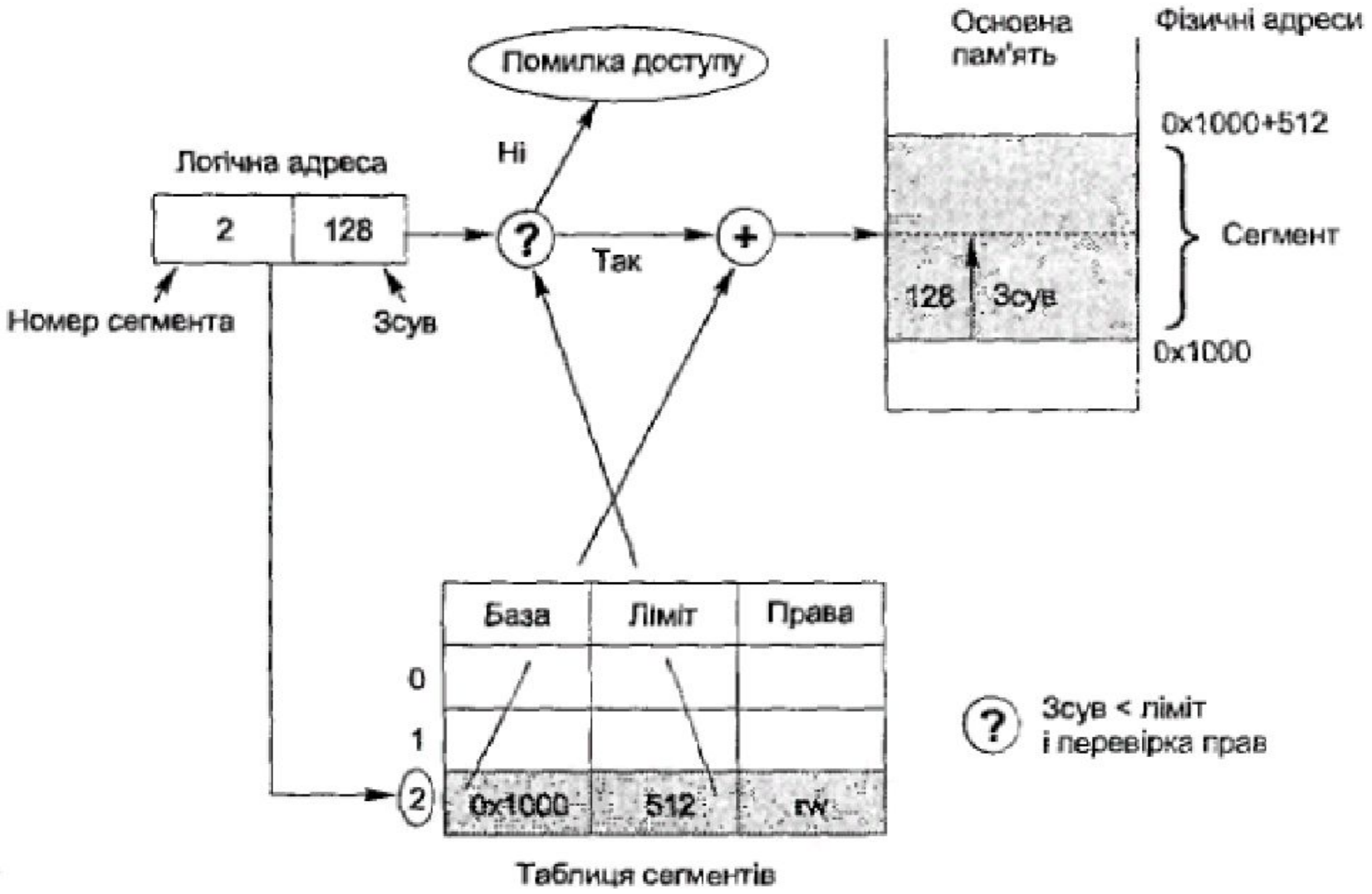


Рис. 2.7.5. Перетворення адреси в разі сегментації

Загальний вигляд пам'яті у разі сегментації показано на рис. 2.7.6.

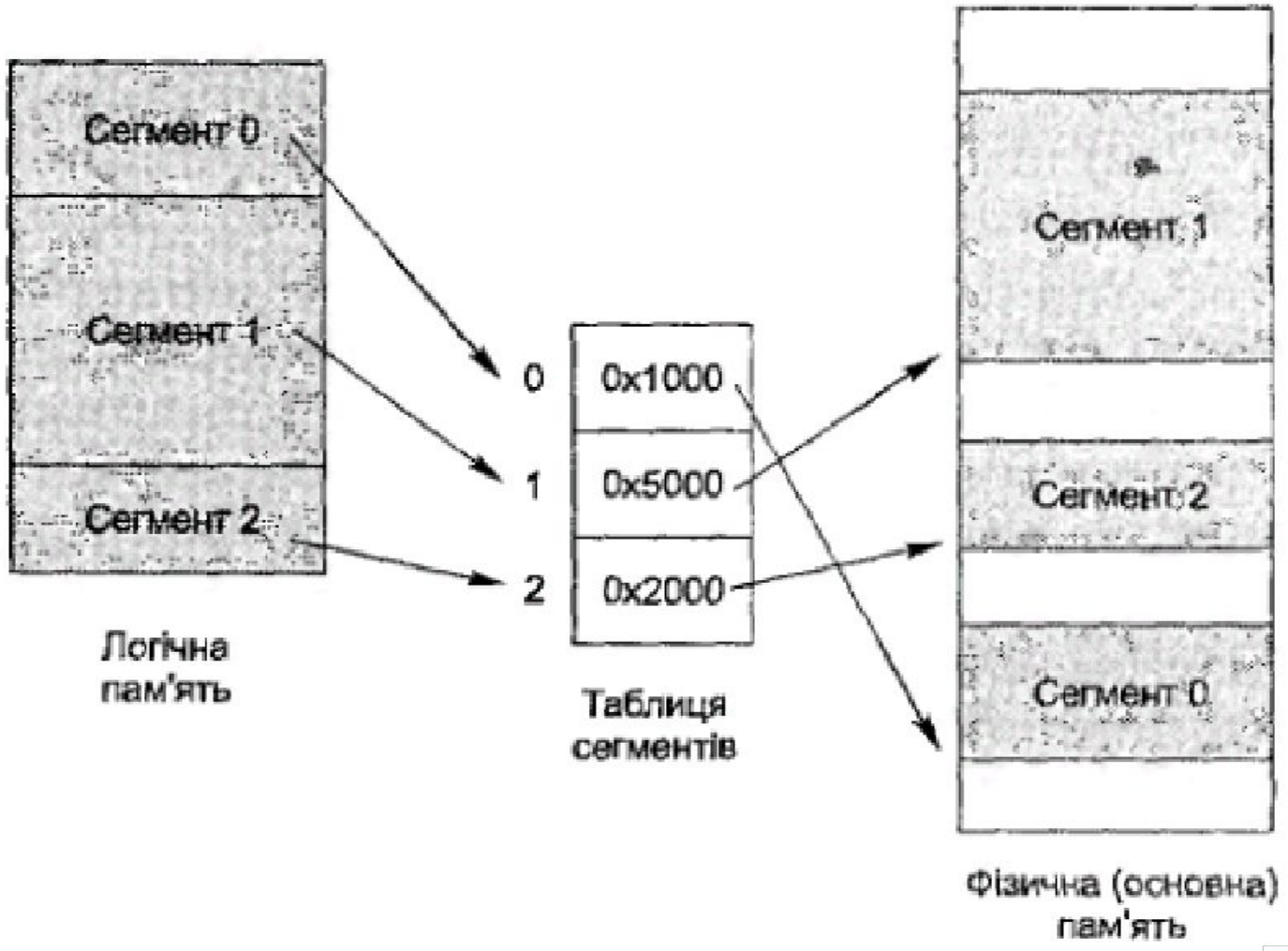


Рис. 2.7.6. Логічний і фізичний адресний простір у разі сегментації

Наведемо переваги сегментації пам'яті.

1. З'явилася можливість організувати кілька незалежних сегментів пам'яті для процесу і використати їх для зберігання даних різної природи. При цьому права доступу до кожного такого сегмента можуть бути задані по-різному.

2. Окремі сегменти можуть спільно використовуватися різними процесами, для цього їхні таблиці дескрипторів сегментів повинні містити однакові елементи, що описують такий сегмент.

3. Фізична пам'ять, що відповідає адресному простору процесу, тепер не обов'язково має бути неперервною. Справді, сегментація дає змогу окремим частинам адресного простору процесу відображатися не в основну пам'ять, а на диск, і довантажуватися з нього за потребою, забезпечуючи виконання процесів будь-якого розміру.

Недоліки сегментації пам'яті.

1. Необхідність введення додаткового рівня перетворення пам'яті спричиняє зниження продуктивності (цей недолік властивий будь-якій повноцінній реалізації віртуальної пам'яті). Для ефективної реалізації сегментації потрібна відповідна апаратна підтримка.

2. Керування блоками пам'яті змінної довжини з урахуванням необхідності їхнього збереження на диску може бути досить складним.

3. Вимога, щоб кожному сегменту відповідав неперервний блок фізичної пам'яті відповідного розміру, спричиняє зовнішню фрагментацію пам'яті. Внутрішньої фрагментації у цьому разі не виникає, оскільки сегменти мають змінну довжину і завжди можна виділити сегмент довжини, необхідної для виконання програми.

Сьогодні сегментацію застосовують доволі обмежено передусім через фрагментацію і складність реалізації ефективного звільнення пам'яті та обміну із диском.

Реалізацію віртуальної пам’яті, подано трьома класами: ***сторінковим***, ***сегментним***, ***сегментно-сторінковим розподілами***. Ширше використання отримав розподіл пам'яті на блоки фіксованої довжини — сторінкова організація пам'яті.

Сторінковий розподіл. За сторінкового розподілу віртуальна пам’ять ділиться на частини однакового та фіксованого для системи розміру – віртуальні сторінки. Вся оперативна пам’ять також ділиться на частини такого ж розміру – фізичні сторінки. Розмір сторінки вибирається рівним мірі двійки: 512, 1024, 4096 і так далі.

Адреса сторінки входить в контекст процесу.

Таблиця сторінок складається з дескрипторів. Кожен ***дескриптор*** включає:

- номер фізичної таблиці;
- ознака наявності в оперативній пам'яті (формується апаратно);
- ознака модифікації (формується апаратно);
- ознака звернення (формується апаратно).

Віртуальна адреса, подана парою (P, SV), перетвориться в (N, SF).

Обсяг сторінки дорівнює мірі 2k, тоді зсув (S) можна отримати відділенням k розрядів.

Наприклад, якщо розмір сторінки становить 1 кбайт (210), то 50718 = 101 000 111 0012, 108 = 28 – номер сторінки.

Апаратно, з регістра витягується адреса таблиці сторінок. На підставі номера сторінки P і довжини запису L визначається адреса дескриптора (A = AT + P·L).

З таблиці витягується номер фізичної сторінки N. До номера N приєднується зсув S.

Розмір сторінок, (часто 4096) впливає на розмір таблиць, а це, у свою чергу, позначається на продуктивності. Для усунення цього недоліку ВАП може ділиться на розділи, а в кожному розділі формується своя таблиця сторінок. Цей варіант пришвидшує пошук.

Сегментний розподіл. За сторінкового розподілу ВАП ділиться на рівні частини механічно без урахування важливого значення даних. На одній сторінці можуть одночасно виявитися код програми і вихідні дані. Такий підхід не дозволяє забезпечити роздільне оброблення, наприклад захист, спільний доступ і т. ін.

Розбиття адресного простору на «осмислені» частини усуває ці недоліки і називається сегментним розподілом. Приклади сегментів: код програми, масив вихідних даних та ін.

На етапі створення процесу ОС будує таблицю сегментів процесу, аналогічну таблиці сторінок.

Недоліки сегментного розподілу:

- використання операції складання під час формування фізичної адреси призводить до зниження продуктивності;
- надмірність. Оскільки сегмент у загальному випадку може бути більшим ніж сторінка, то і одиниця обміну між оперативною пам'яттю і диском більша, що приводить до уповільнення роботи.

Сегментно-сторінковий розподіл. Цей метод є комбінацією сторінкового і сегментного механізмів керування пам'яттю і спрямований на реалізацію переваг обох підходів. Віртуальна пам'ять ділиться на сегменти, а кожен сегмент – на сторінки. Усі сучасні ОС використовують саме такий спосіб організації.

Дякую за увагу!!!