ЛЕКЦІЯ 6. КЕРУВАННЯ ОПЕРАТИВНОЮ ПАМ'ЯТТЮ

- Означення віртуальної пам'яті
- Принципи адресації пам'яті
- Сегментація пам'яті
- Сторінкова організація пам'яті
- Сторінково-сегментна організація пам'яті
- Керування оперативною пам'яттю у Windows

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.

Різні види пам'яті організовані в ієрархію. 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 (secondary storage). Швидшою й дорожчою є оперативна пам'ять, що зберігається в мікросхемах пам'яті, встановлених на комп'ютері. Таку пам'ять називатимемо *основною пам'яттю* (main memory). Ще швидшими засобами зберігання даних є різні кеші процесора, а обсяг цих кешів ще обмеженіший.

Керування пам'яттю в ОС – досить складне завдання. Потрібної за характеристиками пам'яті часто виявляють недостатньо, і щоб це не заважало роботі користувача, необхідно реалізовувати засоби координації різних видів пам'яті. Так, сучасні застосування можуть не вміщатися цілком в основній пам'яті, тоді невикористовуваний код застосування може тимчасово зберігатися на жорсткому диску.

6.1. 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. 6.1).

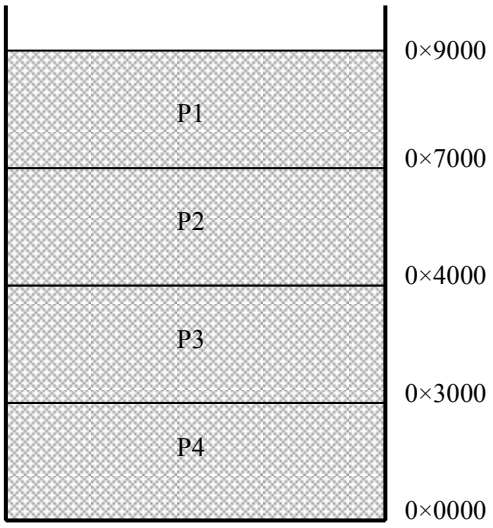


Рис. 6.1. Спільне використання фізичної пам’яті процесами

За цієї ситуації кожний процес завантажують у свою власну неперервну ділянку фізичної пам'яті, ділянка наступного процесу починається відразу після ділянки попереднього. На рис. 6.1 праворуч позначені адреси фізичної пам'яті, починаючи з яких

завантажуються процеси.

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 пам'яті, які давали б змогу процесам використовувати набори адрес, котрі відрізняються від адрес фізичної пам'яті. Перш ніж розібратися в особливостях цих адрес, коротко зупинимося на особливостях компонування і завантаження програм.

Програма зазвичай перебуває на диску у вигляді двійкового виконуваного файла, отриманого після компіляції та компонування. Для свого виконання вона має бути завантажена у пам'ять (адресний простір процесу). Сучасні архітектури дають змогу процесам розташовуватися у будь-якому місці фізичної пам'яті, при цьому одна й та сама програма може відповідати різним процесам, завантаженим у різні ділянки пам'яті. Заздалегідь невідомо, в яку ділянку пам'яті буде завантажена програма.

Під час виконання процес звертається до різних адрес, зокрема в разі виклику функції використовують її адресу (це адреса коду), а звертання до глобальної змінної відбувається за адресою пам'яті, призначеною для зберігання значення цієї змінної (це адреса даних).

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»). Під час виконання програми переміщувані адреси, своєю чергою, 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 P2 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 приналежність адреси цьому діапазону.
- *Відсутність прив'язання 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 такого набору є переміщуваною. Процесор і апаратне забезпечення повинні мати змогу перетворювати такі адреси у фізичні адреси основної пам'яті.

6.1.1. Поняття віртуальної пам'яті

Віртуальна пам'ять – це технологія, в якій вводиться рівень додаткових перетворень між адресами пам'яті, використовуваних процесом, 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. Є так зване правило «дев'яносто до десяти», або правило локалізації, яке стверджує, що 90 % звертань до пам'яті у процесі припадає на 10 % його адресного простору. Адреси можна переміщати так, щоб основній пам'яті відповідали тільки ті розділи адресного простору процесу, які справді використовуються у конкретний момент.

При цьому невикористовувані 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.

6.1.2. 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. 6.2).

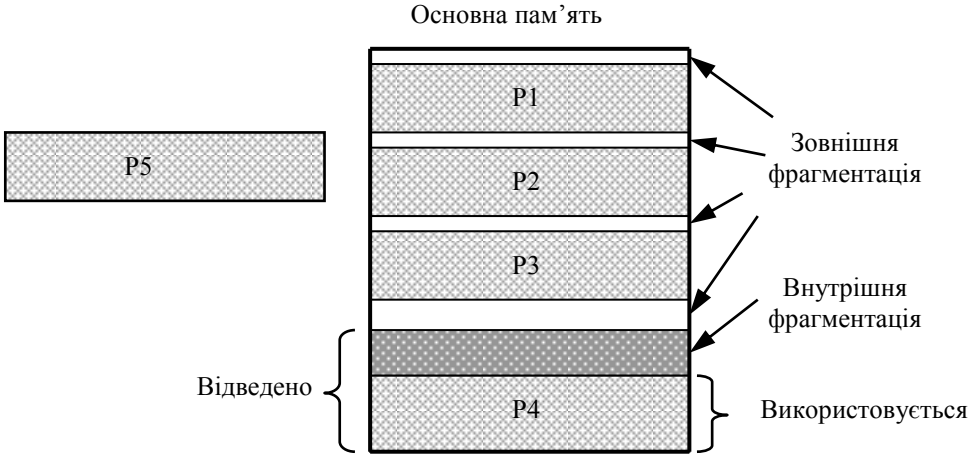


Рис. 6.2. Зовнішня і внутрішня фрагментація

Зовнішня 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 $5 > N$, хоча загалом обсяг

вільного простору пам'яті перевищує *N*. Так, на рис. 6.2 для виконання процесу P5 місця через зовнішню фрагментацію не вистачає.

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.

6.1.3. 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 128 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. 6.3.

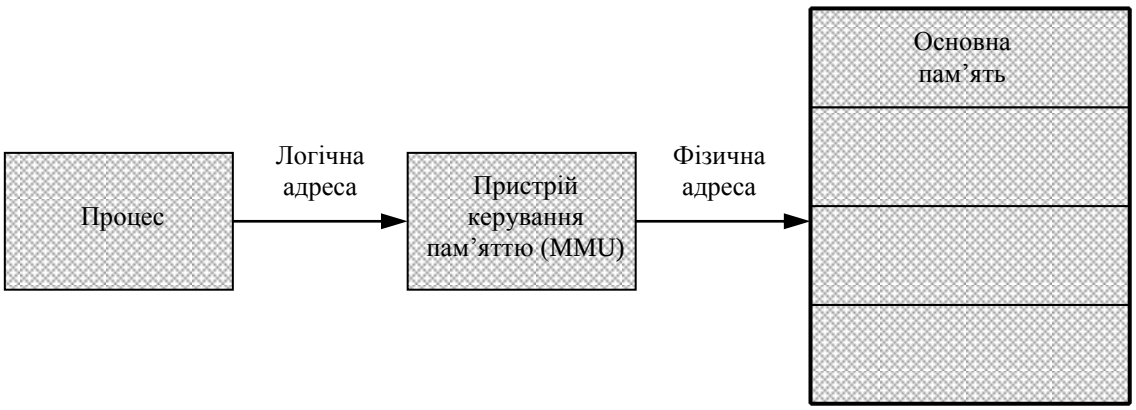


Рис. 6.3. Перетворення логічних адрес пам’яті у фізичні адреси

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

6.1.4. Підхід базового і межового регістрів

Під час реалізації віртуальної пам'яті необхідно забезпечити 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 кожного процесу в двох регістрах процесора зберігають два значення – *базової адреси* (base) і *межі* (bounds). Кожний доступ до логічної адреси апаратно перетворюється 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 генерують помилку (рис. 8.4).

Такий підхід є найпростішим прикладом реалізації динамічного 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. 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.

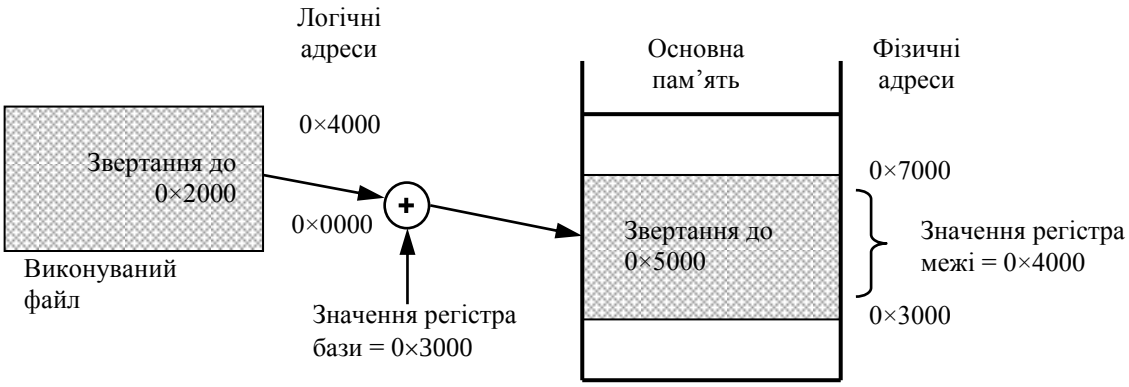


Рис. 6.4. Використання базового і межового регістрів

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 фізичної пам'яті, кожен з яких задають власною парою значень базової адреси і межі. Так виникла концепція сегментації пам'яті.

6.2. Сегментація пам'яті

6.2.1. 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. 6.5

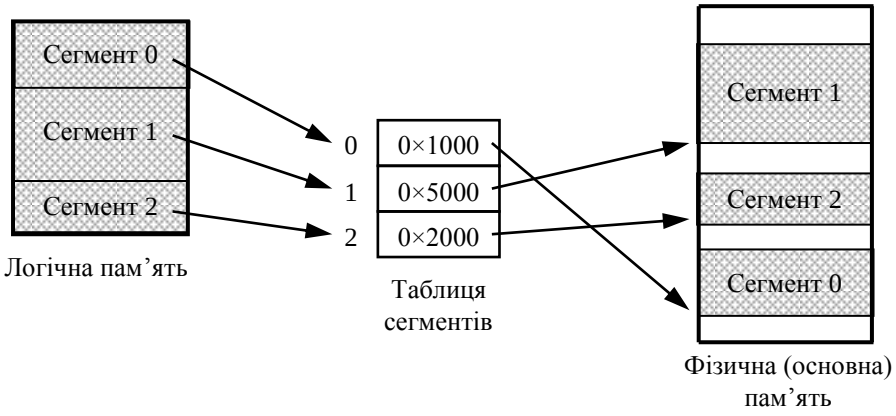


Рис. 6.5. Перетворення адреси у разі сегментації

Загальний вигляд пам'яті у разі сегментації показано на рис. 6.6

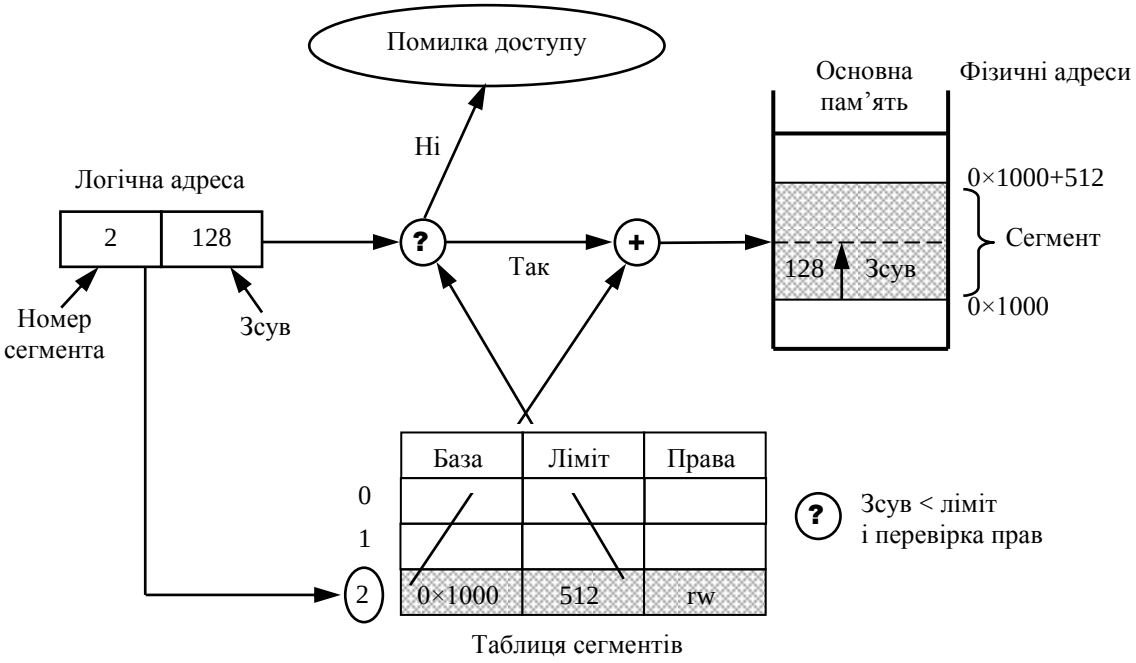


Рис. 6.6. Логічний і фізичний адресний простір у разі сегментації

Наведемо переваги сегментації пам'яті.

- 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.
- Окремі сегменти можуть спільно використовуватися різними процесами, для цього їхні таблиці дескрипторів сегментів повинні містити однакові елементи, що описують такий сегмент.
- 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. Цей підхід не позбавлений і недоліків.
- 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.

- Керування блоками пам'яті змінної довжини з урахуванням необхідності їхнього збереження на диску може бути досить складним.
- 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. Ширше використання отримав розподіл пам'яті на блоки фіксованої довжини – *сторінкова організація пам'яті,* яку розглянемо в подальших лекціях.

6.2.2. Реалізація сегментації в архітектурі IA-32

В архітектурі IA-32 логічні адреси в програмі формуються із використанням сегментації й мають такий вигляд: «селектор-зсув». Значення селектора завантажують у спеціальний регістр процесора (сегментний регістр) і використовують як індекс у таблиці дескрипторів сегментів, що перебуває в пам'яті та є аналогом таблиці сегментів, описаної раніше. В архітектурі IA-32 підтримуються шість сегментних регістрів. Це означає, що виконуваний код в один і той самий час може адресувати шість незалежних сегментів.

Селектор містить індекс дескриптора в таблиці, біт індикатора локальної або глобальної таблиці та необхідний рівень привілеїв.

Для системи задають спільну *глобальну таблицю дескрипторів* (Global Descriptor Table, GDT), а для кожної задачі – *локальну таблицю дескрипторів* (Local Descriptor Table, LOT).

Дескриптори в IA-32 мають довжину 64 біти. Вони визначають властивості програмних об'єктів (наприклад, сегментів пам'яті або таблиць дескрипторів).

Дескриптор містить значення бази (base), яке відповідає адресі об'єкта (наприклад, початок сегмента); значення межі (limit); тип об'єкта (сегмент, таблиця дескрипторів тощо); характеристики захисту.

Звертання до таблиць дескрипторів підтримується апаратно. Якщо задані в дескрипторі характеристики захисту не відповідають рівню привілеїв, визначеному селектором, отримати доступ до пам'яті за його допомогою буде неможливо. Так забезпечують захист пам'яті.

Проте жодного разу не було згадано, що в дескрипторі зберігають фізичну адресу. Справа в тому, що для архітектури IA-32 внаслідок перетворення логічної адреси отримують не фізичну адресу, а ще один вид адреси, який називають *лінійною* адресою.

6.3. Сторінкова організація пам'яті

До основних технологій реалізації віртуальної пам'яті крім сегментації належить *сторінкова організація пам'яті* (paging), її головна ідея – розподіл пам'яті блоками фіксованої довжини. Такі блоки називають *сторінками.*

Ця технологія є найпоширенішим підходом до реалізації віртуальної пам'яті в сучасних операційних системах.

6.3.1. Базові принципи сторінкової організації пам'яті

У разі сторінкової організації пам'яті логічну адресу називають також *лінійною,* або *віртуальною* адресою. Такі адреси належать одній множині (наприклад, лінійною адресою може бути невід'ємне ціле число довжиною 32 біти).

Фізичну пам'ять розбивають на блоки фіксованої довжини – *фрейми,* або сторінкові блоки (frames). Логічну пам'ять, у свою чергу, розбивають на блоки такої самої довжини – *сторінки* (pages). Коли процес починає виконуватися, його сторінки завантажуються в доступні фрейми фізичної пам'яті з диска або іншого носія.

Сторінкова організація пам'яті повинна мати апаратну підтримку. Кожна адреса, яку генерує процесор, ділиться на дві частини: *номер сторінки і зсув сторінки.* Номер сторінки використовують як індекс у таблиці сторінок.

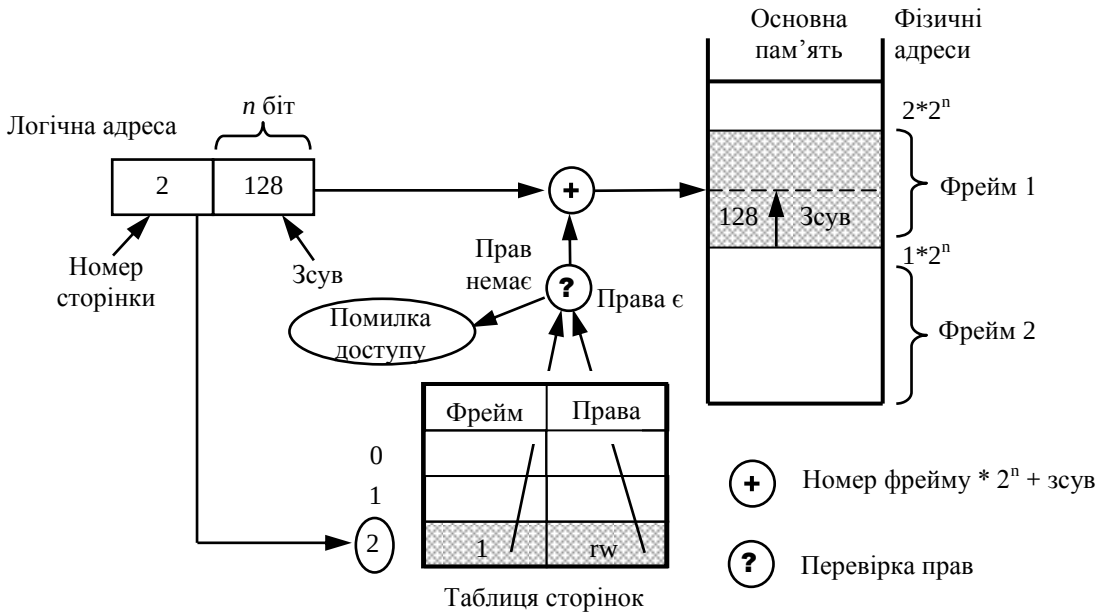
Таблиця сторінок – це структура даних, що містить набір елементів (page-table entries, PTE), кожен із яких містить інформацію про номер сторінки, номер відповідного їй фрейму фізичної пам'яті (або безпосередньо його базову адресу) та права доступу. Номер сторінки використовують для пошуку елемента в таблиці. Після його знаходження до базової адреси відповідного фрейму додають зсув сторінки, чим і визначають фізичну адресу (рис. 6.7).

Рис. 6.7. Перетворення адреси у разі сторінкової організації пам’яті

Розмір сторінки є ступенем числа 2, у сучасних ОС використовують сторінки розміром від 2 до 8 Кбайт. У спеціальних режимах адресації можна працювати зі сторінками більшого розміру.

Для кожного процесу створюють його власну таблицю сторінок. Коли процес починає своє виконання, ОС розраховує його розмір у сторінках і кількість фреймів у фізичній пам'яті. Кожну сторінку завантажують у відповідний фрейм, після чого його номер записують у таблицю сторінок процесу.

Відображення логічної пам'яті для процесу відрізняється від реального стану фізичної пам'яті. На логічному рівні для процесу вся пам'ять зображується неперервним блоком і належить тільки цьому процесові, а фізично вона розосереджена по адресному простору мікросхеми пам'яті, чергуючись із пам'яттю інших процесів (рис. 6.8). Процес не може звернутися до пам'яті, адреса якої не вказана в його таблиці сторінок (так реалізований захист пам'яті)



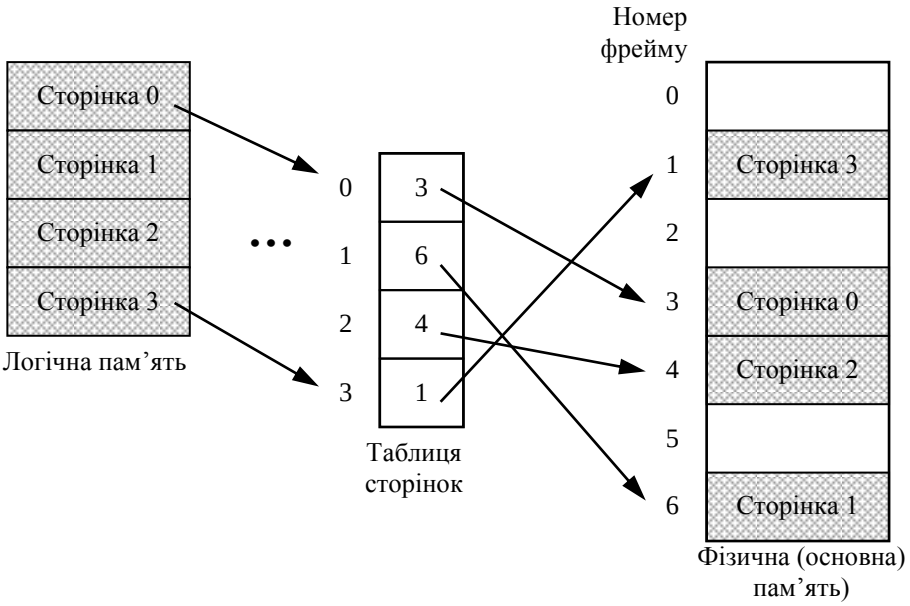


Рис. 6.8. Логічний і фізичний адресний простір у разі сторінкової організації пам’яті

ОС повинна мати інформацію про поточний стан фізичної пам'яті (про зайнятість і незайнятість фреймів, їхню кількість тощо). Цю інформацію звичайно зберігають у *таблиці фреймів.* Кожний її елемент відповідає фрейму і містить всі відомості про нього.

6.3.2. Порівняльний аналіз сторінкової організації пам'яті та сегментації

Сторінкова організація пам'яті та сегментація мають більше спільних рис, аніж відмінностей. Основна відмінність між цими двома підходами полягає в тому, що всі сторінки мають фіксовану довжину, а сегменти – змінну. Інші базові моменти (відсутність вимоги неперервності фізичної пам'яті, можливість вивантаження блоків пам'яті на диск, необхідність підтримувати таблиці перетворення тощо) принципово не відрізняються.

Розглянемо основні переваги сторінкової організації пам'яті порівняно із сегментацією. Вони визначаються насамперед тим, що всі сторінки мають одну й ту саму довжину.

- Реалізація розподілу і звільнення пам'яті спрощується. Усі сторінки з погляду процесу рівноправні, тому можна підтримувати список вільних сторінок і в разі необхідності виділяти першу сторінку із цього списку, а після звільнення повертати сторінку в список. Із сегментами так чинити не можна, оскільки кожен сегмент можна використати лише за його призначенням (спроба використати сегмент для іншої мети призведе швидше за все до того, що виникне потреба у сегменті іншої довжини).
- Реалізація обміну даними з диском також спрощується. Для організації такого обміну ділянка на диску, яку використовують для зберігання інформації про сторінки, вивантажені з пам'яті *(простір підтримки,* backing store) може бути теж розбита на блоки фіксованого розміру, рівного розмірові фрейму.

Сторінкова організація пам'яті не позбавлена й недоліків.

- Насамперед цей підхід спричиняє внутрішню фрагментацію, пов'язану з тим, що розмір сторінки завжди фіксований, і в разі необхідності виділення блоку пам'яті конкретної довжини його розмір буде кратним розміру сторінки. У середньому розмір невикористовуваної пам'яті становить приблизно половину сторінки для кожного виділеного блоку пам'яті (аналогічного до сегмента). Така фрагментація може бути знижена зменшенням кількості та збільшенням розміру блоків, що виділяються.

- Таблиці сторінок мають бути більші за розміром, ніж таблиці сегментів. Так, для виділення неперервного діапазону пам'яті розміром 100 Кбайт знадобиться один елемент таблиці сегментів, що описує сегмент, виділений для цього діапазону. З іншого боку, у разі використання сторінок розміром 4 Кбайт для опису такого діапазону нам знадобиться 25 елементів таблиці сторінок – по одному елементу для кожної сторінки.

6.3.3. Багаторівневі таблиці сторінок

Щоб адресувати логічний адресний простір значного обсягу за допомогою однієї таблиці сторінок, її доводиться робити дуже великою. Наприклад, в архітектурі IA-32 за стандартного розміру сторінки 4 Кбайт (для адресації всередині такої сторінки потрібні 12 біт) на індекс у таблиці залишається 20 біт, що відповідає таблиці сторінок на 1 мільйон елементів.

Щоб уникнути таких великих таблиць, запропоновано технологію *багаторівневих таблиць сторінок.* Таблиці сторінок самі розбиваються на сторінки, інформацію про які зберігають в таблиці сторінок верхнього рівня. Кількість рівнів рідко перевищує 2, але може доходити й до 4.

Коли є два рівні таблиць, логічну адресу розбивають на індекс у таблиці верхнього рівня, індекс у таблиці нижнього рівня і зсув.

Ця технологія має дві основні переваги. По-перше, таблиці сторінок стають менші за розміром, тому пошук у них можна робити швидше. По-друге, не всі таблиці сторінок мають перебувати в пам'яті у конкретний момент часу. Наприклад, якщо процес не використовує якийсь блок пам'яті, то вміст усіх сторінок нижнього рівня невикористовуваного блоку може бути тимчасово збережений на диску.

6.3.4. Реалізація таблиць сторінок в архітектурі IA-32

Архітектура IA-32 використовує дворівневу сторінкову організацію, починаючи з моделі Intel 80386.

Таблицю верхнього рівня називають *каталогом сторінок* (page directory), для кожної задачі повинен бути заданий окремий каталог сторінок, фізичну адресу якого зберігають у спеціальному керуючому регістрі `cr3` і куди він автоматично завантажується апаратним забезпеченням при перемиканні контексту. Таблицю нижнього рівня називають просто *таблицею сторінок* (page table).

Лінійна адреса поділяється на три поля:

- *каталогу* (Directory) – визначає елемент каталогу сторінок, що вказує на потрібну таблицю сторінок;
- *таблиці* (Table) – визначає елемент таблиці сторінок, що вказує на потрібний фрейм пам'яті;
- *зсуву* (Offset) – визначає зсув у межах фрейму, що у поєднанні з адресою фрейму формує фізичну адресу.

Розмір полів каталогу і таблиці становить 10 біт, що дає таблиці сторінок, які містять 1024 елементи, розмір поля зсуву – 12 біт, що дає сторінки і фрейми розміром 4 Кбайт. Одна таблиця сторінок нижнього рівня адресує 4 Мбайт пам'яті (1 Мбайт фреймів), а весь каталог сторінок – 4 Гбайт.

Елементи таблиць сторінок всіх рівнів мають однакову структуру. Виокремимо такі поля елемента:

- *прапорець присутності* (Present), дорівнює одиниці, якщо сторінка перебуває у фізичній пам'яті (їй відповідає фрейм); рівність цього прапорця нулю означає, що сторінки у фізичній пам'яті немає, при цьому операційна система може використати

інші поля елемента для своїх цілей;

- *20 найбільш значущих бітів*, які задають початкову адресу фрейму, кратну 4 Кбайт (може бути задано 1 Мбайт різних початкових адрес);
- *прапорець доступу* (Accessed), який покладають рівним одиниці під час кожного звертання пристрою сторінкової підтримки до відповідного фрейму;
- *прапорець зміни* (Dirty), який покладають рівним одиниці під час кожної операції записування у відповідний фрейм;
- *прапорець читання-записування* (Read/Write), що задає права доступу до цієї сторінки або таблиці сторінок (для читання і для записування або тільки для читання);
- *прапорець привілейованого ре*жиму (User/Supervisor), який визначає режим процесора, необхідний для доступу до сторінки. Якщо цей прапорець дорівнює нулю, сторінка може бути адресована тільки із привілейованого режиму, якщо одиниці – доступна також і з режиму користувача; Прапорці присутності, доступу і зміни можна використовувати ОС для організації віртуальної пам'яті.

6.3.5. Асоціативна пам'ять

Під час реалізації таблиць сторінок для отримання доступу до байта фізичної пам'яті доводиться звертатися до пам'яті кілька разів. У разі використання дворівневих сторінок потрібні *три* операції доступу: до каталогу сторінок, до таблиці сторінок і безпосередньо за адресою цього байта, а для трирівневих таблиць – *чотири* операції. Це сповільнює доступ до пам'яті та знижує загальну продуктивність системи.

Як уже зазначалося, правило «дев'яносто до десяти» свідчить, що більша частина звертань до пам'яті процесу належить до малої підмножини його сторінок, причому склад цієї підмножини змінюється досить повільно. Засобом підвищення продуктивності у разі сторінкової організації пам'яті є кешування адрес фреймів пам'яті, що відповідають цій підмножині сторінок.

Для розв'язання цієї проблеми було запропоновано технологію *асоціативної пам'яті або кеша трансляції* (translation look-aside buffers, TLB). У швидкодіючій пам'яті (швидшій, ніж основна пам'ять) створюють набір із кількох елементів (різні архітектури відводять під асоціативну пам'ять від 8 до 2048 елементів, в архітектурі IA-32 таких елементів до Pentium 4 було 32, починаючи з Pentium 4 – 128). Кожний елемент кеша трансляції відповідає одному елементу таблиці сторінок.

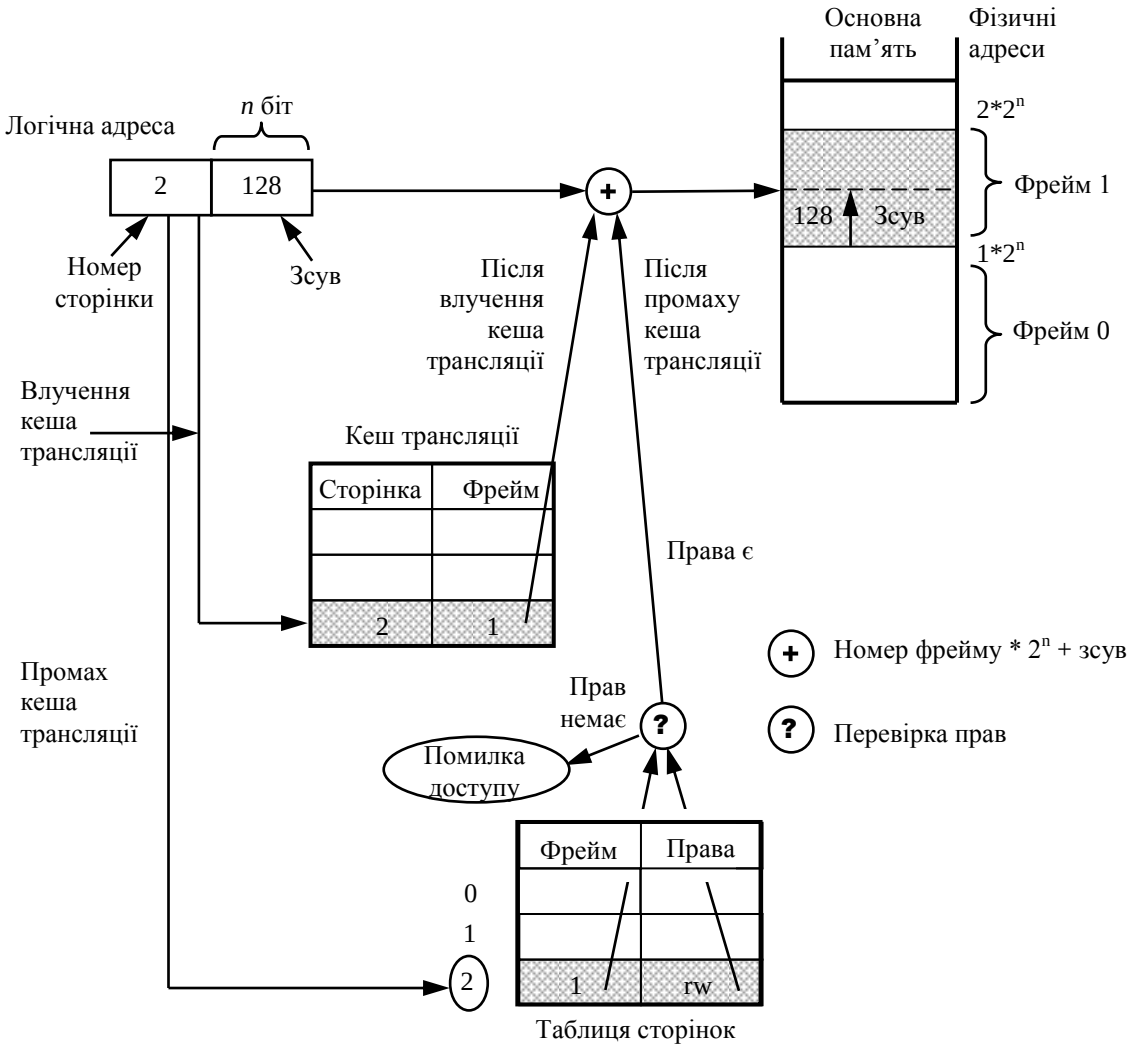
Тепер під час генерування фізичної адреси спочатку відбувається пошук відповідного елемента таблиці в кеші (в IA-32 – за полем каталогу, полем таблиці та зсуву), і якщо він знайдений, стає доступною адреса відповідного фрейму, що негайно можна використати для звертання до пам'яті. Якщо ж у кеші відповідного елемента немає, то доступ до пам'яті здійснюють через таблицю сторінок, а після цього елемент таблиці сторінок зберігають в кеші замість найстарішого елемента (рис. 6.9).

На жаль, у разі перемикання контексту в архітектурі IA-32 необхідно очистити весь кеш, оскільки в кожного процесу є своя таблиця сторінок, і ті ж самі номери сторінок для різних процесів можуть відповідати різним фреймам у фізичній пам'яті. Очищення кеша трансляції є дуже повільною операцією, якої треба всіляко уникати.

Важливою характеристикою кеша трансляції є відсоток влучень, тобто відсоток випадків, коли необхідний елемент таблиці сторінок перебуває в кеші і не потребує доступу до пам'яті. Відомо, що при 32 елементах забезпечується 98 % влучень. Зазначимо також, що за такого відсотку влучень зниження продуктивності у разі використання дворівневих таблиць сторінок порівняно з однорівневими становить 28 %, однак

переваги, отримувані під час розподілу пам'яті, роблять таке зниження допустимим.

Рис. 6.9. Принцип роботи кеша трансляції



6.4. Сторінково-сегментна організація пам'яті

Базові принципи

Оскільки сегменти мають змінну довжину і керувати ними складніше, чиста сегментація зазвичай не настільки ефективна, як сторінкова організація. З іншого боку, видається цінною сама можливість використати сегменти як блоки пам'яті різного призначення змінної довжини.

Для того щоб об'єднати переваги обох підходів, у деяких апаратних архітектурах (зокрема, в IA-32) використовують комбінацію сегментної та сторінкової організації пам'яті. За такої організації перетворення логічної адреси у фізичну відбувається за три етапи.

1. У програмі задають логічну адресу із використанням сегмента і зсуву.
2. Логічну адресу перетворюють у лінійну (віртуальну) адресу за правилами, за даними для сегментації.
3. Віртуальну адресу перетворюють у фізичну за правилами, заданими для сторінкової організації.

Таку архітектуру називають *сторінково-сегментною організацією пам'яті.*

Перетворення адрес в архітектурі IA-32

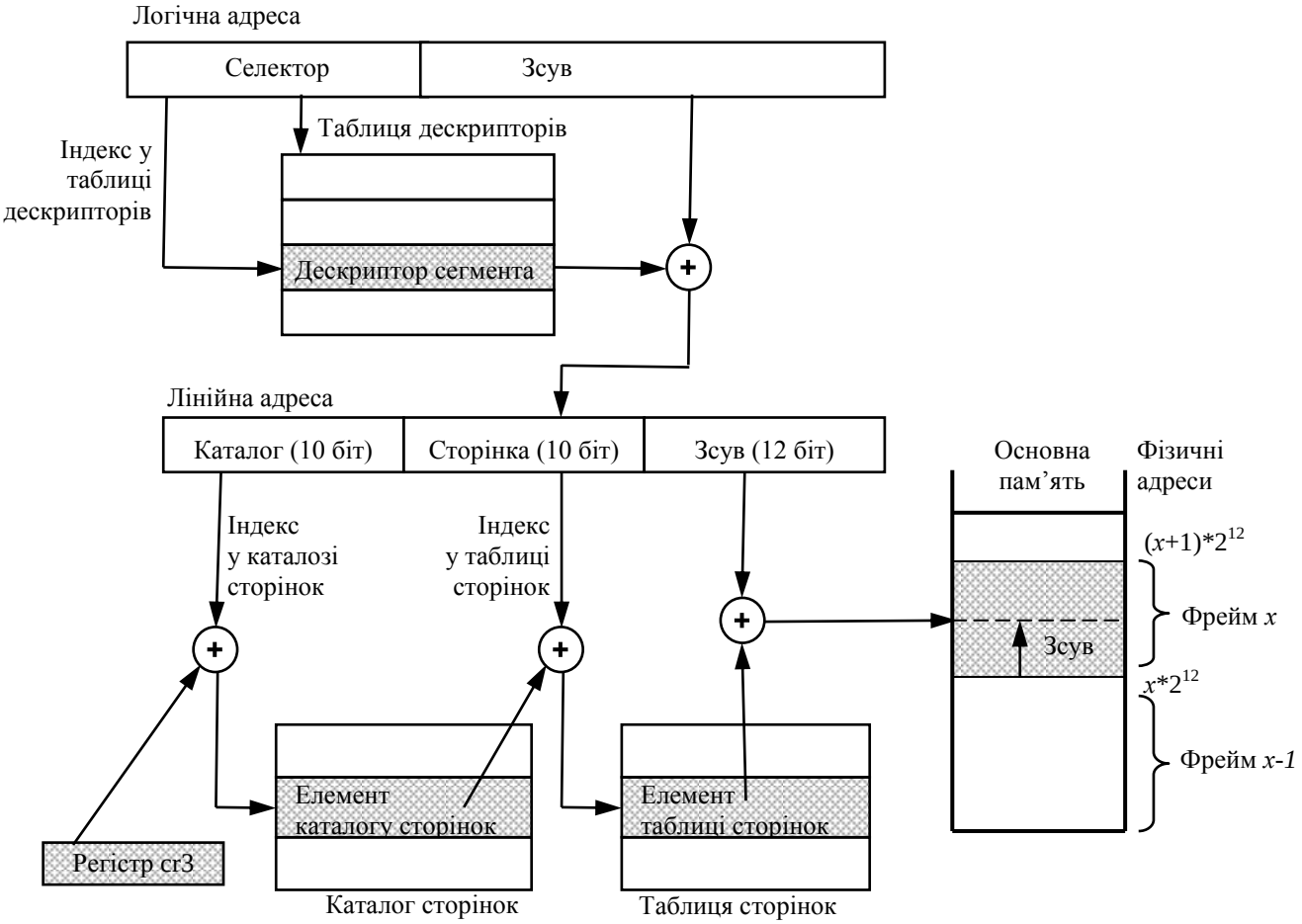
Розглянемо особливості реалізації описаних трьох етапів перетворення адреси в архітектурі IA-32.

1. Машинна мова архітектури IA-32 (а, отже, будь-яка програма, розроблена для цієї архітектури) оперує логічними адресами. Логічна адреса, як було зазначено раніше, складається із селектора і зсуву.
2. Лінійна або віртуальна адреса – це ціле число без знака завдовжки 32 біти. За його допомогою можна дістати доступ до 4 Гбайт комірок пам'яті. Перетворення логічної адреси в лінійну відбувається всередині *пристрою сегментації* (segmentation unit) за правилами перетворення адреси на базі сегментації, описаними раніше.
3. Фізичну адресу використовують для адресації комірок пам'яті в мікросхемах пам'яті, її теж зображають 32-бітовим цілим числом без знака. Перетворення лінійної адреси у фізичну відбувається всередині *пристрою сторінкової підтримки* (paging unit) за правилами для сторінкової організації пам'яті (лінійну адресу розділяють апаратурою на адресу сторінки і сторінковий зсув, а потім перетворюють у фізичну адресу із використанням таблиць сторінок, кеша трансляції тощо).

Формування адреси у разі сторінково-сегментної організації пам'яті показане на рис. 6.10.

Рис. 6.10. Сторінково-сегментна організація пам’яті в архітектурі IA-32

Необхідність підтримки сегментації в IA-32 значною мірою є даниною традиції (це



пов'язано з необхідністю зворотної сумісності зі старими моделями процесорів, у яких була відсутня підтримка сторінкової організації пам'яті). Сучасні ОС часто обходять таку сегментну організацію майже повністю, використовуючи в системі лише кілька загальних сегментів, причому кожен із них задають селектором, у дескрипторі якого поле base дорівнює нулю, а поле limit – максимальній адресі лінійної пам'яті. Зсув логічної адреси завжди буде рівний лінійній адресі, а отже, лінійну адресу можна буде формувати у програмі, фактично переходячи до чисто сторінкової організації пам'яті.

6.5. Реалізація керування основною пам'яттю: Windows

6.5.1. Сегментація у Windows

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

6.5.2. Сторінкова адресація у Windows

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

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

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

Для платформо-незалежного визначення розміру сторінки у Win32 API використовують універсальну функцію отримання інформації про систему `GetSysteminfo()`:

```
SYSTEM_INFO info;  //структура для отримання інформації про систему
GetSysteminfo(&info);
printf("Розмір сторінки: %d\n", info.dwPageSize);
```

6.5.3. Особливості адресації процесів і ядра

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

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

Деякі версії Windows XP дають можливість задати співвідношення 3 Гбайт/1 Гбайт під час завантаження системи.

6.5.4. Структура адресного простору процесів і ядра

В адресному просторі процесу можна виділити такі ділянки:

- 4 перші 64 Кбайт (починаючи з нульової адреси) – це спеціальна ділянка, доступ до якої завжди спричиняє помилки;
- усю пам'ять між першими 64 Кбайт і останніми 136 Кбайт (майже 2 Гбайт) може використовувати процес під час свого виконання;
- далі розташовані два блоки по 4 Кбайт: блоки оточення потоку (TEB) і процесу (PEB);
- наступні 4 Кбайт – ділянка пам'яті, куди відображаються різні системні дані (системний час, значення лічильника системних годин, номер версії системи), тому для доступу до них процесу не потрібно перемикатися в режим ядра; 4 останні 64

Кбайт використовують для запобігання спробам доступу за межі адресного простору процесу (спроба доступу до цієї пам'яті дасть помилку). Системний адресний простір містить велику кількість різних ділянок. Найважливіші з них наведено нижче.

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

Висновки

- Керування пам'яттю є однією з найскладніших задач, які стоять перед операційною системою. Щоб нестача пам'яті не заважала роботі користувача, потрібно розв'язувати задачу координації різних видів пам'яті. Можна використовувати повільнішу пам'ять для збільшення розміру швидшої (на цьому ґрунтується технологія віртуальної пам'яті), а також швидшу – для прискорення доступу до повільнішої (на цьому ґрунтується кешування).
- Технологія віртуальної пам'яті передбачає введення додаткових перетворень між логічними адресами, які використовують програми, та фізичними, що їх розуміє мікросхема пам'яті. На основі таких перетворень може бути реалізований захист пам'яті; крім того, вони дають змогу процесу розміщатися у фізичній пам'яті не неперервно і не повністю (ті його частини, які в цей момент часу не потрібні, можуть бути збережені на жорсткому диску). Ця технологія спирається на той факт, що тільки частина адрес процесу використовується в конкретний момент часу, тому коли зберігати в основній пам'яті тільки її, продуктивність процесу залишиться прийнятною.
- Базовими підходами до реалізації віртуальної пам'яті є сегментація і сторінкова організація. Обидва ці підходи дають можливість розглядати логічний адресний простір процесу як сукупність окремих блоків, кожен з яких може бути відображений на основну пам'ять або на диск. Головна відмінність полягає в тому, що у випадку сегментації блоки мають змінну довжину, а у разі сторінкової організації – постійну. Сьогодні часто трапляється комбінація цих двох підходів (сторінково-сегментна організація пам'яті).
- У разі сторінкової організації пам'яті логічна адреса містить номер у спеціальній структурі даних – таблиці сторінок, а також зсув відносно початку сторінки. Розділення адреси на частини відбувається апаратно. Елемент таблиці сторінок містить адресу початку блоку фізичної пам'яті, у який відображається сторінка (такий блок називають фреймом) і права доступу. Він може також відповідати сторінці, відображеній на диск. Таблиці сторінок можуть містити кілька рівнів. Таблицю верхнього рівня називають каталогом сторінок. Кожний процес має свій набір таких таблиць. Для прискорення доступу останні використані елементи таблиць сторінок кешуються в асоціативній пам'яті.

Контрольні запитання та завдання

1.Чи є необхідність реалізувати в системі віртуальну пам'ять, якщо відомо, що загальний обсяг пам'яті, необхідної для всіх активних процесів, ніколи не перевищить обсяг доступної фізичної пам'яті? Якщо така необхідність є, то які функції системи віртуальної пам'яті варто реалізувати обов'язково, а які – ні?

2. Перелічіть відмінності в реалізації та використанні сегментів даних і сегментів коду.

3. У якій ситуації виконання двох незалежних процесів, що коректно завершуються в системі зі сторінковою організацією пам'яті, призведе до взаємного блокування в системі, де такої організації немає? Передбачено, що запит на виділення пам'яті переводить процес у режим очікування, якщо для його виконання бракує фізичної пам'яті.

4. Чому розмір сторінки повинен бути степенем числа 2?

5. Припустімо, що розмір сторінки пам'яті становить 4 Кбайт, кожен її елемент займає 4 байти. Кожна таблиця сторінок повинна вміщатися на одній сторінці.
Скільки рівнів таблиць сторінок буде потрібно, щоб адресувати:

а) 32-бітний адресний простір;

б) 64-бітний адресний простір?

6. Коли використання звичайного масиву *з* лінійним пошуком виявляється не ефективним для реалізації таблиці сторінок? Які поліпшення в цьому випадку можна запропонувати?

7. Перелічіть можливі переваги сторінково-сегментної організації пам'яті порівняно з чистою сегментацією і чисто сторінковою організацією.

8. Поясніть, чому в разі використання сторінково-сегментної організації пам'яті перетворення адреси не може бути реалізоване у зворотному порядку (коли логічну адресу спочатку перетворюють за допомогою таблиці сторінок, а потім – за допомогою таблиці сегментів).

9. У системі зі сторінково-сегментною організацією пам'яті кожному процесові виділяють 64 Кбайт адресного простору для трьох сегментів: коду, даних і стека. Розмір сегмента коду для процесу дорівнює 32 Кбайт, сегмента даних – 16 400 байт, а стека – 15 800 байт. Чи достатньо адресного простору процесу для розміщення цих сегментів, якщо розмір сторінки дорівнює:

а) 4 Кбайт;

б) 512 байт?

10. Чи можуть під час виконання програми всі сегментні регістри містити однакові значення? Навіщо це може знадобитися?