ПОНЯТТЯ ВІРТУАЛЬНОЇ ПАМ'ЯТІ

Для більшості типових застосувань ОМ характерна ситуація, коли розміщення всієї програми в ОП неможливо через її великий розмір. У цьому, проте, і немає принципової необхідності, оскільки в кожен момент часу «увага» машини концентрується на визначених порівняно невеликих ділянках програми. Таким чином, в ОП досить зберігати тільки використовувані в даний період частини програм, а решта частин може розташовуватися на зовнішніх ЗП (ЗЗП). Складність подібного підходу в тому, що процеси звернення до ОП і ЗЗП істотно розрізняються, і це ускладнює завдання програміста. Виходом з такої ситуації була поява в 1959 році ідеї *віртуалізації пам'яті* [25], під якою розуміється метод автоматичного управління ієрархічною пам'яттю, при якому програмістові здається, що він має справу з єдиною пам'яттю великої ємкості і високої швидкодії. Цю пам'ять називають *віртуальною* (що здається) *пам'яттю*. За своєю суттю віртуалізація пам'яті є способом апаратної і програмної реалізації концепції ієрархічної пам'яті.

В рамках ідеї віртуалізації пам'яті ОП розглядається як лінійний простір N адрес, названий *фізичним простором* пам'яті. Для завдань, де потрібно більш ніж N комірок, надається значно більший простір адрес (зазвичай рівний загальній ємності всіх видів пам'яті), названий *віртуальним простором*, у загальному випадку не обов'язково лінійний. Адреси віртуального простору називають *віртуальними*, а адреси фізичного простору – *фізичними* (рис. 1).

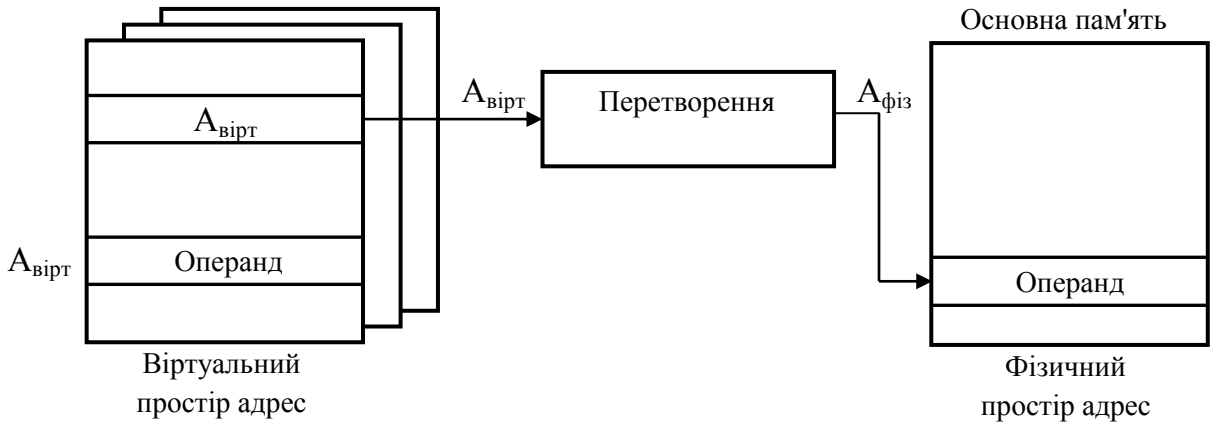

Рисунок 1 - Відображення віртуальної адреси на фізичну

Програма пишеться у віртуальних адресах, але оскільки для її виконання потрібно, щоб оброблювані команди і дані знаходилися в ОП, потрібно, щоб кожній віртуальній адресі відповідала фізична. Таким чином, у процесі обчислень необхідно, перш за все, переписати з ЗЗП в ОП ту частину інформації, на яку вказує віртуальна адреса (відобразити віртуальний простір на фізичний), після чого перетворити віртуальну адресу у фізичну (рис. 1).

Серед систем віртуальної пам'яті можна виділити два класи: системи з фіксованим розміром блоків (сторінкова організація) і системи із змінним розміром блоків (сегментна організація). Обидва варіанти зазвичай суміщають (сегментно-сторінкова організація).

1 Сторінкова організація пам'яті

Цілям перетворення віртуальних адрес у фізичні служить сторінкова організація пам'яті. Її ідея полягає в розбитті програми на частини рівної величини, що називаються *сторінками*. Розмір сторінки зазвичай вибирають в межах 4-8 Кбайт, але так, щоб він був кратний ємності одного сектора магнітного диска. Віртуальний і фізичний адресні простори розбиваються на блоки розміром у сторінку. Блок основної пам'яті, відповідний сторінці, часто називають *сторінковим кадром* або *фреймом* (page frame). Сторінкам віртуальної і фізичної пам'яті привласнюють номери. Процес доступу до даних за їх віртуальною адресою ілюструє рис. 2.

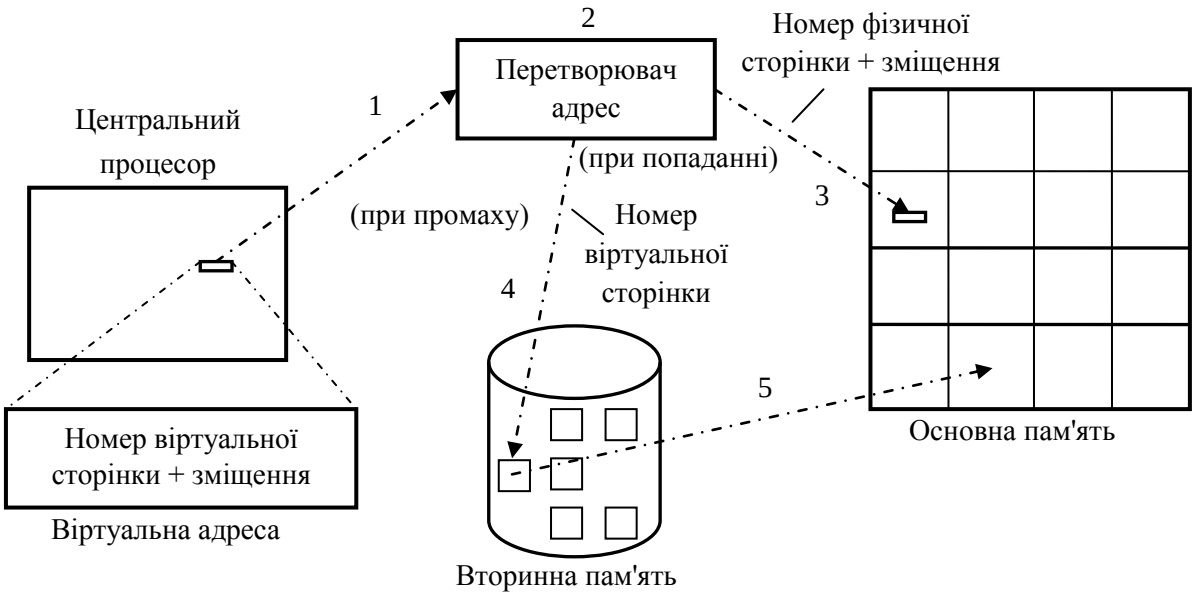

Рисунок 2 - Сторінкова організація віртуальної пам'яті

Центральний процесор звертається до комірки, вказавши її віртуальну адресу (1), яка складається з номера віртуальної сторінки і зсуву щодо її початку. Ця адреса поступає в систему перетворення адрес (2), з метою отримання з неї фізичної адреси комірки в основній пам'яті (3). Оскільки зсув у віртуальній і фізичній адресі однаковий, перетворенню піддається лише номер сторінки. Якщо перетворювач виявляє, що потрібна фізична сторінка відсутня в основній пам'яті (відбувся промах або сторінковий збій), то потрібна сторінка зчитується із зовнішньої пам'яті і заноситься в ОП (4, 5).

Перетворювач адрес – це частина операційної системи, що транслює номер віртуальної сторінки в номер фізичної сторінки, розташованої в основній пам'яті, а також апаратура, що забезпечує цей процес і що дозволяє прискорити його. Перетворення здійснюється за допомогою так званої *сторінкової таблиці*. За відсутності потрібної сторінки в ОП перетворювач адрес виробляє ознаку сторінкового збою, по якому операційна система припиняє обчислення, поки потрібна сторінка не буде зчитана з вторинної пам'яті і поміщена в основну.

Віртуальний простір повністю описується двома таблицями (рис. 4): сторінковою таблицею і картою диска (вважатимемо, що вторинна пам'ять реалізована на магнітних дисках). Таблиця сторінок визначає, які віртуальні сторінки знаходяться в основній пам'яті і в яких фізичних фреймах, а карта диска містить інформацію про сектори диска, де зберігаються віртуальні сторінки на диску.

Число записів у сторінковій таблиці (СТ) в загальному випадку рівне кількості віртуальних сторінок. Кожен запис містить поле номера фізичної сторінки (НФС) і чотири ознаки: **V**, **R**, **M** і **A.**

Ознака присутності **V** встановлюється в одиницю, якщо віртуальна сторінка в даний момент знаходиться в основній пам'яті. В цьому випадку в полі номера фізичної сторінки знаходиться відповідний номер. Якщо **V** = 0, то у разі спроби звернутися до даної віртуальної сторінки перетворювач адреси генерує сигнал сторінкового збою (page fault), і операційна система робить дії із завантаження сторінки з диска в ОП, звертаючись для цього до карти диска. В карті вказано, на якій доріжці і в якому секторі диска розташована кожна з віртуальних сторінок. Завантаження сторінки з диска в ОП супроводжується записом у відповідний рядок сторінкової таблиці (указується номер фізичної сторінки, куди була завантажена віртуальна сторінка).

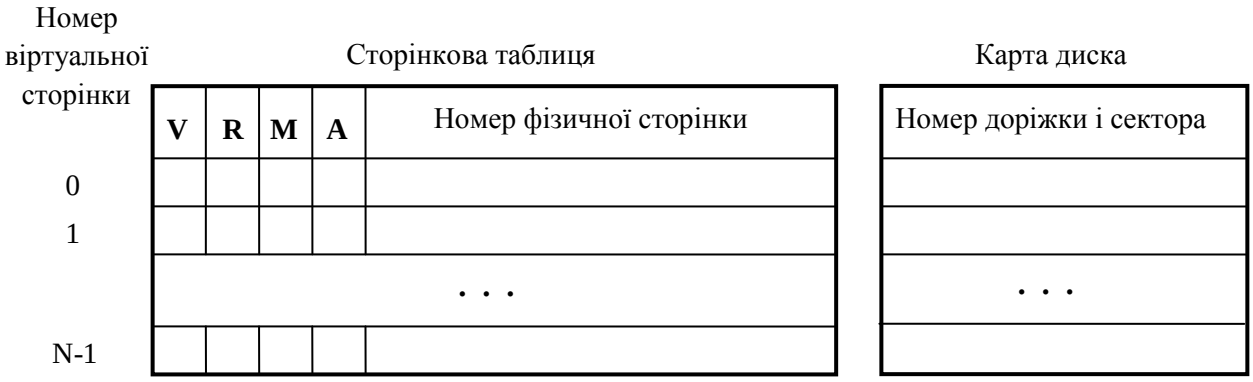

Рисунок 4 - Структура сторінкової таблиці і карти диска

У принципі карта диска може бути суміщена із сторінковою таблицею шляхом додавання до останньої ще одного поля. Іншим варіантом може бути збільшення розрядності поля номера фізичної сторінки і зберігання в нім номерів доріжок і секторів для віртуальних сторінок, відсутніх в основній пам'яті. В цьому випадку вид інформації, що зберігається, визначатиме ознаку **V**.

Ознака використання сторінки **R** встановлюється у разі звернення до даної сторінки. Ця інформація використовується в алгоритмі заміщення сторінок для вибору тієї з них, яку можна найбільш безболісно видалити з ОП, щоб звільнити місце для нової. Проблеми заміщення інформації в ОП вирішуються так само, як і для кеш-пам'яті.

Оскільки в ОП знаходяться лише копії сторінок, а їх оригінали зберігаються на диску, необхідно забезпечити ідентичність оригіналів і копій. У ході обчислень вміст окремих сторінок може змінюватися, що фіксується шляхом установки в одиницю *ознаки модифікації* **M**. Під час видалення сторінки з ОП перевіряється стан ознаки **M**. Якщо **M** = 1, то перед видаленням сторінки з основної пам'яті її необхідно переписати на диск, а при **M** = 0 цього можна не робити.

Ознака прав доступу **A** служить цілям захисту інформації і визначає, який вид доступу до сторінки дозволений: тільки для читання, тільки для запису, для читання і для запису.

Коли програма завантажується в ОП, вона може бути направлена в будь-які вільні в даний момент сторінкові кадри, незалежно від того, чи розташовані вони підряд чи ні. Сторінкова організація дозволяє скоротити об'єм пересилок інформації між зовнішньою пам'яттю і ОП, оскільки сторінку не потрібно завантажувати до тих пір, поки вона дійсно не знадобиться.

Спосіб реалізації СТ життєво важливий для ефективності техніки віртуальної адресації, оскільки кожне звернення до пам'яті припускає звернення до сторінкової таблиці. Найбільш швидкий спосіб – зберігання таблиці в спеціально виділених для цього регістрах, але від нього доводиться відмовлятися у разі великого об'єму СТ. Залишається практично один варіант – виділення сторінкової таблиці області основної пам'яті, незважаючи на те, що це приводить до двократного збільшення часу доступу до інформації. Щоб скоротити цей час, до складу ОМ включають додатковий ЗП, що називається *буфером швидкого перетворення адреси* (TLB – Translation Look-aside Buffer), або *буфером асоціативної трансляції*, або *буфером випереджаючої вибірки* і який є кеш-пам'яттю. Під час кожного перетворення номера віртуальної сторінки в номер фізичної сторінки результат заноситься в TLB: номер фізичної сторінки в пам'ять даних, а віртуальної – в пам'ять тегів. Таким чином, у TLB потрапляють результати декількох останніх операцій трансляції адрес. Під час кожного звернення до ОП перетворювач адрес спочатку проводить пошук у

пам'яті тегів TLB номера необхідної віртуальної сторінки. У разі попадання адреса відповідної фізичної сторінки береться з пам'яті даних TLB. Якщо в TLB зафіксований промах, то процедура перетворення адрес проводиться за допомогою сторінкової таблиці, після чого здійснюється запис нової пари «номер віртуальної сторінки – номер фізичної сторінки» в TLB. Структура TLB показана на рис. 5.

Номер віртуальної сторінки	V	R	M	A	Номер фізичної сторінки
. . .	. . .				

Пам'ять тегів Пам'ять даних

Рисунок 5 - Структура буфера швидкого перетворення адрес

Буфер перетворення адрес зазвичай реалізується у вигляді повністю асоціативної або множинно-асоціативної кеш-пам'яті з високим ступенем асоціативності і часом доступу, що збігається з аналогічним показником для кеш-пам'яті першого рівня (L1). Число входів у типових TLB невелике (64-256). Так, TLB мікропроцесора Pentium III має 64 входи при розмірі сторінки 4 Кбайт, що дозволяє дістати швидкий доступ до адресного простору в 256 Кбайт.

Серйозною проблемою в системі віртуальної пам'яті є великий об'єм сторінкових таблиць, який пропорційний числу віртуальних сторінок. Таблиця займає значну частину ОП, а на пошук іде багато часу, що вкрай небажано.

Один із способів скорочення довжини таблиць заснований на введенні багаторівневої організації таблиць. У цьому варіанті інформація оформляється у вигляді декількох сторінкових таблиць порівняно невеликого об'єму, які утворюють другий рівень. Перший рівень представлений таблицею з каталогом, де вказано місцеположення кожної із сторінкових таблиць (адреса початку таблиці в пам'яті) другого рівня. Спочатку в каталозі визначається розташування потрібної сторінкової таблиці і лише потім проводиться звернення до потрібної таблиці. Про ефект, що досягається, можна судити з наступного прикладу. Хай адресна шина ОМ має ширину 32 біта, а розмір сторінки дорівнює 4 Кбайт. Тоді кількість віртуальних сторінок, а отже, і число входів у єдиній сторінковій таблиці складе 2^{20}. У разі дворівневої організації можна обійтися однією сторінкою першого рівня на 1024 (2^{10}) входів і 1024 сторінковими таблицями на таке ж число входів.

Інший підхід називають *способом обернених* або *інвертованих сторінкових таблиць*.

Таку таблицю в якомусь сенсі можна розглядати як збільшений еквівалент TLB, який відрізняється тим, що вона зберігається в ОП і реалізується не апаратурою, а програмними засобами. Число входів у таблицю визначається ємністю ОП і дорівнює числу сторінок, яке може бути розміщене в основній пам'яті. Одночасно з цим є і традиційна СТ, але зберігається вона не в ОП, а на диску. Для пошуку потрібного запису в інвертованій таблиці використовується хешування, коли номер запису в таблиці обчислюється відповідно до якоїсь хеш-функції. Аргументом цієї функції служить номер шуканої віртуальної сторінки. Хешування дозволяє прискорити операцію пошуку. Якщо потрібна сторінка в ОП відсутня, проводиться звернення до основної таблиці на диску і після завантаження сторінки в ОП коректується і інвертована таблиця.

2 Сегментно - сторінкова організація пам'яті

У разі сторінкової організації передбачається, що віртуальна пам'ять – це безперервний масив з наскрізною нумерацією слів, що не завжди можна визнати оптимальним. Зазвичай програма складається з декількох частин – кодової, інформаційної та стекової. Оскільки заздалегідь невідомі довжини цих складових, то зручно, щоб під час програмування кожна з них мала власну нумерацію слів, відлічуваних з нуля. Для цього організовують систему *сегментованої пам'яті*, виділяючи у віртуальному просторі незалежні лінійні простори змінної довжини, що називаються *сегментами*. Кожен сегмент є окремою логічною одиницею інформації, що містить сукупність даних або програмний код і розташована в адресному просторі користувача. В кожному сегменті встановлюється своя власна нумерація слів, починаючи з нуля. Віртуальна пам'ять також розбивається на сегменти, з незалежною адресацією слів усередині сегмента. Кожній складовій програми виділяється сегмент пам'яті. Віртуальна адреса визначається номером сегмента і адресою всередині сегмента. Для перетворення віртуальної адреси у фізичну використовується спеціальна *сегментна таблиця*.

Недоліком такого підходу є те, що неоднаковий розмір сегментів приводить до неефективного використання ОП. Так, якщо ОП заповнена, то у разі заміщення одного з сегментів потрібно витіснити такий, розмір якого дорівнює або більше розміру нового. У разі багатократного повтору подібних дій в ОП залишається багато вільних ділянок, недостатніх за розміром для завантаження повного сегмента. Вирішенням проблеми служить *сегментно-сторінкова організація пам'яті*. В ній розмір сегмента вибирається не довільно, а задається кратним розміру сторінки.

Структуру віртуальної адреси і процес перетворення її у фізичну адресу ілюструє рис. 6.

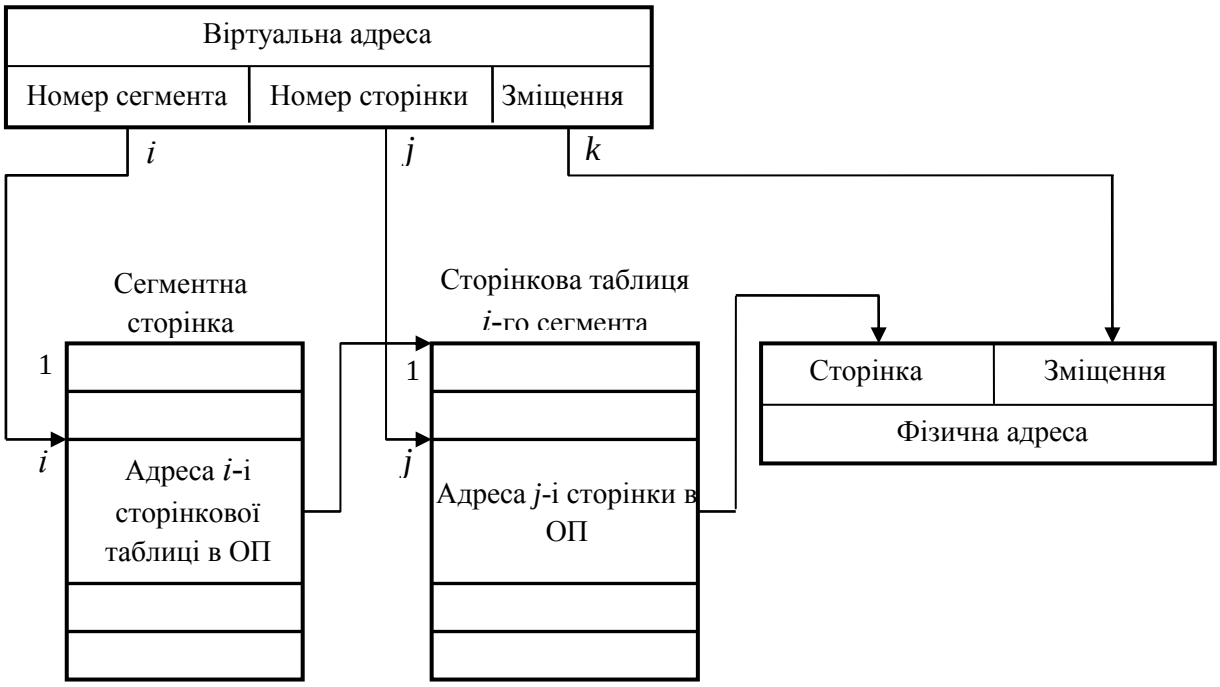

Рисунок 6 - Перетворення адреси при сегментно-сторінковій організації пам'яті

Сегмент може містити те або інше, але обов'язково ціле число сторінок, навіть якщо одна із сторінок заповнена частково. Виникає певна ієрархія в організації доступу до даних, що складається з трьох ступенів: сегмент > сторінка > слово. Цій структурі відповідає ієрархія таблиць, які служать для перекладу віртуальних адрес у фізичні. В сегментній таблиці програми перераховуються всі сегменти даної програми з вказівкою початкових адрес СТ, які відносяться до кожного сегмента. Кількість сторінкових таблиць дорівнює числу сегментів і будь-яка з них визначає розташування кожної із сторінок сегмента в пам'яті, які можуть розташовуватися не підряд – частина сторінок може знаходитися в ОП, останні – у зовнішній пам'яті

Для отримання фізичної адреси необхідний доступ до сегментної та однієї із сторінкових таблиць, тому перетворення адреси може займати багато часу.

ОРГАНІЗАЦІЯ ЗАХИСТУ ПАМ'ЯТІ

Сучасні обчислювальні машини, як правило, працюють у багатокористувацькому і багатозадачному режимах, коли в основній пам'яті одночасно знаходяться програми, що відносяться як до різних користувачів, так і до різних завдань одного користувача. Якщо навіть ОМ виконує тільки одну програму, в ОП, крім цієї програми і даних до неї, завжди присутні фрагменти операційної системи. Кожному завданню в основній пам'яті виділяється свій адресний простір. Такі простори, якщо тільки це спеціально не передбачено, зазвичай незалежні. В той же час у програмах можуть міститися помилки, які приводять

до вторгнення в адресний простір інших завдань. Наслідком цих помилок може стати спотворення інформації, що належить іншим програмам. Отже, в ОМ обов'язково повинні бути передбачені заходи, що запобігають несанкціонованій дії програм одного користувача на роботу програм інших користувачів і на операційну систему.

Щоб перешкодити руйнуванню одних програм іншими, досить захистити область пам'яті даної програми від спроб запису в неї з боку інших програм (захист від запису). В ряді випадків необхідно мати можливість захисту і від читання з боку інших програм, наприклад, у разі обмежень на доступ до системної інформації.

Захист від вторгнення програм у чужі адресні простори реалізується різними засобами і способами, але в будь-якому варіанті до системи захисту пред'являються дві вимоги: її реалізація не повинна помітно знижувати продуктивність ОМ і вимагати дуже великих апаратних витрат.

Задача зазвичай вирішується за рахунок поєднання програмних і апаратних засобів, хоч відповідальність за охорону адресних просторів від несанкціонованого доступу зазвичай покладається на операційну систему. Коротко розглянемо апаратні аспекти проблеми захисту пам'яті.

Кільця захисту. Захист адресного простору операційної системи від несанкціонованого вторгнення з боку призначених для користувача програм зазвичай організовують за рахунок апаратно реалізованого розділення системного і користувацького рівнів привілеїв. Передбачаються як мінімум два режими роботи процесора: системний (режим супервізора – «наглядача») і користувацький. Таку структуру прийнято називати *кільцями захисту* і зображати у вигляді концентричних кіл, де призначений для користувача режим поданий зовнішнім кільцем, а системний – внутрішнім колом. У системному режимі програмі доступні всі ресурси ОМ, а можливості призначеного для користувача режиму істотно обмежені. Перемикання з призначеного для користувача режиму в системний здійснюється спеціальною командою. В більшості сучасних ОМ число рівнів привілеїв (кілець захисту) збільшене. Так, у мікропроцесорах класу Pentium передбачено чотири рівні привілеїв.

Метод граничних регістрів. Даний вид захисту найбільш поширений. Метод припускає наявність у процесорі двох *граничних регістрів*, вміст яких визначає нижню і верхню межі області пам'яті, куди програма має право доступу. Заповнення граничних регістрів проводиться операційною системою під час завантаження програми. Під час кожного звернення до пам'яті перевіряється, чи потрапляє використовувана адреса у встановлені межі. Таку перевірку, наприклад, можна організувати на етапі перетворення віртуальної адреси у фізичну. У разі порушення межі доступ до пам'яті блокується і

формується запит переривання, що викликає відповідну процедуру операційної системи. Нижню межу дозволеної області пам'яті визначає сегментний регістр. Верхня межа підраховується операційною системою відповідно до розміру розміщуваного в ОП сегмента.

У розглянутій схемі необхідно, щоб в ОМ підтримувалися два режими роботи: привілейований і користувацький. Запис інформації в граничні регістри можливий лише в привілейованому режимі.

Метод ключів захисту. Метод дозволяє організувати захист несуміжних областей пам'яті. Пам'ять умовно ділиться на блоки однакового розміру. Кожному блоку ставиться у відповідність деякий код, названий *ключем захисту пам'яті*. Кожній програмі, у свою чергу, привласнюється *код захисту програми*. Умовою доступу програми до конкретного блока пам'яті служить збіг ключів захисту пам'яті і програми або рівність одного з цих ключів нулю. Нульове значення ключа захисту програми дозволяє доступ до всього адресного простору і використовується тільки програмами операційної системи. За розподіл ключів захисту програми відповідає операційна система. Ключ захисту програми зазвичай представлений у вигляді окремого поля *слова стану програми*, яке зберігається в спеціальному регістрі. Ключі захисту пам'яті зберігаються в спеціальній пам'яті. Під час кожного звернення до ОП спеціальна комбінаційна схема проводить порівняння ключів захисту пам'яті і програми. У разі збігу доступ до пам'яті дозволяється. Дії у разі незбігання ключів залежать від того, який вид доступу заборонений: під час запису, під час читання або в обох випадках. Якщо з'ясувалося, що даний вид доступу заборонений, то так само, як і в методі граничних регістрів, формується запит переривання і викликається відповідна процедура операційної системи.