## 10.4 СЕГМЕНТНА ОРГАНІЗАЦІЯ ВІРТУАЛЬНОЇ ПАМ'ЯТІ

При сторінковій організації віртуальний адресний простір процесу ділиться механічно на рівні частини. Це не дозволяє диференціювати способи доступу до різних частин програми (сегментів), а ця властивість часто буває дуже корисною. Наприклад, можна заборонити поводитися з операціями запису і читання в кодовий сегмент програми, а для сегменту даних дозволити тільки читання. Крім того, розбиття програми на «осмислені» частини робить принципово можливим розділення одного сегменту декількома процесами. Наприклад, якщо два процеси використовують одну і ту ж математичну підпрограму, то в оперативну пам'ять може бути завантажена тільки одна копія цієї підпрограми. Покращується захист сегментів, оскільки програміст може визначати певні ділянки програми або даних і призначати їм права доступу.

До сих пір розглянута віртуальна пам'ять була одновимірною, оскільки в ній адреси слідували одна за одною від 0 до деякого максимального значення. Але для вирішення багатьох проблем наявність двох і більше окремих віртуальних адресних просторів може бути раціональнішим варіантом, ніж наявність тільки одного адресного простору. Наприклад, у компілятора є декілька таблиць, що створюються в процесі компіляції, в які можуть входити:

1. Початковий текст, збережений для друку лістингу (у пакетних системах).
2. Таблиця імен, що містить імена і атрибути змінних.
3. Таблиця, що містить усі використовувані константи, як цілочисельні, так і з плаваючою точкою.
4. Дерево розбору, в якому міститься синтаксичний аналіз програми.
5. Стек, який використовується для викликів процедур усередині компілятора.

У процесі компіляції кожна з перших чотирьох таблиць постійно росте. А остання збільшується і зменшується в розмірах абсолютно непередбачуваним чином. В одновимірній пам'яті цим п'яти таблицям мають бути виділені послідовні ділянки віртуального адресного простору, показані на рис. 10.19 [9].


Рисунок 10.19 – Розміщення таблиць компілятора в одновимірному адресному просторі

Розглянемо, що вийде, якщо програма містить набагато більшу, ніж зазвичай, кількість змінних, але цілком звичайну кількість усіх інших компонентів. Ділянка адресного простору, виділена під таблицю імен, може заповнитися повністю, але для інших таблиць може залишитися велика кількість вільного простору. Зрозуміло, що компілятор може просто видати повідомлення про те, що він не може продовжити роботу через занадто велику кількість змінних, але усе це виглядатиме не занадто переконливо на тлі того, що у інших таблиць залишився невикористаний простір.

Існує також інша можливість: забрати простір у таблиць, що мають його в надлишку, і передати його таблицям з дефіцитом простору. У кращому випадку це принесе одні неприємності, а в гіршому – стомливу і нерозумну роботу. Насправді тут потрібний спосіб позбавити програміста від необхідності збільшення і зменшення розмірів таблиць, аналогічно тому, як віртуальна пам'ять усуває недліки з приводу організації програми в оверлеї.

Простим і дуже універсальним рішенням є надання машини з великою кількістю абсолютно незалежних адресних просторів, що називаються **сегментами**. Кожен сегмент складається з лінійної послідовності адрес від 0 до деякого

максимуму. Довжина кожного сегменту може мати будь-яке значення від 0 до максимально дозволеного. Різні сегменти можуть бути різної довжини, як це звичайно і трапляється. Крім того, довжина сегменту може змінюватися в процесі виконання програми. Довжина сегменту стека може збільшуватися при записі в нього даних і зменшуватися при їх витяганні з нього.

Оскільки кожен сегмент містить окремий адресний простір, різні сегменти можуть розростатися або звужуватися незалежно, не впливаючи один на одного. Для вказівки адреси в такій сегментованій, або двовимірній пам'яті, програма повинна надати адресу, що складається з двох частин: номери сегменту і адреси усередині цього сегменту. На рис. 10.20 показано п'ять незалежних сегментів пам'яті, яка використовується для розглянутих раніше таблиць компілятора.

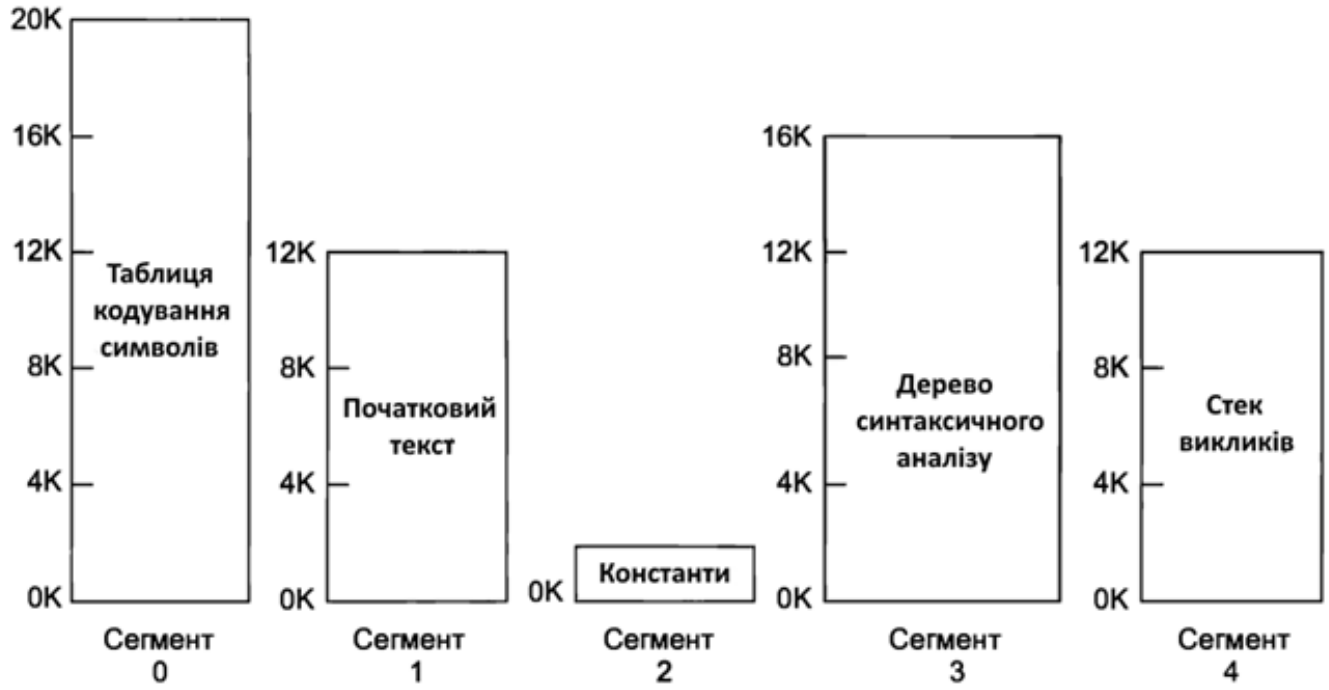


Рисунок 10.20 – Сегментована пам'ять з п'ятьма незалежними сегментами

Після того як усі процедури програми скомпільовані і скомпоновані, то у виклику процедури, яка звертається до процедури в сегменті n, буде використана адреса, що складається з двох частин (n, 0) і адресована до слова 0 (до точки входу).

Якщо згодом процедура в сегменті n буде змінена і перекомпільована, то змінювати інші процедури вже не доведеться (оскільки початкові адреси не будуть змінені), навіть якщо нова версія буде більше за стару. При використанні

одновимірної пам'яті процедури компонуються безпосередньо одна за одною, без якого-небудь адресного простору між ними.

Отже, зміна розмірів однієї процедури вплине на початкові адреси інших (не пов'язаних з нею) процедур. А це, у свою чергу, зажадає зміни усіх процедур, з яких викликаються будь-які з переміщених процедур, щоб врахувати їх нові початкові адреси.

Сегментація також полегшує спільне використання процедур або даних декількома процесами. Типовим прикладом може послужити спільно використовувана бібліотека. У сегментованій системі такі бібліотеки можуть бути поміщені в сегмент і спільно використовуватися декількома процесами, виключаючи потребу у своїй присутності в адресному просторі кожного процесу.

Оскільки кожен сегмент формує логічний об'єкт, наприклад процедуру, або масив, або стек, у різних сегментів можуть бути різні види захисту. Сегмент процедури може бути визначений тільки як виконуваний, із забороною спроб що-небудь в ньому прочитати або зберегти. Масив чисел з плаваючою точкою може бути визначений для читання і запису, але не для виконання, і спроби передати йому управління. Подібний захист дуже корисний при виявленні помилок програмування.

Розглянемо, яким чином сегментний розподіл пам'яті реалізує ці можливості (рис. 10.21). Віртуальний адресний простір процесу ділиться на сегменти, розмір яких визначається програмістом з урахуванням смислового значення інформації, що міститься в них. Окремий сегмент може бути підпрограмою, масивом даних і тому подібне. Іноді сегментація програми виконується за умовчанням компілятором.

При завантаженні процесу частина сегментів поміщається в оперативну пам'ять, а частина сегментів розміщується в дисковій пам'яті. Сегменти однієї програми можуть займати в оперативній пам'яті несуміжні ділянки.

Під час завантаження система створює таблицю сегментів процесу (аналогічно таблиці сторінок), в якій для кожного сегменту вказується початкова фізична адреса сегменту в оперативній пам'яті, розмір сегменту, правила доступу, ознака модифікації, ознака звернення до цього сегменту за останній інтервал часу і деяка інша інформація (дескриптор сегмента).

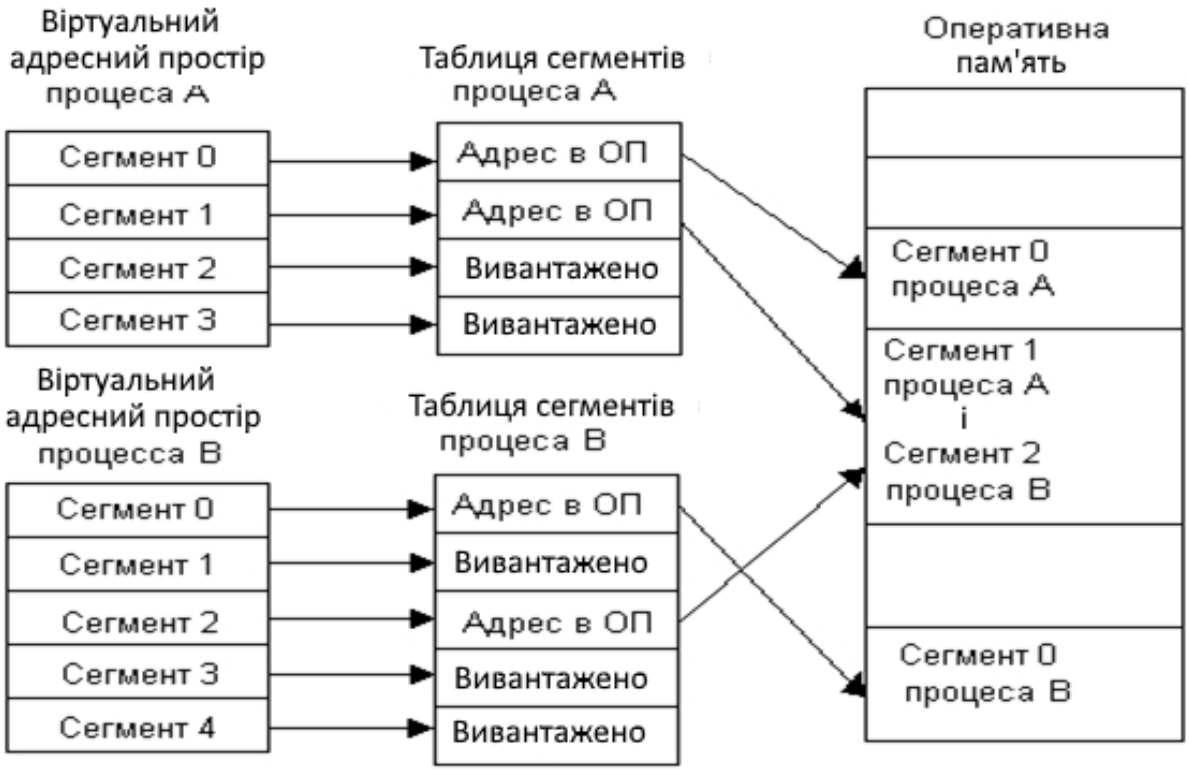


Рисунок 10.21 – Розподіл пам'яті сегментами

Біт модифікації вказує, чи було змінено вміст сегменту з часу його останнього завантаження в основну пам'ять. Якщо змін не було, то при вивантаженні сегменту немає необхідності в його повторному записі на диск.

Якщо віртуальні адресні простори декількох процесів включають один і той же сегмент, то в таблицях сегментів цих процесів робляться посилання на одну і ту ж ділянку оперативної пам'яті, в яку цей сегмент завантажується в єдиному екземплярі. Система з сегментною організацією функціонує аналогічно системі із сторінковою організацією. Час від часу відбуваються переривання, пов'язані з відсутністю потрібних сегментів в пам'яті. При необхідності звільнення пам'яті деякі сегменти вивантажуються. При кожному зверненні до оперативної пам'яті виконується перетворення віртуальної адреси у фізичну адресу. Крім того, при зверненні до пам'яті перевіряється чи дозволений доступ необхідного типу до цього сегменту.

Віртуальна адреса при сегментній організації пам'яті може бути представлена парою (g, s), де g – номер сегменту, а s – зміщення в сегменті. Фізична адреса виходить шляхом складання початкової фізичної адреси сегменту, знайденого в таблиці сегментів за номером g, і зміщення s (рис. 10.22). У даному випадку не можна обійтися операцією конкатенації, як це робиться при сторінковій організації пам'яті.

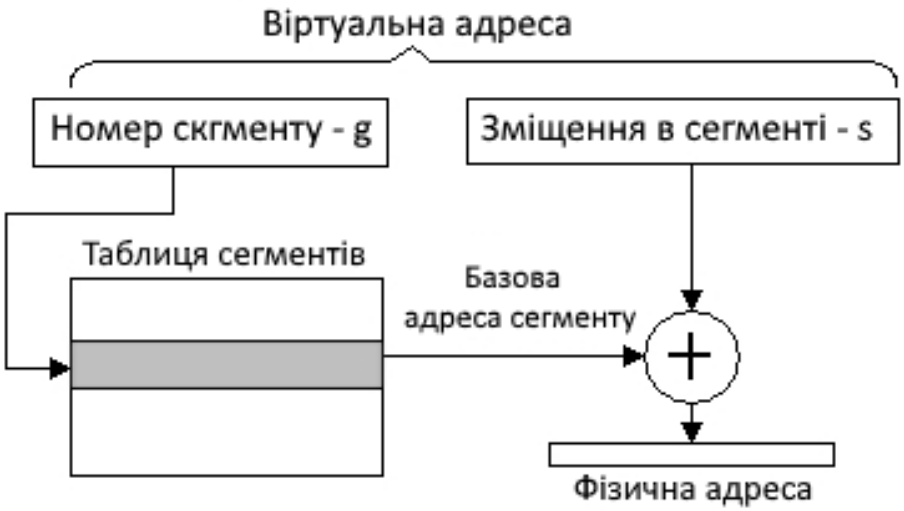


Рисунок 10.22 – Перетворення віртуальної адреси при сегментній організації пам'яті

Дійсно, оскільки розмір сторінки дорівнює степені двійки, то в двійковому виді він виражається числом з декількома нулями в молодших розрядах. Сторінки мають однаковий розмір, тобто їх початкові адреси кратні розміру сторінок і виражаються також числами з нулями в молодших розрядах. Саме тому ОС заносить в таблиці сторінок не повні адреси, а номери фізичних сторінок, які співпадають із старшими розрядами базових адрес. Сегмент же може в загальному випадку розташовуватися у фізичній пам'яті, починаючи з будь-якої адреси (байта). Отже, для визначення місця розташування в пам'яті необхідно задавати його повну **початкову фізичну адресу.**

На рис. 10.23 зображений приклад звернення до елементу пам'яті, віртуальна адреса якого дорівнює сегменту з номером 11 і зміщенням від початку цього сегменту, рівним 612. Операційна система розмістила цей сегмент у пам'яті, починаючи з комірки з номером 19700.

Використання операції складання замість конкатенації уповільнює процедуру перетворення віртуальної адреси у фізичну в порівнянні із сторінковою організацією.

Іншим недоліком сегментного розподілу є надмірність. При сегментній організації одиницею переміщення між пам'яттю і диском є сегмент, що має в загальному випадку об'єм більший, ніж сторінка. Проте в багатьох випадках для роботи програми зовсім не потрібно завантажувати увесь сегмент цілком, досить було б однієї або двох сторінок. Аналогічно за відсутності вільного місця в пам'яті не варто вивантажувати цілий сегмент, коли можна обійтися вивантаженням декількох сторінок.

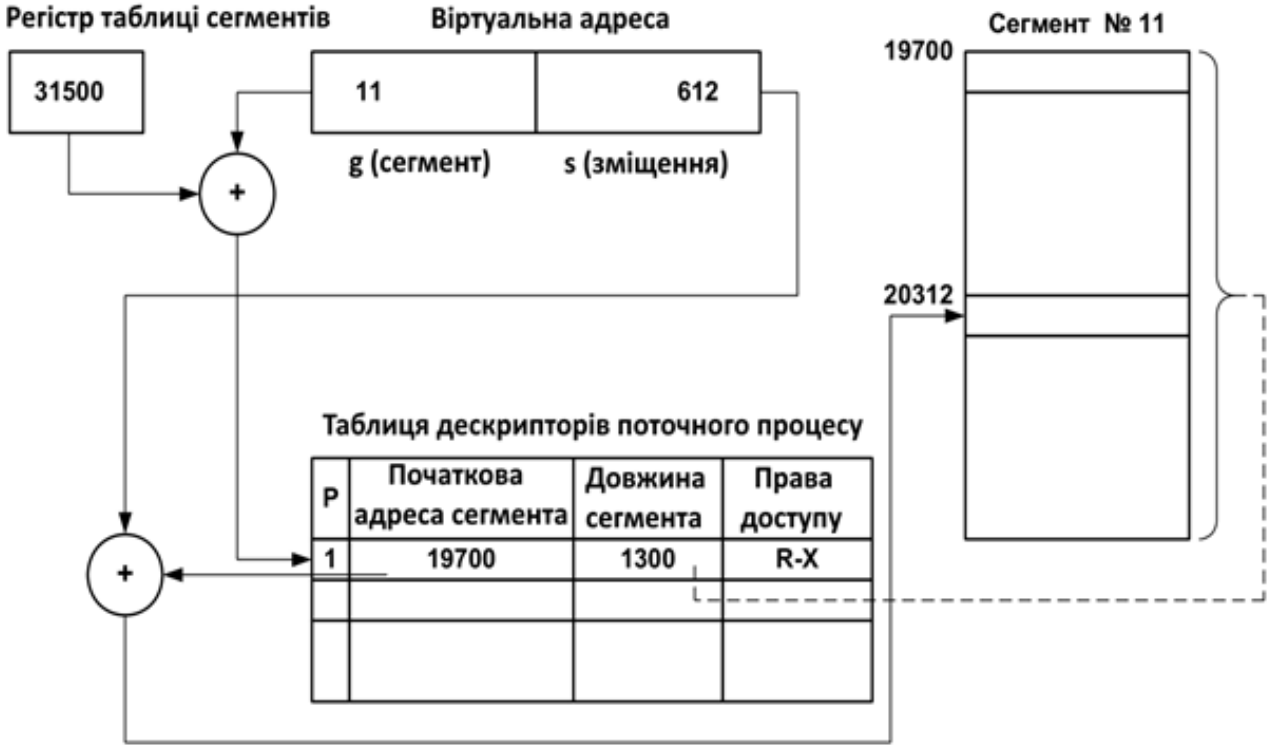


Рисунок 10.23 – Приклад розміщення сегменту в пам'яті

Але головний недолік сегментного розподілу – це фрагментація на рівні сегментів (зовнішня фрагментація), яка виникає із-за непередбачуваності розмірів сегментів. У процесі роботи системи в пам'яті утворюються невеликі ділянки вільної пам'яті, в які не може бути завантажений жоден сегмент. Сумарний об'єм, зайнятий фрагментами, може скласти істотну частину загальної пам'яті системи, призводячи до її неефективного використання.

На рис. 10.24, а показаний приклад фізичної пам'яті, що спочатку має п'ять сегментів. Тепер розглянемо, що вийде, якщо сегмент 1 віддаляється, а на його місце поміщається менший за розміром сегмент 7. У нас вийде конфігурація пам'яті, показана на рис. 10.24, б. Між сегментом 7 і сегментом 2 буде невживана область, тобто діра. Потім сегмент 4 замінюється сегментом 5, як показано на рис. 10.24, в, а сегмент 3 замінюється сегментом 6, як показано на рис. 10.24, г.

Після того як система деякий час попрацює, пам'ять розділиться на декілька ділянок, частина з яких міститимуть сегменти, а частину – діри. Це явище називають явищем шахівниці або зовнішньою фрагментацією, що призводить до марної трати пам'яті на діри. З цим можна впоратися за рахунок ущільнення, показаного на рис. 10.24, д.

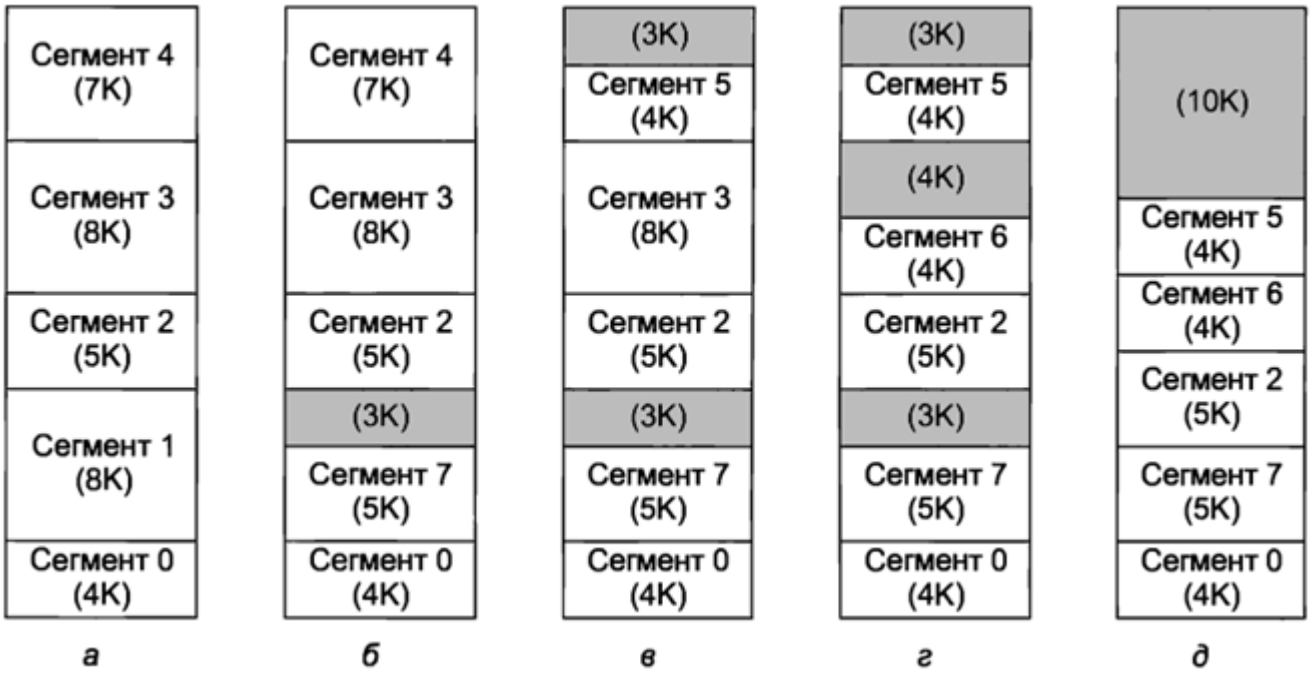


Рисунок 10.24 – Наростання зовнішньої фрагментації (а – г); позбавлення від зовнішньої фрагментації за рахунок ущільнення (д)

Оскільки таблиця сегментів має змінну довжину, залежну від розміру процесу, то найчастіше вона зберігається в оперативній пам'яті, а не в швидких регістрах. Використанням сегментів різного розміру цей спосіб схожий на динамічний розподіл пам'яті. Проте на відміну від динамічного розподілу в цьому випадку сегменти можуть займати декілька несуміжних розділів.