10.2 МЕТОДИ РОЗПОДІЛУ ПАМ'ЯТІ

10.2.1 Розподіл пам'яті фіксованими розділами

Найпростішим способом управління оперативною пам'яттю без використання дискового простору є розділення її на області з фіксованими межами. Це може бути виконано під час генерації ОС. У тому випадку, коли розділи мають однаковий розмір, розміщення процесів в пам'яті є тривіальною задачею. Не має значення, в якому з вільних розділів буде розміщений процес. Але при цьому підході є і свої труднощі:

1. Програма може бути занадто велика для розміщення в розділі. В цьому випадку програміст повинен розробити програму, що використовує оверлеї (перекриття).

2. Використання пам'яті при цьому вкрай неефективно. Так, маленька програма все одно займатиме цілком увесь розділ, при цьому залишається невикористана пам'ять. Цей феномен невикористаної пам'яті називається внутрішньою фрагментацією.

Коли розділи мають різні розміри, чергова задача, що поступило на виконання, поміщається або в загальну чергу (рис. 10.3, а), або в чергу до деякого розділу (рис. 10.3, б).

Підсистема управління пам'яттю в цьому випадку виконує такі задачі:

– вибирає відповідний розділ, порівнюючи розмір програми, що поступила на виконання, і вільних розділів;

– здійснює завантаження програми і налаштування адрес.

Перевага такого підходу полягає в тому, що процеси можуть бути розподілені між розділами пам'яті так, щоб мінімізувати внутрішню фрагментацію. Використання розділів різного розміру в порівнянні з використанням розділів однакового розміру надає додаткову гнучкість цьому методу.

При очевидній перевазі (простоті реалізації) цей метод, у цілому, має істотний недолік – жорсткість. Оскільки в кожному розділі може виконуватися тільки одна програма, то рівень мультипрограмування заздалегідь обмежений числом розділів, не залежно від того, який розмір мають програми.

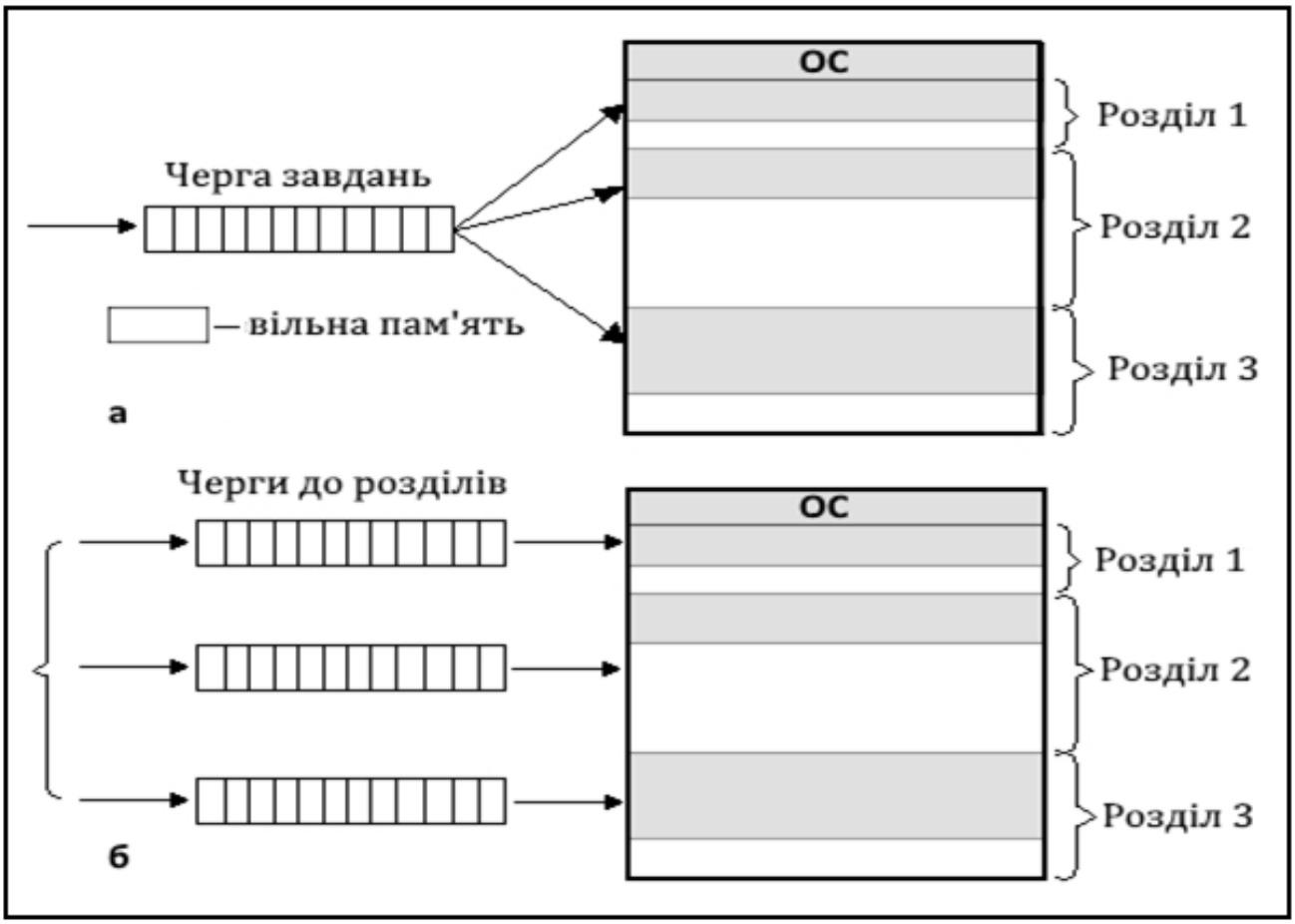

Рисунок 10.3 – Розподіл пам'яті фіксованими розділами:
а) – із загальною чергою; б) – з окремими чергами для кожного розділу

Навіть якщо програма має невеликий об'єм, вона займатиме увесь розділ, що призводить до неефективного використання пам'яті. З іншого боку, навіть якщо об'єм оперативної пам'яті машини дозволяє виконати деяку програму, розбиття пам'яті на розділи не дозволяє зробити цього.

Оскільки модель з фіксованими розділами обмежує віртуальний адресний простір розмірами реальної пам'яті, виникає проблема великих програм, що не вміщуються в доступну пам'ять. Подолання цього обмеження досягається створенням програм з оверлейною структурою (overlay – перекриття).

Структура міжмодульних викликів оверлейної програми має вигляд дерева. В корені дерева (нульовий рівень) знаходиться модуль-монітор, з якого здійснюється звернення до модулів, розташованих на гілках дерева. Кожен модуль першого рівня у свою чергу може бути коренем піддерева і так далі. Принцип оверлеїв або

перекриття полягає в тому, що гілки дерева займають одні і ті ж області у віртуальному адресному просторі програми.

Оскільки модулі, розташовані в різних гілках дерева, не можуть виконуватися одночасно, то тільки монітор є резидентним в оперативній пам'яті весь час виконання програми, гілки ж змінюють одна одну в одній і тій же області пам'яті. Насправді побудувати програму, що має ідеальну деревовидну структуру, досить важко.

У системах, що підтримують розвинені засоби створення оверлейних програм (OS/360), велика робота покладається на редактор зв'язків, який формує вміст адресного простору процесу відповідно до оверлейної структури, що описується програмістом за допомогою спеціальної мови.

У цілому можна відмітити, що схеми з фіксованими розділами відносно прості, тому пред'являються мінімальні вимоги до ОС; накладні витрати роботи процесора на розподіл пам'яті невеликі. Проте у цих схем є серйозні недоліки.

1. Кількість розділів, визначена в момент генерації системи, обмежує кількість активних процесів.

2. Оскільки розміри розділів встановлюються заздалегідь під час генерації системи, невеликі завдання призводять до неефективного використання пам'яті.

Фіксований розподіл нині практично не використовується. Прикладом ОС з використанням цієї технології може служити рання ОС IBM для мейнфреймів OS/MFT (Multiprogramming with a Fixed number of Tasks – багатозадачна з фіксованою кількістю задач).

10.2.2 Динамічний розподіл пам'яті

Для подолання складнощів, пов'язаних з фіксованим розподілом, був розроблений альтернативний підхід, відомий як динамічний розподіл.

При динамічному розподілі пам'яті без використання дискового простору пам'ять машини не ділиться заздалегідь на розділи. Спочатку вся пам'ять вільна. Кожній новій задачі виділяється необхідна їй пам'ять. Якщо достатній об'єм пам'яті відсутній, то задача не приймається на виконання і стоїть в черзі. Після завершення задачі пам'ять звільняється, і на це місце може бути завантажене інша задача. Таким чином, в довільний момент часу оперативна пам'ять є випадковою послідовністю

зайнятих і вільних ділянок (розділів) довільного розміру. На рис. 10.4 показаний стан пам'яті в різні моменти часу при використанні динамічного розподілу.

Так, у момент t0 в пам'яті знаходиться тільки ОС, а до моменту t1 пам'ять розділена між 5 задачами, причому задача П4, завершуючись, покидає пам'ять. На звільнене місце після задачі П4 завантажується задача П6, що поступила в момент t3.

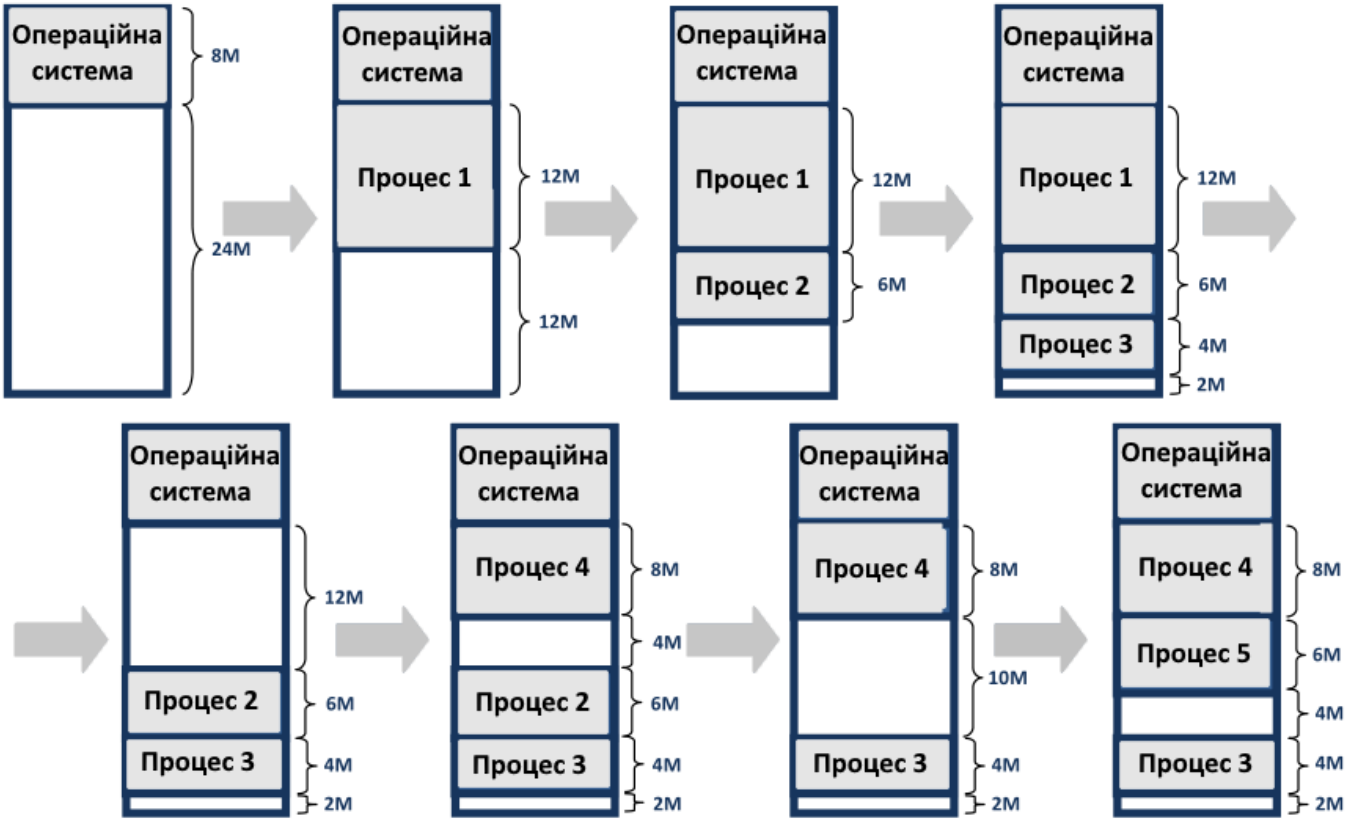

Рисунок 10.4 – Розподіл 32 Мб пам'яті динамічними розділами

Задачами ОС при реалізації цього методу управління пам'яттю є:

– ведення таблиць вільних і зайнятих областей, в яких указуються початкові адреси і розміри ділянок пам'яті;

– при надходженні нової задачі ОС аналізує запит, переглядає таблиці вільних областей і вибирає розділ, розмір якого достатній для розміщення нової задачі;

– завантаження задачі у виділений їй розділ і коригування таблиць вільних і зайнятих областей;

– після завершення задачі ОС коригує таблиці вільних і зайнятих областей. Програмний код не переміщається під час виконання, тобто може бути проведене

одноразове налаштування адрес за допомогою використання переміщаючого завантажувача.

Вибір розділу для нової задачі може здійснюватися за різними правилами, такими, наприклад, як «перший розділ достатнього розміру», що попався, або «розділ, що має найменший достатній розмір». Усі ці правила мають свої переваги і недоліки.

У порівнянні з методом розподілу пам'яті фіксованими розділами цей метод має набагато більшу гнучкість, але йому властивий дуже серйозний недолік – зовнішня фрагментація пам'яті. Зовнішня фрагментація – це наявність великого числа несуміжних ділянок вільної пам'яті дуже маленького розміру (фрагментів). Настільки маленького, що жодна з програм, яка знову поступає, не може поміститися ні в одній з ділянок, хоча сумарний об'єм фрагментів може скласти значну величину, що набагато перевищує необхідний об'єм пам'яті.

Свого часу динамічний розподіл використала ОС IBM для мейнфреймів OS/MVT (Multiprogramming with a Variable number of Tasks – багатозадачна зі змінною кількістю задач). Пізніше цей же підхід до розподілу пам'яті був використаний в ОС ЄС ЕОМ.

10.2.3 Переміщувані розділи

Одним з методів боротьби із зовнішньою фрагментацією є переміщення усіх зайнятих ділянок у бік старших або у бік молодших адрес, так, щоб уся вільна пам'ять утворювала єдину вільну область (рис. 10.5).

На додаток до функцій, які виконує ОС при розподілі пам'яті змінними розділами, в даному випадку вона повинна ще час від часу копіювати вміст розділів з одного місця пам'яті в інше, коригуючи таблиці вільних і зайнятих областей. Ця процедура називається «ущільненням» або «стискуванням».

Перелічимо функції операційної системи з управління пам'яттю в цьому випадку:

1. Переміщення всіх зайнятих ділянок у бік старших або молодших адрес при кожному завершенні процесу або для новоствореного процесу в разі відсутності розділу достатнього розміру.

2. Корекція таблиць вільних і зайнятих областей.

3. Зміна адрес команд і даних, до яких звертаються процеси при їх переміщенні в пам'яті за рахунок використання відносної адресації.

4. Апаратна підтримка процесу динамічного перетворення відносних адрес в абсолютні адреси основної пам'яті.

5. Захист пам'яті, що виділяється процесу, від взаємного впливу інших процесів.

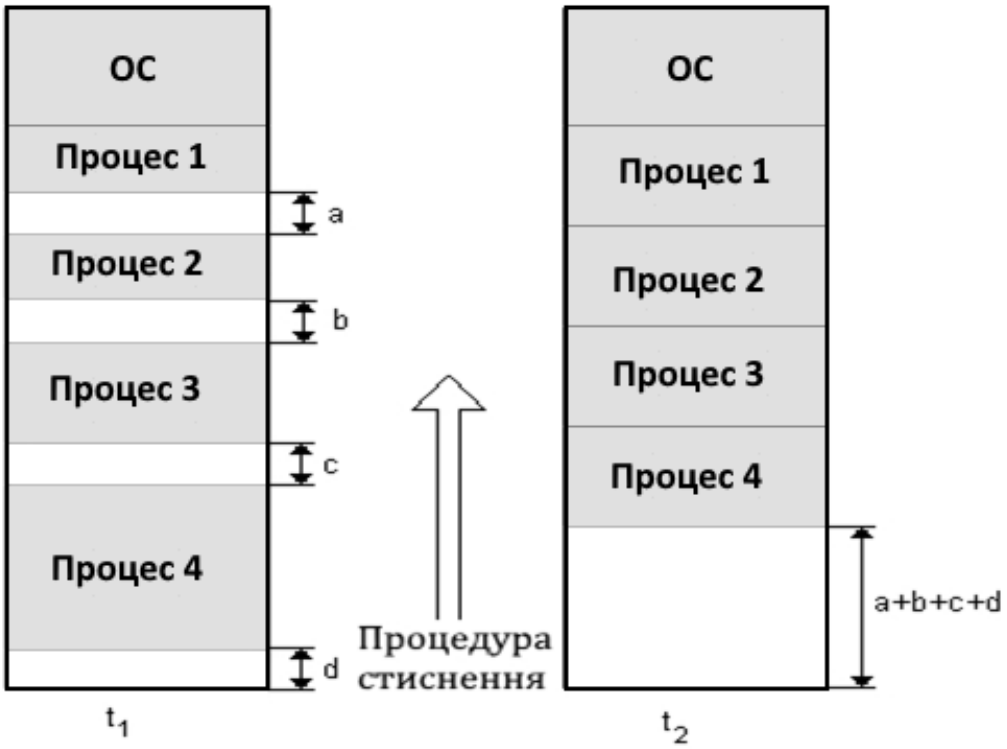

Рисунок 10.5 – Розподіл пам'яті переміщуваними розділами

Стискування (ущільнення) може виконуватися або при кожному завершенні задачі, або тільки тоді, коли для задачі, яка знову поступила, немає вільного розділу достатнього розміру. У першому випадку потрібно менше обчислювальної роботи при коригуванні таблиць, а в другому – рідше виконується процедура стискування. Оскільки програми переміщаються по оперативній пам'яті в ході свого виконання, то перетворення адрес з віртуальної форми у фізичну повинне виконуватися динамічним способом.

Перевагами розподілу пам'яті переміщуваними розділами є ефективне використання оперативної пам'яті, виключення внутрішньої і зовнішньої фрагментації.

Хоча процедура стискування і призводить до ефективнішого використання пам'яті, вона може зажадати значного часу, що часто знижує переваги цього методу.

Ситуація ускладнюється, якщо процеси по ходу роботи можуть займати різне місце розташування в розділах. У цих випадках розташування команд і даних, до яких звертається процес, не є фіксованим і змінюється всякий раз при вивантаженні, завантаженні або переміщенні процесу. Для вирішення цієї проблеми в програмах використовуються відносні адреси. Це означає, що всі посилання на пам'ять у завантажуваному процесі даються відносно початку цієї програми. Таким чином, для коректної роботи програми потрібен апаратний механізм, який би транслював відносні адреси у фізичні в процесі виконання команди, яка звертається до пам'яті. Один із способів трансляції показаний на рис. 10.6.

Коли процес переходить у стан виконання, то в спеціальний базовий регістр процесу завантажується початкова адреса процесу в основній пам'яті. Крім того, використовується «граничний» (bounds) регістр, в якому міститься адреса останньої комірки програми. Ці значення заносяться в регістри при завантаженні програми в основну пам'ять.

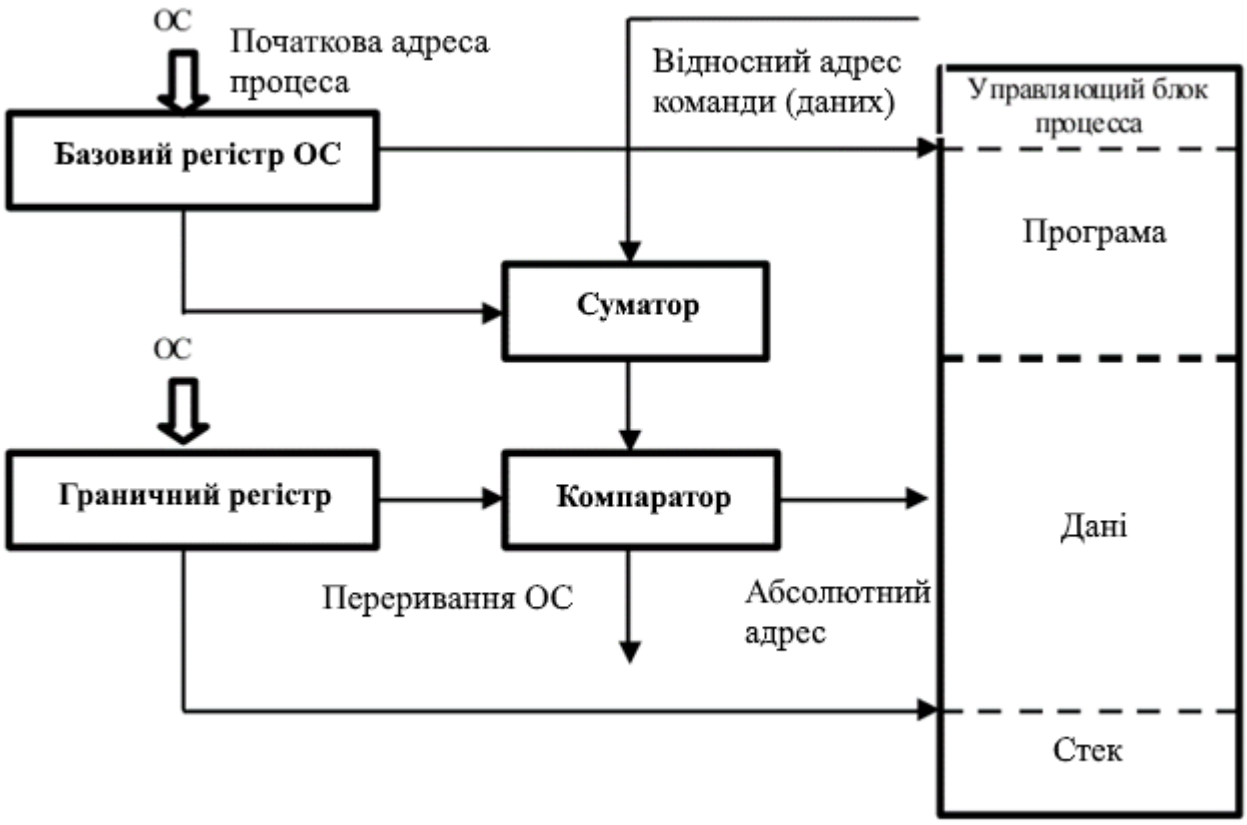

Рисунок 10.6 – Перетворення адрес

При виконанні процесу відносні адреси в командах обробляються процесором в два етапи. Спочатку до відносної адреси додається значення базового регістра для

отримання абсолютної адреси. Потім отримана абсолютна адреса порівнюється зі значенням в граничному регістрі. Якщо отримана абсолютна адреса належить цьому процесу, то команда може бути виконана. Інакше генерується переривання, що відповідає цій помилці.

10.2.4 Система двійників

Як фіксований, так і динамічний розподіл пам'яті мають переваги і недоліки. Компромісним у цьому плані є система двійників, в якій пам'ять розподіляється на блоки розміром $\mathbf{2^K}$**, L <= K <= U**, де: – $\mathbf{2^L}$ – мінімальний розмір блоку пам'яті; **2U** – найбільший розмір блоку пам'яті (спочатку вся доступна пам'ять).

При запиті пам'яті розміром **S**, таким, що **2U-1 < S <= 2U**, виділяється увесь блок. Інакше блок розділяється на два рівних двійники з розмірами **2U-1**. Якщо **2U-2 < S <= 2U-1**, то за запитом виділяється один з двох двійників; інакше один з двійників знову ділиться навпіл.

Цей процес триває до тих пір, поки не буде згенерований найменший блок, розмір якого не менше **S**. Система двійників постійно веде список «дір» (доступних блоків) для кожного розміру **2i**. Діра може бути видалена зі списку (i+1) розділенням її навпіл і внесенням двох нових дір розміру **2i** в список i. Коли пара двійників в списку і виявляється звільненою, вони видаляються зі списку і об'єднуються в єдиний блок в списку i+1.

Нижче наведено приклад (рис. 10.7) використання блоку з початковим розміром 1 Мб. Перший запит A – на 100 Кб (для нього потрібен блок розміром 128 Кб).

Початковий блок	**1Мб**							
Запит A=100K	**A128**	128K		256K		512K		
Запит B=240K	**A128**	128K		**B256**		512K		
Запит C=63K	**A128**	**C64**	64K	**B256**		512K		
Запит D=256K	**A128**	**C64**	64K	**B256**		**D256**		256K
Звільнення B	**A128**	**C64**	64K	256K		**D256**		256K
Звільнення A	128K	**C64**	64K	256K		**D256**		256K
Запит E=75K	**E128**	**C64**	64K	256K		**D256**		256K
Звільнення C	**E128**	128K		256K		**D256**		256K
Звільнення E	512K					**D256**		256K
Звільнення D	**1Мб**							

Рисунок 10.7 – Приклад роботи системи двійників

Для цього початковий блок ділиться на два двійники по 512 Кб. Потім перший з них ділиться на два двійники розміром 256 Кб, і, у свою чергу, перший двійник, який вийшов при цьому поділі, також ділиться навпіл (128 Кб). Один із двійників розміром 128 Кб виділяється процесу А. Наступний запит В вимагає 240 Кб. Такий блок є в наявності і виділяється. Процес триває з поділом і злиттям двійників при необхідності. Зверніть увагу, що після звільнення блоку Е відбувається злиття двійників по 128 Кб в один блок розміром 256 Кб, який, у свою чергу, тут же зливається зі своїм двійником.

Система двійників є розумним компромісом для подолання недоліків схем фіксованого і динамічного розподілу, але в сучасних операційних системах її перевершує віртуальна пам'ять, заснована на сторінковій організації і сегментації. Однак система двійників знайшла застосування в сучасних ОС як ефективний засіб розподілу і звільнення паралельних програм. Модифікована версія системи двійників використовується для розподілу пам'яті ядром UNIX.