ПРАКТИЧНА РОБОТА 05

Керування пам’яттю в операційних системах

План

1. Виділення пам’яті для процесів.
2. Механізм підкачування сторінок.
3. Практичні завдання.

Концепція віртуальної пам’яті допомагає у випадку, коли розмір процесу значно перевищує розмір основної пам’яті, тому неможливо повністю завантажити процес в оперативну пам’ять. У результаті завантажується частина процесу, яка працює до свого завершення, а потім вона вивантажується та заміщається іншою частиною. Основна пам’ять розбивається на фрейми, й кожному процесу (P1, P2, P3, P4, P5 на рис. 1) виділяється відповідний фрейм.

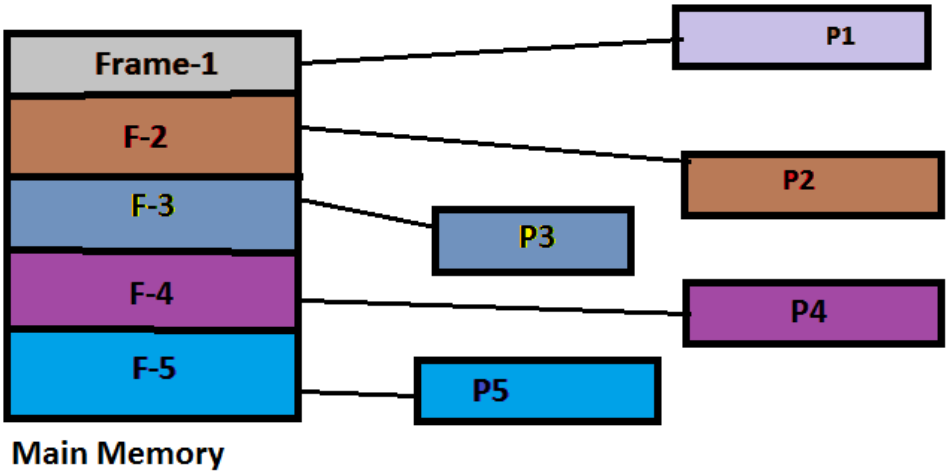


Рис. 1. Розподіл процесів та фреймів у пам’яті

Реалізація віртуальної пам’яті здійснюється на основі ***підкачування сторінок за вимогою (demand paging)***. Підкачування сторінок – техніка управління пам’яттю, яка усуває потребу в неперервному виділенні фізичної пам’яті. Дана схема дозволяє мати розриви в фізичному адресному просторі процесу. Для здійснення операцій у контексті даної технології потрібні ***алгоритм виділення фреймів (frame allocation algorithm)*** та ***алгоритм заміщення сторінок (page-replacement algorithm)***. Алгоритми виділення фреймів використовуються за умови наявності багатьох процесів та допомагають вирішувати, скільки фреймів віддавати кожному процесу. Кілька процесів мають спільну основну пам’ять, тому важливо керувати процесами, щоб кожен з них отримав свою частину пам’яті. Для виділення (алокації) сторінок у пам’яті зазвичай пропонують один з двох алгоритмів, які суттєво впливають на продуктивність системи. Їх погана реалізація може призводити до багатьох проблем, на зразок ***пробуксовки (thrashing)***.

В операційних системах, які використовують підкачування сторінок для управління пам’яттю ***алгоритм заміщення сторінок*** визначає, яку сторінку слід замінити на новоприбувшу. ***Збій сторінки (page fault)*** відбувається, коли запущена програма отримує доступ до сторінки пам’яті, яка відображена у віртуальний адресний простір, проте не завантажена у фізичну пам’ять.

Оскільки реальна фізична пам’ять набагато менша, ніж віртуальна пам’ять, трапляються збої сторінки. У такому випадку ОС може замістити одну існуючу сторінку на нову, потрібну в цей момент. Різні алгоритми заміщення передбачають різні способи вибору, яку сторінку заміщати. Метою усіх цих алгоритмів є зменшення кількості збійних сторінок.

Введемо декілька понять:

- Логічна (віртуальна) адреса – представлена в бітах адреса пам’яті, згенерована центральним процесором;
- Логічний (віртуальний) адресний простір – набір усіх логічних адрес, згенерованих програмою, представлений у байтах або словах;
- Фізична адреса – реально доступна адреса в мікросхемі пам’яті, яка представляється в бітах;
- Фізичний адресний простір – набір усіх фізичних адрес, які відповідають логічним адресам, представлений у байтах або словах.

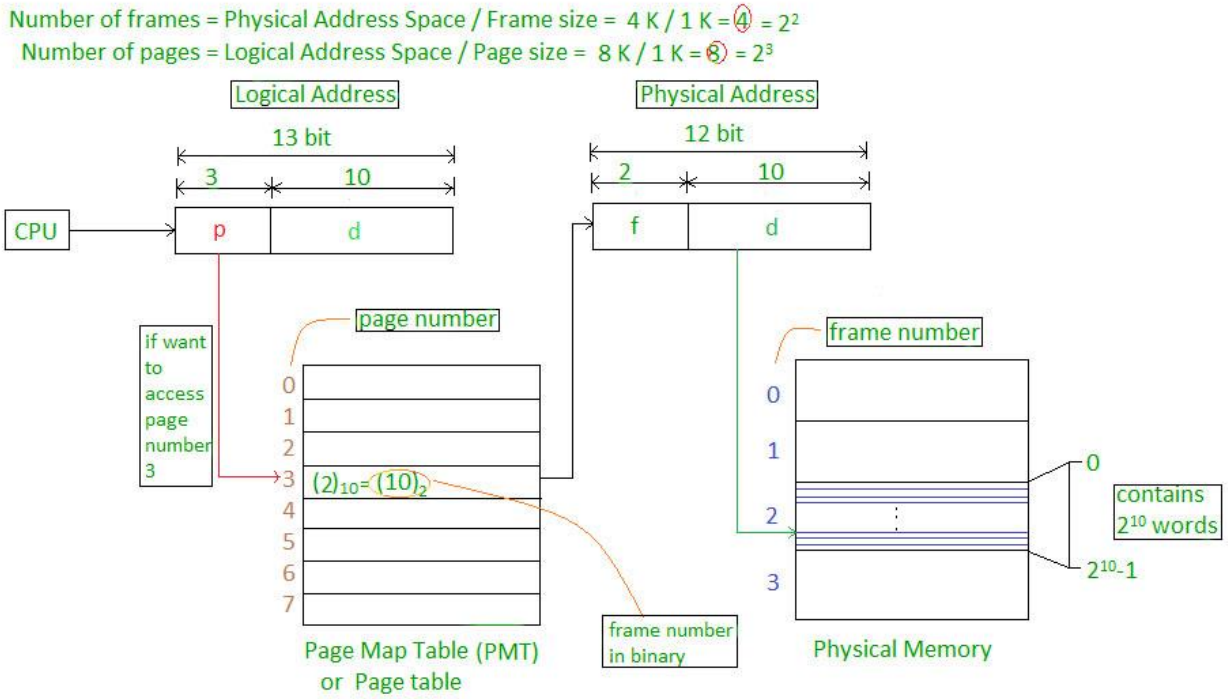
Якщо логічна адреса має 31 біт, то логічний адресний простір - 2^{31} слів, тобто 2Гб слів. Якщо логічний адресний простір має розмір 128Мб слів (2^{27} слів), то логічна адреса складається з $\log_2 2^{27} = 27$ бітів.

Якщо фізична адреса має 22 біти, то фізичний адресний простір містить 2^{22}=4Мб слів. Для фізичного адресного простору розміром 16Мб фізична адреса записується з 24 бітів.

Відображенням віртуальної адреси на фізичну займається пристрій управління пам’яттю (memory management unit, MMU), який є апаратним пристроєм. Техніку відображення адрес називають ***підкачуванням сторінок (paging)***.

Фізичний адресний простір концептуально розбивається на багато блоків фіксованого розміру – ***фреймів***. Логічний адресний простір розбивається на блоки фіксованого розміру – ***сторінки (pages)***. Розмір сторінки дорівнює розміру фрейму.

Приклад. Нехай фізична адреса складається з 12 бітів (фізичний адресний простір має 4Кб слів). Логічний адресний простір містить 8Кб слів, тобто 13 бітів для логічної адреси. Припустимо, що розмір сторінки (і фрейму) дорівнює 1Кб слів.

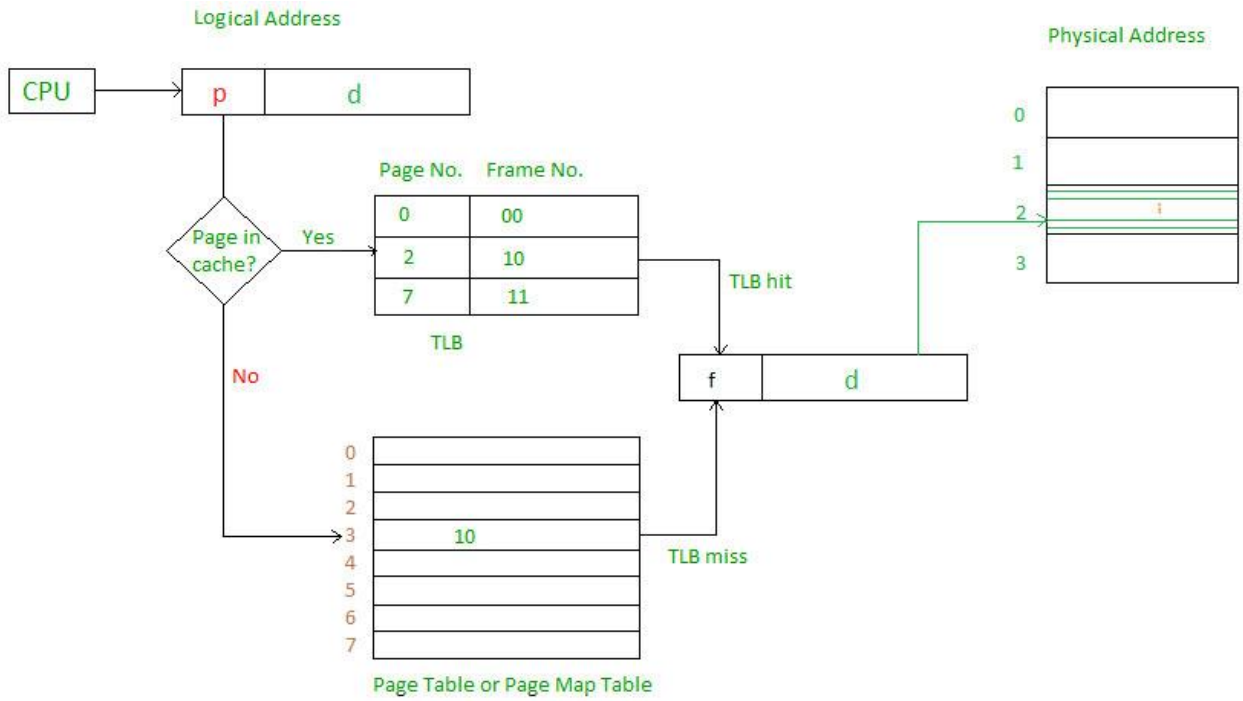


Згенерована ЦП адреса ділиться на:

- ***Номер сторінки (p)*** – кількість бітів, потрібних для представлення сторінок у логічному адресному просторі;
- ***Зміщення (offset) сторінки (d)*** – кількість бітів, необхідних для представлення конкретного слова в сторінці, або розмір сторінки логічного адресного простору, або кількість слів у сторінці.

У фізичні пам'яті аналогічні параметри – ***номер фрейму (frame number, f)*** та з***міщення фрейму (frame offset, d)***.

Апаратна реалізація таблиці сторінок (page table) може здійснюватись за допомогою виділених регістрів, проте використання регістру добре себе рекомендує тільки з малою таблицею. Якщо в таблиці багато входжень, можна використовувати translation Look-aside buffer (TLB) – спеціальний малий апаратний кеш для швидкого пошуку (look up). TLB-кеш – асоціативна, високошвидкісна пам'ять. Кожен запис у TLB складається з двох частин: тегу та значення. Коли дана пам'ять використовується, елемент одночасно порівнюється з усім тегами. Якщо елемент (item) знайдено, повертається відповідне значення.



Для часу доступу до основної пам'яті m можна обчислити ефективний час доступу до таблиці сторінок, яка знаходиться в цій пам'яті:

$$\text{Ефективний час доступу} = m_{\text{таблиці}} + m_{\text{сторінки в таблиці}}$$

За наявності TLB-кешу з часом доступу c та ймовірністю влучання (hit ratio) x (ймовірністю промаху $1 - x$)

Ефективний час доступу

$$= x(c + m_{\text{основної пам'яті}}) + (1 - x)(c + m_{\text{основної пам'яті}} + m_{\text{таблиці}})$$

Існують різні обмеження щодо стратегій виділення фреймів:

- Неможливо виділити більше процесів, ніж наявна кількість фреймів;
- Кожному процесу потрібно виділити принаймні мінімальну кількість фреймів. Чим менше фреймів виділяється, тим вища ймовірність збою сторінки та відповідного зниження продуктивності виконання процесу. Крім того, бажано мати достатню кількість фреймів для зберігання усіляких різних сторінок, на які може посилатись інструкція.

Широко застосовуються два алгоритми для виділення фреймів для процесів:

- ***Рівномірне (equal) виділення.*** У системі з x фреймами та y процесами кожен процес отримує однакову кількість фреймів x/y. Наприклад, якщо система має 48 фреймів та 9 процесів, кожен процес отримає по 5 фреймів. Решта три невиділених фрейми можна використовувати в якості буферу (free-frame buffer pool). Недоліком такого підходу є недоречність подібного розподілу в системах з процесами різного

розміру. Виділення багатьох фреймів маленьким процесам призведе до даремних витрат пам'яті;

– ***Пропорційне виділення.*** Фрейми виділяються кожному процесу відповідно до розміру цього процесу. Для процесу p_i розміром s_i кількість виділених фреймів буде $a_i = \frac{s_i}{S} m$, де S – сума розмірів усіх процесів, а m – кількість фреймів у системі. Наприклад, у системі з 62 фреймами для 2 процесів розмірами 10Кб та 127Кб відповідно буде виділено (10/137)*62 = 4 фрейми та (127/137)*62 = 57 фреймів.

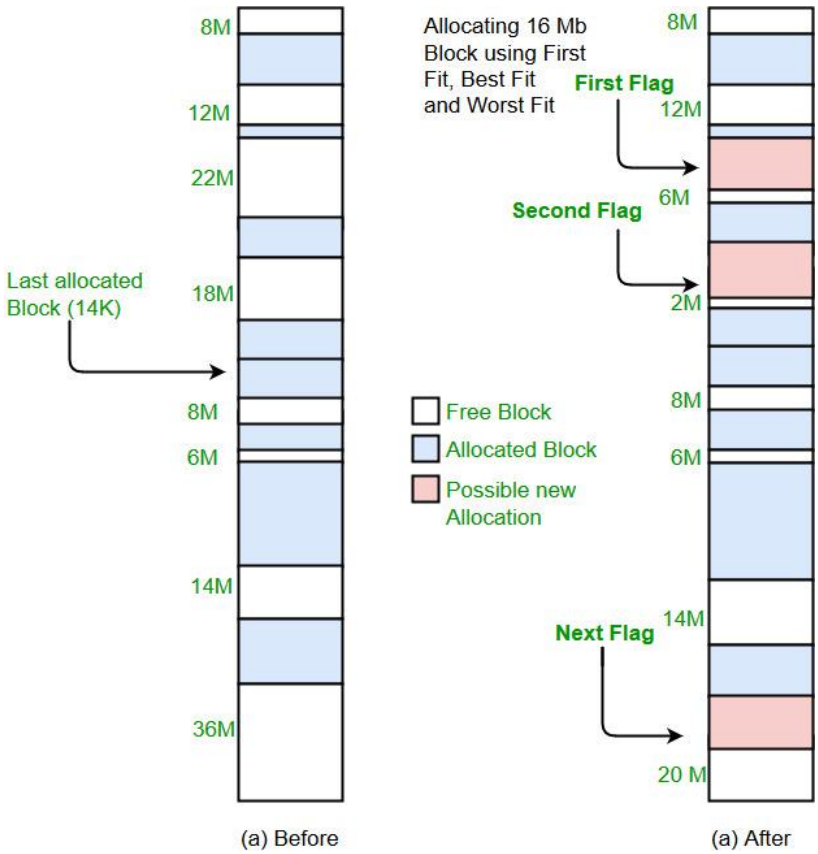
Як для фіксованої, так і для динамічної схем виділення пам'яті ОС повинна слідкувати за кожним місцем у пам'яті, сповіщаючи, які з них зайняті, а які – вільні. Потім при надходженні нових робіт у систему вільні розділи (partitions) пам'яті повинні виділятись. Неперервне (Contiguous) виділення пам'яті передбачає 4 можливих способи такого виділення:

1. First-Fit Memory Allocation

Даний метод тримає перелік вільних/зайнятих робіт, впорядкований за місцем у пам'яті. За умови використання даного алгоритму перша робота вимагає першого доступного розділу пам'яті з розміром >= розміру роботи. ОС не шукає доречний розділ, а лише виділяє найближчий розділ пам'яті достатнього розміру.

Приклад:

```
Input : blockSize[]   = {100, 500, 200, 300, 600};
        processSize[] = {212, 417, 112, 426};
Output:
Process No.    Process Size    Block no.
   1               212            2
   2               417            5
   3               112            2
   4               426        Not Allocated
```



Як продемонстровано на рисунку нижче, система видає J1 найближчий розділ пам’яті. У результаті цього немає розділу достатнього розміру для роботи J3, тому вона ставиться в чергу очікування.

Такий метод швидкий у роботі, оскільки процесор виділяє найближчий можливий розділ для роботи. Проте так втрачається багато пам’яті, оскільки процесор ігнорує, наскільки розмір виділеного розділу перевищує розмір роботи. Таким чином, багато пам’яті даремно витрачається, а багато робіт можуть не отримати простору в пам’яті та очікувати на завершення іншої роботи.

Job Number	Memory Requested
J1	20 K
J2	200 K
J3	500 K
J4	50 K

Memory location	Memory block size	Job number	Job size	Status	Internal fragmentation
10567	200 K	J1	20 K	Busy	180 K
30457	30 K			Free	30
300875	700 K	J2	200 K	Busy	500 K
809567	50 K	J4	50 K	Busy	None
Total available :	980 K	Total used :	270 K		710 K

2. *Best-Fit Memory Allocation*

Даний підхід веде перелік вільних/зайнятих робіт, впорядкований за розміром – від найменшої до найбільшої. У цьому методі ОС спочатку виконує пошук в усій пам’яті відповідно до розміру заданої роботи та виділяє розділ з найближчим до нього розміром. Це дозволяє ефективно використовувати пам’ять.

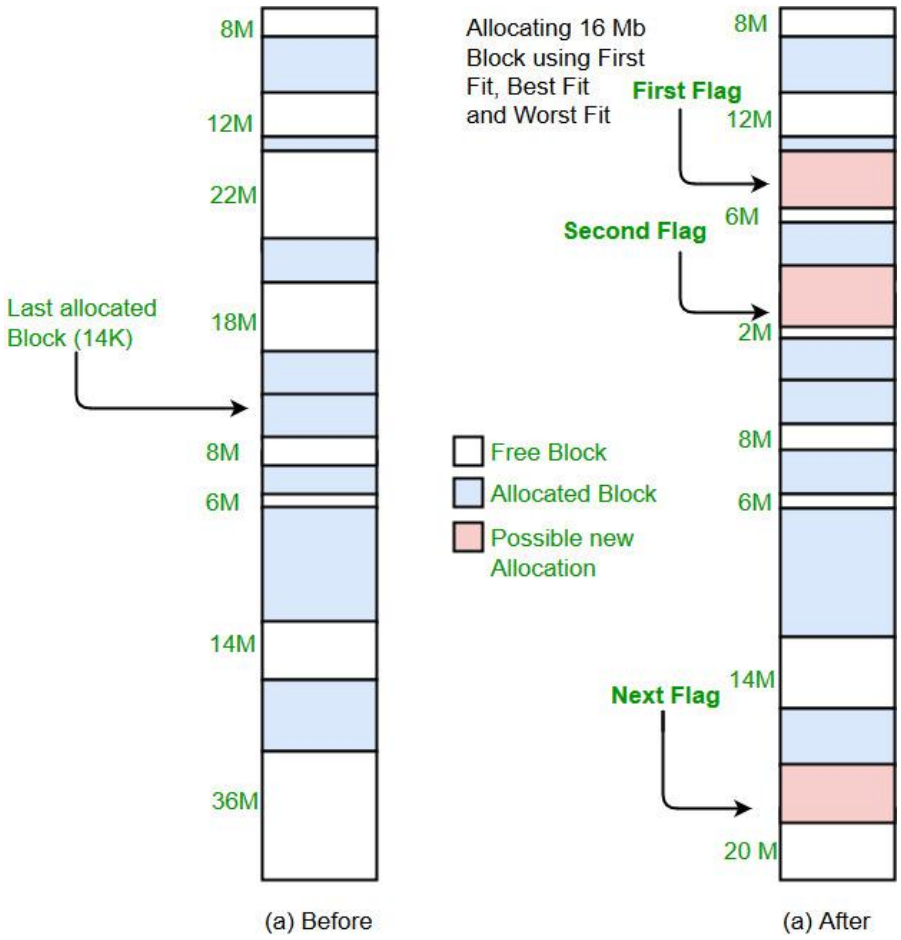
Job Number	Memory Requested
J1	20 K
J2	200 K
J3	500 K
J4	50 K

Memory location	Memory block size	Job number	Job size	Status	Internal fragmentation
10567	30 K	J1	20 K	Busy	10 K
30457	50 K	J4	50 K	Busy	None
300875	200 K	J2	200 K	Busy	None
809567	700 K	J3	500 K	Busy	200 K
Total available :	980 K	Total used :	770 K		210 K

На рисунку проілюстровано наступне: ОС спочатку проводить пошук у пам’яті та виділяє роботу мінімально можливому розділу. Ефективність виділення пам’яті є основною перевагою даного алгоритму, проте сам процес виділення буде повільним: перевірка всієї пам’яті для кожної роботи значно сповільнює роботу ОС.

Приклад:

```
Input : blockSize[]   = {100, 500, 200, 300, 600};
        processSize[] = {212, 417, 112, 426};
Output:
Process No.    Process Size    Block no.
 1                  212          4
 2                  417          2
 3                  112          3
 4                  426          5
```



3. Worst-Fit Memory Allocation

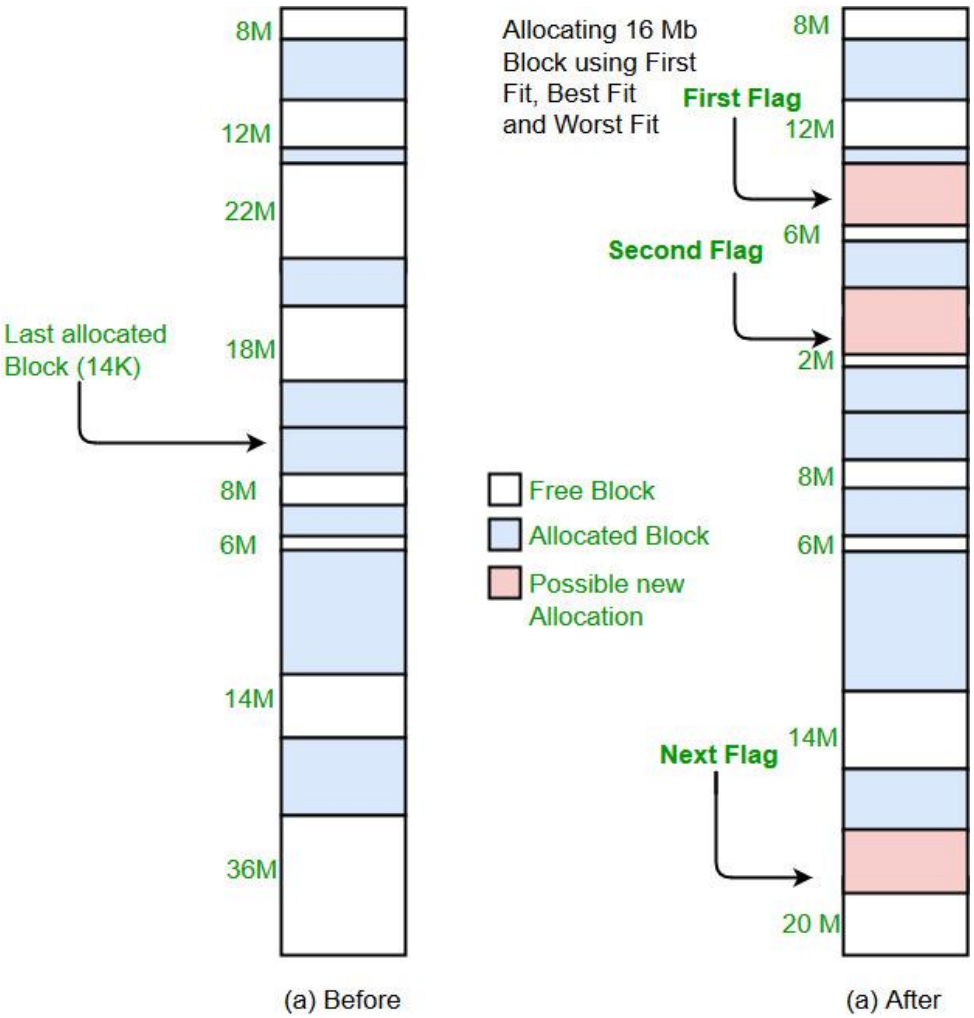
Виділяє процесу розділ пам’яті, найбільший серед доступних вільних розділів основної пам’яті. Якщо великий процес надходить на пізніх етапах виконання, тоді не буде простору для виділення пам’яті цьому процесу. Наприклад:

Ввід : blockSize[] = {100, 500, 200, 300, 600};

processSize[] = {212, 417, 112, 426};

Вивід:

Номер процесу	Розмір процесу	Номер блоку
1	212	5
2	417	2
3	112	5
4	426	Not Allocated



4. *Next-Fit Memory Allocation*

Модифікована версія алгоритму first-fit. Починається так же, як і first fit для знаходження вільного розділу, проте при наступному виклику починає пошук не з початку, а з попереднього місця. Дана політика застосовує бродячий вказівник (roving pointer). Він пересувається по ланцюгу з комірок пам’яті для пошуку наступного доречного розташування. Це допомагає уникнути постійного використання пам’яті на початку (з голови) ланцюга вільних блоків.

Це прямолінійний та швидкий алгоритм, проте він має тенденцію до розрізання великих блоків вільної пам’яті на маленькі частини, що призведе до неможливості виділення пам’яті для крупних процесів, навіть якщо сума всіх маленьких частин перевищує потребу. Дана проблема називається зовнішньою фрагментацією.

Алгоритм next-fit намагається вирішити проблему внутрішньої фрагментації – накопичення дрібних блоків на початку ланцюга вільних блоків у результаті роботи first-fit підходу. Також у порівнянні з First Fit та Best Fit алгоритм вважається дуже швидким. Приклад:

```
Ввід:  blockSize[] = {5, 10, 20};
     processSize[] = {10, 20, 30};
```

```
Вивід:
Process No.     Process Size    Block no.
 1              10              2
 2              20              3
 3              30              Not Allocated
```

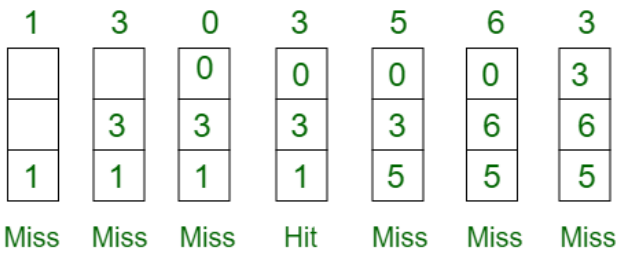
2. ***Механізм підкачування сторінок***

Кількість виділених фреймів для процесу також може динамічно змінюватись залежно від того, який алгоритм заміщення сторінок буде використовуватись:

1) ***Локальне заміщення (Local replacement).*** Коли процесу потрібна сторінка, яка не розміщена в пам'яті, він може внести нову сторінку та виділити їй фрейм тільки з власного набору виділених фреймів. Перевагою такого підходу є вплив поведінки підкачування сторінок лише на даний процес. Недолік: низькопріоритетний процес може заважати високопріоритетному процесу, не допускаючи, щоб виділені йому фрейми стали доступними процесу з вищим пріоритетом.
2) ***Глобальне заміщення (Global replacement).*** Коли процес потребує сторінку, яка не розміщена в пам'яті, він може внести нову сторінку та виділити під неї фрейм з усього набору, навіть якщо цей фрейм зараз виділений іншому процесу. Таким чином один процес може відбирати фрейм в іншого. Перевага: продуктивність процесів не порушується, тому система матиме кращу пропускну здатність. Недолік: ймовірність збою сторінки (page fault ratio) для процесу не може управлятись тільки одним процесом самостійно. Сторінки в пам'яті для процесу також залежать від поведінки їх підкачування інших процесів.

Найбільш поширеними є наступні алгоритми кешування (заміщення) сторінок:

- ***First In First Out (FIFO).*** Найпростіший алгоритм заміщення. ОС відстежує всі сторінки в пам'яті за допомогою черги: найстаріша сторінка знаходиться в голові черги. Коли сторінку потрібно замінити, на заміщення обирається саме сторінка з голови. Припустимо, що є послідовність посилань на сторінки (page reference string) 1, 3, 0, 3, 5, 6, 3. Фрейм відповідає 3 сторінкам. Знайдемо кількість збоїв.



Спочатку всі слоти порожні, тому коли надходять 1, 3, 0, вони алокуються в порожні слоти (3 збої). Далі надходить 3, воно вже є в пам'яті, тому маємо 0 збоїв. Далі – 5, яке заміщає 1 з голови черги (1 збій). Потім 6 заміщає трійку, а наступна 3 – нуль (+ 2 збої). Таким чином маємо 6 збоїв сторінки.

Аномалія Беладі доводить, що можливо мати більше збоїв сторінки, коли кількість фреймів на сторінку збільшується за умови використання FIFO-алгоритму заміщення сторінок. Наприклад, для послідовності посилань на сторінки 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4 та 3 слотів матимемо 9 збоїв, проте якщо збільшити кількість слотів до 4, отримаємо 10 збоїв.

– ***Оптимальне заміщення сторінки (OPT).*** Передбачає, що заміщуються сторінки, які не будуть використовуватись найдовше в майбутньому. Нехай послідовність посилань така: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, з чотирма сторінками, які відповідають фрейму. Знайдемо кількість збоїв.

На початку маємо 4 збої (7, 0, 1, 2). Наступний 0 не дає збій. Надходить 3 і заміщає 7 (1 збій), наступний 0 не дає збій, а 4 заміщає 1. Решта посилань вже доступні в пам'яті, тому збоїв робити не будуть. Таким чином, матимемо 6 збоїв.

Оптимальне заміщення сторінки ідеальне в теорії, проте неможливе на практиці, оскільки ОС не може знати майбутні звернення. Даний алгоритм використовується як бенчмарк для аналізу інших алгоритмів заміщення.

– ***Алгоритм LRU (Least Recently Used).*** Заміщає давно невикористовувані фрейми. Наприклад, для послідовності 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 та 4 сторінки на фрейм. Маємо 6 збоїв.

3. Практичні завдання

Змоделюйте систему роботи з віртуальною пам’яттю відповідно до параметрів, визначених згідно з варіантом:

Варіант	**К-ть адрес пам’яті**	**К-ть процесів/ зайняте місце**	**Первинне виділення фреймів**	**К-ть сторінок на фрейм**	**Алгоритм кешування**	**Виділення доступної пам’яті процесам**
1	512	10/360	Рівномірне	3	FIFO	Best-fit
2	256	8/128	Пропорційне	4	LRU	Worst-fit
3	448	12/320	Рівномірне	4	FIFO	First-fit
4	256	10/192	Пропорційне	3	LRU	Next-fit
5	512	8/320	Рівномірне	3	LRU	First-fit
6	512	12/256	Пропорційне	4	FIFO	Best-fit
7	640	10/512	Рівномірне	3	LRU	Worst-fit
8	448	8/192	Пропорційне	4	FIFO	Next-fit

Принципи побудови подібної системи можна розглянути тут.