Тема 5

Пам'ять

План

1. Фізичні та віртуальні адреси
2. Підхід базового і межового регістрів
3. Сегментна організація пам'яті
4. Сторінкова організація пам'яті
5. Сторінково-сегментна організація пам'яті
6. Підкачування. Заміщення сторінок

1. Фізичні та віртуальні адреси

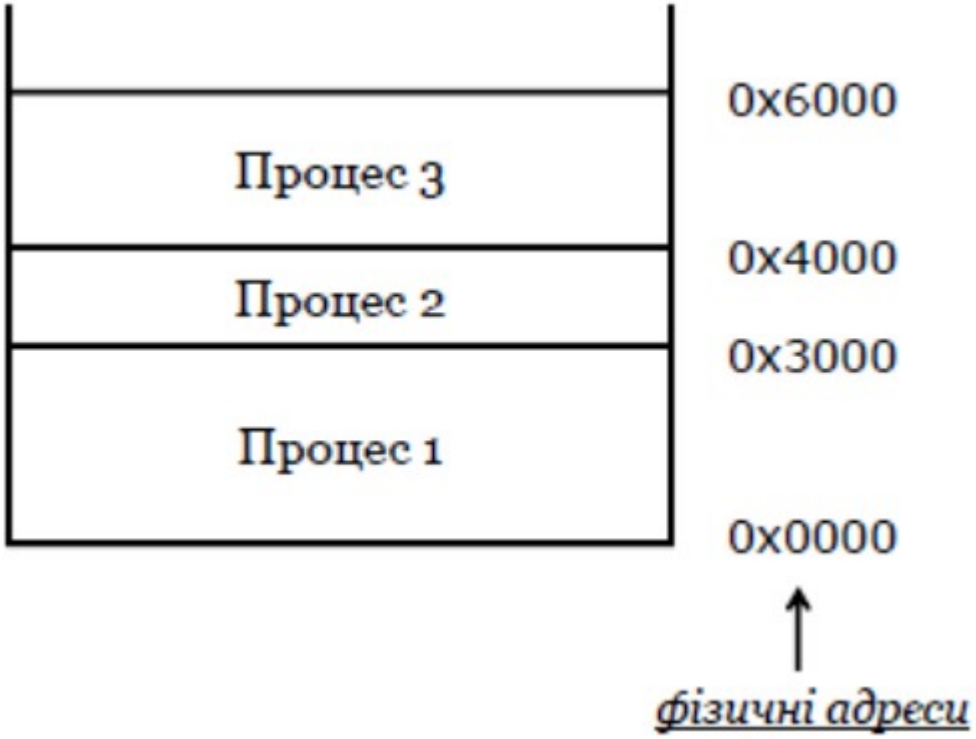
Надалі у цій темі говоримо передусім про **основну пам'ять** (тобто **оперативну пам'ять**).

Основне питання: як взаємно розташувати адресні простори процесів в оперативній пам'яті?

Схему адаптовано з книги Tananbaum, Bos. Modern Operating Systems, 2014

1. Фізичні та віртуальні адреси

Найпростіше рішення: адресні простори процесів послідовно розміщуються у пам'яті один за одним:



Фізична адреса - це реальна адреса, за якою у пам'яті зберігаються дані. Визначається на рівні мікросхеми пам'яті.

1. Фізичні та віртуальні адреси

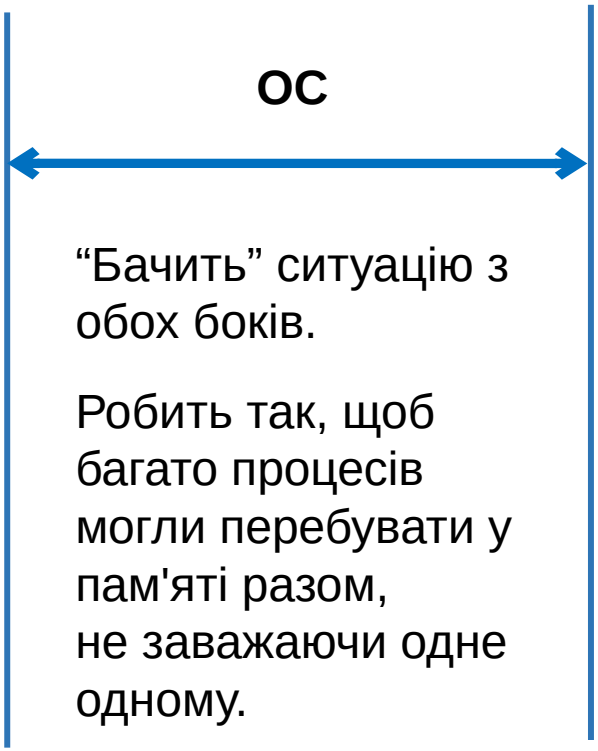
Навіщо потрібні логічні адреси

- Процесу знадобилася додаткова пам'ять (на додачу до тієї, яку вже було виділено йому під час створення).
- Виділеної пам'яті процесу забагато (іншим процесам може тим часом не вистачати пам'яті).
- Процесу потрібний обсяг пам'яті, що перевищує обсяг основної пам'яті на даному комп'ютері.
- Процес помилково звернувся до “чужої” адреси (за даного підходу не вдасться перешкодити йому зробити це).
- Коли кільком процесам треба спільно використовувати окремі блоки пам'яті (наприклад, туди завантажено бібліотеки) та ін.
- Було б добре відокремити програмний код від даних.

1. Фізичні та віртуальні адреси

Пам'ять

з точки зору процесів, апаратного забезпечення та ОС



1. Фізичні та віртуальні адреси

Суть віртуальної пам'яті:

фізичним адресам ставлять у відповідність віртуальні (логічні) адреси.

Віртуальна (логічна) адреса **-** умовна адреса у пам'яті, що у процесі роботи перетворюється у фізичну адресу.

Перетворення логічних адрес у фізичні здійснює **MMU** (*memory management unit*, **блок керування пам'яттю**, пристрій керування пам'яттю).

MMU є *апаратним* компонентом, він найчастіше входить до складу ЦП або чипсета.

Сукупність усіх віртуальних адрес = **віртуальний адресний простір**

Сукупність усіх фізичних адрес = **фізичний адресний простір**

Далі - про різні підходи до організації пам'яті.

2. Підхід базового та межового регістрів

Найпростіший підхід до організації пам'яті. Вже не є популярним, проте лежить в основі інших підходів.

Кожному процесу відповідають:

- **база** (фізична адреса, з якої починається адресний простір процесу);
- **межа** (кількість адрес, відведених процесу).

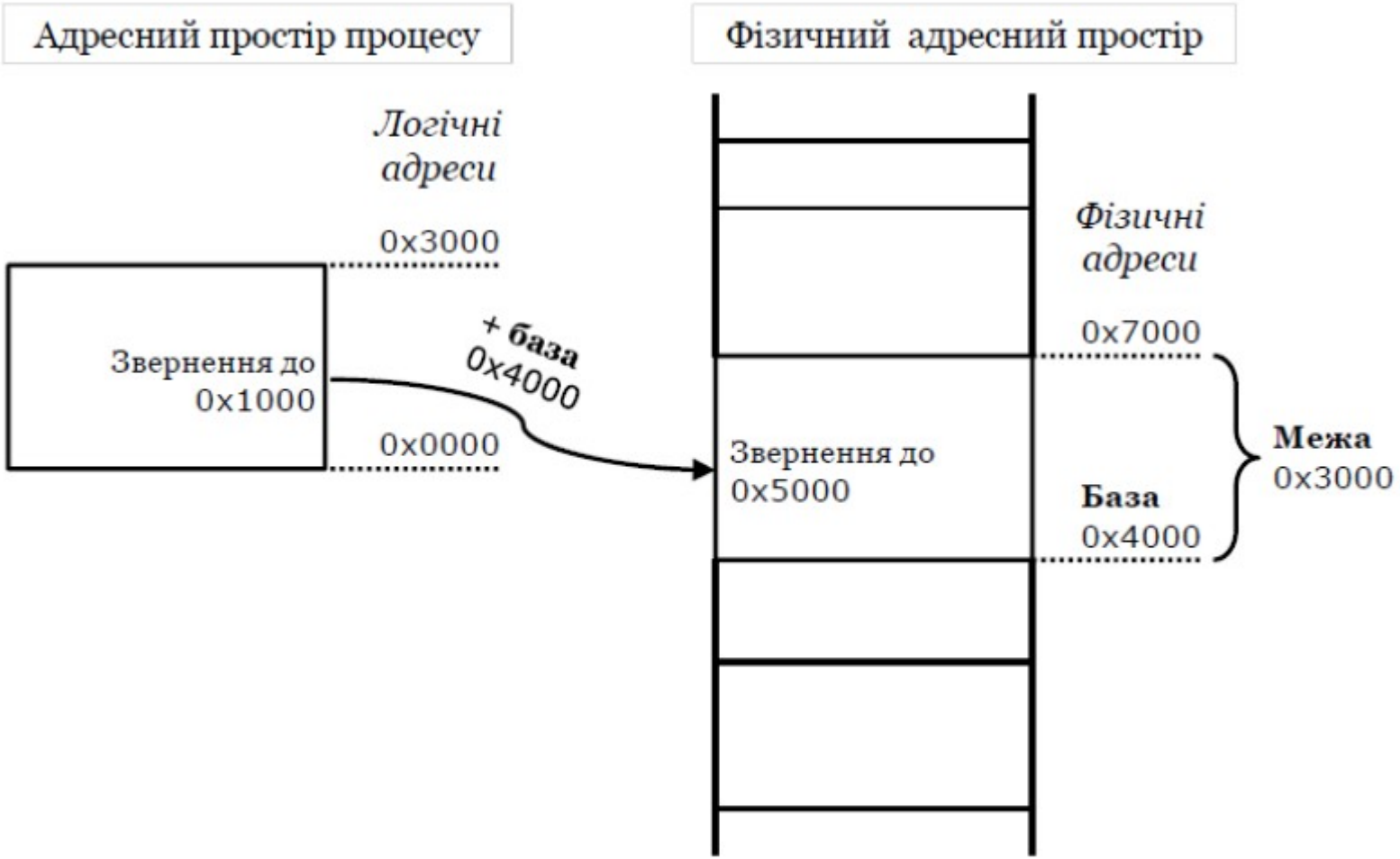
Для зберігання бази і межі відводяться два спеціальні регістри ЦП - ***базовий і межовий регістри***.

Маючи логічну адресу і значення бази, можна обчислити фізичну адресу:

$$\text{Фізична адреса} = \text{Логічна адреса} + \text{База}$$

2. Підхід базового та межового регістрів

Визначення адреси (підхід базового і межового регістрів)



Схему взято з книги: *В. А. Шеховцов* “Операційні системи” (2005)

2. Підхід базового та межового регістрів

Переваги підходу:

- простота реалізації

Недоліки підходу:

- неможливість розширення адресного простору процесу під час його роботи
- неможливість спільного використання адресного простору кількома процесами
- програмний код не відокремлений від даних

3. Сегментна організація пам'яті

Згідно з підходом сегментної організації пам'яті, віртуальний адресний простір ділять на ***сегменти***.

Сегменти нумеруються і можуть мати різний розмір.

Логічна адреса задається:

- номером сегмента
- зсувом усередині сегмента

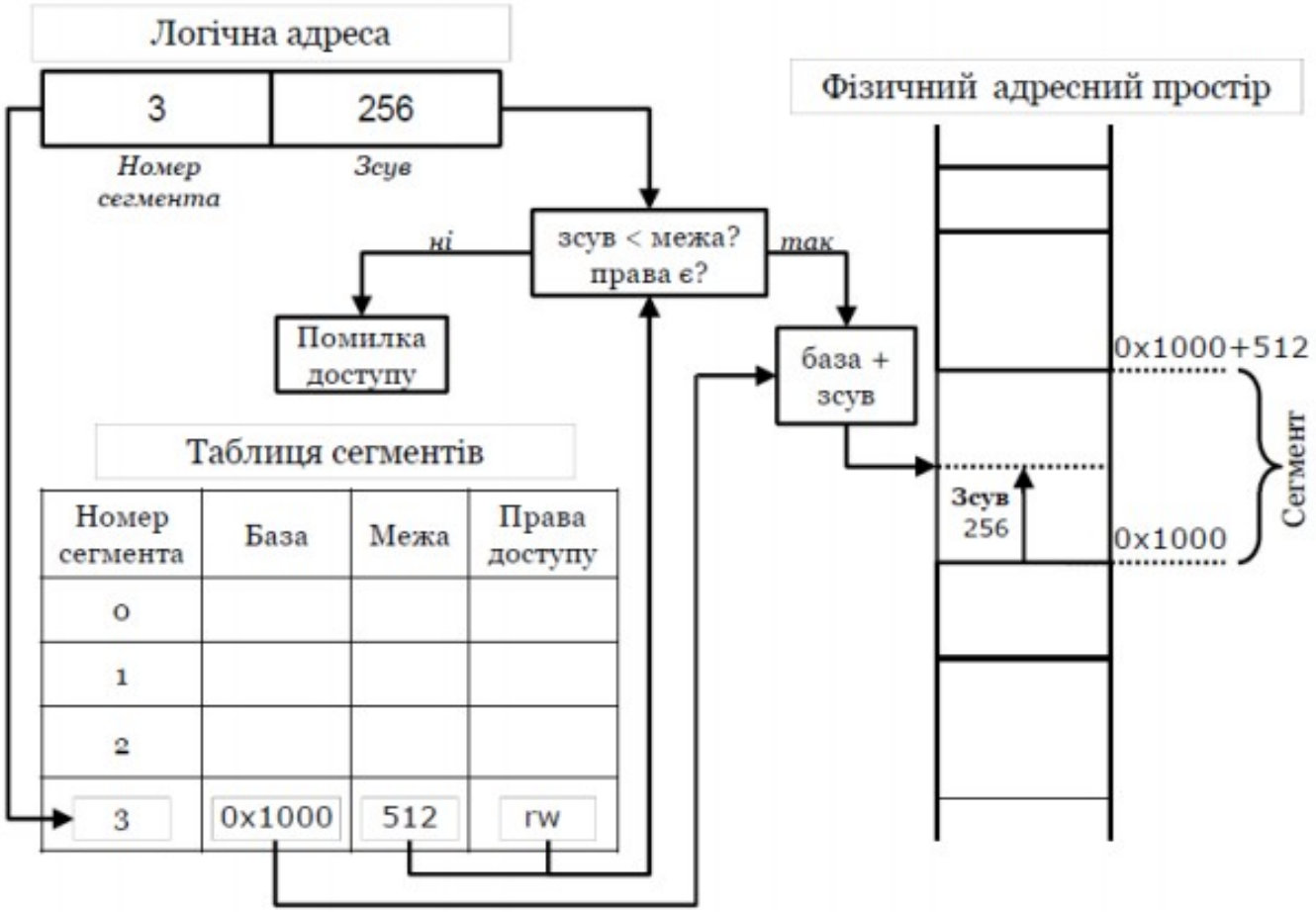
Відомості про всі сегменти процесу зберігаються у таблиці сегментів. Також може бути глобальна таблиця, яка зберігає інформацію про всі сегменти всіх процесів.

Основні **відомості про сегмент** у таблиці:

- *номер* сегмента
- *база* (фізична адреса, з якої починається сегмент)
- *межа* (кількість адрес, відведених під сегмент)
- *права доступу* до сегменту

3. Сегментна організація пам'яті

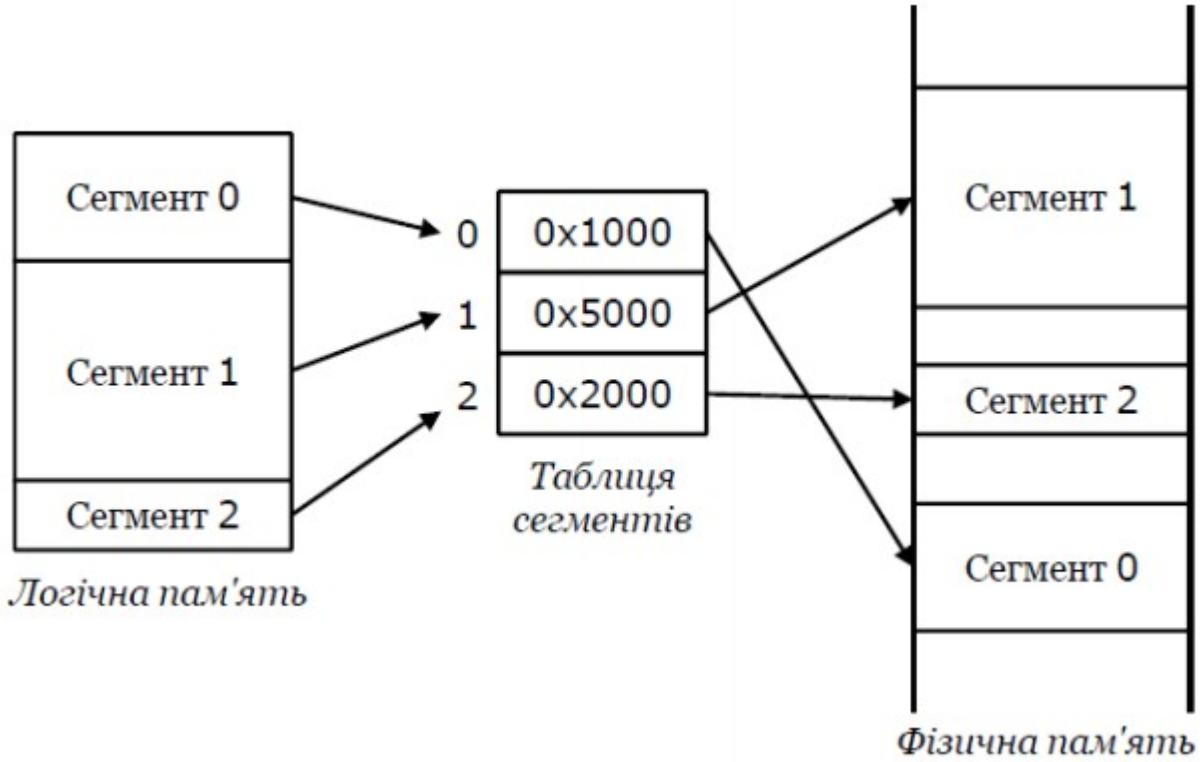
Визначення адреси (сегментна організація пам'яті)



Схему взято з книги: *В. А. Шеховцов* “Операційні системи” (2005)

3. Сегментна організація пам'яті

Співвідношення віртуального і фізичного адресного простору (сегментна організація пам'яті)



Схему взято з книги: *В. А. Шеховцов* “Операційні системи” (2005)

3. Сегментна організація пам'яті

Переваги підходу (порівняно з підходом базового і межового регістрів)**:**

- кілька процесів можуть спільно використовувати деякий сегмент
- можна зберігати програмний код і дані процесу в окремих сегментах
- частину сегментів можна тимчасово вивантажувати на диск

Недоліки підходу:

- обмінюватися сегментами між диском та основною пам'яттю не дуже зручно через різний розмір сегментів
- зовнішня фрагментація

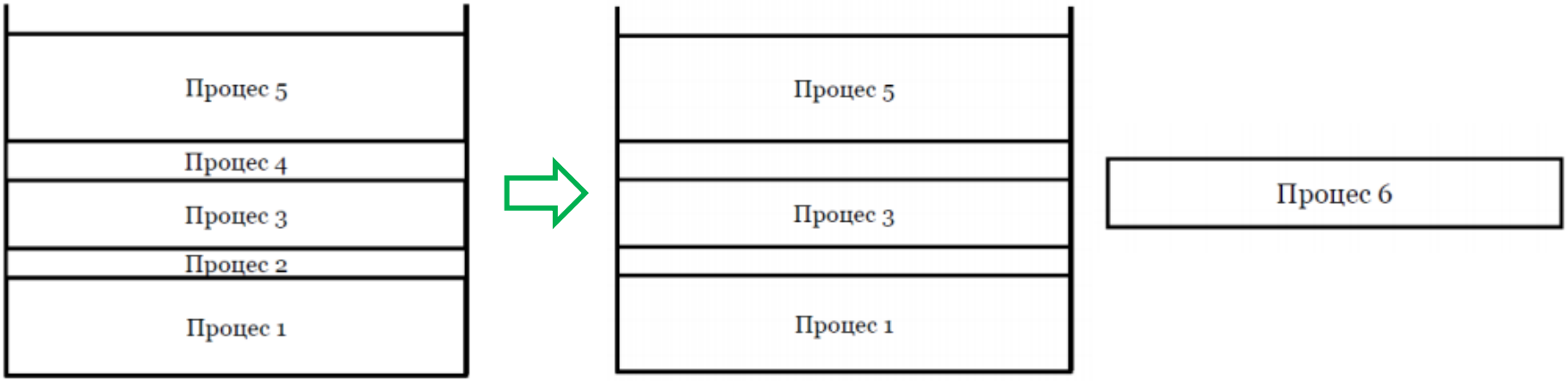
3. Сегментна організація пам'яті

Фрагментація - це явище, за якого у пам'яті з'являється велика кількість коротких несуміжних блоків, внаслідок чого система не в змозі виділити довгий блок, навіть якщо загальний обсяг пам'ять для цього достатній.

Зовнішня фрагментація притаманна підходам до організації пам'яті з поділом адресного простору на частини змінної довжини - наприклад, сегменти.

За ***зовнішньої фрагментації*** короткі несуміжні блоки лежать за межами адресних просторів процесів (тобто *зовні* - звідси назва).

Як виникає: деякий невеликих блок пам'яті вже звільнено, а наступний за ним блок - ще ні. На місці звільненого блоку виникає *діра*.



4. Сторінкова організація пам'яті

За сторінкової організації пам'яті віртуальний адресний простір ділять на ***сторінки***, а фізичний адресний простір на ***фрейми***.

І сторінки, і фрейми мають однакову довжину, фіксовану у межах системи (найчастіше від 512 б до 1 ГіБ, виражається степенем числа 2).

Логічна адреса задається:

- номером сторінки
- зсувом усередині сторінки

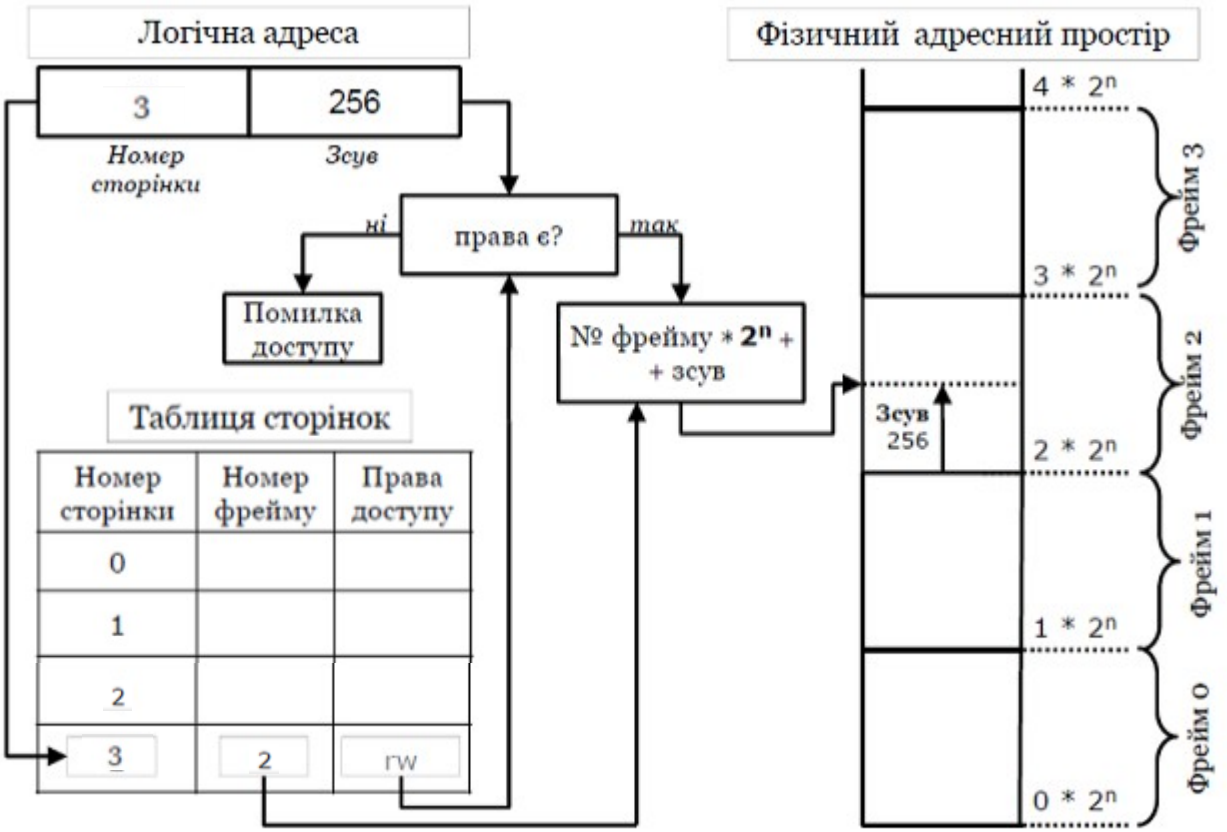
Відомості про сторінки процесу зберігаються у ***таблиці сторінок***. Відомості про фрейми (зокрема які зайняті, а які вільні) зберігається у ***таблиці фреймів***.

Основні **відомості про сторінку** у таблиці:

- *номер* сторінки
- *номер* відповідного фрейму
- *права доступу* до сторінки

4. Сторінкова організація пам'яті

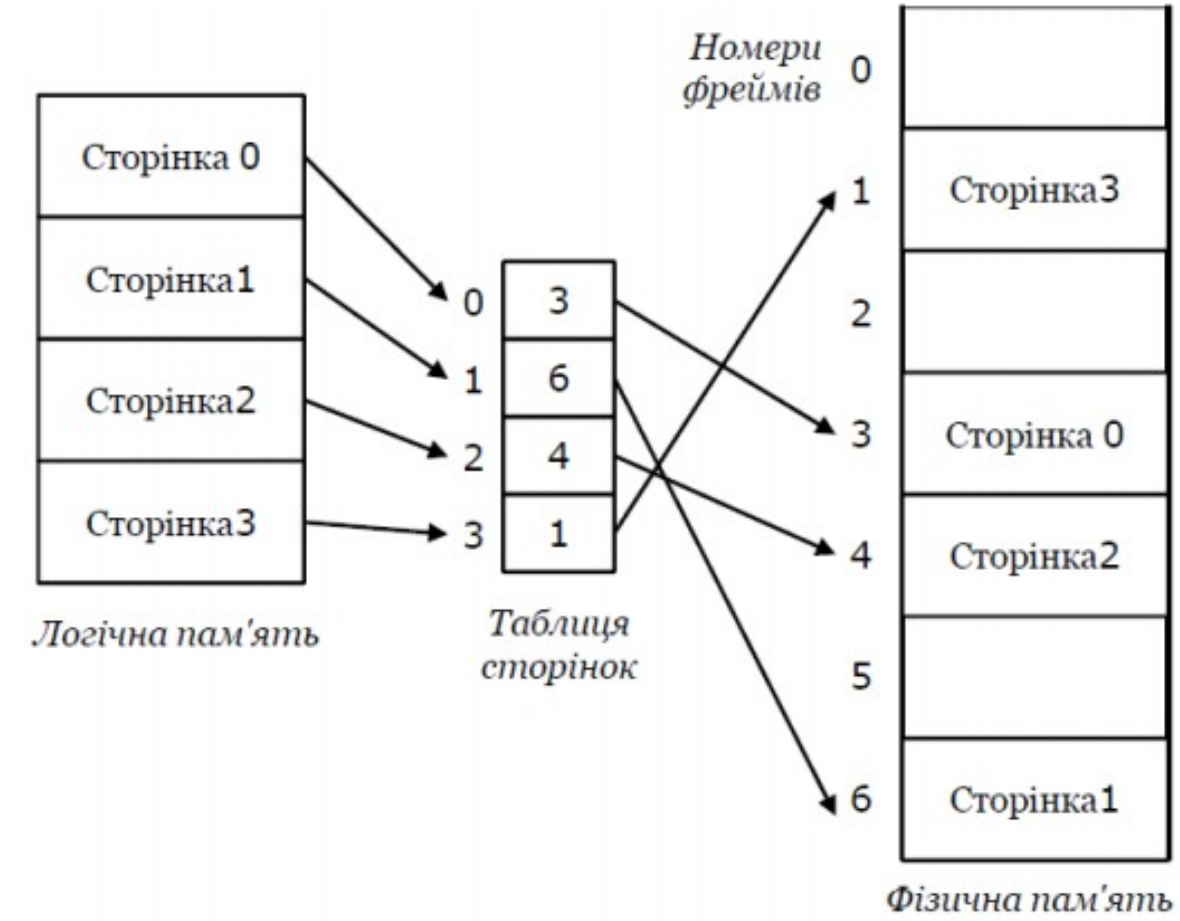
Визначення адреси (сторінкова організація пам'яті)



Схему взято з книги: *В. А. Шеховцов* “Операційні системи” (2005)

4. Сторінкова організація пам'яті

Співвідношення віртуального і фізичного адресного простору (сторінкова організація пам'яті)



Схему взято з книги: *В. А. Шеховцов* “Операційні системи” (2005)

4. Сторінкова організація пам'яті

Переваги підходу (порівняно з сегментною організацією пам'яті)**:**

- сторінку процесу можна передати для використання з іншою метою (у випадку сегментів це було б незручно через змінний розмір сегментів)
- спрощення обміну даними з диском (завдяки однаковому розміру)

Недоліки підходу:

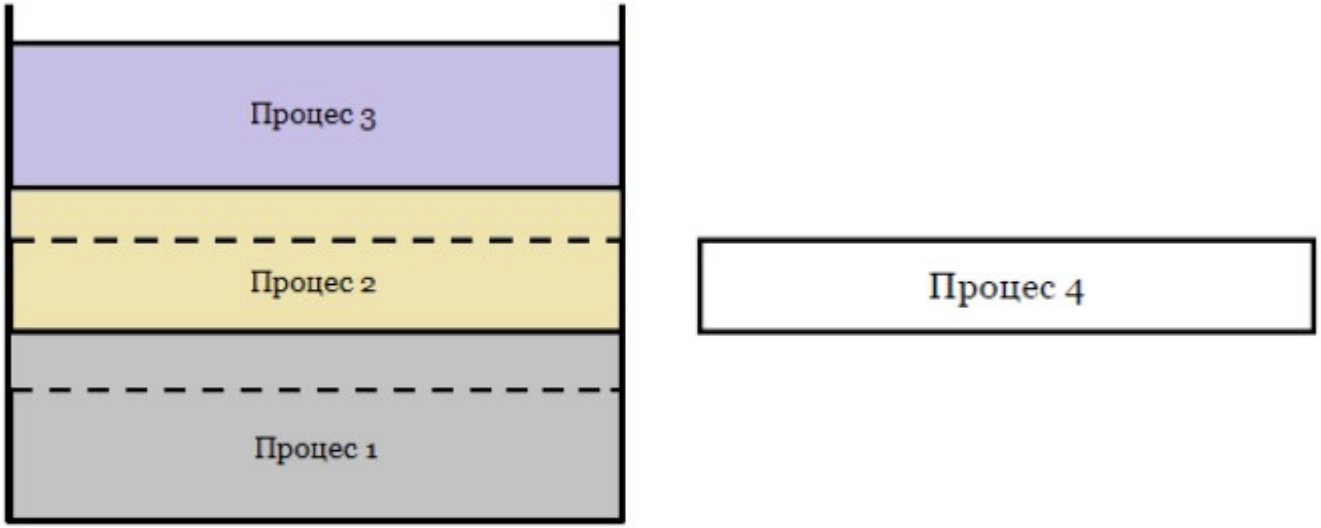
- таблиці сторінок більші за таблиці сегментів
- внутрішня фрагментація

4. Сторінкова організація пам'яті

Внутрішня фрагментація характерна для підходів до організації пам'яті з поділом адресного простору на частини фіксованої довжини - зазвичай *сторінки*.

У випадку ***внутрішньої фрагментації*** короткі несуміжні блоки лежать всередині адресних просторів процесів (виникає *внутрішньо* - звідси назва).

Як виникає. Через фіксований розмір сторінок неможливо виділити черговий блок пам'ять менший за розмір сторінки. Може статися, що чергова сторінка використовується процесом лише частково (процесу стільки не потрібно). Водночас, інші процеси теж не зможуть скористатися вільним місцем (це не їхня сторінка).



4. Сторінкова організація пам'яті

Надто великі сторінки → внутрішня фрагментація

Надто малі сторінки → великі таблиці сторінок → повільний пошук потрібної сторінки

Що тут можна вдіяти:

- намагатися підібрати *оптимальний розмір* сторінки залежно від умов роботи розроблюваної системи;
- зробити *кеш* з відомостями про сторінки, які використовуються найчастіше;
- створити ієрархію сторінок (на зразок змісту книги);
- поєднати сторінки з сегментами
- тощо

Часто перелічені вище ідеї поєднують.

4. Сторінкова організація пам'яті

Багаторівневі таблиці сторінок

Якщо адресний простір процесу великий, то кількість сторінок теж велика і може сягати мільйонів елементів → шукати буде довго.

У випадку багаторівневих таблиць сторінок:

- є декілька рівнів сторінок (небагато, зазвичай 2);
- таблиці сторінок нижнього рівня розбиваються на сторінки
- інформація про сторінки нижнього рівня міститься у таблиці сторінок верхнього рівня (наприклад, в архітектурі IA-32 - у каталозі сторінок)

4. Сторінкова організація пам'яті

Асоціативна пам'ять

До одних сторінок процес звертається частіше, ніж до інших.

Якщо вибрати сторінки, до яких процес звертається частіше, і зберігати відомості про них окремо, то пошук потрібної сторінки може стати швидшим.

Такі сторінки зберігають у спеціальний кеш - ***кеш трансляції*** (***TLB***, *translation lookaside buffer*), або ж ***асоціативна пам'ять***.

Основні відомості про сторінку у кеші TLB:

- номер сторінки
- номер відповідного фрейму

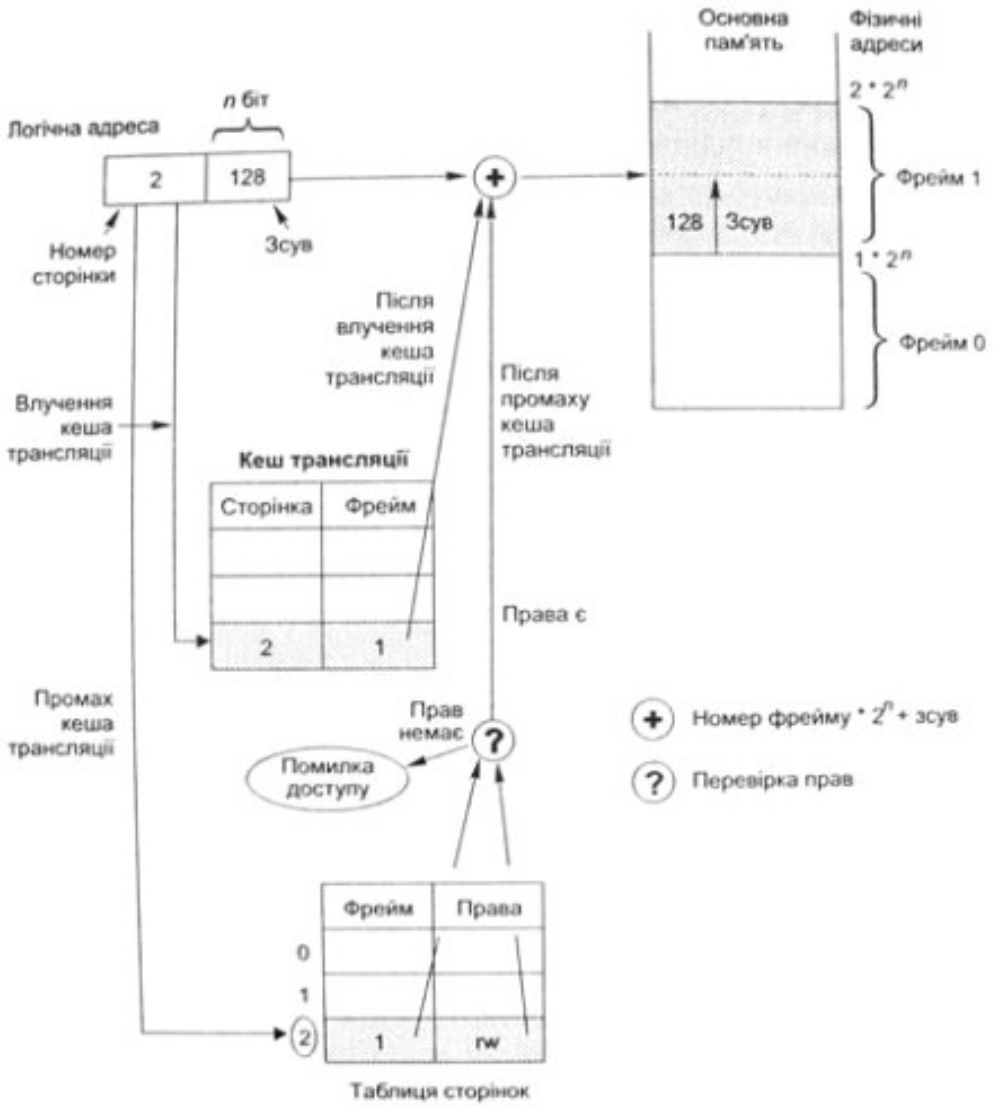
Чому “асоціативна”

Можна шукати *за індексом* (у таблиці сторінок номери - це індекси).

Можна шукати *за вмістом деякого поля* (у кеші TLB номери сторінок не є індексом, бо у кеш потрапляють не всі сторінки, а отже, нумерація не наскрізна). Пошук за вмістом, а не за індексом, є асоціативним.

4. Сторінкова організація пам'яті

Перетворення адреси за участю кешу трансляції



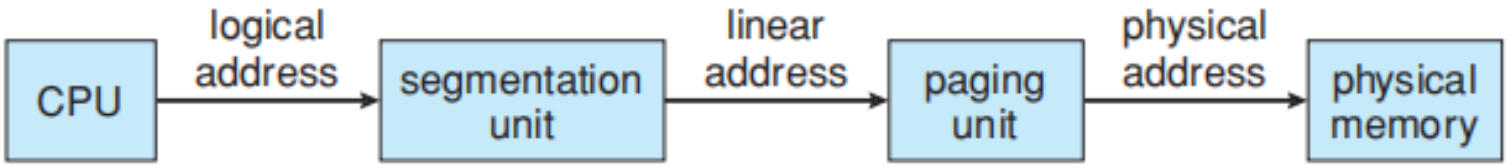
Схему взято з книги: *В. А. Шеховцов* “Операційні системи” (2005)

5. Сторінково-сегментна організація пам'яті

Якщо поєднати сегментну і сторінкову організацію пам'яті, вийде наступне.

Матимемо дві віртуальні адреси: одна з сегментного підходу, інша - зі сторінкового. Назви будуть відрізнятися. Наприклад, в IA-32 це:

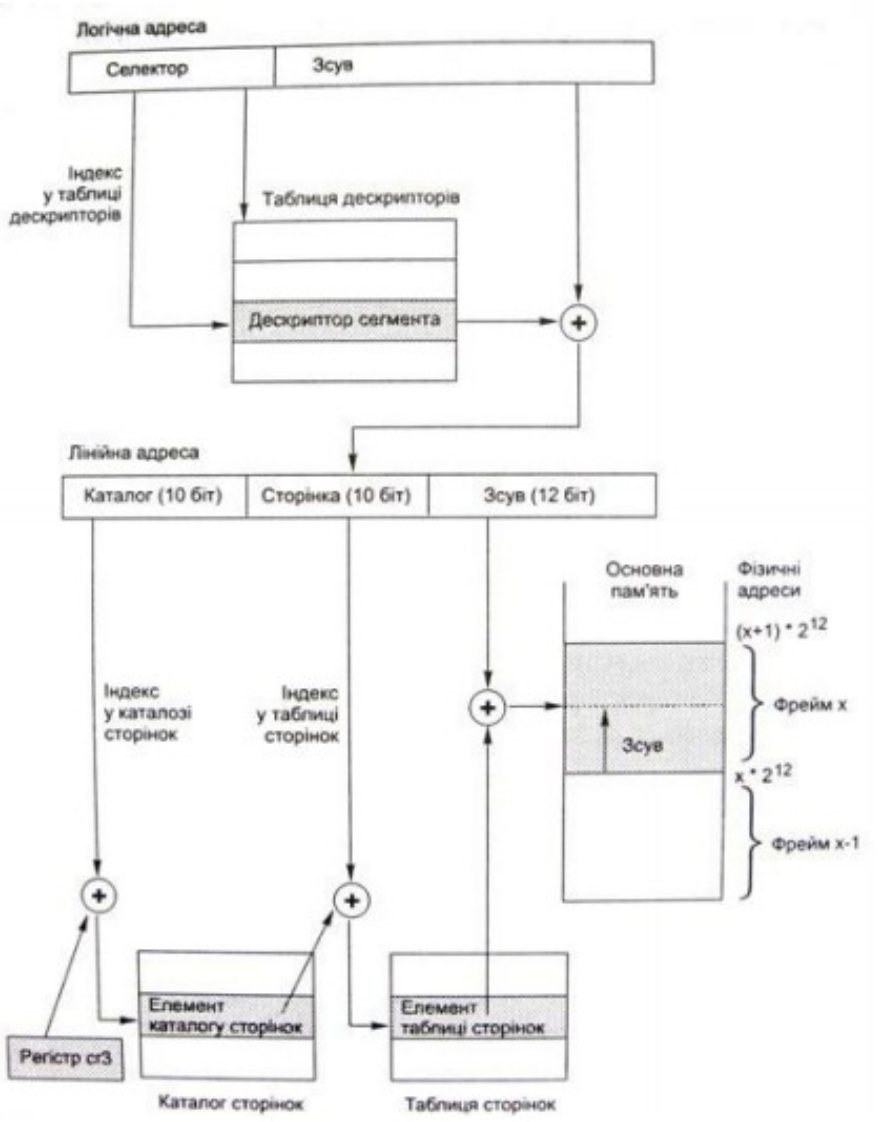
- ***логічна адреса*** (відповідно до сегментного підходу)
- ***лінійна адреса*** (відповідно до сторінкового підходу)



Джерело схеми: *A. Silberschatz, P. Galvin and G. Gagne,* Operating system concepts, 10th ed., Wiley, 2018.

5. Сторінково-сегментна організація пам'яті

Перетворення адреси за сторінково-сегментної організації пам'яті (на прикладі IA-32)



Схему взято з книги: *В. А. Шеховцов* "Операційні системи" (2005)

6. Підкачування. Заміщення сторінок

Чого можна досягнути завдяки підкачуванню?

- було б добре мати більше процесів, запущених одночасно, але так, щоб вони не займали оперативну пам'ять без потреби (більша пропускна здатність, більший % використання ЦП)
- значна частина сторінок процесу використовується дуже рідко (навіщо ж тоді тримати їх усі в оперативній пам'яті)
- економія енергії (оперативна пам'ять енергозалежна, зберігання у ній зайвих даних означає більші витрати не лише місця, а й енергії)

6. Підкачування. Заміщення сторінок

Підкачування (*swapping*, *свопінг*) - технологія передачі сторінки з основної пам'яті на диск і навпаки.

Підкачування можливе і для сегментів, але найчастіше реалізоване для сторінок. Тому надалі говоримо про сторінки.

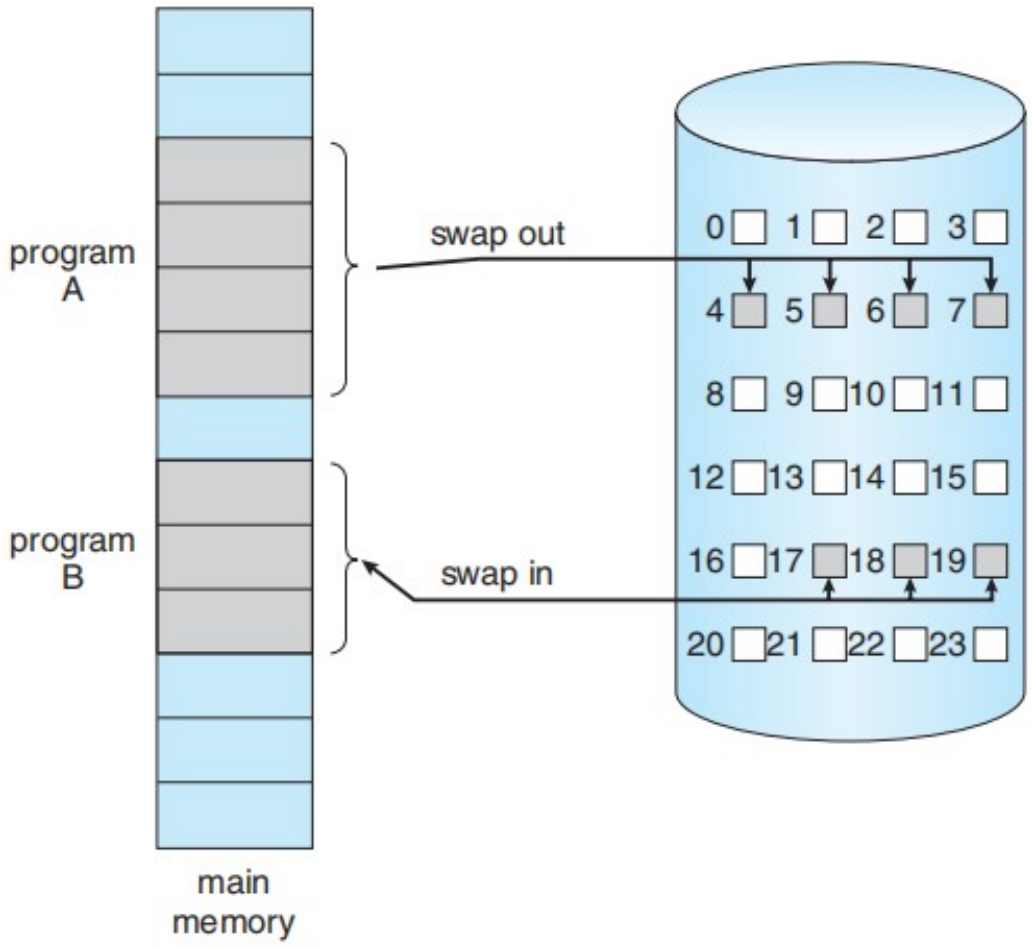
У межах механізму підкачування:

- в основній пам'яті тримається лише частина сторінок (***резидентна множина***) - в ідеалі - найчастіше затребувані;
- решта сторінок зберігається на диску.

Якщо під час виконання процес звертається до сторінки, що на даний момент зберігається на диску, то:

1) відбувається ***сторінкове переривання*** (***помилка відсутності сторінки***, ***сторінкова відмова***);
2) виконання процесу призупиняється;
3) потрібна сторінка завантажується з диску в основну пам'ять;
4) виконання процесу продовжується.

6. Підкачування. Заміщення сторінок



Джерело схеми: *A. Silberschatz, P. Galvin and G. Gagne,* Operating system concepts, 10th ed., Wiley, 2018.

6. Підкачування. Заміщення сторінок

Якщо під час виконання процес звертається до сторінки, що на даний момент зберігається на диску, то:

1) відбувається ***сторінкове переривання*** (***помилка відсутності сторінки***, ***сторінкова відмова***);
2) виконання процесу призупиняється;
3) потрібна сторінка завантажується з диску в основну пам'ять;
4) виконання процесу продовжується.

? Де тут проблема?

Щоб звільнити місце в основній пам'яті для нової сторінки, якусь іншу сторінку може знадобитися вилучити з основної пам'яті (вивантажити на диск).

Є різні алгоритми вибору такої сторінки - ***алгоритми заміщення сторінок****.*

6. Підкачування. Заміщення сторінок

Алгоритми заміщення сторінок

☆ *Оптимальний алгоритм*

Кожна сторінка має спеціальну позначку - кількість команд, що лишилися до моменту, коли процес звернеться до цієї сторінки. Тоді вилучати треба ту сторінку, звернення до якої відбудеться найпізніше.

Цей алгоритм є дуже ефективним, але його неможливо реалізувати. В ОС відсутня інформація для створення таких позначок.

Оптимальний алгоритм використовується для оцінювання ефективності інших алгоритмів (для порівняння).

☆ *Алгоритм FIFO (First In, First Out)*

Вилучається сторінка, яку було завантажено в основну пам'ять першою.

Реалізувати нескладно. Але може призвести до вилучення сторінки, що часто використовується.

Алгоритм має невисоку ефективність і у чистому вигляді застосовується рідко.

6. Підкачування. Заміщення сторінок

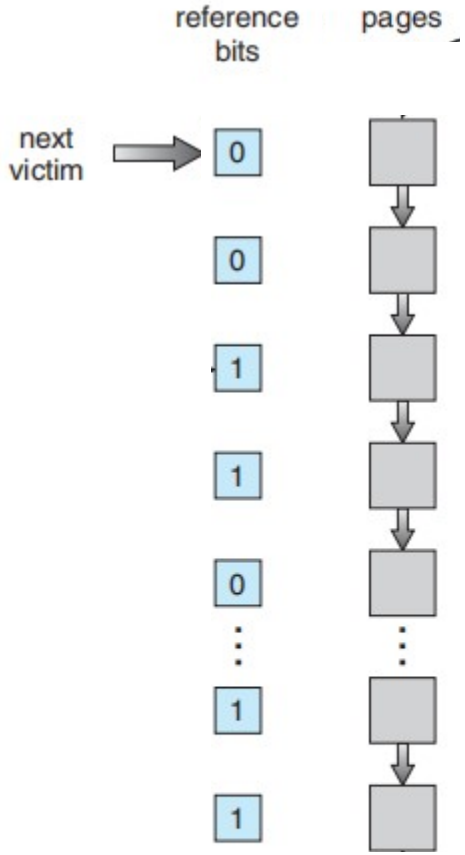
☆ ***Алгоритм “другий шанс” (second chance algorithm)***

Сторінки зберігаються у списку, відсортованому за часом надходження сторінки в основну пам'ять.

Якщо до першої сторінки у списку нещодавно зверталися, її не вилучаються, а переміщують у кінець списку. Замість неї перевіряють сторінку, яка була у списку другою, і т. д.

Мінімальний ризик вилучити активно використовувану сторінку.

Алгоритм дуже повільний через постійні переміщення сторінок у списку.



Схему адаптовано за книгою: *A. Silberschatz, P. Galvin and G. Gagne,* Operating system concepts, 10th ed., Wiley, 2018.

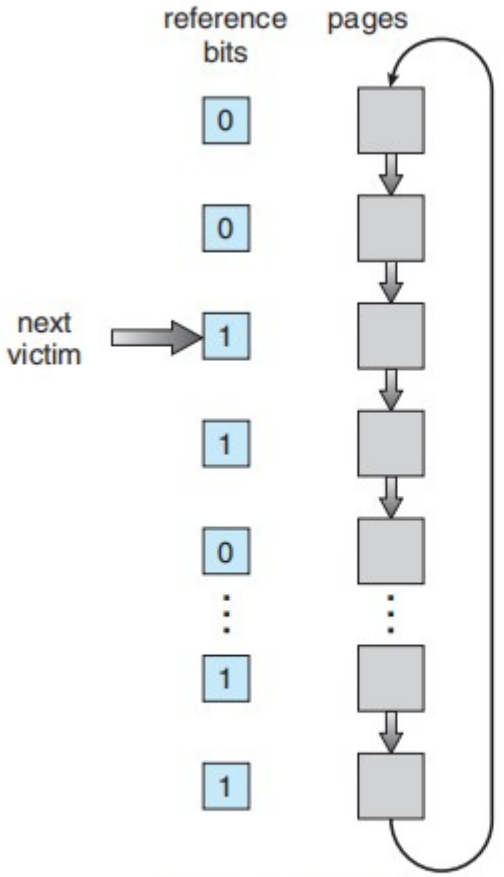
6. Підкачування. Заміщення сторінок

☆ ***Алгоритм “годинник” (clock algorithm)***

Подібний до алгоритму “другий шанс”, але використовує циклічний список.

Найстаріша сторінка у списку (кандидат на вилучення) позначена вказівником. Якщо сторінці дають другий шанс, вказівник пересувається на наступну сторінку.

Завдяки відсутності переміщень елементів у списку, алгоритм значно ефективніший за звичайний “другий шанс”.



Джерело схеми: *A. Silberschatz, P. Galvin and G. Gagne,* Operating system concepts, 10th ed., Wiley, 2018.

6. Підкачування. Заміщення сторінок

☆ ***Алгоритм “покращений другий шанс” (enhanced second-chance algorithm; NRU algorithm - Not Recently Used)***

Резидентна множина сторінок ділиться на 4 класи:

1) Клас (0, 0). Сторінки, до яких останнім часом не було звернень і які не модифікувалися.
2) Клас (0, 1). Сторінки, до яких останнім часом не було звернень, але їх було модифіковано (раніше).
3) Клас (1, 0). Сторінки, до яких останнім часом були звернення, але які не модифіковувалися (взагалі).
4) Клас (1, 1). Сторінки, до яких останнім часом були звернення і які було модифіковано.

Вилучається довільна сторінка з найнижчого непорожнього класу.

Можна поєднати з ”годинником”.

6. Підкачування. Заміщення сторінок

☆ ***Алгоритм LRU (Least Recent Used algorithm)***

Вилучається сторінка, що використовувалася найдавніше.

За ефективністю близький до оптимального, але, як і оптимальний, є дуже складним у реалізації.

Є наближення до алгоритму LRU (наприклад, ***алгоритм старіння***).

☆ ***Алгоритм “робочий набір” (work set algorithm)***

До робочого набору включають ті сторінки, які останнім часом використовувалися найчастіше.

Складний у реалізації. Ресурсомісткий.

☆ ***Алгоритм WSClock***

Поєднання алгоритмів “робочий набір” та “годинник”.

Список такий, як у годиннику. Але замість сортування за часом завантаження сторінку у пам'ять - сортування за часом останнього використання сторінки.

Для самостійного читання

1. **[*Silberschatz, Galvin, Gagne, 2018*]** Chapter 9.
2. **[*Stollings, 2017*]** Chapters 7-8.
3. **[*Tanenbaum, Bos, 2014*]** Chapter 3.
4. **[*Шеховцов, 2009*]** Розділи 8-9.