**1.Призначення та головні ролі операційної системи.**

Операційна система – це красивий та зручний інтерфейс над апаратною частиною (hardware), що також виступає в ролі ресурс менеджера (бере на себе обов’язки щодо розподілу ресурсів при одночасній роботі кількох програм; поділ рекерує тим, як використовується пам’ять;

-----------------------------------

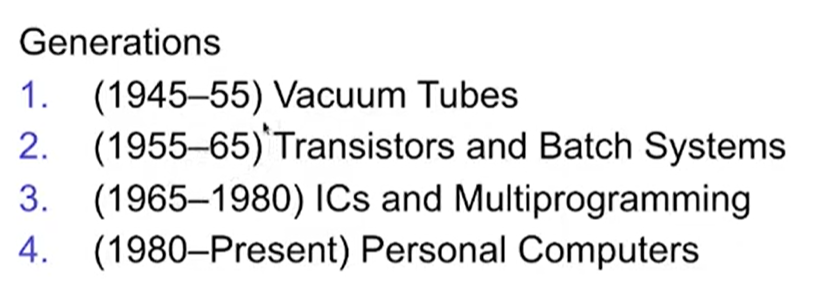
Дві основні функції, які виконує ОС, це функції розширеної машини та диспетчеру ресурсів.

Операційна система як розширена машина приховує безладні деталі, які необхідно виконати та представляє користувачеві гарний інтерфейс, більш простий у використанні.

Операційна система як диспетчер ресурсів дозволяє запускати кілька програм одночасно, здійснює керування та захист пам'яті, пристроїв введення-виводу та інших ресурсів, включає мультиплексування (спільне використання) ресурсів двома різними способами: у часі, або у просторі.

Ще іноді відділяють функцію ОС як віртуальної машини - для введення більшої гранулярності

**2. Покоління архітектур та операційних систем**

1. (1945–55) Електронні лампи

2. (1955–65) Транзистори та системи пакетної обробки

3. (1965-1980) Мікросхеми та багатозадачність

4. (1980-тепер) Персональні комп'ютери

перемички -> перфокарти, FMS -> найпростіший вид мультизадачності; spooling(запуск сканування перфокарт незалежно від виконання більш потужніх обчислювальних задач) -> MultICS (time sharing) -> UNIX -> персональні пк (поява граф інтерфейсів, миші і тд)

**3. Системні виклики. Призначення, механізм, послідовність виконання**

Основою будь-якої операційної системи є набір системних викликів, які вона здатна обробити. Вони свідчать, що реально робить операційна система.

Ми розглянули чотири групи системних дзвінків для UNIX. Перша з них призначалася для створення та припинення процесів. Друга призначалася для читання та запису файлів. Третя група служила керувати каталогами. Четверта група включала системні виклики різного призначення.

**4. Класифікація ОС**

- ОС мейнфреймів (великі універсальні машини)

- Серверні ОС(обслуговування мережі з великою к-стю користувачів)

-Багатопроцесорні (паралельні-, мультикомп’ютери абоб багатопроцесорні системи)

-Персональні комп’ютери

-Карманні(Телефони)

-Вбудовані(Мікрохвильовки, телевізори, авто, dvd)

-ОС сенсорних вузлів

-Реального часу (зварювальний робот)

-Smart card

**5. Різновиди структурування ОС (1.7.3)**

-Монолітні

-Багаторівневі (MULTICS)

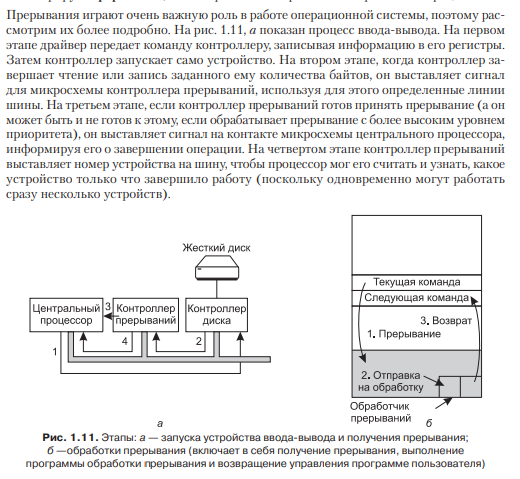
-Мікроядра

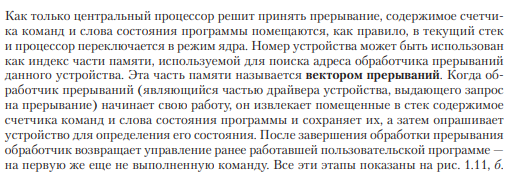
-Клієнт-серверна модель (варіація мікроядер)

-Віртуальні машини

-Екзоядра

**6. Послідовність обробки переривань в класичній операційній системі**





**7. Віртуалізація та пара віртуалізація. Метод бінарної трансляції.**

Віртуалізація - це технологія віртуальної машини, що дає можливість одному комп’ютеру стати основою для декількох віртуальних машин, і на кожній із них можуть знаходитись різні операційні системи. Перевага такого підходу заключається в тому, що відмова на одній з віртуальних машин не призводить до автоматичного збою на всіх інших. На віртуалізованій системі різні сервери можуть працювати на різних віртуальних машинах, підтримуючи модель часткової відмови, наявну на мультикомп’ютерах, але з набагато меншими затратами та легшим керуванням.

Паравіртуалізація - видалення (навмисне) з гостьової операційної системи деяких службових команд.(гостьова ос - ос, зо працює над гіпервізором)

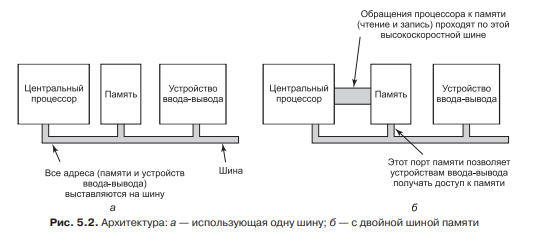
Метод бінарної трансляції – трансляція блоків коду на ходу, зберігання їх у внутрішньому кеші, і повторного використання результата трансляції в випадку їх нового виклику.

**8. Організація вводу-виводу**

Ввод-вывод может осуществляться одним из трех способов.

1. Программный способ ввода-вывода (центральный процессор занимается вводом или выводом каждого байта или слова и пребывает в цикле ожидания, пока сможет получить или отправить следующую порцию данных)
2. Ввод-вывод, управляемый прерываниями (центральный процессор инициирует передачу символа или слова и переходит к решению каких-нибудь других задач, пока не поступит прерывание, сигнализирующее о завершении ввода-вывода.
3. Прямой доступ к памяти — DMA (отдельная микросхема полностью управляет передачей блока данных, выдавая прерывание только тогда, когда будет передан весь блок.

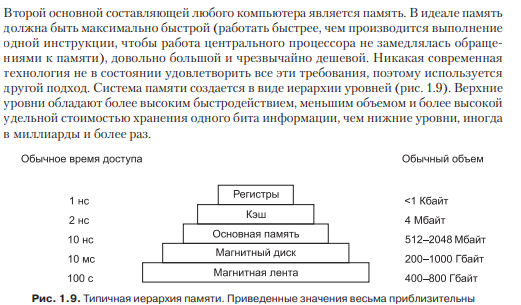
Структура ввода-вывода может состоять из четырех уровней: процедур обработки прерываний, драйверов устройств, программ ввода-вывода, не зависящих от конкретных устройств и библиотек ввода-вывода, и программ спулинга, работающих в пользовательском пространстве. Драйверы устройств учитывают особенности работы устройств и предоставляют единообразные интерфейсы для всех остальных компонентов операционной системы.



**9. Хар-ні риси архітектури сучасного компа. Процесор, ієрархія пам’яті, шини.**

Компьютеры состоят из процессоров, памяти и устройств ввода-вывода. Все эти составные части соединяются с помощью шин

Центральный процессор выбирает команды из памяти и выполняет их. Обычный цикл работы центрального процессора выглядит так: выборка из памяти первой команды, ее декодирование для определения ее типа и операндов, выполнение этой команды, а затем выборка, декодирование и выполнение последующих команд.



Верхний уровень состоит из внутренних регистров процессора. Они выполнены по той же технологии, что и сам процессор, и поэтому не уступают ему в быстродействии.

Затем следует кэш-память, которая управляется главным образом аппаратурой.

Следующей в иерархии, изображенной на рис. 1.9, идет оперативная память.

Дополнительно к оперативной памяти многие компьютеры оснащены небольшой по объему неизменяемой памятью с произвольным доступом — постоянным запоминающим устройством (ПЗУ).

Флеш-память также обычно используется как носитель информации в портативных электронных устройствах.

**10. Основні поняття в галузі знань про операційні системи.**

Процес – програма під час її виконання, контейнер інформації, необхідної для виконання програми.

Адресний простір – список адрес комірок пам’яті, звідки процес може зчитувати дані та записувати туди дані

Образ пам’яті – власний адресний простір процеса

Таблиця процесів – масив стуктур, що представляють собою кожен існуючий на даний момент процес

Інтерпретатор команд (оболонка) – процес, що зчитує команди терміналу

Дочірні процеси – процес, створений іншим процесом

Міжпроцесорна взаємодія – з’язані процесі разом працюють над виконанням якоїсь задачі (+обмін даних, синхронізація)

Сигнал тривоги – сигнал, що посилає система процесу, якщо за вказаний інтервал часу його повідомлення не отримало відповіді

UID – ідентифікатор користувача

GID – ідентифікатор групи

Каталог – спосіб об’єднання файлів в групи

Робочий каталог

Дескриптор файла

(для UNIX) Змонтована файлова система

(для UNIX) Спеціальний файл (для того щоб пристрої вводу-виводу були схожі на файли, 2 види: блочні та символьні)

Коренева файлова система

Канал – різновид псевдо файлу, яким можна скористуватись для об’єднання двох процесів

**11. Концепція процесу та потоку. Діаграма станів процесу**

Щоб згладити вплив переривань, операційні системи надають концептуальну модель, що складається з послідовних процесів, що паралельно запускаються.

Процес - це екземпляр виконуваної програми, включаючи поточні значення лічильника команд, регістрів та змінних. Центральний процесор постійно перемикається між процесами (мультипрограмування, або багатозадачним режим роботи)

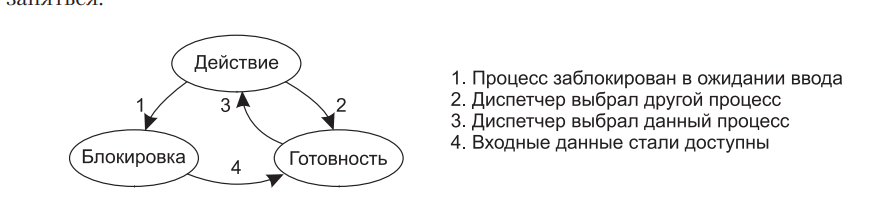
Один процессор може спільно використовуватися декількома процесами в відповідності з деяким алгоритмом планування, який використовується для визначення, коли зупинити один процес і обслужити інший.

Процеси можуть створюватися та знищуватися в динамічному режимі.

Кожен процес має власний адресний простір.

Деяким додаткам вигідно у межах одного процесу мати кілька потоків управління. Ці потоки мають незалежне планування, і кожен з них має власний стек, але всі потоки, що належать до одного процесу, використовують спільний адресний простір. Потоки можуть бути реалізовані у просторі користувача або у просторі ядра

http://um.co.ua/8/8-2/8-228807.html

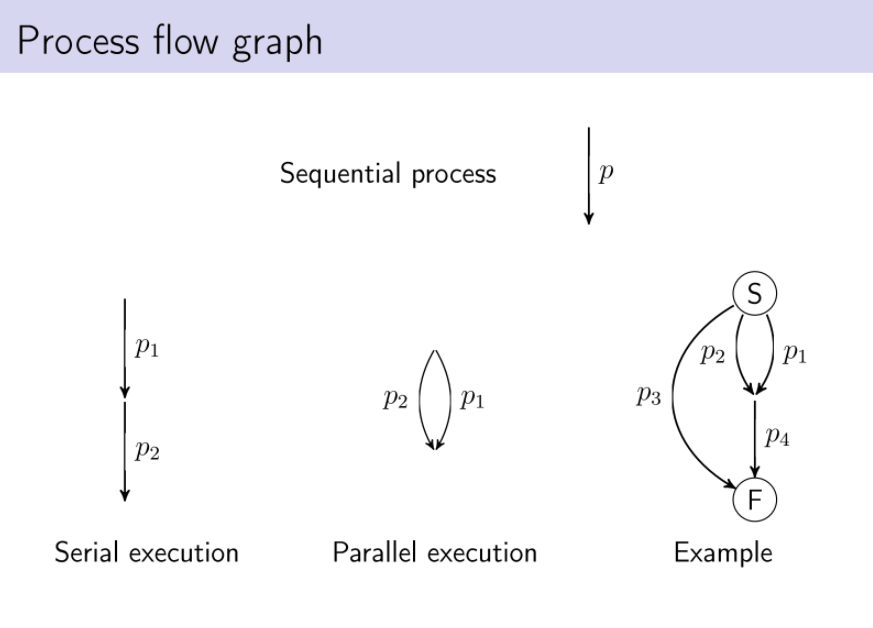


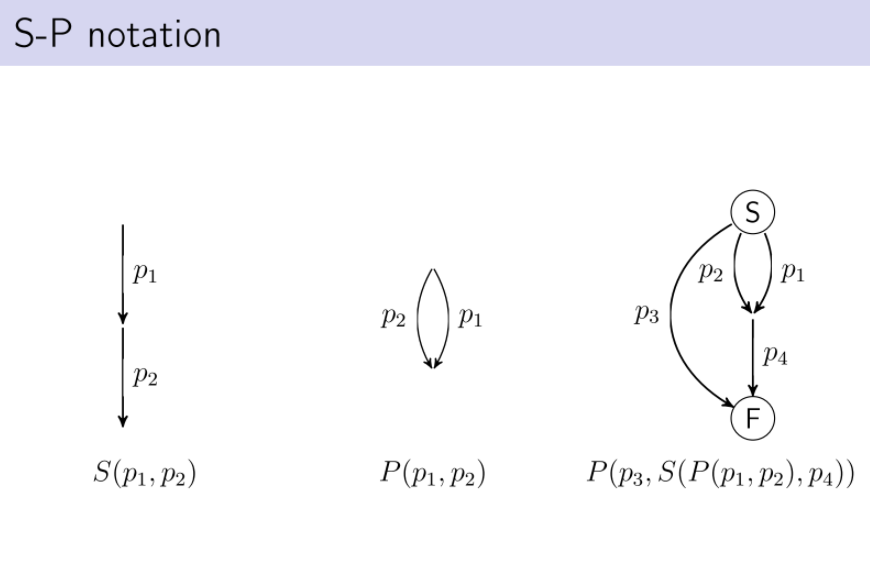
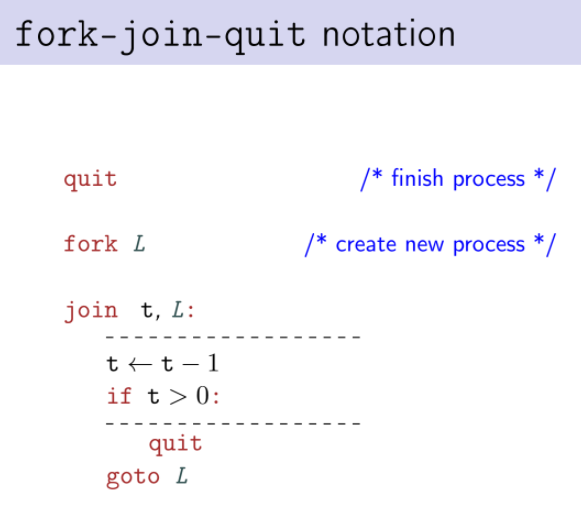
виконується (в даний момент використовує центральний процесор);

готовий (працездатний, але тимчасово припинений, щоб дати можливість виконання іншого процесу);

заблокований (нездатний виконуватися, доки виникне якесь зовнішнє подія).

**12 Граф розвитку процесу. Запис процесів у вигляді термів fork-join-quit**



13. Багатопоточна модель. Мотивація, переваги та проблеми використання.

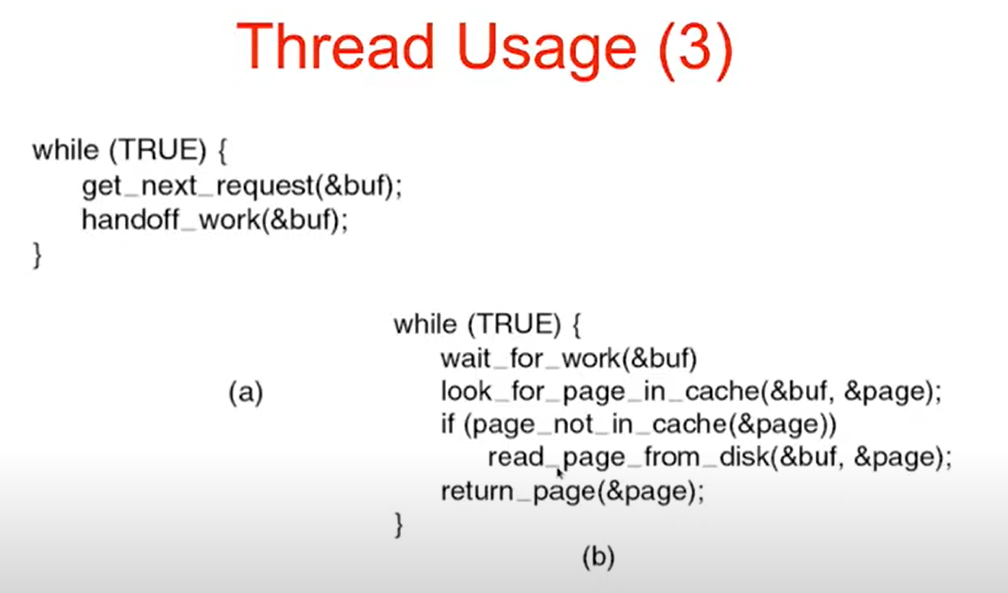
Деяким програмам вигідно в рамках одного процесу мати кілька потоків

управління. Ці потоки мають незалежне планування, і кожен із них має

власний стек, але всі потоки, що належать до одного процесу, використовують загальне адресний простір. Потоки можуть бути реалізовані у просторі користувача або у просторі ядра

(+)

Нехай нам потрібно читати сторінки з кешу, якщо вони там є, або шукати їх на диску, зберігати в кеші і лише після цього читати



Багатопоточність в такому випадку допомагає не втрачати час, коли процес чекає відповіді з диску. Можливо, якась із наступних сторінок вже є у кеші? Це можуть перевірити інши потоки.

Проблеми:

* - взаємовиключення
* - контекст видимості зміни (Thread local storage)
* - ускладнення механізму створення нових процесів
* - обробка сигналів(якому потоку його надсилати)
* - спільні ресурси в сенсі узгодження інформації
* - спільні ресурси в сенсі роботи з файлами
* - ускладнення відлагодження

**14. Підходи до реалізації потоків.**

Існує два основні способи реалізації набору потоків:

1. в користувальницькому просторі (ядро нічого про них не знає)  
    pros: дешеві операції створення, перемикання, легко створити  
    cons: модель не пристосована до блокуючих викликів
2. в просторі ядра

pros: можливість використовувати блокуючі виклики  
 cons: дорожчі операції, бо вони засновані на системних викликах

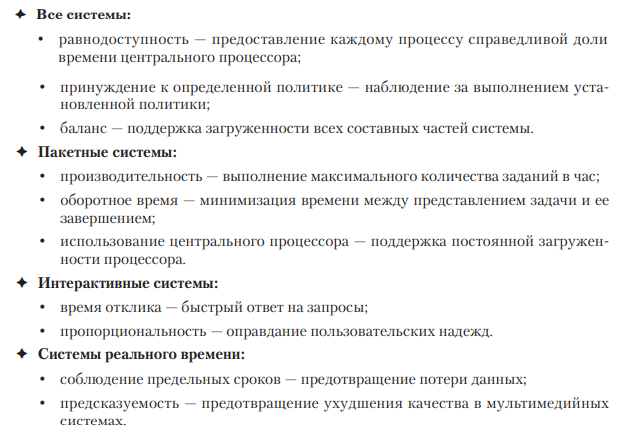
Гібридний: потоки, що підтримуються ядром + потоки в користувальницькому просторі

15. Організація потоків із активацією планувальника (стр. 143-145)

**16. Концепція планування. Задачі планувальника та алгоритм планування. Класифікація алгоритмів планування. Політика та механізм.**

Планувальник – підсистема ОС , яка визначає, який потік виконує кожен з наявних процесорів

Задачі алгоритма планування в різних середовищах



Алгоритми:

* В пакетних системах(FIFO, Shortest job first,
* В інтерактивних системах (циклічне планування, пріоритетне планування, декілька черг, наступний найбільш короткий процес, гарантоване планування, лотерейне планування, справедливе планування)
* В системах реального часу(ст. 195)

Класифікація алгоритмів:

Витісняюча і невитісняюча(процес не знімається з виконання без добровільної передачі процесору) багатозадачність (preeptive & non-preemptive)

Жоден з вищезгаданих процесі не приймає ніяких вхідних даних від користувацьких процесі, тому його рішення можуть бути доволі неефективними. Цю проблему вирішує принцип розділення механізму і політики планування (наявність якогось способу параметризації алгоритму планування, що передбачає можливість поповнення параметрів з боку процесів користувача.)

Механізм знаходиться в ядрі, а політика встановлюється користувацькиим процесом.

1. **Планування в системах пакетної обробки.**

* FIFO "першим прийшов - першим обслужений": при його використанні центральний процесор виділяється запитам у порядку їх надходження. Плюси: простота розуміння та реалізації у програмуванні. Мінуси: час виконання.
* SJF "спочатку найкоротше завдання". Передбачається, що терміни виконання завдань відомі заздалегідь. Коли в очікуванні запуску у черзі знаходиться кілька рівнозначних за важливістю завдань, планувальник вибирає спочатку найкоротше завдання. Плюси: працює швидше алгоритму “першим прийшов – першим обслужений”. Мінуси: завдання мають надійти одночасно.
* shortest remaning time next (алгоритм першочергового виконання завдання з найменшим часом виконання. При використанні цього алгоритму планувальник завжди вибирає процес з найкоротшим часом виконання. Час виконання відомий заздалегідь. При надходженні нового завдання виконується порівняння загального часу його виконання з термінами виконання поточних процесів, що залишилися. Плюси: максимальна швидкість у порівнянні з попередніми алгоритмами.

**18. Планування в інтерактивних системах.**

* Циклічне планування. Кожному процесу призначається

певний інтервал часу, званий квантом, протягом якого йому надається можливість виконання. Якщо до завершення кванта часу все ще виконується, то ресурс центрального процесора у нього відбирається і передається іншому процесу, а сам процес поміщається в кінець черги. Очевидно, якщо процес завершує свою роботу до закінчення кванта часу, то перемикання центрального процесора на інший процес відбувається конкретно в цей момент.

* Пріоритетне планування. Кожному процесу надається значення пріоритетності і запускається той процес, який перебуває у стані готовності та має найвищий пріоритет. Щоб запобігти безкінечному виконанню високо пріоритетних процесів, планувальник повинен знижувати рівень пріоритету поточного виконуваного процесу з кожним сигналом таймера. Пріоритети можуть надаватися процесам у статичному чи динамічному режимі.
* Використання кількох черг. Було засновано класи пріоритетів. Процеси, що відносяться до найвищого класу, запускаються на 1 квант часу, процеси наступного за низхідним класом - на 2 кванта часу, процеси наступного класу - на 4 кванта часу, і т. д. Як тільки процес використовував всі виділені йому кванти часу, його клас знижувався.
* Гарантоване планування. У наданні користувачам реальних обіцянок щодо продуктивності, а потім у виконанні цих обіцянок. Одна з обіцянок, яку можна дати і просто виконати, полягає в наступному: в однокористувацький системі, що має n працюючих процесів, за інших рівних умов кожен з них отримає 1/n від загальної кількості процесорних циклів. Щоб виконати цю обіцянку, система повинна відстежувати, скільки процесорного часу витрачено на кожен процес з моменту створення. Потім вона обчислює кількість процесорного часу, на який кожен із них мав право, а саме час з моменту його створення, поділений на n.
* Лотерейне планування. Основна ідея полягає в роздачі процесів лотерейних квитків на доступ до різних системних ресурсів, у тому числі процесорного часу. Коли планувальнику потрібно приймати рішення, у випадковому порядку обирається лотерейний квиток, і ресурс надається процесу, що має цей квиток. Щодо планування процесорного часу система може проводити лотерейний розіграш 50 разів на секунду, і кожен переможець отримуватиме як приз 20 мс процесорного часу.
* Справедливе планування. Деякі системи перед плануванням роботи процесу беруть до уваги, хто є його власником. У цій моделі кожному користувачеві розподіляється деяка частка процесорного часу і планувальник вибирає процеси, дотримуючись цього розподілу. Таким чином, якщо

кожному з двох користувачів було обіцяно по 50% процесорного часу, то вони його отримають, незалежно від кількості наявних у них процесів.

1. Міжпроцесова взаємодія
2. **Поняття стану змагань та задача критичної секції.**

Стан гонитви - коли два або більше процесу зчитують або записують якісь загальні дані, а остаточний результат залежить від того, який процес і коли саме виконується, називається змагальною ситуацією. Налагодження програм, у яких є змагальна ситуація, особливої ​​радості не приносить. Результати більшості прогонів можуть бути цілком прийнятними, але до певного часу, поки не настане той самий рідкісний випадок, коли

станеться щось таємниче та незрозуміле.

Ключем до попередження проблеми у цій та в багатьох інших ситуаціях використання спільної пам'яті, спільних файлів і взагалі чогось спільного може послужити визначення способу, за якого у кожний конкретний момент часу доступ до загальних даних для читання та запису може отримати лише один процес.

Іншими словами, нам потрібен спосіб взаємного винятку, тобто якийсь спосіб, що забезпечує правило, у якому якщо загальні дані або файл використовуються одним процесом, можливість їх використання усіма іншими процесами виключається.

Та частина програми, в якій використовується доступ до загальної пам'яті, називається критичною областю або критичною секцією. Якби вдалося все побудувати таким чином, щоб жодні два процесу не знаходилися одночасно у своїх критичних сферах, це дозволило б уникнути змагань.

Для прийнятного рішення необхідно дотримання чотирьох умов:

1. Два процеси не можуть одночасно перебувати у своїх критичних сферах.

2. Не повинні вибудовуватися жодні припущення щодо швидкості або кількості центральних процесорів.

3. Жодні процеси, що виконуються за межами своїх критичних областей, не можуть блокуватися будь-яким іншим процесом.

4. Процеси не повинні перебувати у вічному очікуванні входу у свої критичні області.

1. **Примітиви взаємного виключення**.  
   При кожному входженні в критичну ділянку має бути звернення до такого примітиву.

Цей примітив собою являє об’єкт, який буває в двох станах : locked і unlocked

(locked не обов’язково = blocked)

Поділяють на 2 категорії:

* Busy waiting (spinlock) або активне очікування
* Інші (напр. семафор)

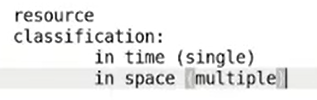
1. **Задача виробників та споживачів та інші класичні задачі на паралелізм.**

Два процеси використовують загальний буфер фіксованого розміру. Один з них, виробник, поміщає інформацію в буфер, а інший споживач витягує її звідти. Проблеми виникають у той момент, коли виробнику потрібно помістити новий запис у вже заповнений буфер. Рішення полягає у блокуванні виробника до того часу, поки споживач не витягне щонайменше один запис. Також, якщо споживачеві потрібно витягти запис з буфера і він бачить, що буфер порожній, він блокується доти, доки виробник не помістить щось у буфер і активізує цього споживача.

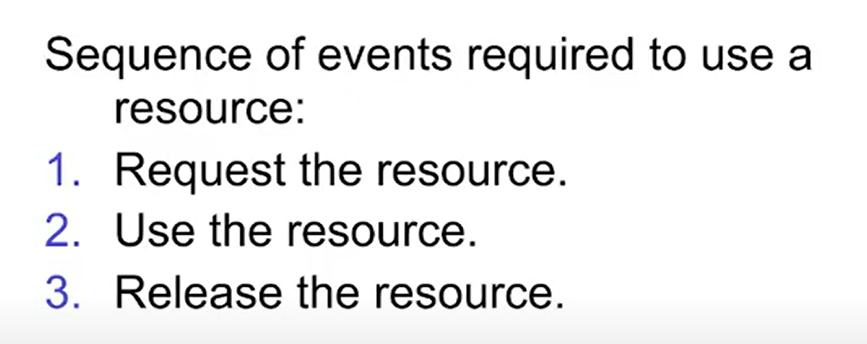
1. Ресурси. Взаємне блокування. Стратегії по управлінню тупиками в операційних системах.

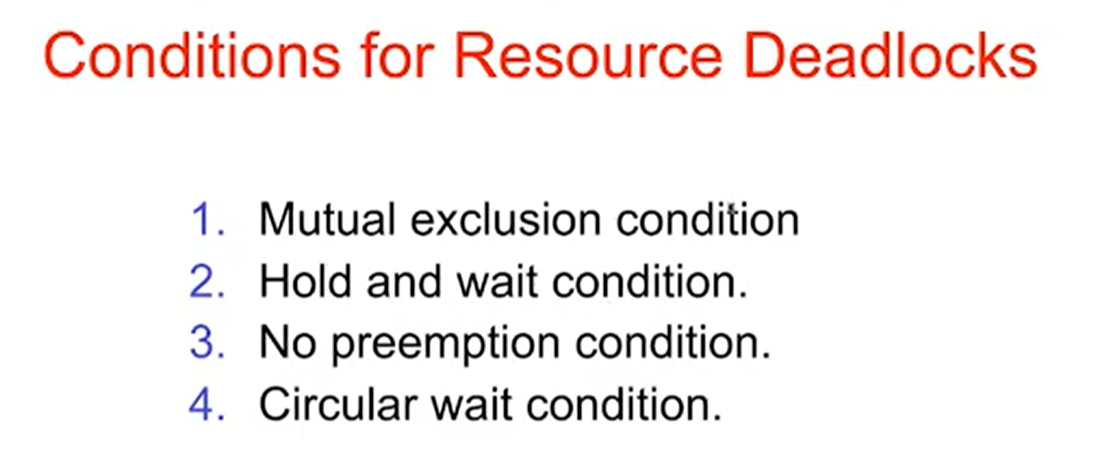
Взаємне блокування в групі процесів виникає в тому випадку, якщо кожен процес з цієї групи чекає на подію, настання якого залежить виключно від другого процесу із цієї групи.

Основна частина взаємного блокування пов’язана з об’єктами для яких надається доступ  - ресурсами. Під ресурсами, мається на увазі, все те, що повинно надаватися, використовуватися і через деякий час вивільнитися, оскільки в один і той же момент часу може використовуватися лише одним процесом.



Цикл роботи з ресурсами

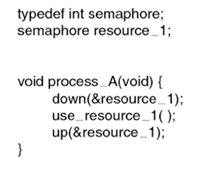




(ексклюзивність ресурсу, умова утримування та запитування, відсутність примусового відбору ресурсу, циклічне очікування)

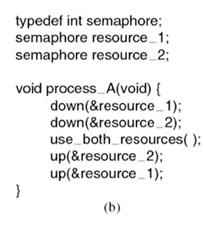
Стратегії по управлінню

1. Відібрати ресурс
2. Відкат
3. Вбити процеси
4. Взаємне блокування в випадку одиночних ресурсів.

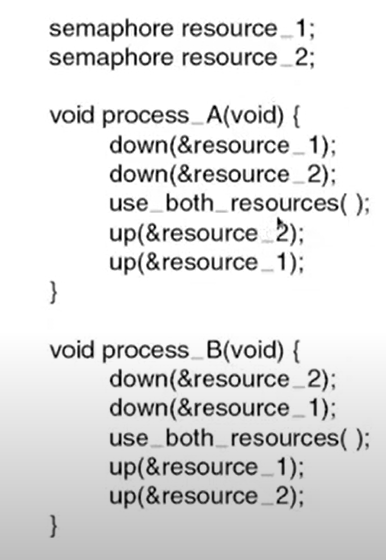


1. Взаємне блокування для множинних ресурсів.

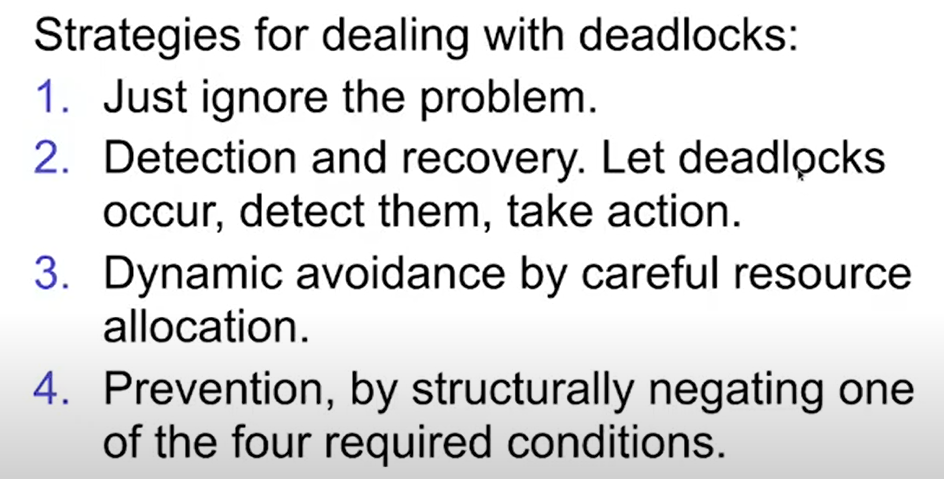
(кілька одиниць однотипних ресурсів, байдуже, який використ)



А тут можливий дедлок:



1. **Стратегії по управлінню тупиками в операційних системах.**

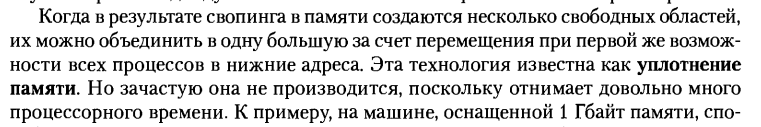


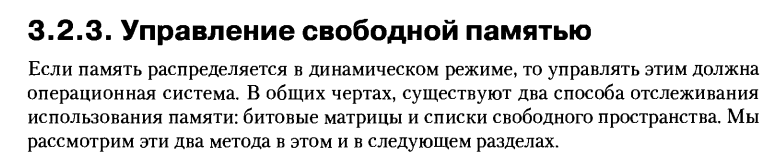
1. **Уникнення (avoidance) взаємного блокування.**

Достатньо усунути хоча б одну з умов для взаємного блокування

1. Активне взаємне блокування, голодування.
2. Основні задачі операційної системи по управлінню пам'яттю при забезпеченні мультизадачності. Загальні підходи до виконання. апаратні можливості, що використовуються. Абстракції пам`яті.
3. Управління пам'яттю в системах з мультизадачністю із фіксованими розділами.
4. **Управління пам'яттю в системах із свопінгом. Зв'язні списки та бітмапи.**

Самий простий метод для подолання проблеми перегрузки пам’яті - свопінг. Полягає він в розміщенні в пам’яті всього процесу цілком, в запуску його на деякий час, а далі в скиданні його на диск. Бездіяльні процеси більшу частину часу зберігаються на диску і в неробочому стані не займають простір оперативної пам’яті (хоча деякі з них періодично активізуються, щоб виконати свою роботу, а потім знову зупиняються) .





* бітмапи (бітові матриці)

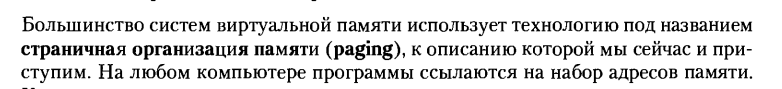
При використанні бітових матриць пам’ять ділиться на одиничні блоки розміром від декількох слов до декількох кілобайт. З кожним одиничним блоком співвідноситься один біт в бітовій матриці, котрий містить 0, якщо одиничний блок вільний, і 1, якщо зайнятий.

* зв’язані списки

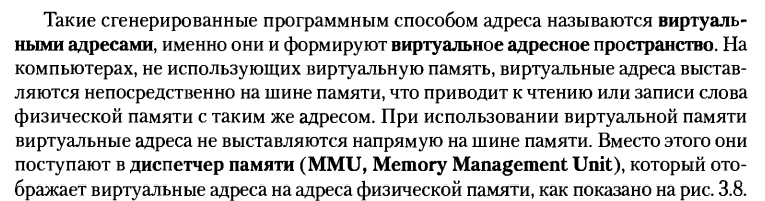
Іншим способом відстежування пам’яті є ведення зв’язаних списків в розподілених і вільних сегментах пам’яті, де сегмент містить в собі або процес, або є пустим простором між двома процесами. Кожен запис в списку зберігає позначення, містить сегмент “дірка” - hole(H) або процес - process(P), адресу, з якої сегмент починається, його довжину та вказівник на наступні записи.

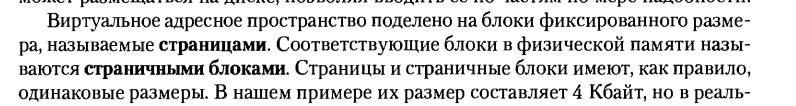
32.Внутрішня та зовнішня фрагментація. Яким видам організації пам’яті властива? Способи її усунення.

33.Сторінкова віртуальна пам'ять. Суть, призначення, апаратна підтримка організації.

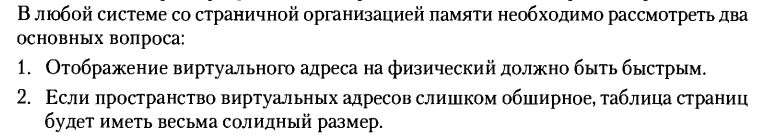


Адреси можуть генеруватися з використанням індексної адресації, базових регістрів, сегментних регістрів та іншими способами

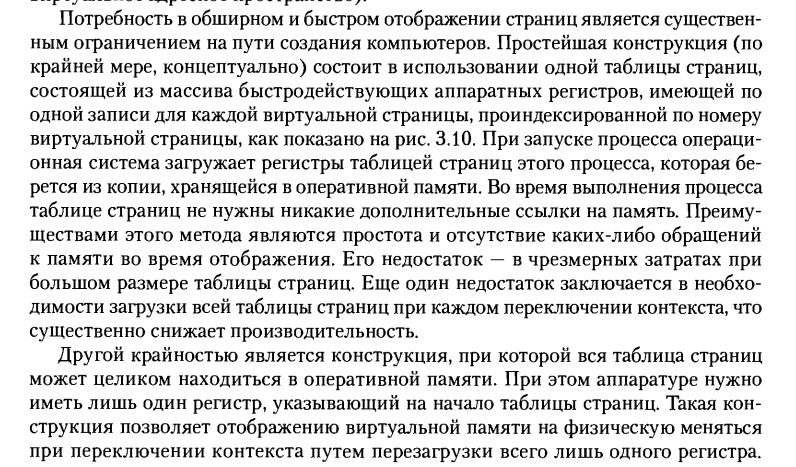


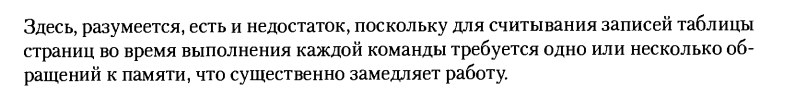


34. Оптимізація використання пам'яті зі сторінковою організацією. Буфер швидкої адресації (TLB), програмний TLB. Багаторівневі та інвертовані таблиці сторінок.

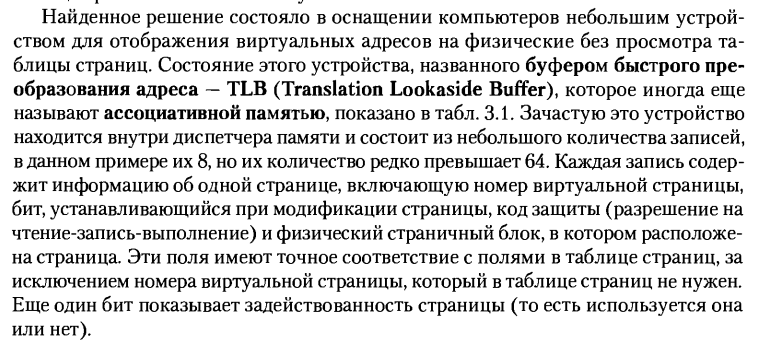


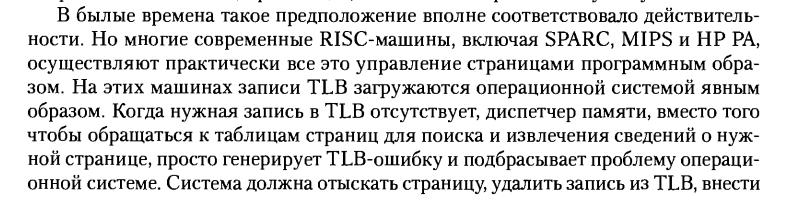
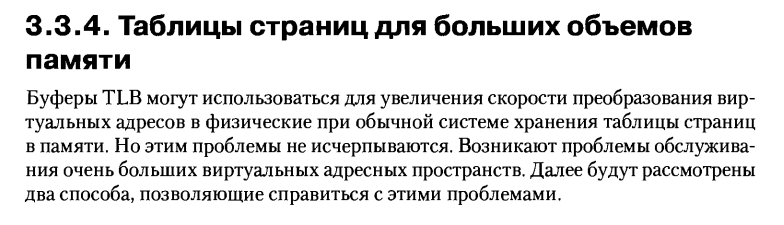
невдалі спроби:





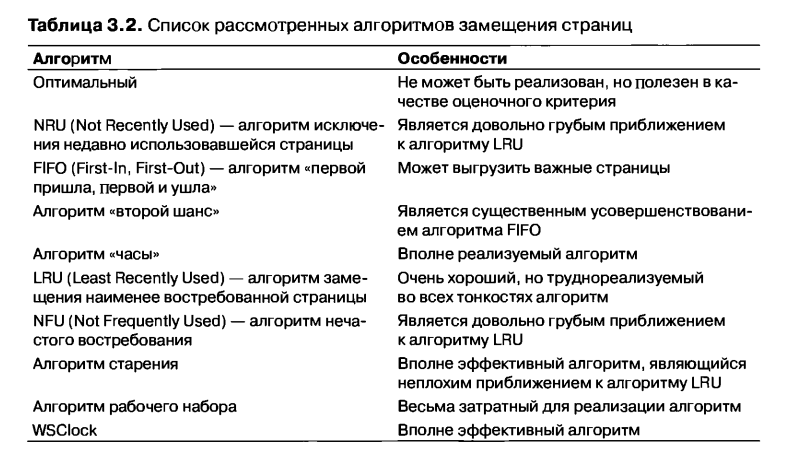
В зв’язку з проблемою продуктивності при використанні сторінкової організації пам’яті був придуманий пристрій.

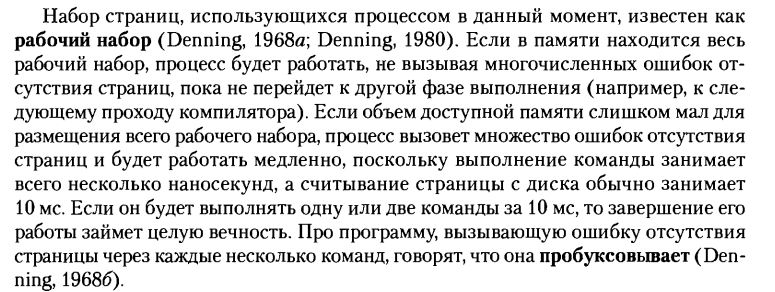
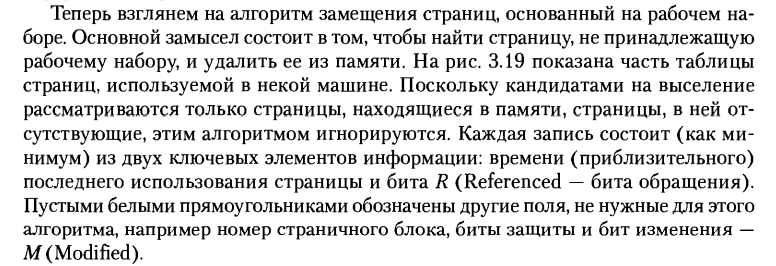
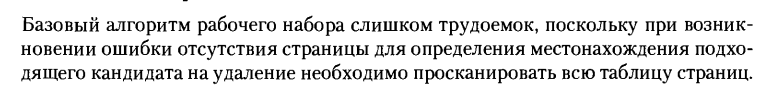
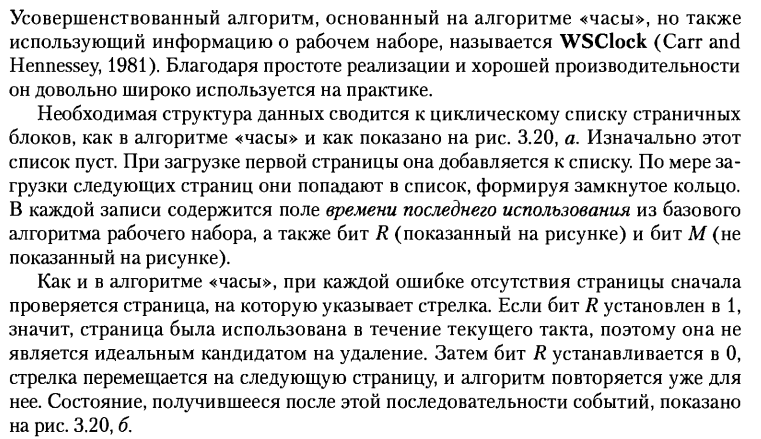


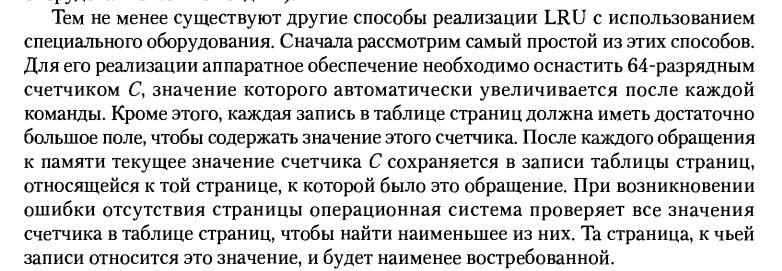
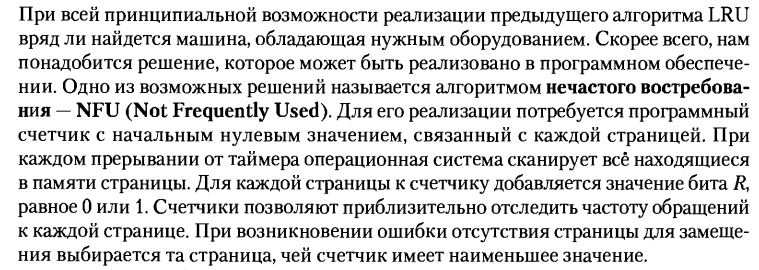
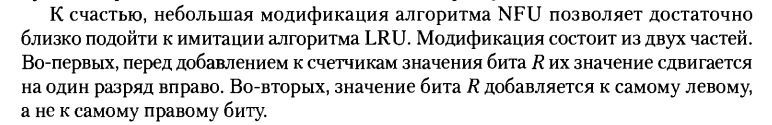
* багаторівневі таблиці сторінок (секрет методу використання багаторівневої таблиці сторінок заключається в відмові від постійного зберігання всіх сторінок таблиці в пам’яті, в частковому випадку, взагалі не повинні зберігатися ті сторінки, в котрих немає необхідності; оптимально для 32-розрядних адресних просторів)
* інвертовані таблиці сторінок (в одній конструкції один запис виділяється не на кожній сторінці віртуального адресно простору, а кожному сторінковому блоку в реальній пам’яті; оптимально для 64-розрядних адресних просторів; перетворення віртуальних адрес в фізичні стає набагато складніше)

35. Відомості про алгоритми заміщення сторінок.



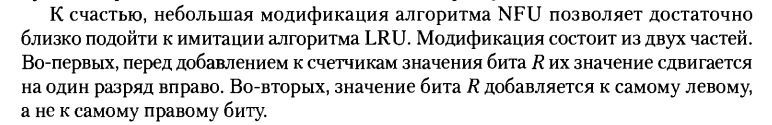
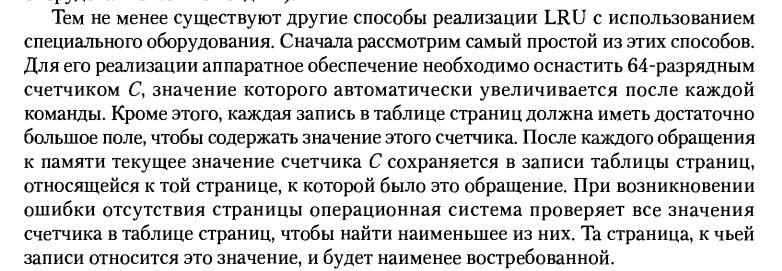
36. Поняття робочого набору. Сторінкове заміщення на основі робочого набору.Алгоритм WS-Clock.    

**37.  Алгоритм заміщення найдовше не використовуваної сторінки, його апроксимації**

Для реалізації алгоритму необхідно ввести зв’язаний список усіх сторінок, що знаходяться в пам’яті. На початку цього списку повинна бути  щойно затребувана сторінка, а в кінці - менш затребувана. Складність полягає у тому, що цей список повинен оновлюватися при кожному зверненні до пам’яті. Для пошуку сторінки в списку, її видалення із нього та наступного переміщення цієї сторінки вперед необхідно досить багато часу.   

**38. Алгоритм заміщення найдовше не використовуваної сторінки, його апроксимації**

Для реалізації алгоритму необхідно ввести зв’язаний список усіх сторінок, що знаходяться в пам’яті. На початку цього списку повинна бути  щойно затребувана сторінка, а в кінці - менш затребувана. Складність полягає у тому, що цей список повинен оновлюватися при кожному зверненні до пам’яті. Для пошуку сторінки в списку, її видалення із нього та наступного переміщення цієї сторінки вперед необхідно досить багато часу.



39. Способи структурування області підкачування

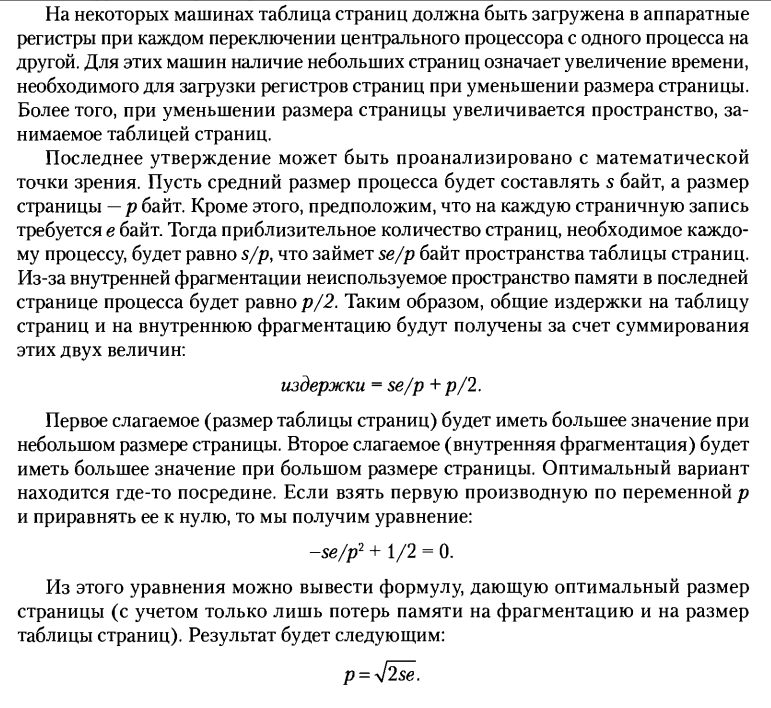
40. Спільна пам’ять та відображення даних

41. Управління очищенням сторінок

**42. Аспекти реалізації: розмір сторінки, обробник помилки сторінок**

Розмір сторінки в більшості випадків є параметром, котрий повинен бути обраний операційною системою.

Визначення найкращого розміру сторінки визначається між декількома конкуруючими факторами. Таким чином, абсолютно оптимального рішення не існує.



На деяких машинах таблиця сторінок повинна бути завантажена в апаратні регістри при кожному перемиканні центрального процеса на інший. Для цих машин наявність невеликих сторінок означає збільшення часу, який необхідний для завантаження регістрів сторінок при зменшенні розміру сторінки. Крім того при зменшенні розміру сторінки збільшується простір, що займає таблиця сторінок.

https://lh3.googleusercontent.com/WIsKJ2CLYVHG5RPjvbLBTnN7Gt78DIvfKSR7aU7WyrLZR-WrgFDrYdDogl2dh52MlZlodYvEnpSkMMRng14kbIdSgNLkJfcGMUDPYsg0qOP9zvtM73OBiNPfvdVkXFFGulddmhc9

оптимальний розмір сторінки (з урахуванням втрати пам’яті на фрагментацію та на розмір таблиці сторінок)

**43. Сегментна організація пам’яті**

У компілятора є множина таблиць, побудованих в процесі компіляції, до яких відносяться:

* Вихідний текст, збережений для друку лістинга (в пакетних системах)
* Таблиця імен, що містить імена та атрибути змінних
* Таблиця, що містить в собі всі константи, що використовуються як цілочисельні так і з плаваючою точкою.
* Дерево розподілу, в якому знаходиться синтаксичний аналіз програми
* Стек, який використовується для виклику процедур в середині компілятора

В процесі компіляції кожна з перших чотирьох таблиць постійно зростає. А остання збільшується або зменшується в розмірах абсолютно непередбачуваним чином.

Простим способом представлення машини з великою кількістю абсолютно незалежних адресних просторів, що мають назву сегменти. Кожен с сегментів складається з лінійної послідовності адрес від 0 до деякого максимуму. Різні сегменти можуть бути різної довжини, яка може змінюватись в процесі виконання програми. Довжина сегмента стека може збільшуватись при надходженні в нього даних і зменшуватись при видаленні з нього.

Сегмент - це логічний елемент, який може зберігати процедуру, або масив, чи стек, або набір скалярних змінних, але зазвичай він містить в собі різні типи даних.

44. Реалізація змішаної організації пам’яті у архітектурі x86

**45. Реалізація змішаної організації пам’яті у MULTICS**

Коли виникає посилання на пам’ять,

виконується алгоритм.

1. Номер сегмента, який використовується для пошуку сегмента

дескриптор.

2. Проводиться перевірка, чи є таблиця сторінок сегмента

знаходиться в пам'яті.

а) Якщо ні, виникає несправність сегмента.

б) Якщо є порушення захисту, виникає несправність (пастка).

3. Запис таблиці сторінок для запитаної віртуальної сторінки

обстежений.

а) Якщо самої сторінки немає в пам'яті, це помилка сторінки

спрацьовує.

b) Якщо він знаходиться в пам'яті, адреса основної пам'яті

початок сторінки витягується із запису таблиці сторінок

4. Зміщення додається до початку початку сторінки, щоб отримати значення

адреса основної пам'яті, де розташоване слово.

5. Читання або збереження нарешті відбувається.