Review

- Virtual Memory with Paging
  - Page Table with Hardware Support
    - Translation Look-Aside Buffer
  - Shared Pages
  - Page Table Structure
    - Multilevel Page Table
    - Hashed Page Table
    - Inverted Page table

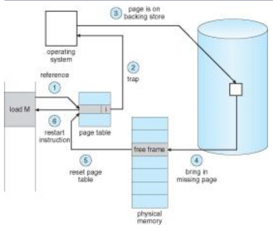
Preview

- Demand Page and Page Fault
- Free-Frame List
- Performance of Demand Page with Page Fault
- Swapping with Paging
- Copy-on-Write between Parent and Child
- Page Table Entries
- Page Replacement Algorithms
  - Optimal Algorithm
  - Not Recently Used
  - First In First Out
  - Second Change
  - The Clock Page Replacement
  - Least Recently Used

Demand Page & Page Fault

- Virtual memory and demand paging are memory management techniques used in operating system.
- Demand paging is a type of swapping done in virtual memory systems.
- In demand paging, the data is not copied from the secondary memory (HDD, SSD) to the main memory until they are needed or being demanded by some program.
- While a process is executing, some pages will be in memory, and some will be in secondary storage.
- For a reference, system check whether a corresponding page is in the memory or not.
- Page fault is a situation such that process tries to access a page that was not brought into memory.

Demand Page & Page Fault



1) Check the location of the referenced page in the page table
2) If a page fault occurred, call on the operating system to fix it
3) Using the frame replacement algorithm, find the frame location
4) Read the data from the secondary memory to memory
5) Update the page map table for the process
6) The instruction that caused the page fault is restarted when the process resumes execution.

Demand Page & Page Fault

- The operating system sets the instruction pointer to the first instruction of the process (which is on a page in secondary memory) before the process running, the process immediately faults for the page.
- Pure demand paging –
  - After this page is brought into memory, the process continues to execute, faulting as necessary until every page that it needs is in memory.
  - At that point, it can execute with no more faults (never bring a page into memory until it is required).

Demand Page & Page Fault

- A given simple instruction could access multiple pages which might cause multiple page faults.
- Consider a three-address instruction such as ADD content of A to B and placing the result in C
  - Fetch and decode instruction ADD
  - Fetch A
  - Fetch B
  - Add A and B
  - Store C
- What if A and B are located in the same page but not C. This instruction might cause two page faults.

Free-Frame List

- When a page fault occurs, the operating system must bring the desired page from secondary storage into to a free page frame in main memory.
- What if there is no free page frame, operating system need extra effort to create a free page frame.
- To resolve page faults with no free page frame, most operating systems maintain a free-frame list, a pool of free frames for satisfying such requests.

Performance of Demand Paging with Page Fault

- Steps for performance of Demand paging with page fault
  1. Trap to the operating system (control change from a process to kernel)
     - OS save the process status in process table
  2. Check page fault or not by checking page table for the process.
  3. Check that the page reference was legal and determine the location of the page on the disk (the secondary memory).
  4. Issue a read from the disk (the secondary memory) to a free frame
  5. Correct the page table. (now the process is ready to implement)
  6. Restore the process status
  7. The process start to run the instruction.

Performance of Demand Paging with Page Fault

- Three major task components of the page-fault service time
  1. Service the page-fault interrupt.
     - Save the process status
     - Find free page frame
     - If there is no page frame, decide victim frame based on replacement algorithm
  2. Read in the page.
  3. Restart the process.
- In case HDD (Hard drive disk)
  - page switch time about 8 milliseconds.
  - A memory access time of 200 nanoseconds.
  - Page-Fault rate = p
  - Effective access time = (1 – p) x 200 +8,000,000 x p
    = 200 + 7,999,800 x p
  - Effective access time is directly proportional to the page-fault rate p

Performance of Demand Paging with Page Fault

- Effective access time = (1 – p) x 200 +8,000,000 x p
  = 200 + 7,999,800 x p
- Effective access time is directly proportional to the page-fault rate p.
- It is important to keep the page-fault rate low in a demand-paging system. Otherwise, the effective access time increases, slowing process execution dramatically.
- An additional aspect of demand paging is the handling and overall use of swap space.
- I/O to swap space is generally faster than that to the file system.
- It is faster because swap space is allocated in much larger blocks, and file lookups and indirect allocation methods are not used

Performance of Demand Paging with Page Fault

- Options to improve better paging throughput with swap space.
  - Copy an entire file image into the swap space at process startup – all demand paging from swap space – overheads for start-up to copy image into the swap space.
  - Initially, demand-page from the file system but to write the pages to swap space as they are replaced. – Linux or Window

Swapping with Paging

- Process instructions and the data must be in memory to be executed. However, a process, or a portion of a process, can be swapped temporarily out of memory to a backing store and then brought back into memory for continued execution.
- Swapping makes it possible for the total physical address space of all processes to exceed the real physical memory of the system.
- Standard swapping involves moving entire processes between main memory and a backing store.
- But it is no longer used in contemporary operating systems, because the amount of time required to move entire processes between memory and the backing store is prohibitive.

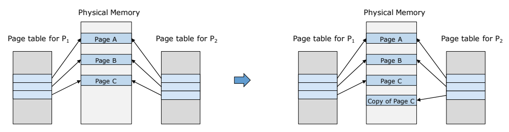
Swapping with Paging

- Most systems, including Linux and Windows, now use a variation of swapping in which pages of a process—rather than an entire process.
- Linux uses virtual memory, so that disk area (swap space) is used as an extension of physical memory for temporary storage when the operating system tries to keep track of processes requiring more physical memory than what is available. When this happens the swap space is used for swapping and paging.

Copy-on-Write between Parent and Child

- Traditionally, fork() worked by creating a copy of the parent's address space for the child, duplicating the pages belonging to the parent.
- Since many child process runs different program by invoking exec() system call, coping of the parent's address space may be unnecessary.
- Copy-on write idea works by allowing the parent and child processes initially to share the same pages.
- These shared pages are marked as copy-on-write pages, meaning that if either process writes (modifies) to a shared page, a copy of the shared page is created.

Copy-on-Write between Parent and Child

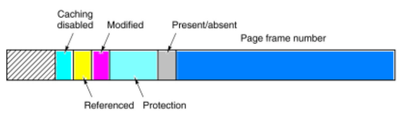


- When process P_1 create a child process p_2 by calling fork() , pages are shared between.
- When process p2 tries to modify page C in RAM, OS find out free page frame and copy page C, then P2 can modify the space now.

Copy-on-Write between Parent and Child

- When a child is created by calling vfork(), the parent process is suspended and the child process use the address space of the parent.
- Since vfork() does not use copy-on-write, if the child process modify any pages of the parent's address space, modified page will be visible to the parent once it resume.

Structure of Page Table Entry



Page Replacement Algorithms

- When a page fault occurs and if there is no free frame, OS need do following steps
  1. Choose a victim page currently allocated in a page frame
  2. If the page has been modified in the memory, rewritten to the disk (secondary memory or swapping area in secondary memory).
  3. A page is allocated into the page frame which was used by the victim page
  4. Change page table.
- First step is dependant on replacement algorithm
- If no frames are free, two page transfers (one for the page-out and one for the page-in) are required.

Page Replacement Algorithms

- A process's Memory access can be characterized by an list of page number
- This list is called the **reference string (sequence of page number)**.
- A paging system can be characterized by three items
  1. The reference string of the executing process
  2. The page replacement algorithm
  3. The number of page frames available in memory for a process

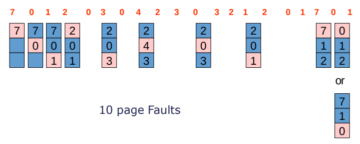
Page Replacement Algorithms
(Optimal Algorithm)

Optimal Algorithm

- Replace the page that will not be used for the longest period of time.
- Optimal algorithm always guarantees the lowest possible page-fault rate for a fixed number of frames
- Unfortunately, the optimal replacement algorithm is difficult to implement, since it requires future knowledge of the reference string.

Page Replacement Algorithms
(Optimal Algorithm)

Optimal Algorithm



10 page Faults

Page Replacement Algorithms
(Not Recently Used (NRU))

Not Recently Used (NRU)

- When page fault occurs, the operating system inspects all the pages and classified into four group base on the page table information (Modified bit, Reference bit).
  - Class 0: Not referenced, not modified
  - Class 1: not referenced, modified
  - Class 2: referenced, not modified
  - Class 3: referenced, modified
- The NRU algorithm removes a page at random from the lowest numbered nonempty class.

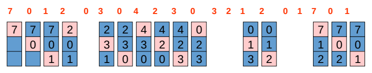
Page Replacement Algorithms
(First In First Out (FIFO))

First In First Out (FIFO)

- Replace the oldest page frame
- The FIFO algorithm is easy to understand and easy to program.
- However, its performance is not always good.

Page Replacement Algorithms
(First In First Out (FIFO))

First In First Out



15 Page Faults

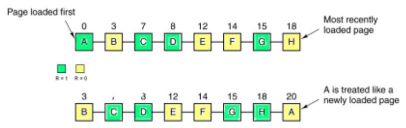
Page Replacement Algorithms
(Second Chance)

Second Chance

- A simple modification of FIFO that avoids the problem of throwing out a heavily used is to inspect the R bit of the oldest pages.
- If the oldest page's R = 0, it is old and not referenced. So it remove from the page frame.
- If the oldest page's R = 1, it is old but referenced. So set R = 0 and set the page from oldest to newest page

Page Replacement Algorithms
(Second Chance)

Second Chance



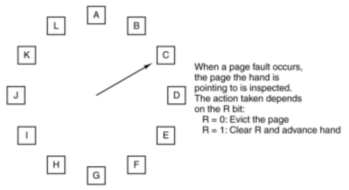
Page Replacement Algorithms
(The Clock Page Replacement Algorithm)

The Clock Page Replacement Algorithm.

- Similar with second chance, but it keep all the page frames on a circular list.
- When page fault occurs, the page the hands is pointing to is inspected.
- Action taken depends on the Reference bit R.
  - If R = 0 evict the page
  - If R = 1, Clear R and advance hand

Page Replacement Algorithms
(The Clock Page Replacement Algorithm)

The Clock Page Replacement Algorithm


A
B
C
D
E
F
G
H
I
J
K
L
When a page fault occurs, the page the hand is pointing to is inspected. The action taken depends on the R bit:
R = 0: Evict the page
R = 1: Clear R and advance hand

Page Replacement Algorithms
(The Least Recently Used)

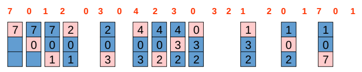
The Least Recently Used

- Replace the page that has not been used for the longest period of time
- This is the optimal page replacement algorithm looking backward in time.
- LRU algorithms works quiet good but it may require substantial hardware assistance to keep track the information.

- LRU can be implemented by
  - Counter – save the reference time
  - Stack – keep a stack of page number

Page Replacement Algorithms
(The Least Recently Used)

Least Recently Used



12 Page Faults