Lecture 10: Virtual Memory

- Reason for virtual memory & what it is.
- Demand Paging
- Performance of Demand Paging
- Page Replacement
- Page-Replacement Algorithms
- Allocation of Frames
- Thrashing

Virtual memory - the need for it.

- Memory is a limited resource
- the memory required by the various processes that want to run on the system may be greater than the total physical memory
- using the memory allocation mechanisms we have seen so far, this would mean that the long term scheduler (one that decides whether to allow processes into the system) would have to refuse to run some of the processes until some of the processes have finished and can release memory back to the OS

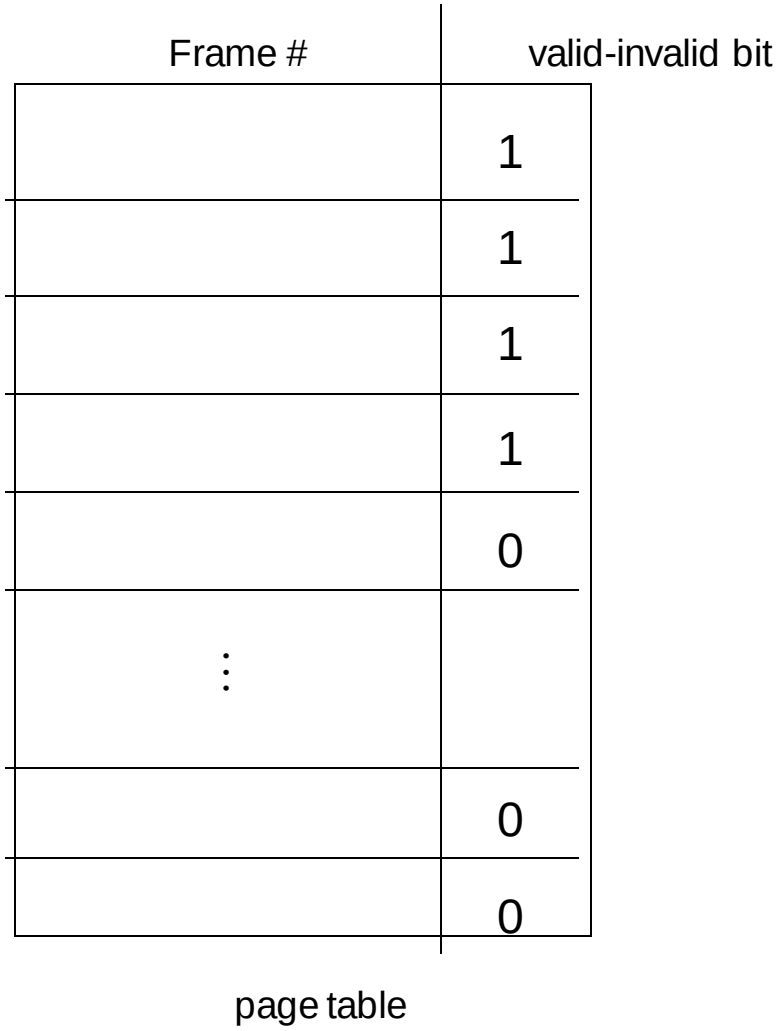
- virtual memory is used to get round this problem - it is an attempt to allow a computer with ***X*** amount of physical memory to be able to run even when the memory required by processes running on the system is greater than ***X***
- to do this only a proportion of a process' memory is held in physical memory (some of it is left on hard disk), so more proceses can be fitted into a given area of physical memory
- So need some scheme which will determine which areas of the process' memory needs to be in physical memory - allocation scheme

- in general the idea is to only keep in physical memory those items of data and instructions that are currently being used or are likely to be used in the near future - implemented using **demand paging** or **demand segmentation**
- also when physical memory that is available to a process becomes full and new memory needs to be brought in, then OS needs some schme which will determine which memory needs to be replaced with the new memory that is being brought in - this is called a **replacement policy**

Demand Paging

- Bring a page into memory only when it is needed.
- Uses a lazy approach to bringing into memory
- a process makes a reference to a page of its memorywhile running on the CPU
  - if reference is invalid (e.g. no such program address) then abort process
  - if page not in memory then fetch into memory

Valid-Invalid Bit



- A valid-invalid bit is associated with each page table entry (1 means the page is in memory, 0 means the page is not in memory)
- Initially valid–invalid bit is set to 0 on all entries.
- During adddress translation, if valid–invalid bit in page table entry is 0 then a **page fault** is generated

Page Fault

- **Page fault** is an interrupt that signals the OS that a page that has been referenced by a running process is not in memory
- OS looks at table in Process Control Block to decide whether:
  - the reference is invalid (address not in processes address space) then abort OR
  - Address not in memory in which case it needs to be brought into memory
- The first reference to a page will generate a page fault

Page Fault (Cont.)

- To bring in new page requires:
- Finding empty frame.
- Swapping page into frame.
- Updating page table and PCB table, setting valid bit = 1.
- Restart instruction that made memory reference, but now page with address in it is in memory

What happens if there is no free frame?

- **Page replacement** – find some page in memory that is not being used very much and replace it with new page.
  - performance – want an algorithm which will result in minimum number of page faults.
- Same page may be brought in/out of memory several times during lifetime of process

Performance of Demand Paging

- Page Fault Rate $0 \le p \le 1.0$
  - if $p = 0$ no page faults
  - if $p = 1$, every reference generates a fault
- Effective Access Time (EAT)

  EAT = $(1 - p)$ * physical memory access time

  + p * (page fault overhead + time to copy page out (if modified) + time to copy page in + restart overhead + physical memory access time)
- Page fault/restart overheads include - switching process contexts, handling interrupt, finding free frame, initiating disk transfer, updating tables, etc.

Demand Paging Example

- For example we take memory access time = 100 nanoseconds and 50% of the time the page that is being replaced has been modified and therefore needs to be copied back out to disk. We will ignore page fault and restart overheads.

- Swap Page Time = 10 msec = 10 million nanosec

  EAT = (1 – p) * 100 + p (15 million + 100) nanosec

  = 100 - p*100 + p*15 million + p*100 (in nanosec)

  = 100 + p * 15 million (in nanosec)

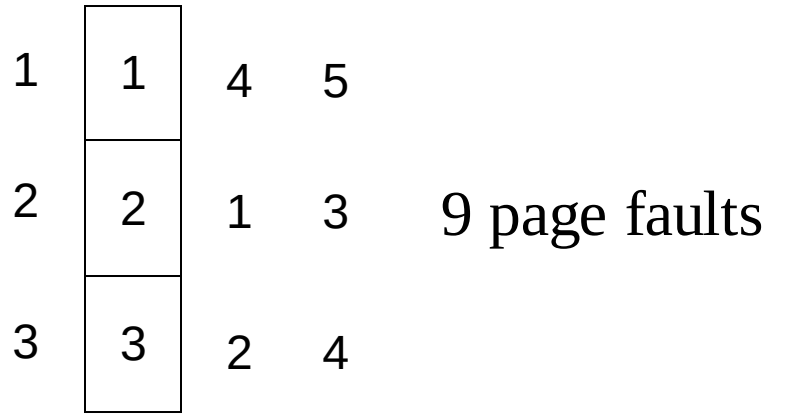
- size of page fault rate has huge affect on effective access time

Page-Replacement Algorithms

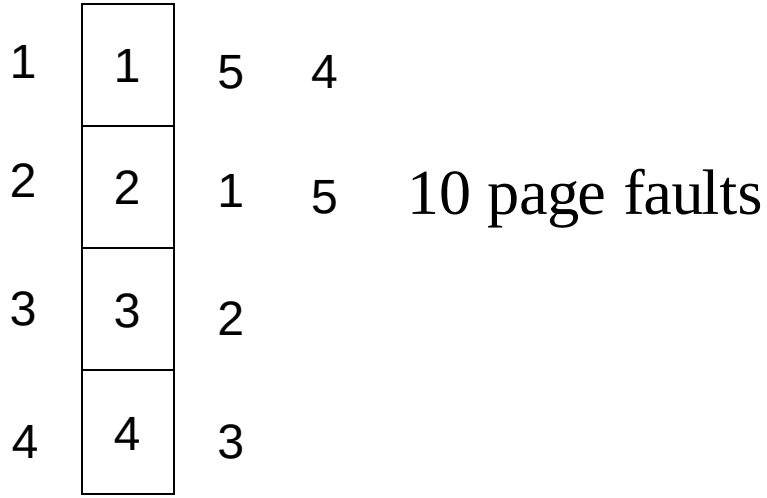
- Want lowest page-fault rate.
- We can examine performance of an algorithm by running it on a particular string (sequence) of memory references (reference string) in which we are only interested in the page number of the memory reference
- we can then compute for each algorithm the number of page faults that would occur with that particular sequence of memory references.
- In all examples that follow, the reference string is

  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

First-In-First-Out (FIFO) Algorithm



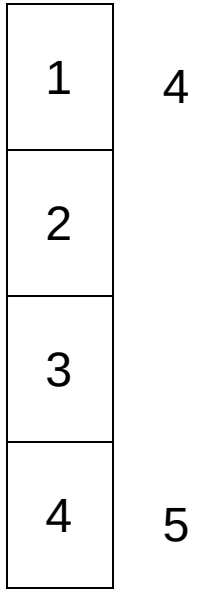
9 page faults



10 page faults

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)
- 4 frames - sometimes using more actual frames can produce more page faults – known as Belady's Anomaly

An Optimal Algorithm



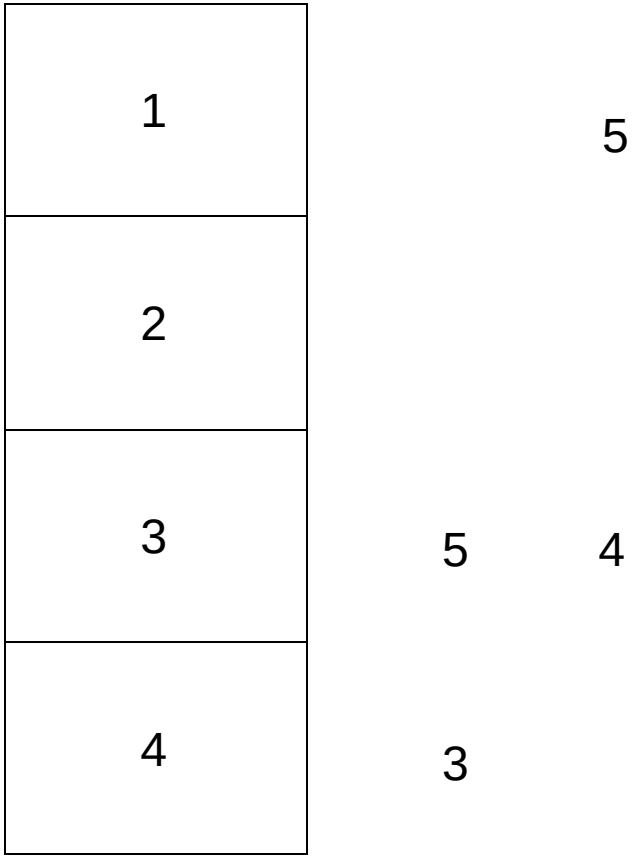
- Replace page that will not be used for longest period of time in the future.
- 4 frames example

  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

- How do you which page will be the one that is not referenced for the longest time in the future? - Can't know!
- It is used as benchmark in comparison of algorithms.
- But it can be approximated by taking recent past as guide to future and replacing page that has not been used for longest period of time - Least Recently Used

Least Recently Used (LRU) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



8 Page Faults

LRU Algorithm (Cont.)

- implementation of LRU using counters
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter.
  - When a page needs to be changed, look at the counters to determine which is oldest.
- implementation of LRU using a stack
  - keep a stack of page numbers in a double linked list:
  - when a page is referenced - move it to the top of the stack
  - Bottom of stack always LRU page
  - Updating stack requires changing set of pointer values

Allocation of Frames

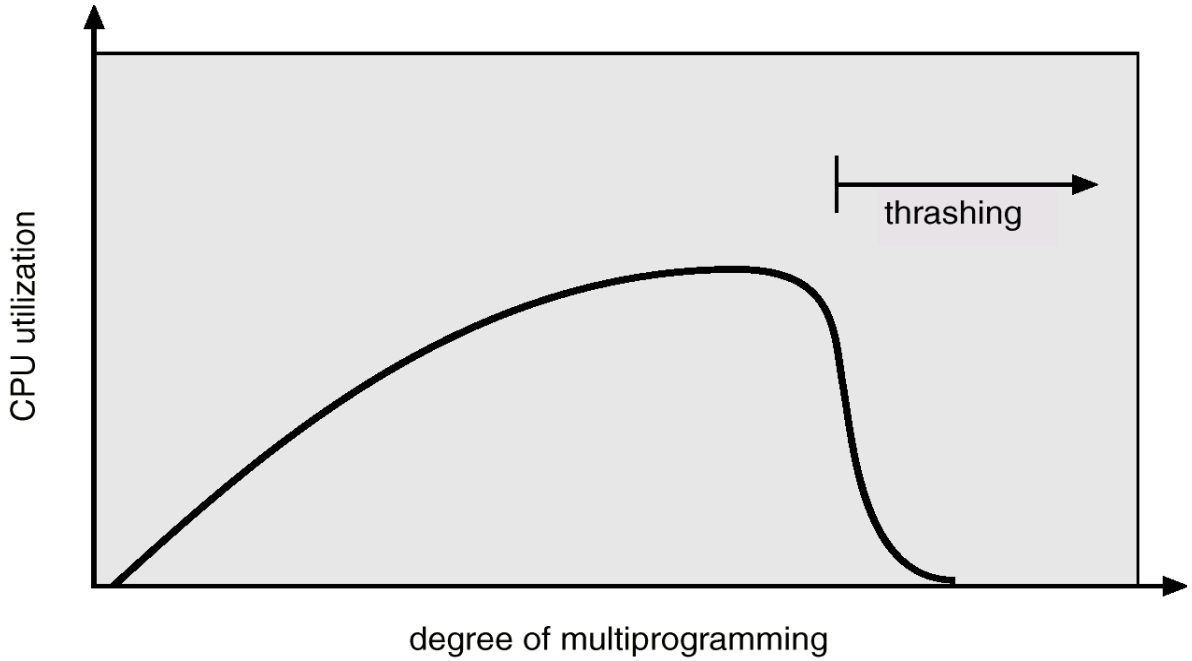
- Each process needs minimum number of pages to be able to operate effectively.
- Two major allocation schemes.
  - Equal allocation
  - proportional allocation
- Equal allocation between processes – e.g. if 100 frames and 5 processes, give each 20 pages.
- Proportional allocation – Allocate according to the size of process.

Global vs. Local Replacement

- Global replacement – replacement mechanism selects a page for replacment in a frame from the set of all frames; replacement of pages can use frames for replacement that currently belong to a different process from the one generating the page fault.
- Local replacement – replacement mechanism only selects a page for replaceemnt from a set of frames that have been allocated to the process generating the page fault.

Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high. This can lead to low CPU utilization.
- Thrashing ≡ a process is busy swapping pages in and out from disk - very time consuming.



Thrashing

- Locality model
  - Set of addresses/pages a process accesses over a period of time during execution tend to form a stable set - known as a **locality**
  - after a period of time the process will start acessing a new set of addresses/pages which in turn will form a new stable set.
  - Localities may overlap i.e. some pages used throughout lifetime of program
- Why does thrashing occur?
  Sum of sizes of current localities of all processes on system > total hardware memory size

Demand Segmentation

- Used when insufficient hardware to implement demand paging.
- OS/2 allocates memory in segments, which it keeps track of through segment descriptors
- Segment descriptor contains a valid bit to indicate whether the segment is currently in memory.
  - If segment is in main memory, access continues,
  - If not in memory, segment fault.