CMSC 412

Virtual Memory (Part II)

Virtual Memory

- Paging is a typical feature of broader support for **virtual memory**
- Pages can be swapped to and from disk
  - Presents the illusion of a larger physical memory
- Virtual addressing simplifies model
  - Swapping: can load pages into different addresses in physical memory
  - Addressing: different processes have their own address spaces, starting at 0, separate from other processes

Elements of Virtual Memory

- Page Fault Handler
  - What happens when the referenced page is not accessible?
- Page Allocation and Replacement
  - If page is on disk, swapper must load it into memory and update the page table.
  - If no pages available to load into memory, must evict a page to make room
    - Which page? Depends on the page replacement algorithm.

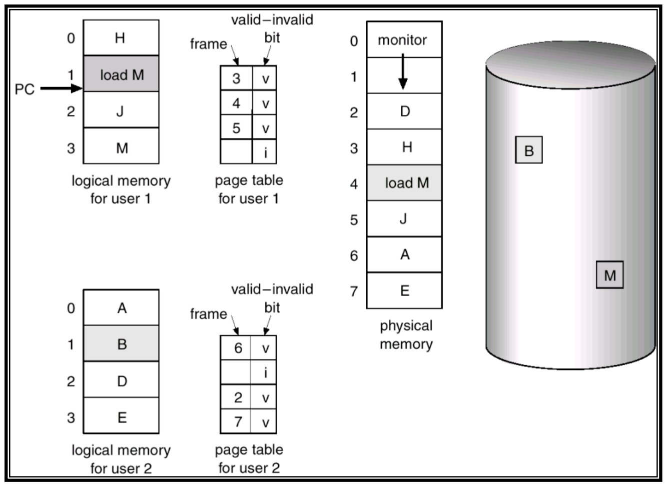
Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement.

- Use *modify* (*dirty*) *bit* to reduce overhead of page transfers – only modified pages are written to disk.

- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory.

Need For Page Replacement



Basic Page Replacement

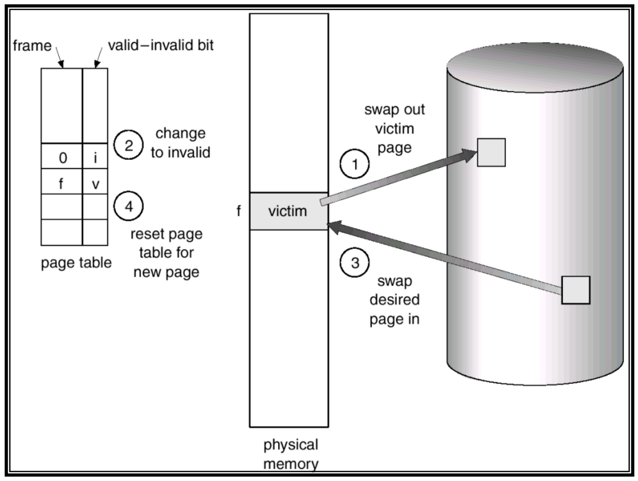
Find the location of the desired page on disk.

Find a free frame:
- If there is a free frame, use it.
- If there is no free frame, use a page replacement algorithm to select a *victim* frame (may need to write this frame to disk).

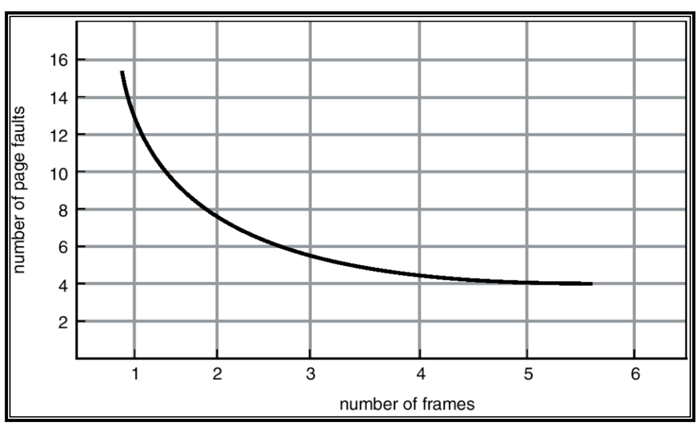
Read the desired page into the (newly) free frame. Update the page and frame tables.

Restart the process.

Page Replacement



Expected Page Faults Versus The Number of Frames



FIFO Page Replacement

- Replace the page that was brought in longest ago
- However
  - Old pages may be great pages (frequently used)
  - Belady's anomaly: number of page faults may increase when one increases number of page frames!

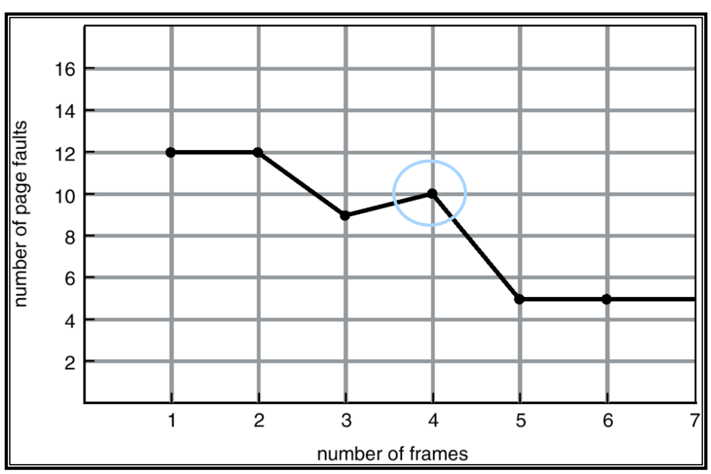
FIFO Example (3 frames)

- Reference string: 1,2,3,4,1,2,5,1,2,3,4,5
  - access 1 - (1) fault
  - access 2 - (1,2) fault
  - access 3- (1,2,3) fault
  - access 4 - (2,3,4) fault, replacement
  - access 1 - (3,4,1) fault, replacement
  - access 2 - (4,1,2) fault, replacement
  - access 5 - (1,2,5) fault, replacement
  - access 1- (1,2,5)
  - access 2 - (1,2,5)
  - access 3 - (2,5,3) fault, replacement
  - access 4 - (5,3,4) fault, replacement
  - access 5 - (5,3,4)
- 9 page faults

FIFO Example (4 frames)

- Reference string: 1,2,3,4,1,2,5,1,2,3,4,5
  - access 1 - (1) fault
  - access 2 - (1,2) fault
  - access 3- (1,2,3) fault
  - access 4 - (1,2,3,4) fault
  - access 1 - (1,2,3,4)
  - access 2 - (1,2,3,4)
  - access 5 - (2,3,4,5) fault, replacement
  - access 1- (3,4,5,1) fault, replacement
  - access 2 - (4,5,1,2) fault, replacement
  - access 3 - (5,1,2,3) fault, replacement
  - access 4 - (1,2,3,4) fault, replacement
  - access 5 - (2,3,4,5) fault, replacement
- 10 Page faults

FIFO: Belady's Anamoly



Optimal Page Replacement

- Replace the page that will be used furthest in the future
- Good algorithm(!) but requires knowledge of the future
  - With good compiler assistance, knowledge of the future is sometimes possible.
- Used as point of comparison with other algorithms.

Optimal Example (4 frames)

- Reference string: 1,2,3,4,1,2,5,1,2,3,4,5
  - access 1 - (1) fault
  - access 2 - (1,2) fault
  - access 3- (1,2,3) fault
  - access 4 - (1,2,3,4) fault
  - access 1 - (1,2,3,4)
  - access 2 - (1,2,3,4)
  - access 5 - (1,2,3,5) fault, replacement
  - access 1 - (1,2,3,5)
  - access 2 - (1,2,3,5)
  - access 3 - (1,2,3,5)
  - access 4 - (4,2,3,5) fault, replacement
  - access 5 - (4,2,3,5)
- 6 Page faults

LRU Page Replacement

- Replace the page that was used longest ago
  - Best known non-clairvoyant algorithm.
- Implementation of LRU can be expensive
  - Maintain a stack of nodes representing pages and put page on top of stack when the page is accessed
  - Maintain a time stamp associated with each page

LRU Example (3 frames)

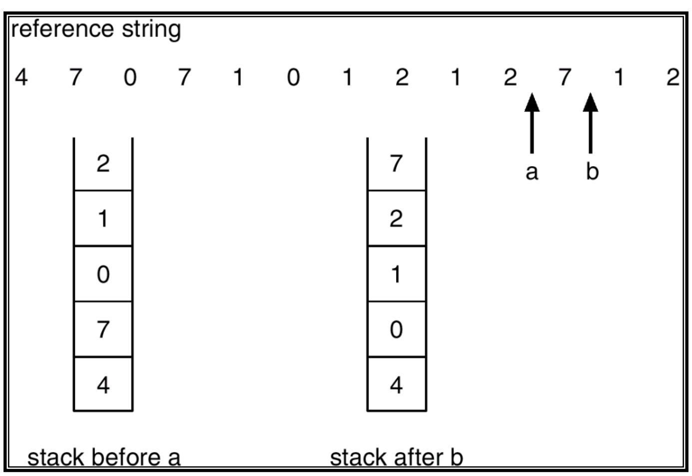
- Reference string: 1,2,3,4,1,2,5,1,2,3,4,5
  - access 1 - (1) fault
  - access 2 - (1,2) fault
  - access 3- (1,2,3) fault
  - access 4 - (2,3,4) fault, replacement
  - access 1 - (3,4,1) fault, replacement
  - access 2 - (4,1,2) fault, replacement
  - access 5 - (1,2,5) fault, replacement
  - access 1- (2,5,1)
  - access 2 - (5,1,2)
  - access 3 - (1,2,3) fault, replacement
  - access 4 - (2,3,4) fault, replacement
  - access 5 - (3,4,5) fault, replacement
- 10 page faults

LRU Example (4 frames)

– Reference string: 1,2,3,4,1,2,5,1,2,3,4,5
  - access 1 - (1) fault
  - access 2 - (1,2) fault
  - access 3- (1,2,3) fault
  - access 4 - (1,2,3,4) fault
  - access 1 - (2,3,4,1)
  - access 2 - (3,4,1,2)
  - access 5 - (4,1,2,5) fault, replacement
  - access 1- (4,2,5,1)
  - access 2 - (4,5,1,2)
  - access 3 - (5,1,2,3) fault, replacement
  - access 4 - (1,2,3,4) fault, replacement
  - access 5 - (2,3,4,5) fault, replacement

– 8 faults

Stack Implementation of LRU



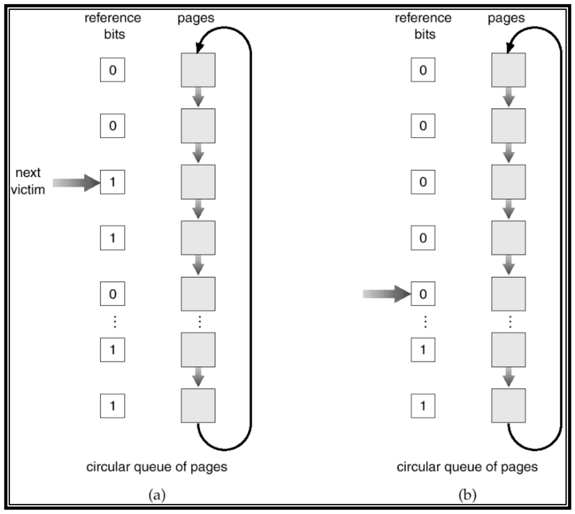
Approximate LRU

- Maintain reference bit(s) which are set (by HW) whenever a page is used
  - at the end of a given time period (e.g., at a page fault, regular timeout, when idle…), reference bits are cleared.
  - How does the interval affect performance?
- Replace the one which is 0 (if one exists, else FIFO).

Second-chance Algorithm

- For each page, maintain
  - Reference bit (by HW)
  - Page Arrival time (by OS)
- If page to be replaced (in clock order) has reference bit = 1. then:
  - set reference bit 0, set arrival time.
  - leave page in memory.
  - replace next page (in clock order), subject to same rules.

Second-Chance (clock) Page-Replacement Algorithm



Allocation of Frames

- Each process needs **minimum** number of pages.
- Example: IBM 370 – 6 pages to handle SS MOVE instruction:
  - instruction is 6 bytes, might span 2 pages.
  - 2 pages to handle **from**.
  - 2 pages to handle **to**.
- Two major allocation schemes.
  - fixed allocation
  - priority allocation

Fixed Allocation

- Equal allocation – e.g., if 100 frames and 5 processes, give each 20 pages.
- Proportional allocation – Allocate according to the size of process.

Priority Allocation

- Use a proportional allocation scheme using priorities rather than size.
- If process P_i generates a page fault,
  - select for replacement one of its frames.
  - select for replacement a frame from a process with lower priority number.

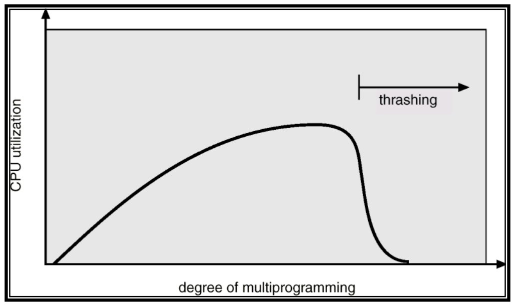
Global vs. Local Allocation

- **Global** replacement – process selects a replacement frame from the set of all frames; one process can take a frame from another.
- **Local** replacement – each process selects from only its own set of allocated frames.
- Tradeoff: predictability vs. throughput

Thrashing

- Can allocate so much virtual memory that the system spends all its time getting pages
  - the situation is called thrashing
- To stop this, swap a process
  - write all of the memory of a process out to disk
  - don't run the process for a period of time
  - part of medium term scheduling
- Why does thrashing happen? How to identify it?

Paging and Thrashing

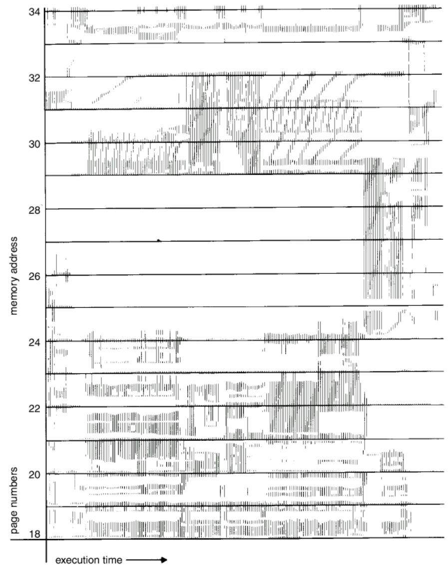


- Why does paging work?
  Locality model
  - Process migrates from one locality to another.
  - Localities may overlap.
- Why does thrashing occur?
  Σ size of locality > total memory size

Working Sets and Locality

- Programs usually display reference locality
  - Temporal locality
    - repeated access to the same memory location
  - Spatial locality
    - consecutive memory references to nearby memory locations
  - *Memory hierarchy* design relies heavily on assumed locality of reference
- Working set
  - set of pages referenced in the last Δ references.

Locality In Memory References



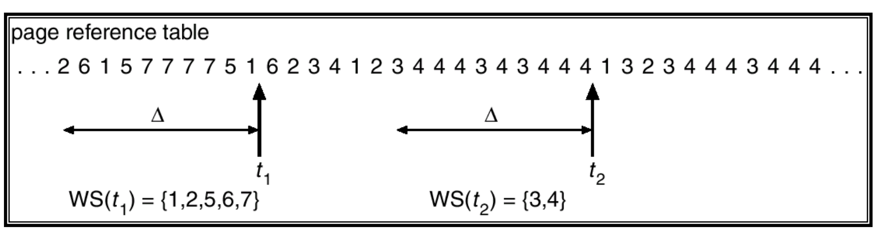
Improving Locality

- Malloc may not ensure spatial locality
  - Two calls to malloc could get memory on different cache lines, pages, etc.
  - What about the stack?
- Option 1:
  - Malloc a large chunk of memory and parcel it out yourself
- Option 2:
  - Add a “near” hint parameter to malloc
  - Indicates that memory should be allocated near the target location
    - It’s only a performance hint, and malloc can ignore it
    - Allows locality improvement without major changes

Considering Working Sets

- $\Delta \equiv$ working-set window $\equiv$ a fixed number of page references
  Example: 10,000 instruction
- WSS_i (working set of Process P_i) = total number of pages referenced in the most recent Δ (varies in time)
  - if Δ too small will not encompass entire locality.
  - if Δ too large will encompass several localities.
  - if $\Delta = \infty \Rightarrow$ will encompass entire program.
- $D = \Sigma\ WSS_i \equiv$ total demand frames
- if $D > m \Rightarrow$ Thrashing
- Policy if D > m, then suspend one of the processes. (Which?)

Working-set Example

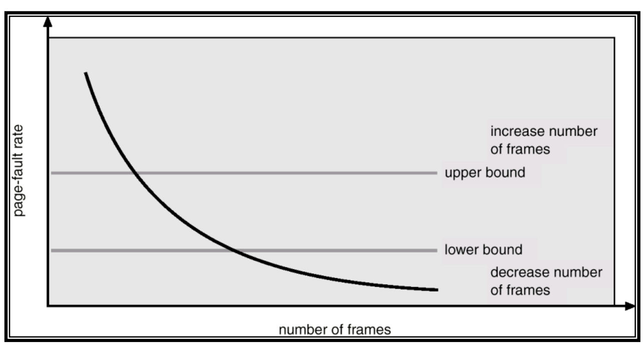


16

Tracking the Working Set

- Approximate with interval timer + a reference bit
- Example: Δ = 10,000
  - Timer interrupts after every 5000 time units.
  - Keep in memory 2 bits for each page.
  - Whenever a timer interrupts copy reference bits to memory, then reset them to 0.
  - If one of the bits in memory = 1 $\Rightarrow$ page in working set.
- Why is this not completely accurate?
- Improvement = 10 bits and interrupt every 1000 time units.

Page-Fault Frequency Scheme



- Establish “acceptable” page-fault rate.
  - If actual rate too low, process loses frame.
  - If actual rate too high, process gains frame.

Implementation Issues

- How big should a page be?
  - Want to trade cost of fragmentation vs.:
    - Cost of **fault**: trap + seek + latency + transfer
    - Cost of access: whether in/out of the TLB
  - Must OS page size be equal the HW page size?
    - no, just needs to be a multiple of it
- How does I/O relate to paging
  - to request I/O for a process, must lock the page
    - if not, the I/O device can overwrite the page
- Can the kernel be paged?
  - most of it can be.
  - what about the code for the page fault handler?

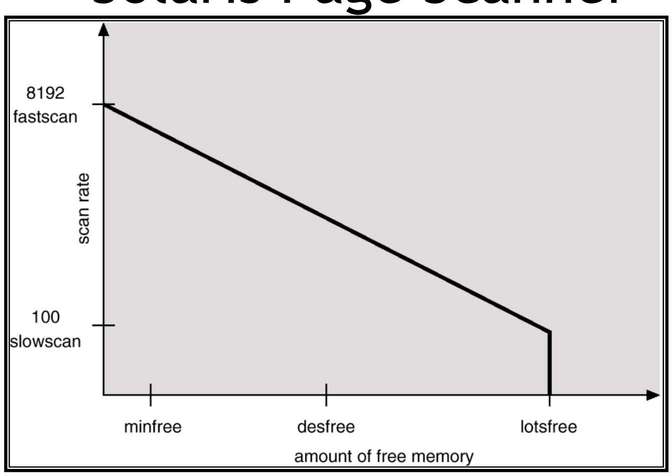
Windows NT

- Uses demand paging with **clustering**. Clustering brings in pages surrounding the faulting page.
- Processes are assigned **working set minimum** and **working set maximum**.
- Working set minimum is the minimum number of pages the process is guaranteed to have in memory.
- A process may be assigned as many pages up to its working set maximum.
- When the amount of free memory in the system falls below a threshold, **automatic working set trimming** is performed to restore the amount of free memory.
- Working set trimming removes pages from processes that have pages in excess of their working set minimum.

Solaris 2

- Maintains a list of free pages to assign faulting processes.
- **Lotsfree** – threshold parameter to begin paging.
- Paging is peformed by *pageout* process.
- Pageout scans pages using modified clock algorithm.
- **Scanrate** is the rate at which pages are scanned. This ranged from **slowscan** to **fastscan**.
- Pageout is called more frequently depending upon the amount of free memory available.

Solaris Page Scanner



19