CS370 Operating Systems

Colorado State University
Yashwant K Malaiya

Fall 2016 Lecture 34

Virtual Memory

Slides based on

- Text by Silberschatz, Galvin, Gagne
- Various sources

FAQ

- Does Belady's Anomaly affect all page replacement algorithms?
- When Belady's Anomaly is present in an algorithm, will adding an additional frame always cause more page faults?
- Paging algorithms seem to assume both temporal and spatial locality… how true is this?

Page Replacement Algorithms

Algorithms

- FIFO
- "Optimal"
- The Least Recently Used (LRU)
  - Exact Implementations
    - Time of use field, Stack
  - Approximate implementations
    - Reference bit
    - Reference bit with shift register
    - Second chance: clock
    - Enhanced second chance: dirty or not?
- Other

Clever Techniques: Page-Buffering Algorithms

- Keep a buffer (pool) of free frames, always
  - Then frame available when needed, not found at fault time
  - Read page into free frame and select victim to evict and add to free pool
  - When convenient, evict victim
- Keep list of modified pages
  - When backing store is otherwise idle, write pages there and set to non-dirty (being proactive!)
- Keep free frame previous contents intact and note what is in them
  - If referenced again before reused, no need to load contents again from disk
  - Generally useful to reduce penalty if wrong victim frame selected

Buffering and applications

- Some applications (like databases) often understand their memory/disk usage better than the OS
  - Provide their own buffering schemes
  - If both the OS and the application were to buffer
    - Twice the I/O is being utilized for a given I/O
  - OS may provide “raw access” disk to special programs without file ystem services.

Allocation of Frames

Allocation of Frames

How to allocate frames to processes?

- Each process needs ***minimum*** number of frames

  Depending on specific needs of the process

- ***Maximum*** of course is total frames in the system

* Two major allocation schemes
  - fixed allocation
  - priority allocation
* Many variations

Fixed Allocation

- **Equal allocation** – For example, if there are 100 frames (after allocating frames for the OS) and 5 processes, give each process 20 frames
  - Keep some as free frame buffer pool
- **Proportional allocation** – Allocate according to the size of process (need based)
  - Dynamic as degree of multiprogramming, process sizes change

$$s_i = \text{size of process } p_i$$

$$S = \sum s_i$$

$$m = \text{total number of frames}$$

$$a_i = \text{allocation for } p_i = \frac{s_i}{S} \times m$$

$$m = 64$$

$$s_1 = 10$$

$$s_2 = 127$$

$$a_1 = \frac{10}{137} \times 62 \approx 4$$

$$a_2 = \frac{127}{137} \times 62 \approx 57$$

Colorado State University

Priority Allocation

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

Colorado State University

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
  - But then process execution time can vary greatly
  - But greater throughput, so more common

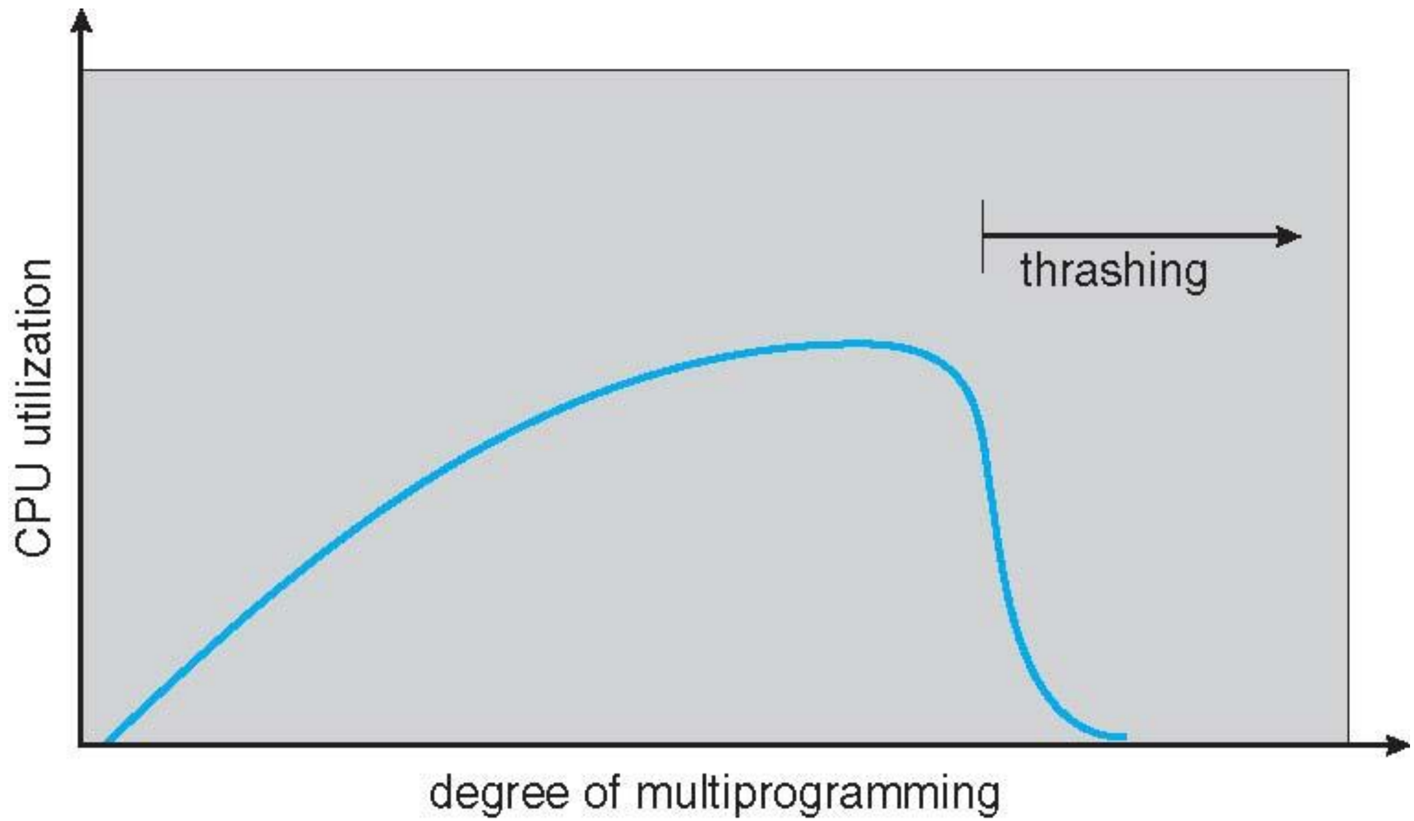
- **Local replacement** – each process selects from only its own set of allocated frames
  - More consistent per-process performance
  - But possibly underutilized memory

Problem: Thrashing

- If a process does not have "enough" pages, the page-fault rate is very high
  - Page fault to get page
  - Replace existing frame
  - But quickly need replaced frame back
  - This leads to:
    - Low CPU utilization, leading to
    - Operating system thinking that it needs to increase the degree of multiprogramming leading to
    - Another process added to the system

- **Thrashing** $\equiv$ a process is busy swapping pages in and out

Thrashing (Cont.)



Colorado State University

Demand Paging and Thrashing

- Why does demand paging work?
  Locality model
  - Process migrates from one locality to another
  - Localities may overlap

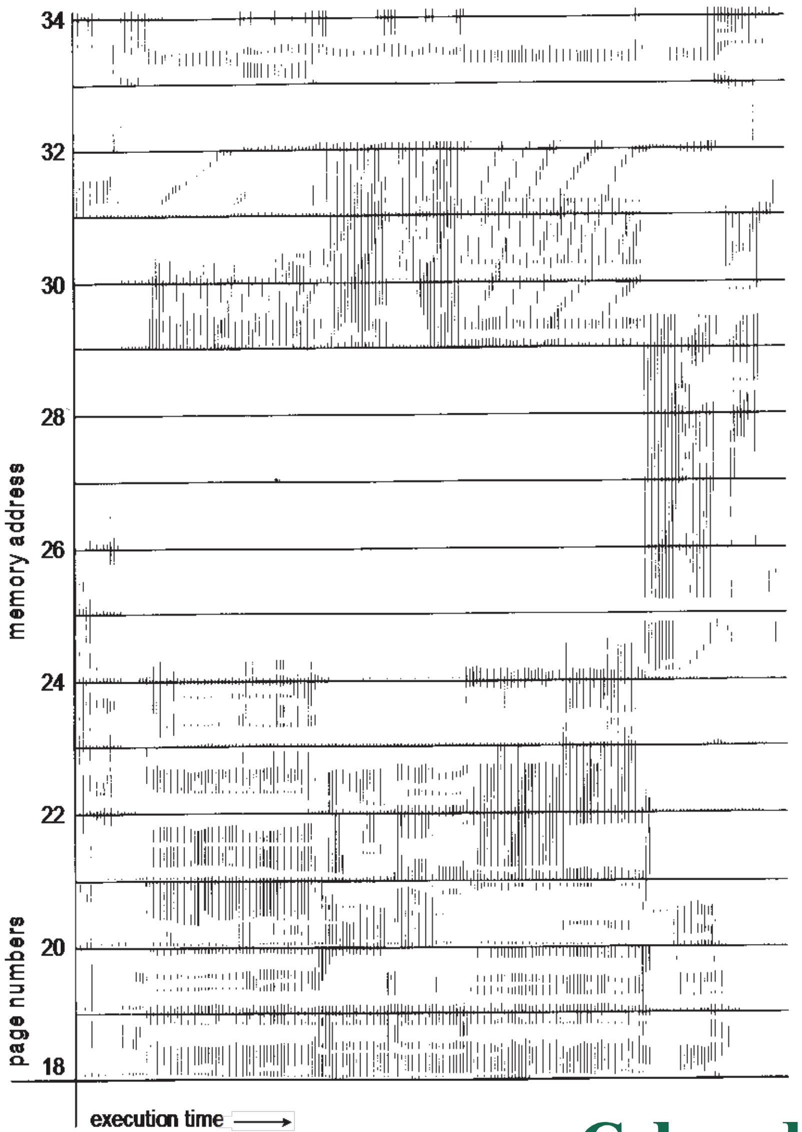
- Why does thrashing occur?

  Σ size of locality > total memory size

  - Limit effects by using local or priority page replacement

Colorado State University

Locality In A Memory-Reference Pattern

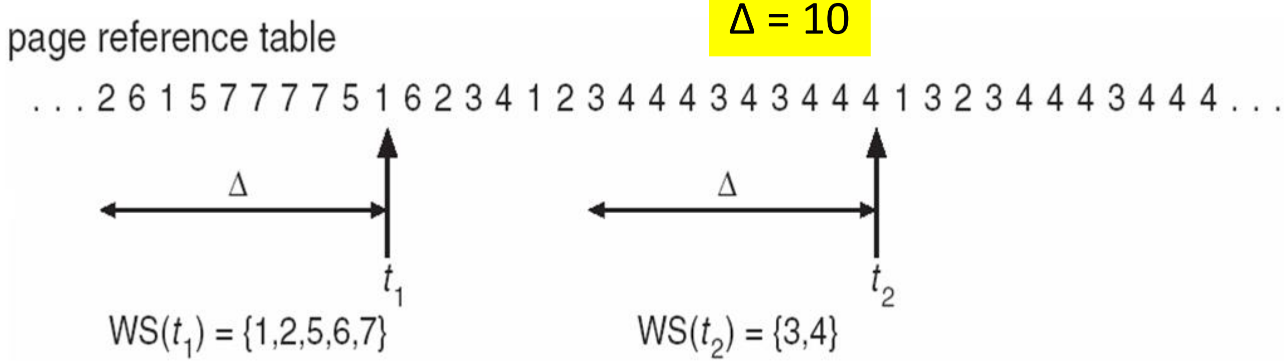


Colorado State University

Working-Set Model

- $\Delta \equiv$ **working-set window** $\equiv$ a fixed number of page references
  Example: 10,000 instructions

$\Delta = 10$

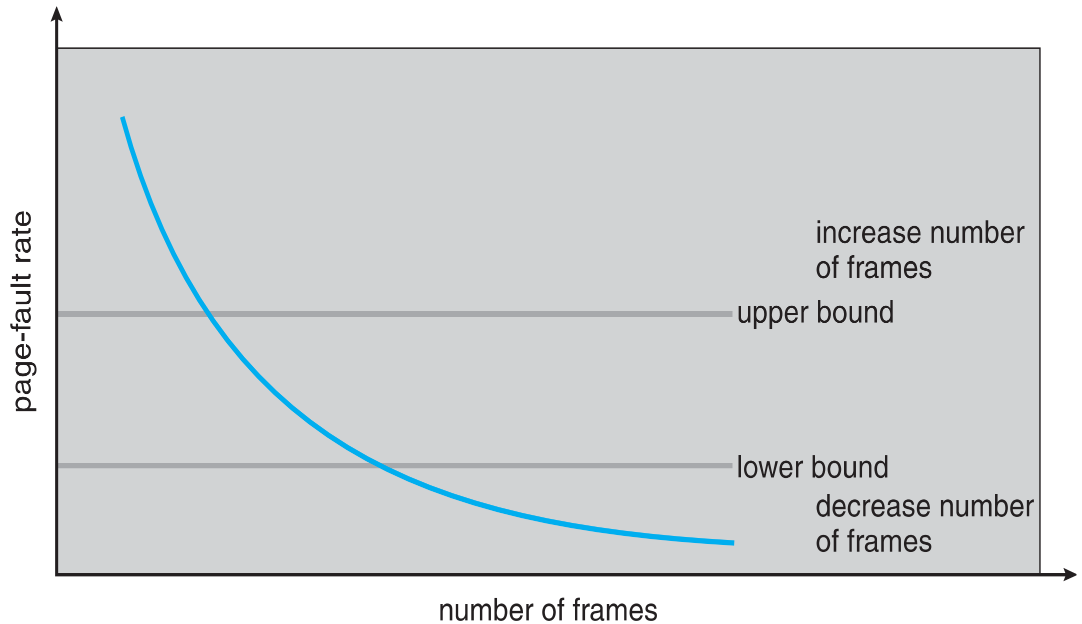


- ***WSS$_i$* (working set of Process *P$_i$*)** =
  total number of pages referenced in the most recent Δ (varies in time)
  - if Δ too small will not encompass entire locality
  - if Δ too large will encompass several localities
  - if $\Delta = \infty \Rightarrow$ will encompass entire program
  - Approximation of locality

- **$D = \Sigma\, WSS_i \equiv$ total demand frames**
  - **if $D > m \Rightarrow$ Thrashing**
  - **Policy** if $D > m$, then suspend or swap out one of the processes

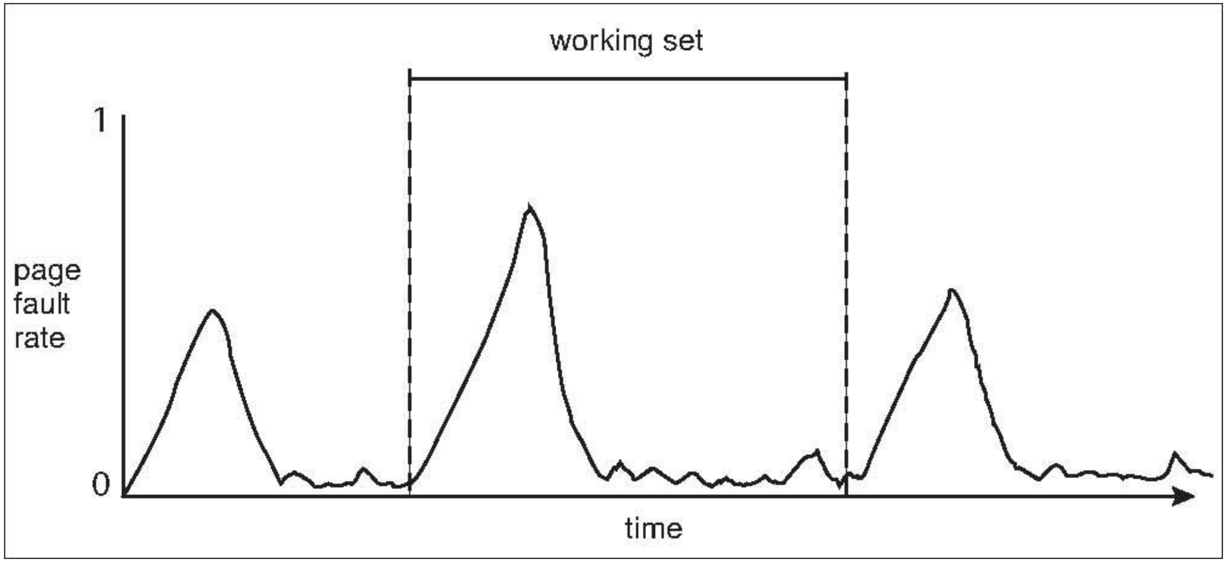
Page-Fault Frequency Approach

- More direct approach than WSS
- Establish "acceptable" **page-fault frequency** (**PFF**) rate and use local replacement policy
  - If actual rate too low, process loses frame
  - If actual rate too high, process gains frame



Working Sets and Page Fault Rates

- Direct relationship between working set of a process and its page-fault rate
- Working set changes over time
- Peaks and valleys over time



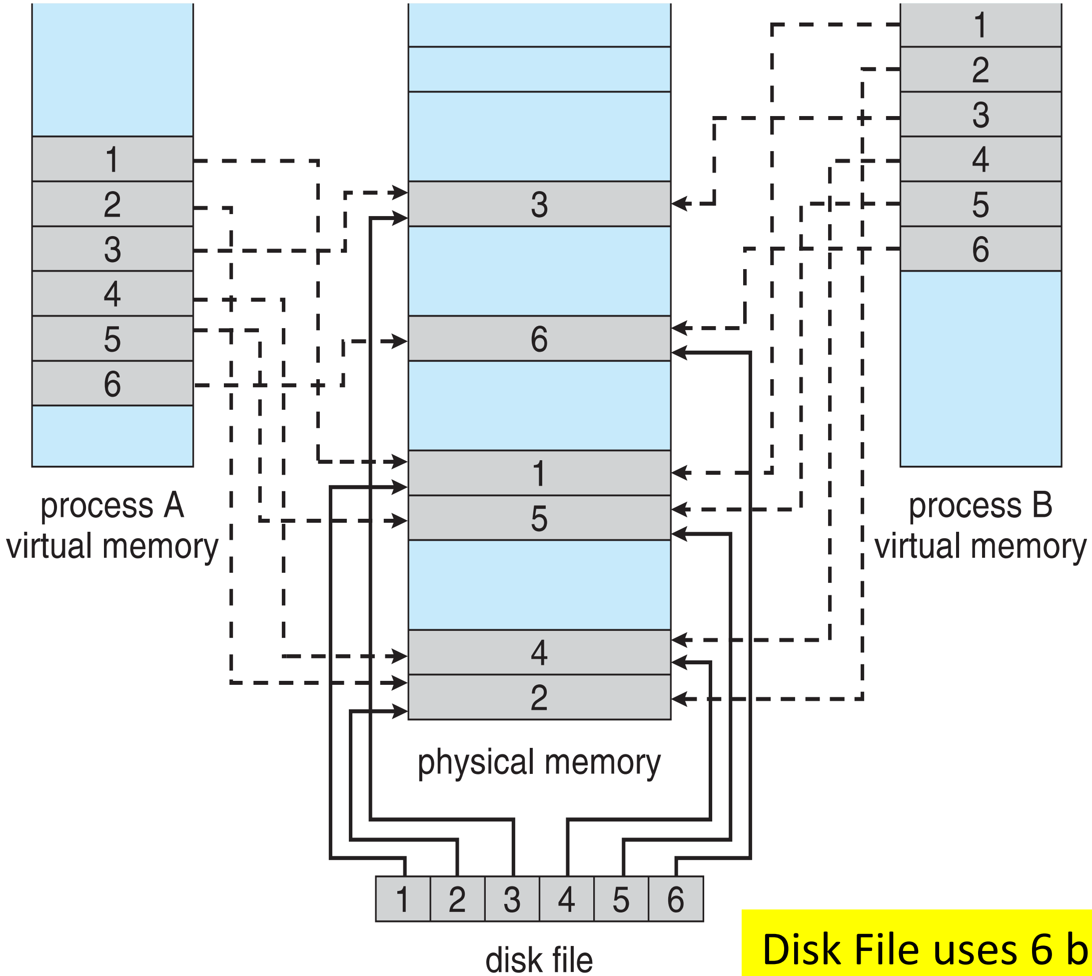
Peaks occur at locality changes: 3 working sets

Colorado State University

Memory-Mapped Files

- Memory-mapped file I/O allows file I/O to be treated as routine memory access by **mapping** a disk block to a page in memory
- File is then in memory instead of disk
- A file is initially read using demand paging
  - A page-sized portion of the file is read from the file system into a physical page
  - Subsequent reads/writes to/from the file are treated as ordinary memory accesses
- Simplifies and speeds file access by driving file I/O through memory rather than `read()` and `write()` system calls
- Also allows several processes to map the same file allowing the pages in memory to be shared
- But when does written data make it to disk?
  - Periodically and / or at file `close()` time
  - For example, when the pager scans for dirty pages

Memory Mapped Files



Disk File uses 6 blocks
Page tables used for mapping

Colorado State University

Allocating Kernel Memory

Allocating Kernel Memory

- Treated differently from user memory
- Often allocated from a free-memory pool
  - Kernel requests memory for structures of varying sizes
    - Process descriptors, semaphores, file objects etc.
    - Often much smaller than page size
  - Some kernel memory needs to be contiguous
    - I.e. for device I/O
  - approaches (skipped)

Other Considerations

Other Considerations -- Prepaging

- Prepaging
  - To reduce the large number of page faults that occurs at process startup
  - Prepage all or some of the pages a process will need, before they are referenced
  - But if prepaged pages are unused, I/O and memory was wasted
  - Assume s pages are prepaged and α of the pages is used
    - Is cost of $s * \alpha$ saved pages faults > or < than the cost of prepaging $s * (1- \alpha)$ unnecessary pages?
    - α near zero $\Rightarrow$ greater prepaging loses

Other Issues – Page Size

- Sometimes OS designers have a choice
  - Especially if running on custom-built CPU
- Page size selection must take into consideration:
  - Fragmentation
  - Page table size
  - **Resolution**
  - I/O overhead
  - Number of page faults
  - Locality
  - TLB size and effectiveness
- Always power of 2, usually in the range 2^{12} (4,096 bytes) to 2^{22} (4,194,304 bytes)
- On average, growing over time

Page size issues – TLB Reach

- TLB Reach - The amount of memory accessible from the TLB
- TLB Reach = (TLB Size) X (Page Size)
- Ideally, the working set of each process is stored in the TLB
  – Otherwise there is a high degree of page faults
- Increase the Page Size
  – This may lead to an increase in fragmentation as not all applications require a large page size
- Provide Multiple Page Sizes
  – This allows applications that require larger page sizes the opportunity to use them without an increase in fragmentation

Other Issues – Program Structure

- Program structure
  - `int[128,128] data;` i: row, j: column
  - Each row is stored in one page
  - Program 1

```
for (j = 0; j <128; j++)
    for (i = 0; i < 128; i++)
                   data[i,j] = 0;
```

128 x 128 = 16,384 page faults

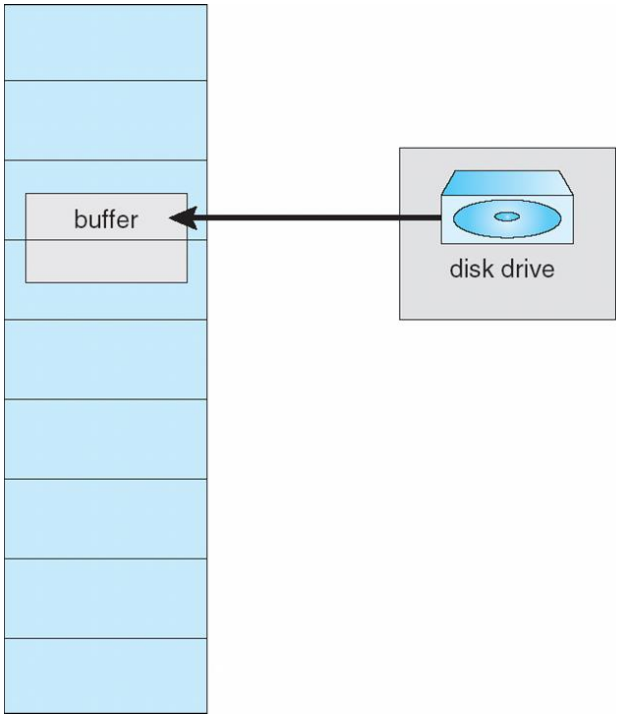
  - Program 2 inner loop = 1 row = 1 page

```
for (i = 0; i < 128; i++)
      for (j = 0; j < 128; j++)
            data[i,j] = 0;
```

128 page faults

Other Issues – I/O interlock

- **I/O Interlock** – Pages must sometimes be locked into memory
- Consider I/O - Pages that are used for copying a file from a device must be locked from being selected for eviction by a page replacement algorithm
- **Pinning** of pages to lock into memory



Colorado State University

Operating System Examples

- Windows

Windows

- 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