Lecture 17

Edited from slides for Operating System Concepts by Silberschatz, Galvin, Gagne

Page Replacement Algorithms

Last Lecture:

- FIFO
- Optimal Page Replacement
- LRU
- LRU Approximation
  - Additional-Reference-Bits Algorithm
  - Second-Chance Algorithm

This Lecture:

- Counting-Based Page Replacement

Counting Algorithms

- Keep a counter of the number of references that have been made to each page
- **LFU (Least Frequently Used) Algorithm**: replaces page with smallest count
- **MFU (Most Frequently Used) Algorithm**: based on the argument that the page with the smallest count was probably just brought in and has yet to be used

Problem

- You have devised a new page replacement algorithm that you think may be optimal.
- However, in some cases, Belady's anomaly occurs.
- Is the new algorithm optimal? Explain.

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 (equal vs. proportional)
  - priority allocation

Fixed Allocation: Equal

- Equal allocation – For example, if there are 100 frames and 5 processes, give each process 20 frames.

Fixed Allocation: Proportional

- Proportional allocation – Allocate according to the size of process

  - s_i = size of process p_i
  - $S = \sum s_i$
  - m = total number of frames
  - a_i = allocation for $p_i = \frac{s_i}{S} \times m$

$$m = 64$$
$$s_1 = 10$$
$$s_2 = 127$$
$$a_1 =$$
$$a_2 =$$

Fixed Allocation: Proportional

- Proportional allocation – Allocate according to the size of process

  - s_i = size of process p_i
  - $S = \sum s_i$
  - m = total number of frames
  - a_i = allocation for $p_i = \frac{s_i}{S} \times m$

$$m = 64$$

$$s_1 = 10$$

$$s_2 = 127$$

$$a_1 = \frac{10}{137} \times 64 \approx 5$$

$$a_2 = \frac{127}{137} \times 64 \approx 59$$

Priority Allocation

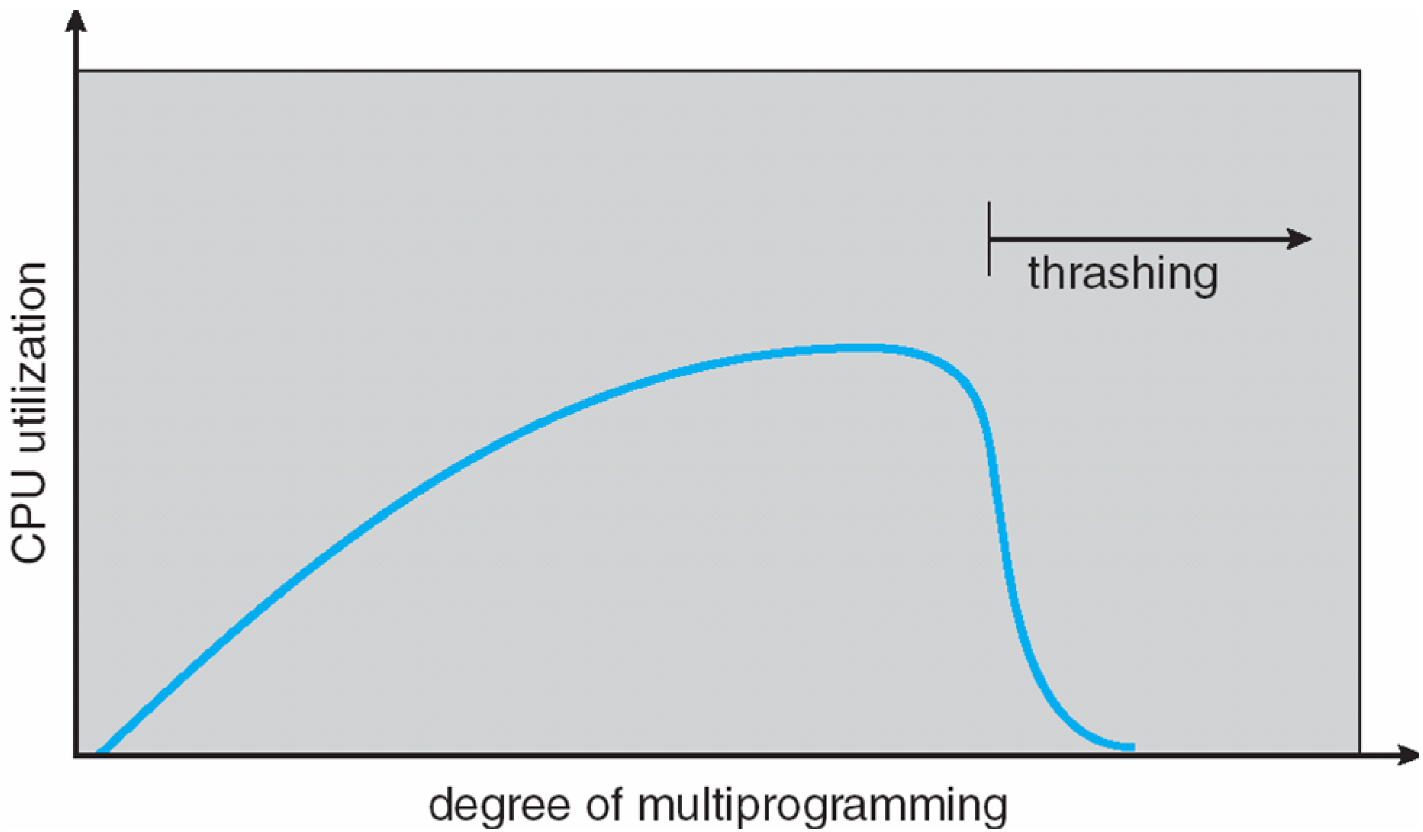
- Use a proportional allocation scheme using priorities rather than size
- If process P_i generates a page fault,
  - select for replacement one of its frames
  - select for replacement a frame from a process with lower priority number

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

Thrashing

- **Thrashing** ≡ process is busy swapping pages, instead of execution
- High page-fault rate =>low CPU utilization => OS thinks that it needs to increase the degree of multiprogramming => another process added to the system



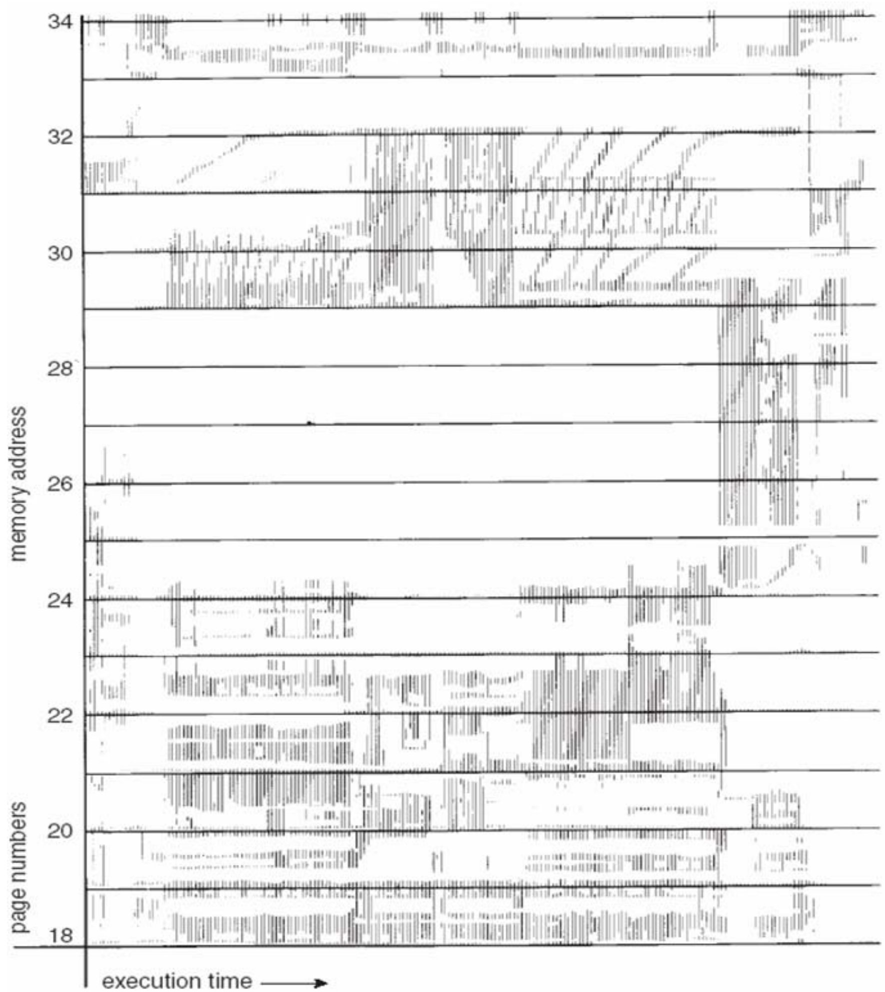
Demand Paging and Thrashing

- Prevent Thrashing by providing enough frames to process

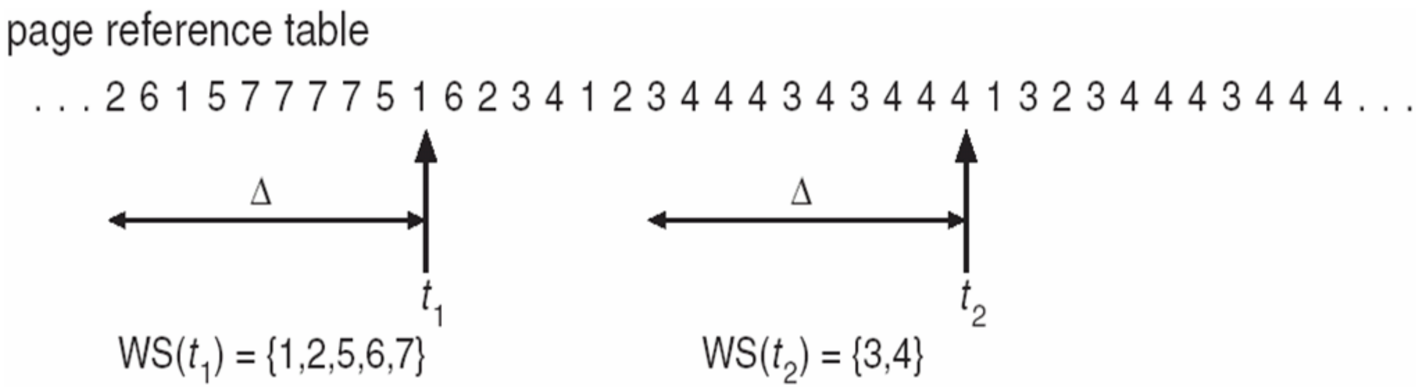
- Locality model
  - Process migrates from one locality to another
  - Localities may overlap

- Why does thrashing occur?
  Σ size of locality > total memory size

Locality In A Memory-Reference Pattern

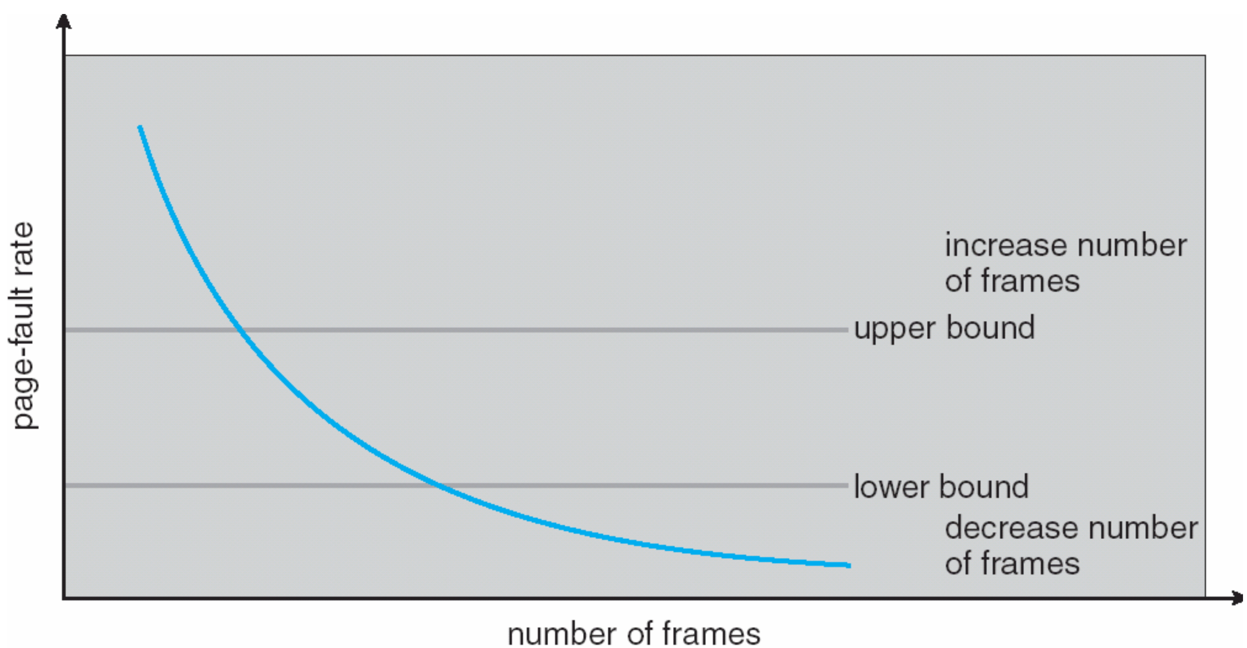


Working-set model

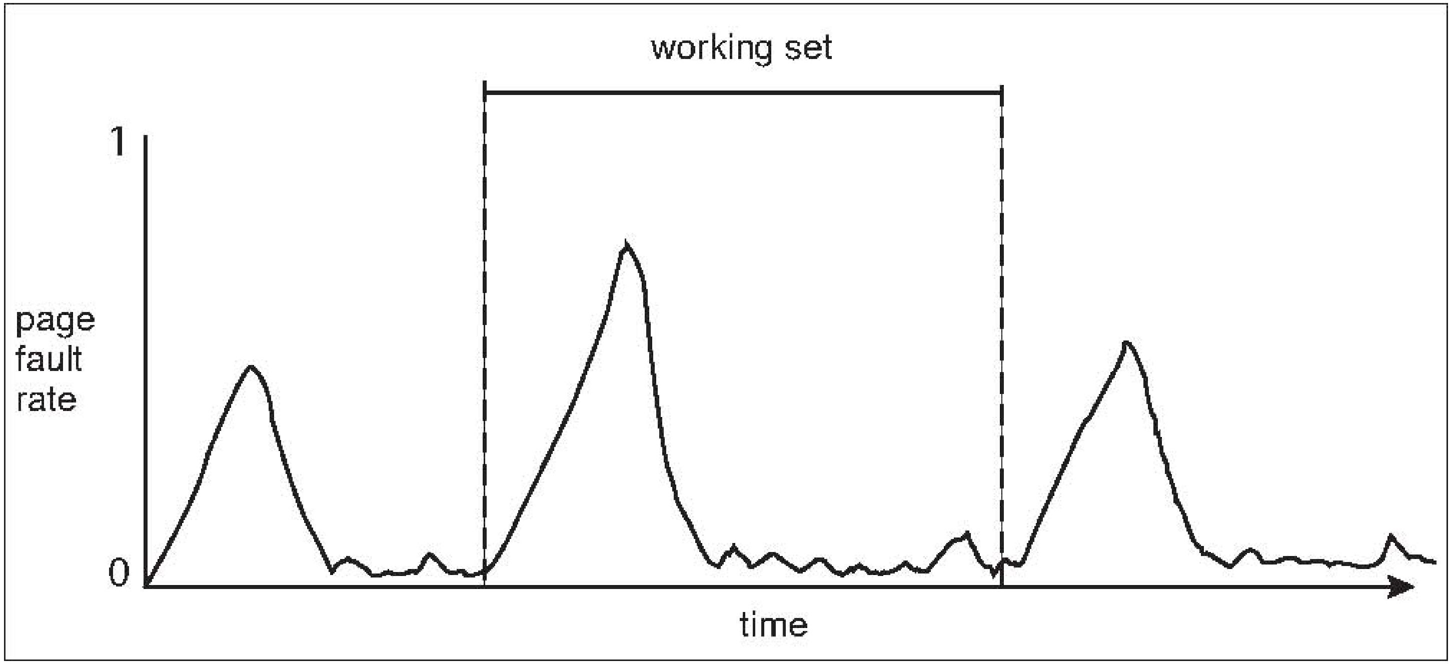


Page-Fault Frequency Scheme

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



Working Sets and Page Fault Rates



Other Issues -- 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 $s * \alpha$ save pages faults > or < than the cost of prepaging
      $s * (1- \alpha)$ unnecessary pages?
    - α near zero $\Rightarrow$ prepaging loses

Other Issues – Page Size

- Page size selection must take into consideration:
  - fragmentation
  - table size
  - I/O overhead
  - locality

Other Issues – Program Structure

- Program structure
  - `Int[128,128] data;`
  - Each row is stored in one page (row major)
  - 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

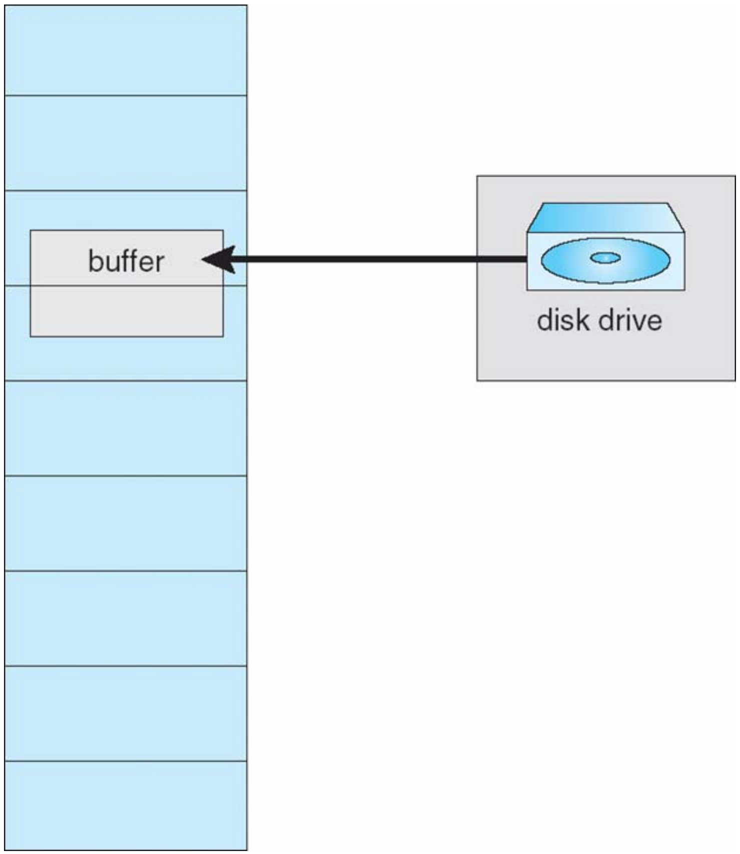
  - Program 2

```
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



Problem

- Demand-paging system with following time-measured utilizations
  - CPU 20%
  - Paging disk 97.7%
  - Other I/O devices 5%
- Comment on improved CPU utilization given if you
  - Install a faster CPU

Problem

- Demand-paging system with following time-measured utilizations
  - CPU 20%
  - Paging disk 97.7%
  - Other I/O devices 5%
- Comment on improved CPU utilization given if you
  - Install a faster CPU
  - Install a bigger paging disk

Problem

- Demand-paging system with following time-measured utilizations
  - CPU 20%
  - Paging disk 97.7%
  - Other I/O devices 5%
- Comment on improved CPU utilization given if you
  - Install a faster CPU
  - Install a bigger paging disk
  - Increase the degree of multiprogramming

Problem

- Demand-paging system with following time-measured utilizations
  - CPU 20%
  - Paging disk 97.7%
  - Other I/O devices 5%
- Comment on improved CPU utilization given if you
  - Install a faster CPU
  - Install a bigger paging disk
  - Increase the degree of multiprogramming
  - Decrease the degree of multiprogramming

Problem

- Demand-paging system with following time-measured utilizations
  - CPU 20%
  - Paging disk 97.7%
  - Other I/O devices 5%
- Comment on improved CPU utilization given if you
  - Install a faster CPU
  - Install a bigger paging disk
  - Increase the degree of multiprogramming
  - Decrease the degree of multiprogramming
  - Install more main memory

Problem

- Demand-paging system with following time-measured utilizations
  - CPU 20%
  - Paging disk 97.7%
  - Other I/O devices 5%
- Comment on improved CPU utilization given if you
  - Install a faster CPU
  - Install a bigger paging disk
  - Increase the degree of multiprogramming
  - Decrease the degree of multiprogramming
  - Install more main memory
  - Install a faster hard disk

Problem

- Demand-paging system with following time-measured utilizations
  - CPU 20%
  - Paging disk 97.7%
  - Other I/O devices 5%
- Comment on improved CPU utilization given if you
  - Install a faster CPU
  - Install a bigger paging disk
  - Increase the degree of multiprogramming
  - Decrease the degree of multiprogramming
  - Install more main memory
  - Install a faster hard disk
  - Add prepaging to the page-fetch algorithms

Problem

- Demand-paging system with following time-measured utilizations
  - CPU 20%
  - Paging disk 97.7%
  - Other I/O devices 5%
- Comment on improved CPU utilization given if you
  - Install a faster CPU
  - Install a bigger paging disk
  - Increase the degree of multiprogramming
  - Decrease the degree of multiprogramming
  - Install more main memory
  - Install a faster hard disk
  - Add prepaging to the page-fetch algorithms
  - Increase the page size

See you next time