CSCI315 – Operating Systems Design

Department of Computer Science
Bucknell University

The Working Set Model And Memory-Mapped Files

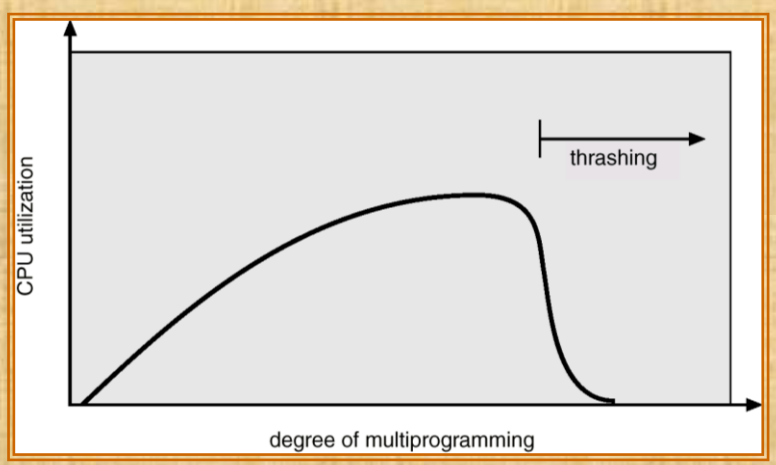
Ch 10.5, 10.8. 10.9

This set of notes is based on notes from the textbook authors, as well as L. Felipe Perrone, Joshua Stough, and other instructors.
Xiannong Meng, Fall 2021.

Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high. This leads to:
  - **Low CPU utilization**.
  - Operating system thinks that it needs to increase the degree of multiprogramming.
  - Another process added to the system.
- **Thrashing** ≡ a process is busy swapping pages in and out.

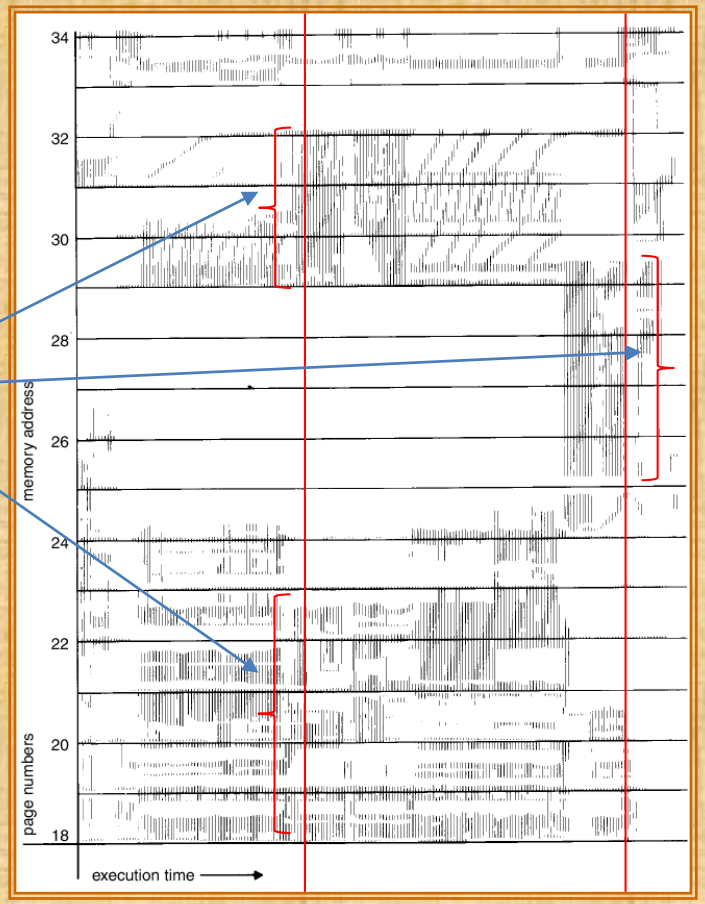
Thrashing



- 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

Locality in Memory-Reference Pattern

Clusters of memory references – working set

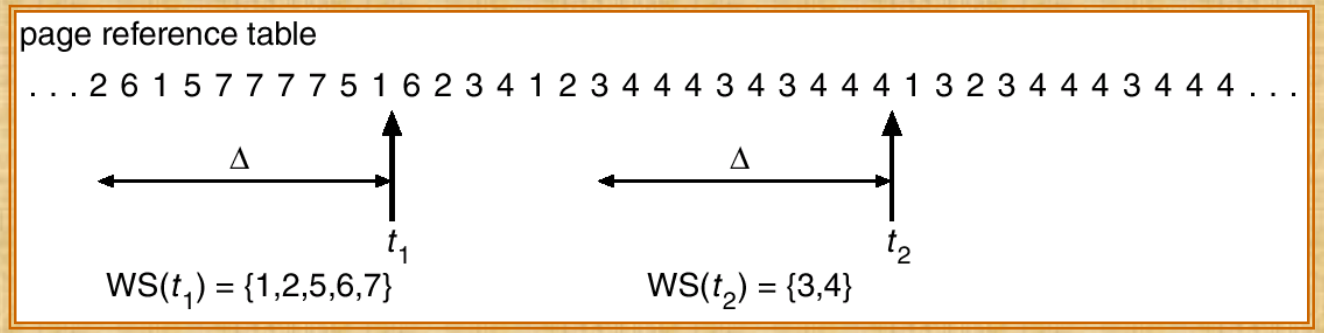


Working-Set Model

- $\Delta \equiv$ **working-set window** $\equiv$ a fixed number of page references.

- 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.

Working-set model

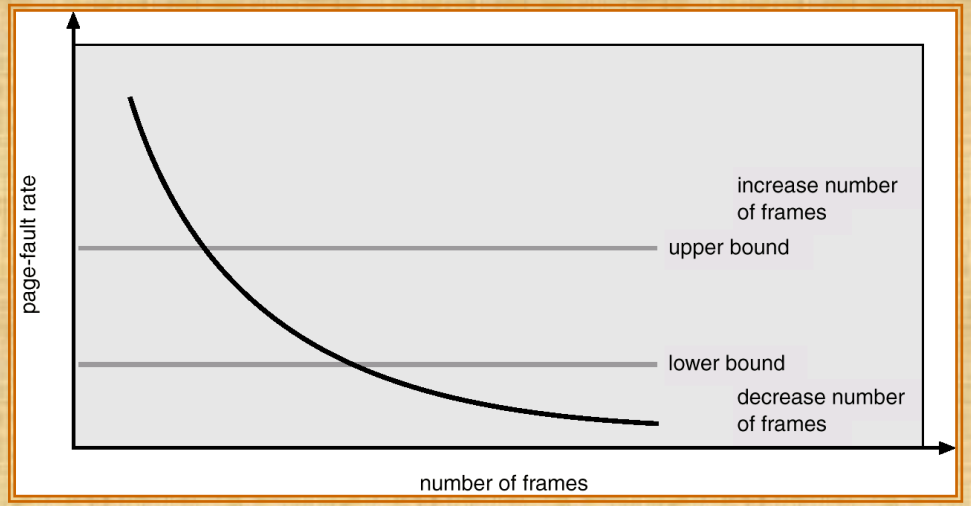


Δ == 10 in this example

Keeping Track of 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 and sets the values of all reference bits 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



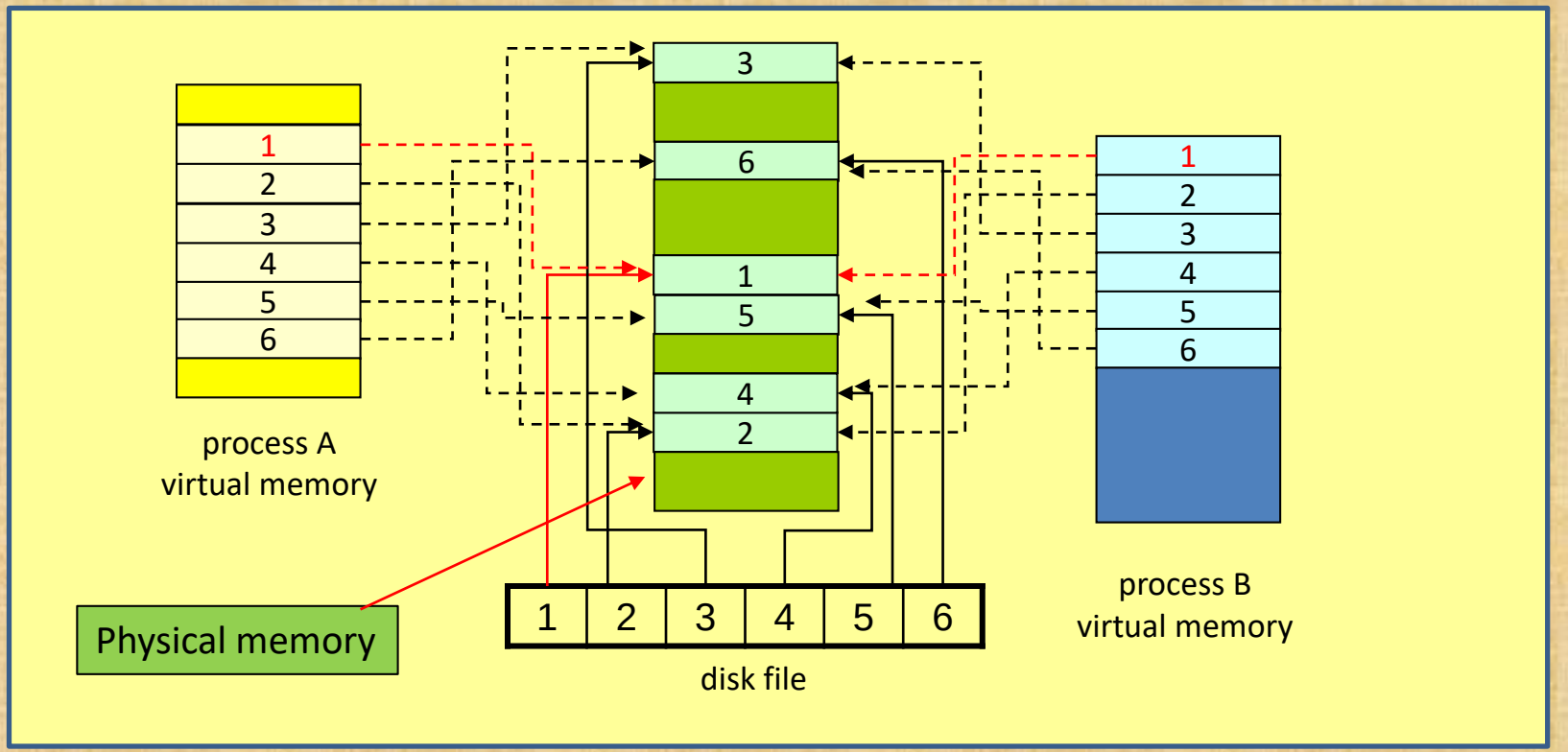
Establish "acceptable" page-fault rate.

- If actual rate too low, process loses frame.
- If actual rate too high, process gains frame.

Memory-mapped Files

- Memory mapping a file can be accomplished by mapping a disk block to one or more pages in memory.
- A page-sized portion of the file is read from the file system into a physical page. Subsequent `read()` and `write()` operations are handled as memory (not disk) accesses.
- Writing to the file in memory is not necessarily synchronous to the file on disk. The file can be committed back to disk when it's closed.

Memory-mapped Files



Prepaging

- **Prepaging**: In order to avoid the initial number of page faults, the system can bring into memory all the pages that will be needed all at once.
- This can also be applied when a swapped-out process is restarted. The smart thing to do is to remember the working set of the process.
- One question that arises is whether all the pages brought in will actually be used…
- Is the cost of prepaging less than the cost of servicing each individual page fault?

Major and Minor Page Faults

- A page fault occurs when a page needed by a process does not have a valid mapping in its page table (valid bit == 0).
- However, the needed frame might be actually in memory! It may just not be mapped to the process.
- If the frame needed is NOT in memory, we call this fault a **major page fault** – the frame has to be loaded from disk.
- If the frame needed is in memory, we call this fault a **minor page fault** – the mapping can be established without a disk I/O.

Reasons for Minor Page Fault

- There are two reasons for minor page fault to occur.
  1. A process may reference a shared page that is in memory, but not mapped to the current process.
  2. A frame has been moved to the free list by the memory management system. (We'll discuss the issue of reclaiming free frames later.)
- In either case, all the MMU needs to do is to update the page table reference, much less expensive than reading the frame from disk.

Observing Major and Minor Faults

- In Linux, one can view the number of major and minor page faults using the **`ps`** (process status) command.
- **`ps -eo min_flt,maj_flt,cmd`**

There is no space in `min_flt,maj_flt,cmd`

The command means “show all processes with minor and major page faults caused by all commands.”

Sample Output

From our Linuxremote1, the following are the sample outputs for the commands sshd -D, system-logind, auditd, and vim.

```
MINFL       MAJFL   CMD
4618573     3819    /usr/sbin/sshd –D
1068498     1206    /usr/lib/systemd/systemd-logind
17611       2312    /sbin/auditd
1851        0       vim
```

Some notes:

1. Because large number of processes are running, you'd use **grep** command to extract the output, e.g.,
   ps –eo min_flt,maj_flt,cmd | grep "sshd –D"
2. Why the number of major fault for "**vim**" is zero?