Operating Systems
CMPSC 473
Virtual Memory - Multiprogramming
March 25, 2008 - Lecture 17
Instructor: Trent Jaeger
• Last class:
  – Virtual Memory

• Today:
  – Virtual Memory Uses
Efficient Use of Physical Memory

• Through virtual memory…
  – $N \cdot 2^{32}$-sized address spaces
  – All isolated by default

• Uses for memory
  – Make a new process
    • Address space
  – Make an IPC
    • Or a cross-address space call

• Challenges in memory use
Shared Pages

• **Shared code**
  – One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).

• **Private code and data**
  – Each process keeps a separate copy of the code and data
  – The pages for the private code and data can appear anywhere in the logical address space
Shared Pages Example

- Process $P_1$
  - Page Table for $P_1$:
    - ed 1
    - ed 2
    - ed 3
    - data 1
    - Page Table for $P_2$:
      - ed 1
      - ed 2
      - ed 3
      - data 2
- Process $P_2$
  - Page Table for $P_2$:
    - ed 1
    - ed 2
    - ed 3
    - data 2
- Process $P_3$
  - Page Table for $P_3$:
    - ed 1
    - ed 2
    - ed 3
    - data 3

Page Addresses:

- 0: data 1
- 1: data 3
- 2: ed 1
- 3: ed 2
- 4: ed 3
- 5: ed 3
- 6: data 2
- 7: data 2
- 8: data 2
- 9:
- 10:
- 11:
Create a New Address Space

• Via fork or clone
  – Copy of the old address space
• Change completely
  – Exec
• Or use the copy independently
Copy-on-Write

• Copy-on-Write (COW) allows both parent and child processes to initially *share* the same pages in memory

  If either process modifies a shared page, only then is the page copied

• COW allows more efficient process creation as only modified pages are copied

• Free pages are allocated from a **pool** of zeroed-out pages
Before Process 1
Modifies Page C
After Process 1
Modifies Page C
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.
- 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 file access by treating file I/O through memory rather than **read()** or **write()** system calls.
- Also allows several processes to map the same file allowing the pages in memory to be shared.
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 (Cont.)

![Diagram showing CPU utilization vs. degree of multiprogramming with a curve indicating thrashing.](attachment:image.png)
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?
  $\Sigma$ size of locality $> \text{total memory size}$
Locality In A Memory-Reference Pattern
Working-Set Model

- $\Delta \equiv$ 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 $\Delta$ (varies in time)
    - if $\Delta$ too small will not encompass entire locality
    - if $\Delta$ too large will encompass several localities
    - if $\Delta = \infty \Rightarrow$ will encompass entire program
- $D = \sum WSS_i$ $\equiv$ total demand frames
- if $D > m \Rightarrow$ Thrashing
- Policy if $D > m$, then suspend one of the processes
Working-set model

page reference table

\[
\begin{align*}
\ldots & 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 \ldots \\
\end{align*}
\]

\[
\begin{align*}
\Delta & \\
t_1 & \\
\Delta & \\
t_2 & \\
\end{align*}
\]

\[
WS(t_1) = \{1,2,5,6,7\} \quad WS(t_2) = \{3,4\}
\]
Keeping Track of the Working Set

• Approximate with interval timer + a reference bit
• Example: $\Delta = 10,000$
  – 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
Summary

• Uses
  – Shared Pages
  – Copy-on-write
  – Memory-mapped files

• Thrashing and the Working Set model
• Next time: Files