CIS 415: Operating Systems

VM Issues

Spring 2014
Prof. Kevin Butler
Administrative Notes

• Project 2: due Friday night
• Project 3 and Assignment 3 out later today
• Final Exam on June 13 at 8 AM

• Good discussions on Piazza - make sure to read and use
System Performance

Table 2.2 Example Time Scale of System Latencies

<table>
<thead>
<tr>
<th>Event</th>
<th>Latency</th>
<th>Scaled</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 CPU cycle</td>
<td>0.3 ns</td>
<td>1 s</td>
</tr>
<tr>
<td>Level 1 cache access</td>
<td>0.9 ns</td>
<td>3 s</td>
</tr>
<tr>
<td>Level 2 cache access</td>
<td>2.8 ns</td>
<td>9 s</td>
</tr>
<tr>
<td>Level 3 cache access</td>
<td>12.9 ns</td>
<td>43 s</td>
</tr>
<tr>
<td>Main memory access (DRAM, from CPU)</td>
<td>120 ns</td>
<td>6 min</td>
</tr>
<tr>
<td>Solid-state disk I/O (flash memory)</td>
<td>50–150 μs</td>
<td>2–6 days</td>
</tr>
<tr>
<td>Rotational disk I/O</td>
<td>1–10 ms</td>
<td>1–12 months</td>
</tr>
<tr>
<td>Internet: San Francisco to New York</td>
<td>40 ms</td>
<td>4 years</td>
</tr>
<tr>
<td>Internet: San Francisco to United Kingdom</td>
<td>81 ms</td>
<td>8 years</td>
</tr>
<tr>
<td>Internet: San Francisco to Australia</td>
<td>183 ms</td>
<td>19 years</td>
</tr>
<tr>
<td>TCP packet retransmit</td>
<td>1–3 s</td>
<td>105–317 years</td>
</tr>
<tr>
<td>OS virtualization system reboot</td>
<td>4 s</td>
<td>423 years</td>
</tr>
<tr>
<td>SCSI command time-out</td>
<td>30 s</td>
<td>3 millennia</td>
</tr>
<tr>
<td>Hardware (HW) virtualization system reboot</td>
<td>40 s</td>
<td>4 millennia</td>
</tr>
<tr>
<td>Physical system reboot</td>
<td>5 m</td>
<td>32 millennia</td>
</tr>
</tbody>
</table>

As you can see, the time scale for CPU cycles is tiny. The time it takes light to travel 0.5 m, perhaps the distance from your eyes to this page, is about 1.7 ns. During the same time, a modern CPU may have executed five CPU cycles and processed several instructions.

Source: “System Performance: Enterprise and Cloud”, Brendan Gregg
Summary of page replacement algorithms

• OPT, FIFO, NRU, second-chance/clock, LRU, approximate LRU

• In practice, OSes use second chance/clock or some variations of it.
Belady’s Anomaly

• Normally you expect number of page faults to decrease as you increase physical memory size.

• However, this may not be the case in certain replacement algorithms
Belady’s Anomaly

• FIFO replacement Algorithm

• Reference string:
  ‣ 1 2 3 4 1 2 5 1 2 3 4 5

• 3 physical frames
  ‣ F F F F F F F - - F F -
  ‣ # of faults = 9

• 4 physical frames
  ‣ F F F F - - F F F F F F F
  ‣ # of faults = 10
• Algorithms which do NOT suffer from Belady’s anomaly are called *stack algorithms*

• E.g. OPT, LRU.
Paging Issues

• Keep the essentials of what you currently need *(working set)* in physical memory.

• When something you need is not in memory, bring it in from disk:
  ‣ On demand (demand-paging)
  ‣ Ahead of need (pre-paging)

• Programs need to exhibit good locality to avoid “thrashing” of pages in memory.

• This usually requires good programming skills!
Efficient Physical Memory

• Through virtual memory…
  ‣ $N \times 2^{32}$-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

```
<table>
<thead>
<tr>
<th>Process P1</th>
<th>Page Table for P1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ed1</td>
<td>3</td>
</tr>
<tr>
<td>ed2</td>
<td>4</td>
</tr>
<tr>
<td>ed3</td>
<td>6 1</td>
</tr>
<tr>
<td>Data 1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process P2</th>
<th>Page Table for P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ed1</td>
<td>3</td>
</tr>
<tr>
<td>ed2</td>
<td>4</td>
</tr>
<tr>
<td>ed3</td>
<td>6 7</td>
</tr>
<tr>
<td>Data 2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process P3</th>
<th>Page Table for P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ed1</td>
<td>3</td>
</tr>
<tr>
<td>ed2</td>
<td>4</td>
</tr>
<tr>
<td>ed3</td>
<td>6 2</td>
</tr>
<tr>
<td>Data 3</td>
<td></td>
</tr>
</tbody>
</table>
```

<table>
<thead>
<tr>
<th>Page 0</th>
<th>Page 1</th>
<th>Page 2</th>
<th>Page 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 1</td>
<td>data 1</td>
<td>ed 1</td>
<td>ed 2</td>
</tr>
<tr>
<td>Data 3</td>
<td>data 3</td>
<td>ed 1</td>
<td>ed 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ed 3</td>
<td>data 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>data 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>data 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>data 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>data 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>data 2</td>
</tr>
</tbody>
</table>
Create 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
  ‣ File is initially read using demand paging
  ‣ 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
  ‣ What is the benefit of doing this?
• Also allows several processes to map the same file allowing the pages in memory to be shared
Memory Mapped Files
Memory-Mapped Shared Mem
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

The diagram shows the relationship between CPU utilization and the degree of multiprogramming, indicating the point of 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 $>$ total memory size
Memory-Reference Locality
Working-Set Model

- \( \Delta \equiv \text{working-set window} \equiv \) a fixed number of page references (e.g., 10,000 instructions)
- \( WSS_i \) (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 \( D > m \), suspend one of the processes
Working-set model

Sliding window that approximates program locality
Tracking the Working Set

• Approximate with interval timer + reference bits

• 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 set 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

• 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