Administrative Notes

• Project 2: due tonight
• Project 3 and Assignment 3 are out
• Final Exam on July 13 at 8 AM

• Good discussions on Piazza - make sure to read and use
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 F -
  ‣ # of faults = 9

• 4 physical frames
  ‣ F 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!
Fragmentation in paging

• Note that there is only internal fragmentation, and that too is only in the last allocated page.

• Smaller the page, smaller the internal fragmentation.

• However, this reduces spatial locality.
Page size trade-offs

- Average process size = $s$ bytes
- Page size = $p$ bytes
- Page Table entry = $e$ bytes

$$\text{Overhead} = s \times \frac{e}{p} + \frac{p}{2}$$
Summary

• Page Replacement
  ‣ Virtual memory
  ‣ Page faults
  ‣ Optimal page replacement not achievable
  ‣ Variety of algorithms
  ‣ Anomalies
Efficient 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

```
<table>
<thead>
<tr>
<th>Process P₁</th>
<th>Process P₂</th>
<th>Process P₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>ed 1</td>
<td>ed 1</td>
<td>ed 1</td>
</tr>
<tr>
<td>ed 2</td>
<td>ed 2</td>
<td>ed 2</td>
</tr>
<tr>
<td>ed 3</td>
<td>ed 3</td>
<td>ed 3</td>
</tr>
<tr>
<td>data 1</td>
<td>data 1</td>
<td>data 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page Table for P₁</td>
<td>Page Table for P₂</td>
<td>Page Table for P₃</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>
```

Data Allocation:
- Page 0: data 1
- Page 1: data 3
- Page 2: ed 1
- Page 3: ed 1
- Page 4: ed 2
- Page 5: ed 3
- Page 6: ed 3
- Page 7: data 2
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
COW in Action

Before Process 1 modifies Page C...
After Process 1 modifies Page C...
• 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.
Memory Mapping Benefits

• 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

process$_1$

shared memory

memory-mapped file

shared memory

process$_2$

shared memory
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

![Diagram showing CPU utilization vs. degree of multiprogramming]

- **CPU utilization** increases as the **degree of multiprogramming** increases, reaching a peak before decreasing, indicating thrashing.
Demand Paging & 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
Memory-Reference Locality
Working-Set Model

• $\Delta \equiv \textit{working-set window} \equiv$ 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
• 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