Lecture Topics

- Today: Virtual Memory
  (Stallings, chapter 8.1-8.4)
- Next: continued

Announcements

- Self-Study Exercise #8
- Project #6 (due 10/26)
- Project #7 (due 11/2)
Virtual Memory

- Memory management technique where virtual addresses (logical addresses as seen by a process) are mapped to physical addresses (actual addresses in RAM).

- Only a subset of the virtual address space for a process needs to be in RAM for it to execute (the virtual addresses which are currently being used by the process).

Virtual Memory: Costs and Benefits

- Every virtual address must be translated to a physical address (takes time, needs hardware support).

- CPU utilization can be increased by having more processes resident in RAM.

- The logical address space of a process can be larger than the physical address space of the system.
Virtual Memory and Paging

- Most virtual memory systems use paging: keep a subset of the pages in primary storage (RAM) and the remainder in secondary storage (disk).

- When necessary, move pages from disk to RAM (and from RAM to disk).

- Page table must keep track of current location of each page (RAM or disk).
OS: Page Fault Processing

- Move process from Running to Blocked
- Choose page to replace (if all frames full)
- If victim page has been modified ("dirty"), copy victim page from RAM to disk
- Copy page from disk to RAM
- Update page table entry
- Move process from Blocked to Ready

Fetch Policy

Determines when a page should be brought into main memory

- Demand paging: copy a page into main memory when a reference is made to a location on that page
- Prepaging: copy more than one page into main memory in response to a page fault
Demand Paging

- When a reference is made to a virtual address on that page, the page is copied from disk to RAM (if it is not already present).
- When a process first begins executing, there will usually be a flurry of page faults.
- Once enough pages have been loaded into RAM, the page fault rate will drop to a very low level (locality of reference).

Prepaging

- More than one page is copied into RAM at once when a page fault occurs.
- Takes advantage of characteristics of disk drives (relatively inexpensive to access adjacent disk sectors).
- Not particularly effective: difficult to predict which pages to load, some pages loaded unnecessarily.
Replacement Policy

- Determines which page should be selected as the "victim", if there are no empty frames

- Related ideas (discussed later):
  - how many frames should be allocated to the current process?
  - should the victim be selected from within the frames allocated to the process, or from within all frames?

Replacement Policy

- Optimal: the page which will not be used for the longest period of time is selected as the victim (results in fewest page faults)

- Not practical to implement – OS does not have perfect knowledge of future events

- Useful as benchmark
Example

- Process with 5 pages, 3 frames allocated

<table>
<thead>
<tr>
<th>Page address stream</th>
<th>2</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>5</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>3</th>
<th>2</th>
<th>5</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>F</td>
<td>-</td>
<td>-</td>
<td>F</td>
</tr>
</tbody>
</table>

- Page address stream condensed: many references to same page

- Diagram only shows faults which result in replacement of a page ("F")

Least Recently Used (LRU)

- Replace the page which has not been used for the longest time

- Locality of reference suggests that page will not be used in the near-term future

- Impractical to implement: expensive to tag each page with time of last reference
Example (cont)

Compare optimal and LRU:

<table>
<thead>
<tr>
<th>Page address</th>
<th>2</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>5</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>3</th>
<th>2</th>
<th>5</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OPT</strong></td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td><strong>LRU</strong></td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

First-In, First-Out (FIFO)

- Replace the page which has been in RAM for the longest time
- If a page has been in RAM for a long time, perhaps it is no longer needed (not always true)
- Easy to implement: treat page frames as a circular buffer
Example (cont)

Compare optimal, LRU and FIFO:

<table>
<thead>
<tr>
<th>Page address stream</th>
<th>2</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>5</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>3</th>
<th>2</th>
<th>5</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>F</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>LRU</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>F</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>F</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>FIFO</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>F</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>F</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Clock

- LRU not practical, FIFO performs poorly
- Find an algorithm which approximates LRU, but can be implemented easily
- Clock policy: bit associated with each frame to help keep track of the fact that it has been used recently (the use bit or referenced bit)
Clock

- Use bit set to 1:
  - when page is initially loaded into RAM
  - whenever the page is referenced again (used)

- Use bit set to 0:
  - when OS scans for a page to replace
Example (cont)

Experimental Comparisons

- Fixed number of frames, victim selected locally
- FIFO worse than Optimal by a factor of 2
- Clock approximates LRU (particularly if improved)
Advanced Algorithms

- Many variations on the Clock algorithm: "second chance" algorithms
- Ideal victim: page has not been modified (does not have to be copied to disk) and has not been referenced recently
- Paging system already has "modified" bit; check that bit as part of replacement algorithm

Advanced Algorithms (cont)

- Four combinations
  - U: 0  M: 0  Not used recently, not modified
  - U: 0  M: 1  Not used recently, modified
  - U: 1  M: 0  Used recently, not modified
  - U: 1  M: 1  Used recently, modified
- Better to select page with U = 0, even if M = 1 (unlikely to be referenced again in the near future)
UNIX: Variation on Clock

front hand sets U bit back to 0, back hand looks for pages where U is still 0

scanrate – rate at which the two hands scan the list (pages per sec)

handspread – the gap between front hand and back hand

Page Buffering

- One problem with all replacement algorithms: the victim page might be referenced in the near future. Since that page has been removed from RAM, it has to be brought back into RAM when it is referenced.

- Rather than overwriting the page immediately, put it into a buffer of pages (leaves time to reclaim it, if it is referenced in the near future).
Page Buffering

- Free frame list: linked list of page frames which are not currently allocated.
- When a page is selected as the victim, its frame is added to the tail of the free frame list.
- When a frame is allocated, the frame at the head of the free frame list is removed.
- Thus, the victim page remains in its frame for some period of time, can be reclaimed.

OS Issues

- Resident set management
- Cleaning policy
- Load control
Resident Set Management

- Resident Set Size
  - Fixed allocation
  - Variable allocation

- Replacement Scope
  - Local replacement
  - Global replacement

Resident Set Size

Trade offs

- If fewer frames are allocated to one process, more processes can be resident in RAM; the pool of processes in the Ready state is larger
- The page fault rate will be higher if too few frames are allocated to a process
- Beyond a certain point, allocating more page frames will not lower the page fault rate for a process
Resident Set Size

- Fixed allocation: number of frames allocated is determined by OS when process is first loaded (based on type of process, info from user)

- Variable allocation: number of frames can vary over lifetime of process (more frames allocated if page fault rate is too high, frames taken away if page fault rate is too low)

Replacement Scope

- Local: page to be replaced is selected from among the resident pages of the process which generated the page fault

- Global: page to be replaced is selected from among all allocated page frames, regardless of which process owns them
Resident Set Management

<table>
<thead>
<tr>
<th>Fixed Allocation</th>
<th>Local Replacement</th>
<th>Global Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of frames allocated to a process is fixed.</td>
<td>Page to be replaced is chosen from among the frames allocated to that process.</td>
<td>Not possible.</td>
</tr>
<tr>
<td>Variable Allocation</td>
<td>Number of frames allocated to a process may vary over time to maintain the working set of the process.</td>
<td>Page to be replaced is chosen from all available frames in main memory; this causes the size of the resident set of processes to vary.</td>
</tr>
<tr>
<td>Page to be replaced is chosen from among the frames allocated to that process.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fixed Allocation, Local Scope

- Process given a fixed number of frames when it is loaded into RAM initially
- When a page fault occurs, a page in one of those frames is selected as the victim
- Too few frames: high page fault rate (system can be affected by extra disk I/O)
- Too many frames: frames allocated, but not used (lowers multiprogramming level)
Variable Allocation, Global Scope

- Process given a certain number of frames when it is loaded into RAM initially, frames can be added or removed over time
- Victim selected from among all frames allocated to all processes; some process loses a member of its resident set
- Page buffering can be used to minimize impact of a poor choice

Variable Allocation, Local Scope

- When a process is initially loaded, allocate some number of page frames (based on type of process, info from user)
- When a page fault occurs, select victim locally
- Periodically check the number of pages allocated, make reasoned decision to add or remove frames to improve overall system performance
Working Set Model

- Resident set: for a given process, the set of pages which are currently resident in RAM
- Working set: the set of pages which have been referenced recently (specific time delta usually given)
- Working set changes over time

Working Set Behavior
Working Set Behavior

- The working set of a process grows when it first starts executing
- It then stabilizes (locality of reference)
- It grows again when the process enters a new locality (transition period), eventually contains pages from two different localities
- It then decreases after sufficient time is spent in the new locality

Working Set Strategy

- Denning: formalization of ideas related to working set of pages
- $W(t,d)$: at virtual time $t$, set of pages which have been referenced in the past $d$ virtual time units
- If $d$ is larger, $W(t,d)$ is larger (wider window implies larger working set)
Working Set Strategy

Overview:

1. Monitor working set of each process

2. Periodically remove pages from resident set if they are no longer in working set

3. Process can only be in Ready state if its working set is contained in its resident set
Working Set Strategy

Problems:

1. Size and membership of working set will change over time

2. Impractical to keep track of working set for each process (time stamp each page reference)

3. Difficult to choose d for \( W(t,d) \)

Page Fault Frequency Algorithm

- Approximate Working Set Strategy by focusing on the page fault rate of a process
  - Page fault rate too high: allocate frame
  - Page fault rate too low: deallocate frame
  - Doesn’t handle transitional periods very well, but variations on the algorithm can account for those spikes in the page fault rate
Cleaning Policy

- Pages which have been modified must be copied to disk (or changes will be lost)

- Demand cleaning: page is copied to disk when it has been chosen as the victim

- Precleaning: pages copied to disk periodically so that they can be copied in blocks

Cleaning Policy

- Problem with demand cleaning: process with page fault has to wait for two disk transfers (modified page from RAM to disk, required page from disk to RAM)

- Problem with precleaning: many pages are copied to disk and then modified again (same pages are copied to disk repeatedly)
Cleaning Policy

- Page buffering can improve performance
- Two free frame lists: unmodified and modified pages
- Unmodified pages do not need to be copied to disk
- Modified pages can be copied to disk, then moved to the other list

Load Control

- Multiprogramming level: number of processes which are resident in RAM
Multiprogramming Level

- Initially, as the multiprogramming level increases, CPU utilization increases (there is less chance that all processes are blocked)

- However, beyond a certain point, the resident set for each process will become too small and there will be many page faults

- Suspend (swap out) one or more processes to lower the multiprogramming level

---

Multiprogramming Level

- Denning: adjust the multiprogramming level so that the mean time between page faults equals the mean time to process a fault

- Empirical studies show that CPU utilization is at a maximum with that approach

- Other strategies are similar: keep disk drive used for paging busy 50% of the time