Virtual Memory

These slides are created by Dr. Huang of George Mason University. Students registered in Dr. Huang’s courses at GMU can make a single machine readable copy and print a single copy of each slide for their own reference as long as the slide contains the copyright statement, and the GMU facilities are not used to produce the paper copies. Permission for any other use, either in machine-readable or printed form, must be obtained form the author in writing.

Background

- Virtual memory – separation of user logical memory from physical memory.
  - Pages in the virtual memory space can be stored in the physical memory or disks.
  - Logical address space can therefore be much larger than physical address space
  - It is as if every process has a clean, dedicated $2^{32}$ memory space.
  - A faulty process has no way to screw up other processes.
Virtual Memory That is Larger Than Physical Memory

Demand Paging

- Bring a page into memory only when it is needed.
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users or processes
- Page is needed $\Rightarrow$ reference to it
  - invalid reference $\Rightarrow$ abort
  - not-in-memory $\Rightarrow$ bring to memory
Transfer of a Paged Memory to Contiguous Disk Space

Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated (1 ⇒ in-memory, 0 ⇒ not-in-memory)
- Initially valid–invalid bit is set to 0 on all entries.
- Example of a page table snapshot.
- During address translation, if valid–invalid bit in page table entry is 0 ⇒ page fault.
Page Table When Some Pages Are Not in Main Memory

- If there is ever a reference to a page, first reference will trap to OS ⇒ page fault
- OS looks at another table to decide:
  - Invalid reference ⇒ abort.
  - Just not in memory.
- Get empty frame.
- Swap page into frame.
- Reset tables, validation bit = 1.
- Restart instruction
Steps in Handling a Page Fault

What happens if there is no free frame?

- **Page replacement** – find some page in memory, hopefully not in use, swap it out.
  - algorithm
  - performance – want an algorithm which will result in minimum number of page faults.
- Use **modify (dirty)** bit to reduce overhead of page transfers – only modified pages are written to disk.
Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory.
Performance of Demand Paging

- Page Fault Rate $0 \leq p \leq 1.0$
  - if $p = 0$, no page faults
  - if $p = 1$, every reference is a fault
- Effective Access Time (EAT)
  \[
  \text{EAT} = (1 - p) \times \text{memory access} \\
  + p \times (\text{page fault overhead} \\
  + \text{swap out overhead} \\
  + \text{swap in overhead} \\
  + \text{restart overhead})
  \]

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.
Page Fault with Page Replacement

1. Find the location of the desired page on disk.
2. Find a free frame:
   If there is a free frame, use it
   If there is no free frame, use a page replacement algorithm to select a *victim* frame.
3. Read the desired page into the (newly) free frame. Update the page and frame tables.
4. Restart the process.

Page Replacement at Work

1. Swap out victim page
2. Change to invalid
3. Swap desired page in
4. Reset page table for new page
Page Replacement Algorithms

- In general more frames produces less page faults (the reason you want to buy more RAM.)
- Give the same memory size, different algorithms produces different page fault rates
  - Smaller rates are desirable
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string.
- In all our examples, the reference string is 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

FIFO Page Replacement

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.
- How many faults does 3 frames produce?  
  -  
- How many faults does 4 frames produce?  
  -  
- FIFO replacement may exhibit Belady’s Anomaly
  - more frames are supposed to produce less page faults
Exercise

Reference String

<p>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Page Frames

Optimal Replacement

- Replace page that will not be used for longest period of time.
- Example: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

- How do you know the future?
- It is used for measuring how well your algorithm performs.
Exercise

Reference String

| 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
| 7 | 7 | 7 | 2 | 2 | 2 | 2 | 2 | 2 | 7 |
| 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 3 | 3 | 3 | 3 | 1 | 1 |

Page Frames

Least-Recently Used (LRU) Replacement

- Example: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.
Counter Implementation

- The processor maintains a (hardware) counter of memory references
- This counter is called a clock (metaphorically speaking).
- Each page has a time-of-use field for the last time (counter value) it was accessed.
- Find the page with the earliest time-of-use for replacement.

Exercise

Reference String

<table>
<thead>
<tr>
<th>7</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>0</th>
<th>4</th>
<th>4</th>
<th>4</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Page Frames
Stack Implementation of LRU

- keep a stack of page numbers in a double link form:
  - Page referenced:
    - move it to the top
    - requires 6 pointers to be changed
  - No search for replacement

Example

<table>
<thead>
<tr>
<th>reference string</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 7 0 7 1 0 1 2 1 2 7 1 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>stack before a</th>
<th>stack after b</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>b</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CS471
LRU Approximation Algorithms

- Reference bit
  - With each page associate a reference bit, initially = 0
  - When page is referenced, the bit set to 1.
  - Replace the one which is 0 (if one exists).
  We do not know the order, however.

- Second Chance
  - FIFO replacement with reference bit
  - FIFO implemented as a circular queue
  - If page to be replaced (according to FIFO) has reference bit = 1, then give it a second chance:
    ➔ set reference bit 0.
    ➔ leave page in memory.
    ➔ replace next page according to FIFO, subject to same rules.
Counting Algorithms

- Keep a counter of the number of references that have been made to each page.

- **Least Frequently Used (LFU)**
  - replaces page with smallest count.

- **Most Frequently Used (MFU)**
  - based on the argument that the page with the smallest count was probably just brought in and has yet to be used.
Problems of counting algorithms: too good a memory is not for your own good
  – A high count might be obtained in distant past; the page could have not been used for a while.
  – What we need is an “aging” mechanism to allow distant history to fade out.
  – One solution is to periodically shift the counter right, effectively cutting its value by half

The Frame Allocation Problem

How many frames does a process need?
  – Obviously the answer varies from application to application
  – How could the OS figure it out for each process?
Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high. This leads to:
  - low CPU utilization
  - System resources consumed by swapping, not real jobs.

Thrashing
- a process is busy swapping pages in and out.

Locality in Memory References

- A process typically does not access its data with an even pattern (say round robin).
- Rather it use a small subset of data at a time before moving on to other parts of the data.
  - These active data are called the working set of the process
- If we keep the working set in the memory, then the process will not thrash even if the physical memory it uses is less than its size.
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.

- $D = \sum WSS_i \equiv$ total demand frames

if $D > m \Rightarrow$ Thrashing

Policy if $D > m$, then suspend one of the processes.
Working-set model

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.
- Establish "acceptable" page-fault rate.
  - If actual rate too low, process loses frame.
  - If actual rate too high, process gains frame.

Page-Fault Frequency Scheme
Other Considerations

- **Prepping**
- **TLB Reach**
  - The amount of memory accessible from the TLB.
  - TLB Reach = (TLB Size) X (Page Size)
- Ideally, the working set of each process is stored in the TLB. Otherwise there is a high degree of page faults.

Virtual Memory and IO

- **I/O Interlock** – Pages must sometimes be locked into memory.

- Consider I/O. Pages that are used for copying a file from a device must be locked from being selected for eviction by a page replacement algorithm.
Example

Windows NT

- Uses demand paging with **clustering**. Clustering brings in pages surrounding the faulting page.
- Processes are assigned **working set minimum** and **working set maximum**.
- Working set minimum is the minimum number of pages the process is guaranteed to have in memory.
- A process may be assigned as many pages up to its working set maximum.
- When the amount of free memory in the system falls below a threshold, **automatic working set trimming** is performed to restore the amount of free memory.
- Working set trimming removes pages from processes that have pages in excess of their working set minimum.