

Virtual Memory

CS 256/456

Dept. of Computer Science, University of Rochester

2/21/2005 CSC 256/456 - Spring 2005 1

Virtual Memory

- **Virtual memory** - separation of user logical memory from physical memory.
 - Only part of the program needs to be in memory for execution.
 - Logical address space can therefore be much larger than physical address space.
 - Allows address spaces to be shared by several processes.
 - Copy-on-write: allows for more efficient process creation.

- With virtual memory, the whole address space of each process has a copy in the backing store (i.e., disk)

2/21/2005 CSC 256/456 - Spring 2005

Page Table with Virtual Memory

- With each page table entry a valid-invalid bit is associated (1 ⇒ in-memory, 0 ⇒ not-in-memory or invalid logical page)

2/21/2005 CSC 256/456 - Spring 2005 3

Page Fault

- A reference to a page with the valid bit set to 0 will trap to OS ⇒ page fault
- Invalid logical page:
 - ⇒ abort.
- Just not in memory:
 - Get a free frame.
 - Swap page into the free frame.
 - Reset the page table entry, valid bit = 1.
 - Restart the program from the fault instruction.
- What if there is no free frame?

2/21/2005 CSC 256/456 - Spring 2005 4

Performance of Demand Paging

- Page Fault Rate $0 \leq p \leq 1.0$
- Effective Access Time (EAT)
 - EAT = $(1 - p) \times \text{memory access}$
 - + $p \times (\text{page fault overhead}$
 - + [swap page out]
 - + swap page in
 - + restart overhead
 - + memory access)

2/21/2005

CSC 256/456 - Spring 2005

5

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.
- At page fault:
 - A certain portion of the file is read from the file system into physical memory.
 - Subsequent reads/writes to/from the file are like ordinary memory accesses.
- Simplifies file access by treating file I/O through memory rather than **read()** **write()** system calls.

2/21/2005

CSC 256/456 - Spring 2005

6

Page Replacement

- Page replacement is necessary when no physical frames are available for demand paging
 - a victim page would be selected and replaced
- A dirty bit for each page
 - indicating if a page has been changed since last time loaded from the backing store
 - indicating whether swap-out is necessary for the victim page.
 - Should this bit be in page table or not?

2/21/2005

CSC 256/456 - Spring 2005

7

Page Replacement Algorithms

- Page replacement algorithm: the algorithm that picks the victim page.
- Metric:
 - low page-fault rate.
 - implementation cost/feasibility.
- For the page-fault rate:
 - Evaluate an algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string.

2/21/2005

CSC 256/456 - Spring 2005

8

First-In-First-Out (FIFO) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time)

1	4	5
2	1	3
3	2	4

9 page faults

- 4 frames

1	5	4
2	1	5
3	2	
4	3	

10 page faults

- Anomaly for the FIFO Replacement
 - more frames not necessarily leading to less page faults

2/21/2005 CSC 256/456 - Spring 2005 9

Optimal Algorithm

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

1	4
2	
3	
4	5

6 page faults

2/21/2005 CSC 256/456 - Spring 2005 10

Least Recently Used (LRU) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1	5
2	
3	5 4
4	3

- Not always better than FIFO, but more frames always lead to less or equal page faults
 - imagine a virtual stack (infinite size) of pages
 - each page is moved to the top after being accessed
 - this virtual stack is independent of the number of frames
 - page fault number when there are N frames:
 - the number of accesses that do not hit the top N pages in the virtual stack.

2/21/2005 CSC 256/456 - Spring 2005 11

Implementations

- FIFO implementation
- Time-of-use LRU implementation:
 - Every page entry has a time-of-use field; every time page is referenced through this entry, copy the clock into the field.
 - When a page needs to be changed, look at the time-of-use fields to determine which are to change.
- Stack LRU implementation - keep a stack of page numbers in a double link form:
 - Page referenced: move it to the top
 - Always replace at the bottom of the stack

2/21/2005 CSC 256/456 - Spring 2005 12

Feasibility of the Implementations

- FIFO implementation.
- LRU implementations:
 - Time-of-use implementation
 - Stack implementation
- What needs to be done at each memory reference?
- What needs to be done at page loading or page replacement?

2/21/2005

CSC 256/456 - Spring 2005

13

LRU Approximation Algorithms

- LRU approximation with a little help from the hardware.
- Reference bit
 - With each page associate a bit, initially = 0
 - When page is referenced, the bit is set to 1 by the hardware.
 - Replace a page whose reference bit is 0 (if one exists). We do not know the order, however.
- Second chance
 - Combining the reference bit with FIFO replacement
 - If page to be replaced (in FIFO order) has reference bit = 1, then:
 - set reference bit 0.
 - leave page in memory.
 - replace next page (in FIFO order), subject to same rules.

2/21/2005

CSC 256/456 - Spring 2005

14

LRU Approximation Algorithms

- Enhancing the reference bit algorithm:
 - it would be nice if there is more information about the reference history than a single bit.
 - with some help from software.
- Maintain more reference bits in software:
 - at every N-th clock interrupt, the OS moves each hardware page reference bit into a page reference history word (with more than one bit).

2/21/2005

CSC 256/456 - Spring 2005

15

Counting-based Page Replacement

- Least frequently used page-replacement algorithm
 - the page with smallest access count (within a period of time) is replaced
- Implementation?

2/21/2005

CSC 256/456 - Spring 2005

16

Disclaimer

- Parts of the lecture slides contain original work of Abraham Silberschatz, Peter B. Galvin, Greg Gagne, Andrew S. Tanenbaum, and Gary Nutt. The slides are intended for the sole purpose of instruction of operating systems at the University of Rochester. All copyrighted materials belong to their original owner(s).

2/21/2005

CSC 256/456 - Spring 2005

17