Administrivia

- Last time
  - Page table data structures
  - Started Page Replacement Policies

- Today: Finish page replacement policies then midterm review

- Homework due today; key will be made available sometime over the weekend (to allow for slack days)
Hierarchical Page Table

Outer Page Table (always in memory)

Page Table

Memory
Inverted Page Table

Logical address

physical address

Pid

p

d

search

I

Pid, p

page table

physical memory
General Flow of Memory Access

- Assume TLB; inverted page table; physically indexed cache; software TLB

1. Physically indexed cache: must Translate address before we can check cache
2. Translation
   - Check TLB – TLB hit means translation ready
   - TLB Miss – trap to OS
     * Check Page table (depending on page table organization); is page in memory?
       - Yes – place entry in TLB (possibly replacing another)
       - No – page fault, bring page from disk, update page table and TLB
         – Redo Translation
3. Check cache
   - Cache hit, get data
   - Cache miss, access memory (address ready)

- What if cache is indexed on virtual address?
Page Replacement; picking a victim

- Important policy because of cost of page fault; must make good choices

- LRU realistic approximation of oracle policy (because of locality)

- Must think about implementation of policy
  - if complex operations required at memory access – problems!
  - Ok to require complex operations at page fault time (or, in general, infrequently)

- We looked at FIFO
LRU Implementation is Difficult

- Time/Counter based implementation
  - record the time of the memory reference to a page
  - select the page with the oldest reference time
  - What is the cost at reference time? What is the cost at page fault time?

- Stack implementation
  - maintain a stack of the pages in memory
  - when referencing a page, remove it from the stack and store it at the top of the stack
  - What about this one?

- 0/1 Hardware Matrix implementation
  - Maintain a square matrix of bits
  - When referencing a page, set all its row bits to 1, and all its column bits to 0
  - Replace the page that has the least number of 1’s in its row
NFU – Simulating LRU in software

• Not Frequently Used (NFU)
  – Keep a counter with each page
  – At every clock interrupt, scan the resident page table and add the current R bit to the counter (reset R)
  – When you need a victim, pick lowest counter value

• Advantages/Disadvantages? How would it compare with optimal/LRU?

• Implementation? Only the update of the R-bit on every memory reference needed
NFU improvement

- Problem with NFU is that it never forgets!

- A heavily used page that is no longer needed will not easily be replaced

- Need a way to decrement counter with age

- Possible algorithm:
  - At every clock interrupt, shift counter to the right one bit (divide by two)
  - Add the R bit to the left most bit (giving it the most weight)
Not Recently Used Algorithm

- Maintain Two bits with the page table entry: R (referenced) and M (modified)

- A page can have (R, M) of:
  - Class 0: (0, 0) – not referenced, not modified
  - Class 1: (0, 1) – not referenced, modified (!)
  - Class 2: (1, 0) – referenced, not modified
  - Class 3: (1, 1) – referenced, modified

- When looking for a victim, pick randomly among the lowest class available

- Every so often (say on time-quanta interrupt) reset all R’s

- What do you think?

- A more crude version with only the R bit
The Clock Algorithm

- Organize the frames (logically) in a circle (clock; so far, identical to FIFO)
- Have a pointer to the current “oldest” page (clock hand)
- When a page replacement is necessary, check the frame at the clock hand
  - If the R-bit is 0 (not referenced), replace that page
  - If the R-bit is 1, reset the R-bit and move the clock hand to the next frame
  - Repeat until you find a page with $R = 0$
- Combines between FIFO and Reference Bits
- Used in unix
- Can also use the modified bit (Macintosh replacement)
Counting Based Algorithms

- Idea: keep a count of the memory references to each page
- Policy: replace the Least Frequently used Page (why?)
- Policy: replace the Most Frequently used Page (why?)
- How expensive?
Another Idea: Page buffering

- Move victims to a victim queue (Death Row!); usually FIFO, limited size

- Real victim comes from this victim queue

- If the page is referenced while still in victim queue, it is back to the regular queue (amnesty!)

- Can be combined with any of the other policies
Working Set Model

• General characteristics of Replacement Policies:
  – Local Replacement – a process will choose a victim among its own pages
  – Global Replacement – all pages are fair game
  – Fixed allocation – A process is given a fixed number of frames to work with
  – Variable allocation – the number of frames allocated to a process can change

• Resident Set: the number of pages that a process currently has in memory

• Working set: the number of pages that the process needs to run effectively (with few page faults)

• Would like to have resident set converge to working set
More Formal Definition of Working Set

- A good approximation of the “working set” is the number of pages that a process has accessed in its recent past.
- \( W(t, \Delta) \) gives the number of pages references in the last \( \Delta \) time units starting from time \( t \).
- How to pick \( \Delta \)? Working set size is a function of \( \Delta \) (\( W \) is nondecreasing function of \( \Delta \)).
- Working set also a function of time as the behavior of the process changes (time is virtual, or execution, time).
- Can we build a replacement policy based on this concept?
Work Set Based Replacement Policy

• Idea:
  – Monitor working set (pages accessed in last $\Delta$)
  – Periodically remove pages not in working set
  – Process executes if its set is resident

• Problems:
  – Past does not always predict the future
  – Impractical to keep track of the resident set
  – Optimal $\Delta$ unknown

• A couple of approximation algorithms:
  – Page Fault Frequency (PFF): if page fault rate of a process is too low, its resident set size can be reduced; if its high, then need to increase it (allow working set to be resident)
  – Variable-interval sampled working set: every interval discard the pages that have 0 use bit; during the interval, any pages that are needed are added to the set
**Problem 5:** (10 points) Consider a byte-addressable computer that supports segmentation. There are 26 bits in a virtual address, of which 5 are the segment number. The computer can access at most 1 GB of physical memory. Answer the following and explain your reasoning.

(a) How many bits in a physical address?  
(b) How many entries in the segment table?  
(c) What is the maximum size of a segment?  
(d) How many bits in a base register?  
(e) How many bits in a bound (segment length) register?  
(f) How many processes will fit into the computer’s memory simultaneously, if each process is using all of its segments and each segment is its maximum size?  
(g) Suppose you decide to increase the maximum segment size. How big can you make it for this computer?
Problem 10.3, Textbook

A certain computer provides its users with a virtual memory space of $2^{32}$ bytes. The computer has $2^{18}$ bytes of physical memory. The virtual memory is implemented by paging, and the page size is 4096 bytes. A user process generates the virtual address 11123456. Explain how the system establishes the corresponding physical location. Distinguish between software and hardware operations.
An operating system supports a paged virtual memory, using a central processor with a cycle time of 1 microsecond. It costs an additional microsecond to access a page other than the current one. Pages have 1000 words and the paging device is a drum that rotates at 3000 revolutions per minute and transfers one million words per second. The following statistical measurements were obtained for the system:

- one percent of all instructions executed accessed a page other than the current one
- Of the instructions that accessed another page, 80 percent accessed a page already in memory.
- When a new page was required, the replaced page was modified 50 percent of the time.

Calculate the effective instruction time on this system assuming that the system is running one process only and that the processor is idle during drum transfers.
A page replacement algorithm should minimize the number of page faults. We can do this minimization by distributing heavily used pages evenly over all of memory, rather than having them compete for a small number of page frames. We can associate with each page frame a counter of the number of pages that are associated with that frame. Then, to replace a page, we search for the page frame with the smallest counter.

a. Define a page replacement algorithm using this basic idea. Specifically, address the problems of: (1) what the initial values of the counters is, (2) when counters are increased, (3) when counters are decreased, (4) how the page to be replaced is selected.

b. how many page faults occur for your algorithm for the following reference string, for four page frames? 1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9, 5, 4, 5, 4, 2

c. What is the minimum number of page faults for an optimal page replacement strategy for the reference string in part b with 4 page frames?
Midterm 2, 2001

Problem 3: (15 points; 15 minutes) For a machine with 3 physical frames. Compare FIFO and LRU for the following reference string: 8, 5, 2, 1, 2, 5, 4, 6, 5, 2, 1, 5, 4, 2. If you had only two frames, would either algorithm suffer from Belady’s anomaly?
Problem 5: (20 points; 15 minutes)
Consider the following deadlock algorithms. Start with deadlock avoidance (banker’s algorithm), but replace the safety check with one that finds a sequence considering the requested resource only. For example, if the new request asks for 2 tape drives, check if there is a sequence of processes that will finish based on their tape drive needs only.

(a) (15 points) Does this algorithm successfully avoid deadlock? If yes, show how. If no, show a counter example.

(b) (5 points) Is this algorithm is more liberal or more conservative than banker’s algorithm?
Problem 6: (10 points) Two processes share two buffers through which they exchange data. Thread 1 places work for Thread 2 in the “input” buffer. Thread 2 removes one item from the input buffer, processes it, and places the result back to the output buffer where it is eventually removed by the other thread. There is a fixed amount of memory available in which both buffers reside. The two buffers can grow dynamically in size as needed. Explain how deadlock can happen. Suggest two ways to prevent it and compare their relative merits.
Problem 3: (20 pts)

- (a) (4) A computer system has d tape drives for which 10 processes compete. Each process may request up to 2 drives. For what values of d is the system deadlock free?
- (b) (8) What value of d would guarantee the same for a system of n processes each requesting up to r drives ($r < d$)?
- (c) (8) A potential deadlock management technique is to chose n such that the formula you developed in (b) holds for every resource. Discuss two more liberal schemes for managing deadlock situations than enforcing the limit above. Demonstrate (using examples if necessary) that they are more liberal.
**Problem 4:** (20 pts)
Which of the following events can lead from a safe to unsafe state; which can lead from a deadlock-free to a deadlocked state: (a) A process finishes (releasing its resources); (b) a process increases its claim; (c) A new process comes in (with a non-zero claim vector); (d) A process makes a request within its claim. Explain (be specific, use examples if necessary).
Problem 5: (20 pts – Token open-ended question)
Because of your recent successes in the scheduling group in Chimera, you have been promoted to the head of the memory management group. As a first step, you are entertaining proposals on how to structure the memory from your team-members. One proposal suggests having fixed partitions with each process getting exactly one partition. Each process is free to manage its partition any way it feels is best for it (simple or virtual segments or pages, buddy system, fixed/dynamic partitions). Comment on the advantages/disadvantages of this solution. It might help to consider some specific organizations and what support would be necessary for them. Be concise and precise.
Problem 6: (20 pts)

• (a) (8) Consider an inverted page table scheme with the hash table pinned (always in memory). The details of the hashing function are unimportant; chaining is used if two values hash into the same location. Explain how a page number is translated using this scheme; what is the maximum number of memory references necessary?

• (b) (12) Explain in detail what happens when a memory reference occurs on a system with virtual memory, a TLB and a cache using an inverted page table. If access to the page table is necessary, reference your answer in part a (do not explain it again). What is the maximum number of page faults possible? What is the maximum number of memory references? (Note: you can answer this part even if you couldn’t get part (a))