CS323 Operating Systems
Memory Management IV

Yuanyuan Zhou
Lecture 16
2/23/2003

Content of this lecture

- Administrative announcements
- Inverted Page Table
- Multi-level page table
- Page Sharing and protection
- Demand paging
- Page Replacement

Administrative

- MP2
- Quiz2

Review

- Page Table
  - Paging mapping hardware
- TLB: cache of page table
  - TLB miss

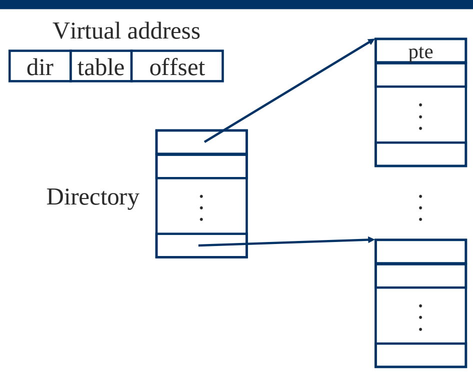
Effective Access Time

- TLB lookup time = ε time unit
- Memory cycle -- m microsecond
- TLB Hit ratio -- α
- Effective access time
  - Eat = $(1m+\varepsilon)\alpha+(2m+\varepsilon)(1-\alpha)$
  - Eat = $2m+\varepsilon-\alpha$

Multilevel Page Tables

- Since the page table can be very large, one solution is to page the page table
- Divide the page number into
  - An index into a page table of second level page tables
  - A page within a second level page table
- Advantage
  - No need to keeping all the page tables in memory all the time
  - Only recently accessed memory's mapping need to be kept in memory, the rest can be fetched on demand

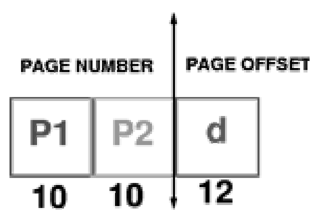
Multiple-Level Page Tables



What does this buy us? Sparse address spaces and easier paging

Example Addressing on a Multilevel Page Table System

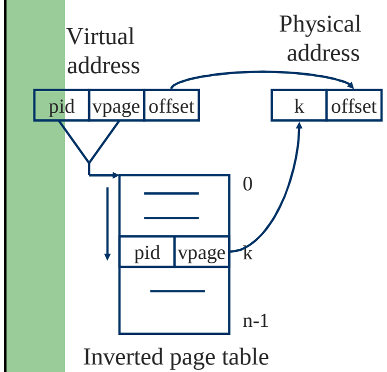
- A logical address (on 32 bit machine with 4k page size) is divided into
  - A page number consisting of 20 bits
  - A page offset consisting of 12 bits
- Divide the page number into
  - A 10-bit page number
  - A 10-bit page offset



Multilevel Paging and Performance

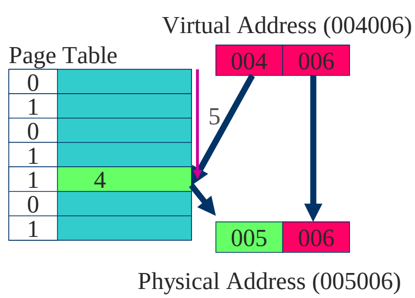
- Since each level is stored as a separate table in memory, converting a logical address to a physical one in a four-level paging may take five memory accesses. Why?

Inverted Page Tables



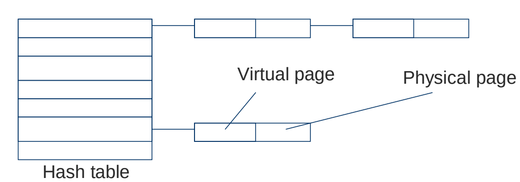
- Main idea
  - One PTE for each physical page frame
  - Hash (Vpage, pid) to Ppage#
  - Trade off space for time
- Pros
  - Small page table for large address space
- Cons
  - Lookup is difficult
  - Overhead of managing hash chains, etc

Inverted Page Table



Inverted Page Table Implementation

- TLB is same as before
- TLB miss is handled by software
- In-memory page table is managed using a hash table
  - number of entries = number of physical frames
  - Not found: page fault



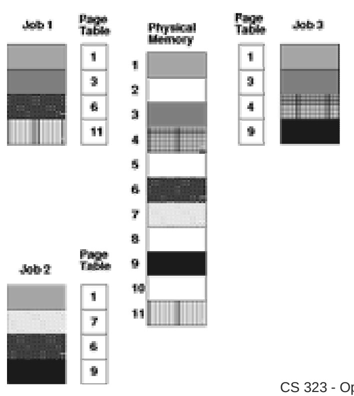
Group Discussion

- Can processes share pages?
- What pages can be shared?
- How do they share?

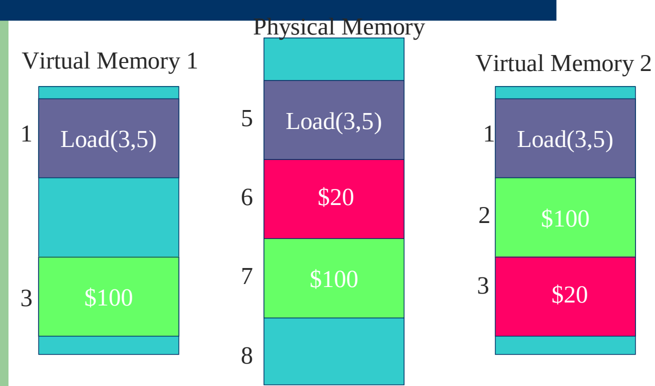
Sharing Pages

- Code and data can be shared among processes
  - mapping them into pages with common page frame mappings
- Code and data must be position independent if VM mappings for the shared data are different
- Code and data cannot store VM addresses for functions and variables not in shared pages
- Shared code and data usually are not position independent resulting in certain regions of memory being reserved for shared libraries and the operating system

Shared Pages



Incorrect Sharing



Protection

- Can add read, write, execute protection bits to page table to protect memory
- Check is done by hardware during access
- shared memory location
  - different protections from different processes
  - Solution:
    - associate protection lock with page frame. Each process has its own key. If the key fits the lock, the process may access the page frame

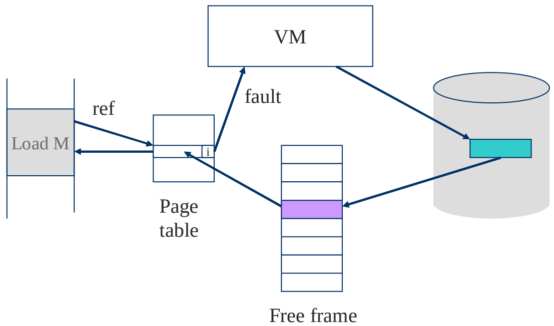
Paging Policies

- Fetch Strategies
  - When should a page be brought into primary (main) memory from secondary (disk) storage.
- Placement Strategies
  - When a page is brought into primary storage, where is it to be put?
- Replacement Strategies
  - Which page now in primary storage is to be removed from primary storage when some other page or segment is to be brought in and there is not enough room.

Demand Paging

- Algorithm Never bring a page into primary memory until its needed.
  1. Page fault
  2. Check if a valid virtual memory address. Kill job if not.
  3. If valid reference, check if its cached in memory already (perhaps for some other process.) If so, skip to 7).
  4. Find a free page frame.
  5. Map address into disk block and fetch disk block into page frame. Suspend user process.
  6. When disk read finished, add vm mapping for page frame.
  7. If necessary, restart process.

Demand Paging Example



Page Replacement

1. Find location of page on disk
2. Find a free page frame
   1. If free page frame use it
   2. Otherwise, select a page frame using the page replacement algorithm
   3. Write the selected page to the disk and update any necessary tables
3. Read the requested page from the disk.
4. Restart the user process.
5. It is necessary to be careful of synchronization problems. For example, page faults may occur for pages being paged out.

Issue: Eviction

- Hopefully, kick out a less-useful page
  - Dirty pages require writing, clean pages don't
    - Hardware has a dirty bit for each page frame indicating this page has been updated or not
  - Where do you write? To "swap space"
- Goal: kick out the page that's least useful
- Problem: how do you determine utility?
  - Heuristic: temporal locality exists
  - Kick out pages that aren't likely to be used again

Terminology

- **Reference string**: the memory reference sequence generated by a program.
- **Paging** – moving pages to (from) disk
- **Optimal** – the best (theoretical) strategy
- **Eviction** – throwing something out
- **Pollution** – bringing in useless pages/lines

Page Replacement Strategies

- **The Principle of Optimality**
  - Replace the page that will not be used again the farthest time in the future.
- **Random page replacement**
  - Choose a page randomly
- **FIFO - First in First Out**
  - Replace the page that has been in primary memory the longest
- **LRU - Least Recently Used**
  - Replace the page that has not been used for the longest time
- **LFU - Least Frequently Used**
  - Replace the page that is used least often
- **NRU - Not Recently Used**
  - An approximation to LRU.
- **Working Set**
  - Keep in memory those pages that the process is actively using.

Principal of Optimality

- Description:
  - Assume that each page can be labeled with the number of instructions that will be executed before that page is first references, i.e., we would know the future reference string for a program.
  - Then the optimal page algorithm would choose the page with the highest label to be removed from the memory.
- This algorithm provides a basis for comparison with other schemes.
- Impractical because it needs future references
- If future references are known
  - should not use demand paging
  - should use pre paging to allow paging to be overlapped with computation.

Optimal Example

12 references, 7 faults

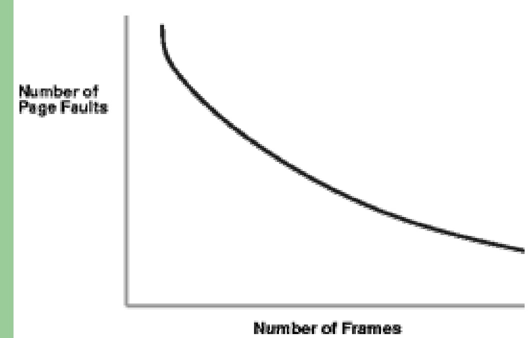
Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A		
B	yes	B	A	
C	yes	C	B	A
D	yes	D	B	A
A	no	D	B	A
B	no	D	B	A
E	yes	E	B	A
A	no	E	B	A
B	no	E	B	A
C	yes	C	E	B
D	yes	D	C	E
E	no	D	C	E

FIFO

12 references, 9 faults

Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A		
B	yes	B	A	
C	yes	C	B	A
D	yes	D	C	B
A	yes	A	D	C
B	yes	B	A	D
E	yes	E	B	A
A	no	E	B	A
B	no	E	B	A
C	yes	C	E	B
D	yes	D	C	E
E	no	D	C	E

Paging Behavior with Increasing Number of Page Frames



Belady's Anomaly (for FIFO)

As the number of page frames increase, so does the fault rate.

12 references, 10 faults

Page Refs	4 Page Frames			
	Fault?	Page Contents			

Page Refs	Fault?	Page Contents			
A	yes	A			
B	yes	B	A		
C	yes	C	B	A	
D	yes	D	C	B	A
A	no	D	C	B	A
B	no	D	C	B	A
E	yes	E	D	C	B
A	yes	A	E	D	C
B	yes	B	A	E	D
C	yes	C	B	A	E
D	yes	D	C	B	A
E	yes	E	D	C	B

LRU

12 references, 10 faults

3 Page Frames

Page Refs	Fault?	Page Contents		
A	yes	A		
B	yes	B	A	
C	yes	C	B	A
D	yes	D	C	B
A	yes	A	D	C
B	yes	B	A	D
E	yes	E	B	A
A	no	A	E	B
B	no	B	A	E
C	yes	C	B	A
D	yes	D	C	B
E	yes	E	D	C

Least Recently Used Issues

- Does not suffer from Belady's anomaly
- How to track “recency”?
  - use time
    - record time of reference with page table entry
    - use counter as clock
    - search for smallest time.
  - use stack
    - remove reference of page from stack (linked list)
    - push it on top of stack
- both approaches require large processing overhead, more space, and hardware support.

LRU and Anomalies

Anomalies cannot occur, why?

12 references, 8 faults

4 Page Frames

Page Refs	Fault?	Page Contents			
A	yes	A			
B	yes	B	A		
C	yes	C	B	A	
D	yes	D	C	B	A
A	no	A	D	C	B
B	no	B	A	D	C
E	yes	E	B	A	D
A	no	A	E	B	D
B	no	B	A	E	D
C	yes	C	B	A	E
D	yes	D	C	B	A
E	yes	E	D	C	B

NUR: A LRU Approximation

- NRU: Evict a page that is NOT recently used;
  LRU: evict a page that is LEAST recently used;
- NRU Implementation: simpler than LRU
  - additional reference bits
  - a register is kept per page
  - a one bit is set in the register if the page is referenced
  - the register is shifted by one after some time interval
  - 00110011 would be accessed more recently than 00010111
  - the page with register holding the lowest number is the least recently used.
  - the value may not be unique. use FIFO to resolve conflicts.

Second Chance

- Only one reference bit in the page table entry.
  - 0 initially
  - 1 When a page is referenced
- pages are kept in FIFO order using a circular list.
- Choose "victim" to evict
  - Select head of FIFO
  - If page has reference bit set, reset bit and select next page in FIFO list.
  - keep processing until reach page with zero reference bit and page that one out.
- system v, r4 uses a variant of second chance

Second Chance

- if page has reference bit set, reset bit and select next page in FIFO list.
- keep processing until reach page with zero reference bit and page that one out.
- system v, r4 uses a variant of second chance.

Second Chance Example

12 references,
9 faults

Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A*		
B	yes	B*	A*	
C	yes	C*	B*	A*
D	yes	D*	C	B
A	yes	A*	D*	C
B	yes	B*	A*	D*
E	yes	E*	B	A
A	no	E*	B	A*
B	no	E*	B*	A*
C	yes	C*	E	B
D	yes	D*	C*	E
E	no	D*	C*	E*

Second Chance Example

12 references,
9 faults

Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A*		
B	yes	B*	A*	
C	yes	C*	B*	A*
D	yes	D*	C	B
A	yes	A*	D*	C
B	yes	B*	A*	D*
E	yes	E*	B	A
A	no	E*	B	A*
B	no	E*	B*	A*
C	yes	C*	E	B
D	yes	D*	C*	E
E	no	D*	C*	E*

Reminder

- Quiz2
- Mp2

- MP2