

Lecture #15: Translation, protection, sharing

Review -- 1 min

Goals of virtual memory:

- protection
- relocation
- sharing
- illusion of infinite memory
- minimal overhead
 - space
 - time

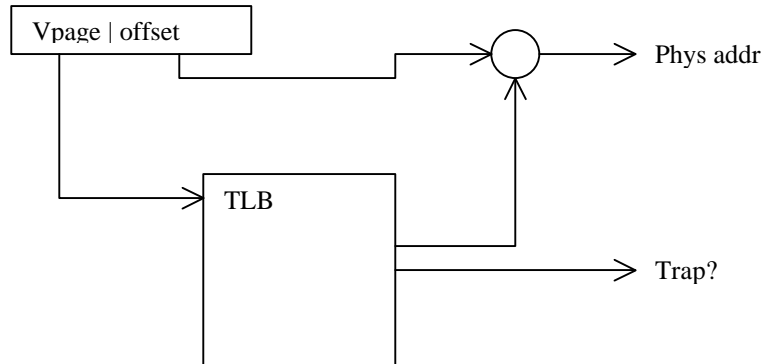
Last time: we ended with page table

Evaluation:

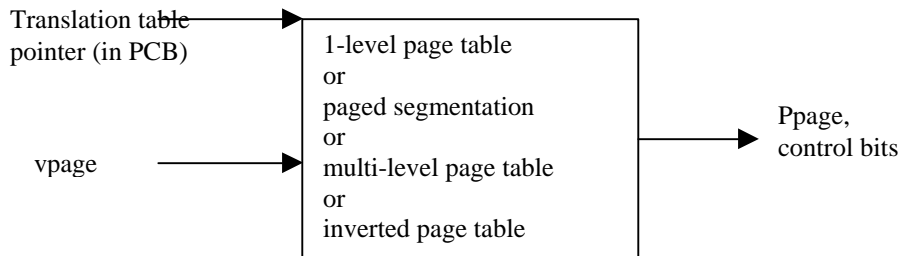
Paging

- + simple memory allocation
- + easy to share
- big page tables if sparse address space QUESTION: WHY?

Modern hardware is always:



Memory data structure is opaque object:



Outline - 1 min

Finish last time:

Scalable schemes:

- Multi-level translation
- Inverted page table
- Others in book: paged page tables, segmented paging, paged segmentation, hashed page tables, ...

Quantitative measures:

- Space overhead: internal v. external fragmentation, data structures
- Time overhead: AMAT: average memory access time

Paging to disk

Basic mechanism

Writing and sharing – dirty bit and core map
performance

Preview - 1 min

Replacement policies

Control bits revisited

Lecture - 35 min

Finish last time:

Scalable schemes:

- Multi-level translation
- Inverted page table
- Others in book: paged page tables, segmented paging, paged segmentation, hashed page tables, ...

Quantitative measures:

- Space overhead: internal v. external fragmentation, data structures
- Time overhead: AMAT: average memory access time

Admin - 3 min

Solutions HW2 available

HW 3 out

Project 2 out

Lecture - 35 min

1. Virtual memory – paging to disk

Reality: finite physical memory

Abstraction: infinite memory

Idea is that we want to be able to run a job that won't fit in memory or run more jobs than will fit

Why infinite memory?

- run large process (> physical memory)
- run many processes ("swap" out processes that are not currently running)

How important is virtual memory paging to disk?

"In operating systems, when you see the word "virtual", substitute the word "slow"."

VM invented in late 60's/early 70's – memory > \$10K/MB (today

<\$0.1) → less important to oversubscribe

70's – many users per MB 90's – many MB's per user

70's – disk a lot slower than CPU or mem 90's disk much much slower than CPU or mem

still – convenient – can start up hundreds of shells @1MB each w/o worrying, etc...

in 1970's inventing VM was quite difficult

Now that we know how to do it, not that hard. So it is worth having around.

1.1 implementing virtual memory paging to disk

Basic idea – page table entry:

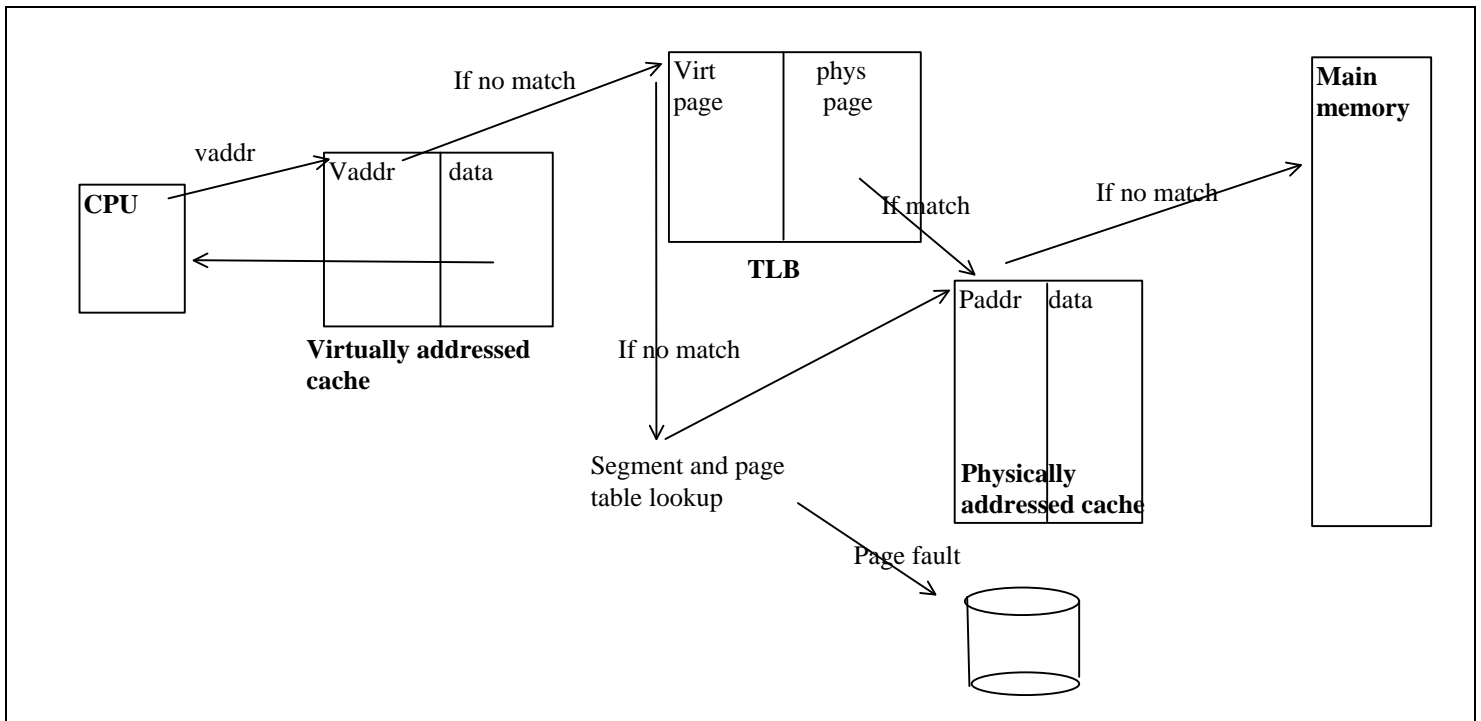
Frame number | control bits

Add a new control bit “paged”

Valid	paged	frame number
1	0	holds frame number of physical memory
1	1	holds pointer to copy of frame on disk
0	X	address is invalid

page fault – memory reference to data that is currently on disk. Need to bring data from disk into memory to complete reference.

Question: How do you handle a page fault?



How do you handle a “page fault”

- 1) TLB lookup – miss; fault to OS
- 2) OS traverses page tables, seg tables, etc. Find page table entry
QUESTION: ok –let’s assume we have pge table – what do we see?
- 3) Find a free physical frame
QUESTION – what would be a good way to implement this?
- 4) Schedule disk operation to read page into newly allocated frame
QUESTION -- how do we know disk address to read from?
- 5) when disk read is complete, update page table and TLB
- 6) restart OS instruction (tricky in general, but knowing what you know about traps/faults how would this work?)

Questions:

- 1) Book says “invalid = paged to disk” (no “paged” bit). In that case, how do we tell difference between "invalid" page and "paged out" page?
- 2) on page fault page-in, what consistency issues are there (e.g., what might need to be invalidated in the picture?)
- 3) on context switch, what consistency issues are there (e.g., what might need to be invalidated in the picture?)

2. Paging to disk: implementation issues – writing and sharing

2.1 Writing

what happens if a page is written?

Write through – send write to immediately lower level (disk)

Write back – send write to lower level when block evicted from higher level

Which one would we use here?

QUESTION: How would you implement write back? How do you know which pages need to be written back and which are OK?

Dirty bit:

- implemented in TLB – when TLB sees a write request to a page, it sets the “dirty bit” in TLB. When evicted from TLB, need to copy dirty bit to page table and core map
- how do you invalidate a dirty page
 - write to disk then mark invalid

2.2 Sharing

Two different virtual addresses can map to the same physical frame

What happens if that frame is paged out (or paged in or moved?)

Solution: core map

coreMap[physPage] is a data structure that tracks info you may want in a replacement policy (e.g., last reference time, dirty, reference count, virtual page to invalidate)

3. Performance of demand paging

Suppose p is probability of a memory reference causing a page fault

$$AMAT = (1-p) * \text{memory time} + p * \text{page fault time}$$

Problem: memory time $O(100\text{ns})$, disk time $O(10\text{ms} = 10^7\text{ns})$

QUESTION: what does p need to be to ensure that paging hurts performance by less than 10%?

$$1.1 * t_{\text{mem}} = (1-p)t_{\text{mem}} + p t_{\text{disk}}$$

$$p = (.1 t_{\text{mem}})/(t_{\text{mem}} + t_{\text{disk}})$$

$$\approx (.1 * 10^2)/(10^7+10^2)$$

$$\approx 10^{-6}$$

At most one access out of 1,000,000 can be a page fault. (Hit rate greater than 99.9999%!)

4. Thrashing

Thrashing – memory overcommitted – pages tossed out while still needed

Example – one program touches 50 pages (each equally likely); only have 40 physical page frames

If have enough pages – 100ns/ref

If have too few pages – assume every 5th reference → page fault
4refs x 100ns

1 page fault x 10ms for disk I/O

→ 5 refs per 10ms + 400ns = 2ms/ref = **20,000x slowdown!!!**

Really important -- even if you don't build OS's, you need to be aware of thrashing.

4.1 Problem: system doesn't know what it is getting in to

Log more and more users into system – eventually:

total number of pages needed > number of pages available

Picture: jobs/sec v. total system throughput

So, what do you do about this?

1) One process alone too big?

Change program so it needs less memory or has better locality

For example, split matrix multiply into smaller sub-matrices that each fit in memory

2) Several jobs?

■ figure out needs/process (working set)

■ run only groups that fit (balance sets) – kick other processes out of memory

e.g., suppose you are paging and running at a 10,000x slowdown, if kicking half of the jobs out would get you to stop paging and run at full speed, you trade a 1.5x slowdown for a 10,000x slowdown

Remember -- issue here is not total size of process, but rather total number of pages being used at the moment.

How do we figure needs/process out?

4.1.1 Working set (denning, MIT mid 60's)

Informally – collection of pages a process is using right now

Formally – set of pages job has referenced in last T seconds

How do we pick T?

1 page fault = 10ms

10ms = 2M instructions

→ T needs to be a lot bigger than 1 million instructions

How do you figure out what working set is?

■ replacement policy keeps track of time to last access – use information from it (next lecture)

- a) modify clock algorithm so that it seeps at fixed intervals (keep idle time/page; how many seconds since last reference)
- b) with second chance list – how many seconds since got put on 2nd chance list

Now you know how many pages each program needs. What do you do?

Balance set

- 1) if all fits? Done
- 2) if not? Throw out fat cats; bring them back eventually

What if T too big?

→ waste memory – too few programs fit in memory

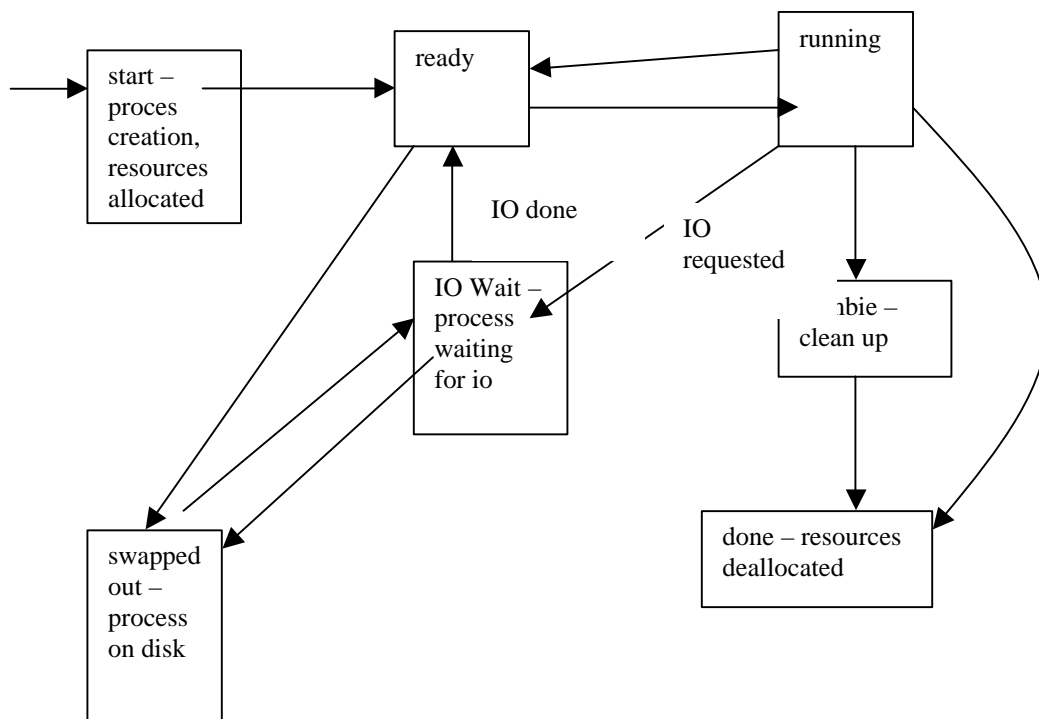
What if T too small?

→ thrashing

5. Swapping v. paging

If system low on memory, may be better off moving an entire process to disk and stopping it from running for a while

System stop thrashing -> system runs more efficiently → average (and perhaps all) jobs run faster



5.1.1 Need two levels of scheduling

Upper level decides on swapping

- when
- who
- for how long

Lower levels decide who on ready queue actually runs on CPU

Upper level invoked when there are processes swapped out and whenever we need to load more programs than can fit in main memory