LECTURE 8:

The Page Fault (+ Policy)

Last time(s):

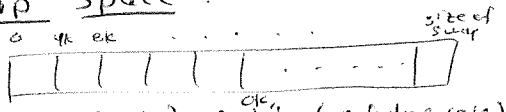
Paging:
too slow → TLB
too big → multi-level PT

Today:
what if all that stuff doesn't fit into memory?
(Q when does this occur?)
A: one program w/ lots of data
many programs sharing mem

Overview:
{ Mechanisms
Policies }

Mechanisms

Swap Space: a disk

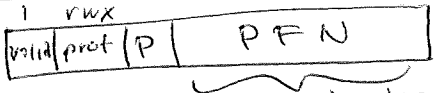


read (addr) => old data (or failure code)
write (addr, data) => old or failure

size (Q) how big is a modern hard drive?

This, page may be in-memory or on disk
=> OS needs to track this for all pages in system
(Q) how? => in page table!

Page Table Entry



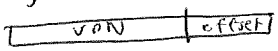
where in phys this page is

new bit: present bit
if page is in memory => P=1
else P=0

note: when present=0, can use PFN field
(Q) what to use it for?

New Control Flow

Trying to fetch instruction @ VA



1) extract VPN
2) check TLB for VPN
2a) TLB Hit: get PFN | offset => Fetch PA from memory
2b) TLB miss: calculate address of PTE
PA_{PTE} = (PTBR + (VPN * sizeof(PTE))
=> Fetch PA_{PTE} Raise Exception (SEG FAULT)
=> if not valid => " (PROT FAULT)
if prot not OK =>
if present:
extract PFN
insert => TLB
retry
else
Raise Exception (PAGE FAULT)

should be PAGE MISS

=> Page Fault Handler
PFN = FindFreePhysPage()
if (PFN == -1)
(may sleep) EvictPage(); // using page replacement policy
(may sleep) DiskRead(PTE.DiskAddr, PFN);
Update PTE: (present=1, PFN)
retry instruction

Policy: which page to evict? (replace)

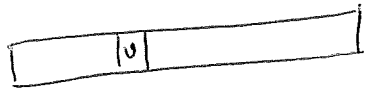
Optimal: kick out page ref'd furthest in future
Random: pick any one, kick it out
FIFO: keep FIFO list of pages
LRU: kick out least-recently-used page
note: difference in updates (LRU updates list on every access)

Example: 0 1 2 0 1 3 0 3 1 2 1
optimal: misses (5)
FIFO: misses (7)
LRU: misses (5)
(history) (also, LFU)

[DO EXAMPLE]

Conclude: Good to do LRU-like policy
But: how to implement?

Approximate: h/w support via use bit (reference bit)

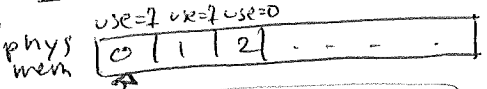


upon every access to a page, h/w sets use bit (to 1)
thus, after running for a while, can differentiate which are in use, which aren't

(Q) is this enough for whole policy?
no, OS must clear bits (use=0) periodically too

clock algorithm

(Q) are all pages equally easy to evict?
modify bit



clock hand

```
if use = 1
    clear (use => 0)
    inc clock
else
    replace (this page)
```

Finally:
Thrashing: (working set): actively used pages
when $\sum$ WS_i > phys memory is over committed
solutions: admission control, kill some procs,
=> Buy more memory
last (Q) (does this always work?)

Belady's Anomaly:
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
w/ FIFO_3 and FIFO_4