

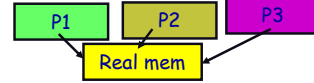
# CS 140: Operating Systems

## Lecture 13: Thrashing

Mendel Rosenblum

### Thrashing: exposing the lie of VM

- ◆ Thrashing: processes on system require more memory than it has.



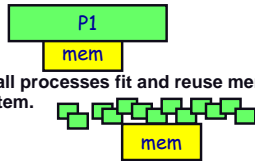
Each time one page is brought in, another page, whose contents will soon be referenced, is thrown out. Processes will spend all of their time blocked, waiting for pages to be fetched from disk I/O devs at 100% utilization but system not getting much useful work done

- ◆ What we wanted: virtual memory the size of disk with access time of physical memory
- ◆ What we have: memory with access time = disk access

### Thrashing

- ◆ Process(es) "frequently" reference page not in mem  
Spend more time waiting for I/O then getting work done
- ◆ Three different reasons  
Process doesn't reuse memory, so caching doesn't work (past != future)

Process does reuse memory, but it does not "fit"



Individually, all processes fit and reuse memory, but too many for system.

### When does thrashing happen?

- ◆ (Over-)simple calculation of average access time:  
Let  $h$  = percentage of references to pages in memory  
Then average access time is  
 $h * (\text{cost of memory access}) + (1-h) * (\text{cost of disk access} + \text{miss overhead})$   
For current technology, this becomes (about)  
 $h * (100 \text{ nanoseconds}) + (1-h) * (10 \text{ milliseconds})$   
Assume 1 out of 100 references misses.  
 $= .99 * (100\text{ns}) + .01 (10\text{ms})$   
 $= .99 (100\text{ns}) + .01 (10,000,000\text{ns})$   
 $= 99 + 100,000 \sim 100 \text{ microseconds}$   
or, 1000x slower than main memory.
- ◆ Even small miss rates lead to unacceptable average access times. What can OS do???

### Making the best of a bad situation

- ◆ Single process thrashing?  
If process does not fit or does not reuse memory, OS can do nothing except contain damage. (cs140?).
- ◆ System thrashing?  
If thrashing arises because of the sum of several processes then adapt:  
Figure out how much memory each process needs  
Change scheduling priorities to run processes in groups whose memory needs can be satisfied (shedding load)  
If new processes try to start, can refuse (admission control)
- ◆ Careful: example of technical vs social.  
OS not only way to solve this problem (and others).  
"Social" solution: go to Fry's and buy more memory.  
Another: use 'ps' to find idiot killing machine and go yell

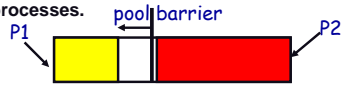
### Methodology for solving?

- ◆ Approach 1: working set  
Thrashing viewed from a caching perspective: given locality of reference, how big a cache does the process need?  
Or: how much memory does process need in order to make "reasonable" progress (its working set)?  
Only run processes whose memory requirements can be satisfied.
- ◆ Approach 2: page fault frequency  
Thrashing viewed as poor ratio of fetch to work  
PFF = page faults / instructions executed  
If PFF rises above threshold, process needs more memory not enough memory on the system? Swap out.  
If PFF sinks below threshold, memory can be taken away

### Working set (1968, Denning)

- ◆ What we want to know: collection of pages process must have in order to avoid thrashing  
This requires knowing the future. And our trick is?
- ◆ Working set:  
Pages referenced by process in last  $T$  seconds of execution considered to comprise its working set  
 $T$ : the working set parameter
- ◆ Uses?  
Cache partitioning: give each app enough space for WS  
Page replacement: preferentially discard non-WS pages  
Scheduling: process not executed unless WS in memory

### Recall: per-process page caches

- ◆ Per-process (per-user same)  
Each process has a separate pool of pages.  
A page fault in one process can only replace one of this process's frames.  
Isolates process and therefore relieves interference from other processes.
- 
- ◆ Adjust cache by making each private cache about as big as process's working set.  
Result: allows process to use other's (comparatively) idle resources while still providing isolation.

### Scheduling details: The balance set

- ◆ Sum of working sets of all runnable processes fits in memory? Scheduling same as before.
- ◆ If they do not fit, then refuse to run some: divide into two groups  
Active: working set loaded.  
Inactive: working set intentionally not loaded.  
Balance set: sum of working sets of all active processes.
- ◆ Long term scheduler:  
Keep moving processes from active -> inactive until balance set less than memory size.  
Must allow inactive to become active. (if changes too frequently?)
- ◆ As working set changes, must update balance set...

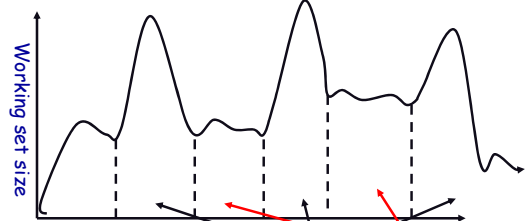
### How to implement working set?

- ◆ Associate an **idle time** with each page frame  
Idle time = amount of CPU time received by process since last access to page  
(why not amount of time since last reference to page?)  
Page's idle time >  $T$ ? Then page not part of working set
- ◆ How to calculate?  
Scan all resident pages of a process  
Use bit on? clear page's idle time, clear use bit.  
Use bit off? add process CPU time (since last scan) to idle time.
- ◆ Unix:  
Scan happens every few seconds.  
 $T$  on order of a minute or more.

### Some problems

- ◆  $T$  is magic  
What if  $T$  too small? Too large?  
How did we pick it? Usually "try and see"  
Fortunately, system's aren't too sensitive.
- ◆ What processes should be in the balance set?  
Large ones so that they exit faster?  
Small ones since more can run at once?
- ◆ How do we compute working set for shared pages?  
Shared Memory.  
Bill for all of the library? Used part?

### Working sets of real programs



- ◆ Typical programs have phases: **transition**, **stable**  
Working set of one may have little to do with other.  
Balloons during transitions....

### Working set less important

---

- ◆ The concept is a good perspective on system behavior.  
As optimization trick, it's less important: Early systems thrashed lots, current systems not so much.
- ◆ Have OS designers gotten smarter?
- ◆ No. It's the hardware guys (cf. Moore's law):  
Obvious: Memory much larger (more to go around).  
Less obvious: CPU faster so jobs exit quicker, return memory to freelist faster.  
Some app can eat as much as you give, the percentage of them that have "enough" seems to be increasing.  
Social implication: while speed very important OS research topic in 80-90s, less so now (fair amount of social inertia though...)