UNIVERSITY of WISCONSIN-MADISON
Computer Sciences Department

CS 537
Introduction to Operating Systems

Andrea C. Arpaci-Dusseau
Remzi H. Arpaci-Dusseau

Virtual Memory: Working Sets

Questions answered in this lecture:

How to allocate memory across competing processes?

What is thrashing? What is a working set?

How to ensure working set of all processes fit?

Allocating Memory across Processes

Scenario:

- Several physical pages allocated to processes A, B, and C. Process B page faults.
- Which page should be replaced?
- Three options...

Per-process replacement

- Each process has separate pool of pages
  - Fixed number of pages (e.g., Digital VMS)
  - Fixed fraction of physical memory (1/P)
  - Proportional to size of allocated address space
- Page fault in one process only replaces pages of that process
  - Perform replacement (e.g., LRU) over only those pages
- Advantage: No interference across processes
- Disadvantage: Potentially inefficient allocation of memory
  - How to handle sharing of pages?

Allocating Memory across Processes

Per-user replacement

- Each user has separate pool of pages
- Advantage: Fair across different users
- Disadvantage: Inefficient allocation

Global replacement

- Pages from all processes lumped into single replacement pool
  - Example: Run clock over all page frames
- Each process competes with other processes for frames
- Advantages:
  - Flexibility of allocation
  - Minimize total number of page faults
- Disadvantages
  - One memory-intensive process can hog memory, hurt all processes

Impact of Additional Processes

What happens to "performance" as add more processes?

- Consider CPU utilization as metric
- Increase number of processes from 1
  - Process blocks: Other processes can run
  - CPU utilization increases with more processes
- Increase number of processes after memory filled
  - Increases number of page faults
  - Memory contention increases with more processes
  - CPU utilization decreases with more processes

Overcommitting Memory

When does the Virtual Memory illusion break?

Example:

- Set of processes frequently referencing 33 important pages
- Physical memory can fit 32 pages

What happens?

- Process A references page not in physical memory
- OS runs another process B
- OS replaces some page in memory with page for A
- How long can B run before it page faults?
  - Cycle continues...

Thrashing

System is reading and writing pages instead of executing useful instructions

- Implication: Average memory access time = disk access time
- Memory appears as slow as disk, instead of disk appearing as fast as memory

Average access time calculation

- H: Percentage of references that hit page in physical memory
- $C_{AccessMemory}$: Cost of referencing page in memory (e.g., 100 ns)
- $C_{PageFault}$: Cost of page fault (e.g., 20 ms or 20,000,000ns)
- $H * C_{AccessMemory} + (1-H) * C_{PageFault}$

Example: 1 out of every 33 references misses, H = 97%

- 0.97 * (100 ns) + (0.03) * (20000000ns) = 750000 ns = 750 us
- More than 1000 times slower than physical memory access

Need very high hit rate for acceptable performance

Motivation for Solution

Thrashing cannot be fixed with better replacement policies

- Page replacement policies do not indicate that a page must be kept in memory
- Only show which pages are better than others to replace

Student's analogy to thrashing: Too many courses

- Solution: Drop a course

OS solution: Admission control

- Determine how much memory each process needs
- Long-term scheduling policy
  - Run only those processes whose memory requirements can be satisfied
- What if memory needs of one process are too large????

Working Set

Informal definition

- Collection of pages the process is referencing frequently
- Collection of pages that must be resident to avoid thrashing

Formal definition

- Assume locality; use recent past to predict future
- Pages referenced by process in last T seconds of execution
- Working set changes slowly over time

Example:

- Page reference stream:
- A B A B C B A C A C D C D E B E D F B F D B E D

Balance Set

Motivation: Process should not be scheduled unless working set is resident in main memory

Divide runnable processes into two groups:

- Active: Working set is loaded
- Inactive: Working set is swapped to disk

Balance set: Sum of working sets of all active processes

Interaction with scheduler

- If balance set exceeds size of memory, move some process to inactive set
  - Which process???
- If balance set is less than size of memory, move some process to active set
  - Which process?
- Any other decisions?

Possible Implementations

Must constantly update working set information

Initial proposal:

- Store capacitor with each page frame
- Charge capacitor on page reference
- Capacitor discharges slowly if page not referenced
- T determined by size of capacitor

Problems with this proposal?

Working Set Implementation

Leverage `use` bits (as in clock algorithm)

OS maintains `idle time` for each page

- Amount of CPU received by process since last access to page
- Periodically scan all resident pages of a process
  - If `use` bit is set, clear page's idle time
  - If `use` bit is clear, add process CPU time (since last scan) to idle time
- If idle time < T, page is in working set

Unresolved Questions

How should value of T be configured?

- What if T is too large?

How should working set be defined when pages are shared?

- Put jobs sharing pages in same balance set

What processes should compose balance set?

How much memory needed for a "balanced system"?

- Balanced system: Each resource (e.g., CPU, memory, disk) becomes bottleneck at nearly same time
- How much memory is needed to keep the CPU busy?
- With working set approach, CPU may be idle even with runnable processes

VM Trends: Interaction with File I/O

Integrated VM and file buffer cache in physical memory

- Physical memory used for both VM and as cache for file I/O
- OS determines how much of physical memory to devote to each role

Can perform file I/O using VM system

- Memory-mapped files: (mmap() in UNIX)
- Programmer perspective: File lives in memory, access through pointer dereferences
- Advantages: Elegant, no overhead for system calls (no read() or write()), use madvise() for prefetching hints
- How?
  - Set up page tables to indicate each page of file initially not present
  - When process accesses page of file, causes page fault
  - OS fetches file page from disk, set present bit

Current Trends

VM code is not as critical

- Reason #1: Personal vs. time-shared machine
  - Why does this matter?
- Reason #2: Memory is more affordable, more memory

Less hardware support for replacement policies

- Software emulation of use and dirty bits

Larger page sizes

- Better TLB coverage
- Smaller page tables
- Disadvantage: More internal fragmentation
  - Multiple page sizes