Memory Management

- Swapping
- Demand Paging
- Page Replacement

Virtual Memory Management

- How does the OS “fit” multiple processes into the main memory?
  - E.g.: 20 processes with 512MB virtual memory each in 1GB of RAM?
- How does a single process fit in the main memory?
  - E.g.: 4GB virtual memory space of process in 1GB physical RAM?
- What happens to the memory of a process:
  - When it is not running on the CPU?
  - If it is not using parts of its memory actively?

Does the OS need to keep All Processes in Memory?

- Needs to keep:
  - Currently executing process
  - Other processes that are likely to execute in near future
- Some processes may be:
  - Inactive. E.g.: blocked on certain events
  - Low priority. E.g.: just executed, so may not be scheduled for long
**Swapping**

- Move inactive process to secondary storage
- Bring in active process when scheduled to run
- Swap space: Portion of hard disk devoted to swapping

![Swapping Diagram]

**Swapping: Issues**

- Performance: Swapping may be slow
  - Depends on disk speed, size of process’s memory
- Swapping out process blocked on I/O:
  - What if pending I/O needs to write to process’s memory?
- When to do swapping?
  - Always?
  - Only when system is low on memory?

**Does a Process Need all its Memory?**

- Several parts of program are never used during process execution:
  - Parts of code. E.g.: Special error-handling functions, control flow paths not taken
  - Parts of data. E.g.: statically allocated array elements
- Process uses only part of the memory *at a time*:
  - Current set of instructions
  - Portion of data, heap, and stack currently being used

**Demand Paging**

- Process keeps only part of its whole address space in RAM at a given time
  - Memory being used actively
- Bring in pages from secondary memory as needed
  - Move inactive pages to disk
- Combines swapping and paging
  - Pager: Swapper at page level
**Demand Paging: Basics**

- How to distinguish between pages in/out of memory?
  - Use Valid-invalid bit in page table
  - Invalid bit => page is invalid or in swap space
- Execution proceeds normally as long as process only accesses memory-resident pages
- What happens when a process accesses an address, but the page is on the disk?

**Page Faults**

- OS generates an exception called page fault
  - Finds the location of the page on the swap space
  - Allocates a physical frame
  - Copies the contents of the page from the swap space into the physical frame
  - Restarts the faulting instruction

**Demand Paging: Performance Overhead**

- Overhead of page fault:
  - Servicing the page fault interrupt
  - Reading in page from disk – slowest step (few msec)
  - Restarting the process
- Page fault rate (p):
  - Probability of page fault per memory access
- Effective memory access time (EAT):
  - $EAT = (1-p) \times ma + p \times pft$
  - Memory access time (ma): Fast (few nsec)
  - pft: Page fault time

**What happens at Process Start?**

- Pure demand paging: Bring in a page only when needed
  - Process faults for initial pages also
- Preparing: Pager brings in a few pages before process start
  - E.g.: code of main function, static data, part of stack, heap
- Copy-on-Write: Two processes share same pages after fork
  - Page is copied when one process tries to write
Handling Full Memory

- When a page fault occurs:
  - What if all physical frames were taken?
- Should the OS:
  - Terminate the faulting process?
  - Swap out another process completely?
  - Replace another page?

Page Replacement

- If no free frame, replace an existing page
- Victim frame: frame to be replaced
- Overhead:
  - Writing victim frame to disk
  - Reading new page from disk

Reducing Page Replacement Overhead

- Dirty (Modified) Bit: Bit maintained for each frame
  - Set by hardware upon page modification
  - Initially set to 0 when page read in
- What if we select a victim frame with dirty-bit=0?

Page Replacement Algorithms

- Question: How to Select A Victim Frame?
- Goal: Minimize the number of page faults
  - Performance depends on memory access pattern
- Reference string: String of memory references
  - Used to evaluate page replacement algorithms
  - Could be synthetic or trace-based
Page Replacement Algorithms

- FIFO
- Optimal
- LRU
- LRU-Approximation
- Counting-based

FIFO Page Replacement

- Replace the oldest page
- Benefit?
- Belady’s Anomaly:
  - Page fault rate *increases* by increasing number of frames

Optimal Page Replacement

- Replace the page that will not be used for the longest period of time
- "Oracle" algorithm: Perfect knowledge of future
  - Not practical
  - Mainly used as a theoretical lower bound

Least Recently Used (LRU)

- Replace the least recently used page
- Intuition: Use past to predict the future
  - If a page not used for long, unlikely to be used again in the near future
  - OPT in reverse time
**Stack Algorithms**

- Class of page replacement algorithms:
  - Set of resident pages for $n$ frames is always a subset of resident pages for $(n+1)$ frames
- Can a stack algorithm suffer from Belady’s anomaly?
- Is the following a stack algorithm?
  - FIFO
  - OPT
  - LRU

**LRU Implementation**

- Needed: Order pages by last access time
- Approaches:
  - Counters
  - Stack
  - LRU-Approximation

**LRU Implementation: Counters**

- Counter added to CPU: Incremented on each memory reference
- Time-of-use field with each page table entry: Copied from counter on memory access to page
- Problems?

**LRU Implementation: Stack**

- Stack of page numbers:
  - Page moved to top of stack on reference
  - Which page is at bottom of stack?
- Stack update on memory reference can be expensive. Why?
**LRU-Approximation Algorithms**

- **LRU approximation:**
  - Uses hardware support for efficiency
- **Reference bit:** Bit maintained for each page
  - Set by hardware upon page reference (read or write)
  - Initially set to 0 when page read in
- **LRU-approximation algorithms:** Use reference bit to emulate LRU

**Additional-Reference-Bits Algorithm**

- Record reference bit for each frame at regular intervals
- **n-bit vector:** reference history of page
  - MSB – most recent reference bit value
  - Can be treated as an unsigned integer
- Pages can be ordered by the integer values
  - Use FIFO between pages with same value

**Second Chance (Clock) Algorithm**

- Pages replaced in FIFO order
- If reference bit set for victim page:
  - give it second chance
  - Reset reference bit
- **Implementation:**
  - Circular queue
  - Pointer moves around the queue to find next victim

**Enhanced Second Chance**

- Consider Dirty bit also
- Order pages by (reference, dirty) pair
  - (0,0): neither recently used nor modified
  - (0,1): not recently used, but modified
  - (1,0): recently used, but not modified
  - (1,1): recently used and modified
- Which should be picked first/last?
Counting-based Page Replacement

- Based on count of page references
- Least Frequently Used (LFU):
  - Replace the page with least no. of references
  - Problem?
- Most Frequently Used (MFU):
  - Replace the page with highest no. of references
  - Problem?

Page Buffering

- Techniques used in addition to page replacement
  - Goal: Reduce page fault costs further
- Free frame buffer:
  - Page is brought into a free frame
  - Victim frame written out lazily, added to free frame buffer
  - Remember the old page in the free frame
- Dirty page list:
  - Write these out in background if swap device is idle