Section 8: Clock Algorithm, Second Chance List Algorithm, and Intro to I/O

CS 162

March 20, 2020

Contents

1 Vocabulary

- **Demand Paging** The process where the operating system only stores pages that are "in demand" in the main memory and stores the rest in persistent storage (disk). Accesses to pages not currently in memory page fault and the page fault handler will retrieve the request page from disk (paged in). When main memory is full, then as new pages are paged in old pages must be paged out through a process called eviction. Many cache eviction algorithms like least recently used can be applied to demand paging, the main memory is acting as the cache for pages which all start on disk.

- **Working Set** The subset of the address space that a process uses as it executes. Generally we can say that as the cache hit rate increases, more of the working set is being added to the cache.

- **Resident Set Size** The portion of memory occupied by a process that is held in main memory (RAM). The rest has been paged out onto disk through demand paging.

- **Thrashing** Phenomenon that occurs when a computer's virtual memory subsystem is constantly paging (exchanging data in memory for data on disk). This can lead to significant application slowdown.

- **Clock Algorithm**: An approximation of LRU. Main idea: replace *an* old page, not the *oldest* page. On a page fault, check the page currently pointed to by the 'clock hand. Checks a use bit which indicates whether a page has been used recently; clears it if it is set and advances the clock hand. Otherwise, if the use bit is 0, selects this candidate for replacement.

  Other bits used for Clock: "modified"/"dirty" indicates whether page must be written back to disk upon pageout; "valid" indicates whether the program is allowed to reference this page; "read-only"/"writable" indicates whether the program is allowed to modify this page.

- **Nth Chance Algorithm**: An approximation of LRU. A version of Clock Algorithm where each page gets N chances before being selected for replacement. The clock hand must sweep by N times without the page being used before the page is replaced. For a large N, this is a very good approximation of LRU.

- **Second-Chance List Algorithm**: An approximation of LRU. Divides pages into two - an active list and a second-chance list. The active list uses a replacement policy of FIFO, while the second-chance list uses a replacement policy of LRU. Not required reading, but if you're interested in the details, this algorithm is covered in detail in this paper: `https://users.soe.ucsc.edu/~sbrandt/221/Papers/Memory/levy-computer82.pdf`. The version presented in lecture and for the purposes of this course includes some significant simplifications.

- **Inverted Page Table** - The inverted page table scheme uses a page table that contains an entry for each phiscial frame, not for each logical page. This ensures that the table occupies a fixed fraction of memory. The size is proportional to physical memory, not the virtual address space. The inverted page table is a global structure – there is only one in the entire system. It stores reverse mappings for all processes. Each entry in the inverted table contains has a tag containing the task id and the virtual address for each page. These mappings are usually stored in associative memory (remember fully associative caches from 61C?). Associatively addressed memory compares input search data (tag) against a table of stored data, and returns the address of matching data. They can also use actual hash maps.

- **I/O** In the context of operating systems, input/output (I/O) consists of the processes by which the operating system receives and transmits data to connected devices.

- **Controller** The operating system performs the actual I/O operations by communicating with a device controller, which contains addressable memory and registers for communicating the the CPU, and an interface for communicating with the underlying hardware. Communication may be done via programmed I/O, transferring data through registers, or Direct Memory Access, which allows the controller to write directly to memory.
- **Interrupt** One method of notifying the operating system of a pending I/O operation is to send a interrupt, causing an interrupt handler for that event to be run. This requires a lot of overhead, but is suitable for handling sporadic, infrequent events.
- **Polling** Another method of notifying the operating system of a pending I/O operating is simply to have the operating system check regularly if there are any input events. This requires less overhead, and is suitable for regular events, such as mouse input.
- **Response Time** Response time measures the time between a requested I/O operating and its completion, and is an important metric for determining the performance of an I/O device.
- **Throughput** Another important metric is throughput, which measures the rate at which operations are performed over time.
- **Asynchronous I/O** For I/O operations, we can have the requesting process sleep until the operation is complete, or have the call return immediately and have the process continue execution and later notify the process when the operation is complete.
- **Memory-Mapped I/O** Memory-mapped I/O (not to be confused with memory-mapped file I/O) uses the same address bus to address both memory and I/O devices – the memory and registers of the I/O devices are mapped to (associated with) address values. So when an address is accessed by the CPU, it may refer to a portion of physical RAM, but it can also refer to memory of the I/O device. Thus, the CPU instructions used to access the memory can also be used for accessing devices.

2 Demand Paging

2.1 Demand Paging

An up-and-coming big data startup has just hired you do help design their new memory system for a byte-addressable system. Suppose the virtual and physical memory address space is 32 bits with a 4KB page size.

Suppose you know that there will only be 4 processes running at the same time, each with a Resident Set Size (RSS) of 512MB and a working set size of 256KB. What is the minimum amount of TLB entries that your system would need to support to be able to map/cache the working set size for one process? What happens if you have more entries? What about less?

Suppose you run some benchmarks on the system and you see that the system is utilizing over 99% of its paging disk IO capacity, but only 10% of its CPU. What is a combination of the of disk space and memory size that can cause this to occur? Assume you have TLB entries equal to the answer from the previous part.

Out of increasing the size of the TLB, adding more disk space, and adding more memory, which one would lead to the largest performance increase and why?

3 Clock Algorithm

3.1 Clock Page Table Entry

Suppose that we have a 32-bit virtual address split as follows:

10 Bits	10 Bits	12 Bits
Table ID	Page ID	Offset

Assume that the physical address is 32-bit as well. Show the format of a page table entry (PTE) complete with bits required to support the clock algorithm.

3.2 Clock Algorithm Step-through

For this problem, assume that physical memory can hold at most four pages. What pages remain in memory at the end of the following sequence of page table operations and what are the use bits set to for each of these pages?

Page	A	B	C	A	C	D	B	D	A	E	F

4 Second Chance List Algorithm

Suppose you have four pages of physical memory: 00, 01, 10, 11 and five pages of virtual memory: 000, 001, 010, 011, 101. Run the second chance list algorithm, with two physical pages delegated to the active list and two physical pages delegated to the second chance list.

The access pattern (assume we write to these pages) for virtual pages is as follows:

Page	000	001	010	011	000	101

The page table looks like this at the start of the algorithm.

Virtual Page	Physical Page	Extra
000		PAGEOUT
001		PAGEOUT
010		PAGEOUT
011		PAGEOUT
101		PAGEOUT

The 'extra' bits should read 'RW', 'INVALID', 'FIFO: 0', 'LRU: 0' where the bit for FIFO / LRU is 0 for more recent and 1 for less recent.

4.1 After Writes to '000', '001'

What does the table look like after these first two writes?

Virtual Page	Physical Page	Extra
000		
001		
010		
011		
101		

4.2 After Writes to ’010’, ’011’

What does the table look like after the next two writes?

Virtual Page	Physical Page	Extra
000		
001		
010		
011		
101		

4.3 After Write to ’000’

What happens when you write to virtual page 000? What does the table look like after this write?

Virtual Page	Physical Page	Extra
000		
001		
010		
011		
101		

4.4 After Write to ’101’

What happens when you write to virtual page 101? What does the table look like after this write?

Virtual Page	Physical Page	Extra
000		
001		
010		
011		
101		

5 Inverted Page Tables

5.1 Inverted Page Tables

Why IPTs? Consider the following case:
- 64-bit virtual address space
- 4 KB page size
- 512 MB physical memory

How much space (memory) needed for a single level page table? Hint: how many entries are there? 1 per virtual page. What is the size of a page table entry? access control bits + physical page #.

How about multi level page tables? Do they serve us any better here?

What is the number of levels needed to ensure that any page table requires only a single page (4 KB)?

Linear Inverted Page Table

What is the size of of the hashtable? What is the runtime of finding a particular entry?

Assume the following:
- 16 bits for process ID
- 52 bit virtual page number (same as calculated above)
- 12 bits of access information

Hashed Inverted Page Table

What is the size of of the hashtable? What is the runtime of finding a particular entry?

Assume the following:
- 16 bits for process ID
- 52 bit virtual page number (same as calculated above)
- 12 bits of access information

6 Input/Output

6.1 Warmup

1. (True/False) If a particular IO device implements a blocking interface, then you will need multiple threads to have concurrent operations which use that device.

2. (True/False) For I/O devices which receive new data very frequently, it is more efficient to interrupt the CPU than to have the CPU poll the device.

3. (True/False) With SSDs, writing data is straightforward and fast, whereas reading data is complex and slow.

4. (True/False) User applications have to deal with the notion of file blocks, whereas operating systems deal with the finer grained notion of disk sectors.

6.2 I/O Devices

What is a block device? What is a character device? Why might one interface be more appropriate than the other?

Why might you choose to use DMA instead of memory mapped I/O? Give a specific example where one is more appropriate than the other.

Explain what is meant by “top half” and “bottom half” in the context of device drivers.