Section 10: Paging, I/O

CS 162

July 27, 2020

Contents

1 Vocabulary

- **Clock** - 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** - 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** - 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.

- **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 Clock Algorithm

2.1 Clock Page Table Entry

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

10 Bits Table ID	10 Bits Page ID	12 Bits 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.

20 Bits PPN	8 Bits Other	1 Bit Dirty	1 Bit Use	1 Bit Writable	1 Bit Valid

2.2 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

E: 1, F: 1, C: 0, D: 0
Recall that the clock hand only advances on page faults. No page replacement occurs until $t = 10$, when all pages are full. At $t = 10$, all pages have the use bit set. The clock hand does a full sweep, setting all use bits to 0, and selects page 1 (currently holding A) to be paged out. The clock hand advances and now points to page 2 (currently holding B). At $t = 11$, we check page 2's use bit, and since it is not set, select page 2 to be paged out. F is brought in to page 2. The clock hand advances and now points to page 3. We reach the end of the input and end.
Note: The table shows the clock hand position before page faults occur.

Page	A	B	C	A	C	D	B	D	A	E	F
1	A: 1	A: 1	A: 1	A: 1	A: 1	A: 1	A: 1	A: 1	A: 1	E: 1	E: 1
2		B: 1	B: 1	B: 1	B: 1	B: 1	B: 1	B: 1	B: 1	B: 0	F: 1
3			C: 1	C: 1	C: 1	C: 1	C: 1	C: 1	C: 1	C: 0	C: 0
4						D: 1	D: 1	D: 1	D: 1	D: 0	D: 0
Clock	1	2	3	4	4	4	1	1	1	1	2

3 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.

3.1 After Writes to '000', '001'

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

Virtual Page	Physical Page	Extra
000	00	RW, FIFO: 1
001	01	RW, FIFO: 0
010		PAGEOUT
011		PAGEOUT
101		PAGEOUT

3.2 After Writes to '010', '011'

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

Virtual Page	Physical Page	Extra
000	00	INVALID, LRU: 1
001	01	INVALID, LRU: 0
010	10	RW, FIFO: 1
011	11	RW, FIFO: 0
101		PAGEOUT

3.3 After Write to '000'

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

Virtual page 000 is translated to physical page 00. However, this page is marked as invalid, so the page fault handler will be invoked. The handler will use the FIFO replacement algorithm on the active list to find a physical page to evict, and insert that page into the second-chance list. Then physical page 00 is inserted into the active list.

Virtual Page	Physical Page	Extra
000	00	RW, FIFO: 0
001	01	INVALID, LRU: 1
010	10	INVALID, LRU: 0
011	11	RW, FIFO: 1
101		PAGEOUT

3.4 After Write to '101'

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

Virtual page 101 does not have a physical page assigned, so this will be a page fault. The page fault handler will use the LRU replacement algorithm on the second chance list to find a physical page to evict, then use this physical page for 101.

Next, it will use the FIFO replacement algorithm on the active list to find a physical page to evict, and insert that page into the second-chance list.

In this case, virtual page 001 is paged out, and virtual page 101 now maps to physical page 01. Finally, physical page 01 is inserted into the newly empty slot in the active list.

Virtual Page	Physical Page	Extra
000	00	RW, FIFO: 1
001		PAGEOUT
010	10	INVALID, LRU: 1
011	11	INVALID, LRU: 0
101	01	RW, FIFO: 0

4 Input/Output

4.1 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.

   True. Only with non-blocking IO can you have concurrency without multiple threads.

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.

   False. It is more efficient to poll, since the CPU will get overwhelmed with interrupts.

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

   False, it is the opposite. SSD's have complex and slower writes because their memory can't be easily mutated.

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.

   False, blocks are also an OS concept and are not exposed to users.

4.2 I/O Devices

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

Both of these are types of interfaces to I/O devices. A block device accesses large chunks of data (called blocks) at a time. A character device accesses individual bytes at a time. A block device interface might be more appropriate for a hard drive, while a character device might be more appropriate for a keyboard or printer.

Describe the difference between port-mapped I/O and memory-mapped I/O.

Port-mapped I/O uses special IN and OUT instructions in the Intel x86 architecture, in a separate address space from main memory. On the other hand, memory-mapped I/O uses load and store instructions in physical address space.

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

The top half of a device driver is used by the kernel to start I/O operations. The bottom half of a device driver services interrupts produced by the device. You should know that Linux has different definitions for "top half" and "bottom half", which are essentially the reverse of these definitions (top half in Linux is the interrupt service routine, whereas the bottom half is the kernel-level bookkeeping).