Carleton UNIVERSITY

Sample Solution to Fall 2017 Final
EXAMINATION

DURATION: 3 HOURS **No. Of Students: 138**

Department Name & Course Number: Systems and Computer Engineering SYSC 4001A

Course Instructor: Thomas Kunz

AUTHORIZED MEMORANDA:
William Stallings, *Operating Systems: Internals and Design Principles*, 9th edition, Pearson 2018, ISBN-9780134670959 (as physical book, no ebook) or earlier versions of that same book. Also, a non-programmable calculator is allowed.

Students MUST count the number of pages in this examination question paper before beginning to write, and report any discrepancy to a proctor. This question paper has 7 pages + cover page = 8 pages in all.

This examination question paper MAY NOT be taken from the examination room.

In addition to this question paper, students require: an examination booklet: NO
Scantron Sheet: NO

Name: _____

Student Number: _____

Question 1: _____ /10

Question 2: _____ /20

Question 3: _____ /20

Question 4: _____ /20

Question 5: _____ /20

Total: _____/90

Exam questions will not be explained, and no hints will be given. If you think that something is unclear or ambiguous, make a reasonable assumption (one that does not contradict the question), write it at the start of the solution, and answer the question. Do not ask questions unless you believe you have found a mistake in the exam paper. If there is a mistake, the correction will be announced to the entire class. If there is no mistake, this will be confirmed, but no additional explanation of the question will be provided.

Question 1: Processes and Threats (10 marks)

1. Consider the following code segment:

```
pid t pid;
pid = fork();
if (pid == 0) { /* child process */
      fork();
      thread_create( . . .);
}
fork();
```

a. How many unique processes are created?
b. How many unique threads are (explicitly) created?

Answer: (6 marks)
There are six processes and two threads. If you only want to count the processes that are created by executing the code, it is 5 (plus the process that is invoked somehow). I allowed both answers.

2. Can a multithreaded solution using multiple user-level threads achieve better performance on a multiprocessor system than on a single processor system? Explain.

Answer: (2 marks)
A multithreaded system comprising of multiple user-level threads cannot make use of the different processors in a multiprocessor system simultaneously. The operating system sees only a single process and will not schedule the different threads of the process on separate processors. Consequently, there is no performance benefit associated with executing multiple user-level threads on a multiprocessor system.

3. Using Amdahl's Law, calculate the speedup gain of an application that has a 60 percent parallel component for
   a. two processing cores
   b. four processing cores.

Answer: (2 marks)
Two processing cores = 1.43 speedup; four processing cores = 1.82 speedup.

Question 2: Concurrency: Mutual Exclusion and Synchronization (20 marks)

1. Servers can be designed to limit the number of open connections. For example, a server may wish to have only N socket connections at any point in time. As soon as N connections are made, the server will not accept another incoming connection until an existing connection is released. Explain how semaphores can be used by a server to limit the number of concurrent connections.

   Answer (10 marks):

   A semaphore is initialized to the number of allowable open socket connections, N. When a connection is accepted, the `semWait()` method is called; when a connection is released, the `semSignal()` method is called. If the system reaches the number of allowable socket connections, subsequent calls to `semWait()` will block until an existing connection is terminated and the `semSignal()` method is invoked.

2. Consider a system consisting of processes P1, P2, ..., Pn, each of which has a unique priority number (denoted by `proc_number` in the code below). The code below claims to be a correct implementation of a monitor that allocates three identical line printers to these processes, using the priority numbers for deciding the order of allocation (processes with higher priority number get printers allocated first). Is this code correct? If so, briefly outline why. If not, state what is missing/wrong and suggest a fix.

```
monitor printers {
   int num_avail = 3;
   int waiting_processes[MAX_PROCS];
   int num_waiting = 0;
   cond c;

   void request_printer(int proc_number) {
      if (num_avail > 0) {
         num_avail--;
         return;
      }
      waiting_processes[num_waiting] = proc_number;
      num_waiting++;
      sort(waiting_processes); // sorts in decreasing order
      while (num_avail == 0)
         cwait(c);
      waiting_processes[0] = waiting_processes[num_waiting-1];
      num_waiting--;
      sort(waiting_processes);
      num_avail--;
   }

   void release_printer() {
      num_avail++;
      csignal(c);
   }
}
```

Answer (10 marks):
This was the one question very few solved (or even understood where the problem was)… Quite a few answers showed problems understanding the concept of a monitor.

The process woken up by the `csignal()` operation may not be the process with the highest priority. So a process needs to keep checking not only whether there is a printer available, but also whether the process is the currently highest priority process (is its `proc_number` the one at the top of the sorted list….)

```
      while (num_avail == 0) {
              cwait(c);
              if (proc_number != waiting_processes[0]) {
                    // implicitly, more than one process blocked
                    csignal(c); // wake up another process
                    cwait(c); // go back to sleep
              }
       }
```

Question 3. Virtual Memory Management (20 marks)

The following page table is for a system with 16-bit virtual and physical addresses and with 4,096-byte pages. The reference bit is set to 1 when the page has been referenced. Periodically, a thread zeroes out all values of the reference bit. A dash for a page frame indicates the page is not in memory. All numbers are provided in decimal.

Page	Page Frame	Reference Bit
0	9	0
1	1	0
2	14	0
3	10	0
4	–	0
5	13	0
6	8	0
7	15	0
8	–	0
9	0	0
10	5	0
11	4	0
12	–	0
13	–	0
14	3	0
15	2	0

1. Convert the following virtual addresses (in hexadecimal) to the equivalent physical addresses (in hexadecimal). Also set the reference bit for the appropriate entry in the page table.
   a. `0xE12C`
   b. `0x3A9D`
   c. `0xA9D9`
   d. `0x7001`
   e. `0xACA1`

   Answer (5 marks):
   The translated addresses in hexadecimal notation (the offset part is 12 bits, we only need to replace the first 4 bits with the frame number):
   a. `0xE12C→0x312C`
   b. `0x3A9D→0xAA9D`
   c. `0xA9D9→0x59D9`
   d. `0x7001→0xF001`
   e. `0xACA1→0x5CA1`

2. Using the above addresses as a guide, provide an example of a logical address (in hexadecimal) that would result in a page fault.

   Answer (2 marks):
   The only choices are pages 4, 8, 12, and 13. Thus, example addresses include anything that begins with the hexadecimal sequence 0x4..., 0x8..., 0xC..., and 0xD....

3. Assume that the page replacement algorithm is Clock. We have no information about the past order in which pages were brought into memory. Furthermore, assume none of the pages is locked and the replacement scope is local. If a page fault were to occur now, would all pages currently in main memory be potential candidates for replacement? If not, which pages would NOT be candidates?

   Answer (3 marks):
   Clock would look for pages with reference bit cleared, and only replace pages with reference bit set if there are no pages with a cleared reference bit. In the example, we started with all pages having their reference bit cleared, but 4 pages having their reference bit set again as a result of executing part 1 of this question (the question above said "If a page fault were to occur **NOW**" – admittedly most missed that aspect of the question). So as there are pages in the page table with reference bit cleared, only those are candidates for page replacement. So the pages that would NOT be considered are 2, 5, 10, and 15. Note that we also only consider pages that are in memory, so pages 4, 8, 12, and 13 are not candidates either really.

4. Consider the following page reference string: 7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0 , 1. Assuming demand paging with three frames, how many page faults (including the ones to bring in the initial pages to fill the frames) would occur for the following replacement algorithms?
   - LRU replacement
   - FIFO replacement
   - Optimal replacement

   Answer (6 marks):
   - 18
   - 17
   - 13

5. Assume that you have a page-reference string for a process with *m* frames (initially all empty). The page-reference string has length *p*; *n* distinct page numbers occur in it. Answer these questions for any page-replacement algorithms:
   - What is a lower bound on the number of page faults (including the initial page faults to fill the empty frames)?
   - What is an upper bound on the number of page faults (again including the initial page faults)?

   Answer (4 marks):
   - *Lower bound: n*
   - *Upper bound: p*

   Note that for both upper and lower bounds, tighter bounds can be given, once you start constraining the values of *p*, *n*, and *m*. For example, if $n <= m$, the upper bound on page faults will ben, irrespective of *p*. But the answers above are the most general cases (and even in the example, *p* is an upper bound, though not necessarily the tightest one, as $p >= n$ is always true).

Question 4. CPU Scheduling (20 marks)

1. Consider the exponential average formula used to predict the length of the next CPU burst. What are the implications of assigning the following values to the parameters used by the algorithm?
    a. $\alpha = 0$ and $S_1 = 100$ milliseconds
    b. $\alpha = 0.99$ and $S_1 = 10$ milliseconds

Answer (10 marks):

When $\alpha = 0$ and $S_1 = 100$ milliseconds, the formula always makes a prediction of 100 milliseconds for the next CPU burst. When $\alpha = 0.99$ and $S_1 = 10$ milliseconds, the most recent behavior of the process is given much higher weight than the past history associated with the process. Consequently, the scheduling algorithm is almost memoryless, and simply predicts the length of the previous burst for the next quantum of CPU execution.

2. Consider a system running ten I/O-bound tasks and one CPU-bound task. Assume that the I/O-bound tasks issue an I/O operation once for every millisecond of CPU computing and that each I/O operation takes 10 milliseconds to complete. Also assume that the context-switching overhead is 0.1millisecond and that all processes are long-running tasks. Describe is the CPU utilization for a round-robin scheduler when:
    a) The time quantum is 1 millisecond
    b) The time quantum is 10 milliseconds

Answer (10 marks):

a) The time quantum is 1millisecond: Irrespective of which process is scheduled, the scheduler incurs a 0.1 millisecond context-switching cost for every context-switch. This results in a CPU utilization of 1/1.1 * 100 = **91%.**

b) The time quantum is 10 milliseconds: The I/O-bound tasks incur a context switch after using up only 1 millisecond of the time quantum. The time required to cycle through all the processes is therefore 10*1.1 + 10.1 (as each I/O-bound task executes for 1 millisecond and then incur the context switch task, whereas the CPU-bound task executes for 10 milliseconds before incurring a context switch). The CPU utilization is therefore 20/21.1 * 100 = **94%**. As expected though, a larger time quantum leads to higher CPU utilization as there are, relatively speaking, fewer context switches. It may also lead to lower responsiveness, but the question concerns itself only with CPU utilization…

The question asked only to DISCUSS CPU utilization, not necessarily calculate it. So answers that ball-parked the CPU utilization and compared the expected values for the two quanta given are okay as well. The key insight here is that with a larger quanta, at the very least the CPU-bound job will experience fewer context switches, resulting in an overall higher CPU utilization. Of course that may come at the cost of lower response time or decreased fairness, but the question did not ask about these aspects.

Question 5. I/O Devices and File Systems (20 marks)

1. Suppose that a disk drive has 5,000 tracks, numbered 0 to 4999. The drive is currently serving a request at track 2150, and the previous request was at track 1805. The queue of pending requests, in FIFO order, is:

   `2069, 1212, 2296, 2800, 544, 1618, 356, 1523, 4965, 3681`

   Starting from the current head position, what is the total distance (in tracks) that the disk arm moves to satisfy all the pending requests for each of the following disk-scheduling algorithms?
   - FCFS
   - SSTF
   - SCAN
   - C-SCAN

   Answer (12 marks):
   - The FCFS schedule is 2150, 2069, 1212, 2296, 2800, 544, 1618, 356, 1523, 4965, 3681. The total seek distance is 13,011.
   - The SSTF schedule is 2150, 2069, 2296, 2800, 3681, 4965, 1618, 1523, 1212, 544, 356. The total seek distance is 7586.
   - The SCAN schedule is 2150, 2296, 2800, 3681, 4965, 2069, 1618, 1523, 1212, 544, 356. The total seek distance is 7424.
   - The C-SCAN schedule is 2150, 2296, 2800, 3681, 4965, 356, 544, 1212, 1523, 1618, 2069. The total seek distance is 9137.

2. Requests are not usually uniformly distributed. For example, we can expect a track containing the file-system metadata to be accessed more frequently than a track containing only files. Suppose you know that 50 percent of the requests are for a small, fixed number of track.
   - Would any of the scheduling algorithms discussed in class be particularly good for this case? Explain your answer.
   - Propose a disk-scheduling algorithm that gives even better performance by taking advantage of this "hot spot" on the disk.

   Answer (8 marks):
   - SSTF would take greatest advantage of the situation. FCFS could cause unnecessary head movement if references to the "high demand" cylinders were interspersed with references to cylinders far away.
   - Here are some ideas. Place the hot data near the middle of the disk. Modify SSTF to prevent starvation. Add the policy that if the disk becomes idle for more than, say, 50 ms, the operating system generates an anticipatory seek to the hot region, since the next request is more likely to be there.