Carleton UNIVERSITY

Sample Solution to Final
EXAMINATION
Fall 2018

DURATION: 3 HOURS **No. Of Students: 28**

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.

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 10 pages + cover page = 11 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: _____ /10

Question 3: _____ /10

Question 4: _____ /10

Question 5: _____ /10

Question 6: _____ /10

Total: _____/60

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. Using the program below, identify the values of `pid` at lines `A, B, C,` and `D`. (Assume that the actual pids of the parent and child are 2600 and 2603, respectively.)

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
    pid_t pid, pidl;
    /* fork a child process */
    pid = fork() ;
    if (pid < 0) { /* error occurred */
       fprintf (stderr, "Fork Failed");
       return 1;
    }
    else if (pid == 0) { /* child process */
       pid1 = getpid();
       printf("child: pid = %d", pid); /* A */
       printf("child: pid1 = %d", pid1); /* B */
    }
    else { /* parent process */
       pid1 = getpid();
       printf("parent: pid = %d",pid); /* C */
       printf("parent: pid1 = %d",pid1); /* D */
       wait(NULL);
    }
    return 0;
}
```

Answer (4 marks):
A= 0, B = 2603, C = 2603, D = 2600

2. Is it possible to have concurrency but not parallelism? Explain.

   Answer (2 marks):
   Yes. Concurrency means that more than one process or thread is progressing at the same time. However, it does not imply that the processes are running simultaneously. The scheduling of tasks allows for concurrency, but parallelism is supported only on systems with more than one processing core.

3. A system with two dual-core processors has four processors available for scheduling. A CPU-intensive application is running on this system. All input is performed at program start-up, when a single file must be opened. Similarly, all output is performed just before the program terminates, when the program results must be written to a single file. Between startup and termination, the program is entirely CPU-bound. Your task is to improve the performance of this application by multithreading it. The application runs on a system that uses the one-to-one threading model (each user thread maps to a kernel thread).
   - How many threads will you create to perform the input and output? Explain.
   - How many threads will you create for the CPU-intensive portion of the application? Explain.

   Answer (4 marks):
   - It only makes sense to create as many threads as there are blocking system calls, as the threads will be spent blocking. Creating additional threads provides no benefit. Thus, it makes sense to create a single thread for input and a single thread for output.
   - Four. There should be as many threads as there are processing cores. Fewer would be a waste of processing resources, and any number > 4 would be unable to run.

Question 2: Concurrency (10 marks)

1. A race condition ____.
   A) results when several threads try to access the same data concurrently
   B) results when several threads try to access and modify the same data concurrently
   C) will result only if the outcome does not depend on the order in which instructions are executed
   D) none of the above

Answer (1 mark): B

2. An instruction that executes atomically ____.
   A) must consist of only one machine instruction
   B) executes as a single, uninterruptible unit
   C) cannot be used to solve the critical section problem
   D) all of the above

Answer (1 mark): B

3. A counting semaphore ____.
   A) is essentially an integer variable
   B) is accessed through only one standard operation
   C) can be modified simultaneously by multiple threads
   D) cannot be used to control access to a thread's critical sections

Answer (1 mark): A

4. In Peterson's solution, the ____ variable indicates if a process is ready to enter its critical section.
   A) turn
   B) lock
   C) flag[i]
   D) turn[i]

Answer (1 mark): C

5. The first readers-writers problem ____.
   A) requires that, once a writer is ready, that writer performs its write as soon as possible
   B) is not used to test synchronization primitives
   C) requires that no reader will be kept waiting unless a writer has already obtained permission to use the shared database
   D) requires that no reader will be kept waiting unless a reader has already obtained permission to use the shared database

Answer (1 mark): C

6. A ___ type presents a set of programmer-defined operations that are provided mutual exclusion within it.
   A) transaction
   B) signal
   C) binary
   D) monitor

Answer (1 mark): D

7. _____ occurs when a higher-priority process needs to access a data structure that is currently being accessed by a lower-priority process.
   A) Priority inversion
   B) Deadlock
   C) A race condition
   D) A critical section

Answer (1 mark): A

8. A solution to the critical section problem does not have to satisfy which of the following requirements?
   A) mutual exclusion
   B) progress
   C) atomicity
   D) bounded waiting

Answer (1 mark): C

9. A(n) _____ refers to where a process is accessing/updating shared data.
   A) critical section
   B) entry section
   C) mutex
   D) test-and-set

Answer (1 mark): A

10. How many philosophers may eat simultaneously in the Dining Philosophers problem with 5 philosophers?
   A) 1
   B) 2
   C) 3
   D) 5

Answer (1 mark): B

Question 3. Memory Management (10 marks)

1. Consider a pure paging system that uses three levels of page tables and 64-bit addresses. Each virtual address is the ordered set $v = (p, m, t, d)$, where the ordered triple (p, m, t) is the page number and d is the displacement into the page. Each page table entry is 64 bits (8 bytes). The number of bits that store p is np, the number of bits that store m is nm and the number of bits to store t is nt.

   - Assume $np = nm = nt = 18$.
     i. How large is the table at each level of the multilevel page table?

        Answer (1 mark):
        Each table can refer to 2^{18} entries. Each page table entry is 8 bytes, meaning that each table is $2^{18} \times 8$ bytes, or 2MB.

     ii. What is the page size, in bytes?

        Answer (1 mark):
        Each address uses 3×18 bits to specify the page number, leaving 10 bits for the displacement. Each page therefore is 2^{10} bytes, or 1KB.

   - Assume $np = nm = nt = 14$.
     i. How large is the table at each level of the multilevel page table?

        Answer (1 mark):
        Each table can refer to 2^{14} entries. Each page table entry is 8 bytes, meaning that each table is $2^{14} \times 8$ bytes, or 128KB.

     ii. What is the page size, in bytes?

        Answer (1 mark):
        Each address uses 3×14 bits to specify the page number, leaving 22 bits for the displacement. Each page therefore is 2^{22} bytes, or 4MB.

2. A simplified view of thread states is Ready, Running, and Blocked, where a thread is either ready and waiting to be scheduled, is running on the processor, or is blocked (for example, waiting for I/O). Assuming a thread is in the Running state, answer the following questions, and explain your answer:
   i. Will the thread change state if it incurs a page fault? If so, to what new state?
   ii. Will the thread change state if it generates a TLBmiss that is resolved in the page table? If so, to what new state?
   iii. Will the thread change state if an address reference is resolved in the page table? If so, to what new state?

   Answer (3 marks):
   i. On a page fault the thread state is set to blocked as an I/O operation is required to bring the new page into memory.
   ii. On a TLB-miss, the thread continues running if the address is resolved in the page table.
   iii. The thread will continue running if the address is resolved in the page table.

3. Assume we have a demand-paged memory. The page table is held in registers. It takes 8 milliseconds to service a page fault if an empty page is available or the replaced page is not modified, and 20 milliseconds if the replaced page is modified. Memory access time is 100 nanoseconds. Assume that the page to be replaced is modified 70 percent of the time. What is the maximum acceptable page-fault rate for an effective access time of no more than 200 nanoseconds?

   Answer (3 marks):
   0.2 μsec = (1 − P) × 0.1 μsec + (0.3P) × 8 millisec + (0.7P) × 20 millisec
   0.2 μsec = (1 − P) × 0.1 μsec + (0.3P) × 8000 μsec + (0.7P) × 20000 μsec
   0.2 μsec = 0.1 μsec – 0.1 P μsec + 2400 P μsec + 14000 P μsec
   0.1 μsec = −0.1P μsec + 16400 P μsec
   0.1 μsec ~16,400 P μsec

   P ~ 0.0000061 (or $6.1 * 10^{-6}$)

   Note: without calculator, I wasn't expecting you to come up with the exact numerical answer. Any answer that formulated the basic equation correctly was accepted. Also, if you started out by saying that the 100 nanosecond access time is paid in any case, that is fine as well (makes little difference in the final answer). In that case, the initial equation is:

   0.2 μsec = 0.1 μsec + (0.3P) × 8 millisec + (0.7P) × 20 millisec

Question 4. CPU Scheduling (10 marks)

1. ____ is the number of processes that are completed per time unit.
   A) CPU utilization
   B) Response time
   C) Turnaround time
   D) Throughput

Answer (1 mark): D

2. ____ scheduling is approximated by predicting the next CPU burst with an exponential average of the measured lengths of previous CPU bursts.
   A) Multilevel queue
   B) RR
   C) FCFS
   D) SJF

Answer (1 mark): D

3. The ____ scheduling algorithm is designed especially for time-sharing systems.
   A) SJF
   B) FCFS
   C) RR
   D) Multilevel queue

Answer (1 mark): C

4. Which of the following scheduling algorithms must be non-preemptive?
   A) SJF
   B) RR
   C) FCFS
   D) priority algorithms

Answer (1 mark): C

5. A significant problem with priority scheduling algorithms is _____.
   A) complexity
   B) starvation
   C) determining the length of the next CPU burst
   D) determining the length of the time quantum

Answer (1 mark): B

6. The rate of a periodic task in a hard real-time system is ____, where p is a period and t is the processing time.
   A) 1/p
   B) p/t
   C) 1/t
   D) pt

Answer (1 mark): A

7. Which of the following is true of the rate-monotonic scheduling algorithm?
   A) The task with the shortest period will have the lowest priority.
   B) It uses a dynamic priority policy.
   C) CPU utilization is bounded when using this algorithm.
   D) It is non-preemptive.

Answer (1 mark): C

8. Which of the following is true of earliest-deadline-first (EDF) scheduling algorithm?
   A) When a process becomes runnable, it must announce its deadline requirements to the system.
   B) Deadlines are assigned as following: the earlier the deadline, the lower the priority; the later the deadline, the higher the priority.
   C) Priorities are fixed; that is, they cannot be adjusted when a new process starts running.
   D) It assigns priorities statically according to deadline.

Answer (1 mark): A

9. What effect does the size of the time quantum have on the performance of an RR algorithm? Explain!

Answer (1 mark): At one extreme, if the time quantum is extremely large, the RR policy is the same as the FCFS policy. If the time quantum is extremely small, the RR approach is called processor sharing and creates the appearance that each of n processes has its own processor running at 1/n the speed of the real processor.

10. What are the advantages of the EDF scheduling algorithm over the rate-monotonic scheduling algorithm? Explain!

Answer (1 mark): Unlike the rate-monotonic algorithm, EDF scheduling does not require that processes be periodic, nor must a process require a constant amount of CPU time per burst. The appeal of EDF scheduling is that it is theoretically optimal. Theoretically, it can schedule processes so that each process can meet its deadline requirements, and CPU utilization will be 100 percent.

Note: RMS only accepts a new job IF the inequality in the textbook holds. So RMS DOES guarantee, for all admitted jobs, that they meet their deadline. Also, RMS uses the CPU utilization of a submitted job, so long-running jobs may well get accepted if their task periodicity is low, resulting in a low CPU utilization. So it DOES NOT, per se, favor short jobs.

Question 5. Disk I/O (10 marks)

1. Would disk scheduling be useful for a single-user, dedicated disk in a sequential file processing application? Why?

   Answer (2 marks):
   Probably not, because only one request is generated at a time and because each request is extremely likely to be for the same or a cylinder adjacent to the last one requested.

2. In what circumstances might disk scheduling actually result in poorer performance than FCFS?

   Answer (2 marks):
   It is entirely possible that in some situations, requests to be serviced FCFS actually come in the same order as a more sophisticated scheduling algorithm might service them. The point is simply that there is overhead associated with each disk scheduling discipline, and scheduling is worthwhile only if the overhead is outweighed by the performance improvement.

3. The VSCAN scheduling strategy combines SSTF and SCAN according to a variable *R*. VSCAN determines the next request to service using SSTF, but each request to a cylinder that requires the read/write head to change direction is weighted by a factor *R*. The "distances" to requests in the opposite direction are computed as follows. If *C* is the number of cylinders on the disk and *S* is the number of cylinders between the current location of the head and the next request, then VSCAN uses the distance S + R * C. Thus, VSCAN uses *R* to add an extra cost to the requests in the non-preferred direction.

   - If $R = 0$, to which disk scheduling policy does VSCAN degenerate?

     Answer (2 marks):
     In this case, VSCAN is simply SSTF, because requests in both directions are treated equally.

   - If $R = 1$, to which disk scheduling policy does VSCAN degenerate?

     Answer (2 marks):
     When *R = 1*, VSCAN becomes C-SCAN, because any request in the non-preferred direction will have a cost equal to the number of tracks/cylinders in the disk plus the actual distance to the request. Thus, any request in the preferred direction will have a smaller cost than any request in the non-preferred direction.

   - Discuss the benefit of choosing a value of *R* somewhere between 0 and 1.

     Answer (2 marks):
     Tuning the value of *R* enables a system administrator to balance high throughput ($R = 0$) with small variance of response times ($R = 1$). The developers of VSCAN showed that setting $R = 0.2$ often provides the best trade-off between high throughput and small variance of response times.

Question 6. File Systems (10 marks)

1. Consider a file system on a disk that has a block size of 512 bytes. Assume that the information about each file (i.e., the directory entry) is already in memory. For each of the three allocation strategies (contiguous, linked, and indexed), answer these questions:
   - How is the logical-to-physical address mapping accomplished in this system? That is, how do we determine, for a logical address L, in which block X it is stored and what the offset into that block is to read the byte. For the indexed allocation, assume that a file is always less than 512 blocks long. For the linked allocation, assume that the links take up 4 bytes of space and are stored at the beginning of each block.
   - If we are currently at logical block 10 (the last block accessed was block 10) and want to access logical block 4, how many physical blocks must be read from the disk?

   Answer (6 marks):
   Let Z be the starting file address (block number).
   - Contiguous. Divide the logical address L by 512 with X and Y the resulting quotient and remainder respectively.
     - Add X to Z to obtain the physical block number. Y is the displacement into that block.
     - 1
   - Linked. Divide the logical physical address by 508 with X and Y the resulting quotient and remainder respectively (we need to account for the space taken up by the link)
     - Chase down the linked list (getting X+1 blocks). Y + 4 is the displacement into the last physical block.
     - 4 (we can't go backward, so we start at the beginning of the file)
   - Indexed. Divide the logical address by 512 with X and Y the resulting quotient and remainder respectively.
     - Get the index block into memory. Physical block address is contained in the index block at location X. Y is the displacement into the desired physical block.
     - 2

2. Consider a system where free space is kept in a free-space list. Suppose that the pointer to the free-space list is lost. Can the system reconstruct the free-space list? Explain your answer.

   Answer (4 marks):
   In order to reconstruct the free list, it would be necessary to perform "garbage collection." This would entail searching the entire directory structure to determine which blocks are already allocated to directories and files. Those remaining unallocated blocks could be relinked as the free-space list.