Fall 2014 COMP 3511 Operating Systems

Midterm Examination Solutions

Date: October 23, 2014 (Thursday) Time: 20:00 - 21:30

Name: _____ Student ID: _____

Email: _____ Lecture Section: _____

Note: Please write your name, student ID, and email address on this page. Read the following instructions carefully.

1. This is a **CLOSED** book exam!
2. This examination paper consists of 4 questions and 10 pages (including this page).
3. You have 90 minutes to complete this exam.
4. Answer all questions within the space provided on the examination paper. You may use back of the pages for your rough work. Please be concise! This is NOT an essay contest.
5. Please read each question very carefully and answer the question clearly to the point.
6. Make sure that your answers are neatly written and legible.
7. Show all the steps used in deriving your answer, wherever appropriate.

Question	Points	Score
1	10	
2	20	
3	40	
4	30	
Total	100	

1. (10 points) Please answer the following questions in a few sentences

   1.1. What are the main differences between a symmetric and asymmetric multiprocessor systems? (3 points)

   Answer: Symmetric multiprocessor systems treat all processors the same, where I/O and computation can be done by any processor (CPU) (1 point). Asymmetric multiprocessor systems have one master processor and multiple slave processors (1 point). Master processor distributes tasks among the slaves, and I/O operations are usually only done by master processors (1 point).

   1.2. Please name three advantages and a disadvantage of a microkernel OS approach? (4 points)

   Answer: (a) adding a new service does not require modifying the kernel, (b) it is more secure as more operations are done in user mode than in kernel mode, (c) a simpler kernel design and functionality results in a more reliable operating system (3 points). User programs and system services interact in a microkernel architecture by using inter-process communication mechanisms such as messaging, which add overheads or turns to be slow.

   1.3. Direct memory access or DMA is often used by high-speed I/O devices to relieve CPU from data transfer. What is the CPU involvement in DMA? Does this affect the memory access speed by CPU? (3 points)

   Answer: CPU initiates the DMA operation (by specifying the instructions into the DMA controller registers), and CPU has to react to the interrupt when I/O operation is completed. (2 points). Yes, it affects the CPU access speed to memory since it has to compete with DMA enabled devices (1 point).

2. (20 points) Process and thread

   2.1. What are one-to-one model and many-to-many model respectively? What is the advantage and disadvantage of one-to-one model comparing with many-to-many model? (5 points)

   Answer: One-to-one model allows a user thread to be mapped into a unique kernel thread, while many-to-many model allows many user threads to be mapped to a smaller or equal number of kernel threads. (2 points) One-to-one model provides the maximum concurrency by allowing each thread to run when another thread is blocked, but the creation of each user thread requires the creation of a corresponding kernel thread, which can be resource-consuming. (3 points)

   2.2. What is the main difference between concurrency and parallelism? (5 points)

   Answer: A parallel system can perform more than one task simultaneously (2 point). A concurrent system supports more than one task by allowing multiple tasks to make progress interleaved with one another. In another word, multiple tasks are multiplexed onto a CPU over the time (3 points)

2.3. Consider the following code segments, what is the total number of processes? Please elaborate (5 points)

```
pid_t pid;

pid = fork();

if (pid >0) {

   fork();

   fork();

}
```

Answer: There are total 5 processes (2 points). The first fork() creates one child (1), the second fork() creates one child (2), and the third fork() run by two processes, original process (0) and child process (2), creates two more child (3 and 4). (3 points)

2.4. Consider the following code segments, what is the total number of processes? Please elaborate (5 points)

```
pid_t pid;

pid = fork();

if(pid == 0){

    for (int i = 0; i < 10; i++)

         if (fork())

             fork();

}

fork();
```

Answer: There are total $(3^{10}+1)*2$ processes (2 points). The first fork() creates one child (1). The second fork(), which is run by the child process (1), generates 3^{10} child processes (2). The third fork() is run by the original process (0) and the 310 child processes (2). Thus in the end there will be $(3^{10}+1)*2$ processes (3 points).

3. (40 points) Scheduling

3.1. What are the differences between non-preemptive scheduling algorithms and preemptive scheduling algorithms? (4 points)

Answer: A non-preemptive scheduling algorithm is evoked only when the current process running on the CPU gives up the CPU voluntarily either due to the termination of the process or block for I/O. Otherwise, the scheduling is preemptive in nature. For instance, scheduling might occur when a new process with high priority joins the ready queue.

3.2. Shortest remaining time first (SRTF, i.e., preemptive shortest job first) scheduling is optimal with respect to the minimal average waiting time. Is this algorithm commonly used in practice? Please briefly justify your answer (3 points)

Answer: This is not commonly used, because it requires the future knowledge of the next CPU burst length of processes (2 points). Even with estimation, this is often not accurate (1 point)

3.3. Please explain the problem encountered in the following code. Can you please think of a way to overcome this problem? (7 points)
Consider a system consisting of two processes, P_0 and P_1, each accessing two semaphores, S and Q, set to the value 1.

P_0	P_1

wait(S);
wait (Q);
…
signal(S);
signal(Q);
	wait(Q);
	wait(S);
	…
	signal(Q);
	signal(S);

Answer: If P0 doe wait(S) and P1 does wait(Q), this results in a deadlock (4 points). One solution is to ensure that a process (either P0 or P1) has to acquire both locks (S and Q) or no lock. In another word, it excludes a process from holding one lock while waiting for the other. (3 points)

3.4. (16 points) Given the following set of processes, with arrival time and length of the CPU-burst time given in milliseconds:

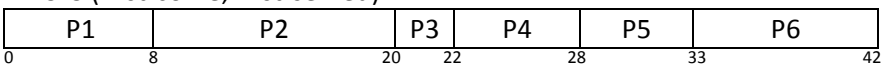
Process	P1	P2	P3	P4	P5	P6
CPU-burst	8	12	2	6	5	9
Arrival Time	0	4	5	9	13	17

For each of the following scheduling algorithms, construct the *Gantt charts* depicting the sequence of process execution and calculate the *average turnaround time*.

a) FCFS (first-come, first-served)
b) SJF (shortest job first) non-preemptive
c) RR (round robin) with quantum of 6 milliseconds
d) SRTF (shortest remaining time first)

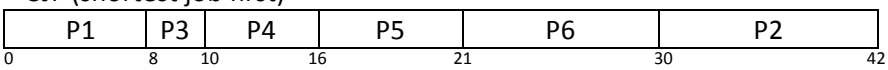
Answer:

a) FCFS (first-come, first-served)



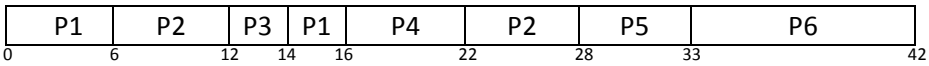
Turnaround time:
P1: 8 – 0 = 8, P2: 20 – 4 = 16, P3: 22 – 5 = 17, P4: 28 – 9 = 19, P5: 33 – 13 = 20,
P6: 42 – 17 = 25
Average turnaround time:
(8 + 16 + 17 + 19 + 20 + 25) / 6 = 17.5 milliseconds.

b) SJF (shortest job first)



Turnaround time:
P1: 8 – 0 = 8, P2: 42 – 4 = 38, P3: 10 – 5 = 5, P4: 16 – 9 = 7, P5: 21 – 13 = 8,
P6: 30 – 17 = 13
Average turnaround time:
(8 + 38 + 5 + 7 + 8 + 13) / 6 = 13.167 milliseconds.

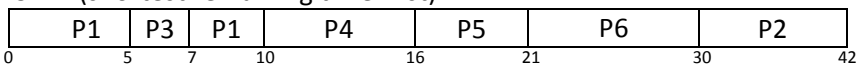
c) RR with quantum of 6 milliseconds (round robin)



Turnaround time:
P1: 16 – 0 = 16, P2: 28 – 4 = 24, P3: 14 – 5 = 9, P4: 22 – 9 = 13, P5: 33 – 13 = 20,

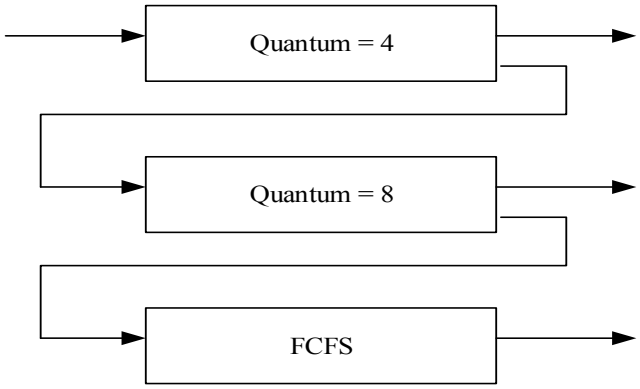
P6: 42 – 17 = 25
Average turnaround time:
(16 + 24 + 9 + 13 + 20 + 25) / 6 = 17.833 milliseconds.

d) SRTF (shortest remaining time first)



Turnaround time:
P1: 10 – 0 = 10, P2: 42 – 4 = 38, P3: 7 – 5 = 2, P4: 16 – 9 = 7, P5: 21 – 13 = 8,
P6: 30 – 17 = 13
Average turnaround time:
(10 + 38 + 2 + 7 + 8 + 13) / 6 = 13 milliseconds.

3.5. (10 point) There is a 3-level feedback queue with Round-Robin Scheduling used for the two high-priority queues and FCFS used for the lowest priority queue, as shown in figure below. The scheduling is **preemptive**.

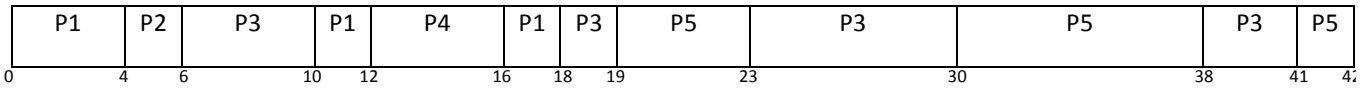


The process below is assumed to arrive in the order P1, P2, P3, P4 and P5. The CPU burst-time and arrival time of each process are listed below, given in milliseconds.

Process	P1	P2	P3	P4	P5
Burst Time	8	2	15	4	13
Arrival Time	0	2	6	12	19

Draw a *Gantt chart* depicting the process scheduling sequence (5 points) and calculate the *average waiting time* (5 points).

Answer:



Waiting time for each process:

P1: 6 + 4 = 10, P2: 2, P3: 8 + 4 + 8 = 20, P4: 0, P5: 7 + 3 = 10

Average waiting time: (10 + 2 + 20 + 0 + 10) / 5 = 8.4 milliseconds.

4. (30 points) Synchronization

4.1. What is a Race Condition? (3 points)

Answer: Several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place.

4.2. What is priority inversion? How this can be overcome? (4 points)

Answer: In scheduling, when a lower-priority process (L) holds a lock needed by a high-priority process (H), and the lower-priority process can be possibly preempted by another higher-priority process (L<M<H), which will affect the waiting time of process H. (2 points) This can be solved via a priority-inheritance protocol in that all processes that are accessing resources needed by a high-priority process inherits the higher priority until they are completed, so they cannot be preempted by any process with a priority M<H.

4.3. What are the two atomic operations on a semaphore? What is a counting semaphore?

Can you give an example when a counting semaphore is useful? (5 points)

Answer: wait() and signal() (2 points). A counting semaphore takes the value of any integer (1 point). A counting semaphore with initial value set = m can allow at most m processes to access shared dataset, (1 point). If semaphore has a negative value, the absolute value indicates the number of processes waiting to access the shared dataset (1 point)

4.4. (10 points) The first Readers-Writers Problem code

```
//The structure of a writer process:
do{
   wait(rw mutex);
   ...
   /*writing is performed*/
   ...
   signal(rw mutex);
} while (true);
```

```
//The structure of a reader process:
do {
    wait(mutex);
    read count++;
    if (read count == 1)
        wait(rw mutex);
    signal(mutex);
    ...
   /* reading is performed */
    ...
    wait(mutex);
    read count--;
    if (read count == 0)
       signal(rw_mutex);
    signal(mutex);
}while (true);
```

a) Why do we need rw_mutex? Please be specific (2 points)

Answer: to ensure mutual exclusion (1 point), so at most one writer or multiple readers can access the critical section (1 point)

b) Why is mutex useful? (2 points)

Answer: to ensure the mutual exclusion when manipulating the shared variable count.

c) If there are readers inside, and there are multiple writers waiting, which semaphore are the writers waiting for? (2 points)

Answer: All writers are waiting on semaphore rw-mutex

d) If there is a writer inside, and there are multiple readers waiting, which semaphore are the readers waiting for? (2 points)

Answer: The first reader waits on rw_mutex, and the remaining readers wait on mutex.

e) What is the potential problem for this solution? Please justify your answer (2 points)

Answer: This can cause starvation for writers (1 point). If there is always reader(s) in side (1 point).

4.5. (8 points) We have defined two atomic operations, wait() and signal() on semaphores shown below to guarantee mutual exclusion. Can you please illustrate why mutual exclusion cannot be ensured if the operations are not atomic.

```
// The definition of wait()
wait (S) {
      while (S <= 0)
            ; // busy wait
      S--;
}
```

```
// The definition of signal()
signal (S) {
      S++;
 }
```

Answer: If these operations are not atomic, there is a race condition (2 points) on the value S.

For example, suppose register1 and register2 are local CPU registers,

In wait(), S-- could be implemented as:

register1 = S; register1 = register1 - 1 ; S = register1;

In signal(), S++ could be implemented as:

register2 = S; register2 = register2 + 1 ; S = register2;

The concurrent execution of "S--" and "S++" is equivalent to a sequential execution in which the lower-level statements presented previously are interleaved in some arbitrary order.

Consider this execution interleaving with "S = 5" initially:

T0: signal() execute register1 = S {register1 = 5}

T1: signal() execute register1 = register1 + 1 {register1 = 6}

T2: wait() execute register2 = S {register2 = 5}

T3: wait() execute register2 = register2 – 1 {register2 = 4}

T4: signal() execute S= register1 {S = 6 }

T5: wait() execute S = register2 {S = 4} (Similar example: 4 points)

We will arrive at the incorrect state in this situation, because we allow several processes to manipulate the variable concurrently (2 points).