Fall 2015 COMP 3511 Operating Systems

Midterm Examination Solutions

Date: October 23, 2015 (Friday) Time: 19:30 - 21:00

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 5 questions and 9 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	10	
3	15	
4	40	
5	25	
Total	100	

1. [10 points] Multiple Choices (Please circle your answer)

1) _____ provide(s) an interface to the services provided by an operating system.
A) Shared memory
B) System calls
C) Simulators
D) Communication
Answer: B

2) The major difficulty in designing a layered operating system approach is ____.
A) appropriately defining the various layers
B) making sure that each layer hides certain data structures, hardware, and operations from higher-level layers
C) debugging a particular layer
D) making sure each layer is easily converted to modules
Answer: A

3) _____ allows a thread to run on only one processor.
A) Processor affinity
B) Processor set
C) NUMA
D) Load balancing
Answer: A

4) Cancellation points are associated with ____ cancellation.
A) asynchronous
B) deferred
C) synchronous
D) non-deferred
Answer: B

5) In Little's formula, λ represents the ____.
A) average waiting time in the queue
B) average arrival rate for new processes in the queue
C) average queue length
D) average CPU utilization
Answer: B

6) _____ involves the decision of which kernel thread to schedule onto which CPU.
A) Process-contention scope
B) System-contention scope
C) Dispatcher
D) Round-robin scheduling
Answer: B

7) 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 of execution does not depend on the order in which instructions are executed
D) None of the above
Answer: B

8) 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: A

9) 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: C

10) _____ 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: A

2. [10 points] Please answer the following questions in a few sentences.

2.1. One of the operating system goals is to manage system resources. Can you please name two hardware resources, and two software resources, respectively. (2 points)

Answer: CPU and memory, files and semaphores (0.5 point each)

2.2. What is the uniqueness in a multicore system comparing with a general multiprocessor system? What are the two main advantages of a multicore system? (4 points)

Answer: It implements multiple computing cores (processors) on a single chip. (1 point) They are more efficient than multiple chips each with single cores because on-chip communication is faster than between-chip communication. (2 points) Also one chip with multiple cores uses significantly less power than multiple single-core chips (1 point)

2.3. In operating system design, why is a *loadable kernel module* approach more commonly used in modern computing systems than a *layered* or *microkernel* approach? (4 points)

Answer: The loadable kernel module approach combines the benefits of both the layered and microkernel design techniques (1 point). In a loadable kernel module design, the kernel needs only to have the capability to perform the required functions and know how to communicate between modules, similar to the microkernel design; functionality can be added to and removed from the kernel while it is running without the need to either recompile or reboot the kernel (3 points)

3. [15 points] Process and Thread

3.1. For a multi-threaded process, can you please name two items each thread within a process shares with others, and two items that are unique to each thread? (2 points)

Answer: share program and data (or file, heap), unique such as program counters, stack, registers, thread local storage. (0.5 point each)

3.2. What is the difference between a *concurrent* system and a *parallel* system? (2 points)

Answer: a parallel system can perform more than one task simultaneously. (1 point) A concurrent system supports more than one task by allowing multiple tasks to make progress interleaved with one another. (1 points)

3.3. Can you please describe two challenges or limitations when designing operating systems for mobile systems when comparing with designing operating systems for traditional PCs? How does this affect the multitasking in mobile systems? (4 points)

Answer: The challenges include less processing capability, or/and less memory, which requires the operating system to handle it carefully (1 point). Also it has to manage the power consumption carefully (1 point). As a result, this usually restricts only one single foreground process to be running and on display while a number of background processes are running without display in multitasking (2 points)

3.4. What is the total number of processes in the following code? Please elaborate. (3 points)

```
pid_t pid1, pid2;
pid1 = fork();
pid2 = fork();
if(pid1>0 && pid2==0){
   if(fork())
      fork();
}
```

Answer: There are 6 processes. After the first two "fork()"s, there would be 4 processes. And only one process enters the "if" part, within this part each "fork()" creates one more process. Finally there are 6 processes.

3.5. What is the total number of processes in the following code? Please elaborate. (4 points)

```
for( int i = 0; i < 5; i ++){
   if(fork()){
       for(int j = 0; j < 3; j ++)
            fork();
   }
}
```

Answer: 9^5. Any process entering the inner "for" part will end up with 8 processes. So after an outer "for" loop, the number of processes will be multiplied by 9. And the total number of processes after 5 outer "for" loops is 9^5.

4. [40 points] Scheduling

4.1. Please describe three CPU scheduling criteria, and how they can be computed. (3 points)

Answer: 1) CPU utilization, which is calculated by the fraction of time that CPU is busy; 2) throughput is defined as the number of processes completed per unit time, which is computed by the total number of processes completed divided by the time period, 3) turnaround time, which can be computed between the time a job is submitted and the time it completes. Waiting time and response time are also fine. (1 point each)

4.2. The CPU scheduling can be *preemptive* and *non-preemptive*, what is the main difference between them? (2 points)

Answer: A non-preemptive scheduling 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 the completion of its current CPU burst (for example waiting for I/O). Otherwise, the scheduling is preemptive in nature. For instance a priority scheduling. (2 points).

4.3. Exponential averaging algorithm (5 points)
Consider the exponential average formula used to predict the length of the next CPU burst. The formula is as follows:

i. $t_n = acutal\ length\ of\ n^{\text{th}}\ CPU\ burst$

ii. $\tau_{n+1} = predicted\ value\ for\ the\ next\ CPU\ burst$

iii. $\alpha,\ 0 \le \alpha \le 1$

iv. Define: $\tau_{n+1} = \alpha \cdot t_n + (1-\alpha) \cdot \tau_n$

Suppose α = 0.6, τ_0= 10, the table below gives the actual CPU burst length t_n, please calculate the predicted CPU burst time at n = 5 (τ_5) (5 points)

n	t_n
0	10
1	5
2	2
3	9
4	6

Answer: According to the formula, we have:

$\tau_1 = 0.6*10 + 0.4*10 = 10$
$\tau_2 = 0.6*5 + 0.4*10 = 7$
$\tau_3 = 0.6*2 + 0.4*7 = 4$
$\tau_4 = 0.6*9 + 0.4*4 = 7$
$\tau_5 = 0.6*6 + 0.4*7 = \underline{6.4}$

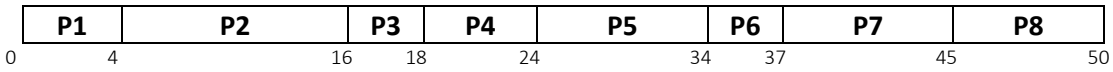
4.4. Given the following set of processes, with arrival time and length of the CPU-burst time given in milliseconds:

Process	Arrival Time (ms)	Burst Time (ms)
P1	0	4
P2	2	12
P3	5	2
P4	6	6
P5	8	10
P6	12	3
P7	15	8
P8	22	5

For each of the following scheduling algorithms, construct the *Gantt chart* depicting the sequence of process execution and calculate the *average waiting time* of each algorithm. (20 points)

a) FCFS (first-come, first-served) (5 points)

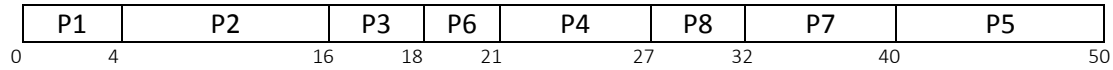
Answer:



Average waiting time = (0+2+11+12+16+22+22+23)/8 = 108/8 =13.5ms

b) SJF (shortest job first) non-preemptive (5 points)

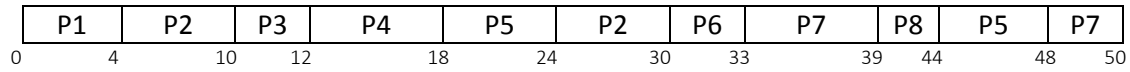
Answer:



Average waiting time = (0+2+11+15+32+6+17+5)/8 = 88/8 =11ms

c) RR (round robin) with quantum of 6 milliseconds (5 points)

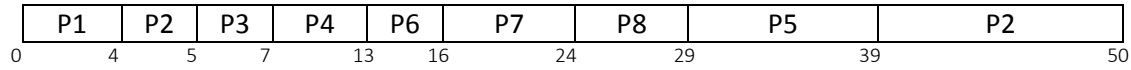
Answer:



Average waiting time = (0+16+5+6+30+18+27+17)/8 = 119/8 =14.875ms

d) SRTF (shortest remaining time first, i.e., preemptive SJF) (5 points)

Answer:

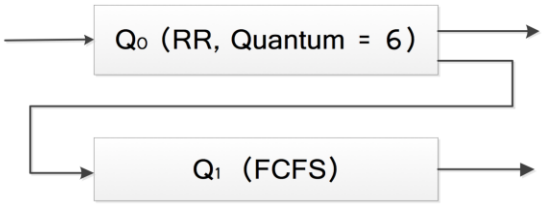


Average waiting time = (0+36+0+1+21+1+1+2)/8 = 62/8 =7.75ms

4.5. Given the arrival time and CPU-burst of 8 processes shown in the following diagram:

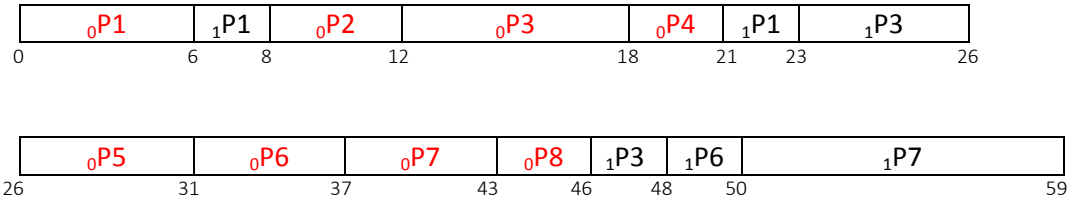
Process	Arrival Time (ms)	Burst Time (ms)
P1	0	10
P2	8	4
P3	10	11
P4	12	3
P5	26	5
P6	29	8
P7	35	15
P8	40	3

Suppose that the operating system uses a 2-level feedback queue to schedule the above 8 processes. Round-Robin scheduling strategy with quantum 6 is used for the queue with the higher priority, and First-Come-First-Serve scheduling strategy is used for the queue with the lower priority. The scheduling is preemptive. (10 points)



Please construct a Gantt chart depicting the scheduling for the set of processes specified in the above diagram using this 2-level feedback queue. For the process currently running on CPU, please also indicate which queue the process is selected from (10 points)

Answer: ($_0$Pi means Pi is selected from Q_0, $_1$Pi for Q_1)



5. [25 points] Process Synchronization

5.1. Please briefly explain the three conditions that a solution to a Critical Section problem must hold. (5 points)

Answer: Mutual exclusion: there is at most one process that can be inside the critical section; progress: If no process is executing in its critical section and there exist some processes that wish to enter their critical sections, the selection of the process that will

enter the critical section next cannot be postponed indefinitely; bounded waiting: a bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

5.2. Suppose there are n instances of a resource in a system, each time one instance of resource can be acquired by a process and utilized exclusively. Explain how semaphores can be used to ensure the correct usage of the resource. (5 points)

Answer: A counting semaphore S can be used, and is initialized to n. A process executes wait(S) before acquiring an instance of the resource, and executes signal(S) after finishing using it. (3 points) If all instances of the resource are used, subsequent request(s) calling wait(S) will be blocked. (2 points)

5.3. Consider the following code, what is the potential problem? (5 points)

Let S and Q be two semaphores initialized to 1：

```
  P0                          P1
wait(S);                    wait(Q);
wait(Q);                    wait(S);
 …                             …
signal(S);                  signal(Q);
signal(Q);                  signal(S);
```

Answer: This can lead to a deadlock (2 points). Specifically, if P0 executes wait(S), gets interrupted, and P1 executes wait(Q), in that P0 and P1 each holds one semaphore waiting for another, this becomes a deadlock (3 points)

5.4. The following code implements a solution to the critical section problem with n processes by using an atomic operation test_and_set(). Please illustrate how the three conditions of the solution are satisfied (10 points)

```
do {
   waiting[i] = true;
   key = true;
   while (waiting[i] && key)
      key = test_and_set(&lock);
   waiting[i] = false;
   /* critical section */
   j = (i + 1) % n;
   while ((j != i) && !waiting[j])
      j = (j + 1) % n;
   if (j == i)
      lock = false;
   else
      waiting[j] = false;
   /* remainder section */
} while (true);
```

Answer:

Mutual-exclusion: P_i enters its critical section only if either waiting[i]==false or key==false**.** The value of key can become false only if test_and_set() is executed. The first process to execute test_and_set() will find key==false; all others must wait. All the variables waiting are set to true. The variable waiting[i] can become false only if another process leaves its critical section; only one waiting[i] is set to false, maintaining the mutual-exclusion requirement.

Progress: since a process existing its critical section either sets lock to false or sets waiting[j] to false. Both allow a process that is waiting to enter its critical section to proceed.

Bounded-waiting; when a process leaves its critical section, it scans the array waiting in cyclic order {i+1, i+2, …,n-1,0,1,…i-1}. It designates the first process in this ordering that is in the entry section (waiting[j]==true) as the next one to enter the critical section. Any process waiting to enter its critical section will thus do so within n-1 turns.