COMP 3511 Operating Systems

Spring Semester 2014

Midterm Examination

Date: April 8, 2014 (Tuesday) Time: 16:30 - 18:00

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 7 questions and 10 pages (including this page).
3. You have **90 minutes** to complete this exam.
4. Please answer all questions within the space provided on the examination paper. You may use back of the pages for your rough work. 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, legible, and readable.
7. Show all the steps used in deriving your answer, wherever appropriate.

Question	Points	Score
1	10	
2	12	
3	18	
4	15	
5	13	
6	17	
7	20	
Total	100	

1. (10 points) Multiple Choice Question.

1) In operating system, each process has its own _____.
A) address space and global variables
B) open files
C) pending alarms, signals and signal handlers
D) all of the mentioned
Answer: D

2) A _____ uses an existing thread — rather than creating a new one — to complete a task.
A) lightweight process
B) thread pool
C) scheduler activation
D) asynchronous procedure call
Answer: B

3) In multithreaded programs, the kernel informs an application about certain events using a procedure known as a(n) ____.
A) signal
B) upcall
C) event handler
D) pool
Answer: B

4) Which one of the following is not shared by threads?
A) program counter
B) stack
C) both (a) and (b)
D) none of the mentioned
Answer: C

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

6) The main problem with a priority scheduling algorithm is _____.
A) complexity
B) starvation
C) determining the length of the next CPU burst
D) determining the length of the time quantum
Answer: B

7) Which of the following scheduling algorithms must be nonpreemptive?
A) SJF
B) RR
C) FCFS
D) priority algorithms

Answer: C

8) 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

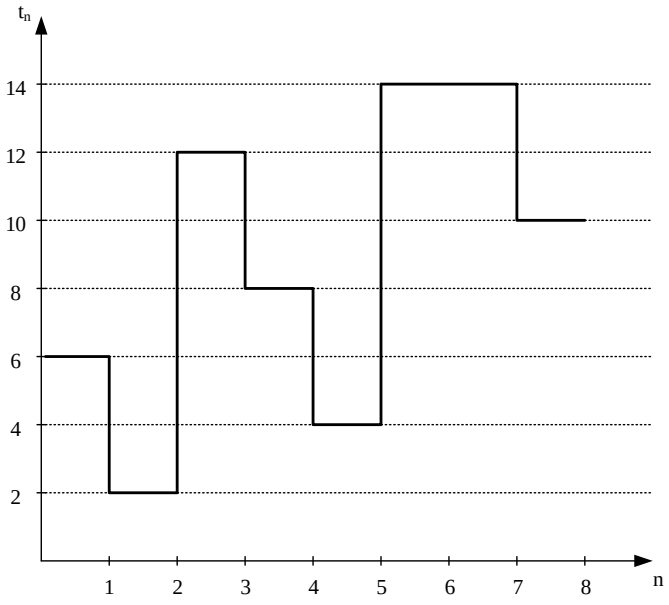
9) How many philosophers may eat simultaneously in the Dining Philosopher problem with 5 philosophers?
A) 1
B) 2
C) 3
D) 5
Answer: B

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. (12 points) Consider the exponential average formula used to predict the length of the next CPU burst. The formula is as follows:

i. $t_n = acutal\ length\ of\ n^{th}\ CPU\ burst$
ii. $\tau_{n+1} = predicted\ value\ for\ the\ next\ CPU\ burst$
iii. $\alpha, 0 \leq \alpha \leq 1$
iv. Define: $\tau_{n+1} = \alpha \cdot t_n + (1 - \alpha) \cdot \tau_n$

2.1. (4 points) Suppose $\alpha = 0.5, \tau_0 = 6$, calculate the predicted CPU burst at $n = 1, 2, \ldots, 8$.



Answer: according to the formula, we have $\tau_n = \alpha \cdot t_{n-1} + (1-\alpha) \cdot \tau_{n-1}$

$\tau_1 = 0.5 \cdot 6 + 0.5 \cdot 6 = 6;$

$\tau_2 = 0.5 \cdot 2 + 0.5 \cdot 6 = 4;$

$\tau_3 = 0.5 \cdot 12 + 0.5 \cdot 4 = 8;$

$\tau_4 = 0.5 \cdot 8 + 0.5 \cdot 8 = 8;$

$\tau_5 = 0.5 \cdot 4 + 0.5 \cdot 8 = 6;$

$\tau_6 = 0.5 \cdot 14 + 0.5 \cdot 6 = 10;$

$\tau_7 = 0.5 \cdot 14 + 0.5 \cdot 10 = 12;$

$\tau_8 = 0.5 \cdot 10 + 0.5 \cdot 12 = 11.$

2.2. (8 points) What are the implications of assigning the following values to the parameters used by the algorithm? Hint: the formula can be expanded as:

$$\tau_n = \alpha \cdot t_{n-1} + \alpha \cdot (1-\alpha) \cdot t_{n-2} + \cdots + \alpha \cdot (1-\alpha)^{k-1} \cdot t_{n-k} + \cdots + (1-\alpha)^n \cdot \tau_0$$

2.2.1.(3 points) $\alpha = 0$, $\tau_0 = 100$ milliseconds.

2.2.2.(5 points) $\alpha = 0.99$, $\tau_0 = 10$ milliseconds.

Answer:

a) When $\alpha = 0$ and $\tau_0 = 100$ milliseconds, the formula is:

$$\tau_n = \tau_0$$

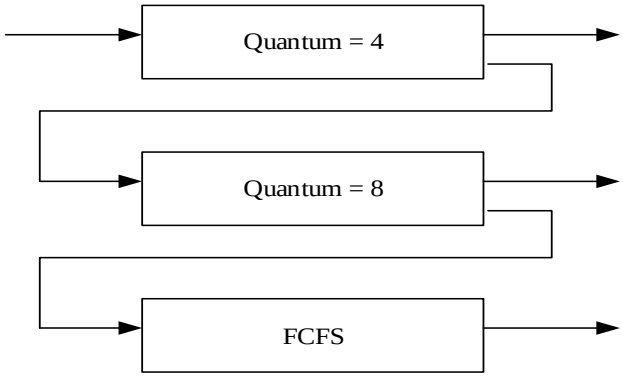
which means it always makes a prediction of 100 milliseconds for the next CPU burst.

b) When α = 0.99 and τ0 = 10 milliseconds, the formula is:

$\tau_n = 0.99 \cdot t_{n-1} + 0.99 \cdot 0.01 \cdot t_{n-2} + \cdots + 0.99 \cdot 0.01^{k-1} \cdot t_{n-k} + \cdots + 0.01^n \cdot 10$

which means 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.

3. (18 points) Consider 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 non-preemptive. A newly arrived process enters the first queue (RR with the quantum 4) before moving down to the other two queues if it is necessary.

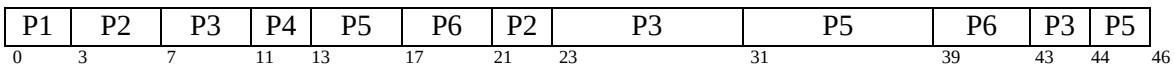


The processes are assumed to arrive in the order P1, P2, P3, P4, P5, and P6 all at time 0.

Process	P1	P2	P3	P4	P5	P6
Burst Time /ms	3	6	13	2	14	8

Draw a Gantt chart depicting the process scheduling sequence (10 points) and calculate the average waiting time (8 points).

Answer:



Waiting time:

P1 = 0 – 0 = 0; P2 = (3-0) + (21 – 7) = 17; P3 = (7 – 0) + (23 – 11) + (43 – 31) = 31;

P4 = (11 – 0) = 11; P5 = (13 – 0) + (31 – 17) + (44 – 39) = 32; P6 = (17 – 0) + (39 – 21) = 35.

Average waiting time = (0 + 17 + 31 + 11 + 32 + 35) / 6 = 21 milliseconds.

4. (15 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
CPU-burst	7	14	3	9	12
Arrival Time	0	4	8	10	19

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

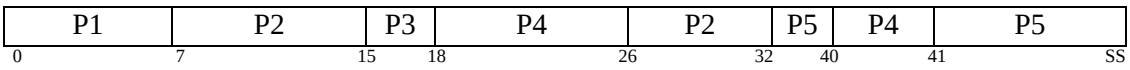
4.1. (6 points) RR with quantum 8 milliseconds (round robin)

4.2. (4 points) SJF (shortest job first)

4.3. (5 points) SRTF (shortest remaining time first)

Answer:

RR



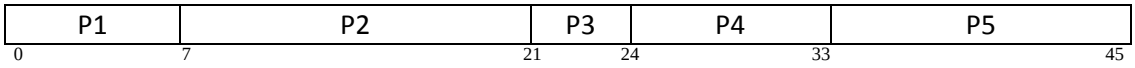
45

Waiting time: P1 = 7 – 0 = 0; P2 = (7 – 4) + (26 – 15) = 14; P3 = 15 – 8 = 7;

P4 = (18 – 10) + (32 – 26) = 14; P5 = 33 – 19 = 14.

Average waiting time = (0 + 14 + 7 + 14 + 14) / 5 = 9.8 milliseconds.

SJF



Waiting time: P1 = 0 – 0 = 0; P2 = 7 – 4 = 3; P3 = 21 – 8 = 13;

P4 = 24 – 10 = 14; P5 = 33 – 19 = 14.

Average waiting time = (0 + 3 + 13 + 14 + 14) / 5 = 8.8 milliseconds.

SRTF

P1	P2	P3	P4	P5	P2

0 7 8 11 20 32 45

Waiting time: P1 = 0 – 0 = 0; P2 = (7 – 4) + (32 – 8) = 27; P3 = 8 – 8 = 0;

P4 = 11 – 10 = 1; P5 = 20 – 19 = 1.

Average waiting time = (0 + 27 + 0 + 1 + 1) / 5 = 5.8 milliseconds.

5. (13 points) In a Producer-Consumer system, producer (P) and consumer (C) processes share a circular buffer of size 10 (buf[10]). P produces computed data into buf[10], while C consumes data from buf[10]. Refer to the following code, fill in the blanks and answer the questions.

```
void main()
{
  in = out = 0; //integer
  semaA = 10;
  semaB = 0;
  mutex = 1;

  Synch begin:
    P();
    C();
  Synch end
}
```

```
Void P()
{ tmp;
  while(true)
  {
    Computing tmp;
    Wait(___①_____);
    Wait(mutex);
    buf[in] = tmp;
    in = (in + 1) % 10;
    Signal(mutex);
    Signal(___②_____);
  }
}
```

```
Void C()
{ tmp;
  while(true)
  {
    Wait(semaB);
    Wait(mutex);
    tmp = buf[out];
    ___③_____;
    Signal(mutex);
    Signal(semaA);
  }
}
```

5.1. (3 points) Fill in the blanks marked by ①, ② and ③.

Answer: Wait(semaA); Signal(semaB); out = (out + 1) % 10

5.2. (5 points) Please briefly explain the purpose of in, out, semaA, semaB, mutex.

Answer:

in: buffer index for writing data in order;

out: buffer index for reading data in order;

semaA: semaphore for empty buffer cells;

semaB: semaphore for full buffer cells;

mutex: mutual exclusion from buffer.

5.3. (5 points) If we remove semaphore "mutex" and the corresponding "Wait()" and "Signal()" code, what kind of problem will occur? Please explain.

Answer: It might enable concurrent access to critical section. For example, if at the beginning, semaA equals to 10, and two producers (i.e. P_i, P_j) both will enter the critical section (semaA equals to 8). What's worse, they will access to buf[0] simultaneously, and incur memory conflicts.

6. (17 points) Answer the questions according to the code below.

```
int main(void){
    pid_t pid;
    pid=fork();
    if (pid < 0)
    {
            fprintf(stderr, "Fork failed!\n");
            exit(1);
    }
    if (pid > 0)
    {
            printf("Parent process.\n");
            wait(NULL);
            printf("Child process ID %d, exited.\n", pid);
            exit(0);
    }
    else
            printf("Child process, line 1.\n");
    printf("Child process, line 2.\n");
    exit(0);
}
```

6.1. (2 points) What is the function of "*wait(NULL);*"?

Answer: The parent process waits until child process terminates.

6.2. (9 points) Assume the child process id is 12345, there are 4 possible output sequence of this program (screen printout), two are listed below, please write down the other two program outputs.

Answer:

1	**Fork failed!**
2	Parent process. Child process line 1. Child process line 2.

	Child process ID 12345, exited.
3	Child process 1. Parent process. Child process line 2. Child process ID 12345, exited.
4	Child process line 1. Child process line 2. Parent process. Child process ID 12345, exited.

6.3. (6 points) Now we replace the "*printf("Child process, line 1.\n");*" with a "exec()" statement to execute a new program which does nothing except printing "exec() exit." Suppose the new program terminates successfully and child process is 12345, what are the other two possible printouts on the screen?

Answer:

1	**Fork failed!**
2	Parent process. exec() exit. Child process ID 12345, exited.
3	exec() exit. Parent process. Child process ID 12345, exited.

7. (20 points) Q&A

7.1. (3 points) Please explain two main advantages in the microkernel design. Please specify how user programs and system services interact in a microkernel architecture.

Answer: Benefits typically include the following: (1) adding a new service does not require modifying the kernel, (2) it is more secure as more operations are done in user mode than in kernel mode, and (3) a simpler kernel design and functionality typically results in a more reliable operating system. User programs and system services interact in a microkernel architecture by using interprocess communication mechanisms such as messaging (1 point)

7.2. (2 points) What is the difference between a short-term scheduler and a long-term scheduler?

Answer: A short-term (CPU scheduler) selects a process or thread from the ready queue to run on a CPU (1 point), while a long-term (job scheduler) determines which jobs are brought into memory for processing or dictates the degree of multiprogramming (1 point).

7.3. (4 points) Modern operating systems are all dual-mode; what are the dual-mode operations, and why is this needed?

Answer: User mode and kernel mode (2 points). The dual-mode provides the means for protecting the operating system from errant users and errant operations (2 points).

7.4. (4 points) What is the difference between concurrency and parallelism? What kind of systems can support parallelism?

Answer: 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 (2 points). The scheduling of tasks allows for concurrency, but parallelism is supported only on systems with more than one processor (2 points).

7.5. (3 points) What three conditions must be satisfied in order to solve the critical section problem?

Answer: mutual exclusion, progressing and bounded waiting

7.6. (4 points) How does the signal() operation associated with a monitor differ from the corresponding operation defined for semaphores?

Answer: The signal() operation associated with monitors is not persistent in the following sense: if a signal is performed and if there are no waiting threads, then the signal is simply ignored and the system does not remember that the signal took place (2 points). If a subsequent wait operation is performed, then the corresponding thread simply blocks. In semaphores, on the other hand, every signal results in a corresponding increment of the semaphore value even if there are no waiting threads. A future wait operation would immediately succeed because of the earlier increment (2 points).