CSE 421/521 - Operating Systems
Fall 2018

Midterm-I Topics & Sample Exam Questions

Tevfik Koşar

University at Buffalo

Midterm-I Exam

October 9th, Tuesday

11:00am - 12:20pm

@NSC 201

Chapters included in the Midterm Exam

- Ch. 1 (Introduction)
- Ch. 2 (OS Structures)
- Ch. 3 (Processes)
- Ch. 4 (Threads)
- Ch. 5 (Synchronization)
- Ch. 6 (CPU Scheduling)
- Ch. 7 (Deadlocks)

1 & 2: Overview

- Basic OS Components
- OS Design Goals & Responsibilities
- OS Design Approaches

3. Processes

- Process Creation & Termination
- Context Switching
- Process Control Block (PCB)
- Process States
- Process Queues & Scheduling

4. Threads

- Concurrent Programming
- Threads vs Processes
- Threading Implementation & Multi-threading Models
- Other Threading Issues
  - Thread creation & cancellation
  - Signal handling
  - Thread pools
  - Thread specific data

5. Synchronization

- Race Conditions
- Critical Section Problem
- Mutual Exclusion
- Semaphores
- Monitors
- Classic Problems of Synchronization
  - Bounded Buffer
  - Readers-Writers
  - Dining Philosophers
  - Sleeping Barber

6. CPU Scheduling

- Scheduling Criteria & Metrics
- Scheduling Algorithms
  - FCFS, SJF, Priority, Round Robing
  - Preemptive vs Non-preemptive
  - Gantt charts & measurement of different metrics
- Multilevel Feedback Queues
- Estimating CPU bursts

7. Deadlocks

- Deadlock Characterization
- Deadlock Detection
  - Resource Allocation Graphs
  - Wait-for Graphs
  - Deadlock detection algorithm
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Recovery

Sample Exam Questions

Question 1

Are each of the following statements True or False? Circle the correct answer.

(a) In multiprogramming, it is safe to have an arbitrary number of threads/ processes reading a piece of data at once. (True / False)

(b) Kernel mode can directly access hardware devices, user mode cannot. (True / False)

(c) Deadlocks cannot arise without semaphores. (True / False)

(d) Semaphores are destroyed by the OS when your process exits. (True / False)

(e) A process that is blocked is not given any processor time by the scheduler until the condition that caused the blocking no longer applies. (True / False)

Question 2-a

A system that meets the four deadlock conditions will **always/ sometimes/never** result in deadlock?

Question 2-b

Round-robin scheduling **always/sometimes/never** results in more context switches than FCFS?

Question 2-c

Which of the following scheduling algorithms can lead to starvation (**FIFO/Shortest Job First/Priority/Round Robin**)?

Question 2-d

What approach to dealing with deadlock does the “Wait-for Graphs” implement (**prevention/avoidance/detection/recovery**)?

Question 3

Process ID	Arrival Time	Priority	Burst Time
A	0	5	20
B	4	1	12
C	12	2	16
D	16	4	4
E	20	3	8

Consider the above set of processes.

a) Draw Gantt chart illustrating the execution of these processes using Shortest Job First (Preemptive) algorithm.

b) What is the waiting time of each process

c) What is the turnaround time of each process

Question 4

In the code below, assume that *(i)* all `fork` and `execvp` statements execute successfully, *(ii)* the program arguments of `execvp` do not spawn more processes or print out more characters, and *(iii)* all `pid` variables are initialized to `0`.

a. What is the total number of processes that will be created by the execution of this code?
b. How many of each character 'A' to 'G' will be printed out?

Question 4 *(cont)*

```
void main()
{
      ...
      pid1 = fork();
      pid2 = fork();
      if (pid1 != 0) {
           pid3 = fork();
           printf("A\n");
      } else {
           printf("B\n");
           execvp(...);
      }
      if (pid2 == 0 && pid3 != 0) {
           execvp(...);
           printf("C\n");
      }
      pid4 = fork();
      printf("D\n");
      if (pid3 != 0) {
           printf("E\n");
           pid5 = fork();
           execvp(...);
      }
      printf("F\n");
      execvp(...);
      pid6 = fork();
      printf("G\n");
      if (pid6 == 0)
           pid7 = fork();
}
```

Question 5

Assume S and T are binary semaphores, and X, Y, Z are processes. X and Y are identical processes and consist of the following four statements:

P(S); P(T); V(T); V(S)

And, process Z consists of the following statements:

P(T); P(S); V(S); V(T)

Would it be safer to run X and Y together or to run X and Z together? Please justify your answer.

Question 6

Remember that if the semaphore operations *Wait* and *Signal* are not executed atomically, then mutual exclusion may be violated. Assume that *Wait* and *Signal* are implemented as below:

```
void Wait (Semaphore S) {
      while (S.count <= 0) {}
      S.count = S.count - 1;
}
```

```
void Signal (Semaphore S) {
      S.count = S.count + 1;
}
```

Describe a scenario of context switches where two threads, T1 and T2, can both enter a critical section guarded by a single mutex semaphore as a result of a lack of atomicity.

Question 7

```
boolean lamp[2];
int book = 0;
```

```
void do_thread0()
{
    while (true) {
        lamp[0] = true;
        while (lamp[1]) {
            if (book == 1) {
                lamp[0] = false;
                while (book == 1);
                    /* nothing */
                lamp[0] = true;
            }
        }
        /*** CRITICAL REGION 0 ***/
        book = 0;
        lamp[0] = false;
        ...
    }
}
```

```
void do_thread1()
{
    while (true) {
        lamp[1] = true;
        while (lamp[0]) {
            if (book == 0) {
                lamp[1] = false;
                while (book == 0);
                    /* nothing */
                lamp[1] = true;
            }
        }
        /*** CRITICAL REGION 1 ***/
        book = 1;
        lamp[1] = false;
        ...
    }
}
```

Question 7 *(cont)*

Does this code guarantee mutual exclusion of the two threads from their respective critical regions?

Question 7-b

Does this code guarantee "progress", i.e., if one thread is currently executing outside its critical region, the other thread will always have the opportunity to enter its own critical region?

Question 8

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) $a = 1$ and $T_0 = 100$ ms

b) $a = 0$ and $T_0 = 10$ ms

Question 9

In the code below, three processes are competing for six resources labeled A to F.

a. Using a resource allocation graph (Silberschatz pp.249-251) show the possiblity of a deadlock in this implementation.

b. Modify the order of some of the `get` requests to prevent the possiblity of any deadlock. You cannot move requests across procedures, only change the order inside each procedure. Use a resource allocation graph to justify your answer.

```
void P0()
{
  while (true) {
    get(A);
    get(B);
    get(C);
    // critical region:
    // use A, B, C
    release(A);
    release(B);
    release(C);
  }
}
```

```
void P1()
{
  while (true) {
    get(D);
    get(E);
    get(B);
    // critical region:
    // use D, E, B
    release(D);
    release(E);
    release(B);
  }
}
```

```
void P2()
{
  while (true) {
    get(C);
    get(F);
    get(D);
    // critical region:
    // use C, F, D
    release(C);
    release(F);
    release(D);
  }
}
```

Question 10

Consider a system using round-robin scheduling with a fixed quantum q. Every context switch takes s milliseconds. Any given process runs for an average of t milliseconds before it blocks or terminates (burst time).

(a) Determine the fraction of CPU time that will be wasted because of context switches for each of the following cases (your answer should be in terms of q, s, and t).

i. $t <= q$:

ii. $t >> q$ (t is much greater than q) :

iii. q approaches 0 :

(b) Under what conditions will the wasted fraction of CPU time be exactly 50%?

Question 11

Consider the Sleeping Barber Problem, in which there is a barbershop which consists of a waiting room with n chairs and a barber room with one barber chair. If there are no customers to be served, the barber goes to sleep. If a customer enters the barbershop and all chairs are occupied, then the customer leaves the shop. If the barber is busy but chairs are available, then the customer sits in one of the free chairs. If the barber is asleep, the customer wakes up the barber.

Consider the solution to this problem which uses three semaphores:

- Semaphore Customers
- Semaphore Barber
- Semaphore accessSeats (mutex)

And an integer to keep track of free seats:

- int NumberOfFreeSeats

Please use the above semaphores correctly in the empty lines of the solution below:

Question 11 (cont.)

The Barber(Thread):	The Customer(Thread):
while(true){	while (notCut){
(1) ________________________	(5) ________________________
	if (NumberOfFreeSeats>0) {
(2) ________________________	NumberOfFreeSeats --;
NumberOfFreeSeats++;	(6) ________________________
(3) ________________________	(7) ________________________
(4) ________________________	(8) ________________________
barber is cutting hair;	notCut = false;
}	}
	else
	(9) ________________________
	}

Question 12

Consider a system with three smoker processes and one agent process. Each smoker continuously rolls a cigarette and then smokes it. But to roll and smoke a cigarette, the smoker needs three ingredients: tobacco, paper and matches. One of the smoker processes has paper, another has tobacco and the third has the matches. The agent has an infinite supply of all three materials. The agent places two of the ingredients on the table. The smoker who has the remaining ingredient then makes and smokes a cigarette, signaling the agent on completion. The agent then puts out another two of the three ingredients, and the cycle repeats.

Given below is a solution to the Cigarette-Smokers Problem. Give initial conditions for the semaphores as well as plausible values for the variables j, r & s, such that the agent and smokers are synchronized. Write a couple of sentences on why these initial conditions are necessary and sufficient.

Question 12 *(cont.)*

var a: array [0..2] of semaphore {**initial condition** = ____________ }

agent: semaphore {**initial condition** = _____________ }

Agent code :

repeat
Set i to a **value** = rand(3)**, and** j to a **value** = ____________
wait(agent);
signal(a[i]);
signal(a[j]);
until false;

Smoker code (for Smoker k):

repeat
Set r to a **value** = ______________**, and** s to a **value** =
____
wait(a[r]);
wait(a[s]);
"smoke"
signal(agent);
until false;

Finally...

- Don't forget to review:

  1. the quiz questions & solutions
  2. the homework questions and solutions