CS 4348/5348 Operating Systems, Homework #1

1.
Consider the following different types of operating systems.
A. Multi-programming batch systems.
B. Multi-programming time sharing systems.
C. An embedded computer that is used on your car to regulate the brake system. (It takes speed and brake pedal pressing level as input and brake pressure as output. The system has a simple CPU and some memory, but no disk.)

Also, consider different types of interrupts:
X: IO completion interrupt
Y: timer interrupt
Z: hardware interrupt, generally occurs when a device has some input or problems
(e.g., keyboard press, network input ready, disk failure, etc.)

For each of the OS given above, give the interrupts that the system should support. For each, put the interrupts by listing their labels. If your answer includes multiple interrupts, put the labels in alphabetical order. For example, if you believe that IO and hardware interrupts are needed, then put XZ. If none is needed, then put none.

For A:
For B:
For C:

A: XZ
B: XYZ
C: none (inputs directly flow to the computer and output comes out directly to the control)

X: When a user submits a command by typing the command and hit the return key, the keyboard generates an interrupt, which is the hardware interrupt. The system invokes the corresponding interrupt handler and process the input (processing the input command may be a new process which is placed in ready queue).

A: Each process is executed till it encounters an IO or has completed ⇒ No need for a timer interrupt (Y is ok, can be used to control the potential infinite loops)
C: The system has only a single task and requires real time reaction ⇒ No need for any OS and OS would incur additional overhead, which is not desirable.

2.
Consider the following pseudo assembly code for computing c = a + b. Assume that a, b, and c are assigned to consecutive memory "words" (memory is generally addressed byte by byte and assume that a word is of 4 bytes) and address for "a" is 0x0000e308. Also, we have a = 20, b = 136, and c = 0 at the starting time. Assume that the first instruction of the code is stored in 0x00001018. Also, each instruction has the opcode in the first byte (most significant byte) and the remaining 3 bytes specify the corresponding address. The opcode for load is 1, store is 2, and add is 3.

```
load a
add b
store c
```

Determine what are in the memory on a draft paper. Write the micro instructions for the code segment given above. Assume that current PC (program counter) is 0x00001018. For each micro-instruction, also indicate the data that is transferred. For all data, use the hexdecimal representation. The following are the first two micro-instructions and the data transferred. Complete the rest.

Micro-instructions	data
PC → MAR	0x00001018
M → MBR	0x0100e308

PC → MAR	0x00001018
M → MBR	0x0100E308
MBR → IR	0x0100E308
IR (operand) → MAR	0x0000E308
M → MBR	0x00000014
MBR → AC	0x00000014
PC+4 → PC	0x0000101C

PC → MAR	0x0000101C
M → MBR	0x0300E30C
MBR → IR	0x0300E30C
IR (operand) → MAR	0x0000E30C
M → MBR	0x00000088
MBR + AC → AC	0x0000009C
PC+4 → PC	0x00001020

PC → MAR	0x00001020
M → MBR	0x0200E310
MBR → IR	0x0200E310
IR (operand) → MAR	0x0000E310
AC → MBR	0x0000009C
MBR → M	0x0000009C
PC+4 → PC	0x00001024

3.
Assume that before time 5, no system resources are in use except for the CPU and memory. Now consider the following events:

At time 5: P1 executes a command to read from disk unit 3.
At time 15: P5's time slice expires.
At time 18: P7 executes a command to write to disk unit 3.
At time 20: P3 executes a command to read from disk unit 2.
At time 24: P5 executes a command to write to disk unit 3.
At time 28: P5 is swapped out.
At time 33: An interrupt occurs from disk unit 2: P3's read has completed.
At time 36: An interrupt occurs from disk unit 3: P1's read has completed.
At time 38: P3 terminates.
At time 40: An interrupt occurs from disk unit 3: P5's write has completed.
At time 44: P5 is swapped back in.
At time 48: An interrupt occurs from disk unit 3: P7's write has completed.

At time 37, which state each process is in. Since we do not always know which process is running, you can ignore the running state. You should use the following codes to represent the states:
R: ready; B: blocked; S: suspended (BS means blocked suspended)

P1:
P3:
P5:
P7:

A: R
B: R
C: BS
D: B

	P1	P3	P5	P7
time 5	wait	ready	ready	ready
time 15			ready	
time 18				wait
time 20		wait		
time 24			wait	
time 28			wait susp	
time 33		ready		
time 36	ready			
time 38		exit		
time 40			ready susp	
time 44			ready	
time 48				ready

4
Consider two functions, P and Q, as shown in the following code. Note that each statement is labeled and the label is just for convenience (no impact to execution).

```
function P()                         function Q()
{   A. a = a + 1;                    {   D. b = 4;
    B. b = a + b;                        E. a = a + b;
}                                    }
```

The pseudo code for the main program is given below (main1). Assume that the program (main1) is compiled into executable “a.out”. And, “a.out” is invoked by the program given second below (main2).

```
main1 ()
{ int a = 0;   int b = 3;
  create-thread (execute function P);
  create-thread (execute function Q);
}

main2 ()
{ int pid;   int a = 2;   int b = 3;
  pid = fork ();
  if (pid == 0) { a = 4;   exec (“a.out”); }
  else { a = 4;   Q(); }
}
```

List all possible values of the pair (a, b) after the execution of the program (main2). Consider the possible values in the parent process and the child process separately. Note that for this question, we assume that each statement is atomic, not each instruction.

Since we use automatic grading, please arrange your answer so that it is sorted by “a” first and then by “b”, from smallest to largest. An example answer is like: (1,2), (1,5), (2,1), …

(a,b) in the parent process:
(a,b) in the child process:

Possible execution sequences and outputs

ABDE	ADBE	ADEB	DABE	DAEB	DEAB
a=5, b=4	a=6, b=5	a=5, b=9	= ADBE	= ADEB	a=5, b=9

Parent: (8,4)
Child: (5,4), (5,9), (6,5)

5
Consider the two functions, P and Q, given below. Variables a and b are initialized to 0. Assume that P and Q are running as two concurrent threads in a program. After execution, what are the possible values of variables a and b. Note: as we have discussed (without being specifically changed in a question), instruction is the level of atomicity.

```
function P()
{   A. a = a + 1;
    B. b = b + 2;
}

function Q()
{   D. b = 4;
    E. a = a + b;
}
```

You need to analyze different execution sequences to get all possible value pairs (a,b). Since we use automated grading, please arrange your answer so that it is sorted by “a” first and then by “b”, from smallest to largest. An example answer is like: (1,2), (1,5), (2,1), …

(Thread P)	
A1	load a
A2	add 1
A3	store a
B1	load b
B2	add 2
B3	store b

(Thread Q)	
D1	load 4
D2	store b
E1	load a
E2	add b
E3	store a

Principle 1:
load only gets impacted by stores of the same variable (not any other)
store impacts loads and other stores of the same variable

Principle 2:
When a constant is loaded and stored directly, only store matters (since load will not be impacted)

Principle3:
E2 is the same as load b;
Combine E1 and E2 $\Rightarrow$ E1; actions; E2;
If actions = store a; store b $\Rightarrow$ actions cannot be moved out of the current position
(move store a before E1 $\Rightarrow$ different effect), (move store b after E2 $\Rightarrow$ different effect)
Combine E2 and E3 $\Rightarrow$ E2; actions; E3; $\Rightarrow$ actions can always be moved out to be before E2 or after E3
If actions = store b; store a; $\Rightarrow$ actions cannot be moved out (but there is no such possibility in P)
No problem for all other cases
$\Rightarrow$ E2 and E3 can be considered together

$\Rightarrow$ So, we only need to consider ordering A1, A3, B1, B3 with D, E1, E3(=E2;E3)

Insert D into P’s sequence (D only impacts B1, B3):
D A1 A3 B1 B3
A1 A3 B1 D B3
A1 A3 B1 B3 D

Now insert E1 (load a) into the above sequences (only depends on store a or b, i.e., A3 and B3)
D E1 A1 A3 B1 B3
D A1 A3 E1 B1 B3
D A1 A3 B1 B3 E1

A1 A3 B1 D E1 B3
A1 A3 B1 D B3 E1

A1 A3 B1 B3 D E1

Now insert E3 (store a) into the above sequences (only impacts A1 and A3)

```
        D E1 E3 A1 A3 B1 B3   DEAB => (5,6)
        D E1 A1 E3 A3 B1 B3   b=4; a-temp-E=0; a-temp-A=0; a=4; a=1; b=6; => (1,6)
        D E1 A1 A3 E3 B1 B3   b=4; a-temp-E=0; a=1; a=4; b=6; => (4,6)
        D E1 A1 A3 B1 B3 E3   b=4; a-temp-E=0; a=1; b=6; a=6; => (6,6)
        ---
        D A1 A3 E1 E3 B1 B3   DAEB => (5,6)    (DA1A3E1B1B3E3 = DA1A3B1B3E1E3 = DABE)
        ---
        D A1 A3 B1 B3 E1 E3   DABE => (7,6)
        ------
        A1 A3 B1 D E1 E3 B3   a=1; b-temp-B=2; b=4; a=5; b=2; => (5,2)
        A1 A3 B1 D B3 E1 E3   a=1; b-temp-B=2; b=4; b=2; a=3; => (3,2)
        ------
        A1 A3 B1 B3 D E1 E3   ABDE => (5,4)
```

(1,6), (3,2), (4,6), (5,2), (5,4), (5,6), (6,6), (7,6)

6.1

Consider the following function implementing a lock solution. Find a counter example (execution sequence) that demonstrates that this solution is incorrect and it will violate mutual exclusion.

```
var blocked: array[0..1] of boolean;
var turn: 0..1;  // could be 0 or 1

function p (id: integer)
{   repeat
        blocked[id] := true;
        while (turn != id) do
        {   while (blocked[1–id]) do;  // do nothing
            turn := id;
        }
        <critical section >
        blocked[id] := false;
        <remainder code for p>
    until false;
}

main ()
{   blocked[0] := false; blocked[1] := false;
    create thread to execute P(0);
    create thread to execute P(1);
}
```

Currently turn = 0
P1: set blocked[1] = true;
P1: check and find turn is not 1 (while (turn != id) do)
P1: check and find blocked[0] = false, exit inner loop (while (blocked[1–id]) do;)
P0: set blocked[0] = true;
P0: check and find turn is 0, exit outer loop (while (turn != id) do)
P0: enter critical section
P1: set turn = 1;
P1: check and find turn is 1, exit outer loop (while (turn != id) do)
P1: enter critical section

6.2

In the Jurassic Park Amusement center, tourists ride on a tour bus to visit the park. Each bus takes two visitors from the visitor center, drives around the park as long as the passengers wish, and return to the visitor center to drop off the passengers. There are N tour buses on service .If the N buses are all out, then a passenger who wants a ride must wait.If a bus is ready to load but there are no waiting visitors, then the bus waits. A bus does not leave until it loads two passengers. A bus does not load passengers if there is another bus that is only partly filled.

Consider the problem of synchronizing the bus processes and the visitor processes using semaphores. The code for visitor process follows. Three semaphores cust, bus, and busReady are used. Write the code for the bus process that performs the proper operations on the three semaphores. Use extra semaphores as needed. In the bus process, you can use the same pseudo-statements "visitor get in bus"', etc. as in the visitor process. Do not forget to properly initialize the semaphores (including the three given in the code and any new ones you may introduce).

```
cust: semaphore (:= ?);
bus: semaphore (:= ?);
busReady: semaphore (:= ?);

procedure visitor;
{   signal (cust);
    wait (bus);
    visitor gets in bus;
    wait (busReady);
    drive around the park;
    visitor gets off bus;
}
```

```
cust: semaphore := 0;
bus: semaphore := 0;
busReady: semaphore := 0;
mutex := 1;

procedure bus;
begin
  wait (mutex);
    signal (bus);
    signal (bus);
    wait (cust);
    wait (cust);
    signal (busReady);
    signal (busReady);
  signal (mutex);
  drive around the park;
  drop visitors;
end;
```

*** a bus needs to wait for two customers in mutex, otherwise, two waiting customers may get on two different buses.