OS Tutorial - 2 (Process Synchronization)

1. Three concurrent processes P1, P2 and P3 are concurrently updating a shared variable var (with initial value of 110) as follows:
   P1: var = var – 15;
   P2: var = var + 25;
   P3: var = var * 3;
   What can be the possible maximum and minimum values of var after execution of the three processes? Show the execution order of P1, P2, P3 in the cases of the possible maximum and minimum values.

   P1:
   1. r1 = var
   2. r1 = r1 - 15
   3. var = r1

   P2:
   1. r2 = var
   2. r2 = r2 + 25
   3. var = r2

   P3:
   1. r3 = var
   2. r3 = r3 * 3
   3. var = r3

   Max - value : P1.1 P1.2 P2.1 P2.2 P2.3 P3.1 P3.2 P1.3 P3.3 - 405
   Min - value : P1.1 P1.2 P2.2 P2.2 P2.3 P3.1 P3.2 P3.3 P1.3 - 95
   Other cases also possible

2. (a) Consider a program consisting of 3 concurrent processes P0, P1 and P2 as shown below, and 3 binary semaphores that are initialized as: S0 = 1, S1 = 0; S2 = 0. How many times will the process P0 print ‘0’?

Process P0	Process P1	Process P2

while (true) {
wait (S0);
print '0';
signal (S1);
signal (S2);
}
	wait (S1);
	signal (S0);
		wait (S2);
		signal (S0);

Maximum no. of time 0 printed is thrice when execute in this order (p0 -> p1 -> p0 -> p2 -> p0).

Minimum no. of time 0 printed is twice when execute in this order (p0 -> p1 -> p2 -> p0)

(b)Consider the following threads, T_1, T_2, and T_3 executing on a single processor, synchronized using three binary semaphore variables,S1, S2, and S3, operated upon using standard *wait()* and *signal()*. The threads can be context switched in any order and at any time.Which initialization of the semaphores would print the sequence BCABCABCA….?

T_1	T_2	T_3

while(true){
wait(S_3);
print("C");
signal(S_2);}
	while(true){
	wait(S_1);
	print("B");
	signal(S_3);}
		while(true){
		wait(S_2);
		print("A");
		signal(S_1);}

As T_2 should execute first, in order to print 'B'.
S_1 must be 1 and S_2 and S_3 must be 0.

3. Consider two processes P1 and P2 accessing the shared variables X and Y protected by two binary semaphores SX and SY respectively, both initialized to 1. Complete the entry and exit sections of the following codes such that the processes can update the shared variables atomically.

P1: while (true) do {	P2: while (true) do {
<entry section>	<entry section>
X = X + 10;	Y = Y + 20;
Y = Y – 20;	X = X – 10;
<exit section>	<exit section>
}	}

P1: wait(SX)
wait(SY)
X=X+10
Y=Y-20
signal(SY)
signal(SX)

P2: wait(SX)
wait(SY)
Y=Y+20
X=X-10
signal(SY)
signal(SX)

4. Consider the following pseudo-code for a process Pi, where "shared boolean flag[2]" is a variable declared in shared memory,initialized as:
flag[0] = flag[1] = FALSE;
Explain whether this code solves the critical section problem for two processes P0 and P1.

```
P (int i) { // i=0 for P0, i=1 for P1
    while (TRUE) {
      flag[i] = TRUE;
      while (flag[(i+1) % 2) == TRUE);
         < Critical Section >
         flag[i] = FALSE;
        < Remainder Section >
    }
 }
```

This solution may result in a deadlock between the two processes, where neither P0 nor P1 can proceed to enter their critical section. This will happen when P0 makes flag[0] = TRUE, and then there is a context switch. P1 makes flag[1] =

TRUE. Now both P0 and P1 will be waiting indefinitely in the while loop before entry to the critical section.

5. Consider the methods used by processes P1 and P2 for accessing their critical sections whenever needed, as given below. The initial values of shared Boolean variables S1 and S2 are randomly assigned. Does this setup guarantee mutual exclusion and progress?

Method used by P_1	**Method used by P_2**
while (S_1 == S_2); Critical Section $S_1 = S_2$	while (S_1 != S_2); Critical Section $S_2 = !S_1$

Progress is not guaranteed

6. Let m[0]…m[4] be mutexes (binary semaphores) and P[0]…P[4] be processes. Suppose each process P[i] executes the following:

```
wait(m[i]);

wait(m[(i+1) mod 4]);

……

release(m[i]);

release(m[(i+1) mod 4]);
```

(i) What is the problem with such a setup?
(ii) Which pairs of processes can execute the critical section concurrently?

7. Consider the following solution to the producer-consumer synchronization problem. The shared buffer size is N. Three semaphores empty, full and mutex are defined with respective initial values of 0, N, and 1. Semaphore empty denotes the number of available slots in the buffer for the consumer to read from. Semaphore full denotes the

number of available slots in the buffer for the producer to write to. The placeholder vavriables denoted P, Q, R & S in the code below can be assigned either empty or full. The valid semaphore operation are: wait() and signal().

Producer:	Consumer:

do{
wait(P);
wait(mutex);
//Add item to buffer
signal(mutex);
signal(Q);
}while(1);
	do{
	wait(R);
	wait(mutex);
	//Consume item from buffer
	signal(mutex);
	signal(S);
	}while(1);

P: *full*, Q: *empty*, R: *empty*, S: *full*