NAME-SHISHU Assignment 4 Reg 2020CA089

Q1 A shared variable x, initialized to zero, is operated on by four concurrent processes W, X, Y, Z as follows. Each processes W and X reads x from memory, increments by one, store it to memory, and then terminate. Each of the processes Y and Z reads x from memory, decrements by two, store it to memory, and then terminates. Each process before reading x invokes the P operation (i.e. wait) on a counting semaphore S and invokes the V operation (i.e. signal) on the semaphore S after storing x to memory. Semaphore S is initialized to two. What is the maximum possible value of x after all processes complete execution?

Process →	W	X	Y	Z
	R(x)	R(x)	R(x)	R(x)
Operation	x++	x++	x=x-2	x=x-2
	W(x)	W(x)	W(x)	W(x)

R(x) is to read x from memory, W(x) is to store x in memory.

(i) W_1 (x 0) [W is Preempted]

(ii) Y_1, Y_2, Y_3 (x-2) [Y is completed]

(iii) Z_1, Z_2, Z_3 (x-4) [Z is completed]

(iv) W_2, W_3 (x 1) [It is increments local copy of x and stores & W is completed]

(v) X_1, X_2, X_3 (x 2) [X is completed]

After all processes completed maximum value of X is 2.

Q2 The following two function P1 and P2 that share a variable B with an initial value of 2 execute concurrently. What is the number of distinct value that B can possibly takes after the execution?

```
P1() {
    C = B-1;
    B = 2*C;
}
```

```
P2() {
    D = 2*B
    B = D-1;
}
```

There are following ways that concurrent processes can followed:

```
C = B-1;  // C=1
B = 2*C;  // B=2
D = 2*B;  // D=4
B = D-1;  // B=3
```

```
C = B-1;  // C=1
D = 2*B;  // D=4
B = 2*C;  // B=2
B = D-1;  // B=3
```

```
C = B-1;  // C=1
D = 2*B;  // D=4
B = D-1;  // B=3
B = 2*C;  // B=2
```

```
D = 2*B;  // D=4
C = B-1;  // C=1
B = 2*C;  // B=2
B = D-1;  // B=3
```

```
D = 2*B;  // D=4
B = D-1;  // B=3
C = B-1;  // C=2
B = 2*C;  // B=4
```

There are 3 different value of B 2, 3 and 4.

Q3 Enabling and disabling interrupts to prevent timer interrupts ..... accuracy of the system clock.

Solution Disabling the interrupts before entering the critical section and enabling them while existing the critical section as a result, multiple threads cannot be these in critical section. The issue with the above procedure is if the interrupts are disabled for a long time. It affects the performance of the I/O system. Moreoves, if there is an ifinite loop, the interrupts are disabled permanently.

```
while (true)
    disable Interrupts();
/* Enter critical section */
/* Infinite loop in critical section */
    enable Interrupts();
```

Consequently the procedures will also affect the system clock. It means that a clock could go off by one or two clocks while waiting for an interrupt to be disabled. This could lead to a lot of lost data.

Q4 Consider the following snap shot of a system in which four resource A, B, C and D are available. The system contains a total of 6 instance of A, 4 of resource B, 4 of resource C, and 2 of resource D

	Allocation				Max				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P_0	2	0	1	1	3	2	1	1					6	4	4	2
P_1	1	1	0	0	1	2	0	2								
P_2	1	0	1	0	3	2	1	0								
P_3	0	1	0	1	2	1	0	1								

(i) Do the following problem
Compute what each might - - - - column Need

(ii) Is the system is a safe state? why or why not?

(iii) Is the system deadlocked? why or why not.

(iv) If a request from P_3 arrives for (2,1,0,0) can request be granted immediatly?

Solution

(i) We first calculate the need array as follows:

	Allocation				Max				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P_0	2	0	1	1	3	2	1	1	1	2	0	0	6	4	4	2
P_1	1	1	0	0	1	2	0	2	0	1	0	2				
P_2	1	0	1	0	3	2	1	0	2	2	0	0				
P_3	0	1	0	1	2	1	0	1	2	0	0	0				

because P_0's Need = [1,2,0,1] ≤ Available = [6,4,4,2]
P_0 can run and return its Allocation = [2,0,1,1]
to Available. The new Available = [6,4,4,2] + [2,0,1,1] = [8,4,5,3]

	Allocation				Max				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P_0					3	2	1	1					8	4	5	3
P_1	1	1	0	0	1	2	0	2	0	1	0	2				
P_2	1	0	1	0	3	2	1	0	2	2	0	0				
P_3	0	1	0	1	2	1	0	1	2	0	0	0				

Now we run P_1. After that P_1 returns its Allocation = [1,1,0,0] to Available, and hence Available = old Available = [8,4,5,3] + P_1's Allocation = [1,1,0,0] = [9,5,5,3].

	Allocation				Max				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P_0					3	2	1	1					9	5	5	3
P_1					1	2	0	2								
P_2	1	0	1	0	3	2	1	0	2	2	0	0				
P_3	0	1	0	1	2	1	0	1	2	0	0	0				

Now we can run P_2 followed by P_3.

Consequently, the system is in a safe state. Because the deadlock state is on the unsafe area and because this system in in a safe state, this system is not deadlocked.

Now get back to the original situtions:-

	Allocation				Max				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P_0	2	0	1	1	3	2	1	1	1	2	0	0	6	4	4	2
P_1	1	1	0	0	1	2	0	2	0	1	0	2				
P_2	1	0	1	0	3	2	1	0	2	2	0	0				
P_3	0	1	0	1	2	1	0	1	2	0	0	0				

If a new request P_3 arrives for [2,1,0,0], this is a mistake because P_3's Request = [2,1,0,0]. & P_3's Need = [2,0,0,0] and this request cannot be granted. see Resource-Request algorithm for the details.

Qs. Clearly justify why deadlocks cannot arise on a bounded buffer producers-consumers System.

Solution mutex = 1, full = 0, empty = n

```
producer:-do {
    // produce item
    wait(empty);
    wait(mutex);
    // place in buffer;
    signal(mutex);
    signal(full);
} while(true)
    wait(s);
while(s <= 0);
```

```
consumer:- do {
    wait(full);
    wait(mutex);
    //remove item from buffer
    signal(mutex);
    signal(empty);
    //consume item
} while(true)
    signal(s);
    s++;
```

```
    s--; //busy
         //waiting
}
```

Necessary condition for deadlock to occur are

1) Mutual Exclusion
2) Hold & wait
3) No Preemption
4) Circular wait.

As we can see producer an consumer are operating on different semaphores; hence they won't be hold and wait. All 4 condition are'nt satisfied, hence deadlock can't arise.