Question 1

Assume that there are 5 processes P0, P1, P2, P3, P4, and 4 types of resources R0 (3 instances), R1 (17 instances), R2 (16 instances), R3 (12 instances). Given the allocation and max requirement tables below, is the system in a safe state?

	Allocation Matrix				**Maximum Requirement Matrix**			
	R0	**R1**	**R2**	**R3**	**R0**	**R1**	**R2**	**R3**
P0	0	1	1	0	0	2	1	0
P1	1	2	3	1	1	6	5	2
P2	1	3	6	5	2	3	6	6
P3	0	6	3	2	0	6	5	2
P4	0	0	1	4	0	6	5	6

Question 2

If the system is in a safe state, can the following requests be granted immediately?

(a) P1 requests (1, 1, 1, 0)
(b) P1 requests (0, 1, 1, 1)
(c) P1 requests (0, 2, 1, 0)

Question 3

A shared variable x, initialized to zero, is operated on by four concurrent processes A,B,C,D as follows. Each of the processes A and D reads x from memory, increments by five, stores it to memory and then terminates. Each of the processes B and C reads x from memory, decrements by five, stores it to memory, and then terminates. Each process before reading x invokes the P operation (wait) on a counting semaphore S and invokes the V operation (signal) on the semaphore S after storing x to memory. Semaphore S is initialized to two. What is the maximum and minimum possible value of x after all processes complete execution?

Question 4

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

Find if the above code enters into Deadlock or not, If enters in which case it will enter. Which possible combinations of processes can be present in the critical section at the same time?

Question 5

It is time for elections amidst the pandemic. Consider that a polling station has one polling agent, one voting machine, and a waiting room with exactly *N* seats (placed maintaining social distancing). Voters come to this polling station to cast their votes following some rules. If a voter arrives, and finds that one voter is casting the vote and all *N* waiting room seats are occupied, he/she immediately leaves the polling station according to the rules. Otherwise, the voter sits on one of the empty waiting room seats and wait. If there is no one casting vote at a particular point of time, any one of the waiting voters can approach the agent and cast his/her vote. The agent goes to sleep if there is no voter in the polling station. If a voter comes and finds the agent sleeping, the voter wakes up the agent.

We want to simulate this situation considering that the agent and each voter is implemented as a thread/process, all of which will run concurrently. Assume that the following three semaphores and one variable are already declared and initialized as shown:

```
semaphore agentReady = 0, voterReady = 0, accessSeats = 1;
int numFreeSeats = N;
```

Every semaphore has `P()` (or, wait( )) and `V()` (or, signal( )) functions available as `<semaphore>.P()` and `<semaphore>.V()`. E.g., the semaphore `agentReady` has the functions `agentReady.P()` and `agentReady.V()` available. Fill in the blanks in the functions below to implement the agent and a voter, so as to ensure that the agent and voters are synchronized and free from deadlock. Note that each blank can have one or more `P()` or `V()` statements.

```
void agent(){
     while (true) {
          __________(i)__________
          numFreeSeats = numFreeSeats + 1;
          __________(ii)__________
          helpVoterCastVote();
          __________(iii)__________
     }
}

void voter(){
     __________(iv)__________
     if (numFreeSeats > 0) {
          numFreeSeats = numFreeSeats - 1;
          __________(v)__________
          castVote();
     }
     else {
          __________(vi)__________
          leaveWithoutCastingVote();
     }
}
```

Solution 1

	Allocation				Maximum				Need			
	R0	**R1**	**R2**	**R3**	**R0**	**R1**	**R2**	**R3**	**R0**	**R1**	**R2**	**R3**
P0	0	1	1	0	0	2	1	0	0	1	0	0
P1	1	2	3	1	1	6	5	2	0	4	2	1
P2	1	3	6	5	2	3	6	6	1	0	0	1
P3	0	6	3	2	0	6	5	2	0	0	2	0
P4	0	0	1	4	0	6	5	6	0	6	4	2
Total	2	12	14	12								
Avail	1	5	2	0								

Safety Algorithm:

Initial:
Work = Available = [1, 5, 2, 0]
Finish = [F, F, F, F, F]

Checking for P0
Need(P0) = [0, 1, 0, 0] ≤ Work = [1, 5, 2, 0]
P0 will be executed
Work = Work + Allocation(P0)
= [1, 5, 2, 0] + [0, 1, 1, 0]
= [1, 6, 3, 0]
Finish = [T, F, F, F, F]

Checking for P1
Need(P1) = [0, 4, 2, 1] ≰ Work = [1, 6, 3, 0]
P1 will not be executed

Checking for P2

Need(P2) = [1, 0, 0, 1] $\nleq$ Work = [1, 6, 3, 0]

P2 will not be executed

Checking for P3

Need(P3) = [0, 0, 2, 0] $\leq$ Work = [1, 6, 3, 0]

P3 will be executed

Work = Work + Allocation(P3)
= [1, 6, 3, 0] + [0, 6, 3, 2]
= [1, 12, 6, 2]

Finish = [T, F, F, T, F]

Checking for P4

Need(P4) = [0, 6, 4, 2] $\leq$ Work = [1, 12, 6, 2]

P4 will be executed

Work = Work + Allocation(P4)
= [1, 12, 6, 2] + [0, 0, 1, 4]
= [1, 12, 7, 6]

Finish = [T, F, F, T, T]

Checking for P1

Need(P1) = [0, 4, 2, 1] $\leq$ Work = [1, 12, 7, 6]

P1 will be executed

Work = Work + Allocation(P1)
= [1, 12, 7, 6] + [1, 2, 3, 1]
= [2, 14, 10, 7]

Finish = [T, T, F, T, T]

Checking for P2

Need(P1) = [1, 0, 0, 1] $\leq$ Work = [2, 14, 10, 7]

P2 will be executed

Work = Work + Allocation(P2)
= [2, 14, 10, 7] + [1, 3, 6, 5]

= [3, 17, 16, 12]
Finish = [T, T, T, T, T]

Thus, the system is in a safe state with the order P0, P3, P4, P1, P2.

<u>Solution 2</u>

(a) Request(P1) = [1, 1, 1, 0] $\nleq$ Need(P1) = [0, 4, 2, 1]
Hence this request cannot be granted at all

(b) Request(P1) = [0, 1, 1, 1] $\nleq$ Available = [1, 5, 2, 0]
Hence this request cannot be granted immediately

(c) Request(P1) = [0, 2, 1, 0] ≤ Need(P1) = [0, 4, 2, 1]
Request(P1) = [0, 2, 1, 0] ≤ Available = [1, 5, 2, 0]
Hence this request can be granted.

After granting request, new tables:

	Allocation				**Maximum**				**Need**			
	R0	**R1**	**R2**	**R3**	**R0**	**R1**	**R2**	**R3**	**R0**	**R1**	**R2**	**R3**
P0	0	1	1	0	0	2	1	0	0	1	0	0
P1	1	4	4	1	1	6	5	2	0	2	1	1
P2	1	3	6	5	2	3	6	6	1	0	0	1
P3	0	6	3	2	0	6	5	2	0	0	2	0
P4	0	0	1	4	0	6	5	6	0	6	4	2
Total	2	14	15	12								
Avail	1	3	1	0								

Safety Algorithm:

Initial:
Work = Available = [1, 3, 1, 0]
Finish = [F, F, F, F, F]

Checking for P0
Need(P0) = [0, 1, 0, 0] ≤ Work = [1, 3, 1, 0]
P0 will be executed
Work = Work + Allocation(P0)
= [1, 3, 1, 0] + [0, 1, 1, 0]
= [1, 4, 2, 0]
Finish = [T, F, F, F, F]

Checking for P1
Need(P1) = [0, 2, 1, 1] ≰ Work = [1, 4, 2, 0]
P1 will not be executed

Checking for P2
Need(P2) = [1, 0, 0, 1] ≰ Work = [1, 4, 2, 0]
P2 will not be executed

Checking for P3
Need(P3) = [0, 0, 2, 0] ≤ Work = [1, 4, 2, 0]
P3 will be executed
Work = Work + Allocation(P3)
= [1, 4, 2, 0] + [0, 6, 3, 2]
= [1, 10, 5, 2]
Finish = [T, F, F, T, F]

Checking for P4
Need(P4) = [0, 6, 4, 2] ≤ Work = [1, 10, 5, 2]
P4 will be executed

Work = Work + Allocation(P4)
= [1, 10, 5, 2] + [0, 0, 1, 4]
= [1, 10, 6, 6]
Finish = [T, F, F, T, T]

Checking for P1

Need(P1) = [0, 2, 1, 1] ≤ Work = [1, 10, 6, 6]
P1 will be executed
Work = Work + Allocation(P1)
= [1, 10, 6, 6] + [1, 4, 4, 1]
= [2, 14, 10, 7]
Finish = [T, T, F, T, T]

Checking for P2

Need(P1) = [1, 0, 0, 1] ≤ Work = [2, 14, 10, 7]
P2 will be executed
Work = Work + Allocation(P2)
= [2, 14, 10, 7] + [1, 3, 6, 5]
= [3, 17, 16, 12]
Finish = [T, T, T, T, T]

Thus, the system is in a safe state with the order P0, P3, P4, P1, P2. Hence the request from P1 can be granted safely.

Solution 3

Clearly processes A and D increments the value of x and Processes B and C decrements the value of x.

To obtain the maximum value of x, the processes must execute in such a way that-

- Only the impact of the processes A and D remains on the value of x.
- The impact of processes B and C gets lost on the value of x.

This can be done as follows:

- First of all, make the process A read the value of x = 0.
- Then, preempt the process A.
- Now, schedule process B and process C to execute one by one.
- After executing them, reschedule process A.
- Now, when process A gets scheduled again, it starts with value x = 0 and increments this value by five.
- This is because before preemption it had read the value x = 0.
- The updates from the processes B and C get lost.
- Later, execute process D which again increments the value of x by five.

To obtain the minimum value of x, the processes must execute in such a way that-

- Only the impact of the processes B and C remains on the value of x.
- The impact of processes A and D gets lost on the value of x.

Solution 4

Given processes have to execute the wait operations as-

- Process P_0 : wait(m[0]); wait(m[1]);
- Process P_1 : wait(m[1]); wait(m[2]);
- Process P_2 : wait(m[2]); wait(m[3]);
- Process P_3 : wait(m[3]); wait(m[0]);
- Process P_4 : wait(m[4]); wait(m[1]);

Consider a case where all processes get preempted after completion of the first wait in each process then it will reach a circular wait state and reach a Deadlock state.

Only the processes which do not have any mutex in common (in the first two wait statements) can be present in the CS at the same time. These combinations are:

- P0 (waiting on m[0], m[1]) and P2 (waiting on m[2] and m[3])
- P1 (waiting on m[1], m[2]) and P3 (waiting on m[3] and m[0])
- P2 (waiting on m[2], m[3]) and P4 (waiting on m[4] and m[1])
- P3 (waiting on m[3], m[0]) and P4 (waiting on m[4] and m[1])

Solution 5

(i) `voterReady.P(); accessSeats.P();`

(ii) `accessSeats.V();`

(iii) `agentReady.V();`

(iv) `accessSeats.P();`

(v) `accessSeats.V(); voterReady.V(); agentReady.P();`

(vi) `accessSeats.V();`