Mohamed Boudiaf University – M'sila
Faculty of Mathematics and Computer Science
Department of Computer Science

3nd year degree in Computer Science (ISIL)
Academic Year 2025-2026

Operating Systems II
Duration: 2h

SOLUTION FOR FINAL EXAM

(Normal Session)

Exercise 1 (6 points)

Consider a system with 3 processes P1, P2, P3, and 4 types of resources: 4 instances of R1, 2 instances of R2, 3 instances of R3, and 1 instance of R4. The state of the system is represented as follows:

ALLOCATION

	R1	R2	R3	R4
P1	0	0	1	0
P2	2	0	0	1
P3	0	1	2	0

MAX

	R1	R2	R3	R4
P1	2	0	1	1
P2	3	0	1	1
P3	2	2	2	0

1) If we impose the following protocol: each process that requests an additional resource must first release the resources it already holds. Which deadlock handling method does this protocol belong to? What is its disadvantage? **(1.5 pts)**

- This protocol belongs to the deadlock prevention method; it violates the hold-and-wait condition. (0.75 pts)
- This protocol cannot be applied in practice because a process may need multiple resources to complete a single operation, which can lead to starvation. (0.75 pts)

2) Determine whether the system is in a safe state or not. **(2 pts)**

NEED = MAX – ALLOCATION (0.5 pts)

	R1	R2	R3	R4
P1	2	0	0	1
P2	1	0	1	0
P3	2	1	0	0

AVAILABLE = Total instances – Current allocation (0.5 pts)

R1	R2	R3	R4
2	1	0	0

Work

Steps	R1	R2	R3	R4
S 0	2	1	0	0
S 1	2	2	2	0
S 2	4	2	2	1
S 3	4	2	3	1

Finish

P1	P2	P3
F	F	F
F	F	**T**
F	**T**	**T**
T	**T**	**T**

The system is in a safe state: the sequence P3, P2, P1 is safe. (1 pt)

3) Determine whether the following requests can be satisfied or not, and justify your answer: **(1.5 pts)**

   - P1 requests 2 instances of R1 and 1 instance of R4 (0.75 pt)

The request R4: (2, 0, 0, 1)
(2, 0, 0, 1) ≤ NEED[P1], but (2, 0, 0, 1) > AVAILABLE.
Therefore, this request cannot be satisfied and P1 must wait (blocked).

   - P3 requests 1 instance of R1 and 2 instances of R2 (0.75 pt)

The request R2: (1, 2, 0, 0)
(1, 2, 0, 0) > NEED[P3]. which is an error because P3 requests more than its need.
Therefore, this request cannot be satisfied.

4) Two new processes are added to the system: **P4** and **P5**, where **P4** has **Allocation = (0,0,0,0)** and **Need = (0,1,1,1),** and **P5** has **Allocation = (1,1,0,0)** and **Need = (1,1,1,0).** Using the Deadlock Detection Algorithm, determine whether the system is in a deadlocked state? If so, identify the processes involved in the deadlock. **(1 pt)**

Work (0.5 pt)

Steps	R1	R2	R3	R4
0	1	0	0	0

Finish (0.5 pt)

P1	P2	P3	P4	P5
F	F	F	**T**	F

Allocation (P4) =(0,0,0,0) so Finish[4] = True
There is no Pi where Need [i] <= Work, so, the system is in a deadlock, the processes involved are: P1, P2, P3 and P5

Exercise 2 (6 pts)

Consider three processes P1, P2, and P3 running in parallel. Process P1 produces items and puts them into a buffer B1 of size N1. Process P2 takes the items from B1, processes them, and then puts the processed items into a buffer B2 of size N2. Process P3 takes the items from B2 and consumes them. Access to each buffer can only be done by one process at a time.

```
Process P1
begin
   repeat
      ProduceItem();
      DepositItem(B1);
   until false;
end;
```

```
Process P2
begin
   repeat
      RemoveItem(B1);
      ProcessItem();
      DepositItem(B2);
   until false;
end;
```

```
Process P3
begin
   repeat
      RemoveItem(B2);
      ConsumeItem();
   until false;
end;
```

1. Using semaphores, modify the source code of processes P1, P2, and P3 to ensure the operation described above.

Solution with semaphores (4 pts)

```
Mutex1, Mutex2: Semaphore := 1;
Empty1 : Semaphore := N1;
Empty2 : Semaphore := N2;
Full1, Full2 : Semaphore := 0;
```

Process P1; (1 pts)

```
Begin
  Repeat
    ProduceItem();
    P(Empty1);
    P(Mutex1);
      DepositItem(B1);
    V(Mutex1);
    V(Full1);
  until false;
End;
```

Process P2; (2 pts)

```
Begin
  Repeat
    P(Full1);
    P(Mutex1);
      RemoveItem(B1);
    V(Mutex1);
    V(Empty1);
    ProcessItem();
    P(Empty2);
    P(Mutex2);
      DepositItem(B2);
    V(Mutex2);
    V(Full2);
  until false;
End;
```

Process P3; (1 pts)

```
Begin
  Repeat
    P(Full2);
    P(Mutex2);
    RemoveItem(B2);
    V(Mutex2);
    V(Empty2);
    ConsumeItem();
  until false;
End;
```

2. Instead of semaphores, can a lock mechanism be used to ensure all synchronization constraints in this problem? Justify your answer. **(2 pts)**

No. A lock mechanism alone is not sufficient because it only ensures **mutual exclusion**, meaning that only one process at a time can access the buffer. However, it does not provide any way to coordinate the **state of the buffer** (such as full or empty conditions). Since the lock is owned by a single process at a time, other processes cannot use it to signal or wait for changes, so proper synchronization between producer and consumer cannot be guaranteed.

Exercise 4 (8 points)

In a small post office, it is forbidden to have more than **6 customers** inside at the same time:

1. Each customer, as soon as they arrive, immediately joins a waiting queue outside the post office.
2. An agent at the door lets customers enter while respecting the maximum limit of 6 customers, then waits until all entered customers leave before allowing the next group to enter.
3. Inside the post office, only two (2) customers can be served at the same time.

Assuming there is always an **unlimited number** of waiting customers, give the **Customer(i)** and **Agent** processes synchronized using either a Semaphore or a Monitor mechanism, **of your choice** (do not forget the Main program).

Semaphore solution

```
Program PostOffice;

Var
   Mutex : semaphore := 1;          // protects the variable Count
   CustomerWaiting : semaphore := 0;  // blocks customers outside the post office
   AgentWaiting : semaphore := 0;     // blocks the agent until the customers inside leaves
   Inside : semaphore := 2;       // manages the waiting line inside the post office
   Count: integer;
```

```
Process Customer(i)
Begin
   P(CustomerWaiting);    (0.5 pt)
   P(Mutex); (0.5 pt)
      Count++;  (0.5 pt)
   V(Mutex); (0.5 pt)
   EnterPostOffice();
   P(Inside);         (0.5 pt)
   GetServed();
   V(Inside);     (0.5 pt)
   P(Mutex);    (0.5 pt)
   Count--;        (0.5 pt)
   if (Count = 0) then V(AgentWaiting);  (1 pt)
   V(Mutex);  (0.5 pts)
End;
```

```
Process Agent
Begin
   Repeat      (0.5 pt)
      for i = 1 to 6  do V(CustomerWaiting); (1 pt)
      P(AgentWaiting);    (0.5 pt)
   Until false;
End;
```

```
Begin (0.5 pt)
   Count := 0;

   ParBegin
      Customer1; Customer2; ... CustomerN;
      Agent;
   ParEnd;
End.
```

Monitor solution

```
Program PostOffice;
```

```
Process Customer(i); (0.25 pts)
Begin
   postOfficeManagement.entry();
   beServed();
   postOfficeManagement.exit();
End;
```

```
Process Agent; (0.25 pts)
Begin
   postOfficeManagement.work();
End;
```

```
Monitor postOfficeManagement;
Var
   A, C, P : condition; // A for the Agent, C for customers and P for the counters
   inside : integer; //  customers inside the post office
  count : integer;  // customers waiting at the counter
```

```
Procedure work()
Begin
   Repeat (0.5 pts)
      For i := 1 to 6 do C.signal;   (1 pts)
      A.wait;   (0.5 pts)
   Until false;
End;
```

```
Procedure entry()
Begin
   C.wait;    (0.5 pts)
   inside++;    (0.5 pts)
   If (count == 2) then P.wait;  (1 pts)
   count++;    (0.5 pts)
End;

Procedure exit()
Begin
   count - -;     (0.5 pts)
   P.signal;     (0.5 pts)
   inside - -;      (0.5 pts)
   If (inside == 0) then A.signal; (1 pts)
End;
```

```
Begin (0.25 pts)
   inside := 0;
   count := 0;
End;
```

```
Begin    (0.25 pts)
   Parbegin
      Customer1; Customer2; …… Customeri;
      Agent;
   Parend;
End;
```