Revision Questions

1. Consider the set of processes with arrival time (in milliseconds), CPU burst time (in milliseconds), and priority shown below: (Higher number represents lower priority)

Process ID	Arrival Time	Burst Time	Priority
P1	0	4	4
P2	1	3	3
P3	2	1	5
P4	3	5	1
P5	4	2	2

By giving the gantt chart in each case, calculate the throughput, average response time, average waiting time, average turn around time, in the following CPU sheduling policies

i) Preemptive Shortest Job First

ii) Preemptive Priority

iii) Round Robin with quantum time 2ms

iv) First Come First Serve

2. Imagine a hypothetical barbershop with one barber, one barber chair, and a waiting room with *n* chairs (*n* may be 0) for waiting customers. The following rules apply:

   - If there are no customers, the barber falls asleep in the chair
   - A customer must wake the barber if he is asleep
   - If a customer arrives while the barber is working, the customer leaves if all chairs are occupied and sits in an empty chair if it's available
   - When the barber finishes a haircut, he inspects the waiting room to see if there are any waiting customers and falls asleep if there is none.

There are two main complications. First, there is a risk that a race condition, where the barber sleeps while a customer waits for the barber to get him/her for a haircut arises because all of the actions—checking the waiting room, entering the shop, taking a waiting room chair—take a certain amount of time. Specifically, a customer may arrive to find the barber cutting another customer hair so he/she returns to the waiting room to take a seat but while walking back to the waiting room the barber finishes the haircut and goes to the waiting room, which he finds empty (because the customer walks slowly or went to the restroom) and thus goes to sleep in the barber chair. Second, another problem may occur when two customers arrive at the same time when there is only one empty seat in the waiting room and both try to sit in the single chair; only the first person to get to the chair will be able to sit.

Propose a solution to this problem that coordinates the actions of the customers and the barber in a way that avoids

3. Consider a system with five processes (P1 to P5) and three resource types (A, B, and C). The maximum resource requirements and current allocation for each process are provided in the table below:

Process	**Max(A, B, C)**	**Allocation(A, B, C)**
P1	(7, 5, 3)	(0, 1, 0)
P2	(3, 2, 2)	(2, 0, 0)
P3	(9, 0, 2)	(3, 0, 2)
P4	(2, 2, 2)	(2, 1, 1)
P5	(4, 3, 3)	(0, 0, 2)

The system currently has available resources: (3, 3, 2).

i) Apply the Banker's algorithm to determine if the system is in a safe state. Show the sequence of resource allocation and deallocation that satisfies all processes' resource requirements without leading to deadlock.

ii) If the system is in a safe state, determine the safe sequence of processes.

iii) If the system is not in a safe state, explain why it is unsafe and identify the processes causing the deadlock. How will you efficiently recover from the deadlock

4. Linux Fundamental

1. Create a directory named "exam" in your home directory. Inside this directory, create three subdirectories named "task1", "task2", and "task3". Create an empty text file named "notes.txt" inside "task1" directory.

```
mkdir ~/exam
cd ~/exam
mkdir task1 task2 task3
touch task1/notes.txt
```

2. In the "task2" directory, create a file named "numbers.txt" containing numbers from 1 to 10, each on a new line. Append the numbers from 11 to 20 to the same file without overwriting the existing content.

```
cd ~/exam/task2
seq 1 10 > numbers.txt
seq 11 20 >> numbers.txt
```

3. Set the permissions of "notes.txt" in "task1" directory to be readable, writable, and executable by the owner only, and readable by the group and others.

```
chmod 744 ~/exam/task1/notes.txt
```

4. Inside the "task3" directory, create a file named "data.txt" containing the text **"This is a sample text containing Linux.".** Search for the word "Linux" in the file and replace it with "Open Source". Save the changes.

```
cd ~/exam/task3

echo "This is a sample text containing Linux." > data.txt

sed -i 's/Linux/Open Source/g' data.txt
```

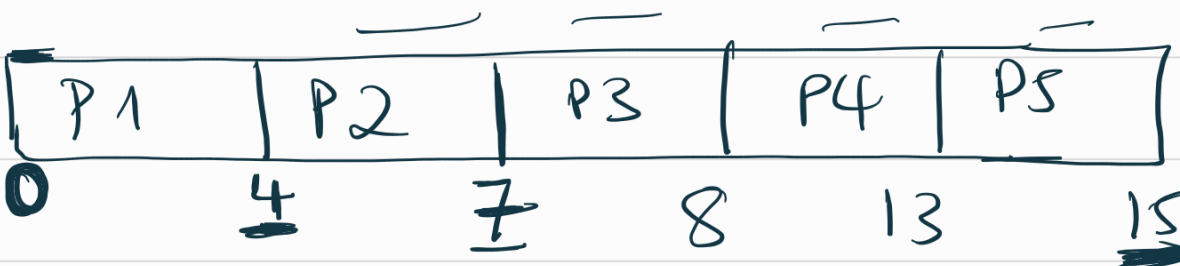
Why is the response time the same as the waiting time?

No Preemption

	AT	BT	Priority	CT	TAT	WT	RT
✓ P1	0	4	4	4	4	0	0
✓ P2	1	3	3	7	6	3	3
✓ P3	2	1	5	8	6	5	5
✓ P4	3	5	1	13	10	5	5
✓ P5	4	2	2	15	11	9	9

Averages

iv) First Come First Serve



Throughput = $\frac{\text{Total n}^{\text{o}} \text{ of Processes}}{\text{Total time taken}}$

$= 5/15 = 1/3$

$= 33.33\%$

Turn around time = Completion time − Arrival time

$= CT - AT$

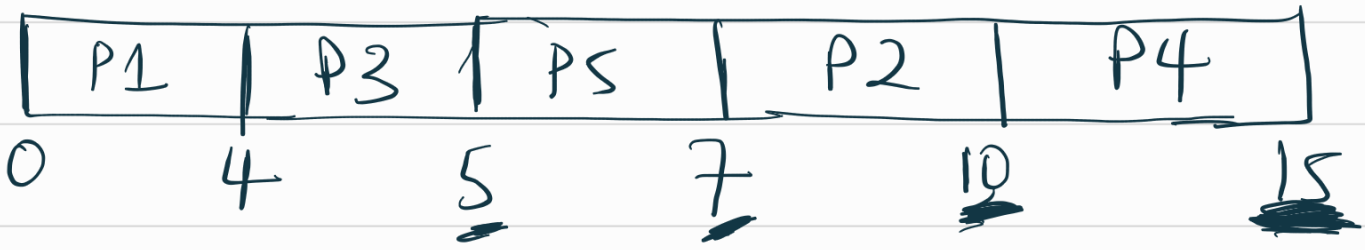
Waiting time = $TAT - BT$

Why is waiting time = Response time?

Response time = Start time − Arrival time

v) Non Preemptive Shortest Job First ✓

	AT	BT	CT	TAT	WT	RT
P1	0	4	4	4	0	0
P2	1	3	10	9	6	6
P3	2	1	5	3	2	2
P4	3	5	15	12	7	7
P5	4	2	7	3	1	1



Waiting time = Response time?

Round Robin ✓

vi)

	AT	BT	CT	TAT	WT	RT	
P1	0	4	7	7	3	0	
P2	1	3	12	11	8	1	
P3	2	1	5	3	2	2	✓
P4	3	5	15	12	7	4	
P5	4	2	11	7	5	5	✓

P1	P2	P3	P1	P4	P5	P2	P4	P4

0 2 4 5 7 9 11 12 14 15

Waiting time → Response time?

P3 and P5 were not preempted

Preemptive Priority

	AT	BT	Priority	CT	TAT	WT	RT
P1	0	4	4	14	14	10	0
P2	1	3	3	11	10	7	0
P3	2	1	5	15	13	12	12 ✓
P4	3	5	1	8	5	0	0 ✓
P5	4	2	2	10	6	4	4 ✓

P1	P2	P2	P4	P4	P5	P2	P1	P3

0 1 2 3 4 8 10 11 14 15

P3, P4 and P5 were not preempted

Need
7 4 3

Exercice 3:

Process	Max(A,B,C)	Allocation(A,B,C)
P1	(7,5,3)	(0,1,0)
P2	(3,2,2)	(2,0,0)
P3	(9,0,2)	(3,0,2)
P4	(2,2,2)	(2,1,1)
P5	(4,3,3)	(0,0,2)

Available: (3,3,2)

i) Need Matrix = Max − Allocation

$$= \begin{pmatrix} 7 & 4 & 3 \\ 1 & 2 & 2 \\ 6 & 0 & 0 \\ 0 & 1 & 1 \\ 4 & 3 & 1 \end{pmatrix}$$

P1 ✓
P2 ✓
P3 ✓
P4 ✓
P5 ✓

Initialize Work = Available, Finish[i] = false

$Work_1 = (3, 3, 2)$

Finish[i] =

P1	~~F~~ T
P2	~~F~~ T
P3	F
P4	~~F~~ T
P5	F

<P2, P4, P1,

Find an i such that

(a) Finish[i] = false

(b) Need ≤ Work

Work = Work + Allocation

Needed Available

- For P1 $(7, 4, 3) \leq (3, 3, 2)$ ✗ blocked
  P2 $(1, 2, 2) \leq (3, 3, 2)$ ✓ executed

$(3, 3, 2) + (2, 0, 0)$

$Work_2 = (3, 3, 2) + (2, 0, 0)$

available $= (5, 3, 2)$

- For P1 $(7, 4, 3) \leq (5, 3, 2)$ ✗ blocked
  P3 $(6, 0, 0) \leq (5, 3, 2)$ ✗ blocked
  P4 $(0, 1, 1) \leq (5, 3, 2)$ ✓ executed

$$Work_3 = (5,3,2) + (2,1,1)$$
$$= (7,4,3)$$

- For P1 $(7,4,3) \leq (7,4,3)$ ✓ executed

$$Work_4 = (7,4,3) + (0,1,0)$$
$$= (7,5,3)$$

- For P3 $(6,0,0) \leq (7,5,3)$ ✓ executed

$$Work_5 = (7,5,3) + (3,0,2)$$
$$= (10,5,5)$$

- For P5 $(4,3,1) \leq (10,5,5)$ ✓ executed

Safe Sequence : $<P2, P4, P1, P3, P5>$

$<P4, P2, P1, P3, P5>$

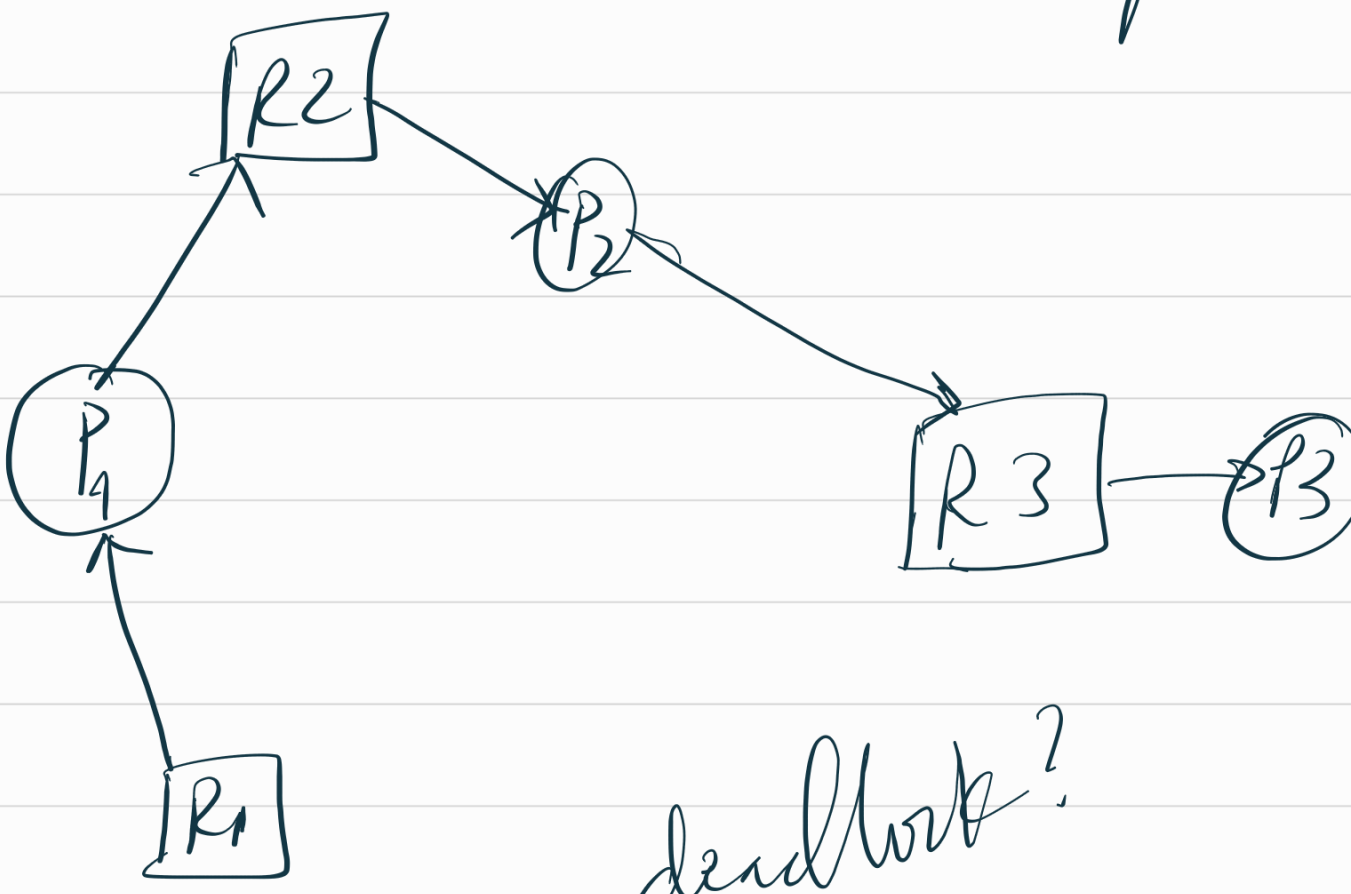
Exercice 2 :

Signal ( ) Wait ( )

Counting Semaphore, Binary Semaphore (mutex)

Semaphores
Waiting_customers (initial value = 0)

Resource Allocation Graph



deadlock?

Safe Sequence <P3, P2, P1>

To solve the classic barbershop problem with one barber, one barber chair, and a waiting room with \(n \) chairs, we can use synchronization mechanisms such as semaphores to coordinate the actions of the barber and the customers. The goal is to ensure mutual exclusion, avoid race conditions, and properly handle the interactions between the barber and the customers.

Here's a step-by-step solution using semaphores:

Definitions
1. **Semaphores**:
   - `waiting_customers` (initial value = 0): Counts the number of waiting customers.
   - `barber_ready` (initial value = 0): Indicates whether the barber is ready to cut hair.
   - `mutex` (initial value = 1): Ensures mutual exclusion when accessing shared resources like the waiting room chairs.
2. **Counters**:
   - `waiting_chairs`: A counter to keep track of available chairs in the waiting room.

Pseudocode
Shared Resources Initialization
```python
waiting_customers = Semaphore(0)
barber_ready = Semaphore(0)
mutex = Semaphore(1)
waiting_chairs = n  # Number of waiting room chairs
```

Barber Process
```python
def barber():
    while True:
        waiting_customers.acquire()  # Wait for a customer to arrive
        mutex.acquire()  # Ensure mutual exclusion
        waiting_chairs += 1  # One waiting chair becomes free
        barber_ready.release()  # Signal that the barber is ready
        mutex.release()  # Release mutual exclusion
        cut_hair()  # Perform the haircut
```

Customer Process
```python
def customer():
    mutex.acquire()  # Ensure mutual exclusion
    if waiting_chairs > 0:
        waiting_chairs -= 1  # Take a waiting chair
        waiting_customers.release()  # Notify the barber
        mutex.release()  # Release mutual exclusion
        barber_ready.acquire()  # Wait for the barber to be ready
        get_haircut()  # Get the haircut
    else:
        mutex.release()  # Release mutual exclusion and leave if no chair is available
```

Explanation

1. **Barber Process**:
   - The barber waits for a customer by calling `waiting_customers.acquire()`.
   - When a customer arrives and releases `waiting_customers`, the barber gains access and decrements the semaphore.
   - The barber then acquires the mutex to ensure mutual exclusion and increases the count of available waiting chairs.
   - The barber releases `barber_ready` to signal that he is ready to cut hair and releases the mutex.
   - Finally, the barber performs the haircut with `cut_hair()`.

2. **Customer Process**:
   - A customer arrives and acquires the mutex to ensure mutual exclusion.
   - If there is an available waiting chair, the customer takes a chair (decrementing `waiting_chairs`) and releases `waiting_customers` to notify the barber.
   - The customer then releases the mutex and waits for the barber to be ready by calling `barber_ready.acquire()`.
   - Once the barber is ready, the customer gets the haircut with `get_haircut()`.
   - If no chairs are available, the customer releases the mutex and leaves.

Additional Considerations
- **Atomic Operations**: Ensure that semaphore operations and counter updates are atomic to prevent race conditions.
- **Fairness**: Ensure that semaphore operations are fair so that customers are served in the order of their arrival.

- **Handling Edge Cases**: Consider edge cases such as multiple customers arriving simultaneously and ensure the implementation handles these correctly without deadlocks or starvation.

By using semaphores and proper synchronization, this solution ensures that the barber and customers coordinate their actions correctly, avoiding race conditions and ensuring smooth operation of the barbershop.