[CLO-3] Evaluate the commonly used mechanisms for scheduling of tasks and implement synchronization mechanisms like Semaphores, TSL, etc. **[Marks: 20]**

Q4:

A system implements a Multi-Level Feedback Queue (MLFQ) scheduler with the following characteristics:

- **Queue 1** (Highest Priority): Time quantum = 3 ms, Round Robin.
- **Queue 2** (Medium Priority): Time quantum = 6 ms, Round Robin.
- **Queue 3** (Lowest Priority): FCFS (First-Come, First-Served).

At time **t = 0 ms**, five processes arrive in the system with the following burst times and priority boost intervals:

Process	Burst Time (ms)	Arrival Time (ms)	Priority Boost Interval (ms)
P1	10	0	20
P2	15	0	30
P3	8	0	16
P4	20	0	45
P5	17	0	24

Rules:

1. All processes start in **Queue 1**.
2. If a process does not finish within its time quantum, it moves to the next lower-priority queue.
3. After every **Priority Boost Interval** (calculated from the arrival time of the process), the process is **immediately moved and placed at the end of Queue 1** (if the process is not using CPU). **[e.g. for P1, 1st booster at 20 , 2nd at 40 , 3rd at 60 and so on. Same goes for others]**
4. If a process is currently executing (using CPU) in **Queue 2** or **Queue 3** when its priority boost interval is reached:
   - The process will **not surrender its current time slice** in Queue 2 but will be boosted to Queue 1 right after its time slice finishes.
   - The process will be **interrupted mid-execution in Queue 3 (FCFS)** and moved to Queue 1 at the end of the queue 1.
5. Processes in the same queue are scheduled based on the respective queue's scheduling policy. Context switching time is ignored.

Tasks:

1. Draw the Gantt chart for the schedule.
2. Calculate the turnaround time (TAT) and waiting time (WT) for each process and average.

1. Randomly select a warehouse section.
2. Randomly choose to add or remove stock (50% probability for each).
3. Perform the operation using **addStock()** or **removeStock()** and print the transaction details.

```
#define NUM_WORKERS 5

void* worker(void* arg) {
//Write your code here
}
```

4. Synchronization

- Use semaphores to ensure that only one thread can access a warehouse section's stock at a time.
- Semaphores ensure there is no race condition or inconsistent state.

Basic Code Structure for Main Function:

```
int main()
{

pthread_t workers[NUM_WORKERS];

int worker_ids[NUM_WORKERS];

// Initialize warehouse

// Create worker threads

// Print final stock

}
```

Sample Output:

```
Added 7 to Section 2. Current Stock: 107
Removed 4 from Section 3. Current Stock: 96
Removed 3 from Section 0. Current Stock: 97
Added 8 to Section 4. Current Stock: 108
...

Final Stock in Warehouse Sections:
Section 0: 102 items
Section 1: 100 items
Section 2: 105 items
Section 3: 98 items
Section 4: 110 items
```

[CLO-4] Deploy OS tools related related to Virtualization and Containers

Q6: **[7+2+2+4=15 Marks]**

Part A:

Consider a single-level paging scheme where:

- The virtual address space is 512MB.
- The page table entry size is 8 bytes.

What is the minimum page size such that the entire page table fits in one page? For the page table to fit within a single page, the following condition must be satisfied:

Page Table Size ≤ Page Size

Part B:

Consider a memory system with the following properties:

- The address length is 30 bits.
- The memory is 2-byte addressable (each address refers to 2 bytes of memory).

What will be the size of the Memory?

Part C:

A computer system has 40-bit virtual address space with a page size of 16K, and 8 bytes per page table entry.

1. How many pages are in the virtual address space?
2. What is the maximum size of the addressable memory in this system? (**Hint:** you can ignore protection bits)

[CLO-4] Deploy OS tools related related to Virtualization and Containers

Q7: **[10 Marks]**

You are given a following **page reference string** for a single process:

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Each page in the reference string has a **criticality factor**, which determines its **cost** during a page fault. The **criticality factor** of a page can be one of the following:

C > 5

- **Critical pages (C)**: Have a **cost of 5 units** per page fault.
- **Non-critical pages (N)**: Have a **cost of 2 units** per page fault.

NC > 2

Page	Criticality Factor
7	C
0	N
1	N
2	C
3	N
4	C
5	N
6	C

Memory starts with 4 empty frames. Your task is to apply **FIFO** and **LRU** Page replacement Algorithms and calculate the total number of page faults and the total cost for each algorithm.

10, 35

9, 33

[CLO-3] Evaluate the commonly used mechanisms for scheduling of tasks and implement synchronization mechanisms like Semaphores, TSL, etc.

[15 Marks]

Q5:

Scenario

In a factory, there is a shared machine that is used for producing goods. The machine can either be operated by workers to produce the goods or undergo maintenance by engineers.

- **Production Workers:** These workers operate the machine to produce goods. They only use the machine to create products, without modifying its internal structure.
- **Engineers:** These workers perform maintenance on the machine, which includes checking its components, fixing issues, or upgrading its software or mechanical parts. During maintenance, the machine cannot be used for production.

Here are the rules for the factory system:

1. **Multiple Production Workers Rule:** If there are several workers using the machine at the same time, they can all work without issue, as long as no engineers are working on the machine.

2. **Maintenance Rule:** If an engineer is performing maintenance on the machine, no one else can use it. This means that no workers (production or maintenance i.e. other engineers) are allowed to access the machine until the engineer finishes their work.

3. **Priority for Engineers:** If an engineer is waiting to perform maintenance on the machine, they must be given priority. If there are any workers currently using the machine, they must finish their work, and no new workers can start operating the machine until the engineer begins the maintenance.

4. **No New Workers When an Engineer is Waiting:** If an engineer is waiting for the machine, no new workers are allowed to start using the machine, even if some workers are already working on it. Once all current workers finish, the engineer can perform maintenance.

Task:

Write a solution for the system using synchronization mechanism (**semaphores**) to manage access to the shared machine ensuring that engineers are given priority and that production workers can continue using the machine if no engineers are waiting. **To receive full credit, you are required to explicitly initialize all semaphores and variables in your solution.**

Production Workers	Engineers

Worker() {
 	
<CRITICAL Section> //Production started
 	
}
	Engineer() {
	
	<CRITICAL Section> // Maintenance started
	}

[2+3+3+2=10 Marks]

Q8:

A resource allocation system that uses the Banker's algorithm for 4 resource types (A, B, C, D) and 5 users (P0, P1, P2, P3 P4) is currently in the following state.

Processes	Allocation (A B C D)	Max (A B C D)	Available (A B C D)
P_0	2001	4212	3321
P_1	3121	5252	
P_2	2103	3216	
P_3	1312	4244	
P_4	1432	3665	

9,9,6,9

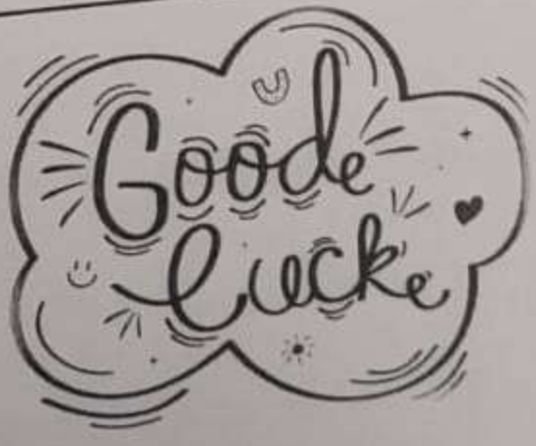
3,3,2,1

(12,12,8,10)

Using Banker's algorithm, answer the following questions:

a. How many resources of type A, B, C and D are there in the system?
b. What are the contents of the **need** matrix?
c. Is the system in a safe state? Why? No
d. If the request from P4 arrives for (0, 0, 2, 0), can the request be granted immediately? No

2,2,1,1
2,1,3,1
1,1,1,3
3,-1,3,2
2,2,3,3



National University of Computer and Emerging Sciences 0348
Lahore Campus

Operating Systems (CS2006)

Date: DEC 23th 2024

Course Instructor(s)

Mr. Razi Uddin
Mr. Mubashar Hussain
Ms. Rubab Anam
Ms. Namra Absar

Final Exam

Total Time (Hrs):	3
Total Marks:	95
Total Questions:	8

Name: Hasan Yahya

22L-7971 Roll No

BSE-5B Section

Student Signature

Do not write below this line

Attempt all the questions in sequence.

[CLO-1] Describe services provided by the modern Operating Systems

Q1: **[Marks:5]**

How operating system use hardware mechanisms like the Memory Management Unit (MMU) to convert logical addresses to physical addresses, ensuring memory protection and efficient resource management. Give an example.

[CLO-2] Implement solutions employing concepts of Processes and Threads

Q2: **[Marks:5]**

Consider the following C code:

```
int main() {
    int x = 3, y = 2;

    if ((x + y) % 2 == 1) {     // Condition 1
        fork();                 // Fork A
    }
    for (int i = 0; i < x; i++) { // Loop 1
        if (fork() == 0) {      // Fork B
            break;
        }
    }
    if (x * y > 5 && fork()) { // Condition 2 and Fork C
```

23 → C
P → 2

```
        fork();                    // Fork D
    }
    return 0;
}
```

Draw the complete fork tree generated by this program. Label each process with its parent-child relationship (e.g., P, C1, C2, etc.). Clearly indicate how the loop and conditions affect process creation.

[CLO-2] Implement solutions employing concepts of Processes and Threads

Q3: Warehouse Stock Management with Threads and Semaphores **[15 Marks]**

Requirements:

1. **Warehouse Initialization:**

- Define an array of NUM_SECTIONS warehouse sections. Each section will have an initial stock of INITIAL_STOCK items (e.g., 100 items per section).
- Use an array of semaphores, where each semaphore corresponds to a warehouse section, to ensure thread-safe access to the stock.
- Skeleton code is given below.

```
#define NUM_SECTIONS 5
#define INITIAL_STOCK 100

int warehouse_stock[NUM_SECTIONS];   // Array to store stock in each section
sem_t section_lock[NUM_SECTIONS];    // Array of semaphores for synchronization

void initializeWarehouse()           // Initialize each semaphore
{
//Write your code here
}
```

2. **Transaction Functions**

- Implement two functions: **addStock()** and **removeStock()**.
- These functions will:
  1. Accept a section number and transaction amount as arguments.
  2. Use semaphores to ensure only one thread modifies a section's stock at a time.
  3. Ensure **removeStock()** checks for sufficient stock before decrementing.

```
void addStock(int section, int amount) {
//Write your code here
}
void removeStock(int section, int amount) {
//Write your code here
}
```

3. **Worker Threads**

- Create multiple threads (e.g., **NUM_WORKERS = 5**), where each thread represents a worker.
- Each thread will: