Revised OSL Practice Problem Statements AY 2025-26

1.1 Write a shell script to implement an address book (address.txt) that contains ID, Name, and Phone with the following functions:

1. Search Address Book
2. Add an address book entry
3. Remove an address book entry
4. Quit the program.

Suggested record format: Use semi-colon (;) to separate fields.

1.2 Write a script program to create a phonebook. Using the phonebook, perform the following tasks:

1. Add a new name to a phone book file
2. Display all matches to a name or phone number
3. Sort the phone book by the last name
4. Delete an entry

Suggested record format: Use tabs to separate fields.

1.3 Create an address book program using the bourne-again shell. It should use functions to perform the required tasks. It should be menu-based, allowing you the options of:

1. Search address book
2. Add entries
3. Remove / edit entries.

You will also need a "display" function to display a record or records when selected.

Suggested record format: Use colons to separate fields.

1.4 Create a file management program using the bourne-again shell. It should use functions to perform the required tasks. It should be menu-based, allowing you the options of:

1. Test if file exists
2. Read a file
3. Delete a file
4. Display a list of files

e.g. book.txt and test.txt files are available in present working directory. The filename = “book.txt” contents:

1. Pro AngularJS
2. Learning JQuery
3. PHP Programming

2.1 Implement the C program to accept ‘n’ integers to be sorted. Main function creates child process using fork system call. Parent process sorts the integers using **bubble sort** and waits for child process using wait system call. Child process sorts the integers using **insertion sort**. Also demonstrate zombie and orphan states.

2.2 Implement the C program in which main program accepts an integer array. Main program uses the fork system call to create a new process called a child process. Parent process sorts an integer array and passes the sorted array to child process through the command line arguments of execve system call. The child process uses execve system call to load new program that uses this sorted array for performing the binary search to search the item in the array.

2.3 Write a C program using the fork() system call that generates the prime number sequence in the child process. The number will be provided from the command line. For example, if 10 is passed as a parameter on the command line, the child process will output 2, 3, 5, 7, 11, 13, 17, 19, 23, 29. Because the parent and child processes have their own copies of the data, it will be necessary for the child to output the sequence. Have the parent invoke the wait() call to wait for the child process to complete before exiting the program. Perform necessary error checking to ensure that a positive integer is passed on the command line.

3.1 Write a menu-driven C program to simulate the following CPU scheduling algorithms to find average turnaround time and average waiting time.
a) FCFS b) SJF (non-preemptive)
Expected Output (e.g):
Enter the number of processes:3

Process	Arrival time	Burst time
p1	0	7
p2	2	4
p3	4	1
p4	5	4

Enter the choice: 1. FCFS 2. SJF (non-preemptive) 3. Exit

Average waiting time for <FCFS/SJF> is:
Average turn-around time for <FCFS/SJF> is:

GANTT CHART (e.g):

```
____________________
| p3 | p2 | p1 |
____________________
0    5    9    12
```

3.2 Write a menu-driven C program to simulate the following CPU scheduling algorithms to find average turnaround time and average waiting time.
a) FCFS b) SJF (preemptive)
Expected Output (e.g):
Enter the number of processes:3

Process	Arrival time	Burst time
p1	0	7
p2	2	4
p3	4	1
p4	5	4

Enter the choice: 1. FCFS 2. SJF (preemptive) 3. Exit
Average waiting time for <FCFS/SJF> is:
Average turn-around time for <FCFS/SJF> is:
GANTT CHART:

```
__________________
| p* | p* | p* |
__________________
0    5    9    12
```

3.3 Write a menu-driven C program to simulate the following CPU scheduling algorithms to find average turnaround time and average waiting time.
a) FCFS b) Round-robin (preemptive) with time quantum = 2 sec.

Expected Output (e.g):
Enter the number of processes:3

Process	Arrival time	Burst time
p1	0	7
p2	2	4
p3	4	1
p4	5	4

Enter the choice: 1. FCFS 2. Round-robin (preemptive) 3. Exit
Average waiting time for <FCFS/RR> is:
Average turn-around time for <FCFS/RR> is:
GANTT CHART:

```
__________________
| p* | p* | p* |
__________________
0    5    9    12
```

4.1 The problem below describes two processes, the producer and the consumer, who share a common, fixed-size buffer used as a queue.
The producer is Mr. Simpson whose job is to bake Pizza and consumer is Joey Tribbiani who consumes Pizza at the same time. Both of them share common Pizza counter to interact with each other. The problem is to make sure that the Simpson won't try to add more pizza on Pizza-counter if it's full. He needs to wait until Joey consumes Pizza. Similarly, Joey can't consume pizza from an empty counter. He needs to wait until Mr. Simpson adds Pizza on counter.
Implement a solution in C using POSIX threads that coordinates the activities of the producer Simpson and the consumer Joey with counting semaphores and mutex.

4.2 A university computer science department has a teaching assistant (TA) who helps undergraduate students with their programming assignments during regular office hours. The TA's office is rather small and has room for only one desk with a chair and computer. There are three chairs in the hallway outside the office where students can sit and wait if the TA is currently helping another student. When there are no students who need help during office hours, the TA sits at the desk and takes a nap. If a student arrives during office hours and finds the TA sleeping, the student must awaken the TA to ask for help. If a student arrives and finds the TA currently helping another student, the student sits on one of the chairs in the hallway and waits. If no chairs are available, the student will come back at a later time. Using POSIX threads, mutex locks, and semaphores, implement a solution that coordinates the activities of the TA and the students.

4.3 The Producer generates an integer between 0 and 9 (inclusive), stores it in a file. To make the synchronization problem more interesting, the Producer sleeps for a random amount of time between 0 and 100 milliseconds before repeating the number-generating cycle. The Consumer consumes all integers from the same file (the exact same file into which the Producer put the integers in the first place) as quickly as they become available. The activities of the Producer and the Consumer must be synchronized in a following way: The two threads must do some simple coordination. That is, the Producer must have a way to indicate to the Consumer that the value is ready, and the Consumer must have a way to indicate that the value has been retrieved. Implement the solution using mutex, and binary semaphore — to help threads wait for a condition and notify other threads when that condition changes.

4.4 A real-world example of the readers-writers problem is an airline reservation system:

- Readers: want to read flight information
- Writers: want to make flight reservations
- Potential problem: if readers and writers can access the shared data simultaneously then readers/writers may view flights as being available when they've actually just been booked.

Implement the solution using mutex and semaphore that controls access to the reader count and database given the priority to readers over writers.

5.1 Develop a C program to implement Banker's algorithm. Assume suitable input required to demonstrate the results. Using the banker's algorithm, determine whether or not the state is unsafe. If the state is safe, illustrate the order in which the processes may complete. Otherwise, illustrate why the state is unsafe.

5.2 Develop a C program to simulate Banker's Algorithm for Deadlock Avoidance with following requirements:

1. Demonstrate safe and unsafe state of the system?
2. Demonstrate Grant and non-grant of new resource request.

Note: Your program may read the current state of the system from a file called "state.txt". In this file, number of processes, number of resources, Allocation Matrix, Max and Available Resources Matrix may be presented in the order below:

State.txt contents:-->

```
Number of Processes     5
Number of Resources     4

3 2 0 0
0 1 1 2
2 1 0 0
0 0 1 0
2 1 1 1

3 0 1 1
0 1 0 0
1 1 1 0
1 1 0 1
0 0 0 0

1 0 2 1
```

6.1 Write a C program to simulate page replacement algorithms a) FIFO b) LRU

First, generate a random page-reference string where page numbers range from 0 to 9. Apply the random page-reference string to each algorithm and record the number of page faults incurred by each algorithm. Implement the replacement algorithms so that the number of page frames can vary from 1 to 7.

6.2 Consider the page reference string of size 12: 1, 2, 3, 4, 1, 2, 5, 1, 1, 2, 3, 4, 5 with frame size 3 and 4 respectively. Write a C program to simulate page replacement algorithms a) Optimal b) LRU.

7.1 Implement two independent processes, Sender and Receiver, using Message Queues (FIFO). The Sender accepts messages from the user and sends them to the queue with different message types. The Receiver reads each message, converts it to uppercase, and displays it on the screen. Ensure proper cleanup of message queues after execution.

7.2 Develop Client and Server processes using Shared Memory.
The Server creates a shared memory segment and writes a message to it. The Client reads the message from the shared memory segment and displays it on the screen. Include appropriate synchronization (using semaphores or sleep/wait) to avoid data inconsistency.

8.1 Implement and compare the total head movement for a given sequence of disk requests using two specific scheduling algorithms:

- System Setup: A disk with cylinders numbered 0 to 499.
- Current Head Position: 85
- Pending Requests (FIFO order): 10, 229, 39, 400, 18, 145, 120, 480, 20, 250
- Algorithms to Implement:
    - C-SCAN (Assume initial movement is towards 499).
    - C-LOOK (Assume initial movement is towards 499).
- Output: Calculate and display the sequence of cylinder movements and the Average Seek Distance for both C-SCAN and C-LOOK. Conclude which algorithm performed better for this request set.

8.2 Implement and compare the total head movement for a given sequence of disk requests using two specific scheduling algorithms:

- System Setup: A disk with cylinders numbered 0 to 499.
- Current Head Position: 185
- Pending Requests (FIFO order): 20, 229, 39, 450, 18, 145, 120, 380, 20, 250
- Algorithms to Implement:
    - SCAN (Assume initial movement is towards 499).
    - LOOK (Assume initial movement is towards 499).
- Output: Calculate and display the sequence of cylinder movements and the Average Seek Distance for both SCAN and LOOK. Conclude which algorithm performed better for this request set.

General Oral Questions

- What is the race condition?
- What is a binary semaphore?
- What is the difference between Semaphore and Mutex.
- Explain the concept of semaphore?
- Explain wait and signal functions associated with semaphores.
- Where the binary and counting semaphores are used in the solution of classical problems.
- Is dining philosopher problem represents processes synchronization or deadlock situation?
- How to avoid deadlock in dining philosopher problem.
- How many operations Banker’s algorithm required to perform in worst case scenario?
- What is the cause of thrashing?
- Why LRU and Optimal Page Replacement algorithm doesn't suffer from Belady's anamoly.
- What is the fundamental difference between an unnamed pipe and a FIFO (named pipe) in Linux, and when would you choose one over the other?
- What is the main advantage of using Shared Memory over message passing mechanisms like pipes or message queues? What is its major drawback?
- Why is a separate synchronization mechanism (like a semaphore or mutex) absolutely essential when using shared memory?
- Define a **race condition** and provide a simple, real-world example different from the bank account simulation.
- What is the key difference between a Mutex and a counting Semaphore? Which one is better suited for protecting the bank balance variable?
- Name the four necessary and sufficient conditions for a deadlock to occur.
- Briefly distinguish between deadlock and livelock.
- Why is the Banker's Algorithm primarily a theoretical tool rather than a widely implemented feature in general-purpose operating systems?
- What are the two main performance metrics that disk scheduling algorithms aim to optimize?
- Which of the six common disk scheduling algorithms (FIFO, SSTF, SCAN, C-SCAN, LOOK, C-LOOK) is most prone to causing starvation for requests located at the edges of the disk? Explain why.
- Why does C-SCAN generally provide a more uniform waiting time than SCAN, despite often resulting in a higher total seek time?
- Based on our earlier comparison, if disk load is light and requests are clustered, why is LOOK preferred over SCAN?