1.Write a syntax and example of some UNIX command.

1. `ls`

- Syntax: `ls [options] [directory]`
- Example: `ls -l /home`
  Lists files in the `/home` directory in long format.

2. `cd`

- Syntax: `cd [directory]`
- Example: `cd /var/log`
  Changes the current directory to `/var/log`.

3. `pwd`

- Syntax: `pwd`
- Example: `pwd`
  Prints the current working directory.

4. `mkdir`

- Syntax: `mkdir [directory_name]`
- Example: `mkdir new_folder`
  Creates a new directory named `new_folder`.

5. `rm`

- Syntax: `rm [options] [file/directory]`
- Example: `rm -r old_folder`
  Removes the directory `old_folder` and its contents recursively.

6. `cp`

- Syntax: `cp [source] [destination]`
- Example: `cp file1.txt /backup/`
  Copies `file1.txt` to the `/backup/` directory.

7. `mv`

- Syntax: `mv [source] [destination]`
- Example: `mv file1.txt file2.txt`
  Renames or moves `file1.txt` to `file2.txt`.

8. `touch`

- Syntax: `touch [file_name]`
- Example: `touch new_file.txt`
  Creates an empty file named `new_file.txt`.

9. `cat`

- Syntax: `cat [file_name]`
- Example: `cat file.txt`
  Displays the content of the file.

10. `head`

- Syntax: `head [options] [file_name]`
- Example: `head -5 file.txt`
  Displays the first 5 lines of the file.

11. `tail`

- Syntax: `tail [options] [file_name]`
- Example: `tail -n 10 log.txt`
  Displays the last 10 lines of the file.

12. `grep`

- Syntax: `grep [pattern] [file_name]`
- Example: `grep "error" log.txt`
  Searches for the word "error" in the file.

13. `find`

- Syntax: `find [path] [options]`
- Example: `find /home -name "*.txt"`
  Finds all `.txt` files in `/home`.

14. `chmod`

- Syntax: `chmod [permissions] [file_name]`
- Example: `chmod 755 script.sh`
  Changes permissions for the file.

15. `ps`

- Syntax: `ps [options]`
- Example: `ps aux | grep apache2`
  Lists processes and filters for "apache2".

16. `kill`

- Syntax: `kill [PID]`
- Example: `kill 1234`
  Terminates the process with PID 1234.

17. `df`

- Syntax: `df [options]`
- Example: `df -h /home/username/Downloads/`
  Shows disk space usage in human-readable format.

2.Write a simple Unix program that uses the fork() system call in C.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
```

```
#include <unistd.h>

int main() {
    pid_t pid;

    /* Create a new process */
    pid = fork();

    if (pid < 0) {
        /* Fork failed */
        perror("fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        /* Child process */
        printf("Hello from the child process! My PID is %d\n",
getpid());
    } else {
        /* Parent process */
        printf("Hello from the parent process! My PID is %d and my
child's PID is %d\n", getpid(), pid);
    }

    return 0;
}
```

3.Unix program in C that simulates the functionality of the ls,opendir(), readdir(), and closedir() system calls command.

```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>

int main(int argc, char *argv[]) {
    struct dirent *entry;
    DIR *dir;

    // Check if directory path is provided
    if (argc < 2) {
        printf("Usage: %s <directory_path>\n", argv[0]);
        return EXIT_FAILURE;
    }

    // Open the directory
    dir = opendir(argv[1]);
    if (dir == NULL) {
        perror("opendir");
        return EXIT_FAILURE;
    }

    // Read and print directory entries
    while ((entry = readdir(dir)) != NULL) {
        printf("%s\n", entry->d_name);
    }

    // Close the directory
    closedir(dir);

    return EXIT_SUCCESS;
}
```

4.Write a Unix C program that demonstrates the usage of fork(), exec(), exit(), and close() system calls in one execution.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <fcntl.h>

int main() {
    pid_t pid;
    int status;

    // Fork a new process
    pid = fork();

    if (pid < 0) {
        perror("fork failed");
        exit(EXIT_FAILURE);
    }
    else if (pid == 0) {
        // Child process

        // Close standard output (stdout)
        close(STDOUT_FILENO);

        // Execute ls command
        execlp("ls", "ls", "-l", NULL);
```

```
        // If exec fails
        perror("exec failed");
        exit(EXIT_FAILURE);
    }
    else {
        // Parent process
        printf("Parent waiting for child process to finish...\n");

        // Wait for child to complete
        waitpid(pid, &status, 0);

        if (WIFEXITED(status)) {
            printf("Child exited with status %d\n",
WEXITSTATUS(status));
        } else {
            printf("Child terminated abnormally\n");
        }
    }

    return 0;
}
```

5.Write a program to implement the FCFS CPU Scheduling Algorithms

```
import java.util.Scanner;

public class FCFS_Scheduling {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // Get number of processes
        System.out.print("Enter the number of processes: ");
        int n = sc.nextInt();

        int processes[] = new int[n];
        int burstTime[] = new int[n];
        int waitingTime[] = new int[n];
        int turnaroundTime[] = new int[n];

        // Input burst times
        System.out.println("Enter the burst time for each process:");
        for (int i = 0; i < n; i++) {
            processes[i] = i + 1; // Process ID
            System.out.print("Process " + (i + 1) + ": ");
            burstTime[i] = sc.nextInt();
        }
        // Calculate Waiting Time
        waitingTime[0] = 0; // First process has no waiting time
        for (int i = 1; i < n; i++) {
            waitingTime[i] = waitingTime[i - 1] + burstTime[i - 1];
        }
```

```
        // Calculate Turnaround Time
        float totalWaitingTime = 0, totalTurnaroundTime = 0;
        for (int i = 0; i < n; i++) {
            turnaroundTime[i] = burstTime[i] + waitingTime[i];
            totalWaitingTime += waitingTime[i];
            totalTurnaroundTime += turnaroundTime[i];
        }

        // Display results
        System.out.println("\nProcess\tBurst Time\tWaiting
Time\tTurnaround Time");
        for (int i = 0; i < n; i++) {
            System.out.println(processes[i] + "\t" + burstTime[i] + "\t\t"
+ waitingTime[i] + "\t\t" + turnaroundTime[i]);
        }

        // Print averages
        System.out.printf("\nAverage Waiting Time: %.2f\n",
totalWaitingTime / n);
        System.out.printf("Average Turnaround Time: %.2f\n",
totalTurnaroundTime / n);

        sc.close();
    }
}
```

6. Write a program to implement the SJF CPU Scheduling Algorithms

```
import java.util.Arrays;
import java.util.Scanner;

class Process {
    int id, burstTime, waitingTime, turnaroundTime;

    Process(int id, int burstTime) {
        this.id = id;
        this.burstTime = burstTime;
    }
}

public class SJF_Scheduling {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // Get number of processes
        System.out.print("Enter the number of processes: ");
        int n = sc.nextInt();

        Process processes[] = new Process[n];

        // Input burst times
        System.out.println("Enter the burst time for each process:");
        for (int i = 0; i < n; i++) {
            System.out.print("Process " + (i + 1) + ": ");
            int burstTime = sc.nextInt();
```

```
            processes[i] = new Process(i + 1, burstTime);
        }

        // Sort processes by burst time (SJF Logic)
        Arrays.sort(processes, (p1, p2) -> p1.burstTime -
p2.burstTime);

        // Calculate Waiting Time and Turnaround Time
        processes[0].waitingTime = 0; // First process has no waiting
time
        float totalWaitingTime = 0, totalTurnaroundTime = 0;

        for (int i = 1; i < n; i++) {
            processes[i].waitingTime = processes[i - 1].waitingTime +
processes[i - 1].burstTime;
        }

        for (int i = 0; i < n; i++) {
            processes[i].turnaroundTime = processes[i].waitingTime +
processes[i].burstTime;
            totalWaitingTime += processes[i].waitingTime;
            totalTurnaroundTime += processes[i].turnaroundTime;
        }
```

```
        // Display results
        System.out.println("\nProcess\tBurst Time\tWaiting
Time\tTurnaround Time");
        for (Process p : processes) {
            System.out.println(p.id + "\t" + p.burstTime + "\t\t" +
p.waitingTime + "\t\t" + p.turnaroundTime);
        }

        // Print averages
        System.out.printf("\nAverage Waiting Time: %.2f\n",
totalWaitingTime / n);
        System.out.printf("Average Turnaround Time: %.2f\n",
totalTurnaroundTime / n);

        sc.close();
    }
}
```

7. Write a program to implement the Priority CPU Scheduling Algorithms

```
import java.util.Arrays;
import java.util.Scanner;

class Process {
    int id, burstTime, priority, waitingTime, turnaroundTime;

    Process(int id, int burstTime, int priority) {
        this.id = id;
        this.burstTime = burstTime;
        this.priority = priority;
    }
}

public class PriorityScheduling {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // Get number of processes
        System.out.print("Enter the number of processes: ");
        int n = sc.nextInt();

        Process processes[] = new Process[n];

        // Input burst times and priorities
        System.out.println("Enter Burst Time and Priority for each process:");
        for (int i = 0; i < n; i++) {
```

```
            System.out.print("Process " + (i + 1) + " Burst Time: ");
            int burstTime = sc.nextInt();
            System.out.print("Process " + (i + 1) + " Priority: ");
            int priority = sc.nextInt();
            processes[i] = new Process(i + 1, burstTime, priority);
        }

        // Sort processes by priority (Lower number = Higher priority)
        Arrays.sort(processes, (p1, p2) -> p1.priority - p2.priority);

        // Calculate Waiting Time and Turnaround Time
        processes[0].waitingTime = 0; // First process has no waiting time
        float totalWaitingTime = 0, totalTurnaroundTime = 0;

        for (int i = 1; i < n; i++) {
            processes[i].waitingTime = processes[i - 1].waitingTime +
processes[i - 1].burstTime;
        }

        for (int i = 0; i < n; i++) {
            processes[i].turnaroundTime = processes[i].waitingTime +
processes[i].burstTime;
            totalWaitingTime += processes[i].waitingTime;
            totalTurnaroundTime += processes[i].turnaroundTime;
        }

        // Display results
        System.out.println("\nProcess\tBurst Time\tPriority\tWaiting
Time\tTurnaround Time");
        for (Process p : processes) {
```

```
            System.out.println(p.id + "\t" + p.burstTime + "\t\t" + p.priority +
"\t\t" + p.waitingTime + "\t\t" + p.turnaroundTime);
        }

        // Print averages
        System.out.printf("\nAverage Waiting Time: %.2f\n",
totalWaitingTime / n);
        System.out.printf("Average Turnaround Time: %.2f\n",
totalTurnaroundTime / n);

        sc.close();
    }
}
```

8. Write a program to implement the Round-Robin CPU Scheduling Algorithms

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;

class Process {
    int id, burstTime, remainingTime, waitingTime, turnaroundTime;

    Process(int id, int burstTime) {
        this.id = id;
        this.burstTime = burstTime;
        this.remainingTime = burstTime;
    }
}
```

```
public class RoundRobinScheduling {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // Get number of processes
        System.out.print("Enter the number of processes: ");
        int n = sc.nextInt();

        // Get time quantum
        System.out.print("Enter the time quantum: ");
        int quantum = sc.nextInt();

        Queue<Process> queue = new LinkedList<>();
        Process processes[] = new Process[n];

        // Input burst times
        System.out.println("Enter Burst Time for each process:");
        for (int i = 0; i < n; i++) {
            System.out.print("Process " + (i + 1) + ": ");
            int burstTime = sc.nextInt();
            processes[i] = new Process(i + 1, burstTime);
            queue.add(processes[i]);
        }

        int currentTime = 0;
        while (!queue.isEmpty()) {
            Process current = queue.poll();

            // Process execution for time quantum or remaining time
```

```
            int executeTime = Math.min(current.remainingTime,
quantum);
            current.remainingTime -= executeTime;
            currentTime += executeTime;

            // If process is still not finished, add it back to queue
            if (current.remainingTime > 0) {
                queue.add(current);
            } else {
                // Calculate turnaround and waiting time
                current.turnaroundTime = currentTime;
                current.waitingTime = current.turnaroundTime -
current.burstTime;
            }
        }

        // Display results
        float totalWaitingTime = 0, totalTurnaroundTime = 0;
        System.out.println("\nProcess\tBurst Time\tWaiting
Time\tTurnaround Time");
        for (Process p : processes) {
            System.out.println(p.id + "\t" + p.burstTime + "\t\t" +
p.waitingTime + "\t\t" + p.turnaroundTime);
            totalWaitingTime += p.waitingTime;
            totalTurnaroundTime += p.turnaroundTime;
        }

        // Print averages
        System.out.printf("\nAverage Waiting Time: %.2f\n",
totalWaitingTime / n);
```

```
        System.out.printf("Average Turnaround Time: %.2f\n",
totalTurnaroundTime / n);

        sc.close();
    }
}
```

9. Write a program to implement the producer-consumer problem using semaphores.

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.concurrent.Semaphore;

class Buffer {
    private final Queue<Integer> queue = new LinkedList<>();
    private final int capacity;

    // Semaphores
    private final Semaphore mutex = new Semaphore(1);  //
Controls access to buffer
    private final Semaphore empty;  // Tracks empty slots
    private final Semaphore full = new Semaphore(0);   // Tracks
filled slots

    public Buffer(int capacity) {
        this.capacity = capacity;
        this.empty = new Semaphore(capacity); // Initialize empty
slots
```

```
    }

    public void produce(int item) throws InterruptedException {
        empty.acquire();  // Wait for an empty slot
        mutex.acquire();  // Lock the buffer

        queue.add(item);
        System.out.println("Produced: " + item);

        mutex.release();  // Unlock the buffer
        full.release();   // Signal that an item is available
    }

    public int consume() throws InterruptedException {
        full.acquire();   // Wait for an available item
        mutex.acquire();  // Lock the buffer

        int item = queue.poll();
        System.out.println("Consumed: " + item);

        mutex.release();  // Unlock the buffer
        empty.release();  // Signal that a slot is free

        return item;
    }
}

class Producer extends Thread {
    private final Buffer buffer;
```

```
    public Producer(Buffer buffer) {
        this.buffer = buffer;
    }

    public void run() {
        try {
            for (int i = 1; i <= 10; i++) {
                buffer.produce(i);
                Thread.sleep(500); // Simulate production time
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

class Consumer extends Thread {
    private final Buffer buffer;

    public Consumer(Buffer buffer) {
        this.buffer = buffer;
    }

    public void run() {
        try {
            for (int i = 1; i <= 10; i++) {
                buffer.consume();
                Thread.sleep(1000); // Simulate consumption time
            }
        } catch (InterruptedException e) {
```

```
            e.printStackTrace();
        }
    }
}

public class ProducerConsumer {
    public static void main(String[] args) {
        Buffer buffer = new Buffer(5); // Shared buffer with a capacity
of 5

        Producer producer = new Producer(buffer);
        Consumer consumer = new Consumer(buffer);

        producer.start();
        consumer.start();
    }
}
```

10. Write a program to implement the FIFO Page Replacement Algorithms

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;

public class FIFOPageReplacement {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
```

```
// Get number of frames in memory
System.out.print("Enter the number of frames: ");
int framesCount = sc.nextInt();

// Get number of pages
System.out.print("Enter the number of pages: ");
int n = sc.nextInt();

int[] pages = new int[n];

// Input page reference string
System.out.println("Enter the page reference string:");
for (int i = 0; i < n; i++) {
    pages[i] = sc.nextInt();
}

// Implement FIFO Page Replacement
Queue<Integer> memory = new LinkedList<>();
int pageFaults = 0;

System.out.println("\nPage Reference\tFrames in Memory");

for (int page : pages) {
    // If page is not in memory, replace the oldest one
    if (!memory.contains(page)) {
        if (memory.size() == framesCount) {
            memory.poll(); // Remove oldest page
        }
        memory.add(page);
```

```
                pageFaults++;
            }

            // Display current memory state
            System.out.print(page + "\t\t" + memory + "\n");
        }

        System.out.println("\nTotal Page Faults: " + pageFaults);
        sc.close();
    }
}
```

11. Write a program to implement the LRU Page Replacement Algorithms

```
import java.util.*;

public class LRUPageReplacement {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // Get number of frames in memory
        System.out.print("Enter the number of frames: ");
        int framesCount = sc.nextInt();

        // Get number of pages
        System.out.print("Enter the number of pages: ");
        int n = sc.nextInt();

        int[] pages = new int[n];
```

```
        // Input page reference string
        System.out.println("Enter the page reference string:");
        for (int i = 0; i < n; i++) {
            pages[i] = sc.nextInt();
        }

        // Implement LRU Page Replacement
        LinkedHashSet<Integer> memory = new
LinkedHashSet<>(framesCount);
        LinkedList<Integer> usageOrder = new LinkedList<>();
        int pageFaults = 0;

        System.out.println("\nPage Reference\tFrames in Memory");

        for (int page : pages) {
            if (!memory.contains(page)) {  // Page Fault
                if (memory.size() == framesCount) {
                    int lru = usageOrder.removeFirst(); // Remove LRU
page
                    memory.remove(lru);
                }
                memory.add(page);
                pageFaults++;
            } else {
                usageOrder.remove((Integer) page); // Update order
            }
            usageOrder.add(page);

            // Display current memory state
```

```
            System.out.print(page + "\t\t" + memory + "\n");
        }

        System.out.println("\nTotal Page Faults: " + pageFaults);
        sc.close();
    }
}
```

12. Write a program to implement the Optimal Page Replacement Algorithms

```
import java.util.*;

public class OptimalPageReplacement {
    public static int predict(int[] pages, List<Integer> memory, int startIndex) {
        int farthest = -1, indexToReplace = -1;
        for (int i = 0; i < memory.size(); i++) {
            int page = memory.get(i);
            int j;
            for (j = startIndex; j < pages.length; j++) {
                if (pages[j] == page) {
                    if (j > farthest) {
                        farthest = j;
                        indexToReplace = i;
                    }
                    break;
                }
            }
            if (j == pages.length) // If a page is never used in future
```

```
                return i;
            }
            return (indexToReplace == -1) ? 0 : indexToReplace;
        }

        public static void optimalPageReplacement(int[] pages, int
framesCount) {
            List<Integer> memory = new ArrayList<>();
            int pageFaults = 0;

            System.out.println("\nPage Reference\tFrames in Memory");

            for (int i = 0; i < pages.length; i++) {
                int page = pages[i];

                if (!memory.contains(page)) {  // Page Fault
                    if (memory.size() == framesCount) {
                        int indexToReplace = predict(pages, memory, i + 1);
                        memory.set(indexToReplace, page);
                    } else {
                        memory.add(page);
                    }
                    pageFaults++;
                }

                // Display current memory state
                System.out.print(page + "\t\t" + memory + "\n");
            }

            System.out.println("\nTotal Page Faults: " + pageFaults);
```

```
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // Get number of frames in memory
        System.out.print("Enter the number of frames: ");
        int framesCount = sc.nextInt();

        // Get number of pages
        System.out.print("Enter the number of pages: ");
        int n = sc.nextInt();

        int[] pages = new int[n];

        // Input page reference string
        System.out.println("Enter the page reference string:");
        for (int i = 0; i < n; i++) {
            pages[i] = sc.nextInt();
        }

        optimalPageReplacement(pages, framesCount);
        sc.close();
    }
}
```