# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**"JnanaSangama", Belgaum -590014, Karnataka.**

**LAB REPORT**
**on**

# Operating Systems (22CS4PCOPS)

***Submitted by***


**BHARATH M (1BM22CS405)**


***in partial fulfillment for the award of the degree of***
**BACHELOR OF ENGINEERING**
***in***
**COMPUTER SCIENCE AND ENGINEERING**

**B.M.S. COLLEGE OF ENGINEERING**
**(Autonomous Institution under VTU)**
**BENGALURU-560019**
**June-2023 to September-2023**

## B. M. S. College of Engineering,

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

## Department of Computer Science and Engineering

**<u>CERTIFICATE</u>**

This is to certify that the Lab work entitled "Operating Systems" carried out by **Bharath M(1BM22CS405),** who is a bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a **Operating Systems - (22CS4PCOPS)** work prescribed for the said degree.

| | |
|---|---|
| Name of the Lab-Incharge | **Sneha S Bagalkot** |
| Designation | Professor and Head |
| Department of CSE | Department of CSE |
| BMSCE, Bengaluru | BMSCE, Bengaluru |

`

# Index

## Laboratory Program 1

Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time.
?FCFS
? SJF (preemptive & Non-pre-emptive)

```
#include<stdio.h>

int num;

int wait_time[100], burst_time[100], tat[100], proc[100], arrival_time[100];

void burstsort() {

 for (int i = 0; i < num - 1; i++)

 {

  for (int j = 0; j < num - i - 1; j++)

  {

   if (burst_time[j] > burst_time[j + 1])

   {

    int temp = burst_time[j];

    burst_time[j] = burst_time[j + 1];

    burst_time[j + 1] = temp;

    temp = proc[j];

    proc[j] = proc[j + 1];

    proc[j + 1] = temp;

    temp = arrival_time[j];

    arrival_time[j] = arrival_time[j + 1];
```

```
      arrival_time[j + 1] = temp;

     }

    }
  }
}
void

waitingtime2() {

  int remaining_time[num];

  int completed = 0;

  // Initialize the remaining_time array
  for (int i = 0; i < num; i++) {

    remaining_time[i] = burst_time[i];

  }
  int current_time = 0;

  while (completed != num) {

    int shortest_index = -1;

    int shortest_burst = __INT_MAX__;

    // Find the process with the shortest remaining burst time among the arrived and
uncompleted processes
    for (int i = 0; i < num; i++) {

      if (arrival_time[i] <= current_time &&
        remaining_time[i] < shortest_burst && remaining_time[i] > 0) {

        shortest_burst = remaining_time[i];

        shortest_index = i;

      }

    }
```

```
    if (shortest_index == -1) {

      current_time++;

    } else {

      // Execute the process for 1 unit of time
      remaining_time[shortest_index]--;

      current_time++;

      // If the process is completed, update the waiting time and completed count
      if (remaining_time[shortest_index] == 0) {

        completed++;

        wait_time[shortest_index] =
          current_time - burst_time[shortest_index] -
          arrival_time[shortest_index] - arrival_time[0];

        if (wait_time[shortest_index] < 0)

          wait_time[shortest_index] = 0;

      }

    }

  }

}

void
waitingtime1() {

  wait_time[0] = 0;

  for (int i = 1; i < num; i++)

  {

    wait_time[i] = burst_time[i - 1] + wait_time[i - 1] - arrival_time[i];

    if (wait_time[i] < 0)
```

```
      wait_time[i] = 0;

  }

}

void
turnaroundtime() {

  for (int i = 0; i < num; i++)

    tat[i] = burst_time[i] + wait_time[i];

}
void

avgtime() {

  double avg_wait = 0.0, avg_tat = 0.0;

  for (int i = 0; i < num; i++)

  {

    avg_wait += wait_time[i];

    avg_tat += tat[i];

  }
  avg_wait = avg_wait / num;

  avg_tat = avg_tat / num;

  printf("Average waiting time is %f\nAverage turnaround time is %f\n",
    avg_wait, avg_tat);

}

int

main() {

  printf("1.FCFS\n2.SJF\n3.SRTF\nEnter your choice:");
```

```
int  ch;

scanf("%d", & ch);

if (ch < 1 || ch > 3)

{

  printf("Invalid choice!");

  return 0;

}

printf("Enter the total number of processes:");

scanf("%d", & num);

for (int i = 0; i < num; i++)

{

  printf("Process %d\n", i + 1);

  printf("Burst Time:");

  scanf("%d", & burst_time[i]);

  proc[i] = i + 1;

  printf("Arrival Time:");

  scanf("%d", & arrival_time[i]);

  printf("\n");

}
switch (ch)

{

case 1:
  waitingtime1();
```

```
  turnaroundtime();

  printf
    ("Process\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n");

  for (int i = 0; i < num; i++)

    printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", proc[i], arrival_time[i],
      burst_time[i], wait_time[i], tat[i]);

  avgtime();

break;

case 2:
  burstsort();

  waitingtime1();

  turnaroundtime();

  printf
    ("Process\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n");

  for (int i = 0; i < num; i++)

    printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", proc[i], arrival_time[i],
      burst_time[i], wait_time[i], tat[i]);

  avgtime();
  break;

case 3:

  waitingtime2();

  turnaroundtime();

  printf
    ("Process\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n");

  for (int i = 0; i < num; i++)
```

```
    printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", proc[i], arrival_time[i],
     burst_time[i], wait_time[i], tat[i]);

   avgtime();

   break;

  }

  return 0;

}
```

## Output

```
C:\Users\admin\CS4SEM>a
1.FCFS
2.SJF
3.SRTF
Enter your choice:1
Enter the total number of processes:4
Process 1
Burst Time:12
Arrival Time:1

Process 2
Burst Time:4
Arrival Time:0

Process 3
Burst Time:3
Arrival Time:2

Process 4
Burst Time:5
Arrival Time:4

Process Arrival Time    Burst Time      Waiting Time    Turnaround Time
1       1               12              0               12
2       0               4               12              16
3       2               3               14              17
4       4               5               13              18
Average waiting time is 9.750000
Average turnaround time is 15.750000
```

```
C:\Users\admin\CS4SEM>a
1.FCFS
2.SJF
3.SRTF
Enter your choice:2
Enter the total number of processes:4
Process 1
Burst Time:12
Arrival Time:1

Process 2
Burst Time:4
Arrival Time:0

Process 3
Burst Time:3
Arrival Time:2

Process 4
Burst Time:5
Arrival Time:4

Process Arrival Time    Burst Time      Waiting Time    Turnaround Time
3       2               3               0               3
2       0               4               3               7
4       4               5               3               8
1       1               12              7               19
Average waiting time is 3.250000
Average turnaround time is 9.250000
```

```
C:\Users\admin\CS4SEM>a
1.FCFS
2.SJF
3.SRTF
Enter your choice:3
Enter the total number of processes:4
Process 1
Burst Time:12
Arrival Time:1

Process 2
Burst Time:4
Arrival Time:0

Process 3
Burst Time:3
Arrival Time:2

Process 4
Burst Time:5
Arrival Time:4

Process Arrival Time    Burst Time      Waiting Time    Turnaround Time
1       1               12              10              22
2       0               4               0               4
3       2               3               1               4
4       4               5               2               7
Average waiting time is 3.250000
Average turnaround time is 9.250000
```

## Laboratory Program 2

Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time.
? Priority (preemptive & Non-pre-emptive)
?Round Robin (Experiment with different quantum sizes for RR algorithm)

```
#include <stdio.h>

#define MAX_PROCESSES 10

void roundRobin(int burst_time[], int arrival_time[], int n, int time_quantum) {
   int remaining_time[MAX_PROCESSES];
   int waiting_time[MAX_PROCESSES] = {0};
   int turnaround_time[MAX_PROCESSES] = {0};

   // Initialize the remaining time array with burst times
   for (int i = 0; i < n; i++) {
      remaining_time[i] = burst_time[i];
   }

   int current_time = 0;
   int completed = 0;
   int front = 0, rear = 0;
   int queue[MAX_PROCESSES];

   while (completed < n) {
      for (int i = 0; i < n; i++) {
         if (arrival_time[i] <= current_time && remaining_time[i] > 0) {
            queue[rear++] = i;
         }
      }

      if (front == rear) {
         current_time++;
         continue;
      }

      int process_index = queue[front];
      front = (front + 1) % MAX_PROCESSES;

      if (remaining_time[process_index] <= time_quantum) {
         current_time += remaining_time[process_index];
```

```
            turnaround_time[process_index] = current_time - arrival_time[process_index];
            waiting_time[process_index] = turnaround_time[process_index] -
burst_time[process_index];
            remaining_time[process_index] = 0;
            completed++;
        } else {
            current_time += time_quantum;
            remaining_time[process_index] -= time_quantum;
        }
    }

    // Calculate average waiting time and turnaround time
    double avg_waiting_time = 0.0;
    double avg_turnaround_time = 0.0;

    for (int i = 0; i < n; i++) {
        avg_waiting_time += waiting_time[i];
        avg_turnaround_time += turnaround_time[i];
    }

    avg_waiting_time /= n;
    avg_turnaround_time /= n;

    // Print the results
    printf("Round Robin Scheduling with Arrival Time\n");
    printf("Process\tBurst Time\tArrival Time\tWaiting Time\tTurnaround Time\n");

    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", i + 1, burst_time[i], arrival_time[i], waiting_time[i],
turnaround_time[i]);
    }

    printf("\nAverage Waiting Time: %.2f\n", avg_waiting_time);
    printf("Average Turnaround Time: %.2f\n", avg_turnaround_time);
}

void preemptivePriority(int burst_time[], int arrival_time[], int priority[], int n) {
    int remaining_time[MAX_PROCESSES];
    int waiting_time[MAX_PROCESSES] = {0};
    int turnaround_time[MAX_PROCESSES] = {0};

    // Initialize the remaining time array with burst times
    for (int i = 0; i < n; i++) {
        remaining_time[i] = burst_time[i];
```

```
    }

    int current_time = 0;
    int completed = 0;

    while (completed < n) {
        int highest_priority = 9999; // Higher value means lower priority
        int selected_process = -1;

        for (int i = 0; i < n; i++) {
            if (arrival_time[i] <= current_time && remaining_time[i] > 0 && priority[i] <
highest_priority) {
                highest_priority = priority[i];
                selected_process = i;
            }
        }

        if (selected_process == -1) {
            current_time++;
            continue;
        }

        remaining_time[selected_process]--;
        current_time++;

        if (remaining_time[selected_process] == 0) {
            turnaround_time[selected_process] = current_time - arrival_time[selected_process];
            waiting_time[selected_process] = turnaround_time[selected_process] -
burst_time[selected_process];
            completed++;
        }
    }

    // Calculate average waiting time and turnaround time
    double avg_waiting_time = 0.0;
    double avg_turnaround_time = 0.0;

    for (int i = 0; i < n; i++) {
        avg_waiting_time += waiting_time[i];
        avg_turnaround_time += turnaround_time[i];
    }

    avg_waiting_time /= n;
    avg_turnaround_time /= n;
```

```
    // Print the results
    printf("\nPreemptive Priority Scheduling with Arrival Time\n");
    printf("Process\tBurst Time\tArrival Time\tPriority\tWaiting Time\tTurnaround Time\n");

    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", i + 1, burst_time[i], arrival_time[i], priority[i],
waiting_time[i], turnaround_time[i]);
    }

    printf("\nAverage Waiting Time: %.2f\n", avg_waiting_time);
    printf("Average Turnaround Time: %.2f\n", avg_turnaround_time);
}

int main() {
    int n;
    printf("Enter the total number of processes (up to %d): ", MAX_PROCESSES);
    scanf("%d", &n);

    int burst_time[MAX_PROCESSES], arrival_time[MAX_PROCESSES], priority[MAX_PROCESSES];

    printf("Enter the burst time, arrival time, and priority for each process:\n");
    for (int i = 0; i < n; i++) {
        printf("Process %d\n", i + 1);
        printf("Burst Time: ");
        scanf("%d", &burst_time[i]);
        printf("Arrival Time: ");
        scanf("%d", &arrival_time[i]);
        printf("Priority: ");
        scanf("%d", &priority[i]);
    }

    int time_quantum;
    printf("Enter the time quantum for Round Robin: ");
    scanf("%d", &time_quantum);

    roundRobin(burst_time, arrival_time, n, time_quantum);
    preemptivePriority(burst_time, arrival_time, priority, n);

    return 0;
}
```

## Output

```
C:\Users\admin\CS4SEM>gcc os.c

C:\Users\admin\CS4SEM>a
Enter the total number of processes (up to 10): 3
Enter the burst time, arrival time, and priority for each process:
Process 1
Burst Time: 12
Arrival Time: 1
Priority: 1
Process 2
Burst Time: 4
Arrival Time: 0
Priority: 2
Process 3
Burst Time: 3
Arrival Time: 2
Priority: 3
Enter the time quantum for Round Robin: 2
Round Robin Scheduling with Arrival Time
Process Burst Time      Arrival Time    Waiting Time    Turnaround Time
1       12              1               0               2
2       4               0               6               0
3       3               2               6               9

Average Waiting Time: 4.00
Average Turnaround Time: 3.67
```

```
Preemptive Priority Scheduling with Arrival Time
Process Burst Time      Arrival Time    Priority        Waiting Time    Turnaround Time
1       12              1               1               0               12
2       4               0               2               12              16
3       3               2               3               14              17

Average Waiting Time: 8.67
Average Turnaround Time: 15.00
```

## Laboratory Program 3

Write a C program to simulate a multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for the processes in each queue.

```
#include<stdio.h>
#include<stdbool.h>

#define MAX_QUEUE_SIZE 100

// Structure to represent a process
struct process {
  int pid;
  int priority;
  int burst_time;
};

// Function to implement FCFS scheduling algorithm
void fcfs_scheduling(struct process queue[], int size) {
  int total_time = 0;
  float average_wait_time = 0;
  float average_turnaround_time = 0;

  printf("\nProcess\tPriority\tBurst Time\tWaiting Time\tTurnaround Time\n");

  for (int i = 0; i < size; i++) {
    int waiting_time = total_time;
    int turnaround_time = waiting_time + queue[i].burst_time;

    printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", queue[i].pid, queue[i].priority, queue[i].burst_time,
waiting_time, turnaround_time);

    total_time += queue[i].burst_time;
    average_wait_time += waiting_time;
    average_turnaround_time += turnaround_time;
  }

  average_wait_time /= size;
  average_turnaround_time /= size;

  printf("\nAverage Waiting Time: %.2f\n", average_wait_time);
  printf("Average Turnaround Time: %.2f\n", average_turnaround_time);
```

```
}

int main() {
   int num_system_processes, num_user_processes;

   printf("Enter the number of system processes: ");
   scanf("%d", &num_system_processes);

   printf("Enter the number of user processes: ");
   scanf("%d", &num_user_processes);

   // Create queues for system processes and user processes
   struct process system_queue[MAX_QUEUE_SIZE];
   struct process user_queue[MAX_QUEUE_SIZE];

   // Accept details for system processes
   printf("\nEnter details for system processes:\n");
   for (int i = 0; i < num_system_processes; i++) {
      printf("\nProcess %d:\n", i + 1);
      system_queue[i].pid = i + 1;
      system_queue[i].priority = 1; // Higher priority for system processes
      printf("Enter burst time: ");
      scanf("%d", &system_queue[i].burst_time);
   }

   // Accept details for user processes
   printf("\nEnter details for user processes:\n");
   for (int i = 0; i < num_user_processes; i++) {
      printf("\nProcess %d:\n", i + 1);
      user_queue[i].pid = i + 1;
      user_queue[i].priority = 2; // Lower priority for user processes
      printf("Enter burst time: ");
      scanf("%d", &user_queue[i].burst_time);
   }

   printf("\n--- System Processes ---\n");
   fcfs_scheduling(system_queue, num_system_processes);

   printf("\n--- User Processes ---\n");
   fcfs_scheduling(user_queue, num_user_processes);

   // Preemptive execution of system processes
   int system_queue_index = 0;
   int user_queue_index = 0;
```

```
    printf("\n--- Execution Order ---\n");
    printf("Process\tPriority\tBurst Time\n");

    while (system_queue_index < num_system_processes || user_queue_index <
num_user_processes) {
        if (system_queue_index < num_system_processes && user_queue_index <
num_user_processes) {
            // Compare the burst times of the current processes in both queues
            if (system_queue[system_queue_index].burst_time <=
user_queue[user_queue_index].burst_time) {
                printf("%d\t%d\t\t%d\n", system_queue[system_queue_index].pid,
system_queue[system_queue_index].priority,
system_queue[system_queue_index].burst_time);
                system_queue_index++;
            } else {
                printf("%d\t%d\t\t%d\n", user_queue[user_queue_index].pid,
user_queue[user_queue_index].priority, user_queue[user_queue_index].burst_time);
                user_queue_index++;
            }
        } else if (system_queue_index < num_system_processes) {
            printf("%d\t%d\t\t%d\n", system_queue[system_queue_index].pid,
system_queue[system_queue_index].priority,
system_queue[system_queue_index].burst_time);
            system_queue_index++;
        } else if (user_queue_index < num_user_processes) {
            printf("%d\t%d\t\t%d\n", user_queue[user_queue_index].pid,
user_queue[user_queue_index].priority, user_queue[user_queue_index].burst_time);
            user_queue_index++;
        }
    }

    return 0;
}
```

## Output

```
C:\Users\admin\CS4SEM>gcc os.c

C:\Users\admin\CS4SEM>a
Enter the number of system processes: 3
Enter the number of user processes: 3

Enter details for system processes:

Process 1:
Enter burst time: 12

Process 2:
Enter burst time: 25

Process 3:
Enter burst time: 3

Enter details for user processes:

Process 1:
Enter burst time: 20

Process 2:
Enter burst time: 6

Process 3:
Enter burst time: 9
```

```
--- System Processes ---

Process Priority        Burst Time      Waiting Time    Turnaround Time
1       1               12              0               12
2       1               25              12              37
3       1               3               37              40

Average Waiting Time: 16.33
Average Turnaround Time: 29.67

--- User Processes ---

Process Priority        Burst Time      Waiting Time    Turnaround Time
1       2               20              0               20
2       2               6               20              26
3       2               9               26              35

Average Waiting Time: 15.33
Average Turnaround Time: 27.00
```

## Laboratory Program 4

Write a C program to simulate Real-Time CPU Scheduling
algorithms:
a) Rate- Monotonic
b) Earliest-deadline First
c) Proportional scheduling

```
#include  <stdio.h>

#define MAX_TASKS 100

// Structure to represent a task
struct task {
   int id;
   int period;
   int execution_time;
   int deadline;
   int priority;
   int response_time;
   int start_time;
   int finish_time;
};

// Function to simulate Rate-Monotonic scheduling algorithm
void rate_monotonic(struct task tasks[], int num_tasks) {
   printf("Rate-Monotonic Scheduling:\n");

   int current_time = 0;
   int total_response_time = 0;
   int total_turnaround_time = 0;

   for (int i = 0; i < num_tasks; i++) {
      struct task current_task = tasks[i];

      if (current_task.start_time > current_time)
         current_time = current_task.start_time;

      current_task.start_time = current_time;
      current_task.finish_time = current_time + current_task.execution_time;
      current_task.response_time = current_task.start_time;
      total_response_time += current_task.response_time;
      total_turnaround_time += current_task.finish_time;
```

```
        current_time += current_task.period;

        printf("Task %d: Start Time = %d, Finish Time = %d\n", current_task.id,
current_task.start_time, current_task.finish_time);
    }

    float average_response_time = (float)total_response_time / num_tasks;
    float average_turnaround_time = (float)total_turnaround_time / num_tasks;

    printf("Average Response Time: %.2f\n", average_response_time);
    printf("Average Turnaround Time: %.2f\n", average_turnaround_time);
    printf("\n");
}

// Function to simulate Earliest-Deadline First scheduling algorithm
void earliest_deadline_first(struct task tasks[], int num_tasks) {
    printf("Earliest-Deadline First Scheduling:\n");

    int current_time = 0;
    int total_response_time = 0;
    int total_turnaround_time = 0;

    for (int i = 0; i < num_tasks; i++) {
        struct task current_task = tasks[i];

        if (current_task.start_time > current_time)
            current_time = current_task.start_time;

        current_task.start_time = current_time;
        current_task.finish_time = current_time + current_task.execution_time;
        current_task.response_time = current_task.start_time;
        total_response_time += current_task.response_time;
        total_turnaround_time += current_task.finish_time;

        current_time += current_task.period;

        printf("Task %d: Start Time = %d, Finish Time = %d\n", current_task.id,
current_task.start_time, current_task.finish_time);
    }

    float average_response_time = (float)total_response_time / num_tasks;
    float average_turnaround_time = (float)total_turnaround_time / num_tasks;

    printf("Average Response Time: %.2f\n", average_response_time);
```

```
    printf("Average Turnaround Time: %.2f\n", average_turnaround_time);
    printf("\n");
}

// Function to simulate Proportional Scheduling algorithm
void proportional_scheduling(struct task tasks[], int num_tasks) {
    printf("Proportional Scheduling:\n");

    int current_time = 0;
    int total_response_time = 0;
    int total_turnaround_time = 0;

    int total_execution_time = 0;
    for (int i = 0; i < num_tasks; i++) {
        total_execution_time += tasks[i].execution_time;
    }

    for (int i = 0; i < num_tasks; i++) {
        struct task current_task = tasks[i];

        if (current_task.start_time > current_time)
            current_time = current_task.start_time;

        current_task.start_time = current_time;
        current_task.finish_time = current_time + (int)((float)current_task.execution_time /
total_execution_time * 100);
        current_task.response_time = current_task.start_time;
        total_response_time += current_task.response_time;
        total_turnaround_time += current_task.finish_time;

        current_time += (int)((float)current_task.execution_time / total_execution_time * 100);

        printf("Task %d: Start Time = %d, Finish Time = %d\n", current_task.id,
current_task.start_time, current_task.finish_time);
    }

    float average_response_time = (float)total_response_time / num_tasks;
    float average_turnaround_time = (float)total_turnaround_time / num_tasks;

    printf("Average Response Time: %.2f\n", average_response_time);
    printf("Average Turnaround Time: %.2f\n", average_turnaround_time);
    printf("\n");
}
```

```
int main() {
    int num_tasks;

    printf("Enter the number of tasks: ");
    scanf("%d", &num_tasks);

    struct task tasks[MAX_TASKS];

    // Accept task details from the user
    for (int i = 0; i < num_tasks; i++) {
        printf("\nTask %d:\n", i + 1);
        tasks[i].id = i + 1;

        printf("Enter the period: ");
        scanf("%d", &tasks[i].period);

        printf("Enter the execution time: ");
        scanf("%d", &tasks[i].execution_time);

        printf("Enter the deadline: ");
        scanf("%d", &tasks[i].deadline);

        printf("Enter the priority: ");
        scanf("%d", &tasks[i].priority);

        printf("Enter the start time: ");
        scanf("%d", &tasks[i].start_time);
    }

    rate_monotonic(tasks, num_tasks);
    earliest_deadline_first(tasks, num_tasks);
    proportional_scheduling(tasks, num_tasks);

    return 0;
}
```

## Output

```
C:\Users\admin\CS4SEM>a
Enter the number of tasks: 3

Task 1:
Enter the period: 100
Enter the execution time: 25
Enter the deadline: 50
Enter the priority: 1
Enter the start time: 0

Task 2:
Enter the period: 50
Enter the execution time: 10
Enter the deadline: 30
Enter the priority: 2
Enter the start time: 10

Task 3:
Enter the period: 150
Enter the execution time: 50
Enter the deadline: 100
Enter the priority: 3
Enter the start time: 0
```

```
Rate-Monotonic Scheduling:
Task 1: Start Time = 0, Finish Time = 25
Task 2: Start Time = 100, Finish Time = 110
Task 3: Start Time = 150, Finish Time = 200
Average Response Time: 83.33
Average Turnaround Time: 111.67

Earliest-Deadline First Scheduling:
Task 1: Start Time = 0, Finish Time = 25
Task 2: Start Time = 100, Finish Time = 110
Task 3: Start Time = 150, Finish Time = 200
Average Response Time: 83.33
Average Turnaround Time: 111.67

Proportional Scheduling:
Task 1: Start Time = 0, Finish Time = 29
Task 2: Start Time = 29, Finish Time = 40
Task 3: Start Time = 40, Finish Time = 98
Average Response Time: 23.00
Average Turnaround Time: 55.67
```

## Laboratory Program 5

Write a C program to simulate producer-consumer problem using semaphores.

```
#include<stdio.h>
#include<stdlib.h>
int mutex=1,full=0,empty,x=0;

int wait(int s)
{
  return (--s);
}
int signal(int s)
{
  return(++s);
}

void producer()
{
  mutex=wait(mutex);
  full=signal(full);
  empty=wait(empty);
  x++;
  printf("\nProducer produces Item %d",x);
  mutex=signal(mutex);
}

void consumer()
```

```
{
  mutex=wait(mutex);
  full=wait(full);
  empty=signal(empty);
  printf("\nConsumer consumes Item %d",x);
  x--;
  mutex=signal(mutex);
}

void printbuffer()
{
  if(full==0)
  {
        printf("Buffer is empty!\n");
        return;
  }
  printf("The contents of the buffer are:");
  for(int i=1;i<=full;i++)
  printf("%d\t",i);
}
int main()
{
  int n,ch;
  printf("Enter the buffer size:");
  scanf("%d",&n);
  empty=n;
  printf("\n1.Producer\n2.Consumer\n3.Print Buffer Contents\n4.Exit");
```

```
  while(1)
  {
        printf("\nEnter your choice:");
        scanf("%d",&ch);
        switch(ch)
        {
        case 1: if((mutex==1)&&(empty!=0))
        producer();
        else
        printf("Buffer is full!!");
        break;
        case 2: if((mutex==1)&&(full!=0))
        consumer();
        else
        printf("Buffer is empty!!");
        break;
        case 3:
        printbuffer();
        break;
        case 4:return 0;
        }
  }
}
```

## Output

```
C:\Users\admin\CS4SEM>a
Enter the buffer size:3

1.Producer
2.Consumer
3.Print Buffer Contents
4.Exit
Enter your choice:1

Producer produces Item 1
Enter your choice:1

Producer produces Item 2
Enter your choice:1

Producer produces Item 3
Enter your choice:1
Buffer is full!!
Enter your choice:3
The contents of the buffer are:1        2       3
Enter your choice:2

Consumer consumes Item 3
Enter your choice:3
The contents of the buffer are:1        2
Enter your choice:2

Consumer consumes Item 2
Enter your choice:2

Consumer consumes Item 1
Enter your choice:3
Buffer is empty!
```

## Laboratory Program 6

Write a C program to simulate the concept of Dining - Philosophers problem.

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (phnum + 4) % N
#define RIGHT (phnum + 1) % N

int eat=0;

int state[N];
int phil[N] = { 0, 1, 2, 3, 4 };

sem_t mutex;
sem_t S[N];

void test(int phnum)
{
  if (state[phnum] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING)
  {
          // state that eating
          state[phnum] = EATING;
          eat++;
          sleep(2);
          printf("Philosopher %d takes fork %d and %d\n",phnum + 1, LEFT + 1, phnum + 1);
          printf("Philosopher %d is Eating\n", phnum + 1);
          sem_post(&S[phnum]);
  }
}

// Take up chopsticks
void take_fork(int phnum)
{
  sem_wait(&mutex);
```

```
  // Set state of thread to hungry
  state[phnum] = HUNGRY;
  printf("Philosopher %d is Hungry\n", phnum + 1);
  // Start eating only if nighbours are not eating
  test(phnum);
  sem_post(&mutex);
  // If neighbour is eating, wait to be signalled
  sem_wait(&S[phnum]);
  sleep(1);
}

// Put down chopsticks
void put_fork(int phnum)
{
  sem_wait(&mutex);
  // Set state of thread to thinking
  state[phnum] = THINKING;
  printf("Philosopher %d putting fork %d and %d down\n",phnum + 1, LEFT + 1, phnum + 1);
  printf("Philosopher %d is thinking\n", phnum + 1);
  test(LEFT);
  test(RIGHT);
  sem_post(&mutex);
}

void* philosopher(void* num)
{
  while (1)
  {
        int* i = num;
        sleep(1);
        take_fork(*i);
        eat=eat+1;
        sleep(0);
        put_fork(*i);
  }
}

int main()
{
  int i;
  pthread_t thread_id[N];
  // Initializing Semaphores
  sem_init(&mutex, 0, 1);
  for (i = 0; i < N; i++)
```

```
  sem_init(&S[i], 0, 0);
  for (i = 0; i < N; i++)
  {
          pthread_create(&thread_id[i], NULL, philosopher, &phil[i]);
          printf("Philosopher %d is thinking\n", i + 1);
  }
  for (i = 0; i < N; i++)
  pthread_join(thread_id[i], NULL);
  return 0;
}
```

## Output

```
C:\Users\admin\CS4SEM>a
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 5 is Hungry
Philosopher 4 is Hungry
Philosopher 2 is Hungry
Philosopher 1 is Hungry
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 3 is Hungry
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 1 is Hungry
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 3 is Hungry
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
```

## Laboratory Program 7

Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.

```
#include<stdio.h>
#include<stdlib.h>
int allocation[25][25],available[25],max[25][25],need[25][25],work[25][25],safe[25];
int main()
{
  int n,m,i,j;
  printf("Enter the number of processes:");
  scanf("%d",&n);
  printf("Enter the number of resources:");
  scanf("%d",&m);
  printf("Enter the allocation matrix:\n");
  for(i=0;i<n;i++)
  {
        for(j=0;j<m;j++)
        scanf("%d",&allocation[i][j]);
  }
  printf("Enter the maximum resources matrix:\n");
  for(i=0;i<n;i++)
  {
        for(j=0;j<m;j++)
        scanf("%d",&max[i][j]);
  }
  for(i=0;i<n;i++)
  {
        for(j=0;j<m;j++)
        need[i][j]=max[i][j]-allocation[i][j];
  }
  printf("Enter the available resources vector:\n");
  for(i=0;i<m;i++)
  scanf("%d",&available[i]);
  printf("Need Matrix :\n");
  for(i=0;i<n;i++)
  {
        for(j=0;j<m;j++)
        printf("%d ",need[i][j]);
        printf("\n");
  }
  int f[n], ans[n], ind = 0;
  for (i = 0; i < n; i++)
```

```
 {
 	f[i] = 0;
 }
int y = 0,k;
for (k = 0; k < 5; k++) {
	for (i = 0; i < n; i++) {
	if (f[i] == 0) {

	int flag = 0;
	for (j = 0; j < m; j++) {
	if (need[i][j] > available[j]){
	flag = 1;
	break;
	}
	}

	if (flag == 0) {
	ans[ind++] = i;
	for (y = 0; y < m; y++)
	available[y] += allocation[i][y];
	f[i] = 1;
	}
	}
	}
}

int flag = 1;

// To check if sequence is safe or not
for(int i = 0;i<n;i++)
{
	if(f[i]==0)
	{
	flag = 0;
	printf("The system is not in safe state.");
	break;
	}
}

if(flag==1)
{
	printf( "The safe sequence for the system is:\n");
	for (i = 0; i < n - 1; i++)
	printf( " P%d->",ans[i]);
```

```
        printf(" P%d\n",ans[n - 1]);
 }


  return 0;
}
```

## Output

```
C:\Users\admin\CS4SEM>a
Enter the number of processes:5
Enter the number of resources:3
Enter the allocation matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter the maximum resources matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter the available resources vector:
3 3 2
Need Matrix :
7 4 3
1 2 2
6 0 0
0 1 1
4 3 1
The safe sequence for the system is:
 P1-> P3-> P4-> P0-> P2
```

## Laboratory Program 8

Write a C program to simulate deadlock detection.

```
#include<stdio.h>
#include<stdlib.h>
int allocation[25][25],available[25],req[25][25];
int main()
{
  int n,m,i,j;
  printf("Enter the number of processes:");
  scanf("%d",&n);
  printf("Enter the number of resources:");
  scanf("%d",&m);
  printf("Enter the allocation matrix:\n");
  for(i=0;i<n;i++)
  {
        for(j=0;j<m;j++)
        scanf("%d",&allocation[i][j]);
  }
  printf("Enter the request matrix:\n");
  for(i=0;i<n;i++)
  {
        for(j=0;j<m;j++)
        scanf("%d",&req[i][j]);
  }
  printf("Enter the available resources vector:\n");
  for(i=0;i<m;i++)
  scanf("%d",&available[i]);
  int f[n], ans[n], ind = 0;
  for (i = 0; i < n; i++)
  {
        f[i] = 0;
  }
 int y = 0,k;
 for (k = 0; k < 5; k++) {
        for (i = 0; i < n; i++) {
        if (f[i] == 0) {

        int flag = 0;
        for (j = 0; j < m; j++) {
        if (req[i][j] > available[j]){
        flag = 1;
        break;
```

```
          }
          }

          if (flag == 0) {
          ans[ind++] = i;
          for (y = 0; y < m; y++)
          available[y] += allocation[i][y];
          f[i] = 1;
          }
          }
          }
 }

 int flag = 1;

 // To check if sequence is safe or not
 for(int i = 0;i<n;i++)
 {
          if(f[i]==0)
          {
          flag = 0;
          printf("Deadlock is encountered.");
          break;
          }
 }

 if(flag==1)
 {
          printf( "Deadlock is not encountered.The safe sequence for the system is:\n");
          for (i = 0; i < n - 1; i++)
          printf( " P%d->",ans[i]);
          printf(" P%d\n",ans[n - 1]);
 }


  return 0;
}
```

## Output

```
C:\Users\admin\CS4SEM>a
Enter the number of processes:5
Enter the number of resources:3
Enter the allocation matrix:
1 1 1
2 0 2
0 2 0
7 0 1
0 0 1
Enter the request matrix:
0 1 0
1 2 2
3 3 0
1 1 1
5 0 1
Enter the available resources vector:
0 2 0
Deadlock is not encountered.The safe sequence for the system is:
 P0-> P3-> P4-> P1-> P2
```

## Laboratory Program 9

Write a C program to simulate the following contiguous memory allocation techniques:
a) Worst-fit
b) Best-fit
c) First-fit

```
#include<stdio.h>
#include<stdlib.h>
int frag[100],block[100],files[100],nf,nb;
int maximum()
{
  int max=block[0];
  for(int i=1;i<nb;i++)
  {
        if(block[i]>max)
        max=block[i];
  }

  return max;
}
int minimum(int filesize)
{
  int min=block[0]-filesize;
  int temp=block[0];
  for(int i=1;i<nb;i++)
  {
        int diff=block[i]-filesize;
        if(diff<min&&diff>0)
        {
        min=diff;
        temp=block[i];
        }
  }

  return temp;
}
void worstfit()
{
  int i,j,blockpos;
  printf("Worst Fit Memory Allocation\nFile Number\tFile Size\tBlock Number\tBlock
Size\tFragment\n");
  for(i=0;i<nf;i++)
```

```
  {
        int tempmax=maximum();
        blockpos=-2;
        if(files[i]<=tempmax)
        {
        for(j=0;j<nb;j++)
        {
        if(block[j]==tempmax)
        {
                blockpos=j;
                break;
        }
        }
        frag[blockpos]=block[blockpos]-files[i];
        block[blockpos]=-1;

printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",i+1,files[i],blockpos+1,tempmax,frag[blockpos]);
        }
        else
        {
        printf("%d\t\t%d\t--------Not Allocated ------ \n",i+1,files[i]);
        }
  }

}
void firstfit()
{
  int i,j,alloc=0;
  printf("First Fit Memory Allocation\nFile Number\tFile Size\tBlock Number\tBlock
Size\tFragment\n");
  for(i=0;i<nf;i++)
  {
        alloc=0;
        for(j=0;j<nb;j++)
        {
        if(files[i]<=block[j])
        {
        alloc=1;
        break;
        }
        }
        if(alloc==1)
        {
        frag[i]=block[j]-files[i];
```

```
           printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",i+1,files[i],j+1,block[j],frag[i]);
          block[j]=frag[i];
           }
           else
           printf("%d\t\t%d\t--------Not Allocated ------ \n",i+1,files[i]);
  }
}
void bestfit()
{
  int i,j,blockpos;
  printf("Best Fit Memory Allocation\nFile Number\tFile Size\tBlock Number\tBlock 
Size\tFragment\n");
  for(i=0;i<nf;i++)
  {
           int tempbest=minimum(files[i]);
           //printf("%d",tempbest);
           blockpos=-2;
           if(files[i]<=tempbest)
           {
           for(j=0;j<nb;j++)
           {
           if(block[j]==tempbest)
           {
                     blockpos=j;
                     break;
           }

           }
           frag[blockpos]=block[blockpos]-files[i];
           block[blockpos]=frag[blockpos];

printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",i+1,files[i],blockpos+1,tempbest,frag[blockpos]);
           }
           else
           {
           printf("%d\t\t%d\t--------Not Allocated ------ \n",i+1,files[i]);
           }
  }
}
int main(int argc,char *argv[])
{
  int i,j;
  printf("Enter the number of blocks:");
  scanf("%d",&nb);
```

```
    printf("Enter the size of each block:\n");
    for(i=0;i<nb;i++)
    {
        printf("Block %d:",i+1);
        scanf("%d",&block[i]);
    }
    printf("\nEnter the number of files:");
    scanf("%d",&nf);
    printf("\nEnter the size of each file:\n");
    for(i=0;i<nf;i++)
    {
        printf("File %d:",i+1);
        scanf("%d",&files[i]);
    }
    int ch;
    printf("\n1.Worst Fit\n2.First Fit\n3.Best Fit\nEnter your choice:");
    scanf("%d",&ch);
    switch(ch)
    {
        case 1:worstfit();
        break;
        case 2:firstfit();
        break;
        case 3:bestfit();
        break;
        default:return 0;
    }
}
```

## Output

```
C:\Users\admin\CS4SEM>gcc os.c

C:\Users\admin\CS4SEM>a
Enter the number of blocks:3
Enter the size of each block:
Block 1:5
Block 2:2
Block 3:7

Enter the number of files:2

Enter the size of each file:
File 1:1
File 2:4

1.Worst Fit
2.First Fit
3.Best Fit
Enter your choice:1
Worst Fit Memory Allocation
File Number     File Size       Block Number    Block Size      Fragment
1               1               3               7               6
2               4               1               5               1
```

```
C:\Users\admin\CS4SEM>a
Enter the number of blocks:3
Enter the size of each block:
Block 1:5
Block 2:2
Block 3:7

Enter the number of files:2

Enter the size of each file:
File 1:1
File 2:4

1.Worst Fit
2.First Fit
3.Best Fit
Enter your choice:2
First Fit Memory Allocation
File Number     File Size       Block Number    Block Size      Fragment
1               1               1               5               4
2               4               1               4               0
```

```
C:\Users\admin\CS4SEM>a
Enter the number of blocks:3
Enter the size of each block:
Block 1:5
Block 2:2
Block 3:7

Enter the number of files:2

Enter the size of each file:
File 1:1
File 2:4

1.Worst Fit
2.First Fit
3.Best Fit
Enter your choice:3
Best Fit Memory Allocation
File Number     File Size       Block Number    Block Size      Fragment
1               1               2               2               1
2               4               1               5               1
```

## Laboratory Program 10

Write a C program to simulate paging technique of memory management.

```
#include<stdio.h>
int main()
{
  int memsize,pagesize,i,j,procpages=4,nproc;
  printf("Enter the memory size:");
  scanf("%d",&memsize);
  printf("\nEnter the page size of main memory:");
  scanf("%d",&pagesize);
  int npages=memsize/pagesize;
  printf("\nEnter the number of processes:");
  scanf("%d",&nproc);
  int processes[nproc][procpages];
  int frame[nproc];
  int  phymem[50][50];
  printf("Enter the page frame:\n");
  for(i=0;i<nproc;i++)
  {
        printf("Process %d:",i+1);
        scanf("%d",&frame[i]);
  }
  for(i=0;i<nproc;i++)
  {
        /*printf("Enter the number of pages required for Process %d:",i+1);
        scanf("%d",&procpages);*/
        printf("\nEnter the page table for Process %d:\n",i+1);
        for(j=0;j<procpages;j++)
        {
        scanf("%d",&processes[i][j]);
        }
  }
  char logi,a;
  while(1)
  {
        printf("\nEnter the logical address:");
        scanf("%c",&a);
        scanf("%c",&logi);
        int logicaladd=(int)logi-97;

        if(logicaladd<=-1||logicaladd>=40)
        {
```

```
        printf("Invalid Logical address!\n");

        printf("Page number\tData\n");
        for(int x=0;x<npages;x++)
        {
        printf("%d\t\t",x);
        for(int y=0;y<4;y++)
        {
                printf("%c\n",(char)phymem[x][y]);
        }
        }

        return 0;
        }
        int offset=logicaladd%procpages;
        for(i=0;i<nproc;i++)
        {
        for(j=0;j<4;j++)
        {
        if(processes[i][j]==logicaladd)
        {
        int phy=(procpages*frame[i])+offset;
        //phyadd[frame[i]][phy]=logicaladd+97;
        printf("Physical address is %d",phy);
        }
        }
        }
  }
}
```

## Output

```
C:\Users\admin\CS4SEM>a
Enter the memory size:100

Enter the page size of main memory:4

Enter the number of processes:3
Enter the page frame:
Process 1:6
Process 2:5
Process 3:1

Enter the page table for Process 1:
0 1 2 3

Enter the page table for Process 2:
4 5 6 7

Enter the page table for Process 3:
8 9 10 11

Enter the logical address:a
Physical address is 24
Enter the logical address:d
Physical address is 27
Enter the logical address:e
Physical address is 20
Enter the logical address:f
Physical address is 21
Enter the logical address:j
Physical address is 5
Enter the logical address:k
Physical address is 6
Enter the logical address:
```

# Laboratory Program 11

Write a C program to simulate page replacement algorithms:
a) FIFO
b) LRU
c) Optimal

```
#include <stdio.h>

#define MAX_FRAMES 10
#define MAX_PAGES 100

int frames[MAX_FRAMES];
int pageQueue[MAX_FRAMES];
int pageQueueSize = 0;
int pageQueueFront = 0;

int findInFrames(int page, int numFrames) {
   for (int i = 0; i < numFrames; i++) {
      if (frames[i] == page) {
         return i;
      }
   }
   return -1;
}

void displayFrames(int numFrames) {
   printf("Current frames: ");
   for (int i = 0; i < numFrames; i++) {
      if (frames[i] != -1) {
         printf("%d ", frames[i]);
      }
   }
   printf("\n");
}

int findLRUIndex(int numFrames) {
   int index = 0;
   int min = pageQueueSize + 1;

   for (int i = 0; i < numFrames; i++) {
      int currentPage = frames[i];
      int j;
      for (j = pageQueueFront; j < pageQueueSize; j++) {
```

```
            if (pageQueue[j] == currentPage) {
                break;
            }
        }

        if (j < min) {
            min = j;
            index = i;
        }
    }

    return index;
}

int main() {
    int numFrames, numPages;

    printf("Enter the number of frames: ");
    scanf("%d", &numFrames);

    printf("Enter the number of pages: ");
    scanf("%d", &numPages);

    printf("Enter the page reference string:\n");
    for (int i = 0; i < numPages; i++) {
        scanf("%d", &pageQueue[i]);
        pageQueueSize++;
    }

    for (int i = 0; i < numFrames; i++) {
        frames[i] = -1;
    }

    int faultsFIFO = 0, faultsLRU = 0, faultsOptimal = 0;

    printf("\nFIFO Page Replacement Algorithm:\n");
    pageQueueFront = 0;
    for (int i = 0; i < numPages; i++) {
        int currentPage = pageQueue[i];
        if (findInFrames(currentPage, numFrames) == -1) {
            frames[pageQueueFront] = currentPage;
            pageQueueFront = (pageQueueFront + 1) % numFrames;
            displayFrames(numFrames);
            faultsFIFO++;
```

```
    }
}

printf("\nLRU Page Replacement Algorithm:\n");
pageQueueFront = 0;
for (int i = 0; i < numPages; i++) {
    int currentPage = pageQueue[i];
    if (findInFrames(currentPage, numFrames) == -1) {
        int index = findLRUIndex(numFrames);
        frames[index] = currentPage;
        pageQueue[pageQueueSize] = currentPage;
        pageQueueSize++;
        displayFrames(numFrames);
        faultsLRU++;
    }
}

printf("\nOptimal Page Replacement Algorithm:\n");
for (int i = 0; i < numFrames; i++) {
    frames[i] = -1;
}

for (int i = 0; i < numPages; i++) {
    int currentPage = pageQueue[i];
    if (findInFrames(currentPage, numFrames) == -1) {
        int optimalIndex = -1;
        int maxDistance = -1;

        for (int j = 0; j < numFrames; j++) {
            int nextPage = frames[j];
            int distance = -1;

            for (int k = i + 1; k < numPages; k++) {
                if (pageQueue[k] == nextPage) {
                    distance = k - i;
                    break;
                }
            }

            if (distance == -1) {
                optimalIndex = j;
                break;
            }
```

```
            if (distance > maxDistance) {
                maxDistance = distance;
                optimalIndex = j;
            }
        }

        frames[optimalIndex] = currentPage;
        displayFrames(numFrames);
        faultsOptimal++;
    }
  }

  printf("\nTotal Page Faults:\n");
  printf("FIFO: %d\n", faultsFIFO);
  printf("LRU: %d\n", faultsLRU);
  printf("Optimal: %d\n", faultsOptimal);

  return 0;
}
```

## Output

```
C:\Users\admin\CS4SEM>gcc os.c

C:\Users\admin\CS4SEM>a
Enter the number of frames: 4
Enter the number of pages: 10
Enter the page reference string:
7 0 1 2 3 4 3 4 0 2

FIFO Page Replacement Algorithm:
Current frames: 7
Current frames: 7 0
Current frames: 7 0 1
Current frames: 7 0 1 2
Current frames: 3 0 1 2
Current frames: 3 4 1 2
Current frames: 3 4 0 2

LRU Page Replacement Algorithm:
Current frames: 3 4 7 2
Current frames: 3 4 0 2
Current frames: 3 4 1 2
Current frames: 3 4 0 2
Current frames: 1 4 0 2
```

```
LRU Page Replacement Algorithm:
Current frames: 3 4 7 2
Current frames: 3 4 0 2
Current frames: 3 4 1 2
Current frames: 3 4 0 2
Current frames: 1 4 0 2

Optimal Page Replacement Algorithm:
Current frames: 7
Current frames: 1
Current frames: 1 2
Current frames: 1 3
Current frames: 1 3 4
Current frames: 1 0 4

Total Page Faults:
FIFO: 7
LRU: 5
Optimal: 6
```

## Laboratory Program 12

Write a C program to simulate the following file allocation strategies:
a) Sequential
b) Indexed
c) Linked

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

#define MAX_BLOCKS 100
#define MAX_FILES 10

// Data structures
struct File {
    int size;
    int blocks[MAX_BLOCKS];
};

struct IndexedFile {
    int size;
    int index_block;
};

struct LinkedBlock {
    int data;
    int next_block;
};

// Functions for Sequential File Allocation
void allocateSequential(struct File files[], int num_files, int total_blocks) {
    int current_block = 0;

    printf("\nSequential File Allocation:\n");
    for (int i = 0; i < num_files; i++) {
        if (current_block + files[i].size <= total_blocks) {
            for (int j = current_block; j < current_block + files[i].size; j++) {
                files[i].blocks[j - current_block] = j;
            }
            current_block +=  files[i].size;
            printf("File %d allocated blocks: ", i + 1);
            for (int j = 0; j < files[i].size; j++) {
                printf("%d ", files[i].blocks[j]);
            }
```

```
            printf("\n");
        } else {
            printf("File %d cannot be allocated due to insufficient space.\n", i + 1);
        }
    }
}

// Functions for Indexed File Allocation
void allocateIndexed(struct IndexedFile files[], int num_files, int total_blocks, int index_blocks) {
    int current_block = index_blocks;

    printf("\nIndexed File Allocation:\n");
    for (int i = 0; i < num_files; i++) {
        if (current_block < total_blocks) {
            files[i].index_block = current_block;
            current_block++;
            printf("File %d index block: %d\n", i + 1, files[i].index_block);
        } else {
            printf("File %d cannot be allocated due to insufficient index space.\n", i + 1);
        }
    }
}

// Functions for Linked File Allocation
void allocateLinked(struct LinkedBlock blocks[], int num_blocks, struct File files[], int num_files)
{
    int current_block = 0;

    printf("\nLinked File Allocation:\n");
    for (int i = 0; i < num_files; i++) {
        if (current_block + files[i].size <= num_blocks) {
            for (int j = 0; j < files[i].size; j++) {
                blocks[current_block + j].data = i + 1;
                blocks[current_block + j].next_block = (j == files[i].size - 1) ? -1 : current_block + j + 1;
            }
            current_block += files[i].size;
            printf("File %d allocated blocks:\n", i + 1);
            for (int j = 0; j < files[i].size; j++) {
                printf("Block %d: Data %d, Next Block %d\n", current_block - files[i].size + j + 1,
blocks[current_block - files[i].size + j].data, blocks[current_block - files[i].size + j].next_block);
            }
        } else {
            printf("File %d cannot be allocated due to insufficient space.\n", i + 1);
        }
```

```
    }
}

int main() {
    int total_blocks, index_blocks;

    printf("Enter the total number of blocks: ");
    scanf("%d", &total_blocks);

    printf("Enter the number of index blocks for indexed allocation: ");
    scanf("%d", &index_blocks);

    int num_files;
    printf("Enter the number of files (up to %d): ", MAX_FILES);
    scanf("%d", &num_files);

    struct File files[MAX_FILES];
    struct IndexedFile indexedFiles[MAX_FILES];
    struct LinkedBlock blocks[MAX_BLOCKS];

    printf("Enter the size of each file:\n");
    for (int i = 0; i < num_files; i++) {
        printf("File %d: ", i + 1);
        scanf("%d", &files[i].size);
        for (int j = 0; j < MAX_BLOCKS; j++) {
            files[i].blocks[j] = -1;
        }
    }

    allocateSequential(files, num_files, total_blocks);
    allocateIndexed(indexedFiles, num_files, total_blocks, index_blocks);
    allocateLinked(blocks, total_blocks, files, num_files);

    return 0;
}
```

## Output

```
C:\Users\admin\CS4SEM>gcc os.c

C:\Users\admin\CS4SEM>a
Enter the total number of blocks: 4
Enter the number of index blocks for indexed allocation: 1
Enter the number of files (up to 10): 3
Enter the size of each file:
File 1: 12
File 2: 4
File 3: 6

Sequential File Allocation:
File 1 cannot be allocated due to insufficient space.
File 2 allocated blocks: 0 1 2 3
File 3 cannot be allocated due to insufficient space.

Indexed File Allocation:
File 1 index block: 1
File 2 index block: 2
File 3 index block: 3
```

```
Linked File Allocation:
File 1 cannot be allocated due to insufficient space.
File 2 allocated blocks:
Block 1: Data 2, Next Block 1
Block 2: Data 2, Next Block 2
Block 3: Data 2, Next Block 3
Block 4: Data 2, Next Block -1
File 3 cannot be allocated due to insufficient space.
```

## Laboratory Program 13

Write a C program to simulate the following file organization techniques:
a) Single level directory
b) Two level directory
c) Hierarchical

```
#include <stdio.h>
#include <string.h>

#define MAX_FILES 100

struct File {
   char name[20];
   int size;
};

struct SingleLevelDirectory {
   struct File files[MAX_FILES];
   int num_files;
};

struct TwoLevelDirectory {
   struct File files[MAX_FILES];
   int num_files;
   struct Directory1 {
      char  name[20];
      int num_files;
   } directories[MAX_FILES];
   int num_directories;
};

struct HierarchicalDirectory {
   struct File files[MAX_FILES];
   int num_files;
   struct Directory2 {
      char  name[20];
      int num_files;
      struct SubDirectory {
         char  name[20];
         int num_files;
      } subdirectories[MAX_FILES];
      int num_subdirectories;
```

```
    } directories[MAX_FILES];
    int num_directories;
};

int main() {
    int choice;

    printf("Select file organization technique:\n");
    printf("1. Single level directory\n");
    printf("2. Two level directory\n");
    printf("3. Hierarchical directory\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1: {
            struct SingleLevelDirectory dir;
            dir.num_files = 0;

            int num_files;
            printf("Enter the number of files: ");
            scanf("%d", &num_files);

            printf("Enter file details:\n");
            for (int i = 0; i < num_files; i++) {
                printf("File %d name: ", i + 1);
                scanf("%s", dir.files[i].name);
                printf("File %d size: ", i + 1);
                scanf("%d", &dir.files[i].size);
                dir.num_files++;
            }

            printf("\nSingle Level Directory:\n");
            for (int i = 0; i < dir.num_files; i++) {
                printf("File name: %s, Size: %d\n", dir.files[i].name, dir.files[i].size);
            }

            break;
        }
        case 2: {
            struct TwoLevelDirectory dir;
            dir.num_files = 0;
            dir.num_directories = 0;
```

```
    int num_files, num_directories;
    printf("Enter the number of directories: ");
    scanf("%d", &num_directories);
    dir.num_directories = num_directories;

    for (int i = 0; i < num_directories; i++) {
        printf("Enter directory %d name: ", i + 1);
        scanf("%s", dir.directories[i].name);
        dir.directories[i].num_files = 0;

        printf("Enter the number of files in directory %s: ", dir.directories[i].name);
        scanf("%d", &num_files);

        printf("Enter file details:\n");
        for (int j = 0; j < num_files; j++) {
            printf("File %d name: ", j + 1);
            scanf("%s", dir.files[dir.num_files].name);
            printf("File %d size: ", j + 1);
            scanf("%d", &dir.files[dir.num_files].size);
            dir.directories[i].num_files++;
            dir.num_files++;
        }
    }

    printf("\nTwo Level Directory:\n");
    for (int i = 0; i < num_directories; i++) {
        printf("Directory name: %s\n", dir.directories[i].name);
        for (int j = 0; j < dir.directories[i].num_files; j++) {
            printf("File name: %s, Size: %d\n", dir.files[j].name, dir.files[j].size);
        }
    }

    break;
}
case 3: {
    struct HierarchicalDirectory dir;
    dir.num_files = 0;
    dir.num_directories = 0;

    int num_files, num_directories;
    printf("Enter the number of directories: ");
    scanf("%d", &num_directories);
    dir.num_directories = num_directories;
```

```
        for (int i = 0; i < num_directories; i++) {
          printf("Enter directory %d name: ", i + 1);
          scanf("%s", dir.directories[i].name);
          dir.directories[i].num_files = 0;
          dir.directories[i].num_subdirectories = 0;

          printf("Enter the number of subdirectories in directory %s: ", dir.directories[i].name);
          scanf("%d", &dir.directories[i].num_subdirectories);

          for (int j = 0; j < dir.directories[i].num_subdirectories; j++) {
            printf("Enter subdirectory %d name: ", j + 1);
            scanf("%s", dir.directories[i].subdirectories[j].name);
            dir.directories[i].subdirectories[j].num_files = 0;

            printf("Enter the number of files in subdirectory %s: ",
dir.directories[i].subdirectories[j].name);
            scanf("%d", &num_files);

            printf("Enter file details:\n");
            for (int k = 0; k < num_files; k++) {
              printf("File %d name: ", k + 1);
              scanf("%s", dir.files[dir.num_files].name);
              printf("File %d size: ", k + 1);
              scanf("%d", &dir.files[dir.num_files].size);
              dir.directories[i].subdirectories[j].num_files++;
              dir.directories[i].num_files++;
              dir.num_files++;
            }
          }
        }

        printf("\nHierarchical Directory:\n");
        for (int i = 0; i < num_directories; i++) {
          printf("Directory name: %s\n", dir.directories[i].name);
          for (int j = 0; j < dir.directories[i].num_subdirectories; j++) {
            printf("Subdirectory name: %s\n", dir.directories[i].subdirectories[j].name);
            for (int k = 0; k < dir.directories[i].subdirectories[j].num_files; k++) {
              printf("File name: %s, Size: %d\n", dir.files[k].name, dir.files[k].size);
            }
          }
        }

        break;
      }
```

```
        default:
            printf("Invalid choice.\n");
            break;
    }

    return 0;
}
```

## Output

```
C:\Users\admin\CS4SEM>a
Select file organization technique:
1. Single level directory
2. Two level directory
3. Hierarchical directory
Enter your choice: 1
Enter the number of files: 4
Enter file details:
File 1 name: file1
File 1 size: 12
File 2 name: file2
File 2 size: 36
File 3 name: file3
File 3 size: 25
File 4 name: file4
File 4 size: 20

Single Level Directory:
File name: file1, Size: 12
File name: file2, Size: 36
File name: file3, Size: 25
File name: file4, Size: 20
```

```
C:\Users\admin\CS4SEM>a
Select file organization technique:
1. Single level directory
2. Two level directory
3. Hierarchical directory
Enter your choice: 2
Enter the number of directories: 2
Enter directory 1 name: direc1
Enter the number of files in directory direc1: 2
Enter file details:
File 1 name: file1direc1
File 1 size: 12
File 2 name: file2direc1
File 2 size: 25
Enter directory 2 name: direc2
Enter the number of files in directory direc2: 1
Enter file details:
File 1 name: file1direc2
File 1 size: 30

Two Level Directory:
Directory name: direc1
File name: file1direc1, Size: 12
File name: file2direc1, Size: 25
Directory name: direc2
File name: file1direc1, Size: 12
```

```
C:\Users\admin\CS4SEM>a
Select file organization technique:
1. Single level directory
2. Two level directory
3. Hierarchical directory
Enter your choice: 3
Enter the number of directories: 3
Enter directory 1 name: user1
Enter the number of subdirectories in directory user1: 2
Enter subdirectory 1 name: user1child1
Enter the number of files in subdirectory user1child1: 0
Enter file details:
Enter subdirectory 2 name: user1child2
Enter the number of files in subdirectory user1child2: 1
Enter file details:
File 1 name: file1_user1child2
File 1 size: 12
Enter directory 2 name: user2
Enter the number of subdirectories in directory user2: 0
Enter directory 3 name: user3
Enter the number of subdirectories in directory user3: 0

Hierarchical Directory:
Directory name: user1
Subdirectory name: user1child1
Subdirectory name: user1child2
File name: file1_user1child2, Size: 12
Directory name: user2
Directory name: user3
```

## Laboratory Program 14

Write a C program to simulate disk scheduling algorithms:
a) FCFS
b) SCAN
c) C-SCAN

```
#include<stdio.h>
#include<stdlib.h>
void fcfs()
{
   int RQ[100],i,n,TotalHeadMoment=0,initial;
   printf("Enter the number of Requests\n");
   scanf("%d",&n);
   printf("Enter the Requests sequence\n");
   for(i=0;i<n;i++)
    scanf("%d",&RQ[i]);
   printf("Enter initial head position\n");
   scanf("%d",&initial);

   // logic for FCFS disk scheduling

   for(i=0;i<n;i++)
   {
      TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
      initial=RQ[i];
   }

   printf("Total head moment is %d",TotalHeadMoment);
}
void scan()
{
   int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
   printf("Enter the number of Requests\n");
   scanf("%d",&n);
   printf("Enter the Requests sequence\n");
   for(i=0;i<n;i++)
    scanf("%d",&RQ[i]);
   printf("Enter initial head position\n");
   scanf("%d",&initial);
   printf("Enter total disk size\n");
   scanf("%d",&size);
   printf("Enter the head movement direction for high 1 and for low 0\n");
   scanf("%d",&move);
```

```
// logic for Scan disk scheduling

   /*logic for sort the request array */
for(i=0;i<n;i++)
{
   for(j=0;j<n-i-1;j++)
   {
      if(RQ[j]>RQ[j+1])
      {
         int temp;
         temp=RQ[j];
         RQ[j]=RQ[j+1];
         RQ[j+1]=temp;
      }


   }
}


int index;
for(i=0;i<n;i++)
{
   if(initial<RQ[i])
   {
      index=i;
      break;
   }
}

// if movement is towards high value
if(move==1)
{
   for(i=index;i<n;i++)
   {
      TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
      initial=RQ[i];
   }
   // last movement for max size
   TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
   initial = size-1;
   for(i=index-1;i>=0;i--)
   {
```

```
          TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
          initial=RQ[i];

      }
    }
    // if movement is towards low value
    else
    {
      for(i=index-1;i>=0;i--)
      {
         TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
         initial=RQ[i];
      }
      // last movement for min size
      TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
      initial =0;
      for(i=index;i<n;i++)
      {
          TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
          initial=RQ[i];

      }
    }

    printf("Total head movement is %d",TotalHeadMoment);
}


void cscan()
{
    int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
    printf("Enter the number of Requests\n");
    scanf("%d",&n);
    printf("Enter the Requests sequence\n");
    for(i=0;i<n;i++)
     scanf("%d",&RQ[i]);
    printf("Enter initial head position\n");
    scanf("%d",&initial);
    printf("Enter total disk size\n");
    scanf("%d",&size);
    printf("Enter the head movement direction for high 1 and for low 0\n");
    scanf("%d",&move);

    // logic for C-Scan disk scheduling
```

```
  /*logic for sort the request array */
for(i=0;i<n;i++)
{
  for( j=0;j<n-i-1;j++)
  {
    if(RQ[j]>RQ[j+1])
    {
      int temp;
      temp=RQ[j];
      RQ[j]=RQ[j+1];


      RQ[j+1]=temp;
    }

  }
}

int index;
for(i=0;i<n;i++)
{
  if(initial<RQ[i])
  {
    index=i;
    break;
  }
}

// if movement is towards high value
if(move==1)
{
  for(i=index;i<n;i++)
  {
    TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
    initial=RQ[i];
  }
  // last movement for max size
  TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
  /*movement max to min disk */
  TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
  initial=0;
  for( i=0;i<index;i++)
  {
```

```
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];

        }
    }
    // if movement is towards low value
    else
    {
        for(i=index-1;i>=0;i--)
        {
           TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
           initial=RQ[i];
        }
        // last movement for min size
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
        /*movement min to max disk */
        TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
        initial =size-1;
        for(i=n-1;i>=index;i--)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];

        }
    }

    printf("Total head movement is %d",TotalHeadMoment);

}
int main()
{
    int i,j,ch;
            printf(" 1.FCFS \n 2.SCAN\n 3.C SCAN\n Enter your choice:");
            scanf("%d",&ch);
            switch(ch)
            {
                        case 1:fcfs();
                        break;
                        case 2:scan();
                        break;
                        case 3:cscan();
                        break;
                        default:
                        return 0;
```

```
    }
  return 0;

}
```

## Output

```
C:\Users\admin\CS4SEM>gcc os.c

C:\Users\admin\CS4SEM>a
 1.FCFS
 2.SCAN
 3.C SCAN
 Enter your choice:1
Enter the number of Requests
5
Enter the Requests sequence
53 19 69 124 193
Enter initial head position
14
Total head moment is 247
C:\Users\admin\CS4SEM>
```

```
C:\Users\admin\CS4SEM>a
 1.FCFS
 2.SCAN
 3.C SCAN
 Enter your choice:2
Enter the number of Requests
5
Enter the Requests sequence
168 93 50 14 155
Enter initial head position
45
Enter total disk size
122
Enter the head movement direction for high 1 and for low 0
1
Total head movement is 277
C:\Users\admin\CS4SEM>
```

```
C:\Users\admin\CS4SEM>a
 1.FCFS
 2.SCAN
 3.C SCAN
 Enter your choice:3
Enter the number of Requests
5
Enter the Requests sequence
193 188 45 12 56
Enter initial head position
33
Enter total disk size
200
Enter the head movement direction for high 1 and for low 0
0
Total head movement is 386
C:\Users\admin\CS4SEM>
```

## Laboratory Program 15

Write a C program to simulate disk scheduling algorithms:
a) SSTF
b) LOOK
c) C-LOOK

```
#include<stdio.h>
#include<stdlib.h>
void sstf()
{
    int RQ[100],i,n,TotalHeadMoment=0,initial,count=0;
    printf("Enter the number of Requests\n");
    scanf("%d",&n);
    printf("Enter the Requests sequence\n");
    for(i=0;i<n;i++)
     scanf("%d",&RQ[i]);
    printf("Enter initial head position\n");
    scanf("%d",&initial);

    // logic for sstf disk scheduling

        /* loop will execute until all process is completed*/
    while(count!=n)
    {
        int min=1000,d,index;
        for(i=0;i<n;i++)
        {
           d=abs(RQ[i]-initial);
           if(min>d)
           {
               min=d;
               index=i;
           }

        }
        TotalHeadMoment=TotalHeadMoment+min;
        initial=RQ[index];
        // 1000 is for max
        // you can use any number
        RQ[index]=1000;
        count++;
    }
```

```
    printf("Total head movement is %d",TotalHeadMoment);
}
void look()
{
    int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
    printf("Enter the number of Requests\n");
    scanf("%d",&n);
    printf("Enter the Requests sequence\n");
    for(i=0;i<n;i++)
     scanf("%d",&RQ[i]);
    printf("Enter initial head position\n");
    scanf("%d",&initial);
    printf("Enter total disk size\n");
    scanf("%d",&size);
    printf("Enter the head movement direction for high 1 and for low 0\n");
    scanf("%d",&move);

    // logic for look disk scheduling

        /*logic for sort the request array */
    for(i=0;i<n;i++)
    {
        for(j=0;j<n-i-1;j++)
        {
            if(RQ[j]>RQ[j+1])
            {
                int temp;
                temp=RQ[j];
                RQ[j]=RQ[j+1];
                RQ[j+1]=temp;
            }

        }
    }

    int index;
    for(i=0;i<n;i++)
    {
        if(initial<RQ[i])
        {
            index=i;
            break;
        }
    }
```

```
   // if movement is towards high value
   if(move==1)
   {
      for(i=index;i<n;i++)
      {
         TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
         initial=RQ[i];
      }

      for(i=index-1;i>=0;i--)
      {
          TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
          initial=RQ[i];

      }
   }
   // if movement is towards low value
   else
   {
      for(i=index-1;i>=0;i--)
      {
         TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
         initial=RQ[i];
      }

      for(i=index;i<n;i++)
      {
          TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
          initial=RQ[i];

      }
   }

   printf("Total head movement is %d",TotalHeadMoment);
}


void clook()
{
   int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
   printf("Enter the number of Requests\n");
   scanf("%d",&n);
   printf("Enter the Requests sequence\n");
```

```
for(i=0;i<n;i++)
 scanf("%d",&RQ[i]);
printf("Enter initial head position\n");
scanf("%d",&initial);
printf("Enter total disk size\n");
scanf("%d",&size);
printf("Enter the head movement direction for high 1 and for low 0\n");
scanf("%d",&move);

// logic for C-look disk scheduling

   /*logic for sort the request array */
for(i=0;i<n;i++)
{
   for( j=0;j<n-i-1;j++)
   {
      if(RQ[j]>RQ[j+1])
      {
         int temp;
         temp=RQ[j];
         RQ[j]=RQ[j+1];
         RQ[j+1]=temp;
      }


   }
}


int index;
for(i=0;i<n;i++)
{
   if(initial<RQ[i])
   {
      index=i;
      break;
   }
}

// if movement is towards high value
if(move==1)
{
   for(i=index;i<n;i++)
   {
```

```
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];
        }

        for( i=0;i<index;i++)
        {
             TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
             initial=RQ[i];

        }
    }
    // if movement is towards low value
    else
    {
        for(i=index-1;i>=0;i--)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];
        }

        for(i=n-1;i>=index;i--)
        {
             TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
             initial=RQ[i];

        }
    }

    printf("Total head movement is %d",TotalHeadMoment);

}
int main()
{
    int i,j,ch;
            printf(" 1.SSTF \n 2.LOOK\n 3.C LOOK\n Enter your choice:");
            scanf("%d",&ch);
            switch(ch)
            {
                        case 1:sstf();
                        break;
                        case 2:look();
                        break;
                        case 3:clook();
                        break;
```

```
                    default:
                    return 0;
            }
    return 0;

}
```

## Output

```
C:\Users\admin\CS4SEM>gcc os.c

C:\Users\admin\CS4SEM>a
 1.SSTF
 2.LOOK
 3.C LOOK
 Enter your choice:1
Enter the number of Requests
5
Enter the Requests sequence
193 55 162 14 78
Enter initial head position
25
Total head movement is 190
C:\Users\admin\CS4SEM>
```

```
C:\Users\admin\CS4SEM>a
 1.SSTF
 2.LOOK
 3.C LOOK
 Enter your choice:2
Enter the number of Requests
5
Enter the Requests sequence
144 25 65 88 147
Enter initial head position
50
Enter total disk size
200
Enter the head movement direction for high 1 and for low 0
0
Total head movement is 147
C:\Users\admin\CS4SEM>
```

```
C:\Users\admin\CS4SEM>a
 1.SSTF
 2.LOOK
 3.C LOOK
 Enter your choice:3
Enter the number of Requests
5
Enter the Requests sequence
156 75 34 91 166
Enter initial head position
50
Enter total disk size
200
Enter the head movement direction for high 1 and for low 0
1
Total head movement is 248
C:\Users\admin\CS4SEM>
```