VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Operating Systems

Submitted by

Amrutha Muralidhar (1BM21CS257)

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 Sept-2023

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "**Operating Systems**" carried out by **Amrutha Muralidhar (1BM21CS257),** who is 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 2023. 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.

Dr. Nandhini Vineeth
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

`

Index

Sl. No.	Date	Experiment Title	Page No.
1	18/06/23	Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time. 1 FCFS 2 SJF (pre-emptive & Non-pre-emptive)	1
2	22/06/23	Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time. 1 Priority (pre-emptive & Non-pre-emptive) 2 Round Robin (Experiment with different quantum sizes for RR algorithm)	5
3	13/07/23	Write a C program to simulate 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	8
4	13/07/23	Write a C program to simulate Real-Time CPU Scheduling algorithms: a) Rate- Monotonic b) Earliest-deadline First c) Proportional scheduling	12
5	20/07/23	Write a C program to simulate producer-consumer problem using semaphores.	16
6	20/07/23	Write a C program to simulate the concept of Dining-Philosophers problem.	18
7	27/07/23	Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance	20
8	27/07/23	Write a C program to simulate deadlock detection	22
9	03/08/23	Write a C program to simulate the following contiguous memory allocation techniques a) Worst-fit b) Best-fit c) First-fit	25
10	10/08/23	Write a C program to simulate paging technique of memory management.	29
11	17/08/23	Write a C program to simulate page replacement algorithms a) FIFO b) LRU c) Optimal	31

12	17/08/23	Write a C program to simulate the following file allocation strategies. a) Sequential b) Indexed c) Linked	35
13	24/08/23	Write a C program to simulate the following file organization techniques a) Single level directory b) Two level directory c) Hierarchical	40
14	24/08/23	Write a C program to simulate disk scheduling algorithms a) FCFS b) SCAN c) C-SCAN	46
15	24/08/23	Write a C program to simulate disk scheduling algorithms a) SSTF b) LOOK c) c-LOOK	52

Course Outcome

CO1	Apply the different concepts and functionalities of Operating System
CO2	Analyse various Operating system strategies and techniques
CO3	Demonstrate the different functionalities of Operating System.
CO4	Conduct practical experiments to implement the functionalities of Operating system.

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

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
void calculateFCFSAverageTime() {
   int n, i;
   int bt[50], wt[50], tt[50];
   printf("Enter the number of processes: ");
   scanf("%d", &n);
   printf("Enter the burst time of each process:\n");
   for (i = 0; i < n; i++) {
      scanf("%d", &bt[i]);
   }
   wt[0] = 0;
   for (i = 1; i < n; i++) {
      wt[i] = wt[i - 1] + bt[i - 1];
   }

   for (i = 0; i < n; i++) {
      tt[i] = bt[i] + wt[i];
   }
   printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");
   for (i = 0; i < n; i++) {
      printf("%d\t%d\t\t%d\t\t%d\n", i, bt[i], wt[i], tt[i]);
   }
   float avg_wt = 0;
   for (i = 0; i < n; i++) {
      avg_wt += wt[i];
   }
   avg_wt /= n;
   printf("Average Waiting Time: %.2f\n", avg_wt);
   float avg_tt = 0;
   for (i = 0; i < n; i++) {
      avg_tt += tt[i];
   }
   avg_tt /= n;
   printf("Average Turnaround Time: %.2f\n", avg_tt);
}
```

```
void calculateSJFAverageTime() {
    int n, i, j;
    int bt[50], wt[50], tt[50];
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    printf("Enter the burst time of each process:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &bt[i]);
    }
    for (i = 0; i < n; i++) {
        int minIndex = i;
        for (j = i + 1; j < n; j++) {
            if (bt[j] < bt[minIndex]) {
                minIndex = j;
            }
        }
        int temp = bt[minIndex];
        bt[minIndex] = bt[i];
        bt[i] = temp;
        temp = wt[minIndex];
        wt[minIndex] = wt[i];
        wt[i] = temp;
    }
    wt[0] = 0;
    for (i = 1; i < n; i++) {
        wt[i] = wt[i - 1] + bt[i - 1];
    }
    for (i = 0; i < n; i++) {
        tt[i] = bt[i] + wt[i];}
    printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\n", i, bt[i], wt[i], tt[i]);}
    float avg_wt = 0;
    for (i = 0; i < n; i++) {
        avg_wt += wt[i];    }
    avg_wt /= n;
    printf("Average Waiting Time: %.2f\n", avg_wt);
    float avg_tt = 0;
    for (i = 0; i < n; i++) {
        avg_tt += tt[i];}
```

```
    avg_tt /= n;
    printf("Average Turnaround Time: %.2f\n", avg_tt);}
int main() {
    int choice;
    while (1) {
        printf("\nMenu:\n");
        printf("1. Calculate Average Time using FCFS\n");
        printf("2. Calculate Average Time using SJF\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                calculateFCFSAverageTime();
                break;
            case 2:
                calculateSJFAverageTime();
                break;
            case 3:
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        } }
    return 0;}
```

Output

```
/Amrutha/fcfs_sjf.c
Menu:
1. Calculate Average Time using FCFS
2. Calculate Average Time using SJF
3. Exit
Enter your choice: 1
Enter the number of processes: 5
Enter the burst time of each process:
6 2 5 3 2
Process Burst Time  Waiting Time    Turnaround Time
0   6       0       6
1   2       6       8
2   5       8       13
3   3       13      16
4   2       16      18
Average Waiting Time: 8.60
Average Turnaround Time: 12.20
```

```
Menu:
1. Calculate Average Time using FCFS
2. Calculate Average Time using SJF
3. Exit
Enter your choice: 2
Enter the number of processes: 5
Enter the burst time of each process:
6 2 5 3 2
Process Burst Time  Waiting Time    Turnaround Time
0   2       0       2
1   2       2       4
2   3       4       7
3   5       7       12
4   6       12      18
Average Waiting Time: 5.00
Average Turnaround Time: 8.60
```

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

Source Code

```
#include <stdio.h>
#define MAX_PROCESSES 100
void roundRobinScheduling(int bt[], int n, int qt) {
   int wt[MAX_PROCESSES] = {0};
   int tat[MAX_PROCESSES] = {0};
   int rt[MAX_PROCESSES];
   for (int i = 0; i < n; i++) {
      rt[i] = bt[i];
   }

   int cp = 0;
   int current_time = 0;

   while (cp < n) {
      for (int i = 0; i < n; i++) {
         if (rt[i] > 0) {
            if (rt[i] <= qt) {
               current_time += rt[i];
               rt[i] = 0;
               tat[i] = current_time;
               cp++;
            } else {
               current_time += qt;
               rt[i] -= qt;
            }
            wt[i] = current_time - bt[i];
         }
      }
   }

   printf("Process\t| Burst Time\t| Waiting Time\t| Turnaround Time\n");
   for (int i = 0; i < n; i++) {
      printf("%d\t| %d\t\t| %d\t\t| %d\n", i + 1, bt[i], wt[i], tat[i]);
   }

   float avg_wt = 0, avg_tat = 0;
```

```
    for (int i = 0; i < n; i++) {
        avg_wt += wt[i];
        avg_tat += tat[i];
    }
    avg_wt /= n;
    avg_tat /= n;

    printf("Average Waiting Time: %.2f\n", avg_wt);
    printf("Average Turnaround Time: %.2f\n", avg_tat);
}
void priorityNonPreemptiveScheduling(int bt[], int priority[], int n) {
    int wt[MAX_PROCESSES] = {0};
    int tat[MAX_PROCESSES] = {0};
    int ct[MAX_PROCESSES] = {0};
    for (int i = 0; i < n; i++) {
        int min_priority = priority[i];
        int min_priority_index = i;

        for (int j = i + 1; j < n; j++) {
            if (priority[j] < min_priority) {
                min_priority = priority[j];
                min_priority_index = j;
            }
        }
        int temp = bt[i];
        bt[i] = bt[min_priority_index];
        bt[min_priority_index] = temp;

        temp = priority[i];
        priority[i] = priority[min_priority_index];
        priority[min_priority_index] = temp;
    }
    int current_time = 0;
    for (int i = 0; i < n; i++) {
        ct[i] = current_time;
        wt[i] = current_time;
        current_time += bt[i];
        tat[i] = current_time;
    }
    printf("Pno|Bt\t|Wt\t|Tat\n");
    for (int i = 0; i < n; i++) {
```

```
        printf("%d  |%d\t|%d\t|%d\n", i + 1, bt[i], wt[i], tat[i]);
    }
    float avg_wt = 0, avg_tat = 0;
    for (int i = 0; i < n; i++) {
        avg_wt += wt[i];
        avg_tat += tat[i];
    }
    avg_wt /= n;
    avg_tat /= n;
    printf("Average Waiting Time: %.2f\n", avg_wt);
    printf("Average Turnaround Time: %.2f\n", avg_tat);
}

int main() {
    int choice;
    int n, qt;
    int bt[MAX_PROCESSES];
    int priority[MAX_PROCESSES];
    printf("Menu:\n1. Round Robin Scheduling\n2. Priority Non-preemptive Scheduling\n");
    scanf("%d", &choice);
    switch (choice) {
        case 1:
            printf("Enter the number of processes: ");
            scanf("%d", &n);

            printf("Enter the time quantum: ");
            scanf("%d", &qt);

            printf("Enter burst time for each process:\n");
            for (int i = 0; i < n; i++) {
                scanf("%d", &bt[i]);
            }

            roundRobinScheduling(bt, n, qt);
            break;

        case 2:
            printf("Enter the number of processes: ");
            scanf("%d", &n);

            printf("Enter burst time and priority for each process:\n");
```

```
            for (int i = 0; i < n; i++) {
                scanf("%d %d", &bt[i], &priority[i]);
            }

            priorityNonPreemptiveScheduling(bt, priority, n);
            break;

        default:
            printf("Invalid choice!\n");
            return 0;
    }
    return 0;
}
```

Output

```
C:\Administrator\Documents>rr
Enter the number of processes
3
Enter the time quantum
4
Enter burst time
5
3
9
Pno | Bt    | Wt    | Tat
1   | 5     | 7     | 12
2   | 3     | 4     | 7
3   | 9     | 8     | 17
Average Waiting Time: 6.33
Average Turnaround Time: 12.00
C:\Administrator\Documents>rp.c
Menu:
1. Round Robin Scheduling
2. Priority Non-preemptive Scheduling
2
Enter the number of processes: 3
Enter burst time and priority for each process:
5 2
3 0
9 1
Pno|Bt  |Wt |Tat
1  |3   |0  |3
2  |9   |3  |12
3  |5   |12 |17
Average Waiting Time: 5.00
Average Turnaround Time: 10.67
```

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.

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
   int p[30], bt[30], su[30], wt[30], tat[30], arrival[30];
   int i, k, n, temp;
   float waiting_avg, turnaround_avg;
   int tr;
   int csource = 0;
   int cuser = 0;
   int btsource[30], btuser[30], puser[30], psource[30];
   printf("Enter the number of processes: ");
   scanf("%d", &n);
   for (i = 0; i < n; i++) {
      printf("System process/User Process (0/1): ");
      scanf("%d", &tr);
      printf("Enter the Burst Time of Process %d: ", i);
      if (tr == 1) {
         scanf("%d", &btuser[cuser]);
         printf("Enter the Arrival Time of Process %d: ", i);
         scanf("%d", &arrival[cuser]);
         puser[cuser] = i;
         cuser++;
      } else if (tr == 0) {
         scanf("%d", &btsource[csource]);
         printf("Enter the Arrival Time of Process %d: ", i);
         scanf("%d", &arrival[csource]);
         psource[csource] = i;
         csource++;
      }
   }
   for (i = 0; i < csource; i++) {
      p[i] = psource[i];
      bt[i] = btsource[i];
      su[i] = 0;
   }
```

```
for (i = 0; i < cuser; i++) {
   p[i + csource] = puser[i];
   bt[i + csource] = btuser[i];
   su[i + csource] = 1;
}
for (i = 0; i < n; i++) {
   printf("%d %d\n", p[i], bt[i]);
}
for (i = 0; i < n - 1; i++) {
   for (int j = 0; j < n - i - 1; j++) {
      if (arrival[j] > arrival[j + 1]) {
         temp = arrival[j];
         arrival[j] = arrival[j + 1];
         arrival[j + 1] = temp;

         temp = p[j];
         p[j] = p[j + 1];
         p[j + 1] = temp;

         temp = bt[j];
         bt[j] = bt[j + 1];
         bt[j + 1] = temp;

         temp = su[j];
         su[j] = su[j + 1];
         su[j + 1] = temp;
      }
   }
}
waiting_avg = wt[0] = 0;
turnaround_avg = tat[0] = bt[0];
for (i = 1; i < n; i++) {
   wt[i] = wt[i - 1] + bt[i - 1] - arrival[i];
   tat[i] = tat[i - 1] + bt[i] - arrival[i];
   waiting_avg = waiting_avg + wt[i];
   turnaround_avg = turnaround_avg + tat[i];
}
printf("\nP\t SYSTEM/USER\t At\t Bt\t Wt\t Tat\n");
for (i = 0; i < n; i++) {
   printf("%d\t\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d\n", p[i], su[i], arrival[i], bt[i], wt[i], tat[i]);
}
```

```
    printf("\nAverage Waiting Time: %.2f", waiting_avg / n);
    printf("\nAverage Turnaround Time: %.2f\n", turnaround_avg / n);
    return 0;
}
```

Output

```
Enter the number of processes: 3
System process/User Process (0/1): 0
Enter the Burst Time of Process 0: 1
Enter the Arrival Time of Process 0: 2
System process/User Process (0/1): 0
Enter the Burst Time of Process 1: 5
Enter the Arrival Time of Process 1: 0
System process/User Process (0/1): 1
Enter the Burst Time of Process 2: 3
Enter the Arrival Time of Process 2: 1
P    SYSTEM/USER     At  Bt  Wt  Tat
1           0           0           5           0           5
2           1           0           3           5           8
0           0           1           1           7           8

Average Waiting Time: 4.00
Average Turnaround Time: 7.00
```

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

Source Code:

```
#include <stdio.h>
#define MAX_PROCESSES 10
Int at[MAX_PROCESSES], bt[MAX_PROCESSES], dl[MAX_PROCESSES], p[MAX_PROCESSES], n, i;

void swap(int *a, int *b) {
        int temp;
        temp = *a;
        *a = *b;
        *b = temp;
}

void rateMonotonic() {
        int j;
        for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
        if (at[j] > at[j + 1]) { // Sort based on arrival time
                swap(&bt[j], &bt[j + 1]);
                swap(&at[j], &at[j + 1]);
                swap(&dl[j], &dl[j + 1]);
                swap(&p[j], &p[j + 1]);
        }
        }
        }
        for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
        if (bt[j] > bt[j + 1] && at[j] <= at[j + 1]) {
                swap(&bt[j], &bt[j + 1]);
                swap(&at[j], &at[j + 1]);
                swap(&dl[j], &dl[j + 1]);
                swap(&p[j], &p[j + 1]);
        }
        }
        }
        printf("\nRate-Monotonic Scheduling Order:\n");
        for (i = 0; i < n; i++) {
```

```
        printf("P%d ", p[i]);
        }
        printf("\n");
}

void earliestDeadlineFirst() {
        int j;
        for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
        if (dl[j] > dl[j + 1]) { // Sort based on deadlines
                swap(&bt[j], &bt[j + 1]);
                swap(&at[j], &at[j + 1]);
                swap(&dl[j], &dl[j + 1]);
                swap(&p[j], &p[j + 1]);
        } else if (dl[j] <= dl[j + 1] && at[j] > at[j + 1]) {
                // If deadlines are equal, sort based on arrival time
                if(bt[j]>bt[j+1]){
                swap(&bt[j], &bt[j + 1]);
                swap(&at[j], &at[j + 1]);
                swap(&dl[j], &dl[j + 1]);
                swap(&p[j], &p[j + 1]);}
        }
        }
        }
        printf("\nEarliest-Deadline First Scheduling Order:\n");
        for (i = 0; i < n; i++) {
        printf("P%d ", p[i]);
        }
        printf("\n");
}

void proportionalScheduling() {
        int totalBurstTime = 0;
        int i;
        float share[10];

        for (i = 0; i < n; i++) {
        totalBurstTime += bt[i];
        }

        for (i = 0; i < n; i++) {
```

```
        share[i] = (bt[i] * 100) / totalBurstTime;
        }
        printf("\nProportional Scheduling Order:\n");
        for (i = 0; i < n; i++) {
        printf("P%d gets %.2f%% CPU\n", p[i], share[i]);
        }
        printf("\n");
}

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

        printf("\nEnter the arrival time, burst time, and deadline for each process:\n");
        for (i = 0; i < n; i++) {
        printf("\nProcess P%d:\n", i);
        printf("Arrival Time: ");
        scanf("%d", &at[i]);
        printf("Burst Time: ");
        scanf("%d", &bt[i]);
        printf("Deadline: ");
        scanf("%d", &dl[i]);
        p[i] = i;
        }
        rateMonotonic();
        earliestDeadlineFirst();
        proportionalScheduling();

        return 0;
}
```

Output

```
bmscecse@bmscecse-HP-Pro-3330-MT:~$ cd 1BM21CS257
bmscecse@bmscecse-HP-Pro-3330-MT:~/1BM21CS257$ gcc rtos.c
bmscecse@bmscecse-HP-Pro-3330-MT:~/1BM21CS257$ ./a.out
Enter the number of processes: 5

Enter the arrival time, burst time, and deadline for each process:

Process P0:
Arrival Time: 5
Burst Time: 10
Deadline: 15

Process P1:
Arrival Time: 0
Burst Time: 22
Deadline: 30

Process P2:
Arrival Time: 0
Burst Time: 1
Deadline: 15

Process P3:
Arrival Time: 3
Burst Time: 4
Deadline: 6

Process P4:
Arrival Time: 7
Burst Time: 4
Deadline: 25

Rate-Monotonic Scheduling Order:
P2 P3 P4 P0 P1

Earliest-Deadline First Scheduling Order:
P3 P2 P0 P4 P1

Proportional Scheduling Order:
P0 gets 9.00% CPU
P1 gets 2.00% CPU
P2 gets 24.00% CPU
P3 gets 9.00% CPU
P4 gets 53.00% CPU

bmscecse@bmscecse-HP-Pro-3330-MT:~/1BM21CS257$
```

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

Source Code:

```
#include<stdio.h>
    #include<stdlib.h>
    int mutex=1,full=0,empty=3,x=0;
    int main()
    {
    int n;
    void producer();
    void consumer();
    int wait(int);
    int signal(int);
    printf("\n1.Producer\n2.Consumer\n3.Exit");
    while(1)
    {
    printf("\nEnter your choice:");
    scanf("%d",&n);
    switch(n)
    {
    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:
    exit(0);
    break;
    }
    }
    return 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 the 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);
}
```

Output

```
bmscecse@bmscecse-HP-Pro-3330-MT:~/1BM21CS257$ gcc pc.c
bmscecse@bmscecse-HP-Pro-3330-MT:~/1BM21CS257$ ./a.out

1.Producer
2.Consumer
3.Exit
Enter your choice:1

Producer produces the item 1
Enter your choice:1

Producer produces the item 2
Enter your choice:1

Producer produces the item 3
Enter your choice:1
Buffer is full!!
Enter your choice:2

Consumer consumes item 3
Enter your choice:2

Consumer consumes item 2
Enter your choice:2

Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
Enter your choice:3
bmscecse@bmscecse-HP-Pro-3330-MT:~/1BM21CS257$
```

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

Source Code:

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

sem_t room;
sem_t chopstick[5];

void * philosopher(void *);
void eat(int);
int main()
{
   int i,a[5];
   pthread_t tid[5];

   sem_init(&room,0,4);

   for(i=0;i<5;i++)
         sem_init(&chopstick[i],0,1);

   for(i=0;i<5;i++){
         a[i]=i;
         pthread_create(&tid[i],NULL,philosopher,(void *)&a[i]);
   }
   for(i=0;i<5;i++)
         pthread_join(tid[i],NULL);
}

void * philosopher(void * num)
{
   int phil=*(int *)num;

   sem_wait(&room);
   printf("\nPhilosopher %d has entered room",phil);
   sem_wait(&chopstick[phil]);
   sem_wait(&chopstick[(phil+1)%5]);

   eat(phil);
```

```
    sleep(2);
    printf("\nPhilosopher %d has finished eating",phil);

    sem_post(&chopstick[(phil+1)%5]);
    sem_post(&chopstick[phil]);
    sem_post(&room);
}
void eat(int phil)
{
    printf("\nPhilosopher %d is eating",phil);
}
```

Output

```
bmscecse@bmscecse-HP-Pro-3330-MT:~/1BM21CS257$ gcc pd.c
bmscecse@bmscecse-HP-Pro-3330-MT:~/1BM21CS257$ ./a.out

Philosopher 0 has entered room
Philosopher 0 is eating
Philosopher 3 has entered room
Philosopher 3 is eating
Philosopher 2 has entered room
Philosopher 1 has entered room
Philosopher 0 has finished eating
Philosopher 4 has entered room
Philosopher 3 has finished eating
Philosopher 2 is eating
Philosopher 4 is eating
Philosopher 2 has finished eating
Philosopher 4 has finished eating
Philosopher 1 is eating
Philosopher 1 has finished eatingbmscecse@bmscecse-HP-Pro-3330-MT:~/1BM21CS257$
```

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

Source Code:

```
#include <stdio.h>
int main() {
int
k=0,a=0,b=0,instance[5],availability[5],allocated[10][5],need[10][5],MAX[10][5],process,P[10],no_
of_resources, op[10], cnt=0,i, j;
   printf("\n Enter the number of resources : ");
   scanf("%d", &no_of_resources);
   printf("\n enter the max instances of each resources\n");
   for (i=0;i<no_of_resources;i++) {
         availability[i]=0;
         printf("%c= ",(i+97));
         scanf("%d",&instance[i]);
   }
   printf("\n Enter the number of processes : ");
   scanf("%d", &process);
   printf("\n Enter the allocation matrix \n     ");
   for (i=0;i<no_of_resources;i++)
   printf(" %c",(i+97));
   printf("\n");
   for (i=0;i <process;i++) {
         P[i]=i;
         printf("P[%d]  ",P[i]);
         for (j=0;j<no_of_resources;j++) {
               scanf("%d",&allocated[i][j]);
               availability[j]+=allocated[i][j];
         }
   }
   printf("\nEnter the MAX matrix \n            ");
   for (i=0;i<no_of_resources;i++) {
         printf(" %c",(i+97));
         availability[i]=instance[i]-availability[i];
   }
   printf("\n");
   for (i=0;i <process;i++) {
         printf("P[%d]  ",i);
         for (j=0;j<no_of_resources;j++)
          scanf("%d", &MAX[i][j]);
   }
   printf("\n");
```

```
    A: a=-1;
    for (i=0;i <process;i++) {
            cnt=0;
            b=P[i];
            for (j=0;j<no_of_resources;j++) {
                    need[b][j] = MAX[b][j]-allocated[b][j];
                    if(need[b][j]<=availability[j])
                     cnt++;
            }
            if(cnt==no_of_resources) {
                    op[k++]=P[i];
                    for (j=0;j<no_of_resources;j++)
                    availability[j]+=allocated[b][j];
            } else
             P[++a]=P[i];
    }
    if(a!=-1) {
            process=a+1;
            goto A;
    }
    printf("\t <");
    for (i=0;i<k;i++)
    printf(" P[%d] ",op[i]);
    printf(">");

    return 0;
}
```

Output:

```
        <>bmscecse@bmscecse-HP-Pro-3330-MT:~/1BM21CS257$ ./a.out

 Enter the number of resources : 3

 enter the max instances of each resources
a= 10
b= 5
c= 7

 Enter the number of processes : 5

 Enter the allocation matrix
      a b c
P[0]  0 1 0
P[1]  2 0 0
P[2]  3 0 2
P[3]  2 1 1
P[4]  0 0 2

Enter the MAX matrix
      a b c
P[0]  7 5 3
P[1]  3 2 2
P[2]  9 0 2
P[3]  4 2 2
P[4]  5 3 3

        < P[1]  P[3]  P[4]  P[0]  P[2] >bmscecse@bmscecse-HP-Pro-
```

8. Write a C program to simulate deadlock detection

Source Code

```
#include<stdio.h>
static int mark[20];
int i,j,np,nr;
int main(){
int alloc[10][10],request[10][10],avail[10],r[10],w[10];
printf("\nEnter the no of process: ");
scanf("%d",&np);
printf("\nEnter the no of resources: ");
scanf("%d",&nr);
for(i=0;i<nr;i++){
printf("\nTotal Amount of the Resource R%d: ",i+1);
scanf("%d",&r[i]);}
printf("\nEnter the request matrix:");
for(i=0;i<np;i++)
for(j=0;j<nr;j++)
scanf("%d",&request[i][j]);
printf("\nEnter the allocation matrix:");
for(i=0;i<np;i++)
for(j=0;j<nr;j++)
scanf("%d",&alloc[i][j]);
for(j=0;j<nr;j++){
avail[j]=r[j];
for(i=0;i<np;i++){
avail[j]-=alloc[i][j];}}
for(i=0;i<np;i++){
int count=0;
 for(j=0;j<nr;j++) {
      if(alloc[i][j]==0)
        count++;
      else
        break; }
 if(count==nr)
 mark[i]=1;}
for(j=0;j<nr;j++)
    w[j]=avail[j];
for(i=0;i<np;i++){
int canbeprocessed=0;
 if(mark[i]!=1){
   for(j=0;j<nr;j++) {
```

```
            if(request[i][j]<=w[j])
              canbeprocessed=1;
            else{
               canbeprocessed=0;
               break;} }
        if(canbeprocessed){
        mark[i]=1;
        for(j=0;j<nr;j++)
        w[j]+=alloc[i][j];}}}
        int deadlock=0;
        for(i=0;i<np;i++)
        if(mark[i]!=1)
        deadlock=1;
        if(deadlock)
        printf("\n Deadlock detected");
        else
        printf("\n No Deadlock possible");}
```

Output

```
Enter the no of process: 4
Enter the no of resources: 5

Total Amount of the Resource R1: 2
Total Amount of the Resource R2: 1
Total Amount of the Resource R3: 1
Total Amount of the Resource R4: 2
Total Amount of the Resource R5: 1

Enter the request matrix:0 1 0 0 1
0 0 1 0 1
0 0 0 0 1
1 0 1 0 1

Enter the allocation matrix:1 0 1 1 0
1 1 0 0 0
0 0 0 1 0
0 0 0 0 0

 Deadlock detected
--------------------------------
```

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

Source Code

```
#include <stdio.h>
#define max 25
void firstFit(int b[], int p[], int nb, int nf) {
   int i, j;
   printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
   for (i = 0; i < nf; i++) {
      for (j = 0; j < nb; j++) {
         if (b[j] >= p[i]) {
            printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i + 1, p[i], j + 1, b[j], b[j] - p[i]);
            b[j] -= p[i];
            break;
         }
      }
      if (j == nb) {
         printf("\n%d\t\t%d\t\tNot Allocated\t\t-\t\t-", i + 1, p[i]);
      }
   }
}
void bestFit(int b[], int p[], int nb, int nf) {
   int i, j, idx;
   printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
   for (i = 0; i < nf; i++) {
      idx = -1;
      for (j = 0; j < nb; j++) {
         if (b[j] >= p[i]) {
            if (idx == -1 || b[j] < b[idx]) {
               idx = j;
            }
         }
      }
      if (idx != -1) {
         printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i + 1, p[i], idx + 1, b[idx], b[idx] - p[i]);
         b[idx] -= p[i];
      } else {
         printf("\n%d\t\t%d\t\tNot Allocated\t\t-\t\t-", i + 1, p[i]);
      }
```

```
    }
}

void worstFit(int b[], int p[], int nb, int nf) {
    int i, j, idx;
    printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
    for (i = 0; i < nf; i++) {
        idx = -1;
        for (j = 0; j < nb; j++) {
            if (b[j] >= p[i]) {
                if (idx == -1 || b[j] > b[idx]) {
                    idx = j;
                }
            }
        }
        if (idx != -1) {
            printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i + 1, p[i], idx + 1, b[idx], b[idx] - p[i]);
            b[idx] -= p[i];
        } else {
            printf("\n%d\t\t%d\t\tNot Allocated\t\t-\t\t-", i + 1, p[i]);
        }
    }
}
int main() {
    int frag[max], b[max], p[max];
    int nb, nf, i, j, ch;
    printf("Enter the number of blocks: ");
    scanf("%d", &nb);
    printf("Enter the number of files: ");
    scanf("%d", &nf);
    printf("Enter the size of the blocks:\n");
    for (i = 0; i < nb; i++) {
        printf("Block %d: ", i + 1);
        scanf("%d", &b[i]);
    }
    printf("Enter the size of the files:\n");
    for (i = 0; i < nf; i++) {
        printf("File %d: ", i + 1);
        scanf("%d", &p[i]);
    }
    do {
```

```
            printf("\nMemory Allocation Techniques:\n");
            printf("1. First Fit\n");
            printf("2. Best Fit\n");
            printf("3. Worst Fit\n");
            printf("4. Exit\n");
            printf("Enter your choice: ");
            scanf("%d", &ch);
            switch (ch) {
                case 1:
                    firstFit(b, p, nb, nf);
                    break;
                case 2:
                    bestFit(b, p, nb, nf);
                    break;
                case 3:
                    worstFit(b, p, nb, nf);
                    break;
                case 4:
                    printf("Exiting...\n");
                    break;
                default:
                    printf("Invalid choice. Please enter a valid option.\n");
            }
        } while (ch != 4);
        return 0;
    }
```

Output

```
Enter the number of blocks: 10
Enter the number of files: 4
Enter the size of the blocks:
Block 1: 200
Block 2: 500
Block 3: 100
Block 4: 600
Block 5: 150
Block 6: 130
Block 7: 20
Block 8: 50
Block 9: 70
Block 10: 250
Enter the size of the files:
File 1: 150
File 2: 120
File 3: 480
File 4: 147
Memory Allocation Techniques:
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 1
File_no:    File_size : Block_no:   Block_size: Fragement
1       150     1       200     50
2       120     2       500     380
3       480     4       600     120
4       147     2       380     233
Memory Allocation Techniques:
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 2
File_no:    File_size : Block_no:   Block_size: Fragement
1       150     5       150     0
2       120     4       120     0
3       480     Not Allocated       -       -
4       147     2       233     86
```

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

Source Code

```
#include<stdio.h>
#define MAX 50
int main()
{
int page[MAX],i,n,f,ps,off,pno;
int choice=0;
printf("\nEnter the no of  peges in memory: ");
scanf("%d",&n);
printf("\nEnter page size: ");
scanf("%d",&ps);
printf("\nEnter no of frames: ");
scanf("%d",&f);
for(i=0;i<n;i++)
page[i]=-1;
printf("\nEnter the page table\n");
printf("(Enter frame no as -1 if that page is not present in any frame)\n\n");
printf("\npageno\tframeno\n-------\t-------");
for(i=0;i<n;i++)
{
printf("\n\n%d\t\t",i);
scanf("%d",&page[i]);
}
do
{
printf("\n\nEnter the logical address(i.e,page no & offset):");
scanf("%d%d",&pno,&off);
if(page[pno]==-1)
printf("\n\nThe required page is not available in any of frames");
else
printf("\n\nPhysical address(i.e,frame no & offset):%d,%d",page[pno],off);
printf("\nDo you want to continue(1/0)?:");
scanf("%d",&choice);
}while(choice==1);
return 1;
}
```

Output

```
Enter the no of  peges in memory: 4

Enter page size: 10

Enter no of frames: 10

Enter the page table
(Enter frame no as -1 if that page is not present in any frame)


pageno  frameno
------- -------

0               -1


1               8


2               -1


3               6


Enter the logical address(i.e,page no & offset):2 200


The required page is not available in any of frames
Do you want to continue(1/0)?:1


Enter the logical address(i.e,page no & offset):1 500


Physical address(i.e,frame no & offset):8,500
Do you want to continue(1/0)?:
```

11. Write a C program to simulate page replacement algorithms

a) FIFO

b) LRU

c) Optimal

Source Code

```
#include <stdio.h>

int findLRU(int time[], int n) {
    int min = time[0], index = 0;
    for (int i = 0; i < n; i++) {
        if (time[i] < min) {
            min = time[i];
            index = i;
        }
    }
    return index;
}

int main() {
    int n, frames, reference[100], pages[100], faults = 0;
    printf("Enter number of pages in reference string: ");
    scanf("%d", &n);

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

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

    int frame[frames], time[frames];
    for (int i = 0; i < frames; i++) {
        frame[i] = -1; // Initialize frames as empty
        time[i] = 0;   // Initialize time for LRU
    }

    printf("\nPage Replacement Algorithms:\n");
    printf("1. FIFO\n");
    printf("2. LRU\n");
    printf("3. Optimal\n");
```

```
printf("Enter your choice: ");
int choice;
scanf("%d", &choice);

switch (choice) {
    case 1: // FIFO
        for (int i = 0; i < n; i++) {
            int flag = 0;
            for (int j = 0; j < frames; j++) {
                if (frame[j] == reference[i]) {
                    flag = 1; // Page found in frames
                    break;
                }
            }

            if (flag == 0) { // Page not found
                frame[faults % frames] = reference[i];
                faults++;
            }
        }
        break;

    case 2: // LRU
        for (int i = 0; i < n; i++) {
            int flag = 0;
            for (int j = 0; j < frames; j++) {
                if (frame[j] == reference[i]) {
                    flag = 1; // Page found in frames
                    time[j] = i; // Update time for LRU
                    break;
                }
            }

            if (flag == 0) { // Page not found
                int index = findLRU(time, frames);
                frame[index] = reference[i];
                time[index] = i; // Update time for LRU
                faults++;
            }
        }
        break;
```

```
    case 3: // Optimal
        for (int i = 0; i < n; i++) {
            int flag = 0;
            for (int j = 0; j < frames; j++) {
                if (frame[j] == reference[i]) {
                    flag = 1; // Page found in frames
                    break;
                }
            }

            if (flag == 0) { // Page not found
                int max = -1, index = -1;
                for (int j = 0; j < frames; j++) {
                    int found = 0;
                    for (int k = i + 1; k < n; k++) {
                        if (frame[j] == reference[k]) {
                            found = 1;
                            if (k > max) {
                                max = k;
                                index = j;
                            }
                            break;
                        }
                    }
                    if (!found) {
                        index = j;
                        break;
                    }
                }
                frame[index] = reference[i];
                faults++;
            }
        }
        break;

    default:
        printf("Invalid choice!\n");
        return 0;
}
```

```
    printf("Number of Page Faults: %d\n", faults);

    return 0;
}
```

Output

case1

```
bmscecse@bmscecse-HP-Pro-3330-MT:~/1BM21CS257$ ./a.out
Enter no of pages:10
Enter the reference string:5 2 4 6 2 1 2 3 6 5
Enter no of frames:3

        5
        5       2
        5       2       4
        6       2       4
        6       2       1
        3       2       1
        3       2       6
        3       5       6

The no of page faults is 8bmscecse@bmscecse-HP-Pro-3330-MT:~/1BM21CS257$
```

case2

```
bmscecse@bmscecse-HP-Pro-3330-MT:~/1BM21CS257$ ./a.out
Incoming         Frame 1         Frame 2         Frame 3
4                        4                       -                       -
1                        4                       1                       -
2                        4                       1                       2
4                        4                       1                       2
5                        5                       1                       2
Total Page Faults:      4
bmscecse@bmscecse-HP-Pro-3330-MT:~/1BM21CS257$
```

case3

```
bmscecse@bmscecse-HP-Pro-3330-MT:~/1BM21CS257$ ./a.out
Stream Frame1 Frame2 Frame3
7               7               -               -
0               7               0               -
1               7               0               1
2               2               0               1
0               2               0               1
3               2               0               3
0               2               0               3
4               2               4               3
2               2               4               3
3               2               4               3
0               2               0               3
3               2               0               3
2               2               0               3
1               2               0               1
2               2               0               1
0               2               0               1
1               2               0               1
7               7               0               1
0               7               0               1
1               7               0               1

Hits: 11
Misses: 9bmscecse@bmscecse-HP-Pro-3330-MT:~/1BM21CS257$
```

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

Source Code

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

#define MAX_BLOCKS 50

void sequentialAllocation();
void indexedAllocation();
void linkedAllocation();

int f[MAX_BLOCKS];

int main() {
   int choice;
   for (int i = 0; i < MAX_BLOCKS; i++) {
      f[i] = 0;
   }

   while (1) {
      printf("\nFile Allocation Strategies:\n");
      printf("1. Sequential Allocation\n");
      printf("2. Indexed Allocation\n");
      printf("3. Linked Allocation\n");
      printf("4. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1:
            sequentialAllocation();
            break;

         case 2:
            indexedAllocation();
            break;

         case 3:
```

```
            linkedAllocation();
            break;

        case 4:
            printf("Exiting...\n");
            exit(0);

        default:
            printf("Invalid choice. Please select a valid option.\n");
      }
   }

   return 0;
}

void sequentialAllocation() {
   int st, len, count = 0;
   printf("Enter starting block and length of files: ");
   scanf("%d%d", &st, &len);

   for (int k = st; k < (st + len); k++) {
      if (f[k] == 0) {
         count++;
      }
   }

   if (len == count) {
      for (int j = st; j < (st + len); j++) {
         f[j] = 1;
         printf("%d\t%d\n", j, f[j]);
      }
      printf("The file is allocated to disk\n");
   } else {
      printf("The file is not allocated\n");
   }
}

void indexedAllocation() {
   int ind, n, count = 0;
   printf("Enter the index block: ");
   scanf("%d", &ind);
```

```
    if (f[ind] != 1) {
        printf("Enter no of blocks needed and no of files for the index %d on the disk: ", ind);
        scanf("%d", &n);
    } else {
        printf("%d index is already allocated\n", ind);
        return;
    }

    for (int i = 0; i < n; i++) {
        scanf("%d", &f[i]);
        if (f[i] == 0) {
            count++;
        }
    }

    if (count == n) {
        printf("Allocated\n");
        printf("File Indexed\n");
        for (int k = 0; k < n; k++) {
            printf("%d-------->%d : %d\n", ind, f[k], f[f[k]]);
        }
    } else {
        printf("File in the index is already allocated\n");
    }
}

void linkedAllocation() {
    int p, st, len, k, a;
    printf("Enter how many blocks already allocated: ");
    scanf("%d", &p);
    printf("Enter blocks already allocated: ");
    for (int i = 0; i < p; i++) {
        scanf("%d", &a);
        f[a] = 1;
    }

    printf("Enter index starting block and length: ");
    scanf("%d%d", &st, &len);

    k = len;
```

```
    if (f[st] == 0) {
        for (int j = st; j < (st + k); j++) {
            if (f[j] == 0) {
                f[j] = 1;
                printf("%d-------->%d\n", j, f[j]);
            } else {
                printf("%d Block is already allocated\n", j);
                k++;
            }
        }
    } else {
        printf("%d starting block is already allocated\n", st);
    }
}
```

Output

```
File Allocation Strategies:
1. Sequential Allocation
2. Indexed Allocation
3. Linked Allocation
4. Exit
Enter your choice: 1
Enter starting block and length of files: 25 3
25  1
26  1
27  1
The file is allocated to disk

File Allocation Strategies:
1. Sequential Allocation
2. Indexed Allocation
3. Linked Allocation
4. Exit
Enter your choice: 3
Enter how many blocks already allocated: 5
Enter blocks already allocated: 1 2 3 4 5
Enter index starting block and length: 3 20
3 starting block is already allocated

File Allocation Strategies:
1. Sequential Allocation
2. Indexed Allocation
3. Linked Allocation
4. Exit
Enter your choice: 3
Enter how many blocks already allocated: 4
Enter blocks already allocated: 1 2 3 4
Enter index starting block and length: 5 20
5 starting block is already allocated

File Allocation Strategies:
1. Sequential Allocation
2. Indexed Allocation
```

13. Write a C program to simulate the following file organisation techniques

a) Single level directory

b) Two level directory

c) Hierarchical

Source Code

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

#define MAX_FILES 50

struct File {
    char name[30];
    int size;
};

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

struct TwoLevelDirectory {
    char userNames[MAX_FILES][30];
    struct SingleLevelDirectory directories[MAX_FILES];
    int userCount;
};

struct HierarchicalDirectory {
    char mainDirectoryName[30];
    struct TwoLevelDirectory subDirectories[MAX_FILES];
    int subDirectoryCount;
};

void singleLevelDirectory();
void twoLevelDirectory();
void hierarchicalDirectory();

int main() {
    int choice;

    while (1) {
```

```
        printf("\nFile Organization Techniques:\n");
        printf("1. Single Level Directory\n");
        printf("2. Two Level Directory\n");
        printf("3. Hierarchical Directory\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                singleLevelDirectory();
                break;

            case 2:
                twoLevelDirectory();
                break;

            case 3:
                hierarchicalDirectory();
                break;

            case 4:
                printf("Exiting...\n");
                exit(0);

            default:
                printf("Invalid choice. Please select a valid option.\n");
        }
    }

    return 0;
}

void singleLevelDirectory() {
    struct SingleLevelDirectory directory;
    int choice;

    directory.fileCount = 0;

    while (1) {
        printf("\nSingle Level Directory:\n");
```

```
        printf("1. Create File\n");
        printf("2. List Files\n");
        printf("3. Back\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                if (directory.fileCount >= MAX_FILES) {
                    printf("Directory is full\n");
                } else {
                    printf("Enter file name: ");
                    scanf("%s", directory.files[directory.fileCount].name);
                    printf("Enter file size: ");
                    scanf("%d", &directory.files[directory.fileCount].size);
                    directory.fileCount++;
                    printf("File created successfully\n");
                }
                break;

            case 2:
                printf("\nFiles in the directory:\n");
                for (int i = 0; i < directory.fileCount; i++) {
                    printf("%s\t%d KB\n", directory.files[i].name, directory.files[i].size);
                }
                break;

            case 3:
                return;

            default:
                printf("Invalid choice. Please select a valid option.\n");
        }
    }
}

void twoLevelDirectory() {
    struct TwoLevelDirectory directory;
    int choice;

    directory.userCount = 0;
```

```
while (1) {
    printf("\nTwo Level Directory:\n");
    printf("1. Create User\n");
    printf("2. Select User\n");
    printf("3. Back\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            if (directory.userCount >= MAX_FILES) {
                printf("Directory is full\n");
            } else {
                printf("Enter user name: ");
                scanf("%s", directory.userNames[directory.userCount]);
                directory.directories[directory.userCount].fileCount = 0;
                directory.userCount++;
                printf("User created successfully\n");
            }
            break;

        case 2:
            printf("\nSelect User:\n");
            for (int i = 0; i < directory.userCount; i++) {
                printf("%d. %s\n", i + 1, directory.userNames[i]);
            }
            int userChoice;
            printf("Enter user choice: ");
            scanf("%d", &userChoice);

            if (userChoice >= 1 && userChoice <= directory.userCount) {
                singleLevelDirectory(&(directory.directories[userChoice - 1]));
            } else {
                printf("Invalid user choice\n");
            }
            break;

        case 3:
            return;
```

```
            default:
                printf("Invalid choice. Please select a valid option.\n");
        }
    }
}

void hierarchicalDirectory() {
    struct HierarchicalDirectory directory;
    int choice;

    printf("Enter main directory name: ");
    scanf("%s", directory.mainDirectoryName);
    directory.subDirectoryCount = 0;

    while (1) {
        printf("\nHierarchical Directory:\n");
        printf("1. Create Sub Directory\n");
        printf("2. Select Sub Directory\n");
        printf("3. Back\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                if (directory.subDirectoryCount >= MAX_FILES) {
                    printf("Directory is full\n");
                } else {
                    printf("Enter sub directory name: ");
                    scanf("%s", directory.subDirectories[directory.subDirectoryCount].userNames[0]);
                    directory.subDirectories[directory.subDirectoryCount].directories[0].fileCount = 0;
                    directory.subDirectories[directory.subDirectoryCount].userCount = 1;
                    directory.subDirectoryCount++;
                    printf("Sub directory created successfully\n");
                }
                break;

            case 2:
                printf("\nSelect Sub Directory:\n");
                for (int i = 0; i < directory.subDirectoryCount; i++) {
                    printf("%d. %s\n", i + 1, directory.subDirectories[i].userNames[0]);
                }
```

```
            int subDirChoice;
            printf("Enter sub directory choice: ");
            scanf("%d", &subDirChoice);

            if (subDirChoice >= 1 && subDirChoice <= directory.subDirectoryCount) {
                twoLevelDirectory(&(directory.subDirectories[subDirChoice - 1]));
            } else {
                printf("Invalid sub directory choice\n");
            }
            break;

        case 3:
            return;

        default:
            printf("Invalid choice. Please select a valid option.\n");
    }
  }
}
```

Output

```
File Organization Techniques:
1. Single Level Directory
2. Two Level Directory
3. Hierarchical Directory
4. Exit
Enter your choice: 1
Single Level Directory:
1. Create File
2. List Files
3. Back
Enter your choice: 1
Enter file name: AS
Enter file size: 20
File created successfully

Single Level Directory:
1. Create File
2. List Files
3. Back
Enter your choice: 2
Files in the directory:
AS  20 KB
```

```
Single Level Directory:
1. Create File
2. List Files
3. Back
Enter your choice: 3
File Organization Techniques:
1. Single Level Directory
2. Two Level Directory
3. Hierarchical Directory
4. Exit
Enter your choice: 2
Two Level Directory:
1. Create User
2. Select User
3. Back
Enter your choice: 1
Enter user name: AA
User created successfully
```

```
Enter your choice: 1
Enter user name: AA
User created successfully

Two Level Directory:
1. Create User
2. Select User
3. Back
Enter your choice: 1
Enter user name: AQ
User created successfully

Two Level Directory:
1. Create User
2. Select User
3. Back
Enter your choice: 2
Select User:
1. AA
2. AQ
```

```
File Organization Techniques:
1. Single Level Directory
2. Two Level Directory
3. Hierarchical Directory
4. Exit
Enter your choice: 3
Enter main directory name: Dir1
Hierarchical Directory:
1. Create Sub Directory
2. Select Sub Directory
3. Back
Enter your choice: 1
Enter sub directory name: Sub1
Sub directory created successfully

Hierarchical Directory:
1. Create Sub Directory
2. Select Sub Directory
3. Back
Enter your choice: 2
Select Sub Directory:
1. Sub1
```

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

Source Code

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

#define MAX_REQUESTS 1000

void fcfs(int head, int requests[], int n);
void scan(int head, int requests[], int n, int maxCylinder);
void cscan(int head, int requests[], int n, int maxCylinder);

int main() {
   int head, n, maxCylinder, choice;
   int requests[MAX_REQUESTS];

   printf("Enter the initial head position: ");
   scanf("%d", &head);
   printf("Enter the number of requests: ");
   scanf("%d", &n);
   printf("Enter the maximum cylinder: ");
   scanf("%d", &maxCylinder);

   printf("Enter the requests:\n");
   for (int i = 0; i < n; i++) {
      scanf("%d", &requests[i]);
   }

   while (1) {
      printf("\nDisk Scheduling Algorithms:\n");
      printf("1. FCFS (First-Come, First-Served)\n");
      printf("2. SCAN\n");
      printf("3. C-SCAN\n");
      printf("4. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
```

```
            case 1:
                fcfs(head, requests, n);
                break;

            case 2:
                scan(head, requests, n, maxCylinder);
                break;

            case 3:
                cscan(head, requests, n, maxCylinder);
                break;

            case 4:
                printf("Exiting...\n");
                exit(0);

            default:
                printf("Invalid choice. Please select a valid option.\n");
        }
    }

    return 0;
}

void fcfs(int head, int requests[], int n) {
    int totalSeekTime = 0;

    printf("\nFCFS (First-Come, First-Served):\n");
    printf("Head Movement Sequence:\n");

    for (int i = 0; i < n; i++) {
        int seek = abs(requests[i] - head);
        printf("%d ", requests[i]);
        totalSeekTime += seek;
        head = requests[i];
    }

    printf("\nTotal Seek Time: %d\n", totalSeekTime);
}

void scan(int head, int requests[], int n, int maxCylinder) {
```

```
int totalSeekTime = 0;
int direction = 1;

printf("\nSCAN:\n");
printf("Head Movement Sequence:\n");

while (1) {
    if (direction == 1) {
        for (int i = 0; i < n; i++) {
            if (requests[i] >= head) {
                int seek = abs(requests[i] - head);
                printf("%d ", requests[i]);
                totalSeekTime += seek;
                head = requests[i];
            }
        }
        direction = -1;
    } else {
        for (int i = n - 1; i >= 0; i--) {
            if (requests[i] <= head) {
                int seek = abs(requests[i] - head);
                printf("%d ", requests[i]);
                totalSeekTime += seek;
                head = requests[i];
            }
        }
        direction = 1;
    }

    if (direction == 1) {
        if (head <= maxCylinder) {
            printf("%d ", maxCylinder);
            totalSeekTime += abs(maxCylinder - head);
            head = maxCylinder;
        }
    } else {
        if (head >= 0) {
            printf("0 ");
            totalSeekTime += head;
            head = 0;
        }
```

```
        }

        break;
    }

    printf("\nTotal Seek Time: %d\n", totalSeekTime);
}

void cscan(int head, int requests[], int n, int maxCylinder) {
    int totalSeekTime = 0;

    printf("\nC-SCAN:\n");
    printf("Head Movement Sequence:\n");

    for (int i = head; i <= maxCylinder; i++) {
        int seek = abs(i - head);
        printf("%d ", i);
        totalSeekTime += seek;
        head = i;
    }

    printf("0 ");

    totalSeekTime += maxCylinder;

    for (int i = 0; i < n; i++) {
        int seek = abs(requests[i] - 0);
        printf("%d ", requests[i]);
        totalSeekTime += seek;
    }

    printf("\nTotal Seek Time: %d\n", totalSeekTime);
}
```

Output

```
Enter the initial head position: 20
Enter the number of requests: 6
Enter the maximum cylinder: 200
Enter the requests:
84 25 62 41 32 77
Disk Scheduling Algorithms:1. FCFS (First-Come, First-Served)
2. SCAN
3. C-SCAN
4. Exit
Enter your choice: 1
FCFS (First-Come, First-Served):
Head Movement Sequence:
84 25 62 41 32 77
Total Seek Time: 235
Disk Scheduling Algorithms:
1. FCFS (First-Come, First-Served)
2. SCAN
3. C-SCAN
4. Exit
Enter your choice: 2
SCAN:
Total Seek Time: 148

Disk Scheduling Algorithms:
1. FCFS (First-Come, First-Served)
2. SCAN
3. C-SCAN
4. Exit
Enter your choice: 3
C-SCAN:
Total Seek Time: 701

Disk Scheduling Algorithms:
1. FCFS (First-Come, First-Served)
2. SCAN
3. C-SCAN
```

15. Write a C program to simulate disk scheduling algorithms

a) SSTF

b) LOOK

c) c-LOOK

Source Code

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

#define MAX_REQUESTS 1000

void sstf(int head, int requests[], int n);
void look(int head, int requests[], int n, int maxCylinder);
void clook(int head, int requests[], int n, int maxCylinder);

int main() {
    int head, n, maxCylinder, choice;
    int requests[MAX_REQUESTS];

    printf("Enter the initial head position: ");
    scanf("%d", &head);
    printf("Enter the number of requests: ");
    scanf("%d", &n);
    printf("Enter the maximum cylinder: ");
    scanf("%d", &maxCylinder);

    printf("Enter the requests:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &requests[i]);
    }

    while (1) {
        printf("\nDisk Scheduling Algorithms:\n");
        printf("1. SSTF (Shortest Seek Time First)\n");
        printf("2. LOOK\n");
        printf("3. C-LOOK\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
```

```
            sstf(head, requests, n);
            break;

        case 2:
            look(head, requests, n, maxCylinder);
            break;

        case 3:
            clook(head, requests, n, maxCylinder);
            break;

        case 4:
            printf("Exiting...\n");
            exit(0);

        default:
            printf("Invalid choice. Please select a valid option.\n");
      }
   }

   return 0;
}

void sstf(int head, int requests[], int n) {
   int totalSeekTime = 0;
   int visited[MAX_REQUESTS] = {0};

   printf("\nSSTF (Shortest Seek Time First):\n");
   printf("Head Movement Sequence:\n");

   for (int i = 0; i < n; i++) {
      int minDistance = INT_MAX;
      int minIndex = -1;

      for (int j = 0; j < n; j++) {
         if (!visited[j] && abs(requests[j] - head) < minDistance) {
            minDistance = abs(requests[j] - head);
            minIndex = j;
         }
      }
```

```
        visited[minIndex] = 1;
        printf("%d ", requests[minIndex]);
        totalSeekTime += minDistance;
        head = requests[minIndex];
    }

    printf("\nTotal Seek Time: %d\n", totalSeekTime);
}

void look(int head, int requests[], int n, int maxCylinder) {
    int totalSeekTime = 0;
    int direction = 1;

    printf("\nLOOK:\n");
    printf("Head Movement Sequence:\n");

    while (1) {
        for (int i = 0; i < n; i++) {
            if (requests[i] == head) {
                printf("%d ", requests[i]);
                requests[i] = -1; // Mark as visited
            }
        }

        if (direction == 1) {
            for (int i = head + 1; i <= maxCylinder; i++) {
                for (int j = 0; j < n; j++) {
                    if (requests[j] == i) {
                        printf("%d ", requests[j]);
                        requests[j] = -1; // Mark as visited
                    }
                }
            }

            direction = -1;
        } else {
            for (int i = head - 1; i >= 0; i--) {
                for (int j = 0; j < n; j++) {
                    if (requests[j] == i) {
                        printf("%d ", requests[j]);
                        requests[j] = -1; // Mark as visited
```

```
            }
        }
    }

    direction = 1;
  }

  int found = 0;
  for (int i = 0; i < n; i++) {
    if (requests[i] != -1) {
      found = 1;
      break;
    }
  }

  if (!found) {
    break;
  }
 }

 printf("\nTotal Seek Time: %d\n", totalSeekTime);
}

void clook(int head, int requests[], int n, int maxCylinder) {
 int totalSeekTime = 0;

 printf("\nC-LOOK:\n");
 printf("Head Movement Sequence:\n");

 for (int i = 0; i < n; i++) {
  if (requests[i] >= head) {
    printf("%d ", requests[i]);
    totalSeekTime += abs(requests[i] - head);
    head = requests[i];
  }
 }

 printf("%d ", maxCylinder);
 totalSeekTime += abs(maxCylinder - head);
 head = 0;
```

```
    for (int i = 0; i < n; i++) {
        if (requests[i] < head) {
            printf("%d ", requests[i]);
            totalSeekTime += abs(requests[i] - head);
            head = requests[i];
        }
    }

    printf("\nTotal Seek Time: %d\n", totalSeekTime);
}
```

Output

```
Enter the initial head position: 20
Enter the number of requests: 6
Enter the maximum cylinder: 199
Enter the requests:
20 14 58 96 21 125
Disk Scheduling Algorithms:
1. SSTF (Shortest Seek Time First)
2. LOOK
3. C-LOOK
4. Exit
Enter your choice: 1
SSTF (Shortest Seek Time First):
Head Movement Sequence:
20 21 14 58 96 125
Total Seek Time: 119

Disk Scheduling Algorithms:
1. SSTF (Shortest Seek Time First)
2. LOOK
3. C-LOOK
4. Exit
```

```
Enter your choice: 2
LOOK:
Head Movement Sequence:
20 21 58 96 125 14
Total Seek Time: 0

Disk Scheduling Algorithms:
1. SSTF (Shortest Seek Time First)
2. LOOK
3. C-LOOK
4. Exit
Enter your choice: 3
C-LOOK:
Total Seek Time: 180

Disk Scheduling Algorithms:
1. SSTF (Shortest Seek Time First)
2. LOOK
3. C-LOOK
4. Exit
Enter your choice: 4
Exiting...
```