VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT on

OPERATING SYSTEMS

(23CS4PCOPS)

Submitted by

SHREYAS GOWDA C (1BM23CS319)

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 Feb-2025 to June-2025

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 – 23CS4PCOPS" carried out by **SHREYAS GOWDA C (1BM23CS319),** 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 2024. The Lab report has been approved as it satisfies the academic requirements in respect of a **OPERATING SYSTEMS - (23CS4PCOPS)** work prescribed for the said degree.

Amruta.BP

Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda

Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Sl. No.	Experiment Title	Page No.
1.	Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time. (Any one) a) FCFS b) SJF	5 -14
2.	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.	15-18
3.	Write a C program to simulate Real-Time CPU Scheduling algorithms a) Rate- Monotonic	19 -21
4.	Write a C program to simulate: a) Producer-Consumer problem using semaphores. b) Dining-Philosopher's problem	22 -27
5.	Write a C program to simulate: a) Bankers' algorithm for the purpose of deadlock avoidance.	28-30
6.	Write a C program to simulate the following contiguous memory allocation techniques. a) Worst-fit b) Best-fit c) First-fit	31-38
7.	Write a C program to simulate page replacement algorithms. a) FIFO b) LRU c) Optimal	39 -47
8.	Write a C program to simulate the following file allocation strategies. a) Sequential b) Indexed c) Linked	48-54

Course Outcome

CO1	Apply the different concepts and functionalities of Operating System
CO2	Analyze 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

Program -1

Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time.

a. FCFS

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

int main()
{
   int ps[100],at[100],bt[100],ct[100],tat[100],wt[100],n;
   float totalTAT=0,totalWT=0;
   printf("Enter the number of process to enter : ");
   scanf("%d",&n);

   for(int i=0;i<n;i++)
   {
      printf("\n Enter the arrival time of process [%d] : ",i+1);
      scanf("%d",&at[i]);
   }

   for(int i=0;i<n;i++)
   {
      printf("\n Enter the burst time of process [%d] : ",i+1);


      scanf("%d",&bt[i]);
   }
```

```
    int sum=at[0];
    for(int i=0;i<n;i++)
    {
        sum+=bt[i];
        ct[i]=sum;
    }

    for(int i=0;i<n;i++)
    {
        tat[i]=ct[i]-at[i];
        totalTAT+=tat[i];
    }

    for(int i=0;i<n;i++)
    {
        wt[i]=tat[i]-bt[i];
        totalWT+=wt[i];
    }

    printf("\n The average TAT is : %.2f ms",(float)totalTAT/n);
    printf("\n The average WT is : %.2f ms\n\n\n",(float)totalWT/n);

    return 0;
}
```

output

```
PS C:\Users\shett\Downloads\os> gcc fcfs.c -ofcfs
PS C:\Users\shett\Downloads\os> ./fcfs
Enter the number of process to enter : 3

 Enter the arrival time of process [1] : 0

 Enter the arrival time of process [2] : 0

 Enter the arrival time of process [3] : 1

 Enter the burst time of process [1] : 7

 Enter the burst time of process [2] : 8

 Enter the burst time of process [3] : 10

 The average TAT is : 15.33 ms
 The average WT is : 7.00 ms


PS C:\Users\shett\Downloads\os>
```

b. SJF(Non-preemptive):

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

struct process
{
    int  at;
    int  bt;
    int  ct;
    int tat;
    int wt;
    int start_time;
} ps[100];

int main()
{
    float totalTAT = 0, totalWT = 0;
    int n;
    int completed = 0;
    bool is_visited[100] = {false};
    int current_time = 0;
    int minimum = 9999999999;
    int min_index = -1;

    printf("Enter the number of process to enter : ");
    scanf("%d", &n);

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

8

```
{
    printf("\n Enter the arrival time of process [%d] : ", i + 1);
    scanf("%d", &ps[i].at);
}

for(int i = 0; i < n; i++)
{
    printf("\n Enter the burst time of process [%d] : ", i + 1);
    scanf("%d", &ps[i].bt);
}

while(completed != n)
{
    minimum = 2147483647;
    min_index = -1;

    for(int i = 0; i < n; i++)
    {
        if(ps[i].at <= current_time && !is_visited[i])
        {
            if(ps[i].bt < minimum)
            {
                minimum = ps[i].bt;
                min_index = i;
            }

            if(ps[i].bt == minimum)
            {
```

```
            if(ps[i].at < ps[min_index].at)
            {
               minimum = ps[i].bt;
               min_index = i;
            }
         }
      }
   }

   if(min_index == -1)
   {
      current_time++;
   }
   else
   {
      ps[min_index].start_time = current_time;
      ps[min_index].ct = ps[min_index].start_time + ps[min_index].bt;
      ps[min_index].tat = ps[min_index].ct - ps[min_index].at;
      totalTAT += ps[min_index].tat;
      ps[min_index].wt = ps[min_index].tat - ps[min_index].bt;
      totalWT += ps[min_index].wt;
      is_visited[min_index] = true;
      completed++;
      current_time = ps[min_index].ct;
   }
  }

  printf("\n The average TAT is : %.2f ms", totalTAT / n);
printf("\n The average WT is : %.2f ms\n\n\n", totalWT / n);
}
```

output

```
PS C:\Users\shett\Downloads\os> gcc sjf_np.c -o sjfnp
PS C:\Users\shett\Downloads\os> ./sjfnp
Enter the number of process to enter : 3

 Enter the arrival time of process [1] : 0

 Enter the arrival time of process [2] : 0

 Enter the arrival time of process [3] : 1

 Enter the burst time of process [1] : 7

 Enter the burst time of process [2] : 8

 Enter the burst time of process [3] : 10

 The average TAT is : 18.33 ms
 The average WT is : 8.00 ms


PS C:\Users\shett\Downloads\os>
```

c. SJF(Pre-Emptive[SRTF]):

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

struct process_struct {
  int pid, at, bt, ct, wt, tat, start_time;
} ps[100];

int main() {
    int n;
    float bt_remaining[100];
    bool is_completed[100] = {false};
    int current_time = 0, completed = 0;
    float sum_tat = 0, sum_wt = 0;
    int prev = 0;

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

    for (int i = 0; i < n; i++) {
        printf("\nEnter Process %d Arrival Time: ", i);
        scanf("%d", &ps[i].at);
        ps[i].pid = i;
    }

    for (int i = 0; i < n; i++) {
        printf("\nEnter Process %d Burst Time: ", i);
```

```
        scanf("%d", &ps[i].bt);
        bt_remaining[i] = ps[i].bt;
    }

    while (completed != n) {
        int min_index = -1, minimum = INT_MAX;
        for (int i = 0; i < n; i++) {
            if (ps[i].at <= current_time && !is_completed[i] && bt_remaining[i] <
minimum) {
                minimum = bt_remaining[i];
                min_index = i;
            }
        }

        if (min_index == -1) {
            current_time++;
        } else {
            if (bt_remaining[min_index] == ps[min_index].bt) {
                ps[min_index].start_time = current_time;
            }
            bt_remaining[min_index]--;
            current_time++;
            prev = current_time;
            if (bt_remaining[min_index] == 0) {
                ps[min_index].ct = current_time;
                ps[min_index].tat = ps[min_index].ct - ps[min_index].at;
                ps[min_index].wt = ps[min_index].tat - ps[min_index].bt;
                sum_tat += ps[min_index].tat;
```

```
            sum_wt += ps[min_index].wt;
            completed++;
            is_completed[min_index] = true;
          }
        }
    }

    printf("\nAverage Turnaround Time = %f", sum_tat / n);
    printf("\nAverage Waiting Time = %f\n", sum_wt / n);
return 0}
```

output

```
PS C:\Users\shett\Downloads\os> gcc sjf_p.c -o sjfp
PS C:\Users\shett\Downloads\os> ./sjfp
Enter total number of processes: 3

Enter Process 0 Arrival Time: 0

Enter Process 1 Arrival Time: 0

Enter Process 2 Arrival Time: 1

Enter Process 0 Burst Time: 7

Enter Process 1 Burst Time: 8

Enter Process 2 Burst Time: 10

Average Turnaround Time = 19.33
Average Waiting Time = 9.00

PS C:\Users\shett\Downloads\os>
```

Program -2

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.

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

struct Process {
    int pid;
    int arrival_time;
    int burst_time;
    int completion_time;
    int waiting_time;
    int turnaround_time;
};

void sort_by_arrival(struct Process p[], int n) {
    struct Process temp;
    for (int i = 0; i < n - 1; i++)
        for (int j = i + 1; j < n; j++)
            if (p[i].arrival_time > p[j].arrival_time) {
                temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
```

```
}

void calculate_fcfs(struct Process p[], int n, int start_time) {
    int current_time = start_time;
    for (int i = 0; i < n; i++) {
        if (current_time < p[i].arrival_time)
            current_time = p[i].arrival_time;
        p[i].completion_time = current_time + p[i].burst_time;
        p[i].turnaround_time = p[i].completion_time - p[i].arrival_time;
        p[i].waiting_time = p[i].turnaround_time - p[i].burst_time;
        current_time = p[i].completion_time;
    }
}

void print_processes(struct Process p[], int n) {
    printf("PID\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++)
        printf("%d\t%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].arrival_time, p[i].burst_time,
p[i].completion_time, p[i].turnaround_time, p[i].waiting_time);
}

int main() {
    int n_sys, n_user;
    printf("Enter number of system processes: ");
    scanf("%d", &n_sys);
    struct Process sys[n_sys];
    for (int i = 0; i < n_sys; i++) {
        printf("Enter PID, Arrival Time and Burst Time for System Process %d: ", i + 1);
        scanf("%d%d%d", &sys[i].pid, &sys[i].arrival_time, &sys[i].burst_time);
    }
```

```
    printf("Enter number of user processes: ");
    scanf("%d", &n_user);
    struct Process user[n_user];
    for (int i = 0; i < n_user; i++) {
        printf("Enter PID, Arrival Time and Burst Time for User Process %d: ", i + 1);
        scanf("%d%d%d", &user[i].pid, &user[i].arrival_time, &user[i].burst_time);
    }

    sort_by_arrival(sys, n_sys);
    sort_by_arrival(user, n_user);

    calculate_fcfs(sys, n_sys, 0);
    int last_sys_completion = n_sys > 0 ? sys[n_sys - 1].completion_time : 0;
    calculate_fcfs(user, n_user, last_sys_completion);

    printf("\nSystem Processes:\n");
    print_processes(sys, n_sys);
    printf("\nUser Processes:\n");
    print_processes(user, n_user);

    return 0;
}
```

output

```
PS C:\Users\shett\Downloads\os> gcc priority_fcfs.c -o priorityfcfs
PS C:\Users\shett\Downloads\os> ./priorityfcfs
Enter number of system processes: 2
Enter PID, Arrival Time and Burst Time for System Process 1: 1 0 3
Enter PID, Arrival Time and Burst Time for System Process 2: 2 1 4
Enter number of user processes: 2
Enter PID, Arrival Time and Burst Time for User Process 1: 3 2 5
Enter PID, Arrival Time and Burst Time for User Process 2: 4 3 6

System Processes:
PID AT  BT  CT  TAT WT
1   0   3   3   3   0
2   1   4   7   6   2

User Processes:
PID AT  BT  CT  TAT WT
3   2   5   12  10  5
4   3   6   18  15  9

PS C:\Users\shett\Downloads\os>
```

Program -3

Write a C program to simulate Real-Time CPU Scheduling algorithms

a. Rate- Monotonic

```
#include <stdio.h>
struct Task {
    int id;
    int execution_time;
    int period;
    int deadline;
    int remaining_time;
    int next_release;
};

int main() {
    int n, hyper_period = 1;
    printf("Enter number of tasks: ");
    scanf("%d", &n);
    struct Task t[n];
    for (int i = 0; i < n; i++) {
        printf("Enter Execution Time and Period for Task %d: ", i + 1);
        scanf("%d%d", &t[i].execution_time, &t[i].period);
        t[i].id = i + 1;
        t[i].deadline = t[i].period;
        t[i].remaining_time = 0;
        t[i].next_release = 0;
        hyper_period *= t[i].period;
    }
```

```
    printf("\nTime\tTask\n");
    for (int time = 0; time < hyper_period; time++) {
        for (int i = 0; i < n; i++)
            if (time == t[i].next_release) {
                t[i].remaining_time = t[i].execution_time;
                t[i].next_release += t[i].period;
            }

        int current = -1, min_period = 1e9;
        for (int i = 0; i < n; i++)
            if (t[i].remaining_time > 0 && t[i].period < min_period) {
                min_period = t[i].period;
                current = i;
            }

        if (current != -1) {
            printf("%d\tT%d\n", time, t[current].id);
            t[current].remaining_time--;
        } else {
            printf("%d\tIdle\n",  time);
        }
    }

    return 0;
}
```

output

```
PS C:\Users\shett\Downloads\os> gcc rm_sched.c -o rmsched
PS C:\Users\shett\Downloads\os> ./rmsched
Enter number of tasks: 2
Enter Execution Time and Period for Task 1: 1 4
Enter Execution Time and Period for Task 2: 2 5

Time    Task
0   T1
1   T2
2   T2
3   Idle
4   T1
5   T2
6   T2
7   Idle
8   T1
9   T2
10  T2
11  Idle
12  T1
13  T2
14  T2
15  Idle
16  T1
17  T2
18  T2
19  Idle

PS C:\Users\shett\Downloads\os>
```

Program -4

Write a C program to simulate:

a. Producer-Consumer problem using semaphores.

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

#define SIZE 5

int buffer[SIZE];
int in = 0, out = 0;

sem_t empty, full, mutex;

void* producer(void* arg) {
    int item;
    while (1) {
        item = rand() % 100;
        sem_wait(&empty);
        sem_wait(&mutex);
        buffer[in] = item;
        printf("Produced: %d\n", item);
        in = (in + 1) % SIZE;
        sem_post(&mutex);
        sem_post(&full);
        sleep(1);
    }
```

```
}

void* consumer(void* arg) {
    int item;
    while (1) {
        sem_wait(&full);
        sem_wait(&mutex);
        item = buffer[out];
        printf("Consumed: %d\n", item);
        out = (out + 1) % SIZE;
        sem_post(&mutex);
        sem_post(&empty);
        sleep(2);
    }
}

int main() {
    pthread_t prod, cons;

    sem_init(&empty, 0, SIZE);
    sem_init(&full, 0, 0);
    sem_init(&mutex, 0, 1);

    pthread_create(&prod, NULL, producer, NULL);
    pthread_create(&cons, NULL, consumer, NULL);

    pthread_join(prod, NULL);
    pthread_join(cons, NULL);

    sem_destroy(&empty);
```

```
    sem_destroy(&full);
    sem_destroy(&mutex);

    return 0;
}
```

output

```
PS C:\Users\shett\Downloads\os> gcc prod_cons.c -lpthread -o prodcons
PS C:\Users\shett\Downloads\os> ./prodcons
Produced: 42
Consumed: 42
Produced: 87
Produced: 29
Consumed: 87
Produced: 14
Consumed: 29
Produced: 73
Consumed: 14
Produced: 63
Consumed: 73
Produced: 38
Consumed: 63
Produced: 15
Consumed: 38
Produced: 90
Consumed: 15
Produced: 11
Consumed: 90
```

b. Dining-Philosopher's problem

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

#define N 5

sem_t forks[N];
sem_t mutex;

void* philosopher(void* num) {
    int id = *(int*)num;
    while (1) {
        printf("Philosopher %d is thinking\n", id);
        sleep(1);
        sem_wait(&mutex);
        sem_wait(&forks[id]);
        sem_wait(&forks[(id + 1) % N]);
        sem_post(&mutex);
        printf("Philosopher %d is eating\n", id);
        sleep(2);
        sem_post(&forks[id]);
        sem_post(&forks[(id + 1) % N]);
    }
}

int main() {
    pthread_t threads[N];
```

```
    int ids[N];

    sem_init(&mutex, 0, 1);
    for (int i = 0; i < N; i++)
        sem_init(&forks[i], 0, 1);

    for (int i = 0; i < N; i++) {
        ids[i] = i;
        pthread_create(&threads[i], NULL, philosopher, &ids[i]);
    }

    for (int i = 0; i < N; i++)
        pthread_join(threads[i], NULL);

    for (int i = 0; i < N; i++)
        sem_destroy(&forks[i]);
    sem_destroy(&mutex);

    return 0;
}
```

output

```
PS C:\Users\shett\Downloads\os> gcc dp.c -lpthread -o dp
PS C:\Users\shett\Downloads\os> ./dp
Philosopher 0 is thinking
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 0 is eating
Philosopher 1 is eating
Philosopher 2 is eating
Philosopher 3 is eating
Philosopher 4 is eating
Philosopher 0 is thinking
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 0 is eating
Philosopher 1 is eating
```

Program -5

Write a C program to simulate:

a. Bankers' algorithm for the purpose of deadlock avoidance.

```
#include <stdio.h>
int main() {
    int n, m;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    printf("Enter number of resources: ");
    scanf("%d", &m);

    int alloc[n][m], max[n][m], need[n][m], avail[m];
    printf("Enter Allocation Matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &alloc[i][j]);

    printf("Enter Maximum Matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &max[i][j]);

    printf("Enter Available Resources:\n");
    for (int i = 0; i < m; i++)
        scanf("%d", &avail[i]);

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

```
    for (int j = 0; j < m; j++)
        need[i][j] = max[i][j] - alloc[i][j];

int finish[n], safe[n], index = 0;
for (int i = 0; i < n; i++)
    finish[i] = 0;

for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
        if (!finish[i]) {
            int flag = 1;
            for (int j = 0; j < m; j++) {
                if (need[i][j] > avail[j]) {
                    flag = 0;
                    break;
                }
            }
            if (flag) {
                for (int j = 0; j < m; j++)
                    avail[j] += alloc[i][j];
                safe[index++] = i;
                finish[i] = 1;
            }
        }
    }
}

int is_safe = 1;
for (int i = 0; i < n; i++) {
    if (!finish[i]) {
```

```
            is_safe = 0;
            break;
        }
    }

    if (is_safe) {
        printf("System is in a safe state.\nSafe sequence: ");
        for (int i = 0; i < n; i++)
            printf("P%d ", safe[i]);
        printf("\n");
    } else {
        printf("System is not in a safe state.\n");
    }

    return 0;
}
```

output

```
PS C:\Users\shett\Downloads\os> gcc banker.c -o banker
PS C:\Users\shett\Downloads\os> .\banker
Enter number of processes: 5
Enter number of resources: 3
Enter Allocation Matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter Maximum Matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter Available Resources:
3 3 2
System is in a safe state.
Safe sequence: P1 P3 P4 P0 P2
```

Program -6

Write a C program to simulate the following contiguous memory allocation techniques.

a. Worst-fit

```
#include <stdio.h>
int main() {
    int b, p;
    printf("Enter number of memory blocks: ");
    scanf("%d", &b);
    int blockSize[b], blockAllocated[b];
    printf("Enter sizes of memory blocks:\n");
    for (int i = 0; i < b; i++) {
        scanf("%d", &blockSize[i]);
        blockAllocated[i] = 0;
    }

    printf("Enter number of processes: ");
    scanf("%d", &p);
    int processSize[p], allocation[p];
    printf("Enter sizes of processes:\n");
    for (int i = 0; i < p; i++) {
        scanf("%d", &processSize[i]);
        allocation[i] = -1;
    }

    for (int i = 0; i < p; i++) {
        int worstIdx = -1;
        for (int j = 0; j < b; j++) {
            if (!blockAllocated[j] && blockSize[j] >= processSize[i]) {
```

```
            if (worstIdx == -1 || blockSize[j] > blockSize[worstIdx])
                worstIdx = j;
        }
    }
    if (worstIdx != -1) {
        allocation[i] = worstIdx;
        blockAllocated[worstIdx] = 1;
    }
}

printf("\nProcess No.\tProcess Size\tBlock No.\n");
for (int i = 0; i < p; i++) {
    printf("%d\t\t%d\t\t", i + 1, processSize[i]);
    if (allocation[i] != -1)
        printf("%d\n", allocation[i] + 1);
    else
        printf("Not Allocated\n");
}

return 0;
}
```

output

```
PS C:\Users\shett\Downloads\os> gcc worstfit.c -o worstfit
PS C:\Users\shett\Downloads\os> .\worstfit
Enter number of memory blocks: 5
Enter sizes of memory blocks:
100 500 200 300 600
Enter number of processes: 4
Enter sizes of processes:
212 417 112 426

Process No. Process Size    Block No.
1       212     5
2       417     2
3       112     3
4       426     Not Allocated
```

b.Best-fit

```
#include <stdio.h>

int main() {
    int b, p;
    printf("Enter number of memory blocks: ");
    scanf("%d", &b);
    int blockSize[b], blockAllocated[b];
    printf("Enter sizes of memory blocks:\n");
    for (int i = 0; i < b; i++) {
        scanf("%d", &blockSize[i]);
        blockAllocated[i] = 0;
    }

    printf("Enter number of processes: ");
    scanf("%d", &p);
    int processSize[p], allocation[p];
    printf("Enter sizes of processes:\n");
    for (int i = 0; i < p; i++) {
        scanf("%d", &processSize[i]);
        allocation[i] = -1;
    }

    for (int i = 0; i < p; i++) {
        int bestIdx = -1;
        for (int j = 0; j < b; j++) {
            if (!blockAllocated[j] && blockSize[j] >= processSize[i]) {
                if (bestIdx == -1 || blockSize[j] < blockSize[bestIdx])
                    bestIdx = j;
```

```
            }
        }
        if (bestIdx != -1) {
            allocation[i] = bestIdx;
            blockAllocated[bestIdx] = 1;
        }
    }

    printf("\nProcess No.\tProcess Size\tBlock No.\n");
    for (int i = 0; i < p; i++) {
        printf("%d\t\t%d\t\t", i + 1, processSize[i]);
        if (allocation[i] != -1)
            printf("%d\n", allocation[i] + 1);
        else
            printf("Not Allocated\n");
    }

    return 0;
}
```

output

```
PS C:\Users\shett\Downloads\os> gcc bestfit.c -o bestfit
PS C:\Users\shett\Downloads\os> .\bestfit
Enter number of memory blocks: 5
Enter sizes of memory blocks:
100 500 200 300 600
Enter number of processes: 4
Enter sizes of processes:
212 417 112 426

Process No. Process Size    Block No.
1       212     4
2       417     2
3       112     3
4       426     5
```

c. First-fit

```
#include <stdio.h>
int main() {
    int b, p;
    printf("Enter number of memory blocks: ");
    scanf("%d", &b);
    int blockSize[b], blockAllocated[b];
    printf("Enter sizes of memory blocks:\n");
    for (int i = 0; i < b; i++) {
        scanf("%d", &blockSize[i]);
        blockAllocated[i] = 0;
    }

    printf("Enter number of processes: ");
    scanf("%d", &p);
    int processSize[p], allocation[p];
    printf("Enter sizes of processes:\n");
    for (int i = 0; i < p; i++) {
        scanf("%d", &processSize[i]);
        allocation[i] = -1;
    }

    for (int i = 0; i < p; i++) {
        for (int j = 0; j < b; j++) {
            if (!blockAllocated[j] && blockSize[j] >= processSize[i]) {
                allocation[i] = j;
                blockAllocated[j] = 1;
                break;
            }
        }
```

```
    }

    printf("\nProcess No.\tProcess Size\tBlock No.\n");
    for (int i = 0; i < p; i++) {
        printf("%d\t\t%d\t\t", i + 1, processSize[i]);
        if (allocation[i] != -1)
            printf("%d\n", allocation[i] + 1);
        else
            printf("Not Allocated\n");
    }

    return 0;
}
```

output

```
PS C:\Users\shett\Downloads\os> gcc firstfit.c -o firstfit
PS C:\Users\shett\Downloads\os> .\firstfit
Enter number of memory blocks: 5
Enter sizes of memory blocks:
100 500 200 300 600
Enter number of processes: 4
Enter sizes of processes:
212 417 112 426

Process No. Process Size    Block No.
1       212     2
2       417     3
3       112     4
4       426     5
```

Program -7

Write a C program to simulate page replacement algorithms.

a.FIFO

```
#include <stdio.h>

int main() {
    int frames, pages;
    printf("Enter number of frames: ");
    scanf("%d", &frames);
    printf("Enter number of pages: ");
    scanf("%d", &pages);

    int pageSeq[pages];
    printf("Enter page reference string:\n");
    for (int i = 0; i < pages; i++)
        scanf("%d", &pageSeq[i]);

    int frame[frames], index = 0, pageFaults = 0;
    for (int i = 0; i < frames; i++)
        frame[i] = -1;

    for (int i = 0; i < pages; i++) {
        int found = 0;
        for (int j = 0; j < frames; j++) {
            if (frame[j] == pageSeq[i]) {
                found = 1;
                break;
```

```
            }
        }
        if (!found) {
            frame[index] = pageSeq[i];
            index = (index + 1) % frames;
            pageFaults++;
        }
        printf("Page %d: ", pageSeq[i]);
        for (int j = 0; j < frames; j++) {
            if (frame[j] != -1)
                printf("%d ", frame[j]);
            else
                printf("- ");
        }
        printf("\n");
    }

    printf("Total Page Faults = %d\n", pageFaults);

    return 0;
}
```

output

```
PS C:\Users\shett\Downloads\os> gcc fifo.c -o fifo
PS C:\Users\shett\Downloads\os> .\fifo
Enter number of frames: 3
Enter number of pages: 12
Enter page reference string:
7 0 1 2 0 3 0 4 2 3 0 3
Page 7: 7 - -
Page 0: 7 0 -
Page 1: 7 0 1
Page 2: 2 0 1
Page 0: 2 0 1
Page 3: 2 3 1
Page 0: 0 3 1
Page 4: 0 4 1
Page 2: 0 4 2
Page 3: 3 4 2
Page 0: 3 0 2
Page 3: 3 0 2
Total Page Faults = 9
```

b.LRU

```
#include <stdio.h>

int main() {
    int frames, pages;
    printf("Enter number of frames: ");
    scanf("%d", &frames);
    printf("Enter number of pages: ");
    scanf("%d", &pages);

    int pageSeq[pages];
    printf("Enter page reference string:\n");
    for (int i = 0; i < pages; i++)
        scanf("%d", &pageSeq[i]);

    int frame[frames];
    int time[frames];
    int pageFaults = 0, counter = 0;

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

    for (int i = 0; i < pages; i++) {
        int found = 0;
        for (int j = 0; j < frames; j++) {
            if (frame[j] == pageSeq[i]) {
                found = 1;
```

```
            counter++;
            time[j] = counter;
            break;
        }
    }
    if (!found) {
        int lru_index = 0, min_time = time[0];
        for (int j = 1; j < frames; j++) {
            if (time[j] < min_time) {
                min_time = time[j];
                lru_index = j;
            }
        }
        frame[lru_index] = pageSeq[i];
        counter++;
        time[lru_index] = counter;
        pageFaults++;
    }

    printf("Page %d: ", pageSeq[i]);
    for (int j = 0; j < frames; j++) {
        if (frame[j] != -1)
            printf("%d ", frame[j]);
        else
            printf("- ");
    }
    printf("\n");
}

printf("Total Page Faults = %d\n", pageFaults);
```

```
    return 0;
}
```

output

```
PS C:\Users\shett\Downloads\os> gcc lru.c -o lru
PS C:\Users\shett\Downloads\os> .\lru
Enter number of frames: 3
Enter number of pages: 12
Enter page reference string:
7 0 1 2 0 3 0 4 2 3 0 3
Page 7: 7 - -
Page 0: 7 0 -
Page 1: 7 0 1
Page 2: 0 1 2
Page 0: 0 1 2
Page 3: 0 2 3
Page 0: 0 2 3
Page 4: 0 3 4
Page 2: 0 4 2
Page 3: 3 4 2
Page 0: 0 4 2
Page 3: 0 3 2
Total Page Faults = 10
```

c. Optimal

```
#include <stdio.h>

int findOptimal(int pageSeq[], int frame[], int frames, int current, int pages) {
    int res = -1, farthest = current;
    for (int i = 0; i < frames; i++) {
        int j;
        for (j = current; j < pages; j++) {
            if (frame[i] == pageSeq[j])
                break;
        }
        if (j == pages)
            return i;
        if (j > farthest) {
            farthest = j;
            res = i;
        }
    }
    return (res == -1) ? 0 : res;
}

int main() {
    int frames, pages;
    printf("Enter number of frames: ");
    scanf("%d", &frames);
    printf("Enter number of pages: ");
    scanf("%d", &pages);

    int pageSeq[pages];
    printf("Enter page reference string:\n");
```

```
for (int i = 0; i < pages; i++)
    scanf("%d", &pageSeq[i]);

int frame[frames];
for (int i = 0; i < frames; i++)
    frame[i] = -1;

int pageFaults = 0, count = 0;

for (int i = 0; i < pages; i++) {
    int found = 0;
    for (int j = 0; j < frames; j++) {
        if (frame[j] == pageSeq[i]) {
            found = 1;
            break;
        }
    }
    if (!found) {
        if (count < frames) {
            frame[count++] = pageSeq[i];
        } else {
            int idx = findOptimal(pageSeq, frame, frames, i + 1, pages);
            frame[idx] = pageSeq[i];
        }
        pageFaults++;
    }
    printf("Page %d: ", pageSeq[i]);
    for (int j = 0; j < frames; j++) {
        if (frame[j] != -1)
            printf("%d ", frame[j]);
```

```
            else
                printf("- ");
        }
        printf("\n");
    }

    printf("Total Page Faults = %d\n", pageFaults);

    return 0;
}
```

output

```
PS C:\Users\shett\Downloads\os> gcc optimal.c -o optimal
PS C:\Users\shett\Downloads\os> .\optimal
Enter number of frames: 3
Enter number of pages: 12
Enter page reference string:
7 0 1 2 0 3 0 4 2 3 0 3
Page 7: 7 - -
Page 0: 7 0 -
Page 1: 7 0 1
Page 2: 2 0 1
Page 0: 2 0 1
Page 3: 3 0 1
Page 0: 3 0 1
Page 4: 3 0 4
Page 2: 2 0 4
Page 3: 2 3 4
Page 0: 0 3 4
Page 3: 0 3 4
Total Page Faults = 9
```

Program -8

Write a C program to simulate the following file allocation strategies.

a. Sequential

```
#include <stdio.h>

int main() {
    int diskSize, start, length;
    printf("Enter total disk size: ");
    scanf("%d", &diskSize);

    int disk[diskSize];
    for (int i = 0; i < diskSize; i++)
        disk[i] = 0;

    printf("Enter starting block and length of file: ");
    scanf("%d%d", &start, &length);

    if (start + length > diskSize) {
        printf("Error: Not enough space.\n");
        return 0;
    }

    for (int i = start; i < start + length; i++) {
        if (disk[i] == 1) {
            printf("Block %d already allocated. Cannot allocate file.\n", i);
            return 0;
        }
```

```
    }

    for (int i = start; i < start + length; i++)
        disk[i] = 1;

    printf("File allocated from block %d to %d\n", start, start + length - 1);

    return 0;
}
```

output

```
PS C:\Users\shett\Downloads\os> gcc sequential.c -o sequential
PS C:\Users\shett\Downloads\os> .\sequential
Enter total disk size: 20
Enter starting block and length of file: 5 6
File allocated from block 5 to 10
```

b. Indexed

```
#include <stdio.h>

int main() {
    int diskSize, indexBlock, fileSize;
    printf("Enter total disk size: ");
    scanf("%d", &diskSize);

    int disk[diskSize];
    for (int i = 0; i < diskSize; i++)
        disk[i] = 0;

    printf("Enter index block number: ");
    scanf("%d", &indexBlock);
    if (indexBlock >= diskSize) {
        printf("Invalid index block.\n");
        return 0;
    }

    printf("Enter file size (number of blocks): ");
    scanf("%d", &fileSize);

    if (disk[indexBlock] == 1) {
        printf("Index block already allocated.\n");
        return 0;
    }

    int blocks[fileSize];
    printf("Enter data block numbers:\n");
    for (int i = 0; i < fileSize; i++) {
```

```
        scanf("%d", &blocks[i]);
        if (blocks[i] >= diskSize) {
            printf("Invalid block number %d.\n", blocks[i]);
            return 0;
        }
    }

    if (disk[indexBlock] == 1) {
        printf("Index block already allocated.\n");
        return 0;
    }

    for (int i = 0; i < fileSize; i++) {
        if (disk[blocks[i]] == 1) {
            printf("Block %d already allocated. Cannot allocate file.\n", blocks[i]);
            return 0;
        }
    }

    disk[indexBlock] = 1;
    for (int i = 0; i < fileSize; i++)
        disk[blocks[i]] = 1;

    printf("File allocated with index block %d and data blocks: ", indexBlock);
    for (int i = 0; i < fileSize; i++)
        printf("%d ", blocks[i]);
    printf("\n");

    return 0;
}
```

output

```
PS C:\Users\shett\Downloads\os> gcc indexed.c -o indexed
PS C:\Users\shett\Downloads\os> .\indexed
Enter total disk size: 30
Enter index block number: 5
Enter file size (number of blocks): 4
Enter data block numbers:
6 7 8 9
File allocated with index block 5 and data blocks: 6 7 8 9
```

c. Linked

```
#include <stdio.h>

int main() {
    int diskSize, fileSize;
    printf("Enter total disk size: ");
    scanf("%d", &diskSize);

    int disk[diskSize];
    for (int i = 0; i < diskSize; i++)
        disk[i] = 0;

    printf("Enter file size (number of blocks): ");
    scanf("%d", &fileSize);

    int blocks[fileSize];
    printf("Enter block numbers in sequence:\n");
    for (int i = 0; i < fileSize; i++) {
        scanf("%d", &blocks[i]);
        if (blocks[i] >= diskSize) {
            printf("Invalid block number %d.\n", blocks[i]);
            return 0;
        }
    }

    for (int i = 0; i < fileSize; i++) {
        if (disk[blocks[i]] == 1) {
            printf("Block %d already allocated. Cannot allocate file.\n", blocks[i]);
            return 0;
        }
```

```
    }

    for (int i = 0; i < fileSize; i++)
        disk[blocks[i]] = 1;

    printf("File allocated in linked manner with blocks: ");
    for (int i = 0; i < fileSize; i++)
        printf("%d ", blocks[i]);
    printf("\n");

    return 0;
}
```

output

```
PS C:\Users\shett\Downloads\os> gcc linked.c -o linked
PS C:\Users\shett\Downloads\os> .\linked
Enter total disk size: 25
Enter file size (number of blocks): 5
Enter block numbers in sequence:
3 5 10 11 15
File allocated in linked manner with blocks: 3 5 10 11 15
```