OPERATING SYSTEM LAB REPORT

Name- Vibhanshu Sharma
Adm. No. : 21JE1039

INDEX

1. **Implementation of the following CPU Scheduling Algorithms:**
   (i)FCFS(First Come First Serve) Scheduling **12-Jan-2023**
   (ii)SJF(Shortest Job First) Scheduling

2. **Implementation of the following CPU Scheduling Algorithms:-**
   (i)SRTF(Shortest Remaining Time First) Scheduling
   (ii)Priority Scheduling **19-Jan-2023**

3. **Implementation of Banker's Algorithm to find all possible safe sequences** **28-Jan-2023**

4. **Basics of UNIX Commands** **02-Feb-2023**

5. **Basics of Shell Programming** **02-Feb-2023**

6. **Implementation of Memory Management Techniques- First Fit, Best Fit and Worst Fit.** **09-Feb-2023**

7. **Producer - Consumer Problem using Semaphores** **16-Feb-2023**

8. **Implementation of the following Disk Scheduling Algorithms :-**
   (i)FCFS(First Come First Serve) Algorithm
   (ii)SSTF(Shortest Seek Time First)Algorithm. **23-Feb-2023**

9. **Implementation of the following Disk Scheduling Algorithms:-**
   (i)SCAN Algorithm
   (ii)LOOK Algorithm **23-Feb-2023**

10. **Implementation of Basic Shell Programming:-**
    (i)Finding the n^{th} term of the Fibonacci Sequence **16-Mar-2023**
    (ii)Checking whether a number is palindrome or not

11. **Implementation of the following Page Replacement Algorithms:-**
    (i)FIFO(First In First Out)Algorithm **23-Mar-2023**
    (ii)LRU(Least Recently Used) Algorithm

12. **Implementation of the Following Page Replacement Algorithms:-**
    (i)Optimal Algorithm
    (ii)MRU(Most Recently Used) Algorithm **06-Apr-2023**

LAB - 01

AIM ⇒To write the program to implement the CPU Scheduling algorithm for First Come First Serve(FCFS) and Shortest Job First(SJF).

THEORY⇒

Given n processes with their burst times, the task is to find average waiting time and average turn around time using FCFS scheduling algorithm.
First in, first out (FIFO), also known as first come, first served (FCFS), is the simplest scheduling algorithm. FIFO simply queues processes in the order that they arrive in the ready queue. In this, the process that comes first will be executed first and the next process starts only after the previous gets fully executed.

Shortest Job First (SJF) is a Scheduling Algorithm where the process are executed in ascending order of their burst time, that is, the process having the shortest burst time is executed first and so on.The processor knows the burst time of each process in advance.It can be thought of as the shortest-next-cpu-burst algorithm, as Scheduling depends on the length of the next CPU burst of a process. (The duration for which a process gets control of the CPU, is the Burst time for a process.)

CODE⇒
CODE FOR FCFS

```
/*
Name: Vibhanhsu Sharma
Adm no. : 21JE1039
*/
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int num_datasets;
    cout << "Enter the number of datasets: ";
    cin >> num_datasets;

    if (num_datasets <= 0)
    {
```

```
        cout << "WRONG INPUT\n";
        exit(0);
    }

    vector<int> burst_time(num_datasets),
arrival_time(num_datasets), completion_time(num_datasets),
turnaround_time(num_datasets), process_id(num_datasets),
waiting_time(num_datasets);

    cout << "PINDEX\tARR. TIME\tBURST TIME\n";
    for (int i = 0; i < num_datasets; i++)
    {
        process_id[i] = i + 1;
        cout << process_id[i] << "\t";
        cin >> arrival_time[i];

        if (arrival_time[i] < 0)
        {
            cout << "WRONG INPUT\n";
            exit(0);
        }

        cin >> burst_time[i];

        if (burst_time[i] < 0)
        {
            cout << "WRONG INPUT\n";
            exit(0);
        }
    }

    vector<pair<int, int>> dataset, backup_dataset;

    for (int i = 0; i < num_datasets; i++)
    {
        dataset.push_back(make_pair(arrival_time[i],
burst_time[i]));
        backup_dataset.push_back(make_pair(arrival_time[i],
```

```
i));
    }

    sort(dataset.begin(), dataset.end());
    sort(backup_dataset.begin(), backup_dataset.end());

    for (int i = 0; i < num_datasets; i++)
    {
        cout << dataset[i].first << "\t" <<
dataset[i].second << endl;
    }

    completion_time[0] = dataset[0].first +
dataset[0].second;
    waiting_time[0] = 0;
    turnaround_time[0] = completion_time[0] -
dataset[0].first;

    for (int i = 1; i < num_datasets; i++)
    {
        if (completion_time[i - 1] < dataset[i].first)
        {
            completion_time[i] = completion_time[i - 1] +
((dataset[i].first) - completion_time[i - 1]) +
dataset[i].second;
        }
        else
        {
            completion_time[i] = completion_time[i - 1] +
dataset[i].second;
        }
    }

    for (int i = 1; i < num_datasets; i++)
    {
        if (completion_time[i - 1] - dataset[i].first >= 0)
        {
            waiting_time[i] = completion_time[i - 1] -
```

```
dataset[i].first;
        }
        else
        {
            waiting_time[i] = 0;
        }
        turnaround_time[i] = completion_time[i] -
dataset[i].first;
    }

    cout << "PINDEX\tCOM. TIME\tWT. TIME\tTAT TIME\n";
    for (int i = 0; i < num_datasets; i++)
    {
        cout << backup_dataset[i].second + 1 << "\t\t" <<
completion_time[i] << "\t\t" << waiting_time[i] << "\t\t"
<< turnaround_time[i] << "\n";
    }

    int total_completion_time = 0, total_waiting_time = 0,
total_turnaround_time = 0;

    for (int i = 0; i < num_datasets; i++)
    {
        total_completion_time += completion_time[i];
        total_waiting_time += waiting_time[i];
        total_turnaround_time += turnaround_time[i];
    }

    float average_completion_time, average_waiting_time,
average_turnaround_time;

    average_completion_time = float(total_completion_time)
/ num_datasets;
    average_waiting_time = float(total_waiting_time) /
num_datasets;
    average_turnaround_time = float(total_turnaround_time)
/ num_datasets;
    cout << "Average Completion Time = " <<
```

```
average_completion_time << "\n";

    cout << "Average Waiting Time = " <<
average_waiting_time << "\n";

    cout << "Average Turn Around Time = " <<
average_turnaround_time << "\n";
    return 0;

}
```

CODE FOR SJF

```
/*
Name: Vibhanhsu Sharma
Adm no. : 21JE1039
*/
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int num_datasets;
    cout << "Enter the number of datasets\n";
    cin >> num_datasets;
    if (num_datasets <= 0)
    {
        cout << "WRONG INPUT\n";
        exit(0);
    }
    int burst_time[num_datasets],
arrival_time[num_datasets], completion_time[num_datasets],
turnaround_time[num_datasets], process[num_datasets],
waiting_time[num_datasets];
    cout << "PINDEX   ARR. TIME    BURST TIME\n";
    for (int i = 0; i < num_datasets; i++)
    {
```

```
        process[i] = i + 1;
        cout << process[i];
        cin >> arrival_time[i];
        if (arrival_time[i] < 0)
        {
            cout << "WRONG INPUT\n";
            exit(0);
        }
        cin >> burst_time[i];
        if (burst_time[i] < 0)
        {
            cout << "WRONG INPUT\n";
            exit(0);
        }
    }
    vector<pair<int, int>> vect, back;
    for (int i = 0; i < num_datasets; i++)
    {
        vect.push_back(make_pair(burst_time[i],
arrival_time[i]));
        back.push_back(make_pair(burst_time[i], i));
    }
    sort(vect.begin(), vect.end());
    sort(back.begin(), back.end());
    for (int i = 0; i < num_datasets; i++)
    {
        cout << vect[i].first << " "
             << vect[i].second << endl;
    }
    completion_time[0] = vect[0].first + vect[0].second;
    waiting_time[0] = 0;
    turnaround_time[0] = completion_time[0] -
vect[0].second;
    for (int i = 1; i < num_datasets; i++)
    {
        if (completion_time[i - 1] < vect[i].second)
        {
            completion_time[i] = completion_time[i - 1] +
```

```
((vect[i].second) - completion_time[i - 1]) +
vect[i].first;
        }
        else
        {
            completion_time[i] = completion_time[i - 1] +
vect[i].first;
        }
    }
    for (int i = 1; i < num_datasets; i++)
    {
        if (completion_time[i - 1] - vect[i].second >= 0)
        {
            waiting_time[i] = completion_time[i - 1] -
vect[i].second;
        }
        else
        {
            waiting_time[i] = 0;
        }
        turnaround_time[i] = completion_time[i] -
vect[i].second;
    }
    for (int i = 0; i < num_datasets; i++)
    {
        cout << "PINDEX\tCOM. TIME\tWT. TIME\tTAT TIME\n";
        cout << back[i].second + 1 << "\t\t" <<
completion_time[i] << "\t\t" << waiting_time[i] << "\t\t"
<< turnaround_time[i] << "\n";
    }
    int total_completion_time = 0, total_waiting_time = 0,
total_turnaround_time = 0;
    for (int i = 0; i < num_datasets; i++)
    {
        total_completion_time += completion_time[i];
        total_waiting_time += waiting_time[i];
        total_turnaround_time += turnaround_time[i];
    }
```

```
    float avg_completion_time, avg_waiting_time,
avg_turnaround_time;
    avg_completion_time = (float)total_completion_time /
num_datasets;
    avg_waiting_time = (float)total_waiting_time /
num_datasets;
    avg_turnaround_time = (float)total_turnaround_time /
num_datasets;
    cout << "Average Completion Time = " <<
avg_completion_time << "\n";
    cout << "Average Waiting Time = " << avg_waiting_time
<< "\n";
    cout << "Average Turn Around Time = " <<
avg_turnaround_time << "\n";
    return 0;
}
```

OUTPUT⇒

Output for FCFS

```
Enter the number of datasets
6
PINDEX   ARR. TIME    BURST TIME
1 0 9
2 1 3
3 1 2
4 1 4
5 2 3
6 3 2
```

```
PINDEX  COM. TIME       WT. TIME        TAT TIME
1               9               0               9
PINDEX  COM. TIME       WT. TIME        TAT TIME
2               11              8               10
PINDEX  COM. TIME       WT. TIME        TAT TIME
3               14              10              13
PINDEX  COM. TIME       WT. TIME        TAT TIME
4               18              13              17
PINDEX  COM. TIME       WT. TIME        TAT TIME
5               21              16              19
PINDEX  COM. TIME       WT. TIME        TAT TIME
6               23              18              20
Average Completion Time = 16
Average Waiting Time = 10
Average Turn Around Time = 14.6667
```

OUTPUT FOR SJF

```
Enter the number of datasets
6
PINDEX   ARR. TIME    BURST TIME
1 0 9
2 1 3
3 1 2
4 1 4
5 2 3
6 3 2
```

```
PINDEX  COM. TIME       WT. TIME        TAT TIME
3               3               0               2
PINDEX  COM. TIME       WT. TIME        TAT TIME
6               5               0               2
PINDEX  COM. TIME       WT. TIME        TAT TIME
2               8               4               7
PINDEX  COM. TIME       WT. TIME        TAT TIME
5               11              6               9
PINDEX  COM. TIME       WT. TIME        TAT TIME
4               15              10              14
PINDEX  COM. TIME       WT. TIME        TAT TIME
1               24              15              24
Average Completion Time = 11
Average Waiting Time = 5
Average Turn Around Time = 9.66667
```

LAB - 02

AIM ⇒To write the program to implement the CPU Scheduling algorithm for Shortest Remaining Time First(SRTF) Scheduling and Priority Scheduling.

THEORY⇒

Shortest remaining time, also known as shortest remaining time first (SRTF), is a scheduling method that is a preemptive version of shortest job next scheduling. In this scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute. Since the currently executing process is the one with the shortest amount of time remaining by definition, and since that time should only reduce as execution progresses, the process will either run until it completes or get preempted if a new process is added that requires a smaller amount of time.

In Priority scheduling, there is a priority number assigned to each process. In some systems, the lower the number, the higher the priority. While, in the others, the higher the number, the higher will be the priority. The Process with the higher priority among the available processes is given the CPU.

CODE ⇒

CODE FOR SRTF

```
/*
Name: Vibhanhsu Sharma
Adm no. : 21JE1039
*/
#include <bits/stdc++.h>
using namespace std;

struct Process
{
    int pid;
    int burst_time;
    int arrival_time;
};

void findWaitingTime(Process proc[], int n, int waiting_time[])
```

```
{
    int remaining_time[n];
    for (int i = 0; i < n; i++)
    {
        remaining_time[i] = proc[i].burst_time;
    }

    int completed = 0, time = 0, minimum = INT_MAX;
    int shortest = 0, finish_time;
    bool found = false;

    while (completed != n)
    {
        for (int j = 0; j < n; j++)
        {
            if ((proc[j].arrival_time <= time) &&
                (remaining_time[j] < minimum) &&
remaining_time[j] > 0)
            {
                minimum = remaining_time[j];
                shortest = j;
                found = true;
            }
        }

        if (found == false)
        {
            time++;
            continue;
        }

        remaining_time[shortest]--;

        minimum = remaining_time[shortest];
        if (minimum == 0)
        {
            minimum = INT_MAX;
        }

        if (remaining_time[shortest] == 0)
        {
```

```
            completed++;
            found = false;
            finish_time = time + 1;
            waiting_time[shortest] = finish_time -
proc[shortest].burst_time - proc[shortest].arrival_time;

            if (waiting_time[shortest] < 0)
            {
                waiting_time[shortest] = 0;
            }
        }

        time++;
    }
}

void findTurnAroundTime(Process proc[], int n, int
waiting_time[], int turnaround_time[])
{
    for (int i = 0; i < n; i++)
    {
        turnaround_time[i] = proc[i].burst_time +
waiting_time[i];
    }
}

void findAvgTime(Process proc[], int n)
{
    int waiting_time[n], turnaround_time[n], total_waiting_time
= 0, total_turnaround_time = 0;

    findWaitingTime(proc, n, waiting_time);
    findTurnAroundTime(proc, n, waiting_time, turnaround_time);

    cout << "ID\t\t"
         << "CT\t\t"
         << "WT\t\t"
         << "TAT\t\t\n";

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

```
        total_waiting_time = total_waiting_time +
waiting_time[i];
        total_turnaround_time = total_turnaround_time +
turnaround_time[i];
        cout << " " << proc[i].pid << "\t\t" <<
proc[i].arrival_time + turnaround_time[i] << "\t\t " <<
waiting_time[i] << "\t\t " << turnaround_time[i] << endl;
    }

    int completion_time[n];
    for (int i = 0; i < n; i++)
    {
        completion_time[i] = turnaround_time[i] +
proc[i].arrival_time;
    }

    int maximum = completion_time[0];
    for (int i = 1; i < n; i++)
    {
        maximum = max(maximum, completion_time[i]);
    }

    cout << "\nCompletion time = " << maximum;
    cout << "\nAverage waiting time = " <<
(float)total_waiting_time / (float)n;
    cout << "\nAverage turn around time = " <<
(float)total_turnaround_time / (float)n;
}

int main()
{
    cout << "Enter the number of processes";
    int N;
    cin >> N;
    if (N < 0)
    {
        cout << "WRONG INPUT";
        exit(0);
    }
    Process proc[N];
    cout << "ARR. TIME    BURST TIME\n";
```

```
    for (int i = 0; i < N; i++)
    {
        cin >> proc[i].arrival_time;
        cin >> proc[i].burst_time;
        proc[i].pid = i + 1;
    }

    findAvgTime(proc, N);
    return 0;
}
```

Code for Priority

```
Enter the number of processes 6
ARR. TIME    BURST TIME
1 9
1 3
1 2
1 4
2 3
3 2
```

```
ID              CT              WT              TAT
 1              24               14              23
 2              8                4               7
 3              3                0               2
 4              15               10              14
 5              11               6               9
 6              5                0               2

Completion time = 24
Average waiting time = 5.66667
Average turn around time = 9.5
```

LAB - 03

AIM ⇒To write a program to find all the possible safe sequences using Banker's Algorithm.

THEORY⇒

The Bankers Algorithm in OS works by maintaining a matrix of maximum and allocated resources for each process, and then checking if the system is in a safe state before allowing a process to request additional resources. The algorithm checks if the request can be granted without compromising the safety of the system, by ensuring that the request does not cause a process to exceed its maximum resource needs and that there are enough resources available to grant the request.

CODE ⇒

```
/*Vibhanshu Sharma
21JE1039*/
#include<bits/stdc++.h>
#define P 3

#define R 3

int total = 0;

using namespace std;

bool is_available(int process_id, int allocated[][R],
                  int max[][R], int need[][R], int available[])
{

    bool flag = true;

    for (int i = 0; i < R; i++) {

        if (need[process_id][i] > available[i])
            flag = false;
    }
```

```
    return flag;
}

void safe_sequence(bool marked[], int allocated[][R], int
max[][R],int need[][R], int available[], vector<int> safe)
{

    for (int i = 0; i < P; i++) {

        if (!marked[i] && is_available(i, allocated, max, need,
available)) {


            marked[i] = true;

            for (int j = 0; j < R; j++)
                available[j] += allocated[i][j];

            safe.push_back(i);

            safe_sequence(marked, allocated, max, need,
available, safe);
            safe.pop_back();


            marked[i] = false;


            for (int j = 0; j < R; j++)
                available[j] -= allocated[i][j];
        }
    }


    if (safe.size() == P) {

        total++;
        for (int i = 0; i < P; i++)
          {
              char X=safe[i]+65;
```

```
            cout <<X;
            if (i != (P - 1))
                cout << " --> ";
        }

        cout << endl;
    }
}


int main()
{

    int allocation[P][R], max[P][R];
    int need[P][R],resource[R];
     int available[R];
     cout<<"ENTER ALLOCATION MATRIX:\n";
     for(int i=0;i<P;i++)
     {
          for(int j=0;j<R;j++)
          {
               cin>>allocation[i][j];
          }
     }

     cout<<"ENTER MAX MATRIX:\n";
     for(int i=0;i<P;i++)
     {
          for(int j=0;j<R;j++)
          {
               cin>>max[i][j];
          }
     }
     cout<<"ENTER TOTAL VECTOR:\n";
     for(int k=0;k<R;k++)
     {
          cin>>resource[k];
     }
    for (int i = 0; i < R; i++) {

        int sum = 0;
```

```
        for (int j = 0; j < P; j++)
            sum += allocation[j][i];

        available[i] = resource[i] - sum;
    }
    vector<int> safe;
    bool marked[P];
    memset(marked, false, sizeof(marked));

    for (int i = 0; i < P; i++)
        for (int j = 0; j < R; j++)
            need[i][j] = max[i][j] - allocation[i][j];

    cout << "Safe sequences are:" << endl;
    safe_sequence(marked, allocation, max, need, available,
safe);

    cout << "\nTotal number of safe sequences are " << total;
    return 0;
}
```

OUTPUT⇒

```
ENTER ALLOCATION MATRIX:
0 1 0
2 0 0
3 0 2
ENTER MAX MATRIX:
7 5 3
3 2 2
9 0 2
ENTER TOTAL VECTOR:
10 5 7
Safe sequences are:
B --> A --> C
B --> C --> A

Total number of safe sequences are 2
```

LAB - 04

AIM ⇒To implement the basic UNIX Commands in C.

THEORY⇒

UNIX is an operating system which was first developed in the 1960s, and has been under constant development ever since. By operating system, we mean the suite of programs which make the computer work. It is a stable, multi-user, multi-tasking system for servers, desktops and laptops.

In the computing field, fork() is the primary method of process creation on Unix-like operating systems. This function creates a new copy called the child out of the original process, that is called the parent. When the parent process closes or crashes for some reason, it also kills the child process.

CODE ⇒

```
/*
Name: Vibhanhsu Sharma
Adm no. : 21JE1039
*/
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

int main()
{
    pid_t process_id;
    process_id = fork();
    if (process_id < 0)
    {
        fprintf(stderr, "Fork Failed");
        return 1;
    }
    else if (process_id == 0)
    {
        printf("Child Process has started\n\n");
```

```
        int num, sum = 0;
        printf("Enter the value of num:\n\n");
        scanf("%d", &num);
        while (num > 0)
        {
            sum += num;
            num--;
        }
        printf("Child Process will sleep for 5s before
displaying result\n\n");
        sleep(5);
        printf("The sum is %d\n\n", sum);
        printf("Child Process has terminated\n\n");
        execlp("/bin/ls", "ls", NULL);
    }
    else
    {
        printf("\n\nParent Process has started\n\n");
        printf("\n\nProgram to print sum of n natural
numbers\n\n");
        wait(NULL);
        printf("Parent Process is now waiting\n\n");
        printf("Parent Process has terminated\n\n");
    }
    return 0;
}
```

OUTPUT ⇒

```
Parent Process has started



Program to print sum of n natural numbers

Child Process has started

Enter the value of n:

45
Child Process will sleep for 5s before displaying result

The sum is 1035

Child Process has terminated

Parent Process is now waiting

Parent Process has terminated
```

LAB - 05

AIM ⇒To implement the Memory Management Techniques - First Fit, Best Fit and Worst Fit.

THEORY⇒

First-Fit Allocation is a memory allocation technique used in operating systems to allocate memory to a process. In First-Fit, the operating system searches through the list of free blocks of memory, starting from the beginning of the list, until it finds a block that is large enough to accommodate the memory request from the process.

Best-Fit Allocation is a memory allocation technique used in operating systems to allocate memory to a process. In Best-Fit, the operating system searches through the list of free blocks of memory to find the block that is closest in size to the memory request from the process.

In the worst fit allocation technique, the process traverses the whole memory and always searches for the largest hole/partition, and then the process is placed in that hole/partition. It is a slow process because it has to traverse the entire memory to search the largest hole.

CODE ⇒

```
/*
Vibhanshu Sharma
21JE1039
*/
#include<bits/stdc++.h>
using namespace std;
int main() {
int m, n;
cout << "Enter the number of blocks\n";
cin >> m;
cout << "Enter the number of files\n";
cin >> n;
vector<pair <int,string> > vec;
int block_size[m], marked[m], internal_fragment[m];
for(int i = 0; i < m; i++) marked[i] = 0;
```

```
cout << "Enter the size of the blocks\n";
for(int i = 0; i < m; i++){
cin >> block_size[i];
internal_fragment[i] = block_size[i];
}
int proc_size[n], block_assigned[n];
cout << "Enter the size of the files\n";
for(int i = 0; i < n; i++){
cin >> proc_size[i];
block_assigned[i] = -1;
}
for(int i = 0; i < n; i++){
for(int j = 0; j < m; j++){
if(marked[j] == 0 && block_size[j] >= proc_size[i]){
marked[j] = 1;
internal_fragment[j] = block_size[j] - proc_size[i];
block_assigned[i] = j+1;
break;
}
}
}
cout << "\n";
cout << "CASE 1:First Fit\n";
cout << "The block number of all files is shown below:\n";
cout << "File No.\tBlock number\n";
for(int i = 0; i < n; i++){
if(block_assigned[i] == -1){
cout<<"\nFile "<<i+1<<" cannot be assigned a memory block\n\n";
continue;
}
cout << " " << i+1 << " \t\t";

if(block_assigned[i] == -1) cout << "Not accomodated";
else cout << block_assigned[i];
cout <<"\n";
}
int sum1=0;
cout << "The internal fragment size in each block is as
follows:\n";
cout << "Block Number\tFragment size\n";
for(int i = 0; i < m; i++){
```

```
cout << " " << i+1 << " \t\t";
cout << internal_fragment[i];
sum1+=internal_fragment[i];
cout <<"\n";
}
vec.push_back(make_pair(sum1,"FIRST FIT"));
cout<<"-----------------------------------------------------------
---------------------------------\n";
cout <<"\n\n";
cout << "CASE 2:Best Fit \n";
for(int i = 0; i < m; i++){
marked[i] = 0;
internal_fragment[i] = block_size[i];
}
for(int i = 0; i < n; i++) block_assigned[i] = -1;
for(int i = 0; i < n; i++){
int ind = -1;
for(int j = 0; j < m; j++){
if(marked[j] == 0 && block_size[j] >= proc_size[i]){
if(ind == -1){
ind = j;
}else{
if(block_size[j] < block_size[ind]) ind = j;
}
}
}
if(ind == -1) continue;
marked[ind] = 1;
block_assigned[i] = ind+1;
internal_fragment[ind] = block_size[ind] - proc_size[i];
}
cout << "The block number of all files is shown below:\n";
cout << "File Number\tBlock number\n";
for(int i = 0; i < n; i++){
if(block_assigned[i] == -1){
cout<<"\nFile "<<i+1<<" cannot be assigned a memory block\n\n";
continue;
}
cout << " " << i+1 << " \t\t";
if(block_assigned[i] == -1) cout << "Not accomodated";
else cout << block_assigned[i];
```

```
cout <<"\n";
}
cout << "The internal fragment size in each block is shown
below\n";
cout << "Block No.\tFragment size\n";
int sum2=0;
for(int i = 0; i < m; i++){
cout << " " << i+1 << " \t\t";
cout << internal_fragment[i];
sum2+=internal_fragment[i];
cout <<"\n";
}
vec.push_back(make_pair(sum2,"BEST FIT"));
cout<<"-----------------------------------------------------------
---------------------------------\n";
cout << "\n\n";
cout << "CASE 3: Worst fit\n";
for(int i = 0; i < m; i++){
marked[i] = 0;
internal_fragment[i] = block_size[i];
}
for(int i = 0; i < n; i++) block_assigned[i] = -1;
for(int i = 0; i < n; i++){
int ind = -1;
for(int j = 0; j < m; j++){
if(marked[j] == 0 && block_size[j] >= proc_size[i]){
if(ind == -1) ind = j;
else if(block_size[j] > block_size[ind]) ind = j;
}
}
if(ind == -1) continue;
marked[ind] = 1;
block_assigned[i] = ind+1;
internal_fragment[ind] = block_size[ind] - proc_size[i];
}
cout << "The block number of all FILES is shown below:\n";
cout << "File Number\tBlock number\n";
for(int i = 0; i < n; i++){
if(block_assigned[i] == -1){
cout<<"\nFile "<<i+1<<" cannot be assigned a memory block\n\n";
continue;
```

```
}
cout << " " << i+1 << " \t\t";
if(block_assigned[i] == -1) cout << "Not accomodated";
else cout << block_assigned[i];
cout <<"\n";
}
int sum3=0;
cout << "The internal fragment size in each block is shown
below\n";
cout << "Block Number\tFragment size\n";
for(int i = 0; i < m; i++){
cout << " " << i+1 << " \t\t";
cout << internal_fragment[i];
sum3+=internal_fragment[i];
cout <<"\n";
}
vec.push_back(make_pair(sum3,"WORST FIT"));
sort(vec.begin(),vec.end());
cout<<"THE MOST OPTIMAL PROCESS IS: ";
cout<<vec[0].second<<" WITH INTERNAL FRAGMENTATION BEING
"<<vec[0].first<<" MEMORY UNITS";
cout<<"\n-----------------------------------------------------------
-----------------------------------\n";
return 0;
}
```

OUTPUT ⇒

```
Enter the number of blocks
5
Enter the number of files
4
Enter the size of the blocks
20 100 40 200 10
Enter the size of the files
90 50 30 40
```

```
CASE 1:First Fit
The block number of all files is shown below:
File No.        Block number
 1              2
 2              4
 3              3

File 4 cannot be assigned a memory block

The internal fragment size in each block is as follows:
Block Number    Fragment size
 1              20
 2              10
 3              10
 4              150
 5              10
```

```
CASE 2:Best Fit
The block number of all files is shown below:
File Number     Block number
 1              2
 2              4
 3              3

File 4 cannot be assigned a memory block

The internal fragment size in each block is shown below
Block No.       Fragment size
 1              20
 2              10
 3              10
 4              150
 5              10
```

```
CASE 3: Worst fit
The block number of all FILES is shown below:
File Number     Block number
 1              4
 2              2
 3              3

File 4 cannot be assigned a memory block

The internal fragment size in each block is shown below
Block Number    Fragment size
 1              20
 2              50
 3              10
 4              110
 5              10
THE MOST OPTIMAL PROCESS IS: BEST FIT WITH INTERNAL FRAGMENTATION BEING 200 MEMORY UNITS
```

LAB - 06

CODE ⇒

```
/*Vibhanshu Sharma
21JE1039 */
#include<bits/stdc++.h>
using namespace std;
int mutx=1,full=0,prc_id=1000;
static int empty;
int main()
{
     cout<<"\nSOLVING THE PRODUCER CONSUMER PROBLEM USING
SEMAPHORES\n";
    int n;
    void producer();
    void consumer();
    int wait(int);
    int signal(int);
    cout<<"Enter the size of your buffer\n";
    cin>>empty;
    cout<<"OPTIONS AVAILABLE: 1.Producer\t2.Consumer\t3.Exit\n";
    while(1)
    {
            cout<<"\nEnter your required choice\n";
            cin>>n;
            if(n==1)
            {
                if((mutx==1) && (empty!=0))
                {
                    producer();
                }
                else
                {
                    cout<<"Buffer is full\n";
                }
            }
            else if(n==2)
            {
                if((mutx==1) && (full!=0))
                {
```

```
                    consumer();
                }
                else
                {
                    cout<<"Buffer is empty\n";
                }
            }
            else if(n==3)
            {
                exit(0);
            }
    }
    return 0;
}
int wait(int s)
{
    return (--s);
}
int signal(int s)
{
    return (++s);
}
void producer()
{
    mutx=wait(mutx);
    full=signal(full);
    empty=wait(empty);
    prc_id++;
    cout<<"\nProducer produces the item with item
id:\n"<<prc_id<<"\n";
    mutx=signal(mutx);
}
void consumer()
{
    mutx=wait(mutx);
    full=wait(full);
    empty=signal(empty);
    cout<<"\nConsumer consumes the item with item
id:\n"<<prc_id<<"\n";
     prc_id--;
mutx=signal(mutx);
```

```
}
```

OUTPUT ⇒

```
SOLVING THE PRODUCER CONSUMER PROBLEM USING SEMAPHORES
Enter the size of your buffer
4
OPTIONS AVAILABLE: 1.Producer   2.Consumer      3.Exit
```

```
Enter your required choice
1

Producer produces the item with item id:
1001

Enter your required choice
1

Producer produces the item with item id:
1002

Enter your required choice
1

Producer produces the item with item id:
1003

Enter your required choice
1
```

```
Producer produces the item with item id:
1004

Enter your required choice
1
Buffer is full

Enter your required choice
2

Consumer consumes the item with item id:
1004

Enter your required choice
2

Consumer consumes the item with item id:
1003
```

```
Enter your required choice
2

Consumer consumes the item with item id:
1002

Enter your required choice
2

Consumer consumes the item with item id:
1001

Enter your required choice
2
Buffer is empty

Enter your required choice
3


...Program finished with exit code 0
```

LAB - 07

CODE ⇒

FCFS

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
int n, startHead;
cout<<"Enter the number of disk requests\n";
cin>>n;
int req[n];
bool finished[n];
vector<int> order;
cout<<"Enter the cylinder number of the disk requests\n";
for(int i=0; i<n; i++)
cin>>req[i];
cout<<"Enter starting position of the head\n";
cin>>startHead;
cout<<"\nFCFS Algorithm\n";
int cyl = startHead;
int finishCount=0;
order.clear();
memset(finished, false, sizeof(finished));
for(int i=0;i<n;i++)
{
	order.push_back(req[i]);
}
int FCFS_count=0;
cout<<"Order of servicing requests:\n";
for(int i=0;i<order.size()-1;i++)
{
	cout<<order[i]<<" ";
	FCFS_count+=abs(order[i]-order[i+1]);
}
cout<<"\nThe total seek time for FCFS algorithm is:
"<<FCFS_count+abs(order[0]-cyl);
cout<<"\n";
	return 0;
}
```

SSTF

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
int n, startHead;
```

```
cout<<"Enter the number of disk requests\n";
cin>>n;
int req[n];
bool finished[n];
vector<int> order;
cout<<"Enter the cylinder number of the disk requests\n";
for(int i=0; i<n; i++)
cin>>req[i];
cout<<"Enter starting position of the head\n";
cin>>startHead;
cout<<"\nSSTF Algorithm\n";
int cyl = startHead;
int finishCount=0;
order.clear();
memset(finished, false, sizeof(finished));
int nearIdx = 0;
while(finishCount<n)
{
nearIdx = -1;
for(int i=0; i<n; i++)
{
if(finished[i])
continue;
if(nearIdx==-1)
{
nearIdx = i;
continue;
}
if(abs(cyl - req[i]) < abs(cyl - req[nearIdx]))
nearIdx = i;
}
finished[nearIdx] = true;
cyl = req[nearIdx];
order.push_back(req[nearIdx]);
finishCount++;
}
int SSTF_count=0;
cout<<"Order of servicing requests:\n";
for(int i=0;i<order.size()-1;i++)
{
	cout<<order[i]<<" ";
	SSTF_count+=abs(order[i+1]-order[i]);
}
cout<<"\nThe total seek time for SSTF algorithm is:
"<<SSTF_count+abs(order[0]-startHead);
cout<<"\n";
	return 0;
}
```

LOOK

```
#include <bits/stdc++.h>
using namespace std;
void LOOK(int arr[], int head, string direction,int size);
int main()
{
int n, startHead;
cout<<"Enter the number of disk requests\n";
cin>>n;
int req[n];
bool finished[n];
vector<int> order;
cout<<"Enter the cylinder number of the disk requests\n";
for(int i=0; i<n; i++)
cin>>req[i];
cout<<"Enter starting position of the head\n";
cin>>startHead;
LOOK(req, startHead, "left",n);
return 0;
}
void LOOK(int arr[], int head, string direction,int size)
{
int seek_count = 0;
int distance, cur_track;
vector<int> left, right;
vector<int> seek_sequence;
for (int i = 0; i < size; i++) {
if (arr[i] < head)
left.push_back(arr[i]);
if (arr[i] > head)
right.push_back(arr[i]);
}
sort(left.begin(), left.end());
sort(right.begin(), right.end());
int run = 2;
while (run--) {
if (direction == "left") {
for (int i = left.size() - 1; i >= 0; i--) {
cur_track = left[i];
seek_sequence.push_back(cur_track);
```

```
distance = abs(cur_track - head);
seek_count += distance;
head = cur_track;
}
direction = "right";
}
else if (direction == "right")
{
for (int i = 0; i < right.size(); i++) {
cur_track = right[i];
seek_sequence.push_back(cur_track);
distance = abs(cur_track - head);
seek_count += distance;
head = cur_track;
direction = "left";
}
}
}
cout << "Order of servicing requests: " << endl;
for (int i = 0; i < seek_sequence.size(); i++) {
cout << seek_sequence[i] <<" ";
}
cout << "\nThe total seek time for LOOK algorithm is: "<<
seek_count <<" ";

}
```

SCAN

```
#include <bits/stdc++.h>
using namespace std;
void SCAN(int arr[], int head, string direction,int size);
int main()
{
int n, startHead;
cout<<"Enter the number of disk requests\n";
cin>>n;
int req[n];
bool finished[n];
vector<int> order;
cout<<"Enter the cylinder number of the disk requests\n";
```

```
for(int i=0; i<n; i++)
cin>>req[i];
cout<<"Enter starting position of the head\n";
cin>>startHead;
 SCAN(req, startHead, "left",n);
return 0;
}
void SCAN(int arr[], int head, string direction,int size)
{
    int seek_count = 0;
    int distance, cur_track;
    vector<int> left, right;
    vector<int> seek_sequence;
if (direction == "left")
left.push_back(0);
for (int i = 0; i < size; i++) {
if (arr[i] < head)
left.push_back(arr[i]);
if (arr[i] > head)
right.push_back(arr[i]);
}
sort(left.begin(), left.end());
sort(right.begin(), right.end());
int run = 2;
while (run--) {
if (direction == "left") {
for (int i = left.size() - 1; i >= 0; i--) {
cur_track = left[i];
seek_sequence.push_back(cur_track);
distance = abs(cur_track - head);
seek_count += distance;
head = cur_track;
}
direction = "right";
}
else if (direction == "right")
{
for (int i = 0; i < right.size(); i++) {
cur_track = right[i];
seek_sequence.push_back(cur_track);
distance = abs(cur_track - head);
seek_count += distance;
head = cur_track;
direction = "left";
}
}
}
cout << "Order of servicing requests: " << endl;
for (int i = 0; i < seek_sequence.size(); i++) {
cout << seek_sequence[i] <<" ";
}
```

```
cout << "\nThe total seek time for SCAN algorithm is: "<< seek_count<<" ";

}
```

OUTPUT ⇒

```
Enter the number of disk requests
8
Enter the cylinder number of the disk requests
98 183 37 122 14 124 65 67
Enter starting position of the head
53

FCFS Algorithm
Order of servicing requests:
98 183 37 122 14 124 65
The total seek time for FCFS algorithm is: 640
```

```
Enter the number of disk requests
8
Enter the cylinder number of the disk requests
98 183 37 122 14 124 65 67
Enter starting position of the head
53

SSTF Algorithm
Order of servicing requests:
65 67 37 14 98 122 124
The total seek time for SSTF algorithm is: 236
```

```
Enter the number of disk requests
8
Enter the cylinder number of the disk requests
98 183 37 122 14 124 65 67
Enter starting position of the head
53
Order of servicing requests:
37 14 65 67 98 122 124 183
The total seek time for LOOK algorithm is: 208
```

```
Enter the number of disk requests
8
Enter the cylinder number of the disk requests
98 183 37 122 14 124 65 67
Enter starting position of the head
53
Order of servicing requests:
37 14 0 65 67 98 122 124 183
The total seek time for SCAN algorithm is: 236
```

LAB - 08

LAB - 09

LAB - 10

Project

CHAT BOT

Indian Institute of Technology (ISM), Dhanbad Semester IV, B.Tech
Instructor : Prof. Hari Om.
Teaching Assistant: Nikhil Kolnurkar.

TEAM MEMBERS:

Vibhanshu Sharma 21JE1039
Vishal Shrivastava 21JE1050
Veeramalla Abhiram 21JE1037
Vejendla Yasasvi 21JE1038

Theory (OS concept Used):

The primary purpose of any page replacement algorithm is to reduce the number of page faults. When a page replacement is required, the LRU page replacement algorithm replaces the least recently used page with a new page. This algorithm is based on the assumption that among all pages, the least recently used page will not be used for a long time. It is a popular and efficient page replacement technique.LRU stands for Least Recently Used. As the name suggests, this

algorithm is based on the strategy that whenever a page fault occurs, the least recently used page will be replaced with a new page. So, the page not utilized for the longest time in the memory (compared to all other pages) gets replaced. This strategy is known as LRU paging. A page fault occurs when a running program tries to access a piece (or page) of memory that is not already present in the main memory (RAM). On the other hand, if that page is already present in the memory, it is called a page hit. The LRU page replacement algorithm comes into the picture whenever a page fault occurs.

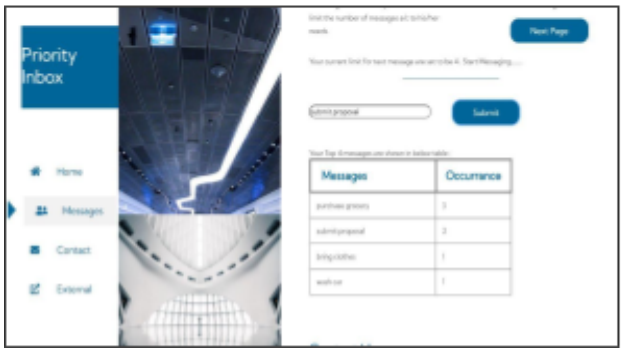
Project Description :

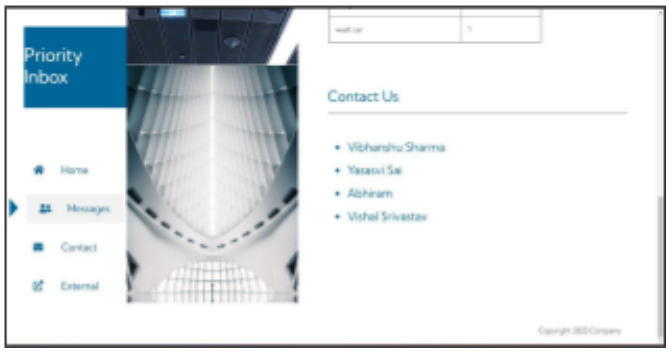
Chat bot is a web-application and message saving tool in which the user can arrange the task according to their priority order.It is used to display messages which are more frequently and recently posted.User can limit the number of messages according to his/her needs.Highest priority message will be shown at the top.Incase of same priority the message which arrived recently will be showing at first. This application is built based on the concepts of LRU algorithm of the Operating Systems course. It displays at most a maximum number of messages to which the limit is set.

Project Methodology:

1. A C++ file has been created which contains the code for the LRU algorithm used to implement the required logic.
2. Basic maps and strings are used to store the messages and the functions in library are used to output the messages.
3. A basic HTML file has been created to take inputs of the maximum message display limit and the messages to be posted and to display the required messages.
4. A CSS file been used for the outlook of the website and basic border design. The .js file contains the code for the integration of C++ file.\

Result Snapshots :



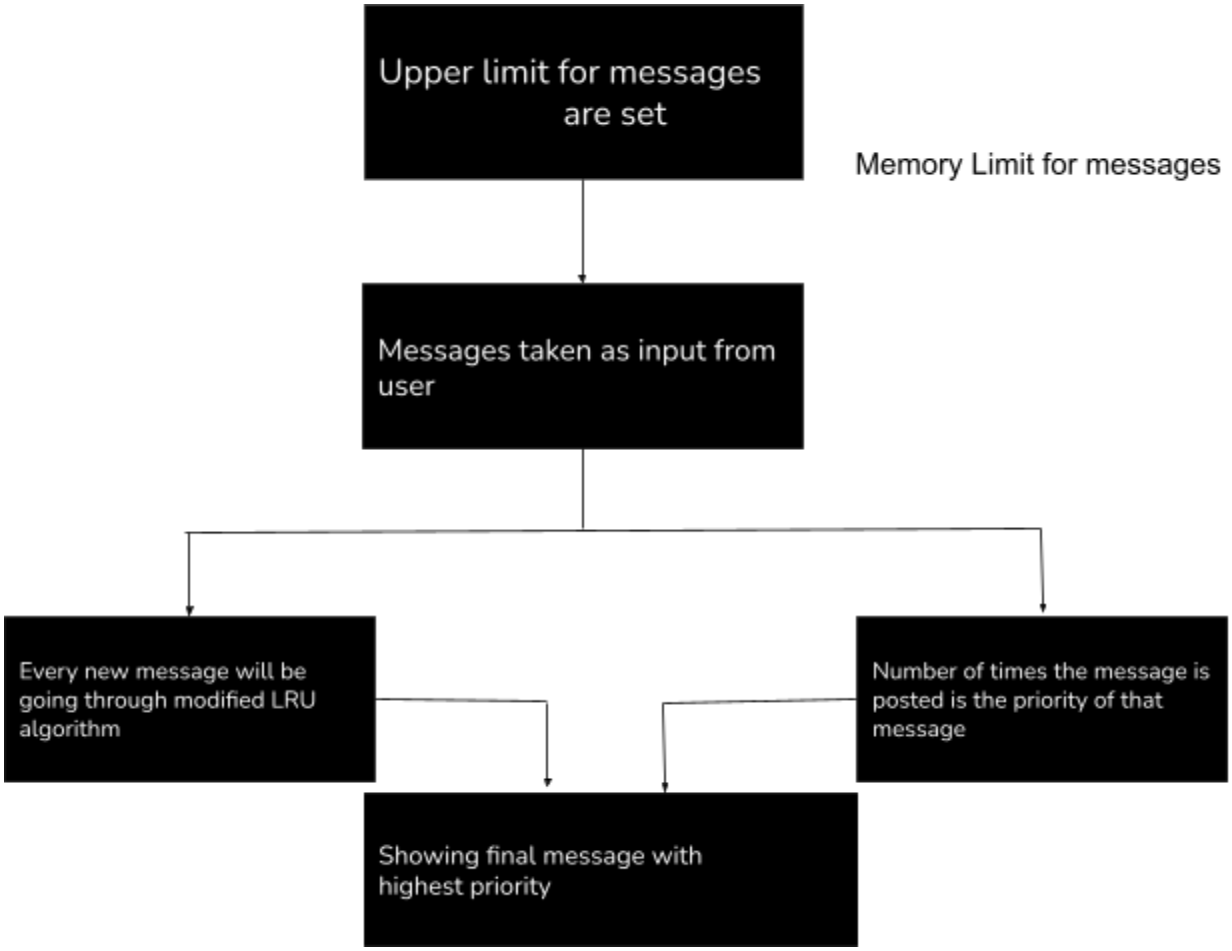


Github Link :https://github.com/vibhanshushrm2025/OS-Project

TechStack Used :

C++,
HTML,
CSS,
Javascript ,
LRU based Algorithms

FlowChart


Upper limit for messages are set
Memory Limit for messages
Messages taken as input from user
Every new message will be going through modified LRU algorithm
Number of times the message is posted is the priority of that message
Showing final message with highest priority