

Maulana Azad National Institute of Technology
(An Institute of National Importance)
Bhopal – 462003 (India)



Department of Computer Science & Engineering

A S S I G N M E N T S U B M I S S I O N

**Operating Systems Lab.
(CSE-317)**

Submitted by : Jishan Shaikh (Scholar no. 161112013)
Submitted to : Prof. Manish Pandey
Department of Computer Science & Engg.
MANIT, Bhopal (India)
Dated by : November 2, 2018 (Friday)
Subject : Operating Systems Lab
CSE-317, V Sem. (B.Tech. in CSE)
Session : Odd Semester 2018

This page intentionally left blank

Index of Programs

S.No.	Program Name	Date of submission	Page number
1	Kruskal algorithm implementation (C++)	Nov 2, 2018	
2	Prim's Algorithm implementation in C++	Nov 2, 2018	
3	Uniprogramming implementation in C++	Nov 2, 2018	
4	Multiprogramming implementation (C++)	Nov 2, 2018	
5	Time-sharing system implementation in C++	Nov 2, 2018	
6	Process Scheduling (FCFS) in C++	Nov 2, 2018	
7	Process Scheduling (SJF) in C++	Nov 2, 2018	
8	Process Scheduling (SRTF) in C++	Nov 2, 2018	
9	Non preemptive Priority Scheduling in C++	Nov 2, 2018	
10	Preemptive Priority Scheduling in C++	Nov 2, 2018	
11	Round Robin Scheduling in C++	Nov 2, 2018	
12	Banker's Algorithm in C++	Nov 2, 2018	
13	Memory Management (First fit) in C++	Nov 2, 2018	
14	Memory Management (Next fit) in C++	Nov 2, 2018	
15	Memory Management (Best fit) in C++	Nov 2, 2018	
16	Memory Management (Worst fit) in C++	Nov 2, 2018	
17	Page Replacement Algorithms (FIFO) in C++	Nov 2, 2018	
18	Optimal Page Replacement Algorithm in C++	Nov 2, 2018	

19	LRU Page Replacement Algorithm in C++	Nov 2, 2018	
20	Producer Consumer Problem in C++	Nov 2, 2018	
21	Critical Section Problem in C++	Nov 2, 2018	
22	Semaphore example in C	Nov 2, 2018	
23	Disk Scheduling implementation (FCFS, SSTF, SCAN) in C++	Nov 2, 2018	

Programs

1. Kruskal Algorithm implementation in C++

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> iPair;

struct Graph{
    int V, E;
    vector< pair<int, iPair> > edges;
    Graph(int V, int E){
        this->V = V;
        this->E = E;
    }
    void addEdge(int u, int v, int w){
        edges.push_back({w, {u, v}});
    }
    int kruskalMST();
};

struct DisjointSets{
    int *parent, *rnk;
    int n;
    DisjointSets(int n){
        this->n = n;
        parent = new int[n+1];
        rnk = new int[n+1];

        for (int i = 0; i <= n; i++){
            rnk[i] = 0;
            parent[i] = i;
        }
    }
    int find(int u){
        if (u != parent[u])
            parent[u] = find(parent[u]);
        return parent[u];
    }
    void merge(int x, int y){
        x = find(x), y = find(y);
        if (rnk[x] > rnk[y])
            parent[y] = x;
        else // If rnk[x] <= rnk[y]
            parent[x] = y;
        if (rnk[x] == rnk[y])
            rnk[y]++;
    }
};

int Graph::kruskalMST(){
    int mst_wt = 0;

    sort(edges.begin(), edges.end());

    DisjointSets ds(V);

    vector< pair<int, iPair> >::iterator it;
    for (it=edges.begin(); it!=edges.end(); it++){
```

```

        int u = it->second.first;
        int v = it->second.second;
        int set_u = ds.find(u);
        int set_v = ds.find(v);
        if (set_u != set_v){
            cout << u << " - " << v << endl;
            mst_wt += it->first;
            ds.merge(set_u, set_v);
        }
    }
    return mst_wt;
}

int main(){
    int V = 9, E = 14;
    Graph g(V, E);
    g.addEdge(0, 4, 8);
    g.addEdge(0, 7, 8);
    g.addEdge(1, 2, 8);
    g.addEdge(1, 7, 11);
    g.addEdge(2, 3, 7);
    g.addEdge(2, 8, 2);
    g.addEdge(2, 5, 4);
    g.addEdge(3, 4, 9);
    g.addEdge(3, 5, 14);
    g.addEdge(4, 5, 10);
    g.addEdge(5, 6, 2);
    g.addEdge(6, 7, 1);
    g.addEdge(6, 8, 6);
    g.addEdge(7, 8, 7);
    cout << "Edges of MST are \n";
    int mst_wt = g.kruskalMST();
    cout << "\nWeight of MST is " << mst_wt;
    return 0;
}

```

2. Prims Algorithm implementation in C++

```

#include<bits/stdc++.h>
using namespace std;

int a,b,u,v,n,i,j,ne=1;

int visited[10]={0},min,mincost=0,cost[10][10];

int main(){
    printf("\nEnter the number of nodes:");
    scanf("%d",&n);
    printf("\nEnter the adjacency matrix:\n");
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++){
            scanf("%d",&cost[i][j]);
            if(cost[i][j]==0)
                cost[i][j]=999;
        }
    visited[1]=1;
    printf("\n");
    while(ne < n){
        for(i=1,min=999;i<=n;i++)
            for(j=1;j<=n;j++)
                if(cost[i][j]< min)

```

```

        if(visited[i]!=0){
            min=cost[i][j];
            a=u=i;
            b=v=j;
        }
        if(visited[u]==0 || visited[v]==0){
            printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
            mincost+=min;
            visited[b]=1;
        }
        cost[a][b]=cost[b][a]=999;
    }
    printf("\n Minimum cost=%d",mincost);
    return 0;
}

```

3. Uniprogramming implementation in C++

```

// Uniprogramming Implementation in C++
// By: Jishan Shaikh

#include <bits/stdc++.h>
using namespace std;

int main(){
    float ttat=0, tat=0, cpuu=0, iou, tp=0, total=0, average=0, ipuu=0;
    int n;
    cin >> n;
    int p[n][5];
    for(int i=0; i<n; i++){
        tat = 0;
        for(int j=0; j<5; j++){
            cin >> p[i][j];
            ttat = ttat + p[i][j];
            tat = tat + p[i][j];
        }
        cout << "Turn around time of process " << i << " is " << ttat << endl;
        total += ttat;
    }
    average = total/n;
    cout << "Average turn around time is " << average << endl;
    for(int i=0; i<n; i++){
        cpuu += p[i][0] + p[i][2] + p[i][4];
    }
    cpuu = (cpuu*100)/ttat;
    ipuu = 100 - cpuu;
    cout << "Total CPU utilization is " << cpuu << "%" << endl;
    cout << "Total I/O utilization is " << ipuu << "%" << endl;
    cout << "Total turn around time is " << ttat << endl;
    cout << "Throughput is " << (n/ttat)*1000 << " processes per second." <<
endl;
    return 0;
}

```

4. Multiprogramming implementation in C++

```

#include <bits/stdc++.h>
using namespace std;

int main(){

```

```

int ms,i,ps[20],n,size,p[20],s,intr=0;
printf("Enter size of memory:");
scanf("%d",&ms);
printf("Enter memory for OS:");
scanf("%d",&s);
ms-=s;
printf("Enter no.of partitions to be divided:");
scanf("%d",&n);
size=ms/n;
for(i=0;i<n;i++){
    printf("Enter process and process size");
    scanf("%d%d",&p[i],&ps[i]);
    if(ps[i]<=size){
        intr=intr+size-ps[i];
        printf("process%d is allocated\n",p[i]);
    }
    else
        printf("process%d is blocked",p[i]);
}
printf("total fragmentation is %d",intr);
return 0;
}

```

5. Time-sharing system implementation in C++

```

#include <bits/stdc++.h>
using namespace std;

int main(){
    int ms,i,ps[20],n,size,p[20],s,intr=0;
    printf("Enter size of memory:");
    scanf("%d",&ms);
    printf("Enter memory for OS:");
    scanf("%d",&s);
    ms-=s;
    printf("Enter no.of partitions to be divided:");
    scanf("%d",&n);
    size=ms/n;
    for(i=0;i<n;i++){
        printf("Enter process and process size");
        scanf("%d%d",&p[i],&ps[i]);
        if(ps[i]<=size){
            intr=intr+size-ps[i];
            printf("process%d is allocated\n",p[i]);
        }
        else
            printf("process%d is blocked",p[i]);
    }
    printf("total fragmentation is %d",intr);
    return 0;
}

```

6. Process Scheduling (FCFS) in C++

```

#include <bits/stdc++.h>
using namespace std;

int main(){
    float
    process[500],aTime[500],bTime[500],abTime[500],wTime[500],tat_time[500];

```



```

int n = 0, i = 0 ;
float aw_time = 0, atat_time = 0;
printf("\nEnter the number of process : ");
scanf("%d", &n);

printf("Enter the Arrival time and Burst time.\n\n");
printf("\tA_Time B_Time\n");
for(i = 0 ; i < n ; i++){
    process[i]=i+1;
    printf("P%d :\t", i+1);
    scanf("%f\t%f", &aTime[i], &bTime[i]);
}
printf("\n\nProcess\tA_Time\tB_Time\n");
for(i = 0 ; i < n ; i++){
    printf("P[%d]\t%.2f\t%.2f\n", i, aTime[i], bTime[i]);
}
wTime[0] = 0;
tat_time[0] = bTime[0];
abTime[0] = bTime[0]+aTime[0];
for( i = 1 ; i < n ; i++){
    abTime[i] = abTime[i-1] + bTime[i];
    tat_time[i] = abTime[i] - aTime[i];
    wTime[i] = tat_time[i] - bTime[i];
}
for(i = 0 ; i < n ; i++){
    aw_time = aw_time + wTime[i];
    atat_time = atat_time + tat_time[i];
}
printf("\tA_time\tB_time\tC_time\tTat_time  W_time\n");
for(i = 0 ; i < n ; i++){
    printf("P[%d]\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", i, aTime[i], bTime[i], abTime[i], tat_time[i], wTime[i]);
}
printf("\nAverage waiting time : %.2f", aw_time/n);
return 0;
}

```

7. Process Scheduling (SJF) in C++

```

#include <bits/stdc++.h>
using namespace std;

int main(){
    int bt[20], p[20], wt[20], tat[20], i, j, n, total=0, pos, temp;
    float avg_wt, avg_tat;
    printf("Enter number of process:");
    scanf("%d", &n);
    printf("\nEnter Burst Time:\n");
    for(i=0; i<n; i++){
        printf("p%d:", i+1);
        scanf("%d", &bt[i]);
        p[i]=i+1;
    }
    for(i=0; i<n; i++){
        pos=i;
        for(j=i+1; j<n; j++){
            if(bt[j]<bt[pos])
                pos=j;
        }
        temp=bt[i];
        bt[i]=bt[pos];
    }
}

```

```

        bt[pos]=temp;

        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }

    wt[0]=0;
    for(i=1;i<n;i++){
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];
        total+=wt[i];
    }

    avg_wt=(float)total/n;
    total=0;
    printf("\nProcess\t\t Burst Time\t\t\tWaiting\t\tTurnaround Time");
    for(i=0;i<n;i++){
        tat[i]=bt[i]+wt[i];
        total+=tat[i];
        printf("\np%d\t\t %d\t\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }
    avg_tat=(float)total/n;
    printf("\n\nAverage Waiting Time=%f",avg_wt);
    printf("\nAverage Turnaround Time=%f\n",avg_tat);
}

```

8. Process Scheduling (SRTF) in C++

```

#include <bits/stdc++.h>
using namespace std;

int main(){
    int a[10],b[10],x[10],i,j,smallest,count=0,time,n;
    double avg=0,tt=0,end;
    printf("enter the number of Processes:\n");
    scanf("%d",&n);
    printf("enter arrival time\n");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    printf("enter burst time\n");
    for(i=0;i<n;i++)
        scanf("%d",&b[i]);
    for(i=0;i<n;i++)
        x[i]=b[i];

    b[9]=9999;

    for(time=0;count!=n;time++){
        smallest=9;
        for(i=0;i<n;i++){
            if(a[i]<=time && b[i]<b[smallest] && b[i]>0 )
                smallest=i;
        }
        b[smallest]--;
        if(b[smallest]==0){
            count++;
            end=time+1;
            avg=avg+end-a[smallest]-x[smallest];
            tt= tt+end-a[smallest];
        }
    }
}

```

```

    }
}
printf("\n\nAverage waiting time = %lf\n", avg/n);
printf("Average Turnaround time = %lf", tt/n);
return 0;
}

```

9. Process Scheduling (Non preemptive Priority Scheduling) in C++

```

#include <bits/stdc++.h>
using namespace std;

int main(){
    int burst_time[20], process[20], waiting_time[20], turnaround_time[20],
    priority[20];
    int i, j, limit, sum = 0, position, temp;
    float average_wait_time, average_turnaround_time;
    printf("Enter Total Number of Processes:\t");
    scanf("%d", &limit);
    printf("\nEnter Burst Time and Priority For %d Processes\n", limit);
    for(i = 0; i < limit; i++){
        printf("\nProcess[%d]\n", i + 1);
        printf("Process Burst Time:\t");
        scanf("%d", &burst_time[i]);
        printf("Process Priority:\t");
        scanf("%d", &priority[i]);
        process[i] = i + 1;
    }
    for(i = 0; i < limit; i++){
        position = i;
        for(j = i + 1; j < limit; j++){
            if(priority[j] < priority[position])
                position = j;
        }
        temp = priority[i];
        priority[i] = priority[position];
        priority[position] = temp;
        temp = burst_time[i];
        burst_time[i] = burst_time[position];
        burst_time[position] = temp;
        temp = process[i];
        process[i] = process[position];
        process[position] = temp;
    }
    waiting_time[0] = 0;
    for(i = 1; i < limit; i++){
        waiting_time[i] = 0;
        for(j = 0; j < i; j++)
            waiting_time[i] = waiting_time[i] + burst_time[j];

        sum = sum + waiting_time[i];
    }
    average_wait_time = sum / limit;
    sum = 0;
    printf("\nProcess ID\t\tBurst Time\t Waiting Time\t Turnaround Time\n");
    for(i = 0; i < limit; i++){
        turnaround_time[i] = burst_time[i] + waiting_time[i];
        sum = sum + turnaround_time[i];
        printf("\nProcess[%d]\t\t\t%d\t\t %d\t\t %d\n", process[i],
burst_time[i], waiting_time[i], turnaround_time[i]);
    }
}

```

```

        average_turnaround_time = sum / limit;
        printf("\nAverage Waiting Time:\t%f", average_wait_time);
        printf("\nAverage Turnaround Time:\t%f\n", average_turnaround_time);
        return 0;
}

```

10. Process Scheduling (Preemptive Priority Scheduling) in C++

```

#include <bits/stdc++.h>
using namespace std;

struct process{
    char process_name;
    int arrival_time, burst_time, ct, waiting_time, turnaround_time, priority;
    int status;
}process_queue[10];

int limit;

void Arrival_Time_Sorting(){
    struct process temp;
    int i, j;
    for(i = 0; i < limit - 1; i++)
        for(j = i + 1; j < limit; j++)
            if(process_queue[i].arrival_time >
process_queue[j].arrival_time){
                temp = process_queue[i];
                process_queue[i] = process_queue[j];
                process_queue[j] = temp;
            }
}

int main(){
    int i, time = 0, burst_time = 0, largest;
    char c;
    float wait_time = 0, turnaround_time = 0, average_waiting_time,
average_turnaround_time;
    printf("\nEnter Total Number of Processes:\t");
    scanf("%d", &limit);
    for(i = 0, c = 'A'; i < limit; i++, c++){
        process_queue[i].process_name = c;
        printf("\nEnter Details For Process[%C]:\n",
process_queue[i].process_name);
        printf("Enter Arrival Time:\t");
        scanf("%d", &process_queue[i].arrival_time );
        printf("Enter Burst Time:\t");
        scanf("%d", &process_queue[i].burst_time);
        printf("Enter Priority:\t");
        scanf("%d", &process_queue[i].priority);
        process_queue[i].status = 0;
        burst_time = burst_time + process_queue[i].burst_time;
    }
    Arrival_Time_Sorting();
    process_queue[9].priority = -9999;
    printf("\nProcess Name\tArrival Time\tBurst Time\tPriority\tWaiting
Time");
    for(time = process_queue[0].arrival_time; time < burst_time;){
        largest = 9;
        for(i = 0; i < limit; i++)

```

```

        if(process_queue[i].arrival_time <= time &&
process_queue[i].status != 1 && process_queue[i].priority >
process_queue[largest].priority)
        {
            largest = i;
        }

        time = time + process_queue[largest].burst_time;
        process_queue[largest].ct = time;
        process_queue[largest].waiting_time = process_queue[largest].ct -
process_queue[largest].arrival_time - process_queue[largest].burst_time;
        process_queue[largest].turnaround_time = process_queue[largest].ct -
process_queue[largest].arrival_time;
        process_queue[largest].status = 1;
        wait_time = wait_time + process_queue[largest].waiting_time;
        turnaround_time = turnaround_time +
process_queue[largest].turnaround_time;
        printf("\n%c\t\t%d\t\t%d\t\t%d\t\t%d",
process_queue[largest].process_name, process_queue[largest].arrival_time,
process_queue[largest].burst_time, process_queue[largest].priority,
process_queue[largest].waiting_time);
    }
    average_waiting_time = wait_time / limit;
    average_turnaround_time = turnaround_time / limit;
    printf("\n\nAverage waiting time:\t%f\n", average_waiting_time);
    printf("Average Turnaround Time:\t%f\n", average_turnaround_time);
}

```

11. Process Scheduling (Round Robin Scheduling) in C++

```

#include <bits/stdc++.h>
using namespace std;

int main() {
    int count,j,n,time,remain,flag=0,time_quantum;
    int wait_time=0,turnaround_time=0,at[10],bt[10],rt[10];
    printf("Enter Total Process:\t ");
    scanf("%d",&n);
    remain=n;
    for(count=0;count<n;count++){
        printf("Enter Arrival Time and Burst Time for Process Process Number
%d :",count+1);
        scanf("%d",&at[count]);
        scanf("%d",&bt[count]);
        rt[count]=bt[count];
    }
    printf("Enter Time Quantum:\t");
    scanf("%d",&time_quantum);
    printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
    for(time=0,count=0;remain!=0) {
        if(rt[count]<=time_quantum && rt[count]>0) {
            time+=rt[count];
            rt[count]=0;
            flag=1;
        }
        else if(rt[count]>0){
            rt[count]-=time_quantum;
            time+=time_quantum;
        }
        if(rt[count]==0 && flag==1) {
            remain--;
        }
    }
}

```

```

        printf("P[%d]\t|\t%d\t|\t%d\n", count+1, time-at[count], time-at[count]-
bt[count]);
        wait_time+=time-at[count]-bt[count];
        turnaround_time+=time-at[count];
        flag=0;
    }
    if(count==n-1)
        count=0;
    else if(at[count+1]<=time)
        count++;
    else
        count=0;
}
printf("\nAverage Waiting Time= %f\n", wait_time*1.0/n);
printf("Avg Turnaround Time = %f", turnaround_time*1.0/n);
return 0;
}

```

12. Bankers Algorithm implementation in C++

```

#include <bits/stdc++.h>
using namespace std;

const int n = 3, r = 1;

bool Safe(int processes[], int avail[], int maxm[][r], int allot[][r]){
    int need[n][r];
    for(int i=0; i<n; i++)
        for(int j=0; j<r; j++)
            need[i][j] = maxm[i][j] - allot[i][j];

    bool finish[n] = {0};
    int safeSeq[n];

    int work[r];
    for(int i=0; i<r; i++)
        work[i] = avail[i];

    int count = 0;
    while(count < n){
        bool found = false;
        for(int p=0; p<n; p++){
            if (finish[p] == 0){
                int j;
                for(j=0; j<r; j++)
                    if(need[p][j] > work[j])
                        break;

                if(j == r){
                    for(int k=0 ; k<r; k++)
                        work[k] += allot[p][k];

                    safeSeq[count++] = p;

                    finish[p] = 1;

                    found = true;
                }
            }
        }
    }
}

```

```

        if(found == false){
            cout << "ERROR: NOT IN SAFE STATE.";
            return false;
        }
    }

    cout << "SYSTEM IS IN SAFE STATE. \nTHE SAFE SEQUENCE IS : ";
    for(int i=0; i<n; i++)
        cout << safeSeq[i] << " ";

    return true;
}

int main(){
    int processes[n] = {0, 1, 2};
    int avail[r] = {1000};
    int maxm[n][r] = {{5000}, {7000}, {9000}};
    int allot[n][r] = {{4000}, {2000}, {3000}};
    Safe(processes, avail, maxm, allot);
    return 0;
}

```

13. Memory Management (First fit) in C++

```

#include <bits/stdc++.h>
using namespace std;

int main(){
    int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;
    for(i = 0; i < 10; i++){
        flags[i] = 0;
        allocation[i] = -1;
    }
    printf("Enter no. of blocks: ");
    scanf("%d", &bno);
    printf("\nEnter size of each block: ");
    for(i = 0; i < bno; i++)
        scanf("%d", &bsize[i]);
    printf("\nEnter no. of processes: ");
    scanf("%d", &pno);
    printf("\nEnter size of each process: ");
    for(i = 0; i < pno; i++)
        scanf("%d", &psize[i]);
    for(i = 0; i < pno; i++)
        for(j = 0; j < bno; j++)
            if(flags[j] == 0 && bsize[j] >= psize[i]){
                allocation[j] = i;
                flags[j] = 1;
                break;
            }
    printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");
    for(i = 0; i < bno; i++){
        printf("\n%d\t\t\t%d\t\t", i+1, bsize[i]);
        if(flags[i] == 1)
            printf("%d\t\t\t%d", allocation[i]+1, psize[allocation[i]]);
        else
            printf("Not allocated");
    }
}

```

14. Memory Management (Next fit) in C++

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    int memory_size[10][2], process_size[10][3];
    int i, j, total_processes = 0, total_memory = 0;
    printf("\nEnter the Total Number of Processes:\t");
    scanf("%d", &total_processes);
    printf("\nEnter the Size of Each Process\n");
    for(int i = 0; i < total_processes; i++){
        printf("Enter Size of Process %d:\t", i + 1);
        scanf("%d", &process_size[i][0]);
        process_size[i][1] = 0;
        process_size[i][2] = i;
    }
    printf("\nEnter Total Memory Blocks:\t");
    scanf("%d", &total_memory);
    printf("\nEnter the Size of Each Block:\n");
    for(i = 0; i < total_processes; i++){
        printf("Enter Size of Block %d:\t", i + 1);
        scanf("%d", &memory_size[i][0]);
        memory_size[i][1] = 0;
    }
    for(i = 0; i < total_processes; i++){
        while(j < total_memory){
            if(memory_size[j][1] == 0 && process_size[i][0]
<=memory_size[j][0]){
                process_size[i][1] = 1;
                memory_size[j][1] = 1;
                printf("\nProcess [%d] Allocated to Memory Block:\t%d",
i + 1, j + 1);
                break;
            }
            j++;
        }
    }
    for(i = 0; i < total_memory; i++){
        if(process_size[i][1] == 0){
            printf("\nProcess [%d] Unallocated\n", i + 1);
        }
    }
    printf("\n");
    return 0;
}
```

15. Memory Management (Best fit) in C++

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    int fragment[20], b[20], p[20], i, j, nb, np, temp, lowest=9999;
    static int barray[20], parray[20];
    printf("\n\t\t\t\tMemory Management Scheme - Best Fit");
    printf("\nEnter the number of blocks:");
    scanf("%d", &nb);
    printf("Enter the number of processes:");
    scanf("%d", &np);
```



```
#include <bits/stdc++.h>
using namespace std;
```

```
int main(){
    int fragments[10], blocks[10], files[10];
    int m, n, number_of_blocks, number_of_files, temp, top = 0;
    static int block_arr[10], file_arr[10];
    printf("\nEnter the Total Number of Blocks:\t");
    scanf("%d",&number_of_blocks);
    printf("Enter the Total Number of Files:\t");
    scanf("%d",&number_of_files);
    printf("\nEnter the Size of the Blocks:\n");
    for(m = 0; m < number_of_blocks; m++) {
        printf("Block No. [%d]:\t", m + 1);
        scanf("%d", &blocks[m]);
    }
    printf("Enter the Size of the Files:\n");
    for(m = 0; m < number_of_files; m++){
        printf("File No. [%d]:\t", m + 1);
        scanf("%d", &files[m]);
    }
    for(m = 0; m < number_of_files; m++){
        for(n = 0; n < number_of_blocks; n++){
            if(block_arr[n] != 1){
                temp = blocks[n] - files[m];
                if(temp >= 0){
                    if(top < temp){
```

```

        file_arr[m] = n;
        top = temp;
    }
}
fragments[m] = top;
block_arr[file_arr[m]] = 1;
top = 0;
}
}
printf("\nFile Number\tFile Size\tBlock Number\tBlock Size\tFragment");
for(m = 0; m < number_of_files; m++){
    printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", m, files[m], file_arr[m],
blocks[file_arr[m]], fragments[m]);
}
printf("\n");
return 0;
}

```

17. Page Replacement Algorithms (FIFO) in C++

```

#include <bits/stdc++.h>
using namespace std;

int pageFaults(vector <int> pages, int n, int f){
    unordered_set <int> s;

    queue <int> indexes;

    int page_faults = 0;
    for(int i=0; i<n; i++){
        if(s.size() < f){ // If MM is not full yet
            if(s.find(pages[i])==s.end()){ // Not found in MM
                s.insert(pages[i]);
                page_faults++;
                indexes.push(pages[i]);
            }
        }
        else{ // MM is full
            if(s.find(pages[i]) == s.end()){ // Not found in MM
                int val = indexes.front(); // Get first page
                indexes.pop();
                s.erase(val);
                s.insert(pages[i]);
                indexes.push(pages[i]);
                page_faults++;
            }
        }
    }
    return page_faults;
}

int main(){
    int n, f;
    cin >> n >> f;
    vector <int> pages(n);
    for(int i=0; i<n; i++)
        cin >> pages[i];
    int page_faults = pageFaults(pages, n, f);
    cout << "Page faults: " << page_faults;
    cout << "\nPage fault percentage: " << float(page_faults/n*100);
}

```

```

    return 0;
}

```

18. Page Replacement Algorithms (Optimal) in C++

```

#include <bits/stdc++.h>
using namespace std;

bool search(int key, vector<int> frame){ // Page exists in frame?
    for(int i=0; i<frame.size(); i++)
        if(frame[i] == key)
            return true;
    return false;
}

// find frame that will not be used in future
int futuree(vector<int> page, vector<int> frame, int n, int index){
    int res = -1, farthest = index;
    for(int i=0; i<frame.size(); i++){
        int j;
        for(j=index; j<n; j++){
            if(frame[i] == page[j]){
                if(j>farthest){
                    farthest=j;
                    res=i;
                }
            }
        }
        break;
    }
    if(j==n)
        return i;
}

if(res==-1)
    return 0;
return res;
}

void optimal(vector<int> page, int n, int m){
    vector<int> frame;

    int hits = 0;
    for(int i=0; i<n; i++){
        if(search(page[i], frame)){
            hits++;
            continue;
        }

        // Page not found in a frame : MISS

        // If there is space available in frames.
        if(frame.size() < m)
            frame.push_back(page[i]);

        // Find page to be replaced.
        else {
            int j = futuree(page, frame, n, i+1);
            frame[j] = page[i];
        }
    }
}

```

```

        cout << "Hits : " << hits << endl;
        cout << "Misses : " << n - hits << endl;
    }

    int main(){
        int m, n; // m frames, n pages
        cin >> n >> m;
        vector <int> page(n);
        for(int i=0; i<n; i++){
            cin >> page[i];
        }
        optimal(page, n, m);
        return 0;
    }

```

19. Page Replacement Algorithms (LRU) in C++

```

#include <bits/stdc++.h>
using namespace std;

int getLeastRecentlyUsed(vector <int> temp, unordered_set <int> s){
    int n = temp.size();
    for(int i=0; i<n; i++){
        if(s.find(temp[i]) != s.end()){ // found in mm
            return temp[i]; // return first hit in used array ""
        }
    }
    return 0;
}

vector <int> removee(int val, vector <int> temp){
    int n = temp.size();
    vector <int> t;
    for(int i=0; i<n; i++){
        if(temp[i] == val){
            for(int a=0; a<i; a++){
                t.push_back(temp[a]);
            }
            for(int a=i; a<n-1; a++){
                t.push_back(temp[a+1]);
            }
            break;
        }
    }
    return t;
}

int pageFaults(vector <int> pages, int n, int f){
    // set s is main memory, and temp is order of processes used
    unordered_set <int> s;

    vector <int> temp;

    int page_faults = 0;
    for(int i=0; i<n; i++){
        if(s.size() < f){ // If MM is not full yet
            if(s.find(pages[i])==s.end()){
                s.insert(pages[i]);
                page_faults++;
                temp.push_back(pages[i]);
            }
        }
    }
}

```

```

    }
    else{ // MM is full
        if(s.find(pages[i]) == s.end()){ // Not found in MM
            int val = getLeastRecentlyUsed(temp, s);
            temp = removee(val, temp); //remove val from temp
            s.erase(val);
            s.insert(pages[i]);
            temp.push_back(pages[i]);
            page_faults++;
        }
    }
}
return page_faults;
}

int main(){
    int n, f;
    cin >> n >> f;
    vector<int> pages(n);
    for(int i=0; i<n; i++)
        cin >> pages[i];
    int page_faults = pageFaults(pages, n, f);
    cout << "Page faults: " << page_faults;
    cout << "\nPage fault percentage: " << float((float)page_faults/(
    return 0;
}

```

20. Producer Consumer Problem in C++

```

#include <bits/stdc++.h>
using namespace std;

int mutex=1,full=0,empty=3,x=0;
int wait(int s){
    return (--s);
}
int signal(int s){
    return(++s);
}
void producer(){
    mutex=wait(mutex);
    full=signal(full);
    empty=wait(empty);
    x++;
    printf("\nProducer produces 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);
}
int main(){

    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;
}

```

21. Critical Section Problem in C++

```

#include "pthread.h"
#include "stdio.h"

// Importing POSIX Operating System API library
#include "unistd.h"

#include "string.h"

#define MEMBAR __sync_synchronize()
#define THREAD_COUNT 8

volatile int tickets[THREAD_COUNT];
volatile int choosing[THREAD_COUNT];

volatile int resource;

void lock(int thread){
    choosing[thread] = 1;

    MEMBAR;

    int max_ticket = 0;

    for (int i = 0; i < THREAD_COUNT; ++i) {
        int ticket = tickets[i];
        max_ticket = ticket > max_ticket ? ticket : max_ticket;
    }

    tickets[thread] = max_ticket + 1;

    MEMBAR;
    choosing[thread] = 0;
    MEMBAR;

    for (int other = 0; other < THREAD_COUNT; ++other) {
        while (choosing[other]) {
        }
    }
}

```

```

MEMBAR;

while (tickets[other] != 0 && (tickets[other]
                                < tickets[thread]
                                || (tickets[other]
                                    ==
tickets[thread]
                                && other < thread)))){
    }
}

void unlock(int thread) {
    MEMBAR;
    tickets[thread] = 0;
}

void use_resource(int thread) {
    if (resource != 0) {
        printf("Resource was acquired by %d, but is still in-use by %d!\n",
               thread, resource);
    }

    resource = thread;
    printf("%d using resource...\n", thread);

    MEMBAR;
    sleep(2);
    resource = 0;
}

void* thread_body(void* arg) {
    long thread = (long)arg;
    lock(thread);
    use_resource(thread);
    unlock(thread);
    return NULL;
}

int main(){
    memset((void*)tickets, 0, sizeof(tickets));
    memset((void*)choosing, 0, sizeof(choosing));
    resource = 0;
    pthread_t threads[THREAD_COUNT];

    for (int i = 0; i < THREAD_COUNT; ++i){
        pthread_create(&threads[i], NULL, &thread_body, (void*)((long)i));
    }
    for (int i = 0; i < THREAD_COUNT; ++i){
        pthread_join(threads[i], NULL);
    }
    return 0;
}

```

22. Semaphore example in C

```

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

```

```

sem_t mutex;
void* thread(void* arg){
    //wait
    sem_wait(&mutex);
    printf("\nEntered..\n");

    //critical section
    sleep(4);

    //signal
    printf("\nJust Exiting...\n");
    sem_post(&mutex);
}

int main(){
    sem_init(&mutex, 0, 1);
    pthread_t t1,t2;
    pthread_create(&t1,NULL,thread,NULL);
    sleep(2);
    pthread_create(&t2,NULL,thread,NULL);
    pthread_join(t1,NULL);
    pthread_join(t2,NULL);
    sem_destroy(&mutex);
    return 0;
}

```

23, 24, 25. Disk Scheduling implementation (FCFS, SSTF, and SCAN) in C++

```

#include <bits/stdc++.h>
using namespace std;

void fcfs(int noq, int qu[10], int st){
    int i,s=0;
    for(i=0;i<noq;i++){
        s=s+abs(st-qu[i]);
        st=qu[i];
    }
    printf("\n Total seek time :%d",s);
}

void sstf(int noq, int qu[10], int st, int visit[10]){
    int min,s=0,p,i;
    while(1){
        min=999;
        for(i=0;i<noq;i++){
            if (visit[i] == 0){
                if(min > abs(st - qu[i])){
                    min = abs(st-qu[i]);
                    p = i;
                }
            }
        }
        if(min == 999)
            break;
        visit[p]=1;
        s=s + min;
        st = qu[p];
    }
    printf("\n Total seek time is: %d",s);
}

```



```

}

void scan(int noq, int qu[10], int st, int ch){
    int i,j,s=0;
    for(i=0;i<noq;i++){
        if(st < qu[i]){
            for(j=i-1; j>= 0;j--){
                s=s+abs(st - qu[j]);
                st = qu[j];
            }
            if(ch == 3){
                s = s + abs(st - 0);
                st = 0;
            }
            for(j = 1;j < noq;j++){
                s= s + abs(st - qu[j]);
                st = qu[j];
            }
            break;
        }
    }
    printf("\n Total seek time : %d",s);
}

int main(){
    int n,qu[20],st,i,j,t,noq,ch,visit[20];
    printf("\n Enter the maximum number of cylinders : ");
    scanf("%d",&n);
    printf("enter number of queue elements");
    scanf("%d",&noq);
    printf("\n Enter the work queue");
    for(i=0;i<noq;i++){
        scanf("%d",&qu[i]);
        visit[i] = 0;
    }
    printf("\n Enter the disk head starting posision: \n");
    scanf("%d",&st);
    while(1){
        printf("\n\n\t\t MENU \n");
        printf("\n\n\t\t 1. FCFS \n");
        printf("\n\n\t\t 2. SSTF \n");
        printf("\n\n\t\t 3. SCAN \n");
        printf("\n\n\t\t 4. EXIT \n");
        printf("\nEnter your choice: ");
        scanf("%d",&ch);
        if(ch > 2){
            for(i=0;i<noq;i++)
                for(j=i+1;j<noq;j++)
                    if(qu[i]>qu[j]){
                        t=qu[i];
                        qu[i] = qu[j];
                        qu[j] = t;
                    }
        }
        switch(ch){
            case 1: printf("\n FCFS \n");
                    printf("\n*****\n");
                    fcfs(noq,qu,st);
                    break;

            case 2: printf("\n SSTF \n");
                    printf("\n*****\n");

```

```
        sstf(noq,qu,st,visit);
        break;
    case 3: printf("\n SCAN \n");
            printf("\n*****\n");
            scan(noq,qu,st,ch);
            break;
    case 4: exit(0);
}
}
}
```

■■■■■