OS -Qbank Solution

//ALL OS PRAC CODES SIMPLIFIED;)

FCFS PROCESS

```
#include <stdio.h>
void fcfs(int bt[],int n)
{
    int total_tat=0;
    int total_wt=0;
    int wt[n],tt[n];
    int strt_t=0;
    for(int i=0;i<n;i++)
    {
        wt[i]=strt_t;
        strt_t=strt_t+bt[i];
        tt[i]=strt_t;

        total_wt=total_wt+wt[i];
        total_tat=total_tat+tt[i];
    }
    printf("Process ID\tBurst Time\tTurnaround time\twaiting Time\n");
    for(int i=0;i<n;i++)
    {
        printf("%d    \t\t%d   \t\t%d \t\t%d\n",i+1,bt[i],tt[i],wt[i]);
    }
    printf("Total turnaround:%d\tTotal Waiting time:%d\t",total_tat,total_wt);

    printf("Avg TT:%f\t Avg WT:%f\t",(float)total_tat/n,(float)total_wt/n);

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

```
    {
        printf("p[%d]: ",i+1);
        scanf("%d",&bt[i]);
    }
    fcfs(bt,n);
    return 0;
}
```

Sjf

```
#include <stdio.h>
void sjf(int bt[],int n)
{
    int total_tat=0;
    int total_wt=0;
    int wt[n],tt[n];
    int strt_t=0;
    int temp;
    for(int i=0;i<n;i++)
    {
        for(int j=i+1;j<n;j++)
        {
            if(bt[j]< bt[i])
            {
                temp=bt[j];
                bt[j]=bt[i];
                bt[i]=temp;


            }
        }
    }

    printf("Process ID\tBurst Time\tTurnaround time\twaiting Time\n");
    for(int i=0;i<n;i++) //for all processes
    {
```

```
        wt[i]=0; //for 1st process
        tt[i]=0;
        for(int j=0;j<i;j++)
        {
            wt[i]=wt[i]+bt[j];
        }
        tt[i]=wt[i]+bt[i];
        printf("%d    \t\t%d   \t\t%d \t\t%d\n",i+1,bt[i],tt[i],wt[i]);
        total_tat+=tt[i];
        total_wt+=wt[i];

    }

    printf("Total turnaround:%d\tTotal Waiting time:%d\t",total_tat,total_wt);

    printf("Avg TT:%f\t Avg WT:%f\t",(float)total_tat/n,(float)total_wt/n);

}
int main()
{
    int n;
    printf("Enter number of processes:\n");
    scanf("%d",&n);
    int bt[n];
    printf("Enter burst time:\n");
    for(int i=0;i<n;i++)
    {
        printf("p[%d]: ",i+1);
        scanf("%d",&bt[i]);
    }
    sjf(bt,n);
    return 0;
}
```


Roundrobin

```
# include <stdio.h>
# define time 4
```

```
int n,
total_waiting_time=0,total_turnaround_time=0,current_time=0,completed_p
rocess=0,time_executed=0;

typedef struct {
int pid;
int remaining_time;
int burst_time;
int arrival_time;
int turnaround_time;
int waiting_time;
}Process ;

Process process[10];

void read()
{
   printf("enter number of processes ");
   scanf("%d",&n);

   for (int i=0;i<n;i++)
   {
      printf("Enter arrival time for process %d ",(i+1));
      scanf("%d",&process[i].arrival_time);
      printf("Enter burst time for process %d ",(i+1));
      scanf("%d",&process[i].burst_time);
      process[i].remaining_time=process[i].burst_time;
      process[i].pid=(i+1);
   }
}

void round_robin()
{
   while(completed_process<n)
   {
      for (int i=0;i<n;i++)
      {
```

```
        if(process[i].remaining_time>0)
        {
            if(process[i].remaining_time>time)
            {
                time_executed=time;
            }
            else{
                time_executed=process[i].remaining_time;
            }
            process[i].remaining_time-=time_executed;

            current_time+=time_executed;

process[i].waiting_time+=(current_time-process[i].arrival_time-time_execut
ed);

            printf("%d\n",current_time);
            if (process[i].remaining_time == 0)
                {
                process[i].turnaround_time = current_time -
process[i].arrival_time;
                completed_process++;
                }

        }

    }
  }
}


void display()
{
  printf("ID\tBurst time\tArrival time\tWaiting time\tTurnaround time\n");
  for (int i=0;i<n;i++)
  {
```

```
printf("%d\t%d\t%d\t%d\t%d\n",process[i].pid,process[i].burst_time,proces
s[i].arrival_time,process[i].waiting_time,process[i].turnaround_time);
        total_waiting_time += process[i].waiting_time;
        total_turnaround_time += process[i].turnaround_time;
    }

    printf("\nTotal waiting time is %d",total_waiting_time);
    printf("\nTotal turnaround time is %d",total_turnaround_time);
    printf("\nAverage waiting time is %f",(float)total_waiting_time/n);
    printf("\nAverage turnaround time is %f",(float)total_turnaround_time/n);
}

int main()
{
    read();
    round_robin();
    display();
    return 0;
}
//roundrobin

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

struct Process {
    int pid;
    int burst_time;
    int remaining_time;
    int waiting_time;
    int turnaround_time;
};

void calculate_waiting_time(struct Process *processes, int num_processes, int
time_quantum) {
    int current_time = 0;
    int completed_processes = 0;
```

```
    while (completed_processes < num_processes) {
        for (int i = 0; i < num_processes; i++) {
            if (processes[i].remaining_time > 0) {
                if (processes[i].remaining_time <= time_quantum) {
                    current_time += processes[i].remaining_time;
                    processes[i].turnaround_time = current_time;
                    processes[i].remaining_time = 0;
                    processes[i].waiting_time = processes[i].turnaround_time -
processes[i].burst_time;
                    completed_processes++;
                } else {
                    current_time += time_quantum;
                    processes[i].remaining_time -= time_quantum;
                }
            }
        }
    }
}

void print_table(struct Process *processes, int num_processes) {
    printf("PID\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < num_processes; i++) {
        printf("%d\t%d\t\t%d\t\t%d\n", processes[i].pid, processes[i].burst_time,
processes[i].waiting_time, processes[i].turnaround_time);
    }
}

float calculate_average_waiting_time(struct Process *processes, int
num_processes) {
    float sum = 0;
    for (int i = 0; i < num_processes; i++) {
        sum += processes[i].waiting_time;
    }
    return sum / num_processes;
}
```

```
float calculate_average_turnaround_time(struct Process *processes, int
num_processes) {
    float sum = 0;
    for (int i = 0; i < num_processes; i++) {
        sum += processes[i].turnaround_time;
    }
    return sum / num_processes;
}

int main() {
    int num_processes, time_quantum;
    printf("Enter the number of processes: ");
    scanf("%d", &num_processes);
    printf("Enter the time quantum: ");
    scanf("%d", &time_quantum);

    struct Process *processes = malloc(num_processes * sizeof(struct Process));
    if (!processes) {
        printf("Error: Unable to allocate memory\n");
        return -1;
    }

    printf("Enter the burst time for each process:\n");
    for (int i = 0; i < num_processes; i++) {
        printf("Process %d: ", i+1);
        scanf("%d", &processes[i].burst_time);
        processes[i].pid = i+1;
        processes[i].remaining_time = processes[i].burst_time;
        processes[i].waiting_time = 0;
        processes[i].turnaround_time = 0;
    }

    calculate_waiting_time(processes, num_processes, time_quantum);

    print_table(processes, num_processes);
```

```
    printf("Average Waiting Time: %.2f\n",
calculate_average_waiting_time(processes, num_processes));
    printf("Average Turnaround Time: %.2f\n",
calculate_average_turnaround_time(processes, num_processes));

    free(processes);

    return 0;
}
```

Prio

```
#include<stdio.h>
void priority(int bt[],int prio[],int n)
{
    int temp;
    int avg_tt,avg_wt;
    int wt[n],tt[n],total_wt=0,total_tt=0;

    for(int i=0;i<n;i++)
    {
        for(int j=i+1;j<n;j++)
        {
            if(prio[i]>prio[j])
            {
                temp=prio[i];
                prio[i]=prio[j];
                prio[j]=temp;

                temp=bt[i];
                bt[i]=bt[j];
                bt[j]=temp;
            }

        }

    }
    wt[0]=0;
```

```
    printf("pid\t\tbt\t\tpriority\t\ttt\t\twt\n");
    for(int i=0;i<n;i++)
    {
         wt[i]=0,tt[i]=0;

        for(int j=0;j<i;j++)
        {
            wt[i]=wt[i]+bt[j];
        }
        tt[i]=wt[i]+bt[i];
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",i+1,bt[i],prio[i],tt[i],wt[i]);
        total_tt=total_tt+tt[i];
        total_wt=total_wt+wt[i];
    }
    printf("\nAvg bt:%f\tAvg wt:%f",(float)total_tt/n,(float)total_wt/n);
}
int main()
{
    int n;
    printf("number of process:\n");
    scanf("%d",&n);
    int bt[n];
    int prio[n];
    printf("Enter processes and priority\n");
    for(int i=0;i<n;i++)
    {
    printf("p[%d]\n",i+1);
    scanf("%d%d",&bt[i],&prio[i]);
    }
    priority(bt,prio,n);
    return 0;
}


//bestfit
#include <stdio.h>
#include<limits.h>
int main()
```

```
{   int n,m,best_fit;
    printf("Enter number of memo blocks:");
    scanf("%d",&n);
    int memory[n];
    for(int i=0;i<n;i++)
    {
        printf("Memory[%d]: ",i);
        scanf("%d",&memory[i]);
    }
    printf("Enter number of processes:");
    scanf("%d",&m);
    int process[m],allocation[m];
    for(int i=0;i<m;i++)
    {
        printf("process[%d]: ",i);
        scanf("%d",&process[i]);
        allocation[i]=-1;
    }

    for(int i=0;i<m;i++)
    {   best_fit=INT_MAX;
        for(int j=0;j<n;j++)
        {
            if(memory[j]>= process[i] && memory[j]-process[i] < best_fit)
            {
                allocation[i]=j;
                best_fit=memory[j]-process[i];
            }
        }
        if(allocation[i]!=-1)
        {
            memory[allocation[i]]=memory[allocation[i]]-process[i];
        }
    }
    printf("\nprocess\t\tallocated blck no\n");
    for(int i=0;i<m;i++)
    {   printf("%d\t",process[i]);
                if(allocation[i]!=-1)
                {
                    printf("%d\n",allocation[i]+1);
                }
                else
                {
                    printf("not allocated\n");
```

```
            }
    }
    return 0;
}
```

//worst fit

Same as best just int_max replace int_min anf if ke and mein > best_fit

```
#include <stdio.h>
#include<limits.h>
int main()
{   int n,m,best_fit;
    printf("Enter number of memo blocks:");
    scanf("%d",&n);
    int memory[n];
    for(int i=0;i<n;i++)
    {
        printf("Memory[%d]: ",i);
        scanf("%d",&memory[i]);
    }
    printf("Enter number of processes:");
    scanf("%d",&m);
    int process[m],allocation[m];
    for(int i=0;i<m;i++)
    {
        printf("process[%d]: ",i);
        scanf("%d",&process[i]);
        allocation[i]=-1;
    }

    for(int i=0;i<m;i++)
    {   best_fit=INT_MIN;
        for(int j=0;j<n;j++)
        {
            if(memory[j]>= process[i] && memory[j]-process[i] > best_fit)
            {
                allocation[i]=j;
                best_fit=memory[j]-process[i];
            }
        }
        if(allocation[i]!=-1)
        {
            memory[allocation[i]]=memory[allocation[i]]-process[i];
        }
```

```
    }
    printf("\nprocess\t\tallocated blck no\n");
    for(int i=0;i<m;i++)
    {   printf("%d\t",process[i]);
            if(allocation[i]!=-1)
            {
                printf("%d\n",allocation[i]+1);
            }
            else
            {
                printf("not allocated\n");
            }
    }
    return 0;
}
```

//First fit

```
#include <stdio.h>
#include<limits.h>
int main()
{   int n,m,best_fit;
    printf("Enter number of memo blocks:");
    scanf("%d",&n);
    int memory[n];
    for(int i=0;i<n;i++)
    {
        printf("Memory[%d]: ",i);
        scanf("%d",&memory[i]);
    }
    printf("Enter number of processes:");
    scanf("%d",&m);
    int process[m],allocation[m];
    for(int i=0;i<m;i++)
    {
        printf("process[%d]: ",i);
        scanf("%d",&process[i]);
        allocation[i]=-1;
    }

    for(int i=0;i<m;i++)
    {
        for(int j=0;j<n;j++)
        {
            if(memory[j]>= process[i])
```

```
            {
                allocation[i]=j;
                memory[j]=memory[j]-process[i];
                break;
            }
        }

    }
    printf("\nprocess\t\tallocated blck no\n");
    for(int i=0;i<m;i++)
    {   printf("%d\t",process[i]);
                if(allocation[i]!=-1)
                {
                    printf("%d\n",allocation[i]);
                }
                else
                {
                    printf("not allocated\n");
                }
    }
    return 0;
}
```

Fifo

```
#include <stdio.h>

 int
main ()
{

int no_req, no_blcks, hit, no_hit = 0, nxt_index = 0;

printf ("Enter no of blocks:\n");

scanf ("%d", &no_blcks);


printf ("Enter no of requests:\n");

scanf ("%d", &no_req);


int blocks[no_blcks], no_miss = 0;
```

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

blocks[i] = -1;

}
printf ("Enter ref strings:\n");

int page;

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

scanf ("%d", &page);

hit = 0;

for (int j = 0; j < no_blcks; j++)
         {

if (blocks[j] == page)
            {

hit = 1;

no_hit++;

break;

}

}

if (!hit)
         {

blocks[nxt_index] = page;

nxt_index = (nxt_index + 1) % no_blcks;

no_miss++;

}
```

```
printf ("Blocks:\n");

for (int j = 0; j < no_blcks; j++)
        {

if (blocks[j] == -1)
          {

printf ("- ");

}
          else
          {

printf ("%d ", blocks[j]);

}

}

printf ("\n");

}
```



```
double hit_ratio = (double) no_hit / no_blcks;

double miss_ratio = (double) no_miss / no_blcks;


printf ("hit ratio:\t%.2f\n", hit_ratio);

printf ("miss ratio:\t%.2f\n", miss_ratio);

return 0;

}
```

Optimal

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

```
int
main ()
{
int page;
int num_pages, num_frames, num_hits = 0, num_misses = 0;
printf ("Enter the number of pages: ");
scanf ("%d", &num_pages);
int pages[num_pages];

printf ("Enter the number of frames: ");
 scanf ("%d", &num_frames);
int frames[num_frames];

int count[num_frames];


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

   {

frames[i] = -1;


count[i] = 0;


     }
printf ("Enter the page reference string: ");


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

   {
scanf ("%d", &page);
int hit = 0;
```

```
for (int j = 0; j < num_frames; j++)
        {
if (frames[j] == page)
            {
hit = 1;
count[j] = 0;
}
           else
            {
count[j]++;
}
}
if (hit)
        {
```

```
num_hits++;

}
      else
         {

int max_count = -1;

int index = -1;


for (int j = 0; j < num_frames; j++)
             {

if (frames[j] == -1)
                    {

index = j;

break;

}
                 else if (count[j] > max_count)
```

```
                {

max_count = count[j];

index = j;

}

}


frames[index] = page;

count[index] = 0;

num_misses++;

}


printf ("Page %d: ", page);

for (int j = 0; j < num_frames; j++)
        {
```

```
if (frames[j] == -1)
            {
printf (" ");
}
          else
            {
printf ("%d", frames[j]);
}
printf (" ");
}
printf ("\n");
}

printf ("Hit ratio: %.2f\n", (float) num_hits / num_pages);
```

```
printf ("Miss ratio: %.2f\n", (float) num_misses / num_pages);


return 0;


}
```

//LRU

```
#include <stdio.h>

#define MAX_PAGES 100

int main() {
    int num_pages, num_frames, page_faults = 0, hit_count = 0;

    // Get the number of pages and frames from the user
    printf("Enter the number of pages: ");
    scanf("%d", &num_pages);
    printf("\nEnter the number of frames: ");
    scanf("%d", &num_frames);

    // Declare arrays to hold the page references, frame buffer, and usage counts
    int reference_string[MAX_PAGES];
    int frame_buffer[num_frames];
    int frame_usage[num_frames];

    // Get the page reference string from the user
    printf("\nEnter the reference string: ");
    for (int i = 0; i < num_pages; i++) {
        scanf("%d", &reference_string[i]);
    }

    // Initialize the frame buffer and usage counts to -1 and 0, respectively
    for (int i = 0; i < num_frames; i++) {
```

```
    frame_buffer[i] = -1;
    frame_usage[i] = 0;
}

// Loop through the page reference string
for (int i = 0; i < num_pages; i++) {
    int page = reference_string[i];
    int page_fault = 1;

    // Check if the page is already in one of the frames
    for (int j = 0; j < num_frames; j++) {
        if (frame_buffer[j] == page) {
            hit_count++;
            page_fault = 0;
            frame_usage[j] = i + 1; // Update the usage count for the frame
            break;
        }
    }

    // If the page is not already in a frame, find the oldest frame and replace it
    if (page_fault) {
        page_faults++;
        int oldest_frame = 0;
        for (int j = 1; j < num_frames; j++) {
            if (frame_usage[j] < frame_usage[oldest_frame]) {
                oldest_frame = j;
            }
        }
        frame_buffer[oldest_frame] = page;
        frame_usage[oldest_frame] = i + 1; // Update the usage count for the frame
    }

    // Print out the current state of the frame buffer after each page reference
    printf("Blocks: ");
    for (int j = 0; j < num_frames; j++) {
        if (frame_buffer[j] == -1) {
            printf("- ");
        } else {
            printf("%d ", frame_buffer[j]);
        }
    }
    printf("\n");
}
```

```
    // Calculate and print out the hit and miss ratios
    float hit_ratio = (float)hit_count / num_pages;
    float miss_ratio = (float)page_faults / num_pages;
    printf("\nHit ratio: %.2f\n", hit_ratio);
    printf("\nMiss ratio: %.2f\n", miss_ratio);

    return 0;
}
```

//FCFS

Disk scheduling

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
int main()
{
    int n,curr;
    printf("Enter number of requests:\n");
    scanf("%d",&n);
    printf("Enter requests:\n");
    int r[n];
    for(int i=0;i<n;i++)
    {
        scanf("%d",&r[i]);
    }
    printf("Enter initial head position:\n");
    scanf("%d",&curr);

    int moment=0;
    moment=moment+abs(curr-r[0]);
    printf("\nqueue will be as follows:%d--->%d--->",curr,r[0]);
    for(int i=1;i<n;i++)
    {   printf("%d--->",r[i]);
        moment=moment+abs(r[i]-r[i-1]);
    }
    printf("\nTotal moment:\t%d",moment);
    return 0;
}
```

//sstf

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
int main()
```

```
{
    int n,curr,count=0;
    printf("Enter number of requests:\n");
    scanf("%d",&n);
    printf("Enter requests:\n");
    int r[n];
    for(int i=0;i<n;i++)
    {
        scanf("%d",&r[i]);
    }
    printf("Enter initial head position:\n");
    scanf("%d",&curr);

    int moment=0,ind;
   printf("Sequence:\n");
    while(count!=n)
    {
        int min=1000;
        for(int i=0;i<n;i++)
        {
            int diff=abs(r[i]-curr);
            if(min>diff)
            {
                min=diff;
                 ind=i;
            }

        }
         printf("%d-->",r[ind]);
        moment=moment+min;
        curr=r[ind];
        r[ind]=1000;
        count++;
    }
    printf("\nTotal moment:\t%d",moment);
    return 0;
}
```

//**scan**

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
int queue[50], queue1[20], queue2[20], current, max, temp1 = 0, temp2 =
  0, temp, n, movement = 0;
```

```
int
main ()
{

int i, j;

printf ("Enter number of requests ");

scanf ("%d", &n);


printf ("Enter current position ");

scanf ("%d", &current);


printf ("Enter max range of disk");

scanf ("%d", &max);


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

   {

printf ("Enter request %d ", (i + 1));

scanf ("%d", &temp);


if (temp > current)

        {

queue1[temp1] = temp;
```

```
temp1++;


}
      else
          {
queue2[temp2] = temp;
temp2++;
}
}

for (i = 0; i < temp1 - 1; i++)
    {
for (int j = 0; j < temp1 - i - 1; j++)
          {
if (queue1[j] > queue1[j + 1])
              {
temp = queue1[j];
queue1[j] = queue1[j + 1];
queue1[j + 1] = temp;
}
```

```
}
}

for (i = 0; i < temp2 - 1; i++)
   {
for (int j = 0; j < temp2 - i - 1; j++)
        {
if (queue2[j] < queue2[j + 1])
            {
temp = queue2[j];
queue2[j] = queue2[j + 1];
queue2[j + 1] = temp;
}
}
}

for (i = 1, j = 0; j < temp1; i++, j++)
   {
queue[i] = queue1[j];
```

```
}

printf ("\n%d\n", i);


queue[i] = max;


for (i = temp1 + 2, j = 0; j < temp2; i++, j++)

   {

queue[i] = queue2[j];

}

queue[0] = current;

queue[i] = 0;

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


   {

movement = movement + abs (queue[j] - queue[j + 1]);

printf ("%d ---> %d : %d\n", queue[j], queue[j + 1], movement);

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

return 0;

}
```

//**cscan**

```
# include <stdio.h>
int
queue[50],queue1[20],queue2[20],current,max,temp1=0,temp2=0,temp,n,m
ovement;
int main()
{
   int i,j;
   printf("Enter number of requests ");
   scanf("%d",&n);

   printf("Enter current position ");
   scanf("%d",&current);

   printf("Enter max range of disk");
   scanf("%d",&max);

   for(i=0;i<n;i++)
   {
      printf("Enter request %d ",(i+1));
      scanf("%d",&temp);

      if(temp>current)
      {
         queue1[temp1]=temp;
         temp1++;

      }
      else
      {
         queue2[temp2]=temp;
         temp2++;
      }
   }
```

```
for( i=0;i<temp1-1;i++)
{
   for(int j=0;j<temp1-i-1;j++)
   {
      if(queue1[j]>queue1[j+1])
      {
         temp=queue1[j];
         queue1[j]=queue1[j+1];
         queue1[j+1]=temp;
      }
   }
}

for( i=0;i<temp2-1;i++)
{
   for(int j=0;j<temp2-i-1;j++)
   {
      if(queue2[j]>queue2[j+1])
      {
         temp=queue2[j];
         queue2[j]=queue2[j+1];
         queue2[j+1]=temp;
      }
   }
}

for( i=1,j=0;j<temp1;i++,j++)
{
   queue[i]=queue1[j];
}
   printf("\n%d\n",i);

queue[i]=max;

 for( i=temp1+2,j=0;j<temp2;i++,j++)
```

```
    {
        queue[i]=queue2[j];
    }
    queue[0]=current;
    queue[i]=0;
    for(int j=0;j<i+1;j++)
    {
        movement=movement+abs(queue[j]-queue[j+1]);
        printf("%d ---> %d : %d\n",queue[j],queue[j+1],movement);
    }
    return 0;
}
```

//Look

```
# include <stdio.h>
# include <stdlib.h>
# include <math.h>
int
i,j,temp,temp1,temp2,queue1[20],queue2[20],queue[20],head,n,movement
=0;
int main ()
{
    printf("Enter number of requests");
    scanf("%d",&n);

    printf("Enter head");
    scanf("%d",&head);

    for(i=0; i<n ; i++)
    {
        printf("Enter size of request %d",(i+1));
        scanf("%d",&temp);
        if(temp>head)
        {
            queue1[temp1]=temp;
```

```
        temp1++;
    }
    else
    {
        queue2[temp2]=temp;
        temp2++;
    }
}

    for(i=0;i<temp1-1;i++)
    {
        for(j=0;j<temp1-i-1;j++)
        {
            if(queue1[j]>queue1[j+1])
            {
                temp=queue1[j];
                queue1[j]=queue1[j+1];
                queue1[j+1]=temp;
            }
        }
    }

    for(i=0;i<temp2-1;i++)
    {
        for(j=0;j<temp2-i-1;j++)
        {
            if(queue2[j]<queue2[j+1])
            {
                temp=queue2[j];
                queue2[j]=queue2[j+1];
                queue2[j+1]=temp;
            }
        }
    }
queue[0]=head;
```

```
    for(i=1,j=0;j<temp1;i++,j++)
        queue[i]=queue1[j];

    for(i=temp1+1,j=0;j<temp2;i++,j++)
        queue[i]=queue2[j];

    for(j=0;j<i-1;j++)
    {
        movement=movement+abs(queue[j]-queue[j+1]);
        printf("%d ---> %d : %d\n",queue[j],queue[j+1],movement);
    }
    printf("\nTotal head movement is %d",movement);
}
```

Clook

```
# include <stdio.h>
# include <stdlib.h>
# include <math.h>
int
i,j,temp,temp1,temp2,queue1[20],queue2[20],queue[20],head,n,movement
=0;
int main ()
{
    printf("Enter number of requests");
    scanf("%d",&n);

    printf("Enter head");
    scanf("%d",&head);

    for(i=0; i<n ; i++)
    {
        printf("Enter size of request %d",(i+1));
        scanf("%d",&temp);
        if(temp>head)
        {
```

```
        queue1[temp1]=temp;
        temp1++;
    }
    else
    {
        queue2[temp2]=temp;
        temp2++;
    }
}

    for(i=0;i<temp1-1;i++)
    {
        for(j=0;j<temp1-i-1;j++)
        {
            if(queue1[j]>queue1[j+1])
            {
                temp=queue1[j];
                queue1[j]=queue1[j+1];
                queue1[j+1]=temp;
            }
        }
    }

    for(i=0;i<temp2-1;i++)
    {
        for(j=0;j<temp2-i-1;j++)
        {
            if(queue2[j]>queue2[j+1])
            {
                temp=queue2[j];
                queue2[j]=queue2[j+1];
                queue2[j+1]=temp;
            }
        }
    }
```

```
    queue[0]=head;
    for(i=1,j=0;j<temp1;i++,j++)
        queue[i]=queue1[j];

    for(i=temp1+1,j=0;j<temp2;i++,j++)
        queue[i]=queue2[j];

    for(j=0;j<i-1;j++)
    {
        movement=movement+abs(queue[j]-queue[j+1]);
        printf("%d ---> %d : %d\n",queue[j],queue[j+1],movement);
    }
    printf("\nTotal head movement is %d",movement);
}
```

//mvt

```
#include<stdio.h>
int main()
{
   int ms,m[100],temp,i,n=0;
   char ch='y';
   printf("Enter total memory size:\n");
   scanf("%d",&ms);
   temp=ms;
   for(i=0;ch=='y';i++,n++)
   {
      printf("Enter memory of p[%d]\n",i+1);
      scanf("%d",&m[i]);

      if(m[i]<=temp)
      {
         printf("Memory is allocated to p[%d]\n",i+1);
         temp=temp-m[i];
      }
      else{
         printf("Memory is full\n");
         break;
      }

      printf("Do you want to con y or n?");
```

```
        scanf(" %c",&ch);
    }
    printf("\nTotal memo allocation\n");
    printf("\nprocss\t\tmemo allcted\n");
    for(i=0;i<n;i++)
    {
        printf("%d\t\t%d\n",i+1,m[i]);
    }
    printf("total memo allocated:\t%d",ms-temp);
    printf("\nFragmentation:\t%d",temp);
}
```

//MFT

```
#include<stdio.h>
#include<conio.h>
int
main ()
{

int ms, bs, nob, ef, n, mp[10], tif = 0;

int i, p = 0;

printf ("Enter the total memory available (in Bytes) -- ");

scanf ("%d", &ms);

printf ("Enter the block size (in Bytes) -- ");

scanf ("%d", &bs);

nob = ms / bs;

ef = ms - nob * bs;

printf ("\nEnter the number of processes -- ");

scanf ("%d", &n);

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

```
   {
printf ("Enter memory required for process %d (in Bytes)-- ", i + 1);
scanf ("%d", &mp[i]);
}
printf ("\nNo. of Blocks available in memory -- %d", nob);
printf
   ("\n\nPROCESS\tMEMORY REQUIRED\t ALLOCATED\tINTERNAL
FRAGMENTATION");
for (i = 0; i < n && p < nob; i++)
   {
printf ("\n %d\t\t%d", i + 1, mp[i]);
if (mp[i] > bs)
printf ("\t\tNO\t\t---");
     else
         {
printf ("\t\tYES\t%d", bs - mp[i]);
tif = tif + bs - mp[i];
p++;
}
}
if (i < n)
```

```
printf ("\nMemory is Full, Remaining Processes cannot be accomodated");

printf ("\n\nTotal Internal Fragmentation is %d", tif);

printf ("\nTotal External Fragmentation is %d", ef);

}
```

Bankers

```
#include <stdio.h>

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

    int allocation[n][m]; // Allocation matrix
    int max[n][m]; // Maximum matrix
    int available[m]; // Available vector
    int need[n][m]; // Need matrix
    int work[m]; // Work vector
    int finish[n]; // Finish vector
    int safe_sequence[n]; // Safe sequence

    // Input allocation matrix
    printf("\nEnter the allocation matrix:\n");
    for (int i = 0; i < n; i++)
    {
        printf("Process %d: ", i + 1);
        for (int j = 0; j < m; j++)
        {
            scanf("%d", &allocation[i][j]);
        }
    }

    // Input maximum matrix
```

```
printf("\nEnter the maximum matrix:\n");
for (int i = 0; i < n; i++)
{
    printf("Process %d: ", i + 1);
    for (int j = 0; j < m; j++)
    {
        scanf("%d", &max[i][j]);
    }
}

// Input available vector
printf("\nEnter the available vector:\n");
for (int i = 0; i < m; i++)
{
    printf("Resource %d: ", i + 1);
    scanf("%d", &available[i]);
}

// Initialize need matrix and finish vector
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < m; j++)
    {
        need[i][j] = max[i][j] - allocation[i][j];
    }
    finish[i] = 0;
}

// Initialize work vector
for (int i = 0; i < m; i++)
{
    work[i] = available[i];
}

// Calculate safe sequence
int count = 0; // Counter to keep track of number of processes executed
while (count < n)
{
    int found = 0; // Flag to indicate if a safe process is found
    for (int i = 0; i < n; i++)
```

```
    {
        if (finish[i] == 0)
        {
            int j;
            for (j = 0; j < m; j++)
            {
                if (need[i][j] > work[j])
                {
                    break;
                }
            }
            if (j == m)
            {
                // Process i can safely execute
                for (int k = 0; k < m; k++)
                {
                    work[k] += allocation[i][k];
                }
                safe_sequence[count] = i;
                finish[i] = 1;
                count++;
                found = 1;
            }
        }
    }
    if (found == 0)
    {
        printf("\nSystem is not in safe state!\n");
        break;
    }
}

if (count == n)
{
    printf("\nSystem is in safe state.\n");
    printf("Safe sequence: ");
    for (int i = 0; i < n; i++)
    {
        printf("%d ", safe_sequence[i] + 1);
    }
```

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

    return 0;
}
```