- Write a C program to add, subtract & multiply the matrices

```
#include <stdio.h>
#include <stdlib.h>
#define max 5
int p, q, n, m, i, j;
int a[max][max], b[max][max], c[max][max];
void add()
{  if (p==m && q==n)
   {  for(i=0; i<p; i++)
      {  for(j=0; j<q; j++)
         {  c[i][j] = a[i][j] + b[i][j];
         }
      }
      printf("matrix addition of given 2 matrices:\n");
   for(i=0; i<p; i++)
   {  for(j=0; j<q; j++)
      {  printf("%d", c[i][j]);
      } printf("\n");
   }
   } else
   {  printf("addition not possible"); }
}
}
void mul()
{  if (q==m)
   {  for(i=0; i<p; i++)
```

```
    for (j=0; j<n; j++)
        c[i][j]=0;
        for (int k=0; k<q; k++)
            c[i][j]+=a[i][k]*b[k][j];
    }
}
printf("multiplication matrix is \n");
for(i=0; i<p; i++)
{
    for(j=0; j<n; j++)
        printf("%d", c[i][j]);
    printf("\n");
}
else
{
    printf("matrix multiplication not possible");
}
}
int main()
{
    int ch;
    printf("enter size of 1st matrix:");
    scanf("%d %d", &p, &q);
    printf("enter elements:\n");
    for(i=0; i<p; i++)
    {
        for(j=0; j<q; j++)
            scanf("%d", &a[i][j]);
    }
```

```
printf("enter size of 2nd matrix:");
scanf("%d, %d", &m, &n);
printf("enter elements");
for(i=0; i<m; i++)
{   for(j=0; j<n; j++)
    {  scanf("%d", &b[i][j]);
    }
}
printf("1.add 2.sub 3.mul 4.exit");
while(1)
{   printf("enter choice:");
    scanf("%d", &ch);
    switch(ch)
{   case 1: add();
    break;
    case 2: sub();
    break;
    case 3: mul();
    break;
    case 4: exit(0);
    default: printf("wrong choice");
    }
}
}
```

Sub
8/5/24

- Write a C code to simulate the following non pre-emptive CPU scheduling algorithm to find turnaround time using i) FCFS ii) SJF

```
#include <stdio.h>
int n, i, j, pos, temp, choice, total = 0;
Int Burst_time[20], Arrival_time[20],
Waiting_time[20], Turn_around_time[20],
process[20];
float avg_Turn_around_time = 0,
      avg_turnaround_time = 0;
int current_time = 0;
void FCFS(){
    Int total_waiting_time = 0,
    total_turnaround_time = 0;
    int current_time = 0;

for(i = 0; i < n - 1; i++){
  for(j = i + 1; j < n; j++){
  if (Arrival_time[i] > Arrival_time[j]){
      temp = Arrival_time[i];
      Arrival_time[i] = Arival_time[j];
      Arrival_time[j] = temp;

temp = Burst_time[i];
Burst_time[j] = Burst_time[j];
```

```
Burst_time[j] = temp;

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

waiting_time[0] = 0;
current_time = Arrival_time[0] + Burst_time[0];

for(i=1; i<n; i++){
    if(current_time < Arrival_time[i]){
    current_time = Arrival_time[i];
}

waiting_time[i] = current_time - Arrival_time[i];
current_time += Burst_time[i];
total_waiting_time += waiting_time[i];
}

printf("\n process \t Arrival Time \t Burst Time
    \t waiting Time \t Turnaround Time");

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

```
Turn_around_time[i] = Burst_time[i] +
                      waiting_time[i];
total_turnaround_time += Turn_around_time[i];
printf("\nP[%d] \t\t %d \t\t %d \t\t %d \t\t %d",
    process[i], Arrival_time[i], Burst_time[i],
    turn_around_time[i]);
}

avg_waiting_time = (float) total_waiting_time / n;
avg_turn_around_time = (float) total_turnaround_time
                       / n;
printf("\n Average waiting Time: %.2f",
        avg_waiting_time);
printf("\n Avg-Turnaround Time: %.2f\n",
    avg_Turn_around_time);
}

void SJF() {
  int total_waiting_time = 0, total_turnaround_time
                           = 0;
  int completed = 0, current_time = 0, min_index;
    int is_completed[20] = {0};

  while (completed != n) {
      int min_burst_time = 9999;
      min_index = -1;
```

```
for (i=0; i<n; i++){
    if (Arrival_time[i] <= current_time && is_completed[i]==0) {
    if (Burst_time[i] < min_burst_time){
        min_burst_time = Burst_time[i];
        min_index = i;
    }

    if (Burst_time[i] == min_burst_time) {
    if (Arrival_time[i] < Arrival_time
                        [min_index] ){
        min_burst_time = Burst_time[i];
        min_index = i;
    }
    }
}
}

if (min_index != -1){
    waiting_time[min_index] = current_time -
                Arrival_time[min_index];
    current_time += Burst_time[min_index];
Turn_around_time[min_index] = current_time -
        Arrival_time[min_index];
    total_waiting_time += waiting_time[min_
                        index];
total_turnaround_time += Turn_around_time
                        [min_index];
```

```
        is_completed[min_index] = 1;
        completed++;
    }
    else
        current_time++;
    }
}
printf("\nProcess\t\tArrival Time\tBurst Time\t
Waiting Time\tTurnaround Time");
for(i=0; i<n; i++)
{ printf("\nP[%d]\t\t%d\t\t%d\t\t%d\t\t%d",
process[i], arrival_time[i], burst_time[i], waiting_time[i], Turn_around_time[i]);
avg_waiting_time = (float) total_waiting_time/n;
avg_turnaroundtime = (float) total_turnaround_time/n;
printf("\n\nAverage Waiting Time = %.2f",
avg_waiting_Time);
Printf("\nAverage Turnaround Time = %.2f\n",
avg_Turn_around_time);
}
int main()
{ printf("Enter total process:");
scanf("%d", &n);
printf("\n Enter arrival time & burst
Time:\n");
```

```
for(i=0; i<n; i++)
{
    printf("P[%d] Arrival Time:", i+1);
    scanf("%d", &Arrival_time[i]);
    printf("P[%d] Burst Time:", i+1);
    scanf("%d", &Burst_Time[i]);
    process[i] = i+1;
}

while(1)
{
    printf(1.FCFS, 2. SJF);
    printf("Enter ur choice:");
    scanf("%d", &choice);
    switch(choice)
    {
        case 1: FCFS();
                break;
        case 2: SJF();
                break;
        default: printf("Invalid input");
    }
}

return 0;
}
```

Output Enter total process 4

P[1] arrival: 0 P[1] burst: 3

P[2] arrival: 1 P[2] burst: 6

P[3] arrival: 4 P[3] burst: 4

P[4] arrival: 6 P[4] burst: 2

main menu

1) FCFS
2) SJF

enter your choice: 1

Process	arrival time	Burst time	Waiting time	Turnaround time
P[1]	0	3	0	3
P[2]	1	6	2	8
P[3]	4	4	5	9
P[4]	6	2	7	9

avg WT = 3.5
ATAT = 7.25

main menu

1) FCFS
2) SJF

enter your choice: 2

Process	arrival time	Burst time	WT	TAT
P[1]	0	3	0	3
P[2]	1	6	2	8
P[3]	4	4	7	11
P[4]	6	2	3	5

AWT = 3
ATAT = 6.75

Sneha
15/5/24

- Write a C program to simulate the following CPU scheduling algorithm to find turn around time & WT
a) Priority (non pre emptive)

→
```
#include <stdio.h>
#include <stdlib.h>
struct process
{ int process_id;
  int burst time;
  int priority;
  int WT
  int TAT;
};
void find_avg_time (struct process [], int);
void priority_scheduling (struct process [], int);
int main()
{ int n, i;
  struct process proc[10];
  printf ("enter no of process:");
  scanf ("%d", &n);
  for (i=0; i<n; i++)
  { printf ("enter process id:");
    scanf ("%d", &proc[i].process_id);
    printf ("enter burst time:");
    scanf ("%d", &proc[i].burst_time);
    printf ("enter priority:");
    scanf ("%d", &proc[i].priority);
```

```
    priority_scheduling(proc, n);
    return 0;
}
void findWT(struct process proc[], int n, int wt[])
{
    int i;
    wt[0] = 0;
    for (i = 1; i < n; i++)
    {
        wt[i] = proc[i-1].burst_time + wt[i-1];
    }
}
void find_TAT(struct process proc[], int n,
    int wt[], int tat[])
{
    int i;
    for(i=0; i<n; i++)
    {
        tat[i] = proc[i].burst_time + wt[i];
    }
}
void find_avg_time(struct process proc[], int n)
{
    int wt[10], tat[10], total_wt = 0,
    total_tat = 0, i;
    find_waiting_time(proc, n, wt);
    find_TAT(proc, n, wt, tat);
    printf("\n Process ID \t Burst time \t Priority
\t WT \t TAT");
    for(i=0; i<n; i++)
    {
        total_wt = total_wt + wt[i];
        total_Tat = total_tat + tat[i];
```

```
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", proc[i].process_id, proc[i].burst_time,
proc[i].priority, wt[i], tat[i]);
}
printf("\n\nAverage WT = %.f", (float) total_wt/n);
printf("\n\nAverage TAT = %.f\n", (float) total_tat/n);
}
void priority_scheduling(struct process proc[], int n)
{
    int i, j, pos;
    struct Process temp;
    for(i=0; i<n; i++)
    {
        pos = i;
        for(j=i+1; j<n; j++)
        {
            if(proc[j].priority < proc[pos].priority)
                pos = j;
        }
        temp = proc[i];
        proc[i] = proc[pos];
        proc[pos] = temp;
    }
    find_avg_time(proc, n);
```

Output:

Process	AT	CPU Time	Priority	WT	TAT
P_1	0	4	2	0	4
P_2	1	3	3	11	14
P_3	2	1	4	9	10
P_4	3	5	5	1	6
P_5	4	2	5	5	7

b) Round Robin (non-preptive)

```
#include <stdio.h>
#include <stdbool.h>
int tAt (int process[], int n, int bt[], int
wt[], int tat[])
{   for (int i=0; i<n; i++)
        tat[i] = bt[i]+wt[i]
        return;
}
int WT (int processes[], int n, int bt[],
int wt[], int quantum)
{
    int rem_bt[n];
    for (int i=0 ; i<n; i++)
      rem_bt[i] = bt[i];
      int t = 0;
    while (1)
{
    bool done = true;
```

```
for (int i=0; i<n; i++)
{
    if (rem_bt[i] >0)
    {
        done = false;
        if (rem_bt[i] > quantum)
        {
            t += quantum;
            rem_bt[i] -= quantum;
        }
        else
        {
            t = t + rem_bt[i];
            wt[i] = t - bt[i];
            rem_bt[i] = 0;
        }
    }
}
if (done == true)
    break;
}
return 1;
}
find avgtime (int processes[], int n, int
bt[], int quantum)
{   int wt[n], tat[n], total_wt=0,
total_tat=0;
    WT (processes, n, bt, wt, quantum);
    printf (tat (processes, n, bt, wt, tat));
```

```
printf("Process \t\t Burst Time \t waiting
time \t turn around time");
    for (int i=0; i<n; i++)
{
    total wt = total_wt + wt[i];
    total_tat = total_tat + tat[i];
    printf("\n\t %d \t %d \t %d \t %d",
i+1, bt[i], wt[i], tat[i]);
}
    printf("average waiting time = %f",
    (float) total_wt / (float) n);
    printf("\n average turnaround time
= %f", (float) total_tat / (float) n);
    return 1;
}

int main()
{
    int n, processes[n], burst_time[n],
quantum;
    printf("Enter quantum time:");
    scanf("%d", &quantum);
    int i=0
    for(i=0; i<n; i++)
{
    printf("enter the process:");
    scanf("%d", &process[i]);
```

```
        printf("Enter the burst time");
        scanf("%d", &burst_time[i]);
    }
    findavgtime(processes, n, burst_time, quantum);
    return 0;
}
```

Output enter no of process = 3
enter Quantum time = 2
enter process : 1 2 3
enter Burst time 5 7 3

P	BT	WT	TAT
1	5	7	12
2	7	8	15
3	3	8	11

avg wt = 7.667
avg TAT = 12.667

- Rate Monotonic and Earliest Deadline first

Code

→
```
#include<stdlib.h>
#include<stdio.h>
#include<math.h>
#include<stdbool.h>
#define max 10

typedef struct
{
    int id;
    int burst_time;
    float priority;
} Task;

int num;
int exec_time[max], period[max], remain_time
[max], deadline[max], remain_deadline
[max];

void get_process_info(int selected_algo)
{
    printf("Enter total no of process(max:%d)
, max_process);
    scanf("%d", &num);
    if(num<1)
        exit(0);
}
```

```
for(int i=0; i<num; i++)
{
    printf("Process %d \n", i+1);
    printf("Execution time :");
    scanf("%d", &execut_time[i]);
    remain_time[i] = execut_time[i];
    if(selected_algo == 2)
    {
        printf("deadline :");
        scanf("%d", &deadline[i]);
    }
    else
    {
        printf("Period :");
        scanf("%d", &period[i]);
    }
}
}

int max(int a, int b, int c)
{
    int max;
    if(a>=b && a>=c)
        max=a;
    else if(b>=a && b>=c)
        max=b;
    else if(c>=a && c>=b)
        max=c;
    return max;
}

int get_obsrvat_time(int selected_algo)
{
```

```
    if (selected_algo == 1)
    {
        return max(period[0], period[1], period[2]);
    }
    else if (selected_algo == 2)
    {
        return max(deadline[0], deadline[1],
    deadline[2]);
}

void print_schedule(int process_list[], int
    cycles)
{
    printf("\n Scheduling:\n");
    printf("Time: ");
    for(int i=0; i<cycles; i++)
    {
        if(i<10)
            printf(" 10%d", i);
        else
            printf("%d", i);
    }
    printf("\n");
}
}
void rate_monotonic(int time)
{
    int process_list[100] = {0}, min=999,
    next_process=0;
```

```
float utilization = 0;
for(int i=0; i<num; i++)
{
    utilization += (1*execution_time[i])/
                    period[i];
}
int n = num;
int m = (float)(n* (pow(2, 1.0/n) -1));
if (utilization > m)
{
    printf("\n Given problem is not schedulable under the said scheduling algorithm");
}
for(int i=0; i<time; i++)
{
    min = 1000;
    for(int j=0; j<num; j++)
    {
        if(remain_time[j] > 0)
        {
            if (min > period[j])
            {
                min = period[j];
                next_process = j;
            }
        }
    }
    if(remain_time[next_process] > 0)
    {
        process_list[i] = next_process + 1;
        remain_time[next_process] -= 1;
    }
}
```

```
for (int k=0; k<num; k++)
{
    if ((i+1) % period[k] == 0)
    {
        remain_time[k] = exec_time[k];
        next_process = k;
    }
}
}
print_schedule(process_list, time);
}

Void earliest_deadline_first(int time)
{
    float utilization = 0;
    for(int i=0; i<num; i++)
    {
        utilization += (1* exec_time[i])/
                       deadline[i];
    }

    int n = num;
    int process[num];
    int max_deadline, current_process = 0;
int min_deadline, process_list[time];
    bool is_ready[num];
    for (int i=0; i<num; i++)
{
        is_ready[i] = true;
        process[i] = i+1;
    }
```

```
max_deadline = deadline[0];
for (int i=1; i<num; i++)
{
    if (deadline[i] > max_deadline)
        max_deadline = deadline[i];
}
for(int i=0; i<num; i++)
{
    for (int j=i+1; j<num; j++)
    {
        if (deadline[j] < deadline[i])
        {
            int temp = execut_time[j];
            execut_time[j] = execut_time[i];
            execut_time[i] = temp;
            temp = deadline[j];
            deadline[j] = deadline[i];
            deadline[i] = temp;
            temp = process[j];
            process[j] = process[i];
            process[i] = temp;
        }
    }
}
for (int i=0; i<num; i++)
{
    remain_time[i] = execut_time[i];
    remain_deadline[i] = deadline[i];
}
for (int t=0; t<time; t++)
{
    if (current_process != -1)
```

```
{   -- exect^n time(current_proc);
    process_list(t) = proc(current_proc);
}
else
    process_list(t) = 0;
for(int i=0; i<num; i++)
    { -- deadline[i]; }
    if((exect^n time(i) == 0) && is_ready[i])
{       deadline[i] += remain_deadline[i];
        is_ready[i] = false;
    }
    if((deadline[i] <= remain_deadline[i])
&& (is_ready[i] == false))
{       exect^n time[i] = remain_time[i];
        is_ready[i] = true;
    }
}
min_deadline = max_deadline;
    current_process = -1;
for(int i=0; i<num; i++)
{       if((deadline[i] <= min_deadline) && (
execution_time[i] > 0))
{       current_process = i;
        min_deadline = deadline[i];
    }
}
print_schedule(process_list, time); }
```

```
int main()
{
    int option;
    int observation_time;
    while(1)
    { printf("\n1. Rate monotonic \n 2. Earliest Deadline first \n Enter your choice:");
      scanf("%d", &option);
      switch(option)
      { case 1: get_process_info(option);
        observation_time = get_observation(option);
        rate_monotonic(observation_time);
        break;
      case 2:
        get_process_info(option);
        observation_time = get_obs_time(opt);
        earliest_deadline_first(obs_time);
        break;
      default:
        printf("\n. Invalid statement\n");
    }
}
    return 0;
}
```

AHH, AAHH ... AHH - AHH

Output:

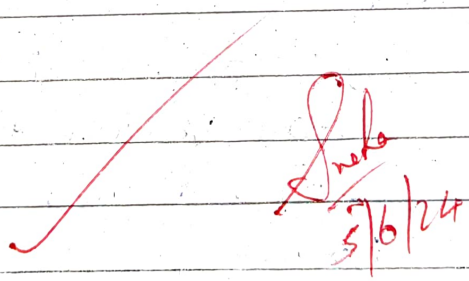
1. Rate Monotonic
2. Earliest Deadline first
3. Proportional Scheduling

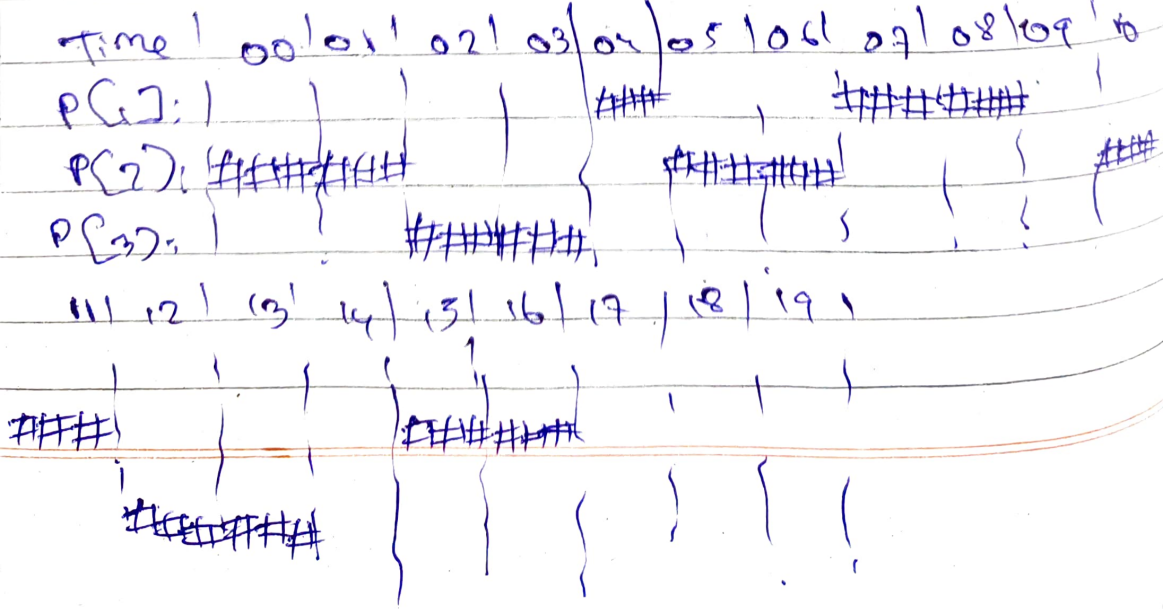
Enter your choice 2
Enter total no of process (no) : 3

Process 1:
execut time : 3
deadline : 20

Process 2:
execut time : 2
deadline : 5

Process 3:
execut time : 2
deadline : 10





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

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

int mutex=1, full=0, empty=3, x=0;

int main()
{
    int n;
    void producer();
    void consumer();
    int wait(int);
    int signal(int);
    printf("\n1. Producer\n2. Consumer\n
3. Exit");
while(1)
{
    printf("Enter 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;
}

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 ("\n Producer produces the
item %d", x);
    mutex = signal (mutex);
}
```

```
void consumer()
{
    mutex = wait(mutex);
    full = wait(full);
    empty = signal(empty);
    printf("\n Consumer consumes
item %d", x);
    x--;
    mutex = signal(mutex);
}
```

output: 1. Producer
2. Consumer
3. Exit

Enter your choice: 2
Buffer is empty
Enter your choice: 1
Producer produces the item 1
Enter your choice: 2
Consumer consumes item 1

- Write a C program to simulate the concept of dining philosophers problem

→
```
#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>
#define N 5
#define Thinking 2
#define Hungry 1
#define Eating 0
#define Left (i+4)%N
#define Right (i+1)%N

int state[N];
int phil[N] = {0,1,2,3,4};
sem_t mutex;
sem_t S[N];

void test(int i)
{
    if (state[i] == Hungry && state[Left]
!= Eating && state[Right] != Eating)
    {
        state[i] = Eating
        sleep(2);
        printf("Philosopher %d takes
fork %d and %d\n", i+1, Left+1, i+1);
        printf("Philosopher %d is Eating\n",
i+1);
```

```
    sem_post(&S[i]);
}
}
void take_fork(int i)
{
    sem_wait(&mutex);
    state[i] = Hungry;
    printf("Philosopher %d putting fork
%d and %d down\n", i+1, LEFT+1, i+1);
    printf("Philosopher %d is Thinking\n"
, i+1);
    test(Left);
    test(Right);
    sem_post(&mutex);
}

void *philosopher(void *num)
{
    while(1)
    {  int *i=num;
        sleep(1);
        take_fork(*i);
        sleep(0);
        put_fork(*i);
    }
}
```

```
int main()
{
    int i;
    pthread_t thread_id[N];
    sem_init(&mutex, 0, 1);
    for (i=0; i<N; i++)
        sem_init(&S[i], 0, 0);
    for(i=0; i<N; i++)
    {
        pthread_create(&thread_id[i],
NULL, philosopher, &phil[i]);
        printf("Philosopher %d is thinking\n",
i+1);
    }
    for(i=0; i<N; i++)
        pthread_join(thread_id[i], NULL);
}
```

Output:

```
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 5 is thinking
Philosopher 1 is hungry
Philosopher 2 is hungry
Philosopher 5 is hungry
Philosopher 5 takes fork 4 & 5
Philosopher 5 is eating
:
Philosopher 5 putting both 4 & 5
```

12/6/24

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

```
#include<stdio.h>
int main()
{
    int n, m, i, j, k;
    printf("Enter no of process:");
    scanf("%d", &n);
    printf("Enter no of resources:");
    scanf("%d", &m);
    int allocation[n][m];
    printf("enter allocation matrix:\n");
    for(i=0; i<n; i++)
    {
        for(j=0; j<m; j++)
        scanf("%d", &allocation[i][j]);
    }
int max[n][m];
    printf("enter max matrix:\n");
    for(i=0; i<n; i++)
    {
        for(j=0; j<m; j++)
            scanf("%d", &max[i][j]);
    }
int available[m];
    printf("enter the available resources:\n");
```

```
for (i=0 ; i<m ; i++)
    scanf("%d", &available[i]);
int f[n], ans[n], ind=0;
for (k=0; k<n; k++)
    f[k]=0;
int need[n][m];
for (i=0; i<n; i++)
{
    for (j=0; j<m; j++)
    {
        need[i][j] = max[i][j] - alloc[i][j];
    }
}
int y=0;
for (k=0; k<n; k++)
{
    for (i=0; i<n; i++)
    {
        if (f[i] == 0)
        {
            int flag=0
            for (j=0; j<m; j++)
            {
                if (need[i][j] > available[j])
                {
                    flag=1;
                    break;
                }
            }
            if (flag==0)
```

```
{
        ans[ind++] = i;
        for (y=0; y<m; y++)
        {
            available[y] += allocat[i][y];
        }
        f[i] = 1
    }
}
}
int flag = 1
for (i=0; i<n; i++)
{
    if(f[i]==0)
    {
        flag = 0;
        printf("The following system
is not safe\n");
        break;
    }
}
if (flag == 1)
{
    printf ("Following is the SAFE
Sequence\n");
    for (i=0; i<n-1; i++)
    {
        printf(" P%d -> ", ans[i]);
    }
```

```
    printf("P%d ", ans[n-1]);
    return 0;
}
```

Output:

```
Enter no of process: 5
Enter no of resources: 3
Enter allocation matrix
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter max matrix
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter available resources
3 3 2
Following is the SAFE sequence
P1 → P3 → P4 → P0 → P2
```

- Write a C program to stimulate deadlock detection

```
#include <stdio.h>
static int mark[20];
int i, j, np, nr;
int main()
{
    int alloc[10][10], request[10][10], avail[10], r[10], w[10];
    printf("Enter no of process:");
    scanf("%d", &np);
    printf("Enter no of resources:");
    scanf("%d", &nr);
    for(i=0; i<nr; i++)
    {
        printf("Total amount of the resource R%d", i+1);
        scanf("%d", &r[i]);
    }
printf("Enter the request matrix");
    for(i=0; i<np; i++)
    {
        for(j=0; j<nr; j++)
            scanf("%d", &request[i][j]);
    }
printf("Enter the allocation matrix:");
    for(i=0; i<np; i++)
    {
        for(j=0; j<nr; j++)
            scanf("%d", &alloc[i][j]);
```

```
{   for(j=0;j<nr;j++)
        avail[j]=r[j];
    {   for(i=0;i<np;i++)
            avail[j] -= alloc[i][j];
    }
}
for(i=0; i<np; i++)
{
    int count=0;
    {   for(j=0; j<nr; j++)
        {
            if(alloc[i][j]==0)
                count++;
            else
                break;
        }
    if(count==nr)
        mark[i]=1;
}
for(j=0; j<nr; j++)
        w[j] = avail[j];
{   for(i=0; i<np; i++)
        if(mark[i]!=1)
            if(canbeprocessed=0)
```

```
{
    for(j=0; j<nr; j++)
        if (request[i][j] <= w[j])
            can be processed = 1;
        else
        { can be processed = 0;
          break;
        }
    }
    if (can be processed)
    {
        mark[i] = 1;
        for (j=0; j<nr; j++)
            w[j] += alloc[i][j];
    }
}

int deadlock = 0;
for (i=0; i<np; i++)
    if (mark[i] != 1)
        deadlock = 1;
if (deadlock)
printf ("Deadlock detected");
else
printf ("No deadlock possible");
}
```

Output: Enter no of processes 5
Enter no of resources: 3
Total amount of Resource R_1 : 5
R_2 : 3
R_3 : 3
Enter request matrix
1 0 1
2 2 2
3 3 3
5 2 1
1 4 2
Enter allocation matrix
2 1 2
3 3 3
4 4 4
6 3 2
2 5 3
deadlock detected

Smita
3/7/24

9 Write a C program to stimulate the following contiguous memory allocation techniques

a) Worst-Fit
b) Best-Fit
c) First-Fit

→

```
#include <stdio.h>
#define max 25
void firstfit(int b[], int nb, int f[], int nf);
void worstfit(int b[], int nb, int f[], int nf);
void BestFit (int b[], int nb, int f[], int nf);

int main()
{
int b[max], f[max], nb, nf;
printf("Memory management Scheme \n");
printf("Enter no of blocks:");
scanf("%d", &nb);
printf("Enter no of files");
scanf("%d", &nf);
printf("Enter the size of blocks:\n");
for(int i=1 to ; i<=nb; i++)
{
Printf("Block %d", i);
scanf("%d", &b[i]);
}
```

```
printf ("Enter the size of the files:\n");
for (int i = 1; i <= nf; i++)
{
    printf ("File %d", i);
    scanf ("%d", &f[i]);
}
printf("\n Memory Management Scheme
- First Fit");
firstFit (b, nb, f, nf);
printf ("\n Memory Management scheme
Worst Fit);
worstFit (b, nb, f, nf);
printf ("\n Memory management scheme
- bestFit);
BestFit (b, nb, f, nf);
return 0;
}

Void FirstFit (int b[], int nb, int f[], int nf)
{
    int bf[max] = {0};
    int ff[max] = {0};
    int frag[max], i, j;
    for (i=0; i<=nf; i++)
    {
        for (j=1; j<=nb; j++)
```

```
        if (bf[j] != 1 && b[j] >= f[i])
        {
            ff[i] = j;
            bf[j] = 1;
            frag[i] = b[j] - f[i];
            break;
        }
    }
}
printf("\n File no:\t File size:\t Block no:\t
Block size:\t Fragment");
for(i=1; i<=nf; i++)
{
    printf("\n %d\t %d\t %d\t %d\t %d", i,
f[i], ff[i], b[ff[i]], frag[i]);
}
}

void worstFit (int b[], int nb, int f[], int nf)
{
    int b[max] = {0};
    int ff[max] = {0};
    int frag[max], i, j, temp, highest=0;
    for (i=1; i<=nf; i++)
    {
        for (j=1; j<=nb; j++)
```

```
        if (bf[j] != 1)
        {
            temp = b[j] - f[i];
            if (temp >= 0 && highest < temp)
            {
                ff[i] = j;
                highest = temp;
            }
        }
    }
    frag[i] = highest;
    bf[ff[i]] = 1;
    highest = 0;
}
printf ("\nFile_no:\tFile_Size:\tBlock_no:\tBlock_size:\tFragment");
for (i=1; i<=nf; i++)
{
    printf ("\n%d\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
}
}

void bestFit (int b[], int nb, int f[], int nf)
{
    int b[max] = {0};
    int ff[max] = {0};
```

```
int frag[max], i, j, temp, lowest=10000;
for (i=1; i<=nf; i++)
{
for (j=1; j<=nb; j++)
{
if (b[j] != 1)
{
temp = b[j] - f[i];
if (temp >= 0 && lowest > temp)
{
ff[i] = j;
lowest = temp;
}
}
}
frag[i] = lowest;
bf[ff[i]] = 1;
lowest = 10000;
}
printf ("\n Fileno: \t Filesize : \t Blockno: \t
Block -size: \t Fragment ");
for(i=1; i<= nf && ff[i] != 0; i++)
{
printf ("\n %d \t %d \t %d \t %d \t %d", i, f[i],
ff[i], b[ff[i]], frag[i]);
}
}
```

Output: Memory management schemes:

Enter no of blocks : 5

Enter no of files : 8

Enter the size of blocks

100 500 200 300 600

Enter the size of the files

212 417 63 124 23 89 73 13

Memory management Scheme - First Fit

File no	File size	Block no	Block size	Fragment
1	124	1	200	76
2	71	2	200	124
3	84	3	300	216
4	76	4	400	324
5	128	6	200	0

~~File no File~~

Memory management scheme - Worst Fit

File no	File size	Block no	Block size	Fragment
1	124	4	400	376
2	71	3	300	229
3	84	1	200	116
4	76	1	200	124
5	128	1	200	0

Memory Management scheme - Best Fit

File no	File size	Block no	Block size	Fragment
1	124	1	200	276
2	76	1	200	224
3	84	1	200	116
4	76	1	200	124
5	128	1	200	0

9 Write a C program to simulate page replacement algorithms

a) FIFO
b) LRU
c) optimal

```
# include<stdio.h>
int n, f, i, j, k;
int in[100];
int p[50];
int hit = 0;
int pgfaultcnt = 0;

void getData()
{
    printf("Enter length of page reference sequence: ");
    scanf("%d", &n);
    printf("\n Enter the page reference sequence: ");
    for(i=0; i<n; i++)
        scanf("%d", &in[i]);
    printf("Enter no of frames");
    scanf("%d", &f);
}
void initialize()
{
    pgfaultcnt = 0
    for(i=0; i<f; i++)
```

```
    p[i]=999;
}
int isHit (int data)
{
    hit=0;
    for (j=0; j<f; j++)
    {
        if (p[j] == data)
        {
            hit=1;
            break;
        }
    }
    return hit;
}

int getHitIndex (int data)
{
    int hitind;
    for (k=0; k<f; k++)
    {
        if (p[k] == data)
        {
            hitind = k;
            break;
        }
    }
```

```
    return hitind;
}

void displayPages()
{
    for(k=0; k<f; k++)
    {
        if(p[k] != 999)
            printf(" %d ", p[k]);
    }
}
void dispPgFaultCnt()
{
    printf("\n Total no of page faults: %d
pgfaultcnt);
}

void fifo()
{
    getdata();
    Initialize();
    for(i=0; i<n; i++)
    {
    printf("\n for %d", in[i]);
        if(search(in[i]) == -1)
        {
            for(k=0; k<f-1; k++)
```

```
            p[k] = p[k+1];
        p[k] = in[i];
        pgfaultcnt++;
        dispPages();
    }
    else
    printf("No page fault");
}
dispPgFaultCnt();
}

void optimal()
{
    initialize();
    int near[50];
    for(i=0; i<n; i++)
    {
    printf("In For %d", in[i]);
        if (isHit(in[i])==0)
    {
        for(j=0; j<f; j++)
    {
        int pg = p[j];
        int found=0;
        for(k=i; k<n; k++)
    {
        if(pg==in[k])
```

```
{
        near[j] = k;
        found = 1;
        break;
    }
    else
        found = 0;
}
if (!found)
    near[j] = 9999;
}
int max = -9999;
int repindex;
for (j=0; j<nf; j++)
{
    if (near[j] > max)
    {
        max = near[j];
        repindex = j;
    }
}
p[repindex] = in[i];
pgfaultcnt++;
dispPages();
}
else
    printf("no page fault");
}
```

```
    disp pgfault cnt();
}

void fun()
{
    initialize();
    int least[50];
    for(i=0; i<n; i++)
    {
        printf("\n For %d :", in[i]);
        if (isHit(in[i])==0)
        {
            for(j=0; j<nf; j++)
            {
                int pg = p[j];
                int found = 0;
                for(k=i-1; k>=0; k--)
                {
                    if(pg == in[k])
                    {
                        least[j] = k;
                        found = 1;
                        break;
                    }
                }
                else
                    found = 0
            }
```

```
        if (!found)
            least[j] = -9999;
    }
    int min = 9999
    int repindex;
    for (j=0; j<fr; j++)
    {
        if (least[j] < min)
        {
            min = least[j];
            repindex = j;
        }
    }
    p[repindex] = in[i];
    pgfaultcnt++;
}
dispPages();
else
    printf("no page fault");
dispPgFaultCnt();
}

int main()
{
    int choice;
```

```
while(1)
{
    printf("Page Replacement Algorithms \n 1.Enter data\n 2. FIFO\n 3. Optimal \n 4. LRU\n 5.Exit \n Enter your choice:");
    scanf("%d", &choice);
    switch(choice)
{
    case 1: getData();
            break;
    case 2: Fifo();
            break;
    case 3: optimal();
            break;
    case 4: lru();
            break;
    default : return 0;
            break;
    }
}
}
```

Output: Page replacement algorithms

1) Enter data
2) FIFO
3) Optimal
4) LRU
5) Exit

Enter length of page reference sequence: 12
Enter the page reference sequence:
1 2 3 4 1 2 5 1 2 3 4 5
Enter no of frames: 3

For 1: Page Fault occured.
Pages in Frams: 1
For 2: Pag Fault occured
Pages in Frame: 1 2
For 3: Page Fault occured
Pages in Frame: 1 2 3
For 4: Page Fault occured
Pages in frame: 2 3 4
For 1: Page: Pag Fault occured
Pages in frame: 3 4 1
For 2: Pag Fault occured
Pages in frame: 4 1 2
For 5: Pag fault occured
Pages in frame: 1 2 5
For 1: no page fault
For 2: no page fault
For 3: pages in frame: 2 5 3
Pages in frame 5 3 4
For 4: no page fault
Total: no page faults: 9

Enter your choice: 3

For 1: Pages in Frame: 1
For 2: Pages in Frame: 1 2
For 3: Pages in Frame 1 2 3
For 4: Pags in frame 1 2 4
For 1: no page fault
for 2: no Page fault
for 5: Pages in frame 1 2 5
for 1: no pag fault
for 2: no pag fault
for 3: pags in frame: 3 2 5
for 4: pags in frames: 4 2 5
for 5: page fault
Total no of page faults: 7

Enter your choice: 4
For 1: Pages in frame 1
for 2: Pages in frame 1 2
for 3: Pages in frame 1 2 3
for 4: page in frames 4 2 3
for 1: page fault occured 4 1 3
for 2: 4 1 2
for 5: 5 1 2
for 1: no page fault
for 2: no page fault
for 3: 3 1 2
for 4: 3 4 2
for 5: 3 4 5
Total no of Pags faults: 10

3/7/24

- Write a C program to simulate disk scheduling algorithms:

a) FCFS
b) SCAN
c) C-SCAN

a)

```
#include <stdio.h>
#include<stdlib.h>
int main()
{
    int RQ[100], i, n, TotalHeadMoment=0, initial;
    printf("Enter the no of Requests\n");
    scanf("%d", &n);
    printf("Enter the request sequence\n");
    for(i=0; i<n; i++)
        scanf("%d", &RQ[i]);
    printf("Enter initial head position\n");
    scanf("%d", &initial);
    for(i=0; i<n; i++)
    {
        TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
        initial = RQ[i];
    }
    printf("Total head moment is %d", TotalHeadMoment);
}
```

- Output:

Enter the no of requests : 5
Enter the request sequence:
5 98 107 45 78
Enter intial head position 45
Total head moment is 237

b)

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int RQ[100], i, j, n, TotalHeadMoment=0,
size, move;
    printf("Enter no of requests\n");
    scanf("%d", &n);
    printf("Enter the requests sequence\n");
    for(i=0; i<n; i++)
        scanf("%d", &RQ[i]);
    printf("Enter intial head position\n");
    scanf("%d", &intial);
    printf("Enter total disk size\n");
    scanf("%d", &size);
    printf("Enter the head movement
direct" for high 1 and for low 0\n");
    scanf("%d", &move);
```

```
for(i=0; i<n; i++)
    for(j=0; j<n-i-1; j++)
        if(RQ[j] > RQ[j+1])
        {
            int temp;
            temp = RQ[j];
            RQ[j] = RQ[j+1];
            RQ[j+1] = temp;
        }
}
}
int indem;
for(int i=0; i<n; i++)
{
    if(intial < RQ[i])
    {
        indem = i;
        break;
    }
}
if(move == 1)
{
    for(i=indem; i<n; i++)
    {
        TotalHeadMoment += abs(RQ[i] - intial);
        intial = RQ[i];
    }
TotalHeadMoment += abs(size - RQ[i-1] - 1);
intial = size - 1;
```

```
for (i= index-1 ; i>=0; i--)
{
    TotalHeadMoment += abs (RQ[i]-initial);
    initial=RQ[i];
}
}
else
{ for(i=index-1; i>=0; i--)
{
    TotalHeadMoment += abs (RQ[i] - initial);
    initial = RQ[i];
}
TotalHeadMoment += abs (RQ[i+1]-0);
initial =0;
for (i= index; i<n; i++)
{
    TotalHeadMoment += abs (RQ[i]-initial);
    initial=RQ[i];
}
}
printf ("Total head movement is %d", TotalHead
Moment);
}
```

Output: Enter the no of requests. 8
Enter the request sequence
98 183 37 122 14 124 65 67

Enter initial head position
53
Enter total disk size
199
Enter the head movement direction for high 1 and for low 0
1
Total head movement is 236

c)

```
# Include <stdio.h>
# include <stdlib.h>
  int main ()
{
    int RQ[100], i, j, n; Total HeadMoment = 0;
  initial, size, move;
    printf ("Enter the no of requests\n");
     scanf ("%d", &n);
     printf ("Enter the requests sequence\n");
     for (i=0; i<n; i++)
       scanf ("%d", &RQ[i]);
     printf ("Enter initial head position");
      scanf ("%d", &initial);
     printf ("Enter total disk size\n");
       scanf ("%d", &size);
      Printf ("Enter the head movement direction
  for high 1 & for low 0\n");
```

```
scanf("%d", &Move);
for (i=0; i<n; i++)
{
    for (j=0; j<n-i-1; j++)
    {
        if (RQ[j] > RQ[j+1])
        {
            int temp;
            temp = RQ[j];
            RQ[j] = RQ[j+1];
            RQ[j+1] = temp;
        }
    }
}
int index;
for (i=0; i<n; i++)
{
    if (intial < RQ[i])
    {
        index = i;
        break;
    }
}
if (Move == 1)
    for (i = index; i<n; i++)
    {
        TotalHeadMoment += abs(RQ[i] - intial);
        intial = RQ[i];
    }
```

```
TotalHeadMoment += abs(size - RQ[i-1] - 1);
TotalHeadMoment += abs(size - 1 - 0);
    initial = 0;
    for (i=0; i< index; i++)
{
        TotalHeadMovement += abs(RQ[i] - initial);
        initial = RQ[i];
    }
}
else
{
    for (i=index-1; i>=0; i--)
    {
        TotalHeadMovement += abs(RQ[i] - initial);
        initial = RQ[i];
    }
TotalHeadMoment += abs(RQ[i+1] - 0);
TotalHeadMoment += abs(size - 1 - 0);
    initial = size - 1;
    for (i=n-1; i>=index; i--)
{
        TotalHeadMovement += abs(RQ[i] - initial);
        initial = RQ[i];
    }
}
printf("Total head movement is %d",
    TotalHeadMovement);
}
```

output: Enter no of requests 7
Enter request sequence
98 183 37 122 14 124 65
Enter initial head position
53
Enter total disk size 200
Enter the head movement direction for high 1 and for low 0 : 1
Total head movement is 236

10/9/24