INDEX

1BM22CS018

NAME: Adityan IV. STD: IV SEC: A ROLL NO: 18

S.No.	Date	Title	Page No.	Teacher's Sign/ Remarks
01	15/5/24	TO find turnaround time and waiting tion for FCFS and SJF	1 - 10	15/5/24
02.	22/5/24	a) Priority (Preemptive & non-Preemptive b) Round-Robin	7	12/6/24
03	5/6/24	a) Rate monotonic b) Earliest deadline first.	7	12/6/24
04	12/6/24	a) Producer consumer b) Philosopher dining.	10	12/6/24
05	19/6/24	i) Bankers algorithm (deadlock avoidance) ii) Deadlock detection.	10	19/6/24
6.	26/6/24	> Worst fit > Fifo	9	
7)	10/7/24	Disk Scheduling	10	10/7/24

Bafna Gold
Date: Page:

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

FCFS
SJF

```
#include<stdio.h>
int n, i, j, Pos, temp, choice, Burst_time[20], waiting_time[20], Turn_around_time[20], process[20], total=0;
float avg_Turn_around_time=0, avg_waiting_time=0, avg_Arrival_time=0;

int FCFS();
{
  Waiting_time[0]=0;
  for (i=1; i<n; i++)
    Waiting_time[i]=0;
    for (j=0; j<i; j++)
      Waiting_time[i]+= Burst_time[j];
  {
    printf("\n Process \t\t Burst Time \t\t Waiting Time \t\t Turnaround Time");
  for (i=0; i<n; i++){
    Turn_around_time[i]= Burst_time[i] + Waiting_time[i];
    avg_waiting_time += Waiting_time[i];
    avg_Turn_around_time += Turn_around_time[i];
Arrival_time[i] = Arrival_time[i];
    printf("\n Process \t\t Burst Time \t\t
```

```
int SJF();
{
  //Sorting
for (i=0; i<n; i++)
{
   pos = i;
for (j = i+1; j<n; j++) {
  if (Burst_Time[j] < Burst_time[pos])
pos = j;
}
  temp = Burst_tim[i];
  Burst_time[i] = Burst_time[pos];
  Burst_time[pos] = temp;

  temp = process[i];
  process[i] = process[pos];
  process[pos] = temp;
  }
  Inserting time value
```

temp = Arrival_time[i];
Arrival_time[i] = Arrival_time[pos];
Arrival_time[pos] = temp;

```
for(i=1; i<n; i++) {
    Waiting_time[i]=0;
    for (j=1; j<n; j++) {
        Waiting_time[i] += Burst_time[j];
        total += waiting_time[i];
    }
        avg_waiting_time = (float)total/n;
        total = 0;

    printf("\n Process \t\t Burst Time
    \t\t Waiting time \t\t Turnaround
    Time");
    for (i=0; i<n; i++)
    {
        Turn_around_time[i] = Burst_Time[j]
```

**x. Regular Expression (RE)
way of describing pattern.

```
        total += waiting_time[i];
    }
    avg_waiting_time = (float)total/n;
    total = 0;
    printf("\n Process \t\t Burst Time
    \t\t Waiting Time \t\t Turnaround Time");
    for(i=0; i<n; i++) {

    Turn_around_time[i] = Burst_time[i]
        + waiting_time[i];
    total += Turn_around_time[i];
```

```
printf("%d", process[i], Burst_time[i]
Waiting_time[i], Turn_around_time[i]);
}
avg_Turn_around_time = (float)total_tt
printf("\n\n Average Waiting Time=
%f", avg_waiting_time);
printf("\n Average Turnaround time=
%f\n", avg_Turnaround_time);
}
int main(){
printf("Enter the total no. of
processes:");
scanf("%d", &n);

printf("Enter Burst Time:\n");
for(i=0;i<n;i++){
printf("P[%d]:", i+1);
scanf("%d", &Burst_time[i]);
process[i]=i+1;
}

while(1){
printf("\n--MAIN MENU--\n");
printf("1.FCFS Scheduling\n2.
SJF Scheduling\n");
printf("\nEnter your choice:");
scanf("%d", &choice);
switch(choice){
case 1: FCFS();
break;
default: Pr
case 2: SJF();
break
```

```
default : Printf("Invalid input!!!");
}
}
return 0;
}
```

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

Output :-

Enter the total number of Processes: 3
Enter Burst Time:
P[1] : 6
P[2] : 8 5
P[3] : 4 3

Enter Arrival Time: 3
P[1] : 3
P[2] : 2
P[3] : 4

--- MAIN MENU ---
1. FCFS Scheduling
2. SJF Scheduling
Enter your choice : 1

Process	Arrival Time	Burst Time	WT	TT
P[2]	2	5	0	5
P[1]	3	6	4	10
P[3]	4	3	9	12

Enter your choice:

Process	Arr Time	B.T	WT	TT
P[2]	2	5	0	5
P[1]	3	6	7	13
P[3]	4	3	3	6

Avg WT = 3.33

Avg TT = 8.00

13/5/24

Perform Addition, Subtraction multiplication of matrix.

```
#include <stdio.h>
#include <stdlib>
ind main() {
  int i,j,c,r,k;
int a[20][20], b[20][20], ma[20,20]
 ms[20][20];
 int mm[20][20];
 printf("Enter the value for
matrix A\n");
 for(i=0; i<c ; i++) {
  for( j=0 ; j<r; j++) {
    scanf("%d", &a[i][j]);
  }
    printf("\n");
 }
printf("\n\t Enter the value for
matrix B\n");
 for (i=0; i<c; i++) {
  for (j=0; j<r; j++) {

   Scanf("%d", &b[i][j]);
 }
   printf("\n");
 }
for(i=0 ; i<c ; i++) {
 for (j=0 ; j <r ; j++) {
 ma[i][j] = a[i][j] +b[i][j];
  ms[i][j] = a[i][j] - b[i][j];
  }
 }
```

```
for(j=0 ; j<r ; j++) {
  mm[i][j] = 0;
  for(k=0; k<c; k++) {
  mm[i][j] += a[i][k] * b[k][j];
    }
  }
}

printf("outPut");
printf("The addition matrix is:
    \n");
for(i=0;i<c;i++) {
for(j=0 ; j<c;j++) {
printf("%d", ma[i][j]);
}
    printf("\n");
  }
printf("The Substraction matrix
is: \n");
for(i=0; i<c; i++) {
for(j=0 ; j<r ; j++) {
printf("%d", ms[i][j]);
}
  printf("\n");
  }
printf("The multipn matrix:\n");
for (i=0; j<c; i++)
{
    for (j=0; j<r; j++) {
  printf("%d", mm[i][j]);
  }
    printf("\n");
  }
```

Output:-

Enter the value of row and column:
3 3
1
2
3
4
5
6
7
8
9
Enter the value of matrix B:
24
5
6
7
8
9
12
15
14
The addition of matrix:-
3 7 9
11 13 15
8 11 13

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

Priority (Pre-emptive & Non-Pre-emptive)

```
#include <stdio.h>
#include <stdbool.h>
#define QUANTUM 2
int turnaroundtime(int processes[],
int n, int bt[], int wt[], int priority[])
{
  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] > 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;
}
int findavgTime(int processes[]
int n, int bt[], int at[], int
priority[]) {
    int wt[n], tat[n], total_wt =
total_tat = 0;

waitingtime(processes, n, bt, wt,
    priority);
turnaroundtime(processes, n, bt, wt,
    tat);

printf("\n\n Processes \t\t Burst
Time \t\t priority \t\t Waiting
Time \t\t Turnaround 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\t %d \t\t %d
    \t\t %d \n", i+1, bt[i], at[i],
    priority[i], wt[i], tat[i]);
}
printf("\n Average waiting time =
%f", (float) total_wt/(float)n);
return 1;
}
int main() {
    int n;
    printf(" Enter the no. of processes
    scanf("%d", &n);
```

int processes[n], burst_time[n], arrival_time[n], priority[n]};

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

findavgTime(processes, n, burst_time, arrival_time, priority);
return 0;
}

output :-

Enter the number of processes : 5

Enter the arrival time for process 1: 0
Enter the burst time for process 1: 4
Enter the priority for process 1: 2

Enter the arrival time for process 2: 1
" " burst time " " " : 3
" " priority " " " : 3

" " arrival " " " : 2
" " burst " " " : 1
" " priority " " " : 4

5 2 4 5 7 9

Average waiting time : 7.4
Average turnaround time : 10.4

Round Robin (Non-Preemptive)

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

int turnaroundtime(int processes[],
int n, int bt[], int wt[], int tat[]
{
    for (int i=0; i<n; i++)
        tat[i] = bt[i] + wt[i];
    return 1;
}
int waitingtime(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;
}
int findavgTime(int processes[],
int n, int bt[], int quantum){
int wt[n], tat[n], total_wt=0,
```

```
waiting(processes, n, bt, wt,
quantum);
turnaroundtime(processes, n, bt, wt,
tat);

printf("\n\n Processes \t\t Burst
Time \t\t waiting time \t\t
turnaround time\n");
for(int i=0; i<n; i++){
total_wt = total_wt + wt[i];
total_tat = total - tat + tat[i];

printf("\n \t %d \t\t %d \t\t %d\n",
i+1, bt[i], wt[i], tat[i]);
}
printf("\n 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 the no. of processes:
scanf("%d", &n);

printf("Enter the quantum time:
scanf("%d", &quantum);
```

```
int i=0;
for(i=0; i<n; i++){
    printf("\n Enter the process:");
    scanf("%d", &process[i]);
    printf(" Enter the Burst time:");
    scanf("%d", &burst_time[i]);
}
findavgTime(process, n, burst_time, quantum);
return 0;
}
```

Output:

```
Enter the no. of PROCESS: 3
Enter quantum : 2
```

P	BT
1	5
2	7
3	3

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

Avg WT = 7.667
Avg TAT = 12.667

```
typedef struct {
    int id;
    int burst_time;
    float priority;
} Task;
int num_of_processes;
int execution_time[MAX_PROCESS], period[MAX_PROCESS],
remain_time[MAX_PROCESS],
deadline[MAX_PROCESS],
remain_deadline[MAX_PROCESS];

void get_process_info(int selected_algo) {
    printf("Enter total no. of process (max num %d): ", MAX_PROCESS);

    scanf("%d", &num_of_process);

    if (num_of_process < 1) {
```

```
    exit(0);
}
for(int i=0; i<num_of_process; i++)
{
    printf("\n process %d\n", i+1);
    printf("==> Execution time: ");
    scanf("%d", &execution_time[i]);
    remain_time[i] = execution_time[i];

    if(selected_algo == 2) {

        printf("==> Deadline: ");
        scanf("%d", &deadline[i]);
    }
    else {
        printf("==> Period: ");
        scanf("%d", &Period);
    }
}
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_observation_time(int selected_
```

```
{
  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\n");
  printf(" Time:");
  for (int i=0 ; i <cycles ; i++)
    {if (i<10)
      printf("| 0%d", i);
    else
printf ("| %d", i);
  }printf("|\n");
  for( int i=0 ; i <num_of_proces
; i++) {
printf("P[%d]:", i+1);
  for( int j=0 ; j<cycles ; j++)
    {
      if (process_list[j]==i+1)
printf("|###");
```

```
        printf("|");
    }
    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_of_process; i++)
{
    utilization+=((1.0*execution_
    time[i]/period[i]);
    int n=num_of_process;
    int m=(float)(n*(pow(2,1.0/n)
    -1));
    if(utilization>m){
printf("\nGiven problem is not
schedulable under the said
scheduling algorithm.\n");
{
    for(int i=0; i<time; i++){
        min=1000;
    for(int j=0; j<num_of_process;
        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;
remain_time[next_process]--;
}
for (int k=0 ; k< num_of_process
 ; k++)
{
 if((i+1)%Period[k]==0)
 {
  remain_time[k] = execution_
 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_of_process
 i++) {
utilization += (1.0* execution_time[
 deadline[i];
}
 int n= num_of_processes;
int process[num_of_process];
int max_deadline, current_process=
 min_deadline, process_list[time];
 bool is_ready[num_of_process];
 for(int i=
```

```
i++) {
is_ready = true;
process[i] = i+1;

}
max_deadline = deadline[0];
for(int i=0 ; i<num_of_process ;
    i++) {
    if(deadline[i] > max_deadline)
    max_deadline = deadline[i];
}

for( int i=0 ; i<num_of_process; i++){
  for( int j= i+1; j<num_of_process;
        j++) {
  if(deadline[j] < deadline[i]) {

  int temp = execution_time[j];
execution_time[j] = execution_
    time[i];
    execution_time[i] = temp;
     temp = deadline[j];
     deadline[j] = deadline[i];
      temp = process[j];
       process[j] = process[i];
      process[i] = temp;
    }
  }
}
```

O/p

5/6/24

Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes of system no are divided into two categories - system processes and user processes. System processes are to be given higher priority than user processes.
Use FCFS scheduling for the processes in each queue.

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

struct process {
    int pid;
    int arrival_time, burst_time, priority,
waiting_time, turnaround_time;
};
void FCFS (struct process *
queue, int n) {

int i, j;
struct process temp;
for (i=0; i<n; i++) {
    for( j= i+1 ; j<n ; j++) {
if( queue[i].arrival_time >
queue[j].arrival_time ) {
    temp = queue[i];
    queue[i] = queue[j];
    queue[j] = temp;
}
```

```
int main(){

int n, i;
struct process * system_queue,
* user_queue;
int system_n=0, user_n=0;
float avg_waiting_time=0,
avg_turnaround_time=0;

printf("Enter the no. of processes:");
scanf("%d",&n);

system_queue=(struct process*) malloc
(n*sizeof(struct process));

user_queue=(struct process*) malloc
(n*sizeof(struct process));

for(i=0; i<n; i++){
struct process p;
printf("Enter arrival time, burst time,
and priority (0-system/1-user)
for process %d:",i+1);
for(i=0; i<n; i++){
struct process p;
printf("Enter arrival
scanf("%d %d %d",&p.arrival_time,
&p.burst_time,&p.priority);
p.pid=i+1;
p.waiting_time=0;
p.turnaround_time=0;
if(p.priority==0){
system_queue[system_n++]=p;
```

arrival time < Sys...
time) {

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
\n3. Exit");
 while(1)
{
   printf("\nEnter your choice:");
   scanf("%d",&n);
    switch(n){
     case1: if((mutex==1)&&(empty!
       =0))
      producer();
       else
      printf("Buffer is full!!");
      break;
     case2: if((mutex==1)&&(full!=0))
      consumer();
       else
      printf("Buffer is empty!!");
       break;
     case3: exit(0);
```

```
break;
}
}
return 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("\n producer Producer
the item %d", x);
mutex = signal(mutex);
}
void consumer()
{
mutex = wait(mutex);
full = wait(full);
empty = signal(empty);
printf("\n consumer item %d", x);
x--;
mutex = signal(mutex);
}
```

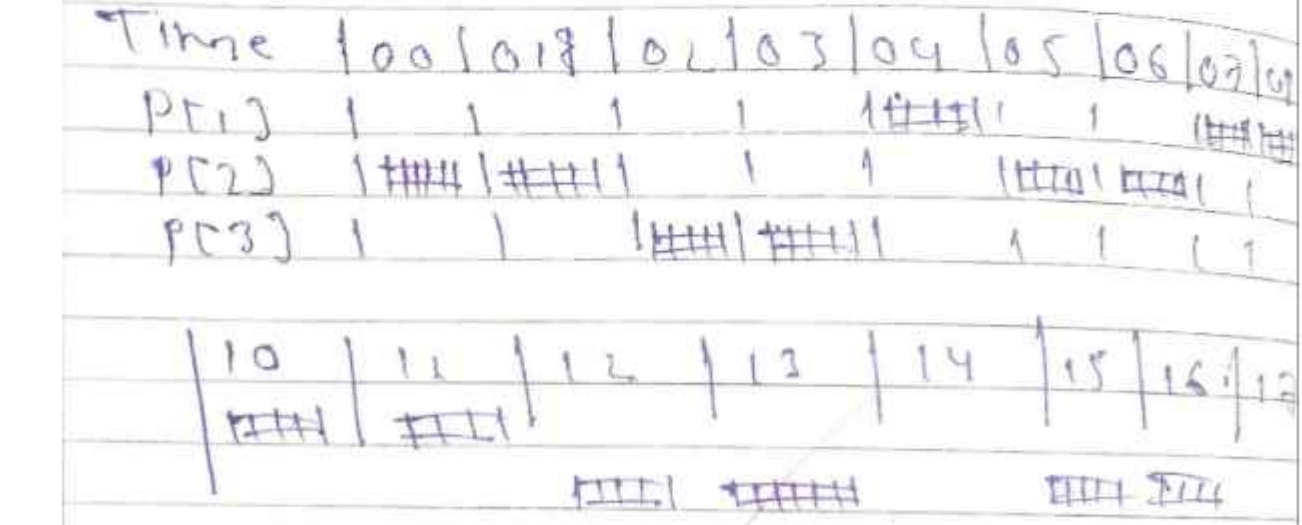
O/P 1. Producer 2. Consumer 3. exit
Enter choice: 1
Producer produces item 1
Enter choice: 1
Producer produces item 2
Enter choice: 1
Producer produces item 3.
Enter choice: 1
Buffer is full
Enter choice: 2
Consumer Consumes item 3.

Output :-
1. Rate Monotonic
2. Earliest deadline first.

Enter Your choice : 2
Enter total no. of process (max) : 3
Process : 1
execution time : 3
deadline : 20

Process 2 :
execution time : 2
deadline : 5

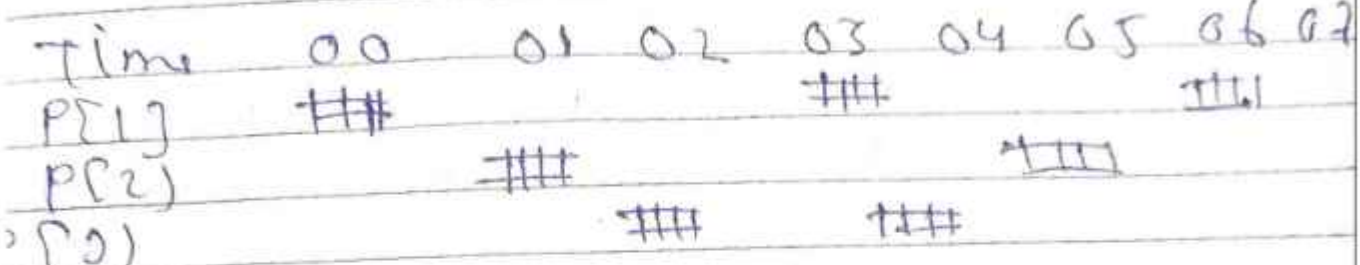
Process 3 :
execution time : 2
deadline : 10



Enter choice : 1
Enter total no. of process (max 10) : 3
Process 1 :
execution time : 1
Period : 3
Process : 2

execution time : 1
period : 4

Process 3:
execution time: 2
period : 8



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);
{
```

```
}
void take_fork(int i)
{
    sem_wait(&mutex);
    state[i] = HUNGRY;

printf("Philosopher %d is Hungry\n",
        i+1);
    test(i)
sem_post(&mutex);
  sem_wait(&S[i]);
    sleep(1)
}
void  put_fork(int i)
{
    sem_wait(&mutex);
    state[i] = THINKING;
printf("Philosopher %d putting
```

```
}

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
P2 ... ...
P3 ... ...
P4 ... ...
P5 ... hungry
P1 ... ...
P2 ... ...
P3 ... ...
P4 ... takes fork 3 and 4
P4 is eating
P4 is putting fork 3 & 4 down
P4 is thinking
P3 takes fork 2 & 3
P3 is eating
}

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 the no. of Processes:");
  scanf("%d", &n);
  printf("Enter the no. of resources:");
  scanf("%d", &m);

  int allocate[n][m];
  printf("Enter the 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 the 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]-allocation[i][j];
}
}
int y=0;
for(k=0; k<n; k++) {
  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]+=allocation[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 ; i++)
  {
    printf(" P.1.d ->", ans[i]);
  }
    printf(" P.1.d\n", ans[n-1]);
}
  return 0;
}
```

Output:-

Enter the no. of processes : 5
Enter the no. of resources : 3
Enter the allocation matrix:

```
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
```

Enter the MAX matrix:

```
7 5 3
3 2 2
9 0 2
2 2 2
```

" Program to simulate deadl
ck 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("\n Total Amount of the Resource
    R%d:", i+1);
    scanf("%d", &r[i]);
    }
printf("\n Enter the request matrix:"
```

```
for (i=0; i<np ; i++)
 for (j=0; j<nr ; j++)
 scanf ("%d", &request[i][j]);

printf ("\n 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<nr 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++)
```

```
}
    int canbeprocessed = 0
    if(mark[i] != 1) {

        for(j=0 ; j<nr ; j++)
        {
            if(request[i][j] <= w[j])
            canbeprocessed = 1;
            else
            {
                canbeprocessed=0;
                break;
            }
        }

    if(canbeprocessed)
    {
        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("\n Deadlock detected");
        else
            printf("\n No Deadlock
                Possible");
    }
```

Output:-

```
Enter the no. of Process : 5
Enter the no. of Resources : 3

Total Amount of the Resource R1: 10
Total amount of the Resource R2: 5
Total Amount of the Resource R3: 7

Enter the request matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2

Enter the allocation matrix:
0 0 1
1 0 0
1 3 5
0 6 3
0 0 1

Deadlock detected.
```

19/6/24

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

Worst-fit
Best-fit
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 Schema\n");
printf("\nEnter the no. of blocks:");
scanf("%d", &nb);

printf("Enter the no. of files:");
scanf("%d", &nf);

printf("Enter the size of the blocks:\n");
for(int i=1; i<=nb; i++)
  {
    printf("Block %d:", i);
    scanf("%d",
```

```
{
  printf("Enter the size of files:");
for(int j=1;j<=nf;j++){

printf("File %d:",j);
 scanf("%d",&f[j]);
}

printf("Memory Management Scheme - First
   Fit");
FirstFit(b,nb,f,nf);

printf("Memory Management Scheme -
   Worst Fit");
WorstFit(b,nb,f,nf);

printf("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=1;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("File_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment");
  for(i=1; i<=nf; i++)
  {
    printf("%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]],
           frag[i]);
  }
}
void worstfit(int b[], int nb, int f[],
              int nf){
  int bf[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("\n File_no:\t File_size:\t Block_
no:\t Block_size:\t fragment");
for(i=1; i<=nf; i++)
{
 printf("\n%d\t\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 bf[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(bf[j]!=1)
     {  temp = b[j]-f[i];
```

4.

```
printf("File_no:\t File_size:\tBlock_no:\t Block_size:\t Fragment");

for(i=1; i<=nf && ff[i]!=0; i++)
{
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i, f[i], ff[i], b[ff[i]], frag[i]);
}
}
```

Output:-

```
Enter the no. of blocks
5
Enter the no of processes
8
Enter the block size
100 500 200 300 600
Enter the process size
212 417 63 124 23 89 73 17
```

1. First fit
2. Best fit
3. Worst fit

3	63	4
4	124	5
5	23	4
6	89	3
7	73	3
8	13	3

Worst fit

NO	Size	block
1	212	NA
2	415	NA
3	63	NA
4	124	NA
5	23	5
6	89	NA
7	73	NA
8	13	5

Write a C program to simulate Page replacement algorithms:

FIFO
LRU
Optimal

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

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

void initialize()
{
   pgfaultcat=0
   for(i=0;i<f;i++)
     P[i]=9999;
   }
```

```
int isHit(int data)
{
  hit=0;
  for(j=0;j<f;j++)
  {
    if(b[j]==data)
    {
      hit=1;
      break;
    }
  }
  return hit;
}
int getHitIndex(int data)
{
  int hitind;
  for(k=0;k<f;k++)
  {
    hitind=k;
    break;
  }
  return hitind;
}
void dispPages()
{
  for(k=0;k<f;k++)
  {
    if(b[k]!=9999)
    printf(" %d",b[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(isHit(in[i])==0)
{
  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("\n for %d:", in[i]);
```

```
if(isHit(in[i])==0)
{
    for(j=0;j<f;j++)
    {
        int pg=b[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;
        }
    }
    b[repindex]=in[i];
    pgfaultcnt++;
```

```
    displayPages();
}
else
    printf("No Page Fault");
    displayPgFaultCnt();
}

void lru()
{
    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<cnt; 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("\n Page Replacement Algorithms
\n1. Enter data \n2. FIFO \n3.
Optimal \n4. LRU \n5. Exit \n Enter
your choice:");
```

```
scanf("%d", &choice);
switch(choice)
{
  case 1: getData();
    breaks;
  case 2: fifo();
    breaks;
  case 5: optimal();
    breaks;
  case 4: lru();
    breaks;
  default: return 0;
    break;
  }
 }
}
```

c/b

Page replacement Algorithm

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

Enter your choice : 2
Enter length of page reference sequence : 12
Enter the page reference : 1 2 3 4 1 2 5 1 2 3 4 5
Enter the no. of eq frame :
frame 1 : 1

2 : 1 2
3 : 1 2 3
4 : 2 3 4
1 : 3 4 1
2 : 4 1 2
5 : 1 2 5
1 : No Page fault
2 : No Page fault
3 : 2 5 3
4 : 5 3 4
5 : No page fault

Total no. of page fault : 9
Enter your choice : 3
for 1 : 1
2 : 1 2
4 : 1 2 4
1 : No PF
2 : No PF
3 : 1 2 5
5 : 1 2 5
1 : No PF
2 : No PF
3 : 3 2 5
4 : 4 2 5
5 : No PF

Total no. of PF's : 7
Enter your choice : 4
for 1 : 1
2 : 1 2
3 : 1 2 3
4 : 4 2 3
1 : 4 1 3
2 : 4 1 2
5 : 5 1 2

Write a C program ... scheduling algorithms.

FCFS

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int RQ[100], i, n, TotalHeadMoment=0, initial;
printf("Enter the number of Requests\n");
for (i=0; i<n; i++)
  scanf("%d", &RQ[i]);
printf("Enter initial head position");
 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);
return 0;
}
```

output:-

Enter the number of Requests
5

Enter the sequences
98
183
37
122
14
Enter initial head position
53
Total head moment is 469.

b) SCAN.

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

int main()
{
    int RQ[100], i, j, n, TotalHeadMovement=0, initial, size, move;
    printf("Enter the number of Requests\n");
    scanf("%d", &n);
    printf("Enter the sequests sequence\n");
    for(i=0; i<n; i++)
        scanf("%d", &RQ[i]);
    printf("Enter initial head position\n");
    scanf("%d", &size);
    printf("Enter the head movement direction for high and for low 0\n");
    scanf("%d", &move);
```

```
for( i=0 ; i<n ; i++)
{
   for (j=0 ; j<n-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 (initial < RQ[i])
         {
               index = i;
               break;
         }
   }
if (inc == 1)
   {
      for (i= index ; i<n ; i++)
   }
TotalHeadMoment = TotalHeadMoment +
   abs(RQ[i] - initial);
      initial = RQ[i];
   }
TotalHeadMoment = TotalHeadMoment + abs(
   size - RQ[i-1]-1);
```

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

- Enter the number of Requests
  8
  Enter the Requests Sequen
  98
  183
  37
  122

14
124
65
67
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 331

C) C-SCAN

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int RQ[100], i, j, n, TotalHeadMovement=0,
initial, size, move;
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);
printf("Enter total disk size\n");
```

```
scanf("%d", &size);
printf("Enter the head movement
direction for high 1 and for low 0");
scanf("%d", &move);

for(i=0; i<n; i++)
{
  for(j=0; j<n-1; j++)
  {
    if(RQ[j]>RQ[j+1])
    {
      int temp;
      temp = RQ[j+1];
      RQ[j+1] = temp;
    }
  }
}
int index;
for(i=0; i<n; i++)
{
  if(initial<RQ[i])
  {
    index = i;
    break;
  }
}
if(move==1)
{
  for(i=index; i<n; i++)
  {
TotalHeadMovement = TotalHeadMovement +
    abs(RQ[i] - initial);
  initial = RQ[i];
```

```
}
TotalHeadMovement = TotalHeadMovement +
  abs(size - RQ[i-1] - 1);

TotalHeadmovement = TotalHeadMovement +
    abs(size - 1 - 0);
  initial = 0;
  for(i=0; i<index; i++)
  {
  TotalHeadMovement = TotalHeadMovement +
 abs(size RQ[i] - initial);
    initial = RQ[i];
  }
  {
    else {
  for(i=index-1; i>=0; i--)
  {
TotalHeadMovement = TotalHeadMovement +
      abs(size - initial);
  initial = RQ[i];
  }
TotalHeadMovement = TotalHeadMovement +
   abs(RQ[i+1]-0);
TotalHeadMovement = TotalHeadMovement + abs
(size-1-0);
  initial = size-1;
  for(i=n-1; i>=index; i--) {

  TotalHeadmovement = TotalHeadMovement
  + abs(RQ[i] - initial);
    initial = RQ[i];
  }
}
```

printf("Total head movement: %d", total_head_movement);
return 0;
}

a/b.

Enter the number of Requests
5
Enter the Requests sequence
12 40 6 22 11
Enter initial head position
20
Enter total disk size
100
Enter the head movement direction for high 1 and for low 0
1
Total head movement is 190

10/7/24