INDEX

Name AKSHAT JAIN Sub. O.S LAB

Std.: CSE (4A) Div. Roll No. 1BM22CS030

Telephone No. E-mail ID.

Blood Group. Birth Day.

Sr.No.	Title	Page No.	Sign./Remarks
1	Matrix Calculations	10	8
2	Cpu Scheduling using FCFS, SJF		
3	Cpu scheduling using priority (non- pre emptive)	8	
4	Round Robin (pre- emptive)		
~~5~~	~~Multi-level Queue scheduling~~		
5	Real -time Scheduling (a) Rate- monotonic (b) Earliest - deadline - first.		
6	(a) Producer Consumer (b) Philosophers dining.	9	19/6/24
7	Bankers Algorithm (1) deadlock avoidance (2) deadlock detection.	10	
8	Contiguous memory Allocation worst, best, First - fit Page Replacment Algorithms FIFO, LRU, Optimal	10	3/7/24
9	Disk Scheduling Algo (1) FCFS (2) Scan (3) C-Scan	10	10/7/24

① Add, Sub, multiply of two matrices.

Sol-

```
#include <stdio.h>
#define Rows 3
#define Cols 3

Void print (int matrix[][Cols]) {
    for(int i=0; i<Rows; i++) {
        for(int j=0; j<Cols; j++) {
            printf("%d\t", matrix[i][j]);
        }
        printf("\n");
    }
}

Void addmatrix (int matrix1[][Cols], int matrix2[][Cols],
                int result[][Cols]) {
    for (int i=0; i<Rows; i++) {
        for (int j=0; j<Cols; j++) {
            result[i][j] = matrix1[i][j] + matrix2[i][j];
        }
    }
}

Void Subtract (int matrix1[][Cols], int matrix2[][Cols],
               int result[][Cols]) {
    for(int i=0; i<Rows; i++) {
        for(int j=0; j<Cols; j++) {
            result[i][j] = matrix1[i][j] - matrix2[i][j];
        }
    }
}
```

```
Void multiply (int matrix1[][Cols], int matrix
                [][Cols], result[][Cols]){

    for (int i=0; i<Rows; i++){
        for (int j=0; j<Cols; j++){
            result[i][j]=0;
            for (int k=0; k<Cols; k++){
            result[i][j] += matrix1[i][k] * matrix2
                                [k][j];
            }
        }
    }
}

int main(){
    int matrix1[Rows][Cols];
    int matrix2[Rows][Cols];
    int result[Rows][Cols];

    printf("enter elements of matrix1: \n");
    for (int i=0; i<Rows; i++){
        for (int j=0; j<Cols; j++){
            printf("matrix1[%d][%d]:", i, j);
            scanf("%d", &matrix1[i][j]);
        }
    }

    printf("enter elements of matrix2: \n");
    for (int i=0; i<Rows; i++){
        for (int j=0; j<Cols; j++){
            printf("matrix2[%d][%d]:", i, j);
            scanf("%d", &matrix2[i][j]);
        }
    }
```

```
addMatrices (matrix1, matrix2, result);
printf("Addition");

Substract (matrix1, matrix2, result);
printf("sub");

multiplyMatrices (matrix1, matrix2, result);
printMatrix (result);
}
```

8/5/24

PAGE NO:
DATE:

① Write a C program to Simulate the following non-pre-empive CPU Scheduling algorithm to find turnaround time and waiting time

(a) FCFS

(b) SJF

Sol:-

```
#include <stdio.h>
int n, i, j, pos, temp, choice, burst_time[20], wait_time[20], arrival_time[20], turn_around_time[20], process[20], total;
float avg_Turn_around_time = 0, avg_waiting_time = 0;
int FCFS()
{
    waiting_time[0] = 0;
    for(int i = 1; i<n; i++)
    {
        waiting_time[i] = 0;
        for (j=0; j<i; j++)
        {
            waiting_time[i] += Burst_time[j];
        }
    }
    printf("\n");
    for (i=0; i<n; i++)
    { avg_arrival_time += arrival_time[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];
    }
    avg_waiting_time = (float)(avg_waiting_time)/(float);
    avg_turn_around_time = (float)(avg_turn_around_time
                                   /(float);
    return 0;
}
```

```
int SJF()
{
    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_time[i];
        burst_time[i] = burst_time[pos];
        burst_time[pos] = temp;
        temp = process[i];
        process[i] = process[pos];
        process[pos] = temp;
    }
    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];
        }
        total += waiting_time[i];
    }

avg_waiting_time = (float) total / n;
total = 0;
printf("\n");
```

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

```
for (i=0; i<n; i++){
Turn-around-time (i) = burst time (i)+ waiting time
total += Turn-around time [i];
}

Avg-Turn-around-time = (float) total/n;
printf
printf(" %.f", Avg-waiting-time);
printf(" %.f %.f", Avg-turn-around-time, Avg-
                                    time);
}
int main() {
  printf(" Enter the total number 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 \n 2. SJF \n");
  printf(" enter your Choice");
  scanf(" %d", &Choice);
  switch (Choice)
  {
    case 1: FCFS();
            break;
```

```
Case 2:  SJF();
         break;

default : printf("Invalid input \n");
}
}
return 0;
}
```

Output-

Enter the total number of Processes : 3

Enter the Burst Time and Arrival Time:
P[1]: 6
0
P[2]: 8
1
P[3]: 4
2

---MAIN MENU---
1. FCFS Scheduling
2. SJF Scheduling

Enter your Choice: 1

Process	Burst Time	Arrival Time	Waiting	Turnaround
P[1]	6	0	0	6
P[2]	8	1	5	13
P[3]	4	2	12	16

Average Arrival Time : 1.00
Average Waiting Time : 5.67
Average Turnaround Time : 11.67

PAGE NO:
DATE:

--- MAZN MENU ---
1. FCFS Scheduling
2. SJF Scheduling

Enter your Choice: 2

Process	Burst	Arrival	Waiting	TurnAround
P[1]	4	2	0	6
P[2]	6	0	9	12
P[3]	8	1	4	8

Average Arival Time: 1.00
Average Waiting Time: 4.33337
Average TurnAround Time: 10.33337

Note
15/5/24

Q Write a C- program to simulate the following CPU Scheduling Algorithm to find turnaround time and waiting time.

(a) priority (non pre-emptive)

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

Struct process {
    int process_id;
    int burst_time;
    int priority;
    int waiting_time;
    int turnaround_time;
};

int main(){
    int n,i;
    Struct process proc[10];
    Printf("enter number of process");
    Scanf("%d",&n);
    for(i=0;i<n;i++){
        Printf("Enter");
        Scanf("%d %d %d", &proc[i].p_id, &proc
            [i].b_t, &proc[i].p);
    }
    priority Scheduling(proc,n);
    Return 0;
}

Void f_w_t (Struct process proc[], int n, int wt[]){
    int i;
    wt[0]=0;
```

```
for (i=1; i<n; i++)
{
  wt[i] = proc[i-1].bt + wt[i-1];
}
}

void f_t_t (struct process proc[], int n,
            int wt, int tat[])
{
  int i;
  for (i=0; i<n; i++)
  {
    tat[i] = proc[i].bt + wt[i];
  }
}

void f_a_t (struct process proc[], int n)
{
  int wt[10], tat[10], total_wt = 0, total_tat = 0;
  f_w_t (proc, n, wt);
  f_t_t (proc, n, wt, tat);
  printf("Process \n Burst \n priority \n
         Waiting \n Turnaround \n");

  for (i=0; i<n; i++)
  {
    total_wt = total_wt + wt[i];
    total_tat = total_tat + tat[i];
    printf("%d \t %d \t %d \t %d \t %d",
           proc[i].process_id, proc[i].bt,
           priority, wt[i], tat[i]);
  }
  printf("Average wt = %f", (float) total_wt
}
```

```
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;
    }
    ~~fcfs~~. f-d-t (proc, n);
}
```

Output - Enter the no of processes : 5
Enter process id, Burst time, and priority.

P1: 11 2
P2: 28 0
P3: 2 3
P4: 10 1
P5: 16 4

Process ID	B.T	priority	wt	TAT
1	28	0	0	28
2	10	1	28	38
3	11	2	38	49
4	2	3	49	51
5	16	4	51	67

Average wt = 33.20
Average tat = 46.59.

DATE :

(b) Round Robin

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

int wt(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 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;
}

int fat (int processes [], int n, int bt[], int quantum){
    int wt[n], tat[n], total_wt = 0, total_tat = 0;
    wat (processes, n, bt, wt, quantum);
    tat (processes, n, bt, wt, tat);
    printf("\n processes \n Burst \n waiting \t
            turnaround time \n");
    for (int i=0; i<n; i++)
    {
     total_wt = total_wt + wt[i];
     total_tat = total_tat + tat[i];
     printf("\t %d \t %d \t %d \t %d \n", i+1,
            bt[i], wt[i], tat[i]);
    }

    printf ("\n Average wt = %f", (float) total_wt
                                / (float) n);

    printf("\n Average tat = %f", (float) total_tat
                                / (float) n);

    return 1;
}
```

```
int main(){

int n, processes[n], burst_time[n], quantum;
printf("Enter the number of Processes");
scanf("%d", &n);

printf("\n Enter the quantum time");
scanf("%d", &quantum);

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

fcfs(processes, n, b_t, quantum);
return 0;
}
```

Output - Enter no of process: 3
Enter quantum: 2

P	BT
1	5
2	7
3	3

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

Avg wt = 7.667 Avg tat = 12.667

PAGE NO:
DATE: 5/6/24

9 Write a C program to simulate Real-time CPU scheduling Algorithms.

(a) Rate-Monotonic

(b) Earliest-deadline-first.

Sol:

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

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

int num_of_process;
int execution_time[MAX_process], period[max_process],
remain_time[max_process], deadline[max_process];
void get_process_info(int selected_algo) {

    printf("Enter total no of processes (max %d): ",
                                    maxprocess);
    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[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_observation_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\n");
  printf(" Time: ");
  for (int i=0; i<cycle; i++)
  {
    if (i<10)
      printf("0 :1:d ", i);
    else
      printf(" 1:1:d ", i);
  }
  printf (" P[:1d]:", i++);
  for (int j=0; j<cycles; j++)
  {
    if (process_list[j] == i+1)
      printf (" ## ## ");
    else
      printf(" | ");
  {
  printf("\n");
}}

Void edd_monotonic (int time)
{
  int process_list[100] = {0}, min = 999,
      next_process = 0;
  float utilization = 0;
```

PAGE NO :
DATE :

```
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("In Given problem is not Schedulable");
}
 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+1;
   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[i]/deadline[i]);
  int n = num_of_process;
  int process[num_of_process];
  int max_deadline, current_process = 0;
  min_deadline, process_list[time];
  bool is_ready[num_of_process];
  for (int j=0; i<num_of_process; i++)
   {
     is_ready[i] = true;
     process[i] = i+1;
   }
  max_deadline = deadline[0];
  for (int i=1; 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-temp[j];
        execution-time = temp;
        deadline[j] = deadline[i];
        temp = process[j];
        process[j] = process[i];
        process[i]=temp;
    }
}
}
for (int i =0; i<num-of-process; i++)
{
  remain-time[i] = execution-time[i];
  remain-deadline[i]= deadline[i];
}

for(int t= 0; t<time; t++) {
  if (current-process != -1);
  {
    execution-time[current-process];
    process-list[t] = process(current-process);
  }
  else
    process-list[t]= 0;

for (int i=0; i<num-of-process; i++)
{
  --deadline[i];
  if((execution-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)) {

  execution_time[i] = remain_time[i];
  is_ready[i] = true;
}
}

min_deadline = max_deadline;
current_process = -1;

for(int i=0; i<num_of_process; i++){
  if (deadline[i] <= min_deadline) &&
      (execution_time[i] > 0)) {

    min_deadline = deadline[i];
    current_process = i;
  }
}}
print_Schedule (process_list, time);
}

int main() {

  int option;
  int observation_time;
  while(1)
  {
    printf("\n 1. Rate Monotonic
                2. Earliest deadline
                3. Proportional Scheduling
\n enter your choice:");
```

```
Scanf("%d", &option);
Switch (option)
{
 Case 1: get_process_info. (option);
         Observation_time = get_observation_time;
         Rate_monotonic (observation_time);
         break;
 Case 2:
         get_process_info (option);
        observation_time = get_observation_time;
        Earliest_deadline_first (observation_time);
        break;
 Case 3:
         exit (0);

 default: printf("\n Invalid Statement");
 }}
 return 0;
}
```

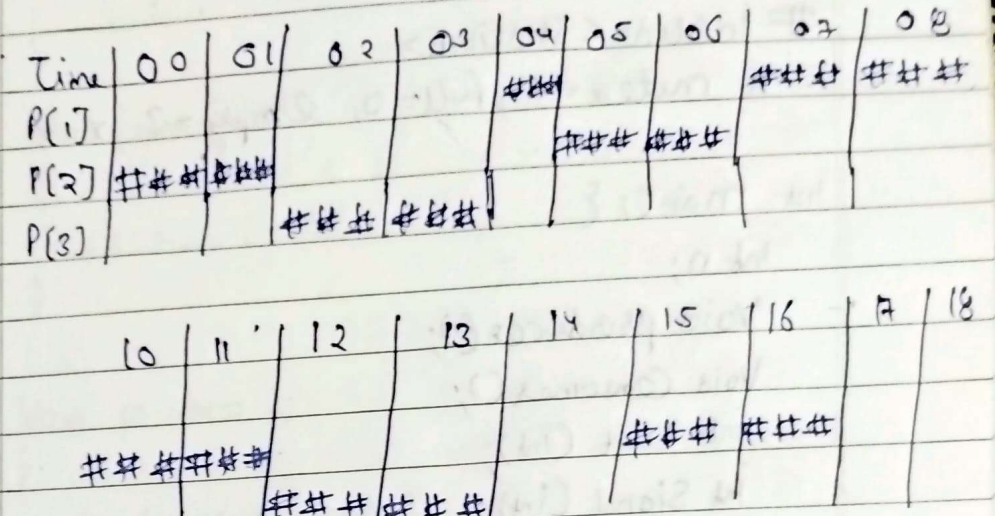
Output-

1. Rate monotonic
2. Earliest Deadline first.

Enter your choice: 2
Enter total no of process (max 10): 3
Process 1:
Execution time: 3
Deadline: 20

Process 2:
Execution Process: 2
Deadline: 5

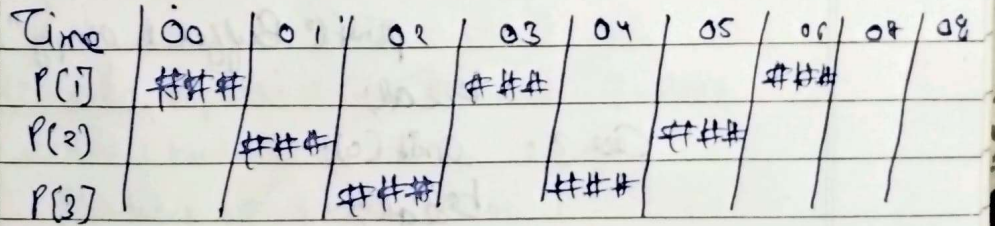
Process 3:
Execution time : 2
deadline : 10



Enter your choice : 1
Enter total no of process (max(3)) : 3
Process: 1
Execution time : 1
period : 3

process: 2
Execution time: 1
period: 4

Process: 3
Execution time : 2
period: 8



PAGE NO
DATE: 12/06/24

Q Write a C program to Simulate producer - Consumer problem using semaphores.

Sol:-

```
#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 2. Consumer 3. Exit\n");
  while(1)
  {
    printf("\n 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 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 produces the item : %d", x);
    mutex = Signal (mutex);
}

Void Consumer ()
{
    mutex = wait (mutex);
    full = wait (full);
    empty = Signal (empty);
    printf(" Consumer Consumes item : %d", x);
    x--;
    mutex = Signal (mutex);
}
```

Output - 1. producer 2. Consumer 3. Exit
Enter choice : 1
Producer produces item 1
Enter choice : 3.

PAGE NO:
DATE: 12/06/24

Q Write a C program to simulate the concept of Dining Philosophers problem.

```
#include <stdio.h>
#include <stdlib.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("Philosophe %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 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 fork %d
        and %d down \n", i+1), left+i,
                  i+1);
  printf("Philosopher %d is Thinking",
                  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, &ph[i]);
      printf ("Philosopher %d is thinking", i+1);
  }
  for (i=0; i<N; i++) {
      pthread_join (thread_id[i], NULL)
  }
}
```

Output - Philosopher 1 is thinking.
2 —"— —"—
3 —"— —
4 —"— —
5 is hungry
1 —"— —
2 —"— —
3 —"— —
4 takes fork 3 and 4 (illegible)
4 is eating
4 is putting fork 3 & 4
4 is thinking
3 takes fork 2 & 3
3 is eating

PAGE NO:
DATE: 19/06/24

Q Write a C-program to simulate bankers algorithm for the purpose of deadlock avoidance.

Sol-

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

    int n, m, i, j, k;
    printf("Enter the no of Processes");
    scanf("%d", &n);
    printf("Enter the number of resources");
    scanf("%d", &m);

    int allocation [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++)
{
   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] += 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-1; i++)
    {
        printf("P%d ->", ans[i]);
    }
```

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

Output -

```
Enter the number of Process: 5
Enter the number 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
4 3 3

Enter the available Resources:
3 3 2

Following is the Safe Sequence:
P1 -> P3 -> P4 -> P0 -> P2
```

PAGE NO.
DATE 13/12/24

Q Write a C program to simulate deadlock detection.

Sol

```
#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], wk
  printf("enter the no of process");
  scanf("%d", &np);
  printf("\n enter the no of Process");
  scanf("%d", &nr);
  for (i=0; i<n; i++)
  {
     printf("Total Amount of the Ressource R%d", i+1)
     scanf("%d", &r[i]);
  }

  printf("\n enter the request matrix");
  for (i=0; i<=np; i++)
  {
     for (j=0; j<npa; 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++)
{
    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 processes: 5
Enter the no of resources: 3

Total Amount of the Resources R1: 10
R2: 5
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.

PAGE NO
DATE 03/07/24

0 Write a C-program to stimulate the following Contignous 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 Mangement Schemes\n");

  printf("Enter the no of blocks: ");
  scanf("%d", &nb);

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

  printf("Enter the size of blocks");
  for(int i=1; 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]);
```

```
firstfit (b,nb,f,nf);
printf("worst fit");
worstfit (b,nb,f,nf);
printf("best fit");
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("\n file");
    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 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 (" File no");
  for (i= 1; i<=nf; i++) {
 printf(" \n %d\t %d\t %d\t %d\t %d\t",
            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 flag[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];
            if (temp >= 0 && lowest > temp)
            {
                ff[i] = j;
                lowest = temp;
            }
        }
    }
flag[i] = lowest;
bf[ff[i]] = 1;
lowest = 10000;
}

printf("File no");
for (i=1; i<=nf && ff[i] != 0; i++)
{
    printf("\n %d \t %d \t %d \t %d \t %d \t", i, f[i], ff[i], b[ff[i]], flag[i]);
}
}
```

Output - Memory management schemes-
Enter the number of blocks: 5
Enter the number of files: 8

Enter the size of the blocks:
block1: 100 2: 500 3: 200 4: 300 5: 600

Enter the size of the files:
212 415 63 124 23 89 73 13

→ First fit

Process no	Process size	Block no.
1	212	2
2	415	5
3	63	1
4	124	2
5	23	1
6	89	2
7	73	2
8	13	1

best fit

Process no	Process size	Block no
1	212	4
2	415	NA
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

PAGE NO:
DATE:

① 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");
    scanf("%d", &n);
    printf("Enter page sequence");
    for (i = 0; i < n; i++)
        scanf("%d", &in[i]);
    printf("\n Enter no of frames");
    scanf("%d", &f);
}

void initialize()
{
    pgfaultcnt = 0;
    for (i = 0; i < f; i++)
        p[i] = 9999;
}

int isHit (int data)
{
```

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

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

void dispage()
{
  for (k=0; k<f; k++)
  {
    if (p[k] != 9999)
      printf("%d", p[k]);
  }
}
```

```
void dispPgfaultCnt()
{
    printf(" Total no of Pages : %d", PgfaultCnt);
}

void fifo()
{
    int getData();
    Initialize();
    for(i=0; i<n; i++)
    {
        printf("\n In 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 (visit[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[j];
        ~~Repindex = j;~~
        pgfault(cnt++);

        dispPages();
    }
    else
        printf("No Page fault");
}
    dispPgfault(cnt);
}

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<nf; j++)
    {
        if(least[j] < min)
        {
            min = least(j);
            repindex=j;
        }
    {
    p[repindex] = in[i];
    pgfaultcnt++;

    dispPages();
    }
    else
        printf("No page fault!");
    }
    dispfaultCnt();
{

int main()
{
    int choice;
    while(1);
    {
        printf(" Page Replacement ");
```

```
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 your choice: 2
Enter length of Page refrence sequence: 12
Enter the Page refrence: 1 2 3 4 1 2 5 1 2 3 4 5
Enter the no of frames: 3
For 1: 1
    2: 1 2
    3: 1 2 3
    4: 2 3 4
    1: 3 4 1
    2: 4 1 2
```

5: 125
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: 12
3: 123
4: 124
5: No pf
2: No Pf
5: 125
1: No Pf
2: No Pf
3: 325
4: 425
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

1: no pf
3: 312
4: 342
5: 345

Total pf: 10.

Q) Write a C-program to disk Scheduling
① FCFS
② SCAN
③ C-SCAN

① #include <stdio.h>
#include <stdlib.h>
int main () {
int RQ[100], i, n, TotalHeadMoment = 0, initial;
printf ("Enter the Number 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);
}

printf ("Total head moment is %d", TotalHeadMoment);
return 0;
}

Output -

Enter the Number of Requests -
3
Enter Request Sequence
2
3
4

PAGE NO :
DATE :

Enter initial head position
2

Total head moment is 2

(2)
```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int RQ[100], i, j, n, TotalHeadMoment=0, initial,
        size, move;
    printf("Enter the number 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");
    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 (initial < RQ[i])
    {
        index = i;
        break;
    }

if (move == 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 = RQ[i];
    }
TotalHeadMoment = TotalHeadmoment + abs(RQ[i+1]-0);
initial = 0;
for (i=index; i<n; i++)
{
    TotalHeadMoment = TotalHeadmoment + abs(RQ[i]-initial);

    initial = RQ[i];
}
}
printf("Total head movement is %d", TotalHeadmoment);
return 0;
}
```

Output-

Enter the number of Requests-
3
Enter the Request Sequence.
2
3
4
Enter initial head position
2
Enter total disk size 20

Enter head Movement-
1
Total head movement is 34

PAGE NO :
DATE :

③

```
#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");
     Scanf("%d", &n);
    printf("Enter the Requests Sequence");
     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 head movement");
     Scanf("%d", & move);

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

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

printf (" Total head moment is %d ", TotalHeadmoment);
return 0;
}
```

Output-

Enter the number of ~~process~~ requests-
3

Enter the Request sequence
90 200 150

Enter initial head position
90

Enter disk size-
200

Enter head movement-
1

Total head movement = 310

10/7/24