INDEX

Name Akshay Raj Aryan Std 4th Sem Sec A.

Roll No. CS032 Subject OS Lab. School/College BMS

School/College Tel. No. Parents Tel. No.

Sl. No.	Date	Title	Page No.	Teacher Sign / Remarks
1.	8/5/24	write a c-Program to calculate addition multiplication and substraction	1-6	10
2.	15/5/24	SJF, FCFS	7-11	
3.	5/6/24	FCFS Scheduling for each queue.	12-17	10
4.	5/6/24	Rate Monotonic	18-28	
5.	12/6/24	Product consumer	29-32	8
6.	12/6/24	Dining Philosopher.	33-36	8
7.	19/6/24	Banker algorithm.	37-40	9
8.	19/6/24	deadlock algorithm	41-43	9
9.	3/7/24	worst Fit, Best Fit, First Fit	44-46	10
10.	3/7/24	FIFO, LRU, optimal.	47-54	10
11.	10/7/24	FCFS, SCAN, C-SCAN	55-64	10

Date __/__/__
Page 1

Q. write a C Program to develope a addition, Substraction and multiplication of a matrix.

Sol^n

```
#include <stdio.h>

void add matrices (int mat1 [Max size] [Max size],
        int mat2 [Max size] [Max size],

int result [Max size], int rows, int cols) {

For (int i = 0 ; i < rows ; i++) {
For (int j = 0 ; j < cols ; j++)) {
 result [i] [j] = mat [i] [j] + mat 2 [i] [j] ;
}
}
}

void Substract matrices (int mat1 [Max size] [Max siz],
int mat2 (Max size] [Max size], int result
[Max size] [Max size], int rows, int cols) {

For (int i = 0 ; i < rows ; i++) {
For (int j = 0 , j < cols ; j++) {
result [i] [j] = mat1 [i] [j] + mat 2 [i] [j];
}}}

void Multiply matrixes (int mat1 [Max size] [Max size],
int mat 2 [Max size] [Max size], int result [Max siz]
[Max size ], int rows, int cols ) {
```

```
For (int i = 0 , i< rows1 ; i++ ){
For (int j = 0 , j< cols ; j++ ){
result [i][j] = 0;
For (int k = 0; k< cols1 ; k++ ){
result [i][j]+= mat1 [i][j] * mat2 [k] [j];
}}}

void display matrixes (int mat [MaxSize] [MaxSize]
, int rows , int cols ){

For ( int i = 0 ; i< rows; i++) {
For ( int j = 0 ; j< cols , j++) {

Printf ("%d ", mat [i][i]);
}
Printf ("\n");
}
}
int main ( ) {
int mat1 [MaxSize][MaxSize], Mat2 [maxSize][cols]
result [Max size][Max size]

int rows1 , cols1 , rows2, cols2;

Printf("Enter the No. of rows and colums
in the first row of matrix :");

Scanf ("%d %d", &rows1 &colums2);
```



```
Printf ("Enter the elements of the first matrix :\n");

For (int i = 0 ; i< row; i++ ){
For ( int j =0; j< colums ; j++) {
Scanf ("%d" , & mat1 [i][j]);
}}

Printf("Enter the No. of rows and colums in the
Second row of matrix :");

Scanf ("%d %d" , &rows2 , &rows2)

Printf (" Enter the elements of the Second matrix :\n");

For ( int i = 0 , i < rows2 , i++ ) {
For ( int j =0 ; j < cols2 , j++ ){
Scanf ("%d" , & mat 2 [i][j] );
}}

if ( rows1 == rows 2 && cols1 == cols 2) {
Printf ("\n Addition of matrices :\n");

add matrices ( mat1 , mat 2 , result , rows1 , cols1);
display matrices ( results , rows1 , cols1);

}
else
{
```

```
Printf("| matrices cannot be added. They
        must have same dimensions|n");

if( rows1 == rows2 && cols1 == cols2){
Printf("| n Subtraction of matrices :|n");
Subtract matrices(mat1, mat2, result, rows1, cols1);

display matrices( result, row1, cols1);
}
else {
Printf("| n matrices cannot be Subtracted
        They must have same dimension.|n");

if (cols1 == rows2) {

Print f("| Multiplication of matrices :|n");

multiply matrices(mat1, mat2, result, rows1, cols1
                  cols2);
display matrix( result, rows1, cols2);
}
else {

Printf("| n Matrices cannot be multiplied.
        Number of columns of first matrix must
        be equal to the number of rows of
        the Second matric|n")
}
return 0;
}
```


OUTPUT

```
Enter the Number of rows and columns of first
matrix : 2    2

Enter the elements of the matrix
2
5
3
6

Enter the Number of rows and colums of the Second
matrix : 2    2.

Enter the elemnt of the matrix.
5
8
9
3

Addition of Matrices :-
7    13
12    9

Subtraction of Matrices :
-3    -3
-6     3

Multiplication of matrices :-

55    31
69    42.
```

Sub 8/5/24

Date / /
Page 7

Q. Write a C Program to Simulate the following non-Pre-emptive CPU Scheduling algorithm to Find turn around time and waiting time.

(a) FCFS
(b) SJF

Sol^n

```
#include <stdio.h>
int n, j, i, 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

int FCFS()
{
waiting time [0] = 0;
for (i = 1, i < n, i++)
{
waiting time [i] += Burst_time [j];
}
Printf("\n Process \t Burst time \t waiting
        time \t Turn around 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]
avg_Arrival_time += Arrival_time[i]
Printf("\nP[%d] \t %d \t %d \t %d", i+1 Burst time
[0] waiting time [i]; Turn around_time[i]);
```

Date / /
Page 8

```
avg_time waiting_time = (float)(avg_waiting_time)/(float);
avg_Turn_around_time = (float)(avg_Turn_around_time)/(flo
Printf("\n Average waiting time :%.2f", avg waiting_time);
Printf("\n Average Turnaround Time :%.2f n", avg_turn
        around Time),
Printf("| Average arrival time :%.2f, Arrival time);
return 0;
}
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_time [i]
Burst_time [i] = Burst_time [Pos];
Burst_time [Pos] = temp ;

temp = Process [i];
Process [i] = Process [Pos] ;
Process [Pos] = temp ;
  temp = Arrival time [i]
Arrival_time[i] = Arrival time[Pos].
waiting_time [0] = 0;          Arrival_time[Pos] = temp;
```

Date / /
Page 9

```
for ( i=1 ; i<n ; i++)
{
waiting_time [j] = 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 Process \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 + = Turn_around_time [i];

Printf("\n P \t| %d", Process [i], Burst_time [i]
      waiting time [i], Turn around_time [i]);
}
avg_Turn_around_time = (float) total /n;

Printf("\n\n Average waiting time = %f", avg waiting time

Printf("\n Average Turnaround Time = %f", avg_turn-
        around_time);
}
```

```
int main()
{
Printf("Enter the total number of Process:");
Scanf("%d", &n);

Printf("\n 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("\n Enter your choice:");
Scanf("%d", &choice);
Switch (choice)
{
case 1: FCFS();
break;

case 2: SJF ();
break;
default: Printf("Invalid input!!!");
}}
return 0;
}
```


OUTPUT

```
Enter the Total Number of process : 5

Enter Burst Time :
P[1] : 15
P[2] : 18
P[3] : 19
P[4] : 14
P[5] : 11

---- Main Menu ----

1. FCFS Scheduling
2. SJF Scheduling.

Enter your choice : 1
```

Process	Burst time	Arrival Time	waiting time	Turn around time
P[1]	15	12	0	15
P[2]	18	45	15	33
P[3]	19	2	33	52
P[4]	14	32	52	66
P[5]	11	5	66	77

```
Average waiting Time : 33.20

Average Turn around Time : 48.60

Average Arrival time :- 19.20
```

15/5/24

Date / /
Page 12

7 write a c-Program to Stimulate multi-level queue Scheduling algorithm considering the following the following Scenario. All the process in the System 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 both process in each queue

Soln ⇒

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

Struct process {
int pid;
int arrival_time;
int burst_time;
int priority;
int waiting_time;
int 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;
???
```

Date / /
Page 13

```
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 number 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);

Scanf("%d%d%d", &P.arrival_time, &P.burst_time,
      &P.Priority);
P.Pid = i+1;
P.waiting time = time = 0
P turnaround time = 0;
if (P.Priority == 0) {
System_queue [System_n++] = P;
} else {
user_queue [user_n++] = P;
}
}
FCFS ( System_queue , System_n);
FCFS (user_queue , user_n);
```

```
int time = 0;
int s=0, u = 0;
while (s< System_n || u< user_n){

if(System_queue[s]. arrival_time <= time){
if(user_queue[u]. arrival_time <= time &&
user_queue[u]. arrival_time < System_queue[s]
. arrival time){

user_queue[u]. waiting_time = time- user_queue[u]
. arrival_time ; &
time + = user_queue[u]. burst_time ;
user_queue[u]. turnaround_time = user_queue[u]
. waiting_time + user_queue[u]. burst_time;

avg_waiting_time + = user_queue[u]. waiting_time;
avg_turnaround_time + = user_queue[u]. turnaround
_time ;
u++;
}
else {
System_queue[s]. waiting_time = time - System_
queue[s]. arrival_time;
time + = System_queue[s]. burst_time;
System_queue[s]. turnaround_time =
System_queue[s]. waiting_time + System_queue[s]
. burst_time ;
avg_waiting_time + = System_queue[s]. waiting_time;
avg_turnaround_time + = System_queue[s].
turnaround_time ;
```



```
s++ ;
}
}
else if(user_queue[u]. arrival_time <= time){
user_queue[u]. waiting_time = time - user_queue[u]
. arrival time;

time + = user_queue[u]. burst time;
user_queue[u]. turnaround_time = user_queue[u].
waiting_time + user_queue[u]. burst_time;

avg_waiting_time + = user_queue[u]. waiting_time;
avg_turnaround_time + = user_queue[u]. turnaround_tim
u++;
}
else {
if(System_queue[s]. arrival_time <= user_queue[u].
arrival_time) {
time = System_queue[s]. arrival_time ;
}
else {

time = user_queue[u]. arrival_time;
}
}
}

avg_waiting_time /= n;
avg_turnaround_time /= n;

Printf("PID | Burst time | Priority | Queue Type | waiting
Time | Turnaround time | n");
```

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

Printf("%-2d\t%-2d\t%d\t%d\t%d\t%d\t System\t%d\t%-2d\t%-2d\n",
, System_queue[i].Pid, System_queue[i]. burst_
time, System_queue[i].priority, System_queue[i].
waiting_time, System_queue[i]. turnaround_
time);

}
For (i=0 ; i< user_n ;i++){

Printf("%-2d\t%-2d\t user\t%d\t%-2d\n", user_queue
[i].Pid, user_queue[i]. burst_time,
user_queue[i].priority, user_queue[i].
waiting_time, user_queue[i].turnaround
_time);

}
Printf("Average waiting Time: %.2f\n" avg_waiting
_time);

Printf("Average Turnaround Time: %.f ,avg_
turnaround_time);

Free (System_queue);
free (System_queue);

return 0;

}
```


OUTPUT

Enter the number of process : 2.

Enter arrival time, burst time, and priority of for process 2 : 6
3
12

PID	Burst time	Priority	Queue type	waiting	Turnaround
1	3	9	user	12779854	12779857
2	3	12	user	12779855	12779860

Average waiting time : 12779854.00

Average Turnaround time : 19169788.00

Date / /
Page 18

9. write a C Program to Simulate Real time CPU Scheduling algorithms:

(a) Rate - Monotonic
(b) Earliest - deadline First

Sol^n

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

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].
remain_deadline [Max_Process];

void get_Process_info (int Selected_algo)
{
Printf("Enter total number of processes (maximum :d):",
        MAX_PROCESS);

Scanf ("%d", & num_of_Process);
if (num_of_process < 1)
{
```

Date / /
Page 19

```
exit (0);
}
for (int i = 0; i < num_of_Process; i++)
{
Printf ("\n Process %d :\n", i+1); 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 [i]);
}
}
}
```

Date __/__/__
Page 20

```
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 < cycles ; i++)
{
if (i < 10)
   Printf ("| 0 %d", i);
 else
```

Date __/__/__
Page 21

```
Printf ("| %d", i);
}
Printf ("|\n");
for (int i=0 ; i < num_of_process ; i++)
{
Printf ("P[%d]: ", i+1);
for (int j = 0; j< cycles ; j++);
{
if(process_list[j] == i+1)
Printf ("|###");
else
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 ("\n Given Problem is not Schedulable under the
      Said Scheduling algorithm.\n");
}
```

```
}
for(int i=0; i< time ; i++)
{
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_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
}
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 i=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_time [j];
execution_time [j] = execution_time [i];
execution_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_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)) {

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 \n2. Earliest Deadline
first \n3. Proportional Scheduling \n\n
Enter your choice :");

Scanf("%d", &option);
switch(option)
{
```

```
case 1: get_process_info (option);
        observation_time = get_observation_time
        (option);
        rate_monotonic (observation_time);
        break;

case 2: get_process_info option.
        observation_time = get_observation_time (option)
        earliest_deadline_first (observation_time);
        break;

case 3:- exit (0);

default: Printf ("\n Invalid Statement");

}
}
return 0;
}
```


OUTPUT

Enter the Number of processes : 2

Enter arrival time, burst time, and priority.
for process 2 :- 6
5
12

PID	Burst time	Priority	Queue Type	waiting	Turnaround
1	5	3	usr	12779854	12779915
2	5	12	usr	12779855	12779916

Average waiting time : 12779854.00

Average Around Time : 19169788.00

OUTPUT

1. Rate Monotonic
2. Earliest Dead line first
3. Proportional Scheduling.

Enter your choice : 1

Enter Total Number of processes (maximum 10) : 2

P

Pf.0

Date 1/1
Page 28

Process 1:-
⇒ Execution time : 2
⇒ Period : 3.

Process 2:-
⇒ Execution time : 5
⇒ Period 6 : 6

Scheduling.

Time :	0 0	01	02	03	04	05
P[1]:	####	####		####	####	
P[2]:			####			####

5/6/24

12/6

Date 1/1
Page 29

Q. write a c program to simulate producer - consumer problem using semaphore.

```
# 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("\n Enter your choice :");
Scanf("%d", &n);
Switch (n)

{
case1: if((mutex == 1) && (empty != 0))
        producer();
        else
        Printf(" Buffer is full !! =0))
        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("\n consumer consumes item %d", x);
x--;
mutex = signal(mutex);
}
```

OUTPUT

1. Producer
2. consumer
3. Exit.

Enter your choice = 1

Producer Produces the item 1
Enter your choice 1

Producer Produces the item 2
Enter your choice = 1

Producer produces the item
Enter your choice : 1
Buffer is full!!
Enter your choice : 2

Consumer consumes item : 3
Enter your choice : 2

Consumer consumes item 2
Enter your choice : 2

Consumer consumes item 1
Enter your choice : 2
Buffer is empty!!

9. write a C Program to simulate the concept of Dining Philosophers Problem.

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

#define N5
#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("Philosophers %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+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[NT];

Sem_unl_init(&mutex, 0, 1);
for(i=0; i<N; i++)
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

```
Phil 1    is thinking
Phil 2
Phil 3
Phil 4
Phil 5
   6
Phil 1 .. 1   Hungry
   2
   3
   4
   5
```


write a c program to simulate Bankers algorithm for the purpose of deadlock avoidance.

```
#include <stdio.h>

int main()
int n, m, j, k;
Printf("Enter the number of process:");
Scanf("%d", &n);
Printf("Enter the number of resources:");
Scanf("%d", &m);

int allocation[n][m];
Print("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, j++)
{
Scanf("%d", &max[i][j];
}
}
```

```
Scanf("%d", &available);
}
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++)
{
if (f[i]==0)
{
int flag = 0;
for (j=0, j<m; j++)
{
if (need[i][j] > available[j])
{
flag = 1;
break;
}
}
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%d ->", ans[i]);
}
Printf("P%d \n", ans[n-1]);
}
return(0)
}
```

Date
Page 40

OUTPUT

Enter the no. of processes : 5
Enter the no. of resources : 3
Enter the allocation matrix : 7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

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

P0 → P1 → P2 → P3 → P4

3/7/24

Date
Page 41

9. write a C program to simulate the following 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 Schemes\n");

Printf("\n Enter the number of blocks:");
Scanf("%d", &nb);

Printf("Enter the number of files:");
Scanf("%d", &nf);

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

```
Printf ("File no.%d :", i);
Scanf ("%d", &f[i]);
}
Printf ("\n Memory Management Scheme - First Fit");
first fit (b, nb, f, nf);

Printf ("\n\n Memory Management Scheme - worst - Fit");
worstFit (b, nb, f, nf);

Printf ("\n\n Memory Management Scheme - Best - Fit");
best Fit (b, nb, f, nf);

return 0;
}
void first fit (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 [j] = 1;
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", i, f[i], ff[i], b[ff[i]], frag[i]);
}
}
void worst Fit (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];

if(temp >= 0 && lowest > temp)
{
ff[i] = j;
lowest = temp;
}
}
}
```



```
frag[i] = lowest
bf[ff[i]] = 1;
lowest = 10000;

}
Printf("\n File no\t File size\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", i, f[i], ff[i],
      b[ff[i]], frag[i]);
}
}
```

OUTPUT

```
Enter the no. of blocks = 2
Enter the no. of files = 2.

Enter the size of the block :

Block 1 : 50
Block 2 : 60

Enter the Size of the files.

File 1 : 24
File 2 : 56
```

Date 4/6
Page

First Fit.

File no.	Filesize	Blockno.	Blocksiz	Fragment
1	24	0	30	26
2	36	1	60	0

Memory management Scheme - worst fit.

File	Filesiz	Blockno	Blocksiz	Fragment
1	24	0	60	36
2	60	1	50	0

Date 4/7
Page

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 pg fault cnt = 0;

void get data()
{
Printf("\n 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("\n Enter no. of frames:");
Scanf("%d", &f);
}

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

```
int ishit ( int data)
{
hit = 0;
for ( j = 0; j < f ; i++)
{
if (P[j] == data)
{
 hit = 1;
 break ;
}}
return hit ;
}
int gethit index ( int data)
{
int hitind ;
for ( k = 0; k < f ; k++)
{
if (P[k] == data)
{
hitind = k ;
break ;
}}
return hitind ;
}
void dish Pages ()
{
for ( k = 0; k < f ; k++)
{
```



```
if (P[k] != 9999)
Printf ("%d", P[k]);
}}
void disp PgFaultcnt ()
{
Printf("\n Total no of Page faults :%d",
       Pg Faultcnt);
}

void fifo()
{
 get data ();
 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[ j ];
Pg faultcnt ++;
 Dish Pages()
}
else
Printf(" No Page Fault");
}
```

```
void optimal()
{
initialize();
int near[50];
for(i=0;i<n;i++)
{
Printf("\n For %d :",in[i]);

if(isHit(in[j])==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];
Pg Faultcnt++;

disp Pages();
}
else
Printf("NO Page Fault");
}
disp Pg Fault cnt();
}
void lru()
{
initialize();

int least[50];
for(i=0;i<n;i++)
{
Printf("\n For %d :",in[i]);
if(isHit(in[j]==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 refindex;
for(j=0; j<nf; j++)
{
if(least[j]<min)
{
min=least[j];
refindex=j;
}}
pt[refindex]=in[i];

Pg Fault cnt++;
display Pages();
}
else
Printf("No Page Fault!")
}.
```



```
}
display Pg Fault Cnt();
}
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: get data();
        break;
case 2: FiFO();
        break;
case 3: optimal();
        break;
case 4: lru()
        break;
default: return 0
    break.
}
}
}
```

Date / / Page 54

OUTPUT

Page replacement Algorithms.

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

Enter your choice : 2.

Enter length of Page reference Sequence : 4

Enter the page reference Sequence : 5
2
3
5

Enter no. of Frames : 2

For 5 : 5
For 2 : 5 2
For 3 : 8 2 3
For 5 : 3 5

3/7/24

10/7/24

Date / / Page 55

write a C program to simulate Disk Scheduling algorithms

(a) FCFS
(b) SCAN
(c) C-SCAN

(a) FCFS:

```
#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 requests Sequences\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);
return 0;
}.
```

OUTPUT

Enter the number of requests : 2

Enter the request sequence : 5
6

Enter the initial head position : 3

Total head moment is : 3

(b) SCAN

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
int RQ[100], i, j, n, Total Head Moment + abs(RQ[i] - initial)
initial = RQ[i];
}
Printf("Total head moment is -> %d , Total Head moment)

Printf("Enter the number 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 \n");
Scanf("%d", &initial);
Printf("Enter total disk size \n")
Scanf("%d", &size);
Printf("Enter the head movement direction
        for high 1 and 0 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 (initial < RQ[i])
{
index = i;
break;
}}
if (move == 1)
{
for (i = index; i<n; i++)
{
Total Head Moment = Total Head Moment + abs(RQ[i] - init
initial = RQ[i];
}
Total Head Moment = Total Head Moment + abs(size - RQ[i-1] -1);
initial = size-1;
for (i = index-1; i>=0; i--)
{
Total Head Moment = Total Head Moment + abs(RQ[i] - initial);
initial = RQ[i];
}
}
```



```
else
{
for (i = index-1; i>=0; i--)
{
Total Head Moment = Total Head Moment + abs(RQ[i] - initial);
initial = RQ[i];

}
Total Head Moment = Total Head Moment + abs(RQ[i] - initial);
initial = 0;
for (i = index; i<n; i++)
{

Total Head Moment = Total Head Moment + abs(RQ[i] - initial);
initial = RQ[i];
}
}
Printf("Total head movement is %d", Total Head moment);
return 0;
}
```

OUTPUT

```
Enter the number of requests : 5

Enter the request sequence
43
98
12
85
47
```

Enter initial head position : 21

Enter total disk size : 65

Enter the head movement direction for high 1 and for low 0

21

Total head moment is : 119.

(C) C- SCAN

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
int RQ[100], i, j, n, total, initial, size, move;
Printf("Enter the number 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\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\n");
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 (more == 1)
{
for(i = index ; i< n ; i++)
{
Total Head Moment = Total Head Moment + abs(RQ[i]- initial
initial= RQ[i];
}
Total Head Moment = Total Head Moment + abs(Size-RQ[i-1]-1);
Total Head Moment= Total Head Moment + abs(Size-1-0);
initial = 0;
for(i=0 ; i< index ; i++)
{
Total Head Moment = Total Head Moment +abs(RQ[i]- initial);
initial = RQ[i];
}}
else
{
for(i = index-1 ; i>=0 ; i--)
{
```



```
Total Head Moment = Total Head Moment + abs(RQ[i] - initial)
initial=RQ[i];
}
Total Head Moment = Total Head Moment + abs(RQ[i+1]-0);
Total Head Moment = Total Head Moment + abs(Size-1-0);
initial = Size-1;
for(i=n-1 ; i>= index ; i--)
{
Total Head Moment = Total Head Moment + abs(RQ[i]-initial
initial = RQ[i];
}
}
Print f("Total head Movement + abs(RQ[i] - is :%d
,Total head Movement);
return 0;
```

OUTPUT

```
Enter the Number of Requests : 4
Enter the Request Sequences :
82
45
50
82
Enter the initial head position : 52
```

Enter total disk size : 30

Enter the head moment direction for high 1 and for low 0 : 1

Total head moment is : 168

Sneha
10/7/24