OS

Date : 8 - 5 - 24

Q Develop a c program to do addition, subtraction & multiplication of two matrices.

=>
```
#include <stdio.h>
int main()
{ int a[3][3], b[3][3], c[3][3],
        d[3][3], e[3][3];
  int i, j, m, n, p, q, k;
printf("Enter the rows and columns
      of matrix A");
scanf("%d %d", &m, &n);
printf("Enter the rows and columns
      of matrix B");
scanf("%d %d", &p, &q);
if ( m != p || n != q)
{ printf(" Addition and Subtraction
      not possible"); }
else if (n != p)
{ printf(" Multiplication is not
      possible"); }
else { printf("Enter the values of
      matrix A");
for(i=0; i<m; i++)
  { for(j=0; j<n; j++)
      { scanf("%d", &a[i][j]);
      } }
```

```
printf(" Enter the values of matrix B");
    for(i=0 ; i<p; i++)
    { for(j=0; j<q; j++)
    { scanf("%d", &b[i][j]); } }

printf(" Addition and subtraction
        of two A & B matrices");
for(i=0; i<m ; i++)
    { for(j=0; j<n; j++)
      { c[i][j] = a[i][j] + b[i][j];
        d[i][j] = a[i][j] - b[i][j];
      printf("%d", &c[i][j]);
       printf(" %d", &d[i][j]);
        } }

printf(" Multiplication of A and
        B matrices");
for ( i=0; i<m ; i++)
    { for(j=0; j<q ; j++)
      { e[i][j]=0;
    for(k=0 ; k<n; k++)
      { e[i][j] = e[i][j] + a[i][k]
                * b[k][i] ; }
    printf("%d", &e[i][j]);
      } } }
    }
```

Output :

Enter rows & columns of matrix A : 2 2

Enter rows & columns of matrix B : 2 2

Enter the values of matrix A :

1 2

3 4

Enter the values of matrix B :

5 6

7 8

Addition and Subtraction of A & B matrices :

6 8

10 12

-4 -4

-4 -4

Multiplication of A & B matrices :

19 22

43 50

Sole
8/5/24

Date: 15-5-24

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

=>
```
#include <stdio.h>
int n, i, j, pos, temp, choice, total = 0;
int Burst_time[20], Arrival_time[20], Waiting_time[20],
Turn_around_time[20], process[20];
float avg_Turn_around_time = 0,
      avg_waiting_time = 0;
void FCFS() {
  int total_waiting_time = 0, ~~turn~~
    total_around_time = 0;
  int current_time = 0;
for(i=0; i<n-1; i++)
 { for(j=i+1; j<n; j++)
    { if(Arrival_time[i] > Arrival_time[j]){
        temp = Arrival_time[i];
      Arrival_time[i] = Arrival_time[j];
      Arrival_time[j] = temp;

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

```
        temp = process[i];
        process[i] = process[j];
        process[j] = temp; }
      } }
    waiting_time[0] = 0;
  current_time = Arrival_time[0] +
                 Burst_time[0];
  for( i=1; i<n; i++)
    { if (current_time < Arrival_time[i])
      { current_time = Arrival_time[i]; }
    Waiting_time[i] = current_time -
                      Arrival_time[i];
    current_time += Burst_time[i];
    total_waiting_time += Waiting_time[i]; }

printf("\n Process \t\t Arrival Time \t Burst
      time \t Waiting Time \t Turnaround
      time");
  for(i=0; i<n; i++)
    { Turn_around_time[i] = Burst_time[i]
                          + waiting_time[i];
  total_turnaround_time += Turn_around_time[i];
  printf("\n P[%d] \t\t %d \t\t %d \t\t %d \t\t %d",
      process[i], Arrival_time[i], Burst_time[i],
    waiting_time[i], Turn_around_time[i]; }
```

```
avg_waiting_time = (float) total_waiting_time/n;
avg_Turn_around_time = (float) total_turn_around_
                       time/n;
printf("\n Average Waiting Time : %.2f",
        avg_waiting_time);
printf("\n Average Turnaround time : %.2f",
        avg_Turn_around_time); }

void SJF(){
 int  total_waiting_time=0, total_turn_around_tim
                =0;
int completed=0, current_time=0, min_index;
  int is_completed[20]={0};
  while(completed!=n) {
      int min_burst_time=9999;
       min_index=-1;
  for(i=0; i<n; i++)
{ if(Arrival_time[i]<=current_time &&
                    is_completed[i]==0) {
    if(Burst_time[i]<min_burst_time){
      min_burst_time = Burst_time[i];
       min_index = i; }
 if( Burst_time[i] == min_burst_time ) {
     If( Arrival_time[i] < Arrival_time[min_index])
       min_burst_time = Burst_time[i];
        min_index=i;
        } }
       } }
```

```
if (min_index != -1) {
waiting_time[min_index] = current_time - Arrival_time
                          [min_index];
current_time += Burst_time[min_index];
Turn_around_time[min_index] = current_time -
                Arrival_time[min_index];
total_waiting_time += Turn_around_time[
                min_index];
is_completed[min_index] = 1;
completed ++; }
else{ current_time++; } }
printf("\n Process \t\t Arrival Time \t
Burst Time \t Waiting Time \t Turnaround
~~for~~ Time");
for(i=0; i<n; i++)
{ printf("\n P[%d] \t\t %d \t\t %d \t\t %d \t
\t %d", process[i], Arrival_time[i],
Burst_time[i], Waiting_time[i],
Turn_around_time[i]); }
avg_waiting_time = (float) total_waiting_time/n;
avg_Turn_around_time = (float) total_Turnaround_time/n;
printf("\n\n ~~Around~~ Average Waiting Time = %.2f",
avg_waiting_time);
printf("\n Average Turnaround Time = %.2f",
avg_Turn_around_time);
}
```

```
int main() {
 printf ("Enter the total number of
        processes");
 scanf (" %d", &n);
printf (" Enter Arrival Time and Burst Time");
  for(i=0; i<n; i++)
     { printf(" P[%d] Arrival Time", i+1);
       scanf(" %d", & Arrival_time[i]);
      printf("P[%d] Burst-time:[i", i+1);
       scanf(" %d", & Burst_time[i]);
        process[i] = i+1; }
while(1) {
  printf(" Main Menu");
  printf(" 1. FCFS");
  printf(" 2. SJF");
  printf(" Enter choice");
   scanf(" %d" &choice);
   switch(choice) {
      case 1 : FCFS();
               break;
     case 2 : SJF();
              break;
   default: printf(" Invalid");
        } }
       return 0; }
```

output:

Enter the total number of processes: 4.

Enter Arrival Time and Burst time:

P[1] Arrival Time: 0

P[1] Burst Time: 3

P[2] Arrival Time: 1

P[2] Burst Time: 6

P[3] Arrival Time: 4

P[3] Burst Time: 4

P[4] Arrival Time: 6

P[4] Burst Time: 2

Main Menu

1. FCFS

2. SJF

Enter your choice: 1

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

Average Waiting Time = 3.50

Average Turn around time = 7.25

Main Menu

1. FCFS

2. SJF

Enter your choice : 1

Process	Arrival time	Burst time	Waiting Time	Turnaround time
P[1]	0	3	0	3
P[2]	1	6	2	8
P[3]	4	4	7	11
P[4]	6	2	3	5

Average Waiting time = ~~6.75~~ 3.5

Average Turn around time = 7.25

15/5/24

Q. Write a C program to simulate CPU scheduling algo to find TAT & WT

(a) Priority (pre-emptive & non)

(b) Round Robin

```
#include<stdio.h>
#include<stdlib.h>
struct process{
        int process_id;
        int burst_time;
        int priority;
        int waiting_time;
        int turnaround_time;};
void find_avg_time(struct process[], int);
void priority_scheduling(struct process[], int);

void main(){
    int n, i;
    struct process proc[10];
    printf("Enter no. of processes");
    scanf("%d", &n);
    for(i=0; i<n; i++)
        { printf("Enter process %d");
          scanf("%d", &proc[i].process_id);
        printf("Enter burst time");
          scanf("%d", &proc[i].burst_time);
        printf("Enter priority");
        scanf("%d", &proc[i].priority); }
          priority_scheduling(proc, n); }
```

```
void find_waiting_time(struct process proc[],
    int i;                          int n, int wt[]) {
    wt[0] = 0;
    for (i=1; i<n; i++)
    {   wt[i] = proc[i-1].burst_time + wt[i-1];
        } }
void find_turn_around_time( struct process proc[],
                            int n, int wt[],
                            int tat[]){
    int i;
    for(i=0; i<n; i++)
    {   tat[i] = proc[i].burst_time + wt[i];
    } }
void find_avg_time(struct process proc[], int n){
    int wt[10], tat[10], total_wt=0, total_tat=0;
    find_waiting_time(proc, n, wt);
    find_turnaround_time(proc, n, wt, tat);
    printf(" Process ID \t Burst time \t priori
        \t Waiting time \t Turnaround time");
    for(i=0; i<n; i++)
    {   pos=i;
        for(j=i+1; j<n; j++){
            if(proc[j].priority < proc[pos].priority)
                pos=j;
            }
        temp = proc[i];
        proc[i] = proc[pos];
        proc[pos] = temp; }
```

```
find_avg_time(proc, n); }
```

6) Round Robin (Non-preemptive)

```
#include <stdio.h>
#include <stdbool.h>
int findaroundtime(int procs[], 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 wt[n], tat[n], total_wt = 0,          int quantum)
        total_tat = 0;
  waiting_time (processes, n, bt, wt, tat);
printf("\n\n Processes \t\t Burst time \t\t
  waiting time \t\t Turnaround time \n");
    for (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 \t\t
    %d \n", i+1, bt[i], wt[i], tat[i]);}
printf("\n Avg time waiting = %f",
          (float) total_wt / (float)n);
printf("\n Avg turnaround time = %f",
          (float) total_tat / (float)n);
    return 1;}
void main(){
int n, processes[n], burst_time[n], q;
  printf("Enter no. of processes");
   scanf("%d", &n);
  printf("Enter quantum time");
   scanf("%d", &q);
    int i=0;
  for(i=0; i<n; i++){
   printf("\n Enter processes");
   scanf("%d", &processes[i]);
   printf("Enter burst time");
   scanf("%d", &burst time[i]));}
```

findavgtime(processes, n, burst time, q);
return 1; }

Output : a) Enter no. of processes : 5

Enter process ID : 1

Enter burst time : 4

Enter priority : 2

Enter process ID : 2

Enter burst time : 3

Enter priority : 3

PID : 3 PID : 4 PID : 5

BT : 1 BT : 5 BT : 2

P : 4 P : 5 P : 5

PID	BT	P	WT	TAT
1	4	2	0	4
2	3	3	4	7
3	1	4	7	8
4	5	5	8	13
5	2	5	13	15

Avg WT = 6.4

Avg TAT = 9.4

5/6/24

Q. Write a C program to simulate Real-time CPU scheduling algos:

a) Rate-Monotonic

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>
#define max_processes 10
typedef struct {
    int id;
    int burst_time;
    float priority; } Task;
int num_of_process;
int execution_time[max_processes], 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 processes (maximum %d)
        , max_processes);
    scanf("%d", &num_of_process);
    if (num_of_process <1)
        { exit(0); }
    for (int i=0; i<num_of_process; i++)
    { printf("Process %d", 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 set_obeservation_time(int selected_algo)
{ if(selected_algo==1){
    return max(period[0], period[1],
               period[2]); }
  else if(selected_algo==2){
    return max(deadline[0], deadline[1],
               deadline[2]); } }
void print_schedule(int process_list[],
     int cycles){
  printf("\n Scheduling \n");
  printf("time");
  for(int i=0; i<cycles; i++)
  { if(i<10){ printf("0%d", i); }
    else{ 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("|");}
    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("Given problem is not schedulable
    under the said scheduling algo");} }
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; }}}
```

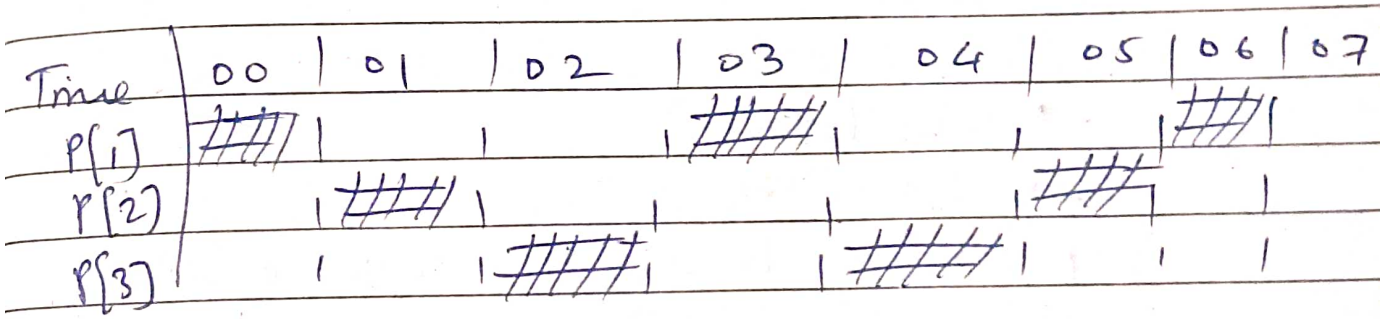
```
if (remain_time[next_process] > 0)
{ process_list[i] = next_process + 1;
  remain_time[next_process] -= 1; }
for (int k = 0; k < num_of_process; k++)
  { if ((i+1) % period[k] == 0)
  { remain_time[k] = execution_time[k];
      next_process = k; }}}
  print_schedule(process_list, time); }
void earliest_deadline_first(int time) {
  float utilization = 0;
  for (int i=0; i<num_of_process; i++)
  { utilization += (1.0 * execution_time[i]/
                    deadline[i]; }
  int n = number_of_processes;
  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];
} 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("1. Rate Monotonic \n 2.
                Earliest Deadline First \n
                3. Exit");
    switch(option) {
        case 1: get_process_info(option);
            observation_time = get_observation_time(option);
            rate(observation_time);
            break;
        case 2: get_process_info(option);
            observation_time = get_observation_time(option);
            rate_monotonic(observation_time);
            break;
        case 3: exit(0);
        default: printf("\n Invalid Statement");
    }
}
return 0;
}
```

Output : 1. Rate Monocotonic
2. Earliest Deadline First

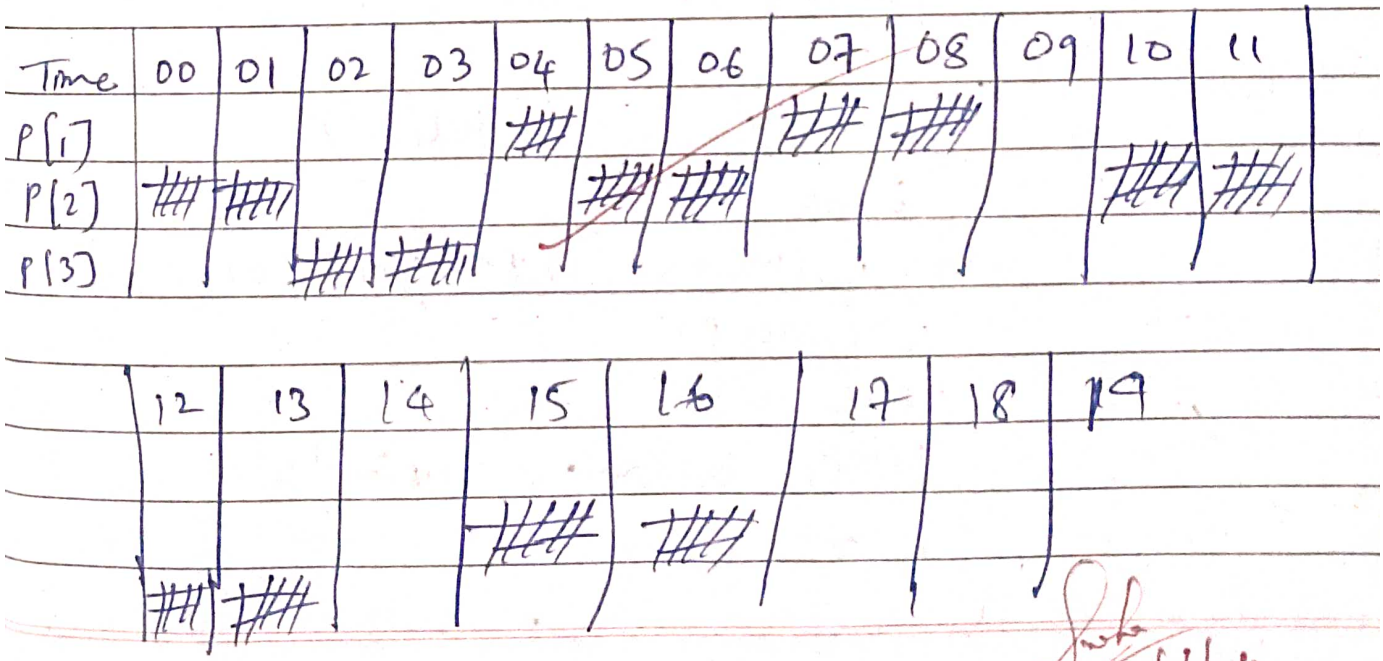
Enter your choice : 1
Enter total no. of process (max : 10) : 3

Process 1:	Process 2	Process 3:
Extime : 1	Extime : 1	Extime : 2
period : 3	period : 4	period : 8



Enter your choice : 2
Enter total no. of processes (max 10) : 3

Process 1:	Process 2:	Process 3:
Extime : 3	Extime : 2	Extime : 2
deadline : 20	deadline : 5	deadline : 10



5/6/24

Date : 12-6-26

Q. 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("\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;
```

Outp

```
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 : 2

~~Producer produces~~

Buffer is empty!!

Enter the choice : 1

Producer produces the item 1.

Enter the choice : 2

Consumer consumes the item 1

Enter your choice : 3

Q 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]={0,1,2,3,4,5};
sem_t mutex;
sem_t s[N];
void test(int i)
{   if(state[i]==HUNGRY && state[LEFT]
                    !=EATING &&
            state[RIGHT]!=EATING)
    { state[i]=EATING;
      sleep(2);
    printf("Philosopher %d takes fork %d
          and %d\n", i+1, LEFT+1, i+1);
    printf("Philosopher %d is Eating\n", i+1);
    sem_post(&s[i]); } }
void take_fork(int i){
    sem_wait(&mutex);
```

```
state[i] = HUNGRY;
printf(" Philosopher %d 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[N];
  sem_init(&mutex, 0, 1);
  for(i=0; i<N; i++) {
```

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

Output:

```
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 4 is hungry
Philosopher 3 is hungry
Philosopher 2 is hungry
Philosopher 5 is Hungry
Philosopher 1 is hungry
Philosopher 1 takes fork 5 and 1
:
:
:
```

12/6/24

Date: 19-6-24

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

⇒
```
#include<stdio.h>
#include<stdlib.h>
int main(){
int n, m, i, j, k;
printf("Enter the number of processes");
scanf("%d", &n);
printf("Enter the no. of resources");
scanf("%d", &m);
int **allocation =(int **)malloc(n*sizeof(int *));
int **max
int allocation[n][m];
printf("Enter the Allocation Matrix");
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");
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");
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; } }
```

```
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] = mav[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 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

4 3 3

Enter the Available Resources:

3 3 2

Following is the SAFE Sequence

$P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_0 \rightarrow P_2$

Q. Write a C program to simulate deadlock detection.

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

```
avail[j] = r->t - 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)
   { if for(j=0; j<nr; j++)
   { if (request[i][j] <= w[j])
        canbe processed = 1
      else {
        canbeprocured = 0;
        break; } }
   if (can be processed)
    { mark[i] = 1;
     for(j=0; j<nr; j++)
    w[j] + = alloc[i][j];
      } } }
int deadlock = 0;
for(i=0; i<np; i++)
```

```
if (marr[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 Resources R1 : 5

Total amount of Resource R2 : 3

Total amount of Resource R3 : 3

Enter the request matrix:

1 0 1

2 2 2

3 3 3

4 4 4

5 5 5

Enter the allocation matrix:

1 1 1

2 2 2

3 3 3

4 5 3

5 6 2

Deadlock detected

19/6/24

Date: 3-7-24

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

(a) Worst-fit (b) Best-fit (c) First-fit

```
#include<stdio.h>
#define max 25
void firstFit(int b[], int nb, int f[], int nf);
void worstFit(int b[], int nb, int f[], int nf);
void bestFit(int b[], int nb, int f[], int nf);
int main(){
    int b[max], f[max], nb, nf;
    printf("Memory Management schemes\n");
    printf("Enter the number of blocks:");
    scanf("%d", &nb);
    printf("Enter the number of files:");
    scanf("%d", &nf);
    printf("Enter the number of blocks:");
    scanf("%d", &nb);
    printf("Enter the size of the blocks\n:");
    for(int i=1; i<=nb; i++)
    {printf("%d", &b[i]);}
    printf("\nEnter the size of the files");
    for(int i=1; i<=nf; i++)
    {printf("File %d:", i);
    scanf("%d", &f[i]);}
    printf("\nMemory management scheme-
                First Fit");
    firstFit(b, nb, f, nf);
    printf("Memory Management Scheme-WorstFit");
```

```
worstfit(b, nb, f, nf);
printf("\n Memory Management Scheme - 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_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 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
    \t\t %d", i, f[i], ff[i], bf[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_no: \t
    Block size: \t Fragment");
        for (i=1; i<=nf && f[i]!=0; i++)
    {printf("\n %d \t %d \t \t %d \t \t %d \t \t
            %d", i, f[i], ff[i], b[ff[i]],
                frag[i]); }}
```

Output:

Memory management schemes

Enter the no. of blocks: 4

Enter the no. of files: 5

Enter the size of the blocks:

Block 1: 200

Block 2: 200

Block 3: 300

Block 4: 400

Enter the size of the files:

File 1: 124

File 2: 76

File 3: 84

File 4: 76

File 5: 128

Memory management scheme - First Fit

File_no	File_size	Block_no	Block_size	Fragment
1	124	1	200	76
2	76	2	200	124
3	84	3	300	216
4	76	4	400	324
5	128	1	200	0

Memory Management scheme - Worst Fit

File_no	File_size	Block_no	Block_size	Fragement
1	124	4	400	276
2	76	3	300	224
3	84	2	200	116
4	76	4	200	124
5	128	1	200	0

Memory management scheme - Best Fit

File_no	File_size	Block_no	Block_size	Fragment.
1	124	1	200	276
2	76	1	200	224
3	84	1	200	116
4	76	1	200	124
5	128	1	200	0

8. 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("\n Enter the length of page reference
                    sequence");
scanf("%d", &n);
printf("\n Enter the page reference sequence");
    for(i=0; i<n; i++)
    { scanf("%d", &in[i]); }
printf("Enter no. of frames");
  scanf("%d", &f); }
void initialize(){
pgfaultcnt = 0;
for(i=0; i<f; i++)
 { p[i]= 9999; }
int isHit(int data)
 { hit=0;
   for(j=0; j<f; j++)
    { if(p[j]=data)
      { hit = 1;
        break; } }
    return hit; }.
```

```
int getHitIndex(int data)
{ int hitind;
  for(k=0;k<f;k++)
    { if(p[k]==data)
      { hitind=k;
        break;}}
    return hitind;}
void dispPages(){
  for(k=0;k<f;k++)
  { if(p[k]!=9999)
      printf(" %d", p[k]);
  }}
void dispPgFault(cnt(){
printf("\n Total no of pages faults :%d",
      pgfault(cnt));}
void fifo(){
  getdata();
  Initialize();
  for(int 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 ");}
      dispPgFault(cnt();}
```

```
void optimal(){
    Initialize()
    int near[50];
    for(i=0; i<n; j++)
    { printf("\n For %d", in[i]);
      if (isHit[in[i]]==0){
       for(j=0; j<f; j++)
       { int pg = p[i];
         int found=0;
        for(k=i; k<n; k++)
        { if(pg==in[k])
          { near[j]=k;
            found=1;
            break; }
          else found=0; }
        if(!found)
          near[j]=9999; }
        int max=-9999;
        int repindex;
        for(j=0; j<nf; j++)
        { if(near[j]<max)
          { max=near[j];
            repindex=j; }}
        P[repindex]=in[i];
        pgfaultcnt++;
        dispPages(); }
      else printf("no page fault"); }
      dispPgFaultCnt(); }
```

```
void lru(){
 initialize();
 int least[50];
 for(i=0; i<n; i++)
 { printf("\n For %d", in[i]);
   if(checkHit(in[i])==0)
   { for(j=0; j<nf; j++){
     { int pg = p[j].pno;
       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"); }
```

```
dispPgFaultCnt();
}

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

output: Page Replacement Algorithms

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

Enter your choice: 2

Enter the length of page reference sequence : 12
Enter the page reference sequence:
1 2 3 4 1 2 5 1 2 3 4 5
Enter the number of frames : 3
For 1 : ~~Page~~ 1
For 2 : 1, 2
For 3 : 1 2 3
For 4 : 2 3 4
For 1 : 3 4 1
For 2 : 4 1 2
For 5 : 1 2 5
For 1 : No page Fault
For 2 : No Page Fault
For 3 : 2 5 3
For 4 : 8 5 3 4
For 5 : No page Fault
Total number of page Faults : 9
Page Replacement Algorithms.
1. Enter Data
2. FIFO
3. Optimal
4. LRU
5. Exit
Enter your choice : 3
For 1 : 1
For 2 : 1 2 3
For 3 : 1 2 3
For 4 : 1 2 4

For 1: No page Fault
For 2: No page Fault
For 3: 3 2 5
For 4: 4 2 5
For 5: No page Fault
Total number of page Faults: 7
& Page Replacement Algorithms
1. Enter data
2. FIFO
3. optimal
4. LRU
5. Exit
Enter your choice: 4
For 1: 1
For 2: 1 2
For 3: 1 2 3
For 4: 4 2 3
For 1: 4 1 3
For 2: 4 1 2
For 5: 5 1 2
For 1: No Page Fault
For 2: No page Fault
For 3: 3 1 2
For 4: 3 4 2
For 5: 3 4 5
Total number of page faults: 10

3/7/24

Date : 10-7-24

Q. Write a C program to simulate disk scheduling algorithm

(a) FCFS

→
```
#include<stdio.h>
#include<stdlib.h>
int main(){
    int RQ[100], i, n, TotalHeadMovement=0, initial;
    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);
    for(i=0; i<n; i++){
        TotalHeadMovement += abs(RQ[i] - initial);
        initial = RQ[i];
    }
    printf("Total Head movement is %d \n",
            TotalHeadMovement);
    return 0;
}
```

Output : Enter the number of Requests 5
Enter the Requests sequence
5 98 107 45 78
Enter initial head position 45
Total Head Movement is 237

(b) SCAN

```
#include<stdio.h>
#include<stdlib.h>
int main(){
int RQ[100], i, n, j, TotalHeadMovement=0, 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; j<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 (move == 1)
{  for(i = index ; i<n; i++)
   { TotalHeadMoment = TotalHeadMoment +
                       abs(RQ[i] - initial);
initial = RQ[i]; }
TotalHeadMoment = TOTALNAD HeadMoment + abs
                  (size - RQ[i-1] - 1);
initial = size - 1;
for( i = index - 1; i >= 0; i--)
{  TotalHeadMoment = TotalHead Moment +
                     abs(RQ[i] - initial);
   initial = RQ[i];
} }
else { for(i = index - 1; i >= 0; i--)
     { TotalHeadMoment = TotalHead Moment
                + abs(RQ[i] - initial);
       initial = RQ[i];
     }
  TotalHeadMoment = TotalHead moment +
                    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
8
Enter the Requests sequence
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 359
```

(c) C - SCAN:

```
#include<stdio.h>
#include<stdlib.h>
int main(){
    int RQ[100], i, j, n, TotalHeadMoment=0,
        initial, size, move;
    printf("Enter the number of Request\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; 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 = Total Head Moment
                    + abs ( RQ [i] - intial ) ;
      initial = RQ [i] ;
    }
  TotalHeadMoment = TOTAL HeadMoment + (size -
                                    RQ [i-1] - 1));
  TotalHeadMoment = TotalHead Moment +
                    abs (size - 1 - 0) ;
  initial = 0 ;
  for ( i = 0 ; i < index ; i++)
    { TotalHeadMoment = TotalHead Moment +
                    abs ( RQ [i] - initial ) ;
      initial = RQ [i] ; }
  }
else { for ( i = index - 1 ; i >= 0 ; i--)
    { TotalHead Moment = TotalHead Moment +
                    abs ( RQ [i] - initial ) ;
      initial = RQ [i] ;
    }
  TotalHeadMoment = TotalMoment Head +
                    abs ( RQ [i+1] - 0 ) ;
  TotalHeadMoment = TotalHead Moment +
                    abs ( size - 1 - 0 ) ;
  initial = size - 1 ;
```

```
for (i=n-1 ; i>=index ; i--)
{ TotalHeadMoment = TotalHeadMoment +
                    abs(RQ[i]-initial);
initial = RQ[i];
} }
printf("Total Head Movement is %d",
        TotalHead Movement );
return 0;
}
```

Output:

```
Enter the number of requests
5
Enter the requests sequence
98 183 37 122 14
Enter the 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 359
```

10/12/24