8/5/2024

Addition, subtraction and multiplication of matrix:

i/p

```
#include <stdio.h>
#define RS 3
#define CS 3
void add (int A[RS][CS], int B[RS][CS], int C[RS][CS])
{
  for (int i = 0; i<RS; i++){
      for (int j = 0; j<CS; j++){
        C[i][j] = A[i][j] + B[i][j];
      }
  }
}
void sub (int B[RS][CS], int B[RS][CS], int C[RS][CS]){
  for (int i=0; i<RS; i++) {
      for (int j=0; j<CS; j++){
        C[i][j] = A[i][j] - B[i][j];
  }
}
}
void mul (int A[RS][CS], int B[RS][CS],
          int C[RS][CS]){
    for (int i = 0; i<RS; i++){
```

```
        for (int j = 0; j < CS; j++ ){
            C[j][j] = 0;
            for (int k = 0; k < CS; k++){
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
}
void dm (int matrix[RS][CS]){
    for(int i=0; i<RS; i++){
        for(int j=0; j<CS; j++){
            printf("%d\t", matrix[i][j]);
        }
        printf("\n");
    }
}
int main(){
    int A[RS][CS] = {{1,2,3},{4,5,6},{7,8,9}};
    int B[RS][CS] = {{9,8,7},{6,5,4},{3,2,1}};
    int C[RS][CS];
```

```
printf ("MA:\n");
dm (A);
printf ("MB:\n");
dm(B);
add (A, B, C);
printf ("\n Addition C:\n");
dm(C);
sub (A, B, C);
printf ("\n subtraction C:\n");
dm (C);
mul (A, B, C);
printf ("\n Multiplication C:\n");
dm (C);
return 0;
}
```

O/P

MA:

1 2 3

4 5 6

7 8 9

MB:

9 8 7

6 5 4

3 2 1

addition:

10	10	10
10	10	10
10	10	10

subtraction:

-8	-6	-4
-2	0	2
4	6	8

Multiplication:

30	24	18
84	69	54
118	114	90

Scale 8|5|24

15/5/24

WAP to simulate the following non-preemptive CPU scheduling algorithm to find turnaround time and waiting time.

1) FCFS
2) SJF.

```
#include <stdio.h>
int n, i, j, pos, temp, choice, BT[20], WT[20], TT[20],
    process[20], total = 0;
float aTT = 0, aWT = 0;
int FCFS(){
    WT[0] = 0;
    for(i = 1; i<n; i++){
        WT[i] = 0;
        for(j = 0; j<i; j++)
            WT[i] += BT[j];
    }
    printf("\n Process \t Burst Time \t Waiting Time
\t Turnaround Time");
    for(i=0; i<n; i++){
        TT[i] = BT[i] + WT[i];
        aWT += WT[i];
        aTT += TT[i];
        printf("\n P[%d] \t %d \t\t %d \t\t %d",
        i+1, BT[i], WT[i], TT[i]);
}
```

```
    avwt = (float)(avwt)/(float)i;
    attt = (float)(attt)/(float)i;
    printf("\n Avg. Waiting Time: %.2f", avwt);
    printf("\n Avg. Turnaround Time: %.2f\n", attt);
    return 0;
}
int SJF(){
    for(i=0; i<n; i++){
        pos = i;
        for(j=i+1; j<n; j++){
            if(BT[j] < BT[pos])
                pos=j;
        }
        temp = BT[i];
        BT[i] = BT[pos];
        BT[pos] = temp;
        temp = process[i];
        process[i] = process[pos];
        process[pos] = temp;
    }
    WT[0] = 0;
    for(i=1; i<n; i++){
        WT[i] = 0;
        for(j=0; j<i; j++);
        WT[i] = 0;
        for(j=0; j<i; j++)
            WT[i] += BT[j];
```

```
    total += WT[i];
  }
aWT = (float) total / n;
total = 0;
printf("\n Process \t Burst Time \t Waiting Time
  \t Turnaround Time");
for(i=0; i<n; i++){
  TT[i] = BT[i] + WT[i];
  total += TT[i];
  printf("\n P[%d] \t %d \t %d \t %d", process[i], BT[i], WT[i], TT[i]);
}
aTT = (float) total / n;
printf("\n\n Average Waiting Time = %f", aWT);
printf("\n Average Turnaround Time = %f\n", aTT);
return 0;
}
int main(){
  printf("Enter the total number of process:");
  scanf("%d", &n);
  if(n<=0){
    printf("Invalid input! Number of process
    must be greater than zero.\n");
    return 1;
  }
  printf("\n Enter Burst Time:\n");
  for(i=0; i<n; i++){
```

```
        printf("P[%d]: ", i+1);
        scanf("%d", &BT[i]);
        if(BT[i]<0){
            printf("Invalid input! Burst time cannot
            be negative.\n");
            return 1;
        }
    }
    process[i]=i+1;
}
while(1){
    printf("1. FCFS\n2. SJF\n3. Exit.\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    switch(choice){
        case 1: FCFS();
                break;
        case 2: SJF();
                break;
        case 3: printf("BYE BYE\n");
                return 0;
        default: printf("Invalid Input!\n");
                break;
    }
}
return 0;
}
```

P

nter the total number of process: 3

nter burst time:

P[1] : 2.

P[2] : 1.

P[3] : 3.

. FCFS.

.. SPF.

3. EXIT.

Enter your choice : 1.

process under Time	Burst time	waiting Time	Turnaround Time
P[1]	2	0	2
P[2]	1	2	3
P[3]	3	3	6

Avg wait time : 1.67.

Avg Turnaround Time : 3.67.

Enter choice : 2

PT	BT	WT	TT
P[2]	1	0	1
P[1]	2	1	3
P[3]	3	3	6

Avg wait time : 1.333

Avg Turnaround time : 5.333.

Enter choice: 3

BYE BYE

15/5/24

22/5/24

NAP to simulate the following CPU scheduling algorithm to find turnaround time and waiting time.

a) Priority (Non pre emptive)

i/p

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

struct process {
    int p_id;
    int b_t;
    int pri;
    int wait;
    int turn;
    int ad;
};
void fwt(struct process proc[], int n, int wt){
    wt[0] = 0;
    for(int i = 1; i<n; i++){
        wt[i] = proc[i-1].b_t + wt[i-1];
    }
}
void ftt(struct process proc[], int n, int wt[],
         int tat[]){
    for(int i=0; i<n; i++){
        tat[i] = proc[i].b_t + wt[i];
    }
}
```

```
void fcfs(struct process proc[], int n) {
    int wt[10], tat[10], total_wt = 0, total_tat = 0;
    fwt(proc, n, wt);
    ftt(proc, n, wt, tat);
    ~~printf("\n %d\t~~
    printf("\n Process ID \t Burst time \t Priority
    Time \t Arrival time \t Turnaround Time");
    for(int i = 0; i < n; i++){
        twt = twt + wt[i];
        ttat = ttat + tat[i];
        printf("\n %d\t\t %d\t\t %d\t\t %d\t\t %d\t\t
        %d\t\t %d", proc[i].pid, proc[i].
        b_t, proc[i].pri, proc[i].at, wt[i],
        tat[i]);
    }
    printf("\n\n Avg time = %f", (float)twt/n);
    printf("\n Avg tt = %f\n", (float)ttat/n);
}
void ps(struct process proc[], int n){
    struct process temp;
    for(int i = 0; i < n; i++){
        int pos = i;
        for(int j = i+1; j < n; j++){
            if(proc[j].pri < proc[pos].pri ||
    (proc[j].pri == proc[pos].pri && proc[j].
    at < proc[pos].at))
```

```
            pos = j;
        }
        temp = proc[i];
        proc[i] = proc[pos];
        proc[pos] = temp;
    }
    ft_at(proc, n);
}
int main(){
    int n;
    struct pro proc[10];
    printf("Enter process ids: ");
    scanf("%d", &n);
    for(int i=0; i<n; i++){
        printf("\n Enter process ID: ");
        scanf("%d", &proc[i].pid);
        printf("Enter priority: ");
        scanf("%d", &proc[i].proc);
        printf("Enter burst time: ");
        scanf("%d", &proc[i].b_t);
        printf("Enter arrival: ");
        scanf("%d", &proc[i].at);
    }
    pri_sche(proc, n);
    return 0;
}
```

o/p

number process : 3

pid : 1
bt : 4

p : 2
at : 1

pid : 2
bt : 3
p : 3
at : 1

pid : 3
bt : 2
p : 4
at : 2

pid	bt	p	at	wt	tt
1	4	2	0	0	4
2	3	3	1	4	7
3	2	4	2	7	9

Avg wt = 3.667..
Avg tt = 6.667..

b) Round Robin (Non pre emptive)

i/p

```
#include <stdio.h>
#include <stdbool.h>
int tt (int process[], 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 process[], 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 fcfs(int processes[], int n, int bt[], int quantum)
{
    int wt[n], total_wt = 0, total_tat = 0;
    wt(processes, n, bt, wt, quantum);
    tt(processes, n, bt, wt, tat);
    for(int i=0; i<n; i++){
        twt = twt + wt[i];
        ttat = ttat + tat[i];
    }
    return 1;
}
int main(){
    int n, quantum;
    printf("Enter process:");
    scanf("%d", &n);
```

```
printf("Enter quantum:");
scanf("%d", &quantum);
int proc[n], bt[n];
for(int i = 0; i<n; i++){
    printf("\n Enter process:");
    scanf("%d", &proc[i]);
    printf("Enter bt:");
    scanf("%d", &bt[i]);
}
at(processes, n, bt, quantum);
return 0;
}
```

o/p

Enter process no: 5
Enter quantum: 2
Enter bt 1: 5
Enter bt 2: 3
Enter bt 3: 1
Enter bt 4: 2
Enter bt 5: 3

Processes	bt	wt	tt
1	5	9	14
2	3	9	12
3	1	4	5
4	2	5	7
5	3	10	13

Avg wat = 7.4

Avg tt = 10.2

5/6/24

WAP to simulate Real time CPU scheduling.

a) Rate Monotonic.

b) Earliest deadline First.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>
#define maxpro 10
typedef struct {
    int id;
    int bt;
    float pri;
} Task;
int npro;
int et[maxpro], pr[maxpro], rt[maxpro],
    dld[maxpro], rd[maxpro];
void getpro(int sched) {
    printf("\nEnter no. of pros. (max %d): ", maxpro);
    scanf("%d", &npro);
    if (npro < 1) {
        exit(0);
    }
    for (int i=0; i<npro; i++) {
        printf("\n pro %d:\n", i+1);
        printf("==> Et: ");
```

```
        scanf("%d", &get[i]);
        out[i] = get[i];
    }
}

int max(int a, b, 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(int salgo){
    if(salgo==1){
        return max(per[0], per[1], per[2]);
    } else if (salgo == 2){
        return max(ded[0], ded[1], ded[2]);
    }
}
void ps(int p[], int ayc){
    printf("\n\n Selecting: \n\n");
    printf("Time: ");
}
```

```
printf("\n");
for(int i=0; i<npro; i++){
  printf("P[%d]: ", i+1);
  for(int j=0; j<qyc; j++){
    if(p[j]==i+1)
      printf("***");
    else
      printf("   ");
  }
}
}
void fcfs(int tm){
  int pl[100]={0}, min=999, np=0;
  float uti=0;
  for(int i=0; i<npro; i++){
    uti+=(1.0*et[i])/per[i];
  }
  int n=npro;
  int m=(float)(n*(pow(2, 1.0/n)-1));
  if(uti>m){
    printf("\nGiven pro is not scheduling.\n");
  }
  for(int i=0; i<tm; i++){
    min=1000;
    for(int j=0; j<npro; j++){
      if(et[j]>0){
        if(min>per[j]){
```

```
            min = per[j]
            mp = j;
        }
    }
}
}
void edf (int tm){
    float alti = 0;
    for (int i = 0 ; i < npro ; i++){
    }
int n = npro;
int pro[npro];
int mxd, cp = 0, md, pl[tm];
bool ter[npro];
for (int i = 1; i <= npro ; i++){
    if (ded[i] > mxd)
        mxd = ded[i];
}
for (int i = 0; j < npro ; i++){
    for (int j = i+1 ; j < npro ; j++){
        if (ded[j] < ded[i]){
            int temp = et[j];
            et[j] = et[i];
            et[i] = temp
            temp = ded[j];
            ded[i] = temp
            temp = pro[j]
            pro[i] = temp;
```

```
for(int t=0; t<tm; t++){
    if(ap!=-1){
        --et[ap];
        pl[t]=pro[ap];
    } else
        pl[t]=0;
    for(int i=0; i<npro; i++){
        --ded[i];
        if((et[i]==0)&&ir[i]){
            ded[i]+=red[i];
            ir[i]=false;
        }
    }
}
int main(){
    int op;
    int at;
    while(1){
        printf("1.RM \n2.EDF \n3.PS\n:");
        scanf("%d",&op);
        switch(op){
            case 1:
                getpro(op);
                at=get(op);
                sm(at);
                break;
```

```
case 2:
    getpro(cp);
    ot = get(cp);
    edf(ot);
    break;
case 3:
    exit(0);
default:
    printf("\n Invalid");
}
}
return 0;
}
```

o/p

1. Rate Monotonic
2. Edf
3. preprocessional scheduling.

Enter choice: 2

Enter total process : 3.

P1 :

==> Enter Execution time: 3

==> Dead time : 20

P2 :

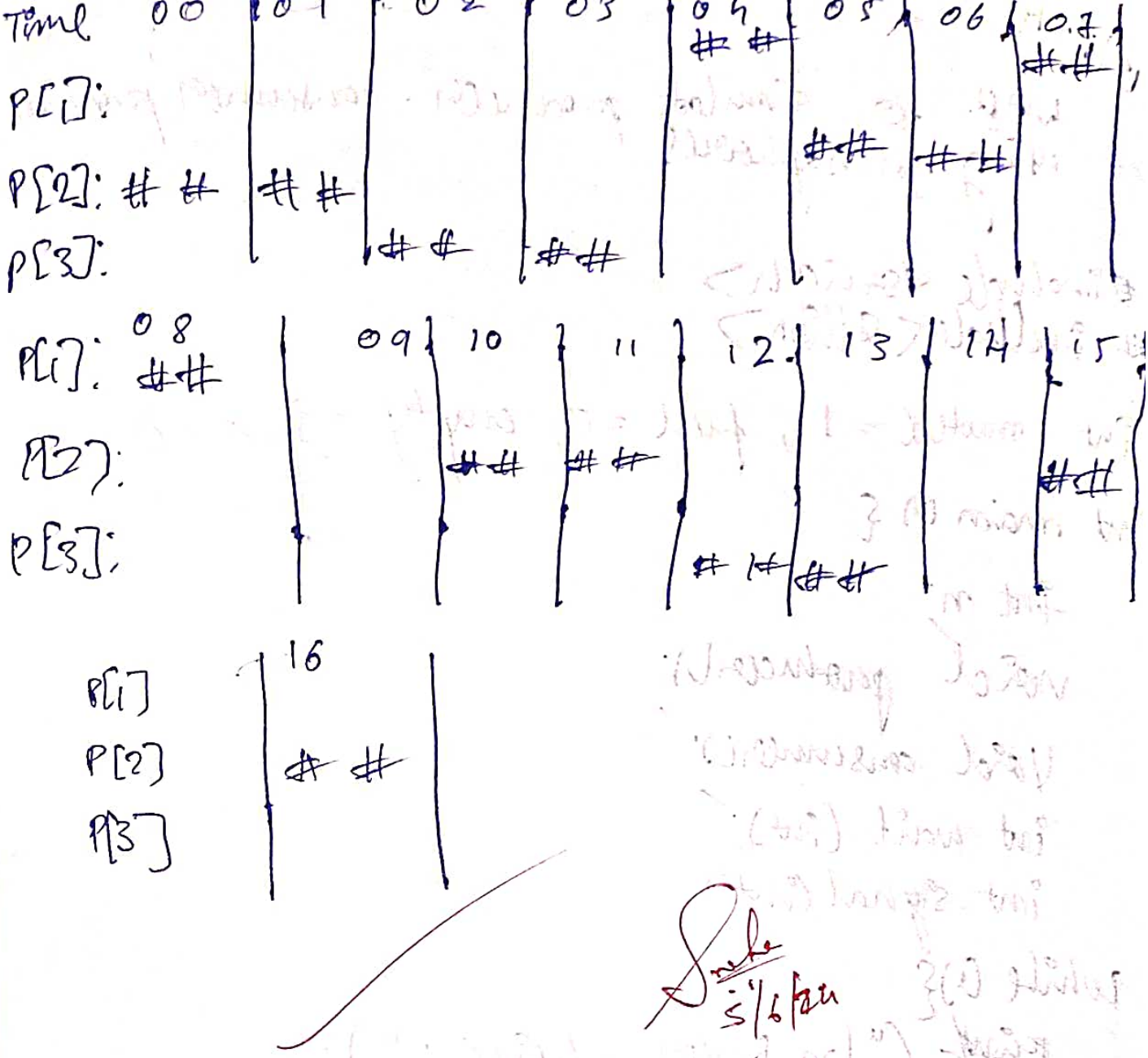
==> Execution time 2

==> Dead time : 5

P3 :

==> Execution time: 2

==> dead time : 10


Time 00 01 02 03 04 05 06 07
P[1]:
P[2]:
P[3]:
P[1]: 08 09 10 11 12 13 14 15
P[2]:
P[3]:

16
P[1]
P[2]
P[3]

Sneha
5/6/24

12/6/24

WAP 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);
while (1){
    printf ("\n Enter choice : ");
    scanf ("%d", &n);
    switch (n) {
        case 1: if ((mutex == 1) && (empty != 0))
                    producer ();
                else
                    printf ("Buffer full !! ");
                break;
        case 2:
            if ((mutex == 1) && (full != 0))
                    consumer ();
```

```
        else
            printf("Buffer 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 item %d", x);
    mutex = signal(mutex);
}
void consumer(){
    mutex = wait(mutex);
```

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

OP

1. producer
2. consumer
3. Exit.

Enter choice: 1
producer produces item 1
Enter choice: 1
producer produces item 1.
Enter choice: 2
consumer consumes item 2.
Enter choice: 2
consumer consumes item 2.
Enter choice: 2
Buffer empty!!
Enter choice 3.

12/6/24

WAP to simulate concept of dining philosophers problem.

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (i+4)%N
#define RIGHT (i+1)%N
int state[N];
int phil[N] = {0, 1, 2, 3, 4};
sem_t mutex;
sem_t S[N];
void test(int i)
{
    if (state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING) {
        state[i] = EATING;
        sleep(2);
        printf("Philosopher %d takes fork %d and %d\n", i+1, LEFT+1, i+1);
```

```
printf("P %d is Eating\n", i+1);
            S - pC&S[i];
        }
    }
}
void tf (int i) {
    S_w (&mutex);
    State[i] = H;
    printf ("P %d is Hungary\n", i+1);
        test (i);
    S-p (&mutex);
    S_w (&S[i]);
    sleep(1);
}
void p-f (int)
{
sem_wait (&mutex);
 State[i] = T;
printf ("P %d puting fork %d and
%d down\n", i+1, L+i, i+1);
printf ("P %d is think\n", i+1);
        test(L);
        test(R);
    S-p (&mutex);
}
void* p(void *num) {
```

```
while(1){
    int *i = num;
    sleep(1);
    v-f(*i);
    sleep(0)
    p-f(*i);
}
}
int main(){
    int i;
  p-thread_id[N];
  s-init(&mutex,0,1);
  for(i=0;i<N;i++)
      s-init(&s[i],0,0);
   for(i=0;i<N;i++){
p-c(&t-id[i],NULL,p,&phi[i])
   printf("P %d is thinking\n",i+1);
   }
   for(i=0;i<N;i++){
     p-join(t-id[i],NULL);
}
}
```

op

P. 1 is T

P 2 is T

P 3 is T

P 4 is T

P 5 is T

P 1 is H

P 2 is H

P 3 is H

P 4 is H

P 5 is H

P 1 takes fork 5 and 1

P 1 is E

P 2 takes fork 1 and 2

P 2 is E

P 3 takes fork 2 and 3.

P 3 is E.

P 4 takes fork 3 and 4

P 4 is E

P 5 takes fork 4 and 5

P 5 is E

P 1 putting fork 5 and 1 down.
P 1 is T
P2 putting fork 1 and 2 down
P2 is T
P3 putting fork 2 and 3 down
P3 is T
P4 putting fork 3 and 4 down.
P4 is T
P5 putting fork 4 and 5 down.
P5 is T

12/6/24

9/6/24

WAP to simulate Bankers algorithm for the purpose of deadlock avoidance

```
#include <stdio.h>
int main(){
    int n, m, i, j, k;
    printf("Enter proc: ");
    scanf("%d", &n);
    printf("Enter res: ");
    scanf("%d", &m);

    int al[n][m];
    printf("Ent al_M: \n");
    for(i=0; i<n; i++){
        for(j=0; j<m; j++){
            scanf("%d", &al[i][j]);
        }
    }
int max[n][m];
printf("Enter MAX matrix: \n");
for(i=0; i<n; i++){
    for(j=0; j<m; j++){
        scanf("%d", &max[i][j]);
    }
}
    int avl[m];
printf("Enter avl res: \n");
```

```
for(i=0; i<m; i++){
    scanf("%d", &dua[i]);
}
int f(n), ans[n], ind=0;
for(k=0; k<n; k++){
    f[k]=0;
}
int nd[n][m];
for(i=0; i<m; i++){
    for(j=0; j<m; j++){
        f[k]=0;
}
int nd[n][m];
for(i=0; i<n; i++){
    for(j=0; j<m; j++){
        nd[i][j]=max[i][j]-al[i][j]
    }
}
int y=0;
for(k=0; k<n; k++){
    for(i=0; i<n; i++){
        if flag(f[i]==0){
            int flag=0;
```

```
for(j=0;j<m;j++){
  if(ind[i][j]>ava[j]){
    flag=1;
    break;
  }
}
if(flag==0){
  ans[ind++]=i;
  for(y=0;y<m;y++){
    ava[y]+=al[i][y];
  }
  f[i]=1;
}
}
}
}
int flag=1;
for(i=0;i<n;i++){
  if(f[i]==0){
    flag=0;
    printf("The following not safe\n");
    break;
  }
}
if(flag==1){
```

```
        printf("Following Safe \n");
        for(i=0; i<n-1; i++){
            printf("P%d ->", ans[i]);
        }
        printf("P%d\n", ans[n-1]);
    }
    return 0;
}
```

op

```
Enter processes : 5
Enter resources : 3
Enter Allocation Matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter MAX matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 2 3
Enter Available resources:
3 3 2
Following Safe: P1 -> P3 -> P4 -> P0 -> P2.
```

WAP to simulate deadlock detection

```
#include<stdio.h>

Static int mark[20];
int i, j, np, nr;
int main(){
    int al[10][10], rq[10][10], av[10], r[10];
    printf("\n Enter pro : ");          w[10];
    scanf("%d", &np);
    printf("Enter res : ");
    scanf("%d", &nr);
    for(i=0; i<nr; i++){
        printf("\n Total amt Res R%d : ", i+1);
        scanf("%d", &r[i]);
    }
    printf("\n Enter req mat : \n");
    for(i=0; i<np; i++){
        int count = 0;
        for(j=0; j<nr; j++){
            if(al[i][j]==0)
                count++;
            else
                break;
        if(count == nr)
```

```
        mk[i] = 1;
    }
    for(j = 0; j < nr; j++)
        w[j] = av[j];
    for(i=0; i<np; i++){
        int cbp = 0;
        if(mk[i] != 1){
            if(rq[i][j] <= w[j])
                cbp = 1;
            else {
                cbp = 0;
                break;
            }
        if(cbp){
            mk[i] = 1;
            for(j=0; j<nr; j++)
                w[j] += al[i][j];
        }
    }
}
int dl = 0;
for(i=0; i<np; i++){
    if(mk[i] != 1)
        dl = 1;
}
```

```
if (dl)
    printf("\n dl detected\n");
else
    printf("\n No dl possible\n");
return 0;
}
```

```
of
Enter process : 5
Enter resources : 3.
Total R1 : 7
Total R2 : 5
Total R3 : 3
Enter req. Matrix :
2 1 0
2 0 2
0 0 2
1 0 0
0 1 0
Enter alloc. Matrix :
2 1 3
0 2 0
3 0 2
1 1 1
0 0 0
Deadlock detected.
```

19/6/24

3/7/24

WAP to simulate following contagious memory allocation techniques:

a) Worst fit
b) best fit
c) First fit.

```
#include <stdio.h>
#define max 25
void ff (int b[], int nb, int f[], int nf);
void wf (int b[], int nb, int f[], int nf);
void bf (int b[], int nb, int f[], int nf);

int main(){
    int b[max], f[max], nb, nf;
    printf("Memory Management \n");
    printf("\n Enter Blocks: ");
    scanf("%d", &nb);
    printf("Enter files: ");
    scanf("%d", &nf);
    printf("Enter block size: \n");
    for(int i=1; i<=nb; i++){
        printf("Block %d: ", i);
        scanf("%d", &b[i]);
    }
    printf("\n Enter the size of files: \n");
    for(int i=1; i<=nf; i++)
    printf("\n Memory Management - FF ");
```

```
ff(b, nb, f, nf);
printf("\n\n Memory management - WF");
wf(b, nb, f, nf);
printf("\n\n Memory management - BF");
bf(b, nb, f, nf);
return 0;
}
void ff(int b[], int nb, int f[], int nf){
int bf[max]={0};
int f[max]={0};
int frag[max],i,j;
for(i=1;i<=nf;i++){
for(j=1;j<=nb;j++){
if(b[j]!=1 && b[j]>=f[i]){
ff[i]=j;
bf[j]=1;
frag[i]=b[j]-f[i];
break;
}
}
}
printf("\n File_no:\tFile_size:\tBlock_no:
\t Block_size:\t Fragment");
for(i=1;i<=nf;i++){
printf("\n %d\t\t %d\t\t %d\t\t %d", i,
f[i], ff[i], b[ff[i]], frag[i]);
}
}
```

```
void wf(int b[], int, int f[], int nf){
    int bf[max]={0};
    int ff[max]={0};
    int flag[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;
                }
            }
        }
    }
    flag[i]=highest;
    bf[ff[i]]=1;
    highest=0;
}
}
}
void bf(int b[], int nb, int f[], int nf){
    int bf[max]={0};
    int ff[max]={0};
    int flag[max], i, j, temp, lowest=1000;
    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 && ff[i]!=0; i++){
printf("\n %d \t\t %d \t\t %d \t\t %d \t\t %d",
    i, f[i], bf[i], b[ff[i]],
    frag[i]);
}
}
```

OP

Memory Management

Enter blocks: 5

Enter files: 8

Enter block size:

100 500 200 300 600

Enter process size:

212 415 63 124 23 89 73 13

1. First
2. Best
3. Worst

1.

Pro no.	Pro. 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

2.

Req no.	Req size	Block no.
1	212	4
2	415	Not Allowed
3	63	4
4	124	5
5	23	4
6	89	3
7	73	3
8	13	3

3.

Req no.	Req size	Block no.
1	212	Not Allowed
2	415	11
3	63	11
4	124	11
5	23	5
6	89	Not Allowed
7	73	11
8	13	5

3/4/24

WAP - 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=0;
void data() {
    printf("Enter page length:");
    scanf("%d", &n);
    printf("\n Enter page sequence:");
    for(i=0; i<n; i++)
        scanf("%d", &in[i]);
    printf("\n Enter frames:");
    scanf("%d", &f);
}
void initialize(){
    pg=0;
    for(i=0; i<f; i++)
        p[i]=9999
}
```

```
int Hit (int data) {
    hit = 0;
    for (j=0 ; j<f ; j++){
        if (p[j] == data) {
            hit=1;
            break;
        }
    }
    return hit;
}
int get Hit (int data){
    int hitind;
    for (k=0 ; k<f ; k++){
        if (p[k] == data) {
            hitind =k;
            break;
        }
    }
    return hitind;
}
void dis (){
for (k=0 ; k<f ; k++){
    if (p[k] != 9999)
    }
}
```

```
void dispPg()
{
  printf("\n Total no. of pages: %d", pg);
}
void fifo(){
    getdata();
    initialize();
    for(i=0; i<n; i++){
      printf("\n For %d:", in[i]);
      if(isHit(in[i]==0){
        for(k=0; k<f-1; k++)
          p[k] = p[k+1];
        p[k] = in[i];
        pg++;
        disp();
      }
      else
      printf("No page fault");
    }
    disPg();
}
void optimal(){
    initialize();
    int near[50];
    for(i=0; i<n; i++){
      printf("\n %d:", in[i]);
```

```
if (hit[pn[i]==0){
  for(j=0; j<f ; j++){
    int pg=p[j];
    int found=0;
    for(k=i; k<n; k++)
  int max=-9999;
  int rep;
  for(j=0; j<nf; j++){
    if(near[j]>max){
      max=near[j];
      rep=j;
    }
  }
  p[rep]=pn[i];
  pg++;
  dis();
}else
  printf("No page fault");
}
dis pg();
}
void fault(){
  initialize();
```

```
int least[50];
for(i=0; i<n; i++){
    printf("\nFor %d : ", in[i]);
if(hit[in[i]==0){
        int pg = p[j];
        int found=0;
        for(k=i-1; k>=0; k--)
        }
        else
        found = 0;
    }
    if(found)
        least[j] = -9999;
}
int min=9999;
int rep;
for(j=0; j<nf; j++){
    if(least[j] < min){
        min = least[j];
        rep=j;
    }
}
p[rep] = in[i];
    pg++;
    chs();
}else
```

```
        printf("No page fault");
    }
    dispPg();
}
int main(){
    int choice()
    while(1){
        scanf("%d", &choice);
        switch(choice);
        {
        case 1: getData();
                break;
        case 2: fifo();
                break;
        case3: optimal();
                break;
        case 4: Lru();
                break;
        default: return 0;
                break
    }
}
}
```

OP

Page replacement Algorithms

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

Enter choice: 2

Enter page length : 12

Enter sequence : 1 2 3 4 1 2 5 1 2 3 4 5

Enter frames : 3

1 : 1

2 : 12

3 : 123

4 : 234

1 : 341

2 : 412

5 : 125

1 : No page fault

2 : "

3 : 253

4 : 534

5 : No page fault

Total no. page : 9

Enter choice : 3

1 : 1

2 : 12

3 : 123

4 : 124

1 : No page fault

2 : "

5 : 125

1 : No page fault

2 : "

3 : 3 2 5

4 : 4 2 5

5 : No page fault

Total page faults : 7

Enter choice : 4

1 : 1

2 : 12

3 : 1 2 3

4 : 2 3 4

1 : 4 1 3

2 : 4 1 2

5 : 5 1 2

1 : No page fault.

2 : "

3 : 5 1 2

4 : 342

5 : 345

Total page faults : 10

Enter choice 5

== code successfull ==

Sneha

3/7/24

10/7/24

WAP to simulate disk scheduling

a) FCFS

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

OP

Enter request : 5

Enter sequence : 34 60 120 180 240

Enter initial : 50

Total head is 222.

b) Scan

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int RQ[100], i, j, n, TM = 0, pt, sz, mv;
    printf("Enter requests\n");
    scanf("%d", &n);
    printf("Enter sequence\n");
    for(i=0; i<n; i++)
        scanf("%d", &RQ[i]);
    printf("Enter initial\n");
    scanf("%d", &pt);
    printf("Enter total disk\n");
    scanf("%d", &sz);
    scanf("%d", &mv);
    for(i=0; i<n-1; j++){
        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 (it < RQ[i]) {
        index = i;
        break;
    }
}
if (move == 1){
    for (i=index; i<n; i++){
        THM = THM + abs(RQ[i] - it);
        it = RQ[i];
    }
    THM = THM + abs(s2 - RQ[i-1] - 0);
    it = s2 - 1;
}
```

```
for (i = index - 1; i >= 0; i--){
    THS = THS + abs(RQ[i] - it);
    it = RQ[i];
}
THM = THM + abs(RQ[i+1] - 0);
it = 0
for (i = index; i < n; i++){
    THM = THM + abs(RQ[i] - it);
    it = RQ[i];
}
printf("Total movement %d", THM);
return 0;
}
```

OP

Enter request: 5

Enter sequence: ~~24~~ 60 120 180 240

Enter initial: 50

Enter total disk size: 250

Enter head movement direction for 1 to 0

1.

Total head movement is 414.

C-Scan

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int RQ[100], i, j, n, TM=0, it, sz, mv;
    printf("Enter Request \n");
    scanf("%d", &n);
    printf("Enter sequence \n");
    for(i=0; i<n; i++)
        scanf("%d", &RQ[i]);
    printf("Enter initial \n");
    scanf("%d", &it);
    printf("Enter total disk size \n");
    scanf("%d", &sz);
    printf("
    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 id;
for(i=0; i<n; i++){
    if(it < RQ[i]){
        id = i;
        break;
    }
}
for(i=id-1; i>=0; i--){
    THM = THM + abs(RQ[i] - it);
    it = RQ[i];
}
THM = THM + abs(RQ[i+1] - 0);
it = sz-1;
for(i=n-1; i>=id; i--){
    THM = THM + abs(RQ[i] - it);
    it = RQ[i];
```

```
    }
  }
  printf("Total head %d", THM);
  return 0;
}
```

OP

Enter requests: 5

Enter sequence: 34 60 120 100 240

Enter initial: 50

Enter movement from: 1 too: 1

Total head movement: 482.

10/11/24