INDEX

1BM22CS019

Name : Aditya Ram S H Class :

Section : Roll No. : Subject :

Sl. No.	Date	Title	Page No.	Teacher Sign. / Remarks
01	08/05/24	Add, Sub, Multiply of Matrices	1-3	10
02	15/05/24	FCFS and SJF	4-7	
03	22/05/24	Priority & round robbin scheduling	8-10	9
04	05/06/24	Rate monotonic & earliest deadline first	11-22	
05	12/06/24	producer - diners using semaphore	23-26	10
06	19/06/24	Deadlock Avoidance, detection	27-29	10
07	03/07/24	Contiguous memory allocation and page memory allocation	30-40	10
08	10/07/24	Disk scheduling algorithms • FCFS, SCAN, C-SCAN	41-47	9

08/05/2024

WAP in C to Add, Subtract and Multiply Matrices

```
# 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 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 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[i][j] = 0;
            for (int k=0; k<CS; k++)
                C[i][j] += A[i][k] * B[k][j];
        }
}
```

```
void dm (int M[RS][CS]) {
    for (int i=0; i < RS; i++)
        for (int j = 0; j < CS; j++)
            printf("%d \t", M[i][j]);
        printf("\n");
}

void 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("Matrix A :\n");
    dm(A);
    printf("Matrix B :\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);
}
```

Output :-

Matrix A :-

1	2	3
4	5	6
7	8	9

Matrix B :-

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
138	114	90

Sneha
8/5/24

15/05/24

FCFS and SJF CPU scheduling

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

int n, i, j, pos, temp, choice, BT[20], WT[20], TT[20], pre[20], total=0;
float aTT=0, aWT=0;

int FCFS() {
    WT[0] = 0;
    for(i=0; i<n; i++){
        WT[i] = 0;
        for(j=0; j<i; j++)
            WT[i] += BT[j];
    }
    printf("\n pre \t\t Burst Time \t\t Waiting time \t\t
        Turaround 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 %d \t %d", i+1, BT[i], WT[i], TT[i]);
    }
    aWT = aWT/(float) i;
    aTT = aTT/(float) i;
    printf("\n Average Waiting Time : %.2f", aWT);
    printf("\n Average Turnaround Time : %.2f \n", aTT);

    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 = prc[i];
        prc[i] = prc[pos];
        prc[pos] = temp;
    }
    WT[0] = 0;
    for (i=1; i<n; i++){
        WT[i] = 0;
        for (j=0; j<i; j++)
            WT[i] += BT[j];
        total += WT[i];
    }
    awt = total/(float)n;
    total = 0;
    printf("\n prc\t Burst-time\t waiting time\tat");
    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", prc[i], BT[i], WT[i]
                                                    TT[i]);
    }
    aTT = total/(float)n;
    printf("\n Average waiting time = %f", awt);
    printf("\n Average Turnaround time = %f\n", aTT);
    return 0;
}
```

```
void main() {
    printf("Enter the total number of process: ");
    scanf("%d", &n);
    if(n <=0) {
        printf("Invalid Input");
        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");
            return 1;
        }
        prc[i] =i+1;
    }
    while (1) {
        printf("\n--- Main Menu ---\n");
        printf("1. FCFS scheduling\n 2. SJF scheduling\n 3. Exit");
        printf("Enter your choice:");
        scanf("%d", &choice);
        switch(choice) {
            case 1: FCFS();
                    break;
            case 2: SJF();
                    break;
            case 3: return 0;
            default: printf("Invalid Input");
                    break;
        }
    }
    return 0;
}
```

OUTPUT :-

Enter the total number of process : 3

Enter Burst time :

P[1] : 2

P[2] : 1

P[3] : 3

--- Main Menu --

1. FCFS scheduling
2. SJS scheduling
3. EXIT

Enter your choice : 1

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

Average Waiting time = 1.67

Average Turnaround time = 3.67

Enter your choice : 2

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

Average waiting time = 1.33

Average Turnaround time = 3.33

Enter your choice : 3

Sneha
15/5/24

22/05/2024

Priority (Non preemptive) scheduling :-

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

struct process {
    int pi;
    int bt;
    int p;
    int at;
    int wt;
    int tt;
};

void fwt(struct process proc[], int n, int wt[]){
    wt[0] =0;
    for(int i=1; i<n; i++) wt[i]=proc[i-1].bt + 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].bt + wt[i];
}

void fat (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);

    for(int i=0; i<n; i++) {
        total_wt += wt[i];
        total_wt += tat[i];
    printf("\n%d\t%d\t%d\t%d", proc[i].pi, proc[i].bt, wt[i],
                                tat[i]);
}
```

```
    printf("\n Average waiting time = %f", total_wt/n);
    printf("\n Average turaround time = %f\n", total_tat/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].p < proc[pos].p ||
                (proc[j].p == proc[pos].p &&
                proc[j].at < proc[pos].at))
                pos = j;
        }
        temp = proc[i];
        proc[i] = proc[pos];
        proc[pos] = temp;
    }
    fat(proc, n);
}

void main () {
    int n;
    struct process proc [10];
    printf("number of processes? : ");
    scanf("%d", &n);
    for(int i= 0; i<n; i++) {
        printf("\n process ID: ");
        scanf("%d", &proc[i].pi);
        printf("burst time :");
        scanf("%d", &proc[i].p);
    }
    ps(proc, n);
}
```

Output :-

number of processes ? : 3

process ID : 1
burst time : 4
p : 2
arrival time : 1

process ID : 2
burst time : 3
p : 3
arrival time : 1

process ID : 3
burst time : 2
p : 4
arrival time : 2

Process ID	Burst time	p	Arrival time	Waiting time	Turnaround time
1	4	2	0	0	4
2	3	3	1	4	7
3	2	4	2	7	9

Average waiting Time = 3.667

Average Turnaround Time = 6.667

05/06/2024

Rate Monotonic & Earliest Deadline First

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


~~int tat(int pr~~
int tat ( int p[], 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 p[], int n, int bt[], int wt[], int q) {

    int rem[n];
    for (int i=0; i<n; i++)
        rem [i] = bt[i];
    int t = 0;
    while(1) {
        int done = 1;
        for(int i=0; i<n; i++){
            if (rem[i] >0) {
                done = false;
                if(rem [i] > q) {
                    t += q;
                    rem[i] -= q;
                }
                else {
                    t += rem [i];
                    wt[i] = t - bt[i];
                    rem [i] = 0;
                }
        }   if(done == 1) break;
```

```
    return 1;
}

int tat(int p[], int n, int bt[], int q) {
    int wt[n], tat[n], twt=0, tat=0;
    wt(p, n, bt, wt, q);
    tat(p, n, bt, wt, tat);

    for(int i=0; i<n; i++) {
        twt += wt[i];
        tat += tat[i];
        printf("\n%d\t%d\t%d", i+1, bt[i], wt[i], tat[i]);
    }
    printf("\n Average waiting time = %f", twt/n);
    printf(" Averge turnaround time = %f", tat/n);
    return 1;
}

void main() {
    int n;
    printf(" Number of process ? :");
    scanf("%d", &n);
    int p[n], bt[n], q; quantum
    printf("\n Enter the ~~number of process~~ :");
    scanf("%d", &q);

    for(int i=0; i<n; i++){
        printf("\n Enter the process: ");
        scanf("%d", &p[i]);
        printf("Enter the burst time : ");
        scanf("%d", &bt[i]);
    }
    tat(p, n, bt, q);
}
```

Output :-

(a)

Process	AT	CPU time	Priority	WT	TAT
P_1	0	4	2	0	4
P_2	1	3	3	11	14
P_3	2	1	4	9	10
P_4	3	5	5	1	6
P_5	4	2	5	5	7

(b) Enter no. of process = 3

Enter Quantum time = 2

Enter process : 1 2 3

Enter Burst time : 5 7 3

P	BT	WT	TAT
1	5	7	12
2	7	8	15
3	3	8	4

avg wt = 7.667

avg TAT = 12.667

05/06/2024

Earliest Deadline First :-

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

#define MP 10

typedef struct {
        int id;
        int bt;
        float pri;
} task;

int npro;
int et[MP], pri[MP], rt[MP], ded[MP], rd[MP];

void gpi(int sa) {
        printf("Enter total number of process (maximum %d): ", MP);
        scanf("%d", &npro);

        if (npro < 1)
                exit(0);

        for(int i=0; i< npro; i++) {
                printf("\npri %d: (n", i+1);
                printf("==> Execution time: ");
                scanf("%d", &et[i]);
                rt[i] = et[i];
        }
}
```

```
int max (int a, int b, int c){
    int max;
    if( a>=b && a>=c)
        max =a;
    else if ( b>=a && b>=c)
        max =b;
    else if( c>=a && c>=b)
        max = c;
    return max;
}

int get_at(int sa) {
    if (sa == 1)
        return max(pr[0], pr[1], pr[2]);
    else if(sa == 2)
        return max (ded[0], ded[1], ded[2]);
}

void ps (int pl[], int cycles){
    printf ("\n Scheduling: \n\n");
    printf ("Time : ");
    for (int i=0; i < cycles; i++) {
        if (i < 10)
            printf("| 0%d ", i);
        else
            printf("| %d ", i);
    }
    printf("\n");
    for (i=0; i< npro; i++) {
        printf("p[%d]: ", i+1);
        for (int j=0; j < cycles; j++) {
            if (pl[j] == i+1)
                printf("| ####");
```

```
        else
            printf("|  ");
        }
        printf("|\n");
    }
}


void rm (int time){
    int pl[100] = {0}, min = 999, np = 0;
    float uti=0;

    for (int i=0; i< npro; i++)
        uti += (1.0*et[i])/pr[i];

    int n = npro;
    int m = (float) (n*(pow(2, 1.0/n) - 1));

    if (uti > m)
        printf("\n Given problem is not schedulable under
                the said scheduling algorithm.\n");

    for (i=0; i<time; i++)
        min= 1000;
        for(j=0; j< npro; j++){
            if(rt[j] >0)
                if (min > pr[j]){
                    min = pr[j];
                    np = j;
                }
        if (rt[np] >0) {
            pl[i] = np+1;
            rt[np] --=1;
        }
```

```
for(K=0; K<npro; K++)
        if( (i+1) % pr(K) == 0) {
                rd[K] = et[K];
                np = K;
        }
    }
    ps(pt, time);
}


void edf (int time) {
    float uti=0;
    for(i=0; i< npro ; i++)
            uti += (1.0 * et[i]) / ded[i];

    int n = npro;
    int pr[npro];
    int ma, up =0, mi, pl [time];
    bool ir[npro];
    for (i=0; i< npro; i++){
            ir[i] = true;
            pr[i] = i+1;
    }
    ma = ded[0];

    for (int i=1; i<npro ; i++)
            if(ded[i] > ma)
                    ma = ded[i];

    for (int i=0; i<npro; i++) {
            for (j=i+1; j<npro; j++) {
                    if(ded[j] < ded[i])
                            int temp = et[i];
    }
}
```

```
for (int i=0; i < npro; i++)
    if((ded[i] <= rd[i]) && (ir[i] == false)){
        et[i] = rt[i];
        ir[i] = true;
    }

    mi = mq;
    cp = -1;
    for (int i=0; i < npro; i++)
        if( (ded[i] <= mi) && (et[i] > 0)){
            cp = i;
            mi = ded[i];
        }
    }
    ps(pl, time);

}

void main() {
    int op;
    int at;
    while(1) {
        printf("\n 1. Rate Monotonic\n
                2. Earliest ded first\n
                3. Proportional scheduling\n\n
                Enter your choice.");

    scanf("%d", &op);
    switch (op){
        case 1:
            gpi(op);
            at = get_at(op);
            rm(at); break;
```

```
        case 2:
            gpi(op)
            at = get_bt(op);
            edf(at);
            break;

        case 3:
            exit(0);

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

OUTPUT :-

1. Rate Monotonic
2. Earliest ded first
3. proportional scheduling.

Enter your choice : 2
Enter total num of process (max 10) : 3

pr 1:
==> Execution time : 3
==> Dead time : 20

pr 2:
==> Execution time : 2
==> dead time : 5

pr 3 :

=> Execution time : 2

=> ded : 10

scheduling :

Time :	00	01	02	03	04	05	06	07	08	09
P[1];					##			##	##	
P[2];	##	##	##	##		##	##			
P[3];										

Time;	10	11	12	13	14	15	16	17	18	19
P[1];										
P[2];	##	##	##	##		##	##			
P[3];										

12/06/2024

Write a C program to simulate producer - consumer problem using semaphores.

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

int m=1, f=0, e=3, x=0;

void main() {
    int n;
    void p();
    void c();
    int w(int);
    int s(int);
    printf("\n1.producer\n2. consumer\n3. Exit");

    while(1){
        printf("\nEnter your choice : ");
        scanf("%d", &n);
        switch(n){
            case 1:
                if ( m==1 && e!=0)
                    p();
                else
                    printf("Buffer is full!!");
                break;
            case 2:
                if( m==1 && f!=0)
                    c();
                else
                    printf("Buffer is empty!!");
                break;
            case 3:
                exit(0);
                break;
        }
    }
```

```
    return 0;
}

int w(int s){
    return (--s);
}

int s(int s){
    return (++s);
}

void p(){
    m = w(m);
    f = s(f);
    e = w(e);
    x++;
    printf("\nproducer produces the item %d", x);
    m = s(m);
}

void c(){
    m = w(m);
    f = w(f);
    e = s(e);
    printf("\nconsumer consumes item %d", x);
    x--;
    m = s(m);
}
```

Output :-

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 is empty !!
Enter choice : 3

12/06/2024

write a C program to simulate the concept of Dining-philosopher problem.

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

#define N 5
#define T 2
#define H 1
#define E 0
#define L (i+4)% N
#define R (i+1)% N

int st[N];
int phil[N] = {0, 1, 2, 3, 4};

sem_t m;
sem_t s[N];

void test (int i) {
    if (st[i] == H && st[L] != E && st[R] != E) {
        st[i] = E;
        sleep (2);
        printf("philosopher %d takes fork %d and %d\n",
                                i+1, i+1, i+1);
        printf("philosopher %d is eating\n", i+1);
        sem_post (&s[i]);
    }
}
```

```
void tf(int i){
    sem_wait(&m);
    st[i] = H;
    printf("philosopher %d is hungry\n", i+1);
    test(i);

    sem_post(&m);
    sem_wait(&S[i]);
    sleep(1);
}

void pf(int i){
    sem_wait(&m);
    st[i]=T;

    printf("philosopher %d putting fork %d and %d down\n",
                                    i+1, i+1, i+1);
    printf("philosopher %d is thinking\n", i+1);
    test(L);
    test(R);
    sem_post(&m);
}

void * P(void * num){
    while(1){
        int * i = num;
        sleep(1);
        tf(*i);
        sleep(0);
        pf(*i);
    }
}
```

```
void main() {
    int i;
    pthread_t thread_id[N];

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

<u>Output:</u>

p1 is t
p2 is t
p3 is t
p4 is t
p5 is h
p1 is h
p2 is h
p3 is h
p4 is h
p5 is h
p1 takes fork 5 and 1
p1 is E
P2 takes fork 1 and 2
p2 is E
p3 takes for 3 and 4

P 3 is E

P 4 takes forks 3 and 4

P 4 is E

P 5 takes fork 4 and 5

P 5 is E

P1 putting fork 5 and 1 down

P2 is t

P2 is putting fork 1 and 2 down

P 2 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

19/06/2024

DeadLock Activation Bankers algorithm

```
#include <stdio.h>

void main() {

    int n, m, i, j, k;

    printf("Enter processes : ");
    scanf("%d", &n);

    printf("Enter resources : ");
    scanf("%d", &m);

    int al[n][m];
    printf("Enter the allocation Matrix : \n");

    for(i=0; i<n; i++) {
        for(j=0; j<m; j++) {
            scanf("%d", &al[i][j]);
        }
    }

    int max[n][m];
    printf("Enter the MAX Matrix : \n");

    for(i=0; i<n; i++)
        for(j=0; j<m; j++)
            scanf("%d", &max[i][j]);

    int ava[m];
    printf("Enter the available resources : \n");

    for(i=0; i<m; i++)
        scanf("%d", &ava[i]);
```

```
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(f[i]==0){
            int flag = 0;
            for(j=0; j<m; j++)
                if(nd[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 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]);
}
```

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

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

19/06/2024

Deadlock Detection :-

```
#include <stdio.h>

static int mark [20];
int i, j, np, nr;

void main() {
    int al[10][10], rq[10][10], av[10], r[10], w[10];

    printf("\nEnter the number of processes : ");
    scanf("%d", &np);

    printf("Enter the number of resources : ");
    scanf("%d", &nr);

    for(i=0; i<nr; i++) {
        printf("\n Total amount of Resources R%d : ", i+1);
        scanf("%d", &r[i]);
    }
    printf("\nEnter the rq matrix: \n");
    for (i=0; i<np; i++)
        for (j=0; j<nr; j++)
            scanf("%d", &rq[i][j]);

    for (j=0; j<nr; j++){
        av[j] = r[j];
        for (i=0; i<np; i++) av[j] -= al[i][j];
    }
```

```
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) mark[i]=1;
}

for (i=0; i<np; i++) {
    w[j] = av[j];

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

```
if(dl)
    printf("\n deadlock detected \n");
else
    printf("\n No deadlock potible \n");
}
```

Output:-

Enter the no. of processes :- 5
Enter the no. of resources : 3

Total Amount of the resources R1 : 70
R2 : 5
R3 : 7

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

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

Deadlock detected.

19/6/24

03/07/2024

Contiguous Memory Allocation.

```
#include <stdio.h>

#define m 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);

void main() {
    int b[m], f[m], 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("\nEnter the size of the blocks :\n");

    for (int i=1; i<= nb; i++) {
        printf("Block %d :", i);
        scanf("%d", &b[i]);
    }

    printf("\n Enter the size of the files :\n");
```

```
for (int i=1; i<=nf; i++) {
        printf("File %d", i);
        scanf("%d", &f[i]);
}
printf("\n Memory Management Scheme - First Fit");

ff(b, nb, f, nf);
printf("\n\n Memory Management scheme - Worst fit");
wf(b, nb, f, nf);
printf("\n\n Memory Management scheme - Best fit");
bf(b, nb, f, nf);
}

void ff(int *b, int nb, int *f, int nf) {
    int bf[m] = {0};
    int ff[m] = {0};
    int frag[m], 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 %d\t %d\t %d\t %d", i, f[i], ff[i], b[ff[i]], frag[i]);
```

```
void wf (int b*, int nb, int*f, int nf) {
    int bf[m] = {0};
    int ff[m] = {0};
    int frag[m], 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 && high < temp) {
                    ff[i] = j;
                    high = temp;
                }
            }
        }

    frag[i] = high;
    bf[ff[i]] = 1;
    high = 0;
    }
    printf("\nFile no:\t File size:\t Block no:\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 bf(int* b, int nb, int *f, int nf){
    int bf[m] = {0};
    int ff[m] = {0};
    int frag[m], i, j, temp, low = 10000;

    for (i=1; i<=nf; i++)
        for (j=1; j<=nb; j++)
            if (bf[j] - f[i]);
            temp = b[j] - f[i];
```

```
if (temp >= 0.4f && low > temp) {
        ff[i] = j;
        low = temp;
    }
    frag[i] = low;
    bf[ff[i]] = 1;
    low = 10000;
}
printf("\nFile_no :\t File_size :\t Block_no\t Fragement");
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 5
Enter the no. of process 8
Enter the block size 100 500 200 300 600
Enter the process size 212 415 63 124 23 89 73 13

1. First Fit 2. Best Fit 3. Worst Fit
Enter your choice 1

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

Enter your choice 2

Process no.	Process 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

choice 3

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

03/07/2024

Page Memory Allocation.

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

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

void initialize(){
    pf =0;
    for (i=0; i<f; i++) p[i] =99999;
}

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 dispp f() {
    printf("\nTotal no. of page faults : %d", pf);
}

void fifo() {
    getD();
    initialize();
    for(i=0; i<n; i++) {
        printf("\nEnter %d :", in[i]);
        if (isHit(in[i])==0){
            for (k=0; k<f-1; k++)
                p[k] = p[k+1];
            p[k] = in[i];
            pf++;
            dispPages();
        }
        else printf("No page fault");
    }
    dispp f()
}
```

```
void optimal() {
    initialize();
    int near[50];
    for(i=0; i<n; i++) {
        printf("\nFor %d :", in[i]);
        if (isHit(in[i]) ==0) {
            for(j=0; j<f; j++) {
                int pg = p[j];
                int found =0;
                for(k=i; k<n; k++) {
                    if (pg==in[k]) {
                        near[j] = k;
                        found =1;
                        break;
                    }
                    else
                        found =0;
                }
                if (!found)
                    near[j] = 99999;
            int max =-9999;
            int repindex;
            for (j=0; j<n; j++) {
                if (near[j] > max) {
                    max = near[j];
                    repindex = j;
                }
            }
            p[repindex] = in[i];
            pf++;
            dispPages();
        }
        else
            printf("No Page fault");
    }
    dispPf();
```

```
void lru() {
    initialize();
    int least[50];
    for (i=0; i<n; i++) {
        printf("\nFor %d : ", in[i]);
        if (isHit(in[i]) ==0) {
            for (j=0; j<n; j++)
                int pg = p[j];
                int found = 0;
                for (k=i-1; k>=0; k--)
                    if (pg == in[k])
                        least[j] = k;
                        found = 1;
                    } break;
                    else
                        found = 0;
                if (!found)
                    least[j] = -9999;
            }
            int min = 9999;
            int repindex;
            for (j=0; j<n; j++)
                if (least[j] < min)
                    min = least[j];
                    repindex = j;
            p[repindex] = in[i];
            pf++;
            displages();
        }
        else
            printf("No page fault!");
    }
    dispPf();
}
```

```
void main() {
    int choice;
    while (1) {
        printf("\n Page Replacement Algorithm
                \n1. Enter data \n2. FIFO
                \n3. optimal
                \n4. LRU \n5. Exit \n Enter your choice:");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                getD();
                break;
            case 2:
                fifo();
                break;
            case 3:
                optimal();
                break;
            case 4:
                lru();
                break;
            default:
                return 0;
                break;
        }
    }
}
```

Output :-

Page Replacement Algorithms

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

Enter your choice : 2

Enter length of page reference sequence : 12

Enter the page reference sequence : 1 2 3 4 1 2 5 1 2 3 4 5

Enter no. of frames : 3

For 1 : 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 : 5 3 4

For 5 : No page fault

Total no. of page faults : 9

~~Page Replacement Algorithms~~

Enter your choice : 3

For 1 : 1

For 2 : 1 2

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 no. of page fault : 7

Enter your choice : 4

For 1 : 1

for 2 : 1 2

for 3 : 1 2 3

for 4 : 4 1 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 no of page faults : 10

Enter your choice : 5

3/7/24

10/07/2024

Disk scheduling algorithms FCFS, SCAN, C-SCAN

(a) FCFS

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

int main() {
    int RQ[100], i, n, thm=0, init;
    printf("Enter the no. of req\n");
    scanf("%d", &n);
    printf("Enter the req sequence\n");
    for (i=0; i<n; i++)
        scanf("%d", &RQ[i]);
    printf("Enter initial head position\n");
    scanf("%d", &init);

    for (i=0; i<n; i++) {
        thm = thm + abs(RQ[i] - init);
        init = RQ[i];
    }
    printf("Total head moment is %d", thm);
    return 0;
}
```

OUTPUT:- Enter the no. of requests 5
Enter the req sequence 34 60 120 180 240
Enter the initial head position: 50
Total head movement is 222

(b) SCAN

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

int main() {
    int RQ[100], i, j, n, thm=0, init, size, m;
    printf("Enter the no. of req\n");
    scanf("%d", &n);
    printf("Enter the req seq\n");
    for (i=0; i<n; i++)
        scanf("%d", &RQ[i]);
    printf("Enter initial head position\n");
    scanf("%d", &init);
    printf("Enter total disk size\n");
    scanf("%d", &size);
    printf("Enter the head movement direction for high-1
        and for low-0 \n");
```

```
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( init < RQ[i]){
        index = i;
    }   break

if( m==1){
    for( i= index; i<n; i++){
        thm = thm + abs.(RQ[i]-init);
        init = RQ[i];
    }
    thm = thm + abs(size - RQ[i-1] - 1);
    init = size-1;
    for( i= index-1; i>=0; i--) {
        thm = thm + abs (RQ[i] - init);
        init = RQ[i];
    }
}
```

```
else {
    for(i=index-1; i>=0; i--){
        thm = thm + abs(RQ[i] - init);
        init = RQ[i]
    }

    thm = thm + abs(RQ[i+1] - 0);
    init = 0;
    for( i=index; i<n; i++){
        thm = thm + abs (RQ[i] - init);
        init = RQ[i];
    }
    printf("Total head movement is %d", thm);
    return 0;
}
```

OUTPUT :- Enter the number of requests : 5

Enter the request sequence : 34 60 120 180 240

Enter the initial head position : 50

Enter the total disk size : 250

Enter the head movement direction : 1

Total head movement is 414

(c) C- SCAN

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

void main() {
    int RQ[100], i, j, n, thm=0, init, size, m;
    printf("Enter the no. of req \n");
    scanf("%d", &n);
    printf("Enter the req sequence\n");
    for(i=0; i<n; i++)
        scanf("%d", &RQ[i]);
    printf("Enter initial head position\n");
    scanf("%d", &init);
    printf("Enter total disk size\n");
    scanf("%d", &size);
    printf("Enter the head movement direction\n");
    scanf("%d", &m);

    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(init< RQ[i])
                index = i;
                break;

if(m==1) {
    for(i= index ; i<n; i++) {
        thm = thm + abs(RQ[i] - init);
        init = RQ[i];
    }

    thm = thm + abs(size-1 - 0);
    init = 0;

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

Output :-

Enter the number of requests : 5

Enter the sequence of requests : 34 60 120 180 241

Enter the initial head position : 50

Enter the total disk size : 250

Enter the head movement direction : 1

Total head movement is 482

15/7/24