

# INDEX

Name : Nakul Dholl Class : 4C

Section : ..... Roll No. : ..... Subject : .....

Sl. No.	Date	Title	Page No.	Teacher's Sign. / Remarks
		5/5/24 First come First serve		
	5/5/24	SJF		
	5/5/24	Round Robin		
	5/6/24	Rate Monotonic		
	5/6/24	Priority scheduling		
	5/6/24	Earliest Deadline first		
12	5/6/24	Proportional Share		
12	5/6/24	Producer Consumer Problem		
	19/6/24	Bomber Alg		
	3/7/24	Deadlock detection		
	10/7/24	Contiguous Memory Allocation		
	10/7/24	Page Replacement - Algorithm.		
		Computer Systems		

8/5/24

## Q) Code for FCFS.

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

```
int AT[4], BT[4], CT[4], TAT[4], WT[4],  
temp = 0, a, b, currentTime, n;
```

```
int main()
```

```
{
```

```
int i;
```

```
printf ("Enter the Arrival times for  
programs");
```

```
for (i=0; i<4; i++)
```

```
{
```

```
scanf ("%d", &AT[i]);
```

```
}
```

```
printf ("Enter the Burst times for program");
```

~~```
for (int i=0; i<4; i++)
```~~~~```
{
```~~~~```
scanf ("%d", &BT[i]);
```~~~~```
}
```~~

printf("In values for completion time\n");

for cont i=0; i<n; i++)

currentTime = (i==0) ? 0 : CT[i-1];

if (currentTime < AT[i])

{

x = AT[i] - currentTime;

printf("CPU is Idle for %d", x);

currentTime = currentTime + x;

}

Temp = currentTime + BT[i];

CT[i] = temp;

printf("%d", CT[i]);

}

temp = 0;

values  
printf("Waiting for turn around time");

```
for(i=0; i<4; i++)
{
    temp = (CTC[i] - ATC[i]);
    TAT[i] = temp;
    printf("\n%.d", TAT[i]);
}
```

temp = 0;

values for waiting time);

{

```
for(i=0; i<4; i++)
{
```

temp = (TAT[i] - BT[i]);

WTC[i] = temp;

printf("\n%.d", WTC[i]);

FW

temp = 0;

printf("in The average TAT is %.m");

for(i=0; i<4; i++)
{

temp = temp + TAT[i];

a = temp;

}

~~average~~ a float a = temp/4;

printf("\n%.d", a);

temp = 0;

i6. printf ("In the average waiting time  
in");

for (int i=0; i<4; i++)

{

temp = temp + WT[i];

b = temp;

}

float avg; b =  $\frac{temp}{4}$ ;

printf ("In %d", b);

3. Gantt chart

Output:

| PID            | AT | BT | CT | TAT | WT |
|----------------|----|----|----|-----|----|
| P <sub>1</sub> | 0  | 5  | 5  | 5   | 0  |
| P <sub>2</sub> | 1  | 3  | 8  | 7   | 4  |
| P <sub>3</sub> | 2  | 8  | 16 | 14  | 6  |
| P <sub>4</sub> | 3  | 6  | 22 | 19  | 13 |

$$\text{Avg TAT} = 11.25 \text{ ms}$$

$$\text{Avg WT} = 5.75 \text{ ms}$$

2)

# include &lt; stdio.h &gt;

void swap ( int \*a , int \*b )

{

\*a = \*a + \*b ;

\*b = \*a - \*b ;

\*a = \*a - \*b ;

}

void sort ( int pid , int \*at , int \*bt , int s , int n )

{

for ( int i = s ; i &lt; n ; i++ ) {

for ( int j = s ; j &lt; n ; j++ ) {

if ( at [ i ] &lt; at [ j ] ) {

{

swap ( &amp; at [ i ] , &amp; at [ j ] );

swap ( &amp; bt [ i ] , &amp; bt [ j ] );

swap ( &amp; pid [ i ] , &amp; pid [ j ] );

}

}

{

}

void sortb ( int \*pid , int \*at , int \*bt , int s , int n )

{

for ( int i = s ; i &lt; n ; i++ ) {

for ( int j = s ; j &lt; n ; j++ ) {

if ( bt [ i ] &lt; bt [ j ] ) {

swap ( &amp; at [ i ] , &amp; at [ j ] );

swap ( &amp; bt [ i ] , &amp; bt [ j ] );

swap ( &amp; pid [ i ] , &amp; pid [ j ] );

}

{

{

{

112 38  
int main ()

{

    int n;  
    printf("Enter the no of process");  
    scanf("%d", &n);  
    int pid[n], at[n], bt[n], ct[n], tat[n],  
        wt[n];

    for (int i=0; i<n; i++)

{

        printf("Enter arrival time & burst time");  
    }

        scanf("%d %d", &aci, &bt[i]);  
        pid[i] = i+1;

}

    sort(pid, at, bt, 0, n);

    int c = at[0] + bt[0];

    ct[0] = c;

    for (int i=1; i<n; i++)

{

        int t = i;

        int x = i;

        while (at[x] < c)

{

            t += 1;

            x += 1;

}

        sort b (pid, at, bt, x, t);

        if (at[x] > c)

            c = at[x];

100%

(3)

```
for (n; x < t; n++)
{
    ct[n] = c + bt[n];
    c = ct[n];
}
```

```
for (int i=0; i<n; i++)
{
    tat[i] = ct[i] - at[i];
    wt[i] = tat[i] - bt[i];
}
float avg_tat = 0;
float avg_wt = 0;
for (int i=0; i<n; i++)
{
    avg_tat += tat[i];
    avg_wt += wt[i];
}
```

Output:

```
for (int i=0; i<n; i++)
{
    printf("%d %d %d %d %d %d\n",
           pid[i], at[i], bt[i], ct[i], tat[i],
           wt[i]);
}
printf("\n");
```

```
printf("Average Turn Around Time : %.2f", avg_tat/n);
```

```
printf("Average Waiting Time : %.2f", avg_wt/n);
```

## Output

Enter number of processes:

Enter arrival time & Burst time: 2 1

Enter arrival time & Burst time: 3 5

Enter arrival time & Burst time: 4 1

Enter arrival time & Burst time: 0 6

Enter arrival time & Burst time: 2 3.

|   |   |   |    |    |    |
|---|---|---|----|----|----|
| 4 | 0 | 6 | 6  | 6  | 0  |
| 1 | 2 | 1 | 7  | 5  | 4  |
| 3 | 4 | 1 | 8  | 4  | 3  |
| 5 | 2 | 3 | 11 | 9  | 6  |
| 2 | 1 | 5 | 16 | 15 | 10 |

Average Turn around Time = 7.8000.

Average Waiting time: 4.6000

Enter arrival time.

(Round Robin)

0 1 2 4.

Enter burst time.

5 4 2 1

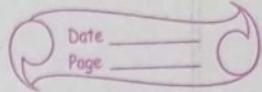
| PID | AT | BT | CT | TAT | WT | RT |
|-----|----|----|----|-----|----|----|
| P1  | 10 | 0  | 5  | 5   | 0  | 0  |
| P2  | 10 | 1  | 12 | 11  | 7  | 7  |
| P3  | 10 | 2  | 18 | 8   | 4  | 4  |
| P4  | 10 | 4  | 16 | 6   | 2  | 1  |

Avg TAT = 6.0ms

WT avg : 3.0ms.

15/5/24

## Round Robin.



#include <stdio.h>.

#include <stdlib.h>.

```
struct queue {
```

```
    int pid;
```

```
    struct queue* next;
```

```
};
```

```
struct queue *rq = NULL;
```

```
struct queue *create (int p)
```

```
{
```

```
    struct queue *nn = malloc (sizeof (struct queue));
```

```
    nn->pid = p;
```

```
    nn->next = NULL;
```

```
    return nn;
```

```
}.
```

```
void enqueue (int p)
```

```
{
```

```
    struct queue *nn = create (p);
```

```
    if (rq == NULL)
```

```
{
```

```
        rq = nn; // empty q added.
```

```
}
```

```
else {
```

```
    struct queue *temp = rq;
```

```
    while (temp->next != NULL)
```

```
    { temp = temp->next; }.
```

```
    temp->next = nn;
```

```
}
```

```
}
```

```

int dequeue()
{
    int x=0;
    if (rq==NULL)
    {
        return x; //empty
    }
    else
    {
        struct queue *temp = rq;
        rq = temp->pid;
        rq = rq->next;
        free(temp);
        return x;
    }
}

```

```

void printq()
{
    struct queue *temp=(struct queue *)malloc(sizeof(struct queue));
    while (temp != NULL)
    {
        printf("%d\n", temp->pid);
        temp = temp->next;
    }
}

```

~~if (temp == NULL)~~  
~~return;~~  
~~else~~  
~~temp = temp->next;~~

```
void swap(int *a, int *b)
```

{

$*a = *a + *b;$

$*b = *a - *b;$

$*a = *a - *b;$

}

```
void sort(int pid[], int at[], int bt[], int tn)
```

{

for (int i=0; i<n; i++)

for (int j=0; j<n; j++)

if (at[i] < at[j])

swap(&at[i], &at[j]);

swap(&bt[i], &bt[j]);

swap(&pid[i], &pid[j]);

}

int main()

{

int n; t, tq;

printf("Enter the no. of processes");

scanf("%d", &n);

printf("Enter the time quantum");

scanf("%d", &t);

int pid[n], at[n], bt[n], ct[n], ta[n],  
wt[n] bt2[n], rt[n];

```

for (int i=0; i<n; i++) {
    cout << "Enter the arrival time: ";
    cin >> a[i];
    cout << "Enter the burst time: ";
    cin >> bt[i];
    pid[i] = i+1;
}

```

```

call sort (pid, at, bt, n);
enqueue (pid[0]);
for (int i=0; i<n; i++) {
    bt[2][i] = bt[i];
    pid[2][i] = -1;
}
int count = 0;
int cvar = at[0];
while (count != n) {
    int curp = rq->pid;
    int curi = 0;
    for (int i=0; i<n; i++) {
        if (pid[i] == curp) {
            if (curi == i)
                break;
        }
    }
}

```

~~if (rt[Curi] == -1)~~

~~rt[Curi] = CVar - at[Curi];~~

~~3~~

~~if (bt2[Curi] <= t) {~~

~~CVar += bt2[Curi];~~

~~bt2[Curi] = 0;~~

~~3~~

```
else { ctvar+=t; bt2[curi]=t; }

while (ct[n]<=ctvar && n<=n) {
    enqueue(pid[n]);
    n+=1;

    if (bt2[curi]>0) { enqueue(pickcuri);}

    if (bt2[curi]==0) { count+=1;
        ct[curi]=ctvar;
        dequeuc();
    }

    for (int i=0; i<n; i++) {
        tot[i] = ct[i]-at[i];
        wt[i] = tot[i]-bt[i];
    }

    float avg_tat=0;
    float avg_wt=0;

    for (int i=0; i<n; i++) {
        avg_tat+=tot[i];
        avg_wt+=wt[i];
    }

    printf("pid\tat\tbt\tct\ttat\twt\n");
    printf("%d\t%d\t%d\t%d\t%.2f\t%.2f\n", i, at[i], bt[i], ct[i], avg_tat/n, avg_wt/n);
}
```

```

printf("At %d & it %d it %d
it %d ", p[i], a[i],
b[i], c[i], e[i]; w[i],
r[i]);
printf("\n");
}

```

```

printf("Average tat = %f", avg_tat);
printf("Avg WT = %f", avg_wt);
return 0;
}

```

### Rate monotonic OPR,

OPR.

Enter no of processes : 2.

Enter the CPU burst time : 100 ms

20 35.

Enter Time period = 100 ms

50 : 100 + 100 = 100 ms

Note monotonic scheduling

PED = 80 ms period.

1 20 50

2 35 100

$$0.7500 \leq 0.8248 \Rightarrow \text{true}$$

0 ms forward P1 is running

20 ms forward process 2 is running,

50 ms " Process 1 is running

70 ms " " Process 2 is running

75 " " CPU is idle.

## Rate Monotonic Scheduling.

```
#include <stdlio.h>
```

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

```
#include <math.h>
```

```
void sort (int proc[], int b[], int p[]  
, int n)
```

{

```
    int temp = 0;
```

```
    for (int i = 0; i < n; i++) {
```

```
        for (int j = 0; j < n; j++) {
```

```
            if (p[i] < p[j]) {
```

```
                temp = p[i];
```

```
                p[i] = p[j];
```

```
                p[j] = temp;
```

```
                b[i] = b[j];
```

```
b[j] = temp;
```

```
b[i] = temp;
```

```
temp = proc[i];
```

```
proc[i] = proc[j];
```

```
proc[j] = temp;
```

```
int gcd (int a, int b)
```

{

```
    int r;
```

```
    while (b > 0) {
```

```
        r = a % b;
```

```
a = b;
```

```
b = r;
```

```
} return a;
```

```

int lcm(int p[0], int n) {
    int LCM = p[0];
    for (int i=1; i<n; i++) {
        LCM = ((LCM * p[i])) / gcd(LCM, p[i]);
    }
    return LCM;
}

```

```

void main() {
    int n;
    printf("Enter no of processes ");
    scanf("%d", &n);
    int proc[n], b[n], pt[n], sum[n];
    printf("Enter CPU Burst times:\n");
    for (int i=0; i<n; i++) {
        scanf("%d", &b[i]);
        sum[i] = b[i];
    }
    printf("Enter the no of time periods:\n");
    for (int i=0; i<n; i++) {
        scanf("%d", &pt[i]);
        proc[i] = i + 1;
    }
    sort(proc, b, 10, n);
    int L = lcm(pt, n);
    printf("LCM = %d\n", L);
    printf("In Rate Monotonic Scheduling");
    printf(" PJD at Burst Time it Period");
    for (int i=0; i<n; i++) {
        printf("\n%d at %d", proc[i], b[i]);
    }
}

```

```

double sum = 0.0;
for (int i=0; i<n; i++) {
    sum += (double) b[i] / pt[i];
}
double rhs = n * pow(2.0, (1.0/n)-1.0);
printf("n/4 (= %f => %.5fn",
       sum, rhs, sum <= rhs ? "True"
                                : "False");
if (sum > rhs) exit(0);

```

printf("scheduling occurs for c.d ? h),

```

int time = 0; prev = 0; n = 0;
while (true (h)) {
    intj = 0;
    for (int i=0; i<n; i++) {
        if (c.time[i].pt[i] == 0)
            rem[i] = b[i];
        if (rem[i] > 0) {
            if (prev != proc(i)) {
                printf("c.d was moved. proc
                       c.d running\n");
                stime, proc(i);
            }
            prev = proc(i);
        }
    }
    rem[i] = -;
    if (i == 1)
        break;
    n = 0
}

```

3

if ( $\neg f$ )  $\vdash \{$

if ( $2 \leq i \leq 1$ ) {

print `C` i. d ns `onval`:

CPU is 2. min; time;

$u=1$ ; extra  $\lambda$  part

3. *Amazilia rutilans* (L.)

3: 55 2002 11/02 2002

∴ time + t,

3. *(3) Una cada una)*

3

CA is the only country producing biogas

Enter the no of process : 2

Enter the CPU's name to search

3-4) no likes

Lo Hui

3. 11. 1998; 09:00 h) ab

(cont'd) by author's file

Individus

## Section 6

3. (iii)  $\sin^{-1}(-\frac{1}{2})$  (iv)  $\tan^{-1}(\sqrt{3})$

*Lepturus* and *Lepturus* (*Lepturus*) *lepturus* Lour.

~~difficulty~~ ~~in~~ ~~the~~ ~~way~~

(iii) ~~newspaper~~

160200-1100

2000 = 60 miles

卷之三

Ward

卷之八

15/5/24

## Priority Scheduling

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

```
void swap(int *a, int *b)
{
    *a = *a + *b;
    *b = *a - *b;
    *a = *a - *b;
}
```

```
or void sort (int pid[], int *at, int *bt, int *prior, int n) {
    for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++) {
            if (at[i] < at[j]) {
                swap(&at[i], &at[j]);
                swap(&bt[i], &bt[j]);
                swap(&pid[i], &pid[j]);
                swap(&prior[i], &prior[j]);
            }
        }
    }
}
```

1. init. location (i) (j)  
 2. swap (i, j)

```

int highest_priority (int *prior, fint, int)
{
    int u = prior[0];
    int j = 0;
    for (int i = 1; i < e; i++)
    {
        if (prior[i] > u)
        {
            u = prior[i];
            j = i;
        }
    }
    return j;
}

```

```

int main ()
{
    int n, t;
    printf ("Enter the no of processes: ");
    scanf ("%d", &n);
    int pid[n], at[n], bt1[n], ct[n], tat[n],
        wt[n], bt2[n], rt[n], prior[n];
    for (int i = 0; i < n; i++)
    {
        printf ("Enter arrival time, burst
                time , priority : ");
        scanf ("%d %d %d", &at[i], &bt[i],
               &prior[i]);
        pid[i] = i + 1;
    }
}

```

sort grid, atbt, sort, n);

for (int i=0; i<n; i++)

{

    bt2[i] = bt[i];

    rt[i] = -1;

    if (arr[0] == count) { front = arr[0], curi = 0;

        while (count != n) {

            if (front < curi) {

                rt[curi] = front - arr[front];

                if (curi == n) {

                    ct[front] = bt2[curi]; bt2[curi] = 0;

            } else {

                ct[front] = 1; bt2[curi] = 1;

                for (int j=i+1; arr[j] <= ct[front]; j++)

            {

                arr[j] += 1; n = j;

            }

            if (bt2[curi] == 0) {

                count += 1; ct[curi] = ct[front];

                rt[curi] = -1;

        }

    for (int i=0; i<n; i++) {

        fat[i] = ct[i] + arr[i];

        wt[i] = fat[i] - bt2[i];

}

516124.

```

for(int i=0; i<n; i++){
    avg = fat + wt[i];
    avg - wt[i];
}

```

points ("in mid vt at \t best ct vt  
-at vt vt (e.vt in in");

```
for (int i = 0; i < n; i++) {
```

printf("In %d it %d t %d go %d it %d  
it %d it %d it %d ");  
scanf("%d", &i), at(i), bt(i), ct(i),  
fat(i), wtc(i), ntc(i);

printf("In Avg fat = %f", avg = fath);  
printf("In Avg w/T = %f", avg - wt  
h);

return 0; /\* + imp \*/

1

Enter the number of moves : 3.

Entry priorities : 10 2020.30

Enter arrival time: 0, 1, 2, 4.

Enter burst time: 5 4 2 1

| PID | Prior | AT | BT | CT | T4T | WT | RP |
|-----|-------|----|----|----|-----|----|----|
| P1  | 10    | 0  | 5  | 12 | 12  | 7  | 0  |
| P2  | 20    | 1  | 4  | 8  | 7   | 3  | 0  |
| P3  | 20    | 2  | 2  | 4  | 2   | 0  | 0  |
| P4  | 40    | 4  | 1  | 5  | 1   | 0  | 0  |

$$\text{Avg TA } \bar{T} = 8.5 \text{ min}$$

$$\text{Ag} \cdot \text{W} = 2.7 \text{ mJ}$$

516124.

## Earliest Deadline First

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

```
void sort (int proc[], int d[], int b[],  
          int pt[], int n);
```

{

```
    int temp = 0;  
    for (int i = 0; i < n; i++) {  
        for (int j = i + 1; j < n; j++) {  
            if (d[j] < d[i]) {
```

$d[j] = d[i]$ ;

$d[i] = temp$ ;

$temp = pt[i]$ ;

$pt[i] = pt[j]$ ;

$pt[j] = temp$ ;

$b[j] = b[i]$ ;

$b[i] = temp$ ;

$temp = proc[i]$ ;

$proc[i] = pt[proc[j]]$ ;

$proc[j] = temp$ ;

3.

```
int gcd (int a, int b) {
    int r;
    while (b > 0) {
        r = a % b;
        a = b;
        b = r;
    }
    return a;
}
```

```
int lcm (int p[ ], int n) {
    int LCM = p[0];
    for (int i = 1; i < n; i++) {
        LCM = (LCM * p[i]) / gcd(LCM,
                                     p[i]);
    }
    return LCM;
}

void main () {
    int n;
    printf("Enter the no of process:");
    scanf("%d", &n);
    int proc[n], b[n], p[n], d[n];
    i = 0;
    printf("Enter the processes:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &b[i]);
        proc[i] = b[i];
        p[i] = 0;
        d[i] = 0;
    }
}
```

printf("Enter the deadline[n]:");

for (int i=0; i<n; i++) { scny[i]=0,  
& d[i]);}

printf("Enter the arrival periods");

for (int i=0; i<n; i++)

{ scny[i] = spec[i]; }

for (int i=0; i<n; i++) preci[i]=i+1;

sort(precd, &b, pt, n);

int L = larrival(pt, n);

printf("In Earliest deadline scheduling

printf("PJD At B + Lt 'Deadline' in  
period \n");

for (int i=0; i<n; i++)

printf("\n Y.d Lt Lt &d Lt &c  
pro[i], pb[i], dc[i], ptc[i]);

printf("Scheduling occurs for jobs in ");

(i.e. arrival = 0, prec = 0, n = 0;

int npd[n];

for (int i=0; i<n; i++) {

if (arr[i] <= d[i]) { npd[i] = 1; } else { npd[i] = 0; }

printf("No. of jobs completed is %d", npd);

```

while (time < 1) {
    for (int i = 0; i < n; i++) {
        if (time % p[i] == 0 &&
            time != 0) {
            ncl[i] = time + dc[i];
            rem[i] = b[i];
        }
    }
    int mcl = t;
    int t += 1;
    if (t >= n) {
        if (rem[0] > 0 && ncl[0] < mcl)
            mcl = ncl[0];
        t -= 1;
    }
    if (t <= -1) {
        printf("At %d ms : total %d ms running in\n"
               "proc , proc %d);\n",
               rem[TLE] = rem[TLE] - 1;
        if (rem[TLE] == 0)
            printf("At %d ms : CPU is idle\n",
                   time);
    }
    time++;
}

```

output. b) Longest

Enter the no of processes: 3.

Enter CPU BT:

3 2 2.

Enter deadline

7 4 8.

Enter the periods.

20 5 10

### Earliest deadline scheduling.

| PID | BT | Deadline | Period |
|-----|----|----------|--------|
| 2   | 2  | 9 ns     | 5      |
| 1   | 3  | 7 ns     | 20     |
| 3   | 2  | 8        | 10     |

Now we start to run our algorithm

Scheduling access for 20ms.

0 ns : 2

1 ns : 1

2 : 1

3 : 1

4 : 3

5 : 3

6 : 2

7 : 2

8 : CPU is idle

9 : 2

10 : 2

11 : 3

12 : CPU is idle

13 : CPU is idle

14 : CPU is idle

15 : CPU is idle

16 : CPU is idle

17 : CPU is idle

18 : CPU is idle

19 : CPU is idle

20 : CPU is idle

21 : CPU is idle

22 : CPU is idle

23 : CPU is idle

24 : CPU is idle

25 : CPU is idle

26 : CPU is idle

27 : CPU is idle

28 : CPU is idle

29 : CPU is idle

30 : CPU is idle

31 : CPU is idle

32 : CPU is idle

33 : CPU is idle

34 : CPU is idle

35 : CPU is idle

36 : CPU is idle

37 : CPU is idle

38 : CPU is idle

39 : CPU is idle

40 : CPU is idle

41 : CPU is idle

42 : CPU is idle

43 : CPU is idle

44 : CPU is idle

45 : CPU is idle

46 : CPU is idle

47 : CPU is idle

48 : CPU is idle

49 : CPU is idle

50 : CPU is idle

51 : CPU is idle

52 : CPU is idle

53 : CPU is idle

54 : CPU is idle

55 : CPU is idle

56 : CPU is idle

57 : CPU is idle

58 : CPU is idle

59 : CPU is idle

60 : CPU is idle

61 : CPU is idle

62 : CPU is idle

63 : CPU is idle

64 : CPU is idle

65 : CPU is idle

66 : CPU is idle

67 : CPU is idle

68 : CPU is idle

69 : CPU is idle

70 : CPU is idle

71 : CPU is idle

72 : CPU is idle

73 : CPU is idle

74 : CPU is idle

75 : CPU is idle

76 : CPU is idle

77 : CPU is idle

78 : CPU is idle

79 : CPU is idle

80 : CPU is idle

81 : CPU is idle

82 : CPU is idle

83 : CPU is idle

84 : CPU is idle

85 : CPU is idle

86 : CPU is idle

87 : CPU is idle

88 : CPU is idle

89 : CPU is idle

90 : CPU is idle

91 : CPU is idle

92 : CPU is idle

93 : CPU is idle

94 : CPU is idle

95 : CPU is idle

96 : CPU is idle

97 : CPU is idle

98 : CPU is idle

99 : CPU is idle

100 : CPU is idle

101 : CPU is idle

102 : CPU is idle

103 : CPU is idle

104 : CPU is idle

105 : CPU is idle

106 : CPU is idle

107 : CPU is idle

108 : CPU is idle

109 : CPU is idle

110 : CPU is idle

111 : CPU is idle

112 : CPU is idle

113 : CPU is idle

114 : CPU is idle

115 : CPU is idle

116 : CPU is idle

117 : CPU is idle

118 : CPU is idle

119 : CPU is idle

120 : CPU is idle

121 : CPU is idle

122 : CPU is idle

123 : CPU is idle

124 : CPU is idle

125 : CPU is idle

126 : CPU is idle

127 : CPU is idle

128 : CPU is idle

129 : CPU is idle

130 : CPU is idle

131 : CPU is idle

132 : CPU is idle

133 : CPU is idle

134 : CPU is idle

135 : CPU is idle

136 : CPU is idle

137 : CPU is idle

138 : CPU is idle

139 : CPU is idle

140 : CPU is idle

141 : CPU is idle

142 : CPU is idle

143 : CPU is idle

144 : CPU is idle

145 : CPU is idle

146 : CPU is idle

147 : CPU is idle

148 : CPU is idle

149 : CPU is idle

150 : CPU is idle

151 : CPU is idle

152 : CPU is idle

153 : CPU is idle

154 : CPU is idle

155 : CPU is idle

156 : CPU is idle

157 : CPU is idle

158 : CPU is idle

159 : CPU is idle

160 : CPU is idle

161 : CPU is idle

162 : CPU is idle

163 : CPU is idle

164 : CPU is idle

165 : CPU is idle

166 : CPU is idle

167 : CPU is idle

168 : CPU is idle

169 : CPU is idle

170 : CPU is idle

171 : CPU is idle

172 : CPU is idle

173 : CPU is idle

174 : CPU is idle

175 : CPU is idle

176 : CPU is idle

177 : CPU is idle

178 : CPU is idle

179 : CPU is idle

180 : CPU is idle

181 : CPU is idle

182 : CPU is idle

183 : CPU is idle

184 : CPU is idle

185 : CPU is idle

186 : CPU is idle

187 : CPU is idle

188 : CPU is idle

189 : CPU is idle

190 : CPU is idle

191 : CPU is idle

192 : CPU is idle

193 : CPU is idle

194 : CPU is idle

195 : CPU is idle

196 : CPU is idle

197 : CPU is idle

198 : CPU is idle

199 : CPU is idle

200 : CPU is idle

201 : CPU is idle

202 : CPU is idle

203 : CPU is idle

204 : CPU is idle

205 : CPU is idle

206 : CPU is idle

207 : CPU is idle

208 : CPU is idle

209 : CPU is idle

210 : CPU is idle

211 : CPU is idle

212 : CPU is idle

213 : CPU is idle

214 : CPU is idle

215 : CPU is idle

216 : CPU is idle

217 : CPU is idle

218 : CPU is idle

219 : CPU is idle

220 : CPU is idle

221 : CPU is idle

222 : CPU is idle

223 : CPU is idle

224 : CPU is idle

225 : CPU is idle

226 : CPU is idle

227 : CPU is idle

228 : CPU is idle

229 : CPU is idle

230 : CPU is idle

231 : CPU is idle

232 : CPU is idle

233 : CPU is idle

234 : CPU is idle

235 : CPU is idle

236 : CPU is idle

237 : CPU is idle

238 : CPU is idle

239 : CPU is idle

240 : CPU is idle

241 : CPU is idle

242 : CPU is idle

243 : CPU is idle

244 :

121692N

## Proportional Code.

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

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

```
#include <math.h>
```

```
int main()
```

```
{ int n, SOT = 0;
```

```
printf("Enter the no of processes : ");
```

```
scanf("%d", &n);
```

```
int pid[n], L[n+1];
```

```
L[0] = 0;
```

```
printf(" Enter the no of tickets in each  
process");
```

```
for (int i=0; i<n; i++)
```

```
{
```

```
printf("\nPID %d:", i+1);
```

```
scanf("%d", &pid[i]);
```

```
SOT += pid[i];
```

```
L[i+1] = pid[i];
```

```
};
```

```
int t=1;
```

```
int sum = SOT;
```

```
for (int i=0; i<n; i++)
```

```
{
```

```
printf (" In probability of servicing process
```

```
%d is %d %.10%.10% \n", i+1,
```

```
(pid[i]/100)/SOT);
```

```
}
```

```
 srand (time(NULL));  
 while (sum > 0) {  
     int x = rand () % SOT;  
     int j;  
     for (j = 0; j < n; j++) {  
         if (x < t[j])  
             {  
                 printf ("%d ns : servicing Tickets of  
processes %d\n", t, j+1);  
                 pid [j] = j;  
                 sum -= j;  
                 f++; break;  
             }  
     }  
     for (int i = 0; i < n; i++) {  
         if (pid [i+1] == -1) {  
             printf ("%d finished executing").  
         }  
     }  
 }.
```

output.

:(((1111) and) lenarz  
:(0 or 2) elthen

Enter the number of processes : 2

Enter the number of tickets for each process.

PSD 1 = 10

PSD 2 = 20.

Probability of servicing process 1 is 33%.

Probability of servicing process 2 is 66%.

1ms: Servicing Tickets of process 2.

2ms: servicing Tickets of process 2.

3ms: Servicing Tickets of process 2.

4ms: Servicing Tickets of process 2.

5ms: Servicing Tickets of process 2.

Operations being performed

12/16/21

Date \_\_\_\_\_  
Page \_\_\_\_\_

## Producer consumer problem

```
#include <stdio.h>
int muter = 1, full = 0, empty = 7, n = 0;
int main()
{
    int n;
    void producer(); (a) producer
    void consumer(); (b) consumer
    int wait(int); (3rd time)
    int signal(int); (3rd time)
    printf("\n1. Producer\n2. Consumer\n3. Exit");
    while (1)
    {
        printf("Enter your choice:");
        scanf("%d", &n);
        switch (n)
        {
            case 1: (1st time) if (muter == 1 && empty != 0)
                producer();
                else
                    printf("Buffer is full");
                break;
            case 2: (2nd time) if (muter == -1 && full != 0)
                consumer();
                else
                    printf("Buffer is empty!!");
                break;
        }
    }
}
```

case 3: holding resource counter

```
printf("In Number of Items remaining  
in buffer: %d\n", x);
```

```
semif(0); // Null semaphore for
```

```
break;
```

```
}
```

```
}
```

```
return (0);
```

```
}
```

int wait (int s)

```
{ ("Wait") & (empty). Sem / counting. If full, skip
```

```
return (-s);
```

```
}
```

int signal (int s)

```
{ ("Signal") every related w/ it's waiting
```

```
return (+s); } // by "final
```

```
}
```

void producer ()

```
muten = wait (muten);
```

```
full = signal (full);
```

```
empty = wait (empty);
```

```
n++;
```

```
printf("producer produces the item %d ", x);
```

```
muten = signal (muten);
```

```
}
```

(0-1) (1-0) - return if

void consumer ()

```
muten = wait (muten);
```

```
full = wait (full);
```

```
empty = signal (empty);
```

privileges in consumer consumes items %d

%--;

mutex = signal ( mutex );

}

1: Producer :

2: Consumer . ?

3: Exit .

Enter your choice : 1.

producer produces item 1.

Enter your choice : 1.

producer produces item 2.

Enter your choice : 1.

producer produces item 3.

Enter your choice : 1.

producer produces item 7. Enter your choice : 3.

Enter your choice : 1.

Buffer is full !!! No of items rem-

Enter your choice : 2. coming in Buffer =

Consumer consumes item 7.

Enter your choice : 2.

Buffer is empty !!!

9/6/2h

## Bankers Algorithm

```
#include <stdio.h>
#include <stdlib.h>
#define NUM_Processes 5
#define NUM_Resources 3
```

```
int available [NUM_Resources];
int maximum [NUM_Processes][NUM_Resources];
int allocation [NUM_Processes][NUM_Resources];
int need [NUM_Processes][NUM_Resources];

void calculateGED () {
    for (int i=0; i<NUM_Processes; i++) {
        for (int j=0; j<NUM_Resources; j++) {
            need[i][j] = maximum[i][j] - allocation[i][j];
        }
    }
}
```

```
bool isSafe () {
    int work [NUM_RESOURCES];
    bool finish [NUM_PROCESSES] = {False};

    for (int i=0; i<NUM_RESOURCES; i++) {
        work[i] = available[i];
    }

    for (int i=0; i<NUM_Processes; i++) {
        if (!finish[i]) {
            for (int j=0; j<NUM_Resources; j++) {
                if (need[i][j] > work[j]) {
                    break;
                }
            }
            if (j == NUM_Resources) {
                finish[i] = True;
            }
        }
    }

    for (int i=0; i<NUM_ProCESSES; i++) {
        if (!finish[i]) {
            return False;
        }
    }

    return True;
}
```

```

while (true) {
    bool found = false;
    for (int i=0; i< NUM_Processes; i++) {
        if (!finish[i]) {
            bool canProceed = true;
            for (int j=0; j < NUM_RESOURCES; j++) {
                if (need[i][j] > work[i][j])
                    canProceed = false;
            }
            if (canProceed) {
                for (int j=0; j < NUM_RESOURCES; j++) {
                    work[j] += allocation[i][j];
                }
                finish[i] = true;
                found = true;
            }
        }
        if (!found) { break; }
    }
    for (int i=0; i< NUM_PROCESSES; i++) {
        if (!finish[i])
            return false;
    }
}

```

return true;

}

bool requestResources (int process; int request)

{

for (int i=0; i< NUM\_Resources; i++)

{

if (request[i] > need [process])

{

printf ("Error: Process has exceeded  
its main claim.\n");

return false;

}

}

for (int i=0; i< NUM\_Resources; i++) {

available [i] -= request [i];

allocation [process][i] += request [i];

need [process][i] -= request [i];

}

if (isSafe ())

printf ("Request granted.\n");

return true;

}

else {

for (int i=0; i< NUM\_Resources; i++) {

available [i] += request [i];

allocation [process][i] -= request [i];

need [process][i] += request [i];

}

printf ("Request cannot be granted. Systems  
would be in an unsafe state.\n");  
return false;

}

}

int main ()  
{

int i, j;

printf ("Enter the Available Resources Vector  
\n"),

for ( i=0; i< NUM\_Resources; i++ ) {  
scanf ("%d", & available[i]);  
}

printf ("Enter the maximum states matrix"),

for ( i=0; i< NUM\_Processes; i++ ) {

for ( j=0; j< NUM\_Resources; j++ ) {

scanf ("%d", & maximum[i][j]);

}

printf ("Enter the Allocation matrix:\n ");

for ( i=0; i< NUM\_Processes; i++ ) {

for ( j=0; j< NUM\_Resources; j++ ) {

{

scanf ("%d", & allocation[i][j]);

}

}

calculate Need();

if(isSafe()) {

printf("The system is in safe state");  
}

else { printf("The system is not in a  
safe state.\n");  
}

int process() {  
int request [NUM\_Resources];  
printf("Enter the no of resources process  
needs for the request (0 to 7-d),"  
NUM\_PROCESS-1);  
scanf("%d", &process);  
printf("Enter the request vector ");  
for (i=0; i< NUM\_Resources; i++)  
scanf("%d", &request[i]);  
requestResource (process, request);  
return 0;  
}

(C) 1/10/2018 11:41 AM

(C) 1/10/2018 11:41 AM

## Output

Enter the available matrix/resource  
matrix

3 3 2

Enter the mom matrix

7 5 3

3 2 2

9 0 2

2 2 2

4 3 3

Enter Allocation matrix.

0 3 0 1 0

2 0 0

3 0 2

2 1 1

0 0 2

Need Matrix.

7 4 3

1 2 2

6 0 0

0 1 1

4 3 1

Process 1 is visited

Process 3 is visited

Process 4 is visited.

Process 0 is visited

Process 2 is visited

The system is in safe state.

Safe sequence.

1 3 4 0 2

Enter the process no for Request(0 to 4):  
Enter the request no vector:

2 0 2

Process 1 is visited (5, 3, P<sub>2</sub>)  
Process 3 is visited (7, 4, 3)  
Process 4 is visited (7, 4, 5)  
Process 0 is visited (7, 5, 5)  
Process 2 is visited (10, 5, 7)

Request Granted.

~~1000~~

0 0 1  
0 0 2  
1 1 0  
1 0 1

1. Allocation is 1 memory

2. Status is in waiting

3. Allocation is 1 memory

4. Status is in ready

5. Allocation is 1 memory

6. Status is in ready

9/7/24

Date \_\_\_\_\_  
Page \_\_\_\_\_

## Deadlock Detection

P.8

#include &lt; stdio.h &gt;

#define MAX\_Processes 10

#define MAX\_Resources 10.

int allocation [MAX\_Processes][MAX\_Resources];

int max\_need [MAX\_Processes][MAX\_Resources];

int avl\_available [MAX\_RESOURCES];

int n\_processes, n\_resources;

void initialize () {

printf ("Enter the no of processes ");

scanf ("%d", &amp;n\_processes);

printf ("Enter no number of resources ");

scanf ("%d", &amp;n\_resources);

printf ("Enter the allocation matrix (%d x %d):\n", n\_processes, n\_resources);

for (i=0; i&lt;n\_processes; i++)

{

for (j=0; j&lt;n\_resources; j++)

scanf ("%d", &amp;allocation[i][j]);

}

3

Enter

```
printf("Enter the maximum need matrix  
(%d x %d): \n", n_processes,  
n_resources);
```

```
for( int i=0 ; i<n_processes; i++ )  
{
```

```
    for( j=0; j<n_resources; j++ )
```

```
        : (scanf("%d", &max_need[i][j]));
```

```
printf("Enter available resources  
vector (%d elements): \n",
```

```
n_resources);
```

```
for( j=0, j<n_resources; j++ )
```

```
{
```

```
    scanf("%d", &available[j]);
```

```
}
```

```
void detect_deadlock() {
```

```
    int i, j, k;
```

```
    int work[MAX_Resources];
```

```
    int finish[MAX_Processes];
```

```
    int safeSequence[MAX_Processes];
```

```
    int safe_sequence_count = 0;
```

```
for (j=0; j<n_resources; j++) {
    work[j] = available[j];
}
```

```
for (i=0; i<n_processes; i++) {
    finish[i] = 0;
```

// Process to allocate to the resources.

```
int found;
do {
```

```
    found = 0; // (b) if
    for (i=0; i<n_processes; i++)
        if (!finish[i]) {
```

```
        int com_allocate = 1;
```

```
        for (j=0; j<n_resources; j++)
```

```
            if (non_need[i][j] - allocation[i][j] >
                work[j])
```

```
{
```

```
            com_allocate = 0;
            break;
```

```
}
```

```
if (com_allocate) {
```

```
if (com-allocate) {  
    for (j=0; j < n-resources; j++)  
    {  
        work[j] = allocation[i][j];  
    }  
    finish[i] = 1;  
    safe sequence (safe sequence - count++) = i;  
    found = 1;  
}
```

```
}  
}  
}  
}  
}
```

```
} while (found);  
for (i=0; i < n-processes; i++) {  
    if (!finish[i]) {  
        printf("Deadlock detected!\n");  
        return;  
    }  
}
```

```
int main()  
{  
    initialize();  
    detect-deadlock();  
    return 0;  
}
```

initial configuration  
output

Enter the processes: 5

Enter the resources: 3

Enter allocation:

~~1 1 1 1 1~~  
~~2 2 2 2 2~~  
~~3 3 3 3 3~~

5 0 3 (XAM) to (XAM) go to tri

3

1 1 1 1 1 (initial)

(initial) 1 1 1 1 1 (initial)

1 1 1 1 1 (initial)

3 (initial)

1 1 1 1 1 (initial)

(initial)

1 1 1 1 1 (initial)

10/7/24

## Contiguous Memory Allocation

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

```
# define MAX 25
```

```
void firstFit (int nb, int nf, int b[], int f[])
{
    int frag[MAX], bf[MAX] = {0}, ff[MAX] = {0};
```

```
    int i, j, temp;
```

```
    for (i = 1; i < nf; i++)
    {
```

```
        for (j = 1; j < nb; j++)
        {
```

```
            if (bf[j] == 1) {
```

~~```
                temp = b[j] = f[i];
```~~~~```
                if (temp >= 0)
```~~~~```
                    ff[i] = j;
```~~~~```
                    frag[i] = temp;
```~~~~```
                    bf[i] = 1;
```~~~~```
                    break;
```~~~~```
}
```~~~~```
3
```~~~~```
3
```~~

```
printf ("In Memory Management scheme  
- First Fit\n");
```

```
printf (" File no : %d File_size : %d Block_no :  
%d Block_size : %d Fragmed\n");
```

```
for ( i=1; i<nf; i++)
```

```
{  
    printf ("%d %d %d %d %d\n",  
           ffc[i], bff[i], frag[i],  
           i, fc[i]);
```

```
if (ffc[i] != 0) {
```

```
    printf ("%d %d %d %d %d\n",  
           ffc[i], bff[i], frag[i]);
```

```
else {
```

```
    printf ("Not allocated\n");
```

```
}
```

```
void bestfit (int nb, int nf, int bc[],  
              int fc[])
```

```
{
```

```
    int frag[MAX], bf[MAX] = {0}, ffrag  
        = {0};
```

```
    int i, j, itemp, lowest = 10000;
```

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

```

for ( i=1; i <= nf; i++ )
    for ( j=1; j <= nb; j++ )
        if ( bfc[j] != 1 ) {
            temp = b[j] - fc[i];
            if ( temp >= 0 && lowest > temp )
                ffc[i] = j;
            lowest = temp;
        }
        frag[i] = lowest;
        bf[ffc[i]] = 1;
        lowest = 10000;
    }

```

printf ("In Memory Management Scheme  
- Best Fit \n");

printf (" File No \t File Size \t Block No  
 \t Block Size \t Fragment \n ");

```

for ( i=1; i <= nf; i++ ) {
    printf ("%d \t %d \t %d \t %d \n", i, fc[i]);
    if ( ffc[i] != 0 ) {
        printf ("%d \t %d \t %d \t %d \n",
            ffc[i], b[ffc[i]], frag[i]);
    }
}

```

```
else {
    printf("Not Allocated \n");
}
```

```
}
```

```
void worstFit (int nB, int nf, int b[],
                int f[])
```

```
{
    int frag[MAX], bf[MAX] = {0}, ff[MAX]
        = {0};
```

```
int 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("In Memory Management Scheme  
- Worst Fit \n");
```

```
printf(" File : no: \t file - size : \t  
Block_no: \t Block_size: \t  
Fragment \n");
```

```
for (int i = 1; i < nF; i++)
```

```
{ printf("%d\t%d\t%d\t", i,  
FC[i]);
```

```
if (FFC[i] != 0) {
```

```
printf("%d\t%d\t%.d\t%d\t%d  
\n", i, FFC[i], b[FFC[i]],  
frag[i]);
```

```
else {
```

```
printf("Not Allocated \n");
```

```
}
```

```
int main()
```

```
{
```

```
int b[MAX], f[MAX], nb, nF;
```

```
printf("Enter the no of blocks: ");
```

```
scanf("%d", &nb);
```

```
printf("Enter the number of files: ");
```

```
scanf("%d", &nF);
```

`printf("In Enter the size of the blocks:-\n")`

```
for (int i = 1; i <= nf; i++)
```

```
    printf("File %d : ", i);  
    scanf("%d", &f[i]);
```

}

```
int b1[MAX], b2[MAX], b3[MAX];
```

```
for (int i = 1; i <= nb; i++) {
```

```
    b1[i] = b[i];
```

```
    b2[i] = b[i];
```

```
    b3[i] = b[i];
```

}

first fit (nb, nf, b1, f)

best fit (nb, nf, b2, f)

worst fit (nb, nf, b3, f)

if return 0, enter temporary number

}

selected address      1512      0x7

2

1

1

1

5

8

9

8

1012 blocks

our blocks

1512?

0x7

F:

E

1

0

2

1

1

0

fit now

printf("In Memory Management Scheme -\nWorst fit"),

(char \*msg) {

    cout << msg;

    cout << endl;

    cout << "File 1 : " << f1 << endl;

    cout << "File 2 : " << f2 << endl;

    cout << "Output" << endl;

#

{

Enter the no of block : 3.

Enter : the no of blocks : 2

Enter the size of the blocks.

b1 : 5

b2 : 2

b3 : 7

Enter size of files.

f1 = 1

f2 = 4

Memory management scheme - first fit.

| F no | f size | blockno | blocksiz |
|------|--------|---------|----------|
| 1    | 1      | 1       | 5        |
| 2    | 4      | 3       | 7.       |

Worst fit

| fno | fsize | block no | blocksiz |
|-----|-------|----------|----------|
| 1   | 1     | 3        | 7        |
| 2   | 4     | 1        | 5.       |

8- do 1

Best fit

| fno | filesize | blockno | blocksize |
|-----|----------|---------|-----------|
| 1   | 1        | 2       | 2         |
| 2   | 5        | 1       | 5         |

at 1, Demand fit, file size 10 at 2nd  
(spare)

(file size 10 at 2nd)

(Demand = 3) demand fit

1st writer

2

3

4 writers

5

at 1, Demand fit, file size 10 at 2nd

at 1, Demand fit, file size 10 at 2nd

(file size 10 at 2nd)

3

3 (1-1-1) (Demand fit)

(Demand fit, 3-3-3 fit)

3 writers

4 writers

5

6

10/7/24

## Lab-8

### Page Replacement Algorithm : LRU

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

```
int isPagePresent (int frames[], int n,  
                   int page);
```

```
{
```

```
    for (int i=0; i<n; i++)
```

```
        if (frames[i] == page)
```

```
{
```

```
        return 1;
```

```
}
```

```
return 0;
```

```
}
```

// Function to print frames.

```
void printFrames (int frames[], int n)
```

```
{
```

```
    for (int i=0; i<n; i++)
```

```
{
```

```
        if (frames[i] != -1) {
```

```
            printf ("%d", frames[i]);
```

```
        }
```

```
        else {
```

```
            printf (" - ");
```

```
}
```

```
}
```

printf("\n");

1.

ii Function to implement FIFO page replacement

```
void fifoPageReplacement (int pages[], int numPages, int numFrames)
{
    int frames [numFrames];
    int front = 0, pageFaults = 0;
```

11 Initialization of frames.

```
for (int i=0; i<numFrames; i++)
{
    frames [i] = -1;
}
```

printf("FIFO Replacement\n");

printf("Reference String \t Frames\n");

for (int i=0; i< numPages ; i++)

```
{
```

printf("%d\t\t", pages[i]);

if (!isPagePresent (frames, numFrames,

pages[i])) {

}

frames [front] = pages[i];

front = (front + 1) % numFrames;

pageFaults++;

printf("Frames: (frames, numFrames);

3

```
cout << "Total page faults : " << pageFaults;
```

3. Function to find the page to replace using the optimal page replacement algorithm

```
int findOptimalReplacementIndex (
```

int pages[], int numPages, int frames[],  
int numFrames, int currentIndex)

```
{
```

```
    int farthest = currentIndex;
```

int index = -1;

```
    for (int i = 0; i < numFrames; i++)
```

```
{
```

```
        if (frames[i] == pages[currentIndex])
```

```
            if (currentIndex > i) {
```

j = currentIndex; j < numPages; j++)

```
                    if (frames[i] == pages[j])
```

```
                        if (j > farthest) {
```

```
                            farthest = j;
```

index = i;

```
}
```

```
        if (index != -1) break;
```

```
    if (index == -1)
```

```
        return index;
```

```
}
```

return (index == -1) ? 0 : index;

// Function to implement Optimal page replacement

```
void optPageReplacement (int pages[],  
                        int numPages, int numFrames)
```

{

int frames[numFrames];  
 int pageFaults = 0;

// Initialize frames.

```
for (int i=0; i< numFrames; i++)
```

{

}

printf("Optimal Replacement\n");

printf("Reference String \tFrames\n");

```
for (int i=0; i< numPages; i++)
```

{

printf("%d\t", pages[i]);

~~if (!isPagePresent(frames, numFrames,  
 pages[i])) {~~

~~if (!isPagePresent(frame, numFrames,  
 pages[i])) {~~

~~p~~

~~-1~~

```
if (frames[i] == -1) {  
    frames[i] = pages[i];  
    break;  
}
```

}

```
else {
```

```
    int index = FindOptimalReplacementIndex(pages, numPages,  
  frames, numFrames, i+1);  
    frames[index] = pages[i];
```

}

```
pageFaults++;
```

}

```
printf("frames : %d\n",
```

}

```
printf("\nTotal page Faults : %d\n",  
      pageFaults);
```

};

// Function to implement LRU page replacement

```
void lruPageReplacement(int pages[], int  
                        numPages, int numFrames)
```

{

```
    int frame, numFrames;
```

```
    int pageFaults = 0;
```

```
    int timestamps[numFrames];
```

```
for (int i=0; i < numFrames; i++) {
```

```
    frames[i] = -1;
```

```
    timestamps[i] = -1;
```

```
}
```

```
printf("LRU Replacement\n");
```

```
printf("Preference String \tframes\n");
```

```
for (int i=0; i < numPages; i++) {
```

```
    printf("%d \t\t", pages[i]);
```

```
    if (!isPagePresent(frames, numFrames, pages))
```

```
{
```

```
    int freeIndex = 0;
```

```
    for (int j=0; j < numFrames; j++)
```

```
{
```

```
        if (timestamps[j] < timestamps[freeIndex])
```

```
[freeIndex] = j;
```

```
    freeIndex = j;
```

```
}
```

```
    frames[freeIndex] = pages[i];
```

```
    timestamps[freeIndex] = i;
```

```
    pageFaults++;
```

```
}
```

```
else {
```

```
for (int j = 0; j < numFrames; j++)
```

```
{
```

```
    if (frames[j] == pages[i])
```

```
{
```

```
    timeStamps[j] = i;
```

```
    break;
```

```
}
```

```
if (i >= numPages - 1) {
```

```
    printf("Total page faults : %d\n",
```

```
        pageFaults);
```

```
}
```

```
else {
```

```
    printf("Page %d is in frame %d at time %d\n",
```

```
        i + 1, frames[timeStamps[i]], timeStamps[i]);
```

```
    pageFaults++;
```

```
}
```

```
int main()
```

```
{
```

```
    int numFrames, numPages;
```

```
    printf("Enter the number of frames: ");
```

```
    scanf("%d", &numFrames);
```

```
    printf("Enter the number of pages: ");
```

```
    scanf("%d", &numPages);
```

```
    int pages[numPages];
```

```
    printf("Enter the reference string: ");
```

```
for (int i=0; i<numPages; i++)
```

leftmost scanf("%d", &pages[i]),  
if pages[i] > 0 then

fifo Page Replacement (pages, numPages,

Opt Page Replacement (pages; numPages,  
numFrames);

drudgeRepacement (page, numPages, numRows)

return 0;

ressssssssssssssssssss

Output : C C S S S 1 1 1 -

Enter no of frames = 3. ~~1000~~ ~~1000~~

enter the no of pages : 20

Entered Job Reference string: 7012300423032

00000 0 000 000 0 000 000 0 0 -

~~CCW~~ FIFO ~~SSSS C S EEE IIII -~~

~~201~~ 201 203 042 303 213  
~~9772224440000.0~~  
- 000  
- 111

FIFO

10120304230321201701  
7772222444000000077  
-00003332222111100  
10003333222111100

page faults = 15

optimal

10120304230321207  
777222222222227  
-00000004440000000  
-11133333333311111

page faults = 9

LRU

10120304230321201701  
777222244400011111  
-00000000033333300000  
-11133322222222777