INDEX

Name : Mukund. R.S Class : 4 C

Section : C Roll No. : Subject : OS Lab observation

Sl. No.	Date	Title	Page No.	Teacher's Sign. / Remarks
1	8/5	Lab 1 STF, FCFS		
2	10/5	Lab 2 Round Robin, priority		
3	5/6	Lab 3 Multiqueue, Rotemono Eauly, proportnd		
4	5/6.	Lab 4 proportional Scheduling.		
5	12/6	Lab -5 consumer, driving problem		
6	19/6	Lab -6 Bankers algorithm		
7	3/7	Lab -7 deadlock detection, memory		
8	10/7	Lab -8 page replacement algo.		

complete

Lab-1

Date
Page

① Write a C program to simulate the following CPU scheduling algorithms.

→ FCFS

→ SJF

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

int *mint(int *AT, int n) {
    int mini = AT
    int *arr = (int *)malloc(2*sizeof(int))
    arr[0] = mini;
    arr[1] = 0;

    for(int i=0; i<n; i++) {
        if(AT[i] < mini && AT[i] != -1)
            mini = AT[i];
            arr[0] = mini;
            arr[1] = i;
            AT[i] = -1;
    }
}
return arr;
}

int *CTA(int * AT, int *BT, char PID[][4]
    int *CT = (int *) malloc
    char GT[n][2]
    int time = 0;
```

```
    for(int i=0; i<n; i++){
        int *mini = mint(AT, n)
        strcpy(GT[i], PID[mini[i]]);
        time += BT[mini[i]];
        CT[i] = time;
        AT[mini[i]] = -1
        free(mini);
    }
    return CT;
}

int main(){

    int AT[4], BT[4];
    char PID[4][2];

    AT = {0, 1, 5, 6};
    BT = {2, 2, 3, 4};
    PID = {"P1", "P2", "P3", "P4"};

int *CT = CTA(AT, BT, PID, n);

    for(int i = 0; i<n; i++){
        printf("CT %s: %d\n", PID[i]
    }
    free(CT);

    return 0;

}
```

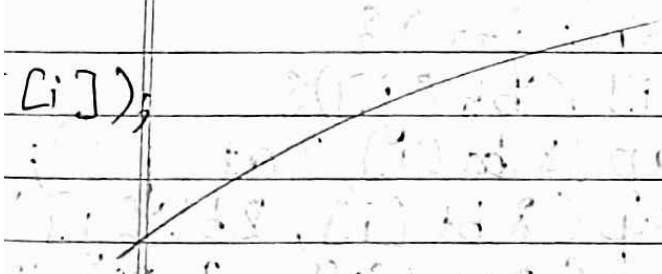
Output

CT P1 : 2

CT P2 : 4

CT P3 : 8

CT P4 : 12



Lab-2

```
#include <stdio.h>

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

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

void sortb (int *pid, int *at, int *bt, ints
    for (i=0; i<n; i++) {
        for (j=s; j<n; j++) {
            if (bt[i] < bt[j]){
                swap (&bt[i], &at[j]);
                swap (&bt[i], &bt[j]);
                swap (&pid[i], &pid[j]);
            }
        }
    }
}.
```

```
int main() {
    int n;
    int pid[n], at[n], bt[n], ct[n], tat[n
    for(int i=0; i<n; i++) {
        scanf("%d %d", &at[i], &bt[i]);
        pid[i] = i+1;
    }
    sort(pid, at, bt, 0, n);
ints) {  int c = at[0] + bt[0];
    ct[0] = c;
    for(int i=1; i<n; i++) {
        int t = i;
        int x = 1;
        while(at[i] < c) {
            t++;
            x++;
        }
        sortb(pid, at, bt, x, t);
        if(at[x] > 0) c = at[x];
        for(x; x<t; x++) {
,int n) {       ct[x] = c + bt[x];
            c = ct[x];
        }
    }
    for(i=0; i<n; i++) {
        tat[i] = ct[i] - at[i];
        wt[i] = tat[i] - bt[i];
    }
}
```

Output

PID	AT	BT	CT	TAT	WT
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

Lab-2

P-7 Round Robin scheduling

```
#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(-
      nn->pid = p
      nn->next = NULL;
      return nn;
}
void enqueue(int p) {
      struct queue *nn = create(p);
      if (rq == NULL)
            rq = nn;
      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;
```

```
    else {
        struct queue *temp = rq;
        x = temp->pid;
        rq = rq->next;
        free(temp);
    }
    return x;
}

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

int main() {
    int n, t, x=1;
    printf("enter no. processes");
    scanf("%d", &n);
    printf("enter time quantum");
    scanf("%d", &t);
    int pid[n], at[n], bt1[n], ct[n],

    sort(pid, at, bt1, n);
    enqueue(pid[0]);
    for(int i=0; i<n; i++) {
        bt2[i] = bt1[i];
        rt[i] = -1;
    }
```

```
int count=0
int ctval = at[0];
while(count!= n){
    int cmp = rq->pid;
    int curi = 0;
    for(int i=0; i<n; i++){
        if(pid[i] == cmp){
            curi = i;
            break;
        }
    }
    if(rt[curi] == -1){
        rt[curi]= ctval- at[curi];
    }
    if(bt2[curi]<=t){
        ctval += bt2[curi];
        bt[curi]=0;
    }
    else{
        ctval +=t;
        bt2[curi]-=t;
    }
}

tat[n]; printf("pid\tat\tbt\tct\ttat\twt\trt");

for(int i=0; i<n; i++){
    printf("%d\t%d\t%d\t%d\t%d\t%d\t%d",
    printf("\n");
}

}
```

Output

pid	at	bt	ct	tat	wt	rt
1	0	5	13	13	8	0
2	1	3	12	11	8	1
3	2	1	5	3	2	2
4	3	2	9	6	4	4
5	4	3	14	10	7	5

Avg tat : 8.6
Avg wt : 5.8

Priority scheduling

```
#include <stdio.h>
int highest_priority (int *prior, int s, inte
    int x = prior[s];
    int j = s;
    for(int i = s; i<e; i++){
        if(prior[i] > x){
            x = prior[i];
            j = i;
        }
    }
    return j;
}

int main(){
    int n, t, x;
    printf("enter no. of processes:");
    scanf(" %d", &n);
    int pid[n], at[n], bt1[n], ct[n
    for(int i=0; i<n; i++){
        printf("enter arrival time :");
        scanf("%d %d %d", &at[i], &bt[
        pid[i] = i+1;
    }
    sort(pid, at, bt1, prior, n);
    for(int i=0, i<n; i++){
        bt2[i] = bt1[i];
        at[i] = -1;
    }.
```

```
while(count != n){
    if(rt[curi] == -1){
        rt[curi] = ctval - at[curi];
    }
    if(curc == n){
        ctval += bt2[curi];
        bt2[curi] = 0;
    }
    else{
        ctval += 1;
        bt2[curi] = 1;
    }
    for(int i=0; at[i]<=ctval; i++){
        curc += 1;
        x = i;
    }
    if bt2[curi] == 0){
        count += 1;
        ct[curi] = ctval;
        prior[curi] = -1;
    }
    curi = highest_priority(prior, 0, x+1);
}
```

output

pid	priority	AT	BT	CT	TAT	WT	RT
1	10	0	5	12	12	7	
2	20	1	4	8	7	3	
3	30	2	2	4	6	4	
4	40	4	1	5	1	0	

Lab-3

Date
Page

(3) Multilevel Queue scheduling.

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

Struct Process {
    int id;
    int bt;
    int at;
    int qt;
    int wt;
    int tat;
    int rt;
    int st;
    int et;
    int v;
};

void main() {
    int np, ct=0
    float awt=0, atat=0, aut=0, th;

    printf("Queue1 sys process 2 user pr");
    printf("enter number of processes:");
    scanf("%d", &np);

    Struct process pr[np];
    Struct process temp;
```

```
for(int i=0; i<np; i++){
    pr[i].id = i+1;
    pr[i].v = 0;
    printf("enter, bt, at, q no of p:%d:", i+1);
    scanf("%d%d%d", &pr[i].bt, &pr[i].at,
}
// sorting.

for(int i=0; i<np; i++){
    for(int j = i+1; j<np; j++){
        if(pr[i].at > pr[j].at){
            temp = pr[i];
            pr[i] = pr[j];
            pr[j] = temp;
        }
    }
}
//scheduling logic
int ct = pr[0].bt;
int mqt = INT_MAX, mqi, pc = 0;
pr[0].v = 1;

while(pc < np-1){
    for(int i=0; i<np; i++){
        if(pr[i].at <= ct && pr[i].v == 0){
            if(pr[i].qt < mqt){
                mqt = pr[i].qt
                mqi = i
            }
        }
    }
}
```

```
pr[mqi].v=1
ct += pr[mqi].bt;
mqt = INT_MAX;
temp = pr[pc+1];
pr[pc+1] = pr[mqi];
pr[mqi] = temp;

pc++;
}.
```

&pr[i].qt

output.

process	wt	tat	Rt
1	0	4	0
2	4	7	4
3	7	15	7
4.	5	10.	5

averge wait time = 4.
Avg turn around : 9
avg response t = 4.

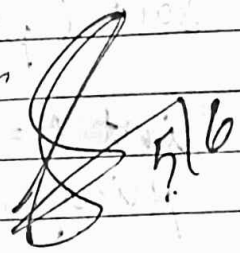
Rate monotonic

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

void sort (int pt[], int b[], int pt, int n){
    int temp=0;
    for (int i=0; i<n; i++){
        for(int j=i; j<n; j++){
            if (pt[j] < pt[i]) {
                temp = pt[i];
                pt[i]= pt[j];
                p[j] = temp;
            }
    } }
}

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

```
void main() {
    int n;
    printf("enter no. of p: ");
    scanf("%d", &n);
    int pr[n], b[n], pt[n], rem[n];
    for(int i=0; i<n; i++) {
        scanf("%d", &b[i]);
        rem[i] = b[i];
    }

    sort(pr, b, pt, n);

    int l = lcmul(pt, n)
    printf("LCM = %d\n", l);

    while(time < l) {
        int f = 0;
        for(int i=0; i<n; i++) {
            if(time % pt[i] == 0)
                rem[i] = b[i];
            if(rem[i] > 0) {
                if(prev != proc[i]) {
                    printf("%d onwards:");
                }
                rem[i]--;
                f = 1;
                break;
                x = 0;
            }
        }
        if(!f) {
            if(x != 1) {
                x = 1;
            }
        }
    }
}
```

Output

0ms onwards : P2
2ms onwards : P3
4ms onwards : P1
5ms onwards : P2
7 ms onwards : P1
9 ms onwards : idle
10ms onwards : P2
12 ms onwards : P3
14 ms onwards : idle
15 ms onwards : P2



P-7

Date
Page

(2) Earliest deadline.

```
void sort (int pr[], int d[], int b[])
    int temp=0;
    for (int i=0; i<n; i++){
        for (int j=i; j<n; j++){
            if (d[j] < d[i]){
                temp = d[j];
                d[j] = d[i];
                d[i] = temp;
            }
        }
    }

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

    return lcm;
}

void main(){
    int n;
    printf(" number of processes : ");
    scanf("%d", &n);
    int pr[n], b[n], pt[n], d[n]
    for (int i=0; i<n; i++){
        scanf("%d", &b[i]);
        rem[i] = b[i];
    }
```

```
sort(pr, d, b, pt, n);

int l = lcmul(pt, n);
printf("\n earliest deadline:");

int time=0, prev=0, x=0;
int nextdeadlines[n];
for(int i=0; i<n; i++){
    nextdeadlines[i] = d[i];
    rem[i] = b[i]
}
while(time<l){
    for(int i=0; i<n; i++){
        if(time%pt[i]==0 && time!=0)
            nextdeadlines[i] = time+d[i]
            rem[i] = b[i];
    }
}
}
```

output

0 ms : 2
1 ms : 1
2 ms : 1
3 ms : 1
4 ms : 3
5 ms : 2

Lab-4

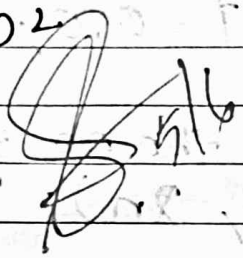
(3) proportional scheduling.

```
#include <stdio.h>

typedef struct {
    char name[5];
    int T;
} task;

int main() {
    int n;
    float total_T = 0.0;

    printf("enter the no. of processes");
    scanf("%d", &n);

    Task p[n];

    for(int i=0; i<n; i++){
        printf("process %d: " i+1);
        sprintf(p[i].name, "p%d");
        printf("Tickets: ");
        scanf("%d", &p[i].T);
        total_T += p[i].T;
    }


}
```

output.

PID1 : 10
PID2 : 20

probability is 33%.
probability is 66%.
1ms : p2
2ms : p1
3ms : p2
4ms : p1
5ms : p2



12/6/24

Lab - 5

```
#include<stdio.h>
#define MAX 10

int sharedMemory[MAX];
int mutex = 1;
int empty = MAX - 1;
int full = 0;

void wait(int *s) {
    s--;
}

void signal(int *s) {
    s++;
}

void producer(int A[], int *m, int *e, int
    if(*e != 0) {
        wait(m);
        A[j] = 1;
        wait(e);
        signal(j);
        signal(m);
    }
}

void consumer(int A[], int *m, int *e, int
    if(*f != 0) {
        wait(m);
        A[j] = 0;
        signal(e);
        signal(m);
    }
}
```

```
void main() {
    int optn;
    printf("buffer size:150");
    while(1) {
        printf("enter optn: ");
        scanf("1.produce \n 2consume");
        if(optn == 1) {
            producer();
        }
        elof(optn == 2) {
            consumer();
        }
        else {
            break;
        }
    }
}
```

output

produced
produced
consumed
produced
produced
produced
produced
produced
produced
produced
produced
produced
produced

```
. size:\n");

nter optn: ");
produce\n2consume");
1) {
ea();

== 2) {
me();
```

```
buffer full!
consumed
consumed
consumed
consumed
consumed
consumed
consumed
consumed
Consumed
consumed
buffer empty!
```

19/6

9/6/24

Lab - 6

→ Banker Algorithm

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

#define NUM_PROCESSES 5
#define NUM_RESOURCES 3

int available [NUM_RESOURCES];
int maximum [NUM_PROCESSES][NUM_RES
int allocation [NUM_PROCESSES][NUM_RES
int need [NUM_PROCESSES][NUM_RESOURCE
int safe_sequence [NUM_PROCESSES];
int scount = 0;

void calculateNeed() {
    for(int i=0; i<NUM_PROCESSES; i++){
        for(int j=0; j<NUM_RESOURCES; j++){
            need[i][j] = maximum[i][j] - allocat
        }
    }
}

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

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

    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_RESO; j
            if (need[i][j]>work[j]
                canProceed=false;
                break;
            }
        }
        if(canProceed) {
            for(int j=0; j<NUM_R; j++)
                work[j] += allocation[i][j]
            finish[i]=true;
            safe_sequence[count]=i;
            count += 1;
            found=true;
        }
    }
}
if(!found){
    block;
}
}
for(int i=0; i<NUM_PROCESSES; i++){
    if(!finish[i]) {
        return false;
    }
}
return true;
}
```

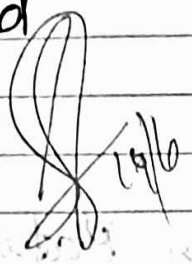
```
bool requestResource(int process, int request[]
    for(int i=0; i<NUM_RESOURCE; i++){
        if(request[i] > need[process][i]){
            printf("Error"); return false;
        }
        if(request[i] > available[i]){
            printf("Error");
            return false;
        }
    }
    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");
        return true;
    }
    else{
        for(int i=0; i<NUM_RESOURCE; i++
            available[i] += request[i];
            allocation[processes][i] -= request[i
            need[process][i] += request[i];
        }
        printf("Req not granted");
        return false;
    }
}
```

Output:

```
Enter available resource vector:
3 3 2
Enter the maximum Matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter allocation Matrix:
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 (5, 3, 2)
process 3 is visited (7, 4, 3)
process 4 is visited (7, 5, 5)
process 0 is visited (10, 5, 7)

Safe sequence
1  3  4  0  2

Enter the process number for request : 1
Request vector : 1 0 2
```

Process 1 is visited (5, 3, 2)
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



3/7/24

Lab - 7

Date
Page

deadlock detection

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

#define NUM_PROCESSES 5
#define NUM_RESOURCE 3

int available[NUM_RESOURCES];
int allocation[NUM_RESOURCES];
int request[NUM_RESOURCES];
int avail_matrix[NUM_RESOURCES];

bool deadlockDetection(int *safeSequence)
    int work[NUM_RESOURCES];
    bool finish[NUM_PROCESSES];

    for(int i=0; i<NUM_PROCESSES; i++){
        work[i] = available[i];
        avail_matrix[0][i] = work[i];
    }
    int count=0;
    while(count< NUM_PROCESSES){
        bool found = false;
        for(int i=0; i<NUM_PROCESS; i++)
            if(req[i][j] > work[j]){
                canProceed = true;
                break;
            }
        }
        if(can proceed){
            safe sequence[count++] = i;
            finish[i] = true;
            found = true;
        }
    }
```

```
        if(found){
            break;
        }
    }

int main(){
    int i, j;

    printf("enter available");
    for(i=0; i<NUM_RESOURCES; i++){
        scanf("%d", &available[i]);
    }
    printf("Available resources");
    for(i=0; i<NUM; i++){
        scanf("%d", &resource[i][i]);
    }

    int safesequence[NUM_PROCESS];

    printf("Request Matrix");
    for(i=0; i<NUM_PROCESSES; i++){
        for(j=0; j<NUM_PROCESSES; j++){
            scanf("%d", &request[i][j]);
        }
    }

    if(deadlockDetection(safesequence)
        printf("Available Matrix");
        for(i=0; i<=NUM_PROCESSES; i++){
            for(j=0; j<=NUM_PROCESSES; j++){
                printf("%d", avail_matrix[i][j]);
            }
        }
    }
```

}

Output.

enter available resource vector:

0 0 0

Enter allocation matrix

0 1 0

2 0 0

3 0 3

2 1 1

0 0 2

Enter request matrix

0 0 0

2 0 2

0 0 0

1 0 0

0 0 2

No dead lock detected. The system is in safe state.

Safe sequence: P0 P2 P3 P4 P1

Memory Management

```
#include<stdio.h>
#define MAX 25

void firstfit(int nb, int nf, int b[], int
    int flag[MAX], bf[MAX]={0}, ff[MA
    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;
                    flag[i]=temp;
                    bf[j]=1;
                    break;
                }
            }
        }
    }

    printf("Memory management: ff");
    printf("File no:\t File Size:\t ff");
    for(i=1; i<=nf; i++){
        printf("%d\t\t%d", i, f[i]);
        if(ff[i]!=0){
            printf("%d\t\t%d", ff[i], flag[i]);
        }
        else
            Printf("not allocated");
    }
}
```

```
void bestfit(int nb, int nj, int b[], int j[]){
    int flag[MAX], bf[MAX]={0};
    int i, j, temp;
    for(i=1; i<=nj; i++){
        for(j=1; j<=nb; j++){
            if(bf[j]!=1){
                temp = b[j]-j[i];
                if(temp>=0 && lowest>temp){
                    flag[i]=j;
                    lowest=temp;
                }
            }
        }
        flag[i]=lowest;
        bf[flag[i]]=1;
        lowest=10000;
    }
}

void worstfit(int nb, int nj, int b[], int j[]){
    int flag[MAX], bf[MAX]={0};
    int i, j, temp;
    for(i=1; i<=nj; i++){
        for(j=1; j<=nb; j++){
            if(bf[j]!=1){
                temp=b[j]-j[i];
                if(temp>=0 && highest<temp){
                    flag[i]=j;
                    highest=temp;
                }
            }
        }
    }
}
```

```
        frag[i] = highest;
        bf[p][i] = 1;
        highest = 0;
    }
}

int main() {
    int b[max], f[max], nb, nf;

    printf("enter no. of block");
    scanf("%d", &nb);
    printf("enter the size of the blocks");
    scanf("%d", &nf);

    for(int i=1; i<=nb; i++) {
        printf("Block %d:", i);
        scanf("%d", &b[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];
    }
    firstfit(nb, nf, b1, f);
    bestfit(nb, nf, b2, f);
    worstfit(nb, nf, b3, f);

}
```

Output

enter the no. of block : 3
enter the no. of files : 2

enter the size of the blocks :
b1 : 5
b2 : 2
b3 : 7

enter the size of files :
f1 : 1
f2 : 4

Memory management scheme - First fit

F no.	F size	block no.	b size
1	1	1	5
2	4	3	7

Worst fit

F no	f size	block no	block size
1	1	3	7
2	4	1	5

Best fit

F no	file size	block no.	block size
1	1	2	2
2	4	1	5

10/7/24

Lab - 8

Page Replacement Algorithms

```
#include<stdio.h>

int isPagePresent(int frame[], int n, int page) {
    for (int i=0; i<n; i++) {
        if (frame[i] == page) {
            return 1;
        }
    }
    return 0;
}

void printFrames(int frames[], int n){
    for(int i=0; i<n; i++) {
        if (frames != -1) {
            printf("%d", frame[i]);
        }
        else{
            printf("-");
        }
    }
    printf("\n");
}

void fifo(int pages[], int numP) {
    int frames[minionFrames];
    int front=0, pageFaults=0;

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

```
printf("FIFO");
for(int i=0; i<numPages; i++){
    printf("%d\t\t", pages[i]);
    if(!isPagePresent(frames, num)){
        frames[front] = pages[i];
        front = (front+1) % numFram
        pageFaults++;
    }
    printFrames(frames, numFram
}
}

int optimal(int pages[], int numpages, int i){
    int farthest = current;
    int index = -1;
    for(int i=0; i<numframes; i++){
        int j;
        for(int j = currentIndex; j<numpgs
            if(frame[i] == pages[j]){
                farthest = j;
                index = i;
            }
            break;
        }
        if(j == numpages){
            return i;
        }
    }
    return (index == -1)? 0 : index;
}
```

```
void lru(int pages[], int numPages, int num
    int frames[numFrames];
    int pageFaults = 0;
    int timestamp[numframe];

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

    printf("LRU Replacement");
    for(int i=0; i<numFrames; i++){
        printf("%d\t\t", pages[i]);
        if(!isPagePresent(frames, num){
            int lruIndex = 0;
            for(int j=1; j<numFrames;j++)
                if(timestamp<timestamp[l
                    lruIndex = j;
                }
                frames[lruIndex] = page[i];
                timestamps[lruIndex] =
                pageFaults++;
            }
        }
        printf(frames[frames, numFrames);
    }

int main(){
    int numFrames, numPages;
    printf("Enter the number: ");
    scanf("%d", &numFrames);
```

numFrame){ printf("Enter the number pages:");
scanf("%d", &numPages);

int pages[numPages];

fifo (pages, numPages, numFra
optimal(pages, numPages, numFra
lruPageReplacement(pages, no.pages);

return 0;
}

output

enter number of frames : 3
){ enter no. of pages : 20
lru]{ enter reference string : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

fifo.

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	7	7	7		
-	0	0	0	3	3	3	2	2	2	2	1	1	1	1	0	0			
-	-	1	1	1	0	0	0	3	3	3	3	2	2	2	1	1			

pagefaults = 15

optimal

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	7			
7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	7			
-	0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0			
-	-	1	1	1	3	3	3	3	3	3	3	3	1	1	1	1			

pagefaults = 9

LRU

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7 7 7 2 2 2 2 4 4 4 0 0 0 1 1 1 1 1 1 1

- 0 0 0 0 0 0 0 0 3 3 3 3 3 0 0 0 0 0

- - 1 1 1 3 3 3 2 2 2 2 2 2 2 2 7 7 7

Page faults = 12.

10/7

Completed.