INDEX

Name MONISH. P Std IVSEM Sec C

Roll No. 168 Subject O.S (Obv) School/College BMSCE (C.S.E)

School/College Tel. No. Parents Tel. No.

8/5/24 LAB 1 DATE: PAGE:

① First come first serve:

```
#include <stdio.h>
int n, i, j;
void main() {
    printf("enter the number of processes: ");
    scanf("%d", &n);
    int arrivalTime[n];
    int burstTime[n];
    int completionTime[n];
    int tat[n];
    int waitingTime[n];
    for(i=0; i<n; i++){
        printf("Enter arrival time of P[%d]: ", i+1);
        scanf("%d", &arrivalTime[i]);
    }
    for(i=0; i<n; i++){
        printf("Enter the burst time of P[%d]: ", i+1);
        scanf("%d", &burstTime[i]);
    }
    completionTime[0] = arrivalTime[0] + burstTime[0];
    for(i=1; i<n; i++){
        if(arrivalTime[i] <= completionTime[i]){
            completionTime[i] = completionTime[i-1] +
                                burstTime[i];
        }
        else{
            completionTime[i] = arrivalTime[i] + burstTime[i];
        }
    for(i=0; i<n; i++){
        tat[i] = completionTime[i] - arrivalTime[i];
        waitingTime[i] = tat[i] - burstTime[i];
    }
```

DATE: PAGE:

```
for (i=0;i<n;i++){
printf("P[%d]\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
}
}
```

O/P:

Taking i/ps

PID	A.T	B.T	C.T	TAT	W.T
P[0]	0	2	2	2	0
P[2]	1	2	4	3	1
P[3]	5	3	8	3	0
P[4]	6	4	12	6	2

DATE: PAGE:

8/5/24 (2) S.J.F:

```
#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, int s, int n){
    for (int i=s; i<n; i++){
        for (int j=s; 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;
    printf ("Enter the number of processes:");
    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", &at[i], &bt[i]);
        pid[i] = i+1;
    }
    sort (pid, at, bt, 0, n);
    int c = at[0] + bt[0];
    ct[0] = c;
```

DATE: PAGE:

```
for (int i=1;i<n;i++){
    int t=i;
    int x=i;
    while(at[i]<c){
        t++;
        i++;
    }
    sortb(pid, at, bt, x, t);
    if(at[x]>c)
        c=at[x];
    for(;x<t;x++){
        at[x]=c+bt[x];
        c=ct[x];
    }
}
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];
}
for(int i=0;i<n;i++){
printf("%d\t%d\t%d\t%d\t%d\t%d\t%d",
printf("\n");
}
printf("\nAverage turn around time: %f", avg_tat/n);
printf("\nAverage waiting time: %f", avg_wt/n);
}
```

DATE: PAGE:

o/p:

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

DATE:

PAGE:

15/5/24

③ Round Robin:

```
#include<stdio.h>
#include<stdlib.h>
struct q{
    int pid;
    struct q* next;
};
struct q* rq = NULL;
struct q* create(int p){
    struct q *nn = (*q) malloc(sizeof(q));
    nn->pid=p;
    nn->next = NULL;
    return nn;
}

void enqueue(int p){
    struct q* nn = create(p);
    if (rq == NULL){ rq = nn; }
    else {
        struct q* temp = rq;
        while (temp->next != NULL){
            temp = temp->next;
        }
        temp->next = nn;
    }
}
int dequeue(){
    int x = 0;
    if (rq == NULL) { return x; }
    else {
        struct q* temp = rq;
        x = temp->pid;
```

```
    rq = rq->next;
    free(temp);
  }
  return n;
}

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

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

void sort(int *pid, int *at, int *bt, 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(&ct[i], &ct[j]);
      }
    }
  }
}
```

DATE:

PAGE:

```
int main(){
    int n, t, x = 1;
    printf("Enter the number of processes");
    scanf("%d", &n);
    printf("Enter the time quantum");
    scanf("%d", &t);
    int pid[n], at[n], bt[n], ct[n], tat[n], wt[n], bt2[n], rt[n];
    for(int i=0; i<n; i++){
        printf("Enter the arrival time & burst time");
        scanf("%d %d", &at[i], &bt[i]);
        pid[i] = i+1;
    }
    sort(pid, at, bt, n);
    enqueue(pid[0]);
    for(int i=0; i<n; i++){
        bt2[i] = bt[i];
        rt[i] = -1;
    }
    int count = 0;
    int ctime = at[0];
    while(count != n){
        int curp = rq->pid;
        int curi = 0;
        for(int i=0; i<n; i++){
            if(pid[i] == curp){
                curi = i;
                break;
            }
        }
        if(rt[curi] == -1){
            rt[curi] = ctime - at[curi];
        }
        if(bt2[curi] <= t){
```

```
        ctvar += bt2[cursi];
        bt2[cursi] = 0;
    } else { ctvar = t;
             bt2[cursi] -= t;
    }
    while (at[x] <= ctvar && x <= n) {
        enqueue(pid[x]); x++;
    }
    if (bt2[cursi] > 0) { enqueue(pid[cursi]); }
    if (bt2[cursi] == 0){
        count++;
        ct[cursi] = ctvar;
    }
    dequeue();
    }
    for (int i=0; i<n; i++){
        tat[i] = ct[i] - at[i];
        wt[i] = tat[i] - bt1[i];
    }
    float avg_tat = 0;
    float avg_wt = 0;
    for (int i = 0; i<n; i++){
        tat[i] = ct[i] - at[i];   avg_tat += tat[i];
        wt[i] = tat[i] - bt1[i];  avg_wt += wt[i];
    }
}
```

DATE:

PAGE:

o/p:

Enter the priorities

10 20 30 40

Enter arrival times

0 1 2 4

Enter burst times

3 4 2 1

PID	Prio	AT	BT	CT	TAT	WT	RT
P1	10	0	3	5	5	0	0
P2	20	1	4	12	11	7	7
P3	30	2	2	8	6	4	4
P4	40	4	1	6	2	1	1

Avg TAT: 6.0 ms

Avg WT: 3.0 ms.

25/5/24 (8) Priority Scheduling:

```
#include <stdio.h>
#include <stdlib.h>
void swap(int *a, int *b){
    *a = *a + *b;
    *b = *a - *b;
    *a = *a - *b;
}
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]);
            }
        }
    }
}
int highest_priority(int *prior, int s, int e){
    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;
}
```

DATE:

PAGE:

```
int main(){
  int n, ti;
  printf("\n Enter the no of processes:");
  scanf("%d", &n);
  int pid[n], at[n], bt[n], ct[n], tat[n], wt[n], bt2[n], pri
  pries[n];
  for(int i=0; i<n; i++){
    printf("Enter assival time & burst time");
    scanf("%d %d %d %d", &at[i], &bt[i] & pries[i]);
    pid[i] = i+1;
  }
  Sort(pid, at, bt, pries, n);
  for(int i=0; i<n; i++){
    bt2[i] = bt[i];
    st[i] = -1;
  }
  int conve=0; count=0; ctvar=at[0], cusi=0;
  while(count != n){
    if(rt[cusi] == -1)
      rt[cusi] = ctvar-at[cusi];
    if(conve==n){
      ctvar++ = bt2[cusi];
      bt2[cusi] = 0;
    }
    else{
      ctvar++;
      bt2[cusi]==1;
    }
  for(int i=0; at[i]<=ctvar; i++){
    conve++;
    n=i;
  }
```

DATE: PAGE:

```
if (ct2[curi] == 0){
    count += 1;
    ct[curi] = ctime;
    prior[curi] = -1;
}
curi = highest_priority(prior, 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];
}
}
```

O/P:

Enter the number of processes: 4.

Enter priorities:

10 20 30 40

Enter arrival times:

0 1 2 4

Enter burst time:

5 4 2 1

PROID	Prior	AT	BT	CT	TAT	WT	RT
P1	10	0	5	12	12	7	0
P2	20	1	4	8	7	3	0
P3	30	2	2	4	2	0	0
P4	40	4	1	5	1	0	0

Avg TAT: 5.5 ms

Avg WT: 2.5 ms.

LAB-3 DATE:

5/6/2024 PAGE:

(5) Rate Monotonic Scheduling:

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
void sort(int proc[],int b[],int pt[],int n){
    int temp=0;
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            if(pt[j]<pt[i]){
                temp=pt[i];
                pt[i]=pt[j];
                pt[j]=temp;
                temp=b[j];
                b[j]=b[i];
                b[i]=temp;
                temp=proc[i];
                proc[i]=proc[j];
                proc[j]=temp;
            }
        }
    }
}
int gcd(int a,int b){
    int x;
    while(b>0){
        x=a%b;
        a=b;
        b=x;
    }
    return a;
}
```

DATE: PAGE:

```
int lcmul (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 no of processes");
scanf ("%d", &n);
int proc[n], b[n], pt[n], rem[n];
printf ("Enter the CPU burst time:\n");
for (int i=0; i<n; i++){
  scanf ("%d", &b[i]);
  rem[i] = b[i];
}
printf ("Enter the time period:\n");
for (int i=0; i<n; i++) { scanf ("%d", &pt[i]);
for (int j=0; j<n; j++) { proc[i] = i+1;
sort (proc, b, pt, n);
int l = lcmul (pt, n);
printf ("lcm = %d\n", l);
printf ("\n Rate monotonic scheduling");
printf ("\n PID \t Bt \t Period \n");
for (int i=0; i<n; i++){
    printf ("%d \t %d \t %d \t %d \n", proc[i], b[i], pt[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 %lf <= %lf => %s \n", sum, rhs, (sum<=rhs)? "True":
"False");
```

DATE:

PAGE:

```
if (sum>pbs) exit(0);
printf("scheduling occurs for %d ms\n\n",i);
int time=0, prev=0, x=0;
while(time<i){
 int f=0;
 for(int i=0; i<n; i++){
  if (time==at[i])
   num[i]=bs[i];
  if (num[i]>0){
   if (prev!=proc[i]){
   printf("%d ms onwards: process %d running\n",time,proc[i]);
   prev=proc[i];
  }
  num[i]--;
  f=1;
  break;
  x=0;
 }
 if(!f){
  if(x!=1){
   printf("%d ms onwards: CPU is idle\n",time);
   x=1;
  }
 }
 time++;
}
```

DATE: PAGE:

o/p:

Enter the number of processes: 2

Enter the CPU burst times:

20 35

Enter the time periods:

50 100

Rate Monotonic Scheduling

PID	BT	Period.
1	20	50
2	35	100.

0.75000 <= 0.82487 ⇒ true.

Scheduling occurs for 100 ms.

0 ms onwards: Process 1 running

20 ms onwards: Process 2 running

50 ms onwards: Process 1 running

70 ms onwards: Process 2 running

75 ms onwards: CPU is idle.

DATE: PAGE:

5/6/2024

⑥ 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=0;j<n;j++){
        if (d[j]<d[i]){
            temp=d[j];
            d[j]=d[i];
            d[i]=temp;
            temp=pt[i];
            pt[i]=pt[j];
            pt[j]=temp;
            temp=b[j];
            b[j]=b[i];
            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;
}
```

DATE: PAGE:

```
int lcmul (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("\n Enter number of processes");
    scanf("%d", &n);
    int proc[n], b[n], pt[n], d[n], rem[n];
    printf("Enter the deadlines: \n");
    for(int i=0; i<n; i++) { scanf("%d", &d[i]);
    printf("Enter the time periods\n");
    for(int i=0; i<n; i++) { scanf("%d", &pt[i]);
    for(int i=0; i<n; i++){ proc[i]= i+1;}
    sort(proc, d, b, pt, n);
    int l = lcmul(pt, n);
    printf("Earliest deadline Scheduling\n");
    printf("PID \t Burst \t deadline \t period \n");
    for(int i=0; i<n; i++) {proc[i]=i+1;}
    printf("%d \t %d \t %d \t %d \n", proc[i], b[i], d[i], pt[i]);
    printf("Scheduling occurs for %d ms \n\n", l);
    int time=0; prev=0; x=0;
    int ndl[n];
    for(int i=0; i<n; i++){
        ndl[i]= d[i]; rem[i] = b[i];
    }
    while (time<l) {
        for(int i=0; i<n; i++){
            if(time % pt[i]==0 && time!=0){
                ndl[i] = time+ d[i];
                rem[i] = b[i];
            }
        }
```

DATE:

PAGE:

```
int mdl = 1000;
int ttE = -1;
for (int i=0; i<n; i++) {
    if (rem[i] > 0 && rd[i] < mdl) {
        mdl = rdl[i];
        ttE = i;
    }
}
if (ttE != -1) {
    printf("%d ms: Task %d is running\n", time, proc[ttE]);
    rem[ttE] = rem[ttE] - 1;
}
else {
    printf("%d ms: CPU is idle\n", time);
}
time++;
}
}
```

o/p:

Enter the no of process: 3

Enter the CPU burst time:

3 2

Enter the deadline

Enter the time periods:

20 5 10

Earliest deadline scheduling:

PID	BT	Deadline	Period
2	2	4	5
1	3	7	20
3	2	8	10

DATE: PAGE:

Scheduling order for above:

0 ms : 2
1 ms : 1
2 ms : 1
3 ms : 1
4 ms : 3
5 ms : 3
6 ms : 2
7 ms : 2
8 ms : CPU is idle
9 ms : 2
10 ms : 2
11 ms : 3
12 ms : 3
13 ms : CPU is idle
14 ms : 2
15 ms : 2
16 ms : CPU is idle
17 ms : 2
18 ms : CPU is idle
19 ms : ——
20 ms : ——

8/16

DATE:

PAGE:

5/6/2024.

(8) PROPORTION SCHEDULING.

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int main(){
    int n, SOT=0;
    printf("\n Enter the total 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("\n PID %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("Probability of surviving process %d is %d \n", i+1, pid[i]);
    }
    srand(time(NULL));
    while(sum>0){
        int x = rand()%SOT;
        int j;
        for(j=0; j<n; j++){
            if(x<l[j+1]){
                printf("%d ms: Surviving ticket of process %d\n", t, j+1);
                pid[j]--;
                sum--;
```

```
        t++;
        break;
      }
    }
  }
  for (int i=0; i<n; i++) {
      if (pid[i+1] == 0) {
          printf("%d finished executing");
      }
  }
}
```

o/p:

Enter number of processes: 3

Enter the number of tickets for the process

10

20

30

Probabality of servicing process 1 : 16 %

Probabality of servicing process 2 : 33 %

Probabality of servicing process 3 : 50 %

Execution

1 ms servicing ticket of process : 1

2 ms Servicing ticket of process : 2

3 ms servicing ticket of process : 3

.

.

.

.

.

.

Process 1 fully completed

Process 2 fully completed

Process 3 fully completed.

12/6/24 LAB-04 DATE: PAGE:

(9) Write a C program to simulate producer-consumer problem using semaphores

```
#include<stdio.h>
#define MAX 7
int full = 0;
int empty = 7;
int mutex = 1;
int array[7];
int index = 0;

void initit(){
    for(int i=0;i<7;i++){
        array[i] = -1;
    }
}

int wait(int x){
    x = x-1;
    return x;
}
int Signal(int x){
    x = x+1;
    return x;
}
void producer(){
  if(mutex==0){
      printf("\n Other process has the lock");
  } else {
      if(empty!=0){
          mutex = wait(mutex);
          empty = wait(empty);
```

```
        printf("\n Enter the value to insert");
        scanf("%d", &array[index]);
        index++;
        full = signal(full);
        printf("Inserted successfully");
        mutex = signal(mutex);
    } else {
        printf("\n The buffer is full");
    }
  }
}

void consume() {
    if(mutex==0){
        printf("\n Other process has the lock");
    }
    else {
      if (full==0) {
          printf("The buffer is empty");
        }
      else {
          mutex = wait(mutex);
          full = wait(full);
          printf("\n%d consumed from buffer", array[index]);
          index = index - 1;
          empty = signal(empty);
          printf("Consumed successfull");
          mutex = signal(mutex);
      }
    }
}
```

DATE:

PAGE:

o/p:

produced

produced

Consumed

produced

produced

produced

produced

produced

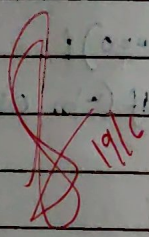
produced

produced

produced

produced

buffer is full!!



DATE: PAGE:

12/6/24

LAB-05

Banker's Algorithm:

```
#include <stdio.h>
#include <stdbool.h>
# define NUM_PROCESS 5
# define NUM_RESOURCES 3.

int available[NUM_RESOURCES];
int maximum[NUM_PROCESSES][NUM_RESOURCES];
int allocation[NUM_PROCESSES][NUM_RESOURCES];
int need [NUM_PROCESS][NUM_PROCESS];
int safeSequence [NUM_PROCESSES];
int count = 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] - 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];
    }
    while (true){
        bool found = false;
        for(int i=0; i< NUM_RESOURCES ; i++){
            if (!finish[i]){
                bool canProcud = true;
```

DATE: PAGE:

```
for (int j = 0; j < NUM_RESOURCES; j++) {
    work[j] += allocation[i][j];
}
printf("Process %d is visited (%d, %d, %d)", );
finish[i] = true;
Safe_Sequence[Scount] = i;
Scount++;
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][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 granted");
    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.");
    return false;
}
```

o/p:

```
Enter available
3 3 2
Enter the maximum
7 5 3
3 2 2
```

DATE: PAGE:

9 0 2
2 2 2
4 3 3

Enter the Allocation

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 (7,4,5)
Process 0 is visited (7,5,5)
Process 2 is visited (10,5,7)

19/6

The System is in a Safe State

Safe Sequence

1 3 4 0 2

Enter the process number for request (0 to 4): 1
Enter 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.

DATE: PAGE:

12/6/24

LAB - 06.

(9) Deadlock detection:

```
#include<stdio.h>
#include<stdbool.h>
#define N_P 5.
#define N_R 3.
int available[N_R];
int allocation[N_P][N_R];
int request
```

```
int main{ int n,m,i,j,k;
printf("Enter the no of process");
scanf("%d", &n);
printf("Enter the no of resource");
scanf("%d", &m);
int alloc[n][m], request[n][m], avail[m]
for (int i=0; i<n; i++){
printf("Enter details for P:%d \n", i);
printf("Enter allocation -:");
for (int j=0; j<m; j++) { scanf("%d", &avail[i]);
int finish[n], safe[n], work[m], flag, f=0;
for (i=0; i<n; i++) { finish[i]=0; }
for (int j=0; j<m; j++) { work[j]=avail[j]; }
int count=0;
while (count<n){
flag=0; f=0;
for (i=0; i<n; i++){
if (finish[i]==0){
for (int j=0; j<m; j++) for
if (alloc[i][j] != 0) { f=1; }
if (f){
int canproceed=1;
for (j=0; j<m; j++){
if (request[i][j] > work[j]){
canproceed=0; break;
}
}
```

DATE: PAGE:

```
if (canBeAllocated){
    for (k=0; k<m; k++){
        work[k] += alloc[i][k];
        safeSeq[count++] = i;
        finish[i] = 1; flag = 1;
    }
}
else{
    safeSequence[count++] = i; finish[i] = 1; flag = 1;
}}
if (flag == 0){ break; }
}
if deadlock = 0;
for(i=0; i<n; i++){
    if (finish[i] == 0){ deadlock = 1;
    printf("\n System is in deadlock state");
    printf("\n The dead process are \n");
    for (j=0; j<n; j++){
        if (finish[j] == 0){ printf("%d", j); }
    }
    printf("\n"); break;
}}
}
```

o/p

Enter the no of processes : 5
Enter the no of resources : 3
Enter details for P0:
Enter allocation : 0 1 0 Enter request : 0 0 0
P1
Allocation : 2 0 0
Request : 2 0 2
P2 Allocation : 3 0 3
Request : 0 0 0
P3 Allocation : 2 1 1
Request : 1 0 0
P4 Allocation : 0 0 2
Request : 0 0 2
System is not in deadlock state !
Safe Sequence is : P0 P2 P3 P4 P1

DATE: PAGE:

3/3/24

(10) CONTINUOUS MEMORY DEALLOCATION

```
# 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=0; i<nf; i++){
    for (j=0; j<nb; j++) {
      if (b[j] != 1){
        temp = b[j] - f[i];
        if (temp >= 0){
          ff[i] = j; frag[i] = temp; bf[j] = 1; break;
  } } }
  printf ("\n Fist fit");
  printf ("\n File no \t File size \t Block no \t Block size \t fragment \n");
  for(int i=0; i<nf; i++){
    printf ("%d \t %d \t %d \t %d \n", i+1, f[i]);
    if (ff[i] != 0){
      printf ("%d %d %d %d %d", i+1, f[i], ff[i], b[ff[i]], frag[i]);
    } else {
      printf ("Not allowed");
  } } }
void bestfit (int nb, nf, b[], f[]) {
  int frag[max], bf[max] = {0}, ff[max] = {0};
  int i, j, temp, lowest;
  for (i=0; i<nf; i++){
    lowest = 10000;
    for (j=0; j<nb; j++){
      if (bf[j] != 1) {
        temp = b[j] - f[i];
        if (temp >= 0 && lowest > temp){
          ff[i] = j;
          lowest = temp;
}}}
```

DATE: PAGE:

```
if(ff[i] != 0 || (ff[i] == 0 && bf[0] == 0 && b[0] > f[i])){
    bf[ff[i]] = 1;
}}
printf("\n Best fit");
printf("\n File no: \t File size: \t Block no: \t Block size: \t Fragment");
for(i=0; i<nf; i++){
    printf("\n%d \t %d \t %d \t %d \t %d", i, f[i], ff[i], b[ff[i]], b[ff[i]]-f[i]);
else
    printf("\n Not allocated");
}
}
int main(){
    int b[max], f[max], nb, nf;
    printf("\n Enter no of blocks");
    scanf("%d", &nb);
    printf("Enter no of files"); scanf("%d", &nf);
    printf("\n Enter size of blocks");
    for(int i=0; i<nb; i++){
        printf("block %d:", i+1); scanf("%d", &b[i]);
    printf("\n Enter size of file");
    for(int i=0; i<nf; i++){
        printf("file %d", i+1);
    int b1[max], b2[max], b3[max];
    for(int i=0; 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);
    return 0;
}
```

o/p:

DATE: PAGE:

Enter the no of blocks: 3

Enter the no of files: 2

Enter the size of blocks:

1: 5

2: 2

3: 7

Enter the size of files:

1: 1

2: 4

First fit :

File no	File Size	Block no	Block size	fragment
1	1	1	5	4
2	4	3	7	3

Best fit

File no	File Size	Block no	Block size	fragment
1	1	2	2	1
2	4	1	5	1

Worst fit

File no	File size	Block no	Block size	fragment
1	1	3	7	6
2	4	1	5	1

DATE:

PAGE:

10/7/24

LAB - 04

(11) Page Replacement Algorithms

```
#include <stdio.h>
int isPageInFrame(int frame[], int n, int page) {
    for (int i = 0; i < n; i++) {
        if (frame[i] == page) {
            return 1;
        }
    }
    return 0;
}

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

void fifoPageReplacement(int pages[], int numPages, int numFrames) {
    int frame[numFrames];
    int front = 0, pageFaults = 0;
    for (int i = 0; i < numFrames; i++) {
        frame[i] = -1;
    }
    printf("FIFO Replacement\n");
    printf("Reference String\tFrames\n");
```

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

int findOptimalReplacement Index (int pages[], int numPages,
 int frames[], int numFrames, int currentIndex) {
 int farthest = currentIndex;
 int index = -1;
 for (int i=0; i< numFrames; i++) {
   int j;
   for (j= currentIndex; j< numPages; j++) {
       if (frames[i] == Pages[j]) {
           if (j > farthest) {
               farthest = j;
               index = i;
           }
           break;
       }
   }
   if (j == Numpages) {
       return i;
   }
}
return (index == -1) ? 0 : index;
```

DATE:

PAGE:

```
void lruPageReplacement(int pages[], int numPages, int numFrames) {
    int frames[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("Reference String\t Frames\n");
    for (int i = 0; i < numPages; i++) {
        printf("%d\t\t", pages[i]);

        if (!isPagepresent(frames, numFrames, pages[i])) {
            int lruIndex = 0;
            for (int j = 1; j < numFrames; j++) {
                if (timestamps[j] < timestamps[lruIndex])
                    lruIndex = j;
            }
            frames[lruIndex] = pages[i];
            timestamps[lruIndex] = i;
            pageFaults++;
        } else {
            for (int j = 0; j < numFrames; j++) {
                if (frames[j] == pages[i]) {
                    timestamps[j] = i;
                    break;
                }
            }
        }
    }
```

```
        printFrames(frames, numFrames);
    }
    printf("\n Total Page faults : %d \n\n", pageFaults);
}

int main() {
    int numFrames, numPages;
    printf("Enter number of frames");
    scanf("%d", &numFrames);
    printf("Enter number of pages");
    scanf("%d", &numPages);
    int pages[numPages];
    printf("Enter the reference string");
    for(int i = 0; i < numPages; i++) {
        scanf("%d", &pages[i]);
    }
    fifoPageReplacement(pages, numPages, numFrames);
    optPageReplacement(pages, numPages, numFrames);
    lruPageReplacement(pages, numPages, numFrames);

    return 0;
}
```

O/P:

Enter the number of frames : 3
Enter the number of pages : 20
Enter the reference string : 7 0 1 2 0 3 0 4 2 3 0
3 2 1 2 0 1 7 0 1

FIFO Replacement

7	7	-
0	7	0 -
1	7	0 1
2	2	0 1

DATE:

PAGE:

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

Total Page Faults is 15

Optimal Replacement:

Reference String	Frames
7	7 - -
0	7 0 -
1	7 0 1
2	2 0 1
0	2 0 1
3	2 0 3
0	2 0 3
4	2 4 3
2	2 4 3
3	2 4 3
0	2 0 3
3	2 0 3

DATE: PAGE:

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

Total Page fault: 9.

LRU Replacement:

Replace String.	Frame.
7	7 - -
0	7 0 -
1	7 0 1
2	2 0 1
0	2 0 1
3	2 0 3
0	2 0 3
4	4 0 3
2	4 0 2
3	4 3 2
0	0 3 2
3	0 3 2
2	0 3 2
1	1 3 2
2	1 3 2
0	1 0 2
1	1 0 2
7	1 0 7
0	1 0 7
1	1 0 7