INDEX

Name : Nakul Dhole Class : 4C

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

Sl. No.	Date	Title	Page No.	Teacher's Sign. / Remarks
5	8/5/24	First come First Serve		
5	8/5/24	SJF		
15	15/5/24	Round Robin		
5	1/6/24	Rate Monotonic		
15	5/6/24	Priority scheduling		
5	1/6/24	Earliest Deadline. first		
12	16/6/24	Proportional Code.		
12	16/6/24	Producer Consumer Problem		
19	19/6/24	Bankers Alg		
3	3/7/24	Deadlock detection		
10	10/7/24	Contiguous Memory Allocation		
10	10/7/24	Page Replacement - Algorithm.		

Completed

8/5/24

8/5/24

1) Code for FCFS.

```
# include <stdio.h>

int AT[4], BT[4], CT[4], TAT[4], WT[4],
temp = 0, a, b, current Time, x;

int main()
{
 int i;

 printf ("\n enter the Arrival times for
          programs");

 for (int i=0; i<4; i++)
 {
    scanf ("\n %d", &AT[i]);
 }

 printf ("enter the Burst times for program");

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

```
printf("\n values for completion time\n");
for (int i = 0; i < 4; 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("\n %d", CT[i]);
}
temp = 0;
```

printf("values for turn around time");

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

printf("\n The values for waiting time");
{
for(i=0; i<4; i++)
{
temp = (TAT C i] - BTC i]);
WTC i] = temp;
printf("\n%d", WTC i]);
}
temp = 0;

printf("\n The avg TAT is \n");

for (i=0; i<4; i++)
{
temp = temp + TAT C i];
a = Temp;
}

float a = temp/4;

printf("\n%d", a);
temp = 0;

```
printf ("\n the average waiting time
\n");

for (int i=0; i<4; i++)
{
    temp= temp + WT[i];
    b= temp;
}
float b = temp/4;

printf ("\n%d", b);

}
```

output

PID	AT	BT	CT	TAT	WT
P_1	0	5	5	5	0
P_2	1	3	8	7	4
P_3	2	8	16	14	6
P_4	3	6	22	18	13.

Avg TAT = 11.25 ms
Avg WT = 5.75 ms

08/5/2024 SJF

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 , 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]);
          }
        }
     }
}
void sort b (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 ( bt [i] < bt [j]) {
            swap ( & at [i], & at [j]);
            swap ( & bt [i], & bt [j];
            swap ( & pid [i] . & pid [j]) ;
           }
        }
     }
```

```
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", &at[i], &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[i] < c)
        {
           t++;
           i++;
        }
     Sort b (pid, at, bt, x, t);
     if (at[x] > c)
       c = at[x];
```

```
for (x; x<t; x++)
  {
    ct[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",
          pid[i], at[i], bt[i], ct[i], tat[i],
                                  wt[i]);
   printf("\n");
  }

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

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

Output

Enter number of processes:

Enter arrival time & Burst time: 2 1
Enter arrival time & Burst time: 1 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
P_1	10	0	5	5	0	0
P_2	20	1	12	11	7	7
P_3	30	2	8	6	4	4
P_4	40	4	6	2	1	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 NULL;
}

void enqueue (int p)
{
    struct queue * nn = create (p);
    if (rq == NULL)
    {
        rq = nn;  // empty
    }
    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;
        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");
}
```

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

  }

int main ()
{
  int n, t, x=0;
  printf (" Enter the no of process");
  scanf ("%d", &n);
  printf ("\n 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++)
  {                                   & burst ti
    printf ("\n enter the arrival 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 = q -> 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) {
    ctime += bt2[curi];
    bt2[curi]=0;
}
```

```
else { ct var += t;
    bt2[curi] = t;
}
while (at[n] <= crtvar && n<n).
{
    enqueue(pid[n]);
    n += 1;
}
if (bt2[curi]>0) { enqueue(pid[curi]);}

if (bt2[curi]==0) { count += 1;
                    ct[curi] = 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++)
    {
        avg_tat += tat[i];
        avg_wt += wt[i];
    }
printf("pid \t at \t bt \t ct \t tat \t wt
    \t \t \n\n");
for(int i=0; i<n; i++);
    {
```

```
printf("%d \t %d \t %d \t %d \t %d
        \t %d \t %d ", pid[i], at[i],
            bt[i], ct[i], tat[i], wt[i],
            rt[i]);
printf("\n");
}
```

```
printf("\n Average tat = %f", avg_tat/n);
printf("\n Avg WT = %f", avg_wT/n);
return(0);
}
```

Rate monotonic CPU,

O/P.

Enter no of process : 2.

Enter the CPU burst time :

20 35.

Enter Time period

50 100

Rate monotonic scheduling

PID	BT	Period.
1	20	50
2	35	100

0.7500 <= 0.8284 => true

0 ms onward P1 is running

20 ms onwards Process 2 is running.

50 ms " Process 1 is running

70 " Process 2 is running

75 " CPU is idel.

5/6/24

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 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 no of processes ");
    scanf("%d", &n);
    int proc[n], b[n], pt[n], rem[n];
    printf("Enter CPU burst times:\n");
    for (int i=0; i<n; i++) {
        scanf("%d", &b[i]);
        rem[i] = b[i];
    }
    printf("Enter the no of 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, b, pt, n);
    int L = LCM(pt, n);
    printf("LCM = %d\n", L);
    printf("\n Rate Monotonic Scheduling ");
    printf("PID \t Burst Time \t Period
                \n");
    for (int i=0; i<n; i++)
    {
        printf("%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/4 <= %f => %.5 \n",
        sum, rhs, sum <= rhs) "True"
                          : "False");
if (sum > rhs) exit(0);

printf ("scheduling occurs for %d ", h);

int time = 0; prev = 0; x = 0;
while (time < h) {
    int j = 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 ms switch. process
                        %d running \n",
                        time, proc[i]);
                prev = proc[i];
            }
            rem[i] = -;
            j = 1;
            break
            x = 0
        }
    }
}
```

```
if (!F) {
    if (x != 1) {
        printf("%d ns ~~one~~ onwards:
                CPU is %d ns\n", time);
        x = 1;
    }
}
time++;
}
}
```

~~Enter the no of process : 2~~
~~Enter the CPU burst~~

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;
}

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;
}

int main () {
    int n, t;
    printf (" \n 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 (" \n Enter arrival time, burst
                time , priority : ");
        scanf ("%d %d %d", &at[i], &bt[i],
                &prior[i]);
        pid[i] = i+1;
    }
```

```
sort(pid, at, bt, prio, n);
for(int i=0; i<n; i++)
{
    bt2[i] = bt1[i];
    rt[i] = -1;
}
int curr=0, count=0, ctime=at[0], cni=0;
while(count!=n) {
if(rt[cni]==-1){
    rt[cni] = ctime - at[cni];
}
if(cni==n){
    ctime += bt2[cni]; bt[cni]=0;
}
else {
    ctime++; bt2[cni]--;
}
for(int i=0; at[i]<=ctime; i++)
{
    cni++; n=i;
}
if(bt2[cni]==0){
    count++; ct[cni] = ctime;
    prio[cni] = -1;
}
}
for(int i=0; i<n; i++) {
    tat[i] = ct[i] - at[i];
    wt[i] = tat[i] - bt[i];
}
```

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

printf("\n pid \t at \t bt \t ct \t
        tat \t wt \t rt \n\n");

for(int i=0; i<n; i++){
printf("\n %d \t %d \t %d \t %d \t %d \t %d
    \t %d \t %d \t %d",
    pid[i], at[i], bt1[i], ct[i],
    tat[i], wt[i], rt[i]);
}

printf("\n Avg tat = %f", avg_tat/n);
printf("\n Avg WT = %f", avg_wt/n);

return 0;
}
```

Enter the number of process : 3.
Enter priorities : 10 20 30.40
Enter arrival time: 0 1 2 4.
Enter burst time : 5 4 2 1

PID	Priority	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 = 5.5ms
Avg WT = 2.7ms.

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; 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;
}

int findlcm (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 the no of process");
    scanf ("%d", &n);
    int proc[n], bt[n], pt[n], d[n],
        rem[n];
    printf("Enter the CPU burst time: \n");
    for (int i = 0; i < n; i++) {
        scanf ("%d", &bt[i]);
        rem[i] = bt[i];
    }
```

printf(" Enter the the deadline \n");

for (int i=0; i<n; i++) { scanf("%d", &d[i]);

printf(" \n Enter the time periods ");

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

{ scanf ("\n %d", &pt[i]);

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

sort (prio, d, b, pt, n);

int L = LCM_nd (pt, n);

printf (" \n Earliest deadline scheduling

printf (" P TD \t B \t Deadline \t

period \n");

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

{

printf (" \n %d \t \t %d \t %d \t %d

prio[i], b[i], d[i], pt[i]);

printf (" Scheduling occurs for %d ms \n\n");

int time = 0, prec = 0, x = 0;

int ndl[n];

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

ndl[i] = d[i]; new[i] = b[i];

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

output:

Enter the no of processes: 3.
Enter CPUBT:
3 2 2.
Enter deadline
7 4 8.
Enter the periods.
20 5 10

Earliest deadline scheduling.

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

Scheduling access for 20ms.

0 ms : 2
1 ms : 1
2 : 1
3 : 1
4 : 3
5 : 3
6 : 2
7 : 2
8 : CPU is idle
9 : 2
10 : 2
11 : 3
12 : 3
13 : "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;

1216724

Proportional code.

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

int main()
{   int n, SOT = 0;

printf("Enter the no of process : ");
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("\n probability of servicing process
             %d is %d %%.\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<L[j+1])
      {
      printf ("%d ns : servicing Tickets of
                  processes %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 eneating"),.
    }.
  }
}.
```

output.

Enter the number of processes : 2

Enter the number of tickets for each process.
PID 1 = 10
PID 2 = 20.
Probability of servicing process 1 is 33%.
Probability of servicing process 2 is 66%.
1ms : Servicing Ticket of process 2.
2ms : Servicing Ticket of process 2.
3ms : Servicing Tickets of process 1.
4ms : Servicing Tickets of process 2.
5ms : Servicing Tickets of process 2.

12/6/24

9.

Producer consumer problem

```
#include <stdio.h>
int muten = 1, full = 0, empty = 7, x = 0;
int main()
{
    int n;
    void producer();
    void consumer();
    int wait(int);
    int signal(int);
    printf("\n1. Producer\n2. Consumer\n3. Exit");

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

```
case 3:
        printf("\n Number of Items remaining
                in buffer: %d \n", x);
        exit(0);
        break;
    }
  }
  return (0);
}

int wait (int s)
{
    return (--s);
}
int signal (int s)
{
    return (++s);
}
void producer() {
    muten = wait (muten);
    full = signal (full);
    empty = wait (empty);
    x++;
    printf("producer produces the item %d", x);
    muten = signal (muten);
}
void consumer() {
  muten = wait (muten);
  full = wait (full);
  empty = signal (empty);
```

printf("\n Consumer consumes items %d", x);

x--;

mutex = signal(mutex);

}

Output;

1: Producer :1

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 : 1

Buffer is full !!!

Enter your choice : 2

Consumer consumes item : 7.

:

Enter your choice : 2.

Buffer is empty !!!

Enter your choice : 3.

No of items remaining in Buffer = 6

19/6/24

Bankers Algorithm

```
#include <stdio.h>
#include <stdbool.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 calculateNEED() {

    for (int i=0; i<NUM_Process; i++) {
        for (int j=0; j<NUM_Process; j++) {
            need[i][j] = maximum[i][j] - allocation[i][j];
        }
    }
}

bool isSafe() {

    int work[NUM_RESOURCES];
    bool finish[NUM_PROCESSES] = {false};
    //initialization
    for (int i=0; i<NUM_RESOURCES; i++) {
        work[i] = available[i];
    }
```

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

        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][i])
        {
            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 Vectors
            \n");
for( i=0; i< NUM_Resources]; i++){
    scanf("%d" & available[i]);
    }.

printf("Enter the maximum 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_Process; i++)
        {
            for(j=0;j< NUM_Resources; j++){
            {
                scanf("%d", & allocation[i][j]);
            }

}
```

```
calculateNeed();

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
    no for the request (0 to %d)",
                NUM_PROCESS-1);
scanf("%d", &process);
printf("Enter the request vector");
    for(i=0; i< NUM_Resources; i++)
        {
            scanf("%d", &request[i]);
        }
    requestResources(process, request);
    return 0;
}
```

Output

Enter the available ~~matrix~~ resource matrix
3 3 2
Enter the max 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
Process 3 is visited
Process 4 is visited.
Process 0 is visited
Process 2 is visited

The system is in safe state.

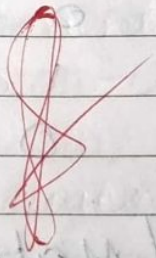
Safe sequence.

1 3 4 0 2.

Enter the process no for Request(0 to 4): 1
Enter the 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.



5/7/24

Deadlock Detection

P.8

```
#include <stdio.h>

#define MAX_Processes 10
#define MAX_Resources 10.

int allocation [MAX_Processes] [MAX_
                                Resources];
int max_need [MAX_Processes] [MAX_Resources]
int ~~or~~ available [MAX_RESOURCES];
int n_processes, n_resources;

void initialize ( ) {.

printf(" Enter the no of process ");
      scanf("%d", & n_processes);

printf(" Enter ~~the~~ number of ~~process~~ resources ");
      scanf ("%d", & n_ resources);

printf (" Enter the allocation matrix (%d &
      %d) : \n", n_ processes, n_ resources);

   for (i=0; i< n_processes ; i++)
      {
         for (j=0; j< n_resources; j++){
            scanf("%d", & allocation [i][j]);
      }
   }
```

~~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 deted_deadlock() {
 int i, j, k;
 int work [MAX_Resources];
  int finish[ MAX_Process];
 int safesequence [MAX_Process];
 int safe_sequence_count =0;
```

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

for( i=0; i<n_process; i++){
    finish[i]=0;
}

// Process to allocate to the resources.

int found;
do {
    found =0;
    for( i=0; i<n_processes; i++)
    {
        if(!finish[i]) {
            int can_allocate =1;
            for( j=0; j<n_resources; j++)
            {
                if (max_need[i][j]-
                    allocation[i][j]>
                    work[j])
                {
                    can_allocate=0;
                    break;
                }
            }

if (can_allocate) {
```

```
if (can_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;
    }

    printf("\n");
}

int main() {

    initialize();
    detect_deadlock();
    return 0;
}
```

Output

Enter the processes : 5
Enter the resources : 3.
Enter allocati
3 7

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[j] = 1;
                    break;
                }
            }
        }
    }
}
```

```
printf("\n Memory Mangement scheme
        - First 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 \t %d \t \t ~~int~~ \n",
      ~~ff[i], b[ff[i]], frag[i]~~);
}          i, f[i]);

if (ff[i] != 0) {

    printf("%d \t \t %d \t \t %d \n",
        ff[i], b[ff[i]], frag[i]);
  }
else {
    printf("Not Allocated \n");
  }
}
}

void bestfit(int nb, int nf, int b[],
                int f[])
{
  int frag[MAX], bf[MAX]={0}, ff[MAX]
                    ={0};
  int i, j, temp, lowest=10000;
```

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

printf("\n 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\t %d \t\t ", i, f[i]);
    if(ff[i]!=0){
        printf("%d \t\t %d \t\t %d\n",
            ff[i], b[ff[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("\n 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 ; ;++)
    {
        printf("%d \t \t %d \t \t", i,
                                    f[i]);
        if (ff[i] != 0) {
            printf("%d \t \t %d \t \t %d
                \n", ff[i], b[ff[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("\nEnter 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];
}

firstFit(nb, nf, b1, f);
bestfit(nb, nf, b2, f);
worstfit(nb, nf, b3, f);

return 0;
}
```

printf("\n Memory Management Scheme - Worst fit\n");

Pages

Output

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

enter the size of the blocks.
b_1 : 5
b_2 : 2
b_3 : 7

enter size of files :
f_1 = 1
f_2 = 4

Memory Management Scheme - first fit.

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

Worst fit

fno	fsize	block no	block size
1	1	3	7
2	4	1	5.

Best fit

fno	filesize	blockno	blocksize
1	1	2	2
2			5

10/7/24

Lab-8

Page Replacement Algorithm

```
#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;
}

// Funct 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");
}

// Function to implement FIFO page replacement

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

    // 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++;
        }
        printFrames(frames, numFrames);
}
```

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

}

// 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++)
      {
      int j;
      for( j = currentIndex ; j < numPa-
                ges ; j++)
        {
          if (frames[i] == pages[j])
            {
              if (j > farthest) {
                farthest = j ;
                index = j;
              }
              break;

      if (j == numPages) {
        return i;
      }
}
```

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++)
{
frames[i] = -1;
}

printf("Optimal 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])) {
if (isPagePresent(frame, numFrames, -1)) {

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

            }
        }
    }
    else {
        int index = findOptimalRepeatReplacement
                    Index(pages, numPages,
                    frames, numFrames, i+1);
        frames[index] = pages[i];
    }
    pageFaults++;
}
printFrames(frames, numFrames);

}
printf("\nTotal page faults : %d\n\n",
        pageFaults);

}

//Function to implement LRU page replacement

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 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++)
{
    scanf("%d", &pages[i]);
}
fifoPageReplacement(pages, numPages,
                    numFrames);
optPageReplacement(pages, numPages,
                   numFrames);
lruPageReplacement(page, numPages, numFrames)
return 0;
}
```

Output

enter no of frames = 3

enter the no of pages = 20

enter the reference string: 7 0 1 2 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	1	1	1	1	7	7	7
	0	0	0	0	3	3	3	2	2	2	3	3	3	2	2	2	0	0	0
		1	1	1	1	0	0	0	3	3	3	2	2	2	0	0	0	1	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 0 7 7 7
- 0 0 0 0 3 3 3 2 2 2 2 1 1 1 1 1 0 0
- - 1 1 1 1 0 0 0 3 3 3 3 3 2 2 2 1 1

page faults = 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

page Routs = 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
- 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.