INDEX

Name MADHUSHREE. S. SHETTY

Standard CSE Section 4C Roll No CS141

Subject OS Observation book.

SL No.	Date	Index	Page No.	Teacher Sign / Remarks
1.	8/5/24	Lab program 1 - FCFS, SJF		8/5/24
~~2.~~	~~15/5/24~~	SJF (Non Pre-Emptive & Pre-Emptive)		21/5
2.	15/5/24	Priority (Pre-Emptive & NPE) and Round Robin Scheduling.		
3.	22/5/24	Lab Question - 3		5/6/24
4.	5/6/24	Lab Question - 4		12/6
5.	12/6/24	Lab Question - 5		19/6
6.	19/6/24	Lab Question - 6 & 7		3/7/24
7.	3/7/24	Deadlock detection and contiguous memory allocation.		
8.	10/7/24	Page replacement algorithm.		10/7/24

8/5/24.

LAB-01. : Lab program.

① write a c program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time.

→ FCFS → SJF (pre-emptive & non-preemptive).

```
//FCFS
# include <stdio.h>
void sort (int proc_id[], int at[], int bt[], int n)
{       int min = at[0], temp = 0;
        for (int i=0 ; i<n ; i++) min = at[0];
        {   for (int j=0; j<n; j++)
            if (at[j] < min)
            {
                temp = at[i]; at[i] = at[j] ;
                at[j] = temp;
                temp = bt[i] ; bt[i] = bt[j];
                bt[j] = temp;
                temp = proc_id[i] ; proc_id[i] = proc_id[j];
                proc_id[j] = temp ;
            }
        }
}
void main()
{       int n, c=0;
        printf("Enter number of processes : ");
        scanf("%d", &n);
        int proc_id[n], at[n], bt[n], ct[n], tat[n], wt[n];
        double avg_tat = 0.0, ttat = 0.0, avg_wt = 0.0, twt = 0.0;
        for (int i = 0 ; i<n ; i++)
                scanf("%d", &at[i]); proc_id[i] = i+1 ;
        printf("Enter arrival times : \n");
        for (int i=0; i<n ; i++)
                scanf("%d", &at[i]);
        printf("Enter burst times : \n");
        for (int i=0; i<n ; i++)
                scanf("%d", &bt[i]);

        sort(proc_id, at, bt, n);
```

```
for(int i=0;i<n;i++)
{
    if(c>=at[i])
        c+=bt[i];
    else
        c+=at[i]-ct[i-1]+bt[i];
    ct[i]=c;
}
for(int i=0; i<n; i++)
    tat[i]=ct[i]-at[i];
for(int i=0;i<n;i++)
    wt[i]=tat[i]-bt[i];
printf("FCFS scheduling:\n");
printf("PID\tAT\tBT\tCT\tTAT\twt\n");
for(int i=0;i<n;i++)
    printf("%d\t%d\t%d\t%d\t%d\t%d\n", proc_id[i],
    at[i], bt[i], ct[i], tat[i], wt[i]);
for(int i=0; i<n; i++)
{   ttat+=tat[i]; twt+=wt[i]; }
printf("\nAverage turnaround time: %lf ms\n",
    ttat/(double)n);
printf("\nAverage waiting time: %lf ms\n", avg-
    twt/(double)n);
}
```

Output:

```
Enter number of processes: 4
Enter arrival times:
0 1 5 6
Enter burst times:
2 2 3 4
FCFS scheduling:
```

PID	AT	BT	CT	TAT	WT
1	0	2	2	2	0
2	1	2	4	3	1
3	5	3	8	3	0
4	6	4	12	6	2

8/5/24

```
Average turnaround time: 3.500000 ms
Average waiting time: 0.750000 ms
```

15/9/24

// SJF (Non - Pre-Emptive).

```
#include<stdio.h>
void main()
{
    int n,c=0;
    printf("Enter number of processes:");
    scanf("%d",&n);
    int proc_id[n],at[n],bt[n],ct[n],tat[n],wt[n],m[n];
    double avg_tat=0.0,ttat=0.0,avg_wt=0.0,
    twt=0.0;
    for(int i=0;i<n;i++)
    { proc_id[i]=i+1; m[i]=0;}
    printf("Enter burst times:\n");
    for(int j=0;j<n;j++)
        scanf("%d",&bt[j]);
    printf("Enter arrival times:\n");
    for(int j=0;j<n;j++)
        scanf("%d",&at[j]);

    // completion time
    int count=0,mb=0,p=0,min=0;
    while(count<n)
    {
        min=bt[0];mb=0;
        for(int i=0;i<n;i++)
        {
            if(at[i]<=c && m[i]!=1)
            {
                min=bt[i];mb=i;
                for(int k=0;k<n;k++)
                {
                    if(bt[k]<min && at[k]<=c &&
                    m[k]!=1)
                    { min=bt[k];mb=k;}
                }
                m[mb]=1;count++;
                if(c>=at[mb])
                    c+=bt[mb];
                else
                    c+=at[mb]-ct[p]+bt[mb];
                ct[mb]=c;
            }
            p=mb;
            if(count==n)
            break;
        }
    }
```

```
// turnaround time
for (int i=0 ; i<n; i++)
    tat[i] = ct[i] - at[i];
// waiting time
for (int i=0; i<n; i++)
    wt[i] = tat[i] - bt[i];

printf("FCFS scheSJF scheduling: \n");
printf("PID \t AT \t BT \t CT \t TAT \t WT \n");
for (int i=0; i<n; i++)
{ printf("p%d \t %d \t %d \t %d \t %d \t %d \n",
    proc_id[i], at[i], bt[i], ct[i], tat[i], wt[i]);
    ttat += tat[i] ; twt += wt[i];
}
avg_tat = ttat/(double) n;
avg_wt = twt/(double) n;
printf("\n Average turnaround time : %.4f ms \n",
    avg_tat );
printf("\n Average waiting time : %.4f ms \n", avg_wt);
}
```

Output:

```
Enter the number of processes: 5
Enter arrival times:
1 4 0 3 1
Enter burst times:
3 1 2 5 4
Enter arrival times:
1 4 0 3 1

SJF scheduling:
```

PID	AT	BT	CT	TAT	WT
P1	1	3	5	4	1
P2	4	1	6	2	1
P3	0	2	2	2	0
P4	3	5	15	12	7
P5	1	4	10	9	5

```
Average turnaround time : 5.800000 ms
Average waiting time : 2.800000 ms
```

```
// Pre-Emptive.
#include <stdio.h>
void main()
{   int n, c=0;
    printf("Enter number of processes:");
    scanf("%d",&n);
    int proc_id[n],at[n],bt[n],ct[n],tat[n],wt[n], m[n],
    b[n];
    double avg_tat=0.0, ttat=0.0, avg_wt=0.0, twt=0.0;
    for(int i=0; i<n; i++)
    {  proc_id[i]=i+1; m[i]=0; }
    printf("Enter arrival times:\n");
    for(int i=0; i<n; i++)
        scanf("%d",&at[i]);
    printf("Enter burst times:\n");
    for(int i=0; i<n; i++)
    {   scanf("%d",&bt[i]); b[i]=bt[i];}

    // completion time
    int count=0, mb, p=0, min=0;
    while(count<n)
    {
        min=b[0]; mb=0;
        for(int i=0; i<n; i++)
        {
            if(at[i]<=c && m[i]!=1)
            {   min=b[i]; mb=i;
                for(int k=0; k<n; k++)
                {
                    if(b[k]<=min && at[k]<=c && m[k]!=1)
                    {  min=b[k]; mb=k; }
                }
                if(b[mb]==1)
                { m[mb]=1; count++; }
                if(c>=at[mb])
                {   c++; b[mb]--; }
                else
                    c+=at[mb]-ct[p];
                if(b[mb]==0)
                    ct[mb]=c;
            }
            p=mb;
            if(count==n)
            break;
        }
}
```

```
// turnaround time & waiting time.
for (int i=0 ; i<n ; i++)
{
    tat[i] = ct[i] - at[i];
    wt[i] = tat[i] - bt[i];
}
printf ("SJF (Pre-Emptive) scheduling : \n");
printf ("PID \t AT \t BT \t CT \t TAT \t WT \n");
for (int i=0; i<n; i++)
{ printf ("P%.d \t%.d \t%.d \t%.d \t%.d \t%.d \n", proc-id[i],
  at[i], bt[i], ct[i], tat[i], wt[i]);
  ttat += tat[i] ; twt += wt[i];
}
avg-tat = ttat/(double)n;
avg-wt = twt/(double)n;
printf ("\n Average turnaround time : %.4f ms \n", avg-tat);
printf ("\n Average waiting time : %.4f ms \n", avg-wt);
}
```

Output:

Enter number of processes : 5
Enter arrival times :
1 4 0 3 1
Enter burst times :
1 1 3 5 4
SJF (Pre-Emptive) scheduling :

PID	AT	BT	CT	TAT	WT
P1	1	1	2	1	0
P2	4	1	5	1	0
P3	0	3	4	4	1
P4	3	5	14	11	6
P5	1	4	9	8	4

Average turnaround time : 5.000000 ms
Average waiting time : 2.200000 ms

15/2/24. LAB-02

② Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time.
→ Priority (Pre-emptive & non pre-emptive)
→ Round Robin (Experiment with different quantum sizes for RR algorithm).

⇒ // Pre-Emptive Priority scheduling and non pre-emptive.

```
#include<stdio.h>
void sort(int proc_id[],int p[],int at[],int bt[],
int b[],int n)
{
    int min=p[0],temp=0;
    for(int i=0;i<n;i++)
    {
        min=p[i];
        for(int j=i;j<n;j++)
        {
            if(p[j]<min)
            {temp=b[i];b[i]=b[j];b[j]=temp;
            temp=at[i];at[i]=at[j];at[j]=temp;
            temp=p[i];p[i]=p[j];p[j]=temp;
            temp=bt[i];bt[i]=bt[j];bt[j]=temp;
            temp=proc_id[i];proc_id[i]=proc_id
            [j];proc_id[j]=temp;
        }
    }
}

void main()
{
    int n,c=0;
    printf("Enter number of processes:");
    scanf("%d",&n);
    int proc_id[n],at[n],bt[n],ct[n],tat[n],wt[n],
    m[n],b[n],rt[n],p[n];
    double avg_tat=0.0,ttat=0.0,avg_tat wt=0.0,
    twt=0.0;
    for(int i=0;i<n;i++)
        proc_id[i]=i+1;
    printf("Enter priorities:\n");
    for(int i=0;i<n;i++)
        scanf("%d",&p[i]);
    printf("Enter arrival times:\n");
```

```
for (int i=0; i<n; i++)
    scanf("%d", &at[i]);
printf("Enter burst times:\n");
for (int i=0; i<n; i++)
{   scanf("%d", &bt[i]); b[i] = bt[i];
    m[i] = -1; wt[i] = -1;
}
sort(proc_id, p, at, bt, b, n);

// completion time
int count = 0, pro = 0, priority = p[0];
int x=0; c=0;
while(count<n)
{   for (int i=0; i<n; i++)
        if (at[i]<=c && p[i]>=priority && b[i]>0
        && m[i]!=1)
        {   x=i; priority = p[i]; }
    if (b[x]>0)
    {   if (wt[x]==-1)
            wt[x] = c-at[x];
        b[x]--; c++;
    }
    if (b[x]==0)
    {   count++; ct[x] = c; m[x]=1;
        while(x>=1 && b[x]==0)
        {   priority = p[--x]; break; }
    }
    if (count==n)
        break;
}
// turnaround time and waiting time
for (int i=0; i<n; i++)
{   tat[i] = ct[i] - at[i];
    wt[i] = tat[i] - bt[i];
}
printf("Priority scheduling (Pre-Emptive):\n");
printf("PID\t Prior\t AT\t BT\t CT\t TAT\t WT
    \t RT\n");
for (int i=0; i<n; i++)
{   printf("P%d\t %d\t %d\t %d\t %d\t %d\t %d\t %d\t
    %d\t", proc_id[i], p[i], at[i], bt[i], ct[i],
    tat[i], wt[i];
```

```
        ttat += tat[i]; twt += wt[i];
    }
    printf("\nAverage turnaround time: %.1f ms\n",
        ttat/(double)n);
    printf("\nAverage waiting time: %.1f ms\n",
        twt/(double)n);
}
```

Output:

```
Enter number of processes: 4
Enter priorities:
10  20  30  40
Enter arrival times:
0  1  2  4
Enter burst times:
5  4  2  1
```

Priority scheduling (Pre-Emptive):

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

Average turnaround time: 5.500000ms

Average waiting time: 2.500000ms

⇒// Non-Pre-Emptive Priority:

```
# completion time include<stdio.h>
void sort (int proc_id[], int p[], int at[], int bt[],
    int n)
{   int min = p[0], temp = 0;
    for (int i=0; i<n; i++)
    {
        min = p[i];
        for (int j=i; j<n; j++)
        {
            if (p[j] < min)
            {
                temp = at[i]; at[i] = at[j]; at[j] = temp;
                temp = bt[i]; bt[i] = bt[j]; bt[j] = temp;
                temp = p[i]; p[i] = p[j]; p[j] = temp;
                temp = proc_id[i]; proc_id[i] = proc_id[j];
                proc_id[j] = temp;
}   }   }   }
```

```
void main()
{
    int n=0,c=0;
    printf("Enter number of processes: ");
    scanf("%d",&n);
    int proc_id[n], at[n], bt[n], ct[n], tat[n], wt[n]
    m[n], rt[n], p[n];
    double ttat=0.0, twt=0.0;
    for(int i=0; i<n; i++)
        proc_id[i] = i+1;
    printf("Enter priorities: \n");
    for(int i=0; i<n; i++)
        scanf("%d", &p[i]);
    printf("Enter arrival times: \n");
    for(int i=0; i<n; i++)
        scanf("%d", &at[i]);
    printf("Enter burst times: \n");
    for(int i=0; i<n; i++)
    { scanf("%d", &bt[i]); m[i]=-1; rt[i]=-1; }

    sort(proc_id, p, at, bt, n);

    // completion time
    int count=0, pre=0, priority = p[0];
    int x=0; c=0;
    while(count<n)
    {
        for(int i=0; i<n; i++)
            if(at[i]<=c && p[i]>=priority && m[i]!=1)
            { x=i; priority = p[i]; }
        if(rt[x]==-1)
            rt[x] = c-at[x];
        if(at[x]<=c)
            c+=bt[x];
        else
            c+=at[x]-c+bt[x];
        count++; ct[x]=c; m[x]=1;
        while(x>=1 && m[--x]!=1)
        { priority = p[x]; break; }
        x++;
        if(count==n)
        break;
    }
```

```
// turnaround time and waiting time
for (int i=0; i<n; i++)
{   tat[i] = ct[i] - at[i];
    wt[i] = tat[i] - bt[i];
}
printf("\n Priority scheduling:\n");
printf("PID\t Prior\t AT\t BT\t CT\t TAT\t WT\t RT\n");
for (int i=0; i<n; i++)
   printf("P%d\t %d\t %d\t %d\t %d\t %d\t %d\t %d\t %d\t",
   proc_id[i], p[i], at[i], bt[i], ct[i], tat[i], wt[i],
   rt[i]);
   for (int i=0; i<n; i++)
   {   ttat += tat[i]; twt += wt[i]; }
printf("\n Average turnaround time : %.4f ms", ttat/
 (double)n);
printf("\n Average waiting time : %.4f ms", twt/(double)n);
}
```

Output:

```
Enter priorities:
10    20    30    40
Enter arrival times:
0    1    2    4
Enter burst times:
5    4    2    1
```

Priority scheduling:

PID	Prior	AT	BT	CT	TAT	WT	RT
P1	10	0	5	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

Average turnaround time : 6.000000 ms

Average waiting time : 3.000000 ms

⇒ // RRS

```
sort(proc_id, at, bt, b, n);
int f, r=0; f=0;
int q[100], count=0; q[0]=proc_id[0];

// completion time
int p=0, i=0;
while(f>=0)
{   p=q[f++]; i=0;
    while(p!=proc_id[i])
        i++;
    if(b[i]>=t)
    {   if(wt[i]==-1)
            wt[i]=c;
        b[i]-=t; c+=t; m[i]=1;
    }
    else
    {   if(wt[i]==-1)
            wt[i]=c;
        c+=b[i]; b[i]=0; m[i]=1;
    } m[0]=1;
    for(int j=0; j<n; j++)
        if(at[j]<=c && proc_id[j]!=p && m[j]!=1)
        {   q[++r]=proc_id[j];
            m[j]=1; }
    if(b[i]==0)
    {   count++; ct[i]=c; }
    else
        q[++r]=proc_id[i];
    if(f>r)
        f=-1;
}

// turnaround time, response time and waiting time
for(int i=0; i<n; i++)
{   tat[i]=ct[i]-at[i];
    wt[i]=wt[i]-at[i];
    wt[i]=tat[i]-bt[i];
}
```

Output:

Enter Number of processes: 5
Enter Time Quantum: 3
Enter arrival times:
0 5 1 6 8
Enter burst times:
8 4 7 3 5

RRS scheduling:

PID	AT	BT	CT	TAT	WT	RT
1	0	8	22	22	14	0
3	1	7	23	22	15	2
2	5	2	11	6	4	4
4	6	3	14	8	5	5
5	8	5	25	17	12	9

Average turnaround time: 15.000000ms
Average waiting time: 10.000000ms

21/5/2024

LAB-02

Q. Write a C program to simulate Real-Time CPU Scheduling algorithms:

a) Rate-Monotonic

b) Earliest-deadline first.

a)

```
#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=i; j<n; j++)
        {   if(pt[j]<pt[i])
            {  temp=pt[i]; pt[i]=pt[j]; pt[j]=temp;
               temp=b[i]; b[i]=b[j]; b[j]=temp;
               temp=proc[i]; proc[i]=proc[j];
               proc[j]=proc[i]temp;
    }  }  }
}

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

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 the number of processes:");
    scanf("%d",&n);
    int proc[n], b[n], pt[n], rem[n];
    printf("Enter the CPU burst times:\n");
    for(int i=0; i<n; i++)
    {  scanf("%d",&b[i]); rem[i]=b[i]; }
```

```
printf ("Enter the time periods: \n");
for (int i=0; i< n; i++)
{  scanf("%d", &pt[i]); proc[i] = i+1; }

sort (proc, b, pt, n);

int l = lcmul(pt, n);

printf("\n Rate Monotone Scheduling: \n");
printf("PID \t Burst \t Period \n");
for (int i=0; i<n; i++)
   printf ("%d \t\t %d \t\t %d \n", proc[i], b[i],
    pt[i]);

// feasibility
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%.4f <= %.4f => %s\n", sum, rhs,
(sum <= rhs) ? "true" : "false");
if (sum > rhs)
   exit(0);
printf("scheduling occurs for %d ms \n\n", l);

//RMS
int time = 0, prev = 0, x = 0;
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 ms onwards: Process %d
               running \n", time, proc[i]);
               prev = proc[i];
             }
             rem[i]--; f=1; break; x = 0;
    }    }
     if (!f)
     {   if (x != 1)
         { printf("%d ms onwards: CPU is idle \n", time);
           x = 1; }}
     time++; }}
```

Output:

Enter number of processes: 2
Enter the CPU burst times:
20 35
Enter the time periods:
50 100

Rate Monotone Scheduling:

PID	Burst	Period
1	20	50
2	35	100

0.750000 <= 0.828427 ⇒ 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.

b)
```
#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[i]; d[i]=d[j]; d[j]=temp;
               temp=b[i]; b[i]=b[j]; b[j]=temp;
               temp=proc[i]; proc[i]=proc[j];
               proc[j]=temp;
               temp=pt[i]; pt[i]=pt[j]; pt[j]=temp;
    }  }  }
}
int gcd(int a, int b)
{   int r;
    while(b>0)
    {  r=a%b; a=b; b=r; }
}   return a;
int lcmul(int p[],int n)
{   int lcm=p[0];
```

```
    for (int i=0; i<n; i++)
        lcm = (lcm * p[i]) / gcd(lcm, p[i]);
    return lcm;
}
void main()
{   int n;
    printf("Enter the number of processes");
    scanf("%d",&n);
    int proc[n], b[n], pt[n], d[n], rem[n];
    printf("Enter the CPU burst times:\n");
    for(int i=0; i<n; i++)
    {   scanf("%d",&b[i]); rem[i]=b[i]; }
    for(int i=0; i<n; i++)
    printf("Enter the deadlines:\n");
    for(int i=0; i<n; i++)
        scanf("%d", &d[i]);
    for(int i=0; i<n; i++)
    printf("Enter the time periods:\n");
    {   scanf("%d", &pt[i]); proc[i]=i+1; }

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

    int l = lcmul(pt, n);

    printf("\nEarliest Deadline scheduling:\n");
    printf("PID \t Burst \t Deadline \t Period \n");
    for(int i=0; i<n; i++)
      printf("%d\t\t%d\t\t%d\t\t%d\n", proc[i],
      b[i], d[i], pt[i]);

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

    //EDF
    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];
            }
        int minDeadline = l+1;
        int taskToExecute = -1;
```

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

time++;
    }
}
```

Output:

Enter the number of processes: 3
Enter the CPU burst times:
3 2 2
Enter the deadlines:
4 4 8
Enter the time periods:
20 5 10

Earliest deadline scheduling:

PID	Burst	Deadline	Period
2	2	4	5
1	3	4	20
3	2	8	10

Scheduling occurs for 20ms.

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

LAB-03

Q. write a c program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into 2 categories - system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for the processes in each queue.

```
#include<stdio.h>
void sort(int proc_id[], int at[], int bt[], int n)
{   int temp=0;
    for(int i=0; i<n; i++)
    {   for(int j=0; j<n; j++)
        {   if(at[j]<at[i])
            {   temp=at[i]; at[i]=at[j]; at[j]=temp;
                temp=bt[i]; bt[j]=bt[i]; bt[i]=temp;
                temp=proc_id[i]; proc_id[i]=
                proc_id[j]; proc_id[j]=temp;
            }
        }
    }
}
void fcfs(int at[], int bt[], int ct[], int tat[], int wt[],
          int n, int *c)
{   // completion time
    for(int i=0; i<n; i++)
    {   if(*c>=at[i])
            *c+=bt[i];
        else
            *c+=at[i]-ct[i-1]+bt[i];
        ct[i]=*c;
    }
    // turnaround time & waiting time
    for(int i=0; i<n; i++)
    {   tat[i]=ct[i]-at[i];
        wt[i]=tat[i]-bt[i];
    }
}
void main()
{   int sn, un, c=0, n=0;
    printf("Enter number of system processes:");
    scanf("%d", &sn); n=sn;
    int sproc_id[n], sat[n], sbt[n], sct[n], stat[n],
    swt[n];
```

```
for (int i=0; i<n; i++)
    sproc_id[i] = i+1;
printf("Enter the arrival times of the system
 processes :\n");
for (int i=0; i<n; i++)
    scanf("%d", &sat[i]);
printf("Enter burst times of the system processes:\n");
for (int i=0; i<n; i++)
    scanf("%d", &sbt[i]);

printf("Enter number of user processes:");
scanf("%d", &un); n=un;
int uproc_id[n], uat[n], ubt[n], uct[n], utat[n], uwt[n];
for(int i=0; i<n; i++)
    uproc_id[i] = i+1;
printf("Enter arrival times of the user processes:\n");
for(int i=0; i<n; i++)
    scanf("%d", &uat[i]);
printf("Enter burst times of the user processes:\n");
for (int i=0; i<n; i++)
    scanf("%d", &ubt[i]);

sort(sproc_id, sat, sbt, sn);
sort(uproc_id, uat, ubt, un);

fcfs(sat, sbt, sct, stat, swt, sn, &c);
fcfs(uat, ubt, uct, utat, uwt, un, &c);

printf("\n Scheduling \n");
printf("System processes :\n");
printf("PID\tAT\tBT\tCT\tTAT\tWT\t\n");
for(int i=0; i<sn; i++)
    printf("%d\t%d\t%d\t%d\t%d\t%d\n", sproc-
    id[i], sat[i], sbt[i], sct[i], stat[i], swt[i]);
printf("User processes:\n");
for(int i=0; i<un; i++)
    printf("%d\t%d\t%d\t%d\t%d\t%d\n",
 uproc-id[i], uat[i], ubt[i], uct[i], utat[i],
 uwt[i]);
```

}

Output.

Enter number of system processes: 3
Enter arrival times of the system processes:
5 1 4
Enter burst times of the system processes:
3 2 1
Enter number of user processes: 3
Enter arrival times of the user processes:
2 6 3
Enter burst times of the system processes:
4 3 2

Scheduling:

System processes:

PID	AT	BT	CT	TAT	WT
2	1	2	3	2	0
3	4	1	5	1	0
1	5	3	8	3	0

User processes:

1	2	4	12	10	6
3	3	2	14	11	9
2	6	3	17	11	8

Rr

5 | 6 | 2n

4

5/6/24

LAB-04 : Proportional scheduling

⇒
```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
void main()
{
    srand(time(NULL));
    int n;
    printf("Enter number of processes:");
    scanf("%d",&n);
    int p[n],t[n],cum[n],m[n],c=0,total=0,
    count=0;
    printf("Enter tickets of the processes:\n");
    for(int i=0;i<n;i++)
    {
        scanf("%d",&t[i]);
        c+=t[i]; cum[i]=c;
        p[i]=i+1; m[i]=0;
        total+=t[i];
    }
    while(count<n)
    {
        int wt=rand()%total;
        for(int i=0;i<n;i++)
        {
            if(wt<cum[i] && m[i]==0)
            {
                printf("The winning number is %d
and winning process is:%d\n",wt,p[i]);
                m[i]=1; count++;
            }
        }
    }
    printf("\nProbabilities\n");
    for(int i=0;i<n;i++)
    {
        printf("The probability of P%d winning:
        %0.2f %\n",p[i],((double)t[i]/
        total*100));
    }
}
```

Output

```
Enter number of processes : 4
Enter tickets of the processes :
10 20 30 40
The winning number is 95 and winning process is : 4
The winning number is 54 and winning process is : 3
The winning number is 3 and winning process is : 1
The winning number is 3 and winning process is : 2

Probabilities:
The probability of P1 winning : 10.00%.
The probability of P2 winning : 20.00%.
The probability of P3 winning : 30.00%.
The probability of P4 winning : 40.00%.
```

Run
6/6/24

12/6/24

LAB-05

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

```
#include<stdio.h>
#include<stdlib.h>
int mutex=1, full=0, empty=50, x=0;
void wait(); void signal(){++mutex;} void wait(){--mutex;}
void producer()
{
    wait();
    --mutex; ++full; --empty; x++;
    printf("Producer has produced : Item %d\n", x);
    mutex++; signal();
}
void consumer()
{
    wait();
    --mutex; --full; ++empty;
    printf("Consumer has consumed : Item %d\n", x);
    x--; mutex++; signal();
}
void main()
{
    int ch;
    printf("Enter 1. Producer 2. Consumer 3. Exit\n");
    while(1)
    {
        printf("Enter your choice:\n");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1:
                if(mutex==1 && empty!=0)
                    producer();
                else
                    printf("Buffer is full!\n");
                break;
            case 2:
                if(mutex==1 && full!=0)
                    consumer();
                else
                    printf("Buffer is empty!\n");
                break;
            case 3: exit(0);
            default: printf("Invalid choice!\n");
        }
    }
}
```

Output:

```
Enter 1. Producer 2. Consumer 3. Exit
Enter your choice:
1
Producer has produced : Item 1
Enter your choice:
2
Consumer has consumed : Item 1
Enter your choice:
2
Buffer is empty!
Enter your choice:
1
Producer has produced : Item 1
Enter your choice:
1
Producer has produced : Item 2
Enter your choice:
1
Producer has produced : Item 3
Enter your choice:
1
Producer has produced : Item 4
Enter your choice:
1
Producer has produced : Item 5
Enter your choice:
1
Buffer is full!
Enter your choice:
3
```

Pm
10/6/24

19/6/24

LAB-06

(Q1). write C program to simulate the concept of ~~Dining - Philosophers~~ problem.
Banker's algorithm

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

void calculateNeed(int p, int r, int need[p][r], int
max[p][r], int allot[p][r])
{       for(int i=0; i<p; i++)
            for(int j=0; j<r; j++)
              need[i][j] = max[i][j] - allot[i][j];
}
bool int isSafe(int p, int r, int proc[], int avail[],
   int max[][r], int allot[][r])
{       int need[p][r];
        calculateNeed(p, r, need, max, allot);
        int finish[p];
        for(int i=0; i<p; i++)
              finish[i] = 0;
        int safeseq[p]; int work[r];
        for(int i=0; i<r; i++)
              work[i] = avail[i];
        int count=0;
        while(count<p)
        {     int found=0;
              for(int pr=0; pr<p; pr++)
              {   if(finish[pr]==0)
                  {   int j;
                      for(j=0; j<r; j++)
                        if(need[p][j]>work[j])
                          break;
                      if(j==r)
                      {  printf("P%d is visited(", pr);
                        for(int k=0; k<r; k++)
                        {  work[k]+=allot[p][k];
                          printf("%d", work[k]);
                        }printf(")\n");
```

```
            safeseq[count++] = p; finish[p] = 1; found = true;
        }
    }
    if (found == 0)
    {
        printf("System is not in safe state\n");
        return 0;
    }
}
printf("System is in safe state\n The safe sequence is
--(");
for(int i=0; i<p; i++)
    printf("P%d", safeseq[i]);
    printf(")\n");
return 1;
}
int main()
{
    int p,r;
    printf("Enter number of processes:");
    scanf("%d", &p);
    printf("Enter number of resources:");
    scanf("%d", &r);
    int processes[p]; int avail[r], max[p][r], allot[p][r];
    for(int i=0; i<p; i++)
    {
        processes[i] = i;
    for(int i=0; i<p; i++)
    {
        printf("Enter details for P%d\n", i);
        printf("Enter allocation--");
        for(int j=0; j<r; j++)
            scanf("%d", &allot[i][j]);
        printf("Enter max--");
        for(int j=0; j<r; j++)
            scanf("%d", &max[i][j]);
    }
    printf("Enter Available Resources--");
    for(int i=0; i<R; i++)
        scanf("%d", &avail[i]);

    isafe(p,r, processes, avail, max, allot);
    printf("\nProcess\tAllocation\tMax\tNeed\n");
```

```
for (int i=0; i<p; i++)
{
    printf("P%d\t", i);
    for (int j=0; j<r; j++)
        printf("%d", allot[i][j]);
    printf("\t");
    for (int j=0; j<r; j++)
        printf("%d", max[i][j]);
    printf("\t");
    for (int j=0; j<r; j++)
        printf("%d", max[i][j]-allot[i][j]);
    printf("\n");
}
return 0;
}
```

Output

```
Enter number of processes: 5
Enter number of resources: 3
Enter details for P0
Enter allocation -- 0 1 0
Enter Max -- 7 5 3
Enter details for P1
Enter allocation -- 2 0 0
Enter Max -- 3 2 2
Enter details for P2
Enter allocation -- 3 0 2
Enter Max -- 9 0 2
Enter details for P3
Enter allocation -- 2 1 1
Enter Max -- 2 2 2
Enter details for P4
Enter allocation -- 0 0 2
Enter Max -- 4 3 3
Enter Available Resources -- 3 3 2
P1 is visited (5 3 2)
P3 is visited (7 4 3)
P4 is visited (7 4 5)
P0 is visited (7 5 5)
P2 is visited (10 5 7)
System is in safe state
The safe sequence is -- (P1 P3 P4 P0 P2)
```

Process	Allocation	Max	Need
P0	0 1 0	7 5 3	7 4 3
P1	2 0 0	3 2 2	1 2 2
P2	3 0 2	9 0 2	6 0 0
P3	2 1 1	2 2 2	0 1 1
P4	0 0 2	4 3 3	4 3 1

Q2) Write C program to simulate the concept of Dining-philosopher's problem.

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

#define MAX_PHILOSOPHERS 100

int mutex=1; int mutex2=2;

int philosophers[MAX_PHILOSOPHERS];

void wait(int *sem)
{   while(*sem<=0);
      (*sem)--;
}
void signal(int *sem)
{   (*sem)++; }

void* one_eat_at_a_time(void* arg)
{   int philosopher = *((int*)arg);
    wait(&mutex);
    printf("Philosopher %d is granted to eat\n",
    philosopher+1);
    sleep(1);
    printf("Philosopher %d has finished eating\n",
    philosopher+1);
    signal(&mutex);
    return NULL;
}
void* two_eat_at_a_time(void* arg)
{   int philosopher = *((int*)arg);
    wait(&mutex2)
    printf("Philosopher %d is granted to eat\n",
      philosopher+1); sleep(1);
    printf("Philosopher %d is granted has finished
     eating\n", philosopher+1);
```

```
    signal(&mutex2);
    return NULL;
}
int main()
{
    int N;
    printf("Enter the total number of philosophers: ");
    scanf("%d",&N);
    int hungry_count;
    printf("How many are hungry: ");
    scanf("%d", &hungry_count);
    int hungry_philosophers[hungry_count];
    for(int i=0; i< hungry_count; i++)
    {
        printf("Enter philosopher %d position (1 to %d): ",
        i+1, N);
        scanf("%d", &hungry_philosophers[i]);
        hungry_philosophers[i]--;
    }
    pthread_t thread[hungry_count];
    int choice;
    do{
    printf("\n1. One can eat at a time \n2. Two can eat
    at a time \n3. Exit \n Enter your choice:");
    scanf("%d", &choice);
    switch(choice)
    {   case 1:
            printf("Allow one philosopher to eat at
            any time \n");
            for(int i=0; i< hungry_count; i++)
            {   philosophers[i] = hungry_philosophers[i];
                pthread_create(&thread[i], NULL,
                one_eat_at_a_time, &philosophers[i]);
            }
            for(int i=0; i< hungry_count; i++)
            {   pthread_join(thread[i], NULL); }
            break;
        case 2:
            printf("Allow two philosophers to eat
            at the same time \n");
            for(int i=0; i< hungry_count; i++)
            {   philosophers[i] = hungry_philosophers[i];
                pthread_create(&thread[i], NULL,
                two_eat_at_a_time, &philosophers[i]);
            }
```

```
        for(int i=0; i<hungry_count; i++)
        {   pthread_join(thread[i], NULL); }
        break;
      case 3: printf("Exit\n"); break;
      default:
          printf("Invalid choice!");
    }
  } while(choice!=3);
  return 0;
}
```

Output

```
Enter the total number of philosophers: 5
How many are hungry: 3
Enter philosopher 1 position (1 to 5): 1
Enter philosopher 2 position (1 to 5): 3
Enter philosopher 3 position (1 to 5): 5

1. One can eat at a time
2. Two can eat at a time
3. Exit
Enter your choice: 1
Allow one philosopher to eat at any time
Philosopher 1 is granted to eat
Philosopher 1 is finished eating.
Philosopher 5 is granted to eat
Philosopher 5 has finished eating.
Philosopher 3 is granted to eat
Philosopher 3 has finished eating

1. One can eat at a time
2. Two can eat at a time
3. Exit
Enter your choice: 2
Allow two philosophers to eat at the same time
Philosopher 1 is granted to eat
Philosopher 5 is granted to eat
Philosopher 1 has finished eating
Philosopher 3 is granted to eat
Philosopher 5 has finished eating
Philosopher 3 has finished eating
```

1. One can eat at a time
2. Two can eat at a time
3. Exit
Enter your choice: 3
Exit.

Rw
19/6/24

3/7/24

LAB-07.

① Deadlock detection:

```
#include <stdio.h>

int main()
{
    int n,m,i,j,k;
    printf("Enter the number of processes:");
    scanf("%d",&n);
    printf("Enter the number of resources:");
    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",&alloc[i][j]);
        printf("Enter Request --");
        for(int j=0; j<m; j++)
            scanf("%d",&request[i][j]);
    }
    printf("Enter available resources--");
    for(int i=0; i<m; i++)
        scanf("%d",&avail[i]);
    int finish[n], safeseq[n], work[m], flag, f=0;
    for(int 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++)
                    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;
        }
        if(canProceed)
        {
            for(int k=0; k<m; k++)
                work[k] += alloc[i][k];
            safeSeq[count++] = i;
            finish[i] = 1;
            flag = 1;
        }
    }
    else
    {
        safeSeq[count++] = i;
        finish[i] = 1;
        flag = 1;
    }
}
    if(flag==0)
        break;
}
    int deadlock = 0;
    for(i=0; i<n; i++)
    {
        if(finish[i]==0)
        {
            deadlock = 1;
            printf("\nSystem is in a deadlock state.\n");
            printf("The deadlocked processes are : ");
            for(j=0; j<n; j++)
                if(finish[j]==0)
                    printf("P%d ", j);
            printf("\n");
            break;
        }
    }
    if(deadlock==0)
    {
        printf("\nSystem is not in a deadlock state.\n");
        printf("Safe sequence is : ");
        for(i=0; i<n; i++)
            printf("P%d", safeSeq[i]);
        printf("\n");
    }
    return 0;
}
```

Output:

```
Enter the number of processes: 5
Enter the number of resources: 3
Enter details for P0
Enter allocation -- 0 1 0
Enter Request -- 0 0 0
Enter details for P1
Enter allocation -- 2 0 0
Enter Request -- 2 0 2
Enter details for P2
Enter allocation -- 3 0 3
Enter Request -- 0 0 0
Enter details for P3
Enter allocation -- 2 1 1
Enter Request -- 1 0 0
Enter details for P4
Enter allocation -- 0 0 2
Enter Request -- 0 0 2
Enter Available Resources -- 0 0 0
System is not in a deadlock state.
Safe sequence is: P0 P2 P3 P4 P1
```

Rm
3/7/24

② Contigious Memory Allocation.

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 25
void firstFit(int nb, int nf, int b[], int f[])
{
    int ff[MAX]={0};
    int allocated[MAX]={0};
    for(int i=0;i<nf;i++)
    {
        ff[i]=-1;
        for(int j=0;j<nb;j++)
        {
            if(allocated[j]==0 && b[j]>=f[i])
            {
                ff[i]=j;
                allocated[j]=1;
                break;
            }
        }
    }
    printf("\nFile_no:\tFile_size:\tBlock_no:\t
    Block_size:");
    for(int i=0;i<nf;i++)
    {
        if(ff[i]!=-1)
            printf("\n%d\t\t%d\t\t%d\t\t%d",
            i+1,f[i],ff[i]+1,b[ff[i]]);
        else
            printf("\n%d\t\t%d\t\t-\t\t-",i+1,
            f[i]);
    }
}
void bestfit(int nb, int nf, int b[], int f[])
{
    int ff[MAX]={0};
    int allocated[MAX]={0};
    for(int i=0;i<nf;i++)
    {
        int best=-1;
        ff[i]=-1;
        for(int j=0;j<nb;j++)
        {
            if(allocated[j]==0 && b[j]>=f[i])
            {
                if(best==-1 || b[j]<b[best])
                    best=j;
            }
        }
        if(best!=-1)
        {
            ff[i]=best;
            allocated[best]=1;
        }
```

```
printf("\nFile_no:\tFile_size:\tBlock_no:\t
Block_size:");
for(int i=0; i<nf; i++)
{
    if(ff[i]!=-1)
        printf("\n%d\t\t%d\t\t%d\t\t%d", i+1,
        f[i], ff[i]+1, b[ff[i]]);
    else printf("\n%d\t\t%d\t\t-\t\t", i+1,
        f[i]);
}
}
void worstFit(int nb, int nf, int b[], int f[])
{
    int ff[MAX]={0};
    int allocated[MAX]={0};
    for(int i=0; i<nf; i++)
    {
        int worst=-1;
        ff[i]=-1;
        for(int j=0; j<nb; j++)
        {
            if(allocated[j]==0 && b[j]>=f[i])
                if(worst==-1 || b[j]>b[worst])
                    worst=j;
        }
        if(worst!=-1)
        { ff[i]=worst; allocated[worst]=1; }
    }
    printf("\nFile_no:\tFile_size:\tBlock_no:\t
    Block_size:");
    for(int i=0; i<nf; i++)
    {
        if(ff[i]!=-1)
            printf("\n%d\t\t%d\t\t%d\t\t%d",
            i+1, f[i], ff[i]+1, b[ff[i]]);
        else
            printf("\n%d\t\t%d\t\t-\t\t", i+1,
            f[i]);
    }
}
int main()
{
    int nb, nf, choice;
    printf("Memory Management Scheme");
    printf("\nEnter the number of blocks:");
    scanf("%d", &nb);
    printf("Enter the number of files:");
    scanf("%d", &nf);
```

```
int b[nb], f[nf];
printf("\nEnter the size of the blocks:\n");
for(int i=0; i<nb; i++)
{
    printf("Block %d:", i+1);
    scanf("%d", &b[i]);
}
printf("Enter the size of the files:\n");
for(int i=0; i<nf; i++)
{
    printf("File %d:", i+1);
    scanf("%d", &f[i]);
}
while(1)
{
    printf("\n1. First Fit \n 2. Best Fit \n3. Worst
    Fit \n 4. Exit\n");
    printf("Enter your choice:");
    scanf("%d", &choice);
    switch(choice)
    {
        case 1:
            printf("\n\t Memory Management
            Scheme - First Fit\n");
            firstFit(nb, nf, b, f);
            break;
        case 2:
            printf("\n\t Memory Management
            Scheme - Best Fit\n");
            bestFit(nb, nf, b, f);
            break;
        case 3:
            printf("\n\t Memory Management
            Scheme - Worst Fit\n");
            worstFit(nb, nf, b, f);
            break;
        case 4:
            printf("\nExiting...\n");
            exit(0);
            break;
        default: printf("\nInvalid choice");
                 break;
    }
}
return 0;
}
```

Output:

Memory Management Scheme
Enter the number of blocks: 3
Enter the number of files: 2
Enter the size of the blocks:
Block 1: 5
Block 2: 2
Block 3: 7
Enter the size of the files:
File 1: 1
File 2: 4

1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 1

Memory Management Scheme - First Fit

File_no:	File_size:	Block_no:	Block_size:
1	1	1	5
2	4	3	7

1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 2

Memory Management Scheme - Best Fit

File_no:	File_size:	Block_no:	Block_size:
1	1	2	2
2	4	1	5

1. First Fit
2. Best Fit
3. Worst Fit
4. Exit

3/7/24

Enter your choice: 3

Memory Management Scheme - Worst Fit

File_no:	File_size:	Block_no:	Block_size:
1	1	3	7
2	4	1	5

1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 4

10/7/24

LAB-05 : Page replacement.

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

#define MAX_FRAMES 100
#define MAX_PAGES 100

int x=0;
void printFrames(int frames[], int framesCount, bool fault)
{
    for(int i=0; i<framesCount ; i++)
    {
        if(frames[i]== -1)
            printf("  ");
        else
            printf("%d ", frames[i]);
    }
    if(fault)
        printf(" - PF %d", ++x);
    else printf("  ");
    printf("\n");
}
int isPageInFrames(int page, int frames[], int framesCount)
{
    for(int i=0; i<framesCount; i++)
        if(frames[i]== page)
            return 1;
    return 0;
}
int getOptimal(int pages[], int currentIndex, int frames[],
    int framesCount, int pagesCount)
{
    int farthest = currentIndex;
    int index = -1;
    for(int i=0; i<framesCount ; i++)
    {
        int j;
        for(j= currentIndex; j< pagesCount ; j++)
            if(frames[i]== pages[j])
            {
                if(j> farthest)
                {
                    farthest = j; index = i;
                }
                break;
            }
        if(j== pagesCount)
            return i;
    }
    return index == -1 ? 0: index;
}
```

```
void fifo(int pages[], int pagesCount, int framesCount)
{
    x=0;
    printf("FIFO Page Replacement Algorithm\n");
    int frames[MAX_FRAMES], currentFrame = 0,
    pageFaults = 0;
    for (int i=0; i< framesCount; i++)
        frames[i] = -1;
    for (int i = 0; i< pagesCount; i++)
    {
        bool fault = false;
        if (! isPageInFrames (pages[i], frames,
            framesCount))
        {
            frames [currentFrame] = pages[i];
            currentFrame = (currentFrame+1)%
                framesCount;
            fault = true; pageFaults++;
        }
        printFrames (frames, framesCount, fault);
    }
    printf("Total Page Faults : %d \n\n", pageFaults);
}
void optimal (int pages[], int pagesCount, int framesCount)
{
    x = 0;
    printf("Optimal Page Replacement Algorithm \n");
    int frames [MAX_FRAMES], pageFaults = 0;
    for (int i=0; i< framesCount; i++)
        frames [i] = -1;
    for (int i=0; i< pagesCount; i++)
    {
        bool fault = false;
        if (! isPageInFrames (pages [i], frames, framesCount))
        {
            if (frames[i % framesCount] == -1)
                frames [i % framesCount] = pages[i];
            else
            {
                int index = getOptimal (pages, i+1, frames,
                framesCount, pagesCount);
                frames[index] = pages[i];
            }
            fault = true;
            pageFaults ++;
        }
        printFrames (frames, framesCount, fault);
    }
    printf("Total Page Faults : %d \n\n", pageFaults);
}
```

```
void lru(int pages[], int pagesCount, int framesCount)
{
    x=0;
    printf("LRU Page Replacement Algorithm\n");
    int frames[MAX_FRAMES], pageFaults=0;
    recent[MAX_FRAMES];
    for(int i=0; i<framesCount; i++)
    {
        frames[i]=-1;
        recent[i]=-1;
    }
    for(int i=0; i<pagesCount; i++)
    {
        bool fault=false;
        if(!isPageInFrames(pages[i], frames, framesCount))
        {
            int lruIndex=0;
            for(int j=1; j<framesCount; j++)
                if(recent[j]<recent[lruIndex])
                    lruIndex=j;
            frames[lruIndex]=pages[i];
            fault=true; pageFaults++;
        }
        for(int j=0; j<framesCount; j++)
            if(frames[j]==pages[i])
                recent[j]=i;
    printFrames(frames, framesCount, fault);
    }
    printf("Total Page Faults: %d\n\n", pageFaults);
}
int main()
{
    int pages[MAX_PAGES], pagesCount, framesCount;
    printf("Enter number of frames:");
    scanf("%d", &framesCount);
    printf("Enter number of pages:");
    scanf("%d", &pagesCount);
    printf("Enter the page reference string:");
    for(int i=0; i<pagesCount; i++)
        scanf("%d", &pages[i]);

    fifo(pages, pagescount, framesCount);
    optimal(pages, pagesCount, framesCount);
    lru(pages, pagesCount, framesCount);
    return 0;
}
```

Output:

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

FIFO Page Replacement Algorithm.

7 - page fault 1 (PF)
7 0 - PF 2
7 0 1 - PF 3
2 0 1 - PF 4
2 0 1
2 3 1 - PF 5
2 3 0 - PF 6
4 3 0 - PF 7
4 2 0 - PF 8
4 2 3 - PF 9
0 2 3 - PF 10
0 2 3
0 2 3
0 1 3 - PF 11
0 1 2 - PF 12
0 1 2
0 1 2
7 1 2 - PF 13
7 0 2 - PF 14
7 0 1 - PF 15

Total Page Faults: 15

Optimal Page Replacement Algorithm

7 - PF 1
7 0 - PF 2
7 0 1 - PF 3
2 0 1 - PF 4
2 0 1
2 0 3 - PF 5
2 0 3
2 4 3 - PF 6
2 4 3
2 4 3
2 0 3 - PF 7
2 0 3
2 0 3
2 0 1 - PF 8
2 0 1
2 0 1
7 0 1 - PF 9
7 0 1
7 0 1

Total Page Faults: 9

LRU Page Replacement Algorithm.

7 - PF 1
7 0 - PF 2
7 0 1 - PF 3
2 0 1 - PF 4
2 0 1
2 0 3 - PF 5
2 0 3
4 0 3 - PF 6
4 0 2 - PF 7
4 3 2 - PF 8
0 3 2 - PF 9
0 3 2
0 3 2
1 3 2 - PF 10
1 3 2
1 0 2 - PF 11
1 0 2
1 0 7 - PF 12
1 0 7
1 0 7

Total Page Faults : 12

10/7/24