Teacher's Sign. Remark

LAB-1

Date 8/5/2024
Page No. 1

FCFS

①

```
#include <stdio.h>
#include <stdlib.h>
void sort (int at[], int pn[], int bt[], int n)
  { int temp;
  for(int i=0; i<n ; i++)
   { for (int j=0; j<n-1 -i ;j++)
    { if (at[j] > at[j+1])
     { temp = at[j];
       at[j] = at[j+1];
       at[j+1] = temp;

       temp = pn[j];
       pn[j] = pn[j+1];
       pn[j+1] = temp;

       temp = bt[j];
       bt[j] = bt[j+1];
       bt[j+1] = temp;
     }
   }
  }
}
int main()
{ int n;
 printf("Enter the number of processes: ");
 scanf("%d", &n);
 int pn[n], bt[n], at[n];
 printf("Enter the process details:\n");
 for (int i=0; i<n; i++)
 { printf("Process %d:\n", i+1);
```

```
printf("Arrival time : ");
scanf("%d", &at[i]);
printf("Burst time : ");
scanf("%d", &bt[i]);
pn[i] = i+1;
}
sort(at, pn, bt, n);
int ct[n], tat[n], wt[n];
    ct[0] = bt[0];
    tat[0] = ct[0] - at[0];
    wt[0] = tat[0] - bt[0];
    for(int i=1; i<n; i++)
    { ct[i] = ct[i-1] + bt[i];
      tat[i] = ct[i] - at[i];
      wt[i] = tat[i] - bt[i];
    }
double avgtat=0.0, avgwt=0.0;
  for(int i=0; i<n; i++)
  { avgtat += tat[i];
    avgwt += wt[i];
  }
    avgtat /= n;
    avgwt /= n;
  printf("\n Process\t Arrival time\t Burst Time\t Completion Time
          \t at \t wt \n");
  for(int i=0; i<n; i++)
{ printf("%d\t %d \t %d \t %d \t %d \t %d \n", pn[i],
    at[i], bt[i], ct[i], tat[i], wt[i]); }
```

```
printf("\n
printf("
return 0;
}
```

Output:

Enter the pro
Enter the pro
Process 1 :
Arival Time
Burst time :
Process 2:
Arrival Tim
Burst Time
Process 3 :
Arrival Ti
Burst Time

Process

1
2
3

Average
Average

printf ("\n Average Turnaround time : %.0.2f\n", avgtat);
printf (" Average waiting time : %.0.2f\n", avgwt);
return 0;
}

Output:

Enter the number of process : 3
Enter the process details :

Process 1 :
Arival Time : 0
Burst time : 24

Process 2:
Arrival Time: 0
Burst Time : 3

Process 3 :
Arrival Time : 0
Burst Time : 3.

Process	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time
1	0	24	24	24	0
2	0	3	27	27	24
3	0	3	30	30	27

Average Turnaround Time : 27.00
Average Waiting Time : 17.00.

8/5/24

LAB-2

```
#include <stdio.h>
struct P {
 int id;
 int at;
 int bt;
 int ct;
 int tt;
 int wt;
};
void sort (struct P p[], int n);
void sjf (struct P p[], int n);
int main()
{ int n;
  int total_t = 0; int total_wt = 0;
  printf("Enter the number of processes: ");
  scanf("%d", &n);
 struct p p[n];
 printf("Enter the arrival time and burst time for each process:\n")
  for (int i=0; i<n; i++)
  { printf("Process %d\n", i+1);
    p[i].id = i+1;
    printf("Arrival Time: ");
    scanf("%d", &p[i].at);
    printf("Burst Time: ");
    scanf("%d", &p[i].bt);
  }
sort (p, n);
sjf (p, n);
```

```
printf("\n Process schedule :\n");
printf("Process ID \t Arrival Time \t Burst Time \t completion time \t
turnaround time\n");
for (int i=0; i<n; i++)
{ printf("%d \t %d \t %d \t %d \t %d \t", p[i].id, p[i].at,
  p[i].ct, p[i].tt, p[i].wt);
    total_tat += p[i].tt;
    total_wt += p[i].wt;
}
printf("\n Avg TAT : %.2f", (float)total_tat/n);
printf("\n Avg wt : %.2f", (float) total_tat/n);
return 0;
}
void sort (struct P p[]; int n) {
  for (int P=0; i<n-1; i++)
  { for (int j=0; j<n-i-1; j++) {
 if (p[j].at > p[j+1].at || (p[j].at == p[j+1].at &&
  p[j].bt > p[j+1].bt)) {
      struct ptemp = p[j]
        p[j] = p[j+1];
    } } p[j+1] = temp;
  }
}
void sjf (struct P p[], int n)
 {int currentime = 0;
 for (int i=0; i<n; i++)
  { int sj-index = i;
    for (int j = i+1; j<n && p[j].at <= current_time; j++)
   { if (p[j].bt < p[sj-index].bt)
    { sj-index = j; }
  }
```

```
P[sj-index].ct = current time + p[sj-index].bt;
P[sj-index].tt = p[sj-index].ct + p[sj-index].at;
P[sj-index].wt = P[sj-index].tt - p[sj-index].bt;
current time = p[sj-index].ct;
struct temp = p[i];
P[i] = P[sj.index];
P[sj.index] = temp;
}
}
```

Output:

Process 1:
Arrival time: 2
Burst time: 1

Process 2:
Arrival time: 1
Burst time: 5

Process 3:
Arrival time: 4
Burst time: 1

Process 4:
Arrival time: 0
Burst time: 6

Process 5:
Arrival time: 2
Burst time: 3

P_id	Arrival time	Burst time	Comptime	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

Avg TAT:- 7.80
Avg WT:- 4.60

ROUND ROBIN

```
#include<stdi
void sort(int
{ int min =
for(int i
{ min =
for(in
{ if
{ te
at[i
at[j
tc
bt
bt
te
b
b
tu
pr
pr
}
}
}
}
void mo
{int n,
printf("
scanf("
printf
scanf("
int proc_i
m[n];
```

LAB-8

Date / /
Page No.

ROUND ROBIN SCHEDULING

```
#include<stdio.h>
void sort(int proc_id[], int at[], int bt[], int b[], int n)
{ int min = at[0], temp = 0;
  for(int i=0; i<n; i++)
  { min = at[i];
    for(int j=1; j<n; j++)
    { if(at[j]<min)
      { temp = at[i];
        at[i] = at[j];
        at[j] = temp;
        temp = bt[j];
        bt[j] = bt[i];
        bt[i] = temp;
        temp = b[j];
        b[j] = b[i];
        b[i] = temp;
        temp = proc_id[i];
        proc_id[i] = proc_id[j];
        proc_id[j] = temp;
      }
    }
  }
}
void main()
{ int n, c=0, t=0;
  printf("Enter number of processes:");
  scanf("%d", &n);
  printf("Enter Time Quantum:");
  scanf("%d", &t);
  int proc_id[n]; at[n], bt[n], ct[n], tat[n], wt[n]; b[n], st[n],
  m[n];
```

TAT	WT
6	0
5	4
4	3
9	6
15	10

```
int f=-1, r=-1;
int q[100];
int count=0;
double avg_tat=0.0, tat=0.0, avg_wt=0.0, twt=0.0;
for(int i=0; i<n; i++)
{ proc_id[i]=i+1;
printf("Enter arrival time:\n");
for(int i=0; i<n; i++)
scanf("%d", &at[i]);
printf("Enter burst time:\n");
for(int i=0; i<n; i++)
{ scanf("%d", &bt[i]);
b[i]=bt[i];
m[i]=0;
rt[i]=-1;
}
sort(proc_id, at, bt, b, n);
f=r=0;
q[0]=proc_id[0];
int p=0, i=0;
while(f>=0)
{ p=q[f++];
i=0;
while(p!=proc_id[i])
i++;
if(b[i]>=t)
{ if(rt[i]==-1)
rt[i]=c;
b[i]=b[i]-t;
c+=t;
m[i]=1;
}
```

```
else
{ if(rt[
rt[i]
c+=
b[i]
m[i]
}
m[0]=1;
for(int j
{ if(at
{ q
}
if(b[i
{ cou
ct[i
}
else { q
if(f>
f=-
}
for(in
{ ta
rt[i
}
for(int
wt[i
printf
printf
for(i
{ prin
proc
```

```
else
{ if (rt[i]==-1)
    rt[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;
}
for(int i=0; i<n; i++)
{ tat[i]=ct[i]-at[i];
  wt[i]=rt[i]-at[i];
}
for(int i=0; i<n; i++)
  wt[i]=tat[i]-bt[i];
printf("\n RRS scheduling:\n");
printf("PID \tAT \tBT \tCT \tTAT \tWT \tRT \n");
for(int i=0; i<n; i++)
{ printf("%d \t%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], rt[i]);
```

```
for (int i=0; i<n; i++)
{ tot += wt[i];
  ttat += tat[i];
}
avg_tat = ttat/(double) n;
avg_wt = twt/(double) n;
printf("\n Avg turnaround time : %.4fms\n", avg_tat);
printf("\n Avg waiting time: %.4fms\n", avg_wt);
}
```

Output:

Enter no of process : 5
Enter time quantum: 2
Enter arrival time : 0, 1, 2, 3, 4
Enter burst time : 5, 3, 1, 2, 3.

P	AT	BT	ST	CT	TAT	WT	RT
1	0	5	0	13	13	8	0
2	1	3	2	12	11	8	1
3	2	1	4	5	3	2	2
4	3	2	7	9	6	4	4
5	4	3	9	14	10	7	5

Avg tat = 8.60
Avg wt = 5.80
Avg responsetime = 2.40.

PRIORITY
#include
void sort
{ int n
for (in
{ ni
for
{
{ tem
at[i
at[j
tem
bt[j
bt[i
tem
b[j
b[i
tem
P[j]
P[i]
temp
proc-i
proc-
}
}
}
void ma
{ int n
int c
printf(
scanf(
int proc
at[n]

PRIORITY (PE)

Date / /
Page No.

```
# 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 = at[i];
    at[i] = at[j];
    at[j] = temp;
    temp = bt[j];
    bt[j] = bt[i];
    bt[i] = temp;
    temp = b[j];
    b[j] = b[i];
    b[i] = temp;
    temp = p[j];
    p[j] = p[i];
    p[i] = temp;
    temp = proc_id[i];
    proc_id[i] = proc_id[j];
    proc_id[j] = temp;
  }
}
}
void main()
{ int n;
  int c=0;
  printf("Enter number of processes: ");
  scanf("%d", &n);
  int proc_id[n], at[n], bt[n], bt[n], tat[n], wt[n], m[n], b[n],
  st[n], p[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 priorities:\n");
for(int i=0; i<n; i++)
  scanf("%d", &p[i]);
printf("Enter arrival time:\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);
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];
    }
    if(count==n)
        break;
}
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("Priority Scheduling:\n");
printf("PID\tPrior\tAT\tBT\tCT\tTAT\tWT\tRT\n");
for(int i=0; i<n; i++)
{
    printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n",
    proc_id[i], p[i], at[i], bt[i], ct[i], tat[i], wt[i], st[i]);
for(int i=0; i<n; i++)
{
    ttat+=tat[i];
    twt+=wt[i];
}
avg_tat=ttat/(double)n;
avg_wt=twt/(double)n;
printf("Avg tat\n:%lfms\n", avg_tat);
printf("Avg wt:%lfms\n", avg_wt);
}
```

Output:

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

Priority scheduling (Pre-emptive)

PID	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 turnaround time : 5.5 ms.

Avg waiting time : 2.5 ms

NON- PRE

```
#include
void sort
{ int mi
  for (int
  { mi
  for (int
  { if
  {
  }
}
void ma
{ int n,
  printf(
  scanf(
  int proc
  double
  for (int
  { proc
```

NON- PRE EMPTIVE PRIORITY

Date / /
Page No.

```
#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[j];
        bt[j] = bt[i];
        bt[i] = temp;
        temp = p[j];
        p[j] = p[i];
        p[i] = 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 process:");
scanf("%d", &n);
int proc_id[n], at[n], bt[n], ct[n], tat[n], wt[n], m[n], st[n], p[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 priorities:\n");
for(int i=0; i<n; i++)
scanf("%d", &p[i]);
printf("Enter arrival time:\n");
for(int i=0; i<n; i++)
scanf("%d", &at[i]);
printf("Enter burst time:\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);
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 && 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
  { pr
    b
  }
x++;
if(co
  br
}
for(int i
  tat
for(in
  wt
printf
printf
printf
}
```

Output:

```
Enter nu
Enter pri
10
20
30
40
Enter arriv
0
1
2
4
Enter burst
5
4
2
1
```

```
while (x>=1 && m[--x] != 1)
  { priority = p[x];
    break;
  }
x++;
if (count == n)
   break;
}
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("\n Priority scheduling: \n");
printf("Avg tat: %.1f ms\n", avg_tat);
printf("Avg wt: %.1f ms\n", avg_wt);
}
```

Output:

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

Avg turnaround time : 6 ms

Avg waiting time : 3 ms

21/5/24

MULTI LEVEL QUEUE SCHEDULING

Date / /
Page No.

```
#include <stdio.h>
void sort (int proc_id[], int at[], int bt[], int n)
{ int min, temp;
for (int i=0; i<n-1; i++)
 { for (int j= i+1; j<n; j++)
 { if (at[j] < at[i])
   { 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 simulate FCFS (int proc_id[], int at[], int bt[], int n,
  int start_time)
 { int c = start_time, ct[n], tat[n], wt[n];
   double ttat = 0.0, twt = 0.0;
// completion time
   for (int i=0; i<n; i++)
   { if (c >= at[i])
       c += bt[i];
     else
       c = at[i] + bt[i];
     ct[i] = c;
   }
```

```
// 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("PID\t AT\t BT\t CT\t TAT\t WT\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]);
    ttat += tat[i];
    twt += wt[i];
}
printf("Avg TAT : %.2f ms\n", ttat/n);
  printf("Avg WT : %.2f ms\n", twt/n);
  }
  void main()
  { int n;
  printf("Enter number of processes : ");
  scanf("%d", &n);
int proc_id[n], at[n], bt[n], type[n];
int sys_proc_id[n], sys_at[n], sys_bt[n] user_proc_id[n];
user_at[n], user_bt[n];
int sys_count = 0; user_count = 0;
for(int i=0; i<n; i++)
  { proc_id[i] = i+1;
  printf("Enter arrival time, burst time and type (0 for
  system, 1 for user) for process %d: ", i+1);
  scanf("%d %d %d", &at[i], &bt[i], &type[i]);
if(type[i] == 0)
```

```
{ sys_proc_id[sys_count] = proc_id[i];
  sys_at[sys_count] = at[i];
  sys_bt[sys_count] = bt[i];
} else
{ user_proc_id[user_count] = proc_id[i];
  user_at[user_count] = at[i];
  user_bt[user_count] = bt[i];
  user_count++;
}}
sort(sys_proc_id, sys_at, sys_bt, sys_count);
sort(user_proc_id, user_at, user_bt, user_count);
printf("System Processes scheduling:\n");
simulateFCFS(sys_proc_id, sys_at, sys_bt, sys_count, 0);
int system_end_time = 0.
  if(sys_count > 0)
  { system_end_time = sys_at[sys_count-1] + sys_bt
      [sys_count-1];
    for(int i=0; i< sys_count-1; i++)
      { if(sys_at[i+1] > system_end_time)
        { system_end_time = sys_at[i+1]; }
      system_end_time = sys_at[i+1];
      } system_end_time += sys_bt[i]; } }
printf("\n User Processes scheduling:\n");
  simulateFCFS(user_proc_id, user_at, user_bt, user_count,
  system_end_time);
}
```

Output:

```
Enter number of processes: 4
Enter AT, BT and type for process: 0 2 0
0 1 1
0 5 0
0 3 1
```

System Process Scheduling:

PID	AT	BT	CT	TAT	WT
1	0	2	2	2	0
3	0	5	7	7	2

Avg TAT : 4.50ms
Avg WT : 1.00MS

User Process Scheduling:

PID	AT	BT	CT	TAT	WT
2	0	1	8	8	7
4	0	3	11	11	8

Avg TAT : 9.50ms
Avg WT : 7.50ms.

Result
5/6/24

RATEMONOTONIC

Date / /
Page No.

```
#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[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 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]);
  for(int i=0; i<n; i++)
  proc[i] = i+1;
  sort(proc, b, pt, n);
  int I = lcmul(pt, n);
  printf("LCM = %d\n", I);
  printf("\n Rate Monotone scheduling:\n");
  printf("PID\tBurst\tPeriod\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%f <= %f is %s\n", sum, rhs, (sum<=rhs)?
  "true":"false");
  if(sum > rhs)
  exit(0);
```

```
printf("Sched
  int time=0
  while(ti
{ int f=0;
  for(int i=
  { if(tim
    rem
    if(rem
  { if(pre
    { print
      proc[i
      prev
    }
    rem
    f=1;
    break;
    x=0;
  }}
  if(!f)
  { if(x!=
  { printf(
    x=1;
  }
}
  time++;
}
}
```

```
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%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++;
}
}
```

Date / /
Page No.

Output:

Enter the number of processes: 2
Enter the CPU burst time:
20
35
Enter the time period:
50
100
LCM = 100
Rate Monotonic scheduling:

PID	BT	Period
1	20	50
2	30	100

0.750000 <= 0.828427 => true.

Scheduling occurs for 100 ms.

0ms: process 1 running
20ms: Process 2 running
50ms: process 1 running
70ms: Process 2 running.
75ms: CPU is idle.

5/6/24

#include <st
#include <
#include <m
void sort (int
{ int temp =
for (int i=0;
{ for (int j=
{ if (d[j] <d[i
{ temp = d[j];
d[j] = d[i];
d[i] = temp;
temp = pt[
pt[i] = pt[j]
pt[j] = tem
temp = b[j]
b[j] = b[i];
b[i] = temp;
temp = proc
proc[i] = pr
proc[j] = t
}}}}
int gcd (int a,
{ int r;
while (b>0)
{ r = a%b;
a = b;
b = r;
}
return a;
int lcmul (i
{ int lcm = p
for (int i=1;
{ lcm = (lcm

EARLIEST DEADLINE FIRST

Date / /
Page No.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void sort (int proc[], int d[], int b[], int pt[], int n)
{ int temp=0;
for(int i=0;i<n;i++)
{ for (int j=i+1; j<n ;j++)
{ if (d[j] <d[i])
{ 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 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],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];
  }
 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 I=lcmcal(pt,n);
 printf("\n Earliest 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",I);
int time=0; int prev=0,x=0;
int nextDeadlines[n];
for(int i=0;i<n;i++)
{ nextDeadlines[i]=d[i];
```

```
 rem[i]=b
}
while(time
{ for(int i=0
 { if(time!
 { nextDead
   rem[i]=
 }
}
 int minDea
  int task
  for(int i=0
   { if(rem
   { minDea
     taskTo
   }
 }
 if(taskTo
 { printf("
  rem[task
  }else
 { printf("
  } time++
}}
```

Output:

```
Enter numb
Enter arri
 5 1 4
Enter bur
 3 2 1
```

```
    rem[i] = b[i];
}
while (time < I)
{ 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 = I + 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: Task %d is running \n", time, proc[taskToExecute]);
    rem[taskToExecute]--;
  } else
  { printf ("%d: CPU is idle \n", time);
  } time++;
}}
```

Output:

Enter number of system processes: 3

Enter arrival time 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 time of the user processes:

2 6 3

Enter burst time 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

5/6/24

SJF (Preemptive)

Date / /
Page No.

```
#include <stdio.h>
#include <limits.h>
int main()
{ int n;
  printf("Enter the number of processes: ");
  scanf("%d", &n);
  int pid[n], arrival[n], burst[n], remaining[n], completion[n],
  waiting[n], tat[n];
  float avg_wt=0, avg_tat=0;
for (int i=0; i<n; i++)
  { pid[i]= i+1;
  printf("Enter arrival time and burst time for process %d: ", i+1);
  scanf("%d %d", &arrival[i], &burst[i]);
  remaining[i]= burst[i];
} int completed=0, ct=0, shortest=0;
  int min_rem_time = INT_MAX;
  int finish_time;
  int check=0;

  if (check==0)
  { current_time++;
    continue; }
  remaining[shortest]--;
  min_rem_time = remaining[shortest];

  if (min_rem_time==0) {
     min_rem_time = INT_MAX;
  } if(rem_time==0)
```

```
if (remaining[shortest] == 0)
{ completed++;
  check=0;
  finish_time = current_time+1;
  completion[shortest] = finish_time;
  tat[shortest] = finish_time - arrival[shortest];
  waiting[shortest] = tat[shortest] - burst[shortest];
  avg_wt += waiting[shortest];
  avg_tat += tat[shortest];
}
}
  avg_wt /= n;
  avg_tat /= n;
printf("\n PID\t\tAT\t\tBT\t\tTAT\t\tWT\n");
for(int i=0; i<n; i++)
{ printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
  pid[i], arrival[i], burst[i], completion[i], tat[i], wt[i]);
}
  printf("\n Avg_WT : %.2f", avg_wt);
  printf("\n Avg_TAT : %.2f", avg_tat);
return 0;
}
```

Output:

Date / /
Page No.

Enter the number of processes: 4
Enter at and bt:

0 6
0 8
0 7
0 3

PID	AT	BT	CT	TAT	WT
1	0	6	9	9	3
2	0	8	24	24	16
3	0	7	16	16	9
4	0	3	3	3	0

Avg wt: 7.00 ms
Avg tat: 13.00 ms.

5/6/24

PROPORTIONALITY

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
  typedef struct
{  char name[5];
   int tickets;
}process;
int main()
{ int n; int total_tickets=0;
   float total_T=0.0;
   printf("Enter the number of processes:");
  scanf("%d",&n);
  Process p[n];
  srand(time(NULL));
  for(int i=0;i<n;i++)
{ printf("\n Process %d:\i+1);
  sprintf(p[i].name,"P.%d",i+1);
  printf("Tickets:");
  scanf("%d",&p[i].tickets);
  total_tickets+= p[i].tickets;
  total_T+=p[i].tickets;
}
printf("\n--Proportional Share scheduling--\n");
printf("Enter the Time Period for scheduling:");
int m;
scanf("%d",&m);
for(int i=0;i<m;i++)
{ int winning_ticket=rand()%.total_ticket+1;
  int accumulated_tickets=0;
  int winner_index;
```

```
for(int j=0;
{ accumul
  if(current
    { winne
      brea
    }
}
printf("Ti
p[winner_i
}
for(int i=0
{ printf("
  p[i].nam
 }return
}
```

Output:

Enter no of

Enter ticket

10 20 30

The winning

The winnin

The winnin

The winning

Probabilities

The probab

The probabi

The probabi

The probabi

```
for (int j=0 ; j<n ; j++)
{ accumulated_tickets += p[j].tickets;
  if (winning_tickets <= accumulated_tickets)
    { winner_index = j;
      break;
    }
}
printf("Tickets picked: %d winner, %s\n", winning_ticket,
p[winner_index].name);
}
for (int i=0; i<n; i++)
{ printf("\n The process: %s gets %0.2f%% of process Time.\n",
  p[i].name, ((p[i].tickets / total_T) * 100));
 } return 0;
}
```

Output:

Enter no 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 : 0.

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%.

6/6/24

SEMAPHORES [PRODUCER AND CONSUMER]

```
#include<stdio.h>
#include<stdlib.h>
int mutex=1;
int full=0;
int empty=10; x=0;
void producer()
{ --mutex;
  ++full;
  --empty;
  x++;
printf("\n Producer produces item %d", x);
  ++mutex;
}
void consumer()
{ --mutex;
  --full;
  ++empty;
  printf("\n Consumer consumes item %d", x);
  x--;
++mutex;
}
int main()
{ int n, i;
printf("\n 1. Press 1 for producer"
       "\n 2. Press 2 for consumer"
       "\n 3. Press 3 for Exit");
for(i=1; i>0; i++)
{ printf("\n Enter your choice:");
  scanf("%d", &n);
```

```
switch(n)
{ case 1:
    else {
    } b
Case 2:
  if ((n
    { co
  else {
    brc
case 3: e
  }
}
}
```

Output:

1. Press 1 f
2. Press 2 f
3. Press 3 f

Enter your
Produce
Enter your
Consumer
Enter you
Produce
Enter you
Consumer
Enter you
Buffer is

CONSUMER()

```
switch(n)
{ case 1: if ((mutex == 1)
          && (empty != 0)){
          producer(); }
  else { printf("Buffer is full!");
  } break;

Case 2:
  if ((mutex == 1) && (full != 0))
  { consumer(); }
  else { printf("Buffer is empty"); }
  break;
case 3: exit(0);
        break;
}
}
}
```

Output:

1. Press 1 for Producer
2. Press 2 for Consumer
3. Press 3 for exit

Enter your choice: 1
Producer produces item 1
Enter your choice: 2
Consumer consumes item 1
Enter your choice: 1
Producer produces item 1
Enter your choice: 2
Consumer consumes item 1
Enter your choice: 2
Buffer is empty!

Rm 12/6/24

BANKERS 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 isSafe( int P, int R, int processes[], int avail[], int max[][R],
    int allot[ ][R])
{   int need[P][R] ;
  CalculateNeed (P, R, need, max, allot);
 bool 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)
{ bool found = false;
  for (int p=0; p <P; p++)
   { if ( finish[p] ==0)
     { int j;
      for (j=0 ; j<R; j++)
```

```
        if (need[p][j] > work[j])
            break;
        if (j == R)
        printf ("P%d is visited (", p);
        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 == false)
    { printf ("System is not in safe state\n");
      return false;
    }
}
printf ("System is in safe state\nThe Safe Sequence is --- (");
    for (int i=0; i<P; i++)
    { printf ("P%d", safeSeq[i]);
    }
    printf (")\n");
    return true;
}
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];
int max[P][R];
int 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]);
 }

isSafe (P, R, processes, avail, max, allot);
printf ("\n Process \t Allocation \t Max \t Need \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 =
  { pri
  }
printf(
 for (int
  { pri
  }
  print
} return 0
}
```

Output:

Enter nu
Enter nu
Enter de
Enter all
Enter n
Enter de
Enter al
Enter n
Enter de
Enter al
Enter n
Enter de
Enter
Enter
Enter de
Enter a
Enter

```
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;
}
```

Date / /
Page No.

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
Enter New Request Details --
Enter pid -- 1
Enter Request for Resources -- 1 0 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
P_0	0 1 0	7 5 3	7 4 3
P_1	2 0 0	3 2 2	1 2 2
P_2	3 0 2	9 0 2	6 0 0
P_3	2 1 1	2 2 2	0 1 1
P_4	0 0 2	4 3 3	4 3 1

Rm
20/6/24

DINING PHILOSOPHERS PROBLEM

Date / / Page No.

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

#define MAX_PH 100

sem_t mutex;
sem_t mutex2;
int philosophers [MAX_PH];

void *one_eat_at_a time (void *arg)
{ int philosopher = ((int)arg);
sem_wait (&mutex);
printf ("Philosopher %d is granted to eat \n", philosopher);
sleep (1);
printf ("Philosopher %d has finished eating \n", philosopher);
sem_post (&mutex);

return NULL;
}
void *two_eat_at_a_time, (void * arg)
{int philosopher = ((int)arg);
sem_wait (&mutex2);
printf ("Philosopher %d is granted to eat \n", philosopher);
sleep(1);
printf ("Philosopher %d has finished eating \n", philosopher);
sem_post (&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_phi[hungry_count];
  for(int i=0; i<hungry_count; i++)
  { printf("Enter philosopher %d position (1 to %d): ", i+1, N);
  scanf("%d", &hungry_phi[i]);
    hungry_phi[i]--;
  }
  pthread_t thread[hungry_count];
  int choice;
  sem_init(&mutex, 0, 1);
  sem_init(&mutex2, 0, 2);
  do{
  printf("\n1. One can eat at a time\n2. Two can eat at a time\n3. Exit\nEnter your choice: ");
  scanf("%d", &choice);
    { case 1: printf("\n\t1. Allow one philosopher to eat at any time\n");
      for(int i=0; i<hungry_count; ++)
      { philosophers[i] = hungry_phi[i];
        pthread_create(&thread[i], &philosophers[i]);
      }
```

Output

Enter the
How ma
Enter ph
Enter ph
Enter ph

```
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. Please try again\n");
 }
}
while(choice!=3);
sem_destroy(&mutex);
sem_destroy(&mutex2);
return 0;
}
```

Output

```
Enter the total number of philosophers: 5
How many are hungry: 3
Enter philosopher 1 position: 2
Enter philosopher 2 position: 4
Enter philosopher 3 position: 5
```

1. One eat at a time.
2. Two eat at a time
3. Exit.

Allow one philosopher to eat at a time.
P2 is granted to eat
P2 is ~~eating~~ finished eating
P4 is waiting
P5 is waiting
P5 is granted to eat
P5 is ~~waiting~~ finished eating
P4 is waiting
P4 is granted to eat
P4 is ~~waiting~~ finished eating

1. Enter your choice : 2
Allow 2 philosophers to eat at same time.
Combination 1.
P2 and P4 granted to eat
P2 and P4 finished eating
P5 waiting
P2 and P5 granted to eat.
P2 and P5 finished eating
P4 waiting.

20/6/24

Deadlock Detection

D: 31/7/24.
Pa_ No.

```
#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; j++)
  { 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 availability Resources --");
 for (int i=0 ; i<m; i++)
   { scanf("%d", &avail[i]);
   }
int finish[n], safeseq[n], work[m], flag, f=0;
 for (i=0 ; i<n ; i++)
       { finish[i]=0;
       }
```

```
for (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 (k=0; k<m; k++)
{ work[k] += alloc[i][k];
}
safeseq[count++] = i;
finish[i] = 1;
flag = 1;
}
}
}
```

```
else
{ safeseq
finish
flag =
}
}
}
if(flag ==
{ break;
}
}
int dead
for (i=0;
{ if (fin
{d
prin
br
}
}
if(deac
{prin
print
for(i
{ prin
}
print
}
retu
}
```

```
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[i] == 0){
          printf("P.%d", j);
         } }
     printf("\n");
      break;
   }
}
 if(deadlock == 0)
  {printf("\n System is not in a deadlock state.\n");
  printf("
  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 the 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
Safe Seq is: P0 P2 P3 P4 P1

Rm 3/7/24

3/7/24

```
#include <
#include <
#define MA
void firstFit
{ int ff[M
  int alloc
for (int i=0
{ ff[i] = 0
for (int j=
{ if (alloc
{ ff[i] =
  alloca
  break;
}
}
}
printf("\n File
for (int i
{ if (
print
b[
else
print
}
}
void bestfit(in
{ int ff[M
int alloc
for (int i=
{ int
```

3/1/24 CONTIGIOUS_MEMORY

Date / /
Page No.

```
#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++) // file
{ ff[i]=-1;
for(int j=0;j<nb;j++) // blocks
{ if(allocated[j]==0 && b[j]>=f[i])
{ ff[i]=j;
  allocated[j]=1;
  break;
}
}
}
printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_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\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("\n File no: \t File size : \t Block_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;
```

```
}}
printf("\n File No: H
  for (int i=0; i<n
  { if (ff[i] != -
    printf("\n %d %d
    b[ff[i]]);
  else
  { printf("\n %d %d
  }
}
int main()
{ int nb, nf, c
  printf("Mem
  printf("\n En
  scanf("%d",
  printf("Enter t
  scanf("%d",
  int b[nb], f[
  printf("\n En
  for (int i=0;
    printf("Bloc
    scanf("%
  }
  printf("Ente
  for (int i=0;
    { printf(
      scanf(
    }
  while(1)
{ printf("\n
  printf("
  scanf("%
```

```
}}
printf("\n File_No: \t File_size: \t Block_No: \t Block_size:");
for (int i=0; i<nf ; i++)
{ if (ff[i] != -1)
printf("\n %d \t\t %d \t\t %d", i+1, ff[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("\n Enter the number of blocks");
scanf("%d", &nb);
printf("Enter the number of files:");
scanf("%d", &nf);
int b[nb], f[nf];
printf("\n Enter 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("\n 1. First fit \n 2. Best fit \n 3. Worst Fit \n 4. Exit \n");
printf("Enter your choice:");
scanf("%d", &choice);
```

```
switch (choice)
{ case 1: printf("\n\t Memory Management Scheme-FirstFit\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 - WorstFit\n");
          worstFit (nb,nf,b,f);
          break;
  case 4: printf ("\n Exiting...\n");
          exit(0);
          break;
  default: printf ("\n Invalid choice.\n");
          break;
  }
}
return 0;
}
```

Output:

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

Memory management. (First fit)

File No	Filesize	BlockNo	Blocksize	Fragize
1	1	1	5	4
2	4	3	7	3

Memory management (Best-fit)

FileNo	Filesize	BlockNo	Blocksize	Fragize
1	1	2	2	1
2	4	1	5	1

Memory management (worst-fit)

Date / /
Page No.

FileNo	Filesize	BlockNo	Blocksize	Fragsize
1	1	3	7	6
2	4	1	5	1

Rm
3/7/24

PAGE REPLACEMENT

10/7/24

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

```
    pageFrames(frames, numFrames);
  }
  printf("\nTotal Page Faults: %d\n\n", PageFaults);
}
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<numPages; j++)  // Here it keeps iterating
   { if(frames[i] == Pages[j])             through all the pages
   { if(j > farthest)                      until it get the
      { farthest = j;                      page inside the frame
        Index = j;
      }
   break;
  }
 }
 if(j == numPages)
  { return i; } }
return (index == -1) ? 0 : index;
}
void optPageReplacement(int pages[], int numPages, int numFrames)
 { int frames[numFrames];
 int PageFaults = 0;
 for(int j=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(frames, numFrames, -1)) {
        for (int j=0; j<numFrames; j++)
            if (frames[j] == -1) {
                frames[j] = pages[i];
                break;
            }
    } else { int index = findOptimalReplacementIndex(pages, numPages,
        frames, numFrames, i+1);
        frames[index] = pages[i];
    }
    pageFaults++;
}
printFrames(frames, numFrames);
}
printf("\n Total Page Faults : %d\n\n", pageFaults);
}
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\tFrames\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;
  { if (timest
    { lruIndex
    }
  } frames[lru
    timestamp
    pageFault
  } else {
  for (int j=0
  { if (frames
    { timest
      break;
    }
  }
  pageFram
  }
printf("\n
}
int main()
{ int num
  printf("
  scanf("
  printf("
  scanf("
  int pages
  printf("
  for (int i=
    { scanf
  fifoPageR
  optPageR
  lruPageR
  return 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;
    }
  }
  pageFrames(frames, numFrames);
  }
  printf("\nTotal 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(pages, numpages, numFrames);
  return 0;
}
```

Output:

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

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

Total Page Faults : 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
2	2 0 3
1	2 0 1
2	2 0 1
0	2 0 1
1	7 0 1
7	7 0 1
0	
1	

Total Page Faults : 9.

LRU 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	4 0 3
4	4 0 2
2	4 3 2
3	
0	

Reference string	Frames
3	0 3 2
2	0 3 2
1	0 3 2
2	1 3 2
0	1 3 2
1	1 0 2
7	1 0 7
0	1 0 7
1	

Total Page Fault