INDEX

Name Pooja. Rajshekhar Arabi Sub. OS

Std.: IV th Sem CSE Div. 4C Roll No. 413

Telephone No. – E-mail ID. –

Blood Group. – Birth Day. –

Date -	Sr.No.	Title	Page No.	Sign./Remarks
8/6/24	1	FCFS and SJF	1	
10/6/24	2	Round Robin & priority	9	
5/6/24	3	multilevel Queue	15	
5/6/24	4)	Rate Monotonic, Earliest deadline first & proportional scheduling.	17	
12/6/24	5)	producer consumer	25	
19/6/24	6)	Dinning philosophers	30	
19/6/24	7)	Banker's Algorithms	27	
3/7/24	8)	Deadlock detection problem	33	
3/7/24	9)	Worst fit, Best fit, first fit	35	
10/7/24	10)	FIFO LRU, optimal	38	

All 10 programs completed

2024.07.17 11:21

8/5/2024 ①

Times in cpu scheduling.

1) Arrival Time → The time at which process entres the rediqu
2) Burst Time → The time require by a process to get executed on the cpu
3) completion time :- The time at which the process completes its execution
4) Turn around time :- completion time - Arrival time
5) waiting Time - TAT Brust time
6) Response Time :- The time at which theprocess gets the cpu for
7) the first time - Arrival time.

① FCFS (first come first serve) -

```
#include <stdLib.h>
# define MAX 10
void fcfs (int n, int at[], int bt[]) {
    int ct[MAX];
    int tat[MAX];
    int wt[MAX];
  int total_wt=0;
  int total_tat=0;
  int current_time;
  for (int i=0; i<n; i++){
      ct[i]=-1; }
  for (int i=0; i<n; i++) {
      if (current_time < at[i]){
          current_time = at[i]; }
      ct[i] = current_time + bt[i];
        current_time = ct[i];
      }
  for (int i=0 ; i<n ; i++) {
        tat[i] = ct[i] - at[i]
        total_wt += wt[i];
      }
  for (int i=0 ; i<n; i++){
      wt[i] = tat[i] - bt[i];
      total_wt += wt[i];
  }
```

```
for (int i=0; i<n; i++) {
    printf ("%d\t %d\t %d\t\t%d", i+1, at[i], bt[i], ct[i], tat[i],
    wt[i]); }
    printf ("\n Average waiting time : %.2f ", (float) total_wt/n);
    printf ("\n Average waiting tourned around time : %.2f ",
        (float) total_tat/n);
    }

int main () {
    int n, i;
    printf (" Enter no of process: ");
    scanf ("%d", &n);
    int at[n], bt[n];
    printf (" Enter the arrival time: \n");
    for (i=0; i<n; i++){
        scanf ("%d", &at[i]);
    }
    printf (" Enter the brust time : \n");
    for (i=0; i<n; i++) {
        scanf ("%d", &bt[i]);
    }
    fcfs (n, at, bt);
    return 0;
}
```

output -

Enter no of processes - 4

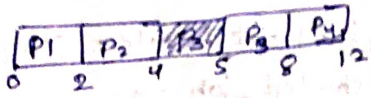
Enter the arrival time - 0
1
5
6

Enter the brust time 2
2
3
4

Process	Arrival time	Brust time	completion TIME	TAT TIME	waiting TIME
1	0	2	2	2	0
2	1	2	4	3	1
3	5	3	8	3	0
4	6	4	12	6	2

Average waiting time - 0.75

Average turnaround time - 3.50



Code for SJF - Preemtive

```
#include<stdlib.h>
#include<stdio.h>
#define MAX 10
void sjf_non_preemptive (int n, int at[], int bt[]){
    int ct[MAX];
    int at[MAX];
    int wt[MAX];
    int rt[MAX];
    int total_wt=0
    int total_tat=0;
    int completed=0;
    int current_time=0;
    int shortest_job=0;
    int Min_bt = 9999;
    int is_completed[MAX]={0};
    for (int i=0; i<n; i++){
        rt[i] = bt[i];
    }
    while (completed <n){
     for (int i=0; i<n; i++){
      if [at[i] <= current_time && rt[i]< Min_bt && ! is_
      completed [i]){
            shortest_job=i;
            Min_bt = rt[i];
        }
    }
    rt [shortest_Job]--;
    if (rt[shortest_job==0){
        completed++;
        Min_bt = 9999;
        is_completed [shortest_Job]=1;
        ct[shortest_Job] = current_time+1;
        tat[shortest_job] = ct[shortest_job]-at[shortest_
                                                job]
```

```
        total_tat += tat[shortest_job];
        wt[shortest_job] = tat[shortest_job] - bt[shortest_job];
        if (wt[shortest_job] < 0) wt[shortest_job] = 0;
            total_wt += wt[shortest_job];
        }
        current_time++;
    }
    printf("\n Process \t Arrival time \t brust time \t completion time \t
      Turnaround time \t waiting time \n");
    for (int i=0; i<n; i++){
        printf(i+1, at[i], bt[i], ct[i], tat[i], wt[i]);
        printf("\n Average waiting time: %.2f", (float) total_wt/n);
        printf("\n Average waiting time: %.2f", (float) total_wt/n);
    }
int main(){
    int n, i;
    printf("Enter the no of process");
    scanf("%d", &n);
    int at[n], bt[n];
    printf("Enter arrival time");
    for (i=0; i<n; i++){
        scanf("%d", &at[i]);
    }
    printf("Enter brust time");
    for (i=0; i<n; i++){
        scanf("%d", &bt[i]);
    }
    sjf_non_preemptive(n, at, bt);
    return 0;
}
```

O/P - Enter the arrival time - 0
0
0
0

Enter the brust time 6
8
7
3

Process	Arival time	Brust time	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

Average waiting time - 7.00 Average turnaround time - 13.00

```
#include<stdio.h>
#define MAX 10
void sjf_preemptive (int n, int at[], int bt[]) {
  int ct[MAX];
  int tat[MAX];
  int wt[MAX];
  int rt[MAX];
  int total_wt=0;
  int total_tat=0;
  int total_wt=0;
  int total_rt=0;
  int is_completed[MAX]={0};

for(int i=0; i<n; i++){
    rt[i]=bt[i];
}
while(completed<n){
  int shortest_job=-1;
  int min_bt=9999;
  for(int i=0; i<n; i++){
    if(at[i]<=current_time && rt[i]<min_bt && rt[i]>0){
    shortest_job=i;
    min_bt=rt[i];
      }
    }
  if(shortest_job==-1){
    current_time++;
      continue;
  }
  rt[shortest_job]--;
  if(rt[shortest_job]==0){
    completed++;
  ct[shortest_job]=current_time+1;
  tat[shortest_job]=ct[shortest_job]-at[shortest_job];
  total_tat+=tat[shortest_job];
  wt[shortest_job]=tat[shortest_job]-bt[shortest_job];
  if(wt[shortest_job<0)wt[shortest_job]=0;
    total_wt+=wt[shortest_job];
      is_completed[shortest_job]=1;
  }
```

```
        current_time++
    }
    printf("\n Process\t Arrival time\t Burst time\t Completion Time\t
        Turnaround Time\t waiting time\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", i+1,
        at[i], bt[i], ct[i], tat[i], wt[i]);
    }
    printf("\n Avg waiting time : %.2f", (float) total_wt/n);
    printf("\n Avg turaround time: %.2f", (float) total_tat/n);
}
printf("\n");
int main(){
    int n;
    printf("Enter the no of processor");
    scanf("%d", &n);
    int at[n], bt[n];
    printf("Enter the arrival time\n");
    for(i=0; i<n; i++){
        scanf("%d", &at[i]);
    }
    printf("Enter the burst time:\n");
    for(i=0; i<n; i++){
        scanf("%d", &bt[i]);
    }
    sjf_preemptive(n, at, bt);
    return 0;
}
```

O/P →

```
Enter the no of process: 4
Enter the arrival time:
0 0 0 0
Enter the burst time:
6 8 7 3
```

Process	AT	BT	CT	TOT	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 waiting Time - 7

Avg turnaround Time - 13

lab-1(b)
2.a)

```
#include<stdio.h>
#define MAX 10
void round_robin (int n, int bt[], int quantum) {
    int wt[MAX]={0};
    int tat[MAX]={0};
    int remaining_bt[MAX];
    int total_wt=0, total_tat=0;
    int time=0;
    for(int i=0; i<n; i++){
        remaining_bt[i]=bt[i];}
    while(1){
        int done=1;
        for(int i=0; i<n; i++){
            if(remaining_bt[i]>0){
                done=0;
                if(remaining_bt[i]>quantum){
                    time+=quantum;
                    remaining_bt[i]-=quantum;
                }else{
                    time+=remaining_bt[i];
                    wt[i]=time-bt[i];
                    remaining_bt[i]=0;
                }
            }
        }
        if(done==1)break;
    }
    for(int i=0; i<n; i++){
        tat[i]=bt[i]+wt[i]
        total_wt+=wt[i]
        total_tat+=tat[i];
    }
printf("\n process \n Brust time \t waiting time \t TAT \n");
    for(int i=0; i<n; i++){
        printf("%d %d %d %d", i+1, bt[i], wt[i], tat[i]);
    }
printf("Avg waiting time: %.2f", (float)total_wt/n);
printf("Avg turnaround time: %.2f", (float)total_wt/n);
```

2024.07.17 11:23

```
int main(){
  int n, quantum;
   printf("Enter no of process");
    scanf("%d", &n);
     int bt[max];
    printf("Enter Brust time for each process\n");
      for (int i=0; i<n; i++){
        printf("process %d", i+1);
          scanf("%d", &bt[i]);
      Printf("Enter size of time slice (quantum):");
      Scanf("%d", &quantum);
       round_robin (n, bt, quantum);
         return 0;
           }
```

o/p-

Enter no of process: 3

Enter Brust time:

Process 1: 24

Process 2: 3

Process 3: 3

Enter the size of time slice

3

process	Brust time	waiting time	TAT
1	24	6	30
2	3	3	6
3	3	6	9

Avg waiting time: 5.66

Avg TAT: 15.66

2b) Lab

void sa

int n =

for (i

2024.07.17 11:23

2b) Lab2 - Priority - Preemptive

```
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=0; 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, c=0;
    printf("Enter no of processes: ");
    scanf("%d", &n);
    int proc-id[n], at[n], bt[n], ct[n], tat[n], wt[n], rt[n], b[n],
    rt[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;
        rt[i] = 0;}
    printf("Enter priorities:\n");
    for(int i=0; i<n; i++){
        scanf("%d", &p[i]);
    printf("Enter arrival times:\n");
    for(i=0; i<n; i++){
        scanf("%d", &at[i]);
```

```
Printf ("Enter brust times:\n");
for (int i=0; i<n; i++) {
        scanf ("%d", &bt[i]);
        b[i] = bl[i];
        M[i] = -1;
        al[i] = -1;
    }
sort (Proc-id, p, at, bl, 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 (al[i] <= c && p[i] >= priority && b[i] > 0 && M[i]
            x = i;
            priority = p[i];
        }
    }
    if (b[x] > 0){
        if (rl[x] == -1) {
            rl[x] = c - at[x];
            b[x] --;
            c++; }
    if (b[x] == 0){
        count++;
        cl[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] = cl[i] - al[i]; }
for (int i=0; i<n; i++){
    wl[i] = tat[i] - bl[i]
printf ("priority Scheduling (pre-Emptive):\n");
printf ("PID \t Prior \tAT \tBT \tCT \tTAT \tWT \tRT\n");
for (int i=0; i<n; i++){
    printf ("%d %d %d %d %d %d %d %d\n", pid[i], p[i], at[i], bt[i], ct[i],
```

```
for (int i=0; i<n; i++)
    ttal += tal[i];
    twt += wt[i];
}
avg_tal = ttal/(double)n;
avg_wt = twt/(double)n;
printf("\n Average turnaround time \n", avg_tal);
printf("\n Average ~~turnaround~~ time \n", avg_wt);
waiting
}
```

O/P -

Enter Turnaround time :- 0 1 2 4.

Enter ~~waiting~~ Brust Time :- 5 4 2 1

Enter priority - 10 20 30 40

Process ID	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	

Average TAT Time - 5.50 MS

Average waiting time - 2.50 MS

lab-2 2c → priority_non-preemptive

```
void sort (int proc-id[], int p[], int al[], 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] = al[j];
            al[j] = temp;
            temp = bt[j];
            bt[j] = bt[i];
            bl[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, i=0;
    printf (" Enter no of processes ");
    scanf ("%d", &n);
    int proc-id[n], al[n], bt[n], ct[n], tat[n], wt[n], m[n],
    b[n], r[n], p[n];
    double avg-tat = 0.0, ttat = 0.0, avg-wt = 0.0, twt = 0.0;
    for (i=0; i<n; i++){
        proc-id[i] = i+1;
        r[i] = 0; }
    printf (" Enter priority ");
    for (i=0; i<n; i++) {
```

int b[], int n);

]], m[n],

t=0.0;


```
printf("Enter the arrival times:\n");
  for(i=0 ; i<n;i++){
      scanf("%d",&a[i]);}
printf("Enter brust time:\n");
  for(int i=0; i<n;i++){
     scanf("%d", &b[i]);
        b[i] = b[i]
        M[i] = 1
        r[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(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 (r[x]==-1)
              r[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 for (pre-Emptive):\n");
for(i=0; i<n; i++){
        printf("%d", proc_id[i], p[i], a[i], bt[i], ct[i],
        tat[i], wt[i], rt[i]);

for(i=0; i<n; i++){
    ttat += tat[i];
    twt += wt[i] }
avg_tat = ttat/(double)n;
avg_wt = twt/(double)n;
printf("\n avarage Trunaround TIME %d", avg_tat);
printf("\n avarage waiting TIME %d", avg_wt);
}
```

Enter no of process: 4
Enter priority - 10 20 30 40
Enter arrival time - 0 1 2 4
Enter borust time - 5 4 2 1

Non-preemptive scheduling

PID	Priority	AT	BT	CT	TAT	WT	RT
1	10	0	5	12	5	0	0
2	20	1	4	8	8	4	0
3	30	2	2	4	9	7	0
4	40	4	1	5	8	7	0

Lab 3 – Multilevel Queue: Program no - 3

39)

```
#include <stdio.h>
void findwaitingTime (int processes[], int n, int bt[], int at[],
int w[]){
    w[0]=0;
  for (i=0; i<n; i++){
    w[i] = bt[i-1] + wt[i-1] - at[i-1];
     if (wt[i]<0) {
         w[i]=0; } }

void findTurnaroundTime (int processes[], int n, int bt[], int wt[],
    int tat[]) {
    for (int i=0; i<n; i++){
     tat[i] = bt[i] + w[i] } }

void roundrobin (int process[], int n, int bt[], int at[], int quantum)
  {
    int w[n], tat[n], ct[n], total_wt = 0, total_tat = 0;
    int remaining_bt[n];
    int completed =0;
    int time = 0; }

  for (i=0; i<n; i++) {
      remaining_bt[i] = bt[i] }

  while (completed <n){
    for (int i=0; i<n; i++){
       if (remaining_bt[i] >0 && at[i] <= time){
         if (remaining_bt[i] <= quantum){
              time += remaining_bt[i];
             remaining_bt[i] =0;
              ct[i] = time)
               completed; }

       else {
            time += quantum
            remaining_bt[i] -= quantum } } } }

  findwaitingTime (process, n, bt, at, wt);
  findTurnaroundTime (processes, n, bt, wt, tat);
printf ["BT AT WT TAT CT");
  for (int i=0; i<n; i++){
```

```
printf("%d", process[i], bt[i], at[i], wt[i], tat[i], ct[i]);
        total_wt += wt[i]
        total_tat += tat[i]; }
printf("Avg waiting time = %f\n", (float) total_wt/n);
printf("Avg Turnaround time = \n", (float) total_tat/n);

void fcfs (int processes[], int n, int bt[], int at[]) {
    int wt[n], tat[n], ct[n], total_wt=0, total_tat=0;
    findwaiting time (process, n, bt, at, wt);
    find Turnaround time (process, n, wt, tat);
    printf("P BT AT WT TAT CT");
    for(int i=0; i<n; i++) {
        ct[i] = at[i] + bt[i];
        printf("%d", process[i], bt[i], wt[i], tat[i],
        ct[i]);
        total_wt += wt[i];
        total_tat += tat[i]; }

    printf("Avg waiting time (FCFS) = %f\n", (float) total_wt/n)
    printf("Avg TAT (FCFS) = %f\n", (float) total_tat/n);
    }
int main(){
    int processes[] = {1,2,3,4,5};
    int n = sizeof(processes)/sizeof(processes[0]);
    int bt[] = {10,5,8,12,15};
    int at[] = {0,1,2,3,4};
    int quantam = 2;
    roundRobin (processes, n, bt, at, quantam);
    fcfs (processes, n, bt, at);
    return 0; }
```

O/P -

P	BT	AT	WT	TAT	CT
P1	10	0	0	10	39
P2	5	1	10	15	23
P3	8	2	14	22	33
P4	12	3	20	32	45
P5	15	4	29	44	50

Avg WT (RoundRobin) = 14.600
Avg TAT (RoundRobin) = 24.600

P	BT	AT	WT	TAT	CT
P1	10	0	0	10	10
P2	5	1	10	15	6
P3	8	2	14	22	10
P4	12	3	20	32	15
P5	15	4	29	44	19

Avg WT (FCFS) = 14.6000
Avg TAT (FCFS) = 24.6000

lab-4a) Earliest Deadline first

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void sort(int proc[], int d[], int b[], int p[], 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=p[i];
        p[i]=p[j];
        p[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=0;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], pl[n], d[n], rem[n];
printf("cpu Burst time\n");
  for(int i=0; i<n; i++){
      scanf("%d", &b[i]);
       rem[i]=b[i]; }
printf("Enter the dead lines");
  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", p[i]);
  for(int i=0; i<n; i++){
       proc[i]=i+1; }

sort(proc, d, b, pl, n);
  int l = lcm(pl, n);
  printf("\nEDS\n");
  for(int i=0; i<n; i++){
       printf("%d", proc[i], b[i], d[i], pl[i]);
       printf("Scheduling for %d\n", l);

int time=0, prev=0, x=0;
int nextDeadlines[n];
  for(int i=0; i<n; i++){
       nextDeadlines[i]=b[i];
       rem[i]=b[i]; }
  while(time<l){
       for(int i=0; i<n; i++){
            if(time%p[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]<min
                                  Deadline){
       minDeadline=nextDeadline[i];
       taskToExecute=i; } }
```

```
if (tasktoexecute != -1) {
    printf ("%d task is running \n", time, proc[tasktoexecute]);
    rem[tasktoexecute]--; }
else {
    printf ("%d.dMs: cpu is idle\n", time); }
    time++;
}
```

```
Enter the no of process: 3
Enter the cpu Brust time:
0 1 2
Enter the deadlines:
8 5 4
Enter the time periods:
3 4 6
```

Earliest Deadline Scheduling

PID	Brust	Deadline	Period	
3	2	4	6	
2	1	5	4	
1	0	8	3	

0MS Task 3 is running
1 MS Task 3 is running
2MS Task 2 is running
3 MS cpu is IDLE
4MS Task 2 is running
5MS cpu is idle
6MS Task 3is running
7 MS Task 3 is running
8 MS cpu is IDLE
9MS CPU is IDLE
10 MS cpu is IDLE
11 MS cpu is IDLE.

Lab 4 - Rate - Monotonic 3/6/24

```
#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 lcmul = 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 no of process");
    scanf("%d", &n);
    int proc[n], b[n], pt[n], rem[n];
    printf("Enter the cpu Brust time");
    for(int i=0; i<n; i++){
```

```
scanf ("%d", &b[i]);
    rem[i] = b[i];
  }
printf ("Enter the time period \n");
for (int i=0; i<n; i++){
    scanf ("%d", &pl[i]); }
for (int i=0; i<n; i++) {
    proc[i] = i+1; }
sort (proc, b, pl, n);
int l = lcuml (pl, n);
  printf ("LCM \n", l);
printf ("Rate Monotone Scheduling ");
printf ("PID \t Brust \t period \n");
 for (int i=0; i<n; i++){
    printf ("%d", proc[i], b[i], pl[i]);
double sum = 0.0;
for (int i=0; i<n; i++){
    sum = (double) b[i] / pl[i];
  }
double rhs = n * (pow (2.0, (1.0/n)) - 1.0);
printf ("%.5f\n", sum, rhs, (sum <= rhs) ? "true" : "false");
if (sum > rhs) {
    exit(0); }
printf ("Scheduling occure %d ms \n\n", l);
int time = 0; prev = 0, x = 0;
while (time < l) {
   int f = 0;
   for (int i=0; i<n; i++) {
     if (time % pl[i] == 0) {
        rem[i] = b[i]; }
if (rem[i] > 0) {
    if (prev != proc[i]) {
        printf ("%d onwards : process %d ", time);
         prev = proc[i];
        rem[i]--;
         f = 1
         break;
         x = 0; } }
```

```
if (!f) {
    if (x! = 1) {
        printf ("%.dms onwards : cpu is idle \n", time);
            x = 1;
        }
    }
time++;
}
}
```

o/p-

Enter noof process -3

Enter no of cpu burst time

3 2 2

Enter the time period

20 5 10

LCM = 20

Rate Monotone Scheduling

PID	Burst	Period
2	2	5
3	2	10
1	3	20

0.750000 < 0.779763 ⇒ True

Scheduling occurs for 20MS

0MS onwards : process 2 running
2MS onwards : process 3 running
4MS onwards : process 1 running
5MS onwards : process 2 running
7MS onwards : process 1 running
8MS onwards : cpu is Idle
10MS onwards : process 2 running.

5/6

4c → Propotional Scheduling →

```
#include <stdio.h>
# include <stdio.h>
# include <time.h>
# define MAX_TASKS 10
# define MAX_TICKETS 100
# define TIME_UNIT_DURATIONS_MS 100

struct time{
    int tid;
    int tickets; };
void schedule (struct Task tasks[], int num_tasks, int * time_span_
    ms ){
  int total_tickets = 0;
      for (int i = 0; i < num_tasks ; i++) {
          total_tickets += task[i] . tickets; }
      srand (time (NULL));
      int current_time = 0;
     int completed_task = 0;
      printf ("process scheduling ");
    while (completed_tasks < num_tasks){
        int winning_ticket = rand() % total_tickets;
      int cumulative_tickets = 0;
      for (int i = 0; i < num_tasks; i++){
        cumulative_tickets += tasks[i]. tickets;
        if (wining ticket < cumulative_tickets){
            printf ("Time %d - %d: task %d ", current_tm,
          current_tim + 1, task[i]. tid);
              break; }
          }
        completed_task ++;
        }
    * time_span_ms = current time * Time_unit_
          duration_ms; }
```

```
int main(){
  struct Task Tasks[MAX_TASKS];
    int num_tasks;
    int time_span_ms;
    printf("Enter no of tasks");
    scanf("%d", &num_tasks");

if (num_tasks <=0 || num_tasks > MAX_TASKS){
    printf("Invalid no", MAX_TASKS);
      return 1;

printf("Enter Ticket");
  for(i=0; i<n; i++){
   task[i].tid = i+1;
    printf("task ", tasks(i).tid);
     scanf("%d", &tasks(i).tickets);
        }
  printf("\n Running Task");
   schedule(tasks, num_tasks, &time_span_ms);
   Printf("\n Timespan of the gant chart: %d Milliseconds\n",
                                          timespan_ms);
return 0
}
```

o/p-

Enter noof Tickets -> 3

Enter no of Tickets ofor each task.

Task 1 Ticket 10

Task 2 Ticket 20

Task 3 Ticket 30

Running Task

process Scheduling

Time 0-1 Task 2 is running

Time 1-2 Task 2 is running

Time 2-3 Task 3 is running

Time span of the gant chost = 300 Milliseconds

Lab-5

Write a c program to simulates LAB-5- producer-consumer problem using simaphones.c

```
#include<stdio.h>
#include<stdlib.h>
 int Muter=1,
 int full=0;
 int Empty=5;
  int x=0;
int main(){
void producer();
 void consumer();
int ~~void~~ wait(int);
  int signal(int);
printf("1n1. producer \n2. consumer\n3. Exit");
 while(1){
    printf("Enter your choice")
~~while(1)~~ scanf("%d",&n);

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

 — case 2:
      if ((Muxer==1)&&(full!=0))
          consumer();

      else  printf("Buffer is Empty");
          break)

 — case 3:
      exit(0)
       break; } }

 return 0; } }
```

```
int wait(int s)
    return (--s);

int signal (int s)
    ++s;

void producer() {
    Mutex = wait (Mutex)
    full = signal (full)
    empty = wait (empty);
     x++
    printf ("\n producer produces the item %d ", x);
    Mutex = signal (Mutex); }

void consumer() {
    Mutex = wait (Mutex)
    full = wait (full)
    empty = signal (empty)
    printf ("\n producer produces the item %d", x)
     x--
    Mutex = signal (Mutex); }
}
```

O/P →

1) producer
2) consumer
3) Exit
producer produces the item 1
Enter your choice 1
Producer produce the item 2
Enter your choice 1
Producer produce the item 3
Enter your choice 1
producer produce the item 4
Enter your choice 1
producer produce the item 5
Enter your choice 1
Buffer is full!!
consumer will consume item 5
Enter your choice: 2
Consumer will consume item 4
Enter your choice: 2
Consumer will consume item 3
Enter your choice: 2
Consumer will consume item: 2
Enter your choice: 2

consumer will consume item: 1
Enter your choice: 2
Buffer is empty!!
Enter your choice

Lab-7

Banker's Algorithm

```
int main(){
  int n, m;
printf("Enter the no of process : ");
scanf("%d", &n);
printf("Enter the no of Resources");
scanf("%d", &m);
int available[m];
printf("Enter the available Resources : ");
  for(int i=0; i<m; i++){
    scanf("%d", &available[i][j]);
  }
  int allocation[n][M];
printf("Enter the allocation Resources for each process : \n");
  for(int i=0; i<n; i++){
    for(int j=0; j<n; j++){
      scanf("%d", &allocation[i][j]); } }

int need[n][m];
  for(int i=0; i<n; i++){
    for(int j=0; j<m; j++){
      need[i][j] = MAXIMUM[i][j] - allocation[i][j]; } }

printf("process Allocation Max need \n");

  for(int i=0; i<n; i++){
    printf("\nP%d   ", i+1);
    for(int j=0; j<M; j++){
      printf("%d", allocation[i][j]);
    }
  printf("| ");
  for(int j=0; j<m; j++){
    printf("%d", MAXIMUM[i][j]); }

printf("|");
  for(int j=0; j<m; j++){
    printf("%d", need[i][j]); }
    printf("| \n"); }
```

```
int work[m];
for(int i=0; i<m; i++){
        work[i] = available[i];
    }
int finish[n];
for(int i=0; i<n; i++){
        finish[i] = 0; }
int safesequence[n];
int count = 0;
int safe = 1
while(count<n){
    int found = 0;
    for(int j=0; i<n; i++){
        if(finished==0){
            int j;
            for(j=0; j<m; j++){
                if(need[i][j] > work[j]){
                    break; } }
        if(j==M){
            for(j=0; j<M; j++){
                work[j] += allocation[i][j]; }
            finish[i] = 1;
            safe sequence[count+1] = i;
            found = 1; }
        }
    }
if(!found){
    safe = 0;
    break; } }
if(safe){
    printf("the system is in a safe state");
    printf("safety sequence");
    for(int i=0; i<n; i++){
        printf("P%d", safesequence[i]+1);
    }
    printf("\n"); }
else{
    printf("read to deadlock\n"));
}
```

Output-

Enter the no of process: 5
Enter the no of Resources: 3
Enter the available resources: 3 3 2
Enter the MAXIMUM resources for each process:

7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

Enter the allocated Resources for each process:

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

Process	Allocation	Max	Need
P1	0 1 0	7 5 3	7 4 3
P2	3 0 2	3 2 2	0 2 0
P3	3 0 2	9 0 2	6 0 0
P4	2 1 1	2 2 2	0 1 1
P5	0 0 2	4 3 3	4 3 1

The system in a safe state

safety sequence: P2, P3 P4 P5 P1

19/6

Lab-6

Dinning philosophers 10

```
#define MAX_PHILOSOPHERS 10
typedef enum {Thinking, Hungry, Eating} state_t;
state_t states[MAX_PHILOSOPHER];
int num_philosophers;
int num_hungry;
int hungry_philoshophers[MAX_PHILOSOPHER];
int front[MAX_PHILOSOPHER];

void print_state(){
    printf("\n");
    for(int i=0; i<num_philosophers; ++i){
      if(states[i]==Thinking) printf("P %d is thinking\n", i+1);
      else if(states[i]==Hungry) printf("P %d is eating", i+1);
      else if(states[i]==EATing) printf("P %d is eating\n", i+1);
    }
}

int can_eat(int philosopher_id){
    int left_fork = philoshopher_id;
    int right_fork = (philoshopher_id + 1) % num_philosophers;
    if(forks[left_fork]==0 && fork[right_fork]==0){
        forks[left_fork] = forks[right_fork]=1;
          return 1; }
        return 0;
    }
void simulate(int allow_two){
    int eating_count=0;
    for(int i=0; i<num_hungry; ++i){
      int philosopher_id = hungry_philosopher[i];
      if(states[philoshopher_id]==Hungry){
        if(can_eat[philosopher_id]){
          states[philosopher_id] = EATING;
          eating_count++;
    printf("P %d is granted to eat\n", philosopher_id+1);
    if(allow_two && eating_count==1) break;
    if(!allow_two && eating_count==2) break;
```

```
}
}
}
for (int i=0; i< NUM_hungry ;i++) {
   if philosopher_id = hungry_philosopher[i];
    if (states[philoshoper_id] == EATING) {
       int left_fork = philosopher_id;
       int right_fork = (philosopher_id + 1) % NUM_philosophers;
       forks[left_fork] = forks[right_fork] = 0;
       states[philosopher_id] = THINKING;
    }
  }
}

int main() {
 printf("Enter philosophers (max %d): ", MAX_PHILOSOPHERS);
 scanf("%d", &NUM_philosophers);

 if (NUM_philosophers < 2 || NUM_philosophers > MAX_PHILOSOPHERS) {
      printf("Invalid no of philosophers. Exiting\n");
      return 1;
   }
 printf("How many are hungry : ");
  scanf("%d", &num_hungry);

 for (int i=0; i< NUM_hungry; i++) {
    printf("Enter philosopher %d position ", i+1);
    int position;
    scanf("%d", &position);
    hungry_philosophers[i] = position-1;
    states[hungry_philosophers[i]] = Hungry;
   }
 for (int i=0; i<NUM_philosophers; ++i) {
      fork[i]=0; }
int choice;
 do {
 print_state();
  printf("\n1. one can eat at a time\n");
  printf("2. two can eat at a time\n");
  printf("3 Exit\n")
   printf("Enter your choice");
```

```
scanf ("%d", &choice);
switch (choice) {
    case 1:
        simulate (0);
        break;
    case 2:
        simulate (1);
        break;
    case 3:
        printf ("Exiting\n");
        break;
    default:
        Printf ("Invalid choice. please Try again\n");
        break;
    }
} while (choice != 3);
    return 0;
}
```

o/p - P2 is granted to eat

P1 is thinking
P2 is thinking
P3 is waiting
P4 is thinking
P5 is waiting

(1) one can eat at a time
(2) two can eat at a time
(3) exit.

Enter your choice 2

P3 is granted to eat
P5 is granted to eat

P1 is thinking
P2 is thinking
P3 is thinking
P4 is thinking
P5 is thinking

1 one can eat at a time
2 Two can eat at a time
3 exit

Enter choice :3

Deadlock Detection problem

```
#include <stdio.h>
void main(){
    int n,m,i,j
    printf("Enter no of process & noof resources type");
    scanf("%d %d", &n,&m);
   int max[n][m], need[n][m], all[n][m], ava[m], flag=1, dead[n];
    c=0;
  printf("Enter max no of each type of resources need by each process\n");
for(i=0;i<n;i++){
        for(j=0;j<m;j++){
            scanf("%d",&max[i][j]);
        }
    }
printf("Enter the allocate no of each type of resources need by each
    process\n");
 for(i=0;i<n;i++){
     for(j=0;j<m;j++){
         scanf("%d", &all[i][j]); }
                    }
 printf("Enter the available number of each type of resources\n");
    for(j=0;j<m;j++){
        scanf("%d", &ava[j]);
    }
    for(i=0;j<n;j++){
            need[i][j] = max[i][j]-all[i][j];
        }
    }
   for(i=0;i<n;i++){
       finish[i]=0; }
   while(flag){
           flag=0;
           for(i=0;i<n;i++){
               c=0;
               for(j=0;j<m;j++){
                   c=0;
```

```
if (finish[i]==0 && need[i]
    c++;
    if(c==m){
    for(j=0;j<m;j++){
        ava[j]+= all[i][j]
          finish[i]=1
           flag=1;}
    if(finish[i]==1){
          i=n;}}}}}

j=0;
flag=0;
for(i=0;i<n;i++){
    if(finish[i]==0){
        dead[j]=i;
          j++;
          flag=1;}}
    if(flag==1){
        printf("Deadlock has occured\n");
        printf("The dead lock process are:\n");
          for(i=0;i<n;i++){
              printf("%d", dead[i]);}}
    else printf("No deadlock");
                                    }
```

o/p -

Enter no of process & no of type of resources: 4 3

Enter the allocated no of each type of resource need by each process: 1 0 2

2 1 1

1 0 3

1 2 2

Enter the available no of each type of resources:

0 0 0

Enter the request no of each type of resource needed by each process

0 0 1

1 0 2

0 0 0

3 3 0

Deadlock has occured.

The Deadlock processes are;

P_3.

LAB-9

3/7/24

write a program to simulates the following contiguous memory allocation - Worst fit, Best fit, first fit

```
#include <stdio.h>
struct Block {
    int Block_No;
    int block_size;
    int is_free; };
struct file {
    int file_no;
    int file_size; };
void firstfit (struct Block Blocks[], int n_blocks, struct file files[],
  int n_files) {
    printf ("MMS - first fit \n");
    printf ("file_no: \t file_size \t Block_no: \t Block_size: \t frag
    ment\n");
    for (int i=0; i<n_files ;i++) {
      for (int j=0; j<n_blocks ; j++) {
        if (blocks[j]. is_free && block[j] . block_size) = files[i]. file_size){
          block [j] . is_free =0)
        printf ("%d \t\t %d \t\t %d \t\t %d\n", files[i].
    fileno, files [i]. file_size , Blocks[j]. block_no , block[j]. block_size
  _files[i]. file_size);
        break; } } }
void worstfit (struct Block blocks[], int n_blocks, struct file files[],
  int n_files) {
    printf (" MMS - worst file\n");
    printf (" file_no: \t file_size: \t Block_no: \t Block_size: \t Fragment\n");
    for (int i=0; i<n_files ; i++) {
      int worst_fit_block = -1;
      int Max_fragment = -1;
      for (int j=0; j<n_blocks ; j++) {
        if (block[j]. is_free && block [j]. block_size >= files [i]. file_
                                                              size) {
      int fragment = blocks [j]. block_size - files [i]. file_size;
      if (fragment > Max_fragment) {
          Max_fragment = fragment;
          worst_fit_block = j; } }
      if (worst_fit_block != -1) {
        blocks [worst_fit_block]. is_free =0;
        printf ("%d \t\t %d \t\t %d \t\t %d\n", files[i].
```

```
size, Max_fragment) { y y
void best_fit (struct Block blocks [], int n_blocks, struct file files [], int n_files){
printf (" MMS - Best fit \n");
printf (" File_no \t File_size \t Block no: \t block_size : \t fragment \n");
    for (int i=0 ; i < n_files ; i++){
     int best_fit_block = -1;
     int min_fragment = 1000;
     for (int j=0 ; j < n_blocks ; j++){
         if (blocks [j]. is_free && blocks [j] . block_size >= files[i].
                                  file_size){
     int fragment = blocks [j]. block_size - files [i]. file_size;
      if (fragment < min_fragment){
           min_fragment = fragment;
           best_fit_block = j; } }
     if (best_fit_block != -1){
        block [best_fit_block]. is_free = 0;
        printf (" ");
     } }
 }
int main(){
    int n_blocks, n_files;
    printf ("Enter no of blocks ");
    scanf ("%d", &n_blocks);
    printf ("Enter the no of files: ");
    scanf ("%d" &n_files);
  struct Block blocks [n_blocks];
    for (int i=0; i < n_blocks ; i++){
        blocks[i]. block_no = i+1;
        printf ("Enter no of size of block %d ", i+1);
        scanf ("%d", &blocks [i]. block_size);
        blocks [i] is_free = 1; }
  struct file files [n_files];
  for (int i=0; i < n_files ; i++){
        files[i]. file_no = i+1;
        printf (" Enot of file %d ", i+1);
        scanf ("%d", &files [i]. file_size);
```

```
firstfit(blocks, n_blocks, files, n_files);
printf("\n");
for(int i=0; i<n_blocks; i++){
    block[i].is_free=1; }
worstfit(blocks, n_blocks, files, n_files);
printf("\n");
for(int i=0; i<n_blocks; i++){
    blocks[i].isfree=1;}
bestfit(blocks, n_blocks, files, n_files);
    return 0; }
```

o/p-

```
Enter the no of block: 3
Enter the no of files: 2
Enter the size of block1: 5
Enter the size of block2: 2
Enter the size of block3: 7
Enter the size of file1: 1
Enter the size of file2: 4
```

MMS - first fit

file no:	file size:	Block_no:	Block_size:	fragment
1	1	1	5	4
2	4	3	7	3

MMS - worst fit

file no:	file_size:	Block_no:	Block_size	fragment
1	1	3	7	6
2	4	1	5	1

MMS - Best fit

file NO	filesize	Block_NO	Block_size	fragment
1	1	2	2	1
2	4	1	5	1

Lab-10

write c program page replacement. a) FIFO b) LRU c) optimal

```
#include <stdio.h>
#include <limits.h>
#include <stdlib.h>
void fifo(int pages[], int n, int capacity){
    int frame[capacity], index=0, page_faults=0;
    for(int i=0; i<capacity; i++){
        frame[i]=-1
    for(int i=0; i<n; i++){
        int found=0;
        for(int j=0; j<capacity; j++){
            if(frame[j]==page[i]){
                found=1
                break; } }
        if(!found){
            frame[index]=page[i];
            index=(index+1)%capacity;
            page_faults++; }
        }
    printf("FIFO page faults: %d\n", page_faults); }
void lru(int page[], int n, int capacity){
    int frame[capacity], counter[capacity], time=0,
    page_fault=0;
    for(int i=0; i<capacity; i++){
        frame[i]=-1;
        frame[i]=0; }
    for(int i=0; i<n; i++){
        int found=0;
        for(int j=0; j<capacity; j++){
            if(frame[j]==pages[i]){
                found=1;
                counter[j]=time++;
                break; }
        if(!found){
            int min=INT_MAX, min_index=-1
            for(int j=0; j<capacity; j++){
```

```
    if (counter[j] < min) {
        min = counter[j];
        min_index = j;
    }
}
frame[min_index] = pages[i];
counter[min_index] = time++;
page_faults++; } }
printf("LRU Page faults : %d\n", page_faults); }

void optimal(int pages[], int n, int capacity){
int frame[capacity], page_faults = 0;
for(int i=0; i<capacity; i++){
    frame[i] = -1;
for(int i=0; i<n; i++){
    int found = 0;
    for(int j=0; j<capacity; j++){
        if(frame[j] == pages[i]){
            found = 1;
            break;
        }
    }
    if(!found){
        int farthest = i+1; index = -1;
        for(int j=0; j<capacity; j++){
            int k;
            for(k=i+1; k<n; k++){
                if(frame[j] == pages[k]){
                    break; }
                if(k > fastest){
                    fastest = k;
                    index = j; } }
    if(index == -1){
        for(int j=0; j<capacity; j++){
            if(frame[j] == -1){
                index = j;
                break } } }
```

```
frame[index] = pages[i];
page_faults++;
}
}
printf("optimal page faults : %d\n", page_faults);
}
int main(){
int n, capacity;
printf("Enter the no of pages");
scanf("%d",&n);
int *pages = (int*)malloc(n * sizeof(int));
printf("Enter the ~~noof~~ pages:")
for(int i=0; i<n; i++){
scanf("%d", &pages[i]);}
printf("Enter noof frame capacity;");
scanf("%d", &capacity);
printf("\n pages:");
for(int i=0; i<n; i++){
printf("%d", pages[i]);
printf("\n\n");
fifo(page, n, capacity);
lru(pages, 0, capacity);
optimal(pages, n, capacity)
free(pages)
return 0
}
```

Output -

Enter the no of pages -20

Enter the pages - 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

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

FIFO page faults: 15

LRO page faults: 12

optimal page faults: 9

10/7/24

Complete!