INDEX

Name : Manav Kumar Class : .....

Section : C Roll No. : ..... Subject : OS Lab,

Sl. No.	Date	Title	Page No.	Teacher's Sign. / Remarks
1.	08/05/24	FCFS Scheduling	01-02	
2.	15/05/24	SJF Scheduling	03-08	
3.	15/05/24	Priority Scheduling	09-12	
4.	15/05/24	Round Robin Scheduling	13-16	
5.	22/05/24	Multilevel Queue Scheduling	17-28	19/6
		Rate Monotonic		
		Earliest Deadline		
6.	05/06/24	Proportional Scheduling	29-32	
7.	12/06/24	Producer consumer problem	33-35	
8.	19/06/24	Dining philosopher's	36-41	
		Banker's problem		
9.	03/07/24	Deadlock detection.	42-44	10/7
10.	03/07/24	Contigous Memory Allocation	45-49	
11.	10/07/24	Page Replacement Algorithms	50-54	

08/05/2024

Date
Page 01

Q. Program 1:
FCFS and SJF Scheduling.

Q) FCFS Scheduling:

→
```
#include<stdio.h>
void compTime(int n, int a[], int b[], int c[]) {
int t=0;
for (int i=0; i<n; i++){
  if(a[i]>t) {
    t= a[i];
    t+= b[i];
    c[i] = t;
  }
}
void tatTime (int n, int a[], int c[], int t[]) {
for (int i=0; i<n; i++)
    t[i] = c[i] - a[i];
void waitTime (int n, int b[], int t[], int w[]) {
for (int i=0; i<n; i++)
    w[i] = t[i] - b[i];
void avgTime (int n, int a[], int b[])
{
int c[n], t[n], w[n];
int sumTAT=0, sumWT=0;
compTime(n,a,b,c);
tatTime(n,a,c,t);
waitTime(n,b,t,w);
printf("p A B C T W \n");
for(int i=0; i<n; i++)
{
  printf("%d %d %d %d %d %d \n", i+1,
    a[i], b[i], c[i], t[i],
    w[i]);
```

```
sumTAT += t[i];
sumWT += w[i];
float avgTAT = (float)sumTAT/n;
float avgWT = (float) sumWT/n;
printf("\n Average TAT: %f \n", avgTAT);
printf(" Average WT: \n", avgWT);}
int main(){
    int n=4;
    int a[] = {0,1,5,6};
    int b[] = {2,2,3,4};
    avgTime(n,a,b);
}
```

Output:

P	A	B	C	T	W
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 TAT: 3.500000

Average WT: 0.750000

Date
Page 03

15/05/2024.

Q. SJF (Non- preamptive) (1-B)

→
```
#include<stdio.h>
void findCompletionTime(int prosessess[], int n, int bt[],
                        int at[], int wt[], int tat[], int v[],
                        int ct[])
{
  int completion[n];
  for(int i=0; i<n; i++){
      remaining[i] = bt[i];
  }
  int currentTime=0;
  for (int i=0; i<n; i++)
  {
      int shortest = -1;
      for (int j=0; j<n; j++)
      {
          if (at[j] <= currentTime && remaining[j] >0)
          {
              if (shortest == -1 || remaining[j] < remaining
                                                  [shortest])
              {
                  shortest=j;
              }
          }
      }
      if (shortest == -1)
      {
          currentTime++;
          continue;
      }
      completion[shortest] = currentTime + remaining[shortest];
      currentTime = completion[shortest];
      wt[shortest] = currentTime - bt[shortest] - at[shortest];
      tat[shortest] = currentTime - at[shortest];
```

```
rt[shortest] = rt[shortest];
}
printf("Process\t Arrival Time\t Burst Time\t Waiting Time
\t Turn Around Time\t Response Time\t completion Time\n");
for(int i=0 ;i<n;i++)
{
ct[i] = completion[i];
printf("%d\t\t %d\t\t %d\t\t %d\t\t %d\t\t %d\n",
processes[i], at[i], bt[i], wt[i], tat[i], rt[i],
ct[i]);
}
}
void main(){
int n;
printf("Enter the number of processes:");
scanf("%d",&n);
int processes[n], burst_time[n], arrival_time[n];
printf("Enter process Number:\n");
for(int i=0; i<n; i++)
{
scanf("%d", &processes[i]);
}
printf("Enter Arrival Time:\n");
for(int i=0; i<n; i++)
{
scanf("%d", &arrival_time[i]);
}
printf("Enter Burst Time:\n");
for(int i=0; i<n; i++)
{
scanf("%d", &burst_time[i]);
}
```

Date
Page

```
int wt[n], tat[n], rt[n], ct[n];
for(int i=0; i<n; i++)
    rt[i]=-1;
printf("\n SJF (Non-preemtive) Scheduling: \n");
find completionTime( processes, n, burst-time, arrival-time,
                    wt, tat, rt, ct);
}
```

Output:

Enter the number of process: 5

Enter process number:

1 2 3 4 5

Enter Arrival Time:

2 1 4 0 2

Enter Burst Time:

1 5 1 6 3

SJF(Non-preemtive) Scheduling:

Process	AT	BT	WT	TAT	RT	CT
1	2	1	4	5	4	7
2	1	5	10	15	10	16
3	4	1	3	4	3	8
4	0	6	0	6	0	6
5	2	3	6	9	6	11

Average Turn Around Time: 7.800000

Average Waiting Time: 4.600000

Q. SJF (Preemtive) :-

→

```
#include <stdio.h>
void findCompletionTime (int processes[], int n, int bt[], int
                         at[], int wt[], int tat[], int ct[])
{
    int remaining[n];
    int currentTime = 0;
    int completed = 0;
    for (int i=0; i<n; i++) {
        remaining[i] = bt[i];
    }
    while (completed < n)
    {
        int shortest = -1;
        for (int i=0; i<n; i++)
        {
            if (at[i] <= currentTime && remaining[i] > 0)
            {
                if (shortest == -1 || remaining[i] <= remaining
                                      [shortest])
                {
                    shortest = i;
                }
            }
        }
        if (shortest == -1)
        {
            currentTime ++;
            continue;
        }
        remaining[shortest]--;
        if (remaining[shortest] == 0)
        {
```

```
        completed++;
        ct[shortest] = currentTime + 1;
        wt[shortest] = ct[shortest] - bt[shortest] - at[shortest];
        tat[shortest] = ct[shortest] - at[shortest];
      }
      currentTime++;
    }
printf("Process \t ArrivalTime \t BurstTime \t WaitingTime \t
    TurnAround Time \t ResponseTime \t CompletionTime \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", process[i], at[i], bt[i],
    ct[i], tat[i], wt[i]);
  }
}
void main()
{
  int n;
printf("Enter number of processess:");
scanf("%d", &n);
int processess[n];
int burst_time[n];
int arrival_time[n];
printf("Enter process numbers :\n");
for(int i=0; i<n; i++)
{
  scanf("%d", &processes[i]);
}
printf("Enter Arrival Time:\n");
for(int i=0; i<n; i++)
  {
```

```
    scanf("%d", &arrival_time[i]);
    }
    printf("Enter Burst Time:\n");
    for(int i=0; i<n; i++)
    {
        scanf("%d", &burst_time[i]);
    }
    int wt[n], tat[n], ct[n];
    printf("\n SJF (Preemtive) Scheduling:\n");
    findCompletionTime(processes, n, burst_time,
        arrival_time, wt, tat, ct);
    for(int i=0; i<n; i++){
        atat= atat+tat[i];
        awt= awt+wt[i];
    }
    printf("Average Turn Around Time=%f", atat/n);
    printf("Average Waiting Time=%f", awt/n);
}
```

Output:

Enter the number of processes: 5
Enter process Number: 1 2 3 4 5
Enter Arrival Time: 2 1 4 0 2
Enter Burst Time: 1 5 1 6 3
SJF(Preemptive) Scheduling:

Process	AT	BT	WT	TAT	RT	CT
1	2	1	0	1		3
2	1	5	10	15		16
3	4	1	0	1		5
4	0	6	5	11		11
5	2	3	2	5		7

Average Turn Around Time = 6.600000
Average Waiting Time = 3.400000

Program 2 (a)

Date
Page

Q. Priority Scheduling:

```
#include <stdio.h>
#include <stdbool.h>
void findCompletionTime(int processes[], int n, int bt[],
  int at[], int wt[], int tat[], int ct[], int rt[],
  int priority[], bool isLowerPriorityHigher)
{
  int remaining[n];
  int currentTime = 0;
  int completed = 0;
  bool isFinished[n];
  for (int i=0; i<n; i++) {
      remaining[i] = bt[i];
      isFinished[i] = false;
      rt[i] = -1;
  }
  while (completed < n) {
      int highestPriorityIndex = -1;
      int highestPriority = isLowerPriorityHigher ? 1000000 : -1;
      for(int i=0; i<n; i++) {
          if (at[i] <= currentTime && !isFinished[i] &&
          ((isLowerPriorityHigher && priority[i] < highestPriority) ||
          (!isLowerPriorityHigher && priority[i] > highestPriority))) {
          highestPriority = priority[i];
          highestPriorityIndex = i;
      }
  }
  if (highestPriorityIndex == -1) {
      currentTime++;
      continue;
  }
```

```
int currentProcess = highestPriorityIndex;
if (rt[currentProcess] == -1) {
    rt[currentProcess] = currentTime - at[currentProcess];
}
remaining[currentProcess]--;
currentTime++;
if (remaining[currentProcess] == 0) {
    isFinished[currentProcess] = true;
    completed++;
    ct[currentProcess] = currentTime;
    tat[currentProcess] = ct[currentProcess] - at[currentProcess];
    wt[currentProcess] = tat[currentProcess] - bt[currentProcess];
    }
}
printf("Process\t Arrival Time\t Burst Time\t Priority\t
    Completion Time\t Turn Around Time\t Waiting Time\t
    Response 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\t\t
    %d\t\t %d\n", process[i], at[i], bt[i],
    priority[i], ct[i], tat[i], wt[i], rt[i]);
}
}
void main()
{
    int n;
    printf("Enter the number of processess:");
    scanf("%d", &n);
    int processes[n];
    int burst_time[n];
    int arrival_time[n];
    int priority[n];
```

```
int priorityType;
bool isLowerPriorityHigher;
printf("Enter 1 if lower number indicates higher priority, 0
if Number indicates higher priority: ");
scanf("%d", &priorityType);
isLowerPriorityHigher = (priorityType == 1);
printf("Enter Process Number: \n");
for (int i=0; i<n; i++) {
    scanf("%d", &processes[i]);
}
printf("Enter priority :\n");
for(int i=0; i<n; i++) {
    scanf("%d", &priority[i]);
}
printf("Enter Arival Time :\n");
for(int i=0 ; i<n; i++) {
    scanf("%d", &arival_Time[i]);
}
printf("Enter Burst Time :\n");
for(int i=0; i<n; i++) {
    scanf("%d", &burst_time[i]);
}
int wt[n], tat[n], ct[n], rt[n];
printf("\n Priority Scheduling (Preemptive): \n");
findCompletionTime(processes, n, burst_time, arival_time,
    wt, tat, ct, rt, priority, isLowerPriorityHigher);
}
```

iii. Output:

Enter the number of processes: 4
Enter 1 if lower number indicates higher priority, 0 if higher number indicates higher priority: 0
Enter process Number: 1 2 3 4
Enter priority:
10 20 30 40
Enter Arrival Time:
0 1 2 4
Enter Burst Time:
5 4 2 1

Priority Scheduling (Preemptive):

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

Average Turnaround Time = 5.500000
Average Waiting Time = 2.500000

Program 2-(b):

Q. Round Robin :-

```
#include <stdio.h>
#include <stdbool.h>
void findCompletionTime (int processess[], int n, int bt[],
 int at[], int wt[], int tat[], int ct[], int rt[], int quantum)
{
  int remaining[n];
  bool firstResponse[n];
  int currentTime=0;
  int completed=0;
  for(int i=0; i<n; i++) {
    remaining[i] = bt[i];
    firstResponse[i] = true;
  }
  int queue[n];
  int front=-1, rear=-1;
  void enque (int process) {
     if (rear == n-1)
        rear=-1;
     queue[++rear]= process;
      if (front ==-1)
         front=0;
  }
  int deque(){
   int process = queue[front];
   if (front == rear)
       front = rear=-1;
   else {
       front++;
       if (front==n)
          front=0;
     }
```

```
    return process;
}
bool inQueue[n];
for(int i=0; i<n; i++)
    inQueue[i]= false;
while (completed <n) {
    for (int i=0; i<n; i++) {
        if (at[i] <= currentTime && !inQueue[i]) {
            enqueue(i);
            inQueue[i] = true;
        }
    }
    if ( front == -1) {
        currentTime++;
        continue;
    }
int currentProcess = dequeue();
if (firstResponse[currentProcess]) {
    rt[currentProcess] = currentTime - at[currentProcess];
    firstResponse[currentProcess] = false;
}
if (remaining[currentProcess] > quantum) {
    remaining[currentProcess] -= quantum;
    currentTime += quantum;
} else {
    currentTime += remaining[currentProcess];
    remaining[currentProcess] =0;
    completed++;
    ct[currentProcess] = currentTime;
    tat[currentProcess] = ct[current process] - at[current process]
    wt[current process] = tat[current process] - bt[current process];
}
```

Date
Page

```
for (int i=0 ; i<n; i++ ) {
    if (at[i] <= currentTime && !inQueue[i]) {
        enqueue(i);
        inQueue[i] = True;
    }
}
if (remaining[current process]>0) {
    enqueue (current process);
}
}
printf ("Process \t Arrival Time \t Burst Time \t Completion
        Time \t Turn Around Time \t Waiting Time \t
        Response 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", processes[i], at[i], bt[i], ct[i], tat[i]
    wt[i], rt[i]);
    }
}
void main() {
    int n;
    printf("Enter the number of processes:");
    scanf ("%d", &n);
    int processcss[n];
    int burst_time[n];
    int arrival_time[n];
    printf("Enter Process Number:\n");
    for(int i=0; i<n; i++){
        scanf("%d", &processes[i]);
    }
printf ("Enter Arrival Time:\n");
```

```
for(int i=0; i<n; i++){
    scanf("%d", &arrival_time[i]);
}
printf("Enter Burst Time:\n");
for(int i=0; i<n; i++){
    scanf("%d", &burst_time[i]);
}
int quantum;
printf("Enter the time quantum:");
scanf("%d", &quantum);
int wt[n], tat[n], ct[n], rt[n];
printf("\n Round Robin scheduling:\n");
findCompletionTime(process, n, burst_time,
arrival_time, wt, tat, ct, rt, quantum);
}
```

#). Output:

```
Enter the number of processesses: 5
Enter process Number: 1 2 3 4 5
Enter Arrival Time: 0 1 2 3 4
Enter Burst Time: 5 3 1 2 3
Enter the time quantum: 2
```

Round Robin Scheduling:

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

Average Turn Around Time: 8.600000

Average Waiting Time: 5.800000

Program 3:

Date
Page

Q. Multi Level Queue Scheduling :-

→
```
#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=i; j<n; j++)
    {
      if (at[j] < at[i])
      {
        temp=at[i];
        at[i]= at[j];
        at[j]= temp;
        temp= bt[j];
        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 *c)
{
  double ttat=0.0, twt=0.0;
  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];
}
void main() {
int sn, un, c=0; int 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<sn; i++)
    sproc_id[i] = i+1;
printf("Enter arrival times of the system processes:\n");
for(int i=0; i<sn; i++)
    scanf("%d", &sat[i]);
printf("Enter burst times of the system processes:\n");
for(int i=0; i<sn; 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<un; i++)
    uproc_id[i] = i+1;
```

```
printf("Enter arrival times of the user processes: \n");
for(int i=0; i<un; i++)
    scanf("%d", &uat[i]);
printf("Enter burst times of the user processes: \n");
for(int i=0; i<un; 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 \t AT \t BT \t CT \t TAT \t WT \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 \t \n",
    uproc_id[i], uat[i], ubt[i], uct[i], utat[i], uwt[i]);
}
```

Output:

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

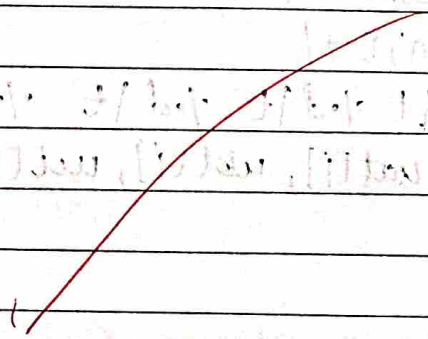
Scheduling:

System processess:

PID	AT	BT	CT	TAT	WT
1	0	4	4	4	0
2	2	3	7	5	2

User processess:

1	1	2	9	8	6
2	3	3	12	9	6
3	5	1	13	8	7



Program 4(a):

Q. Rate Monotonic Scheduling:-

→

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void sort (int proc[], int b[], int pt[], int n)
{
  int temp=0;
  for(int i=0; i<n; i++)
   {
    for( int j= 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)
{
  ont 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 l= lcmul (pt,n);
printf ("LCM= %d \n", l);
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]);
double sum=0.0;
for (int i=0; i<n; i++)
 {
   sum += (double) b[i] /pt[i];
 }
double rhs = n * (pow (2.0, (1.0/n)) - 1.0);
printf("\n %lf <= %lf => %s \n", sum, rhs, (sum<=rhs)?
      "true" : "false");
if (sum > rhs)
   exit(0);
printf (" Scheduling occurs for %d ms \n\n", l);
int time=0, prev=0, n=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 = prd[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 the number of processes: 3
Enter the CPU burst times:
2 1 2
Enter the time periods:
5 3 7
LCM = 105
```

Rate Monotone Scheduling:

PID	Burst ~~Period~~	Period
2	1	3
1	2	5
3	2	7

1.019048 <= 0.779763 ⇒ false.

Program 4(b):

Q. Earliest Deadline scheduling:-

→

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

```
while(b>0)
{
  r= a%b;
  a=b;
  b=r;
 }
return a;
 }
int 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 l = lcmul(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", l);
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: 2 1 2
Enter the deadlines: 5 3 7
Enter the time periods: 5 3 7
Earliest Deadline Scheduling:
```

PID	Burst	Deadline	Period
2	1	3	3
1	2	5	5
3	2	7	7

Scheduling occurs for 105 ms.

Date: 05/06/2024.
Program 4(c):

(#) Proportional scheduling:-

→
```
#include<stdio.h>
#include<stdlib.h>
void sort_by_burst(int proc[], int burst[], int tickets[],
    int remaining[], int n){
 for(int i=0; i<n-1; i++){
  for(int j=0; j<n-i-1; j++){
   if(burst[j] > burst[j+1]){
      int temp= proc[j];
      proc[j] = proc[j+1];
      proc[j+1]= temp;

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

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

      temp= remaining[i];
      remaining[j] = remaining[j+1];
      remaining[j+1]= temp;
    }
  }
 }
}
int main(){
 int n;
 printf("Enter the number of processes:");
 scanf("%d", &n);
 int proc[n], burst[n], tickets[n], remaining[n];
```

```
int total_tickets = 0;
printf("Enter the CPU burst times and number of
    tickets for each process: \n");
for (int i=0; i<n; i++) {
    proc[i] = i+1;
    printf ("Process %d burst time : ", i+1);
    scanf("%d", &burst[i]);
    printf("process %d tickets : ", i+1);
    scanf ("%d", &tickets[i]);
    remaining[i] = burst[i];
    total_tickets += tickets[i];
}
sort_by_burst(proc, burst, tickets, remaining, n);
printf("\n Proportional share scheduling (Lottery
    Scheduling) : \n");
printf ("PID \t Burst \t Tickets \n");
for(int i=0; i<n; i++) {
    printf("%d \t %d \t %d \n", proc[i], burst[i],
        tickets[i]);
}
int time=0;
while(1) {
    int active_processes = 0;
    for(int i=0; i<n; i++) {
        if(remaining[i]>0) {
            active_processes++;
        }
    }
    if (active_processes == 0) {
        break;
    }
    int winning_ticket = rand() % total_tickets;
```

```
int cumulative_tickets = 0;
int selected_process = -1;
for (int i=0; i<n; i++) {
    if (remaining[i] > 0) {
        cumulative_tickets += tickets[i];
        if (winning_ticket < cumulative_tickets) {
            selected_process = i;
            break;
        }
    }
}
if (selected_process != -1) {
    printf("%d ms: Process %d is running.\n", time,
    proc[selected_process]);
    remaining[selected_process]--;
} else {
        printf("%d ms: CPU is idle.\n", time);
}
time++;
}
printf("Scheduling completed.\n");
}
```

#) Output:

```
Enter the number of processes: 3
Enter the CPU burst times and number of tickets for
each process:
Process 1 burst time: 5
Process 1 tickets: 3
Process 2 burst time: 3
Process 2 tickets: 2
Process 3 burst time: 7
Process 3 tickets: 4
```

Proportional share scheduling (Lottery Scheduling):

PID	Burst	Tickets
2	3	2
1	5	3
3	7	4

0ms: process 2 is running
1ms: " 3 " "
2ms: " 2 " "
3ms: " 3 " "
4ms: " 3 " "
5ms: " 3 " "
6ms: " 2 " "
7ms: " 3 " "
8ms: " 3 " "
9ms: " 1 " "
10ms: " 3 " "
11ms: cpu is Idle
12ms: "
13ms: "
14ms: "
15ms: "
16ms: "
17ms: process 1 is running
18ms: cpu is idle
19ms: process 1 is running
20ms: cpu is idle
21ms: "
22ms: "
23ms: "
24ms: "
25ms: "
26ms: "
27ms: "
28ms: process 1 is runnin
29ms: "

Program 5:

Date: 12/06/2024:-

Page 33

Producer Consumer Problem in C using Semaphores:

→

```
#include <stdio.h>
#include <stdlib.h>
int mutex = 1, full = 0, empty = 7, x = 0;

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 Consumer consumes items %d", x);
  x--;
  mutex = signal(mutex);
}

int wait(int s){
  return(--s);
}

int signal(int s){
  return(++s);
}
int main()
  {
```

```
int n;
void producer();
void consumer();
int wait(int);
int signal(int);
printf("\n1. Producer \n2. Consumer \n3. Exit");
while(1) {
    printf("\n Enter your choice:");
    scanf("%d", &n);
    switch(n) {
        case 1:
            if ((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:
            printf("\n Number of items remaining in
                buffer : %d \n", n);
            exit(0);
            break;
    }
}
}
```

Date
Page

Output:

1. Producer
2. Consumer
3. Exit

Enter your choice: 1
Producer produces the item 1
Enter your choice: 1
Producer produces the item 2
Enter your choice: 1
Producer produces the item 3
Enter your choice: 1
Producer produces the item 4
Enter your choice: 1
Producer produces the item 5
Enter your choice: 1
Producer produces the item 6
Enter your choice: 1
Producer produces the item 7
Enter your choice: 1
Buffer is full!!
Enter your choice: 2
Consumer consumes item 7
Enter your choice: 2
Consumer consumes item 6
Enter your choice: 3
Number of items remaining in buffer: 5

12/6

Program 6:
Date: 19/06/2024:-

Dining - Philosopher's problem:-

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

#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (phnum+N-1)%N
#define RIGHT (phnum+1)%N

int state[N];
int phil[N] = {0, 1, 2, 3, 4};
pthread_mutex_t mutex;
```

→
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define MAX_PHILOSOPHERS 10
#define MAX_CHOPSTICKS MAX_PHILOSOPHERS
void dine (int phil_position);
pthread_t philosophers[MAX_PHILOSOPHERS];
pthread_mutex_t chopsticks[MAX_CHOPSTICKS]

int main() {
    int num_total_philosophers, num_hughy_philosophers,
    i, status;
    int philosopher_positions[MAX_PHILOSOPHERS];
```

```
printf("Enter the total number of philosophers (maximum %d)
    :", MAX_PHILOSOPHERS);
if (num_total_philosophers < 2 || num_total_philosophers
  > MAX_PHILOSOPHERS) {
  printf("Invalid number of philosophers. Must be between
    2 and %d.\n", MAX_PHILOSOPHERS);
  return 1;
}

printf("Enter the no. of philosphers who are hungry
    (maximum %d):", &num_~~total~~hungry_philosphers);
  return 1;
  }
printf("Enter positions of %d hungry philosophers (1 to
      %d):\n", num_hungry_philosophers, num_total
    philosophers);
for(i=0; i<num_hungry_philosphers; i++) {
  scanf("%d", &philosopher_positions[i]);
  }
}
for(i=0; i< num_total_philosophers; i++) {
  status = pthread_mutex_init(&chopsticks[i], NULL);
  if(status != 0) {
    printf(stderr, "Error");
  }
}
for(i=0; i< num_hungry_philosophers; i++) {
  status= pthread_join(philosophers[i], NULL);
  if(status != 0) {
    printf(stderr, "Error");
  }
}
```

```
void dine (int phil_position) {
    printf ("\n philosopher %d is thinking \n", phil_position);
    pthread_mutex_wait ( & chopsticks [phil_position - 1]);
    pthread_mutex_lock ( & chopsticks [phil_position %
                                MAX_CHOPSTICKS]);
    printf (" philosopher %d is eating \n", phil_position);
    pthread_mutex_unlock ( & chopsticks [phil_position - 1]);
    pthread_mutex_unlock ( & chopsticks [phil_position %
                                MAX_CHOPSTICKS]);
    printf ("philosopher %d finished eating \n", phil_
                                position);
}
```

(##). Output:

```
Enter total no. of philosophers : 5
Enter no. of philosophers who are hungry : 3
Enter position of 3 hungry philosophers (1 to 5): 2 4 5
philosopher 2 is thinking
philosopher 2 is eating
philosopher 4 is thinking
philosopher 4 is eating
philosopher 5 is thinking
philosopher 4 finished eating
philosopher 2 finished eating
philosopher 5 finished eating
```

Program: 7
Date: 19/08/24

(#). Banker's Problem:-

→
```
#include<stdio.h>
#include<stdbool.h>
#define MAX_PROCESS 10
#define MAX_RESOURCES 10

void calculateNeed(int need[MAX_PROCESSES][MAX_
RESOURCES], int max[MAX_PROCESSES][MAX_RESOURCES],
int allot[MAX_PROCESSES][MAX_RESOURCES], int np,
int nr){
    for(int i=0; i<np; i++)
        for(int j=0; j<nr; j++)
            need[i][j]= max[i][j] - allot[i][j];
}
bool isSafe(int processes[], int avail[], int max[][MAX_
RESOURCES], int allot[][MAX_RESOURCES], int np, int nr)
{
    for (int i=0; i<nr; i++)
        work[i]= avail[i];
    int count=0;
    while(count<np){
        bool found= false;
        for(int p=0; p<np; p++){
            if(finish[p]==0){
                int j;
                for(j=0; j<nr; j++)
                    if(need[p][j]>work[j])
                        break;
                if(j==nr){
                    for(int k=0; k<nr; k++)
                        work[k]+= allot[p][k];
                }
            }
        }
    }
```

```
if (found == false) {
    printf("System is not in safe state");
}
}
int main() {
    int np, nr;
    printf("Enter number of processes: ");
    scanf("%d", &np);
    printf("Enter no. of resource types: ");
    scanf("%d", &nr);

    int max[MAX_PROCESSES][MAX_RESOURCES];
    printf("Enter maximum resource matrix: \n");
    for(int i=0; i<np; i++) {
        printf("Process %d: ", i);
        for(int j=0; j<nr; j++)
            scanf("%d", &max[i][j]);
    }
    int allot[MAX_PROCESSES][MAX_RESOURCES];
    for(i=0; i<np; i++) {
        printf("Process %d: ", i);
        for(int j=0; j<nr; j++)
            scanf("%d", &allot[i][j]);
    }
    isSafe(processes, avail, max, allot, np, nr);
    return 0;
}
```

Date
Page

(##)- Output:

```
Enter number of processes: 5
Enter number of resource types: 3
Enter available resources: 3 3 2
Enter maximum resource matrix:
Process 0: 7 5 3
" 1: 3 2 2
" 2: 9 0 2
" 3: 2 2 2
" 4: 4 3 3
Enter allocation resource matrix:
Process 0: 0 1 0
Process 1: 2 0 0
" 2: 3 0 2
" 3: 2 1 1
" 4: 0 0 2
System is in safe state.
Safe sequence: 1 3 4 0 2
```

19/6

Program 8:
Date: 03/07/2024

Deadlock detection:-

```
#include <stdio.h>
#define MAX_PROCESSES 10
#define MAX_RESOURCES 10
void initialize() {
    int i, j;
    printf("Enter no. of processes: ");
    scanf("%d", &n_processes);
    printf("Enter no. of resources: ");
    scanf("%d", &n_resources);
    printf("Enter allocation matrix");
    for(i=0; i<n_resources; i++) {
        for(j=0; j<n_resources; j++) {
            scanf("%d", &allocation[i][j]);
        }
    }
    printf("Enter maximum need matrix:", n_proc);
    for(i=0; i<n_processes; i++) {
        for(j=0; j<n_resources; j++) {
            scanf("%d", &max_need[i][j]);
        }
    }
    printf("Enter available resource vector", n);
    for(j=0; j<n_resources; j++) {
        scanf("%d", &available[j]);
    }
}
void detect_deadlock() {
    int i, j, k;
    for(j=0; j<n_resources; j++) {
        work[j] = available[j];
    }
```

Date
Page

```
for(i=0; i<n_processes; i++) {
  finish[i]=0;
}
int found;
do {
   found=0;
   for(i=0; i<n_processes; i++) {
     if(!finish[i]) {
       int can_allocate=1;
       for(j=0; j<n_resources; j++) {
         if(max_need[i][j] - allocation[i][j] > work[j]) {
            can_allocate=0;
            break;
         }
       }
     if(can_allocate) {
        for(j=0; j<n_process; j++) {
          work[j] += allocation[i][j];
        }
        finish[i] = 1;
        safe_sequence[safe_sequence_count++] = i;
        found=1;
     }
   }
}
}while(found);
for(i=0; i<n_process; i++) {
    if(!finish[i]) {
       printf("Deadlock detected!\n");
       return;
    }
  }
```

```
printf("No deadlock detected.\n Safe sequence:");
for(i=0; i< n- processes; i++) {
  printf("%d ", safe- sequence[i]);
 }
  printf("\n");
 }
int main() {
    initialize();
    detect_deadlock();
    return 0;
}
```

##. Output:

```
Enter number of processes: 5
Enter number of resources: 3
Enter allocation matrix:
0  1  0
2  0  0
3  0  3
2  1  1
0  0  2
Enter maximum need matrix:
0  0  0
2  0  2
0  0  0
1  0  0
0  0  2
Enter Available resource vector: 0 0 0
No deadlock detected.
Safe Sequence: 0 2 3 4 1
```

3/4

Program: 9
Date: 03/07/2024

Date 03/07/24
Page

(#). Contiguous 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:\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-",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;
      }
    }
```

Date
Page

```
if (coorst != -1) {
    ff[i] = coorst;
    allocated [worst] = 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]);
  }
}
int main() {
 int nb, nf, choice;
 printf(" Memory Management Scheme");
 printf(" \n Enter no. of blocks:");
 scanf("%d", &nb);
 printf(" Enter no. 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]);
 }
```

```
printf("\n\t Memory First fit \n");
firstfit(nb, nf, b, f);
printf("\n\t Best fit \n");
bestfit(nb, nf, b, f);
printf("\n\t Worst fit \n");
worstfit(nb, nf, b, f);
return 0;
}
```

(#). Output:-

→

```
Memory Management scheme
Enter the number of blocks: 5
Enter the number of files: 4
Enter the size of the blocks:
Block 1: 500
Block 2: 250
Block 3: 350
Block 4: 100
Block 5: 150
Enter the size of the files:
File 1: 320
File 2: 150
File 3: 100
File 4: 450
First Fit
```

File-no:	File_size:	Block-no:	Block-size:
1	320	1	500
2	150	2	250
3	100	3	350
4	450	-	-

Date
Page

Best Fit

File-no:	File-size:	Block-no:	Block-size:
1	320	3	350
2	150	5	150
3	100	4	100
4	450	1	500

Worst Fit

File_no:	File-size:	Block-no:	Block-size:
1	320	1	500
2	150	3	350
3	100	2	250
4	450	-	-

Program: 10

Date:- 10/07/2024:-

#). Page Replacement Algorithms:-

→
```
#include<stdio.h>
#include<stdbool.h>
#include<limits.h>
void fifo(int F[], int n, int P[], int m)
{
  int index=0;
  int P-fault=0;
  for(int i=0; i<m; i++)
  {
    bool found= false;
    for(int j=0; j<n; j++)
    {
      if (F[j]==P[i])
      {
        found= true;
        break;
      }
    }
    if(! found)
    {
      F[index]= P[i];
      index= (index+1) %n;
      P-fault++;
    }
  }
  printf("FIFO Page faults: %d \n", P-fault);
}
void fru(int F[], int n, int P[], int m)
{
  int LU[n];
  int P-fault=0;
```

Date
Page

```
int count=0;
for(int i=0; i<n; i++)
{
 F[i] = -1;
 LU[i] = -1;
}
for(int i=0; i<m; i++)
 {
    int j;
    bool found= false;
    for(j=0; j<n; j++)
     {
       if (F[j] == P[i])
        {
          found= true;
         LU[j] = count++;
          break;
       }
     }
    if (! found)
    {
    int lru- index=0;
    for(j=1 ; j<n; j++)
     {
       if(LU[j] <LU[lru-index])
        {
          lru- index =j;
       }
     }
    F[lru-index] = P[i];
    LU[lru-index]= count++;
    P-fault++;
  }
}
```

```
printf("LRU Page Faults: %d \n", P-fault);
}
void optimal(int F[], int n, int P[], int m)
{
 int P-fault=0;
 for(int i=0; i<n; i++)
  {
    F[i] = -1;
 }
 for(int i=0; i<m; i++)
 {
  bool found=false;
 for (int j=0; j<n; j++)
   {
    if (F[j]==P[i])
      {
        found=true;
        break;
      }
   }
  if (!found)
  {
   int replace_i = -1;
   int far = i+1;
   for(int j=0; j<n; j++)
   {
     int k;
    for(k=i+1; k<m; k++)
    {
      if(F[j]==P[k])
        {
          if (k> far)
```

Date
Page

```
{
far=k;
replace-i=j;
}
break;
}
}
if (k==m)
{
replace-i=j;
break;
}
}
if (replace-i==-1)
{
replace-i=0;
}
F[replace-i]=P[i];
P-fault++;
}
}
printf("optimal page faults: %d \n", P-fault);
}
int main() {
int n;
int m;
printf("Enter the number of frames:");
scanf("%d", &n);
printf("Enter the length of reference string:");
scanf("%d", &m);
int P[m];
printf("Enter the reference string:");
```

```
for( int i=0;  i<M; i++)
  {
    scanf("%d", &P[i]);
  }
int F[n];
for(int i=0; i<n; i++)
  {
    F[i] = -1;
  }
fifo(F, n, P, M);
lru(F, n, P, M);
optimal(F, n, P, M);
return 0;
}
```

##. Output:

Enter the Number of Frames: 3
Enter the length of reference string: 20
Enter the reference string: 0 9 0 1 8 1 8 7 8 7 1 2 8 2 7 8 2 3 8 3
FIFO Page faults: 8
LRU Page faults: 9
Optimal Page faults: 7.

10/7 compute!