INDEX

1BM23CS412

Name Varsha V. N. Sub. OS Lab BK

Std.: 4th C Div. Roll No.

Telephone No. E-mail ID.

Blood Group. Birth Day.

Sr.No.	Title	Page No.	Sign,/Remarks
1.	FCFS & SJF (Pre & NonPre)	1	
2.	Priority (Pre & NonPre) & Round Robin	13	
3.	Multilevel Scheduling	23	
4.	a) Rate Monotonic	27	
4.	b) Earliest Deadline First	30	
4.	c) Proportional scheduling	33	
5.	Producer consumer Problem	36	
6.	Dining Philosophers	39	
7.	Banker's Algorithm	44	
8.	Deadlock Detection	48	
9.	Contiguous Mem Allocation	51	
10.	Page Replacement	58	

10/1 Completed

8/5/24 Lab - 1

PAGE NO: 1
DATE:

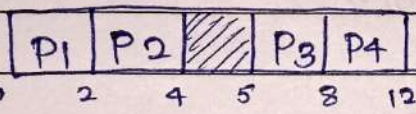
* Scheduling Algorithms

1) FCFS - First Come First Serve Scheduling

1.

Process ID	Arrival Time(ms)	Burst Time(ms)	Completion Time(ms)	Turn Around Time	Waiting Time (ms)
P1	0	2	2	2	0
P2	1	2	4	3	1
P3	5	3	8	3	0
P4	6	4	12	6	2

→ Gantt Chart



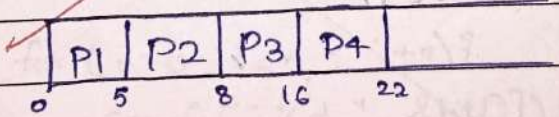
Average Turn Around Time = (2+3+3+6)/4 = 3.5 ms

Average Waiting Time = (0+1+0+2)/4 = 0.7 ms

2.

PID	AT	BT	CT	TAT	WT
P1	0	5	5	5	0
P2	1	3	8	7	4
P3	2	8	16	14	6
P4	3	6	22	19	13

→ Gantt chart



Average TAT = 11.25

Average WT = 5.75

DATE :

* Code:

```
#include <stdio.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 = 0;

    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_tat += tat[i];
    }

    for (int i=0; i<n; i++){
        wt[i] = tat[i] - bt[i];
        total_wt += wt[i];
    }
```

```
    printf("\nProcess \t Arrival Time \t Burst Time \t Completion Time \t Turn Around Time \t Waiting Time");
    for (int i=0; i<n; i++){
        printf("%d \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("\nAverage waiting time : %.2f",
                (float) total_wt/n);
    printf("\nAverage turnaround time: %.2f",
                (float) total_tat/n);
}

int main(){
    int n, i;
    printf("Enter the number of processes: ");
    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]);
    }

    fcfs(n, at, bt);
    return 0;
}
```

PAGE NO :
DATE :

* Output:

Enter the number of processes: 4

Enter the arrival times:

0

1

5

6

Enter the burst times:

2

2

3

4

Process	Arrival Time	Burst Time	Completion Time	Turnaround 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

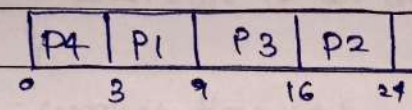
Average turnaround time: 3.50

2) SJF - Shortest Job First Scheduling

1.

PID	AT	BT	CT	TAT	WT
P1	0	6	9	9	3
P2	0	8	24	24	16
P3	6	7	16	16	9
P4	0	3	3	3	0

→ Gantt chart



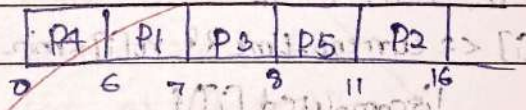
Average Turn around Time = 13

Average Waiting Time = 7

2.

PID	AT	BT	CT	TAT	WT
P1	2	1	7	5	4
P2	1	5	16	15	10
P3	4	1	8	4	3
P4	6	6	6	6	0
P5	2	3	11	9	6

→ Gantt Chart



Average Turn Around Time = 7.8

Average Waiting Time = 4.6

PAGE NO :
DATE :

* Code:

```
#include<stdio.h>
#define MAX 10

void sjf(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 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("\nProcess\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 %d\t\t %d\t\t %d\t\t %d\n", i+1, at[i],
        bt[i], ct[i], tat[i], wt[i]);
}
}

int main(){
    int n, i;
    printf("Enter the number of processes: ");
    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_non_preemptive(n, at, bt);
    return 0;
}
```

PAGE NO :
DATE :

* Output:

Enter the number of processes: 4

Enter the arrival time:

0

0

0

0

Enter the burst time:

6

8

7

3

Process	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

Average waiting Time: 7.00

Average Turn around Time: 13.00

2) SJF - Shortest Job First - Preemptive Scheduling.

* Code:

```
#include <stdio.h>
#define MAX 10

int find_min(int arr[], int n){
    int min = arr[0];
    int index=0;
    for (int i=1; i<n; i++){
        if arr[i] <min){
            min = arr[i];
            index = i;
        }
    }
    return index;
}

void sjf_preemptive(int n, int at[], int bt[]){
    int ct[MAX] = {0};
    int tat[MAX]={0};
    int wt[MAX] = {0};
    int rt[MAX];

    int total_wt=0;
    int total_tat=0;

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

    int current_time=0;
    int completed_processes=0;
```

PAGE NO :
DATE :

```
while(completed_processes<n){
    int available_processes[MAX];
    int available_count=0;
    for(int i=0; i<n; i++){
        if(at[i]<=current_time && rt[i]>0){
            available_processes[available_count]=i;
            available_count++;
        }
    }
    if(available_count==0){
        current_time++;
        continue;
    }
    int sj_index=available_processes[find_min(rt,
                                available_count)];
    rt[shortest_job_index]--;
    current_time++;
    if(rt[sj_index]==0){
        completed_processes++;
        ct[shortest_job_index]=current_time;
        tat[shortest_job_index]=tat[sj_index]-at[sj_index];
        wt[sj_index]=tat[sj_index]-bt[sj_index];
        total_wt+=wt[sj_index];
        total_tat+=tat[sj_index];
    }
}
printf("\nProcess\tArrival Time\tBurst Time\t
    Completion Time\tTurnaround Time\t Waiting Time\n");
for(int i=0; i<n; i++){
    printf("%d\t%d\t%d\t%d\t%d\t%d\n",i+1,
        at[i], bt[i], ct[i], tat[i], wt[i]);
}
```

```
    printf("\nAverage waiting time: %.2f",
                                (float) total_wt / n);
    printf("\nAverage turnaroundtime : %.2f",
                                (float) total_tat / n);
}

int main(){
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    int at[MAX], bt[MAX];
    printf("Enter the arrival time:\n");
    for (int i=0; i<n; i++){
        scanf("%d", &at[i]);
    }

    printf("Enter the burst time:\n");
    for (int i=0; i<n; i++){
        scanf("%d", &bt[i]);
    }

    sjf_preemptive(n, at, bt);
    return 0;
}
```

* Output:

```
Enter the number of processes:5
Enter the arrival time:
2 1 4 0 2
Enter the burst time:
1 5 1 6 3
```

PAGE NO :
DATE :

Process	AT	BT	CT	TAT	WT
1	2	1	3	1	0
2	1	5	7	6	1
3	4	1	8	4	3
4	0	6	13	13	7
5	2	3	16	14	11

Average waiting time: 4.40

Average turnaround time: 7.60

3) Priority Scheduling Preemptive and Non-preemptive

```
#include <stdio.h>

typedef struct {
    int id;
    int at;
    int bt;
    int priority;
    int rt;
} Proc;

void swap(Proc *a, Proc *b){
    Proc temp = *a;
    *a = *b;
    *b = temp;
}

void sort_by_priority(Proc p[], int n){
  for (int i = 0; i < n-1; i++){
    for (int j=0; j < n-i-1; j++){
      if (p[j].priority > p[j+1].priority){
        swap(&p[j], &p[j+1]);
      }
    }
  }
}

void priority_preemptive(Proc p[], int n){
    int wt[n], tat[n];
    int total_time = 0;
    int completed = 0;
    while (completed < n){
        int highest_priority_index = -1;
```

```
int highest_priority = -1;
for (int i=0; i<n; i++){
    if (p[i].at <= total_time && p[i].priority >
        highest_priority && p[i].rt >0){
        highest_priority_index = i;
        highest_priority = p[i].priority;
    }
}
if (highest_priority_index == -1){
    total_time++;
    continue;
}

p[highest_priority_index].rt--;
if (p[highest_priority_index].rt ==0){
    completed++;
    wt[highest_priority_index] = total_time + 1 -
    p[highest_priority_index].at - p[highest_priority_index].bt;
    if (wt[highest_priority_index] <0){
        wt[highestpriority_index] = 0;
    }
    tat[highest_priority_index] = wt[highest_priority_index]
    + p[highest_priority_index].bt;
}
total_time++;
}

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

```
printf("\nPreemptive Priority scheduling:\n");
printf("ID \t Arrival Time \t Burst Time \t Priority \t
    Waiting Time \t Turnaround Time \n");
for (int i=0; i<n; i++){
    printf("%d \t %d \t %d \t \t %d \t \t %d \n",
    p[i].id, p[i].at, p[i].bt, p[i].priority, wt[i],
                                    tat[i]);
}
printf("Avg waiting time: %.2f\n", avg_wt);
printf("Avg Turnaround time: %.2f \n", avg_tat);
}

void priority_non_preemptive (Proc p[], int n) {
    int wt[n], tat[n];
    sort_by_priority (p, n);
    int total_time = p[0].at;
    for (int i=0; i<n; i++){
        wt[i] = total_time - p[i].at;
        if (wt[i]<0){
            wt[i] = 0;
            total_time = p[i].at;
        }
        tat[i] = wt[i] + p[i].bt;
        total_time += p[i].bt;
    }
    float avg_wt = 0, avg_tat = 0;
    for (int i=0; i<n; i++){
        avg_wt += wt[i];
        avg_tat += tat[i];
    }
    avg_wt /= n;
    avg_tat /= n;
    printf("\n Non-preemptive Priority Scheduling:\n");
    printf("ID \t Arrival Time \t Burst time \t Priority \t
        Waiting Time \t Turnaround Time \n");
```

```
for (int i=0; i<n; i++){
  printf ("%d\t %d\t %d\t\t %d\t\t %d\n",
    p[i].id, p[i].at, p[i].bt, p[i].priority, wt[i],
    tat[i]);
}
printf("Avg Waiting Time: %.2f\n", avg_wt);
printf("Avg Turnaround Time: %.2f\n", avg_tat);
}

int main(){
  int n;
  printf("Enter the number of processes: ");
  scanf("%d", &n);
  Proc p[n];
  printf("Enter arrival time, burst time, and priority
      for each process:\n");
  for (int i=0; i<n; i++){
    printf("Process %d:\n", i+1);
    p[i].id = i+1;
    printf("Process Arrival Time: ");
    scanf("%d", &p[i].at);
    printf("Burst Time: ");
    scanf("%d", &p[i].bt);
    printf("Priority: ");
    scanf("%d", &p[i].priority);
    p[i].rt = p[i].bt;
  }
  priority_non_preemptive(p,n);
  priority_preemptive(p,n);

  return 0;
}
```

Output:

Enter the number of processes: 4

Enter arrival time, burst time, and priority for each processes:

Process 1:

Arrival Time : 0

Burst Time: 5

Priority: 10

Process 2:

Arrival Time: 1

Burst Time: 4

Priority: 20

Process 3:

Arrival Time: 2

Burst Time: 2

Priority: 30

Process 4:

Arrival Time: 4

Burst Time: 1

Priority: 40

- Non-Preemptive Priority scheduling:

ID	Arrival Time	BT	Priority	WT	TAT
1	0	5	10	0	5
2	1	4	20	4	8
3	2	2	30	7	9
4	4	1	40	7	8

Avg. waiting time : 4.50

Avg Turnaround Time: 7.50

- Preemptive Priority Scheduling:

ID	Arrival Time	BT	Priority	WT	TAT
1	0	5	10	7	12
2	1	4	20	3	7
3	2	2	30	0	2
4	4	1	40	0	1

Avg WT : 2.50 Avg TAT : 5.50

PAGE NO :
DATE :

4) Round Robin Scheduling

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

struct queue {
  int pid;
  struct queue * next;
};

struct queue * rq = NULL;
struct queue * create (int p){
   struct queue * nn = malloc (sizeof (struct queue));
   nn -> pid = p;
   nn -> next = NULL;
   return nn;
}

void enqueue (int p){
   struct queue* nn = create (p);
   if (rq == NULL)
      rq = nn;
   else {
      struct queue * temp = rq;
      while (temp -> next != NULL)
         temp = temp -> next;
      temp -> next = nn;
   }
}

int dequeue (){
   int x = 0;
   if (rq == NULL)
      return x;
   else{
```

```
    struct queue* temp = rq;
    x = temp -> pid;
    rq = rq -> next;
    free (temp);
  }
  return x;
}

void printq(){
  struct queue* temp = rq;
  while (temp != NULL){
    printf ("%d\t", temp -> pid);
    temp = temp -> next;
  }
  printf ("\n");
}

void swap (int *a, int *b){
  *a = *a + *b;
  *b = *a - *b;
  *a = *a - *b;
}

void sort (int *pid, int *at, int *bt, int n){
  for (int i=0; i<n; i++){
    for (int j=0; j<n; j++){
      if (at[i]<at[j]){
        swap (&at[i], &at[j]);
        swap (&bt[i], &bt[j]);
        swap (&pid[i], &pid[j]);
      }
    }
  }
}
```

PAGE NO :
DATE :

```
int main(){
  int n, t, x=1;
  printf("Enter the number of processes: ");
  scanf("%d", &n);
  printf("Enter the time quantum: ");
  scanf("%d", &t);
  int pid[n], at[n], bt1[n], ct[n], tat[n], wt[n], bt2[n], rt[n];
  for (int i=0; i<n; i++){
    printf("Enter arrival time and burst time: ");
    scanf("%d%d", &at[i], &bt1[i]);
    pid[i] = i+1;
  }
  sort(pid, at, bt1, n);
  enqueue(pid[0]);
  for (int i=0; i<n; i++){
    bt2[i] = bt1[i];
    rt[i] = -1;
  }
  int count = 0;
  int ctvar = at[0];
  while (count != n){
    int curp = rq->pid;
    int curi = 0;
    for (int i=0; i<n; i++){
      if (pid[i] == curp){
        curi = i;
        break;
      }
    }
    if (rt[curi] == -1){
      rt[curi] = ctvar - at[curi];
    }
    if (bt2[curi] <= t){
```

```
        ctvar += bt2[curi];
        bt2[curi] = 0;
    }
    else{
        ctvar += t;
        bt2[curi] -= t;
    }
    while(at[x] <= ctvar && x<n){
        enqueue(pid[x]);
        x += 1;
    }
    if(bt2[curi] > 0)
        enqueue(pid[curi]);
    if(bt2[curi] == 0){
        count += 1;
        ct[curi] = ctvar;
    }
    dequeue();
}
for(int i=0; i<n; i++){
    tat[i] = ct[i] - at[i];
    wt[i] = tat[i] - bt1[i];
}
float avg_tat = 0;
float avg_wt = 0;
for(int i=0; i<n; i++){
    avg_tat += tat[i];
    avg_wt += wt[i];
}
printf("pid\tat\tbt\tct\ttat\twt\trt\n\n");
for(int i=0; i<n; i++){
    printf("%d\t%d\t%d\t%d\t%d\t%d\t%d", pid[i], at[i],
        bt1[i], ct[i], tat[i], wt[i], rt[i]);
    printf("\n");
}
```

printf ("\nAverage Turn around Time: %.f", avg-tat);
printf ("\nAverage waiting time : %.f", avg-wt);
return 0;
}

* Output:

Enter the number of processes : 5
Enter the time quantum: 2
Enter arrival time and burst time: 0 5
_____ : 1 3
_____ : 2 1
_____ : 3 2
_____ : 4 3

pid	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 TAT: 8.600000
Average WT: 5.800000

25/6/24 Lab-3

5) Multilevel Scheduling.

```
#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 simulateFCFS(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;
    for (int i=0; i<n; i++){
        if (c >= at[i])
            c += bt[i];
        else
            c = at[i] + bt[i];
        ct[i] = c;
    }
```

PAGE NO :
DATE :

```
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 ("PID\tAT\t BT\tCT\t TAT \t tWT\n");
for (int i=0; i<n; i++){
    printf ("%d\t %d\t %d\t %d\t %d\t %d\n", proc_id[i],
        at[i], ct[i], bt[i], tat[i], wt[i]);
    ttat += tat[i];
    twt += wt[i];
}

printf ("Average Turnaround Time: %.2lfms\n",
        ttat/n);
printf ("Average Waiting Time: %.2lfms\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 proces
```

```
%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];
    sys_count++;
} 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_bt[i];
    }
}
printf("\n User Processes Scheduling:\n");
simulateFCFS(user_proc_id, user_at, user_bt, user_count,
system_end_time);
}
```

PAGE NO :
DATE :

* Output:

Enter number of processes: 4

Enter arrival time, burst time and type (0 for system, 1 for user) for process 1 : 0 2 0

——"—— process 2 : 0 1 1

——"—— process 3 : 0 5 0

——"—— process 4 : 0 3 1

System Processes Scheduling:

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

Average Turnaround Time: 4.50 ms

Average Waiting Time: 1.00 ms

User Processes Scheduling:

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

Average Turnaround Time: 9.50 ms

Average Waiting Time: 7.50 ms

5/6/24 Lab-4 a)

6) Rate Monotonic Scheduling

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_TASKS 10

typedef struct {
    int Ti;
    int Ci;
    int deadline;
    int RT;
    int id;
} Task;

void Input (Task tasks[], int *n_tasks) {
    printf ("Enter number of tasks: ");
    scanf ("%d", n_tasks);
    for (int i=0; i<*n_tasks; i++) {
        tasks[i].id = i+1;
        printf ("Enter Ti of task %d: ", i+1);
        scanf ("%d", &tasks[i].Ti);
        printf ("Enter execution time of task %d:", i+1);
        scanf ("%d", &tasks[i].Ci);
        tasks[i].deadline = tasks[i].Ti;
        tasks[i].RT = tasks[i].Ci;
    }
}

int compare_by_period (const void *a, const void *b) {
    return ((Task*)a)->Ti - ((Task*)b)->Ti;
}

void RMS (Task tasks[], int n_tasks, int time_frame) {
    qsort (tasks, n_tasks, sizeof (Task), compare_by_period);
```

```
printf("\n Rate Monotonic Scheduling:\n");
for (int time=0; time<time_frame; time++){
    int s_task = -1;
    for (int i=0; i<n_tasks; i++){
        if (time % tasks[i].T; == 0){
            tasks[i].RT = tasks[i].C;
        }
        if (tasks[i].RT>0 && (s_task == -1 || tasks[i].T<
                              tasks[s_task].T;)){
            s_task = i;
        }
    }
    if (s_task != -1){
        printf("Time %d: Task %d \n", time, tasks[s_task]
        tasks[s_task].RT--;
    }else{
        printf("Time %d: Idle \n", time);
    }
}
}

int main(){
    Task tasks[MAX_TASKS];
    int n_tasks;
    int time_frame;
    Input(tasks, &n_tasks);
    printf("Enter time frame for simulation:");
    scanf("%d", &time_frame);
    RMS(tasks, n_tasks, time_frame);
    return 0;
}
```

* Output:

Enter number of tasks: 3
Enter Ti of task 1: 20
Enter execution time of task 1: 3
Enter Ti of task 2: 5
Enter execution time of task 2: 2
Enter Ti of task 3: 10
Enter execution time of task 3: 2
Enter time frame for simulation: 20

Rate Monotonic Scheduling-
Time 0: Task 2
Time 1: Task 2
Time 2: Task 3
Time 3: Task 3
Time 4: Task 1
Time 5: Task 2
Time 6: Task 2
Time 7: Task 1
Time 8: Task 1
Time 9: Idle
Time 10: Task 2
Time 11: Task 2
Time 12: Task 3
Time 13: Task 3
Time 14: Idle
Time 15: Task 2
Time 16: Task 2
Time 17: Idle
Time 18: Idle
Time 19: Idle

5/6/24 Lab-4 b)

PAGE NO :
DATE :

7) Earliest Deadline Scheduling

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_TASKS 10

typedef struct{
  int Ti;
  int Ci;
  int deadline;
  int RT;
  int n_deadline;
  int id;
}Task;

void Input (Task tasks[], int *n_tasks){
  printf("Enter number of tasks: ");
  scanf("%d", n_tasks);
  for (int i=0; i<*n_tasks; i++){
    tasks[i].id= i+1;
    printf("Enter Ti of task %d: ", i+1);
    scanf("%d", &tasks[i].Ti);
    printf("Enter execution time of task %d: ", i+1);
    scanf("%d", &tasks[i].Ci);
    printf("Enter deadline of task %d: ", i+1);
    scanf("%d", &tasks[i].deadline);
    tasks[i].RT= tasks[i].Ci;
    tasks[i].n_deadline = tasks[i].deadline;
  }
}

void EDF (Task tasks[], int n_tasks, int time_frame){
  printf("\nEarliest-Deadline First Scheduling:\n");
  for (int time =0; time< time_frame; time++){
```

```
    int s_task = -1;
    for (int i=0; i<n_tasks; i++) {
        if (time % tasks[i].T == 0) {
            tasks[i].RT = tasks[i].C;
            tasks[i].n_deadline = time + tasks[i].deadline;
        }
    }

    for (int i=0; i<n_tasks; i++) {
        if (tasks[i].RT > 0 && (s_task == -1 || tasks[i].n_dead
                < tasks[s_task].n_deadline)) {
            s_task = i;
        }
    }

    if (s_task != -1) {
        printf("Time %d: Task %d\n", time, tasks[s_task].i
        tasks[s_task].RT--;
    } else {
        printf("Time %d: Idle\n", time);
    }
  }
}

int main() {
    Task tasks[MAX_TASKS];
    int n_tasks;
    int time_frame;
    Input(tasks, &n_tasks);
    printf("Enter time frame for simulation: ");
    scanf("%d", &time_frame);
    EDF(tasks, n_tasks, time_frame);
    return 0;
}
```

PAGE NO :
DATE :

* Output:

Enter Ti of task 3:10

Enter number of tasks:3

Enter Ti of task 1:20

Enter execution time of task 1:3

Enter deadline of task 1: 7

Enter Ti of task 2: 5

Enter ET of task 2: 2

Enter deadline of task 2:4

Enter Ti of task 3:10

Enter ET of task 3: 2

Enter deadline of task 3: 8

Enter time frame for simulation: 20

Earliest - Deadline First scheduling:

Time 0 : Task 2

Time 1 : Task 2

Time 2 , Task 1

Time 3 : Task 1

Time 4 : Task 1

Time 5 : Task 3

Time 6 : Task 3

Time 7 : Task 2

Time 8 : Task 2

Time 9 : Idle

Time 10 : Task 2

Time 11 : Task 2

Time 12 : Task 3

Time 13 : Task 3

Time 14 : Idle

Time 15 : Task 2

Time 16 : Task 2

Time 17 : Idle

Time 18 : Idle

Time 19 : Idle

5/6/24 Lab - 4 c)

8) Proportional Scheduling

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

struct Task {
    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 += tasks[i].tickets;
    }
    srand(time(NULL));
    int current_time = 0;
    int completed_tasks = 0;
    printf("process scheduling :\n");
    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 (winning_ticket < cumulative_tickets){
                printf(" Time %d - %d : Task %d is running\n
                    current_time, current_time+1, tasks
                    [i].tid);
                current_time++;
                break;
            }
        }
    }
```

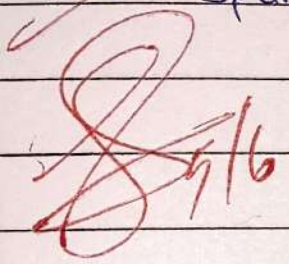
PAGE NO :
DATE :

```
        completed_tasks++;
    }
    *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 the number of tasks: ");
    scanf("%d", &num_tasks);
    if (num_tasks <= 0 || num_tasks > MAX_TASKS){
        printf("Invalid number of tasks. Please enter a
            number between 1 and %d.\n", MAX_TASKS);
    }
    printf("Enter number of tickets for each task:\n");
    for (int i=0 ; i< num_tasks ; i++){
        tasks[i].tid = i+1;
        printf("Task %d tickets: ", tasks[i].tid);
        scanf("%d", &tasks[i].tickets);
    }
    printf("\nRunning tasks:\n");
    schedule(tasks, num_tasks, &time_span_ms);
    printf("\nTime span of the Gantt Chart : %d
        milliseconds\n", time_span_ms);
    return 0;
}
```

* Output:

Enter the number of tasks: 3
Enter number of tickets for each task:
Task 1 tickets: 10
Task 2 tickets: 20
Task 3 tickets: 30

Running tasks:
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 Gantt chart: 300 millisconds.



12/6/24 Lab-5

PAGE NO :
DATE :

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

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

int mutex=1, full=0, empty=5, x=0;

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

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

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

void consumer(){
    mutex = wait(mutex);
    full = wait(full);
    empty = signal(empty);
    printf("\n Consumer consumes item %d", x);
    x--;
    mutex = signal(mutex);
}
```

```
int main(){
  int n;
  void producer();
  void consumer();
  int wait(int);
  int signal(int);

  printf("\n1.Producer\n2.Consumer \n3.Exit");
  while(1){
    printf("\nEnter 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:
        exit(0);
        break;
    }
  }
  return 0;
}
```

PAGE NO:
DATE:

* 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: 2
Consumer consumes the item 3

Enter your choice: 2
Consumer consumes the item 2

Producer - 2 → 3 → 4 → 5

Enter your choice: 1
Buffer is full!!

Enter Consumer - 5 → 4 → 3 → 2 → 1

Enter your choice: 2
Buffer is empty!!

Enter your choice: 3

19/6/24 Lab-6

PAGE NO: 39
DATE:

10) Write a C program to simulate the concept of Dining-Philosophers problem

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_PHILOSOPHERS 10

typedef enum {THINKING, HUNGRY, EATING} state_t;
state_t states[MAX_PHILOSOPHERS];
int num_philosophers;
int num_hungry;
int hungry_philosophers[MAX_PHILOSOPHERS];
int forks[MAX_PHILOSOPHERS];

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 waiting\n", i+1);
        else if (states[i]== EATING) printf("P %d is eating\n", i+1);
    }
}

int can_eat(int philosophers_id){
    int left_fork = philosopher_id;
    int right_fork = (philosopher_id + 1) % num_philosophers;
    if (forks[left_fork] ==0 && forks[right_fork] ==0){
        forks[left_fork] = forks[right_fork] =1;
        return 1;
    };
    return 0;
}
```

PAGE NO :
DATE :

```
void simulate(int allow_two){
    int eating_count = 0;
    for (int i=0; i<num_hungry; ++i){
        int philosopher_id = hungry_philosophers[i];
        if (states[philosopher_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++){
        int philosopher_id = hungry_philosophers[i];
        if (states[philosopher_id] == EATING){
            int left_fork = philosopher_id;
            int right_fork = (philosopher_id + 1) % num_philosop
            forks[left_fork] = forks[right_fork] = 0;
            states[philosopher_id] = THINKING;
        }
    }
}

int main(){
    printf("Enter the total number of philosophers
        (max %d): ", MAX_PHILOSOPHERS);
    scanf("%d", &num_philosophers);
    if (num_philosophers < 2 || num_philosophers >
        MAX_PHILOSOPHERS){
```

```
printf ("Invalid number 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++){
    forks [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;
        case 4:
            printf("
        default:
            printf("Invalid choice. Please try again.\n");
            break;
    }
} while (choice != 3);
return 0;
}
```

*Output:

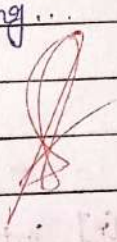
```
Enter the total number of philosophers (max 10): 5
How many are hungry: 3
Enter philosopher 1 position: 1
Enter philosopher 2 position: 3
Enter philosopher 3 position: 5

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

1. One can eat at a time
2. Two can eat at a time
3. Exit
Enter your choice: 1
P1 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: 1
P3 is granted to eat

P1 is thinking
P2 is thinking
P3 is thinking
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: 3
Exiting...



19/6/24

Lab-7

11) Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance

```
#include <stdio.h>
int p, r;
void avai(int all[p][r], int tot[r], int avail[r]){
    for (int j=0; j<r; j++){
        avail[j]=0;
        for (int k=0; k<p; k++){
            avail[j] += all[k][j];
        }
    }
    for (int j=0; j<r; j++){
        avail[j] = tot[j] - avail[j];
    }
}

void need(int all[p][r], int max[p][r], int needmat[p][r]){
    for (int i=0; i<p; i++){
        for (int j=0; j<r; j++){
            needmat[i][j] = max[i][j] - all[i][j];
        }
    }
}

void safety(int p, int r, int all[p][r], int avail[r], int needmat[p][r], int seq[p]){
    for (int i=0; i<p; i++){
        f[i]=0;
    }
    while (count<p && h<p){
        for (int i=0; i<p; i++){
            if (f[i]==0){
```

```
c=0)
for (int j=0; j<r ; j++){
  if (need mat [i] [j] <= avail[j]){
    c++;
  }
}
if (c==r){
  printf("P%d is visited (Tt", i);
  for (int k=0; k<r; k++){
    avail[k] += all [i] [k];
    printf ("%d", avail [k]);
    printf ("\t");
  }
  printf("\n");
  f[i]=1;
  count++;
  seq [h]=i;
  h++;
 }
}
}
}
}
int main(){
printf ("Enter the number of processes : ");
scanf ("%d", &p);
printf (" Enter the number of resources : ");
scanf ("%d", &r);
int tot[r], need mat [p] [r], avail [r], seq [p];
printf ("Enter the total instances of each
  resource : ");
for (int i=0 ; i<r; i++){
```

```
    scanf ("%d ", &tot[r]);
}

int all [p][r], max [p][r];
printf ("Enter the details of each process
    (Allocation matrix) : \n");
for (int i=0; i<p; i++){
    for (int j=0; j<r; j++){
        scanf ("%d", &all [i][j]);
    }
}

printf ("Enter the maximum matrix : \n");
for (int i=0; i<p; i++){
    for (int j=0; j<r; j++){
        scanf ("%d ", &max [i][j]);
    }
}

avail (all, tot, avail);
needi (all, max, needmat);

printf ("Need matrix : \n");
for (int i=0; i<p; i++){
    for (int j=0; j<r; j++){
        printf ("%d \t", needmat [i][j]);
    }
    printf ("\n");
}

safety (p, r, all, avail, needmat, seq);
printf ("Safe sequence is: ");
for (int i=0; i<p; i++){
```

```
        printf("P%d", seq[i]);
    }
    return 0;
}
```

* Output :

```
Enter the number of processes: 5
Enter the number of resources: 3
Enter the total instances of each resource: 10 5 7
Enter the details of each process (allocation matrix):
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter the maximum matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Need matrix:
7 4 3
1 2 2
6 0 0
0 1 1
4 3 1
P1 is visted (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)
Safe sequence is: P1 P3 P4 P0 P2
```

3/7/24 Lab-8

PAGE NO :
DATE :

12) Write a C program to simulate deadlock detection

```
#include <stdio.h>
void main(){
  int n, m, i, j;
  printf("Enter the number of processes and number
      of types of resources:\n");
  scanf("%d %d",&n,&m);
  int max[n][m], need[n][m], all[n][m], ava[m],
  flag=1, finish[n], dead[n], c=0;
  printf("Enter the request number of each type of
      resource needed by each process:\n");
  for(i=0; i<n; i++){
    for(j=0; j<m; j++){
      scanf("%d", &request[i][j]);
    }
  }
  for(i=0; i<n; i++){
    finish[i]=0;
  }
  printf("Enter the number of allocated of each type
   of resource needed by each process:\n");
  for(i=0; i<n; i++){
    for(j=0; j<m; j++){
      scanf("%d", &all[i][j]);
    }
  }
  while(flag){
    flag=0;
    for(i=0; i<n; i++){
      c=0;
      for(j=0; j<m; j++){
        if(finish[i]==0 && request[i][j]<=ava[j]){
          c++;
```

```
if (c==m){
  for (j=0;j<m;j++){
    ava[j] -= request[i][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 deadlock processes are: \n");
  for (i=0; i<j; i++){
    printf("P%d", dead[i]);
  }
}
else
  printf("\nNo deadlock has occured!\n");
}
```

PAGE NO :
DATE :

Output:

Enter the number of processes and number of types of resources: 4 3

Enter the allocated number of each type of resource needed by each process:

1 0 2

2 1 1

1 0 3

1 2 2

Enter the available number of each type of resource:

0 0 0

Enter the request number 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:

P3.

3/7/24 Lab-9

10) Write a C program to simulate the following CMA techniques a) 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("Memory Management Scheme-First Fit\n");
    printf("File no:\tFile_size:\tBlock_no:\tBlock_size:
        \tFragment\n");
    for (int i=0; i<n_files; i++){
      for (int j=0; j<n_blocks; j++){
        if (blocks[j].is_free && blocks[j].block_size>=
            files[i].file_size){
          blocks[j].is_free =0;
          printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
            files[i].file_no, files[i].file_size, blocks[j].
            block_no, blocks[j].block_size);
          break;
        }
      }
    }
}
```

PAGE NO :
DATE :

```
void worstFit(struct Block blocks[], int n_blocks,
    struct File files[], int n_files){
  printf("Memory Management scheme - Worst Fit\n");
  printf("File no : \tFile_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 (blocks[j].is_free && blocks[j].block_size>=
          files[i].file_size){
        int fragment = blocks[j].block_size -
            files[i].filesize;
        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\n", files[i].file
          files[i].filesize, blocks[worst_fit_block].block
          blocks[worst_fit_block].blocksize);
    }
  }
}

void bestFit(struct Block blocks[], int n_blocks,
    struct File files[], int n_files){
  printf("Memory Management scheme - BestFit\n");
  printf("File_no :\tFile_Size:\t Block_no :\t BlockSize
      :\tFragment\n");
```

```
for (int i=0; i<n_files; i++){
    int best_fit_block = -1;
    int min_fragment = 10000;
    for (int j=0; j<n_blocks; j++){
        if (blocks[j].is_free && blocks[j].block_size
            >= files[i].file_size){
            int fragment = fragment;
            best_fit_block = j;
        }
    }
}
if (best_fit_block != -1){
    blocks [best_fit_block]. is_free = 0;
    printf("%d \t\t %d \t\t %d \t\t %d \n",
    files[i].file_no; files[i]. file_size, blocks [
    best_fitblock].block_no, blocks [best_fit_block].
    block_size);
}
}
}

int main (){
    int n_blocks, n_files;
    printf ("Enter the number of blocks: ");
    scanf ("%d", &n_blocks);
    printf ("Enter the number 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 the 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("Enter the size 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++){
    blocks[i].is_free = 1;
}

worstFit(blocks, n_blocks, files, n_files);
printf("\n");

for (int i=0; i<n_blocks; i++){
    blocks[i].is_free = 1;
}

bestFit(blocks, n_blocks, files, n_files);

return 0;
}
```

Output:

```
Enter the number of blocks: 5
Enter the number of files: 4
Enter the size of block 1: 400
Enter the size of block 2: 700
Enter the size of block 3: 200
Enter the size of block 4: 800
Enter the size of block 5: 600
```

Enter the size of file 1: 212
Enter the size of file 2: 517
Enter the size of file 3: 312
Enter the size of file 4: 526

Memory Management Scheme - First-Fit

File_no:	File_size:	Block_no:	Block_size:	Fragment
1	212	1	400	188
2	517	2	700	183
3	312	5	600	288

Memory Management Scheme - Worst-Fit

File_no:	File_Size:	Block_no:	Block_size:	Fragment
1	212	2	700	488
2	517	5	600	83
3	312	1	400	88

Memory Management Scheme - Best-Fit

File_no:	File_Size:	Block_no:	Block_size:	Fragment
1	212	4	300	88
2	517	5	600	83
3	312	1	400	88
4	526	2	700	174

3/7

10/7/24 Lab-10

14) Write a C program to simulate paging technique of memory management.

```
#include <stdio.h>
#include <limits.h>
#include<stdlib.h>
void print_frames(int frame[], int capacity, int page_faults){
  for (int i=0; i<capacity; i++){
    if (frame[i] == -1)
      printf("- ");
    else
      printf("%d ", frame[i]); }
  if (page_faults>0)
    printf("PF No. %d", page_faults);
  printf("\n");
}
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;
  printf("FIFO Page Replacement Process:\n");
  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){
      frame[index]=pages[i];
      index=(index+1)%capacity;
      page_faults++; }
    print_frames(frame, capacity, found?0:page_faults); }
  printf("Total Page Faults using FIFO: %d\n\n",
  page_faults); }
```

PAGE NO :
DATE :

```
void lru (int pages[], int n, int capacity) {
    int frame[capacity], counter[capacity], time=0,
        page-faults=0;
    for (int i=0 ; ; i<capacity ; i++){
        frame [i]=-1;
        counter [i]=0; }
    printf ("LRU . Page Replacement Process:\n");
    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 ++; }
        print_frames(frame, capacity, found ? 0 : page_faults); }
    printf ("Total Page Faults using LRU: %d\n\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;
    printf ("Optimal Page Replacement Process :\n");
    for (int i=0; i<n; i++){
        int found=0;
        for (int j=0; j<capacity ; j++) {
            if (frame[j] == pages [i]){
```

PAGE NO :
DATE :

```
        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>farthest){
                        farthest=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++; }
    print_frames(frame, capacity, found ? 0 : page_faults); }
    printf("Total Page faults using Optimal: %d\n\n", page_faults);}
int main(){
    int n, capacity;
    printf("Enter the number of pages: ");
    scanf("%d", &n);
    int *pages=(int*) malloc(n*sizeof(int));
    printf("Enter the pages: ");
    for (int i=0; i<n; i++) scanf("%d", &pages[i]);
    printf("Enter the frame capacity: ");
    scanf("%d", &capacity);
    printf("\n Pages: ");
    for (int i=0; i<n; i++){
        printf("%d", pages[i]);
    printf("\n\n");
    fifo(pages, n, capacity);
    lru(pages, n, capacity);
    Optimal(pages, n, capacity);
    free(pages);
    return 0;
}
```

PAGE NO :
DATE :

Output:

Enter the no. of pages : 20
Enter the pages : 0 9 0 1 8 1 8 7 8 7 1 2 8 2 7 8 2 3 8 3
Enter the frame capacity : 3
Pages : 0 9 0 1 2 1 8 7 8 7 1 2 8 2 7 8 2 3 8 3

FIFO page Replacement Process :
0 - - PF No. 1
0 9 - PF No. 2
0 9 1 PF No. 3
8 9 1 PF No. 4
8 7 1 PF No. 5
8 7 2 PF No. 6
3 7 2 PF No 7
3 8 2 PF No. 8
Total Page Faults using FIFO : 8

LRU page Replacement Process :
0 - - PF No. 1
9 - - PF No. 2
9 0 - PF No. 3
9 0 1 - PF No. 4
8 0 1 PF No. 5
8 7 1 PF No. 6
2 7 1 PF No. 7
2 8 1 PF No. 8
2 8 7 PF No. 9
2 8 3 PF No. 10
Total Page Faults using LRU : 10.

Optimal Page Replacement Process :
0 - - PF No. 1
0 9 - PF No. 2
1 9 - PF No. 3
1 8 - PF No. 4

PAGE NO :
DATE :

1 8 7 PF No. 5

2 8 7 PF No. 6

3 8 7 PF No. 7

Total Page Faults using Optimal : 7