Bafna Gold
Date: Page:

Stack

```
#include<stdio.h>
#define size 5

int stack[size], top = -1;
void push(int value)
{
    if(top == size-1)
    {
        printf("overflow");
    }
    top++;
    stack[top] = value;
}
void pop()
{
    if(top == -1)
    {
        printf("Underflow");
    }
printf("The element is deleted :", stack[top]);
    top--;
}
void display()
{   if(top == -1)
    {
        printf("Underflow");
    }
    for(int i=0; i<=top; i++)
        printf("stack : %d", stack[]);
    }
}
```

Bafna Gold
Date: Page:

Stack

```
#include<stdio.h>
#define size 5

int stack[size], top = -1;
void push(int value)
{
    if (top == size-1)
    {
        printf("Overflow");
    }
    top++;
    stack[top] = value;
}
void pop()
{
    if (top == -1)
    {
        printf("Underflow");
    }
printf("The element is deleted :", stack[top]);
    top--;
}
void display()
{   if(top == -1)
    {
    printf("Underflow");
    }
    for(int i=0; i<=top; i++)
        printf("stack : %d", stack[i])
    }
}
```

```
int main()
{
    int value, choice;
    while(1)
    {
        printf("1. push \n 2. pop \n 3- display \n 4. exit\n");
        printf("Enter your choice");
        scanf("%d", &choice);
        switch(choice)
        {
        case 1: printf("Enter your elements");
                scanf("%d", &value);
                push(value);
                break;
        case 2: pop()
                break;
        case 3: display()
                break;
        case 4:
                return 0;
        default:
                printf("Invalid choice");
        }
    }
}
```

output :-

1. Push
2. Pop
3. Display
4. Exit

Enter your choi : 1

Enter your elemt : 2

Bafna Gold
Date: Page:

Enter your choice : 1
Enter your choice : 3

Enter your choice : 3

Stack : 3 2

Enter your choice : 2

The element is deleted : 3
Enter your choice : 3
Stack : 2

Queues

```
#include <stdio.h>
#define size 5

int queue[size], front = -1, rear = -1;
void enqueue(int value)
{
  if (rear == size - 1)
  {
    printf("Overflow");
  }
  if (front == -1)
  {
    front = 0;
  }
  rear++;
  queue[rear] = value;
  printf("The element is pushed :", value);
```

```
void dequeue()
{
    if(front == -1 || rear < front)
    {
        printf("Empty");
    }
    printf("The element is deleted : %d", queue[front]);
    front++;
    if(front > rear)
    {
        front = rear = -1;
    }
}
void display()
{
    if(front == -1 || front > rear)
    {
        printf("Empty");
    }
    printf("Queue elements");
    for(int i = front; i <= rear; i++)
    {
        printf("%d", queue[i]);
    }
}
int main()
{
    int choice, value;
    while(1)
    {
        printf("1. Enqueue \n 2. Dequeue \n
            3. Display \n 4. Exit \n");
        printf("%d", &choice);
```

```
switch (choice)
{
    case 1 : printf ("Enter value : ");
             scanf ("%d", &value);
             enqueue (value);
             break;
    case 2 : dequeue ();
             break;
    case 3 : display ();
             break;
    case 4 : printf ("Exit");
             return 0;
    default : printf ("Invalid");
}
```

Output :->

1. Enqueue
2. Dequeue
3. Display
4. Exit

24/12/25

Enter your choice : 1
Enter value : 2

Enter your choice : 1
Enter value : 3

Enter your choice : 2
The element is deleted : 3

Enter your choice : 3

LAB-2

Q) Write a C program to simulate cpu scheduling algorithms
1. FCFS
2. SJF

1) FCFS (First come first serve)

```
#include <stdio.h>
void findWaitingTime(int bt[], int n, int wt[]) {
    wt[0] = 0;
    for (int i = 1; i < n; i++) {
        wt[i] = bt[i-1] + wt[i-1];
    }
}
void findTurnaroundTime(int bt[], int wt[], int n) {
    for (int i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
    }
}
void findAverageTime(int bt[], int n, int at[]) {
    int wt[n], tat[n];
    findWaitingTime(bt, n, wt, at);
    findTurnaroundTime(bt, wt, n, tat);
    int total_wt = 0, total_tat = 0;
    printf("Process\t Burst Time\t Waiting Time\t Turnaround Time\n");
    for (int i = 0; i < n; i++) {
        total_wt += wt[i];
        total_tat += tat[i];
        printf("%d\t\t %d\t\t %d\t\t %d\n", i+1, bt[i], wt[i], tat[i]);
    }
}
```

```
printf ("Average Waiting Time = %.2f\n",
                    (float) total_wt / n);
printf ("Average Turnaround Time = %.2f\n",
                    (float) total_tat / n);
}
int main () {
int n;
printf ("Enter the number of processes: ");
scanf ("%d", &n);
int bt[n], at[n];
printf ("Enter burst Times for each process:\n");
for (int i = 0; i < n; i++) {
printf ("Burst time for process %d: ", i+1);
scanf ("%d", &bt[i]);
}
findAverageTime (bt, at, n);
return 0;
}
```

Output :→ Enter the number of processes: 4
Enter burst time for each process.
Burst time for process 1: 7
Burst time for process 2: 3
Burst time for process 3: 4
Burst time for process 4: 6

Process	Arrival time	Burst Time	Waiting time	Turnaround Time
1	0	7	0	7
2	0	3	7	10
3	0	4	10	14
4	0	6	14	20

Average Waiting Time = 7.75

Average Turnaround Time = 12.75

Manual output :-

Process	AT	BT	CT	TAT (CT-AT)	WT (TAT-BT)	RT (first time)
P1	0	7	7	7	0	0
P2	0	3	10	10	7	7
P3	0	4	14	14	10	10
P4	0	6	20	20	14	14
				51	31	

Avg waiting time = $\frac{31}{4} \Rightarrow 7.75$

Avg Turnaround time = $\frac{51}{4} = 12.75$

Gantt chart

Bafna Gold
Date: Page:

SFJ Algorithm

```
#include <stdio.h>

struct process
{
    int pid;
    int arrival_time;
    int burst_time;
    int remaining_time;
    int waiting_time;
    int turnaround_time;
    int completed;
};
void sortbybursttime(struct process p[], int n)
{
    for (int i = 0; i < n-1; i++) {
        for (int j = 0; j < n-i-1; j++)
        {
            if (p[j].burst_time > p[j+1].burst_time)
            {
                struct process temp = p[j];
                p[j] = p[j+1];
                p[j+1] = temp;
            }
        }
    }
}
void SFJ_nonpreemtive(struct process p[], int n)
{
    sort by burst time (p, n);
    p[0].waiting_time = 0;
```

```
for (int i = 1;  i < n; i++) {
    p[i].waiting time = P[i-1].waiting_time +
                        P[i-1].burst_time;
}
for (int i = 0; i < n ; i++)
{
    P[i].turnaround_time = p[i].waiting_time +
                           P[i].burst_time;
}
}
void SFJ_preemtive (struct process P[], int n)
{
    int completed = 0, time = 0, min_index;
    while( completed != n)
    {
        min_index = -1;
        int min_time = Int MAX;
        for (int i=0; i < n ; i++)
        {
            if ( P[i].arrival_time <= time && p[i].
                 remaining time > 0 &&
                 p[i].remaining_time < min_time)
            {
                min_time = P[i].remaining_time;
                min_index = i;
            }
        }
        if (min index == -1)
        {
            time ++;
            continue;
        }
        P[min_index].remaining_time --;
```

```
        time ++;
        if (P[min_index].remaining_time == 0)
        {
            p[min_index].completed = 1;
            completed ++;
            p[min_index].turnaround_time =
                time - P[min_index].arrival_time
            p[min_index].waiting_time = p[min_index].
                turnaround_time = p[min_index].burst_time;
        }
    }
}
void displayprocess (struct Process P[], int n)
{
    float total_wt = 0, total_tat = 0;
    printf("\n Process \t Arrival Time \t Burst time \t
            waiting time \t turnaround time \n");
    for (int i=0; i<n; i++)
    {
        printf("p %d \t %d \t \t %d \t \t %d \t
            \t %d \n", P[i].pid, P[i].
            arrival_time, P[i].burst_time,
            P[i].waiting_time, P[i].turnaround_time);
        total_wt += P[i].waiting_time;
        total_tat += P[i].turnaround_time;
    }
    printf("\n Average Waiting time: %2f, total_wt/n);
    printf("\n Average Turnaround time: %.2f \n", total_tat/n);
}
int main()
{
    int n, choice;
```

```
printf("Enter number of process");
Scanf("%d", &n);
struct process p[n];
for(int i=0 ; i<n ; i++)
{
    p[i].pid = i+1;
    printf("Enter Arrival time & Burst time", i+1);
    scanf("%d %d", &p[i] arrival time, p[i].
                                    burst_time);
    p[i].remaining_time = p[i] burst_time;
    p[i].completed = 0;
}
Printf("Choose Algorithm : \n 1. Non-preemptive
                    SFJ \n Enter : ");
Scanf("%d", &choice);
if(choice == 1)
{
    SFJ_no_preemptive(p,n)
}
else if (choice == 2)
{
    SJF_preemptive(p,n)
}
else
{
    printf("Invalid choice ! \n");
}
display process(p,n);
}
```

Bafna Gold
Date: Page:

Output :-
Enter number of processes : 4
Enter Arrival time and Burst time for P1 : 0 6
Enter Arrival time and Burst time for P2 : 2 8
Enter Arrival time and Burst time for P3 : 4 7
Enter Arrival time and Burst time for P4 : 5 3

Choose scheduling Algorithm :
Enter Choice : 1

Output for Non-preemptive SFJ:

Process	Arrival Time	BT	WT	TAT
P1	0	6	0	6
P2	2	8	3	9
P3	4	7	9	16
P4	5	3	16	19

Avg WT : 7
Avg TAT : 13

Enter number of process : 4

Enter Arrival time & Burst time for P1 : 0 6
Enter Arrival time & Burst time for P2 : 2 8
Enter Arrival time & Burst time for P3 : 4 7
Enter Arrival time & Burst time for P4 : 5 3

Choose scheduling Algorithm :
Enter choice : 2

Output for Preemptive SJF

Process	AT	BT	WT	TAT
P1	0	6	0	6
P2	2	8	14	22
P3	4	7	5	12
P4	5	8	10	14

Avg WT : 5.00

Avg TAT : 19.00

NLP
20/1/25

19/03/25

Bafna Gold
Date: Page:

LAB - 3

Write a C program to simulate the following CPU scheduling algorithm to find turnaround time & waiting time.

→ Priority (pre-emptive & Non-pre-emptive)

→ Round Robin (Experiment with different quantum sizes for RR algorithm)

Round Robin -

```
# include <Stdio.h>
int main()
{
    int i, j, n, bu[10], wa[10], tat[10], t, ct[10],
        max;
    float awt=0, att=0, temp=0;
    clrscr();
    printf("Enter the number of processes -- ");
    scanf("%d", &n);
    for(i=0; i<n; i++)
    {
        printf("\n Enter Burst time for process %d -- ", i+1);
        scanf("%d", &t);
        bu[i]=t;
    }
    max=bu[0];
    for(i=1; i<n; i++)
    {
        if(max<bu[i])
        {
            max=bu[i];
        }
    }
    for(j=0; j<(max/t)+1; j++)
```

```
for (i=0 ; i<n ; i++) {
    if (bn[i] != 0)
    {
        if (bn[i] <= t)
        {
            tat[i] = temp + bn[i];
            temp = temp -> ba[i];
            bn[i] = 0;
        }
        else
        {
            bn[i] = bn[i] - t;
            temp = temp + 1;
        }
    }
}
for (i=0 ; i<n ; i++)
{
    wa[i] = tat[i] - ct[i];
    att += tat[i];
    awt += wa[i];
}
printf("The Average turnaround time is = %f", att/n);
printf("The Average waiting time is = %f", awt/n);
printf("\n Process \t Burst time \t Waiting time \t
    turnaround time \n");
for (i=0; i<n; i++)
{
    printf("\t %d \t %d \t \t %d \t \t %d \t \t
    \n", i+1, ct[i], wa[i], tat[i]);
    getch();
}
}
```

Bafna Gold
Date: Page:

Priority

```
#include <stdio.h>
#define MAX 100
typedef struct
int pid;
int arival;
int burst;
int priority;
int remaining;
int completion;
int waiting;
int turnaround;
}
process;
void calculateTime(Process process[], int n, int
                   preemptive)
{
    int time=0, completed=0, min_index;
    int is_completed[MAX]={0};
    while(completed!=n){
    min_index=-1;
    for(int i=0; i<n; i++)
    {
        if(Process[i].arrival<=time &&
                !is_completed[i])
        {
            if(min_index==-1||(
                (process[i].priority<process
                (min_index).priority)||(process[i].
            priority==process[min_index].priority
                && process[i].arrival<process[min_index
            ].arrival)){
```

```
        min_index = i;
    }
  }
}
if (min_index == -1)
{
    time++;
}
else
{
    if (pre-emptive)
    {
        process[min_index].remaining--;
        {
            is_completed[min_index] = 1;
            completed++;
        process[min_index].completion = time+1;
    }
else {
    time += process[min_index].burst;
    process[min_index].completion = time;
    iscompleted[min_index] = 1;
    completed++;
  }
    if (!preemptive)
    {
        process[min_index].remaining = 0;
    }
    else {
        time++;
    }
}
}
```

```
for (int i=0 ; i<n ; i++)
{
    process[i].turnaround = process[i].
    completion - process[i].arrival;
process[i].waiting = process[i].
    turnaround - process[i].burst;
}
}
int main()
{
    int n, choice;
    process process[MAX];
    printf("Enter no. of process:");
    scanf("%d", &n);
    printf("Enter process details AT, BT,
                priority &:\n");
    for (int i=0; i<n ; i++)
    {
        process[i].pid = i+1;
    printf("Process %d = i+1);
    scanf("%d %d %d", &process[i].
        arrival, &process[i].burst
        & process[i].priority);
process[i].remaining = process[i].burst;
}
    printf("Choose scheduling :\n 1. Preemption
        & Non-Preemption \n");
    scanf("%d", &choice);
        if (choice == 1)
        {
            printf("\n executing preemptive
                priority Scheduling \n");
```

```
        calculateTimes(process, n, 1);
    }
    else {
        printf("\n Executing non-preemptive Priority
            Scheduling \n");
        calculateTimes(processes, n, 0);
    }
    printResults(Processes, n);
    return 0;
```

Bafna Gold
Date: Page:

Output : 1 (Round Robin)

Enter the number of process : 4

Enter Burst Time for Process 1 : 10
" " " " 2 : 50
" " " " 3 : 24
" " " " 4 : 12

Enter the Size of time : 3

The Average Turnaround time is : 62.250000
The Average Waiting time is : 38.250000

Process	Burst time	Waiting Time	Turnaround time
1	10	27	37
2	50	46	96
3	24	46	70
4	12	34	46

Output : (Priority)

Enter the number of process : 3
Enter Process details (Arrival time, Burst time, priority)

Process 1 : 1 2 3
Process 2 : 4 5 6
Process 3 : 7 8 9

Choose scheduling Type :
1. Preemptive priority
2. Non - Preemptive Priority
1

Executing Preemptive Priority scheduling.

PID	AT	BT	tpriority	CT	TAT	WT
1	1	2	3	0	3	2
2	4	5	6	0	9	5
3	7	8	9	2	17	10

Output :→

Enter the number of process :3
Enter Process details
Process 1 : 1 2 3
Process 2 : 4 5 6
Process 3 : 7 8 9

Choose Scheduling Type.

1. Preemptive Priority
2 Non-Preemptive Priority
2
Executing.

PID	AT	BT	tpriority	CT	TAT	WT
1	1	2	3	0	3	2
2	4	5	6	0	9	5
3	7	8	9	2	17	10

2/04/25

LAB-4

Bafna Gold
Date: Page:

Multilevel Queue Scheduling

```
#include <stdio.h>
#define MAX 10
#define Time_Quantum 3
typedef struct {
    int pid;
    int arrival_time;
    int burst_time;
    int priority;
} process;
void sort_by_priority (Process queue[], int n) {
    for (int i = 0; i < n-1; i++) {
        for (int j = i+1; j < n; j++) {
            if (queue[i].priority > queue[j].priority) {
                process temp = queue[i];
                queue[i] = queue[i];
                queue[j] = temp;
            }
        }
    }
}
void execute_fcfs (Process queue[], int n) {
    printf ("\n Executing processes using FCFS:\n");
    for (int i = 0; i < n; i++) {
        printf ("Process %d | Arrival Time: %d | Burst Time: %d | Priority: %d \n",
            queue[i].pid, queue[i].arrival_time,
```

```
        queue[i].burst_time, queue[i].priority);
    }
}

void execute_round_robin(Process queue[], int n)
{
    int remaining_time[MAX];
    for(int i=0; i<n; i++){
        remaining_time[i] = queue[i].burst_time;
    }
    int time = 0;
    int done;
    printf("\n Executing processes using
            Round Robin :\n");
    do{
        done = 1;
        for(int i=0; i<n; i++){
            if(remaining_time[i]>0){
                done = 0;
            if(remaining_time[i] >0){
                done = 0;
                if(remaining_time[i]>
                    Time_Quantum){
                printf("Process %d executed for
                    %d units\n", queue[i].pid,
                    Time_Quantum);
                remaining_time[i] -= Time_Quantum;
                time += Time_Quantum;
            } else {
                printf("Process %d executed for
                    %d units and finished\n",
                    queue[i].pid, remaining_time[i]);
                time += remaining_time[i];
```

remaining_time[i] = 0

}

{

}

while (!done);

}

int main() {

int n1, n2;

Process high_priority_queue[MAX],

low_priority_queue[MAX];

printf("Enter the number of processes in high priority queue (FCFS): ");

scanf("%d", &n1);

for (int i=0; i<n1; i++) {

printf("Enter process ID, Arrival Time, Burst Time, Priority: ");

scanf("%d %d %d %d",

&high_priority_queue[i].pid,

&high_priority_queue[i].arrival_tim,

&high_priority_queue[i].burst_tim,

&high_priority_queue[i].priority);

}

printf("Enter the number of processes in low priority queue (Round Robin): ");

scanf("%d", &n2);

for (int i=0; i<n2; i++) {

printf("Enter Process ID, Arrival Time, Burst Time, Priority: ");

scanf("%d %d %d %d",

&low_priority_queue[i].pid,

&low_priority_queue[i].arrived_tim,

&low_priority_queue[i].burst_tim,

&low_priority_queue[i].priority);

```
}
sort_by_priority(high_priority_queue, n1);
sort_by_priority(low_priority_queue, n2);
execute_fcfs(high_priority_queue, n1);
execute_Round_robin(low_priority_queue, n2);
return 0;
}
```

Output:

```
Enter process : 5

Enter process details: (PID Arrival Burst type)

Process 1: 1 0 2 0
P 2: 2 2 5 1
P 3: 3 6 8 1
P 4: 4 5 3 0
P 5: 5 7 2 1

Enter time quantum for Round Robin: 2

System Queue [FCFS]:

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

User Qum (Round Robin) :-

PID	AT	BT	CT	TAT	WT
2	2	5	19	17	12
3	6	8	23	17	9
5	7	2	14	7	5

21/11/25

9/9/22

LAB-5

Q.) Write a C program to simulate Real-Time CPU scheduling algorithms.
(a.) Rate-Monotonic
(b.) Earliest-deadline First

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

#define MAX_PROCESS 10
typedef struct{
  int id;
  int burst_time;
  float priority;
} Task;

int num_of_process;
int execution_time[MAX_PROCESS], period[MAX_PROCESS]
remain_time[MAX_PROCESS], deadline[MAX_PROCESS]
remain_deadline[MAX_PROCESS];

Void get_process_info(int selected_algo)
{
  printf("Enter total number of processes (maximum %d):",
                                    MAX_PROCESS);
  Scanf("%d", &num_of_process);
  if(num_of_process < 1)
  {
    exit(0);
  }
```

```
for (int i = 0 ; i < num_of_process; i++) {

int num_of_process;
int execution_time[MAX_PROCESS], period[MAX_PROCESS],
    remain_time[MAX_PROCESS], deadline[MAX_PROCESS],
    remain_deadline[MAX_PROCESS];
void get_process_info (int selected_algo) {
    printf("Enter total number of processes (maximum: "
                                    MAX_PROCESS);
    scanf("%d", &num_of_process);
    if (num_of_process < 1) {
        exit(0);
    }
    for (int i = 0; i < num_of_process; i++) {
        printf("\nProcess %d:\n", i+1);
        printf("  ==> Execution time: ");
        scanf("%d", &execution_time[i]);
        remain_time[i] = execution_time[i];
        if (selected_algo == 2) {
            printf("  ==> Deadline: ");
            scanf("%d", &deadline[i]);
        } else {
            printf("  ==> Period: ");
            scanf("%d", &period[i]);
        }
    }
}

int max(int a, int b, int c) {
    int max;
    if (a >= b && a >= c)
        max = a;
    else if (b >= a && b >= c)
```

```
        max = b;
    else
        max = c;
    return max;
}

int get_observation_time(int selected_algo){
if (selected_algo == 1){
    return max(period[0], period[1], period[2]);
} else if (selected_algo == 2){
    return max(deadline[0], deadline[1], deadline[2]);
}
    return 0;
}

void print_schedule(int process_list[], int cycles){
    printf("\n Scheduling :\n\n");
    printf("Time :");
    for(int i = 0; i < cycles; i++){
        if(i<10)
            printf("| 0%d", i);
        else
            printf("| %d", i);
}
    printf("|\n");
    for (int i=0; i<num_of_process; i++){
    printf("P[%d] :", i+1);
        for(int j=0; j<cycles; j++){
        if(process_list[j] == i+1)
        printf("|####");
        else
            printf("|    ");
        }
        printf("|\n");
}}
```

```
void rate_monotonic (int time){
    int process_list[100] = {0}, min = 999, next_proc=0;
    float utilization = 0;
    for (int i=0; i < num_of_process; i++){
        utilization += (1.0 * execution_time[i])/
                        period[i];
    }
    int n = num_of_process;
    int m = (float)(n * (pow(2,1.0/n)-1));
    if (utilization > m){
    printf("\n Given problem is not schedulable
            under the said scheduling algorithm\n");
    return;
    for (int i=0; i < time; i++){
        min = 1000;
        for (int j=0; j < num_of_process; j++){
        if (remain_time[j] > 0){
            if (min > period[j]){
            min = period[j];
            next_process = j;
            }
        }
    }
    if (remain_time[next_process] > 0){
        process_list[i] = next_process + 1;
        remain_time[next_process] -= 1;
    }
    for (int k = 0; k < num_of_process; k++){
        if ((i+1) % period[k] == 0){
        remain_time[k] = execution_time[k];
    }
}
```

```
print_schedule (process_list, time);
}

void earliest_deadline_first (int tim){
float utilization = 0;
for (int i = 0 ; i < num_of_process; i++ ){
utilization += (1.0 * execution_time [i])/
                deadline [i];
}
int n = num_of_process;
int process [ num_of_process];
int max_deadline, current_proces = 0
min_deadline, process_list [time].
bool is_ready [ num_of_process ];
for ( int i = 0; i < num_of process ; i++){
is_ready [i] = true;
process [i] = i+1;
}
max_deadline = deadline [0];
for (int i = 1; i < num_of_process ; i++){
if ( deadline [i] > max_deadline )
max_deadline = deadline [i];
}
for (int i = 0; i < num_of_process ; i++){
for (int j = i+1 ; j < num_of_process ; j++){
if ( deadline [j] < deadline [i]){
int temp = execution_time [j];
execution_time [j] = execution_time [i];
execution_time [i] = temp;
temp = deadline [j];
deadline [j] = deadline [i];
deadline [i] = temp;
}
}
}
```

```
for (int i = 0; i < num_of_process; i++){
remain_time[i] = execution_time[i];
remain_deadline[i] = deadline[i];
}
for (int t = 0; t < time; t++){
if (current_process != -1){
--execution_time[current_process];
process_list[t] = process[current_process];
}else{
process_list[t] = 0;
}
for (int i = 0; i < num_of_process; i++){
--deadline[i];
if ((execution_time[i] == 0) &&
!is_ready[i]){
execution_time[i] = remain_time[i];
is_ready[i] = true;
}
}
min_deadline = max_deadline;
current_process = -1;
for (int i = 0; i < num_of_process; i++){
if ((deadline[i] <= min_deadline) &&
(execution_time[i] > 0)){
current_process = i;
min_deadline = deadline[i];
}
}
}
print_schedule(process_list, time);
}
```

```
int main () {
int option;
int observation_time;
while (1) {
printf (" \n 1. Rate Monotonic \n 2. Earliest
Deadline first \n 3. proportional Scheduling \n Enter
your choice : ");
Scanf ("%d", & option);
Switch (option) {
Case 1:
get - process - info (option);
observation_time = get_observation_time
(option);
rate_monotonic (observation_time);
break;
Case 2:
get - observation_time (option);
Observation_time = get - observation_time
break;
Case 3:
exit (0);
default:
printf ("\n Invalid Statement");
}
return 0;
}
```

Output :⇒

(a) Rate Monotonic

Enter your choice : 1

Enter total number of processes : 3

Process 1:

⇒ Execution time : 3

⇒ Period : 20

Process 2:

⇒ Execution time : 2

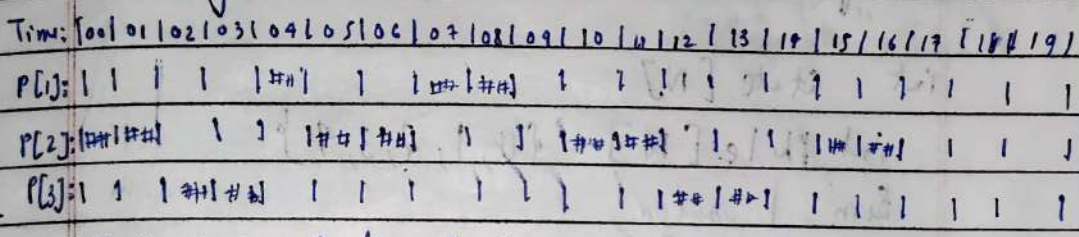
⇒ Period : 5

Process 3:

⇒ Execution time : 2

⇒ Period : 10

Scheduling:



(b) Earliest - deadline First

Enter your choice : 2

Enter total number of processes : 3

Process 1:

⇒ Execution time : 3

⇒ Deadline : 7

Process 2:

⇒ Execution time : 2

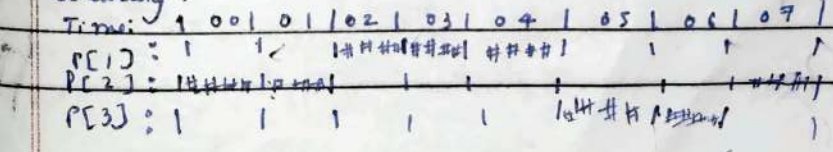
⇒ Deadline : 4

Process 3:

⇒ Execution time : 2

⇒ Deadline : 8

Scheduling :



LAB-5

Dining Philosophers problem

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

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (i+4)%N
#define RIGHT (i+1)%N

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

void test(int i)
{
  if (state[i] == HUNGRY && state[LEFT] !
                = EATING &&
        state[RIGHT] != EATING)
  {
    state[i] = EATING;
    sleep(2);
    printf("Philosopher %d takes fork %d and %d\n",
                i+1, LEFT+1, i+1);
    printf("Philosopher %d is Eating\n", i+1);
    sem_post(&S[i]);
  }
}
```

```
void take_fork(int i){
sem_wait(&mutex);
State[i] = HUNGARY;
printf("Philosopher %d is Hungry\n", i+1);
test(i);
sem_post(&mutex);
sem_wait(&S[i]);
sleep(1);
}
void put_fork(int i){
sem_wait(&mutex);
State[i] = THINKING;
printf("Philosopher %d putting fork %d and %d down\n",
                    i+1, LEFT+1, i+1);
printf("Philosopher %d is thinking\n", i+1);
test(LEFT);
test(RIGHT);
sem_post(&mutex);
}
void * philosopher(void * num){
while(1){
int * i = num;
sleep(1);
take_fork(*i);
sleep(0)
put_fork(*i);
}
}
int main(){
int j;
pthread_t thread_id[N];
sem_init(&mutex, 0, 1);
```

```
for (i=0 ; i<N; i++)
    sem_init (&mutex, 0, 1);
for (i=0; i<N; i++){
    pthread_create (&thread_id[i], NULL,
                    philosopher, &phil[i]);
    printf ("Philosopher %d is thinking \n", i+1);
}
    for (i=0; i<N; i++){
        pthread_join (thread_id[i], NULL);
    }
}
```

Output :-

Philosopher 1 is thinking	Philosopher 5 takes 3 & 4
" " is "	" 5 is eating
" " is Hungry	" 5 putting fork 3&2 down
" " Hungry	" 5 is thinking
" " takes fork 5 and 1 down	" 5 Hungry
" " is eating	" 5 takes 2 & 3
" " putting fork 5&1 down	" 5 is eating
" " is thinking	" 5 putting 2 & 3 down
" 2 is Hungry	Philosopher 5 is eating.
" 2 takes 1 & 5	
" 1 is eating	
" 2 putting fork 1&5 down	
" 2 is thinking	
" 2 Hungry	
" 3 takes 4 & 2	
" 3 is Eating	
" 4 putting fork 1 & 2 down	
" 4 is thinking	
" 5 Hungry	

Bafna Gold
Date: Page:

Producer - Consumer problem

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

```
#include <stdio.h>
#include <stdlib.h>
int mutex = 1, full = 0, empty = 3, x = 0;
void producer();
void consumer();
int wait(int);
int signal(int
int main()
{
int n;
printf("\n 1. Producer \n 2. Consumer \n 3. 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:
    exit (0);
    break;
default:
    printf("Invalid choice! Please enter 1, 2, or 3");
    break;
  }
}
return 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 ("\n Producer produces the items
                %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);
}
```

Output :

1. Producer
2. Consumer
3. Exit.

Enter your choice : 1

Producer produces the item 1
Enter your choice : 2
Consumer consumes item 1
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
Buffer is full !!
Enter your choice : 2
Consumer consumes item 3
Enter your choice 2
Consumer consumes item 2
Enter your choice : 1
Producer produces the item 2
Enter your choice : 2
Consumer consumes item 2
Enter your choice :

LAB - 6

Q.1 Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.

```
#include <stdio.h>
int main()
{
  int n, m, i, j, k;
  printf("Enter the number of processes:");
  scanf("%d", &n);
  printf("Enter the number of resources:");
  scanf("%d", &m);
  int allocation[n][m];
  printf("Enter the Allocation Matrix:\n");
  for(i=0; i<n; i++)
  {
    for(j=0<m; j++)
    {
      scanf("%d", &allocation[i][j]);
    }
  }
  int max[n][m];
  printf("Enter the MAX Matrix:\n");
  for(i=0; i<n; i++)
  {
    for(j=0; j<m; j++)
    {
      scanf("%d", &max[i][j]);
    }
  }
  int available[m];
  printf("Enter the Available Resources:\n");
  for(i=0; i<m; j++)
```

```
}
scanf("%d", &available[i]);
}
int f[n], ans[n], ind = 0
for (k = 0; k < n; k++)
{
    f[k] = 0
}
int need[n][m];
for (i = 0; i < n; k++)
{
    f[k] = 0
}
int need[n][m];
for (int i = 0; i < n; i++)
{
    for (j = 0; j < m; j++)
    {
        need[i][j] = max[i][j] - allocation[i][j];
    }
}
int y = 0;
for (k = 0; k < n; k++)
{
    for (ig = 0; i < n; i++)
    {
        if (f[i] == 0)
        {
            int flag = 0;
            for (j = 0; j < m; j++)
            {
                if (need[i][j] > available[j])
```

```
            {
                flag = 1;
                break;
            }
        }
        if (flag == 0)
        {
            ans[ind++] = i;
            for (y = 0; y < m; y++)
                available[y] += allocation[i][y];
            f[i] = 1;
        }
    }
}
int flag = 1;
for (i = 0; i < n; i++)
{
    if (f[i] == 0)
    {
        flag = 0;
        printf("The following System is not safe\n");
        break;
    }
}
if (flag == 1)
{
    printf("Following is the Safe seq\n");
    for (i = 0; i <= n-1; i++)
```

```
        printf("P %d → ", ans[i]);
    }
    printf("P %d\n", ans[n-1]);
}
    return 0;
}
```

Output:

```
Enter the number of processes: 3
Enter the number of resources: 3
Enter the allocation matrix:
7 5 3
2 1 3
6 2 1
Enter the MAX matrix:
1 3 2
1 0 0
0 1 1
Enter the available resources:
2 1 3
Following is the SAFE sequence
P0 → P1 → P2
```

Q.6 Write a C program to simulate deadlock detection.

```
#include <stdio.h>
static int mark[20];
int i, j, np, nr;
int main()
{
    int alloc[10][10], request[10]
    [10], avail[10], r[10], w[10];
    printf("\n Enter the no. of process: ");
    scanf("%d", &np);
    printf("\n Enter the no. of resources");
    scanf("%d", &nr);
    for(i=0; i<nr; i++)
    {
        printf("\n Total Amount of the Resource
                R %d : ", i+1);
        scanf("%d", &r[i]);
    }
    printf("\n Enter the request matrix");
    for(j=0; j<np; j++)
    for(j=0; j<nr; j++)
    scanf("%d", &request[i][j]);
    printf("\n Enter the allocation matrix");
    for(i=0; i<np; i++)
    for(j=0; j<nr; j++)
    scanf("%d", &alloc[i][j]);
    for(j=0; j<nr; j++)
    {
        avail[j] = r[j];
        for(i=0; i<np; i++)
    }
```

```
avail[j] = alloc[i][j];
}
}
for(i=0; i<np; i++)
{
int count = 0;
for(j=0; j<nr; j++)
w[j] = avail[j];
for(i=0; i<np; j++)
w[j] = avail[j];
for(i=0; i<np; i++)
{
int canbeprocessed = 0;
if(mark[i] != 1)
{
for(j=0; j<nr; j++)
if(request[i][j] <= w[j])
canbeprocessed = 1;
else
{
canbeprocessed = 0;
break;
}
}
if(canbeprocessed)
{
mark[i] = 1;
for(j=0; j<nr; j++)
w[j] += alloc[i][j];
}
}
}
```

```
int deadlock = 0
for (i=0; i< n; i++)
if (mark[i] != 1)
deadlock = 1;
if (deadlock)
printf(" Deadlock detected.");
else
printf("\n deadlock possible");
}
```

Output:

```
Enter the no. of process : 3
Enter the no. of resources : 3
Total Amount of the resources R1 : 3
                              R2 : 3
                              R3 : 2
Enter the request matrix : 2 1 1
1 1 1
1 0 1
Enter the allocation matrix : 1 0 1
1 1 0
0 1 1
Deadlock detected.
```

LAB-7

Bafna Gold

Q.) Write a C program to simulate the following contiguous memory allocation techniques:

(a) Worst-fit
(b) Best-fit
(c) First-fit

```
#include <stdio.h>
#define max 25
void firstFit(int b[], int nb, int f[], int nf);
void worstFit(int b[], int nb, int f[], int nf);
void bestFit(int b[], int nb, int f[], int nf);
int main()
{
    int b[max], f[max], nb, nf;
    printf("Memory Management Schemes\n");
    printf("\n Enter the number of blocks:");
    scanf("%d", &nb);
    printf("Enter the number of files:");
    scanf("%d", &nf);
    printf("\n Enter the size of the blocks:\n");
    for(int i=1; i<=nb; i++)
    {
        printf("Block %d:", i);
        scanf("%d", &b[i]);
    }
    printf("\n Enter the size of the files:\n");
    for(int i=1; i<=nf; i++){
        printf("File %d:", i);
        scanf("%d", &f[i]);
    }
    printf("\n Memory Management scheme - First fit");
```

```
firstFit(b, nb, f, nf);
printf("\n\n Memory Management scheme - First Fit");
firstworstfit(b, nb, f, nf);
printf("\n\n Memory Management scheme - Worst Fit");
worstfit(b, nb, f, nf);
printf("\n\n Memory Management scheme - Best Fit");
bestfit(b, nb, f, nf);

return 0;
}

Void firstFit(int b[], int nb, int f[], int nf){
int bf[max] = {0};
int ff[max] = {0};
int frag[max], i, j;
for(i=1; i<=nf; i++){
for(j=1; j<=nb; j++){
if(bf[j] != 1 && b[j] >= f[i]){
ff[i] = j;
bf[j] = 1;
frag[i] = b[j] - f[i];
break;
}
}
}

printf("\nFile no:\t File size:\t Block no:\t Block size:\t
Fragment");
for(i=1; i<=nf; i++){
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],
frag[i]);
}
}
```

```
void worstFit (int b[], int nb, int f[], int nf ){
  int bf[max] = {0};
  int ff[max] = {0};
  int frag[max], i, j, temp, highest = 0;
  for (i=1; i<=nf; i++){
    for (j=1; j<=nb; j++){
      if (bf[j] != 1){
        temp = b[j] - f[i];
        if (temp >= 0 && highest < temp ){
          ff[i] = j;
          highest = temp;
        }
      }
    }
  frag[i] = highest;
  bf[ff[i]] = 1;
  highest = 0;
  }
  printf ("\nFile_no:\tFile_size:\tBlock_no:
          \tBlock_size:\tFragment");
  for (i=1; i<=nf; i++){
    printf ("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",
            i, f[i], ff[i],
            b[ff[i]], frag[i]);
  }
}
Void bestFit (int b[], int nb, int f[], int nf ){
    int bf[max] = {0};
    int ff[max] = {0};
    int frag[max], i, j, temp, lowest = 10000;
  for (i=1; i<=nf; i++){
    for (j=1; j<=nb; j++){
```

```
        if ( bf [i] != 1 ) {
            temp = b[j] - f[i];
            if ( temp >= 0 && lowest > temp ) {
                ff[i] = j;
                lowest = temp;
            }
        }
    }
    frag[i] = lowest;
    bf[ff[i]] = 1;
    lowest = 10000;
}
printf ("\n File_no: \t File_size: \t Block_no
        \t Block_size: \t Fragment");
for (i = 1; i <= nf && ff[i] != 0; i++)
printf ("\n %d \t \t %d \t \t %d \t \t %d \t \t
        \t \t %d", i, f[i], ff[i], bf[ff[i]],
                frag[i]);
    }
}
```

Output :->

Enter the number of the blocks : 5
Enter the number of the files : 4

Enter the size of the blocks :
Block 1 : 100
Block 2 : 500
Block 3 : 200
Block 4 : 300
Block 5 : 600

Bafna Gold

Enter the size of the files :

File 1 : 212

File 2 : 417

File 3 : 112

File 4 : 426

Memory Management Scheme - First Fit

File_no:	File_size:	Block_no:	Block_size:	Fragment
1	212	2	500	288
2	417	5	600	183
3	112	3	200	88
4	426	not Allocated		

Memory Management Scheme - Worst Fit

File_no:	File_size:	Block_no:	Block_size:	Fragment
1	212	5	600	388
2	417	2	500	83
3	112	4	300	188
4	426	Not Allocated		

Memory Management Scheme - Best Fit

File_no:	File_size:	Block_no:	Block_size	Fragment
1	212	4	300	88
2	417	2	500	83
3	112	3	200	88
4	426	5	600	174

LAB-8

Q) Write a C program to simulate page replacement algorithms:

(a) FIFO
(b) LRU
(c) Optimal

```
#include <stdio.h>
int n, f, i, j, k;
int in[100];
int p[50];
int hit = 0;
int pgfaultcnt = 0;

void getData() {
    printf("\n Enter length of page reference sequence:");
    scanf("%d", &n);
    printf("\n Enter the page reference sequence:");
    for(i=0; i<n; i++)
        scanf("%d", &in[i]);
    printf("\n Enter no. of frames:");
    scanf("%d", &f);
}

void initialize() {
    pgfaultcnt = 0;
    for(i=0; i<f; i++)
        p[i] = 9999;
}
int isHit(int data) {
    hit = 0;
    for(j=0; j<f; j++) {
        if(p[j] == data) {
            hit = 1;
```

```
            break;
        }
    }
    return hit;
}
int getHitIndex(int data){
    int hitind;
    for(k=0; k<f; k++){
        if(p[k]==data){
            hitind = k;
            break;
        }
    }
    return hitind;
}
void dispPages(){
    for(k=0; k<f; k++){
        if(p[k]!=9999)
            printf("%d", p[k]);
    }
}

void dispPgFault(){
    printf("\nTotal no. of page faults: %d", pgfaultcnt);
}

void fifo(){
    getData();
    initialize();
    for(i=0; i<n; i++){
        printf("\nFor %d: ", in[i]);
        if(isHit(in[i])==0){
```

```
for (k=0; k<f-1; k++)
    p[k] = p[k+1];
p[k] = ~~p[k+1]~~ in[i];
pgfault cnt++;
dispPages();
} else
printf("No. page fault");
}
dispPgFaultCnt();
}
void optimal(){
initialize();
int near[50];
for(~~int~~ i=0; i<n; i++){
printf("\nFor %d :", in[i]);
if(isHit(in[i])==0){
for(j=0; j<f; j++){
int pg = p[j];
int found=0;
for(k=i; k<n; k++){
if(pg==in[k]){
near[j]=k;
found=1;
break;
}else
found=0;
}
if(!found)
near[j]=9999;
}
int max=-9999;
int repindex;
```

```
for (j=0; j<f; j++){
    if(near[j] > max){
        max = near[j];
        repindex = j;
    }
}
p[repindex] = in[i];
pgfault(cnt++);
display pages();
} else
    printf("No pages fault");
}
displayPgFaultCnt();
}
void lru(){
    initialize();
    int least[50];
    for(i=0; i<n; i++){
        printf("\n For %d : ", in[i]);
        if(isHit(in[i]) == 0){
            for(j=0; j<f; j++){
                int pg = p[j];
                int found = 0;
                for(j=0; j<f; j++){
                    int found = 0;
                    for(k=i-1; k>=0; k--){
                        if(pg == in[k]){
                            least[j] = k;
                            found = 1;
                            break;
                        } else
                            found = 0;
```

```
    if(!k found)
        least[j] = -9999;
}
int min = 9999;
int repindex;
for(j=0; j<f; j++){
    if(least[j] < f; j++){
        min = least[j];
        repindex = j;
    }
}
p[repindex] = in[i];
pgfault cnt++;
display();
} else
printf("No page fault!");
}
display Fault cnt();
}
int main(){
int choice;
while(1){
printf("In Page Replacement Algorithms \n1. Enter data
\n2. FIFO \n3. Optimal \n4. LRU \n5. Exit
\n Enter your choice:");
scanf("%d", &choice);
switch(choice){
case 1: getData();
        break;
case 2: fifo();
        break;
case 3: optimal();
        break;
```

```
        case 4: lru();
                break;
        default : return 0;
                break;
    }
}
```

Output :→

Page replacement algorithm...

1. Enter data.
2. FIFO
3. Optimal
4. LRU
5. Exit

Enter your choice : 1

Enter length of page reference sequence : 8

Enter the page reference sequence : 2 3 4 2 3 5 6 2

Enter the number of frames : 3

FIFO

Enter your choice : 2

for 2 : 2

for 3 : 2 3

for 4 : 2 3 4

for 2 : No page fault

for 3 : No page fault.

for 5 : 5 3 4

for 6 : 5 6 4

for 2 : 5 6 2

Total number of page faults : 6

Optimal Enter your choice : 3
for 2 : 2
for 3 : 2 3
for 4 : 2 3 4
for 2 : No page fault
for 3 : No page fault
for 5 : 2 5 4
for 6 : 2 6 4
for 2 : No page fault.
Total number of page faults : 5

LRU Enter your choice : 4
for 2 : 2
for 3 ; 2 3
for 4 : 2 3 4
for 2 : No page fault !
for 3 : No page fault !
for 5 : 2 3 5
for 6 : 6 3 5
for 2 : 6 2 5