

# **VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**"JnanaSangama", Belgaum -590014, Karnataka.**



## **LAB REPORT**

**on**

## **OPERATING SYSTEMS**

**(23CS4PCOPS)**

*Submitted by*

**Nallabothula Sai Sruthi(1BM22CS170)**

*in partial fulfillment for the award of the degree of*  
**BACHELOR OF ENGINEERING**  
*in*  
**COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

**BENGALURU-560019**

**Apr-2024 to Aug-2024**

**B. M. S. College of Engineering,  
Bull Temple Road, Bangalore 560019**  
(Affiliated To Visvesvaraya Technological University, Belgaum)  
**Department of Computer Science and Engineering**



**CERTIFICATE**

This is to certify that the Lab work entitled “OPERATING SYSTEMS – 23CS4PCOPS” carried out by **Nallabothula Sai Sruthi(1BM22CS170)**, who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024. The Lab report has been approved as it satisfies the academic requirements in respect of a **OPERATING SYSTEMS - (23CS4PCOPS)** work prescribed for the said degree.

**Dr. Radhika A D**  
Assistant Professor  
Department of CSE  
BMSCE, Bengaluru

**Dr. Jyothi S Nayak**  
Professor and Head  
Department of CSE  
BMSCE, Bengaluru

## Index Sheet

<b>Sl. No.</b>	<b>Experiment Title</b>	<b>Page No.</b>
1.	Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time. →FCFS → SJF (pre-emptive & Non-preemptive)	4
2.	Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time. → Priority (pre-emptive & Non-pre-emptive) →Round Robin (Experiment with different quantum sizes for RR algorithm)	13
3.	Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for the processes in each queue.	19
4.	Write a C program to simulate Real-Time CPU Scheduling algorithms: a) Rate-Monotonic b) Earliest-deadline First c) Proportional scheduling	22
5.	Write a C program to simulate producer-consumer problem using semaphores.	31
6.	Write a C program to simulate the concept of Dining-Philosophers problem.	33
7.	Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.	37
8.	Write a C program to simulate deadlock detection	41
9.	Write a C program to simulate the following contiguous memory allocation techniques a) Worst-fit b) Best-fit c) First-fit	44
10.	Write a C program to simulate page replacement algorithms a) FIFO b) LRU c) Optimal	47

### Course Outcome

CO1	Apply the different concepts and functionalities of Operating System
CO2	Analyze various Operating system strategies and techniques
CO3	Demonstrate the different functionalities of Operating System
CO4	Conduct practical experiments to implement the functionalities of Operating system

1. Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time. →FCFS → SJF (pre-emptive & Non-preemptive)

a.FCFS scheduling using array

```
#include<stdio.h>

int main(){
    int n,i;
    float atat=0,awt=0;
    printf("enter the number of process");
    scanf(" %d",&n);
    int atime1[n],btime2[n],ctime3[n],tattime4[n],wtime5[n];
    printf("enter arrival time of process");
    for(i=0;i<n;i++){
        scanf("%d",&atime1[i]);
    }
    printf("enter burst time of process");
    for(i=0;i<n;i++){
        scanf("%d",&btime2[i]);
    }
    for(i=0;i<n;i++){
        if(i==0){
            ctime3[i]=atime1[i]+btime2[i];
        }
        else{
            if(ctime3[i-1]<atime1[i]){
                ctime3[i]=(atime1[i]-ctime3[i-1])+ctime3[i-1]+btime2[i];
            }
            else{
                ctime3[i]=ctime3[i-1]+btime2[i];
            }
        }
        atat+=ctime3[i]-atime1[i];
        awt+=ctime3[i]-atime1[i]-btime2[i];
    }
    atat=n;
    awt=awt/n;
    printf("Average Turnaround Time = %f",atat);
    printf("Average Waiting Time = %f",awt);
}
```

```

for(i=0;i<n;i++){
    tattime4[i]=ctime3[i]-atime1[i];
}

for(i=0;i<n;i++){
    wtime5[i]=tattime4[i]-btime2[i];
}

for(i=0;i<n;i++){
    atat=atat+tattime4[i];
}

atat=(atat/n);

for(i=0;i<n;i++){
    awt=awt+wtime5[i];
}

awt=(awt/n);

for(i=0;i<n;i++){
    printf("process id %d arrival time %d burst time %d complete time %d turn around time %d
waiting time %d\n",i+1,atime1[i],btime2[i],ctime3[i],tattime4[i],wtime5[i]);
}

printf("average turn around time is %f",atat);
printf("average working time is %f",awt);
}

output:

```

```
C:\Users\saisr\OneDrive\Desktop > + v
enter the number of process4
enter arrival time of process0
1
5
6
enter burst time of process2
2
3
4
process id 1 arrival time 0 burst time 2 complete time 2 turn around time 2 waiting time 0
process id 2 arrival time 1 burst time 2 complete time 4 turn around time 3 waiting time 1
process id 3 arrival time 5 burst time 3 complete time 8 turn around time 3 waiting time 0
process id 4 arrival time 6 burst time 4 complete time 12 turn around time 6 waiting time 2
average turn around time is 3.500000average working time is 0.750000
Process returned 0 (0x0)  execution time : 141.043 s
Press any key to continue.
```

### b.SJF(non-preemptive)scheduling using array

```
#include<stdio.h>

void findCompletionTime(int processes[], int n, int bt[], int at[], int wt[], int tat[], int rt[], int ct[])

{
    int completion[n];
    int remaining[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;
            }
        }
        completion[i] = currentTime + bt[shortest];
        remaining[shortest] = 0;
        tat[i] = completion[i] - at[i];
        rt[i] = currentTime - at[i];
        ct[i] = completion[i];
        currentTime = completion[i];
        wt[i] = currentTime - at[i];
    }
}
```

```

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] = wt[shortest];
remaining[shortest] = 0;
}

printf("Process\tArrival Time\tBurst Time\tWaiting Time\tTurnaround
Time\tResponse Time\tCompletion Time\n");

for (int i = 0; i < n; i++)
{
    ct[i] = completion[i];
    printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n", processes[i], at[i], bt[i], wt[i], tat[i], rt[i], ct[i]);
}

float avg_tat=tat[0];
for(int i=1;i<n;i++)
{
    avg_tat+=tat[i];
}

printf("\n Average TAT=%f ms",avg_tat/n);

float avg_wt=wt[0];
for(int i=1;i<n;i++)
{
    avg_wt+=wt[i];
}

printf("\n Average WT= %f ms",avg_wt/n);

```

```
}

void main()
{
int n;

printf("Enter the number of processes: ");

scanf("%d", &n);

int processes[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 wt[n], tat[n], rt[n], ct[n];

for (int i = 0; i < n; i++)

rt[i] = -1;

printf("\nSJF (Non-preemptive) Scheduling:\n");

findCompletionTime(processes, n, burst_time, arrival_time, wt, tat, rt, ct);

}

output:
```

```

C:\Users\saisr\OneDrive\Desktop > + v
Enter the number of processes: 4
Enter Process Number:
1
2
3
4
Enter Arrival Time:
0
0
0
0
Enter Burst Time:
6
8
7
3

SJF (Non-preemptive) Scheduling:
Process Arrival Time    Burst Time    Waiting Time   Turnaround Time ResponseTime   Completion Time
1              0            6             3              9                3                 9
2              0            8             16             24               16                24
3              0            7             9              16               9                 16
4              0            3             0              3                0                 3

Average TAT=13.000000 ms
Average WT= 7.000000 ms
Process returned 25 (0x19)  execution time : 57.452 s
Press any key to continue.
|

```

### c.SJF(preemptive)scheduling using array

```

#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;

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 shortest_job_index = available_processes[find_min(rt, available_count)];
    rt[shortest_job_index]--;
    current_time++;

    if (rt[shortest_job_index] == 0) {

        completed_processes++;
        ct[shortest_job_index] = current_time;
        tat[shortest_job_index] = ct[shortest_job_index] - at[shortest_job_index];
        wt[shortest_job_index] = tat[shortest_job_index] - bt[shortest_job_index];
        total_wt += wt[shortest_job_index];
        total_tat += tat[shortest_job_index];
    }
}

```

```

printf("\nProcess\tArrival Time\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting
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 turnaround time: %.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:

```
C:\Users\saisr\OneDrive\Desktop
```

Enter the number of processes: 5  
Enter the arrival time:  
2  
1  
4  
0  
2  
Enter the burst time:  
1  
5  
1  
6  
3  
Process Arrival Time Burst Time Completion Time Turnaround Time Waiting Time  
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  
Process returned 0 (0x0) execution time : 162.144 s  
Press any key to continue.

2. Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time. → Priority (pre-emptive & Non-pre-emptive) → Round Robin (Experiment with different quantum sizes for RR algorithm)

a.priority(non preemptive)scheduling

```
#include<stdio.h>

#define MAX 10

void priority_non_preemptive(int n, int at[], int bt[], int p[]) {

    int ct[MAX] = {0};

    int tat[MAX] = {0};

    int wt[MAX] = {0};

    int total_wt = 0;

    int total_tat = 0;

    int bt_copy[MAX];

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

        bt_copy[i] = bt[i];

    }

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

        for (int j = i + 1; j < n; j++) {

            if (p[i] < p[j]) {

                int temp = at[i];

                at[i] = at[j];

                at[j] = temp;

                temp = bt[i];

                bt[i] = bt[j];

                bt[j] = temp;

                temp = p[i];

                p[i] = p[j];

                p[j] = temp;

            }

        }

    }

    ct[0] = at[0] + bt[0];
```

```

tat[0] = ct[0] - at[0];
wt[0] = tat[0] - bt_copy[0];
total_wt += wt[0];
total_tat += tat[0];
for (int i = 1; i < n; i++) {
    ct[i] = ct[i - 1] + bt[i];
    tat[i] = ct[i] - at[i];
    wt[i] = tat[i] - bt_copy[i];
    total_wt += wt[i];
    total_tat += tat[i];
}
printf("\nProcess\tArrival Time\tBurst Time\tPriority\tCompletion Time\tTurnaround
Time\tWaiting Time\n");
for (int i = 0; i < n; i++) {
    printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n", i+1, at[i], bt_copy[i], p[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;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    int at[MAX], bt[MAX], p[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]);
    }
}

```

```

    }

printf("Enter the priority:\n");

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

    scanf("%d", &p[i]);

}

priority_non_preemptive(n, at, bt, p);

return 0;

}

```

output:

```

C:\Users\sair\OneDrive\Desktop > Enter the number of processes: 4
Enter the arrival time:
0
1
2
4
Enter the burst time:
5
4
2
1
Enter the priority:
10
20
30
40
Process  Arrival Time    Burst Time    Priority      Completion Time Turnaround Time Waiting Time
1        4                5              40            5                  1                 -4
2        2                4              30            7                  5                 1
3        1                2              20            11                 10                8
4        0                1              10            16                 16                15

Average waiting time: 5.00
Average turnaround time: 8.00
Process returned 0 (0x0)  execution time : 108.063 s
Press any key to continue.
|

```

b.Round robin scheduling

```

#include <stdio.h>

struct Process {
    int pid;
    int burst_time;
    int arrival_time;
    int remaining_time;
};

void roundRobin(struct Process processes[], int n, int time_quantum) {
    int remaining_processes = n;
    int current_time = 0;

```

```

int completed[n];
int ct[n], wt[n], tat[n], rt[n];
for (int i = 0; i < n; i++) {
    completed[i] = 0;
}
while (remaining_processes > 0) {
    for (int i = 0; i < n; i++) {
        if (completed[i] == 0 && processes[i].arrival_time <= current_time) {
            if (processes[i].remaining_time > 0) {
                if (processes[i].remaining_time <= time_quantum) {
                    current_time += processes[i].remaining_time;
                    processes[i].remaining_time = 0;
                    completed[i] = 1;
                    remaining_processes--;
                    ct[i] = current_time;
                    tat[i] = ct[i] - processes[i].arrival_time;
                } else {
                    current_time += time_quantum;
                    processes[i].remaining_time -= time_quantum;
                }
                wt[i] = ct[i] - processes[i].arrival_time - processes[i].burst_time;
                rt[i] = wt[i];
            }
        }
    }
    printf("PID\tAT\tBT\tCT\tWT\tTAT\tRT\n");
    float avg_tat = 0, avg_wt = 0;
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n", processes[i].pid, processes[i].arrival_time,
processes[i].burst_time, ct[i], wt[i], tat[i], rt[i]);
    }
}

```

```

avg_tat += tat[i];
avg_wt += wt[i];
}

avg_tat /= n;
avg_wt /= n;
printf("\nAverage Turnaround Time: %.2f\n", avg_tat);
printf("Average Waiting Time: %.2f\n", avg_wt);
}

int main() {
int n, time_quantum;
printf("Enter the number of processes: ");
scanf("%d", &n);
printf("Enter the time quantum: ");
scanf("%d", &time_quantum);
struct Process processes[n];
printf("Enter Arrival Time and Burst Time for each process:\n");
for (int i = 0; i < n; i++) {
printf("Enter Arrival Time for process %d: ", i+1);
scanf("%d", &processes[i].arrival_time);
printf("Enter Burst Time for process %d: ", i+1);
scanf("%d", &processes[i].burst_time);
processes[i].pid = i+1;
processes[i].remaining_time = processes[i].burst_time;
}
roundRobin(processes, n, time_quantum);
}

output:

```

```
C:\Users\saisr\OneDrive\Desktop > + ^

Enter the number of processes: 5
Enter the time quantum: 2
Enter Arrival Time and Burst Time for each process:
Enter Arrival Time for process 1: 0
Enter Burst Time for process 1: 5
Enter Arrival Time for process 2: 1
Enter Burst Time for process 2: 3
Enter Arrival Time for process 3: 2
Enter Burst Time for process 3: 1
Enter Arrival Time for process 4: 3
Enter Burst Time for process 4: 2
Enter Arrival Time for process 5: 4
Enter Burst Time for process 5: 3
PID    AT     BT     CT     WT      TAT     RT
1      0      5      14     9      14      9
2      1      3      12     8      11      8
3      2      1      5      2      3      2
4      3      2      7      2      4      2
5      4      3     13     6      9      6

Average Turnaround Time: 8.20
Average Waiting Time: 5.40

Process returned 0 (0x0)  execution time : 108.783 s
Press any key to continue.
```

3. Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for the processes in each queue.

```
#include <stdio.h>

// Function to find waiting time for FCFS

void findWaitingTime(int processes[], int n, int bt[], int at[], int wt[]) {
    wt[0] = 0;

    for (int i = 1; i < n; i++) {
        wt[i] = bt[i-1] + wt[i-1] - at[i-1];

        if (wt[i] < 0)
            wt[i] = 0;
    }
}

// Function to find turnaround time

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

// Function to implement Round Robin scheduling

void roundRobin(int processes[], int n, int bt[], int at[], int quantum) {
    int wt[n], tat[n], ct[n], total_wt = 0, total_tat = 0;
    int remaining_bt[n];
    int completed = 0;
    int time = 0;

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

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

```

        if (remaining_bt[i] <= quantum) {

            time += remaining_bt[i];

            remaining_bt[i] = 0;

            ct[i] = time;

            completed++;

        } else {

            time += quantum;

            remaining_bt[i] -= quantum;

        }

    }

}

findWaitingTime(processes, n, bt, at, wt);

findTurnaroundTime(processes, n, bt, wt, tat);

printf("Processes Burst Time Arrival Time Waiting Time Turnaround Time Completion Time\n");

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

    printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n", processes[i], bt[i], at[i], wt[i], tat[i], ct[i]);

    total_wt += wt[i];

    total_tat += tat[i];

}

printf("Average Waiting Time (Round Robin) = %f\n", (float)total_wt / n);

printf("Average Turnaround Time (Round Robin) = %f\n", (float)total_tat / n);

}

// Function to implement FCFS scheduling

void fcfs(int processes[], int n, int bt[], int at[]) {

    int wt[n], tat[n], ct[n], total_wt = 0, total_tat = 0;

    findWaitingTime(processes, n, bt, at, wt);

    findTurnaroundTime(processes, n, bt, wt, tat);

    printf("Processes Burst Time Arrival Time Waiting Time Turnaround Time Completion Time\n");

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

        ct[i] = at[i] + bt[i];

```

```

printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\n", processes[i], bt[i], at[i], wt[i], tat[i], ct[i]);

total_wt += wt[i];
total_tat += tat[i];

}

printf("Average Waiting Time (FCFS) = %f\n", (float)total_wt / n);
printf("Average Turnaround Time (FCFS) = %f\n", (float)total_tat / n);

}

int main() {

    int processes[] = {1, 2, 3, 4, 5};

    int n = sizeof(processes) / sizeof(processes[0]);

    int bt[] = {10, 5, 8, 12, 15};

    int at[] = {0, 1, 2, 3, 4};

    int quantum = 2;

    roundRobin(processes, n, bt, at, quantum);

    fcfs(processes, n, bt, at);

    return 0;
}

```

output:

Processes	Burst Time	Arrival Time	Waiting Time	Turnaround Time	Completion Time
P1	10	0	0	10	39
P2	5	1	10	15	23
P3	8	2	14	22	33
P4	12	3	20	32	45
P5	15	4	29	44	50

Average Waiting Time (Round Robin) = 14.600000  
Average Turnaround Time (Round Robin) = 24.600000

Processes	Burst Time	Arrival Time	Waiting Time	Turnaround Time	Completion Time
P1	10	0	0	10	10
P2	5	1	10	15	6
P3	8	2	14	22	10
P4	12	3	20	32	15
P5	15	4	29	44	19

Average Waiting Time (FCFS) = 14.600000  
Average Turnaround Time (FCFS) = 24.600000

Process returned 0 (0x0) execution time : 0.063 s  
Press any key to continue.

4. Write a C program to simulate Real-Time CPU Scheduling algorithms:

a) Rate- Monotonic

```
#include <stdio.h>

// Structure to represent a process

struct Process {
    int execution_time;
    int time_period;
};

// Function to calculate the least common multiple (LCM)

int lcm(int a, int b) {
    int max = (a > b) ? a : b;
    while (1) {
        if (max % a == 0 && max % b == 0)
            return max;
        max++;
    }
}

// Function to check if the set of processes is schedulable

int is_schedulable(struct Process processes[], int n) {
    float utilization = 0.0;
    for (int i = 0; i < n; i++) {
        utilization += (float)processes[i].execution_time / processes[i].time_period;
    }
    return utilization <= 1.0;
}

int main() {
    struct Process processes[] = {
        {3, 20}, // P1
        {2, 5}, // P2
        {2, 10} // P3
    };
}
```

```

int n = sizeof(processes) / sizeof(processes[0]);


// Check if the processes are schedulable

if (!is_schedulable(processes, n)) {
    printf("The given set of processes is not schedulable.\n");
    return 0;
}

// Calculate the scheduling time (LCM of time periods)

int scheduling_time = lcm(processes[0].time_period, processes[1].time_period);
scheduling_time = lcm(scheduling_time, processes[2].time_period);

// Display the execution order

printf("Execution order:\n");

for (int t = 0; t < scheduling_time; t++) {
    if (t % processes[1].time_period == 0)
        printf("P2 ");
    if (t % processes[2].time_period == 0)
        printf("P3 ");
    if (t % processes[0].time_period == 0)
        printf("P1 ");
}
printf("\n");

return 0;
}

output:

```

```
C:\Users\saisr\OneDrive\Desktop > Execution order:  
P2 P3 P1 P2 P2 P3 P2  
Process returned 0 (0x0) execution time : 0.053 s  
Press any key to continue.
```

b) Earliest-deadline First

```
#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);
    //LCM
    int l = lcmul (pt, n);
    printf ("\nEarliest Deadline Scheduling:\n");
    printf ("PID\t Burst\tDeadline\tPeriod\n");
    for (int i = 0; i < n; i++)
        printf ("%d\t%d\t%d\t%d\n", proc[i], b[i], d[i], pt[i]);
    printf ("Scheduling occurs for %d ms\n", l);
    //EDF
}

```

```

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 ("%dms : Task %d is running.\n", time, proc[taskToExecute]);
        rem[taskToExecute]--;
    }
}

```

```

        }

    else

    {

        printf ("%dms: CPU is idle.\n", time);

    }

    time++;

}

}

```

output:

```

C:\Users\sair\OneDrive\Desktop > +
Enter the number of processes:3
Enter the CPU burst times:
3
2
2
Enter the deadlines:
7
4
8
Enter the time periods:
20
5
10
Earliest Deadline Scheduling:
PID      Burst  Deadline      Period
2          2       4           5
1          3       7           20
3          2       8           10
Scheduling occurs for 20 ms

0ms : Task 2 is running.
1ms : Task 2 is running.
2ms : Task 1 is running.
3ms : Task 1 is running.
4ms : Task 1 is running.
5ms : Task 3 is running.
6ms : Task 3 is running.
7ms : Task 2 is running.
8ms : Task 2 is running.
9ms: CPU is idle.
10ms : Task 2 is running.
11ms : Task 2 is running.

```

```

C:\Users\sair\OneDrive\Desktop > +
10
Earliest Deadline Scheduling:
PID      Burst  Deadline      Period
2          2       4           5
1          3       7           20
3          2       8           10
Scheduling occurs for 20 ms

0ms : Task 2 is running.
1ms : Task 2 is running.
2ms : Task 1 is running.
3ms : Task 1 is running.
4ms : Task 1 is running.
5ms : Task 3 is running.
6ms : Task 3 is running.
7ms : Task 2 is running.
8ms : Task 2 is running.
9ms: CPU is idle.
10ms : Task 2 is running.
11ms : Task 2 is running.
12ms : Task 3 is running.
13ms : Task 3 is running.
14ms: CPU is idle.
15ms : Task 2 is running.
16ms : Task 2 is running.
17ms: CPU is idle.
18ms: CPU is idle.
19ms: CPU is idle.

Process returned 20 (0x14)  execution time : 63.708 s
Press any key to continue.
|
```

c) 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 // Duration of each time unit in milliseconds

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;
            }
        }
        completed_tasks++;
    }

    // Calculate time span in milliseconds
    *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);
    return 1;
}

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:

```

C:\Users\sair\OneDrive\Desktop > +
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 1 is running
Time 1-2: Task 3 is running
Time 2-3: Task 2 is running

Time span of the Gantt chart: 300 milliseconds

Process returned 0 (0x0)   execution time : 47.068 s
Press any key to continue.
|
```

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

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

#define BUFFER_SIZE 5
#define MAX_ITEMS 20

int buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
int produced_count = 0;
int consumed_count = 0;
sem_t mutex;
sem_t full;
sem_t empty;

void* producer(void* arg) {
    int item = 1;
    while (produced_count < MAX_ITEMS) {
        sem_wait(&empty);
        sem_wait(&mutex);
        buffer[in] = item;
        printf("Produced: %d\n", item);
        item++;
        in = (in + 1) % BUFFER_SIZE;
        produced_count++;
        sem_post(&mutex);
        sem_post(&full);
    }
    pthread_exit(NULL);
}

void* consumer(void* arg) {
    while (consumed_count < MAX_ITEMS) {
        sem_wait(&full);
        sem_wait(&mutex);
        int item = buffer[out];
        printf("Consumed: %d\n", item);
        out = (out + 1) % BUFFER_SIZE;
        consumed_count++;
        sem_post(&mutex);
        sem_post(&empty);
    }
    pthread_exit(NULL);
}

int main() {
    pthread_t producerThread, consumerThread;
    sem_init(&mutex, 0, 1);
    sem_init(&full, 0, 0);
    sem_init(&empty, 0, BUFFER_SIZE);
```

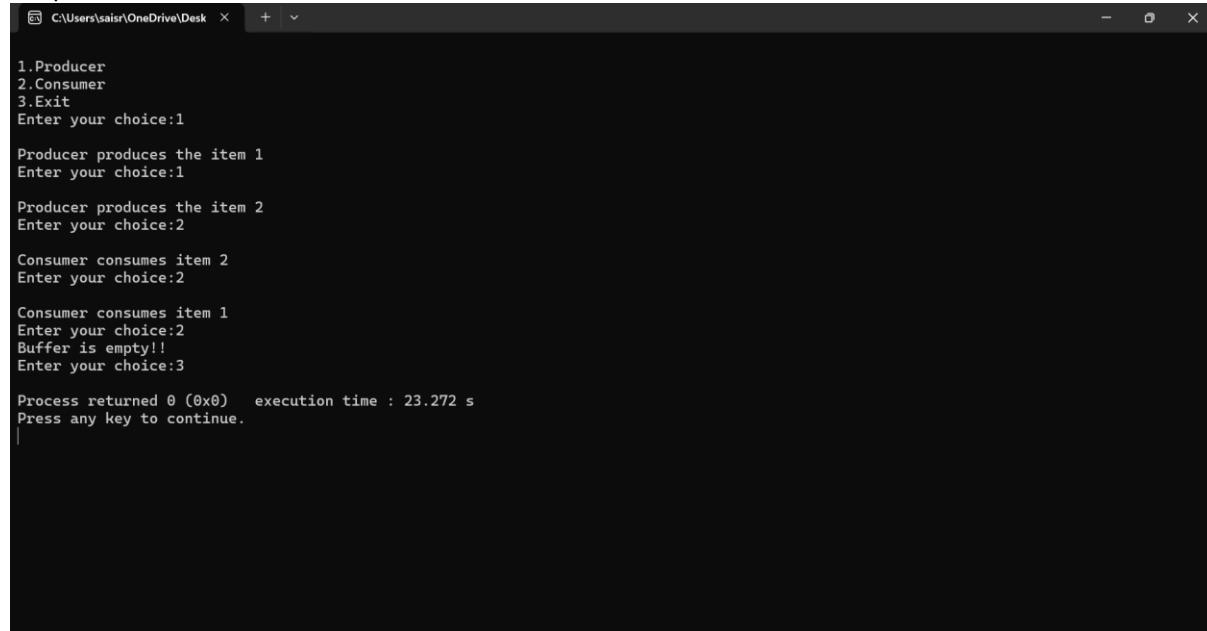
```
pthread_create(&producerThread, NULL, producer, NULL);
pthread_create(&consumerThread, NULL, consumer, NULL);

pthread_join(producerThread, NULL);
pthread_join(consumerThread, NULL);

sem_destroy(&mutex);
sem_destroy(&full);
sem_destroy(&empty);

return 0;
}
```

Output:



```
C:\Users\saisr\OneDrive\Desktop > + ^

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:2

Consumer consumes item 2
Enter your choice:2

Consumer consumes item 1
Enter your choice:2
Bufffer is empty!!
Enter your choice:3

Process returned 0 (0x0)  execution time : 23.272 s
Press any key to continue.
```

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

```
#include <stdio.h>

#include <stdlib.h>

#define MAX_PHILOSOPHERS 5

void allow_one_to_eat(int hungry[], int n) {

    int isWaiting[MAX_PHILOSOPHERS];

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

        isWaiting[i] = 1;

    }

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

        printf("P %d is granted to eat\n", hungry[i]);

        isWaiting[hungry[i]] = 0;

        for (int j = 0; j < n; j++) {

            if (isWaiting[hungry[j]]) {

                printf("P %d is waiting\n", hungry[j]);

            }

        }

        for (int k = 0; k < n; k++) {

            isWaiting[k] = 1;

        }

        isWaiting[hungry[i]] = 0;

    }

}

void allow_two_to_eat(int hungry[], int n) {

    if (n < 2 || n > MAX_PHILOSOPHERS) {

        printf("Invalid number of philosophers.\n");

        return;

    }

    for (int i = 0; i < n - 1; i++) {

        for (int j = i + 1; j < n; j++) {

            printf("P %d and P %d are granted to eat\n", hungry[i], hungry[j]);

        }

    }

}
```

```

for (int k = 0; k < n; k++) {
    if (k != i && k != j) {
        printf("P %d is waiting\n", hungry[k]);
    }
}
}

int main() {
    int total_philosophers, hungry_count;
    int hungry_positions[MAX_PHILOSOPHERS];
    printf("DINING PHILOSOPHER PROBLEM\n");
    printf("Enter the total no. of philosophers: ");
    scanf("%d", &total_philosophers);
    if (total_philosophers > MAX_PHILOSOPHERS || total_philosophers < 2) {
        printf("Invalid number of philosophers.\n");
        return 1;
    }
    printf("How many are hungry: ");
    scanf("%d", &hungry_count);
    if (hungry_count < 1 || hungry_count > total_philosophers) {
        printf("Invalid number of hungry philosophers.\n");
        return 1;
    }
    for (int i = 0; i < hungry_count; i++) {
        printf("Enter philosopher %d position: ", i + 1);
        scanf("%d", &hungry_positions[i]);
        if (hungry_positions[i] < 0 || hungry_positions[i] >= total_philosophers) {
            printf("Invalid philosopher position.\n");
            return 1;
        }
    }
}

```

```
}

int choice;

while (1) {

    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:
            allow_one_to_eat(hungry_positions, hungry_count);
            break;

        case 2:
            allow_two_to_eat(hungry_positions, hungry_count);
            break;

        case 3:
            exit(0);

        default:
            printf("Invalid choice\n");
    }
}

return 0;
}

output:
```

```
C:\Users\saisr\OneDrive\Desktop DINING PHILOSOPHER PROBLEM
Enter the total no. of philosophers: 5
How many are hungry: 2
Enter philosopher 1 position: 1
Enter philosopher 2 position: 4

1. One can eat at a time
2. Two can eat at a time
3. Exit
Enter your choice: 1
P 1 is granted to eat
P 4 is waiting
P 4 is granted to eat

1. One can eat at a time
2. Two can eat at a time
3. Exit
Enter your choice: 2
P 1 and P 4 are granted to eat

1. One can eat at a time
2. Two can eat at a time
3. Exit
Enter your choice: 3

Process returned 0 (0x0)   execution time : 59.936 s
Press any key to continue.
```

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

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

```
int main() {
    int n, m;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    printf("Enter the number of resources: ");
    scanf("%d", &m);
    int available[m];
    printf("Enter the available resources: ");
    for (int i = 0; i < m; i++) {
        scanf("%d", &available[i]);
    }
    int maximum[n][m];
    printf("Enter the maximum resources for each process:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            scanf("%d", &maximum[i][j]);
        }
    }
    int allocation[n][m];
    printf("Enter the allocated resources for each process:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            scanf("%d", &allocation[i][j]);
        }
    }
    int need[n][m];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            need[i][j] = maximum[i][j] - allocation[i][j];
        }
    }
}
```

```

    }

}

printf(" Process Allocation Max Need      \n");
for (int i = 0; i < n; i++) {
    printf(" | P%d    | ", i + 1);
    for (int j = 0; j < m; j++) {
        printf("%d ", allocation[i][j]);
    }
    printf(" | ");
    for (int j = 0; j < m; j++) {
        printf("%d ", maximum[i][j]);
    }
    printf(" | ");
    for (int j = 0; j < m; j++) {
        printf("%d ", need[i][j]);
    }
    printf(" | \n");
}

int work[m];
for (int i = 0; i < m; i++) {
    work[i] = available[i];
}

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

int safeSequence[n];
int count = 0;
int safe = 1;
while (count < n) {
    int found = 0;

```

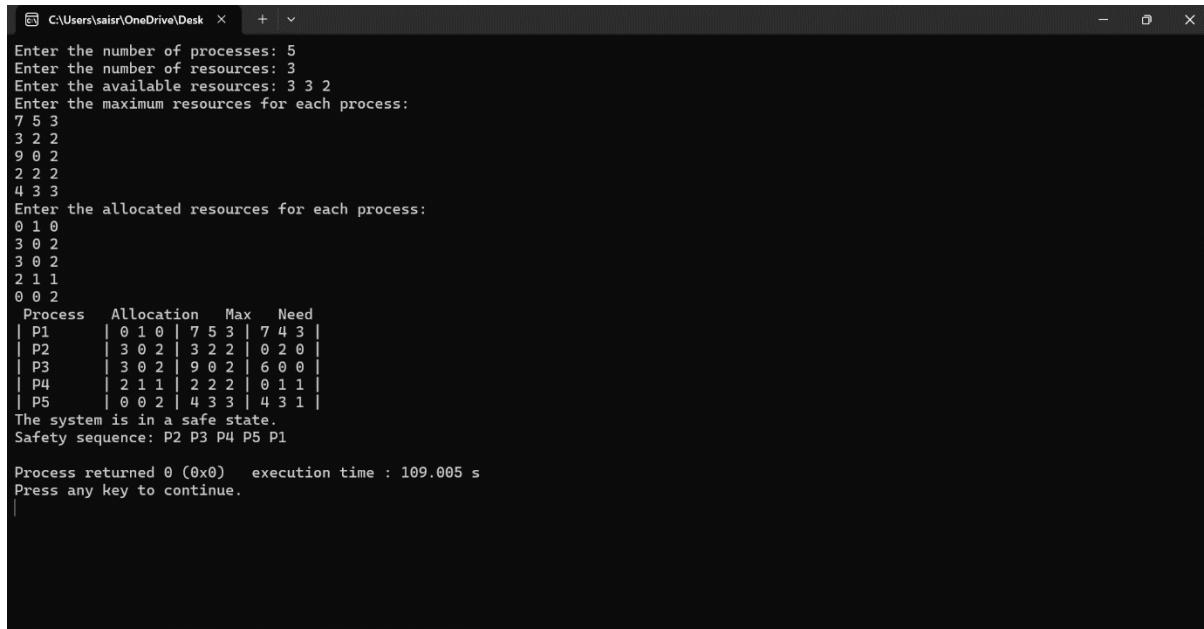
```

for (int i = 0; i < n; i++) {
    if (finish[i] == 0) {
        int j;
        for (j = 0; j < m; j++) {
            if (need[i][j] > work[j]) {
                break;
            }
        }
        if (j == m) {
            for (j = 0; j < m; j++) {
                work[j] += allocation[i][j];
            }
            finish[i] = 1;
            safeSequence[count++] = i;
            found = 1;
        }
    }
}
if (!found) {
    safe = 0;
    break;
}
if (safe) {
    printf("The system is in a safe state.\n");
    printf("Safety sequence: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", safeSequence[i] + 1);
    }
    printf("\n");
} else {

```

```
    printf("The system is in an unsafe state and might lead to deadlock.\n");  
}  
  
return 0;  
}
```

output:



```
C:\Users\saisr\OneDrive\Desktop  
Enter the number of processes: 5  
Enter the number of resources: 3  
Enter the available resources: 3 3 2  
Enter the maximum resources for each process:  
7 5 3  
3 2 2  
9 0 2  
2 2 2  
4 3 3  
Enter the allocated resources for each process:  
0 1 0  
3 0 2  
3 0 2  
2 1 1  
0 0 2  
Process Allocation Max Need  
| P1 | 0 1 0 | 7 5 3 | 7 4 3 |  
| P2 | 3 0 2 | 3 2 2 | 0 2 0 |  
| P3 | 3 0 2 | 9 0 2 | 6 0 0 |  
| P4 | 2 1 1 | 2 2 2 | 0 1 1 |  
| P5 | 0 0 2 | 4 3 3 | 4 3 1 |  
The system is in a safe state.  
Safety sequence: P2 P3 P4 P5 P1  
Process returned 0 (0x0)  execution time : 109.005 s  
Press any key to continue.
```

8. 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 maximum 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",&max[i][j]);
        }
    }
    printf("Enter the allocated 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",&all[i][j]);
        }
    }
    printf("Enter the available number of each type of resource:\n");
    for(j=0;j<m;j++)
    {
        scanf("%d",&ava[j]);
    }
    for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++)
        {
            need[i][j]=max[i][j]-all[i][j];
        }
    }
    for(i=0;i<n;i++)
    {
        finish[i]=0;
    }
    while(flag)
    {
        flag=0;
        for(i=0;i<n;i++)
        {
            c=0;
            for(j=0;j<m;j++)
            {
                if(finish[i]==0 && need[i][j]<=ava[j])
                {
                    c++;
                    if(c==m)
                    {
                        for(j=0;j<m;j++)
                        {
                            ava[j]=ava[j]+all[i][j];
                            all[i][j]=0;
                            need[i][j]=max[i][j];
                        }
                        finish[i]=1;
                    }
                }
            }
        }
    }
}
```

```

{
    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 occurred:\n");
    printf("The deadlock processes are:\n");
    for(i=0;i<n;i++)
    {
        printf("P%d ",dead[i]);
    }
}
else
    printf("No deadlock has occurred!\n");
}

```

**output:**

```
C:\Users\saisr\OneDrive\Desktop + v
Enter the number of processes and number of types of resources:
5 3
Enter the maximum number of each type of resource needed by each process:
7 5 3
3 2 2
9 0 2
2 2 4
4 3 3
Enter the allocated number of each type of resource needed by each process:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter the available number of each type of resource:
3 3 2
No deadlock has occurred!
Process returned 0 (0x0) execution time : 120.785 s
Press any key to continue.
|
```

9. Write a C program to simulate the following contiguous memory allocation techniques

a) Worst-fit

b) Best-fit

c) First-fit

```
#include <stdio.h>

struct Block {
    int block_no;
    int block_size;
    int is_free; // 1 for free, 0 for allocated
};

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%d\t%d\t%d\t%d\n", files[i].file_no, files[i].file_size, blocks[j].block_no,
blocks[j].block_size, blocks[j].block_size - files[i].file_size);
                break;
            }
        }
    }
}

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:\tBlock_no:\tBlock_size:\tFragment\n");

    for (int i = 0; i < n_files; i++) {
        int worst_fit_block = -1;
        int max_fragment = -1; // Initialize with a negative value
        for (int j = 0; j < n_blocks; j++) {
            if (blocks[j].is_free && blocks[j].block_size >= files[i].file_size) {
                int fragment = blocks[j].block_size - files[i].file_size;
                if (fragment > max_fragment) {
                    max_fragment = fragment;
                    worst_fit_block = j;
                }
            }
        }
        if (worst_fit_block != -1) {
```

```

        blocks[worst_fit_block].is_free = 0;
        printf("%d\t%d\t%d\t%d\t%d\n", files[i].file_no, files[i].file_size,
blocks[worst_fit_block].block_no, blocks[worst_fit_block].block_size, max_fragment);
    }
}
}

void bestFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {
    printf("Memory Management Scheme - Best Fit\n");
    printf("File_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment\n");

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

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");

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

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

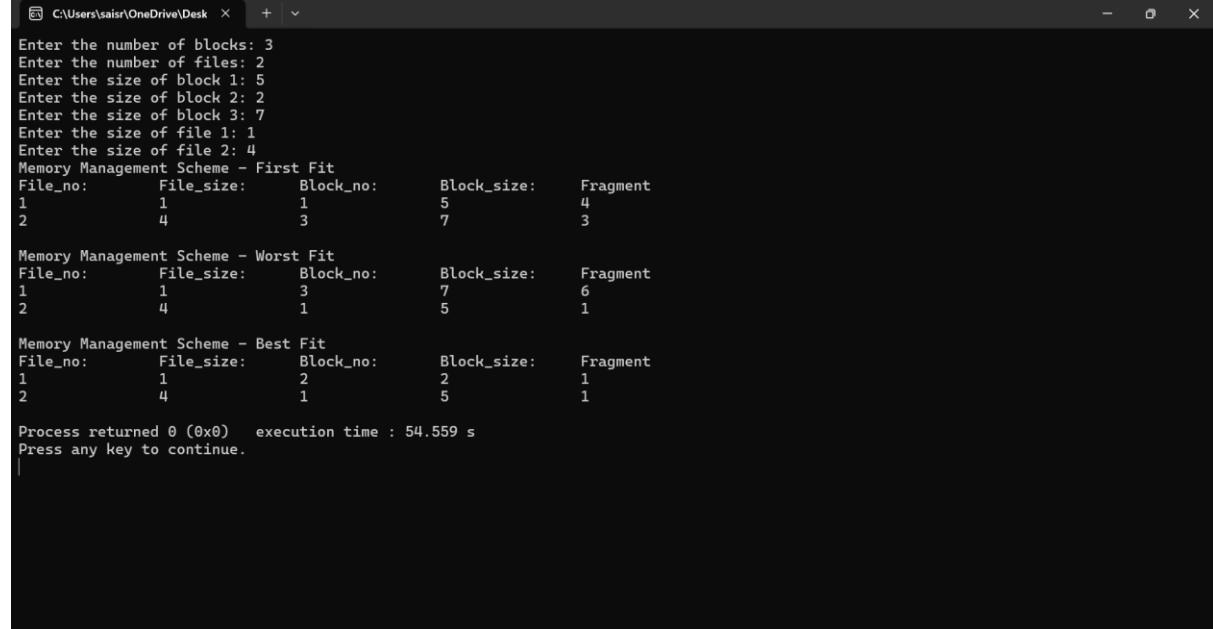
// Reset blocks for best fit
for (int i = 0; i < n_blocks; i++) {
    blocks[i].is_free = 1;
}

bestFit(blocks, n_blocks, files, n_files);

return 0;
}

```

**output:**



The screenshot shows a terminal window with the following content:

```

C:\Users\saisr\OneDrive\Desktop X + v
Enter the number of blocks: 3
Enter the number of files: 2
Enter the size of block 1: 5
Enter the size of block 2: 2
Enter the size of block 3: 7
Enter the size of file 1: 1
Enter the size of file 2: 4
Memory Management Scheme - First Fit
File_no:      File_size:      Block_no:      Block_size:      Fragment
1            1             1              5                4
2            4             3              7                3

Memory Management Scheme - Worst Fit
File_no:      File_size:      Block_no:      Block_size:      Fragment
1            1             3              7                6
2            4             1              5                1

Memory Management Scheme - Best Fit
File_no:      File_size:      Block_no:      Block_size:      Fragment
1            1             2              2                1
2            4             1              5                1

Process returned 0 (0x0)  execution time : 54.559 s
Press any key to continue.
|
```

10. 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);
}

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]) {
                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("\nPages: ");
    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;
}

```

**output:**

```

C:\Users\saisr\OneDrive\Desktop > Enter the number of pages: 20
Enter the pages: 7
0
1
2
0
3
0
4
2
3
0
3
2
1
2
0
1
7
0
1
Enter the frame capacity: 3
Pages: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

FIFO Page Replacement Process:
7 - PF No. 1
7 0 - PF No. 2
7 0 1 PF No. 3
2 0 1 PF No. 4
2 0 1
2 3 1 PF No. 5
2 3 0 PF No. 6

```

```
C:\Users\saisr\OneDrive\Desktop + × - ⌂ X  
2 3 0 PF No. 6  
4 3 0 PF No. 7  
4 2 0 PF No. 8  
4 2 3 PF No. 9  
0 2 3 PF No. 10  
0 2 3  
0 2 3  
0 1 3 PF No. 11  
0 1 2 PF No. 12  
0 1 2  
0 1 2  
7 1 2 PF No. 13  
7 0 2 PF No. 14  
7 0 1 PF No. 15  
Total Page Faults using FIFO: 15  
  
LRU Page Replacement Process:  
7 - - PF No. 1  
0 - - PF No. 2  
0 1 - PF No. 3  
0 1 2 PF No. 4  
0 1 2  
0 3 2 PF No. 5  
0 3 2  
0 3 4 PF No. 6  
0 2 4 PF No. 7  
3 2 4 PF No. 8  
3 2 0 PF No. 9  
3 2 0  
3 2 0  
3 2 1 PF No. 10  
3 2 1  
0 2 1 PF No. 11  
  
C:\Users\saisr\OneDrive\Desktop + × - ⌂ X  
0 2 1 PF No. 11  
0 2 1  
0 7 1 PF No. 12  
0 7 1  
0 7 1  
Total Page Faults using LRU: 12  
  
Optimal Page Replacement Process:  
7 - - PF No. 1  
7 0 - PF No. 2  
7 0 1 PF No. 3  
2 0 1 PF No. 4  
2 0 1  
2 0 3 PF No. 5  
2 0 3  
2 4 3 PF No. 6  
2 4 3  
2 4 3  
2 0 3 PF No. 7  
2 0 3  
2 0 3  
2 0 1 PF No. 8  
2 0 1  
2 0 1  
2 0 1  
7 0 1 PF No. 9  
7 0 1  
7 0 1  
Total Page Faults using Optimal: 9  
  
Process returned 0 (0x0) execution time : 73.003 s  
Press any key to continue.
```

Scheduling Programming  
Scheduling Programming

Lab-1 8/5/24

- (1) write a c program to simulate the following non preemptive cpu scheduling algorithm  
algorithm 2 find turn around time and waiting time

(i) FCFS

```
#include <stdio.h>
int main(){
    int n, i;
    float atime = 0, awt = 0;
    printf("enter the number of process");
    scanf("%d", &n);
    int atime[n], btime[n], chime3[n], tatime4[n], wtimes[n];
    printf("enter arrival time of process");
    for(i=0; i<n; i++){
        scanf("%d", &atime[i]);
    }
    printf("enter burst time of process");
    for(i=0; i<n; i++){
        scanf("%d", &btime[i]);
    }
    for(i=0; i<n; i++){
        if(i==0){
            chime3[i] = atime[i] + btime[i];
        }
        else{
            if(chime3[i-1] < atime[i]){
                chime3[i] = (atime[i] - chime3[i-1]) + chime3[i-1] + btime[i];
            }
            else{
                chime3[i] = chime3[i-1] + btime[i];
            }
        }
    }
    for(i=0; i<n; i++){
        tatime4[i] = chime3[i] - atime[i];
    }
}
```

```

}
for(i=0; i<n; i++){
    wtimes[i] = tattime4[i] - btime2[i];
}

for(i=0; i<n; i++){
    atat = atat + tattime4[i];
}

atat = (atat/n);

for(i=0; i<n; i++){
    awt = awt + wtimes[i];
}

awt = (awt/n);

for(i=0; i<n; i++){
    printf("process id %d arrival time %d burst time %d complete
          time %d turn around time %d waiting time %d \n", i,
          atime1[i], btime2[i], ctime3[i], tattime4[i], wtimes[i]);
}

printf("average turn around time is %.f", atat);
printf("average working time is %.f", awt);
}

```

### Output:

enter the number of process4  
enter arrival time of process 0

1

5

6

enter burst time of process 2

2

3

4

process id 1 arrival time 0 burst time 2 complete time 2 turn around time 2  
waiting time 0

process id 2 arrival time 1 burst time 2 complete time 4 turn around time 3  
Waiting time 1

Process id 3 arrival time 5 burst time 3 complete time 8 turn around time 3  
Waiting time 0

process id 4 Arrival time 6 burst time 4 complete time 12 turn around time 6  
waiting time 2

average

average turn around time

SJF (shortest job first)

non preemptive

(b) SJF (shortest job first)

non preemptive

1 #include <stdio.h>

int main()

int n, i, j, temp1, temp2;

float atat = 0, awt = 0;

printf(" enter the numbe

scanf("%d", &n);

int atime[n], btime[n];

printf(" enter process

printf(" enter process

for (i=0; i<n; i++) {

scanf("%d", &pid[i]);

}

printf(" enter arrival t

for (i=0; i<n; i++) {

scanf("%d", &atime[i]);

}

printf(" enter burst t

for (i=0; i<n; i++) {

scanf("%d", &btime[i]);

}

for (i=0; i<n-1; i++) {

for (j=i+1; j<n; j++) {

if (btime2[j] >

temp1 = bt

btime2[j]

btime2[j+1]

temp2 = P

Pid[j] =

Pid[j+1]

}

}

average turn around time is 3.50000 average waiting time is 0.75000

```
#include <stdio.h>
int main(){
    int n, i, j, temp1, temp2;
    float totat = 0, awt = 0;
    printf(" enter the number of process ");
    scanf("%d", &n);
    int atime1[n], btime2[n], chime3[n], tattime4[n], wtimes[n], pid[n];
    printf(" enter process id's ");
    printf(" enter arrival times ");
    for(i=0; i<n; i++){
        scanf("%d", &pid[i]);
    }
    printf(" enter burst times ");
    for(i=0; i<n; i++){
        scanf("%d", &btime2[i]);
    }
    for(i=0; i<n-1; i++){
        for(j=i+1; j<n; j++){
            if(btime2[j] > btime2[i]){
                temp1 = btime2[i];
                btime2[i] = btime2[j];
                btime2[j] = temp1;
                temp2 = pid[i];
                pid[i] = pid[j];
                pid[j] = temp2;
            }
        }
    }
    for(i=0; i<n; i++){
        totat += tattime4[i];
        awt += (tattime4[i] - btime2[i]) / n;
    }
    printf(" turn around time 1 ");
    printf(" turn around time 2 ");
    printf(" turn around time 3 ");
    printf(" turn around time 4 ");
    printf(" turn around time 5 ");
    printf(" turn around time 6 ");
}
```

```

}
for(i=0; i<n; i++){
    if(i==0){
        ctime3[i] = atime1[i] + btime2[i];
    }
    else{
        if(ctimes[i-1] < atime1[i]){
            ctimes[i] = (atime1[i] - ctimes[i-1]) + ctime3[i-1] +
                btime2[i];
        }
        else{
            ctime3[i] = ctime3[i-1] + btime2[i];
        }
    }
}

for(i=0; i<n; i++){
    tattime4[i] = ctimes[i] - atime1[i];
}

for(i=0; i<n; i++){
    wtimes[i] = tattime4[i] - btime2[i];
}

for(i=0; i<n; i++){
    atat = atat + tattime4[i];
}

atat = (atat/n);

for(i=0; i<n; i++){
    awt = awt + wtimes[i];
}

awt = (awt/n);

for(i=0; i<n; i++){
    printf("process id %d arrival time %d burst time %d complete
time %d turnaround time %d waiting time %d\n",
pid[i], atime1[i], btime2[i], ctime3[i], tattime4[i],
wtimes[i]);
}

```

```

}
printf("average turn around time : %f", awt);
printf("average waiting time : %f", awt);
}

int output();
enter the number of processes : 4
enter process ids : 1 2 3 4
enter arrival times : 0 0 0 0
enter burst times : 6 3 3 4
process id 4 arrival time : 0
process id 1 arrival time : 0
process id 3 arrival time : 0
process id 2 arrival time : 0
average turnaround time : 3.5
average waiting time : 1.5
#include<stdio.h>
#define max 10
void sif(int n, int at[])
{
    int ct[max];
    int tat[max];
    int wr[max];
    int total_wt;
    int total_wt=0;
    int total_tat=0;
    for(int i=0; i<n; i++)
    {
        ct[i]=-1;
    }
}

```

```

        wtimes[i]);
    }
    printf("average turnaround time %f\n", atat);
    printf("average waiting time %f\n", awt);
    return 0;
}

int main()
{
    int n, i, j, k, l, m, p, q, r, s, t, u, v, w, x, y, z;
    float atat, awt, atime, btimes, ttimes, wtimes, atime3, btimes3, ttimes3, wtimes3;
    char ch;

    printf("enter the number of processes ");
    scanf("%d", &n);

    printf("enter process ids ");
    for(i = 1; i <= n; i++)
        scanf("%d", &p);

    printf("enter arrival times ");
    for(i = 1; i <= n; i++)
        scanf("%f", &atime);

    printf("enter burst times ");
    for(i = 1; i <= n; i++)
        scanf("%f", &btimes);

    for(i = 1; i <= n; i++)
    {
        atime3 = atime[i];
        btimes3 = btimes[i];
        ttimes3 = atime3 + btimes3;
        wtimes3 = 0;
        for(j = 1; j < i; j++)
            wtimes3 += btimes[j];
        atat = (atime3 + btimes3) / n;
        awt = (wtimes3) / n;
    }

    printf("process id %d arrival time %f burst time %f complete time %f turnaround time %f waiting time %f\n",
           p, atime, btimes, ttimes, atat, awt);
    printf("process id %d arrival time %f burst time %f complete time %f turnaround time %f waiting time %f\n",
           p, atime, btimes, ttimes, atat, awt);
    printf("process id %d arrival time %f burst time %f complete time %f turnaround time %f waiting time %f\n",
           p, atime, btimes, ttimes, atat, awt);
    printf("process id %d arrival time %f burst time %f complete time %f turnaround time %f waiting time %f\n",
           p, atime, btimes, ttimes, atat, awt);
    printf("process id %d arrival time %f burst time %f complete time %f turnaround time %f waiting time %f\n",
           p, atime, btimes, ttimes, atat, awt);
    printf("average turnaround time %f\n", atat);
    printf("average waiting time %f\n", awt);

    #include <stdio.h>
    #define max 10

    void SJF(int n, int atf[], int btl[])
    {
        int ct[max];
        int tat[max];
        int wt[max];
        int total_wt;
        int total_tat = 0;
        for(int i = 0; i < n; i++)
        {
            ct[i] = -1;
        }
    }

```

```
for(int i=0; i<n; i++)
    for(int j=0; j<n; j++)
        if(bt[j] > bt[j+i])
            int temp = bt[j];
            bt[j] = bt[j+i];
            bt[j+i] = temp;
            int temp_at = at[i];
            at[i] = at[j+i];
            at[j+i] = temp_at;
```

```
for(int i=0; i<n; i++)
    if(current_time < at[i])
        current_time = at[i];
    ct[i] =
```

1000

~~SJF (shortest job first)~~

~~Preemptive~~

```
#include <stdio.h>
#define max 10
int find_min(int arr[], int n){
    int min = arr[0];
    int index = 0;
```

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

void SJF\_Preemptive(int n, int

```
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;
int completed_processes;
while(completed_processes <
```

```
int available_processes;
int available_processes;
for(int i=0; i<n; i++)
    if(at[i] <= current_time)
        available_processes++;
        available_processes++;
        available_processes++;
        available_processes++;

if(available_processes == n)
    current_time = at[n-1];
```

```

for(int i=0; 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;
    int completed_processes = 0;
    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;
                rt[i] = rt[i] - 1;
                available_count++;
            }
        }
        if(available_count == 0){
            current_time++;
        }
    }
}

```

```

        continue;

    }

    int shortest_job_index = available_processes[find_min(rt, + available_time)];
    rt[shortest_job_index] -= current_time++;
    if(rt[shortest_job_index] == 0) {
        completed_processes++;
        ct[shortest_job_index] = current_time;
        tat[shortest_job_index] = ct[shortest_job_index] - at[shortest_job_index];
        wt[shortest_job_index] = tat[shortest_job_index] - bt[shortest_job_index];
        total_wt += wt[shortest_job_index];
        total_tat += tat[shortest_job_index];
    }
}

printf("\n process \t arrival time \t burst time \t completion time \t turnaround \n
       . 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", it[i],
           at[i], bt[i], ct[i], tat[i], wt[i]);
}

printf("\n average waiting time %.2f ", (float)total_wt/n);
printf("\n average turnaround time %.2f ", (float)total_tat/n);

int main(){
    int n;
    printf("enter the number of processes ");
    scanf("%d", &n);
    int at[max], bt[max];
    int ot[max], bt[max];
}

```

```

printf("enter the burst
for(int i=0;
printf(" enter the arry
for(int i=0; i<n; i++)
scanf("%d", &a[i]);
}
printf("enter the burst
for(int i=0; i<n; i++)
scanf("%d", &b[i]);
}
Sif_Preemptive(n, a, b);

```

~~input~~  
output  
enter the number of processes  
output  
enter the number of processes  
enter the arrival time

2  
 1  
 4  
 0  
 2  
 enter the burst time  
 1  
 5  
 1  
 6  
 3

```

printf("enter the burst time\n");
for(int i=0;
    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);
}

```

~~output~~

~~enter the number of processes~~

~~output~~

~~enter the number of process 5~~

~~enter the arrival time~~

2  
1  
4  
0  
2  
enter the burst time

process	arrival time	burst time	completion time	turnaround time	waiting time
1	2	1	3	1	0
2	1	5	7	6	1
3	4	1	8	7	3
4	0	6	13	13	7
5	2	3	16	14	11

~~average waiting time 4/10~~

```

    ct[0] = at[0] + bt[0];
    tot[0] = ct[0] - at[0];
    wt[0] = tot[0] - bt - copy[0];
    total_wt += wt[0];
    total_tat += tot[0];
    for(i=1; i<n; i++)
    {
        ct[i] = ct[i-1] + bt[i];
        tot[i] = ct[i] - at[i];
        wt[i] = tot[i] - bt - copy[i];
        total_wt += wt[i];
        total_tat += tot[i];
    }
    printf("\n process \t arrival time \t burst time \t priority \t completion time
           \t turnaroundtime \t waiting time\n");
    for(i=0; i<n; i++)
    {
        printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n",
               i+1, at[i], bt - copy[i], p[i], ct[i], tot[i], wt[i]);
    }
    printf("\n average waiting time %.2f", (float)total_wt/n);
    printf("\n average turnaroundtime %.2f", (float)total_tat/n);
}

int main()
{
    int n;
    printf("enter the number of processes ");
    scanf("%d", &n);
}

```

```

int at[max], bt[max], p[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]);
}
printf("enter the priority\n");
printf("enter the priority\n");
for(int i=0; i<n; i++){
    scanf("%d", &p[i]);
}
priority_non_preemptive(n, at, bt, p);

```

Output:

enter the number of process 4  
 enter the arrival time

0  
 1  
 2  
 3  
 4  
 enter the burst time

5  
 4  
 2  
 1  
 enter the burst time  
 5  
 4  
 2  
 1

1  
 enter the priority  
 10  
 20  
 30  
 40  
 process arrival time burst time  
 1 4 5  
 2 2 4  
 3 1 2  
 4 3 1  
 average waiting time 5.00  
 average turnaround time 8.00

Enter the priority

10  
20  
30  
40  
50

process	arrival time	bursttime	priority	tat	wt	ct
1	4	5	40	1	4	5
2	2	4	30	5	1	7
3	1	2	20	10	8	11
4	0	1	10	16	15	16

average waiting time 5.00

average turnaround time 8.00

1) (b) round robin (experiment with different quantum sizes for n algorithm)  
2) round robin scheduling

```

#include <stdio.h>
struct process{
    int pid;
    int burst_time;
    int arrival_time;
    int remaining_time;
};

void roundRobin(struct process processes[], int n, int time_quantum) {
    int remaining_processes = n;
    int current_time = 0;
    int completed[n];
    int ct[n], wt[n], tat[n], rt[n];
    for(int i=0; i<n; i++){
        completed[i] = 0;
    }
    while(remaining_processes > 0){
        for(int i=0; i<n; i++){
            if(completed[i] == 0 && processes[i].arrival_time <= current_time){
                if((processes[i].remaining_time > 0) && (processes[i].remaining_time <= time_quantum)){
                    current_time += processes[i].remaining_time;
                    processes[i].remaining_time = 0;
                    completed[i] = 1;
                    remaining_processes--;
                    ct[i] = current_time;
                    tat[i] = ct[i] - processes[i].arrival_time;
                } else {
                    ct[i] = current_time;
                    tat[i] = ct[i] - processes[i].arrival_time;
                }
            }
        }
    }
}

```

```

    wt[i] = ct[i] - processes[i];
    rt[i] = wt[i];
}

if ("pid" <= i) {
    avg_tat = 0, avg_wt = 0;
    for (int j = 0; j < n; j++) {
        printf ("%d\t%dt\t%dt\t%dt\t%dt\n",

```

```
avg_lat += lat[i];  
avg_wt += wt[i];
```

```

avg_tat] = n;
|
avg_wt] = n;
printf("In average turnaround time
printf(" average waiting time

```

```
int main()
{
    int n; time quantum;
    printf(" enter the number of
    scanf("%d", &n);
    printf(" enter the Time quantum
    scanf("%d", &time quantum);
    struct process processes[n];
}
```

algorithm)

```
    }
    wt[i] = ct[i] - processes[i].arrival_time - processes[i].burst_time;
    rt[i] = wt[i];
}

for(i=0; i<n; i++)
{
    printf("pid|at|bt|ct|wt|tat|rt\n");
    float avg_tat=0, avg_wt=0;
    for(int j=0; j<n; j++)
        printf("%d|%d|%d|%d|%d|%d|%d\n", processes[j].pid, processes[j].arrival_time, processes[j].burst_time, ct[j], wt[j], tat[j], rt[j]);
    avg_tat += tat[i];
    avg_wt += wt[i];
}

avg_tat /= n;
avg_wt /= n;
printf("average turnaround time %.2f\n", avg_tat);
printf("average waiting time %.2f\n", avg_wt);

int main()
{
    int n, time_quantum;
    printf("enter the number of processes");
    scanf("%d", &n);
    printf("enter the Time quantum");
    scanf("%d", &time_quantum);
    struct process processes[n];
    quantum;
```

```

printf("enter arrival time and burst time for each process\n");
printf(" enter arrival time and burst time for each process\n");
for(int i=0;i<n;i++){
    printf("enter arrival time for process %d ",i+1);
    scanf("%d",&processes[i].arrival-time);
    printf("enter burst time for process %d ",i+1);
    scanf(" %d",&processes[i].burst-time);
    processes[i].pid = i+1;
    processes[i].remaining-time = processes[i].burst-time;
}
roundrobin(processes,n,time quantum);
}

```

#### OUTPUT:

enter the number of processes 5  
 enter the time quantum 2  
~~enter arrival time and burst time for each pr~~  
 enter arrival time and burst time for each process  
 enter arrival time for Process 1 0  
 enter burst time for process 1 5  
 enter arrival time for process 2 1  
 enter burst time for process 2 3  
 enter arrival time for process 3 2  
 enter burst time for process 3 1  
 enter arrival time for process 4 3  
 enter burst time for process 4 2  
~~enter arrival time for process 5 4~~  
~~enter burst time~~  
 enter burst time for process 5 3

pid	at	bt	ct
1	0	5	14
2	1	3	12
3	2	1	5
4	3	2	7
5	4	3	13

~~average turnaroundtime 8.2~~  
~~average turnaroundtime 8.2~~  
~~average turnaroundtime 8.2~~  
~~average waitingtime 5.4~~

average turnaround time 8  
 average waiting time 5.4

~~earliest deadline first sche~~  
~~earliest deadline first scheduling~~

```

#include<stdio.h>
typedef struct {
    int capacity;
    int deadline;
    int period;
}task;
int compare_tasks(const void*
)
task* task1 = (task*)
task* task2 = (task*)
return (task1->dead
}
int main(){
    int num_tasks;
}

```

pid	at	bt	ct	wt	tat	rt
1	0	5	14	9	14	9
2	1	3	12	8	11	8
3	2	1	5	2	3	2
4	3	2	7	2	4	2
5	4	3	13	6	9	6

average turnaround time = 8.20

average turnaround time = 8.20

average waiting time

average waiting time = 5.40

average turnaround time = 8.20

average waiting time = 5.40

earliest deadline first sche

earliest deadline first scheduling

```
#include<stdio.h>
typedef struct{
    int capacity;
    int deadline;
    int period;
}
```

task;

```
int compare_tasks(const void *a, const void *b){
```

task\*

task1 = (task\*)a;

task\* task2 = (task\*)b;

return (task1->deadline - task2->deadline);

```
}int main(){
```

int num\_tasks;

3) ~~lab-3~~ write a C program to simulate the multilevel queue scheduling algorithm.

#include <stdio.h>

void findwaitingtime(int process[], int n, int bt[], int qt[], int wt[])

{

    wt[0] = 0;  
    for (int i=1; i<n; i++)

    {  
        wt[i] = bt[i-1] + wt[i-1] - qt[i-1];

        if (wt[i] < 0)  
            wt[i] = 0;

}

}

}

}

Void findturnaroundtime(int process[], int n, int bt[], int wt[], int tat[])

{

    int tat[];  
    for (int i=0; i<n; i++)

    {  
        tat[i] = bt[i] + wt[i];

}

}

}

~~void roundrobin()~~

void roundrobin(int proce

{

    int wt[n], tat[n], ct[n],

    int remaining\_bt[n];

    int completed = 0;

    int time = 0;

    for (int i=0; i<n; i++)

    {  
        remaining\_bt[i] = bt[i];

    }

    while (completed < n)

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

        {  
            if (remaining\_bt[i] > 0)

            {  
                if (time >= remaining\_bt[i])

                    if (remaining\_bt[i] >= quantum)

                        remaining\_bt[i] -= quantum;

                        ct[i] = time;

                        completed++;

            }

            else

                time += quantum;

                remaining\_bt[i] -= quantum;

                ct[i] = time;

                completed++;

            }

        }

    }

}

    }

    }

    }

```

void roundrobin(int process[], int n, int bt[], int at[], int quantum)
{
    int wt[n], tat[n], ct[n], total_wt = 0, total_tat = 0;
    int remaining_bt[n];
    int remaining_bt[n];
    int completed = 0;
    int time = 0;
    for(int i=0; i<n; i++)
    {
        remaining_bt[i] = bt[i];
    }
    while (completed < n)
    {
        for(int t=0; t<n; t++)
        {
            if (remaining_bt[t] > 0 & at[t] < time)
            {
                if (remaining_bt[t] > quantum)
                {
                    time += remaining_bt[t];
                    remaining_bt[t] = 0;
                    ct[t] = time;
                    completed++;
                }
                else
                {
                    time += quantum;
                    remaining_bt[t] = quantum;
                    ct[t] = time;
                    completed++;
                }
            }
        }
    }
}

```

```

findWaitingTime(process, n, bt, at, wt);
findTurnaroundTime(process, n, bt, wt, tat);

for(int i=0; i<n; i++)
{
    printf(" P %d |t %d|t %d|t %d|t %d\n", process[i],
           bt[i], at[i], wt[i], tat[i]);
}

total_wt += wt[i];
total_tat += tat[i];
total_bt += bt[i];
total_tat += tat[i];
total_wt += wt[i];
total_tat += tat[i];

printf(" average waiting time (round robin) = %.2f\n", (float) total_wt/n);
printf(" average turnaround time (round robin) = %.2f\n", (float) total_tat/n);

```

```

void fcfs(int process[], int n
{
    int wt[n], tat[n], ct[n];
    findWaitingTime();
    findTurnaroundTime();
    printf(" process burst time arrival time waiting time turnaround time
completion completion time \n");

    for(int i=0; i<n; i++)
    {
        ct[i] = at[i] + bt[i];
        printf(" P %d |t %d|t %d|t %d|t %d|t %d\n", process[i],
               bt[i], at[i], wt[i], tat[i], ct[i]);
    }

    total_wt += wt[i];
    total_tat += tat[i];
    printf(" average waiting time \n");
    printf(" average turnaround time \n");
}

int main()
{
    int process[] = {1, 2, 3, 4, 5};
    int n = sizeof(process) / sizeof(int);
    int bt = {10, 5, 8, 12, 15};
    int at[] = {0, 1, 2, 3, 4};
    int quantum = 3;
}

```

```

void FCFS()

void FCFS(int process[], int n, int bt[], int at[])
{
    int wt[n], tat[n], ct[n];
    int wt[n], tat[n], ct[n], total_wt = 0, total_tat = 0;
    findWaitingTime();
    findTurnaroundTime();
    printf(" process bursttime arrivaltime waitingtime turnaroundtime
           completion time \n");
    for (int i=0; i<n; i++)
    {
        ct[i] = at[i] + bt[i];
        printf(" %d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n",
               process[i],
               process[i], bt[i], at[i], wt[i], tat[i], ct[i]);
        total_wt += wt[i];
        total_tat += tat[i];
    }
    printf(" average waiting time (FCFS) = %f \n", (float)total_wt/n);
    printf(" average turnaround time (FCFS) = %f \n", (float)total_tat/n);
}

int main()
{
    int process[] = {1, 2, 3, 4, 5};
    int n = sizeof(process)/sizeof(process[0]);
    int bt[] = {10, 5, 8, 12, 15};
    int at[] = {0, 1, 2, 3, 4};
    int quantum = 2;
}

```

~~int quantum~~

~~int quantum=2;~~

~~int quantum=2;~~

~~roundrobin (process, n, bt, at, quantum);~~

~~fdfs (process, n, bt, at);~~

~~}~~

Output:

process	bt	at	wt	tat	ct
P1	10	0	0	10	39
P2	5	1	10	15	23
P3	8	2	14	22	33
P4	12	3	20	32	45
P5	15	4	29	44	50

average wt

Average wt (Round robin) = 14.60

average turnaround time = 20

average turnaround time (Round robin) = 24.60

process	bt	at	wt	tat	ct
P1	10	0	0	10	10
P2	5	1	10	15	6
P3	8	2	14	22	10
P4	12	3	20	32	15
P5	15	4	29	44	19

average wt (FCFS) = 14.60

average tat (FCFS) = 24.6

(a) y  
rate monotonic

#include<stdio.h>

struct process {

int execution\_time;

int time\_periods;

};

int

int lcm(int a, int b)

{ int max = (a>b)?a:b;

while(1)

{ if(max%a==0&&max%b==0)

return max;

} else max+=lcm(a,b);

return(max);

}; max+=lcm(a,b);

} else max+=lcm(a,b);

}; max+=lcm(a,b);

} else max+=lcm(a,b);

```

int lcm(int a, int b)
{
    int max = (a>b)?a:b;
    while(1)
    {
        if(max%a==0 && max%b==0)
            return max;
        if(max%a==0 || max%b==0)
            return (max+1);
        max++;
    }
}

int is_schedulable(struct process processes[], int n)
{
    float utilization = 0.0;
    for(int i=0; i<n; i++)
    {
        utilization += (float)process[i].execution_time;
        if(process[i].time> period)
            return (Utilization >= 1.0);
    }
}

int main()
{
    struct process processes[2];
    int main()
    {
        struct processes processes[2] =
        {{3,20}, {2,5}};
    }
}

```

```

process
{2,10};
};

int n = sizeof(process) / sizeof(process[0]);
if (!is_schedulable(processes, n))
{
    printf("the given set of processes is not schedulable\n");
    return 0;
}

int scheduling_time = lcm(processes[0].time_period, processes[1].time_period);
printf("execution order\n");
if (processes[0].time_period == 0)
    printf("P0\n");
if (processes[1].time_period == 0)
    printf("P1\n");
if (processes[2].time_period == 0)
    printf("P2\n");
if (processes[3].time_period == 0)
    printf("P3\n");
}

```

```

}

```

~~output~~  
execution order

~~output~~  
execution order  
~~P1 P2 P3~~

~~output~~  
execution order

P2 P3 P1 P2 P2 P3 P2

(b) earliest deadline first scheduling

```

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

```

```

d[i] = temp;
temp = pt[i];
pt[i] = pt[j];
pt[j] = temp;
temp = b[i];
temp = b[j];
b[j] = b[i];
b[i] = temp;
temp = proc[i];
proc[i] = proc[j];
proc[j] = temp;
}
}
}
int gcd();
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++)
    {
        int lcm = p[0];
        for(int j=1; j<n; j++)
        {
            if(p[j] % lcm == 0)
                lcm *= p[j];
        }
        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];
    printf(" enter the cpu burst times");
    for(int i=0; i<n; i++)
    {
        scanf("%d", &b[i]);
        rem[i] = b[i];
    }
    printf(" enter the deadline");
    for(int i=0; i<n; i++)
    {
        scanf("%d", &d[i]);
    }
    printf(" enter the home position");
    for(int i=0; i<n; i++)
    {
        proc[i] = i+1;
    }
    sort(proc, d, b, pt, n);
    int l = lcmul(pt, n);
    printf(" earliest deadline first scheduling");
    for(int i=0; i<n; i++)
    {
        printf("%d\t", i+1);
        printf("%d\t", pt[i]);
    }
}

```

```

lcm = (lcm * p[i]) / gcd(lcm, p[i]);
}

return lcm;
}

void main()
{
    int n;
    printf(" enter the number of process ");
    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++)
    {
        proc[i] = i+1;
    }
    sort(proc, d, b, pt, n);
    int l = lcmul(pt, n);
    printf(" earliest deadline scheduling \n ");
    printf(" pid\tburst\tdeadline\tperiod \n ");
    for(int i=0; i<n; i++)
    {
        printf("%d\t%d\t%d\t%d\n", proc[i], b[i], d[i], pt[i]);
    }
}

```

```

printf("Scheduling occurs for %d ms\n", n);
int time = 0;
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 < 1)
{
    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 = dt[1];
    int task_to;
    int tasktoexecute = -1;
    for(int i=0; i<n; i++)
    {
        if(rem[i] > 0 && nextdeadlines[i] < mindeadline)
        {
            mindeadline = nextdeadlines[i];
            tasktoexecute = i;
            tasktoexecute = i;
        }
    }
    if(tasktoexecute != -1)
    {
        printf("%dms : task %d is running\n", time, proc[tasktoexecute]);
    }
}

```

rem[tasktoexecute]

```

}
else
{
    printf("%dms : CPU is
    printf("%dms : CPU is
}
}
time++;
}
}

```

Output  
enter the number of process 3  
enter cpu burst time

3  
2  
2  
enter the deadlines:  
7  
4  
8  
~~enter~~

enter the timeperiod  
20  
5  
10

earliest deadline scheduling

Pid	burst	deadlines	pe
2	2	4	
1	3	4	
3	2	8	

Scheduling occurs for 20ms

0ms: task 2 is running

1ms: task 2 is running

2ms: task 1 is running

3ms: task 1 is running

4ms: task 1 is running

5ms: task 3 is running

6ms: task 3 is running

7ms: task 2 is running

8ms: task 2 is running

9ms: CPU is idle

```

rem[tasktoexecute]--;

if(eve)
{
    printf("%dms - CPU is idle\n");
    printf("%dms - CPU is idle \n", time);
}

// Time++
time++;
}

Output
enter the number of process 3
enter CPU burst time
3
2
enter the deadlines
7
4
8
enter the time period
20
5
10
earliest deadline scheduling
pid   burst  deadlines  period
2     2       4           5
1     3       7           20
3     2       8           10

Scheduling occurs for 20ms
0ms: task 2 is running
1ms: task 2 is running
2ms: task 1 is running
3ms: task 1 is running
4ms: task 1 is running
5ms: task 3 is running
6ms: task 3 is running
7ms: task 2 is running
8ms: task 2 is running
9ms: CPU is idle

```

```

10 ms task 2 is running
11 ms task 2 is running
12 ms task 3 is running
13 ms task 3 is running
14 ms CPU is idle
15 ms task 2 is running
16 ms task 2 is running
17 ms CPU is idle
18 ms CPU is idle
19 ms CPU is idle

```

(c) 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;
    int total_ticks = 0;
    for (int i = 0; i < num_tasks; i++) {
        total_tickets += tasks[i].tickets;
    }
    srand(time(NULL));
    int current_time = 0;
    int completed = 0;
    int c;
    int completed_tasks = 0;
    printf("process scheduling\n");
    while (completed_tasks < num_tasks) {
        int winning_ticket = rand() % total_tickets;
    }
}

```

```

int cumulative_ticks = 0;
for (int i = 0; i < num_tasks; i++) {
    cumulative_ticks += tasks[i].tickets;
}
if (winning_ticket <= cumulative_ticks) {
    if (winning_ticket == cumulative_ticks) {
        if (current_time == 0) {
            current_time = 1;
        } else {
            current_time += time_unit_duration_ms;
        }
        break;
    }
    completed_tasks++;
}
current_time += time_unit_duration_ms;
*time_span_ms = current_time;
}

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\n");
        return 1;
    }
}

```

```

int cumulative_ticks = 0;
for(int i=0; i<num_tasks; i++) {
    cumulative_ticks += tasks[i].ticks;
    if(winning_ticket < cumulative_ticks) {
        winning_ticket = cumulative_ticks;
    }
}
if(winning_ticket < cumulative_ticks) {
    printf("time %d - %d task %d is running\n",
          current_time, (current_time - task[i].tid));
    current_time = current_time + task[i].duration;
    break;
}
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 number between
              1 and %d\n", max_tasks);
        return(1);
    }
}

```

```

printf("enter\n");
printf("enter number of tickets for each task \n");
for(int i=0; i<num-tasks; i++)
    for(int i=0; i<num-tasks; i++)
        tasks[i].tid = i+1;
    printf("task %d ticket", tasks[i].tid);
    scanf("%d", &tasks[i].tickets);
}
printf("\n running tasks \n");
schedule(tasks, num-tasks, 8*time-span-ms);
printf("\n time span of the gantt chart: %d milliseconds\n");
printf("\n time span of the gantt chart %d milliseconds\n");
}

```

### Output

enter the number of tasks 3

enter the number of tickets for each task

task 1 tickets 10

task 2 tickets 20

task 3 tickets 30

running tasks

~~process scheduling~~

~~process scheduling~~

~~process scheduling~~

~~process scheduling~~

time 0-1 task 3 is running

~~task~~

time 2-2 task 3 is running

time 2-3 task 1 is running

~~task~~

time span of gantt chart

time span of gantt chart

12-06-2024

write a c program to  
Semaphores

```

#include <stdio.h>
#include <stdlib.h>
int mutex = 1, full = 0, empty = 0;
int main()
{
    int n;
    void producer();
    void consumer();
    int wait();
    int signal();
    printf("\n producer\n");
    while(1)
    {
        printf("In enter your choice\n");
        printf("In enter your choice\n");
        scanf("%d", &n);
        switch(n)
        {
            case 1 : if(mutex == 1)
            case 1 : if((mutex == 1) && (full == 0))
            {
                mutex = 0;
                full = 1;
                printf("producer inserted item\n");
            }
            else {
                printf("producer is waiting\n");
            }
        }
    }
}
```

```

case 2 : if((mutex == 0) && (full == 1))
case 2 : if((mutex == 0) && (full == 1))
{
    mutex = 1;
    full = 0;
    printf("consumer removed item\n");
}
else {
    printf("consumer is waiting\n");
}
break;
}
case 2 : if((mutex == 0) && (full == 1))
{
    mutex = 1;
    full = 0;
    printf("consumer removed item\n");
}
else {
    printf("consumer is waiting\n");
}
break;
}
}
```

time span of gantt chart.

time span of gantt chart 300 milliseconds

12-06-2024

to simulate producer consumer problem using semaphores

```
#include <stdio.h>
#include <stdlib.h>
int mutex = 1, full = 0, empty = 5, x = 0;
int main()
{
    int n;
    void producer();
    void consumer();
    int wait(int);
    int signal(int);
    printf("\n 1.producer\n 2.consumer\n 3.exit");
    while(1)
    {
        printf("\n enter your choice");
        printf("\n enter your choice");
        scanf("%d", &n);
        switch(n)
        {
            case 1 : if((mutex == 1) && (empty == 0))
            case 1 : if((mutex == 1) && (empty == 0))
            {
                producer();
            }
            else
            {
                printf("buffer full");
            }
            break;
            case 2 : if((mutex == 1) && (full != 0))
            {
                consumer();
            }
        }
    }
}
```

```

    }
    else {
        printf("buffer is empty");
        break;
    }
}

case 3 : exit(0);
break;

}

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 item %d", x);
    mutex = signal(mutex);
}

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

```

outputs

1. producer
2. consumer
3. exit

enter your choice 1

~~producer~~  
producer produces item 1  
enter your choice 1

~~producer~~  
producer produces item 2  
enter your choice 2  
enter your choice 2  
consumer consumes item 2  
~~enter your choice 2~~  
enter your choice 2  
consumer consumes item 1  
~~enter your choice 2~~

~~enter your choice 2~~  
buffer is empty  
~~enter your choice 2~~  
buffer is empty  
~~enter your choice 3~~  
enter your choice 3

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

### Output

- 1. producer
- 2. consumer

3. exit

enter your choice 1

~~producer~~  
producer produces item 1

enter your choice 1

~~producer~~

producer produces item 2

enter your choice 2

enter your choice 2

consumer consumes item 2

~~enter your choice 2~~

enter your choice 2

consumer consumes item 1

~~enter your choice~~

~~choice~~

enter your choice 2

buffer is empty

enter yo~~r~~

buffer is empty

~~enter your choice 3~~

enter your choice 3

8/16

→ 19/06/2024  
6) 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[10];
int hungry_philosophers[max_philosophers];
int forks[max_philosophers];
void print_state(){
    printf("\n");
    for(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 philosopher_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;
}
void simulate(int allow_ru
int eating_count = 0;
for(int i = 0; i < num_hu
int philosopher_id =
if(states[philosopher_id]
if(can_eat(philosopher_id))
{
    states[philosopher_id] = eating;
    eating_count++;
    printf("P%d is eating\n", philosopher_id);
    if(!allow_ru_time)
        break;
}
if(allow_ru_time)
    break;
}
}
for(int i = 0; i < num_hu
int philosopher_id =
if(states[philosopher_id]
int left_fork =
int right_fork =
int left_fork =
int right_fork =
```

### 3 Philosophers

```
return 0;

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 rig
                int left_fork = philosopher_id;
                int right_fork = (philosopher_id + 1) % num_philosophers;
            }
        }
    }
}
```

```

forks[left_fork], forks[right_fork] = 0;
states[philosopher_id] = thinking;

int main(){
    printf(" enter the total number of philosophers (max %d)", max_philosophers);
    scanf("%d", &num_philosophers);
    if(num_philosophers < 2 || num_philosophers > max_philosophers){
        printf("invalid number");
        printf("invalid number of philosophers exiting\n");
        return 1;
    }
    printf(" how many are hungry\n");
    printf(" how many are hungry");
    scanf("%d", &num_hungry);
    for(int i=0; i<num_hungry; i++){
        printf(" enter philosopher %d position",
        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("\n 1. one can eat at a time\n 2 two can eat at a time\n 3 exit\n");
    printf("enter your choice");
    printf(" enter your choice");
    scanf("%d", &choice);
    switch(choice){
        case 1 : simulate();
        break;
        case 2 : simulate();
        break;
        case 3 : printf("exiting\n");
        break;
        default: printf("invalid choice\n");
        break;
    }
} while(choice != 3);

```

Output

```

enter total number of philosophers
how many are hungry : 2
enter philosopher 1 position 1
enter philosopher 2 position 1
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

```

```

int choice;
do{
    print-state();
    printf("\n 1 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(" enter your choice");
    scanf("%d", &choice);
    switch(choice){
        case 1 : simulate(0);
        break;
        case 2 : simulate(1);
        break;
        case 3 : printf("exiting\n");
        break;
        default: printf("invalid choice please try again\n");
        break;
    }
}while(choice!=3);

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

```

P<sub>4</sub> is granted to eat

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

enter your choice : 2

P<sub>1</sub> and P<sub>4</sub> are granted

P<sub>1</sub> and P<sub>4</sub> are granted to eat

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

enter your choice : 3

7) write a c program to simulate bankers algorithm for the purpose of deadlock avoidance

#include<st

#include<stdio.h>

int P, Y;

void avai(int all[P][Y], int tot[Y], int avail[Y]) {

for (int j = 0; j < Y; j++) {

avail[j] = 0;

}

for (int k = 0; k < P; k++) {

avail[j] =

avail[j] + all[k][j];

}

for (int j = 0; j < Y; j++) {

}

for (int j = 0; j < Y; j++) {

avail[j] = tot[j] - avail[j];

}

y  
void needt(int all[P][Y], int max)

void needl(int all[P][Y], int max)

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

for (int j = 0; j < Y; j++) {

needmat[i][j] = max

}

}

y  
void safety(int P, int Y, int all[P]

int f[P], C, count = 0, h = 0;

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 < Y; j++) {

if (needmat[i][j] <= C) {

C = C + 1;

}

if (C == Y) {

printf("P%d is", i + 1);

for (int k = 0; k < P; k++) {

avail[k] +=

avail[k];

```

void need(int all[P][R], int max[R][C], -)
void need(int all[P][R], int max[R][C], int needmat[R][C]){
    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[R][C], int seq[R]){
    int f[P], c, count=0, h=0;
    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(j=0; j<R; j++){
                    for(int k=0; k<C; k++){
                        if(needmat[i][j]<=avail[j]){
                            C=C+1;
                        }
                    }
                }
                if(C==R){
                    printf("%d is visited\n", i);
                    for(int k=0; k<R; k++){
                        avail[k] += needmat[i][k];
                    }
                }
                count++;
            }
        }
    }
}

```

```

1     printf("%d", p[i]);
2     printf("%d", tot[i]);
3     printf("\n");
4
5     printf("%d", avail[k]);
6     printf("\n");
7 }
8
9 printf("J\n");
10 F[i]=1;
11 Count=Count+1;
12 seq[h]=i;
13 h=h+1;
14
15 }
16
17 }
18
19 int main()
20 {
21     printf("enter the number of processes");
22     scanf("%d",&P);
23
24     printf("enter the number of resources");
25     scanf("%d",&R);
26
27     int tot[R], needmat[P][R], avail[R], seq[P];
28
29     printf("enter the total instances of each resource");
30
31     for(int i=0; i<P; i++)
32     {
33         for(int j=0; j<R; j++)
34         {
35             scanf("%d", &tot[j]);
36             scanf("%d", &needmat[i][j]);
37             scanf("%d", &avail[j]);
38         }
39     }
40
41     int all[P][R], max[R][R];
42
43     for(int i=0; i<P; i++)
44     {
45         for(int j=0; j<R; j++)
46         {
47             all[i][j] = max[j][j];
48         }
49     }
50
51     for(int i=0; i<P; i++)
52     {
53         for(int j=0; j<R; j++)
54         {
55             if(needmat[i][j] > all[i][j])
56             {
57                 all[i][j] = needmat[i][j];
58             }
59         }
60     }
61
62     for(int i=0; i<P; i++)
63     {
64         for(int j=0; j<R; j++)
65         {
66             if(all[i][j] < tot[j])
67             {
68                 all[i][j] = tot[j];
69             }
70         }
71     }
72
73     for(int i=0; i<P; i++)
74     {
75         for(int j=0; j<R; j++)
76         {
77             if(all[i][j] > tot[j])
78             {
79                 all[i][j] = tot[j];
80             }
81         }
82     }
83
84     for(int i=0; i<P; i++)
85     {
86         for(int j=0; j<R; j++)
87         {
88             if(all[i][j] > tot[j])
89             {
90                 all[i][j] = tot[j];
91             }
92         }
93     }
94
95     for(int i=0; i<P; i++)
96     {
97         for(int j=0; j<R; j++)
98         {
99             if(all[i][j] > tot[j])
100            {
101                all[i][j] = tot[j];
102            }
103        }
104    }
105
106    for(int i=0; i<P; i++)
107    {
108        for(int j=0; j<R; j++)
109        {
110            if(all[i][j] > tot[j])
111            {
112                all[i][j] = tot[j];
113            }
114        }
115    }
116
117    for(int i=0; i<P; i++)
118    {
119        for(int j=0; j<R; j++)
120        {
121            if(all[i][j] > tot[j])
122            {
123                all[i][j] = tot[j];
124            }
125        }
126    }
127
128    for(int i=0; i<P; i++)
129    {
130        for(int j=0; j<R; j++)
131        {
132            if(all[i][j] > tot[j])
133            {
134                all[i][j] = tot[j];
135            }
136        }
137    }
138
139    for(int i=0; i<P; i++)
140    {
141        for(int j=0; j<R; j++)
142        {
143            if(all[i][j] > tot[j])
144            {
145                all[i][j] = tot[j];
146            }
147        }
148    }
149
150    for(int i=0; i<P; i++)
151    {
152        for(int j=0; j<R; j++)
153        {
154            if(all[i][j] > tot[j])
155            {
156                all[i][j] = tot[j];
157            }
158        }
159    }
160
161    for(int i=0; i<P; i++)
162    {
163        for(int j=0; j<R; j++)
164        {
165            if(all[i][j] > tot[j])
166            {
167                all[i][j] = tot[j];
168            }
169        }
170    }
171
172    for(int i=0; i<P; i++)
173    {
174        for(int j=0; j<R; j++)
175        {
176            if(all[i][j] > tot[j])
177            {
178                all[i][j] = tot[j];
179            }
180        }
181    }
182
183    for(int i=0; i<P; i++)
184    {
185        for(int j=0; j<R; j++)
186        {
187            if(all[i][j] > tot[j])
188            {
189                all[i][j] = tot[j];
190            }
191        }
192    }
193
194    for(int i=0; i<P; i++)
195    {
196        for(int j=0; j<R; j++)
197        {
198            if(all[i][j] > tot[j])
199            {
200                all[i][j] = tot[j];
201            }
202        }
203    }
204
205    for(int i=0; i<P; i++)
206    {
207        for(int j=0; j<R; j++)
208        {
209            if(all[i][j] > tot[j])
210            {
211                all[i][j] = tot[j];
212            }
213        }
214    }
215
216    for(int i=0; i<P; i++)
217    {
218        for(int j=0; j<R; j++)
219        {
220            if(all[i][j] > tot[j])
221            {
222                all[i][j] = tot[j];
223            }
224        }
225    }
226
227    for(int i=0; i<P; i++)
228    {
229        for(int j=0; j<R; j++)
230        {
231            if(all[i][j] > tot[j])
232            {
233                all[i][j] = tot[j];
234            }
235        }
236    }
237
238    for(int i=0; i<P; i++)
239    {
240        for(int j=0; j<R; j++)
241        {
242            if(all[i][j] > tot[j])
243            {
244                all[i][j] = tot[j];
245            }
246        }
247    }
248
249    for(int i=0; i<P; i++)
250    {
251        for(int j=0; j<R; j++)
252        {
253            if(all[i][j] > tot[j])
254            {
255                all[i][j] = tot[j];
256            }
257        }
258    }
259
260    for(int i=0; i<P; i++)
261    {
262        for(int j=0; j<R; j++)
263        {
264            if(all[i][j] > tot[j])
265            {
266                all[i][j] = tot[j];
267            }
268        }
269    }
270
271    for(int i=0; i<P; i++)
272    {
273        for(int j=0; j<R; j++)
274        {
275            if(all[i][j] > tot[j])
276            {
277                all[i][j] = tot[j];
278            }
279        }
280    }
281
282    for(int i=0; i<P; i++)
283    {
284        for(int j=0; j<R; j++)
285        {
286            if(all[i][j] > tot[j])
287            {
288                all[i][j] = tot[j];
289            }
290        }
291    }
292
293    for(int i=0; i<P; i++)
294    {
295        for(int j=0; j<R; j++)
296        {
297            if(all[i][j] > tot[j])
298            {
299                all[i][j] = tot[j];
300            }
301        }
302    }
303
304    for(int i=0; i<P; i++)
305    {
306        for(int j=0; j<R; j++)
307        {
308            if(all[i][j] > tot[j])
309            {
310                all[i][j] = tot[j];
311            }
312        }
313    }
314
315    for(int i=0; i<P; i++)
316    {
317        for(int j=0; j<R; j++)
318        {
319            if(all[i][j] > tot[j])
320            {
321                all[i][j] = tot[j];
322            }
323        }
324    }
325
326    for(int i=0; i<P; i++)
327    {
328        for(int j=0; j<R; j++)
329        {
330            if(all[i][j] > tot[j])
331            {
332                all[i][j] = tot[j];
333            }
334        }
335    }
336
337    for(int i=0; i<P; i++)
338    {
339        for(int j=0; j<R; j++)
340        {
341            if(all[i][j] > tot[j])
342            {
343                all[i][j] = tot[j];
344            }
345        }
346    }
347
348    for(int i=0; i<P; i++)
349    {
350        for(int j=0; j<R; j++)
351        {
352            if(all[i][j] > tot[j])
353            {
354                all[i][j] = tot[j];
355            }
356        }
357    }
358
359    for(int i=0; i<P; i++)
360    {
361        for(int j=0; j<R; j++)
362        {
363            if(all[i][j] > tot[j])
364            {
365                all[i][j] = tot[j];
366            }
367        }
368    }
369
370    for(int i=0; i<P; i++)
371    {
372        for(int j=0; j<R; j++)
373        {
374            if(all[i][j] > tot[j])
375            {
376                all[i][j] = tot[j];
377            }
378        }
379    }
380
381    for(int i=0; i<P; i++)
382    {
383        for(int j=0; j<R; j++)
384        {
385            if(all[i][j] > tot[j])
386            {
387                all[i][j] = tot[j];
388            }
389        }
390    }
391
392    for(int i=0; i<P; i++)
393    {
394        for(int j=0; j<R; j++)
395        {
396            if(all[i][j] > tot[j])
397            {
398                all[i][j] = tot[j];
399            }
400        }
401    }
402
403    for(int i=0; i<P; i++)
404    {
405        for(int j=0; j<R; j++)
406        {
407            if(all[i][j] > tot[j])
408            {
409                all[i][j] = tot[j];
410            }
411        }
412    }
413
414    for(int i=0; i<P; i++)
415    {
416        for(int j=0; j<R; j++)
417        {
418            if(all[i][j] > tot[j])
419            {
420                all[i][j] = tot[j];
421            }
422        }
423    }
424
425    for(int i=0; i<P; i++)
426    {
427        for(int j=0; j<R; j++)
428        {
429            if(all[i][j] > tot[j])
430            {
431                all[i][j] = tot[j];
432            }
433        }
434    }
435
436    for(int i=0; i<P; i++)
437    {
438        for(int j=0; j<R; j++)
439        {
440            if(all[i][j] > tot[j])
441            {
442                all[i][j] = tot[j];
443            }
444        }
445    }
446
447    for(int i=0; i<P; i++)
448    {
449        for(int j=0; j<R; j++)
450        {
451            if(all[i][j] > tot[j])
452            {
453                all[i][j] = tot[j];
454            }
455        }
456    }
457
458    for(int i=0; i<P; i++)
459    {
460        for(int j=0; j<R; j++)
461        {
462            if(all[i][j] > tot[j])
463            {
464                all[i][j] = tot[j];
465            }
466        }
467    }
468
469    for(int i=0; i<P; i++)
470    {
471        for(int j=0; j<R; j++)
472        {
473            if(all[i][j] > tot[j])
474            {
475                all[i][j] = tot[j];
476            }
477        }
478    }
479
480    for(int i=0; i<P; i++)
481    {
482        for(int j=0; j<R; j++)
483        {
484            if(all[i][j] > tot[j])
485            {
486                all[i][j] = tot[j];
487            }
488        }
489    }
490
491    for(int i=0; i<P; i++)
492    {
493        for(int j=0; j<R; j++)
494        {
495            if(all[i][j] > tot[j])
496            {
497                all[i][j] = tot[j];
498            }
499        }
500    }
501
502    for(int i=0; i<P; i++)
503    {
504        for(int j=0; j<R; j++)
505        {
506            if(all[i][j] > tot[j])
507            {
508                all[i][j] = tot[j];
509            }
510        }
511    }
512
513    for(int i=0; i<P; i++)
514    {
515        for(int j=0; j<R; j++)
516        {
517            if(all[i][j] > tot[j])
518            {
519                all[i][j] = tot[j];
520            }
521        }
522    }
523
524    for(int i=0; i<P; i++)
525    {
526        for(int j=0; j<R; j++)
527        {
528            if(all[i][j] > tot[j])
529            {
530                all[i][j] = tot[j];
531            }
532        }
533    }
534
535    for(int i=0; i<P; i++)
536    {
537        for(int j=0; j<R; j++)
538        {
539            if(all[i][j] > tot[j])
540            {
541                all[i][j] = tot[j];
542            }
543        }
544    }
545
546    for(int i=0; i<P; i++)
547    {
548        for(int j=0; j<R; j++)
549        {
550            if(all[i][j] > tot[j])
551            {
552                all[i][j] = tot[j];
553            }
554        }
555    }
556
557    for(int i=0; i<P; i++)
558    {
559        for(int j=0; j<R; j++)
560        {
561            if(all[i][j] > tot[j])
562            {
563                all[i][j] = tot[j];
564            }
565        }
566    }
567
568    for(int i=0; i<P; i++)
569    {
570        for(int j=0; j<R; j++)
571        {
572            if(all[i][j] > tot[j])
573            {
574                all[i][j] = tot[j];
575            }
576        }
577    }
578
579    for(int i=0; i<P; i++)
580    {
581        for(int j=0; j<R; j++)
582        {
583            if(all[i][j] > tot[j])
584            {
585                all[i][j] = tot[j];
586            }
587        }
588    }
589
590    for(int i=0; i<P; i++)
591    {
592        for(int j=0; j<R; j++)
593        {
594            if(all[i][j] > tot[j])
595            {
596                all[i][j] = tot[j];
597            }
598        }
599    }
599
600    for(int i=0; i<P; i++)
601    {
602        for(int j=0; j<R; j++)
603        {
604            if(all[i][j] > tot[j])
605            {
606                all[i][j] = tot[j];
607            }
608        }
609    }
610
611    for(int i=0; i<P; i++)
612    {
613        for(int j=0; j<R; j++)
614        {
615            if(all[i][j] > tot[j])
616            {
617                all[i][j] = tot[j];
618            }
619        }
620    }
621
622    for(int i=0; i<P; i++)
623    {
624        for(int j=0; j<R; j++)
625        {
626            if(all[i][j] > tot[j])
627            {
628                all[i][j] = tot[j];
629            }
630        }
631    }
632
633    for(int i=0; i<P; i++)
634    {
635        for(int j=0; j<R; j++)
636        {
637            if(all[i][j] > tot[j])
638            {
639                all[i][j] = tot[j];
640            }
641        }
642    }
643
644    for(int i=0; i<P; i++)
645    {
646        for(int j=0; j<R; j++)
647        {
648            if(all[i][j] > tot[j])
649            {
650                all[i][j] = tot[j];
651            }
652        }
653    }
654
655    for(int i=0; i<P; i++)
656    {
657        for(int j=0; j<R; j++)
658        {
659            if(all[i][j] > tot[j])
660            {
661                all[i][j] = tot[j];
662            }
663        }
664    }
665
666    for(int i=0; i<P; i++)
667    {
668        for(int j=0; j<R; j++)
669        {
670            if(all[i][j] > tot[j])
671            {
672                all[i][j] = tot[j];
673            }
674        }
675    }
676
677    for(int i=0; i<P; i++)
678    {
679        for(int j=0; j<R; j++)
680        {
681            if(all[i][j] > tot[j])
682            {
683                all[i][j] = tot[j];
684            }
685        }
686    }
687
688    for(int i=0; i<P; i++)
689    {
690        for(int j=0; j<R; j++)
691        {
692            if(all[i][j] > tot[j])
693            {
694                all[i][j] = tot[j];
695            }
696        }
697    }
698
699    for(int i=0; i<P; i++)
700    {
701        for(int j=0; j<R; j++)
702        {
703            if(all[i][j] > tot[j])
704            {
705                all[i][j] = tot[j];
706            }
707        }
708    }
709
710    for(int i=0; i<P; i++)
711    {
712        for(int j=0; j<R; j++)
713        {
714            if(all[i][j] > tot[j])
715            {
716                all[i][j] = tot[j];
717            }
718        }
719    }
720
721    for(int i=0; i<P; i++)
722    {
723        for(int j=0; j<R; j++)
724        {
725            if(all[i][j] > tot[j])
726            {
727                all[i][j] = tot[j];
728            }
729        }
730    }
731
732    for(int i=0; i<P; i++)
733    {
734        for(int j=0; j<R; j++)
735        {
736            if(all[i][j] > tot[j])
737            {
738                all[i][j] = tot[j];
739            }
740        }
741    }
742
743    for(int i=0; i<P; i++)
744    {
745        for(int j=0; j<R; j++)
746        {
747            if(all[i][j] > tot[j])
748            {
749                all[i][j] = tot[j];
750            }
751        }
752    }
753
754    for(int i=0; i<P; i++)
755    {
756        for(int j=0; j<R; j++)
757        {
758            if(all[i][j] > tot[j])
759            {
760                all[i][j] = tot[j];
761            }
762        }
763    }
764
765    for(int i=0; i<P; i++)
766    {
767        for(int j=0; j<R; j++)
768        {
769            if(all[i][j] > tot[j])
770            {
771                all[i][j] = tot[j];
772            }
773        }
774    }
775
776    for(int i=0; i<P; i++)
777    {
778        for(int j=0; j<R; j++)
779        {
780            if(all[i][j] > tot[j])
781            {
782                all[i][j] = tot[j];
783            }
784        }
785    }
786
787    for(int i=0; i<P; i++)
788    {
789        for(int j=0; j<R; j++)
790        {
791            if(all[i][j] > tot[j])
792            {
793                all[i][j] = tot[j];
794            }
795        }
796    }
797
798    for(int i=0; i<P; i++)
799    {
800        for(int j=0; j<R; j++)
801        {
802            if(all[i][j] > tot[j])
803            {
804                all[i][j] = tot[j];
805            }
806        }
807    }
808
809    for(int i=0; i<P; i++)
810    {
811        for(int j=0; j<R; j++)
812        {
813            if(all[i][j] > tot[j])
814            {
815                all[i][j] = tot[j];
816            }
817        }
818    }
819
820    for(int i=0; i<P; i++)
821    {
822        for(int j=0; j<R; j++)
823        {
824            if(all[i][j] > tot[j])
825            {
826                all[i][j] = tot[j];
827            }
828        }
829    }
830
831    for(int i=0; i<P; i++)
832    {
833        for(int j=0; j<R; j++)
834        {
835            if(all[i][j] > tot[j])
836            {
837                all[i][j] = tot[j];
838            }
839        }
840    }
841
842    for(int i=0; i<P; i++)
843    {
844        for(int j=0; j<R; j++)
845        {
846            if(all[i][j] > tot[j])
847            {
848                all[i][j] = tot[j];
849            }
850        }
851    }
852
853    for(int i=0; i<P; i++)
854    {
855        for(int j=0; j<R; j++)
856        {
857            if(all[i][j] > tot[j])
858            {
859                all[i][j] = tot[j];
860            }
861        }
862    }
863
864    for(int i=0; i<P; i++)
865    {
866        for(int j=0; j<R; j++)
867        {
868            if(all[i][j] > tot[j])
869            {
870                all[i][j] = tot[j];
871            }
872        }
873    }
874
875    for(int i=0; i<P; i++)
876    {
877        for(int j=0; j<R; j++)
878        {
879            if(all[i][j] > tot[j])
880            {
881                all[i][j] = tot[j];
882            }
883        }
884    }
885
886    for(int i=0; i<P; i++)
887    {
888        for(int j=0; j<R; j++)
889        {
890            if(all[i][j] > tot[j])
891            {
892                all[i][j] = tot[j];
893            }
894        }
895    }
896
897    for(int i=0; i<P; i++)
898    {
899        for(int j=0; j<R; j++)
900        {
901            if(all[i][j] > tot[j])
902            {
903                all[i][j] = tot[j];
904            }
905        }
906    }
907
908    for(int i=0; i<P; i++)
909    {
910        for(int j=0; j<R; j++)
911        {
912            if(all[i][j] > tot[j])
913            {
914                all[i][j] = tot[j];
915            }
916        }
917    }
918
919    for(int i=0; i<P; i++)
920    {
921        for(int j=0; j<R; j++)
922        {
923            if(all[i][j] > tot[j])
924            {
925                all[i][j] = tot[j];
926            }
927        }
928    }
929
930    for(int i=0; i<P; i++)
931    {
932        for(int j=0; j<R; j++)
933        {
934            if(all[i][j] > tot[j])
935            {
936                all[i][j] = tot[j];
937            }
938        }
939    }
940
941    for(int i=0; i<P; i++)
942    {
943        for(int j=0; j<R; j++)
944        {
945            if(all[i][j] > tot[j])
946            {
947                all[i][j] = tot[j];
948            }
949        }
950    }
951
952    for(int i=0; i<P; i++)
953    {
954        for(int j=0; j<R; j++)
955        {
956            if(all[i][j] > tot[j])
957            {
958                all[i][j] = tot[j];
959            }
960        }
961    }
962
963    for(int i=0; i<P; i++)
964    {
965        for(int j=0; j<R; j++)
966        {
967            if(all[i][j] > tot[j])
968            {
969                all[i][j] = tot[j];
970            }
971        }
972    }
973
974    for(int i=0; i<P; i++)
975    {
976        for(int j=0; j<R; j++)
977        {
978            if(all[i][j] > tot[j])
979            {
980                all[i][j] = tot[j];
981            }
982        }
983    }
984
985    for(int i=0; i<P; i++)
986    {
987        for(int j=0; j<R; j++)
988        {
989            if(all[i][j] > tot[j])
990            {
991                all[i][j] = tot[j];
992            }
993        }
994    }
995
996    for(int i=0; i<P; i++)
997    {
998        for(int j=0; j<R; j++)
999        {
1000            if(all[i][j] > tot[j])
1001            {
1002                all[i][j] = tot[j];
1003            }
1004        }
1005    }
1006
1007    for(int i=0; i<P; i++)
1008    {
1009        for(int j=0; j<R; j++)
1010        {
1011            if(all[i][j] > tot[j])
1012            {
1013                all[i][j] = tot[j];
1014            }
1015        }
1016    }
1017
1018    for(int i=0; i<P; i++)
1019    {
1020        for(int j=0; j<R; j++)
1021        {
1022            if(all[i][j] > tot[j])
1023            {
1024                all[i][j] = tot[j];
1025            }
1026        }
1027    }
1028
1029    for(int i=0; i<P; i++)
1030    {
1031        for(int j=0; j<R; j++)
1032        {
1033            if(all[i][j] > tot[j])
1034            {
1035                all[i][j] = tot[j];
1036            }
1037        }
1038    }
1039
1040    for(int i=0; i<P; i++)
1041    {
1042        for(int j=0; j<R; j++)
1043        {
1044            if(all[i][j] > tot[j])
1045            {
1046                all[i][j] = tot[j];
1047            }
1048        }
1049    }
1050
1051    for(int i=0; i<P; i++)
1052    {
1053        for(int j=0; j<R; j++)
1054        {
1055            if(all[i][j] > tot[j])
1056            {
1057                all[i][j] = tot[j];
1058            }
1059        }
1060    }
1061
1062    for(int i=0; i<P; i++)
1063    {
1064        for(int j=0; j<R; j++)
1065        {
1066            if(all[i][j] > tot[j])
1067            {
1068                all[i][j] = tot[j];
1069            }
1070        }
1071    }
1072
1073    for(int i=0; i<P; i++)
1074    {
1075        for(int j=0; j<R; j++)
1076        {
1077            if(all[i][j] > tot[j])
1078            {
1079                all[i][j] = tot[j];
1080            }
1081        }
1082    }
1083
1084    for(int i=0; i<P; i++)
1085    {
1086        for(int j=0; j<R; j++)
1087        {
1088            if(all[i][j] > tot[j])
1089            {
1090                all[i][j] = tot[j];
1091            }
1092        }
1093    }
1094
1095    for(int i=0; i<P; i++)
1096    {
1097        for(int j=0; j<R; j++)
1098        {
1099            if(all[i][j] > tot[j])
1100            {
1101                all[i][j] = tot[j];
1102            }
1103        }
1104    }
1105
1106    for(int i=0; i<P; i++)
1107    {
1108        for(int j=0; j<R; j++)
1109        {
1110            if(all[i][j] > tot[j])
1111            {
1112                all[i][j] = tot[j];
1113            }
1114        }
1115    }
1116
1117    for(int i=0; i<P; i++)
1118    {
1119        for(int j=0; j<R; j++)
1120        {
1121            if(all[i][j] > tot[j])
1122            {
1123                all[i][j] = tot[j];
1124            }
1125        }
1126    }
1127
1128    for(int i=0; i<P; i++)
1129    {
1130        for(int j=0; j<R; j++)
1131        {
1132            if(all[i][j] > tot[j])
1133            {
1134                all[i][j] = tot[j];
1135            }
1136        }
1137    }
1138
1139    for(int i=0; i<P; i++)
1140    {
1141        for(int j=0; j<R; j++)
1142        {
1143            if(all[i][j] > tot[j])
1144            {
1145                all[i][j] = tot[j];
1146            }
1147        }
1148    }
1149
1150    for(int i=0; i<P; i++)
1151    {
1152        for(int j=0; j<R; j++)
1153        {
1154            if(all[i][j] > tot[j])
1155            {
1156                all[i][j] = tot[j];
1157            }
1158        }
1159    }
1160
1161    for(int i=0; i<P; i++)
1162    {
1163        for(int j=0; j<R; j++)
1164        {
1165            if(all[i][j] > tot[j])
1166            {
1167                all[i][j] = tot[j];
1168            }
1169        }
1170    }
1171
1172    for(int i=0; i<P; i++)
1173    {
1174        for(int j=0; j<R; j++)
1175        {
1176            if(all[i][j] > tot[j])
1177            {
1178                all[i][j] = tot[j];
1179            }
1180        }
1181    }
1182
1183    for(int i=0; i<P; i++)
1184    {
1185        for(int j=0; j<R; j++)
1186        {
1187            if(all[i][j] > tot[j])
1188            {
1189                all[i][j] = tot[j];
1190            }
1191        }
1192    }
1193
1194    for(int i=0; i<P; i++)
1195    {
1196        for(int j=0; j<R; j++)
1197        {
1198            if(all[i][j] > tot[j])
1199            {
1200                all[i][j] = tot[j];
1201            }
1202        }
1203    }
1204
1205    for(int i=0; i<P; i++)
1206    {
1207        for(int j=0; j<R; j++)
1208        {
1209            if(all[i][j] > tot[j])
1210            {
1211                all[i][j] = tot[j];
1212            }
1213        }
1214    }
1215
1216    for(int i=0; i<P; i++)
1217    {
1218        for(int j=0; j<R; j++)
1219        {
1220            if(all[i][j] > tot[j])
1221            {
1222                all[i][j] = tot[j];
1223            }
1224        }
1225    }
1226
1227    for(int i=0; i<P; i++)
1228    {
1229        for(int j=0; j<R; j++)
1230        {
1231            if(all[i][j] > tot[j])
1232            {
1233                all[i][j] = tot[j];
1234            }
1235        }
1236    }
1237
1238    for(int i=0; i<P; i++)
1239    {
1240        for(int j=0; j<R; j++)
1241        {
1242            if(all[i][j] > tot[j])
1243            {
1244                all[i][j] = tot[j];
1245            }
1246        }
1247    }
1248
1249    for(int i=0; i<P; i++)
1250    {
1251        for(int j=0; j<R; j++)
1252        {
1253            if(all[i][j] > tot[j])
1254            {
1255                all[i][j] = tot[j];
1256            }
1257        }
1258    }
1259
1260    for(int i=0; i<P; i++)
1261    {
1262        for(int j=0; j<R; j++)
1263        {
1264            if(all[i][j] > tot[j])
1265            {
1266                all[i][j] = tot[j];
1267            }
1268        }
1269    }
1270
1271    for(int i=0; i<P; i++)
1272    {
1273        for(int j=0; j<R; j++)
1274        {
1275            if(all[i][j] > tot[j])
1276            {
1277                all[i][j] = tot[j];
1278            }
1279        }
1280    }
1281
1282    for(int i=0; i<P; i++)
1283    {
1284        for(int j=0; j<R; j++)
1285        {
1286            if(all[i][j] > tot[j])
1287            {
1288                all[i][j] = tot[j];
1289            }
1290        }
1291    }
1292
1293    for(int i=0; i<P; i++)
1294    {
1295        for(int j=0; j<R; j++)
1296        {
1297            if(all[i][j] > tot[j])
1298            {
1299                all[i][j] = tot[j];
1300            }
1301        }
1302    }
1303
1304    for(int i=0; i<P; i++)
1305    {
1306        for(int j=0; j<R; j++)
1307        {
1308            if(all[i][j] > tot[j])
1309            {
1310                all[i][j] = tot[j];
1311            }
1312        }
1313    }
1314
1315    for(int i=0; i<P; i++)
1316    {
1317        for(int j=0; j<R; j++)
1318        {
1319            if(all[i][j] > tot[j])
1320            {
1321                all[i][j] = tot[j];
1322            }
1323        }
1324    }
1325
1326    for(int i=0; i<P; i++)
1327    {
1328        for(int j=0; j<R; j++)
1329        {
1330            if(all[i][j] > tot[j])
1331            {
1332                all[i][j] = tot[j];
1333            }
1334        }
1335    }
1336
1337    for(int i=0; i<P; i++)
1338    {
1339        for(int j=0; j<R; j++)
1340        {
1341            if(all[i][j] > tot[j])
1342            {
1343                all[i][j] = tot[j];
1344            }
1345        }
1346    }
1347
1348    for(int i=0; i<P; i++)
1349    {
1350        for(int j=0; j<R; j++)
1351        {
1352            if(all[i][j] > tot[j])
1353            {
1354                all[i][j] = tot[j];
1355            }
1356        }
1357    }
1358
1359    for(int i=0; i<P; i++)
1360    {
1361        for(int j=0; j<R; j++)
1362        {
1363            if(all[i][j] > tot[j])
1364            {
1365                all[i][j] = tot[j];
1366            }
1367        }
1368    }
1369
1370    for(int i=0; i<P; i++)
1371
```

```

    printf("enter 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 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);
    avail(all, tot, avail);
    need(all, max, needmat);
    need1(all, max, needmat);

    printf("need mat");
    for(int i=0; i<P; i++){
        for(int j=0; j<R; j++){
            printf("%d", needmat[i][j]);
        }
    }
    printf("\n");

    Safety(P, R, all, avail, needmat, seq);
    printf("safe sequence is");
    printf("\n");
}

```

```

    if(i==0, i<p; i++)

for(int i=0; i<n; i++)
    for(int j=0; j<m; j++)
        printf("%d %d", seq[i]);
}

```

### Output

enter the number of processes 5

enter the number of resources 3

enter available resources 3 3 2

enter maximum resources for each processes

3 2 2

9 0 2

2 2 2

4 3 3

enter the allocated proc

enter the allocated resources for each processes 0 1 0

3 0 2

3 0 2

2 1 1

0 0 2

process	allocation	max	need
P1	0 1 0	7 5 3	7 4 3
P2	3 0 2	3 2 2	0 2 0
P3	3 0 2	9 0 2	6 0 0
P4	2 1 1	2 2 2	0 1 1
P5	0 0 2	4 3 3	4 3 1

Safety sequence P2 P3

Safety sequence P2 P3 P4 P5 P1

03/07/2024

8) write a c program to simulate deadlock detection

#include<

#include<stdio.h>

void main()

{

int n,m,i,j;

printf("enter the number of processes and number of types of resources\n");

\n");

```

scanf("%d %d", &n, &m);
scanf("%d %d", &un, &lm);
int max[n][m], need[n][m];
c = 0;
printf("enter the maximum
each process \n");

```

```

for(i=0; i<n; i++)
{
    for(j=0; j<m; j++)
    {
        scanf("%d", &max[i][j]);
    }
}

```

```

printf("enter the allocated
each process \n");

```

```

for(i=0; i<n; i++)
{
    for(j=0; j<m; j++)
    {
        scanf("%d", &alloc[i][j]);
    }
}

```

printf("enter the available resources\n");

```

for(j=0; j<m; j++)
{
    scanf("%d", &ava[j]);
}

for(i=0; i<n; i++)
{
    for(j=0; j<m; j++)
    {
        need[i][j] = max[i][j] - alloc[i][j];
    }
}

```

```

scanf("%d %d", &n, &m);
scanf("%d %d", &n, &m);
int max[n][m], need[n][m], all[n][m], ava[m], flag=1, finish[n];
c = 0;
printf("enter the maximum 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", &max[i][j]);
    }
}
printf("enter the allocated 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", &all[i][j]);
    }
}
printf("enter the available number of each type of resource \n");
for(j=0; j<m; j++)
{
    scanf("%d", &ava[j]);
}
for(i=0; i<n; i++)
{
    for(j=0; j<m; j++)
    {
        need[i][j] = max[i][j] - all[i][j];
    }
}

```

```

for(i=0; i<n; i++)
{
    finish[i]=0;
}
while(flag)
{
    flag=0;
    for(i=0; i<n; i++)
    {
        for(j=0; j<m; j++)
        {
            if(finish[i]==0 && need[i][j]<=available[j])
            {
                c=c+1;
                if(c==m)
                {
                    for(j=0; j<m; j++)
                    {
                        available[j]=available[j]+alloc[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=j+1;
        flag=1;
    }
}
if(flag==1)
{
    printf("deadlock has occurred");
    printf("the deadlock processes are");
    printf("%d", dead[0]);
    for(i=1; i<j; i++)
    {
        printf(", %d", dead[i]);
    }
    else
    {
        printf("no deadlock has occurred");
    }
}

```

Output

enter the number of process  
5  
enter the maximum number each process

```

}
j=0;
flag=0;
for(i=0; i<n; i++)
{
    if(finish[i]==0)
    {
        dead[j]=i;
        j=j+1;
        flag=1;
    }
}
if(flag==1)
{
    printf(" deadlock has occurred \n");
    printf(" the deadlock processes are ");
    printf(" the deadlock processes are \n");
    for(i=0; i<n; i++)
    {
        printf("%d", dead[i]);
    }
}
else
{
    printf(" no deadlock has occurred \n");
}

```

#### Output:

enter the number of process and number of types of resources  
 5 3  
 enter the maximum number of each type of resource needed by each process

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

~~enter the allocated number of each~~

enter the allocated number of each type of resource needed by each process

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

enter the available number of each type of resource

3 3 2

~~deadlock~~

no deadlock has occurred

~~q) write a c program to simulate the follow~~

q) write a c program to simulate the following contiguous memory allocation techniques

- (a) worst-fit
- (b) best-fit
- ~~worst-fit~~
- (c) 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("m~~

```
'printf("memory man
printf("file.no")
printf(" file.no")
for(int i=0; i<n_file
```

```
for(int j=0; j<
```

```
if(blocks[j].is_f
```

```
blocks[j].is_f
```

```

    printf("memory management scheme - First fit\n");
    printf("file-no\t file-size\t block-no\t block-size\t fragment\n");
    for(int i=0; i<n_files; i++){
        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%dt\t%d\t%d\t", files[i].file_no,
                       files[i].file_size, blocks[j].block_no, blocks[j].block_size);
                blocks[j].block_size -= files[i].file_size;
                break;
            }
        }
    }
}

void worstfit(struct block blocks[], int n_blocks, struct file files[], int n_files) {
    printf("memory management scheme - Worst fit\n");
    printf("file-no\t file-size\t block-no\t block-size\t fragment\n");
    for(int i=0; i<n_files; i++){
        int 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].file_size;
                if(fragment > max_fragment){
                    max_fragment = fragment;
                    worst_fit_block = j;
                }
            }
        }
    }
}

```

```

        worst-fit-block = j;
    }
}

if(worst-fit-block) == -1 {
    blocks[worst-fit-block].is_free = 0;
    printf("%d\t%d\t%d\t%d\n", files[i].file_no,
           files[i].file_size, blocks[worst-fit-block].block_no,
           blocks[worst-fit-block].block_size, max_fragment);
}
}

void bestfit(struct block blocks[], int n_blocks, struct file files[],
             int n_files) {
    printf("memory management scheme best-fit\n");
    printf("file_no\tfile_size\tblock_no\tblock_size\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 = blocks[j].block_size - files[i].file_size;
                if(fragment < min_fragment) {
                    min_fragment = fragment;
                    best-fit-block = j;
                }
            }
        }
        if(best-fit-block != -1) {
    }
}

```

```

block[best_fi] = block
printf("%d\n", best_fi);
file[best_fi] = file - 1;
blocks[best_fi] = 0;
}
}

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

struct file filest;
struct file files[n_files];
for (int i = 0; i < n_files; i++)
{
    printf("enter size of file %d", i);
    scanf("%d", &files[i].size);
}

firstfit(blocks, n_blocks, fi);

```

```

blocks[best-fit-block].is-free = 0;
printf("%d\t%d\t%d\t%d\t%d\n", files[i].file-no,
      file-size, blocks[best-fit-block].block-no,
      blocks[best-fit-block].file-size, min-fragment);
}

int main()
{
    int n-blocks, n-files;
    printf("enter number of blocks ");
    scanf("%d", &n-blocks);
    printf("enter 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 size of block %d", i+1);
        blocks[i].is-free = 1;
    }
    struct file files;
    struct file files[n-files];
    for(int i=0; i<n-files; i++)
    {
        printf("enter 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);

```

V bestfit(blocks, n-blocks, files, n-files);

### Output

enter the number of blocks 3  
 enter the number of files 2  
 enter the size of block1 5  
 enter the size of block2 2  
 enter the size of block3 7  
 enter the size of file1 1  
 enter the size of file2 4

memory management scheme - first fit

file no	file size	block no	block size	fragment
1	1	1	8	4
2	4	3	7	3

memory management scheme worst-fit

file no	file size	block no	block size	fragment
1	1	3	7	1
2	4	1	5	1

memory management scheme best-fit

file no	file size	block no	block size	fragment
1	1	2	2	1
2	4	1	5	1

10/07/2024  
 10) write a c program to simulate pag memory management

```

#include<stdio.h>
#include<climits.h>
#include<cslib.h>

```

void print\_frames(int frame[], int capacity,

```
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(" PF No %d", page-faultb);}
```

```
}
```

```
printf("\n");
```

void fifo(int pages[], int n, int capacity)

```
int frame[capacity], index=0, page-fa
```

```
for(int i=0; i<capacity; i++)
```

```
{
```

```
frame[i]=-1;
```

```
}
```

```
printf("FIFO page replacement process");
```

```
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;
```

```
}
```

10/03/2024  
10) 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("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 Processes\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==0)
        {
            frame[index]=pages[i];
            index=(index+1)%capacity;
            page_faults++;
        }
    }
}
```

```

        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\n");
printf("total page faults using fifo %d\n\n", page_faults);
}

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;
    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, min_index;
            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);
}

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

```

```

        min_index=j;
    }

    frame[min_index]=pages[i];
    counter[min_index]=time+i;
    page_faults++;

}

printf("print_frames(frame, capacity, found ? 0 : page_faults);");

printf("total page faults using LRU %d\n\n", page_faults);

};

faults=0;
}

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]){
                found=1;
                break;
            }
        }
        if(!found){
            int farthest=i+1, index=-1;
            for(int j=0; j<capacity; j++){
                for(int k=0; 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++;
}
printf("frames(%d, %d, found? %d): page-faults %d\n", frame, capacity, found ? 0 : page_faults);
printf("total page faults using optimal %d\n", page_faults);
printf("Total page faults using Optimal %d\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(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");
fifo(pages, n, capacity);
FIFO(pages, n, capacity);
LRU(pages, n, capacity);
Optimal(pages, n, capacity);
free(pages);
}

```

Output

enter the number of pages 20  
 enter the pages 7,0,1,2,0,3,0  
 enter the frame capacity 3  
 pages: 7 0 1 2 0 3 0 4 2 3 0  
~~free~~ page  
 FIFO page replacement process

7 - - PF no 1  
 7 0 - PF no 2  
 7 0 1 - PF no 3  
 2 0 1 - PF no 4  
 2 0 1  
 2 3 1 - PF no 5  
 2 3 0 - PF no 6

~~4 2 0~~  
~~4 2 1~~  
 4 3 0 - PF no 7  
 4 2 0 - PF no 8  
 4 2 3 - PF no 9  
 0 2 3 - PF no 10  
 0 2 3  
 0 2 3  
 0 1 3 - PF no 11  
 0 1 2 - PF no 12  
 0 1 2  
 0 1 2  
 7 1 2 - PF no 13  
 7 0 2 - PF no 14  
 7 0 1 - PF no 15

total page faults using FIFO 15

```

    for(int i=0; i<n; i++){
        printf("%d", pages[i]);
    }
    printf("\n");
    fifo(pages, n, capacity);
    fifo(pages, n, capacity);
    lru(pages, n, capacity);
    optimal(pages, n, capacity);
    free
    free(pages);
}

```

output

enter the number of pages 20

enter the pages 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

enter the frame capacity 3

pages: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

fifo

fifo page replacement process

7 - - PF no 1

7 0 - PF no 2

7 0 1 - PF no 3

2 0 1 - PF no 4

2 0 1

2 3 1 - PF no 5

2 3 0 - PF no 6

~~4 3 0~~

4 3 0 - ~~PF no 7~~

4 2 0 - PF no 8

4 2 3 - PF no 9

0 2 3 - PF no 10

0 2 3

0 2 3

0 1 3 - PF no 11

0 1 2 - PF no 12

0 1 2

0 1 2

7 1 2 - PF no 13

7 0 2 - PF no 14

7 0 1 - PF no 15

total page faults using fifo 15

## Colour Me



lru page replacement process:

lru page replacement process:

7 - -	PF no 1	7 - -	PF no 1
0 - -	PF no 2	0 - -	PF no 2
0 1 -	PF no 3	0 1 -	PF no 3
0 1 2	PF no 4	0 1 2	PF no 4
0 3 2	PF no 5	0 1 2	PF no 5
0 3 2	PF no 5	0 3 2	PF no 6
0 3 4	PF no 6	0 3 4	PF no 6
0 2 4	PF no 7	0 2 4	PF no 7
3 2 4	PF no 8	3 2 4	PF no 8
3 2 0	PF no 9	3 2 0	PF no 9
3 2 1		3 2 0	
0 2 1	PF no 10	3 2 1	PF no 10
0 2 1		3 2 1	PF no 10
0 7 1		0 2 1	PF no 11
0 7 1		0 2 1	PF no 11
0 7 1		0 7 1	PF no 12
		0 7 1	
		0 7 1	

Total page faults using lru 12

Optimal page replacement process

7 - -	PF no 1
7 0 -	PF no 2
7 0 1	PF no 3
2 0 1	PF no 4
2 0 1	PF no 5
2 0 3	PF no 5 PF no 3
2 0 3	
2 4 3	PF no 6
2 4 3	
2 4 3	
2 0 3	PF no 7
2 0 3	
2 0 3	
2 0 1	PF no 8
2 0 1	
2 0 1	
2 0 1	PF no 9
7 0 1	
7 0 1	

Total page faults op'

Total page faults using optimal 9