VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

OPERATING SYSTEMS

Submitted by

YASHWANTH S(1WA23CS050)

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

Feb-2025 to June-2025

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 Yashwanth S(1WA23CS050), who is Bonafide student of B. M. S. College of Engineering. It is in partial fulfilment for the award of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum during the year Feb 2025- June 2025. 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.

Prof. Sandhya A Kulkarni
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

10.	Write a C program to simulate page replacement algorithms a) FIFO b) LRU c) Optimal	95-105

Course Outcomes

C01	Apply the different concepts and functionalities of Operating System
C02	Analyse various Operating system strategies and techniques
C03	Demonstrate the different functionalities of Operating System.
C04	Conduct practical experiments to implement the functionalities of Operating system.

Program 1:

Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time.

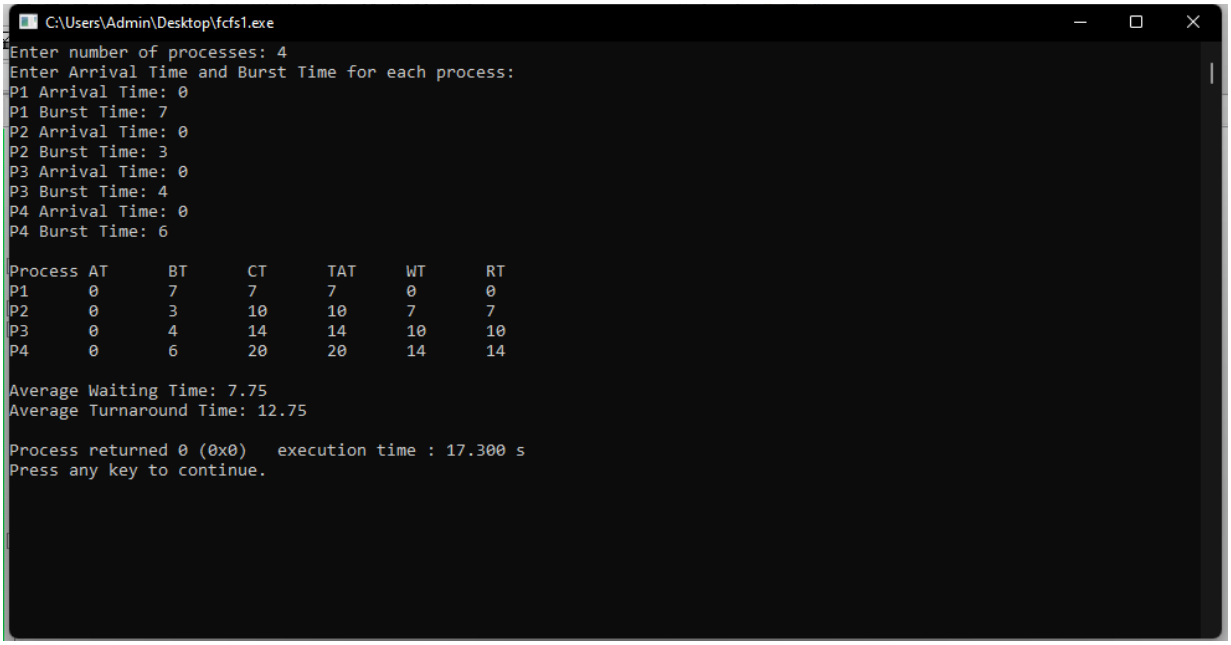
→FCFS

→ SJF (pre-emptive & Non-preemptive)

→FCFS

```
#include <stdio.h>

struct Process {
    int at;
    int bt;
    int ct;
    int tat;
    int wt;
    int rt;
};

void calculateFCFS(struct Process proc[], int n) {
    int time = 0;
    for (int i = 0; i < n; i++) {
        if (time < proc[i].at) {
            time = proc[i].at;
        }
        proc[i].ct = time + proc[i].bt;
        proc[i].tat = proc[i].ct - proc[i].at;
        proc[i].wt = proc[i].tat - proc[i].bt;
        proc[i].rt = time - proc[i].at;
        time = proc[i].ct;
    }
}
int main() {
```

```
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    struct Process proc[n];
    printf("Enter Arrival Time and Burst Time for each process:\n");
    for (int i = 0; i < n; i++) {
        printf("P%d Arrival Time: ", i + 1);
        scanf("%d", &proc[i].at);
        printf("P%d Burst Time: ", i + 1);
        scanf("%d", &proc[i].bt);
    }
    calculateFCFS(proc, n);
    printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\tRT\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\n", i + 1, proc[i].at, proc[i].bt, proc[i].ct, proc[i].tat,
proc[i].wt, proc[i].rt);
    }
    float totalWT = 0, totalTAT = 0;
    for (int i = 0; i < n; i++) {
        totalWT += proc[i].wt;
        totalTAT += proc[i].tat;
    }
    printf("\nAverage Waiting Time: %.2f", totalWT / n);
    printf("\nAverage Turnaround Time: %.2f\n", totalTAT / n);
    return 0;
}
```


C:\Users\Admin\Desktop\fcfs1.exe
Enter number of processes: 4
Enter Arrival Time and Burst Time for each process:
P1 Arrival Time: 0
P1 Burst Time: 7
P2 Arrival Time: 0
P2 Burst Time: 3
P3 Arrival Time: 0
P3 Burst Time: 4
P4 Arrival Time: 0
P4 Burst Time: 6
Process AT BT CT TAT WT RT
P1 0 7 7 7 0 0
P2 0 3 10 10 7 7
P3 0 4 14 14 10 10
P4 0 6 20 20 14 14
Average Waiting Time: 7.75
Average Turnaround Time: 12.75
Process returned 0 (0x0) execution time : 17.300 s
Press any key to continue.

Teacher Sign / Remarks

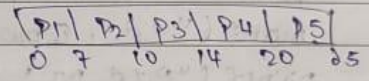
Date / /
Page

1. Write a C program to simulate the following CPU Scheduling algorithm to find average turnaround time & waiting time.
   i) FCFS
   ii) SJF (non-preemptive & preemptive)

(i)

Process	AT	BT	CT	TAT	WT	RT
1	0	7	7	7	0	0
2	0	3	10	10	7	7
3	0	4	14	14	10	10
4	0	6	20	20	14	14
5	0	5	25	25	20	20

Gantt Chart



Program

```
#include<stdio.h>
struct Process {
    int at;
    int bt;
    int ct;
    int tat;
    int wt;
    int rt;
};
void calculateFCFS(struct Process Proc[], int n) {
    int time=0;
    for(int i=0;i<n;++i) {
        if (time < proc[i].at)
            time = proc[i].at; }
    proc[i].ct = time + proc[i].bt;
    proc[i].tat = proc[i].ct - proc[i].at;
```

O/P

Enter number of process: 4

Enter arrival time and burst time for each process:

P1 AT: 0

P1 BT: 7

P2 AT: 0

P2 BT: 3

P3 AT: 0

P3 BT: 4

P4 AT: 0

P4 BT: 6

Process	AT	BT	CT	TAT	WT	RT
P1	0	7	7	7	0	0
P2	0	3	10	10	7	7
P3	0	4	14	14	10	10
P4	0	6	20	20	14	14.

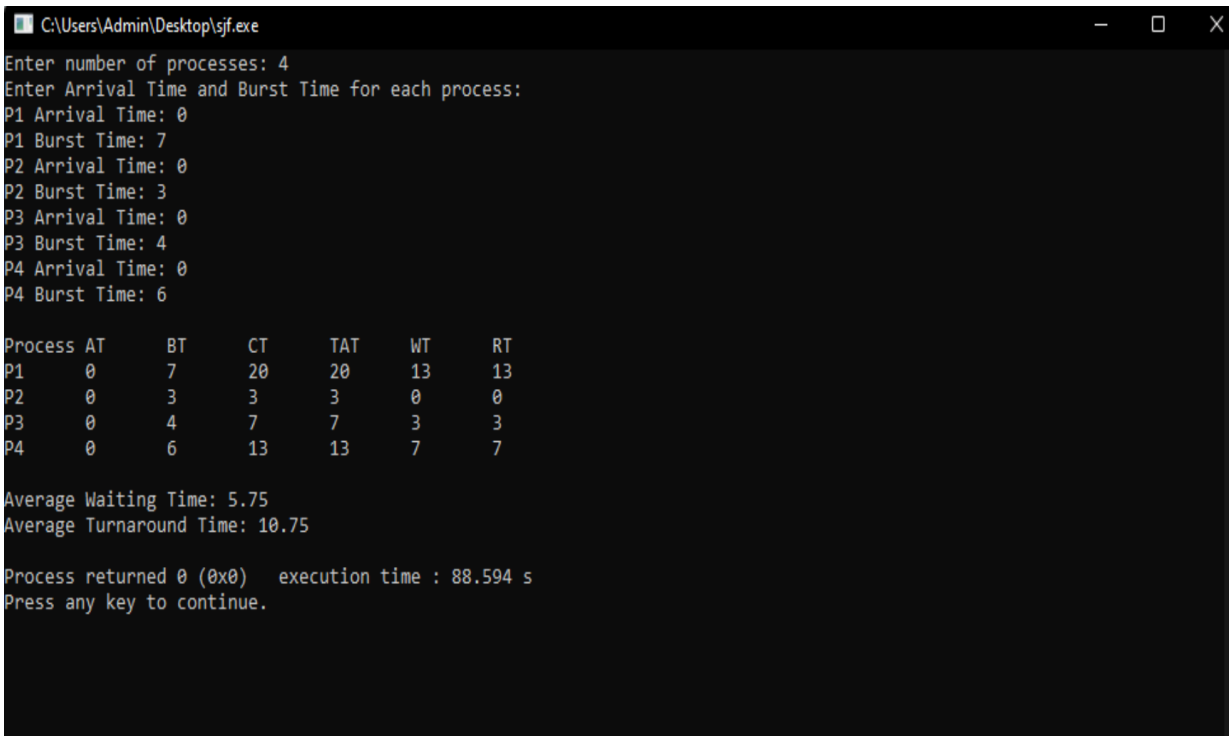
```
        scanf("%d", &proc[i].bt);
    }
calcFCFS(proc, n);
printf("\n Process \t AT \t BT \t TAT \t WT \t RT \n");
for(int i=0; i<n; i++){
    printf("P%d \t %d \t %d \t %d \t %d \t %d \t %d\n",
    i+1, proc[i].at, proc[i].bt, proc[i].ct, proc[i].TAT,
    proc[i].wt, proc[i].RT);
    }
return 0;
}
```

→SJF(Non Preemptive)

```
#include <stdio.h>
#include <limits.h>
struct Process {
    int at;
    int bt;
    int ct;
    int tat;
    int wt;
    int rt;
    int pid;
};
void calculateSJF(struct Process proc[], int n) {
    int time = 0;
    int completed = 0;
    int min_index;
    int is_completed[n];
    for (int i = 0; i < n; i++) {
        is_completed[i] = 0;
    }
    while (completed < n) {
        min_index = -1;
        int min_bt = INT_MAX;
        for (int i = 0; i < n; i++) {
            if (proc[i].at <= time && !is_completed[i] && proc[i].bt < min_bt) {
                min_bt = proc[i].bt;
                min_index = i;
            }
```

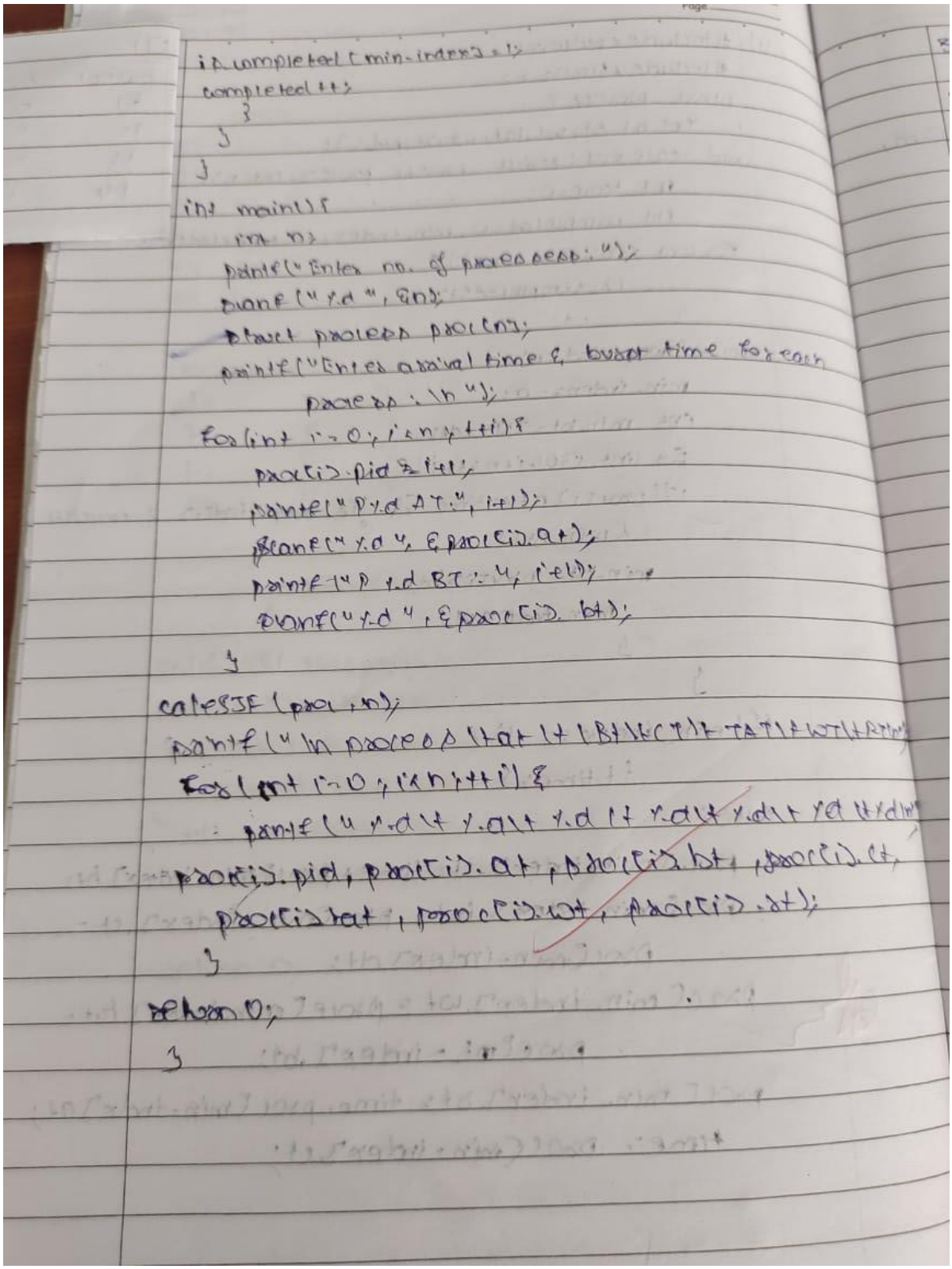
```
        }
        if (min_index == -1) {
            time++;
        } else {
            proc[min_index].ct = time + proc[min_index].bt;
            proc[min_index].tat = proc[min_index].ct - proc[min_index].at;
            proc[min_index].wt = proc[min_index].tat - proc[min_index].bt;
            proc[min_index].rt = time - proc[min_index].at;
            time = proc[min_index].ct;
            is_completed[min_index] = 1;
            completed++;
        }
    }
}
int main() {
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    struct Process proc[n];
    printf("Enter Arrival Time and Burst Time for each process:\n");
    for (int i = 0; i < n; i++) {
        proc[i].pid = i + 1;
        printf("P%d Arrival Time: ", i + 1);
        scanf("%d", &proc[i].at);
        printf("P%d Burst Time: ", i + 1);
        scanf("%d", &proc[i].bt);
    }
    calculateSJF(proc, n);
    printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\tRT\n");
    for (int i = 0; i < n; i++) {
```

```
        printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\n", proc[i].pid, proc[i].at, proc[i].bt, proc[i].ct, proc[i].tat, proc[i].wt, proc[i].rt);
    }
    return 0;
}
```

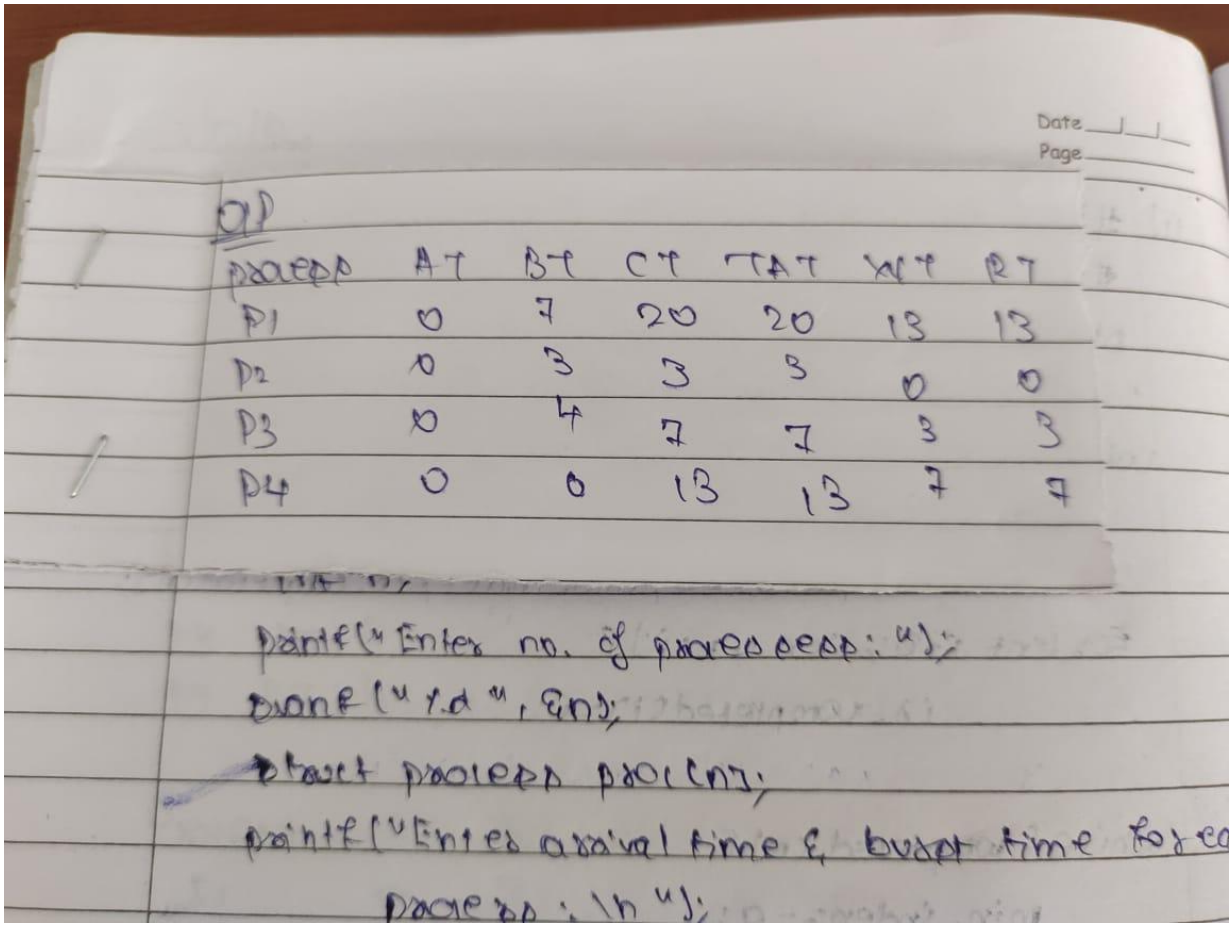


ii)

```
#include <stdio.h>
#include <limits.h>
struct process {
    int at, bt, wt, tat, ct, id, pid ; };
void calc_fcfs (struct process proc[], int n){
    int time=0;
    int completed=0, min_index, is_completed[n];
    for(int i=0; i<n ; ++i){
        is_completed[i]=0;
    }
while(completed<n){
    min_index=-a;
    int min_bt=INT_MAX;
    for(int i=0; i<n ;++i){
    if(proc[i].at<=time && !is_completed[i] && proc[i].at
        <min_bt){
        min_bt=proc[i].bt;
        min_index=i;
        }
    }
if(min_index==-1){
        ++time;}
else{
    proc[min_index].ct=time+proc[min_index].bt;
    proc[min_index].tat=proc[min_index].ct-
        proc[min_index].at;
proc[min_index].wt=proc[min_index].tat-
        proc[int_index].bt;
proc[min_index].st=time-proc[min_index].at;
    time=proc[min_index].ct;
```



```
        is_completed[min_index] = 1;
        completed++;
      }
    }
    current_time++;
  }
}

int main() {
  int n;
  printf("Enter no. of processes: ");
  scanf("%d", &n);
  struct process proc[n];
  printf("Enter arrival time & burst time for each
        process:\n");
  for(int i=0; i<n; ++i) {
    proc[i].pid = i+1;
    printf("P%d AT: ", i+1);
    scanf("%d", &proc[i].at);
    printf("P%d BT: ", i+1);
    scanf("%d", &proc[i].bt);
  }
calcSJF(proc, n);
printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\tRT\n");
for(int i=0; i<n; ++i) {
  printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n",
proc[i].pid, proc[i].at, proc[i].bt, proc[i].ct,
  proc[i].tat, proc[i].wt, proc[i].rt);
}
return 0;
}
```



Date __/__/__
Page

O/P

process	AT	BT	CT	TAT	WT	RT
P1	0	7	20	20	13	13
P2	0	3	3	3	0	0
P3	0	4	7	7	3	3
P4	0	0	13	13	7	7

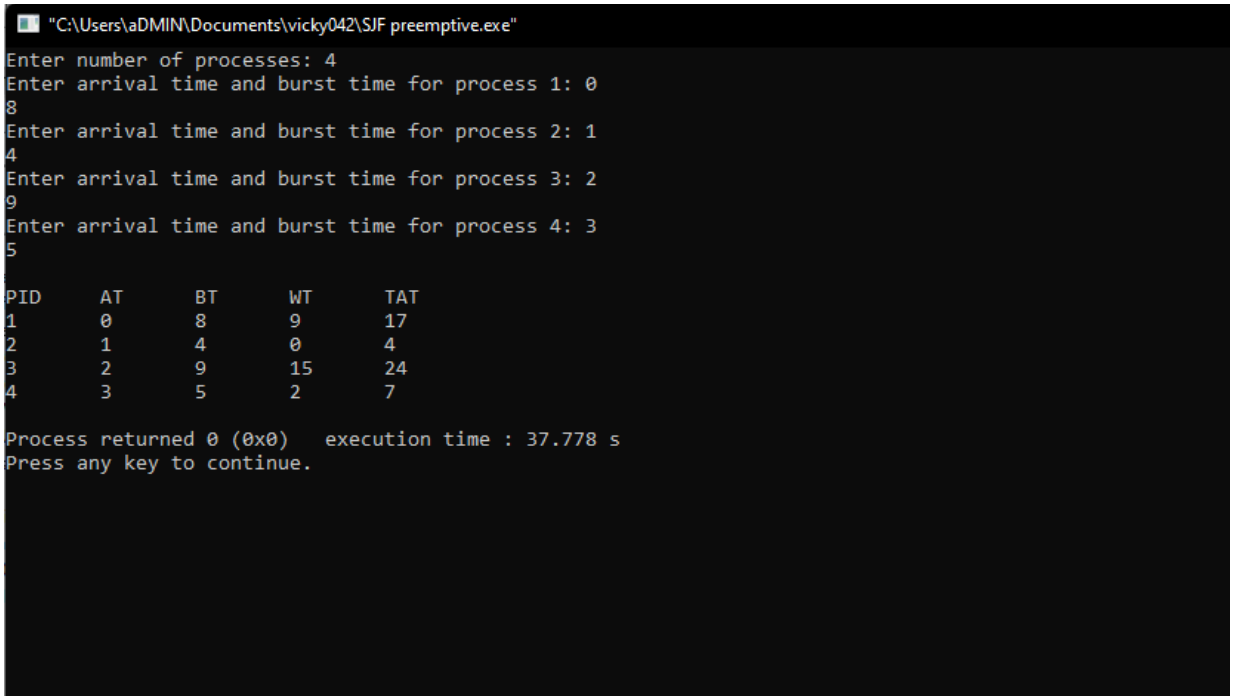
```
printf("Enter no. of processes: ");
scanf("%d", &n);
struct process p[n];
printf("Enter arrival time & burst time for ea
        process : \n");
```

→SJF(Pre-emptive)

```
#include <stdio.h>

#define MAX 10

typedef struct {
   int pid, at, bt, rt, wt, tat, completed;
} Process;

void sjf_preemptive(Process p[], int n) {
   int time = 0, completed = 0, shortest = -1, min_bt = 9999;

   while (completed < n) {
      shortest = -1;
      min_bt = 9999;

      for (int i = 0; i < n; i++) {
         if (p[i].at <= time && p[i].rt > 0 && p[i].rt < min_bt) {
            min_bt = p[i].rt;
            shortest = i;
         }
      }

      if (shortest == -1) {
         time++;
         continue;
      }

      p[shortest].rt--;
      time++;
```

```
        if (p[shortest].rt == 0) {
            p[shortest].completed = 1;
            completed++;
            p[shortest].tat = time - p[shortest].at;
            p[shortest].wt = p[shortest].tat - p[shortest].bt;
        }
    }
}

int main() {
    Process p[MAX];
    int n;

    printf("Enter number of processes: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("Enter arrival time and burst time for process %d: ", p[i].pid);
        scanf("%d %d", &p[i].at, &p[i].bt);
        p[i].rt = p[i].bt;
        p[i].completed = 0;
    }

    sjf_preemptive(p, n);

    printf("\nPID\tAT\tBT\tWT\tTAT\n");
    for (int i = 0; i < n; i++)
        printf("%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].at, p[i].bt, p[i].wt, p[i].tat);
```

```
    return 0;
}
```


"C:\Users\aDMIN\Documents\vicky042\SJF preemptive.exe"
Enter number of processes: 4
Enter arrival time and burst time for process 1: 0
8
Enter arrival time and burst time for process 2: 1
4
Enter arrival time and burst time for process 3: 2
9
Enter arrival time and burst time for process 4: 3
5
PID AT BT WT TAT
1 0 8 9 17
2 1 4 0 4
3 2 9 15 24
4 3 5 2 7
Process returned 0 (0x0) execution time : 37.778 s
Press any key to continue.

Date / /
Page

8. SJF Preemptive.

```
#include<stdio.h>
#define MAX 100
typedef struct {
    int at, bt, rt, ct, tat, wt;} process;
int main(){
  int n, completed=0, time=0, min_rt, sht;
  process p[MAX];
  printf("Enter the no. of processes: ");
  scanf("%d", &n);
  for(int i=0; i<n; ++i){
    printf("AT & BT of P%d: ", i+1);
    scanf("%d %d", &p[i].at, &p[i].bt);
    p[i].rt = p[i].bt;
  }
while(completed<n){
    sht=-1; min_rt=10000;
  for(int i=0; i<n; ++i)
      if(p[i].at<=time && p[i].rt>0 &&
          p[i].rt<min_rt)
        min_rt=p[i].rt; sht=i;
    if(sht==-1){
      ++time; continue;}
    p[sht].rt--;
    if(p[sht].rt==0){
      p[sht].ct=time+1;
      p[sht].tat=p[sht].ct-p[sht].at;
      p[sht].wt=p[sht].tat-p[sht].bt;
      completed++;
    } time++;
  }
```

O/P

Enter the no. of process : 5

Enter AT, BT & priority for each process

Process 1 : 0

10 3

Process 2 : 0 1 1

Process 3 : 0 2 4

Process 4 : 0 1 5

Process 5 : 0 5 2

PID	AT	BT	PR	CT	TAT	WT	RT
P1	0	10	3	16	16	6	6
P2	0	1	1	1	1	0	0
P3	0	2	4	18	18	16	16
P4	0	1	5	19	19	18	18
P5	0	5	2	6	6	1	1

Program 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)

→Priority(Non-pre-emptive)

```
#include <stdio.h>


struct Process {
   int pid, at, bt, pr, ct, wt, tat, rt;
   int isCompleted; // Flag to check if process is completed
};


void sortByArrival(struct Process p[], int n) {
   for (int i = 0; i < n - 1; i++) {
      for (int j = i + 1; j < n; j++) {
         if (p[i].at > p[j].at) {
            struct Process temp = p[i];
            p[i] = p[j];
            p[j] = temp;
         }
      }
   }
}


void findPriorityScheduling(struct Process p[], int n) {
   sortByArrival(p, n);
   int time = 0, completed = 0;
   float totalWT = 0, totalTAT = 0;
```

```
while (completed < n) {
    int idx = -1, highestPriority = 9999;

    for (int i = 0; i < n; i++) {
        if (p[i].at <= time && p[i].isCompleted == 0) {
            if (p[i].pr < highestPriority) {
                highestPriority = p[i].pr;
                idx = i;
            }
        }
    }

    if (idx == -1) {
        time++; // CPU idle
    } else {
        p[idx].rt = time - p[idx].at;
        time += p[idx].bt;
        p[idx].ct = time;
        p[idx].tat = p[idx].ct - p[idx].at;
        p[idx].wt = p[idx].tat - p[idx].bt;
        p[idx].isCompleted = 1;

        totalWT += p[idx].wt;
        totalTAT += p[idx].tat;
        completed++;
    }
}

printf("PID\tAT\tBT\tPR\tCT\tTAT\tWT\tRT\n");
for (int i = 0; i < n; i++) {
```

```
        printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n",
               p[i].pid, p[i].at, p[i].bt, p[i].pr, p[i].ct, p[i].tat, p[i].wt, p[i].rt);
    }

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

int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    struct Process p[n];

    printf("Enter Arrival Time, Burst Time, and Priority for each process:\n");
    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("Process %d: ", i + 1);
        scanf("%d %d %d", &p[i].at, &p[i].bt, &p[i].pr);
        p[i].isCompleted = 0;
    }

    findPriorityScheduling(p, n);
    return 0;
}
```

```
Enter the number of processes: 5
Enter Arrival Time, Burst Time, and Priority for each process:
Process 1: 0
10
3
Process 2: 0
1
1
Process 3: 0
2
4
Process 4: 0
1
5
Process 5: 0
5
2
PID AT  BT  PR  CT  TAT WT  RT
1   0   10  3   16  16  6   6
2   0   1   1   1   1   0   0
3   0   2   4   18  18  16  16
4   0   1   5   19  19  18  18
5   0   5   2   6   6   1   1

Average Turnaround Time: 12.00
Average Waiting Time: 8.20
```

```
int time = 0, completed = 0;
float total wt = 0, total tat = 0;
while(completed < n){
    int idx = -1, hp = 99999;
    for(int i = 0; i < n; ++i){
        if(p[i].at <= time && p[i].iscompleted == 0){
            if(p[i].pr < hp){
                hp = p[i].pr;
                idx = i;
            }
        }
    }
    if(idx == -1){
        ++time;
    }
    else{
        p[idx].st = time - p[idx].at;
        time += p[idx].bt;
        p[idx].ct = time;
        p[idx].tat = p[idx].ct - p[idx].at;
        p[idx].wt = p[idx].tat - p[idx].bt;
        p[idx].iscompleted = 1;
        totalWT += p[idx].wt;
        totalTAT += p[idx].tat;
        completed++;
    }
}
printf("PID\tAT\tBT\tPRI\tCT\tTAT\tWT\tRT\n");
for(int i = 0; i < n; ++i){
    printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].at, p[i].bt,
p[i].pr, p[i].ct, p[i].tat, p[i].wt, p[i].st);
}
printf("\nAverage TAT: %.2f\n", totalTAT/n);
printf("Average WT: %.2f\n", totalWT/n);
}
int main(){
```

O/P

Enter number of processes: 4

AT & BT of P1: 0

8

AT & BT of P2: 1

4

AT & BT of P3: 2

9

AT & BT of P4: 3

5

AT	BT	CT	TAT	WT
0	8	17	17	9
1	4	5	4	0
2	9	21	24	15
3	5	10	7	2

Avg TAT: 13.00

Avg WT: 6.50.

}

Priority Preemptive

Priority Non-preemptive.

```
#include<stdio.h>
struct process {
    int pid, at, bt, pr, ct, wt, tat, st;
    int isCompleted;
};
void sortByArrival(struct process p[], int n){
    for(int i=0; i<n-1; ++i){
        for(int j=i+1; j<n; ++j){
            if(p[i].at > p[j].at){
                struct process temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
        }
    }
}
void findpriorityScheduling(struct process p[], int n){
    sortByArrival(p, n);
```

→Priority(Pre-emptive)

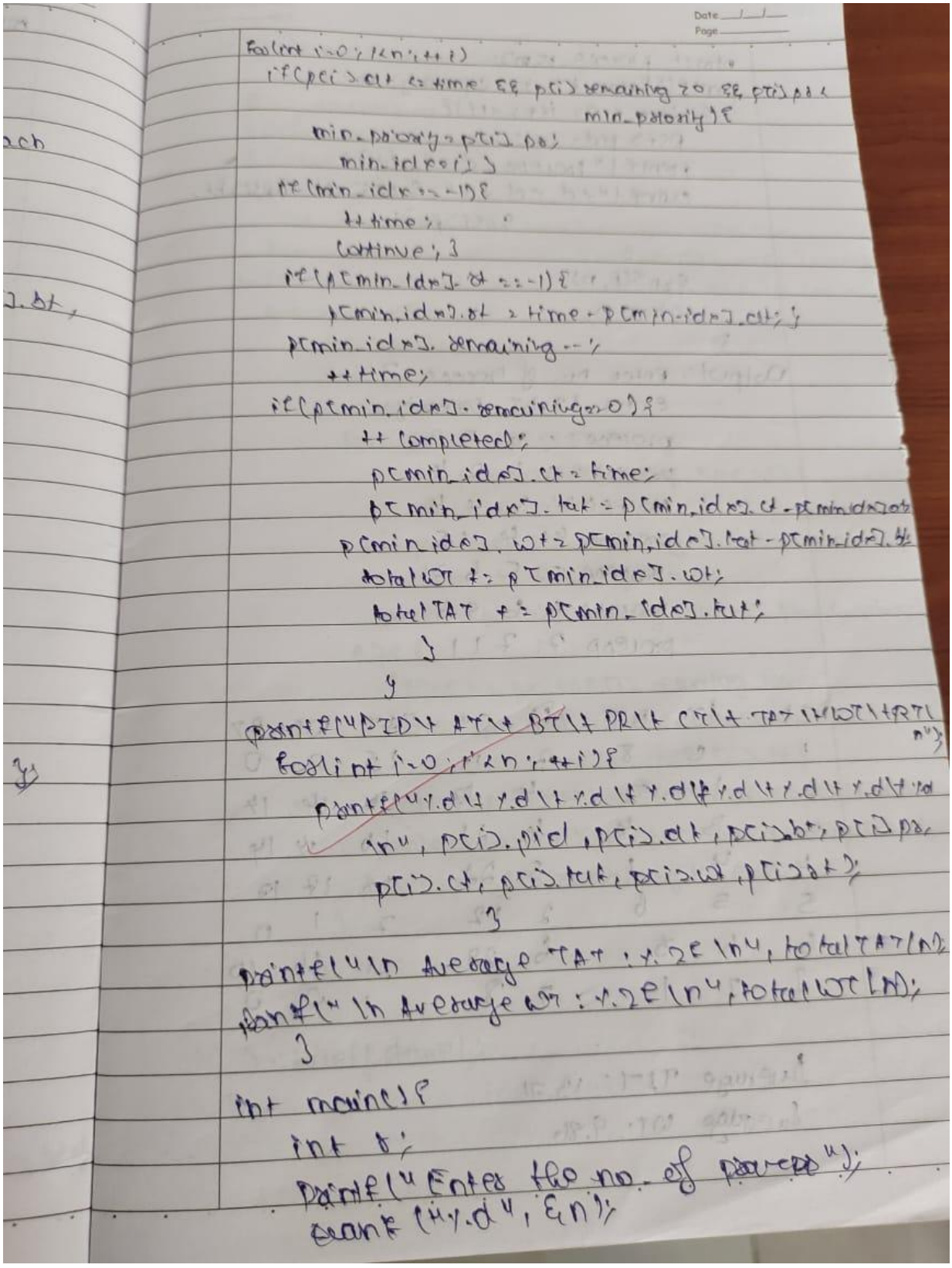
```
#include <stdio.h>
#include <limits.h>

struct Process {
    int pid, at, bt, pr, ct, wt, tat, rt, remaining;
};

void findPreemptivePriorityScheduling(struct Process p[], int n) {
    int completed = 0, time = 0, min_idx = -1;
    float totalWT = 0, totalTAT = 0;

    for (int i = 0; i < n; i++) {
        p[i].remaining = p[i].bt;
        p[i].rt = -1;
    }

    while (completed != n) {
        int min_priority = INT_MAX;
        min_idx = -1;

        for (int i = 0; i < n; i++) {
            if (p[i].at <= time && p[i].remaining > 0 && p[i].pr < min_priority) {
                min_priority = p[i].pr;
                min_idx = i;
            }
        }

        if (min_idx == -1) {
            time++;
```

```
            continue;
        }

        if (p[min_idx].rt == -1) {
            p[min_idx].rt = time - p[min_idx].at;
        }

        p[min_idx].remaining--;
        time++;

        if (p[min_idx].remaining == 0) {
            completed++;
            p[min_idx].ct = time;
            p[min_idx].tat = p[min_idx].ct - p[min_idx].at;
            p[min_idx].wt = p[min_idx].tat - p[min_idx].bt;
            totalWT += p[min_idx].wt;
            totalTAT += p[min_idx].tat;
        }
    }

    printf("PID\tAT\tBT\tPR\tCT\tTAT\tWT\tRT\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n",
               p[i].pid, p[i].at, p[i].bt, p[i].pr, p[i].ct, p[i].tat, p[i].wt, p[i].rt);
    }

    printf("\nAverage Turnaround Time: %.2f\n", totalTAT / n);
    printf("Average Waiting Time: %.2f\n", totalWT / n);
}
```

```
int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    struct Process p[n];

    printf("Enter Arrival Time, Burst Time, and Priority for each process:\n");
    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("Process %d: ", i + 1);
        scanf("%d %d %d", &p[i].at, &p[i].bt, &p[i].pr);
    }

    findPreemptivePriorityScheduling(p, n);
    return 0;
}
```

Output Clear

```
Enter the number of processes: 7
Enter Arrival Time, Burst Time, and Priority for each process:
Process 1: 0
8
3
Process 2: 1
2
4
Process 3: 3
4
4
Process 4: 4
1
5
Process 5: 5
6
2
Process 6: 6
5
6
Process 7: 7
1
1
PID AT  BT  PR  CT  TAT WT  RT
1   0   8   3   15  15  7   0
2   1   2   4   17  16  14  14
3   3   4   4   21  18  14  14
4   4   1   5   22  18  17  17
5   5   6   2   12  7   1   0
6   6   5   6   27  21  16  16
7   7   1   1   8   1   0   0

Average Turnaround Time: 13.71
Average Waiting Time: 9.86
```



```
for(int i=0; i<n; ++i)
  if(p[i].at <= time && p[i].remaining > 0 && p[i].pr <
                                    min_priority){
    min_priority = p[i].pr;
    min_idx = i; }
  if(min_idx == -1){
    ++time;
    continue; }
  if(p[min_idx].st == -1){
    p[min_idx].st = time - p[min_idx].at; }
  p[min_idx].remaining--;
    ++time;
  if(p[min_idx].remaining == 0){
      ++completed;
      p[min_idx].ct = time;
      p[min_idx].tat = p[min_idx].ct - p[min_idx].at
    p[min_idx].wt = p[min_idx].tat - p[min_idx].bt;
      totalWT += p[min_idx].wt;
      totalTAT += p[min_idx].tat;
        }
      }
printf("\nPID\t AT\t BT\t PR\t CT\t TAT\t WT\t RT\
n");
  for(int i=0; i<n; ++i){
    printf("%d\t %d\t %d\t %d\t %d\t %d\t %d\t %d
    \n", p[i].pid, p[i].at, p[i].bt, p[i].pr,
    p[i].ct, p[i].tat, p[i].wt, p[i].st);
        }
printf("\n Average TAT : %.2f\n", totalTAT/n);
printf("\n Average WT : %.2f\n", totalWT/n);
  }
int main(){
  int n;
  printf("Enter the no. of processes");
  scanf("%d", &n);
```

```
struct process p[n];
printf("Enter AT, BT, & priority: \n");
for(int i=0; i<n; ++i){
    p[i].pid = i+1;
    printf("process %d: ", i+1);
    scanf("%d %d %d", &p[i].at, &p[i].bt,
                      &p[i].pp);
}
fcfs(p, n);
return 0;
}
```

Output: Enter no. of process: 7
Enter AT, BT, priority:
process 1: 0 8 3
process 2: 1 2 4
process 3: 3 4 4
process 4: 4 1 5
process 5: 5 6 2
process 6: 6 5 6
process 7: 7 1 1

PID	AT	BT	PR	CT	TAT	WT	RT
1	0	8	3	15	15	7	0
2	1	2	4	17	16	14	14
3	3	4	4	21	18	14	14
4	4	1	5	22	18	17	17
5	5	6	2	12	7	1	0
6	6	5	6	27	21	16	10
7	7	1	7	8	1	0	0

Average TAT: 13.71
Average WT: 9.86.

Avg WT: 6.50.

Priority Preemptive

```
#include<stdio.h>
#include<limits.h>
struct process{
    int pid, at, bt, pr, ct, wt, tat, rt, remaining;};
void FPPS(struct process p[], int n){
    int completed=0, time=0, min_idx=-1;
    float totalWT=totalTAT=0;
    for(int i=0; i<n; i++){
        p[i].remaining=p[i].bt;
        p[i].rt=-1;
    }
    while(completed!=n){
        int min_priority=INT_MAX;
        min_idx=-1;
```

→Round Robin

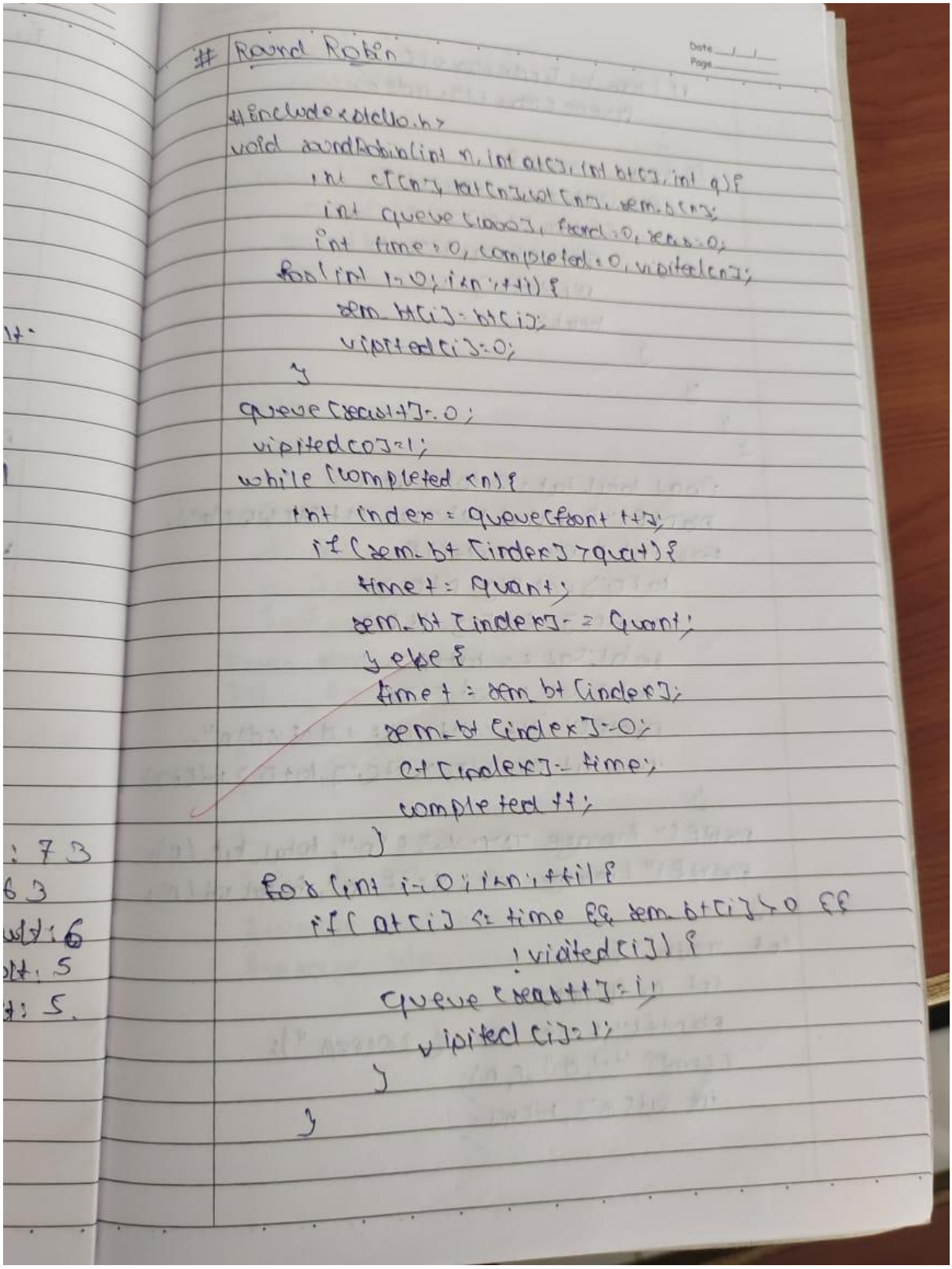
```
#include <stdio.h>

#define MAX 100

void roundRobin(int n, int at[], int bt[], int quant) {
    int ct[n], tat[n], wt[n], rem_bt[n];
    int queue[MAX], front = 0, rear = 0;
    int time = 0, completed = 0, visited[n];

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

    queue[rear++] = 0;
    visited[0] = 1;

    while (completed < n) {
        int index = queue[front++];

        if (rem_bt[index] > quant) {
            time += quant;
            rem_bt[index] -= quant;
        } else {
            time += rem_bt[index];
            rem_bt[index] = 0;
            ct[index] = time;
            completed++;
        }
```

```
    for (int i = 0; i < n; i++) {
        if (at[i] <= time && rem_bt[i] > 0 && !visited[i]) {
            queue[rear++] = i;
            visited[i] = 1;
        }
    }

    if (rem_bt[index] > 0) {
        queue[rear++] = index;
    }

    if (front == rear) {
        for (int i = 0; i < n; i++) {
            if (rem_bt[i] > 0) {
                queue[rear++] = i;
                visited[i] = 1;
                break;
            }
        }
    }
}

float total_tat = 0, total_wt = 0;
printf("P#\tAT\tBT\tCT\tTAT\tWT\n");
for (int i = 0; i < n; i++) {
    tat[i] = ct[i] - at[i];
    wt[i] = tat[i] - bt[i];
    total_tat += tat[i];
    total_wt += wt[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("Average TAT: %.2f\n", total_tat / n);
    printf("Average WT: %.2f\n", total_wt / n);
}

int main() {
    int n, quant;
    printf("Enter number of processes: ");
    scanf("%d", &n);

    int at[n], bt[n];
    for (int i = 0; i < n; i++) {
        printf("Enter AT and BT for process %d: ", i + 1);
        scanf("%d %d", &at[i], &bt[i]);
    }

    printf("Enter time quantum: ");
    scanf("%d", &quant);

    roundRobin(n, at, bt, quant);
    return 0;
}
```

```
Enter number of processes: 5
Enter AT and BT for process 1: 0 8
Enter AT and BT for process 2: 5 2
Enter AT and BT for process 3: 1 7
Enter AT and BT for process 4: 6 3
Enter AT and BT for process 5: 8 5
Enter time quantum: 3
P#      AT      BT      CT      TAT     WT
1       0       8       22      22      14
2       5       2       11      6       4
3       1       7       23      22      15
4       6       3       14      8       5
5       8       5       25      17      12
Average TAT: 15.00
Average WT: 10.00
```

```
        if(rem_bt[index]>0){
            queue[rear+1]=index;
        }
    if(front==rear){
        for(int i=0;i<n;++i){
            if(rem_bt[i]>0){
                queue[rear+1]=i;
                visited[i]=1;
                break;
            }
        }
    }
}

float total_tat=0,total_wt=0;
printf("P# \tBT\tCT\tTAT\tWT\n");
for(int i=0;i<n;++i){
    tat[i]=ct[i]-at[i];
    wt[i]=tat[i]-bt[i];
    total_tat+=tat[i];
    total_wt+=wt[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("Average TAT: %.2f\n",total_tat/n);
printf("Average WT: %.2f\n",total_wt/n);
}
int main(){
    int n,quant;
    printf("Ented no. of process");
    scanf("%d",&n);
    int at[n],bt[n];
```



Round Robin

```
#include<stdio.h>
void roundRobin(int n, int at[], int bt[], int q){
    int ct[n], wt[n], tat[n], rem_bt[n];
    int queue[1000], front=0, rear=0;
    int time=0, completed=0, visited[n];
    for(int i=0; i<n; ++i){
        rem_bt[i]=bt[i];
        visited[i]=0;
    }
    queue[rear++]=0;
    visited[0]=1;
    while(completed<n){
        int index=queue[front++];
        if(rem_bt[index]>quant){
            time+=quant;
            rem_bt[index]-=quant;
        }else{
            time+=rem_bt[index];
            rem_bt[index]=0;
            ct[index]=time;
            completed++;
        }
    for(int i=0; i<n; ++i){
        if(at[i]<=time && rem_bt[i]>0 &&
                !visited[i]){
            queue[rear++]=i;
            visited[i]=1;
        }
    }
}
```

```
for(int i=0; i<n; i++){
    printf("Enter AT & BT for Process %d: ", i+1);
    scanf("%d %d", &at[i], &bt[i]);
}
printf("Enter time quantum: ");
scanf("%d", &quant);
roundRobin(n, at, bt, quant);
return 0;
}
```

OP

Enter number of process: 5
Enter AT and BT for Process
0 8
5 2
1 7
6 3
8 5

Enter time Quantum: 3

PID	AT	BT	CT	TAT	WT
1	0	8	22	22	14
2	5	2	11	6	4
3	1	7	23	22	15
4	6	3	14	8	5
5	8	5	25	17	12

Average TAT: 15.00
Average WT: 10.00

Program 3:

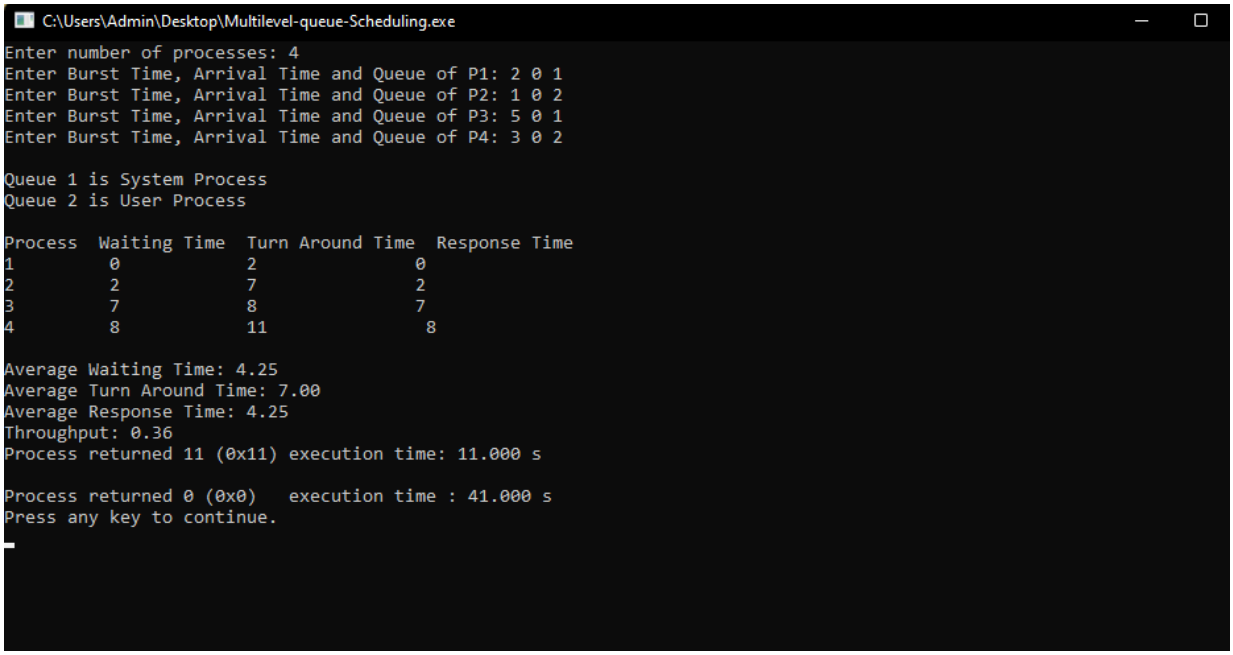
Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for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

```
#include <stdio.h>


#define MAX_PROCESSES 10
#define TIME_QUANTUM 2


typedef struct {
   int burst_time, arrival_time, queue_type, waiting_time, turnaround_time, response_time, remaining_time;
} Process;


void round_robin(Process processes[], int n, int time_quantum, int *time) {
   int done, i;
   do {
      done = 1;
      for (i = 0; i < n; i++) {
         if (processes[i].remaining_time > 0) {
            done = 0;
            if (processes[i].remaining_time > time_quantum) {
               *time += time_quantum;
               processes[i].remaining_time -= time_quantum;
            } else {
               *time += processes[i].remaining_time;
               processes[i].waiting_time = *time - processes[i].arrival_time - processes[i].burst_time;
               processes[i].turnaround_time = *time - processes[i].arrival_time;
               processes[i].response_time = processes[i].waiting_time;
               processes[i].remaining_time = 0;
```

```
            }
        }
    }
  } while (!done);
}

void fcfs(Process processes[], int n, int *time) {
  for (int i = 0; i < n; i++) {
    if (*time < processes[i].arrival_time) {
      *time = processes[i].arrival_time;
    }
    processes[i].waiting_time = *time - processes[i].arrival_time;
    processes[i].turnaround_time = processes[i].waiting_time + processes[i].burst_time;
    processes[i].response_time = processes[i].waiting_time;
    *time += processes[i].burst_time;
  }
}

int main() {
  Process processes[MAX_PROCESSES], system_queue[MAX_PROCESSES],
user_queue[MAX_PROCESSES];
  int n, sys_count = 0, user_count = 0, time = 0;
  float avg_waiting = 0, avg_turnaround = 0, avg_response = 0, throughput;

  printf("Enter number of processes: ");
  scanf("%d", &n);

  for (int i = 0; i < n; i++) {
    printf("Enter Burst Time, Arrival Time and Queue of P%d: ", i + 1);
```

```
        scanf("%d %d %d", &processes[i].burst_time, &processes[i].arrival_time,
&processes[i].queue_type);
        processes[i].remaining_time = processes[i].burst_time;

        if (processes[i].queue_type == 1) {
            system_queue[sys_count++] = processes[i];
        } else {
            user_queue[user_count++] = processes[i];
        }
    }

    // Sort user processes by arrival time for FCFS
    for (int i = 0; i < user_count - 1; i++) {
        for (int j = 0; j < user_count - i - 1; j++) {
            if (user_queue[j].arrival_time > user_queue[j + 1].arrival_time) {
                Process temp = user_queue[j];
                user_queue[j] = user_queue[j + 1];
                user_queue[j + 1] = temp;
            }
        }
    }

    printf("\nQueue 1 is System Process\nQueue 2 is User Process\n");
    round_robin(system_queue, sys_count, TIME_QUANTUM, &time);
    fcfs(user_queue, user_count, &time);

    printf("\nProcess  Waiting Time  Turn Around Time  Response Time\n");

    for (int i = 0; i < sys_count; i++) {
        avg_waiting += system_queue[i].waiting_time;
```

```
        avg_turnaround += system_queue[i].turnaround_time;
        avg_response += system_queue[i].response_time;
        printf("%d %d %d %d\n", i + 1, system_queue[i].waiting_time,
system_queue[i].turnaround_time, system_queue[i].response_time);
    }

    for (int i = 0; i < user_count; i++) {
        avg_waiting += user_queue[i].waiting_time;
        avg_turnaround += user_queue[i].turnaround_time;
        avg_response += user_queue[i].response_time;
        printf("%d %d %d %d\n", i + 1 + sys_count, user_queue[i].waiting_time,
user_queue[i].turnaround_time, user_queue[i].response_time);
    }

    avg_waiting /= n;
    avg_turnaround /= n;
    avg_response /= n;
    throughput = (float)n / time;

    printf("\nAverage Waiting Time: %.2f", avg_waiting);
    printf("\nAverage Turn Around Time: %.2f", avg_turnaround);
    printf("\nAverage Response Time: %.2f", avg_response);
    printf("\nThroughput: %.2f", throughput);
    printf("\nProcess returned %d (0x%d) execution time: %.3f s\n", time, time, (float)time);

    return 0;
}
```


C:\Users\Admin\Desktop\Multilevel-queue-Scheduling.exe
Enter number of processes: 4
Enter Burst Time, Arrival Time and Queue of P1: 2 0 1
Enter Burst Time, Arrival Time and Queue of P2: 1 0 2
Enter Burst Time, Arrival Time and Queue of P3: 5 0 1
Enter Burst Time, Arrival Time and Queue of P4: 3 0 2
Queue 1 is System Process
Queue 2 is User Process
Process Waiting Time Turn Around Time Response Time
1 0 2 0
2 2 7 2
3 7 8 7
4 8 11 8
Average Waiting Time: 4.25
Average Turn Around Time: 7.00
Average Response Time: 4.25
Throughput: 0.36
Process returned 11 (0x11) execution time: 11.000 s
Process returned 0 (0x0) execution time : 41.000 s
Press any key to continue.

```
void fcfs(process process[], int n, int *time){
    for(int i=0; i<n; ++i){
    if(*time < process[i].arrival_time){
     *time = process[i].arrival_time;}
process[i].wt = *time - process[i].at;
process[i].tat = process[i].wt + process[i].bt;
process[i].st = process[i].wt;
*time += process[i].bt;
    }
}
int main(){
  process process[max_Process], system_queue
   [MAX_Processes], user_queue[MAX_PROCESSES]
  float avg_waiting=0; avg_turnaround=0,
    avg_response=0, throughput;
printf("Enter no. of process: ");
  scanf("%d", &n);
for(int i=0; i<n; ++i){
  printf("Enter Burst Time, Arrival Time & Queue of
      P%d:", i+1);
  scanf("%d %d %d", &process[i].burst_time,
    &process[i].arrival_time, &process[i].queue_
      type);
process[i].remaining_time = process[i].burst_time;
if(process[i].queue_type == 1)
  system_queue[sys_count++] = process[i];
else
  user_queue[user_count++] = process[i];
for(int i=0; i<user_count-1; ++i){
    for(int j=0; j<user_count-i-1; ++j){
    if(user_queue[j].at > user_queue[j+1].at){
        process temp = user_queue[j];
        user_queue[j] = user_queue[j+1];
```

Date __/__/__
Page _____

→ Multilevel Queue Scheduling.

```
#include<stdio.h>
#define MAX_PROCESS 10
#define TIME_QUANTUM 2
typedef struct {
    int burst_time, arrival_time, queue_type, waiting_time,
    turnaround_time, response_time, remaining_time; } process;
void round_robin(Process process[], int n, int time_
    quantum, int * time){
        int done, i;
        do {
            done = 1;
            for(int i=0; i<n; ++i) {
                if(process[i].remaining_time > 0) {
                    done = 0;
                if(process[i].remaining_time > time_quantum){
                    *time += time_quantum;
                process[i].remaining_time -= time_quantum;}
                else {
                    *time += process[i].remaining_time;
                    process[i].wt = *time - process[i].at -
                                    process[i].bt;
                    process[i].tat = *time - process[i].at;
                    process[i].response_time = process[i].wt;
                    process[i].remaining_time = 0;
                }
            }
        }
    } while(!done);
}
```

```
        user_queue[j].temp;
    }
  }
}
printf("\n Queue 1 is System Process \n Queue 2 is
        User Process\n");
roundrobin(system_queue, sys_count, time_Quantum,
        &time);
fcfs(user_queue, user_count, &time);
printf("\n Process  Waiting Time  Turn Around Time
        Response Time \n");
for(int i=0 ; i< sys_count ; i++){
    avg_waiting += system_queue[i].wt;
    avg_turnaround += system_queue[i].turnaround_time;
    avg_response += system_queue[i].response_time;
printf("%d  %d  %d  %d \n", i+1, system_queue[i].
    waiting_time, system_queue[i].turnaround_time,
    system_queue[i].response_time);
}
for(int i=0; i< user_count; i++){
    avg_wt += user_queue[i].wt;
    avg_turnaround_time += user_queue[i].turnaround_time;
    avg_response += user_queue[i].response_time;
printf("%d  %d  %d  %d\n", i+1+sys_count,
    user_queue[i].wt, user_queue[i].tat, user_queue
    [i].response_time);
}
avg_waiting /= n;
avg_turnaround /= n;
avg_response /= n;
throughput = (float) n/time;
printf("\n Average wt: %.2f", avg_waiting);
printf("\n Average Turnaround time: %.2f ",
                                avg_turnaround);
```

O/P

Enter process: 4

Enter BT, AT & Queue:

P1: 2 0 1

P2: 1 0 2

P3: 5 0 1

P4: 3 0 2

Process	WT	TAT	RT
P1	0	2	0
P3	2	7	2
P2	7	8	7
P4	8	11	8

Avg WT: 4.25

Avg TAT: 7

Avg RT: 4.25

Throughput: 0.57

int id, word, place,

} stack;

Program 4:

Write a C program to simulate Real-Time CPU Scheduling algorithms:
a) Rate- Monotonic
b) Earliest-deadline First

→Rate- Monotonic

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

typedef struct {
    int id, burst, period;
} Task;

int gcd(int a, int b) {
    return (b == 0) ? a : gcd(b, a % b);
}

int lcm(int a, int b) {
    return (a * b) / gcd(a, b);
}

int findLCM(Task tasks[], int n) {
    int result = tasks[0].period;
    for (int i = 1; i < n; i++)
        result = lcm(result, tasks[i].period);
    return result;
}

void rateMonotonic(Task tasks[], int n) {
    float utilization = 0;
```

```
    printf("\nRate Monotonic Scheduling:\nPID\tBurst\tPeriod\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\n", tasks[i].id, tasks[i].burst, tasks[i].period);
        utilization += (float)tasks[i].burst / tasks[i].period;
    }

    float bound = n * (pow(2, (1.0 / n)) - 1);
    printf("\nUtilization: %.6f, Bound: %.6f\n", utilization, bound);
    if (utilization <= bound)
        printf("Tasks are Schedulable\n");
    else
        printf("Tasks are NOT Schedulable\n");
}

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

    Task tasks[n];
    printf("Enter the CPU burst times: ");
    for (int i = 0; i < n; i++)
        scanf("%d", &tasks[i].burst);

    printf("Enter the time periods: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &tasks[i].period);
        tasks[i].id = i + 1;
    }
```

```
    rateMonotonic(tasks, n);
    return 0;
}
```

Output

```
Enter the number of processes: 3
Enter the CPU burst times: 3 6 8
Enter the time periods: 3 4 5

Rate Monotonic Scheduling:
PID Burst   Period
1   3   3
2   6   4
3   8   5

Utilization: 4.100000, Bound: 0.779763
Tasks are NOT Schedulable
```

Rate-Monotonic

```
#include<stdio.h>
#include<math.h>
  typedef struct {
     int id, burst, period;
  }Task;
int gcd(int a, int b){
   return (b==0)? a: gcd(b, a%b);}
int lcm(int a, int b){
   return (a*b)/gcd(a,b);
     }
int findLcm(Task task[], int n){
   int result = task[0].period;
   for(int i=1; i<n; ++i)
     result = lcm(result, task[i].period);
     return result;
   }
int findLCM(Task task[], int n){
    int result = task[0].period;
   for(int i=1; i<n; ++i)
       result = lcm(result, task[i].period);
   return result;
}
```

```
void rateMonotonic(Task taskArr[], int n){
    float utilization=0;
    printf("\n Rate Monotonic Scheduling : \n PID\t
      Burst\t period \n");
    for(int i=0; i<n; ++i)
      printf("%d \t %d \t %d \t\t %d\n", taskArr[i].id,
    taskArr[i].burst, taskArr[i].period);
    utilization += (float) taskArr[i].burst / taskArr[i].period;
    }
float bound = n* (pow(2,(1.0/n))-1);
printf("\n utilization : %d. 6f, Bound: %. 6f \n"
  utilization bound);
if(utilization <= bound)
      printf(" taskp are schedulable \n");
  else
    Printf(" Taskp are Not Schedulable \n");
  }
int main(){
  int n;
  printf("Enter the no. of processes : ");
  scanf("%d", &n);
  Task taskp[n];
  printf(" Enter the CPU Burst Times : ");
  for(int i=0; i<n; ++i){
    scanf("%d", &taskp[i].period);
    taskp[i]. id = i+1;
      }
  rateMonotonic(taskp, n);
    return 0;
}
```

QP

Enter the no. of processes: 3
Enter the cpu bt: 3 6 8
Enter time periods: 3 4 5
LCM = 60
RMS:

PID	Bt	period
1	3	3
2	6	4
3	8	5

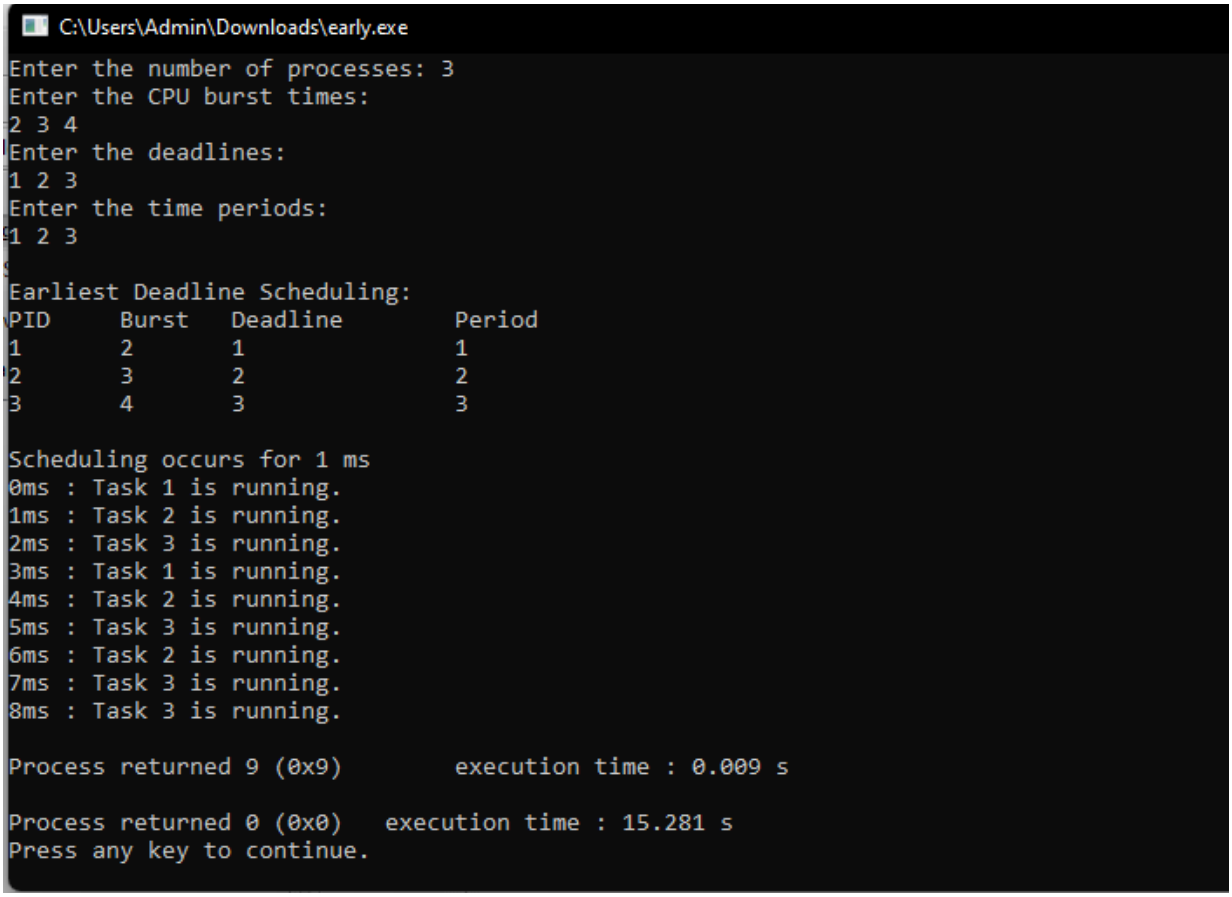
4.100000 <= 0.779763 ⇒ False
System may not be Schedulable.

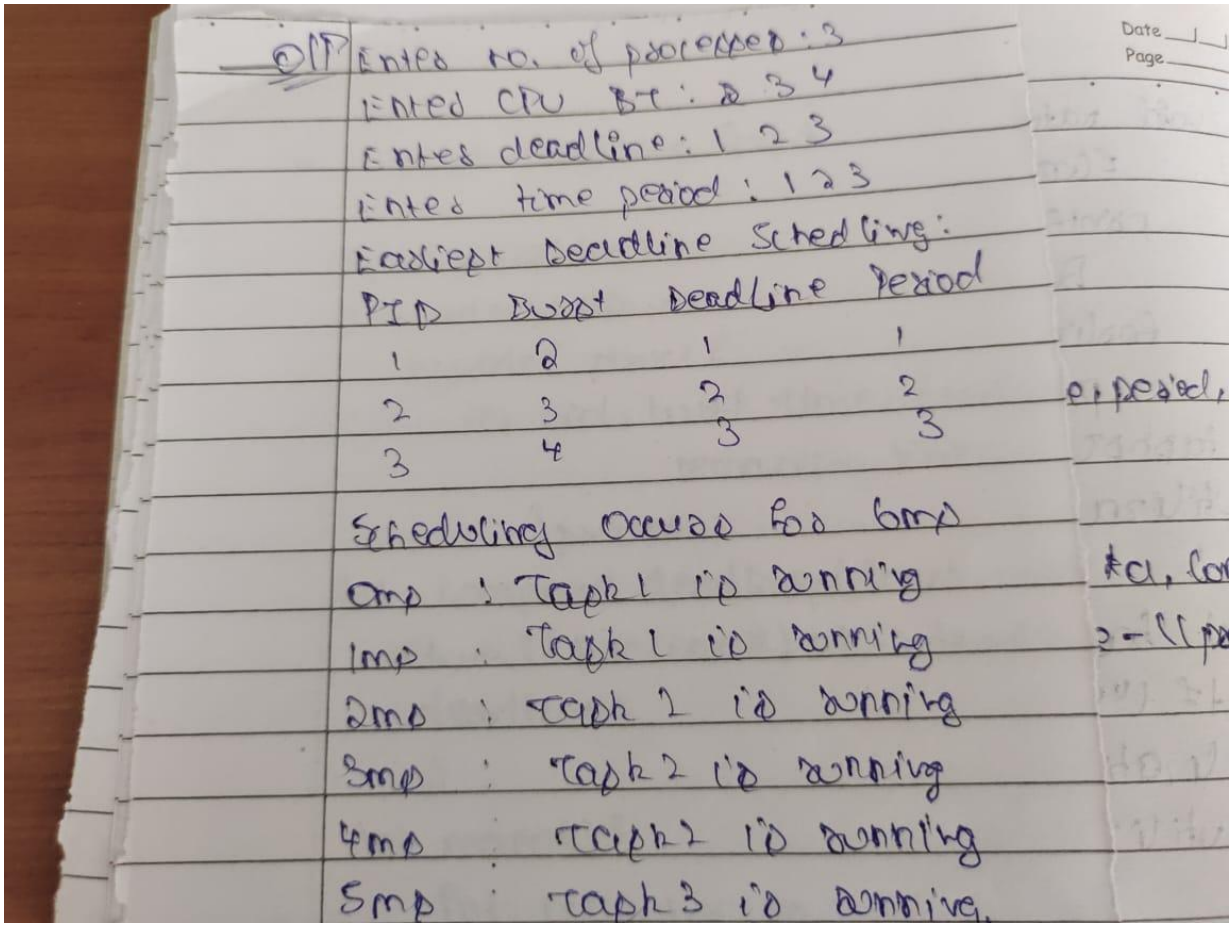
→Earliast-Deadline First

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

typedef struct {
    int pid;
    int burst_time;
    int deadline;
    int period;
    int remaining_time;
} Process;

// Function to compare processes based on their deadlines
int compare_deadlines(const void *a, const void *b) {
    return ((Process *)a)->deadline - ((Process *)b)->deadline;
}

int main() {
    int num_processes;

    printf("Enter the number of processes: ");
    scanf("%d", &num_processes);

    Process processes[num_processes];

    printf("Enter the CPU burst times:\n");
    for (int i = 0; i < num_processes; i++) {
        scanf("%d", &processes[i].burst_time);
        processes[i].pid = i + 1;
        processes[i].remaining_time = processes[i].burst_time;
```

```
    }

    printf("Enter the deadlines:\n");
    for (int i = 0; i < num_processes; i++) {
        scanf("%d", &processes[i].deadline);
    }

    printf("Enter the time periods:\n");
    for (int i = 0; i < num_processes; i++) {
        scanf("%d", &processes[i].period);
    }

    // Sort processes based on deadlines
    qsort(processes, num_processes, sizeof(Process), compare_deadlines);

    printf("\nEarliest Deadline Scheduling:\n");
    printf("PID\tBurst\tDeadline\tPeriod\n");
    for (int i = 0; i < num_processes; i++) {
        printf("%d\t%d\t%d\t\t%d\n", processes[i].pid, processes[i].burst_time, processes[i].deadline,
processes[i].period);
    }

    int current_time = 0;
    int completed_processes = 0;

    printf("\nScheduling occurs for %d ms\n", processes[0].deadline); // Assuming the first process has
the earliest deadline

    while (completed_processes < num_processes) {
        for (int i = 0; i < num_processes; i++) {
```

```
            if (processes[i].remaining_time > 0) {
                printf("%dms : Task %d is running.\n", current_time, processes[i].pid);
                processes[i].remaining_time--;
                current_time++;

                if (processes[i].remaining_time == 0) {
                    completed_processes++;
                }
            }
        }
    }

    printf("\nProcess returned %d (0x%X)\texecution time : %.3f s\n", current_time, current_time,
(float)current_time / 1000.0);


    return 0;
}
```


C:\Users\Admin\Downloads\early.exe
Enter the number of processes: 3
Enter the CPU burst times:
2 3 4
Enter the deadlines:
1 2 3
Enter the time periods:
1 2 3
Earliest Deadline Scheduling:
PID Burst Deadline Period
1 2 1 1
2 3 2 2
3 4 3 3
Scheduling occurs for 1 ms
0ms : Task 1 is running.
1ms : Task 2 is running.
2ms : Task 3 is running.
3ms : Task 1 is running.
4ms : Task 2 is running.
5ms : Task 3 is running.
6ms : Task 2 is running.
7ms : Task 3 is running.
8ms : Task 3 is running.
Process returned 9 (0x9) execution time : 0.009 s
Process returned 0 (0x0) execution time : 15.281 s
Press any key to continue.



O/P
Enter no. of processes: 3
Enter CPU BT: 2 3 4
Enter deadline: 1 2 3
Enter time period: 1 2 3
Earliest Deadline Scheduling:

PID	Burst	Deadline	Period
1	2	1	1
2	3	2	2
3	4	3	3

Scheduling occurs for 6ms
0ms : Task 1 is running
1ms : Task 1 is running
2ms : Task 2 is running
3ms : Task 2 is running
4ms : Task 2 is running
5ms : Task 3 is running.

Earliest deadline first.

```
#include<stdio.h>
#include<stdlib.h>
    typedef struct {
        int pid, burst_time, deadline, period,
            remaining_time;
        } process;
int compare_deadlines(const void *a, const void *b)
    return ((process *)a)->deadline - ((process *)b)->
    deadline;
    }
int main(){
    int num_process;
    printf("Enter the no of process: ");
    scanf("%d", &num_process);
process Processes[num_process];
    printf("Enter the CPU burst time: \n");
    for(int i=0; i<num_process; ++i)
        scanf("%d", &process[i].burst_time);
    process[i].pid=i+1;
    process[i].remaining_time = process[i].burst_time
    }
    printf("Enter the deadline: \n");
    for(int i=0; i<num_process; ++i) {
        scanf("%d", &process[i].deadline);
        }
    printf("Enter the time period: \n");
    for(int i=0; i<num_process; ++i){
        scanf("%d", &process[i].period);
qsort(process, num_process, sizeof(process),
    compare_deadlines);
```

```
printf("\n Earliest Deadline scheduling: \n");
printf("PID\t Burst\t Deadline\t period\n");
for(int i=0; i<num_processes; ++i){
    printf("%d\t %d\t %d\t %d\n",
    processes[i].pid, process[i].burst_time,
process[i].deadline, process[i].period);
}
int current_time=0;
int completed_processes=0;
    printf("\n scheduling occurs for %d ms\n",
        processes[0].deadline);
while(completed_processes<num_proc){
for(int i=0; i<num_processes; ++i)
  if(processes[i].remaining_time>0){
printf("%d ms: task %d is running.\n",
current_time, processes[i].pid);
process[i].remaining_time--;
current_time++;
  if(process[i].remaining_time==0){
      completed_processes++;
    }
  }
}
printf("\n process returned %d (0x%x)\t
    execution time %f sin", current_time,
current_time, current_time, (float) current_time/
1000.0);
  return 0;
}
```

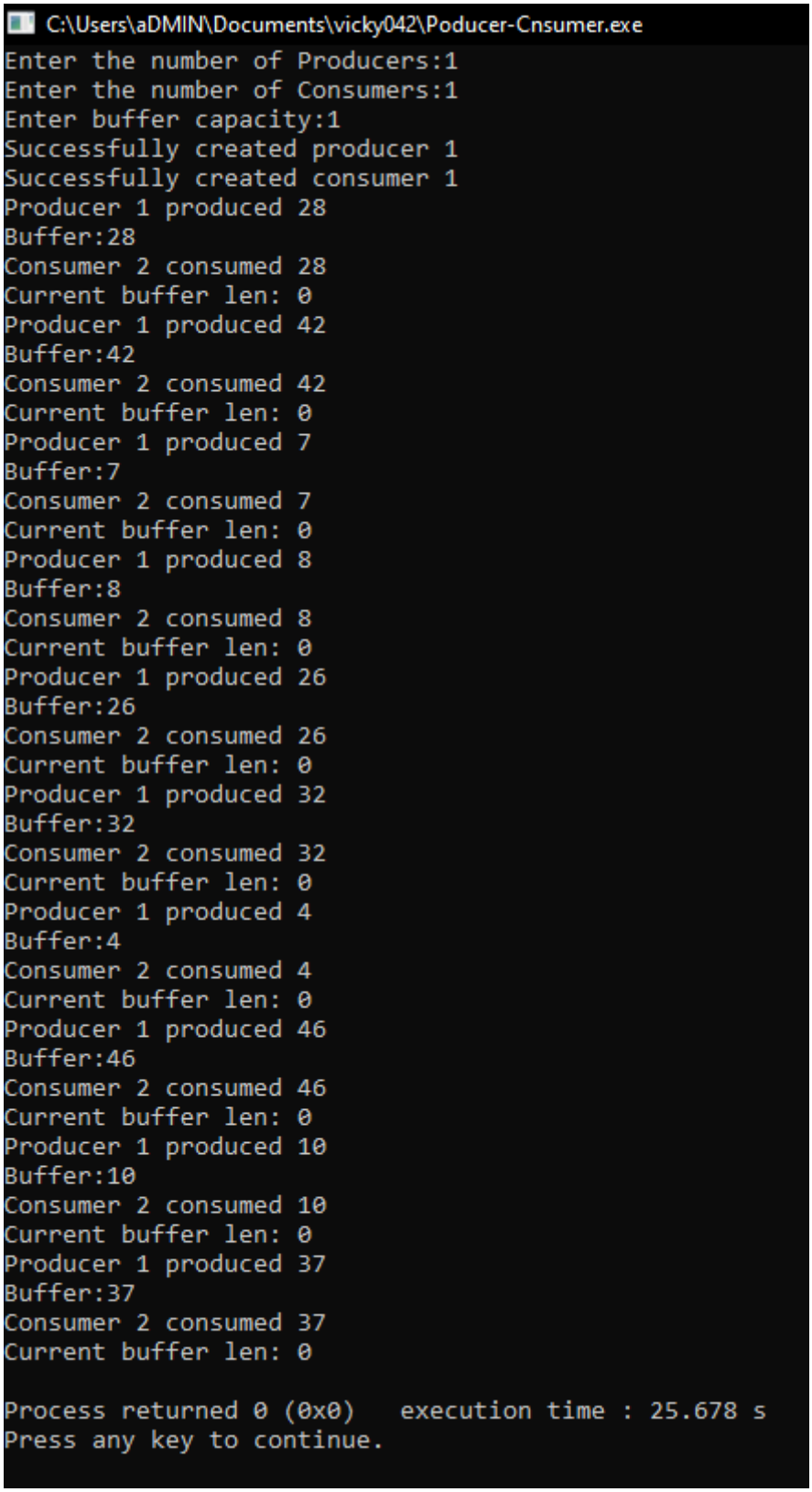
Program 5:

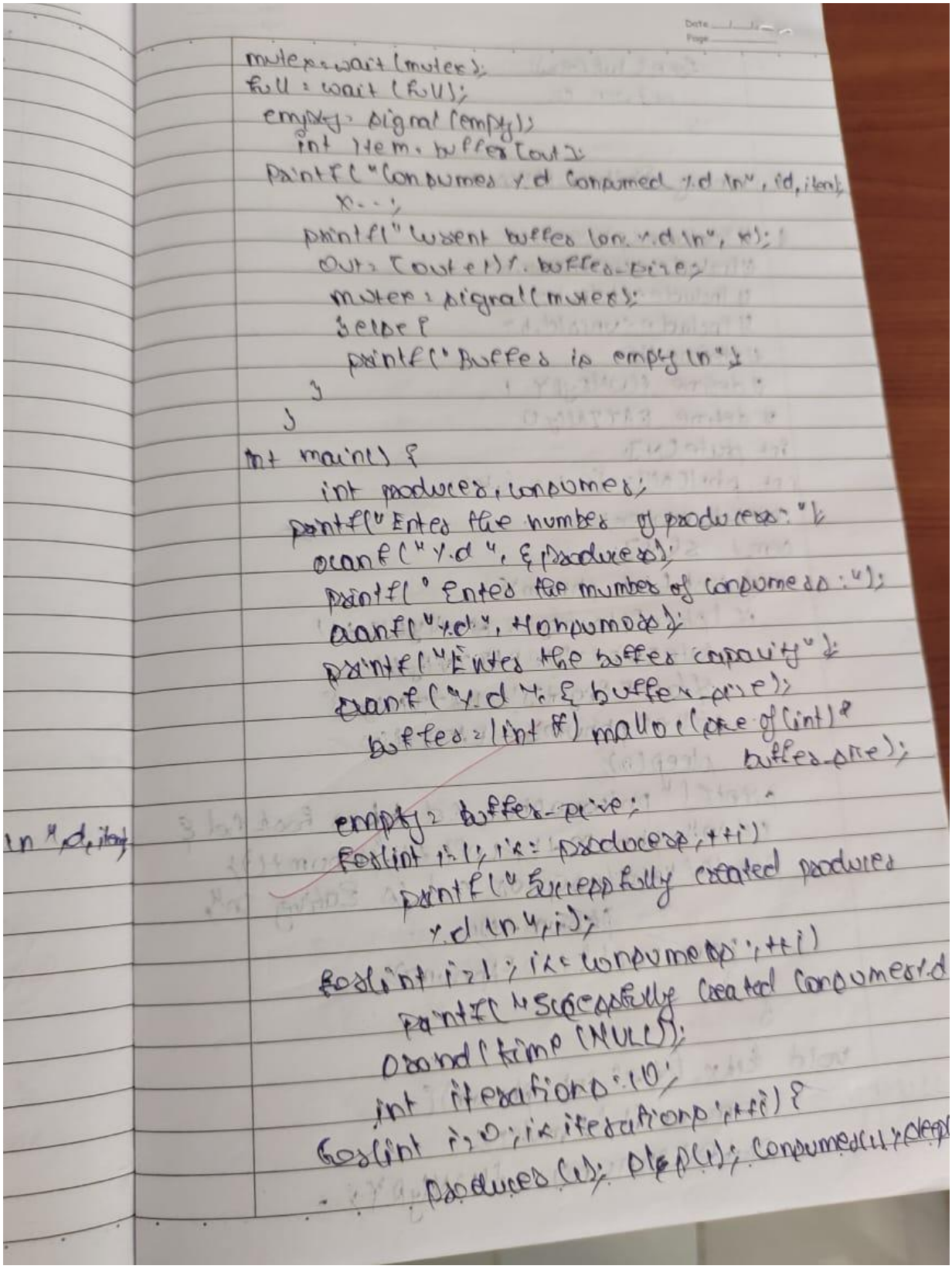
Write a C program to simulate producer-consumer problem using semaphores

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

int mutex = 1, full = 0, empty, x = 0;
int *buffer, buffer_size;
int in = 0, out = 0;

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

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

void producer(int id) {
   if ((mutex == 1) && (empty != 0)) {
      mutex = wait(mutex);
      full = signal(full);
      empty = wait(empty);
      int item = rand() % 50;
      buffer[in] = item;
      x++;
      printf("Producer %d produced %d\n", id, item);
      printf("Buffer:%d\n", item);
      in = (in + 1) % buffer_size;
```

```
        mutex = signal(mutex);
    } else {
        printf("Buffer is full\n");
    }
}

void consumer(int id) {
    if ((mutex == 1) && (full != 0)) {
        mutex = wait(mutex);
        full = wait(full);
        empty = signal(empty);
        int item = buffer[out];
        printf("Consumer %d consumed %d\n", id, item);
        x--;
        printf("Current buffer len: %d\n", x);
        out = (out + 1) % buffer_size;
        mutex = signal(mutex);
    } else {
        printf("Buffer is empty\n");
    }
}

int main() {
    int producers, consumers;
    printf("Enter the number of Producers:");
    scanf("%d", &producers);
    printf("Enter the number of Consumers:");
    scanf("%d", &consumers);
    printf("Enter buffer capacity:");
    scanf("%d", &buffer_size);
```

```
    buffer = (int *)malloc(sizeof(int) * buffer_size);
    empty = buffer_size;

    for (int i = 1; i <= producers; i++)
        printf("Successfully created producer %d\n", i);
    for (int i = 1; i <= consumers; i++)
        printf("Successfully created consumer %d\n", i);

    srand(time(NULL));

    int iterations = 10;
    for (int i = 0; i < iterations; i++) {
        producer(1);
        sleep(1);
        consumer(2);
        sleep(1);
    }

    free(buffer);
    return 0;
}
```

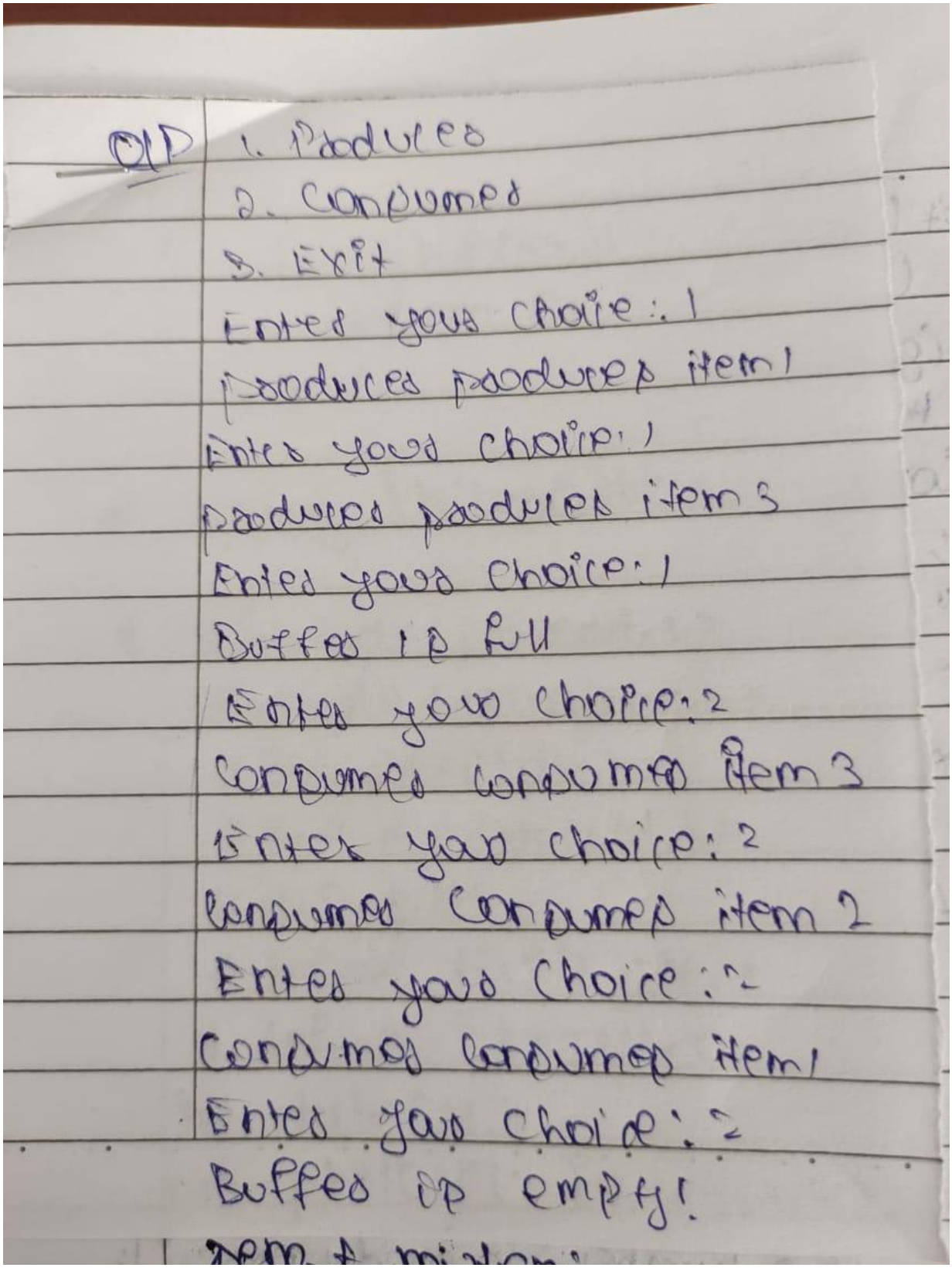

C:\Users\aDMIN\Documents\vicky042\Poducer-Cnsumer.exe
Enter the number of Producers:1
Enter the number of Consumers:1
Enter buffer capacity:1
Successfully created producer 1
Successfully created consumer 1
Producer 1 produced 28
Buffer:28
Consumer 2 consumed 28
Current buffer len: 0
Producer 1 produced 42
Buffer:42
Consumer 2 consumed 42
Current buffer len: 0
Producer 1 produced 7
Buffer:7
Consumer 2 consumed 7
Current buffer len: 0
Producer 1 produced 8
Buffer:8
Consumer 2 consumed 8
Current buffer len: 0
Producer 1 produced 26
Buffer:26
Consumer 2 consumed 26
Current buffer len: 0
Producer 1 produced 32
Buffer:32
Consumer 2 consumed 32
Current buffer len: 0
Producer 1 produced 4
Buffer:4
Consumer 2 consumed 4
Current buffer len: 0
Producer 1 produced 46
Buffer:46
Consumer 2 consumed 46
Current buffer len: 0
Producer 1 produced 10
Buffer:10
Consumer 2 consumed 10
Current buffer len: 0
Producer 1 produced 37
Buffer:37
Consumer 2 consumed 37
Current buffer len: 0
Process returned 0 (0x0) execution time : 25.678 s
Press any key to continue.



```
mutex = wait(mutex);
full = wait(full);
empty = signal(empty);
int item = buffer[out];
printf("Consumes %d Consumed %d \n", id, item);
    x--;
printf("\nSent buffer loc. %d \n", x);
out = (out+1) % buffer_size;
mutex = signal(mutex);
}else{
printf("Buffer is empty \n");
}
}
int main(){
int producer, consumer;
printf("Enter the number of producers: ");
scanf("%d", &producer);
printf("Enter the number of consumers: ");
scanf("%d", &noconsumers);
printf("Enter the buffer capacity");
scanf("%d", &buffer_size);
buffer = (int *) malloc(size of(int) *
                        buffer_size);

empty = buffer_size;
for(int i=1; i<=producer; ++i)
printf("Successfully created producer
%d \n", i);
for(int i=1; i<=consumer; ++i)
printf("Successfully created consumer %d
srand(time(NULL));
int iterations=10;
for(int i=0; i<iterations; ++i){
produce(i); sleep(1); consume(i); sleep
```

Producer-Consumer

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<time.h>
int mutex=1, full=0, empty=x=0,
int *buffer, buffer_size;
int in=0, out=0;
int wait(int s){
    return (--s);
}
int signal(int s){
    return ++s;
}
void producer(int id){
    if((mutex==1) && (empty!=0)){
        mutex=wait(mutex);
        full=signal(full);
        empty=wait(empty);
        int item=void()%50;
        buffer[in]=item;
        x++;
    printf("producer %d produced %d in %d",
    printf("Buffer: %d\n", item);
        in=(in+1)%buffer_size;
        mutex=signal(mutex);
    }else{
    printf("Buffer is full\n");
    }
}
void consumer(int id){
    if(mutex==1 && full!=0){
```


O/P
1. Produces
2. Consumes
3. Exit
Enter your choice: 1
Produces produces item 1
Enter your choice: 1
produces produces item 3
Enter your choice: 1
Buffer is full
Enter your choice: 2
Consumes consumes item 3
Enter your choice: 2
Consumes Consumes item 2
Enter your choice: 2
Consumes consumes item 1
Enter your choice: 2
Buffer is empty!
Item is mixed

Program 6:

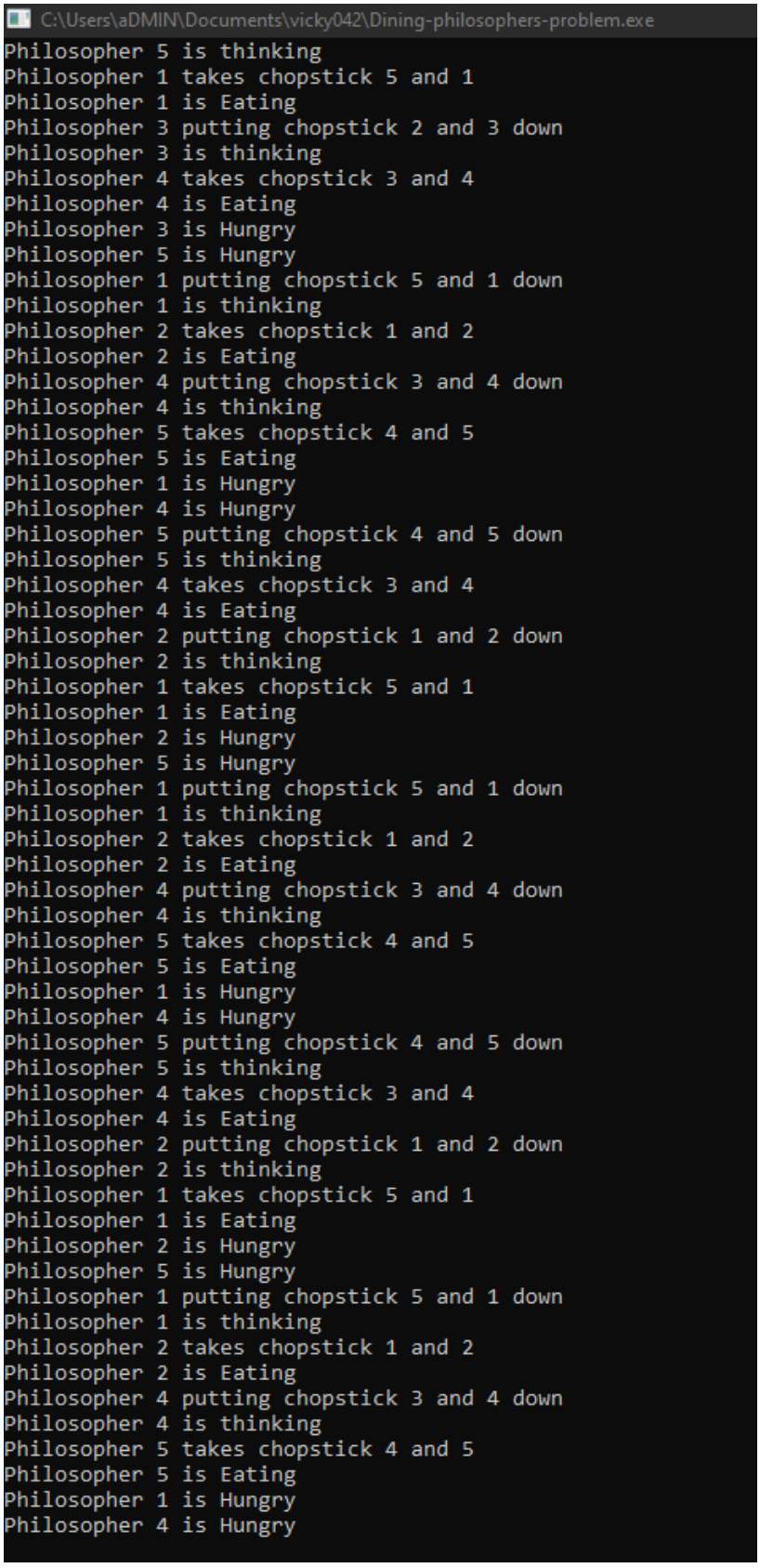
Write a C program to simulate the concept of Dining Philosophers problem.

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

#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (phnum + 4) % N
#define RIGHT (phnum + 1) % N

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

sem_t mutex;
sem_t S[N];

void test(int phnum)
{
   if (state[phnum] == HUNGRY &&
       state[LEFT] != EATING &&
       state[RIGHT] != EATING) {
       // Eating state
       state[phnum] = EATING;

       printf("Philosopher %d takes chopstick %d and %d\n", phnum + 1, LEFT + 1, phnum + 1);
```

```
        printf("Philosopher %d is Eating\n", phnum + 1);

        sem_post(&S[phnum]);
    }
}

void take_fork(int phnum)
{
    sem_wait(&mutex);

    state[phnum] = HUNGRY;
    printf("Philosopher %d is Hungry\n", phnum + 1);

    test(phnum);

    sem_post(&mutex);

    sem_wait(&S[phnum]);
    sleep(1);
}

void put_fork(int phnum)
{
    sem_wait(&mutex);

    state[phnum] = THINKING;
    printf("Philosopher %d putting chopstick %d and %d down\n", phnum + 1, LEFT + 1, phnum + 1);
    printf("Philosopher %d is thinking\n", phnum + 1);

    test(LEFT);
```

```
    test(RIGHT);

    sem_post(&mutex);
}

void* philosopher(void* num)
{
    while (1) {
        int* i = num;

        sleep(1);
        take_fork(*i);
        sleep(1);
        put_fork(*i);
    }
}

int main()
{
    int i;
    pthread_t thread_id[N];

    sem_init(&mutex, 0, 1);

    for (i = 0; i < N; i++)
        sem_init(&S[i], 0, 0);

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

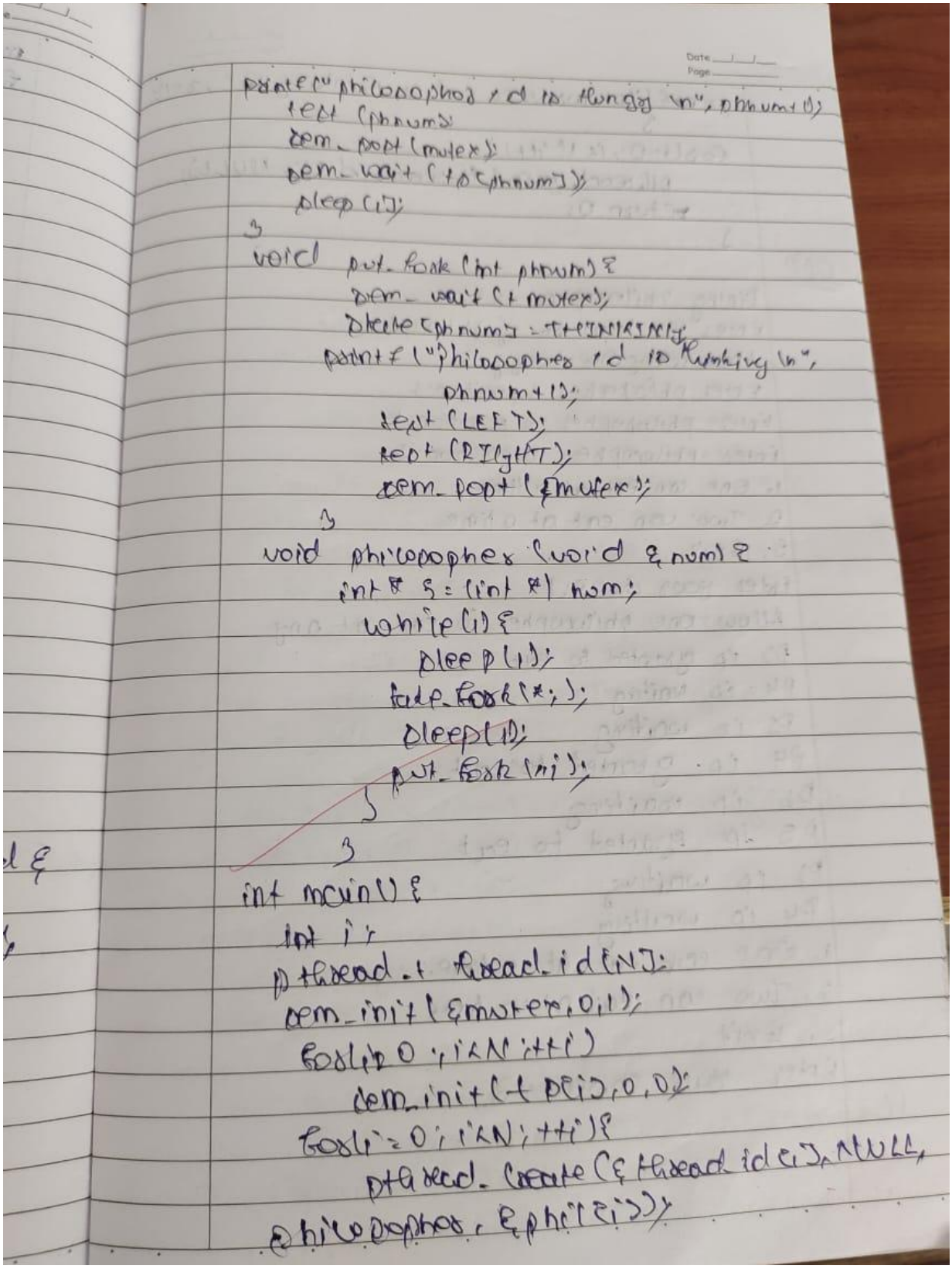
```
    }

    for (i = 0; i < N; i++)
        pthread_join(thread_id[i], NULL);

    return 0;
}
```


C:\Users\aDMIN\Documents\vicky042\Dining-philosophers-problem.exe
Philosopher 5 is thinking
Philosopher 1 takes chopstick 5 and 1
Philosopher 1 is Eating
Philosopher 3 putting chopstick 2 and 3 down
Philosopher 3 is thinking
Philosopher 4 takes chopstick 3 and 4
Philosopher 4 is Eating
Philosopher 3 is Hungry
Philosopher 5 is Hungry
Philosopher 1 putting chopstick 5 and 1 down
Philosopher 1 is thinking
Philosopher 2 takes chopstick 1 and 2
Philosopher 2 is Eating
Philosopher 4 putting chopstick 3 and 4 down
Philosopher 4 is thinking
Philosopher 5 takes chopstick 4 and 5
Philosopher 5 is Eating
Philosopher 1 is Hungry
Philosopher 4 is Hungry
Philosopher 5 putting chopstick 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes chopstick 3 and 4
Philosopher 4 is Eating
Philosopher 2 putting chopstick 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes chopstick 5 and 1
Philosopher 1 is Eating
Philosopher 2 is Hungry
Philosopher 5 is Hungry
Philosopher 1 putting chopstick 5 and 1 down
Philosopher 1 is thinking
Philosopher 2 takes chopstick 1 and 2
Philosopher 2 is Eating
Philosopher 4 putting chopstick 3 and 4 down
Philosopher 4 is thinking
Philosopher 5 takes chopstick 4 and 5
Philosopher 5 is Eating
Philosopher 1 is Hungry
Philosopher 4 is Hungry
Philosopher 5 putting chopstick 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes chopstick 3 and 4
Philosopher 4 is Eating
Philosopher 2 putting chopstick 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes chopstick 5 and 1
Philosopher 1 is Eating
Philosopher 2 is Hungry
Philosopher 5 is Hungry
Philosopher 1 putting chopstick 5 and 1 down
Philosopher 1 is thinking
Philosopher 2 takes chopstick 1 and 2
Philosopher 2 is Eating
Philosopher 4 putting chopstick 3 and 4 down
Philosopher 4 is thinking
Philosopher 5 takes chopstick 4 and 5
Philosopher 5 is Eating
Philosopher 1 is Hungry
Philosopher 4 is Hungry

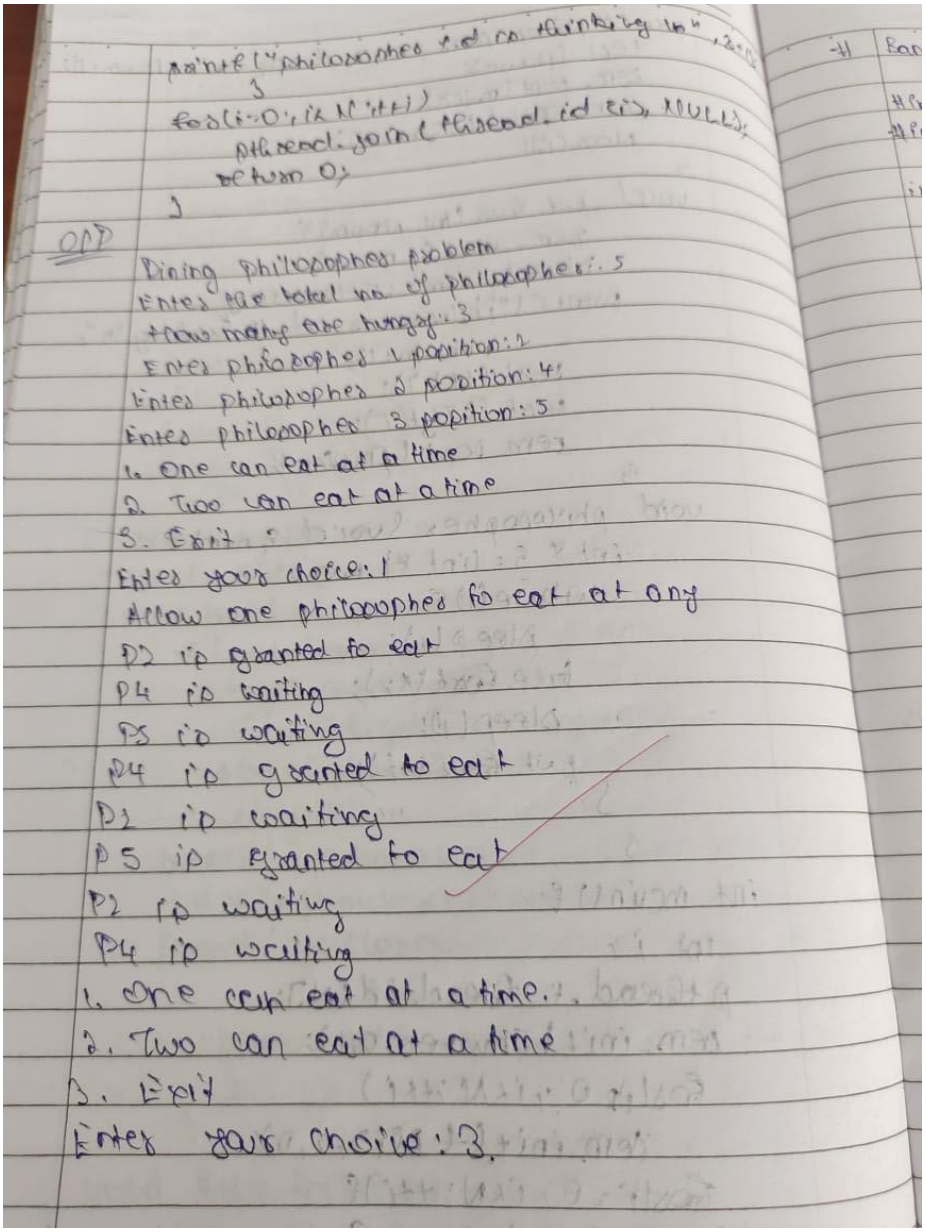
free buffer
return 0;
}

Dining Philosophers.c

```
#include<pthread.h>
#include<semaphore.h>
#include<stdio.h>
#include<unistd.h>
#define NS
#define HUNGRY 1
#define EATING 0
int state[N];
int phil[N]={0,1,2,3,4};
sem_t mutex;
sem_t S[N];
void test(int phnum){
  if(state[phnum]==HUNGRY){
    state[LEFT]=EATING;
    state[Right]=EATING;
    state[phnum]=EATING;
    sleep(2);
  printf("philosophers %d taken fork %d &
  %d\n", phnum+1, left+1, phnum+1);
  printf("philosophor %d is Eating\n",
          phnum+1);
    sem_post(&S[phnum]);
  }
}
void take_fork(int phnum){
  sem_wait(&mutex);
  state[phnum]=HUNGRY;
```



```
printf("philosopher %d is hungry\n", phnum+1);
test(phnum);
sem_post(mutex);
sem_wait(&s[phnum]);
sleep(1);
}
void put_fork(int phnum){
    sem_wait(&mutex);
    state[phnum] = THINKING;
    printf("philosopher %d is thinking\n",
            phnum+1);
    test(LEFT);
    test(RIGHT);
    sem_post(&mutex);
}
void philosopher(void *num){
    int *i = (int *) num;
    while(1){
        sleep(1);
        take_fork(*i);
        sleep(1);
        put_fork(*i);
    }
}
int main(){
    int i;
    pthread_t thread_id[N];
    sem_init(&mutex,0,1);
    for(i=0;i<N;i++)
        sem_init(&s[i],0,0);
    for(i=0;i<N;i++){
        pthread_create(&thread_id[i],NULL,
philosopher,&phil[i]);
```



```
printf("philosophers %d is thinking\n", i+1);
}
for(i=0; i<N; i++)
    pthread_join(phiseod_id[i], NULL);
return 0;
}
```

OUTPUT

```
Dining philosophers problem
Enter the total no of philosophers: 5
How many are hungry: 3
Enter philosopher 1 position: 1
Enter philosopher 2 position: 4
Enter philosopher 3 position: 5
1. One can eat at a time
2. Two can eat at a time
3. Exit
Enter your choice: 1
Allow one philosophers to eat at any
P1 is granted to eat
P4 is waiting
P5 is waiting
P4 is granted to eat
P1 is waiting
P5 is granted to eat
P2 is waiting
P4 is waiting
1. One can eat at a time.
2. Two can eat at a time
3. Exit
Enter your choice: 3.
```

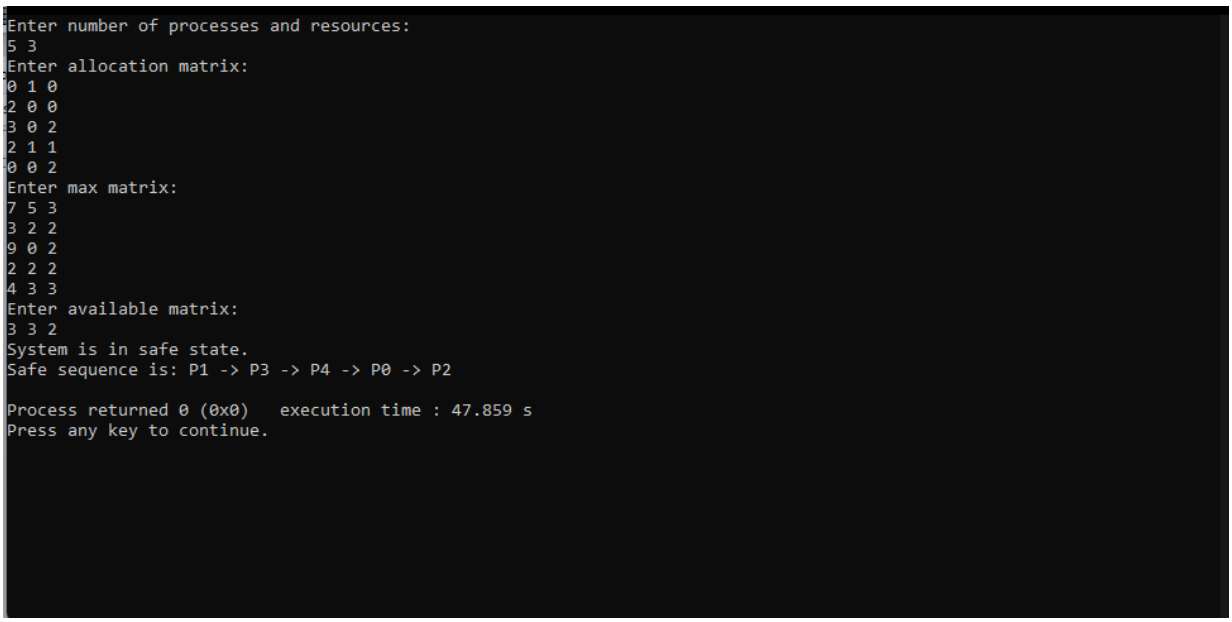
Program 7:

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

```
#include <stdio.h>

int main() {
    int n, m, i, j, k;
    printf("Enter number of processes and resources:\n");
    scanf("%d %d", &n, &m);

    int alloc[n][m], max[n][m], avail[m];
    int need[n][m], f[n], ans[n], ind = 0;

    printf("Enter allocation matrix:\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < m; j++)
            scanf("%d", &alloc[i][j]);

    printf("Enter max matrix:\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < m; j++)
            scanf("%d", &max[i][j]);

    printf("Enter available matrix:\n");
    for (i = 0; i < m; i++)
        scanf("%d", &avail[i]);

    for (i = 0; i < n; i++)
        f[i] = 0;
```

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

for (k = 0; k < n; k++) {
    for (i = 0; i < n; i++) {
        if (f[i] == 0) {
            int flag = 0;
            for (j = 0; j < m; j++) {
                if (need[i][j] > avail[j]) {
                    flag = 1;
                    break;
                }
            }
            if (flag == 0) {
                ans[ind++] = i;
                for (j = 0; j < m; j++)
                    avail[j] += alloc[i][j];
                f[i] = 1;
            }
        }
    }
}

int safe = 1;
for (i = 0; i < n; i++)
    if (f[i] == 0)
        safe = 0;

if (safe) {
```

```
        printf("System is in safe state.\nSafe sequence is: ");
        for (i = 0; i < n - 1; i++)
            printf("P%d -> ", ans[i]);
        printf("P%d\n", ans[n - 1]);
    } else {
        printf("System is not in safe state\n");
    }

    return 0;
}
```



// Bankers Algo

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

int condition(int * need, int * work, int i, int m) {
    for(int j=0; j<m; ++j)
        if( need[i][j] > work[j])
            return 0;
    return 1;
}
int safety(int n, int m, int * alloc, int ** max,
        int * available, int * sequence){
    int ** need = (int**) malloc(n * sizeof(int *));
    for(int i=0; i<n; ++i){
    need[i] = (int * ) malloc(m * sizeof(int));
    for(int j=0; j<m; ++j)
      need[i][j] = max[i][j] - all[i][j];
    }
int * work = (int*) malloc(m * sizeof(int));
    for(int i=0; i<m; ++i)
        work[i] = available[i];
int * finish = (int *) malloc(n * sizeof(int));
for(int i=0; i<n; ++i)
    finish[i]=0;
int safeIndex=0;
int changed;
do {
    changed=0;
    for(int i=0; i<n; ++i) {
if(!finish[i] && condition(need, work, i, m)){
for(int j=0; j<m; ++j)
    work[j] += all[i][j];
```

```
finish[i] = 1;
sequence[safeIndex++] = i;
changed = 1;
}
}
}while(changed);
for(int i=0; i<n; ++i){
    if(!finished[i])
        return 0;
}
return 1;
}
int main(){
int n=5, m=3;
printf("Enter Allocation matrix: \n");
int ** all = (int **) malloc(n * sizeof(int *));
for(int i=0; i<n; ++i){
allocated[i] = (int *) malloc(m * sizeof(int));
for(int j=0; j<m; ++j)
    scanf("%d", &all[i][j]);}
printf("Enter max matrix: \n");
int ** max = (int **) malloc(n * sizeof(int *));
for(int i=0; i<n; ++i){
max[i] = (int *) malloc(m * sizeof(int));
for(int j=0; j<m; ++j){
scanf("%d", &max[i][j]);}
}
printf("Enter available matrix: \n");
int *avai = (int *) malloc(m * sizeof(int));
for(int i=0; i<m; ++i){
    scanf("%d", &available[i]);
}
int * seq = (int *) malloc(n * sizeof(int));
```

```
int safe-safety(m,n,all,max,available,seq)
if(safe){
    printf("System is in a safe state\n Safe
sequence");
    for(int i=0;i<n;i++)
        printf("%d",sequence[i]);
printf("\n");
}
else{
    printf("System is not in a safe state\n");
}
return 0;
}
```

OUTPUT

Enter Alloc matrix:

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

Enter max matrix:

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

Enter Available matrix:

3 3 2

System is in safe state.

Safe Sequence: P1 -> P3 -> P4 -> P0 -> P2.

Program 8:

Write a C program to simulate deadlock detection

```
#include <stdio.h>

int main() {
   int n, m, i, j, k;
   printf("Enter number of processes and resources:\n");
   scanf("%d %d", &n, &m);

   int alloc[n][m], req[n][m], avail[m], finish[n];

   printf("Enter allocation matrix:\n");
   for (i = 0; i < n; i++)
      for (j = 0; j < m; j++)
         scanf("%d", &alloc[i][j]);

   printf("Enter request matrix:\n");
   for (i = 0; i < n; i++)
      for (j = 0; j < m; j++)
         scanf("%d", &req[i][j]);

   printf("Enter available matrix:\n");
   for (i = 0; i < m; i++)
      scanf("%d", &avail[i]);

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

   int done;
   do {
```

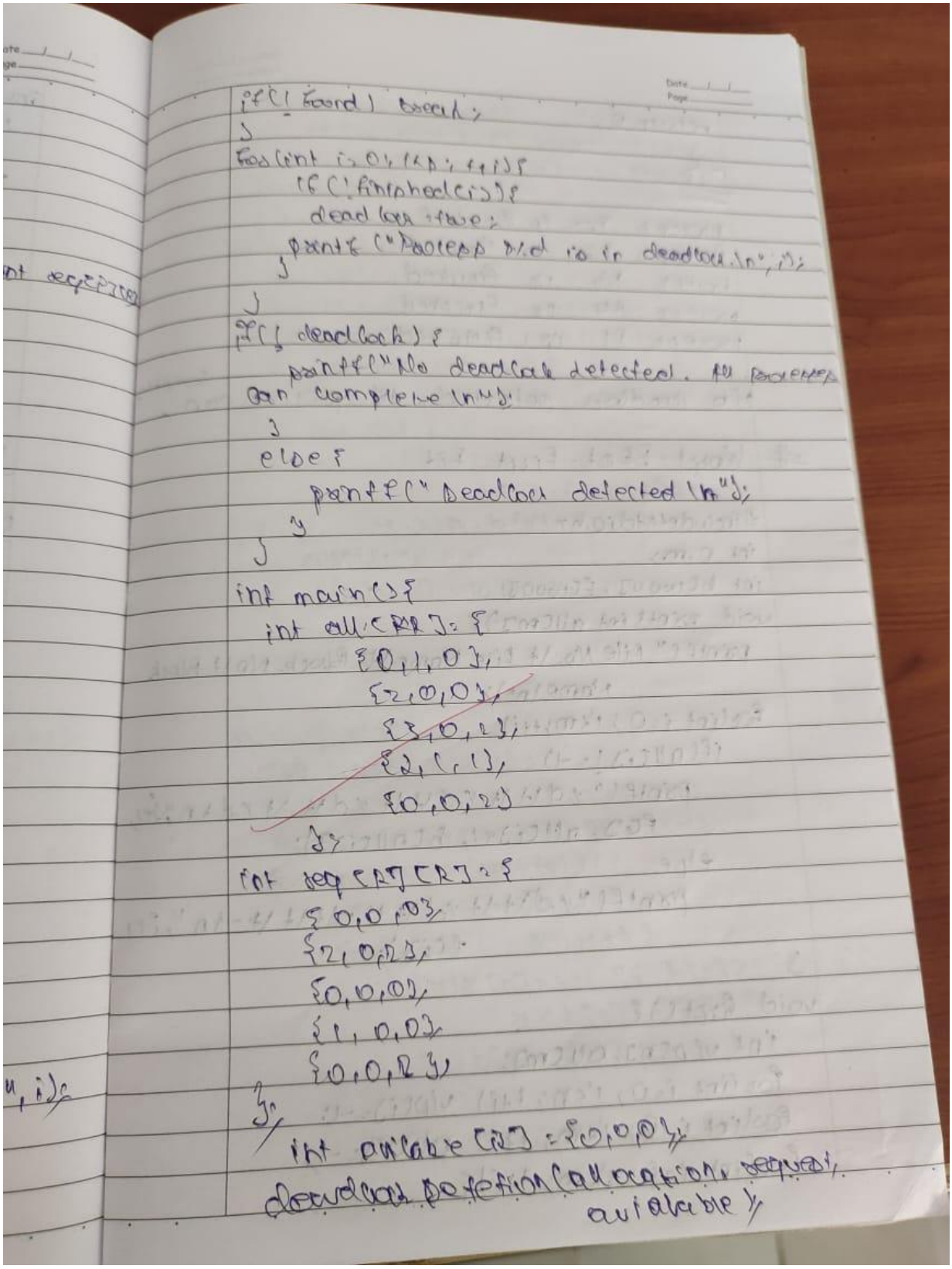
```
    done = 0;
    for (i = 0; i < n; i++) {
        if (finish[i] == 0) {
            int canFinish = 1;
            for (j = 0; j < m; j++) {
                if (req[i][j] > avail[j]) {
                    canFinish = 0;
                    break;
                }
            }
            if (canFinish) {
                for (j = 0; j < m; j++)
                    avail[j] += alloc[i][j];
                finish[i] = 1;
                done = 1;
                printf("Process %d can finish.\n", i);
            }
        }
    }
} while (done);

int deadlock = 0;
for (i = 0; i < n; i++)
    if (finish[i] == 0)
        deadlock = 1;

if (deadlock)
    printf("System is in a deadlock state.\n");
else
    printf("System is not in a deadlock state.\n");
```

```
    return 0;
}
```

```
Enter number of processes and resources:
5 3
Enter allocation matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter request matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter available matrix:
3 3 2
Process 1 can finish.
Process 3 can finish.
Process 4 can finish.
System is in a deadlock state.

Process returned 0 (0x0)   execution time : 47.372 s
Press any key to continue.
```

Deadlock Detection

```
#include<stdio.h>
#include<stdbool.h>
#define P5
#define R3
void deadlockDetection(int all[P][R], int req[P][R],
  int avi[R]){
int work[R];
bool finish[P]={false};
bool deadlock=false;
for(int i=0;i<R;++i){
   work[i]=available[i];}
int count=0;
  while(count<P){
    bool found=false;
for(int i=0;i<P;++i){
   if(!finish[i]){
    bool canProceed=true;
for(int j=0;j<R;++j)
 if(req[i][j]>work[j]){
    canProceed=false;
    break;
   }
}
if(canProceed){
   for(int j=0;j<R;++j){
work[j]=all[i][j];}
finish[i]=true;
  printf("Proceeded P%d and finished\n",i);
found=true;
  count++;
}
}
}
```



```
if(!foord) break;
}
for(int i=0;i<n;i++){
    if(!finished[i]){
        deadlock=true;
        printf("Process %d is in deadlock.\n",i);
    }
}
if(!deadlock){
    printf("No deadlock detected. All processes
can complete\n");
}
else{
    printf("Deadlock detected\n");
}
}
int main(){
int all[n][m]={
        {0,1,0},
        {2,0,0},
        {3,0,2},
        {2,1,1},
        {0,0,2}
    };
int req[n][m]={
    {0,0,0},
    {2,0,2},
    {0,0,0},
    {1,0,0},
    {0,0,2}
};
int avlabe[m]={0,0,0};
deadlock detection(allocation, request,
                   available);
```

Date / /
Page

return 0;
}

Process P0 is finished
Process P2 is finished
Process P3 is finished
Process P4 is finished
Process P1 is finished

No deadlock detected. All process can complete

Worst-Best-First Fit

Program 9:

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

a) **Worst Fit**
b) **Best Fit**
c) **First Fit**

→Worst Fit

```
#include <stdio.h>

struct Block {
   int block_no;
   int block_size;
   int is_free;
};

struct File {
   int file_no;
   int file_size;
};

void worstFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {
   printf("\nMemory 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;

      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\t%d\t\t%d\t\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);
```

```
        } else {
            printf("%d\t%d\t\tNot Allocated\n", files[i].file_no, files[i].file_size);
        }
    }
}

int main() {
    int n_blocks, n_files;

    printf("Enter the number of blocks: ");
    scanf("%d", &n_blocks);

    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;
    }

    printf("Enter the number of files: ");
    scanf("%d", &n_files);

    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);
    }

    worstFit(blocks, n_blocks, files, n_files);

    return 0;
}
```

```
Enter the number of blocks: 5
Enter the size of block 1: 100
Enter the size of block 2: 500
Enter the size of block 3: 200
Enter the size of block 4: 300
Enter the size of block 5: 600
Enter the number of files: 4
Enter the size of file 1: 212
Enter the size of file 2: 417
Enter the size of file 3: 112
Enter the size of file 4: 426

Memory Management Scheme - Worst Fit
File_no File_size       Block_no        Block_size      Fragment
1       212             5               600             388
2       417             2               500             83
3       112             4               300             188
4       426             Not Allocated
```

→Best Fit

```
#include <stdio.h>

struct Block {
    int block_no;
    int block_size;
    int is_free;
};

struct File {
    int file_no;
    int file_size;
};

void bestFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {
    printf("\nMemory 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; // Large initial 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\t%d\t\t%d\t\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);
        } else {
            printf("%d\t%d\t\tNot Allocated\n", files[i].file_no, files[i].file_size);
        }
    }
}

int main() {
    int n_blocks, n_files;

    printf("Enter the number of blocks: ");
    scanf("%d", &n_blocks);

    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;
    }

    printf("Enter the number of files: ");
    scanf("%d", &n_files);

    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);
    }

    bestFit(blocks, n_blocks, files, n_files);

    return 0;
}
```

```
Enter the number of blocks: 5
Enter the size of block 1: 100
Enter the size of block 2: 500
Enter the size of block 3: 200
Enter the size of block 4: 300
Enter the size of block 5: 600
Enter the number of files: 4
Enter the size of file 1: 212
Enter the size of file 2: 417
Enter the size of file 3: 113
Enter the size of file 4: 426

Memory Management Scheme - Best Fit
File_no File_size       Block_no        Block_size      Fragment
1       212             4               300             88
2       417             2               500             83
3       113             3               200             87
4       426             5               600             174
```

→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("\nMemory Management Scheme - First Fit\n");
   printf("File_no\tFile_size\tBlock_no\tBlock_size\tFragment\n");

   for (int i = 0; i < n_files; i++) {
      int allocated = 0;

      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;
            blocks[j].is_free = 0;

            printf("%d\t%d\t\t%d\t\t%d\t\t%d\n",
                  files[i].file_no,
```

```
                files[i].file_size,
                blocks[j].block_no,
                blocks[j].block_size,
                fragment);

            allocated = 1;
            break;
        }
    }

    if (!allocated) {
        printf("%d\t%d\t\tNot Allocated\n", files[i].file_no, files[i].file_size);
    }
  }
}

int main() {
  int n_blocks, n_files;

  printf("Enter the number of blocks: ");
  scanf("%d", &n_blocks);

  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;
  }

  printf("Enter the number of files: ");
  scanf("%d", &n_files);

  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);

  return 0;
}
```

```
Enter the number of blocks: 5
Enter the size of block 1: 100
Enter the size of block 2: 500
Enter the size of block 3: 200
Enter the size of block 4: 300
Enter the size of block 5: 600
Enter the number of files: 4
Enter the size of file 1: 212
Enter the size of file 2: 417
Enter the size of file 3: 112
Enter the size of file 4: 426

Memory Management Scheme - First Fit
File_no File_size       Block_no        Block_size      Fragment
1       212             2               500             288
2       417             5               600             183
3       112             3               200             88
4       426             Not Allocated
```

Worst - Best - First Fit

```
#include<stdio.h>
int n,m;
int b[1000],f[1000];
void print(int all[m]){
  printf(" File No. \t File Name \t Block No \t Block
          Name \n");
for(int i=0;i<m;++i)
   if(all[i]!=-1)
     printf(" %d \t %d \t %d \t %d \n",i+1,
       f[i], all[i]+1, b[all[i]]);
   else
     printf(" %d \t %d \t %d \t - \t \t - \n",i+1,
                  f[i]);
}
void first(){
  int vis[n], all[m];
  for(int i=0; i<n; ++i) vis[i]=-1;
  for(int i=0; i<m; ++i) all[i]=-1;
  for(int i=0; i<m; ++i){
```

```
for(int i
  if(vi
    y=b
  all[i]=y;
  if(y!=-1
}
print(all);
}
void worst(){
  int vis[n]
  for(int i=
  for(int x=0
  for(int i=
    int
    for(
    if(
      vis
  }
  print(all
}
```

File No.	File Name	Block No	Block Name
1	212	2	500
2	417	5	600
3	112	3	200
4	426	-	-

choice:
1. First Fit
2. Best Fit
3 Worst Fit
4. Exit
2

File No.	File Name	Block No.	Block Name
1	212	4	300
2	417	2	500
3	112	3	200
4	426	5	600

choice:
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
3

File No.	File Name	Block No.	Block Name
1	212	5	600
2	417	2	500
3	112	4	300
4	426	-	-

choice:
1. First Fit
2. Best fit
3. Worst Fit
4. Exit
4.

```
int main(){
printf("Memory Management Scheme \n");
printf("Enter the number of blocks and files\n");
scanf("%d %d",&n,&m);
printf("Enter the Block size:\n");
for(int i=0;i<n;i++){
    printf("Block %d: ",i+1);
scanf("%d",&b[i]);}
printf("Enter the File size:\n");
for(int i=0;i<m;i++){
    printf("File %d: ",i+1);
scanf("%d",&f[i]);}
while(1){
  int x;
  printf("choice:\n1. First fit\n2. Best fit
  \n3. Worst Fit\n4. Exit\n");
  scanf("%d",&x);
  if(x==4) break;
  else if(x==1) first();
  else if(x==2) best();
  else worst();
 }
}
```

Op)

```
Enter the no. of block & file: 5 4
Enter the Block Size: 100 500 200 300 600
Enter the file Size: 212 417 112 426
choice:
1. First fit
2. Best fit
3. Worst fit
4. Exit
1
```

```
for(int j=0; j<n; ++j)
    if(vip[j]==-1 && b[j]>=f[i]){
        vip[j]=1; all[i]=j; break;}}
print(all);
}
void best(){
  int vip[n], all[m];
  for(int i=0; i<n; ++i) vip[i]=-1;
  for(int i=0; i<m; ++i) all[i]=-1;
  for(int i=0; i<m; ++i){
      int x=10000; int y=-1;
      for(int j=0; j<n; ++j)
          if(vip[j]==-1 && b[j]>=f[i] &&
                  b[j]-f[i]<x){
              y=j; x=b[j]-f[i];}
      all[i]=y;
      if(y!=-1) vip[y]=1;
  }
  print(all);
}
void worst(){
  int vip[n], all[m];
  for(int i=0; i<n; ++i) vip[i]=-1;
  for(int i=0; i<m; ++i) all[i]=-1;
  for(int i=0; i<m; ++i){
      int x=-1, y=-1;
      for(int j=0; j<n; ++j)
          if(vip[j]==-1 && b[j]>=f[i] &&
                  b[j]-f[i]>x){
              x=b[j]-f[i]; y=j;}
      if(y!=-1)
          vip[y]=1; all[i]=y;
  }
  print(all);
}
```

Program 10:
Write a C program to simulate page replacement algorithms

a) **FIFO**
b) **LRU**
c) **Optimal**

→FIFO

```
#include <stdio.h>

int main() {
   int n, frames, i, j, k, found, index = 0, page_faults = 0, hits = 0;
   char pages[100];

   printf("Enter the size of the pages:\n");
   scanf("%d", &n);
   printf("Enter the page strings:\n");
   scanf("%s", pages);
   printf("Enter the no of page frames:\n");
   scanf("%d", &frames);

   int mem[frames];
   for (i = 0; i < frames; i++) mem[i] = -1;

   for (i = 0; i < n; i++) {
      found = 0;
      for (j = 0; j < frames; j++) {
         if (mem[j] == pages[i] - '0') {
            hits++;
            found = 1;
            break;
         }
      }
      if (!found) {
         mem[index] = pages[i] - '0';
         index = (index + 1) % frames;
         page_faults++;
      }
   }

   printf("FIFO Page Faults: %d, Page Hits: %d\n", page_faults, hits);
   return 0;
}
```

```
Enter the size of the pages:
7
Enter the page strings:
103563
Enter the no of page frames:
3
FIFO Page Faults: 6, Page Hits: 1
```

→LRU

```
#include <stdio.h>

int main() {
    int n, frames, i, j, k, page_faults = 0, hits = 0;
    char pages[100];
    int mem[10], used[10];

    printf("Enter the size of the pages:\n");
    scanf("%d", &n);
    printf("Enter the page strings:\n");
    scanf("%s", pages);
    printf("Enter the no of page frames:\n");
    scanf("%d", &frames);

    for (i = 0; i < frames; i++) {
        mem[i] = -1;
        used[i] = -1;
    }

    for (i = 0; i < n; i++) {
        int page = pages[i] - '0';
        int found = 0;

        for (j = 0; j < frames; j++) {
            if (mem[j] == page) {
                hits++;
                used[j] = i;
                found = 1;
                break;
            }
        }

        if (!found) {
            int lru = 0;
            for (j = 1; j < frames; j++) {
                if (used[j] < used[lru]) lru = j;
            }
            mem[lru] = page;
```

```
            used[lru] = i;
            page_faults++;
        }
    }

    printf("LU Page Faults: %d, Page Hits: %d\n", page_faults, hits);
    return 0;
}
```

```
Enter the size of the pages:
7
Enter the page strings:
1303563
Enter the no of page frames:
3
LU Page Faults: 5, Page Hits: 2
```

→Optimal

```
#include <stdio.h>

int main() {
    int n, frames, i, j, k, page_faults = 0, hits = 0;

    printf("Enter the size of the pages:\n");
    scanf("%d", &n);

    char pages[n + 1];
    printf("Enter the page strings:\n");
    scanf("%s", pages);

    printf("Enter the no of page frames:\n");
    scanf("%d", &frames);

    int mem[frames], next_use[frames];
    for (i = 0; i < frames; i++) {
        mem[i] = -1;
    }

    for (i = 0; i < n; i++) {
        int page = pages[i] - '0';
        int found = 0;

        for (j = 0; j < frames; j++) {
            if (mem[j] == page) {
                hits++;
                found = 1;
                break;
            }
```

```
        }

        if (!found) {
            if (page_faults < frames) {
                mem[page_faults++] = page;
            } else {
                for (j = 0; j < frames; j++) {
                    next_use[j] = -1;
                    for (k = i + 1; k < n; k++) {
                        if (mem[j] == pages[k] - '0') {
                            next_use[j] = k;
                            break;
                        }
                    }
                }

                int farthest = 0;
                for (j = 1; j < frames; j++) {
                    if (next_use[j] > next_use[farthest]) {
                        farthest = j;
                    }
                }

                mem[farthest] = page;
                page_faults++;
            }
        }
    }

    printf("Optimal Page Faults: %d, Page Hits: %d\n", page_faults, hits);
    return 0;
}
```

```
Enter the size of the pages:
7
Enter the page strings:
13035631
Enter the no of page frames:
3
Optimal Page Faults: 6, Page Hits: 1
```

#1 FIFO-LRU-OPTIMAL.

```
#include<stdio.h>
int n,m;
int p[1000], f[1000];
void fifo(){
  for(int i=0;i<m;++i) f[i]=-1;
  int h=0,o=0;
  for(int i=0;i<n;++i){
      int k=0;
    for(int j=0;j<m;++j)
    if(p[i]==f[j]){++h;k=1;break;}
    if(k==1) continue;
    for(int j=0;j<m;++j)
      if(f[j]==-1){
        f[j]=p[i];k=2;break;}
    if(k==2) continue;
    f[o]=p[i];
    o=(o+1)%m;
  }
printf("FIFO:\n\tPage-Hit:%d\n\tPage-Fault:%d\n",h,n-h);
}
void lru(){
  for(int i=0;i<m;++i) f[i]=-1;
  int h=0;
  int t[m];
  for(int i=0;i<m;++i) t[i]=0;
  for(int i=0;i<n;++i){
    int k=0;
    for(int j=0;j<m;++j)
      if(f[j]==-1) ++t[j];
```

```
for(int d=0; d<m; ++d)
    if(f[d]==p[i]) {t[d]=1; k=1; break;}
if(k==1) {++h; continue;}
for(int d=0; d<m; ++d)
    if(f[d]==-1) {f[d]=p[i]; t[d]=1;
                  k=d; break;}
if(k==2) continue;
int x=0;
for(int d=1; d<m; ++d)
    if(t[d]>t[x]) x=d;
f[x]=1; f[x]=p[i];
}
printf("\nLRU:\t\t\t Page Hit %d\t\t
        Page Fault : %d \n", h, n-h);
}
void optimal(){
  for(int i=0; i<m; ++i) f[i]=-1;
  int h=0;
  for(int i=0; i<n; ++i){
      int k=0;
    for(int d=0; d<m; ++d)
        if(f[d]==p[i]) {k=1;
              ++h; break;}
    if(k==1) continue;
    for(int d=0; d<m; ++d)
        if(f[d]==-1) {
      k=d; f[d]=p[i]; break;}
    if(k==2) continue;
    int l=-1, dx=-1, j=1;
    for(int j=0; j<m; ++j){
        y=1
      for(k=i+1; k<n; ++k)
          if(f[j]==p[k]){
```

```
        j=0;
        if(k!=0) {temp=j; j=k; k=j; break;}
    }
    if(!j) {
        f[j]=p[i]; break;}
    }
    if(j) continue;
    f[temp]=p[i];
}
printf("OPTIMAL: \n Page Hit: %d \n Page Fault: %d \n",h,n-h);
}
int main(){
printf("Enter no. of page String and Page-Frame: ");
scanf("%d %d", &n, &m);
printf("Enter the page String: ");
for(int i=0; i<n; ++i)
scanf("%d", &p[i]);
fifo(); lru(); optimal();
}
```

OUTPUT

```
Enter No. of Page String and Page-frame: 7 3
Enter the Page String: 1 3 0 3 5 6 3
FIFO:    Page Hit: 1    Page Fault: 6
LRU:     Page Hit: 2    Page Fault: 5
OPTIMAL: Page Hit: 2    Page Fault: 5.
```