VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

OPERATING SYSTEMS

Submitted by

T. TILAK REDDY(1WA23CS005)

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 Venugopal M(1WA23CS038), 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. Sheetal V A
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;
}
```

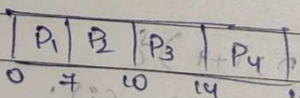
Output

```
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


=== Code Execution Successful ===
```

```
TAT += Turnaround time (i);
}
wT /= n;
CT /= n;
TAT /= n;
Pf ("completion time : %f \n turnaround time :
        %f \n waiting time : %f", CT, TAT, wT);
}
```

1: "write a C program to simulate following non primitive CPU.

Processor	Arrival	Burst	completion	TAT	WT	RT
P_1	0	7	7	7	0	0
P_2	0	3	10	10	7	7
P_3	0	4	14	14	10	10
P_4	0	6	20	20	14	14



output :

no of process 4
AT 0 0 0 0
BT 7 3 4 6
CT 12.75
TAT 12.75
WT 7.75

(1) First come first serve:

```
int main(){
  int n;
  Printf("no. of process:");
  Scanf("%d",&n);
  int arrival time(n);
  int completion time(n);
  int waiting time(n);
  if("Arrival time of each Process");
    for(int i=0 ; i<n ; i++){
      Scanf("%d", & Arrival time);
    }
    if("Burst time of each process:");
      for(int i=0; i<n; i++){
        if("%d", & Burst time(i));
}
int Sum=0;
for(int i=0; i<n; i++){
      Sum+ = Burst time(i);
      Completion time(i) = Sum;
}
for(i=0; i<n; i++){
  Turnaround time(i) = Completion time(i)-
                       Arrival time(i);
}
for(int i=0; i<n; i++){
  waiting time(i) = turnaround time(i)- Burst time
                                        (i);
}
float (T, TAT, WT;
  for(int i=0 ; i<n ; i++){
      WT= + waiting time(i);
      CT+ = Completion time(i);
```

→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;
}
```

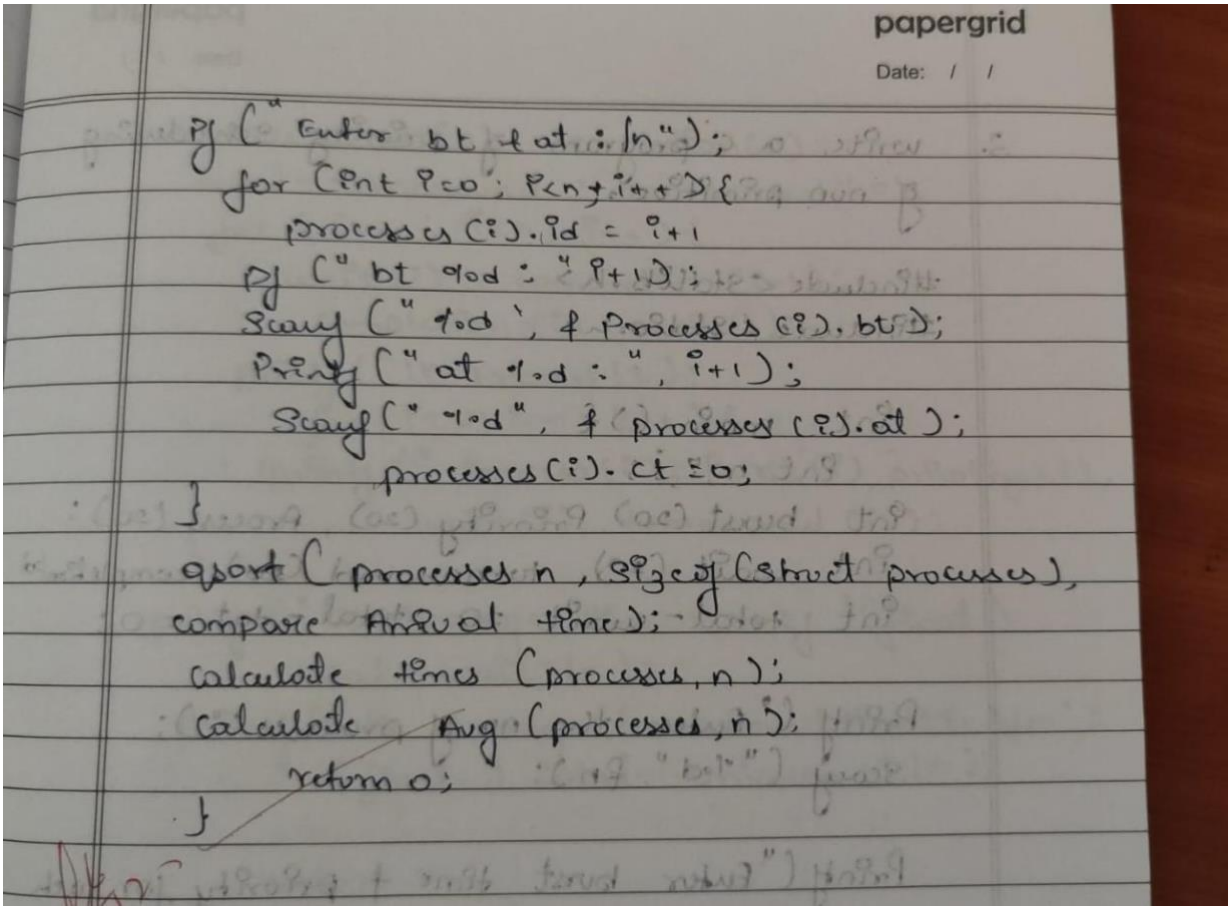
Output

```
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   20  20  13  13
P2  0   3   3   3   0   0
P3  0   4   7   7   3   3
P4  0   6   13  13  7   7


=== Code Execution Successful ===
```

2. Write a C program to shortest job first (non-primitive)

```
#include <stdio.h>
#include <stdlib.h>
Struct Process {
    int id;
    int bt;
    int at;
    int ct;
    int tat;
    int wt;
};
int compare Arrival time (const void *a, const
void *b){
    return ((struct process *) a)->at -((struct
Process *) b)->at;
}
void calculate time (struct Process process[], int n
{
int time=0;
int completed=0;
while (completed<n){
    int shortest = -1;
    int min-burst=100000;
    for (int i=0; i<n; i++){
        if (Processes (i).at <= time && proces
        -s(i).ct==0){
            if (processes(i).bt<min-burst){
                min-burst = Processes (i).bt;
                shortest=i;
            }
        }
    }
}
```

```
if (shortest == -1){
        time++;
    } else {
        processes[shortest].ct = time + Processes[
    shortest].ct - Processes[shortest].at;
        processes[shortest].wt = Processes[shortest]
        .bt;

        time = Processes[shortest].ct;
        Completed++;
    }
}
}
void calculateAvg[struct Processes processes[]
    int n){
    int total_wt=0, total_tat=0;
    for(int i=0; i<n; i++){
        total_wt += Processes[i].wt;
        total_tat += Processes[i].tat;
    }
    printf("\n avg wt = %.2f", (float)total_wt/n);
    printf("\n avg tat = %.2f", (float)total_tat/n");
int main(){
    int n;
    pf("no of processes:");
    Scanf("%d", &n);
    struct processes processes[n];
```



```
pf("Enter bt & at :\n");
for(int i=0; i<n; i++){
    processes[i].id = i+1
    pf("bt %d : ", i+1);
    scanf("%d", &processes[i].bt);
    Printf("at %d : ", i+1);
    Scanf("%d", &processes[i].at);
        processes[i].ct=0;
}
qsort(processes n, sizeof(struct process),
compare Arrival time);
calculate times (processes, n);
calculate Avg (processes, n);
    return 0;
}
```

→SJF(Pre-emptive)

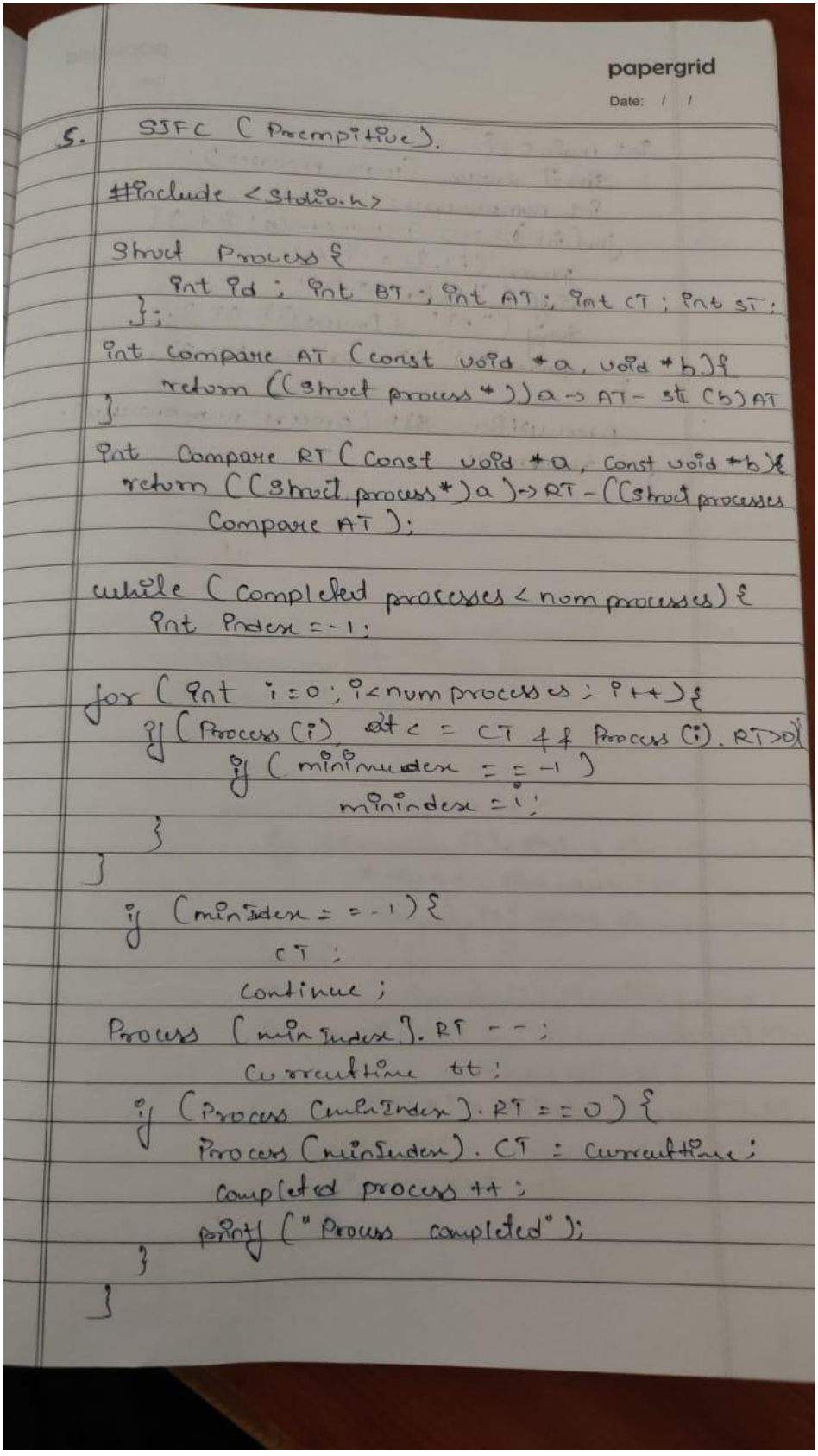
```
#include <stdio.h>

#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;
}
```

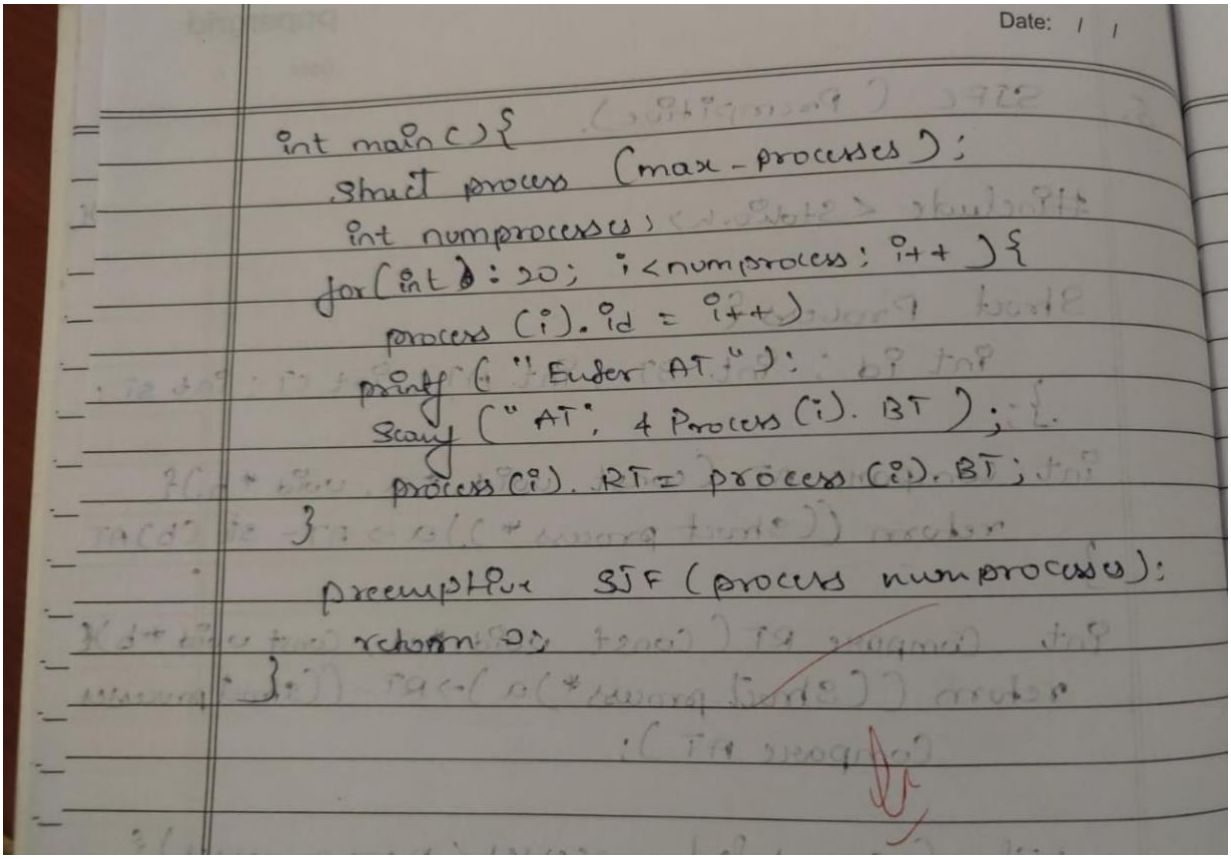
Output

```
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


=== Code Execution Successful ===
```



5. SJFC (Preemptive).

```
#include <stdio.h>
struct Process {
    int id; int BT; int AT; int CT; int ST;
};
int compareAT (const void *a, void *b){
    return ((struct process *)a -> AT - st (b) AT
}
int Compare RT (const void *a, const void *b){
 return ((struct process*)a)-> RT - ((struct processes
        Compare AT );

while (completed processes < num processes) {
    int Index = -1;

for (int i=0; i<num processes; i++){
    if (Process (i) at c = CT && Process (i). RT>0)
        if (minimuderx == -1)
            minindex = i;
    }
}
    if (minIndex = = -1){
            CT ;
        continue ;
Process [minIndex]. RT --;
        Current time tt;
    if (Process (CurIndex). RT == 0) {
     Process (minIndex). CT : Current time;
      completed process ++;
      printf ("Process completed");
    }
}
```



Date: / /

```
int main(){
    struct process (max - processes);
    int numprocesses;
for(int i=0; i<numprocess; i++){
    process(i).id = i++;
printf("Enter AT");
scanf("AT", &Process(i).BT);
process(i).RT = process(i).BT;
}
preemptive SJF (process numprocesses);
return 0;
}
```

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
```

papergrid
Date: / /

3. write a C program of priority scheduling of non primitive:

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

int main (){
    int n, i, j;
int burst (20), Priority (20), Process (20);
int wait (20), turnaround (20), completion[20]
int total-wait =0, total-tat =0;

Printf ("Enter the no of processes:");
scanf ("%d", &n);

Printf ("Enter burst time & priority for each
    process : \n");
for (i=0; i<n; i++){
    printf ("Process %d : ", i+1);
    scanf ("%d %d", &burst(i), &priority(i));
    process (i) = i+1;
    }
for (i=0; i<n; i++) {
    if (priority (i) < Priority (j) {
        int temp = Priority (i);
        priority (i) = Priority (j);
        priority (j) = temp;
        temp = burst (i);
        burst (i) = burst (j);
        burst (j) = Process;
        Process (i) = Process (j)
        wait (0) = 0;
        completion (0) = burst (0);
```

papergrid

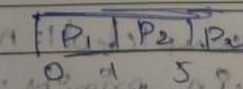
Date: / /

```
turnaround(0) = completion(0);
total-turnaround = turnaround(0);
for (i=0; i<n; i++) {
    wait(i) = completion(i-1);
    completion(i) = wait(i);
printf("\n Process\t");
    for(i=0; i<n; i++) {
printf(" Process(i), burst(i), priority(i),
        wait(i), turnaround(i));
}
printf("\n Average time : %.2f ", (float)
    total-wait/n);
printf("\n Average turnaround time: %.2f\n",
    (float) total-turnaround/n);
return 0;
}
```

output:

Enter the no. of processes:

P	AT	BT	CT	TAT	WT	RT
P_1	0	8	17	17	9	0
P_2	1	4	5	4	0	0
P_3	2	9	26	24	15	15
P_4	3	5	10	7	2	2
					6.50	



→Priority(Pre-emptive)

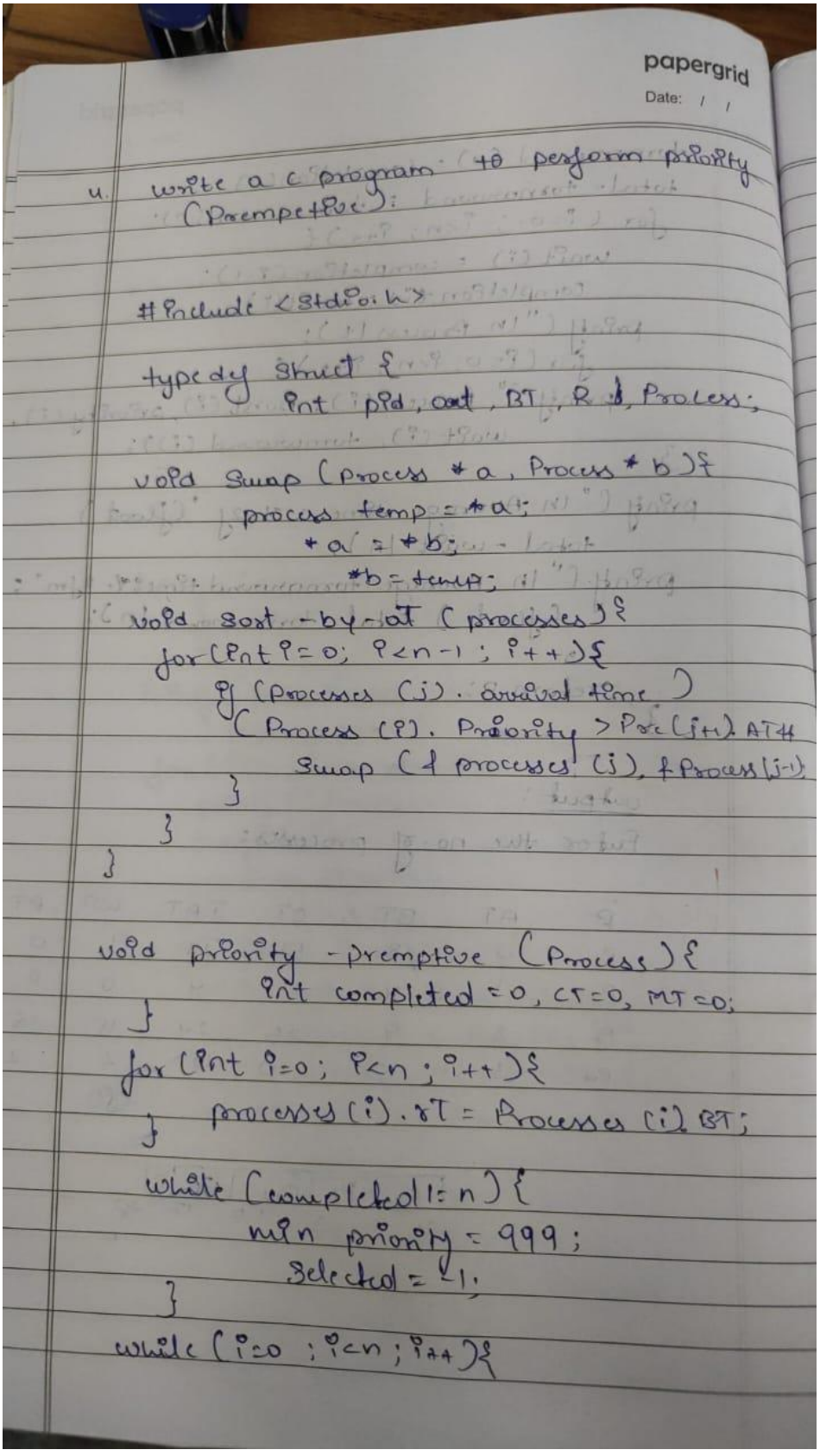
```
#include <stdio.h>
#include <limits.h>

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;
}
```

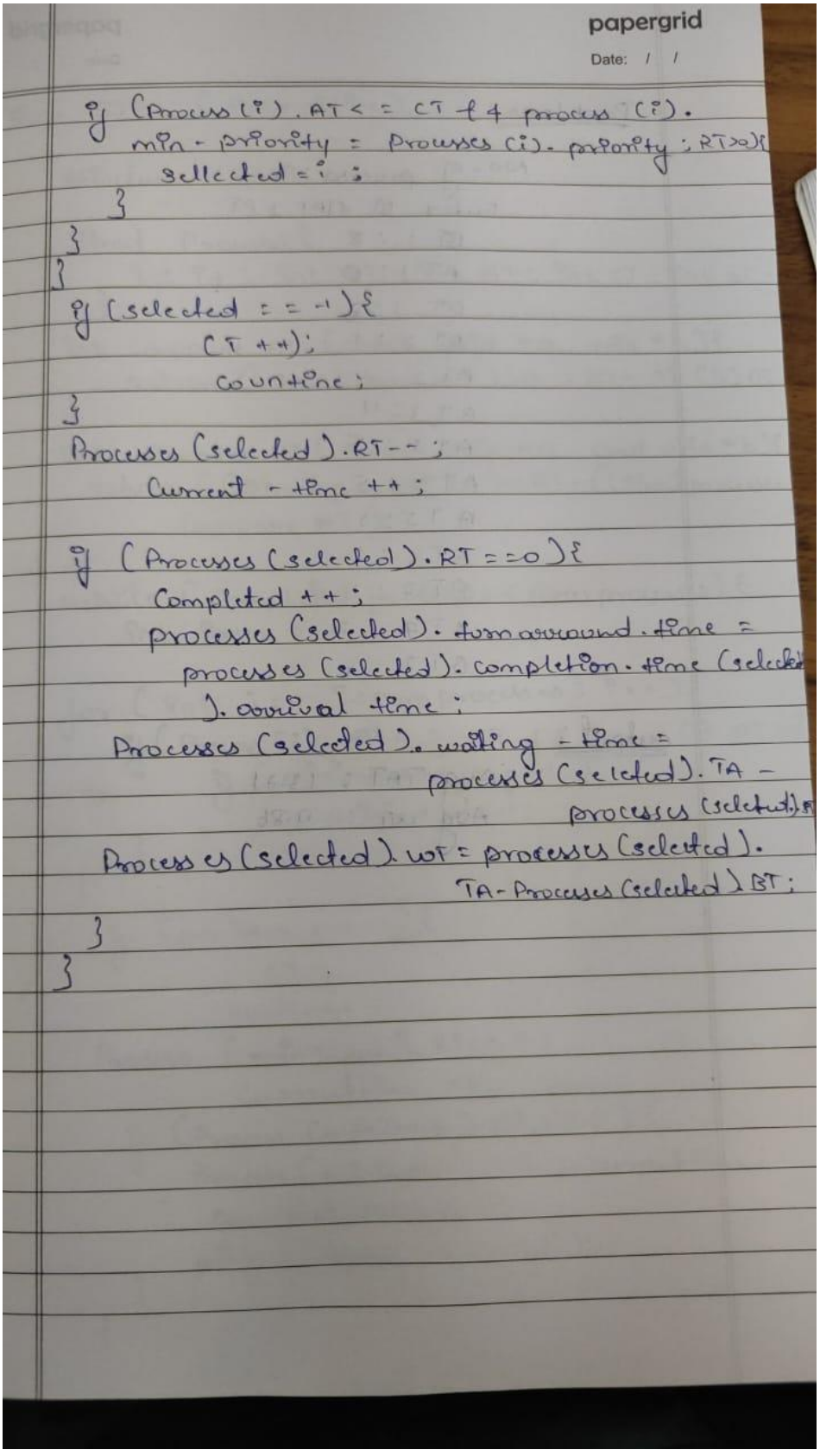
Output Clear

```
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
```

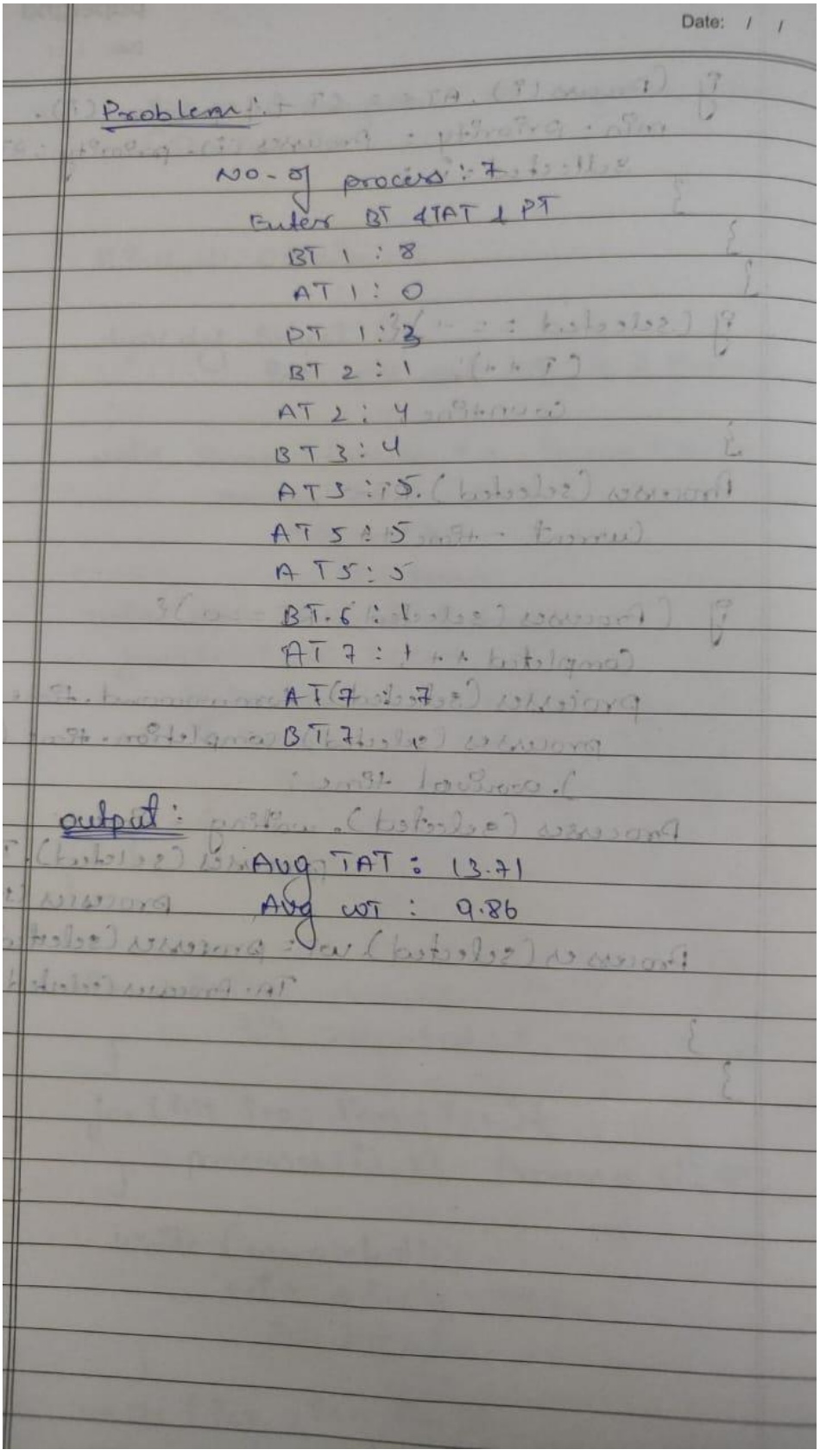


4. write a c program to perform priority (Preemptive):

```
#include <stdio.h>

typedef struct {
    int pid, at, BT, RT, Process;

void Swap (Process *a, Process *b){
    process temp = *a;
    *a = *b;
    *b = temp;
void sort_by_at (processes){
  for(int i=0; i<n-1; i++){
      if (Process (j). arrival time )
      ( Process (i). Priority > Pr(j+1) AT4
          Swap (& processes (j), & Process (j-1)
      }
  }
}

void priority - premptive (Process){
      int completed = 0, CT=0, MT=0;
  }
  for (int i=0; i<n; i++){
  }  processes (i). RT = Process (i) BT;
  while (completed != n){
      min priority = 999;
      selected = -1;
  }
while (i=0 ; i<n; i++){
```



```
if (Process(i).AT <= CT && process(i).
min-priority = Processes(i).priority; RT>0){
    selected = i;
  }
}
}
if (selected == -1){
        CT++;
        countime;
}
Processes(selected).RT--;
    Current-time++;

if (Processes(selected).RT==0){
    Completed++;
    processes(selected).turnaround.time =
      processes(selected).completion.time(selected
      ).arrival time;
  Processes(selected).waiting-time =
                    processes(selected).TA -
                                processes(selected).
  Processes(selected).WT = processes(selected).
                    TA-Processes(selected)BT;
  }
}
```


Date: / /
Problem:
No. of process: 7
Enter BT & TAT & PT
BT 1 : 8
AT 1 : 0
PT 1 : 3
BT 2 : 1
AT 2 : 4
BT 3 : 4
AT 3 : 5
AT 5 : 5
AT 5 : 5
BT 6 : 1
AT 7 : 1
Output:
Avg TAT : 13.71
Avg WT : 9.86

→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
```

papergrid

Date: / /

5. Round Robin:

```
#include <stdio.h>
    struct process {
        int id, bt, at, ct, tat, wt;
    };
void RR(struct process process[], int n, int quantum)
    } int time=0; completed=0;
    while (completed<n) {
        int done=1;
    for (int i=0; i<n; i++)
        if (Process(i).rt > quantum) {
            time + quantum;
        P(i).rt = quantum;
        } else {
        time += Process(i).rt;
        Process(i) ct = time;
        Process(i) tat = Process(i).ct -
                         Process(i) at;
        Process(i).rt=0;
                completed++;
            }
        }
    }
    if (done) ++;
}
}
call Avg (struct process procus(), int n) {
        int total wt=0; total.tat=0;
    for (int i=0; i<n; i++) }
        total_wt += Process(i).wt;
        total_wt += Process(i).tat;
    }
```

```
int main() {
    int n, quantum;
    pf("No of processes:");
    scanf("%d", &n);
    struct processes process(n);
    for (int i=0; i<n; i++) {
        process(i); d = i+1;
        pf("BT%d: ", i++);
        sf("%d", &Process(i).bt);
        pf("%d", &Process(i).at);
    }
    return 0;
```

output:

No of process: 5
Enter BT & AT
BT: 8
AT: 0
BT: 2: 2
AT: 5
BT: 7
AT: 1
BT: 3

P	AT	BT	CT	TAT	WT
P_1	0	8	20	19	11
P_2	5	2	8	3	1
P_3	1	7	23	22	15
P_4	6	3	17	11	8
P_5	8	5	26	18	13

$\Sigma = 48$

P_1	P_3	P_1	P_2	P_4	P_5	P_1	P_3	P_4	P_1	P_5	P_3	P_5
0	2	4	6	8	10	12	14	16	17	19	21	23

Program 3:

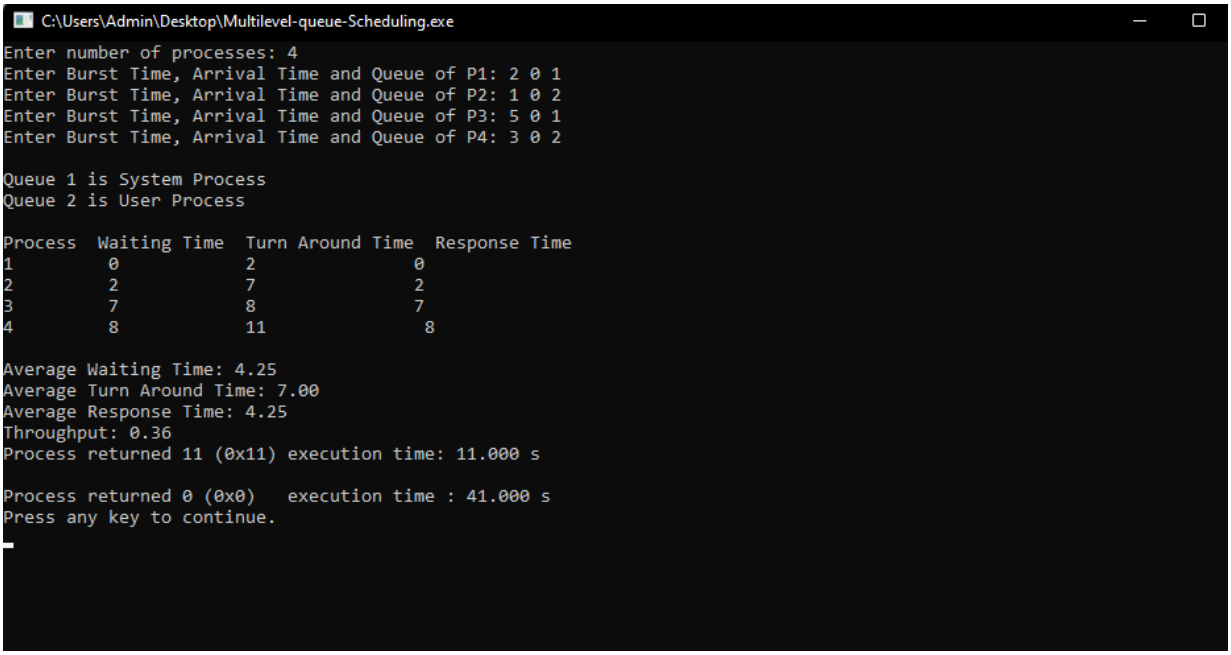
Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for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

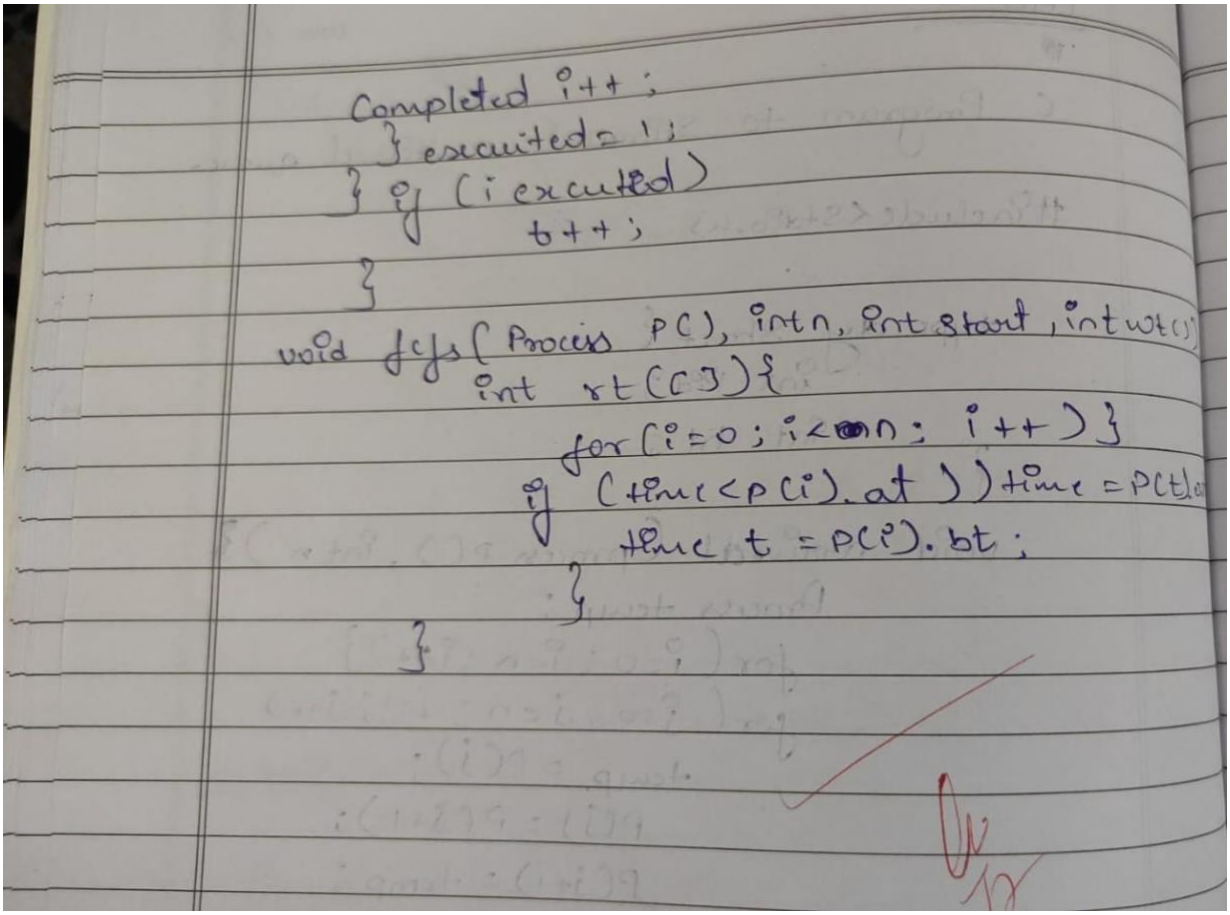
papergrid

Date: / /

LAB - 3.

1. C Program to stimulate multilevel queue.

```
#include<stdio.h>

typedef struct {
    int Pid;
    int bt;
}

void arrivedl (process P(), int n ){
    Process temp;
    for( i=0; i<n; i++)}
    for( j=0; j<n; j-1; j++)
        temp = P(j);
        P(j) = P(j+1);
        P(j+1) = temp;
    }
}

void round robin (process (), int n, int quantum)
    int remain_bt (MAX_Process );
    remaining bt(i) = P(i).bt;

    int t=0, Completed=0;
    while (comp<n){
        int executed=0;
        for(int i=1; i<n; i++ ){
        if ( remaining - bt(i) >0){
            if(remaining -bt(i) > quantum){
                t+ = quantum;
            remaining bt = quantum;
        } else{
        t + = remaining - bt (i)
        tat (i) = t - p(i).at;
        wt (i) = tat (i) - P(i). at;
```



```
Completed i++;
  } executed = 1;
} if (i executed)
      t++;
}
void fcfs( Process P(), int n, int start, int wt(),
     int rt(C]){
          for(i=0; i<n; i++){
     if (time<P(i).at )) time = P(t].a
       time t = P(i).bt;
      }
}
```

Program 4:
Write a C program to simulate Real-Time CPU Scheduling algorithms:
a) Rate- Monotonic
b) Earliest-deadline First

→Rate- Monotonic

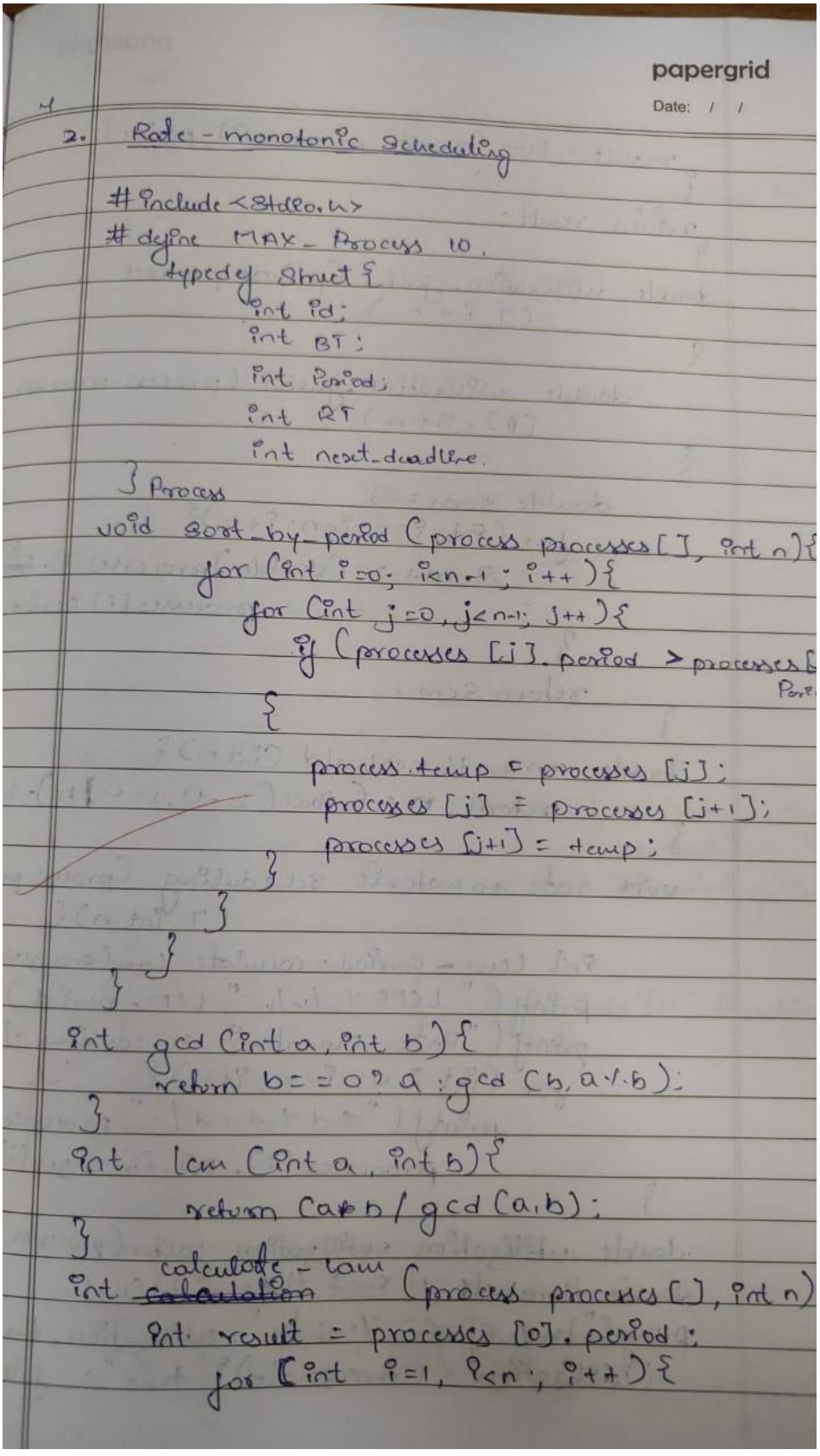
```
#include <stdio.h>
#include <math.h>

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
```

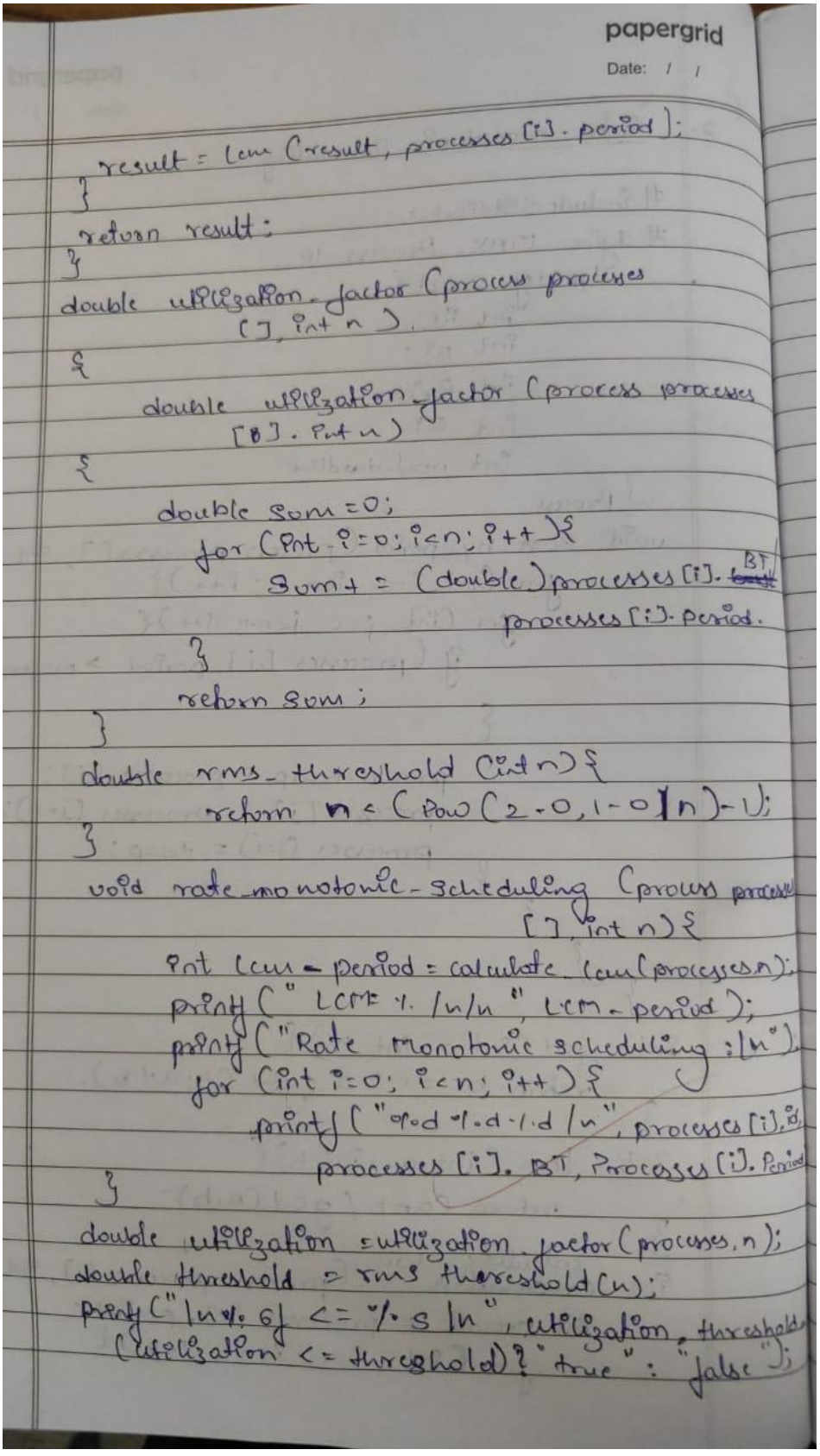


papergrid

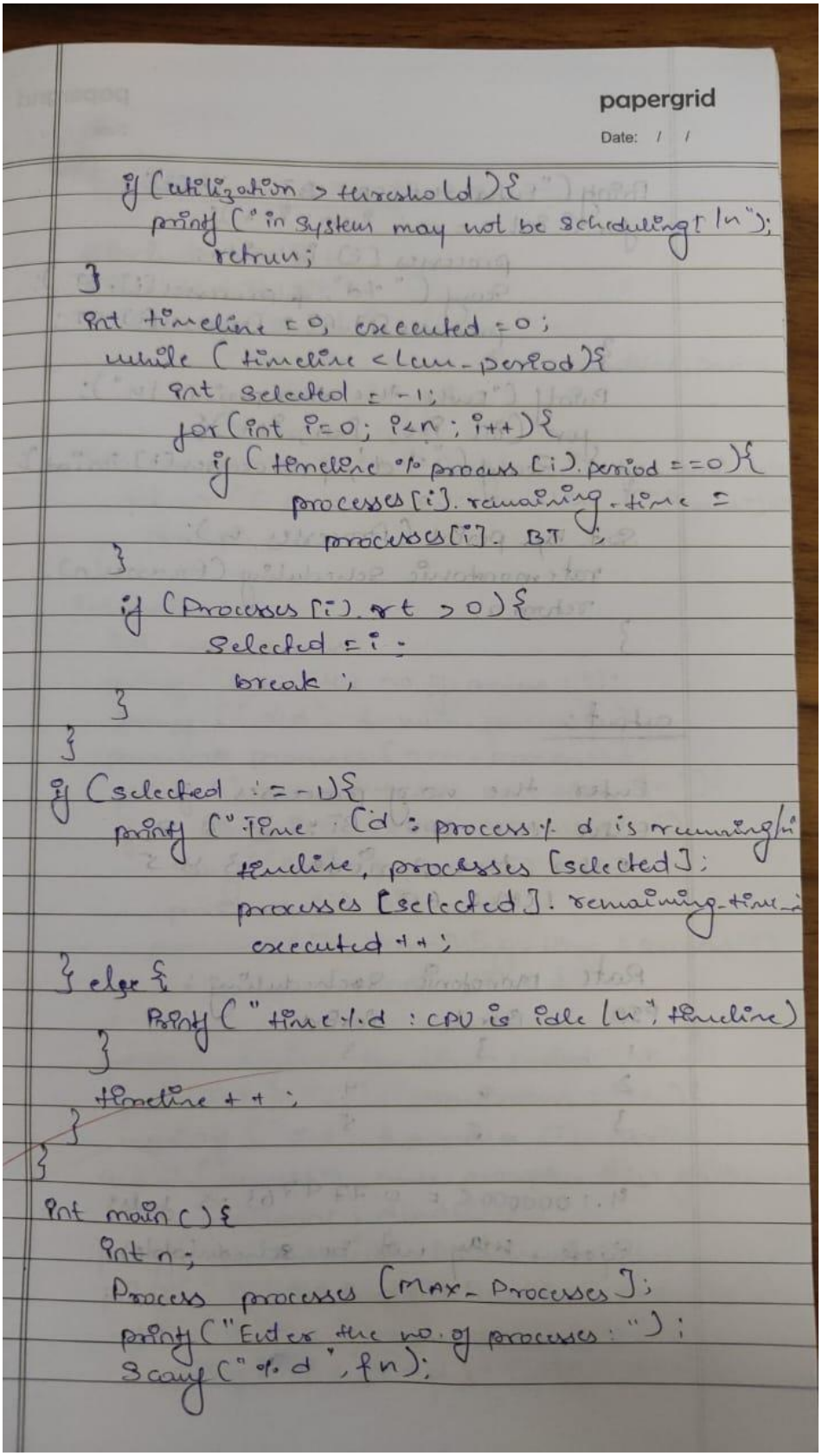
Date: / /

2. Rate - monotonic scheduling

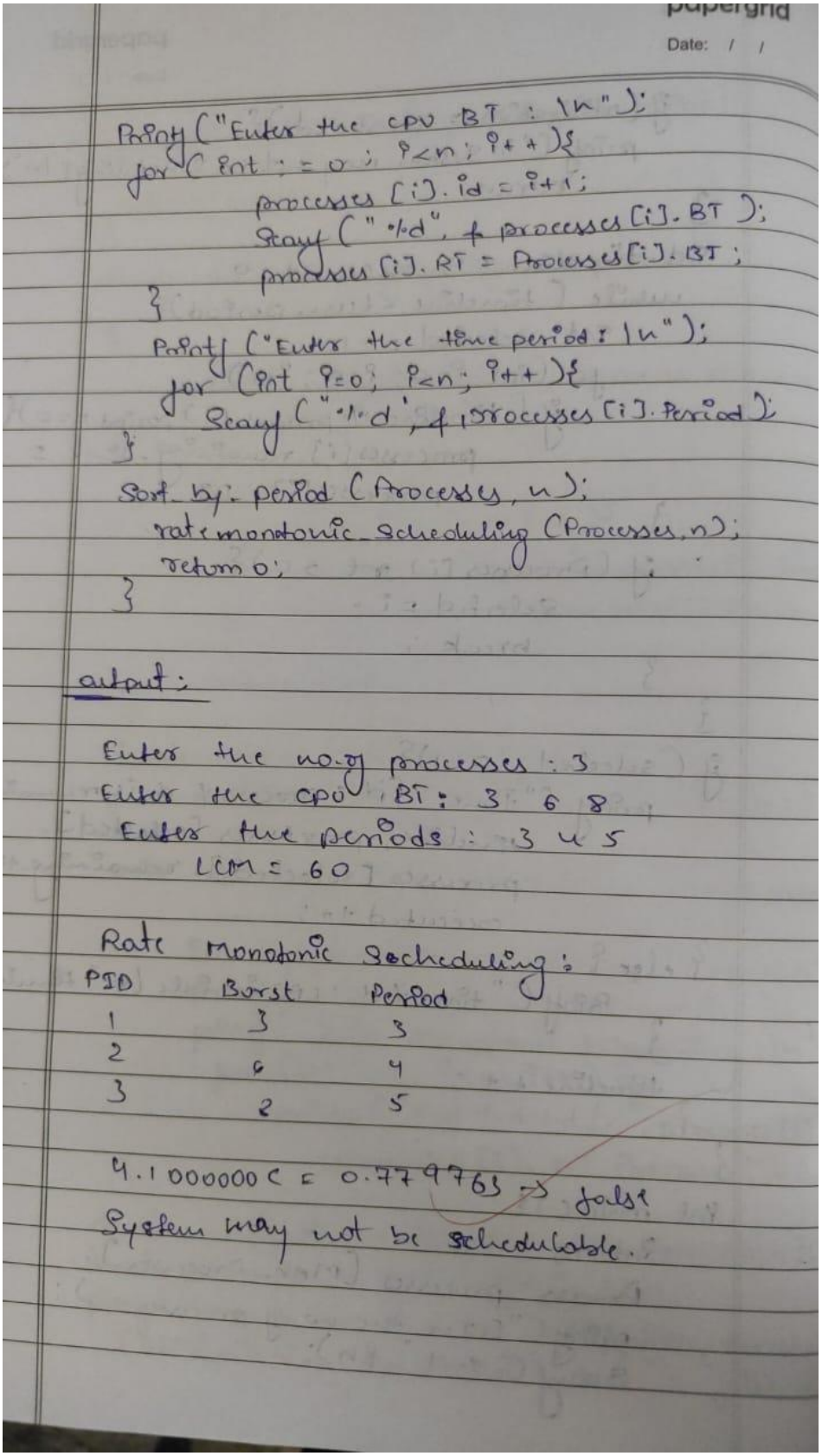
```
#include <stdio.h>
#define MAX_Process 10.
    typedef struct {
        int id;
        int BT;
        int Period;
        int RT
        int next_deadline.
    } Process
void sort_by_period (process processes[], int n){
    for (int i=0; i<n-1; i++){
        for (int j=0, j<n-1; j++){
            if (processes[j].period > processes[
                                        Peri
            {
                process temp = processes[j];
                processes[j] = processes[j+1];
                processes[j+1] = temp;
            }
        }
    }
}
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 calculate - lcm (process processes[], int n)
    int result = processes[0].period;
    for (int i=1, i<n; i++){
```



```
    result = lcm(result, processes[i].period);
}
return result;
}
double utilization_factor(process processes
        [], int n)
{
    double utilization_factor(process processes
        [0], int n)
{
    double sum=0;
    for(int i=0; i<n; i++){
        sum+ = (double)processes[i].BT
                processes[i].period.
    }
    return sum;
}
double rms_threshold(int n){
    return n*(pow(2.0,1.0/n)-1);
}
void rate_monotonic_scheduling(process process
        [], int n){
    int lcm_period = calculate_lcm(processes,n);
    printf("LCM: %d\n", lcm_period);
    printf("Rate Monotonic scheduling:\n");
    for(int i=0; i<n; i++){
        printf("%d %d %d\n", processes[i].id,
            processes[i].BT, Processes[i].Period
}
double utilization=utilization_factor(processes, n);
double threshold = rms_threshold(n);
printf("%d %f <= %f %s\n", utilization, threshold,
    (utilization <= threshold)? "true": "false");
```



```
if (utilization > threshold) {
    printf ("in system may not be scheduling! \n");
        return;
}
int timeline = 0, executed = 0;
while (timeline < lcm_period) {
    int selected = -1;
    for (int i=0; i<n; i++) {
        if (timeline % process[i].period == 0) {
            processes[i].remaining_time =
                processes[i].BT;
    }
    if (Processes[i].rt > 0) {
        selected = i;
        break;
    }
}
if (selected != -1) {
    printf ("Time [d: process %d is running\n"
        timeline, processes[selected];
        processes[selected].remaining_time--;
        executed++;
} else {
    Printf ("time %d : CPU is idle \n", timeline)
}
timeline++;
}
}
int main() {
    int n;
    Process processes [MAX_Processes];
    printf ("Enter the no. of processes: ");
    scanf ("%d", &n);
```



```
Printf("Enter the CPU BT : \n");
for(int i=0; i<n; i++){
        processes[i].id = i+1;
        Scanf("%d", &processes[i].BT);
        processes[i].RT = Processes[i].BT;
}
Printf("Enter the time period : \n");
for(int i=0; i<n; i++){
    Scanf("%d", &processes[i].Period);
}
Sort_by_period(Processes, n);
rate_monotonic_scheduling(Processes, n);
return 0;
}
```

output:

```
Enter the no.of processes : 3
Enter the CPU BT : 3 6 8
Enter the periods : 3 4 5
    LCM = 60
```

Rate Monotonic Scheduling:

PID	Burst	Period
1	3	3
2	6	4
3	2	5

4.1000000 < 0.779763 -> false

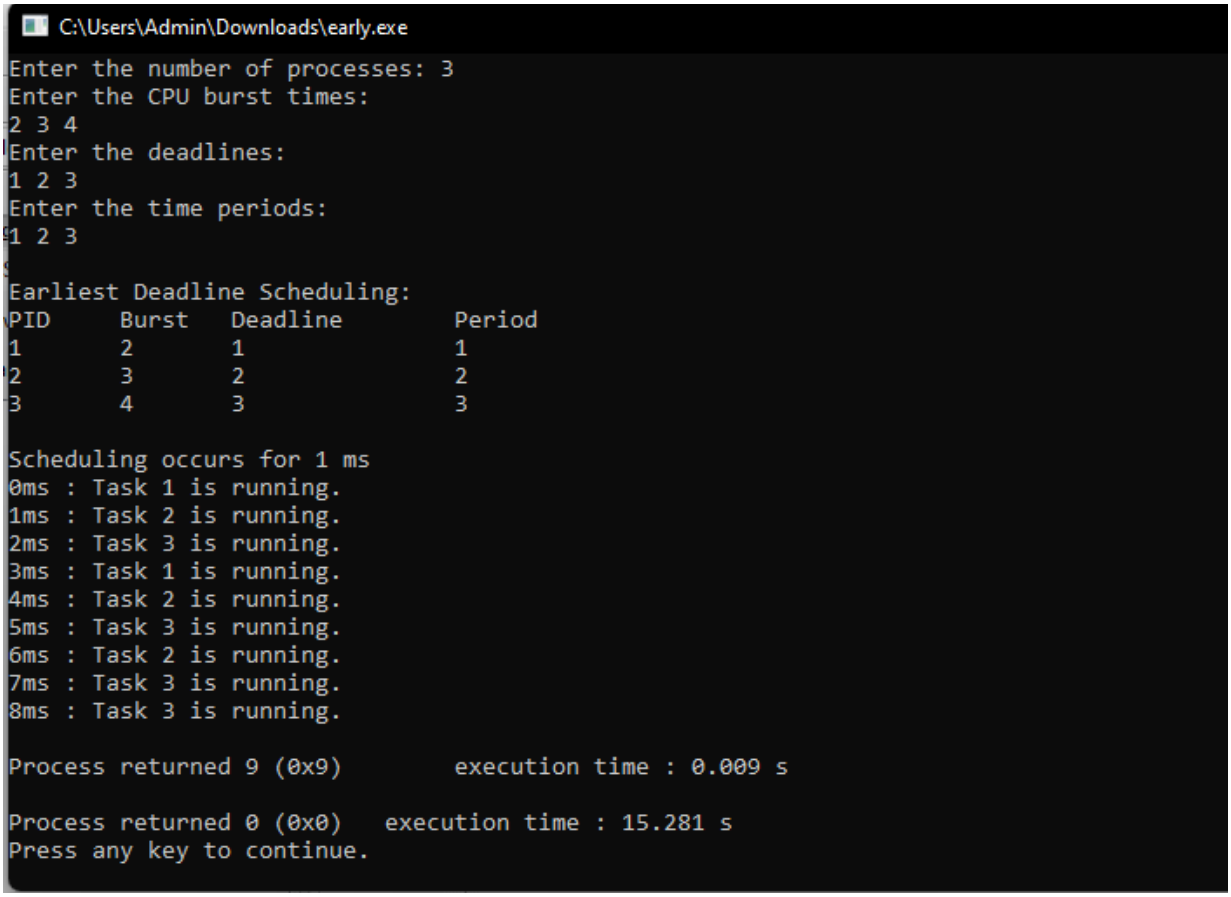
System may not be schedulable.

→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;
}
```

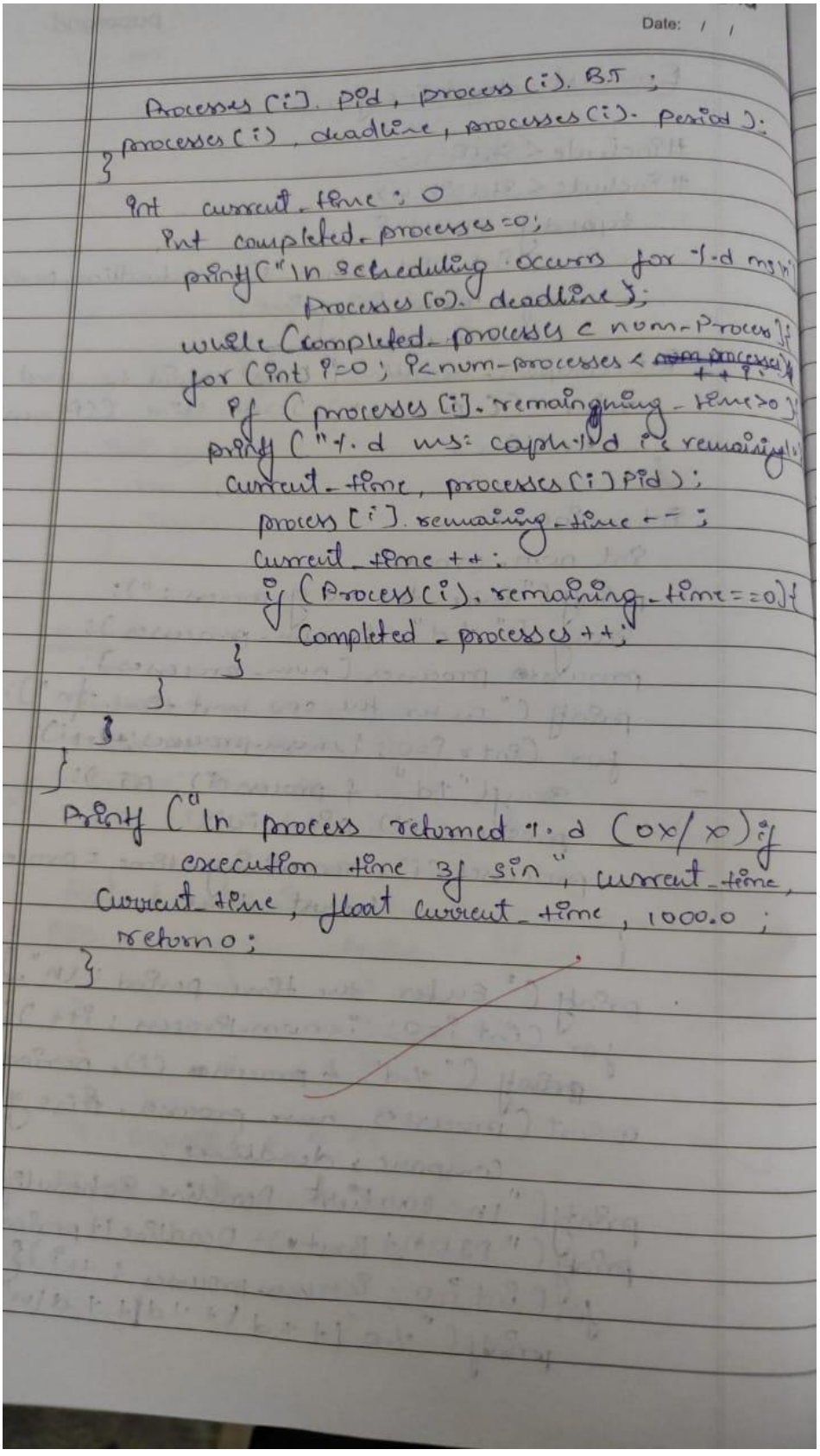

C:\Users\Admin\Downloads\early.exe
Enter the number of processes: 3
Enter the CPU burst times:
2 3 4
Enter the deadlines:
1 2 3
Enter the time periods:
1 2 3
Earliest Deadline Scheduling:
PID Burst Deadline Period
1 2 1 1
2 3 2 2
3 4 3 3
Scheduling occurs for 1 ms
0ms : Task 1 is running.
1ms : Task 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.

papergrid

Date: / /

Earliest - deadline first :

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



Date: / /

```
Processes[i].Pid, process(i).BT ;
processes(i), deadline, processes(i).Period );
}

int current_time = 0
int completed_processes=0;
printf("In scheduling occurs for %d ms\n",
        Processes[0].deadline);
while(completed_processes < num_Process){
for(int i=0; i<num_processes; ++i){
  if(processes[i].remaining_time>0){
  printf("%d ms: copy%d is remaining\n",
   current_time, processes[i].Pid);
    process[i].remaining_time --;
    current_time ++;
    if(Process(i).remaining_time==0){
      completed_processes ++;
    }
  }
}
}
Printf("In process returned %d (0x/x);
    execution time 3f sin", current_time,
Current_time, float current_time, 1000.0 ;
return 0;
}
```

Date: / /

Earliest Deadline first

Output :

```
Enter the no.of processes : 3
Enter the cpu burst time:
  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 4 is running
4ms : Task 1 is running
5ms : Task 3 is running
6ms : Task 2 is running
7ms : Task 3 is running
8ms : Task 3 is running.
```

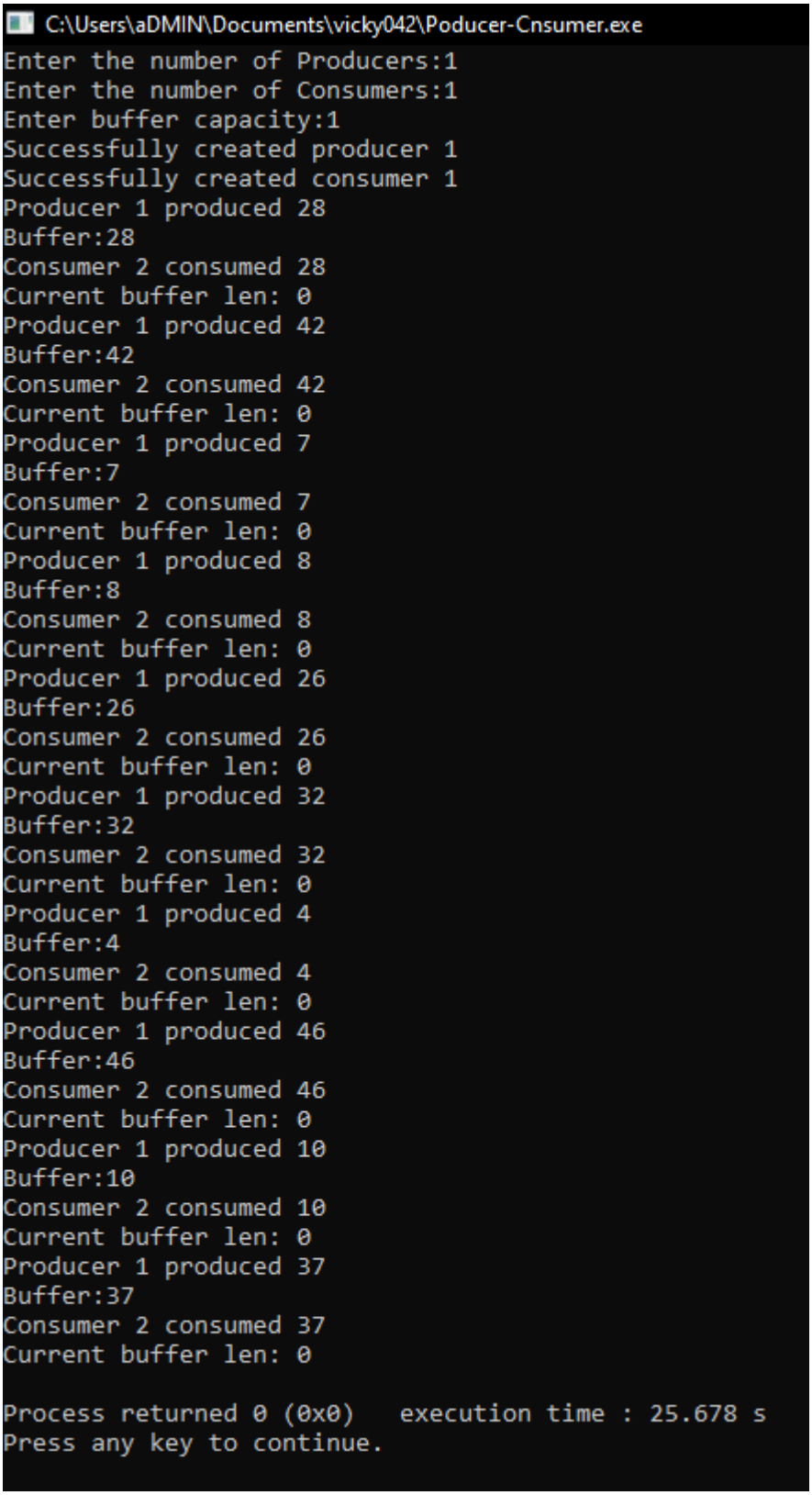
Program 5:

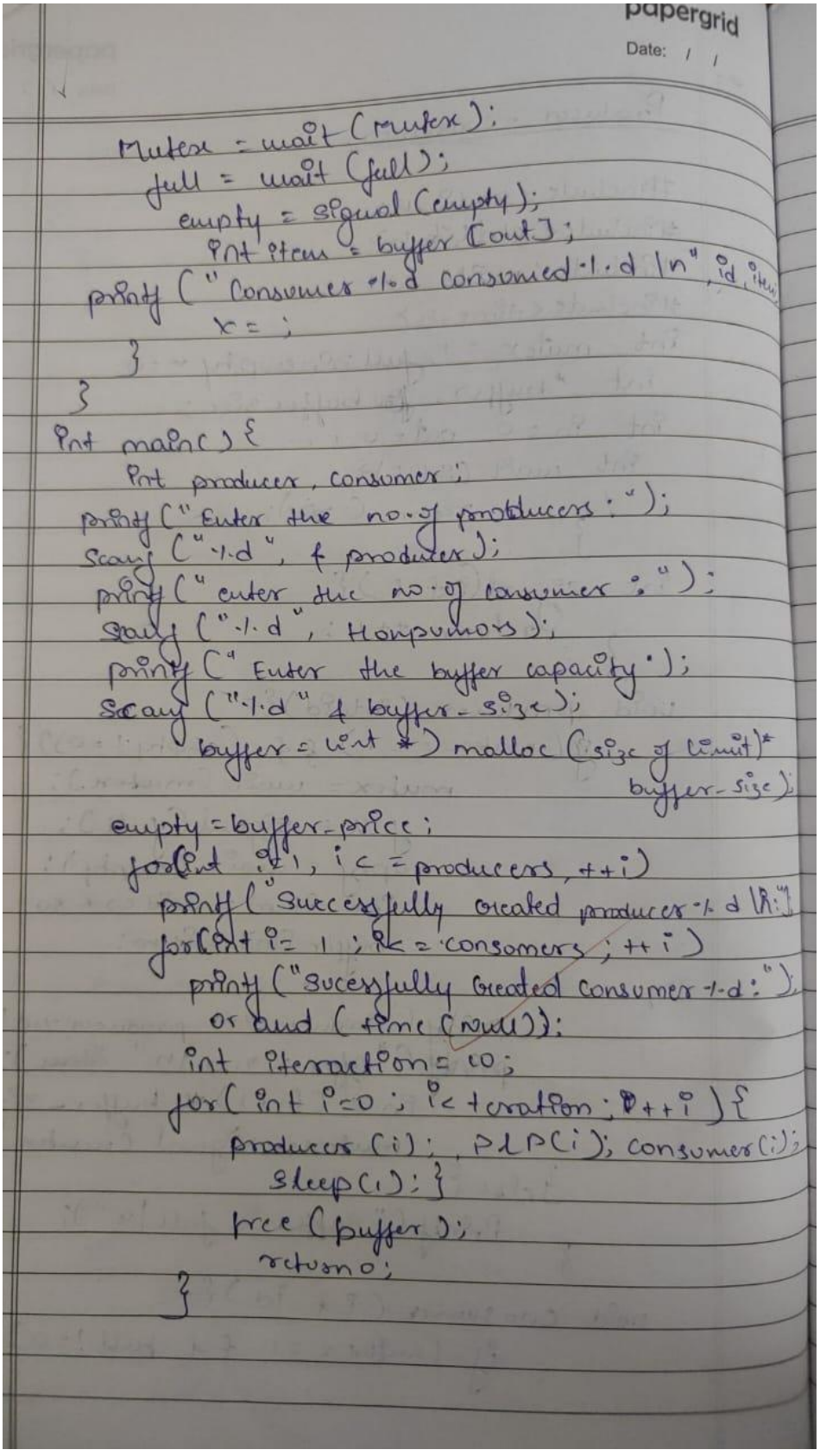
Write a C program to simulate producer-consumer problem using semaphores

```
#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;
```

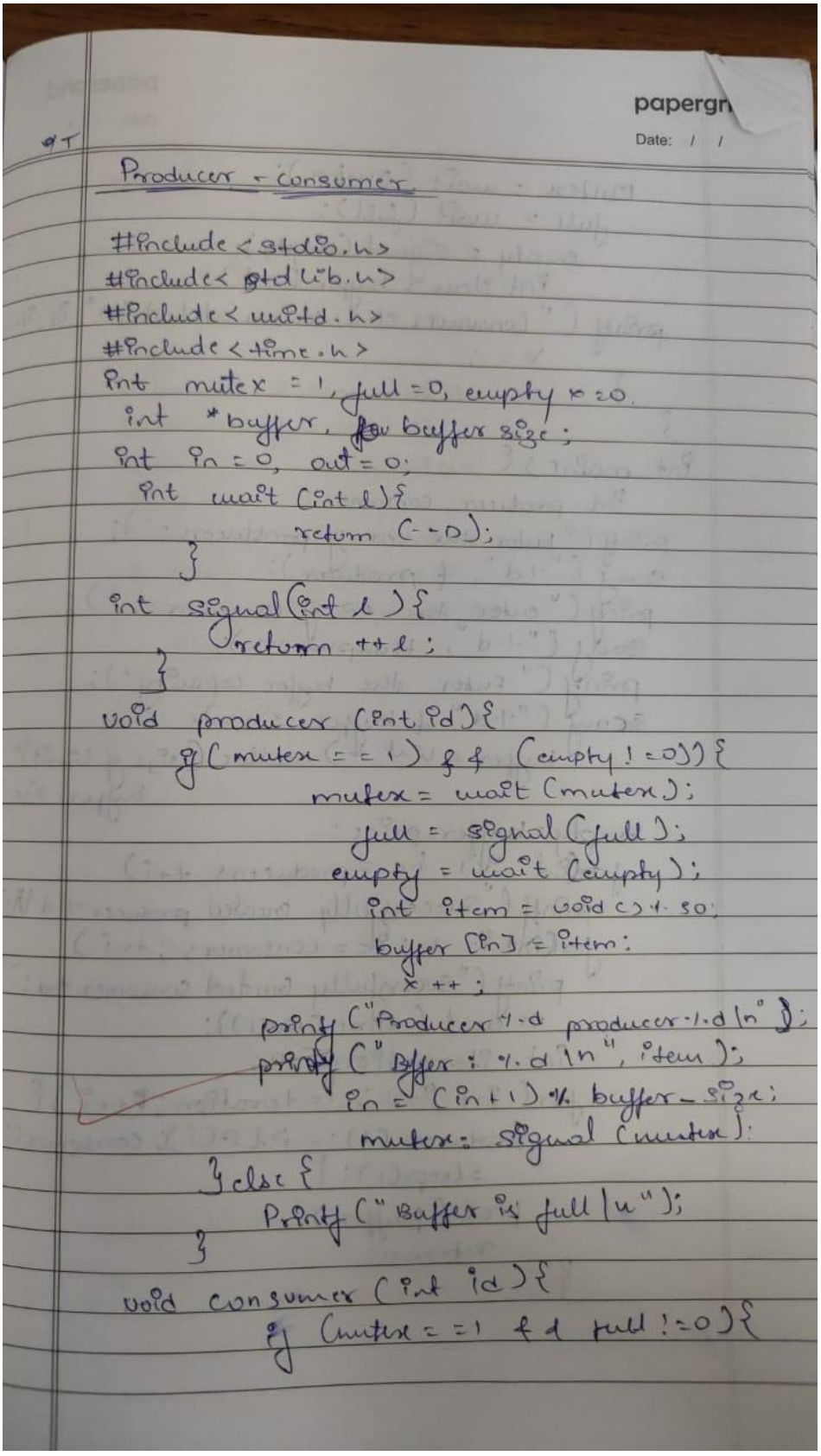
```
        mutex = signal(mutex);
    } else {
        printf("Buffer is full\n");
    }
}

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.

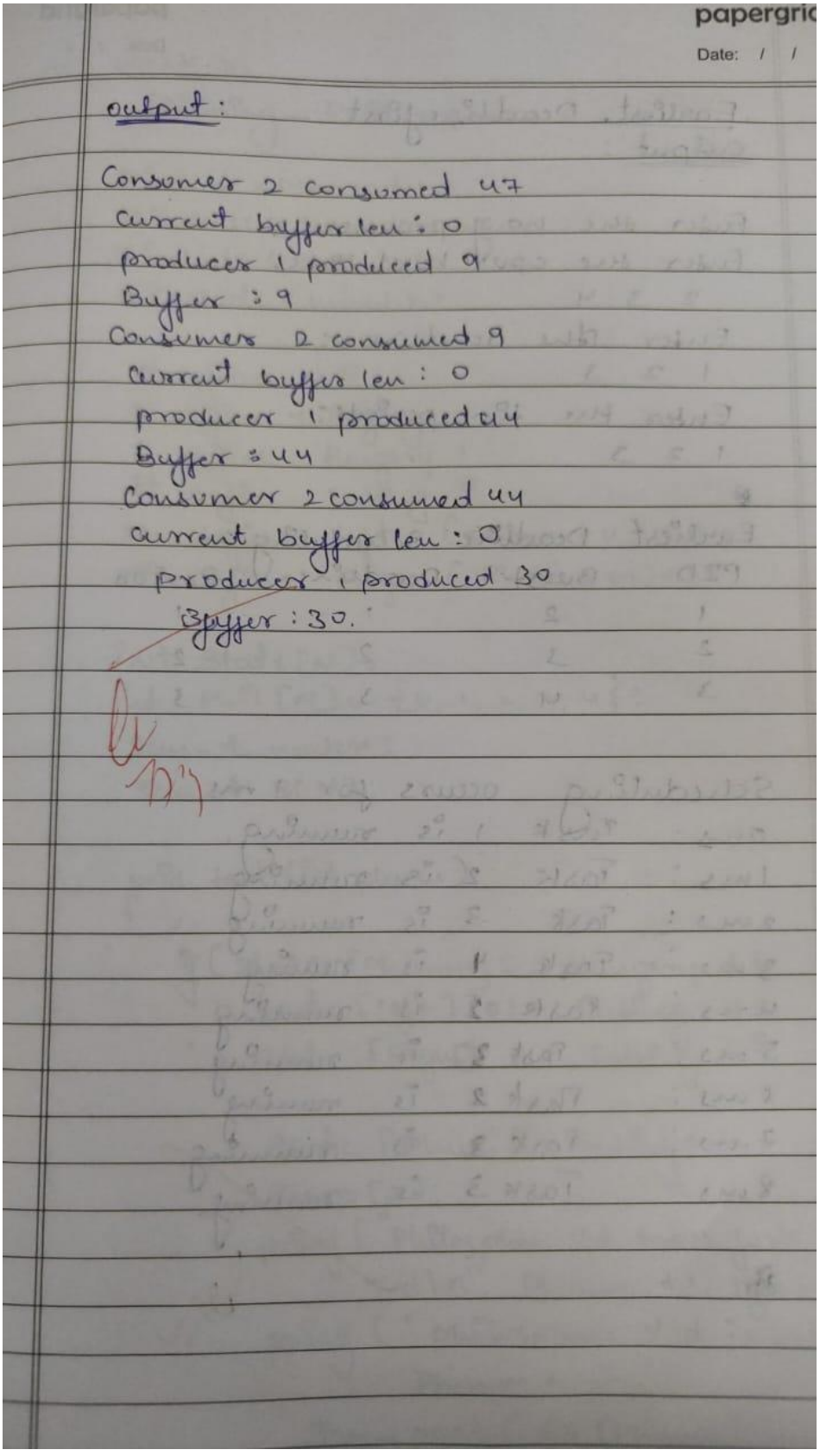


```
Mutex = wait(mutex);
full = wait(full);
empty = signal(empty);
int item = buffer[out];
printf("Consumer %d consumed %d \n", id, item);
x = ;
}
}
int main(){
int producer, consumer;
printf("Enter the no. of producers: ");
scanf("%d", &producer);
printf("enter the no. of consumer: ");
scanf("%d", &consumers);
printf("Enter the buffer capacity");
scanf("%d", &buffer_size);
buffer = (int *) malloc(sizeof(int)*
buffer_size);
empty = buffer_price;
for(int i=1; i<=producers; ++i)
printf("Successfully created producer %d \n");
for(int i=1; i<=consomers; ++i)
printf("Sucessfully Created consumer %d:");
srand(time(NULL));
int iteraction=10;
for(int i=0; i<iteration; ++i){
producer(i); P2P(i); consumer(i);
sleep(1);}
free(buffer);
return 0;
}
```



Producer - consumer

```
#include <stdio.h>
#include <stdlib.h>
#include <wait.h>
#include <time.h>
int mutex = 1, full = 0, empty = 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 ();
            buffer [in] = item;
            x ++;
        printf ("Producer %d producer %d \n");
        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){
```


papergri
Date: / /
output:
Consumer 2 consumed 47
current buffer len : 0
producer 1 produced 9
Buffer : 9
Consumer 2 consumed 9
current buffer len : 0
producer 1 produced 44
Buffer : 44
Consumer 2 consumed 44
current buffer len : 0
producer 1 produced 30
Buffer : 30.

Program 6:

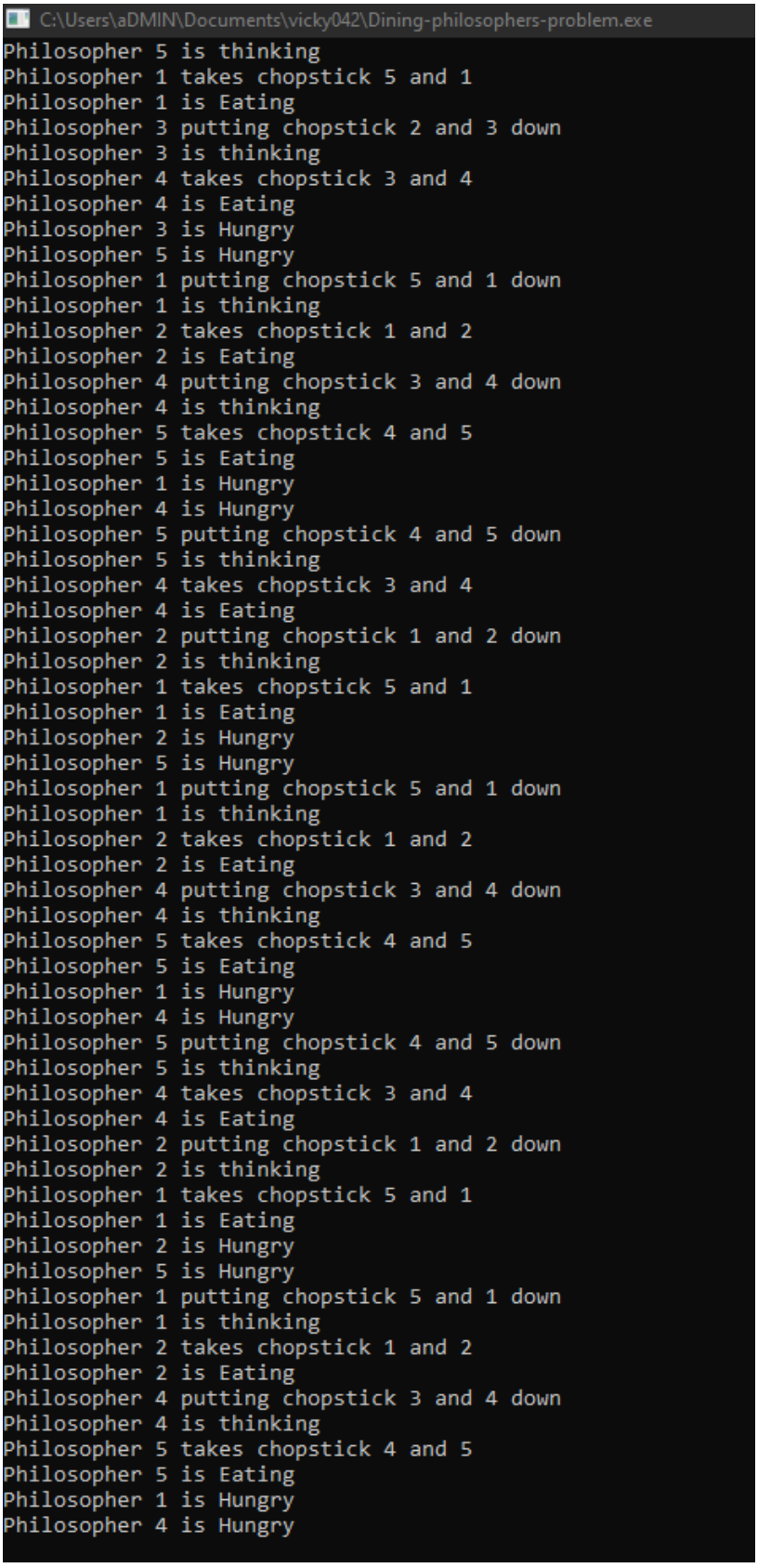
Write a C program to simulate the concept of Dining Philosophers problem.

```
#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;
}
```

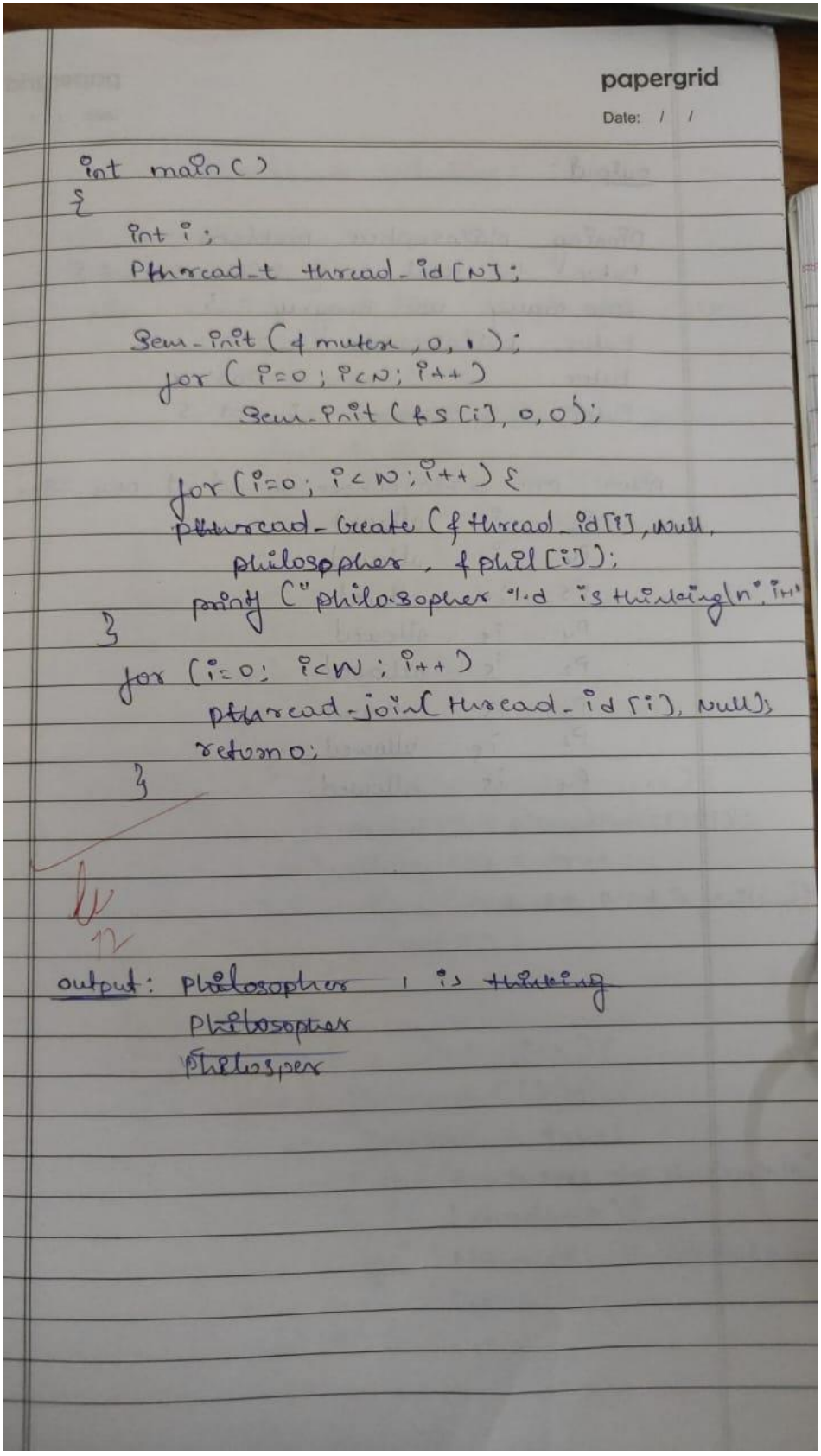

C:\Users\aDMIN\Documents\vicky042\Dining-philosophers-problem.exe
Philosopher 5 is thinking
Philosopher 1 takes chopstick 5 and 1
Philosopher 1 is Eating
Philosopher 3 putting chopstick 2 and 3 down
Philosopher 3 is thinking
Philosopher 4 takes chopstick 3 and 4
Philosopher 4 is Eating
Philosopher 3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

Week-5.

Date: / /

Dinning – Philosophers

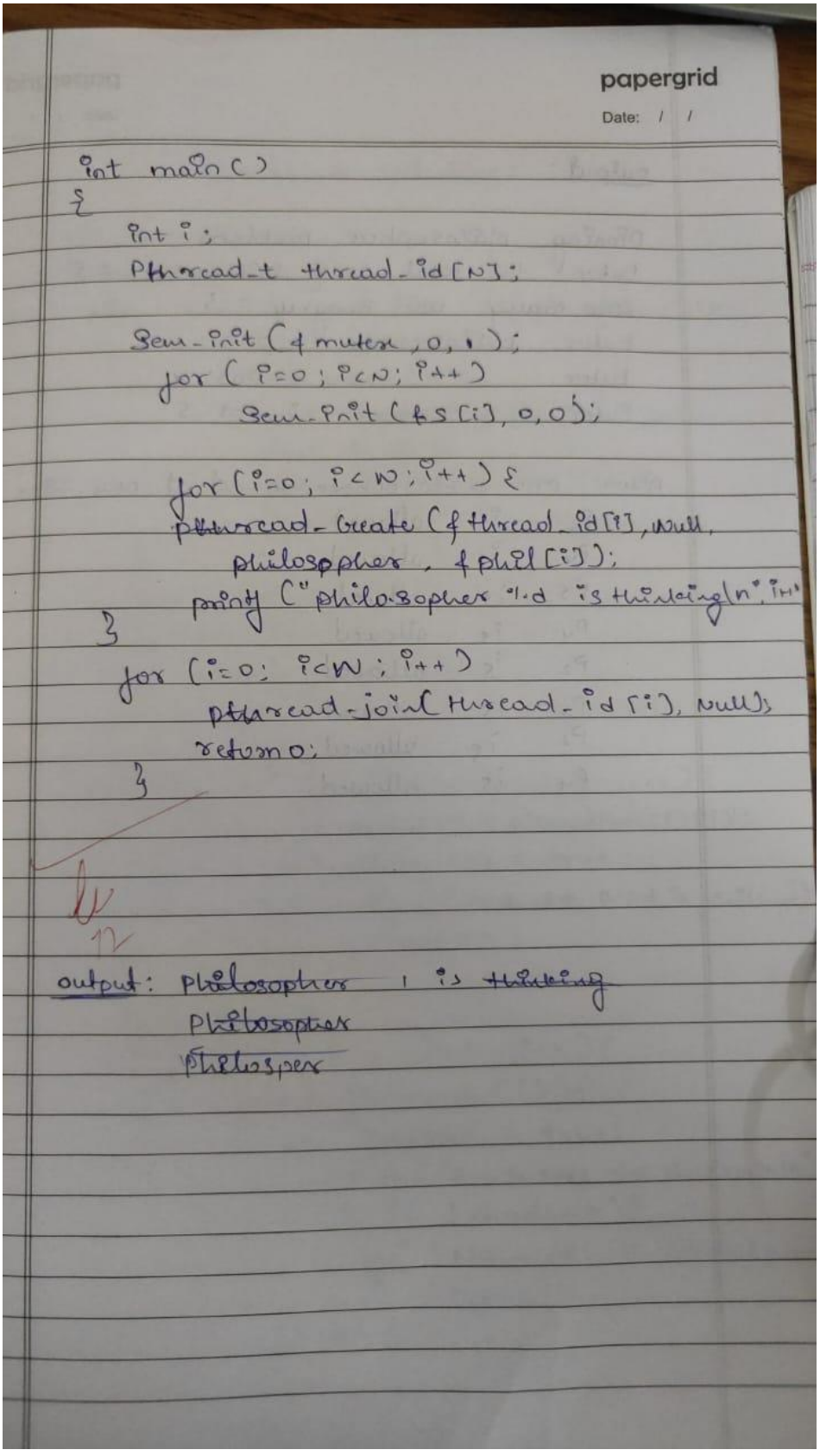
```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <unistd.h>

#define N 5
#define thinking 2
#define Hungary 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] == Hungary &&
        state[left] != Eating &&
        state[Right] != Eating) {

        state[phnum] = Eating;
        sleep(2);
        printf("Philosphes %d takes fork %d and
            %d\n", phnum+1, left+1, phnum+1);
        printf("philosopher %d is Eating\n",
                phnum+1);
        sem_post(&S[phnum]);
    }
}
```



papergrid
Date: / /

```
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;
    }
```

output: Philosopher 1 is thinking
Philosopher
Philosper



papergrid
Date: / /

```
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;
}
```

output: Philosopher 1 is thinking
Philosopher
Philosoper

Date: / /

Output:

Dining philosopher problem
Enter the total no. of philosophers : 5
How many are hungry : 3
Enter philosophers 1 pos : 2
Enter " 2 pos: 4
Enter " 3 pos : 5

Allow one philosophers to eat at any time
P_2 is allowed
P_4 is allowed
P_5 is allowed
P_4 is allowed
P_2 is allowed
P_5 is allowed
P_2 is allowed
P_4 is allowed.

Date: / /

Output:

Dining philosopher problem
Enter the total no. of philosophers : 5
How many are hungry : 3
Enter philosophers 1 pos : 2
Enter " 2 pos: 4
Enter " 3 pos : 5

Allow one philosophers to eat at any time
P_2 is allowed
P_4 is allowed
P_5 is allowed
P_4 is allowed
P_2 is allowed
P_5 is allowed
P_2 is allowed
P_4 is allowed.

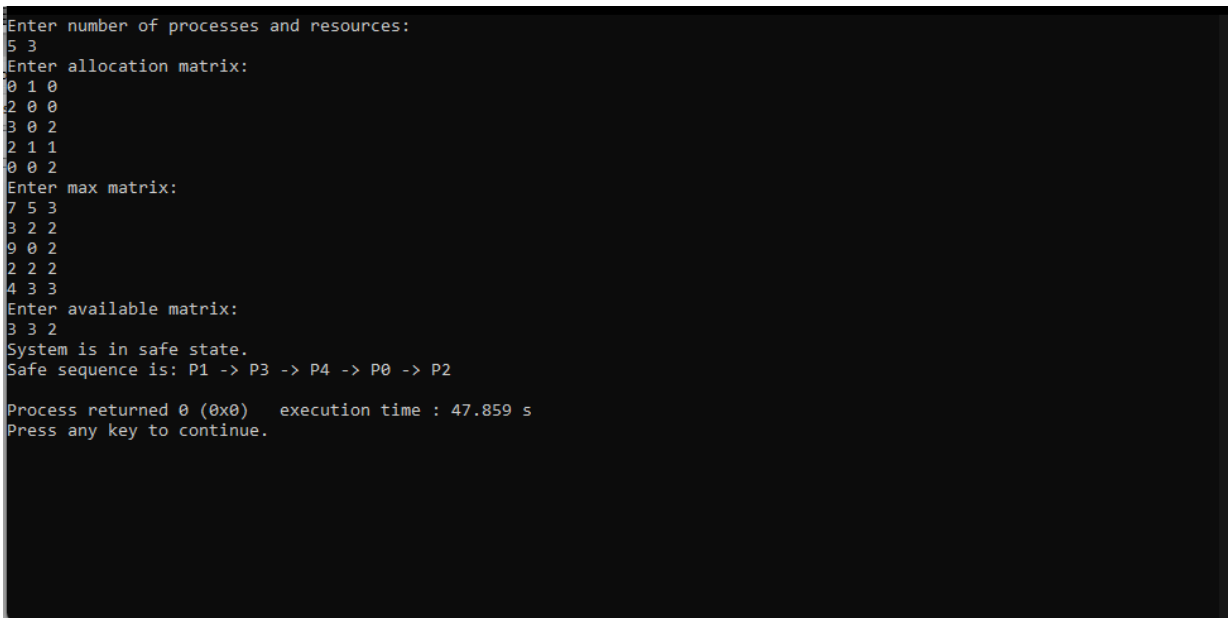
Program 7:

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) {
```

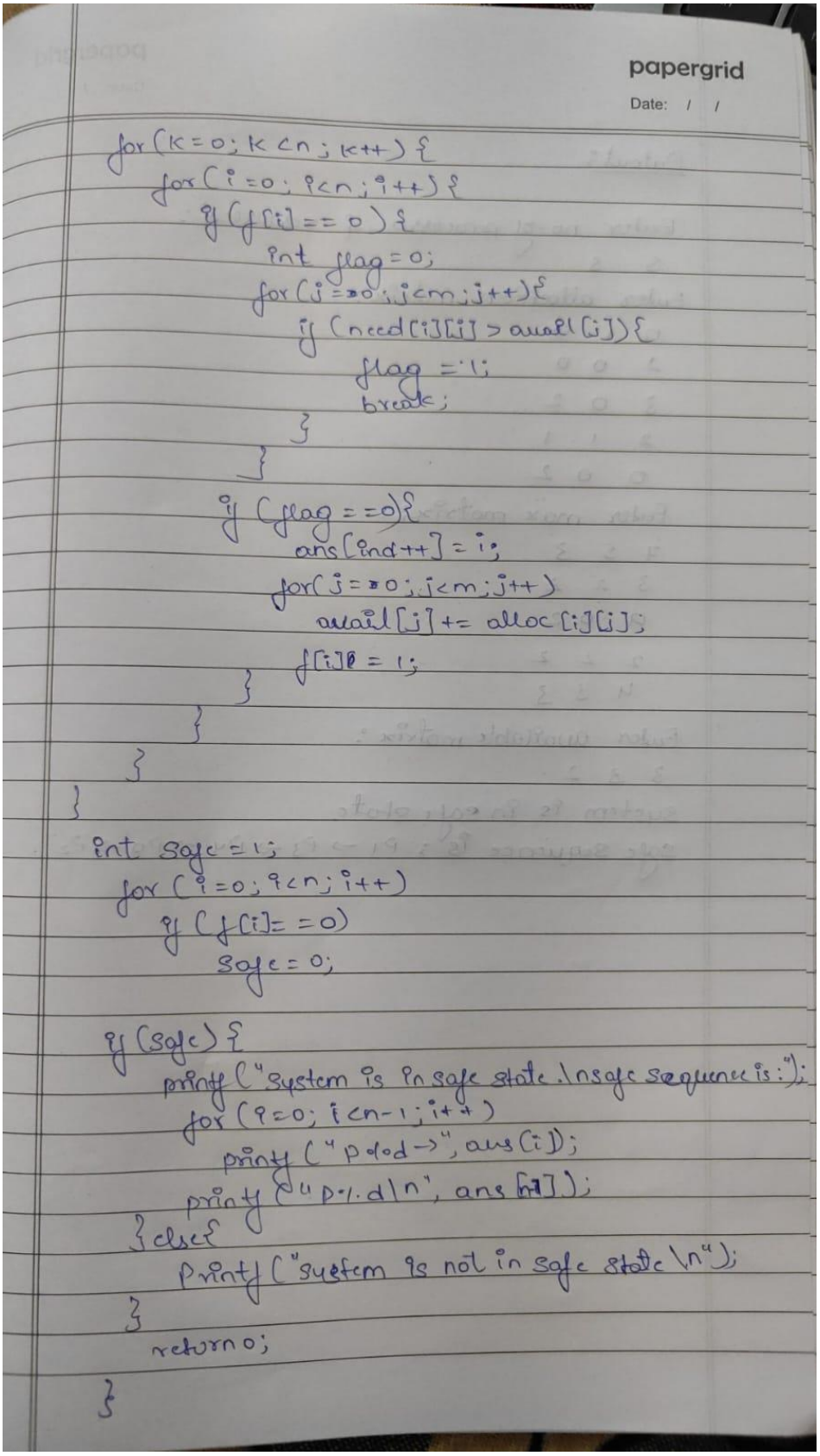
```
        printf("System is in safe state.\nSafe sequence is: ");
        for (i = 0; i < n - 1; i++)
            printf("P%d -> ", ans[i]);
        printf("P%d\n", ans[n - 1]);
    } else {
        printf("System is not in safe state\n");
    }

    return 0;
}
```



Date: / /

Bankers Algorithm.

```
#include <stdio.h>

int main() {
    int n, m, i, j, k;
    printf("Enter no. of processes & 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;
}
```

Date: / /

Output:

```
Enter no. of processes & resources:
5 3
Enter allocation 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 is: P1 -> P3 -> P4 -> P0 -> P2.
```

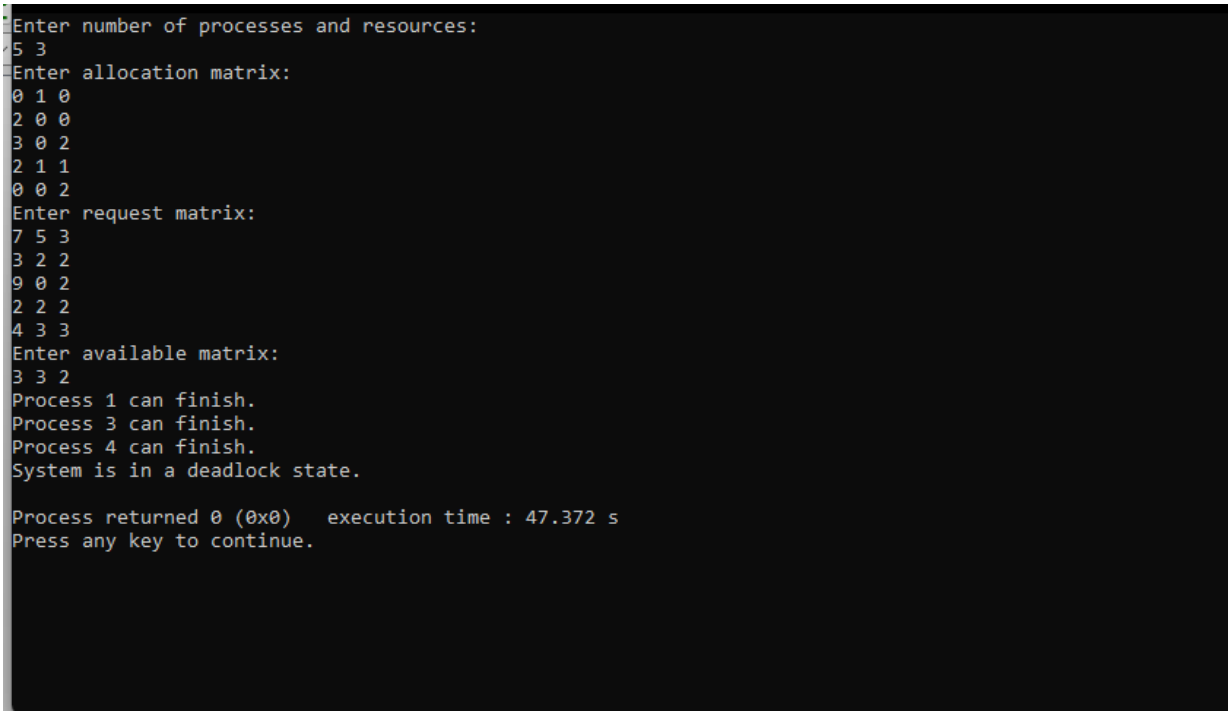
Program 8:

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.

Date: / /

Deadlock Detection:

```
#include <stdio.h>
#include <stdbool.h>
void deadlock_detection (int allocation[i][R],
    int req[i][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;
(count < P)
    bool found = false;
    for (int j=0; j<R; j++){
      if (comp){
        for (int j=0; j<R; j++){
        work[j] += allocation[i][j];
        finish[i] = true;
          pf ("Process p.i.d is finished");
          count++;
        }}}
if (! found){
  for (int i=0; i<P; i++){
    if (! finished[i]){
        deadlock = true;
      printf ("Processes %d deadlock\n");
        if (! deadlock){
          pf (" No deddlock detected process
              complete");
  } else {
```

papergrid

Date: / /

```
int main(){
    int allocation(P)(R)= 3
        { 0, 1, 0}
        { 2, 0, 0}
        { 3, 0, 0}
        { 2, 1, 1}
        { 0, 0, 1}
    int request(P)(R)= 3
        { 0, 0, 0}
        { 0, 0, 0}
        { 0, 0, 0}
        { 0, 0}
    int available(R) = {0,0,0};
    deadlock detection(allocation, request,
                       available).
        return;
    }
```

output :

Process P_0 is finished
Process P_1 is finished
P_2 is finished
P_3 is finished
P_4 is finished
P_1 is finished

No deadlock detected

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
```

papergrid

Date: / /

Memory management.

Write a scheme (Best fit).

```
#include <stdio.h>
struct block {
int block_no;
int block_size;
int is_free : 3;
struct file {
    int file_no;
    int file_size;
};
void bestfit (struct block blocks[], int n_blocks,
        struct file files[], int n_files)
{
printf("In Memory management scheme");
printf("File no\t file size | block_no\n");
for (int i=0; i<n_files; i++)
  {
    int best_file_block = -1;
    int min_fragment = 10000;
  for(int j=0; j<n; j++){
    int fragment = blocks[j] - block size_files
    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\t\t %d\t\t %d", 
    files(i)- file no;
    files(i)- file size;
    blocks (best_fit_block)- block_no
    blocks (best_fit_block) block_size
        min_fragment;
```

Date: / /

```
} else {
    pf("%d \t \t N/A \t \t %d \t \t");
    files(i) - file no;
    files(i) - file size;
}
}

int main() {
  int n_blocks, n_files;
  pf("%d", & n_blocks);
  pf("Enter the no. of files");
  sf("%d" & n_files);
  struct block blocks (n_blocks);
    struct block files (n_files);
      for (int i=0; i<n; i++)
  {
    blocks(i) - blocks_no = (i);
    pf("Enter the size of blocks");
    sf("%d" & blocks(i), block_size");
      blocks(i) - is free = 1;
  }
  for (int i=0; i<n_files; i++) {
      files(i) file no = 1;
  pf("Enter size of file %d");
}
  for(int i=0; i<n_blocks; i++)
    {
      blocks_is_free = 1;
  }
```

output : output :

Enter no. of blocks : 5
Enter no. of files : 4
Enter size of blocks : 100
Enter size of blocks : 200
Enter Size of blocks : 400

Date: / /

memory	Fullsize	Block no	Block size
1	212	3	300
2	146	4	200
3	426	2	500
4	404	N/A	N/A.

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;
}
```

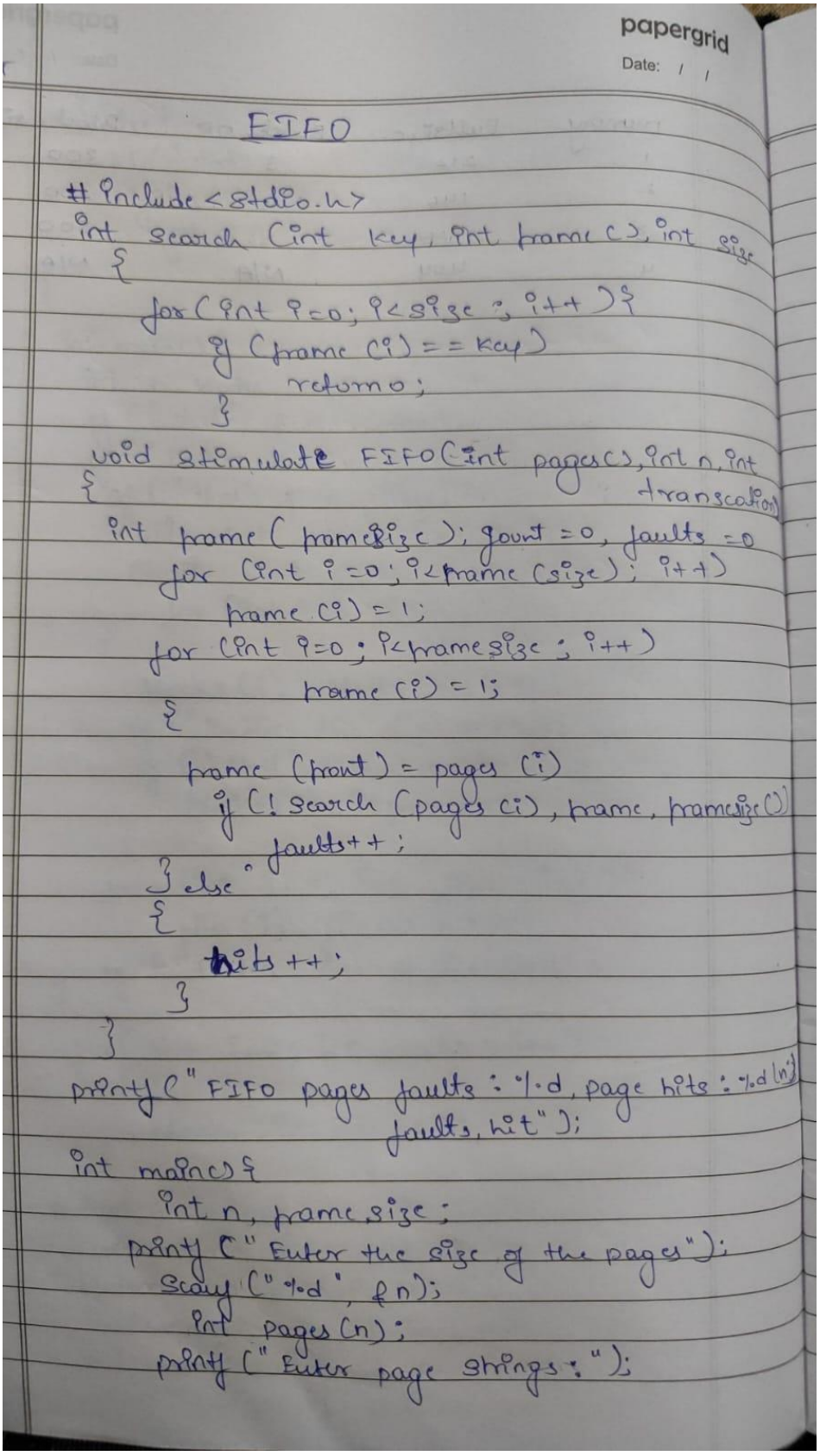
```
Enter the size of the pages:
7
Enter the page strings:
1303563
Enter the no of page frames:
3
LU Page Faults: 5, Page Hits: 2
```

→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;
}
```

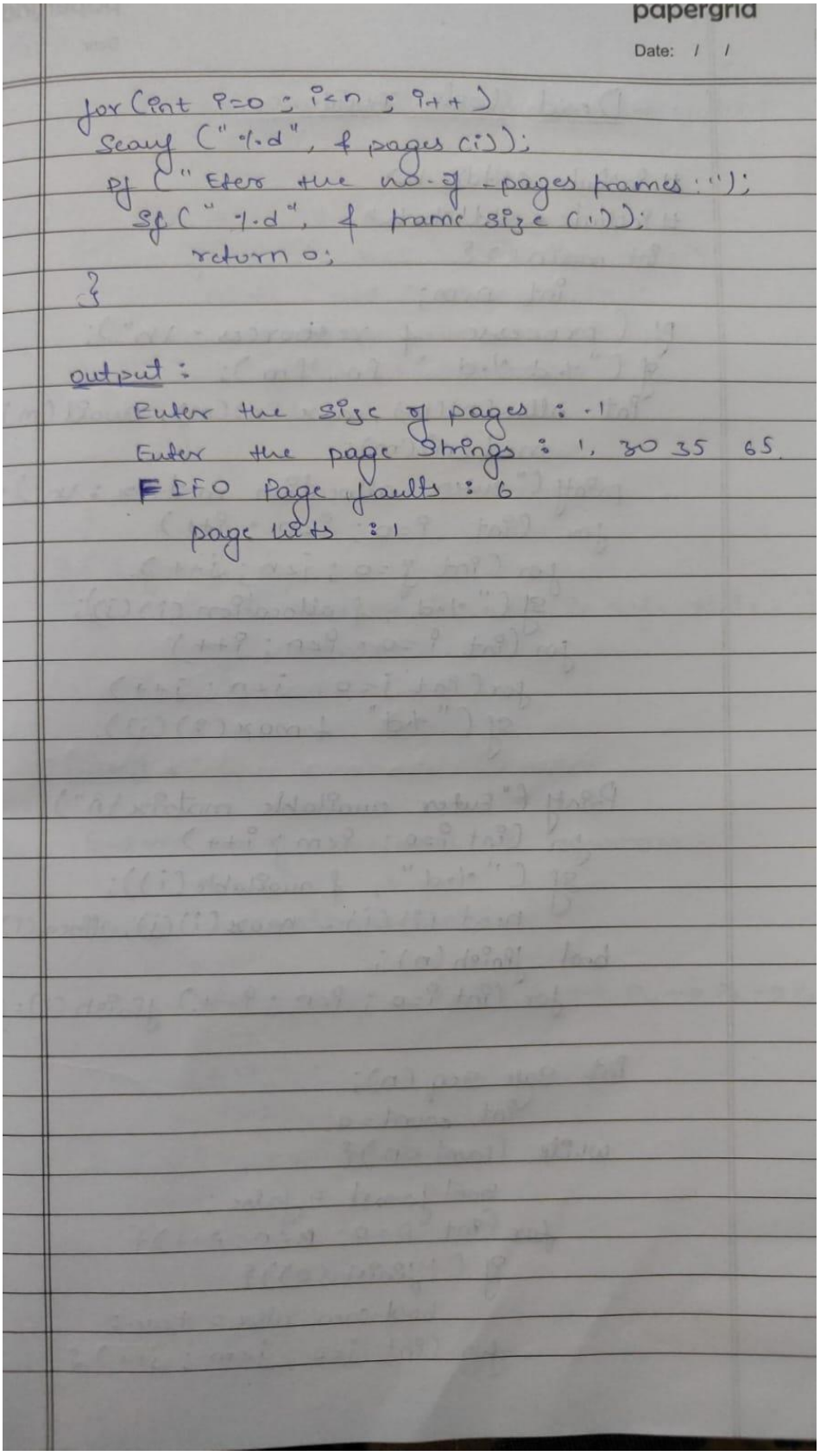
```
Enter the size of the pages:
7
Enter the page strings:
13035631
Enter the no of page frames:
3
Optimal Page Faults: 6, Page Hits: 1
```



papergrid
Date: / /

FIFO

```
#include <stdio.h>
int search (int key, int frame[], int size
{
    for (int i=0; i<size ; i++ ){
        if (frame[i] == key)
            return 0;
        }
void simulate FIFO(int pages[], int n, int
{                                   transcation)
    int frame[ framesize ]; count =0, faults =0
    for (int i =0; i<frame(size); i++)
        frame[i] = 1;
    for (int i=0 ; i<framesize ; i++)
            frame[i] = 1;
    {
        frame [front] = pages [i]
        if (! search (pages[i], frame, framesize())
            faults++ ;
    } else
    {
        hits ++;
    }
}
printf (" FIFO pages faults : %d, page hits : %d \n",
                        faults, hit" );
int main() {
    int n, frame size ;
    printf (" Enter the size of the pages");
    scanf ("%d", &n);
    int pages (n);
    printf (" Enter page strings : ");
```



papergrid

Date: / /

```
for(int i=0 ; i<n ; i++)
scanf("%d", &pages[i]);
printf("Enter the no. of pages frames:");
scanf("%d", &frame_size[i]);
    return 0;
}
```

output :

Enter the size of pages : 1

Enter the page strings : 1, 30 35 65.

FIFO Page faults : 6

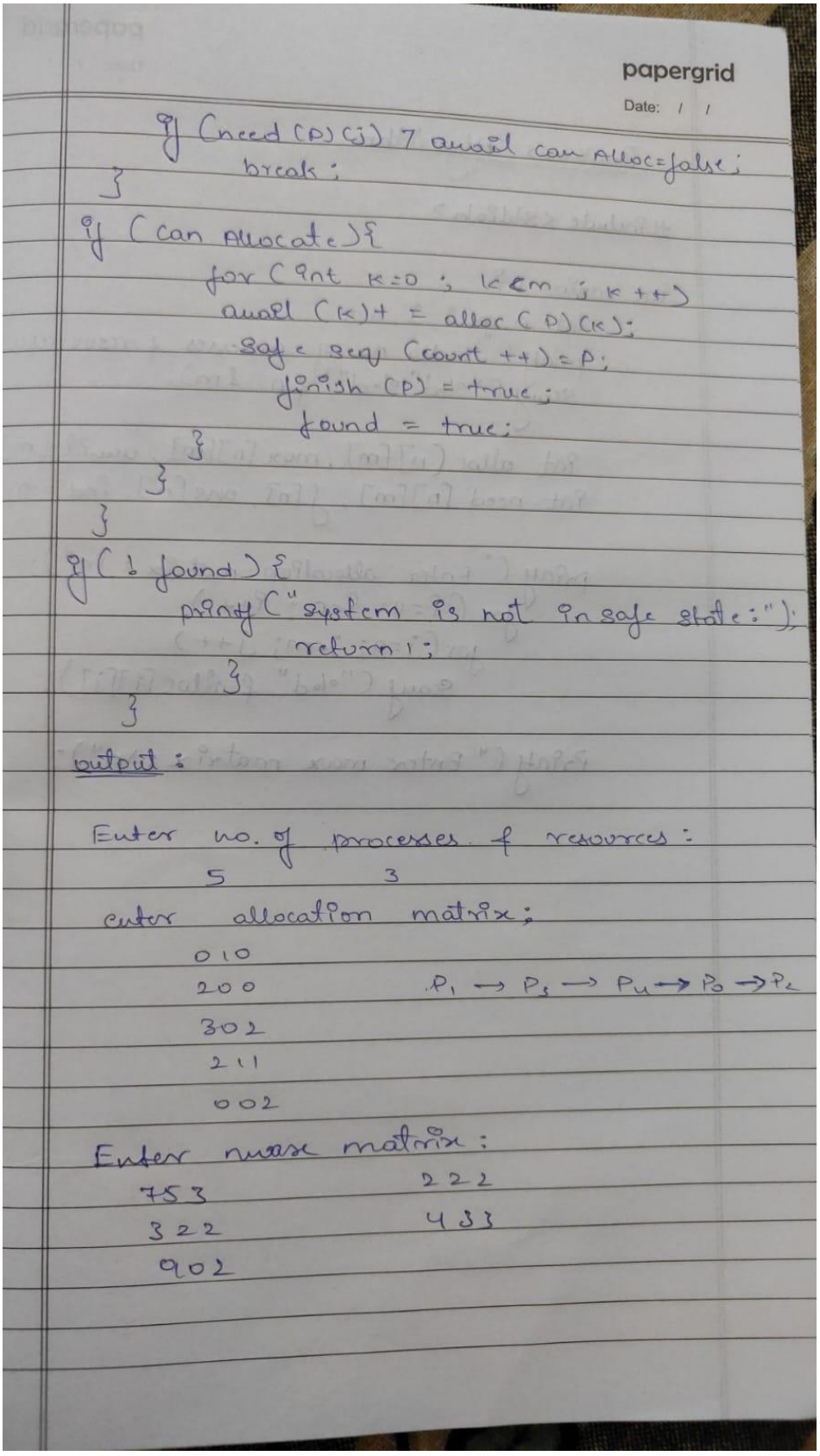
page hits : 1

papergrid

Date: / /

Dead lock Avoidance.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int n,m;
printf("processes & resources : \n");
scanf("%d %d ", &n, &m);
int alloc(n)(m), max(n)(m), avail(m),
    need(n)(m);
printf("enter allocation matrix : \n");
for(int i=0; i<n; i++)
    for(int j=0; j<n; j++)
    scanf("%d", &allocation(i)(j));
for(int i=0; i<n; i++)
    for(int j=0; j<n; j++)
    scanf("%d", &max(i)(j));

Printf("Enter available matrix \n");
for(int i=0; i<m; i++)
    scanf("%d", &available(i));
    need(i)(j) = max(i)(j), alloca(i)(j);
bool finish(n);
    for(int i=0; i<n; i++) finish(i)=false;

int safe seq(n);
    int count=0;
while(count<n){
        bool found = false;
    for(int p=0; p<n; p++){
        if(!finish(p)){
            bool can allo = true;
    for(int j=0; j<m; j++){
```



```
        if (need[p][j] > avail) canAlloc=false;
            break;
    }
    if (canAllocate){
        for (int k=0; k<em; k++)
            avail[k] += alloc[p][k];
            safeSeq[count++]=p;
            finish[p] = true;
            found = true;
        }
    }
}
if (!found){
    printf("system is not in safe state:");
        return 1;
    }
}
```

output :

```
Enter no. of processes & resources :
5        3
enter allocation 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
```

$P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_0 \rightarrow P_2$