VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT

on

# OPERATING SYSTEMS

Submitted by

SUSHRAVYA R (1WA23CS004)

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 Sushravya R (1WA23CS004), 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.

Dr.Seema Patil Dr. Kavitha Sooda

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

Index Sheet

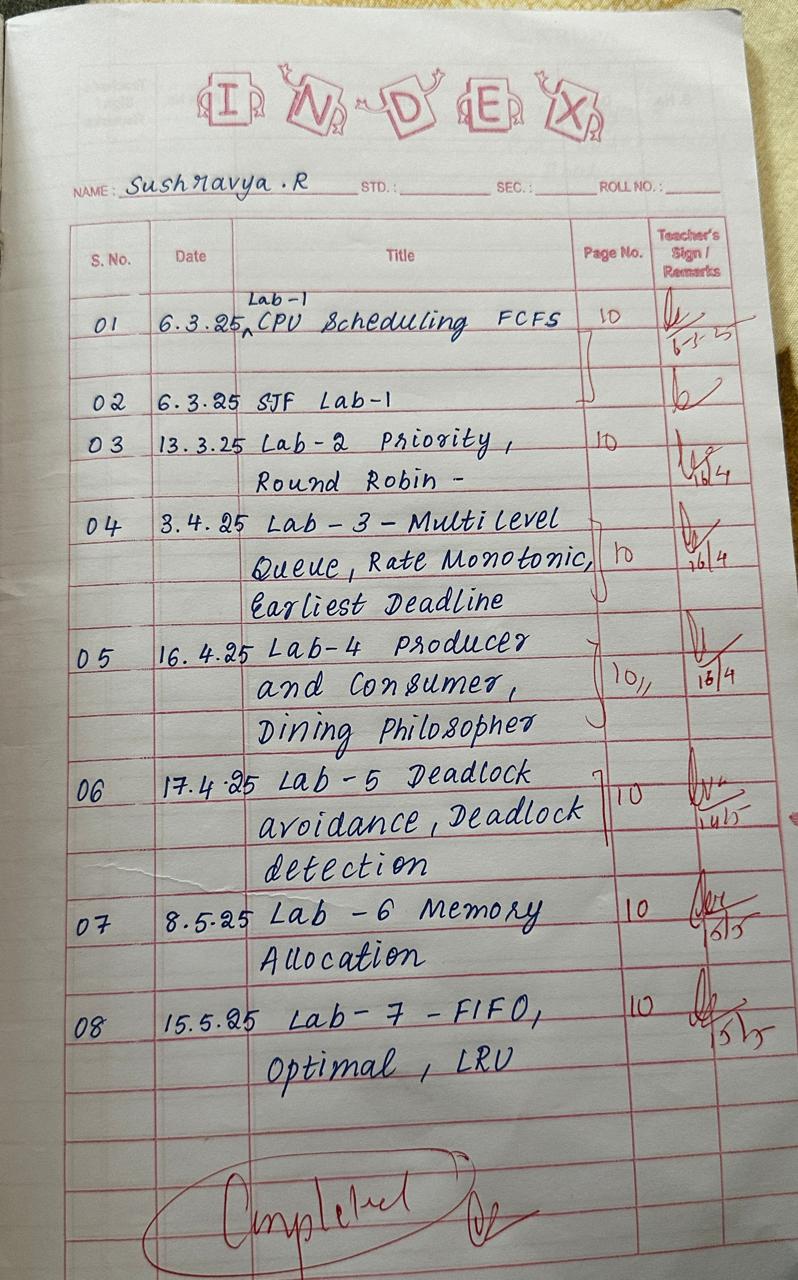
|  |  |  |
| --- | --- | --- |
| Sl.  No. | Experiment Title | Page No. |
| 1. | Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time.  →FCFS  → SJF (pre-emptive & Non-preemptive) | 1-10 |
| 2. | Write a C program to simulate the 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) | 11-19 |
| 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. | 20-24 |
| 4. | Write a C program to simulate Real-Time CPU Scheduling algorithms:   1. Rate- Monotonic 2. Earliest-deadline First 3. Proportional scheduling | 24-34 |
| 5. | Write a C program to simulate producer-consumer problem using semaphores | 35-38 |
| 6. | Write a C program to simulate the concept of Dining Philosophers problem. | 39-43 |
| 7. | Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance. | 44-47 |
| 8. | Write a C program to simulate deadlock detection | 48-51 |
| 9. | Write a C program to simulate the following contiguous memory allocation techniques a) Worst-fit b) Best-fit  c) First-fit | 52-56 |
| 10. | Write a C program to simulate page replacement algorithms a) | 57-63 |

FIFO

|  |  |  |
| --- | --- | --- |
|  | 1. LRU 2. Optimal |  |

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

Question:

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)

Code:

=>FCFS:

#include<stdio.h>

void sort(int proc\_id[],int at[],int bt[],int n)

{

int min=at[0],temp=0; for(int i=0;i<n;i++)

{ min=at[i]; for(int j=i;j<n;j++)

{ if(at[j]<min)

{ temp=at[i]; at[i]=at[j]; at[j]=temp; temp=bt[j]; bt[j]=bt[i]; bt[i]=temp; temp=proc\_id[i]; proc\_id[i]=proc\_id[j]; proc\_id[j]=temp;

}

}

}

}

void main()

{

int n,c=0; printf("Enter number of processes: "); scanf("%d",&n); int proc\_id[n],at[n],bt[n],ct[n],tat[n],wt[n]; double avg\_tat=0.0,ttat=0.0,avg\_wt=0.0,twt=0.0; for(int i=0;i<n;i++) proc\_id[i]=i+1; printf("Enter arrival times:\n"); for(int i=0;i<n;i++) scanf("%d",&at[i]); printf("Enter burst times:\n"); for(int i=0;i<n;i++) scanf("%d",&bt[i]);

sort(proc\_id,at,bt,n); //completion time for(int i=0;i<n;i++)

{ if(c>=at[i]) c+=bt[i]; else c+=at[i]-ct[i-1]+bt[i]; ct[i]=c;

}

//turnaround time for(int i=0;i<n;i++) tat[i]=ct[i]-at[i]; //waiting time for(int i=0;i<n;i++) wt[i]=tat[i]-bt[i];

printf("FCFS scheduling:\n"); printf("PID\tAT\tBT\tCT\tTAT\tWT\n"); for(int i=0;i<n;i++) printf("%d\t%d\t%d\t%d\t%d\t%d\n",proc\_id[i],at[i],bt[i],ct[i],tat[i],wt[i]);

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

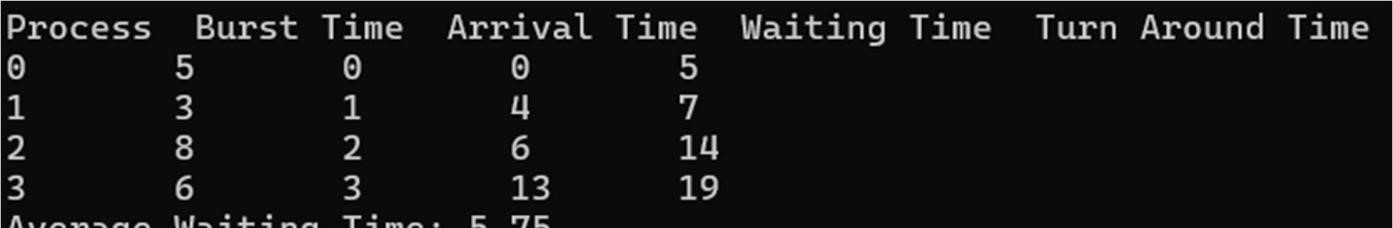
{ ttat+=tat[i];twt+=wt[i];

}

avg\_tat=ttat/(double)n; avg\_wt=twt/(double)n; printf("\nAverage turnaround time:%lfms\n",avg\_tat); printf("\nAverage waiting time:%lfms\n",avg\_wt);

}

Result:



Insert signed Observation Book scan accordingly to programs

=>SJF(Preemptive):

#include <stdio.h>

#include <limits.h>

struct Process {

int pid, bt, at, wt, tat, rt, ct, remaining\_bt;

};

void sortByArrival(struct Process p[], int n) {

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

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

if (p[j].at > p[j + 1].at) {

struct Process temp = p[j];

p[j] = p[j + 1];

p[j + 1] = temp;

}

}

}

}

void sjfPreemptive(struct Process p[], int n) {

sortByArrival(p, n);

int completed = 0, currentTime = 0, shortest = -1;

int totalWT = 0, totalTAT = 0, totalRT = 0;

int isFirstResponse[n];

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

p[i].remaining\_bt = p[i].bt;

isFirstResponse[i] = 1;

}

while (completed < n) {

int minBT = INT\_MAX;

shortest = -1;

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

if (p[i].at <= currentTime && p[i].remaining\_bt > 0 && p[i].remaining\_bt < minBT) {

minBT = p[i].remaining\_bt;

shortest = i;

}

}

if (shortest == -1) {

currentTime++;

continue;

}

if (isFirstResponse[shortest]) {

p[shortest].rt = currentTime - p[shortest].at;

isFirstResponse[shortest] = 0;

}

p[shortest].remaining\_bt--;

currentTime++;

if (p[shortest].remaining\_bt == 0) {

p[shortest].ct = currentTime;

p[shortest].tat = p[shortest].ct - p[shortest].at;

p[shortest].wt = p[shortest].tat - p[shortest].bt;

totalWT += p[shortest].wt;

totalTAT += p[shortest].tat;

totalRT += p[shortest].rt;

completed++;

}

}

printf("\nPID\tAT\tBT\tCT\tWT\tTAT\tRT\n");

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

printf("%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].ct, p[i].wt, p[i].tat, p[i].rt);

}

printf("\nAverage WT: %.2f", (float)totalWT / n);

printf("\nAverage TAT: %.2f", (float)totalTAT / n);

printf("\nAverage RT: %.2f\n", (float)totalRT / n);

}

int main() {

int n;

printf("Enter number of processes: ");

scanf("%d", &n);

struct Process p[n];

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

printf("Enter AT & BT for P%d: ", i + 1);

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

p[i].pid = i + 1;

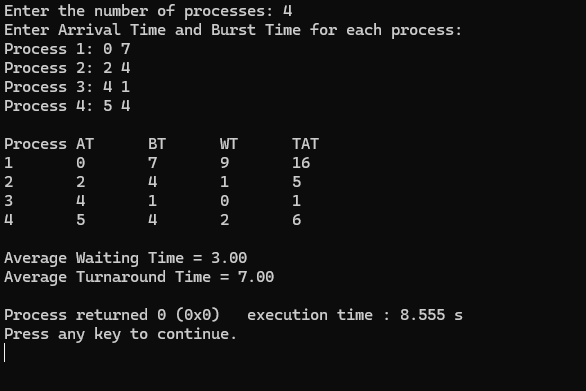
p[i].wt = p[i].tat = p[i].rt = p[i].ct = 0;

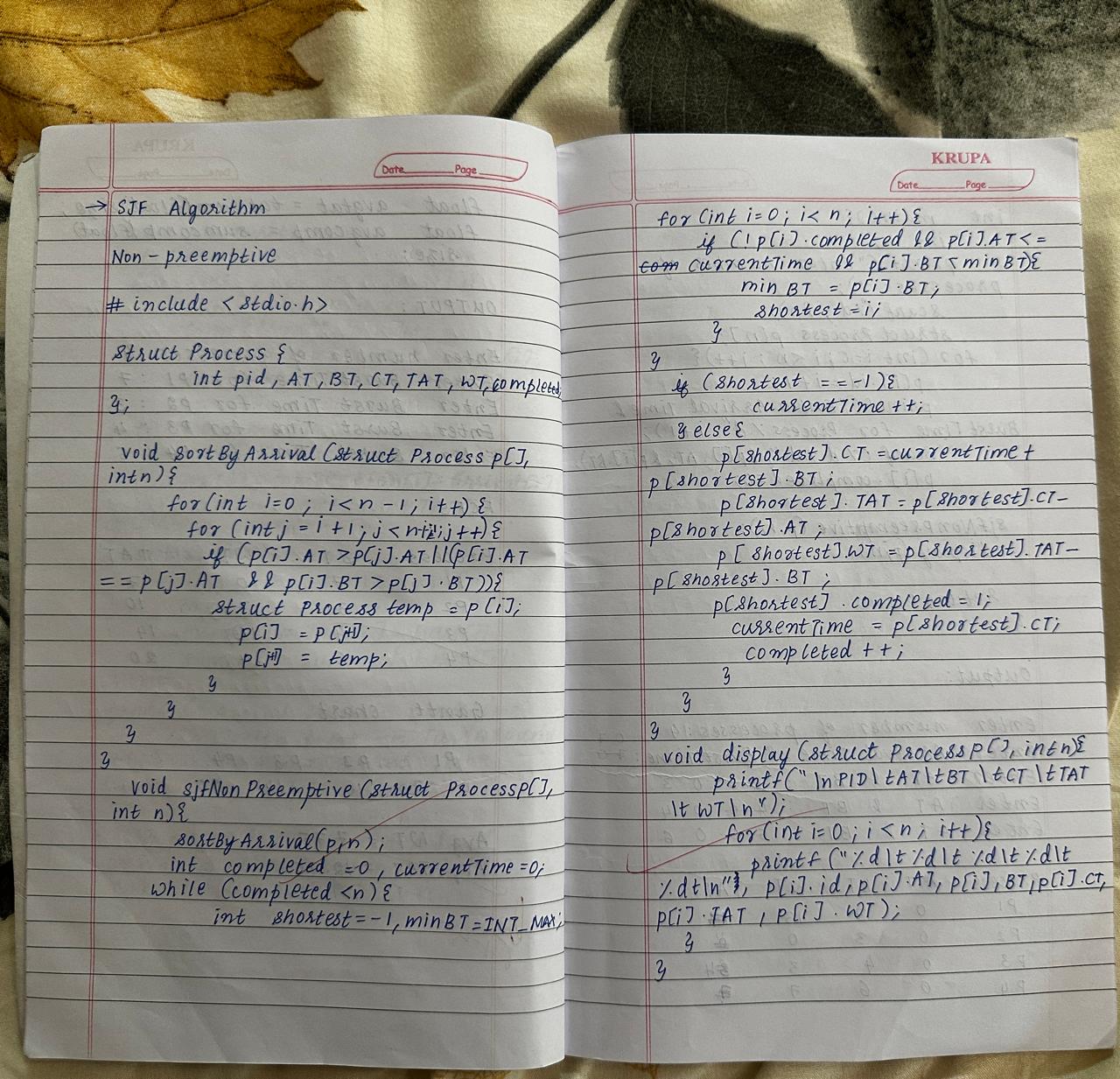
}

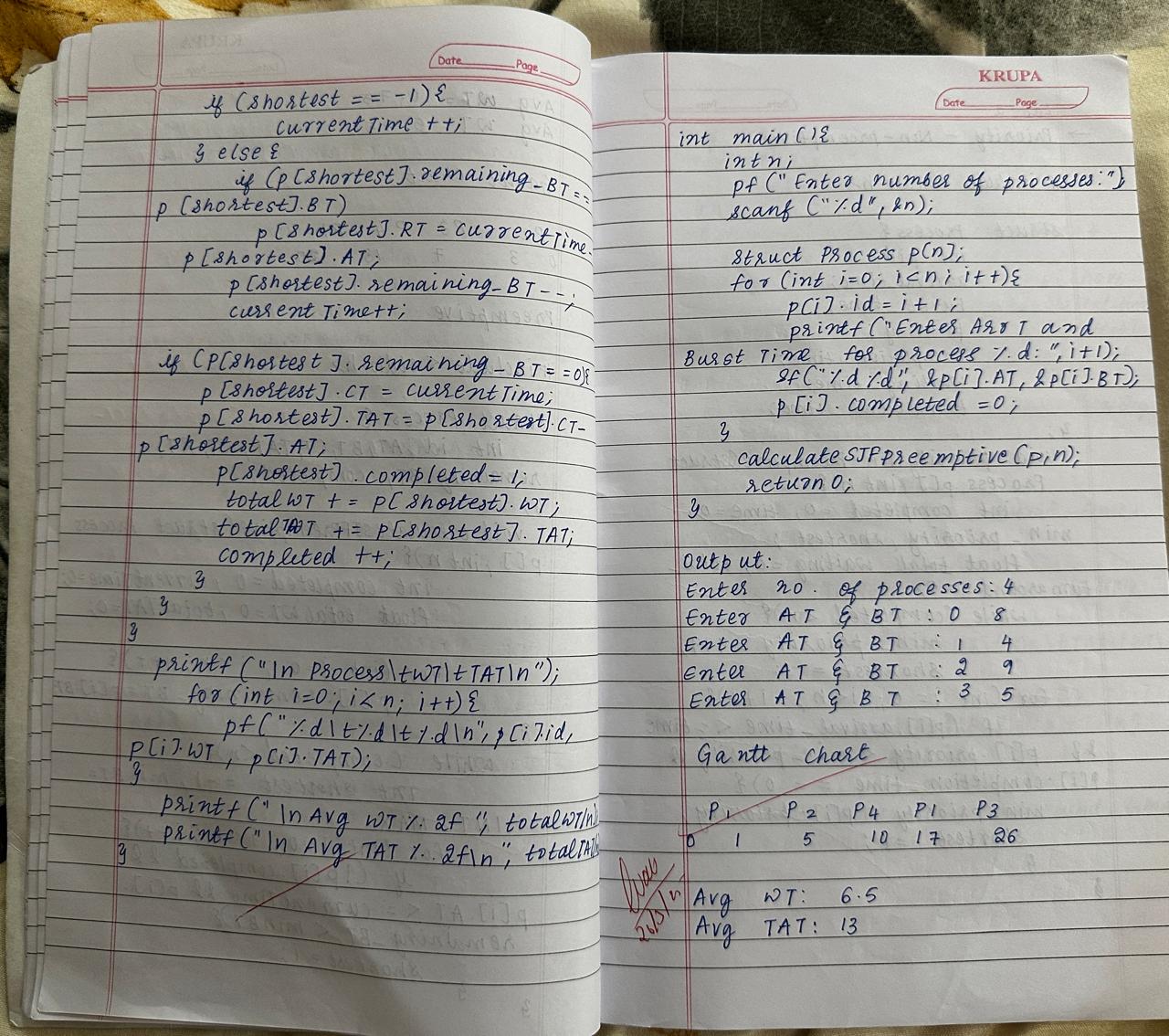
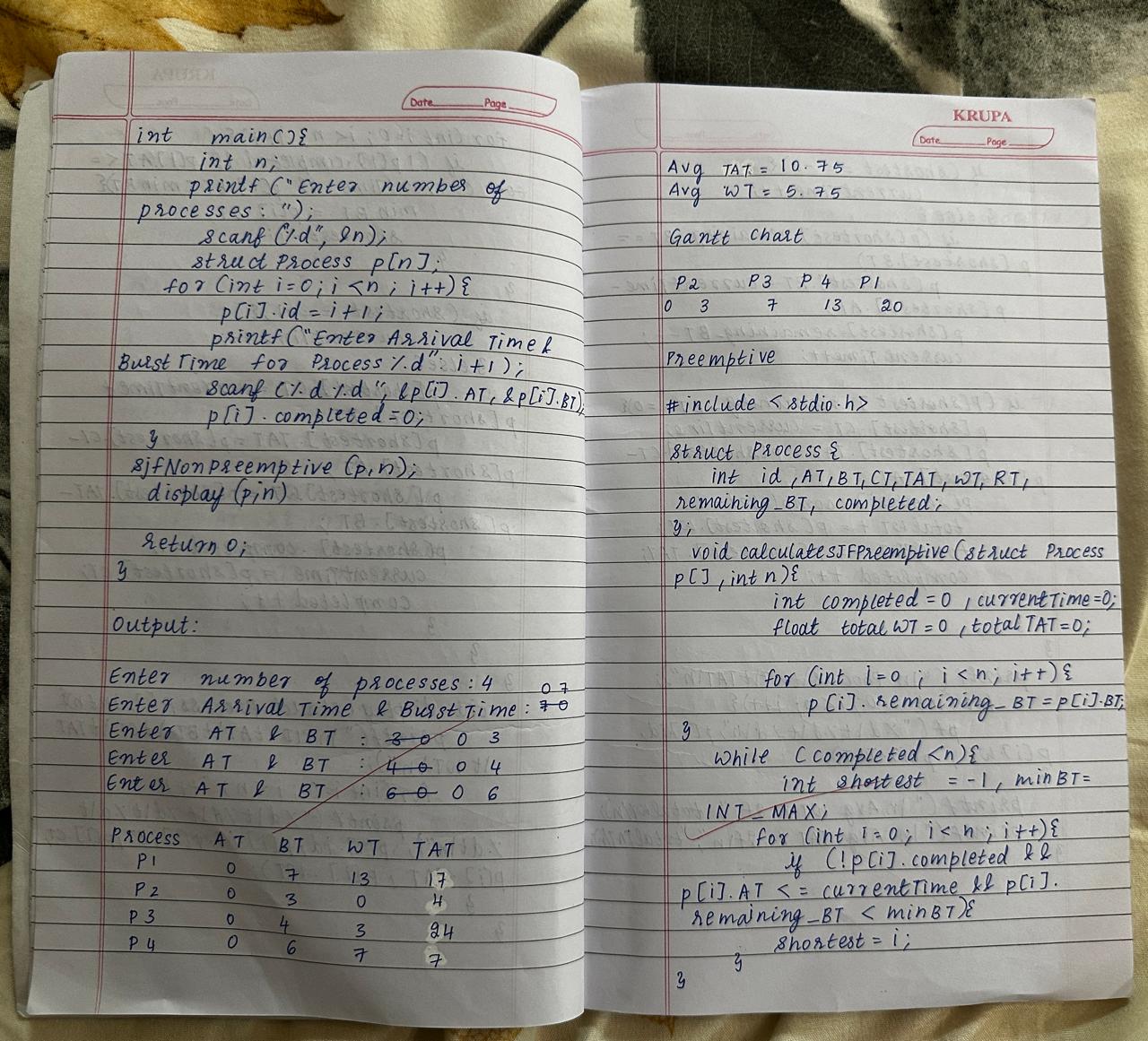
sjfPreemptive(p, n);

return 0;

}







Program -2

Question:

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

→Priority (preemptive and non preemptive)

#include <stdio.h>

struct Process {

int pid, arrival\_time, burst\_time, priority, remaining\_time, completion\_time, waiting\_time, turnaround\_time;

};

void priorityPreemptive(struct Process p[], int n) {

int time = 0, completed = 0, min\_priority, shortest;

float total\_waiting = 0, total\_turnaround = 0;

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

p[i].remaining\_time = p[i].burst\_time;

while (completed < n) {

min\_priority = 9999;

shortest = -1;

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

if (p[i].arrival\_time <= time && p[i].remaining\_time > 0 && p[i].priority < min\_priority) {

min\_priority = p[i].priority;

shortest = i;

}

}

if (shortest == -1) {

time++;

continue;

}

p[shortest].remaining\_time--;

if (p[shortest].remaining\_time == 0) {

completed++;

p[shortest].completion\_time = time + 1;

p[shortest].turnaround\_time = p[shortest].completion\_time - p[shortest].arrival\_time;

p[shortest].waiting\_time = p[shortest].turnaround\_time - p[shortest].burst\_time;

total\_waiting += p[shortest].waiting\_time;

total\_turnaround += p[shortest].turnaround\_time;

}

time++;

}

printf("\nPreemptive Priority Scheduling:\n");

printf("PID\tAT\tBT\tP\tCT\tTAT\tWT\n");

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

printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].arrival\_time, p[i].burst\_time, p[i].priority, p[i].completion\_time, p[i].turnaround\_time, p[i].waiting\_time);

printf("Average Waiting Time: %.2f\n", total\_waiting / n);

printf("Average Turnaround Time: %.2f\n", total\_turnaround / n);

}

int main() {

int n;

printf("Enter number of processes: ");

scanf("%d", &n);

struct Process p[n];

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

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

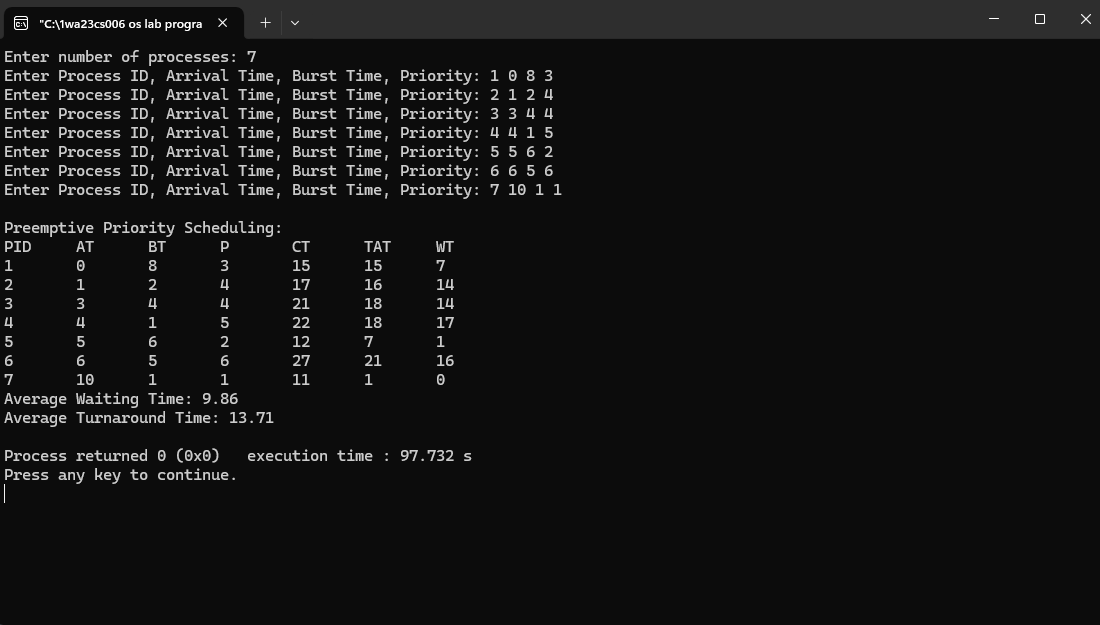
scanf("%d %d %d %d", &p[i].pid, &p[i].arrival\_time, &p[i].burst\_time, &p[i].priority);

}

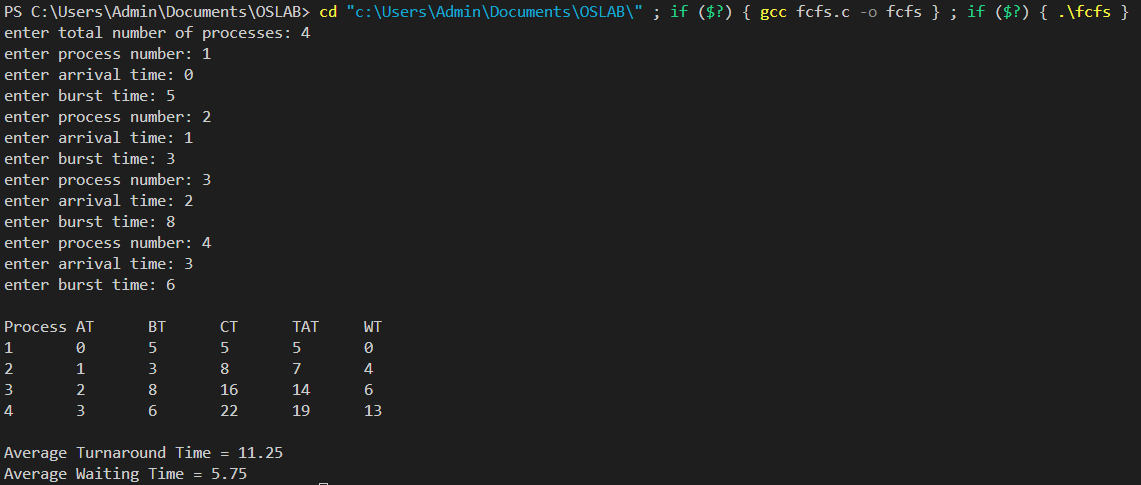
priorityPreemptive(p, n);

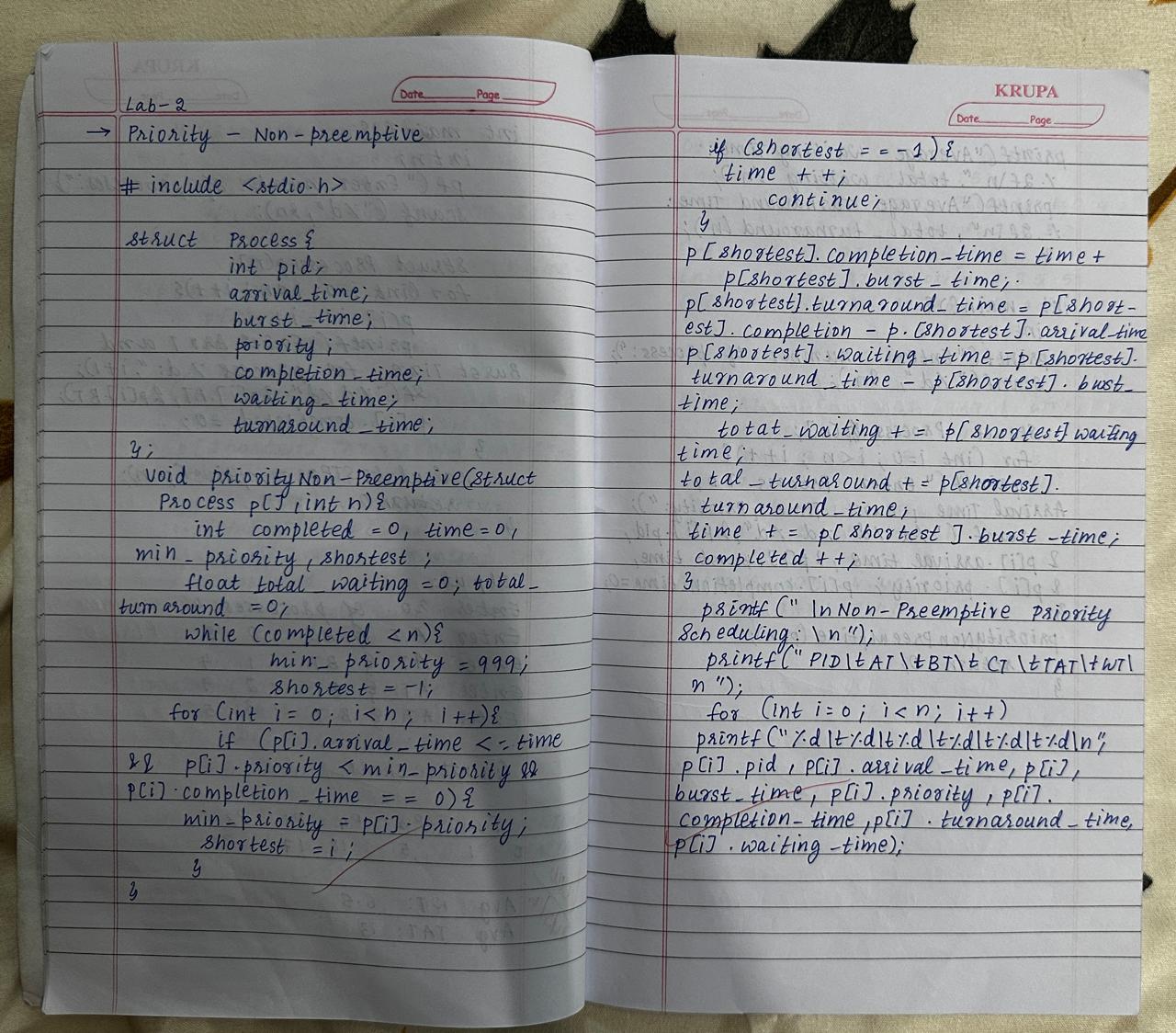
return 0;

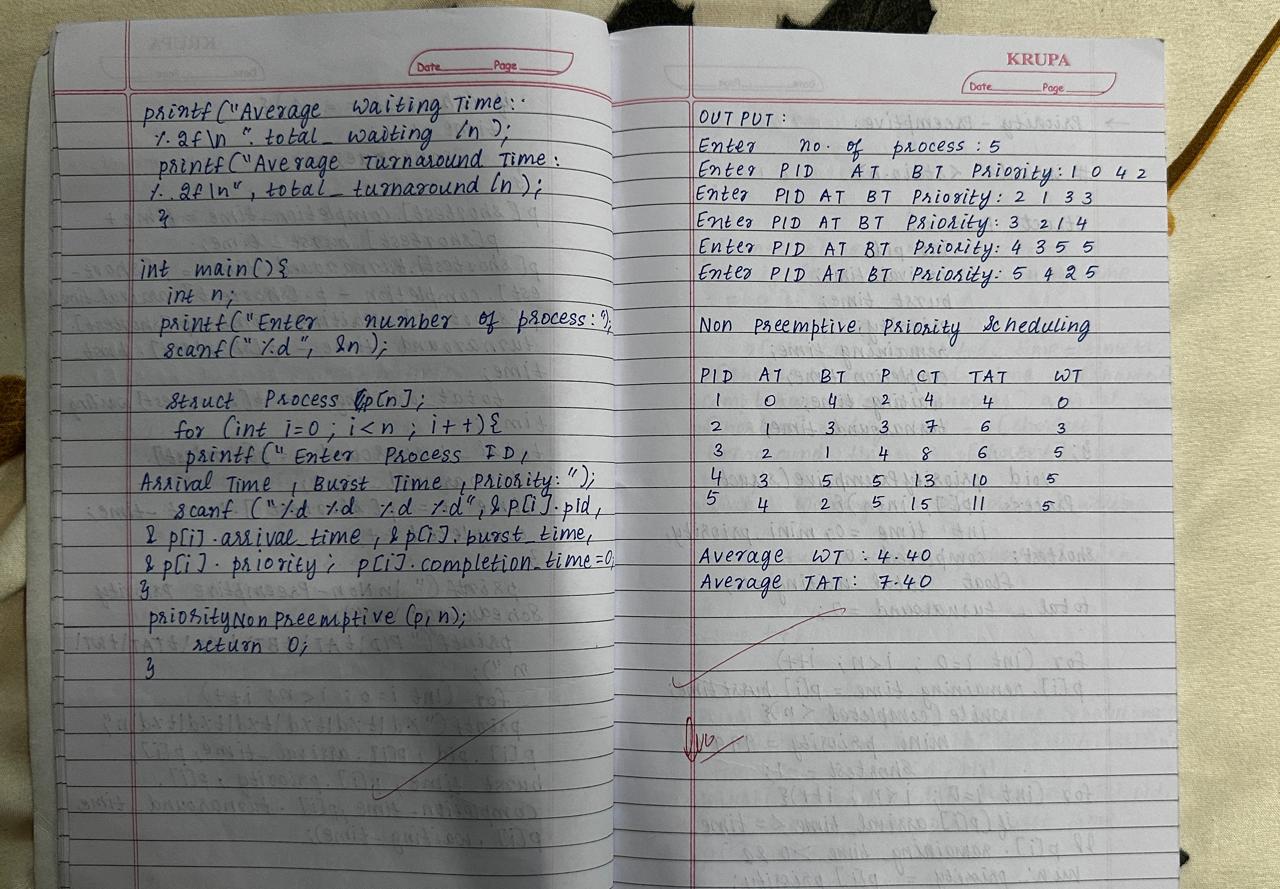
}

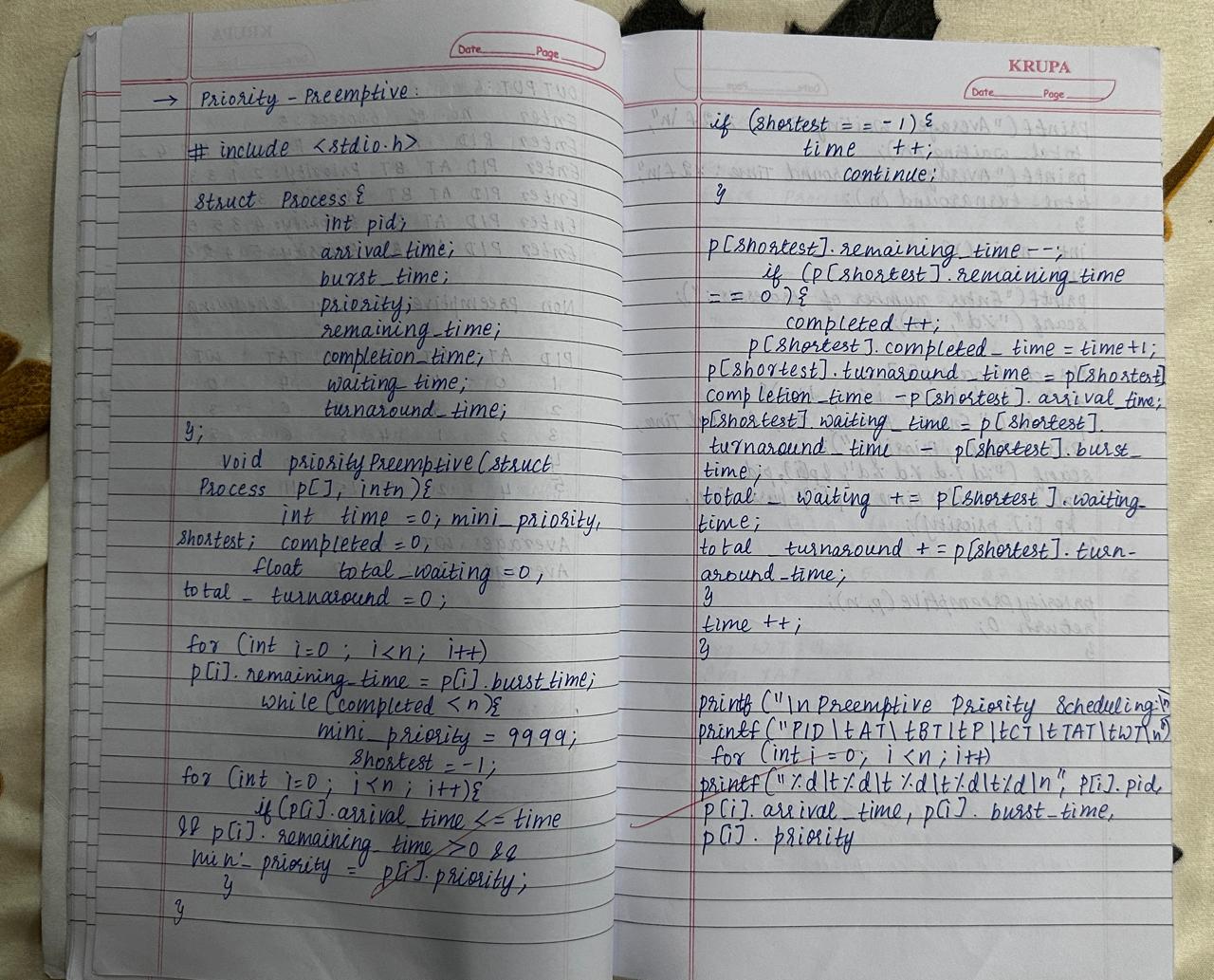


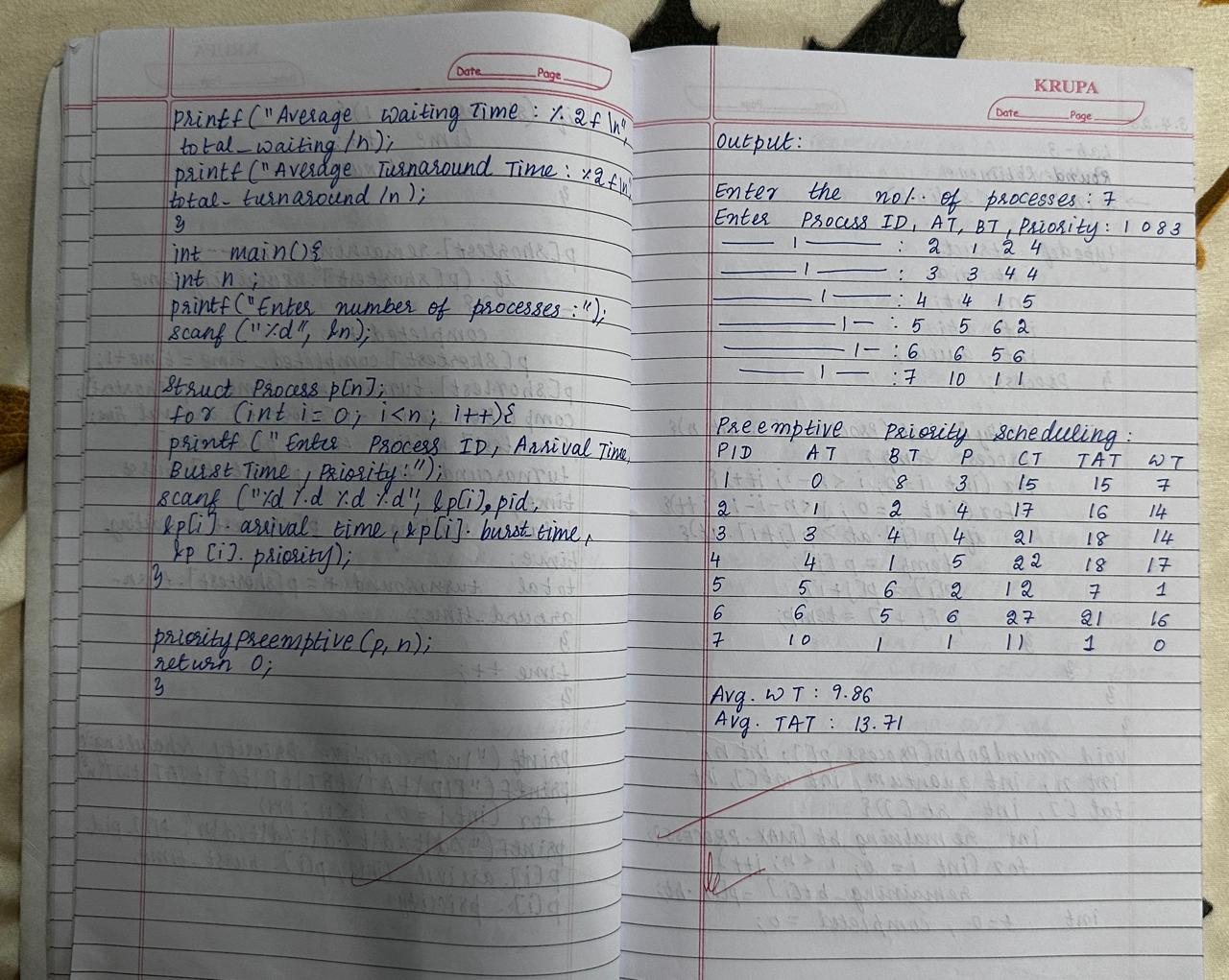
#include <stdio.h>  
  
struct Process {  
int pid, arrival\_time, burst\_time, priority, completion\_time, waiting\_time, turnaround\_time;  
};  
  
void priorityNonPreemptive(struct Process p[], int n) {  
int completed = 0, time = 0, min\_priority, shortest;  
float total\_waiting = 0, total\_turnaround = 0;  
  
while (completed < n) {  
min\_priority = 9999;  
shortest = -1;  
  
for (int i = 0; i < n; i++) {  
if (p[i].arrival\_time <= time && p[i].priority < min\_priority && p[i].completion\_time == 0) {  
min\_priority = p[i].priority;  
shortest = i;  
}  
}  
  
if (shortest == -1) {  
time++;  
continue;  
}  
  
p[shortest].completion\_time = time + p[shortest].burst\_time;  
p[shortest].turnaround\_time = p[shortest].completion\_time - p[shortest].arrival\_time;  
p[shortest].waiting\_time = p[shortest].turnaround\_time - p[shortest].burst\_time;  
  
total\_waiting += p[shortest].waiting\_time;  
total\_turnaround += p[shortest].turnaround\_time;  
time += p[shortest].burst\_time;  
completed++;  
}  
  
printf("\nNon-Preemptive Priority Scheduling:\n");  
printf("PID\tAT\tBT\tP\tCT\tTAT\tWT\n");  
for (int i = 0; i < n; i++)  
printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].arrival\_time, p[i].burst\_time, p[i].priority, p[i].completion\_time, p[i].turnaround\_time, p[i].waiting\_time);  
  
printf("Average Waiting Time: %.2f\n", total\_waiting / n);  
printf("Average Turnaround Time: %.2f\n", total\_turnaround / n);  
}  
  
int main() {  
int n;  
printf("Enter number of processes: ");  
scanf("%d", &n);  
  
struct Process p[n];  
  
for (int i = 0; i < n; i++) {  
printf("Enter Process ID, Arrival Time, Burst Time, Priority: ");  
scanf("%d %d %d %d", &p[i].pid, &p[i].arrival\_time, &p[i].burst\_time, &p[i].priority);  
p[i].completion\_time = 0;  
}  
  
priorityNonPreemptive(p, n);  
return 0;  
}











Program -3

Question:

Write a C program to simulate the following CPU scheduling

→Multilevel Queue

→ Rate Monotonic

→Earliest Deadline

#include <stdio.h>

#define MAX\_PROCESSES 10

typedef struct {

int pid;

int bt;

int at;

int queue;

} Process;

void sortByArrival(Process p[], int n) {

Process temp;

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

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

if (p[j].at > p[j + 1].at) {

temp = p[j];

p[j] = p[j + 1];

p[j + 1] = temp;

}

}

}

}

void roundRobin(Process p[], int n, int quantum, int wt[], int tat[], int rt[]) {

int remaining\_bt[MAX\_PROCESSES];

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

remaining\_bt[i] = p[i].bt;

int t = 0, completed = 0;

while (completed < n) {

int executed = 0;

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

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

if (rt[i] == -1) rt[i] = t;

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

t += quantum;

remaining\_bt[i] -= quantum;

} else {

t += remaining\_bt[i];

tat[i] = t - p[i].at;

wt[i] = tat[i] - p[i].bt;

remaining\_bt[i] = 0;

completed++;

}

executed = 1;

}

}

if (!executed) t++;

}

}

void fcfs(Process p[], int n, int start\_time, int wt[], int tat[], int rt[]) {

int time = start\_time;

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

if (time < p[i].at) time = p[i].at;

rt[i] = time - p[i].at;

wt[i] = rt[i];

tat[i] = wt[i] + p[i].bt;

time += p[i].bt;

}

}

int main() {

int n, quantum;

Process p[MAX\_PROCESSES], sys[MAX\_PROCESSES], usr[MAX\_PROCESSES];

int sys\_count = 0, usr\_count = 0;

int wt[MAX\_PROCESSES], tat[MAX\_PROCESSES], rt[MAX\_PROCESSES];

printf("Enter number of processes: ");

scanf("%d", &n);

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

printf("Enter Burst Time, Arrival Time and Queue (1=System, 2=User) for P%d: ", i + 1);

p[i].pid = i + 1;

scanf("%d %d %d", &p[i].bt, &p[i].at, &p[i].queue);

if (p[i].queue == 1)

sys[sys\_count++] = p[i];

else

usr[usr\_count++] = p[i];

wt[i] = 0;

tat[i] = 0;

rt[i] = -1;

}

printf("Enter time quantum for Round Robin scheduling: ");

scanf("%d", &quantum);

sortByArrival(sys, sys\_count);

sortByArrival(usr, usr\_count);

roundRobin(sys, sys\_count, quantum, wt, tat, rt);

int last\_sys\_time = (sys\_count > 0) ? tat[sys\_count - 1] + sys[sys\_count - 1].at : 0;

fcfs(usr, usr\_count, last\_sys\_time, &wt[sys\_count], &tat[sys\_count], &rt[sys\_count]);

printf("\nProcess\tQueue\tWaiting Time\tTurn Around Time\tResponse Time\n");

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

printf("P%d\t%d\t%d\t\t%d\t\t\t%d\n", p[i].pid, p[i].queue, wt[i], tat[i], rt[i]);

float avg\_wt = 0, avg\_tat = 0, avg\_rt = 0;

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

avg\_wt += wt[i];

avg\_tat += tat[i];

avg\_rt += rt[i];

}

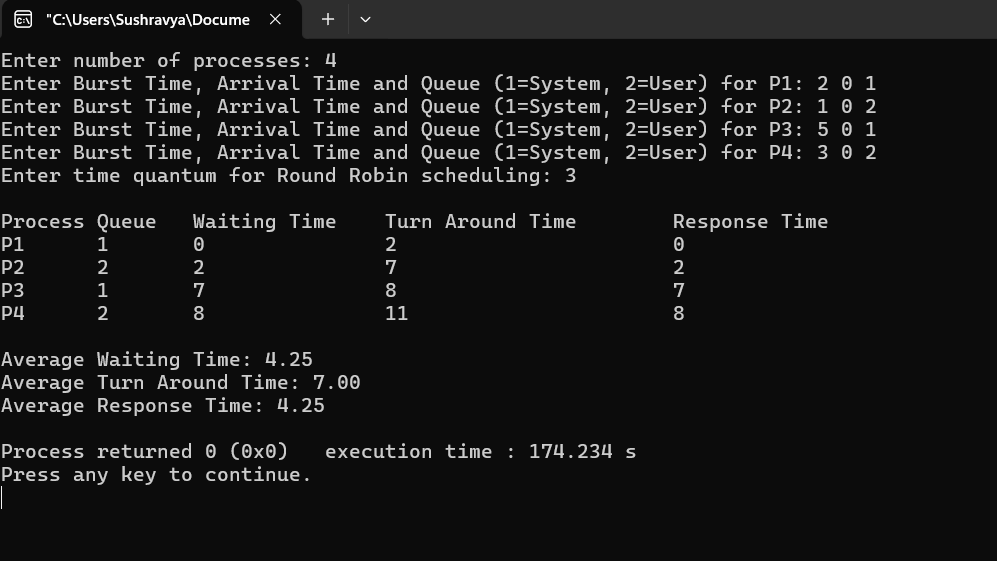
printf("\nAverage Waiting Time: %.2f", avg\_wt / n);

printf("\nAverage Turn Around Time: %.2f", avg\_tat / n);

printf("\nAverage Response Time: %.2f\n", avg\_rt / n);

return 0;

}



#include <stdio.h>

#define MAX\_PROCESSES 10

typedef struct {

int id;

int burst\_time;

int period;

int remaining\_time;

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 - i - 1; j++) {

if (processes[j].period > processes[j + 1].period) {

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 sum = 0;

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

sum += (double)processes[i].burst\_time / processes[i].period;

}

return sum;

}

double rms\_threshold(int n) {

return n \* (pow(2.0, 1.0 / n) - 1);

}

void rate\_monotonic\_scheduling(Process processes[], int n) {

int lcm\_period = calculate\_lcm(processes, n);

printf("LCM=%d\n\n", lcm\_period);

printf("Rate Monotone Scheduling:\n");

printf("PID Burst Period\n");

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

printf("%d %d %d\n", processes[i].id, processes[i].burst\_time, processes[i].period);

}

double utilization = utilization\_factor(processes, n);

double threshold = rms\_threshold(n);

printf("\n%.6f <= %.6f => %s\n", utilization, threshold, (utilization <= threshold) ? "true" :

"false");

if (utilization > threshold) {

printf("\nSystem may not be schedulable!\n");

return;

}

int timeline = 0, executed = 0;

while (timeline < lcm\_period) {

int selected = -1;

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

if (timeline % processes[i]×period == 0) {

processes[i].remaining\_time = processes[i].burst\_time;

}

if (processes[i].remaining\_time > 0) {

selected = i;

break;

}

}

if (selected != -1) {

printf("Time %d: Process %d is running\n", timeline, processes[selected].id);

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 number of processes: ");

scanf("%d", &n);

printf("Enter the CPU burst times:\n");

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

processes[i]×id = i + 1;

scanf("%d", &processes[i].burst\_time);

processes[i].remaining\_time = processes[i].burst\_time;

}

printf("Enter the time periods:\n");

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

scanf("%d", &processes[i].period);

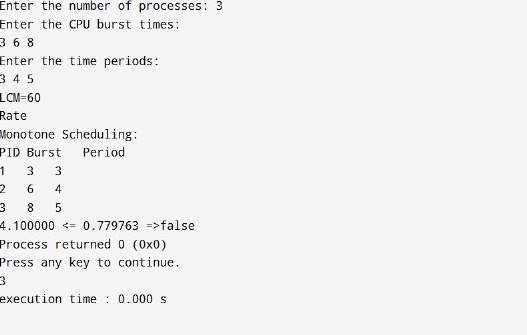
}

sort\_by\_period(processes, n);

rate\_monotonic\_scheduling(processes, n);

return 0;

}



#include <stdio.h>

int gcd(int a, int b) {

while (b != 0) {

int temp = b;

b = a % b;

a = temp;

}

return a;

}

int lcm(int a, int b) {

return (a \* b) / gcd(a, b);

}

struct Process {

int id, burst\_time, deadline, period;

};

void earliest\_deadline\_first(struct Process p[], int n, int time\_limit) {

int time = 0;

printf("Earliest Deadline Scheduling:\n");

printf("PID\tBurst\tDeadline\tPeriod\n");

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

printf("%d\t%d\t\t%d\t\t%d\n", p[i].id, p[i].burst\_time, p[i].deadline, p[i].period);

}

printf("\nScheduling occurs for %d ms\n", time\_limit);

while (time < time\_limit) {

int earliest = -1;

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

if (p[i].burst\_time > 0) {

if (earliest == -1 || p[i].deadline < p[earliest].deadline) {

earliest = i;

}

}

}

if (earliest == -1) break;

printf("%dms: Task %d is running.\n", time, p[earliest].id);

p[earliest].burst\_time--;

time++;

}

}

int main() {

int n;

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

scanf("%d", &n);

struct Process processes[n];

printf("Enter the CPU burst times:\n");

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

scanf("%d", &processes[i].burst\_time);

processes[i]×id = i + 1;

}

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

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

scanf("%d", &processes[i].deadline);

}

printf("Enter the time periods:\n");

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

scanf("%d", &processes[i].period);

}

int hyperperiod = processes[0]×period;

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

hyperperiod = lcm(hyperperiod, processes[i].period);

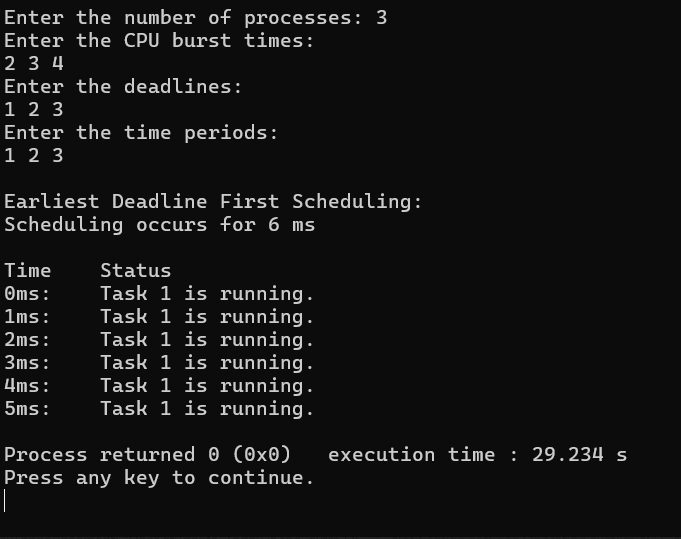
}

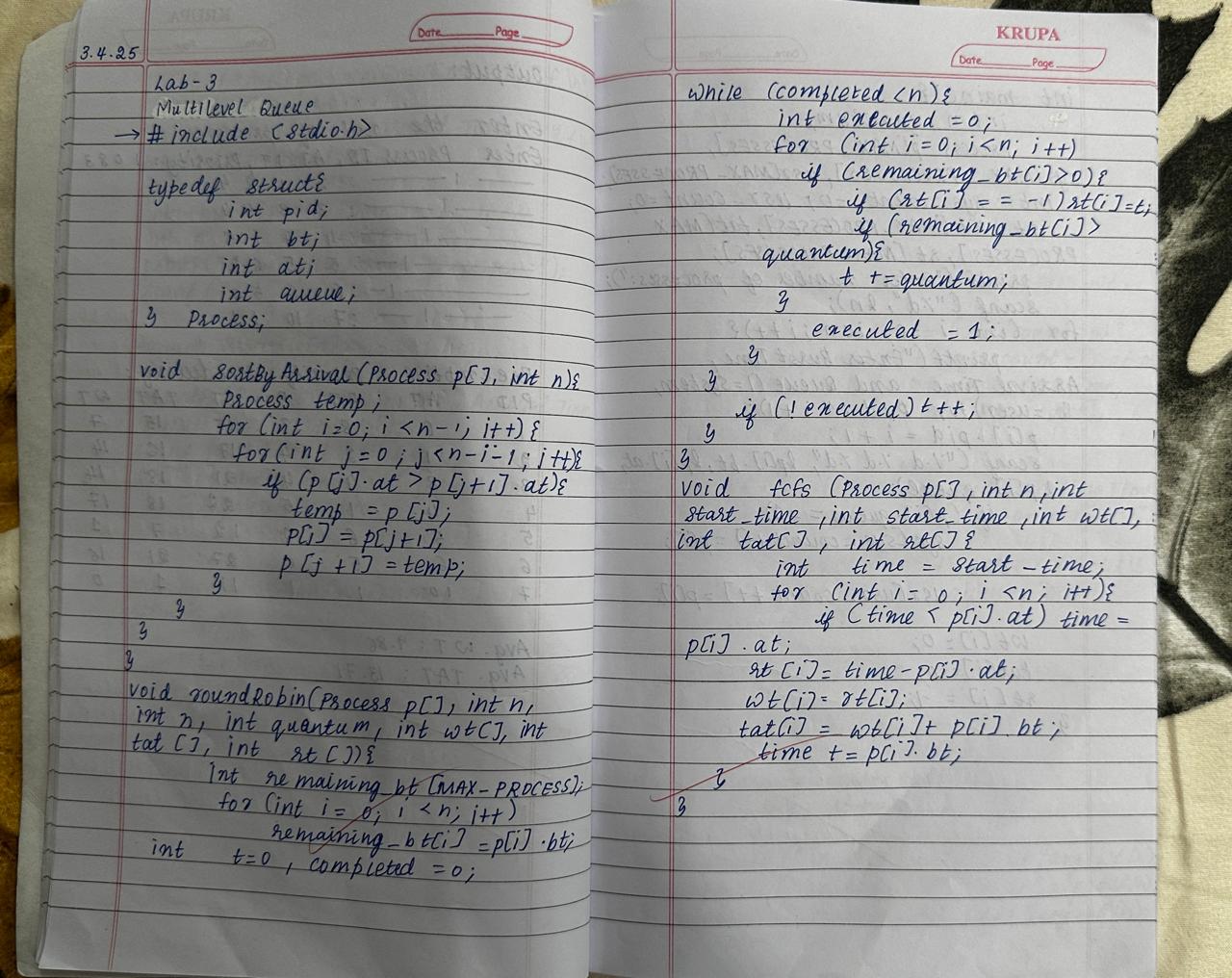
printf("\nSystem will execute for hyperperiod (LCM of periods): %d ms\n", hyperperiod);

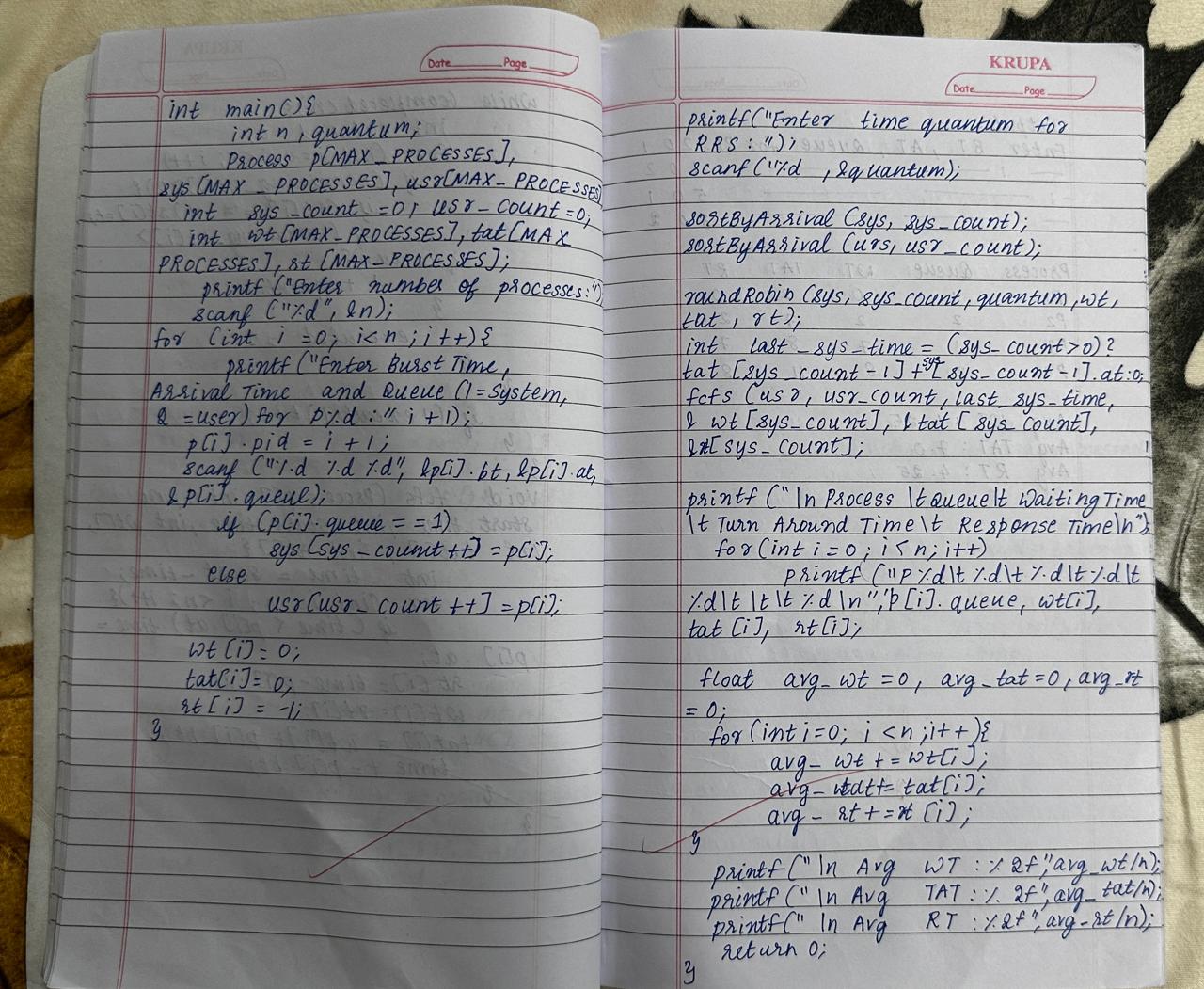
earliest\_deadline\_first(processes, n, hyperperiod);

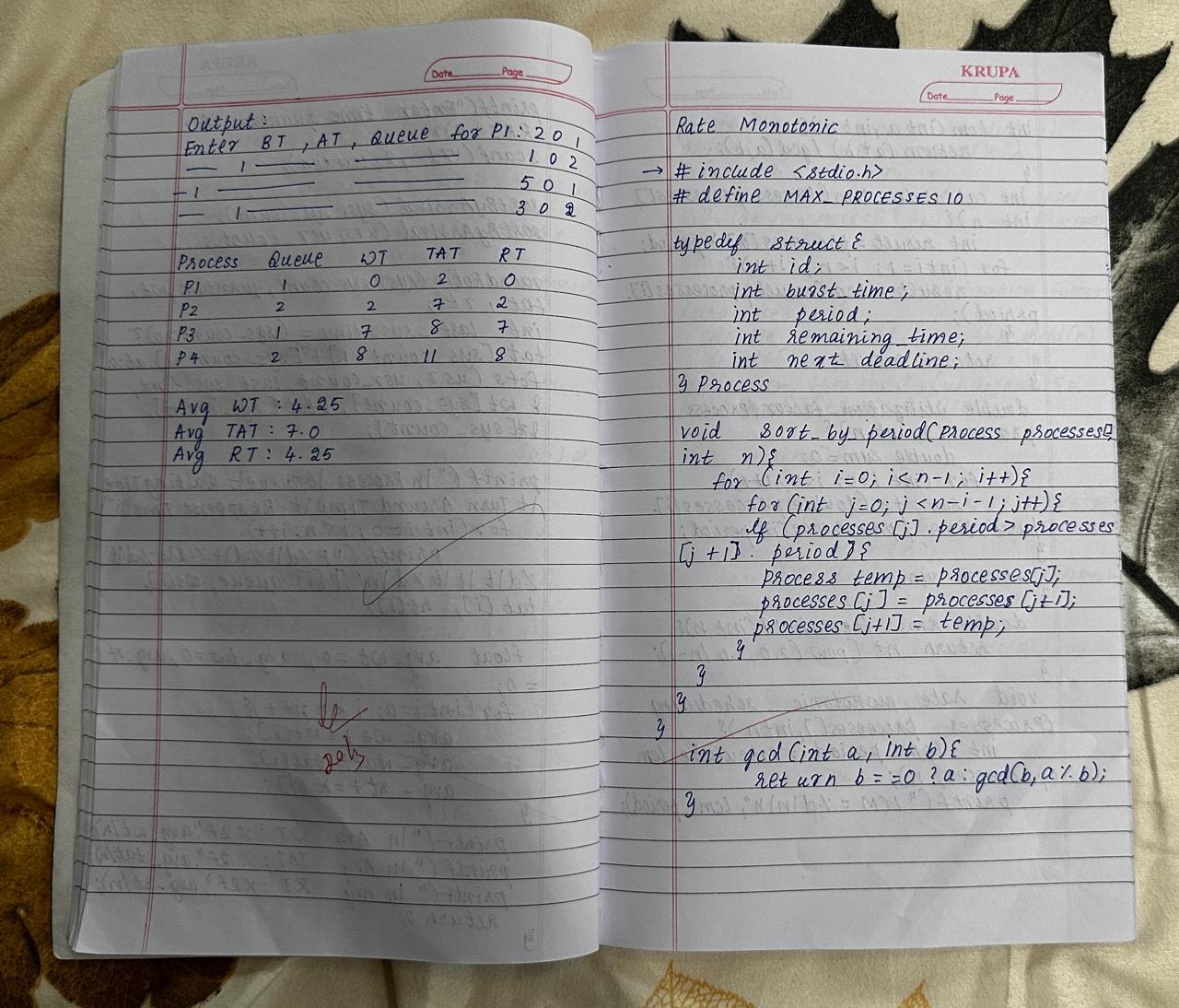
return 0;

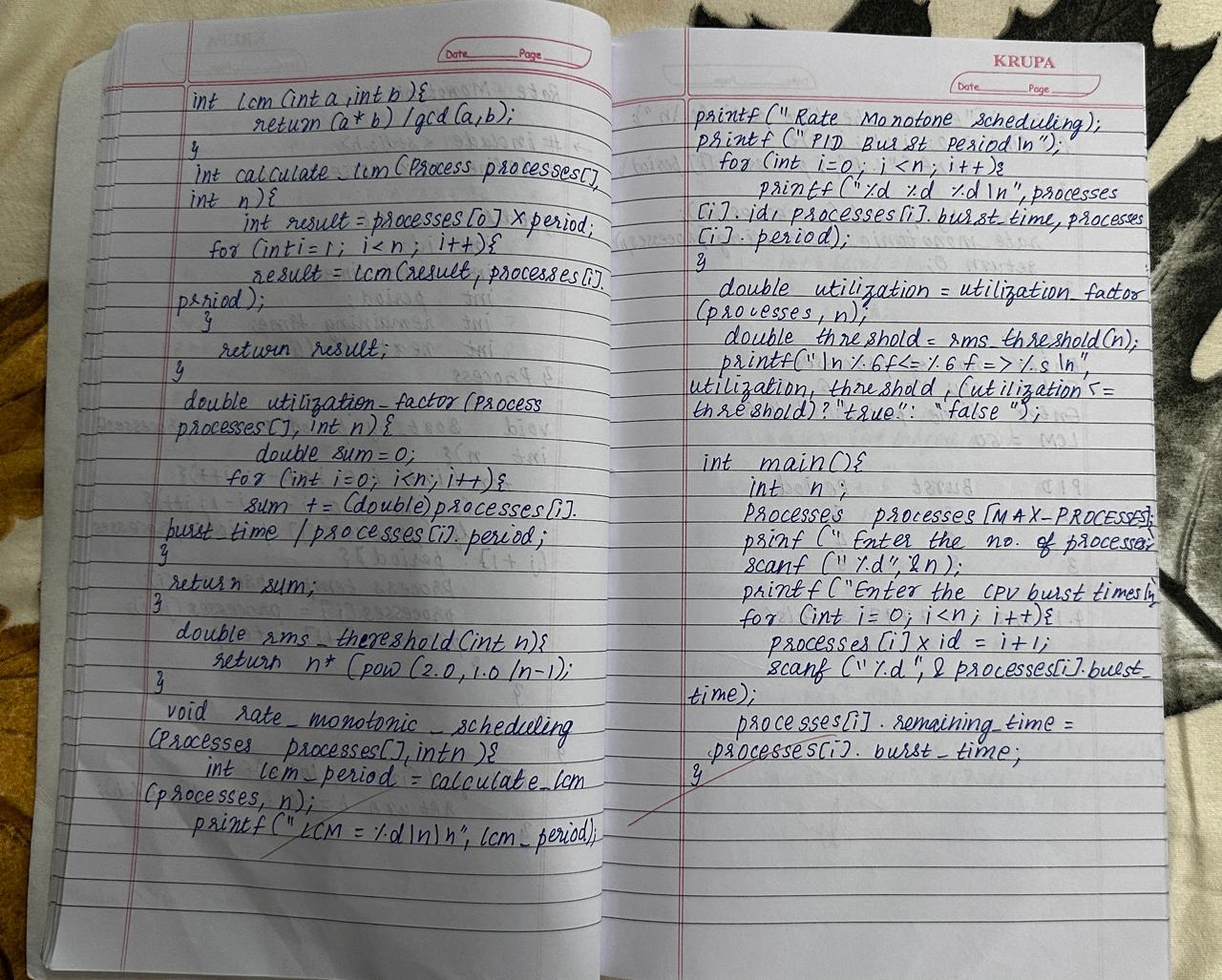
}

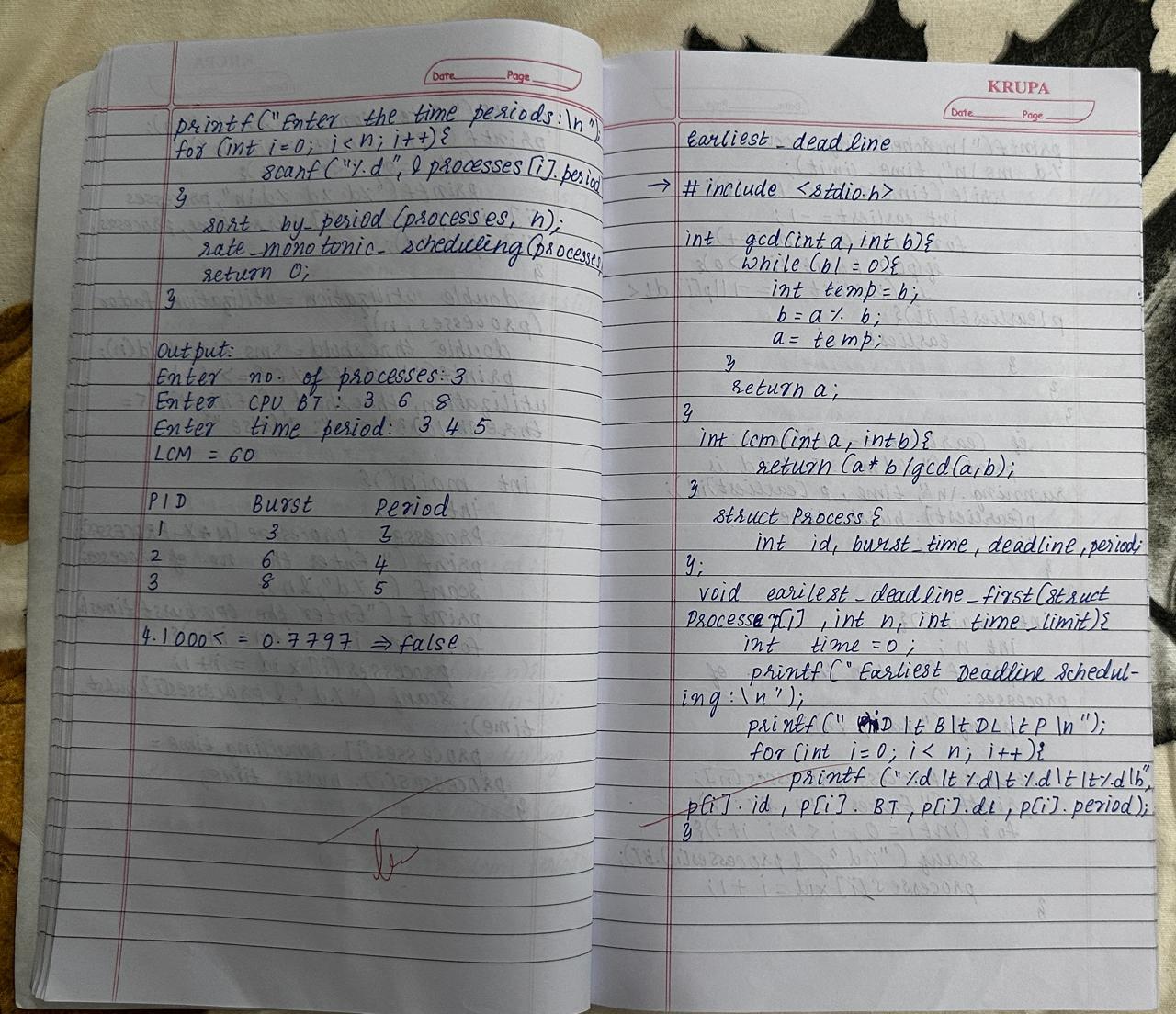


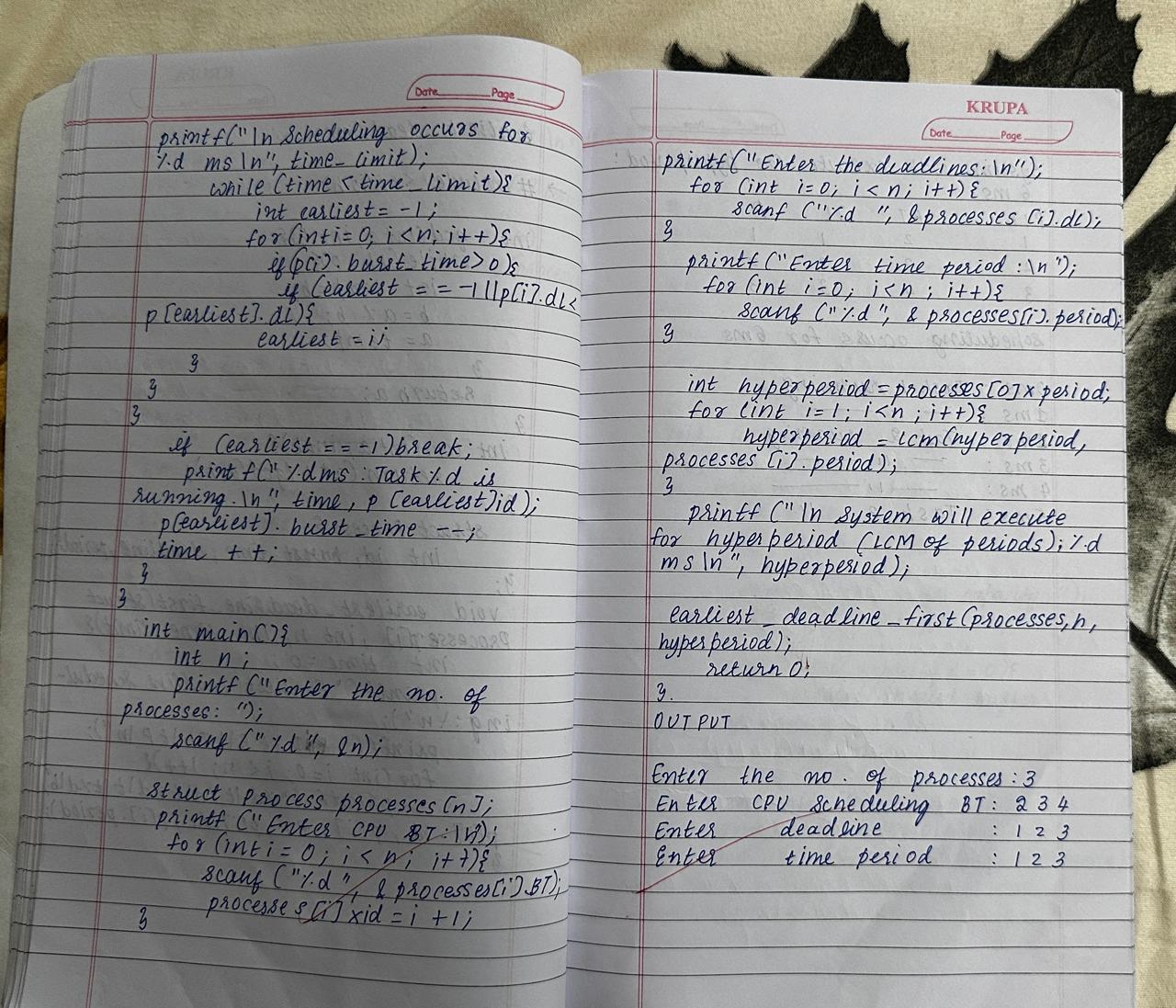


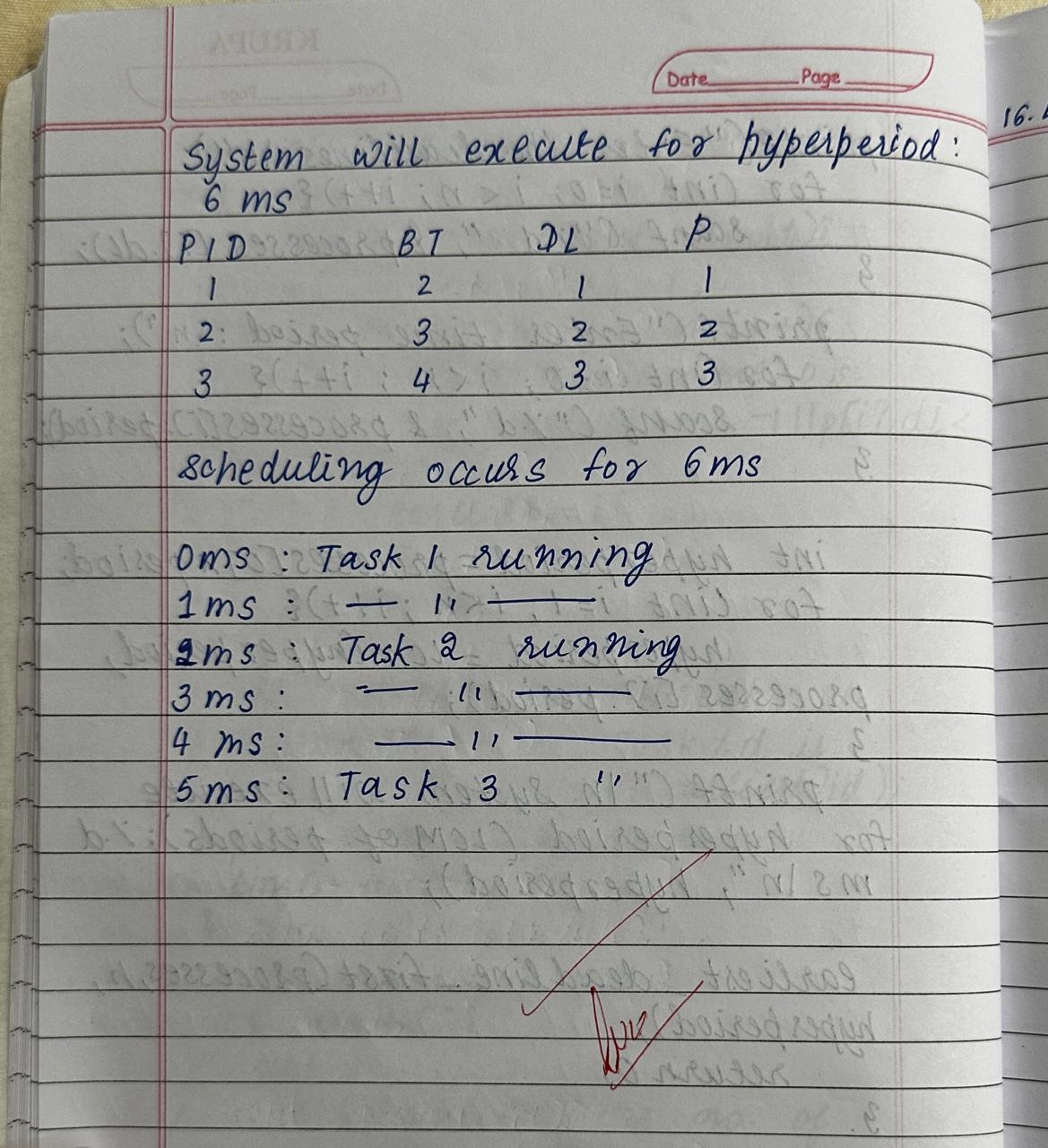












Program - 4

Question:

Write a C program to simulate the following CPU scheduling

→Producer and Consumer

→ Dining Philosopher

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

sem\_t mutex;

sem\_t full;

sem\_t empty;

int x = 0;

void producer();

void consumer();

int main() {

int choice;

int buffer\_capacity;

printf("Enter buffer capacity: ");

scanf("%d", &buffer\_capacity);

if (buffer\_capacity <= 0) {

printf("Invalid buffer capacity. Exiting...\n");

return 1;

}

sem\_init(&mutex, 0, 1);

sem\_init(&full, 0, 0);

sem\_init(&empty, 0, buffer\_capacity);

while (1) {

printf("\nChoose an option:\n");

printf("1. Producer\n");

printf("2. Consumer\n");

printf("3. Exit\n");

printf("Enter choice: ");

scanf("%d", &choice);

if (choice == 1) {

producer();

} else if (choice == 2) {

consumer();

} else if (choice == 3) {

break;

} else {

printf("Invalid choice, please try again.\n");

}

}

sem\_destroy(&mutex);

sem\_destroy(&full);

sem\_destroy(&empty);

return 0;

}

void producer() {

sem\_wait(&mutex);

if (sem\_trywait(&empty) == 0) {

x++;

printf("Produced item: %d\n", x);

sem\_post(&full);

} else {

printf("Buffer is full. Cannot produce more items.\n");

}

sem\_post(&mutex);

}

void consumer() {

sem\_wait(&mutex);

if (sem\_trywait(&full) == 0) {

printf("Consumed item: %d\n", x);

x--;

sem\_post(&empty);

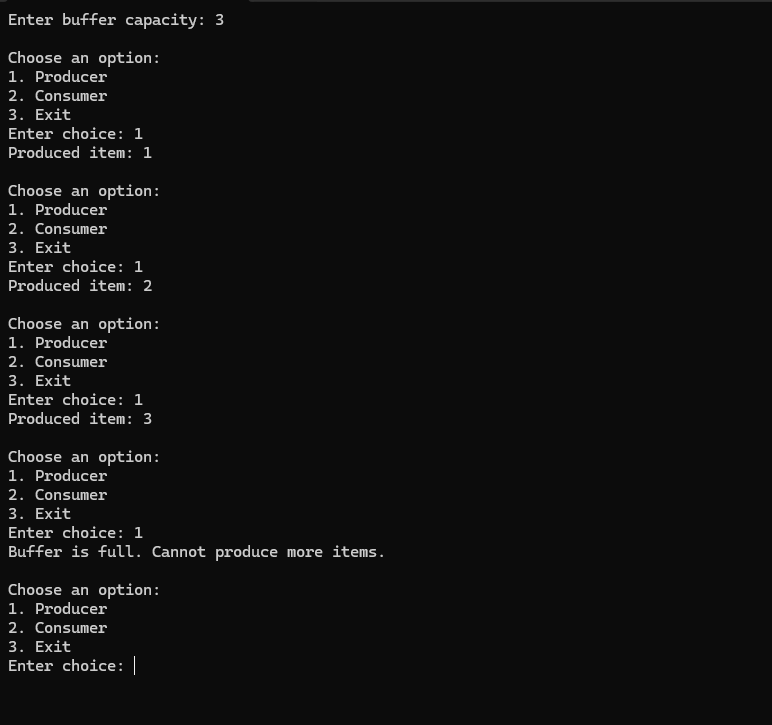
} else {

printf("Buffer is empty. Cannot consume items.\n");

}

sem\_post(&mutex);

}



#include <stdio.h>

#include <stdlib.h>

#define MAX 10

int totalPhilosophers;

int hungry[MAX];

int areNeighbors(int a, int b) {

return (abs(a - b) == 1 || abs(a - b) == totalPhilosophers - 1);

}

void option1(int count) {

printf("\nAllow one philosopher to eat at any time\n");

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

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

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

if (j != i) {

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

}

}

}

}

void option2(int count) {

printf("\nAllow two philosophers to eat at same time\n");

int combination = 1;

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

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

if (!areNeighbors(hungry[i], hungry[j])) {

printf("combination %d\n", combination++);

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

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

if (k != i && k != j) {

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

}

}

printf("\n");

}

}

}

if (combination == 1) {

printf("No combinations found where two non-neighbor philosophers can eat.\n");

}

}

int main() {

int hungryCount;

printf("DINING PHILOSOPHER PROBLEM\n");

printf("Enter the total no. of philosophers: ");

scanf("%d", &totalPhilosophers);

printf("How many are hungry: ");

scanf("%d", &hungryCount);

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

printf("Enter philosopher %d position: ", i + 1);

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

}

int choice;

do {

printf("\n1. One can eat at a time 2. Two can eat at a time 3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

option1(hungryCount);

break;

case 2:

option2(hungryCount);

break;

case 3:

printf("Exiting...\n");

break;

default:

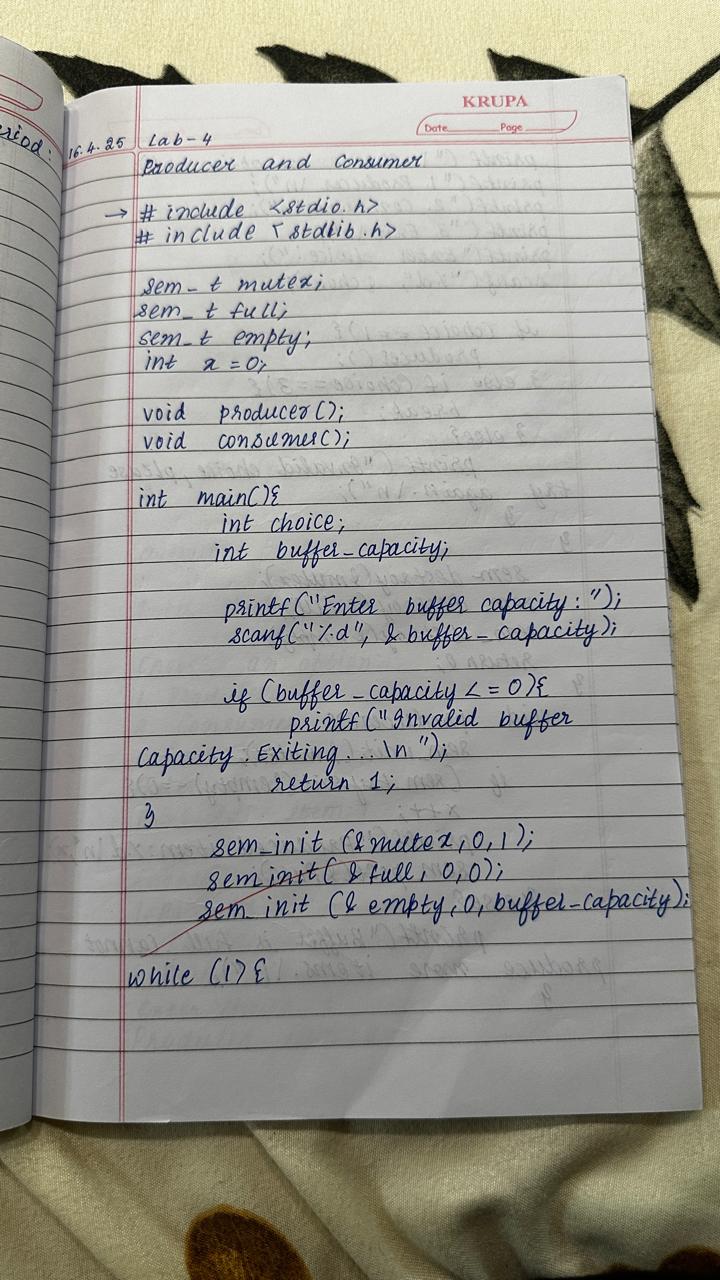
printf("Invalid choice!\n");

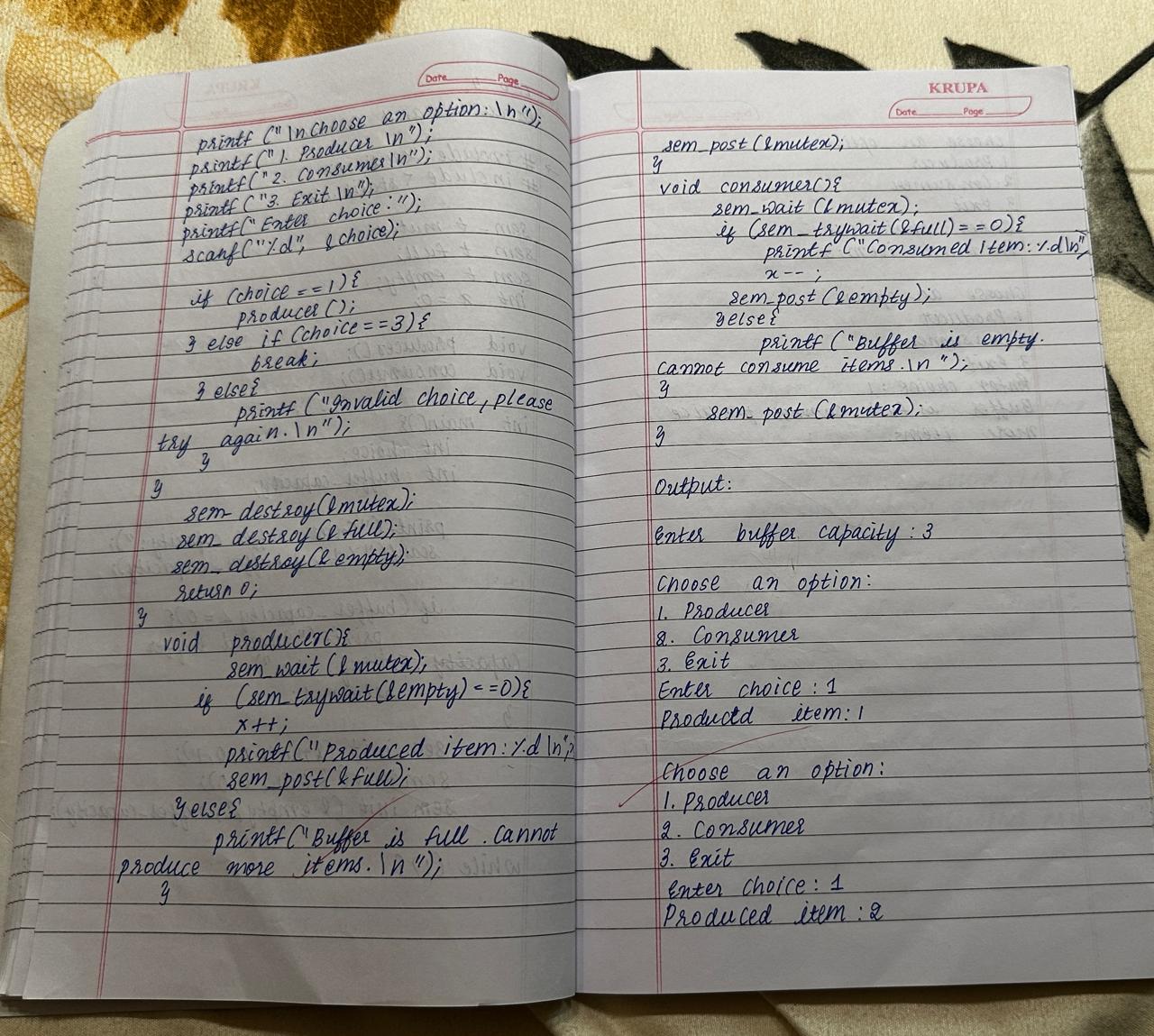
}

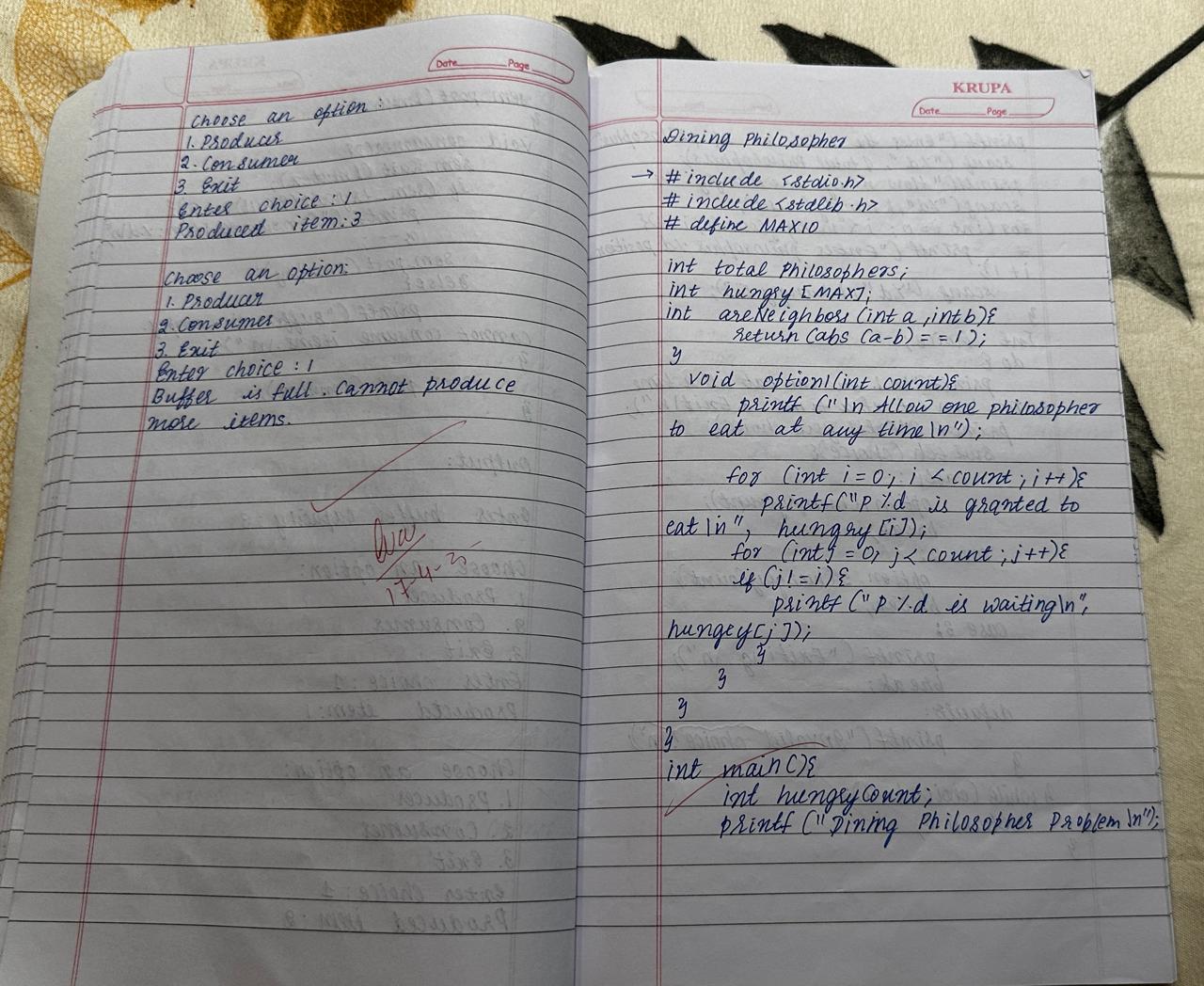
} while (choice != 3);

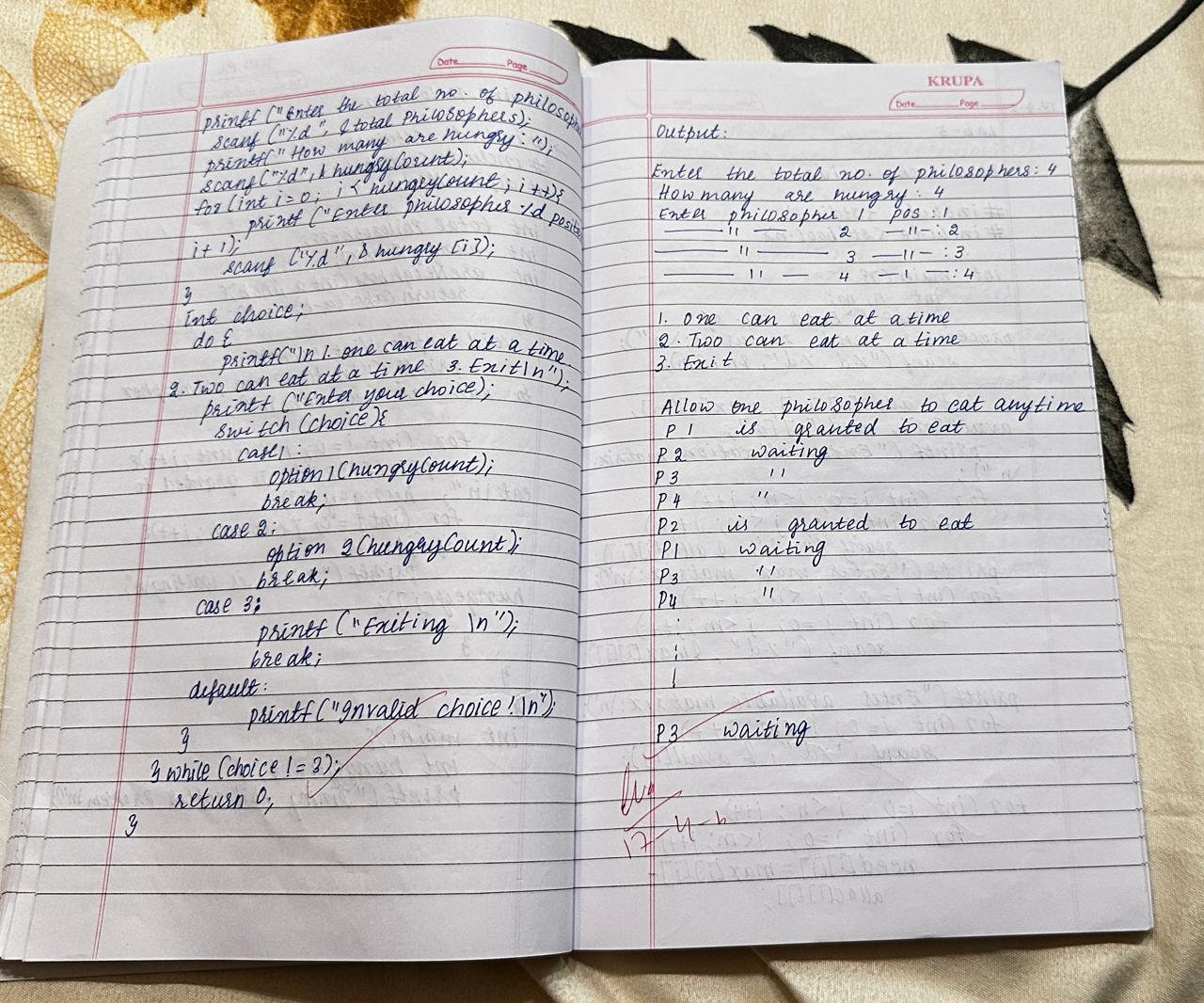
return 0;











Program - 5

Question:

Write a C program to simulate the following CPU scheduling

→Deadlock avoidance

→Deadlock detection

#include <stdio.h>

#include <stdbool.h>

int main() {

int n, m;

printf("Enter number of processes and 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 < m; j++)

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

printf("Enter max matrix:\n");

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

for (int j = 0; j < m; j++)

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

printf("Enter available matrix:\n");

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

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

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

for (int j = 0; j < m; j++)

need[i][j] = max[i][j] - alloc[i][j];

bool finish[n];

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

int safeSeq[n];

int count = 0;

while (count < n) {

bool found = false;

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

if (!finish[p]) {

bool canAllocate = true;

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

if (need[p][j] > avail[j]) {

canAllocate = false;

break;

}

}

if (canAllocate) {

for (int k = 0; k < m; k++)

avail[k] += alloc[p][k];

safeSeq[count++] = p;

finish[p] = true;

found = true;

}

}

}

if (!found) {

printf("System is not in a safe state.\n");

return 1;

}

}

printf("System is in safe state.\n");

printf("Safe sequence is: ");

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

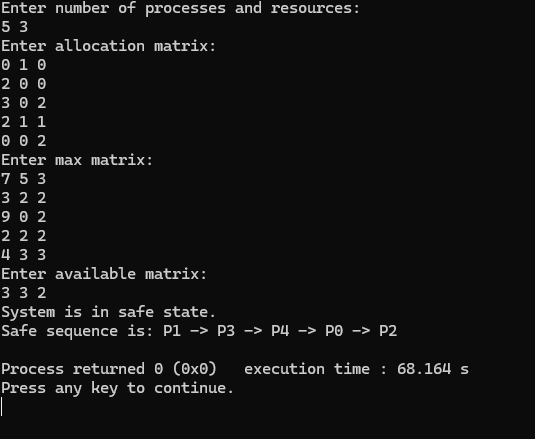
printf("P%d", safeSeq[i]);

if (i != n - 1) printf(" -> ");

}

printf("\n");

return 0;



#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

int main() {

int P, R;

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

scanf("%d", &P);

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

scanf("%d", &R);

int alloc[P][R], max[P][R], need[P][R], finish[P];

int avail[R];

printf("Enter Allocation Matrix:\n");

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

printf("Process %d: ", i);

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

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

}

}

printf("Enter Maximum Matrix:\n");

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

printf("Process %d: ", i);

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

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

}

}

printf("Enter Available Resources:\n");

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

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

}

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

finish[i] = 0;

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

need[i][j] = max[i][j] - alloc[i][j];

}

}

int count = 0;

bool deadlock = false;

while (count < P) {

bool found = false;

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

if (!finish[i]) {

int j;

for (j = 0; j < R; j++) {

if (need[i][j] > avail[j])

break;

}

if (j == R) {

for (int k = 0; k < R; k++)

avail[k] += alloc[i][k];

finish[i] = 1;

found = true;

count++;

printf("Process %d can finish.\n", i);

}

}

}

if (!found) {

deadlock = true;

break;

}

}

if (!deadlock)

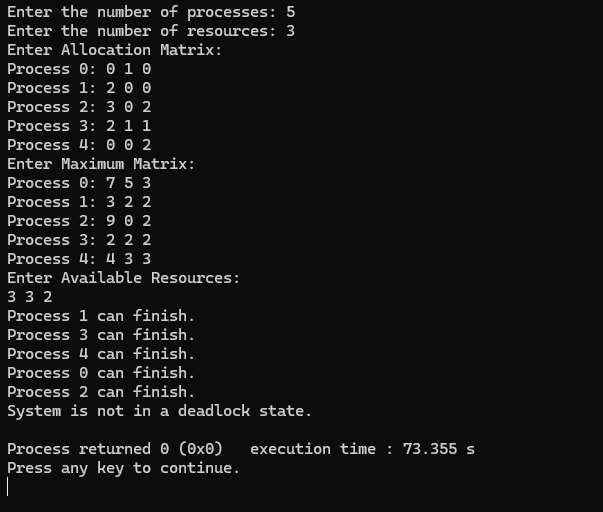
printf("System is not in a deadlock state.\n");

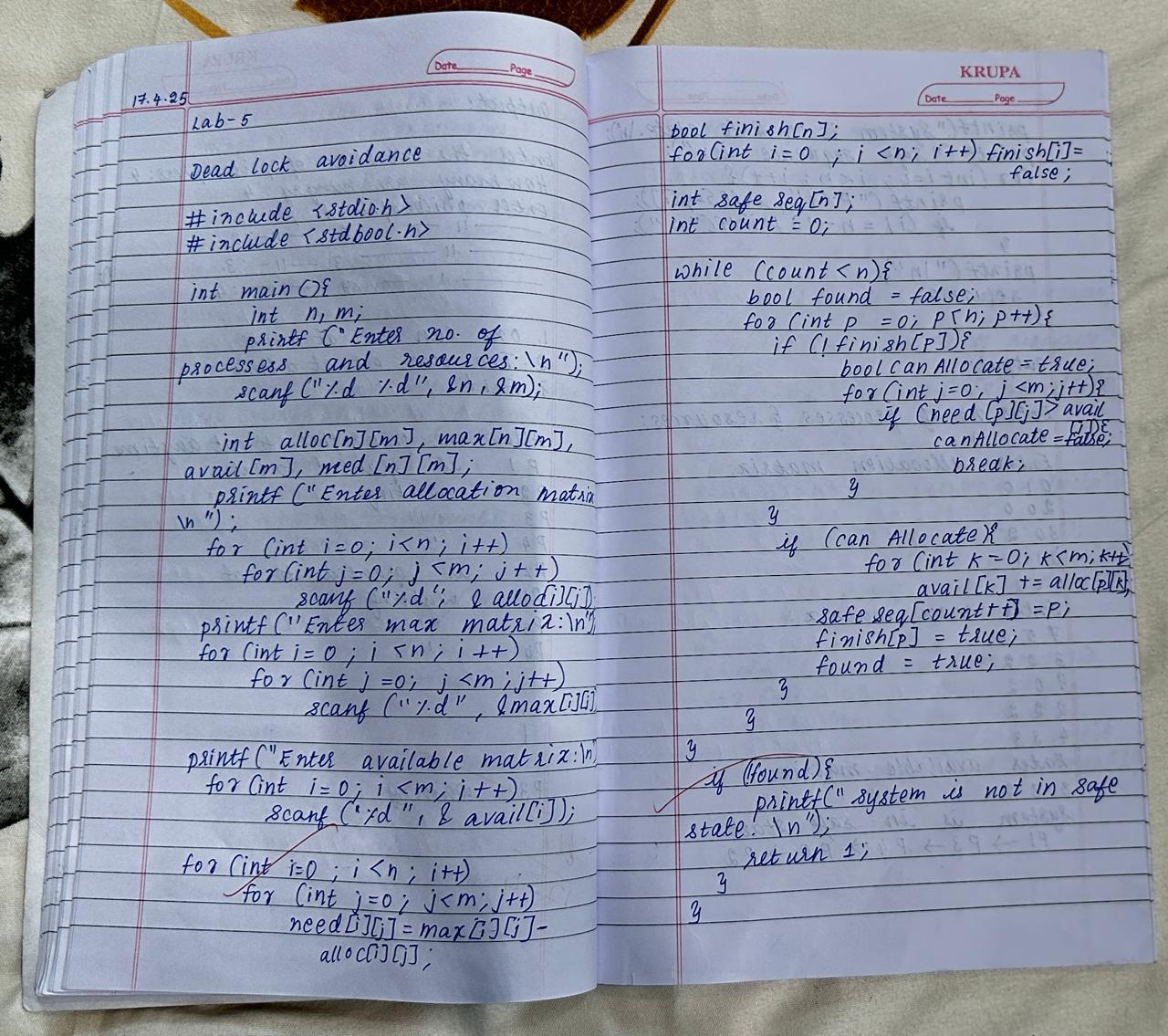
else

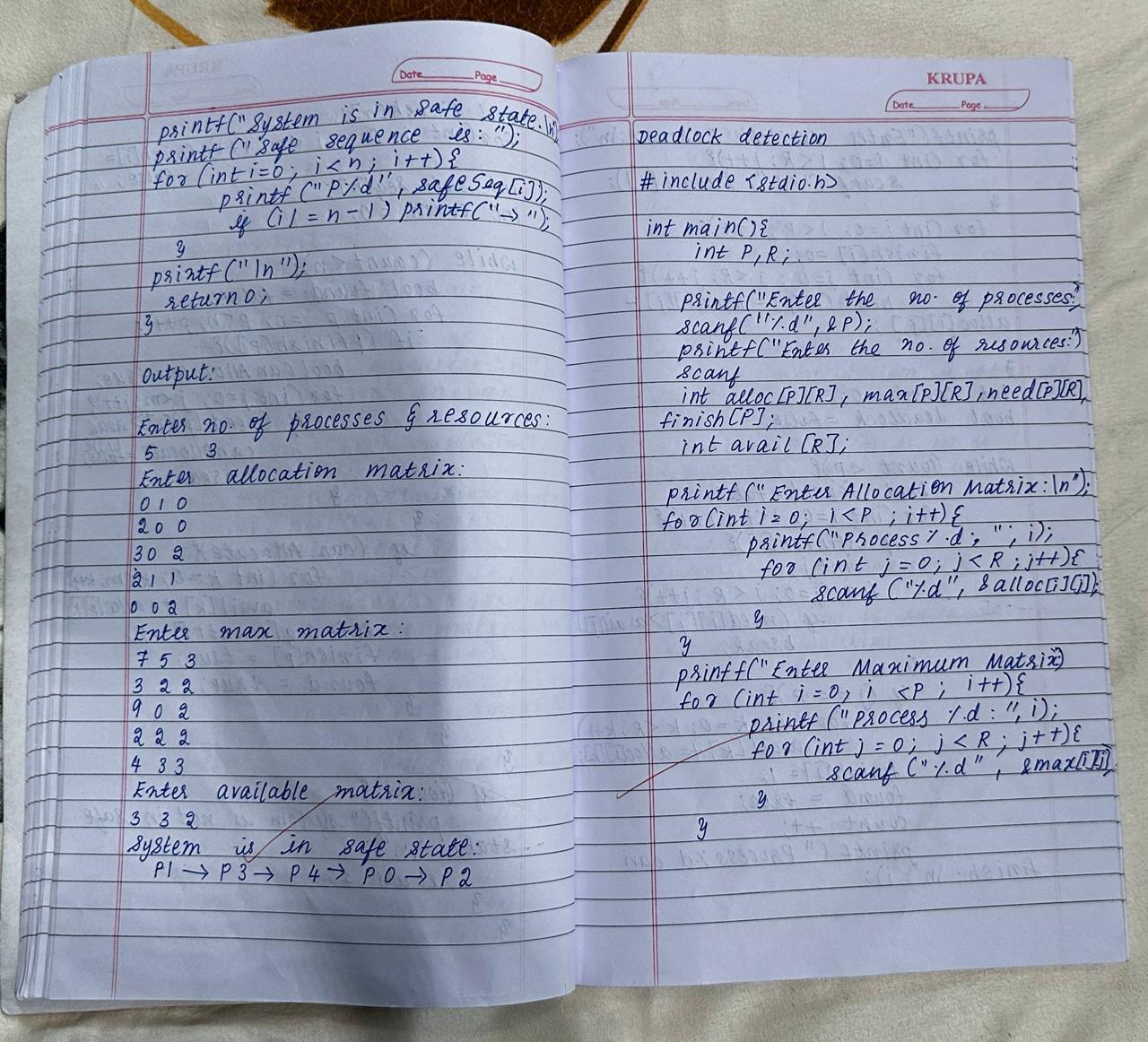
printf("System is in a deadlock state.\n");

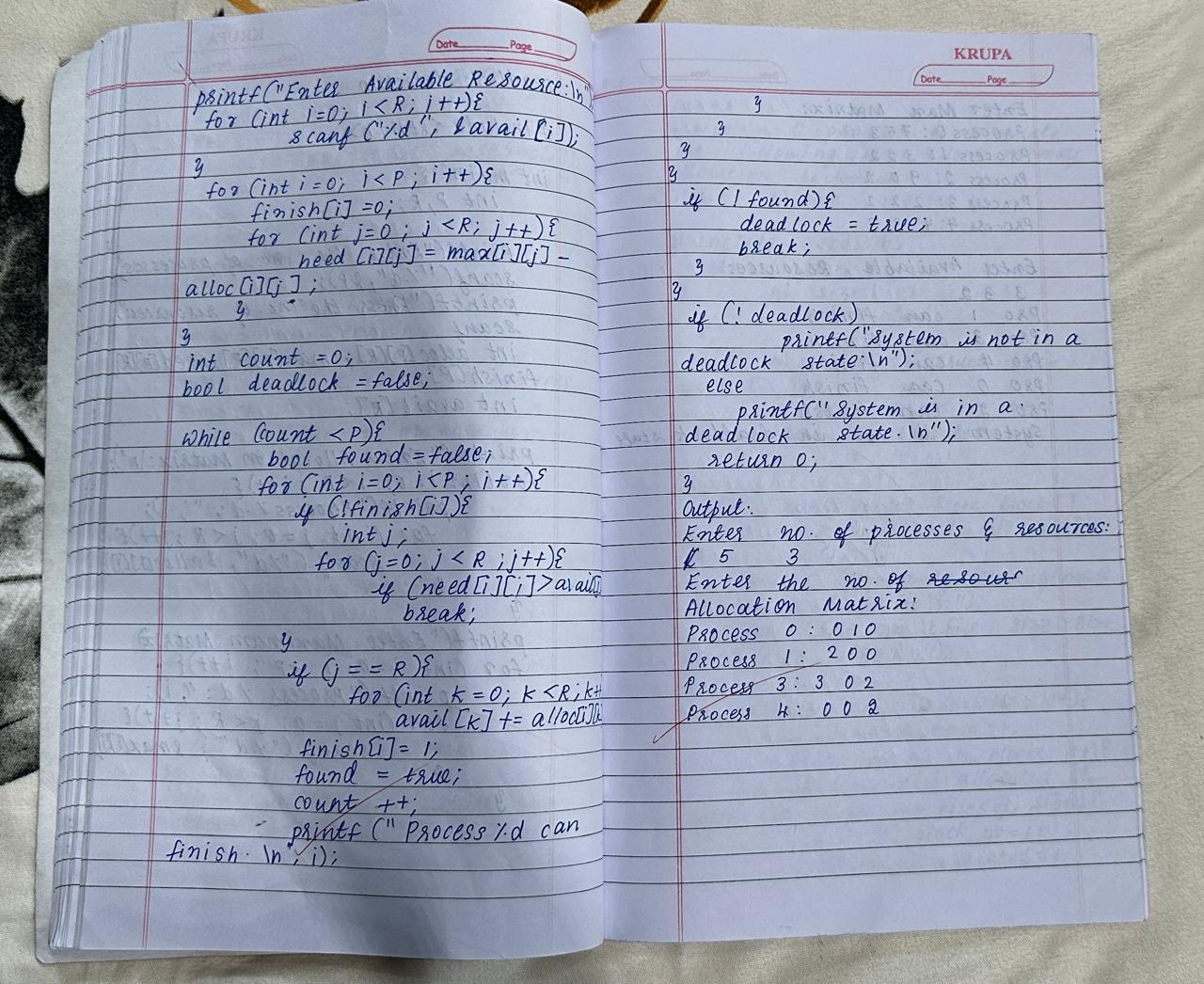
return 0;

}









Program - 6

Question:

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

→Worst Fit

→ Best Fit

→ First Fit

#include <stdio.h>

struct Block {

int size;

int allocated;

};

struct File {

int size;

int block\_no;

};

void resetBlocks(struct Block blocks[], int n) {

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

blocks[i].allocated = 0;

}

}

void firstFit(struct Block blocks[], int n\_blocks, struct File files[], int n\_files) {

printf("\n\tMemory Management Scheme – First Fit\n");

printf("File\_no:\tFile\_size\tBlock\_no:\tBlock\_size:\n");

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

files[i].block\_no = -1;

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

if (!blocks[j].allocated && blocks[j].size >= files[i].size) {

files[i].block\_no = j + 1;

blocks[j].allocated = 1;

printf("%d\t\t%d\t\t%d\t\t%d\n", i + 1, files[i].size, j + 1, blocks[j].size);

break;

}

}

if (files[i].block\_no == -1) {

printf("%d\t\t%d\t\t\_\t\t\_\n", i + 1, files[i].size);

}

}

}

void bestFit(struct Block blocks[], int n\_blocks, struct File files[], int n\_files) {

printf("\n\tMemory Management Scheme – Best Fit\n");

printf("File\_no:\tFile\_size\tBlock\_no:\tBlock\_size:\n");

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

int bestIdx = -1;

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

if (!blocks[j].allocated && blocks[j].size >= files[i].size) {

if (bestIdx == -1 || blocks[j].size < blocks[bestIdx].size) {

bestIdx = j;

}

}

}

if (bestIdx != -1) {

blocks[bestIdx].allocated = 1;

files[i].block\_no = bestIdx + 1;

printf("%d\t\t%d\t\t%d\t\t%d\n", i + 1, files[i].size, bestIdx + 1, blocks[bestIdx].size);

} else {

printf("%d\t\t%d\t\t\_\t\t\_\n", i + 1, files[i].size);

}

}

}

void worstFit(struct Block blocks[], int n\_blocks, struct File files[], int n\_files) {

printf("\n\tMemory Management Scheme – Worst Fit\n");

printf("File\_no:\tFile\_size\tBlock\_no:\tBlock\_size:\n");

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

int worstIdx = -1;

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

if (!blocks[j].allocated && blocks[j].size >= files[i].size) {

if (worstIdx == -1 || blocks[j].size > blocks[worstIdx].size) {

worstIdx = j;

}

}

}

if (worstIdx != -1) {

blocks[worstIdx].allocated = 1;

files[i].block\_no = worstIdx + 1;

printf("%d\t\t%d\t\t%d\t\t%d\n", i + 1, files[i].size, worstIdx + 1, blocks[worstIdx].size);

} else {

printf("%d\t\t%d\t\t\_\t\t\_\n", i + 1, files[i].size);

}

}

}

int main() {

int n\_blocks, n\_files, choice;

printf("Memory Management Scheme\n");

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

scanf("%d", &n\_blocks);

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

scanf("%d", &n\_files);

struct Block blocks[n\_blocks];

struct File files[n\_files];

printf("\nEnter the size of the blocks:\n");

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

printf("Block %d: ", i + 1);

scanf("%d", &blocks[i].size);

blocks[i].allocated = 0;

}

printf("Enter the size of the files:\n");

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

printf("File %d: ", i + 1);

scanf("%d", &files[i].size);

}

do {

printf("\n1. First Fit\n2. Best Fit\n3. Worst Fit\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

resetBlocks(blocks, n\_blocks); // Reset block allocation before each strategy

switch (choice) {

case 1:

firstFit(blocks, n\_blocks, files, n\_files);

break;

case 2:

bestFit(blocks, n\_blocks, files, n\_files);

break;

case 3:

worstFit(blocks, n\_blocks, files, n\_files);

break;

case 4:

printf("\nExiting...\n");

break;

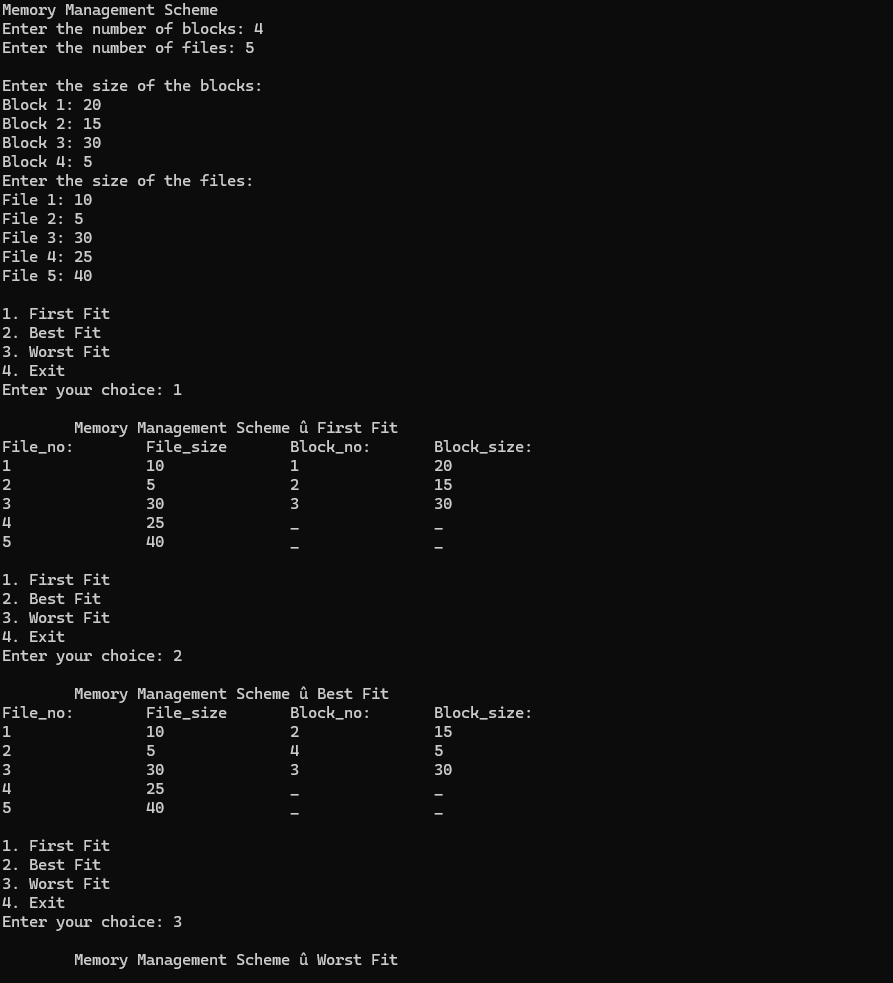
default:

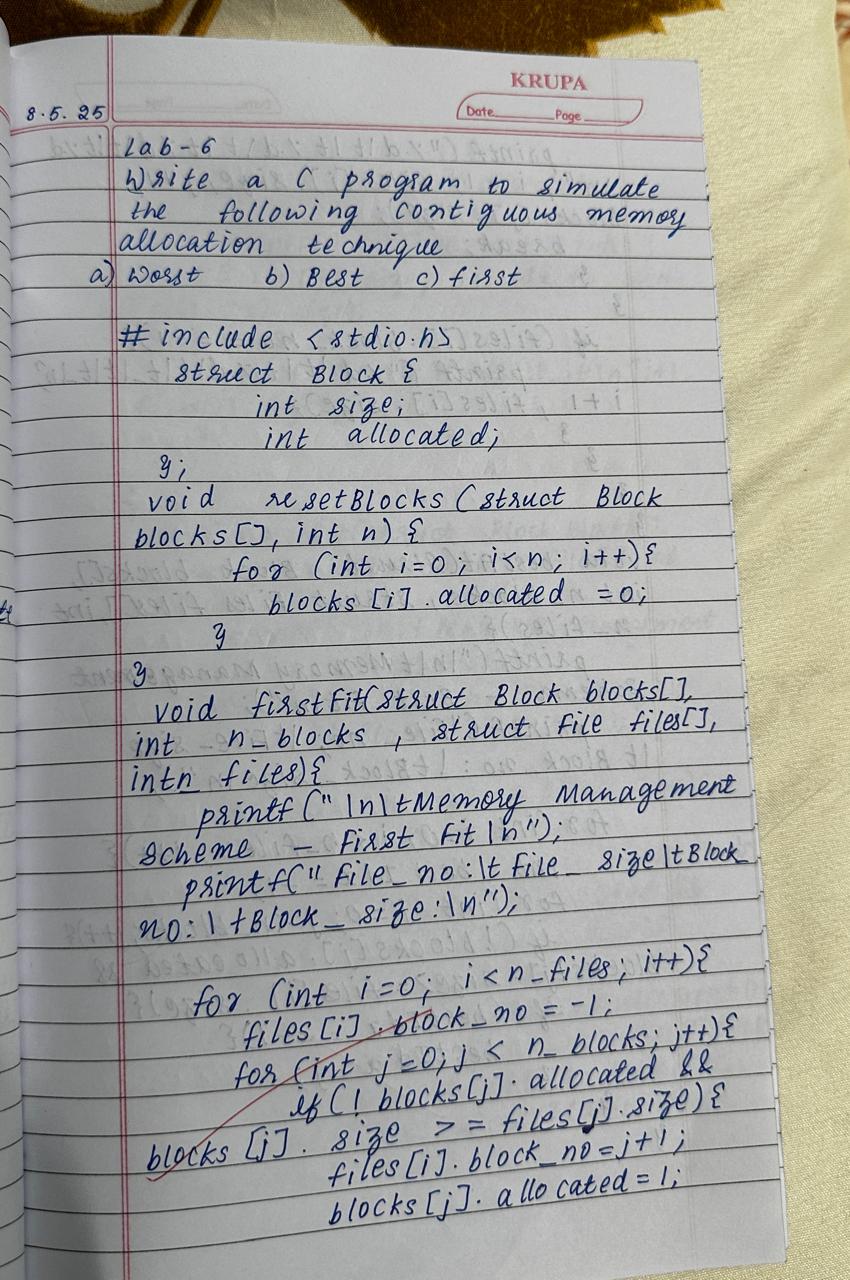
printf("Invalid choice.\n");

}

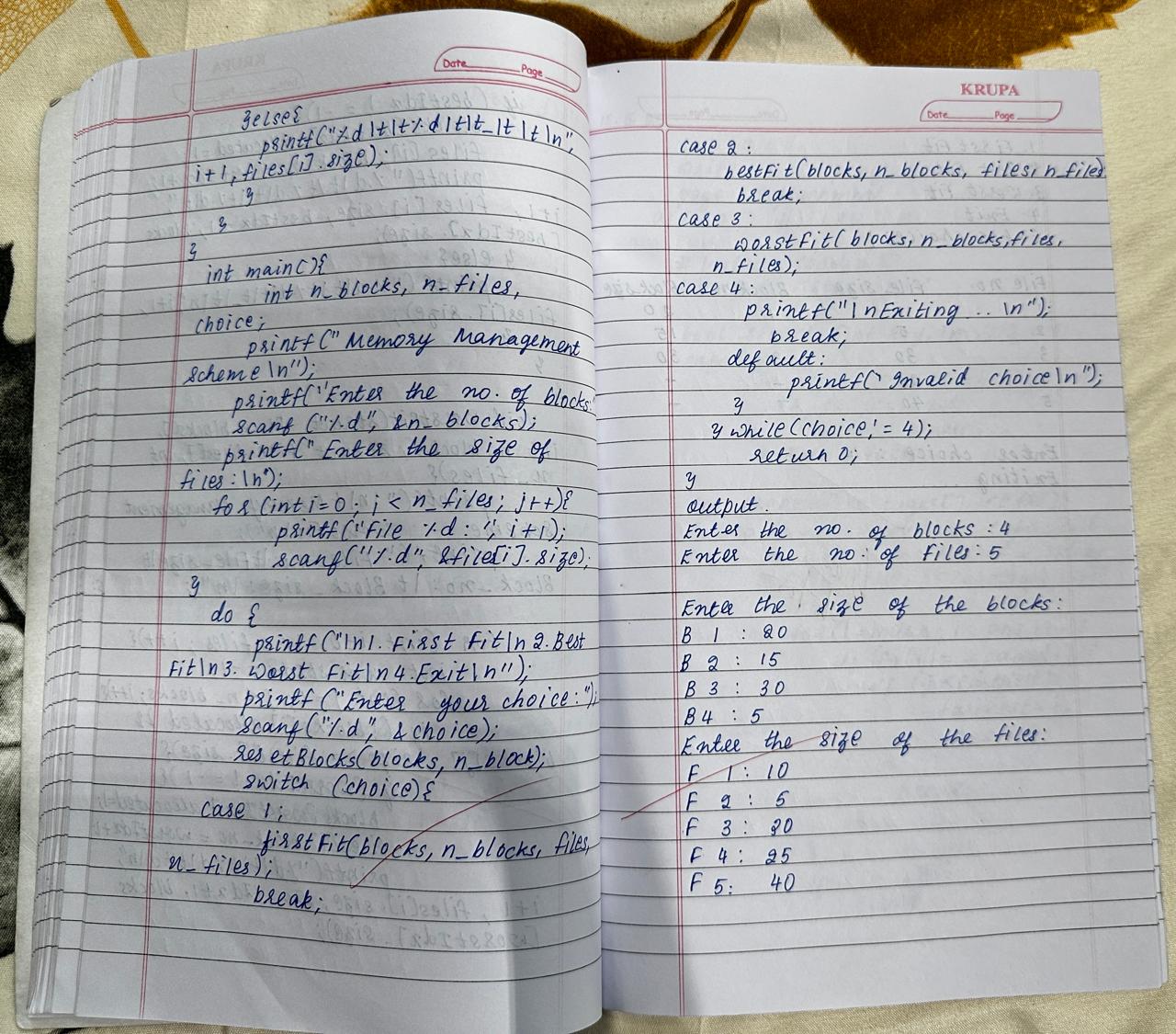
} while (choice != 4);

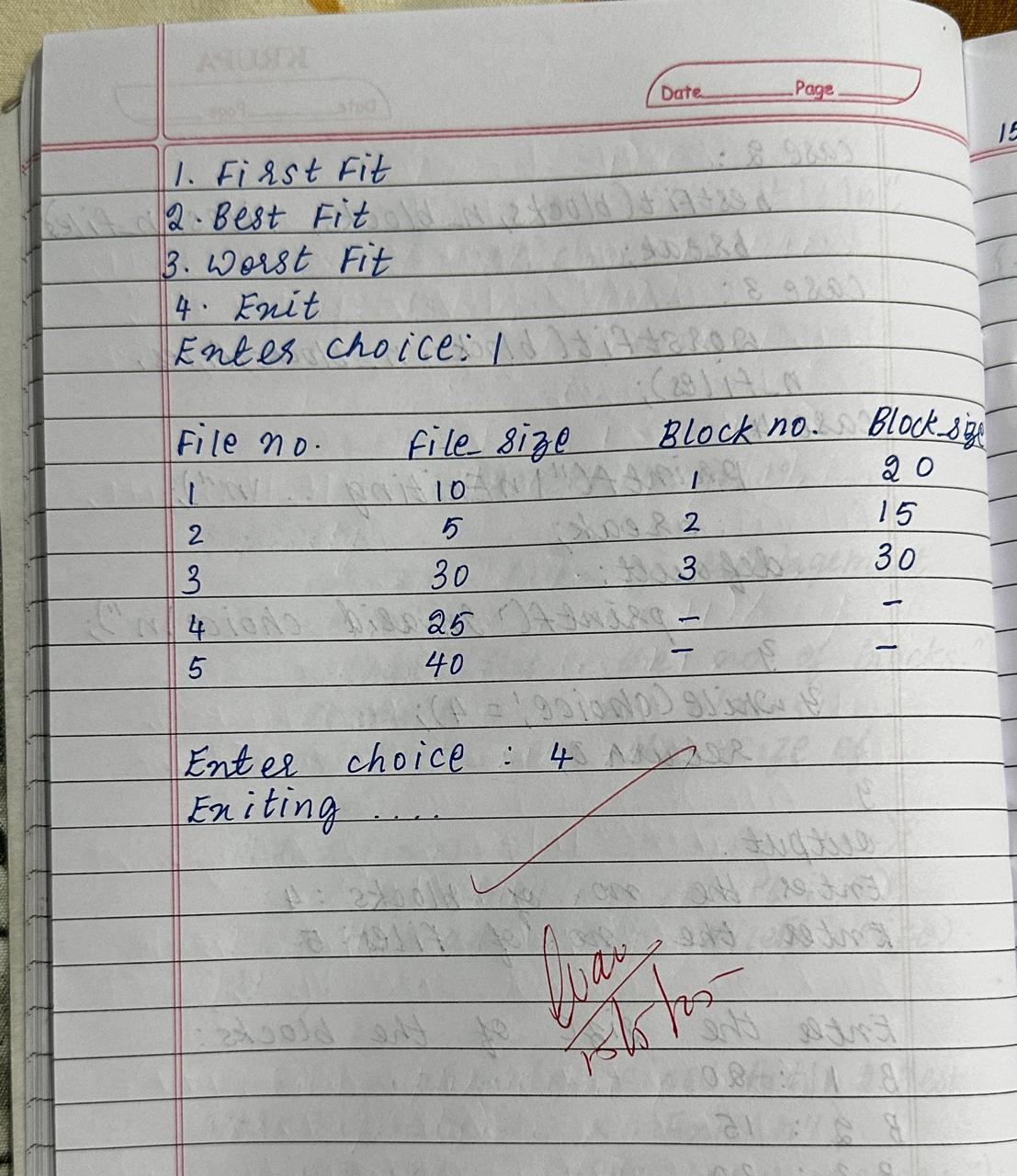
return 0;











Program - 7

Question:

Write a C program to simulate the following page replacement algorithms

→FIFO

→ LRU

→ Optimal

#include <stdio.h>

#include <stdbool.h>

bool search(int key, int fr[], int capacity) {

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

if (fr[i] == key)

return true;

return false;

}

int predict(int pages[], int fr[], int pn, int index, int capacity) {

int res = -1, farthest = index;

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

int j;

for (j = index; j < pn; j++) {

if (fr[i] == pages[j]) {

if (j > farthest) {

farthest = j;

res = i;

}

break;

}

}

if (j == pn)

return i;

}

return (res == -1) ? 0 : res;

}

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

int fr[capacity];

int page\_faults = 0, page\_hits = 0;

int index = 0;

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

if (!search(pages[i], fr, capacity)) {

fr[index] = pages[i];

index = (index + 1) % capacity;

page\_faults++;

} else {

page\_hits++;

}

}

printf("FIFO Page Faults: %d, Page Hits: %d\n", page\_faults, page\_hits);

}

void optimal(int pages[], int n, int capacity) {

int fr[capacity];

int page\_faults = 0, page\_hits = 0;

int filled = 0;

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

if (search(pages[i], fr, capacity)) {

page\_hits++;

continue;

}

if (filled < capacity) {

fr[filled++] = pages[i];

} else {

int j = predict(pages, fr, n, i + 1, capacity);

fr[j] = pages[i];

}

page\_faults++;

}

printf("Optimal Page Faults: %d, Page Hits: %d\n", page\_faults, page\_hits);

}

void lru(int pages[], int n, int capacity) {

int fr[capacity];

int recent[capacity];

int page\_faults = 0, page\_hits = 0;

int time = 0;

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

fr[i] = -1;

recent[i] = -1;

}

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

bool hit = false;

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

if (fr[j] == pages[i]) {

time++;

recent[j] = time;

page\_hits++;

hit = true;

break;

}

}

if (!hit) {

int lru = 0;

for (int j = 1; j < capacity; j++)

if (recent[j] < recent[lru])

lru = j;

time++;

fr[lru] = pages[i];

recent[lru] = time;

page\_faults++;

}

}

printf("LRU Page Faults: %d, Page Hits: %d\n", page\_faults, page\_hits);

}

int main() {

int n, capacity;

printf("Enter the size of the pages:\n");

scanf("%d", &n);

int pages[n];

printf("Enter the page strings:\n");

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

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

printf("Enter the no of page frames:\n");

scanf("%d", &capacity);

fifo(pages, n, capacity);

optimal(pages, n, capacity);

lru(pages, n, capacity);

return 0;

}

