VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



## LAB REPORT

on

OPERATING SYSTEMS

Submitted by

TEJAS JOSHI (1WA23CS016)

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 Tejas Joshi (1WA23CS016), 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

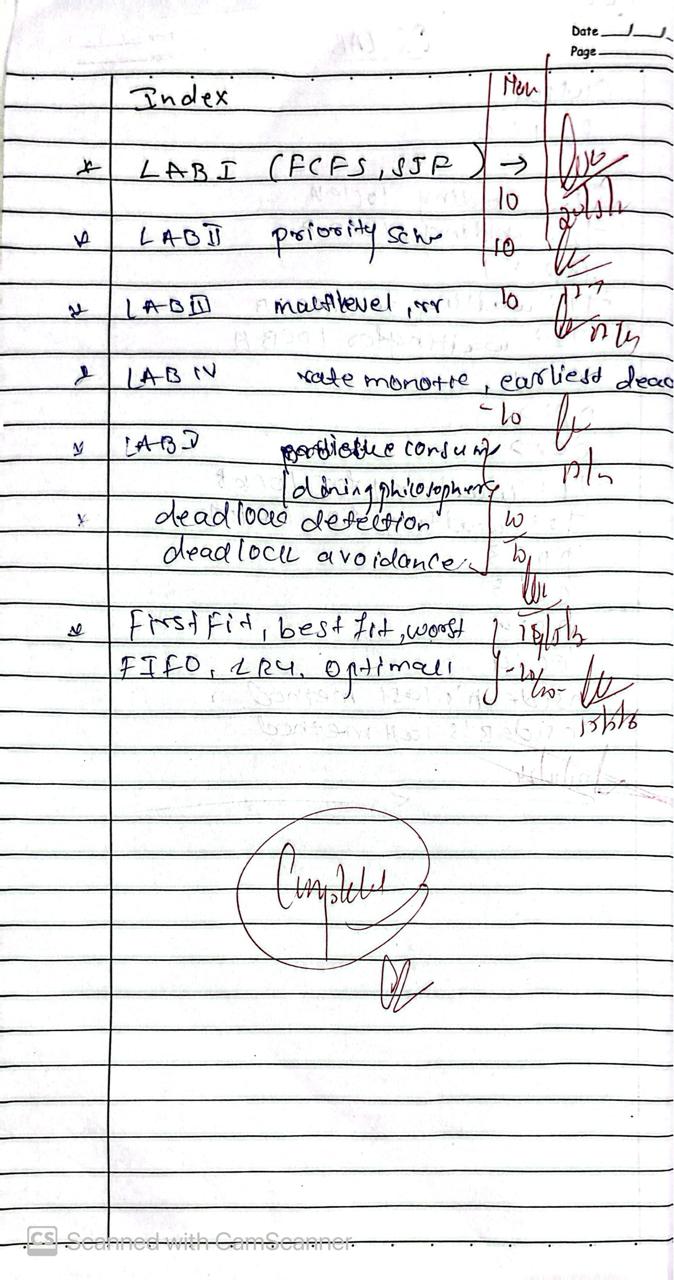
|  |  |  |
| --- | --- | --- |
| Sl.  No. | Experiment Title | Page No. |
| 1. | Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time.  →FCFS  → SJF (pre-emptive & Non-preemptive) | 1-10 |
| 2. | Write a C program to simulate the 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   1. Best-fit 2. 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: ");

5

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);

}

**=>SJF(NON-PRE-EMPTIVE):**

#include <stdio.h>

typedef struct {

int pid;

int burst\_time;

int waiting\_time;

int turnaround\_time;

} Process;

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

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

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

if (p[j].burst\_time > p[j + 1].burst\_time) {

Process temp = p[j];

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

p[j + 1] = temp;

}

}

}

}

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

p[0].waiting\_time = 0;

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

p[i].turnaround\_time = p[i].waiting\_time + p[i].burst\_time;

}

}

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

printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");

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

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

}

}

int main() {

Process processes1[] = {{1, 6}, {2, 8}, {3, 7}, {4, 3}};

int n1 = sizeof(processes1) / sizeof(processes1[0]);

printf("Test Case 1:\n");

sortByBurstTime(processes1, n1);

calculateTimes(processes1, n1);

printProcesses(processes1, n1);

Process processes2[] = {{1, 2}, {2, 4}, {3, 1}, {4, 3}};

int n2 = sizeof(processes2) / sizeof(processes2[0]);

printf("\nTest Case 2:\n");

sortByBurstTime(processes2, n2);

calculateTimes(processes2, n2);

printProcesses(processes2, n2);

    return 0;

}

**=>SJF(PRE-EMPTIVE):**

#include <stdio.h>

#include <limits.h>

typedef struct {

int pid;

int arrival\_time;

int burst\_time;

int remaining\_time;

int waiting\_time;

int turnaround\_time;

int completion\_time;

} Process;

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

int complete = 0, time = 0, shortest = -1;

int min\_time = INT\_MAX, finish\_time;

int is\_completed[n];

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

is\_completed[i] = 0;

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

}

while (complete < n) {

min\_time = INT\_MAX;

shortest = -1;

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

if (p[i].arrival\_time <= time && !is\_completed[i] && p[i].remaining\_time < min\_time) {

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

shortest = i;

}

}

if (shortest == -1) {

time++;

continue;

}

p[shortest].remaining\_time--;

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

complete++;

finish\_time = time + 1;

p[shortest].completion\_time = finish\_time;

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

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

is\_completed[shortest] = 1;

}

time++;

}

}

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

printf("\nProcess\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n");

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

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

}

}

int main() {

Process processes1[] = {{1, 0, 6}, {2, 2, 8}, {3, 4, 7}, {4, 5, 3}};

int n1 = sizeof(processes1) / sizeof(processes1[0]);

printf("Test Case 1:\n");

calculateTimes(processes1, n1);

printProcesses(processes1, n1);

Process processes2[] = {{1, 0, 3}, {2, 1, 1}, {3, 2, 6}, {4, 3, 2}};

int n2 = sizeof(processes2) / sizeof(processes2[0]);

printf("\nTest Case 2:\n");

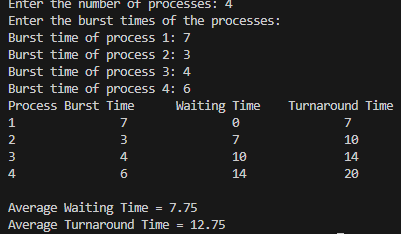
calculateTimes(processes2, n2);

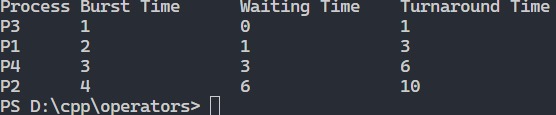
printProcesses(processes2, n2);

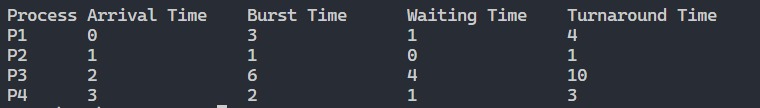
    return 0;

}

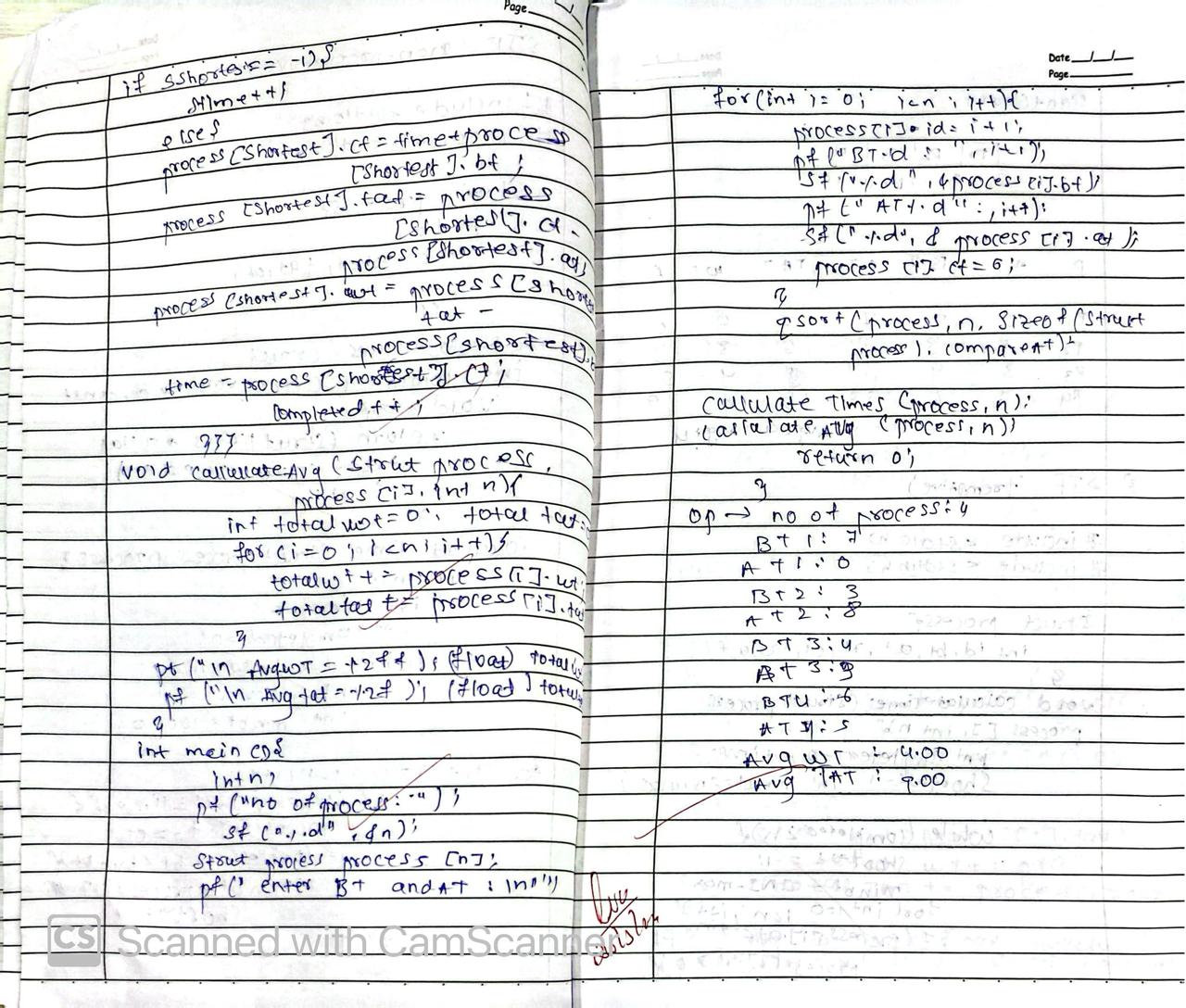
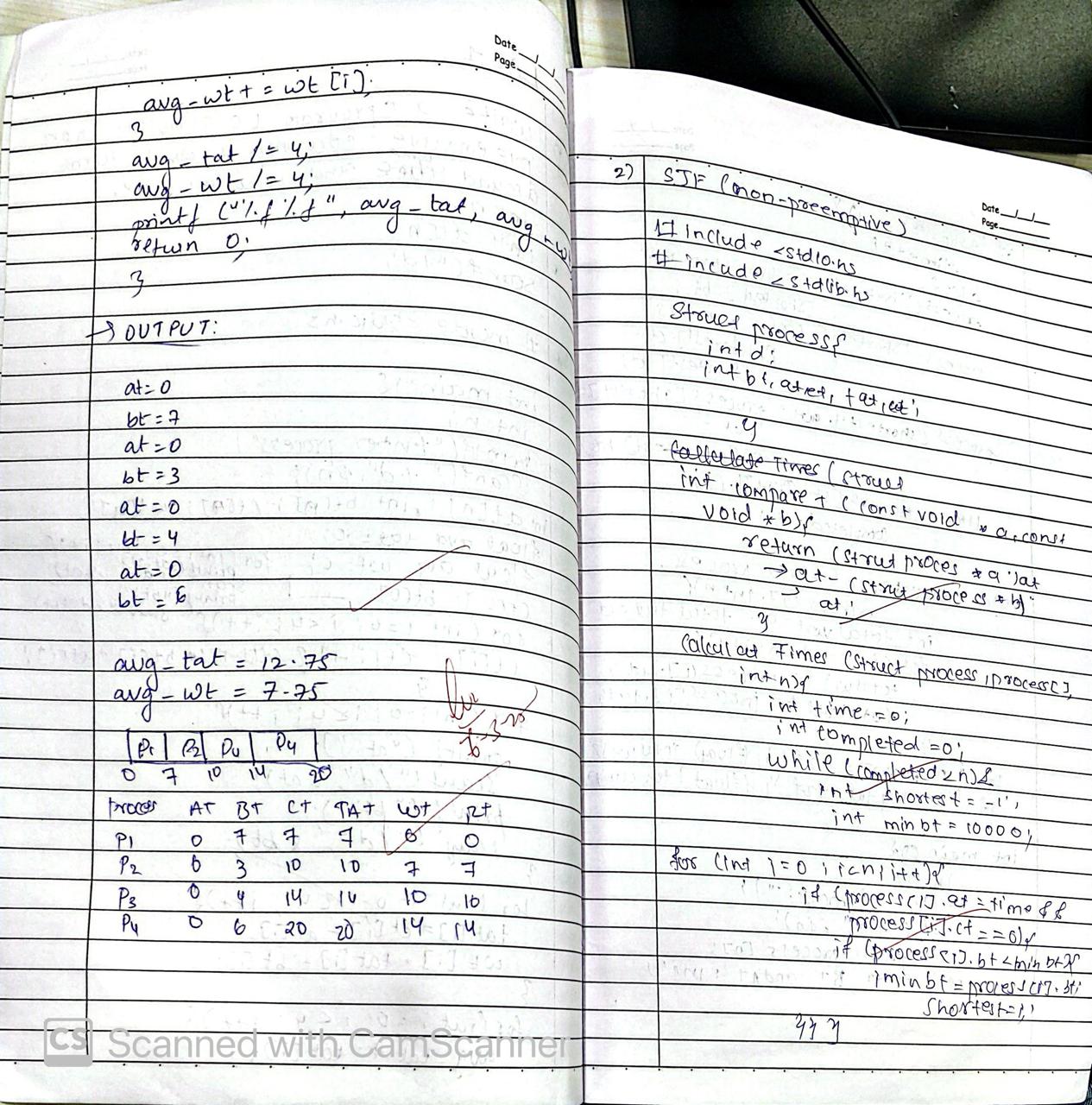
**OUTPUT:**

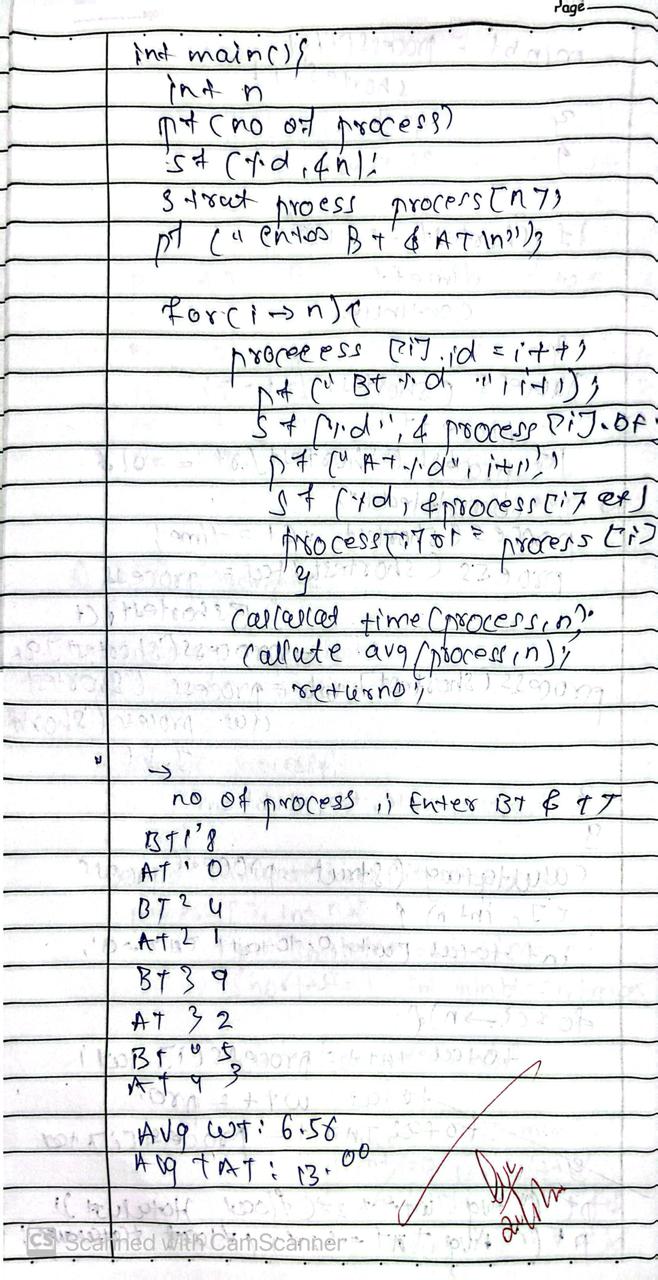
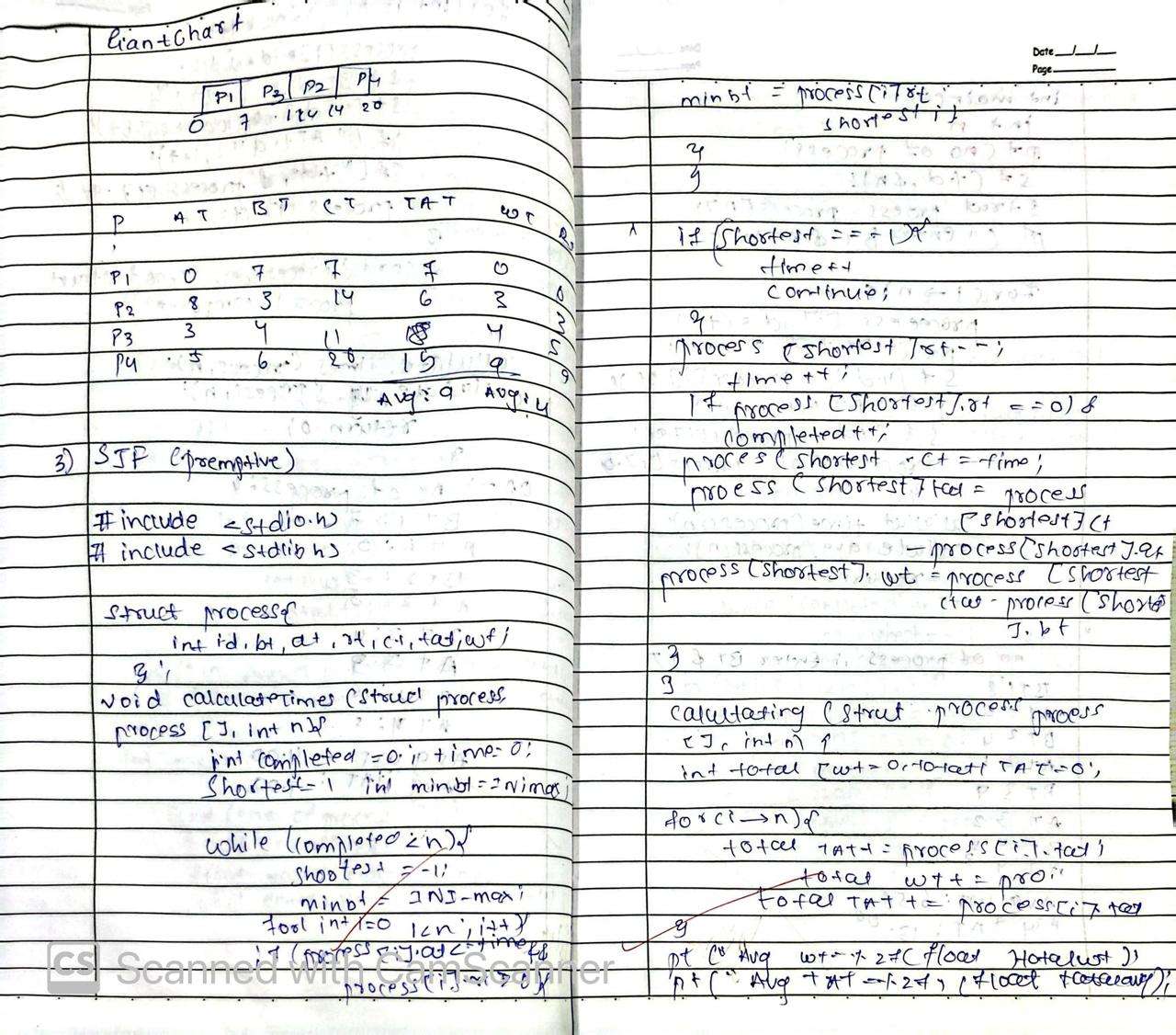
****











**Program – 2 Question:**

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)

**Code:**

**🡪PRIORITY (NON-PRE-EMPTIVE)**

#include<stdio.h>  
  
typedef struct {  
 int pid, bt, at, priority;  
} Process;  
  
void priorityScheduling() {  
 int n;  
 printf("Enter number of processes: ");  
 scanf("%d", &n);  
 Process p[n];  
 int i, j;  
 for(i = 0; i < n; i++) {  
 p[i].pid = i+1;  
 printf("Enter Arrival Time, Burst Time and Priority for process %d: ", i+1);  
 scanf("%d%d%d", &p[i].at, &p[i].bt, &p[i].priority);  
 }  
  
 // Sort based on priority (higher number = higher priority)  
 for(i = 0; i < n-1; i++) {  
 for(j = i+1; j < n; j++) {  
 if(p[i].priority < p[j].priority) {  
 Process temp = p[i];  
 p[i] = p[j];  
 p[j] = temp;  
 }  
 }  
 }  
  
 int wt[n], tat[n], ct = 0;  
 float avgwt = 0, avgtat = 0;  
 printf("\nPID\tAT\tBT\tPriority\tWT\tTAT\n");  
 for(i = 0; i < n; i++) {  
 if(ct < p[i].at) ct = p[i].at;  
 wt[i] = ct - p[i].at;  
 ct += p[i].bt;  
 tat[i] = wt[i] + p[i].bt;  
 avgwt += wt[i];  
 avgtat += tat[i];  
 printf("%d\t%d\t%d\t%d\t\t%d\t%d\n", p[i].pid, p[i].at, p[i].bt, p[i].priority, wt[i], tat[i]);  
 }  
 printf("Average Waiting Time: %.2f\n", avgwt/n);  
 printf("Average Turnaround Time: %.2f\n", avgtat/n);  
}  
int main() {  
 printf("Priority Scheduling (Non-Preemptive)\n");  
 priorityScheduling();  
}

🡪**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);

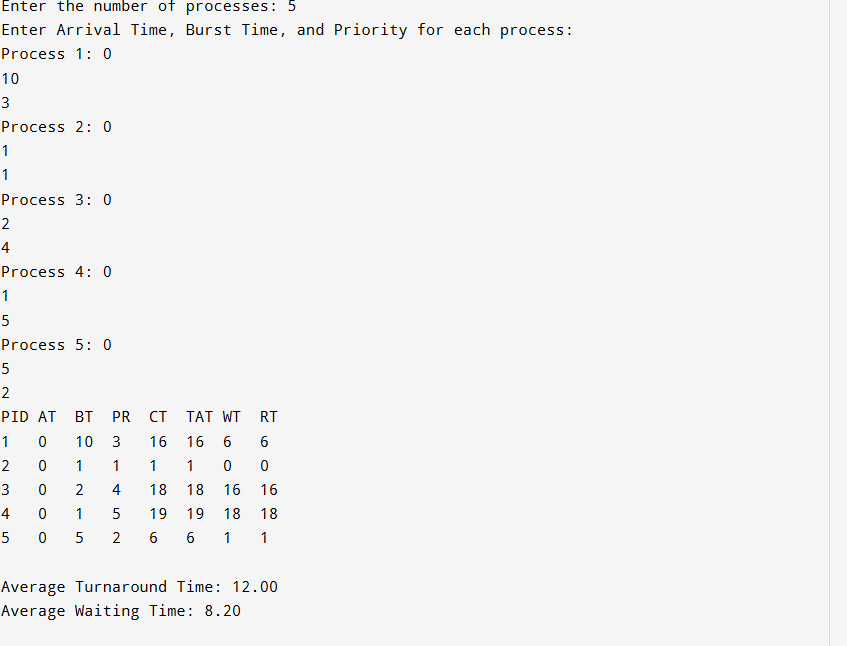
}

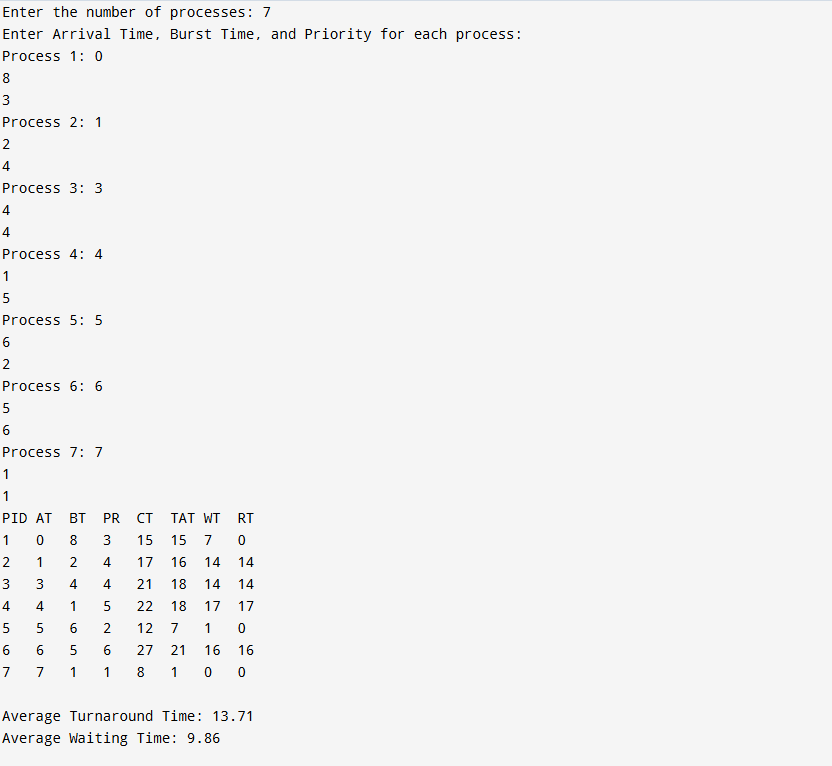
findPreemptivePriorityScheduling(p, n);

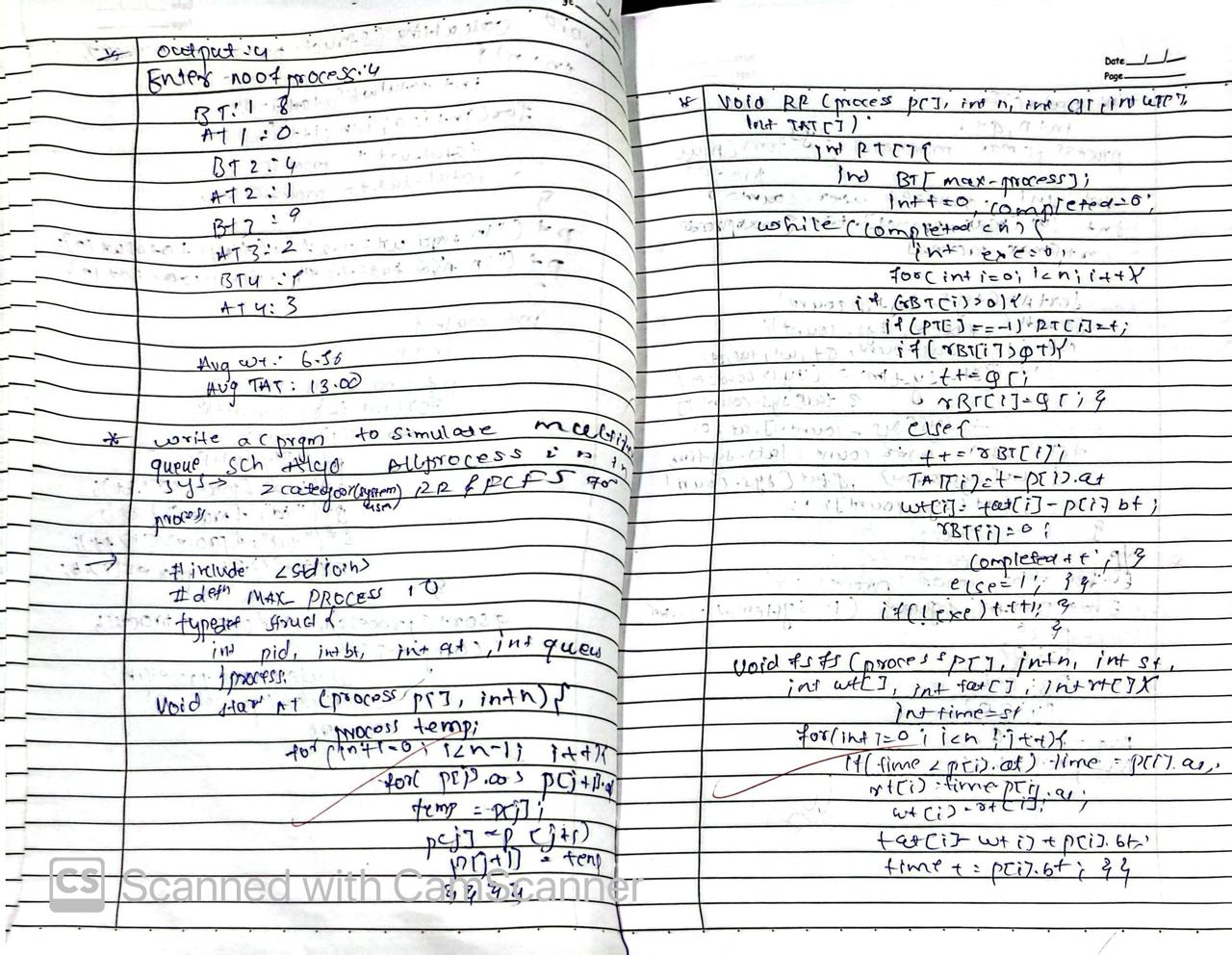
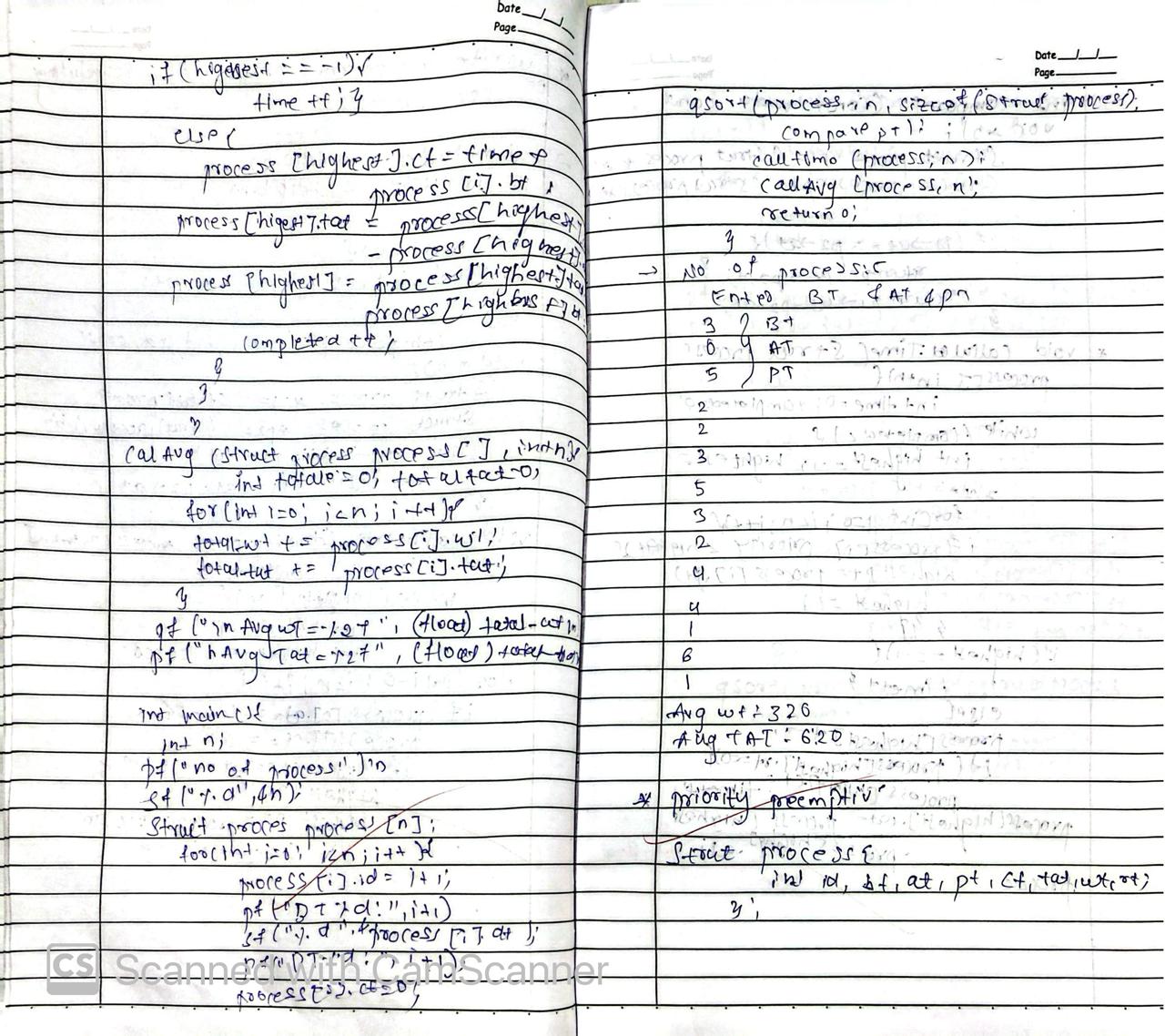
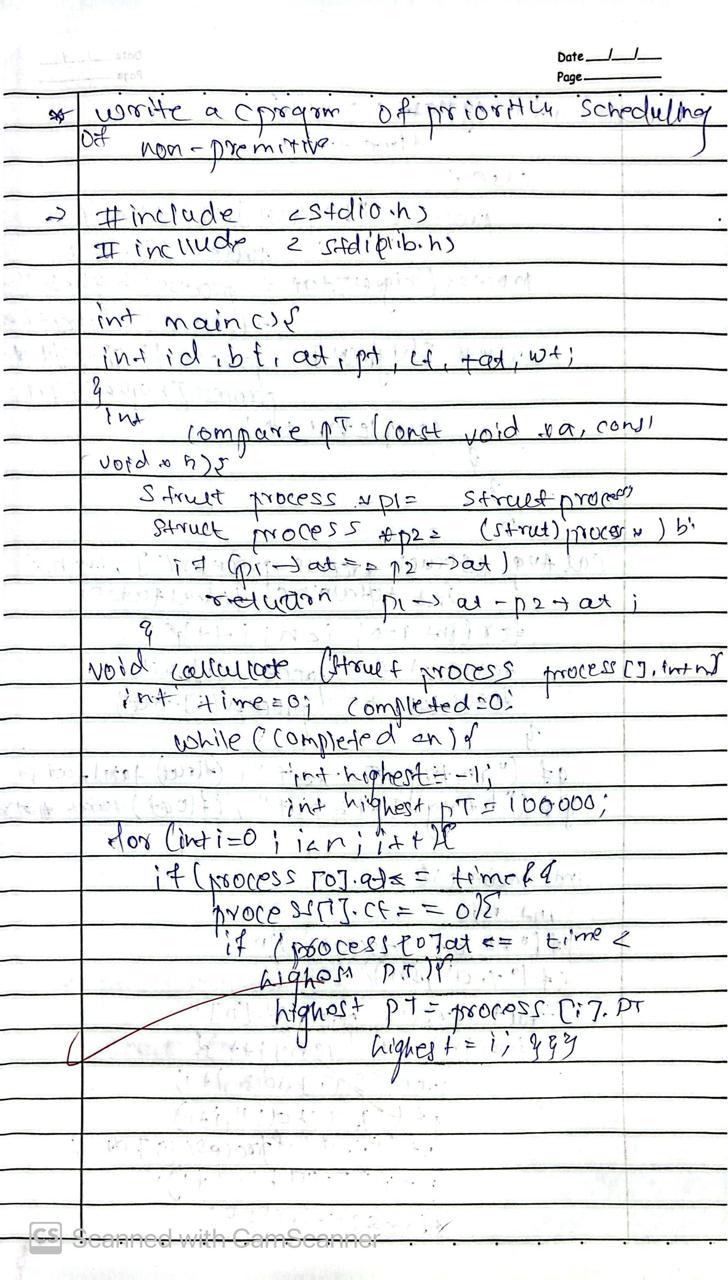
    return 0;

}

**OUTPUT:**







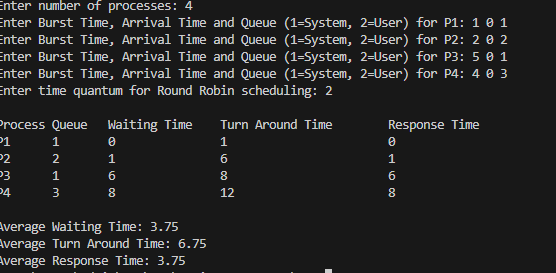
**Program - 3 Question:**

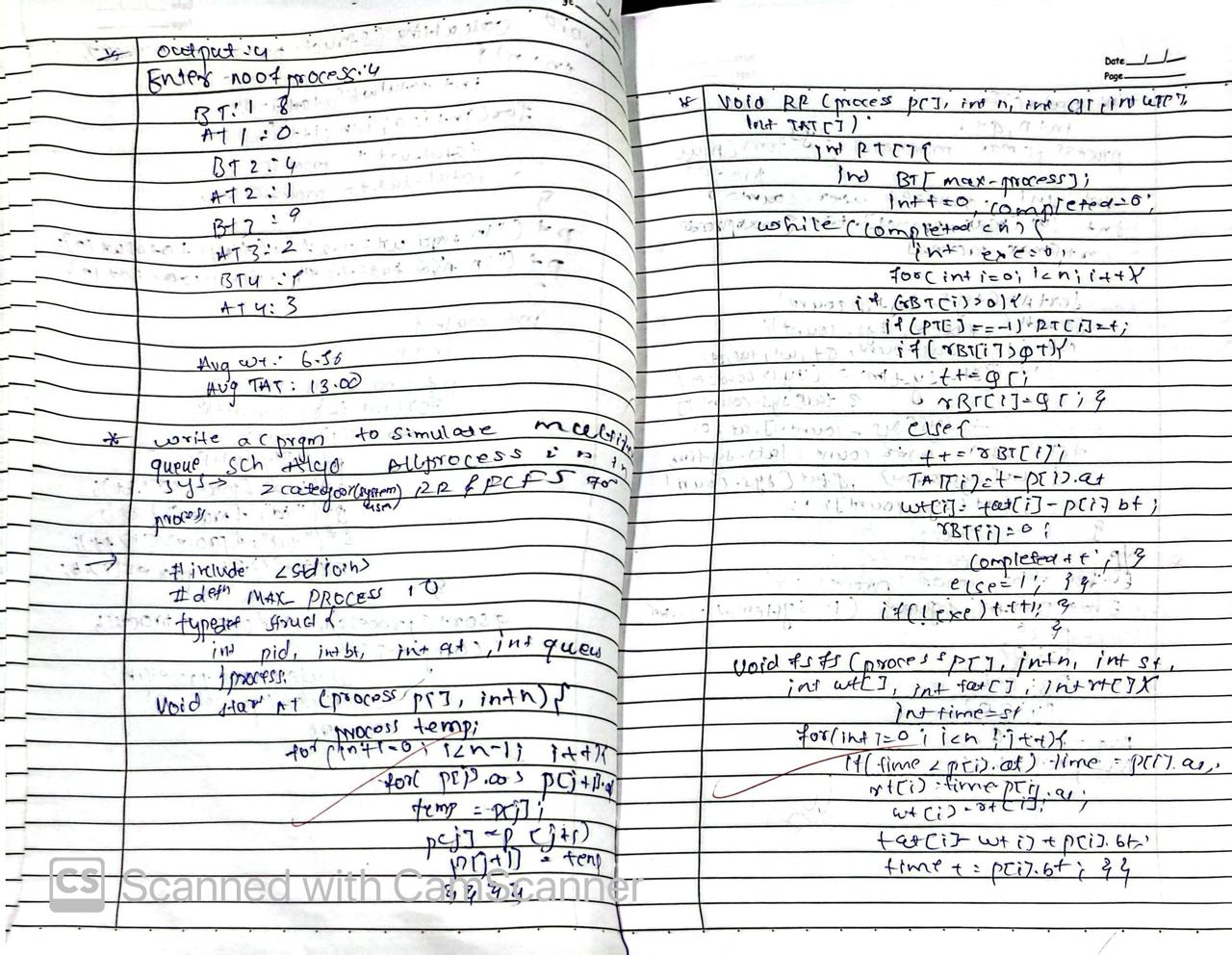
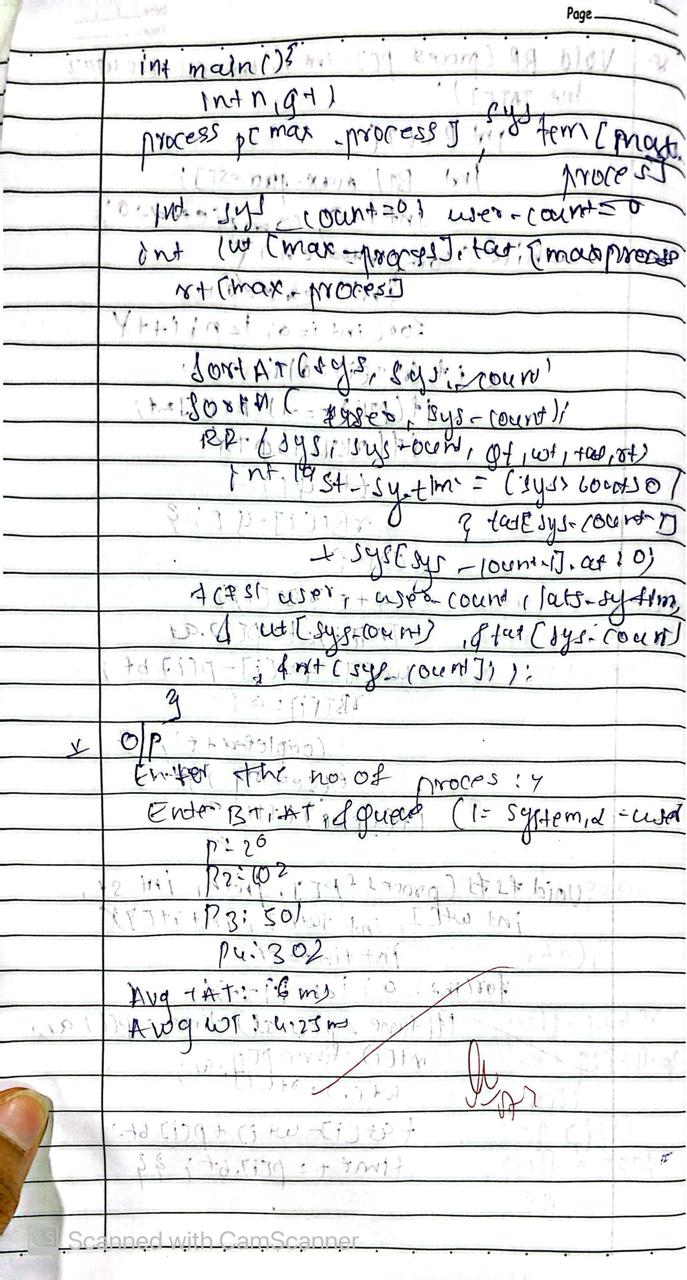
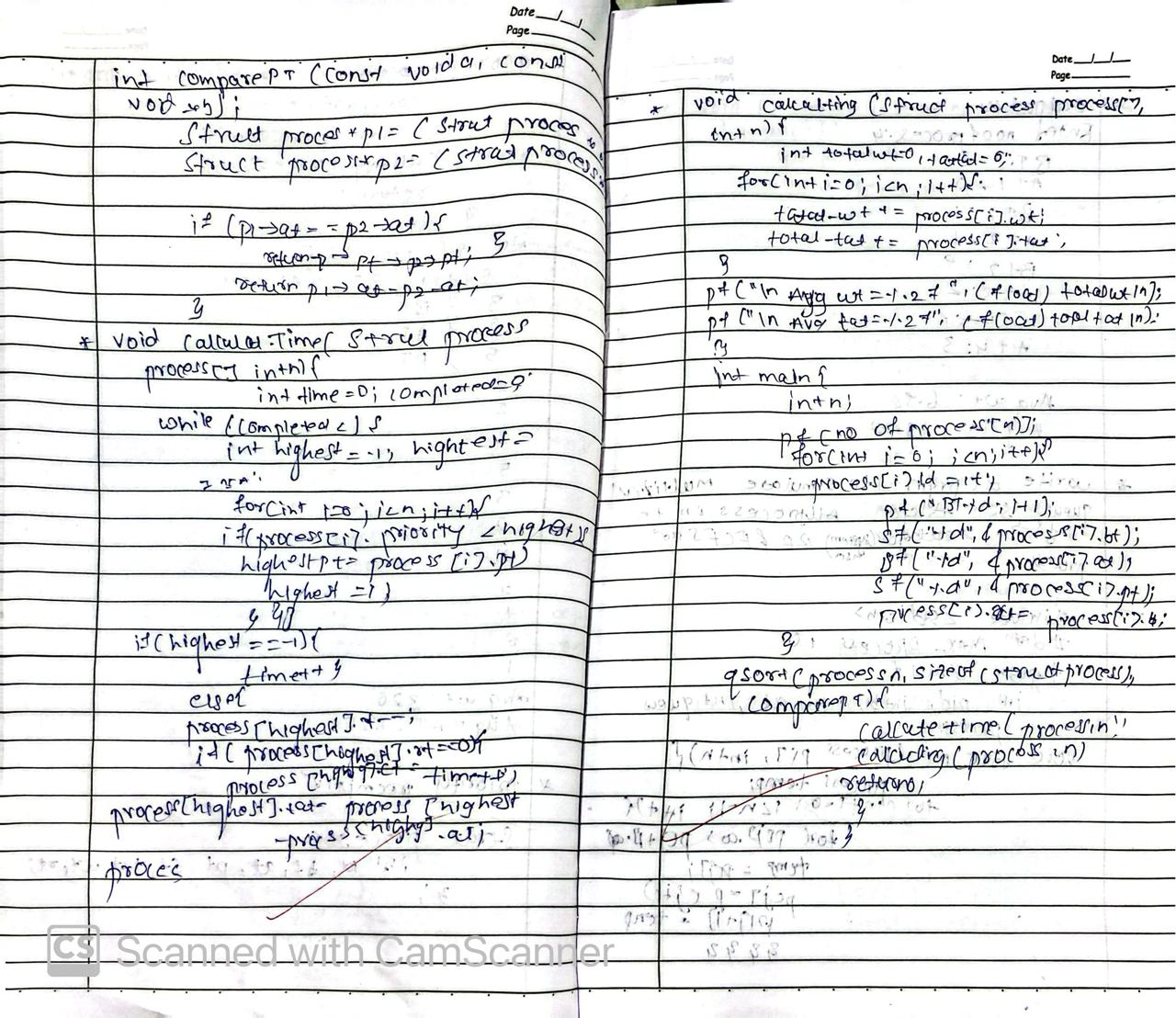
Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario.  
All the processes in the system are divided into two categories – system processes and user processes.  
System processes are to be given higher priority than user processes.  
Use FCFS scheduling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

**Code:**

#include <stdio.h>  
  
typedef struct {  
 int pid, bt, at;  
} Process;  
  
void fcfs(Process p[], int n, char \*queue\_type) {  
 int wt[n], tat[n], ct = 0;  
 float avgwt = 0, avgtat = 0;  
 printf("\n%s Queue:\n", queue\_type);  
 printf("PID\tAT\tBT\tWT\tTAT\n");  
 for (int i = 0; i < n; i++) {  
 if (ct < p[i].at) ct = p[i].at;  
 wt[i] = ct - p[i].at;  
 ct += p[i].bt;  
 tat[i] = wt[i] + p[i].bt;  
 avgwt += wt[i];  
 avgtat += tat[i];  
 printf("%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].at, p[i].bt, wt[i], tat[i]);  
 }  
 printf("Average Waiting Time: %.2f\n", avgwt / n);  
 printf("Average Turnaround Time: %.2f\n", avgtat / n);  
}  
  
int main() {  
 int n\_sys, n\_user;  
 printf("Enter number of system processes: ");  
 scanf("%d", &n\_sys);  
 Process sys[n\_sys];  
 for (int i = 0; i < n\_sys; i++) {  
 sys[i].pid = i + 1;  
 printf("Enter AT and BT for system process %d: ", i + 1);  
 scanf("%d%d", &sys[i].at, &sys[i].bt);  
 }  
 printf("Enter number of user processes: ");  
 scanf("%d", &n\_user);  
 Process user[n\_user];  
 for (int i = 0; i < n\_user; i++) {  
 user[i].pid = i + 1 + n\_sys;  
 printf("Enter AT and BT for user process %d: ", i + 1);  
 scanf("%d%d", &user[i].at, &user[i].bt);  
 }  
 fcfs(sys, n\_sys, "System");  
 fcfs(user, n\_user, "User");  
 return 0;  
}

**OUTPUT:**



**Program - 4 Question**

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

**Code:**

**🡪RATE MONOTONIC**

#include <stdio.h>  
#include <stdlib.h>  
  
#define MAX\_PROCESSES 10  
#define MAX\_TIME 50  
  
typedef struct {  
 int pid;  
 int burst;  
 int period;  
 int remaining\_time;  
 int next\_arrival;  
} Process;  
  
void rate\_monotonic\_scheduling(Process p[], int n) {  
 int time = 0, executed;  
 printf("\nRate Monotonic Scheduling:\n");  
 while (time < MAX\_TIME) {  
 executed = -1;  
 for (int i = 0; i < n; i++) {  
 if (p[i].next\_arrival <= time && p[i].remaining\_time > 0) {  
 if (executed == -1 || p[i].period < p[executed].period)  
 executed = i;  
 }  
 }  
 if (executed != -1) {  
 printf("%dms : process %d is running.\n", time, p[executed].pid);  
 p[executed].remaining\_time--;  
 if (p[executed].remaining\_time == 0) {  
 p[executed].next\_arrival += p[executed].period;  
 p[executed].remaining\_time = p[executed].burst;  
 }  
 }  
 time++;  
 }  
}

**🡪EARLIEST DEADLINE**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_PROCESSES 10

#define MAX\_TIME 50 // Maximum simulation time

typedef struct {

int pid;

int burst;

int deadline;

int period;

int remaining\_time;

int next\_arrival;

} Process;

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

int time = 0, executed;

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

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

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

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

while (time < MAX\_TIME) {

executed = -1;

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

if (p[i].next\_arrival <= time && p[i].remaining\_time > 0) {

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

executed = i;

}

}

if (executed != -1) {

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

p[executed].remaining\_time--;

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

p[executed].next\_arrival += p[executed].period;

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

}

}

time++;

}

}

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].pid = i + 1;

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

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

}

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

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

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

processes[i].next\_arrival = 0;

}

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

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

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

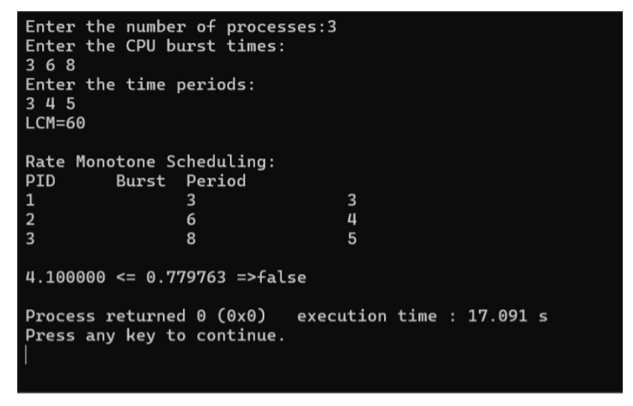
}

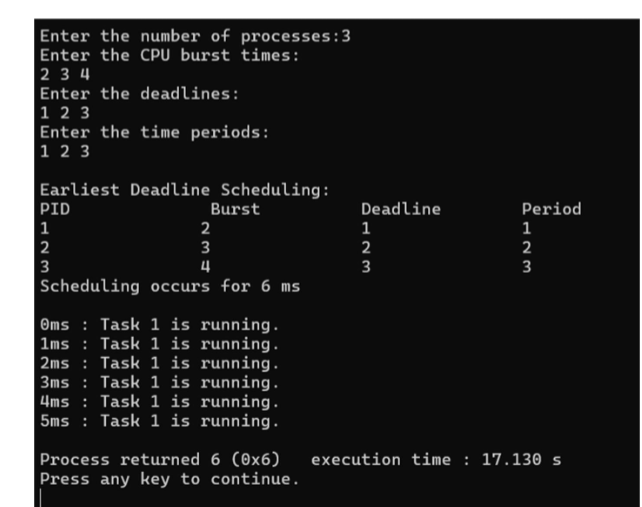
earliest\_deadline\_first(processes, n);

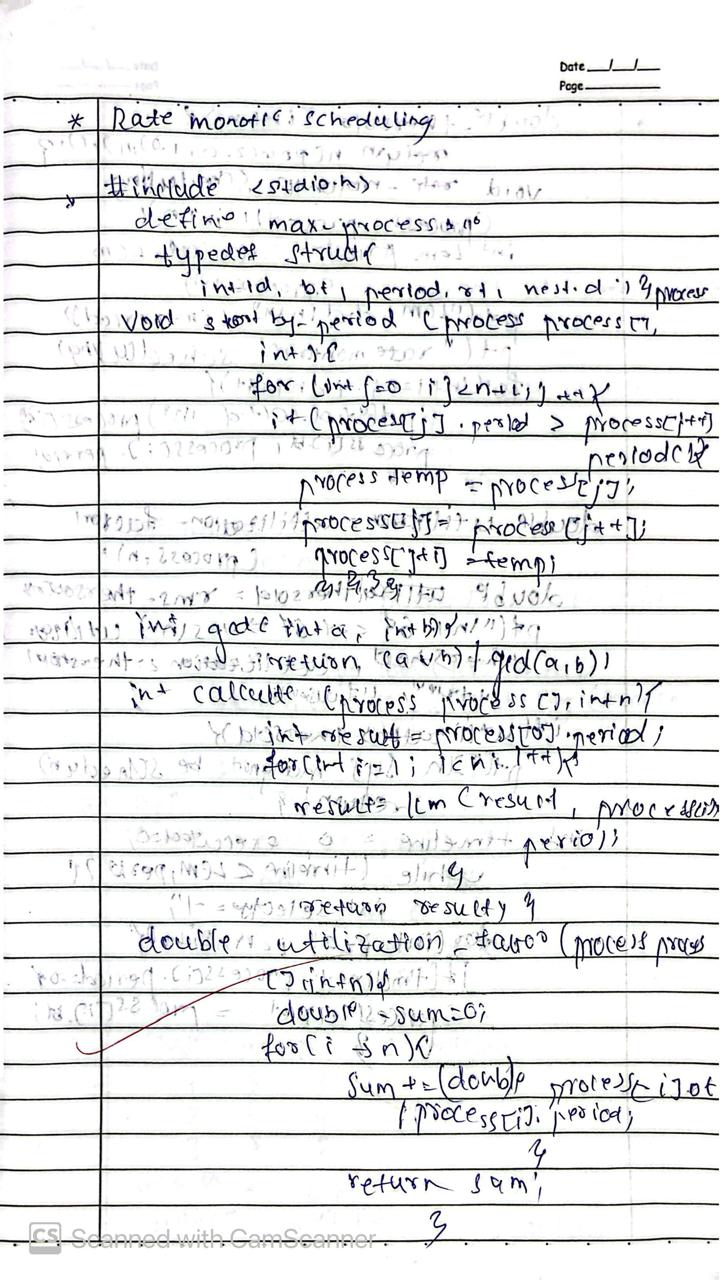
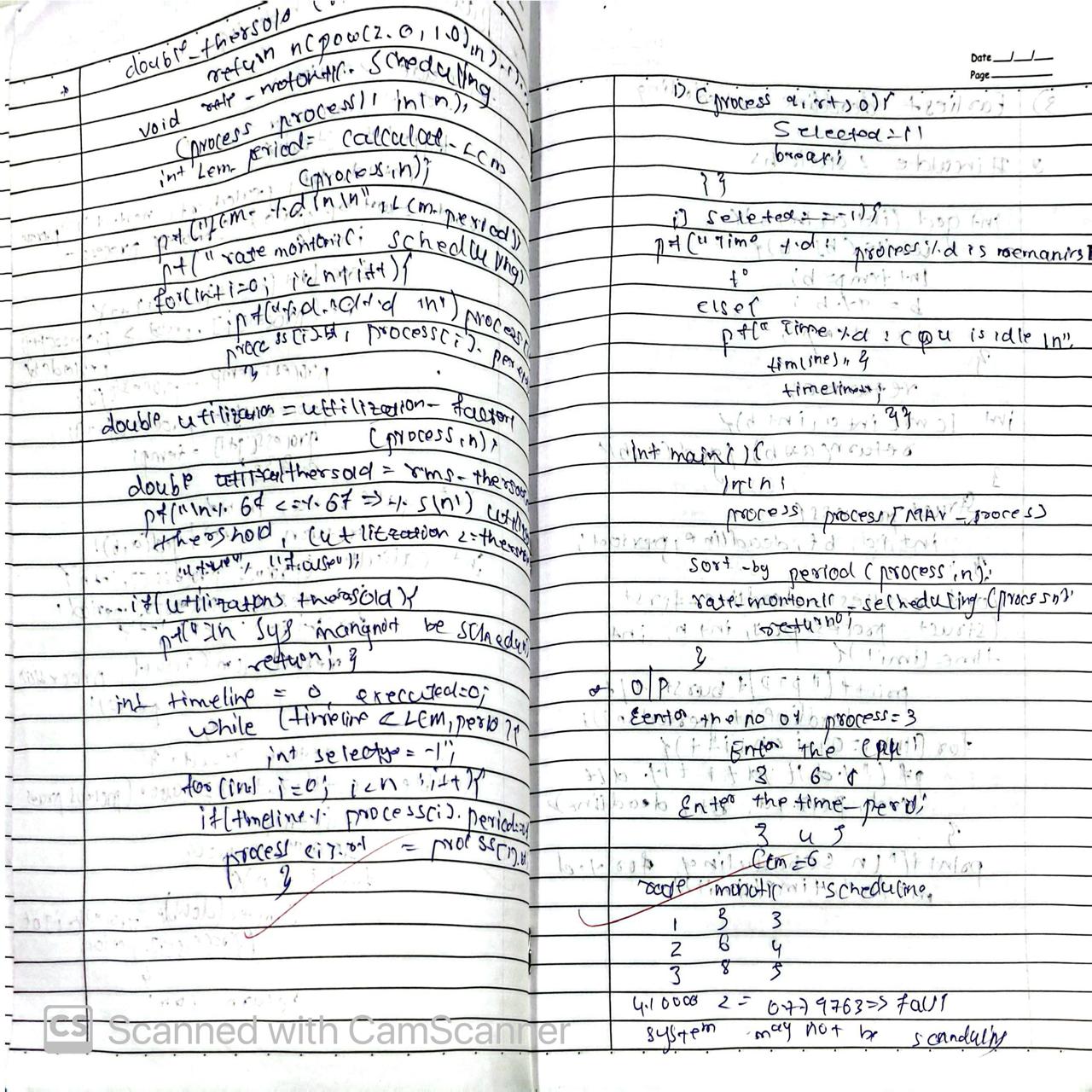
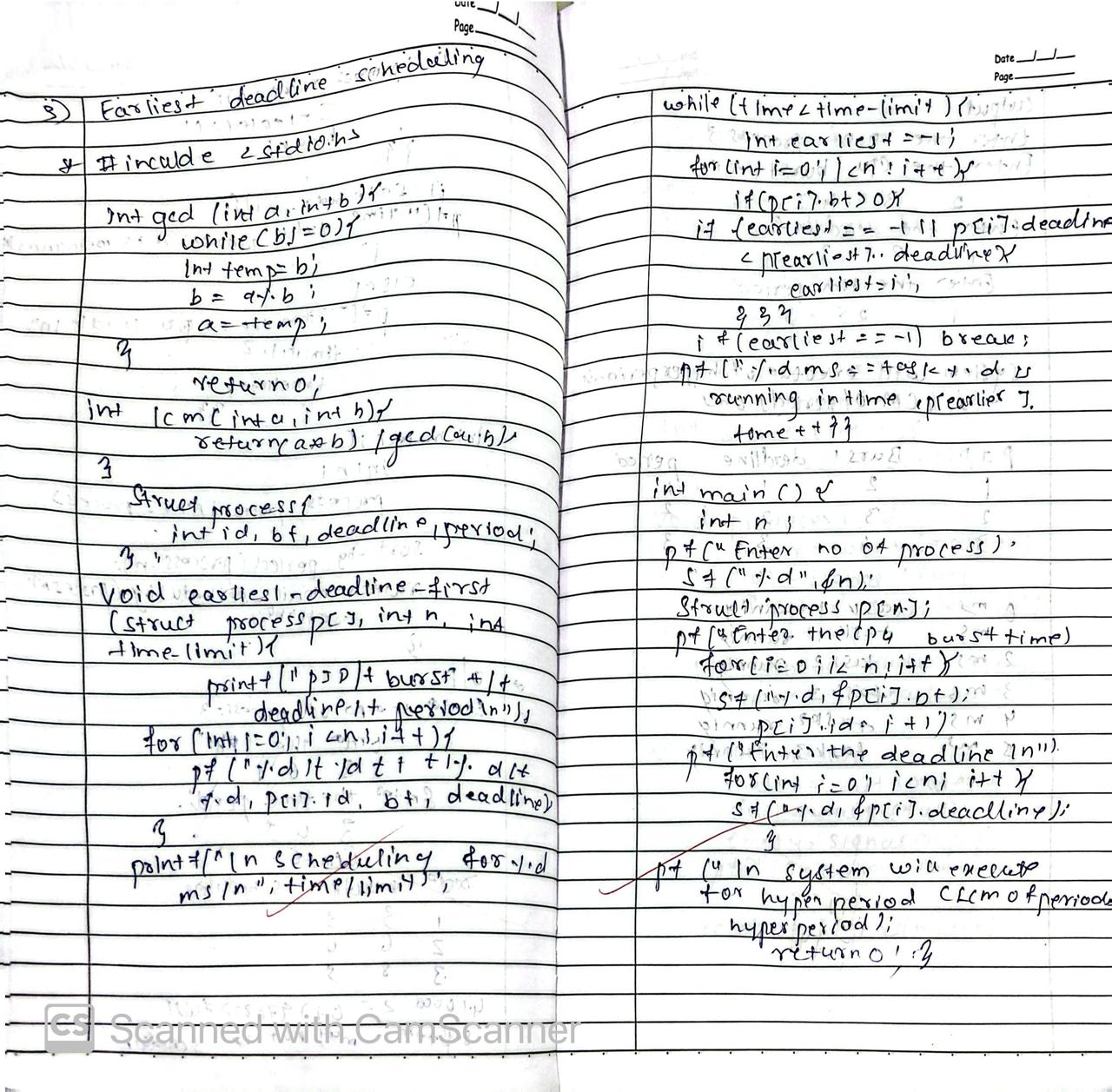
return 0;

}

**OUTPUT:**





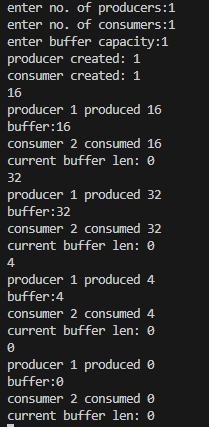
**Program - 5 Question**

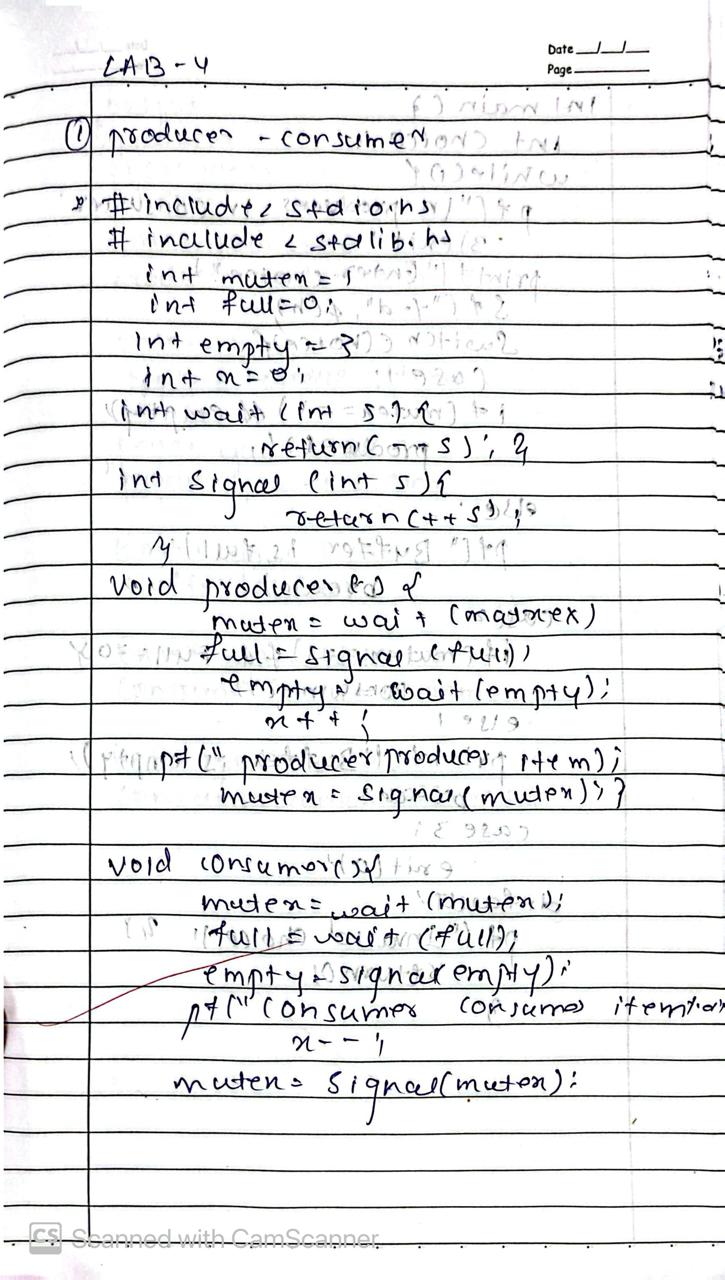
Write a C program to simulate producer-consumer problem using semaphores

**Code:**

#include <stdio.h>  
  
int mutex = 1;  
int full = 0;  
int empty = 3;  
int buffer\_item = 0;  
  
int wait(int s) { return (--s); }  
int signal(int s) { return (++s); }  
  
void producer(int id) {  
 if ((mutex == 1) && (empty != 0)) {  
 int value;  
 scanf("%d", &value);  
 printf("producer %d produced %d\n", id, value);  
 mutex = wait(mutex);  
 full = signal(full);  
 empty = wait(empty);  
 buffer\_item = value;  
 printf("buffer:%d\n", buffer\_item);  
 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);  
 printf("consumer %d consumed %d\n", id, buffer\_item);  
 printf("current buffer len: 0\n");  
 mutex = signal(mutex);  
 } else {  
 printf("buffer is empty!\n");  
 }  
}

**OUTPUT:**



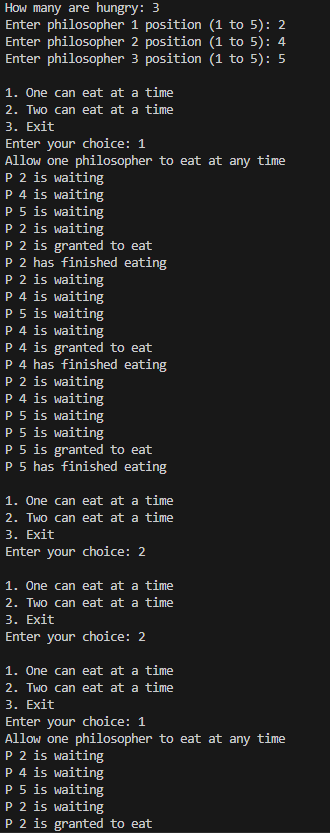
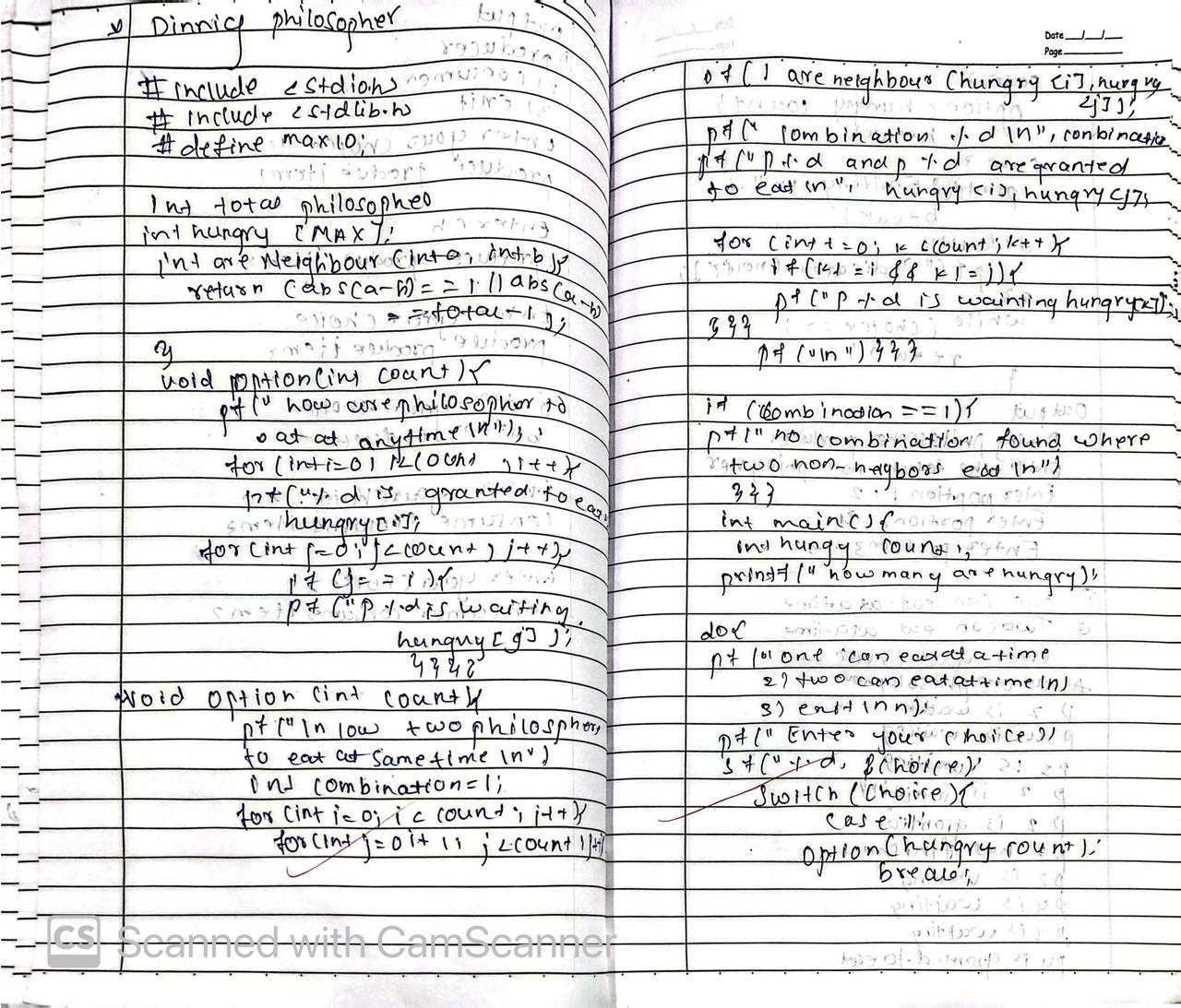
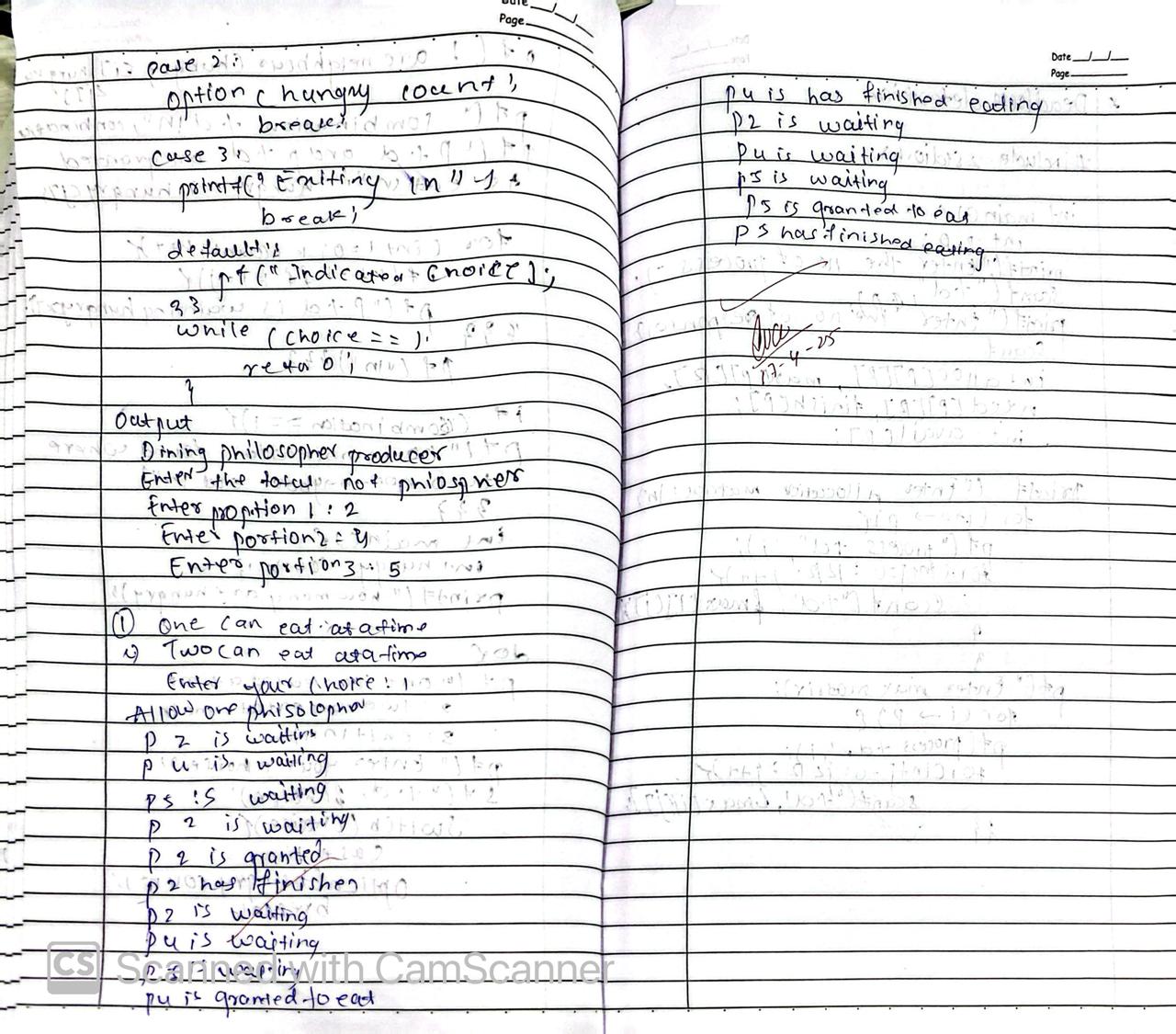
**Program - 6 Question**

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

**Code:**

#include <stdio.h>  
#include <stdlib.h>  
#include <pthread.h>  
#include <semaphore.h>  
#include <unistd.h>  
  
#define MAX 5  
sem\_t mutex;  
sem\_t chopstick[MAX];  
int totalPhilosophers;  
int hungryCount;  
int hungryPhilosophers[MAX];  
  
void \*philosopher(void \*arg) {  
 int id = \*(int \*)arg;  
 sem\_wait(&mutex);  
 sem\_wait(&chopstick[id]);  
 sem\_wait(&chopstick[(id + 1) % totalPhilosophers]);  
 printf("P %d is granted to eat\n", id + 1);  
 sleep(1);  
 printf("P %d has finished eating\n", id + 1);  
 sem\_post(&chopstick[id]);  
 sem\_post(&chopstick[(id + 1) % totalPhilosophers]);  
 sem\_post(&mutex);  
 pthread\_exit(NULL);  
}

**OUTPUT:**

**Program - 7 Question**

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

**Code:**

#include <stdio.h>  
#include <stdbool.h>  
  
int main() {  
 int n, m; // n = number of processes, m = number of resources  
 printf("Enter number of processes and resources:\n");  
 scanf("%d %d", &n, &m);  
  
 int allocation[n][m], max[n][m], available[m];  
  
 printf("Enter allocation matrix:\n");  
 for (int i = 0; i < n; i++)  
 for (int j = 0; j < m; j++)  
 scanf("%d", &allocation[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", &available[i]);  
  
 int need[n][m];  
 for (int i = 0; i < n; i++)  
 for (int j = 0; j < m; j++)  
 need[i][j] = max[i][j] - allocation[i][j];  
  
 bool finish[n];  
 for (int i = 0; i < n; i++)  
 finish[i] = false;  
  
 int count = 0;  
  
 while (count < n) {  
 bool found = false;  
 for (int i = 0; i < n; i++) {  
 if (!finish[i]) {  
 int j;  
 for (j = 0; j < m; j++) {  
 if (need[i][j] > available[j])  
 break;  
 }  
  
 if (j == m) {  
 for (int k = 0; k < m; k++)  
 available[k] += allocation[i][k];  
 finish[i] = true;  
 found = true;  
 printf("Process %d can finish.\n", i);  
 count++;  
 }  
 }  
 }  
  
 if (!found) {  
 break;  
 }  
 }  
  
 bool deadlock = false;  
 for (int i = 0; i < n; i++) {  
 if (!finish[i]) {  
 deadlock = true;  
 break;  
 }  
 }  
  
 if (deadlock)  
 printf("System is in a deadlock state.\n");  
 else  
 printf("System is not in a deadlock state.\n");  
  
 return 0;  
}

**OUTPUT:**

# 

# 

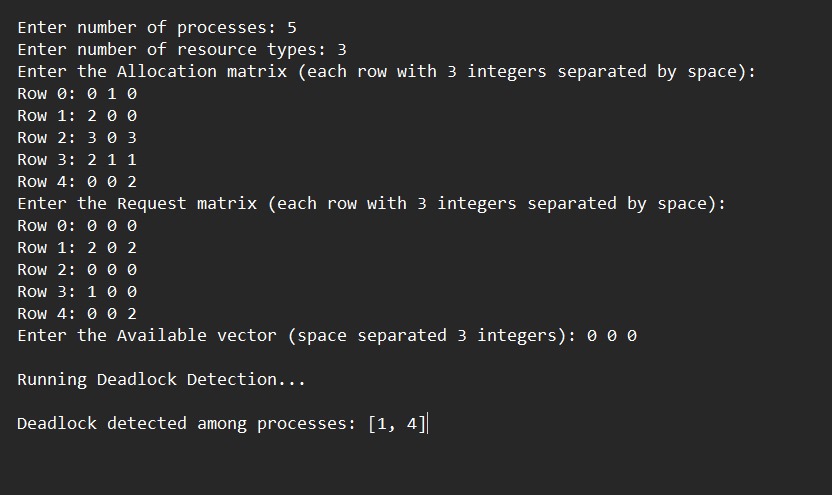
# **Program - 8 Question**

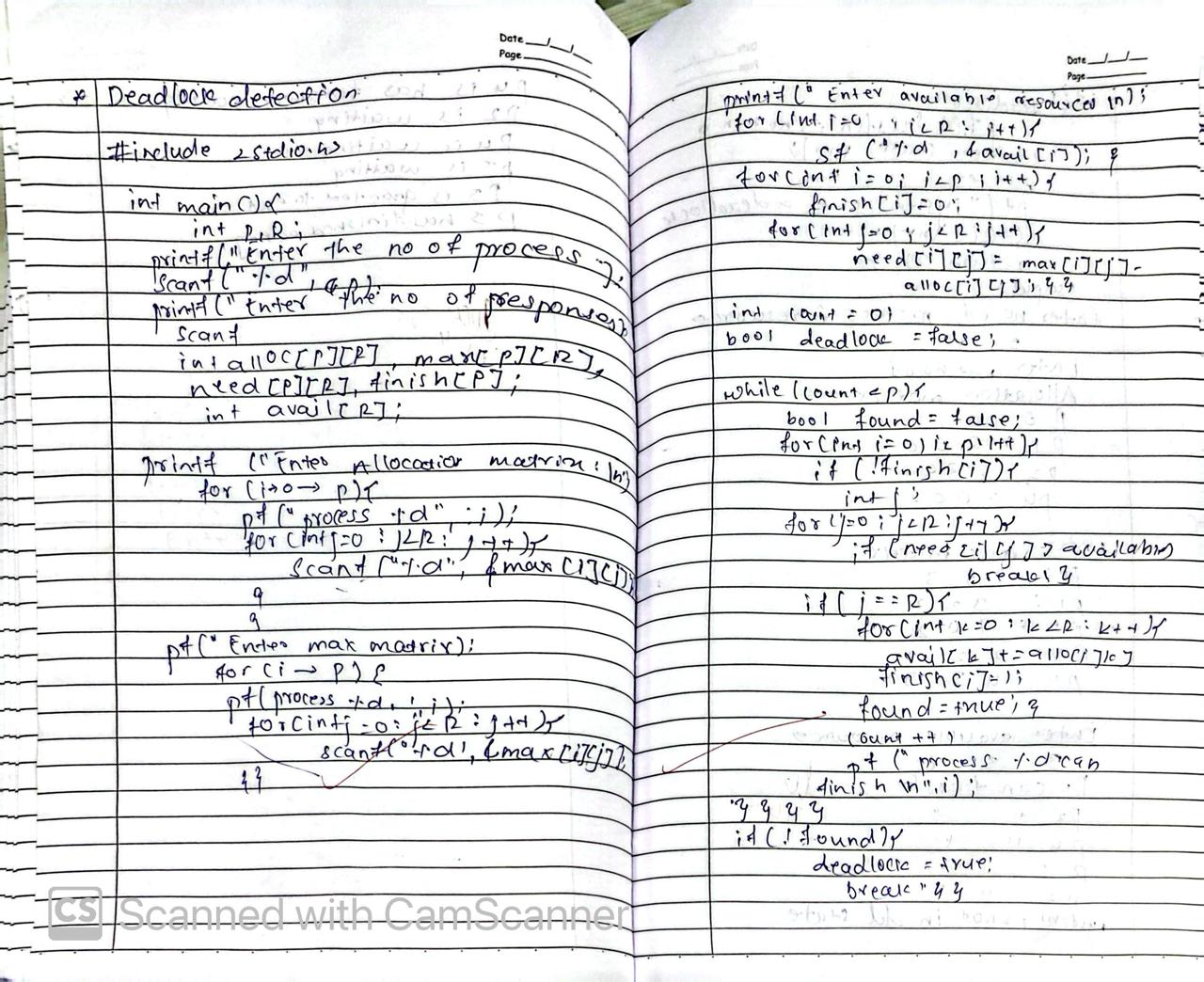
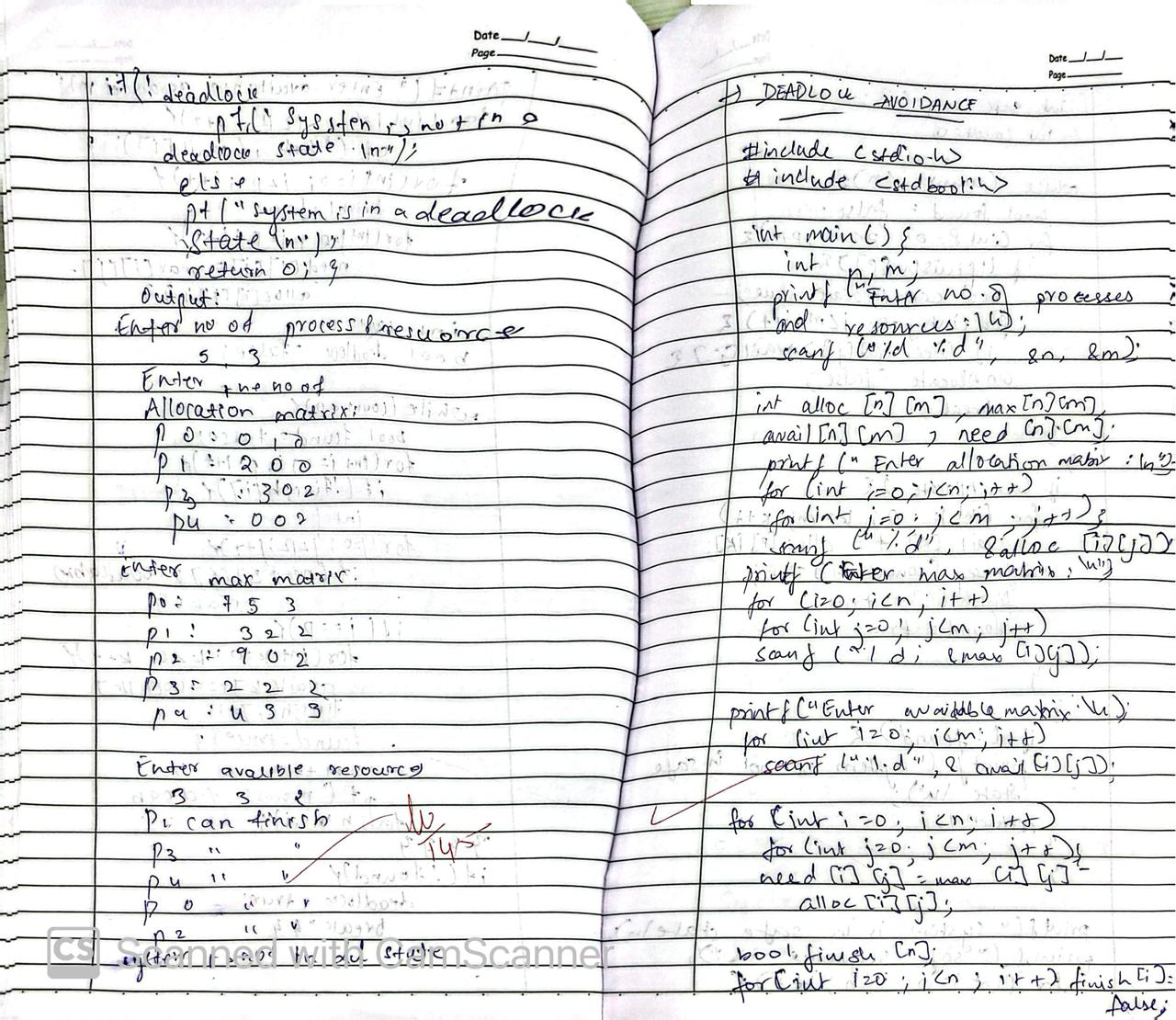
Write a C program to simulate deadlock detection

## **Code:**

#include <stdio.h>  
  
int main() {  
 int n, m; // n = number of processes, m = number of resources  
 printf("Enter number of processes and resources: ");  
 scanf("%d %d", &n, &m);  
  
 int allocation[n][m], request[n][m], available[m];  
 int finish[n];  
  
 printf("Enter allocation matrix:  
");  
 for (int i = 0; i < n; i++)  
 for (int j = 0; j < m; j++)  
 scanf("%d", &allocation[i][j]);  
  
 printf("Enter request matrix:  
");  
 for (int i = 0; i < n; i++)  
 for (int j = 0; j < m; j++)  
 scanf("%d", &request[i][j]);  
  
 printf("Enter available resources:  
");  
 for (int i = 0; i < m; i++)  
 scanf("%d", &available[i]);  
  
 for (int i = 0; i < n; i++)  
 finish[i] = 0;  
  
 int changed;  
 do {  
 changed = 0;  
 for (int i = 0; i < n; i++) {  
 if (!finish[i]) {  
 int j;  
 for (j = 0; j < m; j++) {  
 if (request[i][j] > available[j])  
 break;  
 }  
 if (j == m) {  
 for (int k = 0; k < m; k++)  
 available[k] += allocation[i][k];  
 finish[i] = 1;  
 changed = 1;  
 }  
 }  
 }  
 } while (changed);  
  
 int deadlock = 0;  
 for (int i = 0; i < n; i++) {  
 if (!finish[i]) {  
 deadlock = 1;  
 printf("Process %d is in deadlock.  
", i);  
 }  
 }  
  
 if (!deadlock)  
 printf("No deadlock detected.  
");  
  
 return 0;  
}

**OUTPUT:**



# **Program - 9 Question**

Write a C program to simulate the following contiguous memory allocation techniques a) Worst-fit b)Best-fit c)First-fit

## **Code:**

#include <stdio.h>

#define MAX 100

void firstFit(int blockSize[], int blocks, int processSize[], int processes) {

int allocation[MAX];

for (int i = 0; i < processes; i++) allocation[i] = -1;

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

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

if (blockSize[j] >= processSize[i]) {

allocation[i] = j;

blockSize[j] -= processSize[i];

break;

}

}

}

printf("\nFirst Fit Allocation:\n");

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

printf("Process %d (%dKB) -> ", i + 1, processSize[i]);

if (allocation[i] != -1)

printf("Block %d\n", allocation[i] + 1);

else

printf("Not Allocated\n");

}

}

void bestFit(int blockSize[], int blocks, int processSize[], int processes) {

int allocation[MAX];

for (int i = 0; i < processes; i++) allocation[i] = -1;

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

int bestIndex = -1;

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

if (blockSize[j] >= processSize[i]) {

if (bestIndex == -1 || blockSize[j] < blockSize[bestIndex])

bestIndex = j;

}

}

if (bestIndex != -1) {

allocation[i] = bestIndex;

blockSize[bestIndex] -= processSize[i];

}

}

printf("\nBest Fit Allocation:\n");

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

printf("Process %d (%dKB) -> ", i + 1, processSize[i]);

if (allocation[i] != -1)

printf("Block %d\n", allocation[i] + 1);

else

printf("Not Allocated\n");

}

}

void worstFit(int blockSize[], int blocks, int processSize[], int processes) {

int allocation[MAX];

for (int i = 0; i < processes; i++) allocation[i] = -1;

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

int worstIndex = -1;

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

if (blockSize[j] >= processSize[i]) {

if (worstIndex == -1 || blockSize[j] > blockSize[worstIndex])

worstIndex = j;

}

}

if (worstIndex != -1) {

allocation[i] = worstIndex;

blockSize[worstIndex] -= processSize[i];

}

}

printf("\nWorst Fit Allocation:\n");

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

printf("Process %d (%dKB) -> ", i + 1, processSize[i]);

if (allocation[i] != -1)

printf("Block %d\n", allocation[i] + 1);

else

printf("Not Allocated\n");

}

}

void copyArray(int dest[], int src[], int size) {

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

dest[i] = src[i];

}

int main() {

int blockSize[MAX], processSize[MAX];

int m, n;

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

scanf("%d", &m);

printf("Enter size of each block:\n");

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

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

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

}

printf("\nEnter number of processes: ");

scanf("%d", &n);

printf("Enter size of each process:\n");

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

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

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

}

int b1[MAX], b2[MAX], b3[MAX];

copyArray(b1, blockSize, m);

copyArray(b2, blockSize, m);

copyArray(b3, blockSize, m);

firstFit(b1, m, processSize, n);

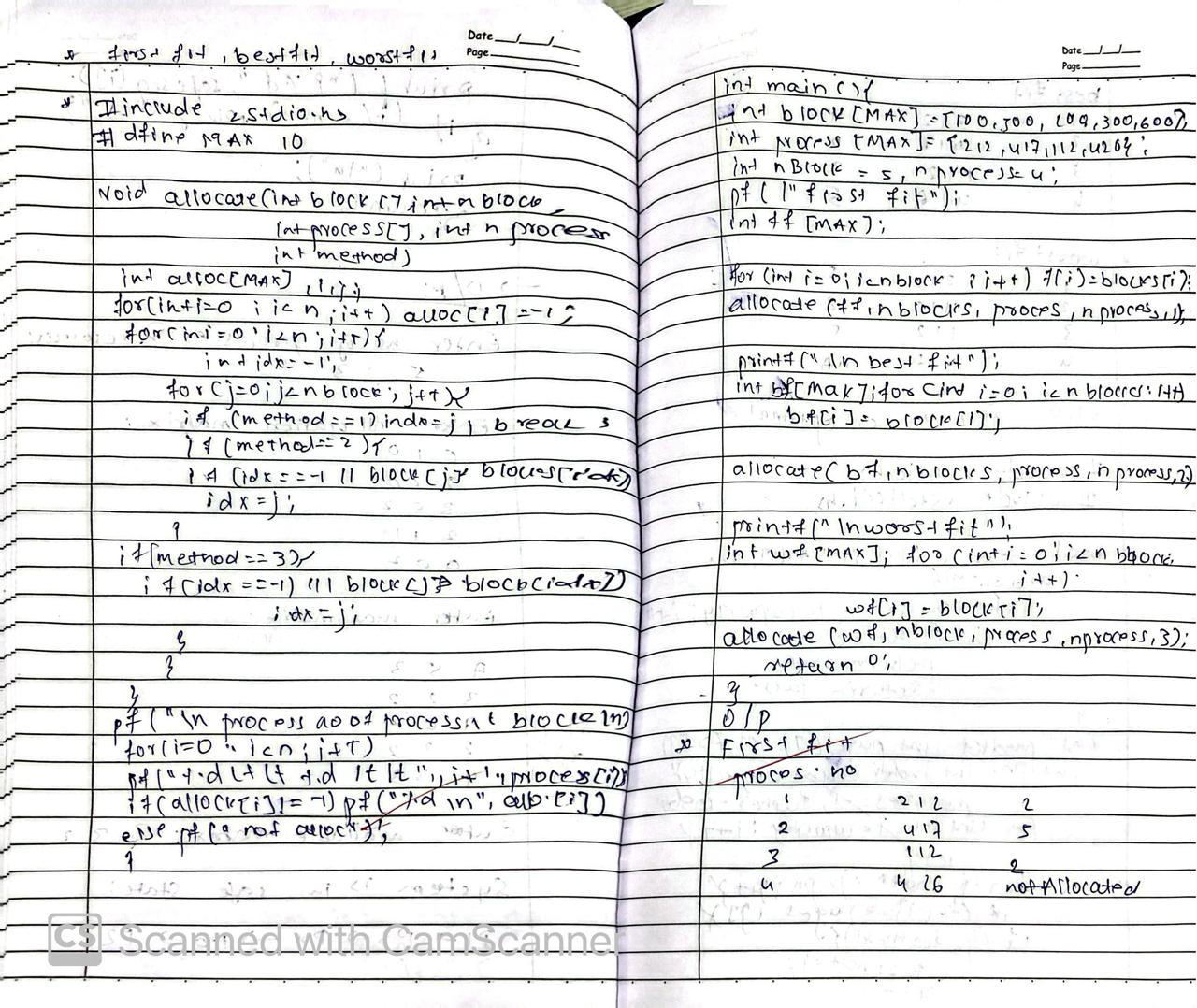
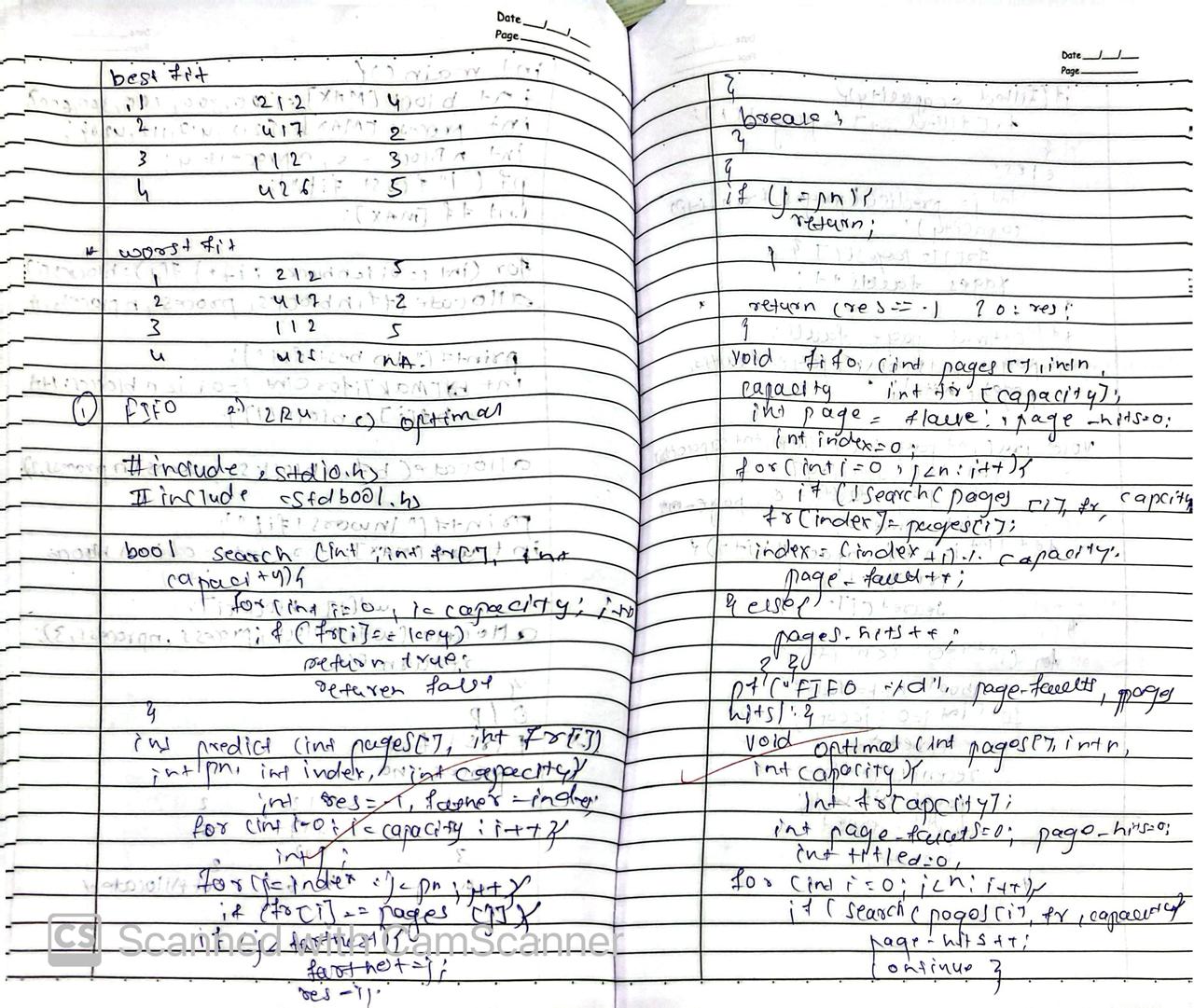
bestFit(b2, m, processSize, n);

worstFit(b3, m, processSize, n);

return 0;

}

**OUTPUT:**

**Program - 10 Question**

Write a C program to simulate page replacement algorithms:  
a) FIFO  
b) LRU  
c) Optimal

**Code:**

#include <stdio.h>  
#include <stdlib.h>  
  
#define MAX 100  
  
void fifo(int pages[], int n, int capacity) {  
 int frame[capacity], index = 0, pageFaults = 0;  
 for (int i = 0; i < capacity; i++) frame[i] = -1;  
 for (int i = 0; i < n; i++) {  
 int found = 0;  
 for (int j = 0; j < capacity; j++) {  
 if (frame[j] == pages[i]) {  
 found = 1;  
 break;  
 }  
 }  
 if (!found) {  
 frame[index] = pages[i];  
 index = (index + 1) % capacity;  
 pageFaults++;  
 }  
 }  
 printf("\nFIFO Page Replacement:\nPage Faults: %d\n", pageFaults);  
}  
  
void lru(int pages[], int n, int capacity) {  
 int frame[capacity], lastUsed[MAX], pageFaults = 0;  
 for (int i = 0; i < capacity; i++) frame[i] = -1;  
 for (int i = 0; i < n; i++) {  
 int found = 0, leastRecent = -1, leastTime = MAX;  
 for (int j = 0; j < capacity; j++) {  
 if (frame[j] == pages[i]) {  
 found = 1;  
 lastUsed[j] = i;  
 break;  
 }  
 if (frame[j] != -1 && lastUsed[j] < leastTime) {  
 leastTime = lastUsed[j];  
 leastRecent = j;  
 }  
 }  
 if (!found) {  
 if (leastRecent != -1) {  
 frame[leastRecent] = pages[i];  
 lastUsed[leastRecent] = i;  
 }  
 pageFaults++;  
 }  
 }  
 printf("\nLRU Page Replacement:\nPage Faults: %d\n", pageFaults);  
}  
  
void optimal(int pages[], int n, int capacity) {  
 int frame[capacity], pageFaults = 0;  
 for (int i = 0; i < capacity; i++) frame[i] = -1;  
 for (int i = 0; i < n; i++) {  
 int found = 0, farthest = -1, replaceIndex = -1;  
 for (int j = 0; j < capacity; j++) {  
 if (frame[j] == pages[i]) {  
 found = 1;  
 break;  
 }  
 }  
 if (!found) {  
 for (int j = 0; j < capacity; j++) {  
 int k;  
 for (k = i + 1; k < n; k++) {  
 if (frame[j] == pages[k]) {  
 if (k > farthest) {  
 farthest = k;  
 replaceIndex = j;  
 }  
 break;  
 }  
 }  
 if (k == n) {  
 replaceIndex = j;  
 break;  
 }  
 }  
 frame[replaceIndex] = pages[i];  
 pageFaults++;  
 }  
 }  
 printf("\nOptimal Page Replacement:\nPage Faults: %d\n", pageFaults);  
}  
  
int main() {  
 int pages[MAX], n, capacity;  
 printf("Enter number of pages: ");  
 scanf("%d", &n);  
 printf("Enter pages:\n");  
 for (int i = 0; i < n; i++) {  
 scanf("%d", &pages[i]);  
 }  
 printf("Enter capacity of page frames: ");  
 scanf("%d", &capacity);  
   
 fifo(pages, n, capacity);  
 lru(pages, n, capacity);  
 optimal(pages, n, capacity);  
   
 return 0;  
}

**OUTPUT:**

