**End Term Project On**

Design of Operating Systems (CSE 4049)

**Submitted by**

**Name : Saswat Mohanty**

**Reg. No. 1941012407**

**Semester : 5th**

**Branch : CSE**

**Section : D**

**Session : 2021-2022 Admission Batch : 2019**



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING FACULTY OF ENGINEERING & TECHNOLOGY (ITER) SIKSHA ‘O’ ANUSANDHAN DEEMED TO BE UNIVERSITY BHUBANESWAR, ODISHA – 751030**

# Objective of this Assignment:

To design a CPU scheduler for simulating a few CPU scheduling policies

# Project Description:

Program to provides an interface to the user to implement the following scheduling policies as per the choice provided:

1. First Come First Served (FCFS)
2. Shortest Job First (SJF)
3. Shortest Remaining Time First (SRTF)
4. Round Robin (RR)

**Programs FCFS** FCFS.h

void FCFS();

FCFS.c

#include <stdio.h> #include <stdlib.h> #include "FCFS.h" typedef struct node

{

int no;

float at, bt, pc, tat, wt, rt, rd; struct node \*next;

} NODE;

void create\_insert(NODE \*\*p, int no, float at, float bt, float \*fr)

{

NODE \*q, \*r = \*p;

q = (NODE \*)malloc(sizeof(NODE)); q->no = no;

q->at = at; q->bt = bt;

q->rt = \*fr - at; q->pc = \*fr + bt;

q->tat = q->pc - at;

q->wt = q->tat - bt;

q->rd = q->tat / bt;

\*fr = \*fr + bt; q->next = NULL; if (\*p == NULL)

\*p = q; else

{

while (r->next != NULL) r = r->next;

r->next = q;

}

}

void gantt\_chart(NODE \*p, int process)

{

int i;

NODE \*r = p; printf("\n\nGannt Chart:\n"); for (i = 1; i <= process; i++)

printf(" ");

printf("\n");

for (i = 1; i <= process; i++)

{

printf("|\tP%d\t", p->no); p = p->next;

}

printf("|\n");

for (i = 1; i <= process; i++) printf(" ");

printf("\n");

printf("%.1f \t", r->at); for (i = 1; i <= process; i++)

{

printf("%.1f \t", r->pc); r = r->next;

}

}

void display(NODE \*p, int process)

{

float ttat, twt, trd, trt, tbt; ttat = twt = trd = trt = tbt = 0;

printf("\n\n\nProcess Details:\n"); printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"); printf("Pro\tArTi\tBuTi\tTaTi\tWtTi\tRTi\n"); printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"); while (p != NULL)

{

printf("%d\t", p->no);

printf("%.2f\t", p->at);

printf("%.2f\t", p->bt);

printf("%.2f\t", p->tat);

printf("%.2f\t", p->wt);

printf("%.2f\n", p->rt); ttat += p->tat;

twt += p->wt; trd += p->rd; trt += p->rt; tbt += p->bt;

p = p->next;

}

printf("\n\n\nOverall Details:\n");

printf("Average Turn Around Time: %.2f\n", ttat / process); printf("Average Waiting Time: %.2f\n", twt / process); printf("Average Response Time: %.2f\n", trt / process);

}

void FCFS()

{

NODE \*head = NULL; int process, i;

float arrival\_time, burst\_time, first\_response; printf("First- Come, First-Served (FCFS) Scheduling\n"); printf("Enter Number of Processes\n");

scanf("%d", &process);

for (i = 1; i <= process; i++)

{

printf("\nEnter the Details for Process %d: \n", i); printf("Arrival Time: ");

scanf("%f", &arrival\_time); printf("Burst Time: "); scanf("%f", &burst\_time); if (i == 1)

first\_response = arrival\_time;

create\_insert(&head, i, arrival\_time, burst\_time, &first\_response);

}

printf("\n<

>\n"); gantt\_chart(head, process); display(head, process);

printf("\n<

>\n");

}

START

END

# SJF

SJF.h

void SFJ();

SJF.c

#include <stdio.h> #include "SJF.h" struct time

{

int p, art, but, wtt, tat, st;

};

int process(struct time a[], int pro, int t)

{

int i, minpro, mintime = 999; for (i = 0; i < pro; i++)

{

if (a[i].art <= t && a[i].st == 0)

{

if (mintime > a[i].but)

{

mintime = a[i].but; minpro = i;

}

}

}

a[minpro].st = 1; return minpro;

}

void ganttchart(struct time a[], int gc[], int pro)

{

int i, x = 0; printf("Gantt Chart\n"); printf("0");

for (i = 0; i < pro; i++)

{

x = x + a[gc[i]].but;

printf(" -> [P%d] <- %d", a[gc[i]].p, x);

}

printf("\n"); return;

}

void SFJ()

{

int i, pro, curpro, t = 0, gc[100]; struct time a[100];

float avgwt = 0, avgtt = 0;

printf("Shortest-Job-First (SJF) Scheduling\n"); printf("Enter Number of Processes\n"); scanf("%d", &pro);

for (i = 0; i < pro; i++)

{

printf("Enter Arrival Time & Burst Time for Process P%d\n", i); a[i].p = i;

scanf("%d%d", &a[i].art, &a[i].but); a[i].st = 0;

}

for (i = 0; i < pro; i++)

{

curpro = process(a, pro, t); a[curpro].wtt = t - a[curpro].art;

a[curpro].tat = a[curpro].art + a[curpro].but; t = t + a[curpro].but;

avgwt = avgwt + a[curpro].wtt; avgtt = avgtt + a[curpro].tat; gc[i] = curpro;

}

printf("\n<

>\n"); printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"); ganttchart(a, gc, pro); printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"); printf("\n\n\nProcess Details:\n"); printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"); printf("Pro\tArTi\tBuTi\tTaTi\tWtTi\n"); printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

START

for (i = 0; i < pro; i++)

{

printf("%d\t%d\t%d\t%d\t%d\n", a[i].p, a[i].art, a[i].but, a[i].tat, a[i].wtt);

}

avgwt = avgwt / pro; avgtt = avgtt / pro;

printf("\n\n\nOverall Details:\n"); printf("Average Waiting Time : %.2f\n", avgwt); printf("Average Turnaround Time : %.2f\n", avgtt);

printf("\n<

>\n");

}

END

# SRTF

SRTF.h

void SRTF();

SRTF.c

#include <stdio.h> #include <stdbool.h> #include <limits.h> #include "SRTF.h" struct process\_struct

{

int pid; int at; int bt;

int ct, wt, tat, rt, start\_time;

} ps[100];

int findmax(int a, int b)

{

return a > b ? a : b;

}

int findmin(int a, int b)

{

return a < b ? a : b;

}

void SRTF()

{

int n;

float bt\_remaining[100];

bool is\_completed[100] = {false}, is\_first\_process = true; int current\_time = 0;

int completed = 0;

;

float sum\_tat = 0, sum\_wt = 0, sum\_rt = 0, total\_idle\_time = 0, length\_cycle, prev = 0;

float cpu\_utilization;

int max\_completion\_time, min\_arrival\_time;

printf("Shortest-Remaining-Time-First (SRTF) Scheduling\n"); printf("Enter Number of Processes\n");

scanf("%d", &n);

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

{

printf("\nEnter Process %d Arrival Time: ", i); scanf("%d", &ps[i].at);

ps[i].pid = i;

}

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

{

printf("\nEnter Process %d Burst Time: ", i); scanf("%d", &ps[i].bt);

bt\_remaining[i] = ps[i].bt;

}

while (completed != n)

{

int min\_index = -1; int minimum = INT\_MAX;

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

{

if (ps[i].at <= current\_time && is\_completed[i] == false)

{

if (bt\_remaining[i] < minimum)

{

minimum = bt\_remaining[i]; min\_index = i;

}

if (bt\_remaining[i] == minimum)

{

if (ps[i].at < ps[min\_index].at)

{

minimum = bt\_remaining[i]; min\_index = i;

}

}

}

}

if (min\_index == -1)

{

current\_time++;

}

else

{

if (bt\_remaining[min\_index] == ps[min\_index].bt)

{

ps[min\_index].start\_time = current\_time; total\_idle\_time += (is\_first\_process == true) ? 0 :

(ps[min\_index].start\_time - prev);

is\_first\_process = false;

}

bt\_remaining[min\_index] -= 1; current\_time++;

prev = current\_time;

if (bt\_remaining[min\_index] == 0)

{

ps[min\_index].ct = current\_time;

ps[min\_index].tat = ps[min\_index].ct - ps[min\_index].at; ps[min\_index].wt = ps[min\_index].tat - ps[min\_index].bt; ps[min\_index].rt = ps[min\_index].start\_time - ps[min\_index].at;

sum\_tat += ps[min\_index].tat; sum\_wt += ps[min\_index].wt; sum\_rt += ps[min\_index].rt; completed++; is\_completed[min\_index] = true;

}

}

}

max\_completion\_time = INT\_MIN; min\_arrival\_time = INT\_MAX;

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

{

max\_completion\_time = findmax(max\_completion\_time, ps[i].ct); min\_arrival\_time = findmin(min\_arrival\_time, ps[i].at);

}

length\_cycle = max\_completion\_time - min\_arrival\_time; printf("\n<

>\n"); printf("\n\n\nProcess Details:\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"); printf("Pro\tArTi\tBuTi\tTaTi\tWtTi\tRTi\n"); printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"); for (int i = 0; i < n; i++)

START

printf("%d\t%d\t%d\t%d\t%d\t%d\n", ps[i].pid, ps[i].at, ps[i].bt, ps[i].tat, ps[i].wt, ps[i].rt);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"); printf("\n");

printf("\n\n\nOverall Details:\n");

printf("\nAverage Turn Around time= %f ", (float)sum\_tat / n); printf("\nAverage Waiting Time= %f ", (float)sum\_wt / n); printf("\nAverage Response Time= %f ", (float)sum\_rt / n);

printf("\n<

>\n");

}

END

# RR

RR.h

void RR();

RR.c

#include <stdio.h> #include "RR.h" struct times

{

int p, art, but, wtt, tat, rnt;

};

void sortart(struct times a[], int pro)

{

int i, j;

struct times temp;

for (i = 0; i < pro; i++)

{

for (j = i + 1; j < pro; j++)

{

if (a[i].art > a[j].art)

{

temp = a[i]; a[i] = a[j]; a[j] = temp;

}

}

}

return;

}

void RR()

{

int i, j, pro, time, remain, flag = 0, ts; struct times a[100];

float avgwt = 0, avgtt = 0; printf("Round Robin (RR) Scheduling\n"); printf("Enter Number of Processes\n"); scanf("%d", &pro);

remain = pro;

for (i = 0; i < pro; i++)

{

printf("Enter arrival time and Burst time for Process P%d : ", i); scanf("%d%d", &a[i].art, &a[i].but);

a[i].p = i; a[i].rnt = a[i].but;

}

sortart(a, pro);

printf("Enter Time Quantum Number : "); scanf("%d", &ts);

printf("\n<

>\n"); printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"); printf("Gantt Chart\n");

printf("0");

for (time = 0, i = 0; remain != 0;)

{

START

if (a[i].rnt <= ts && a[i].rnt > 0)

{

time = time + a[i].rnt;

printf(" -> [P%d] <- %d", a[i].p, time); a[i].rnt = 0;

flag = 1;

}

else if (a[i].rnt > 0)

{

a[i].rnt = a[i].rnt - ts; time = time + ts;

printf(" -> [P%d] <- %d", a[i].p, time);

}

if (a[i].rnt == 0 && flag == 1)

{

remain--;

a[i].tat = time - a[i].art;

a[i].wtt = time - a[i].art - a[i].but; avgwt = avgwt + time - a[i].art - a[i].but; avgtt = avgtt + time - a[i].art;

flag = 0;

}

if (i == pro - 1) i = 0;

else if (a[i + 1].art <= time) i++;

else

i = 0;

}

printf("\n\n"); printf("\n\n\nProcess Details:\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"); printf("Pro\tArTi\tBuTi\tTaTi\tWtTi\n"); printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"); for (i = 0; i < pro; i++)

{

printf("P%d\t%d\t%d\t%d\t%d\n", a[i].p, a[i].art, a[i].but, a[i].tat, a[i].wtt);

}

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"); avgwt = avgwt / pro;

avgtt = avgtt / pro; printf("\n\n\nOverall Details:\n");

printf("Average Waiting Time : %.2f\n", avgwt); printf("Average Turnaround Time : %.2f\n", avgtt);

printf("\n<

>\n");

}

END

main.c

#include <stdio.h> #include <stdlib.h> #include "FCFS.h" #include "SJF.h" #include "SRTF.h" #include "RR.h" int main()

{

int choice;

printf("CPU scheduler for simulating a few CPU scheduling policies.\n"); while (1)

{

printf("\nChosse Option\n1->FCFS\n2->SJF\n3->SRTF\n4->RR\n5->Exit\n"); scanf("%d", &choice);

switch (choice)

{

case 1:

FCFS();

break; case 2:

SFJ();

break; case 3:

SRTF();

break; case 4:

RR();

break; case 5:

exit(0); default:

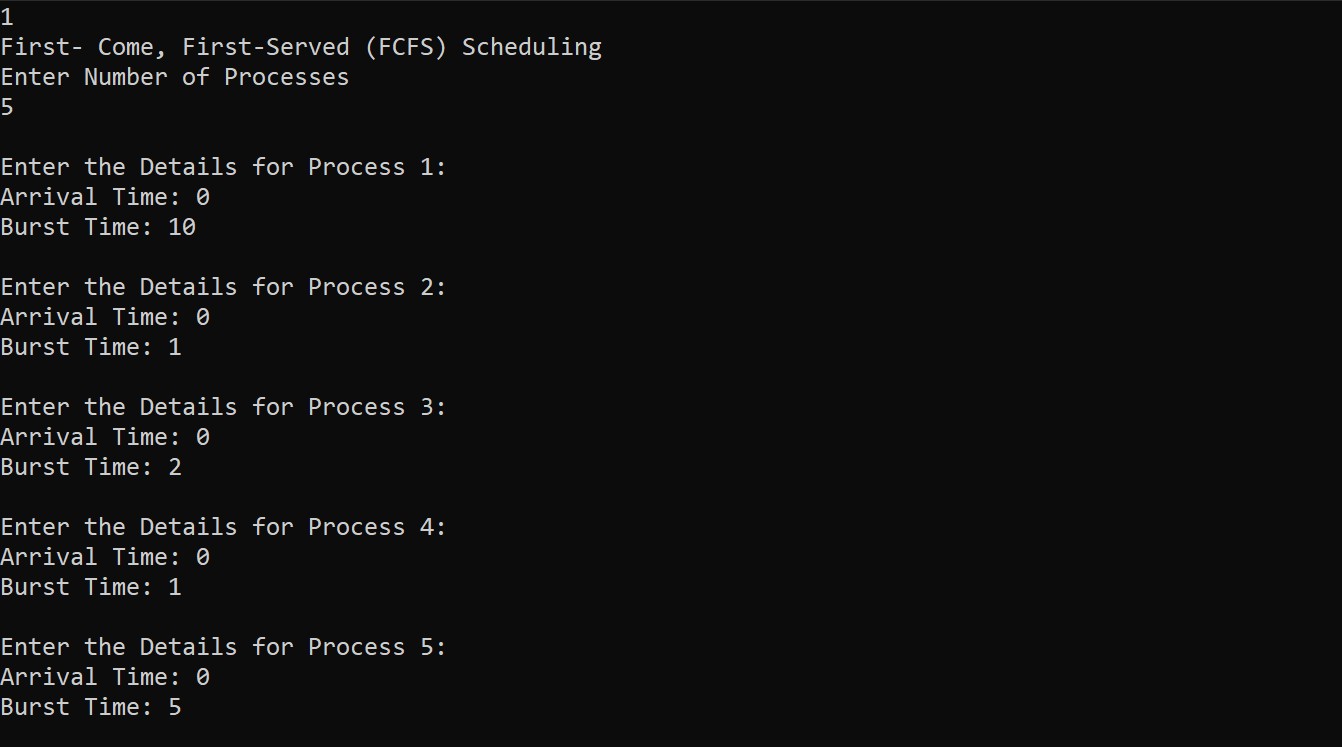
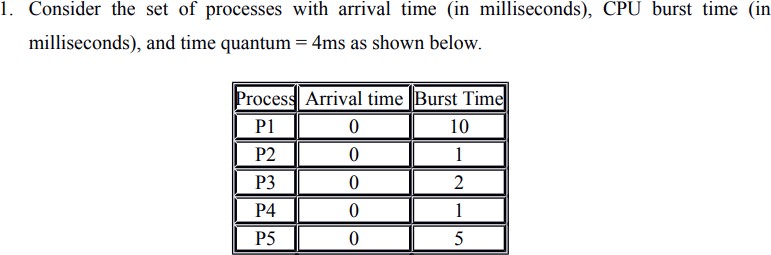
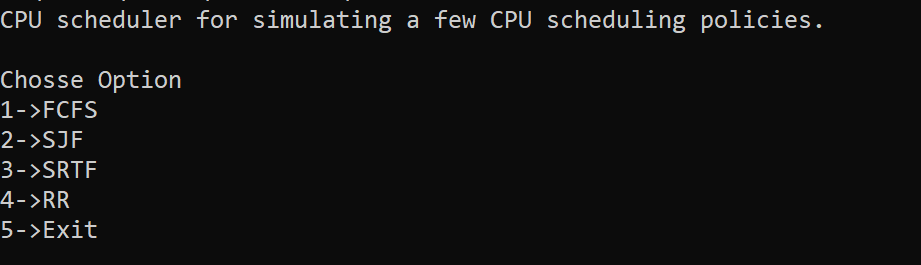
printf("Wrong Input\n"); break;

}

}

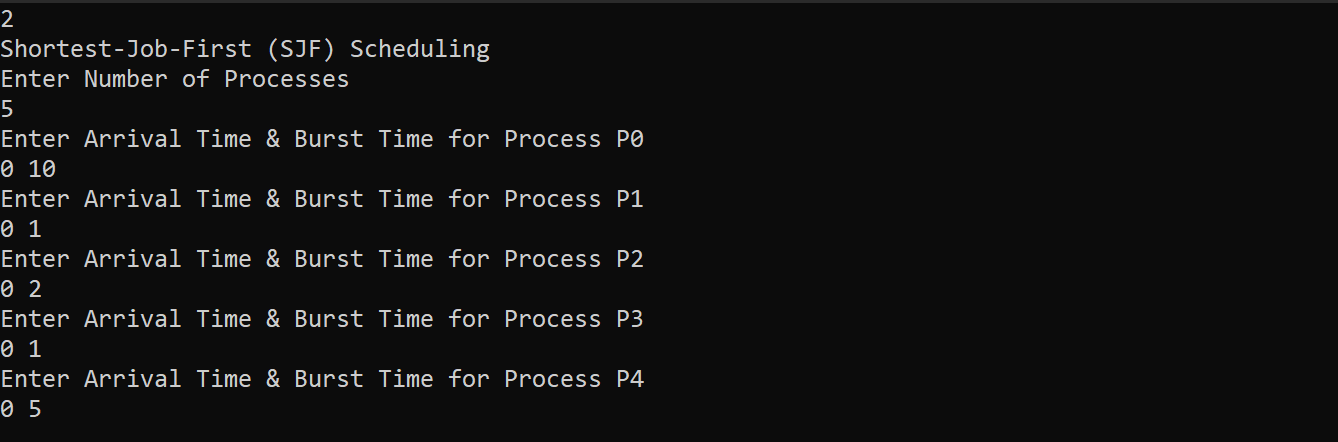
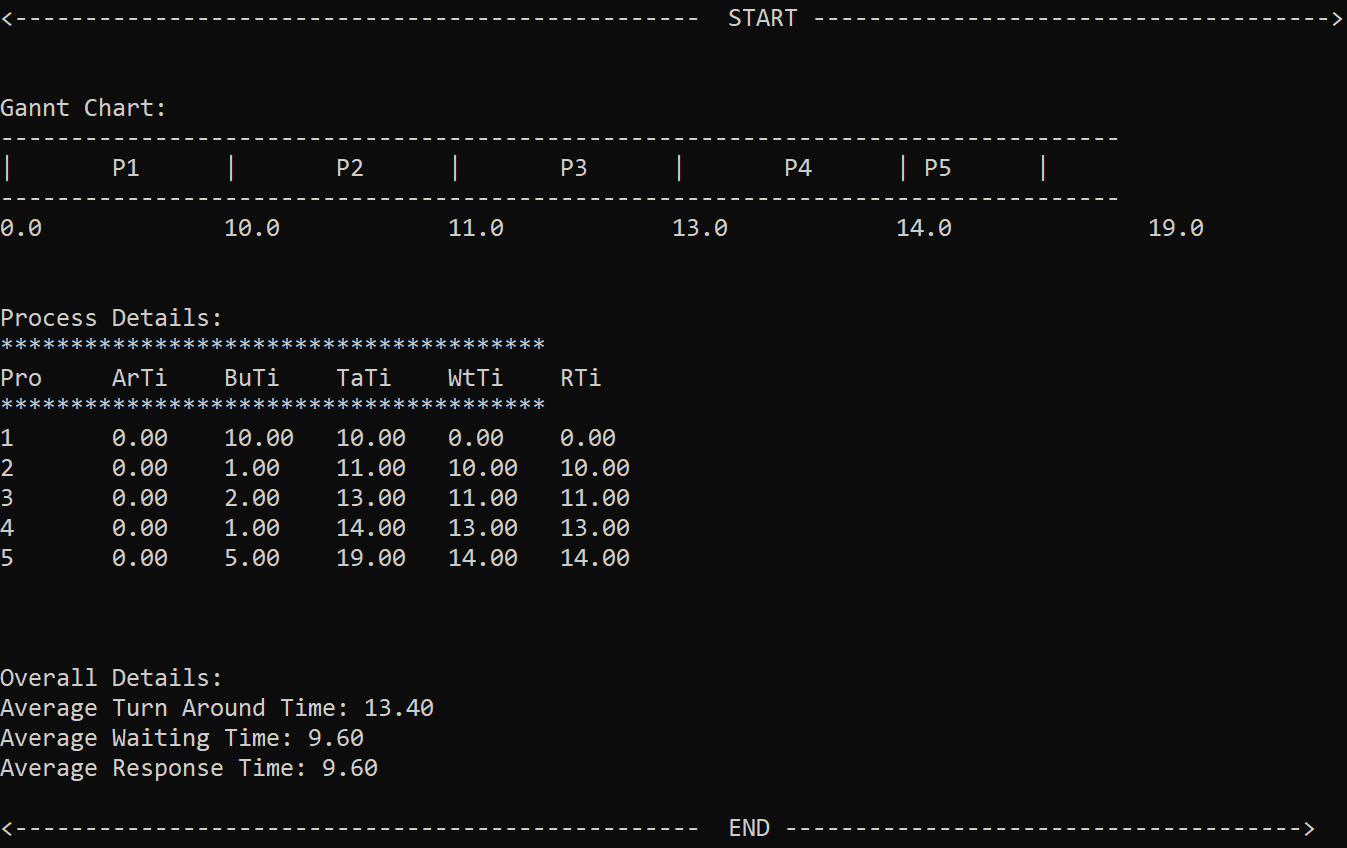
}

# Output Console:

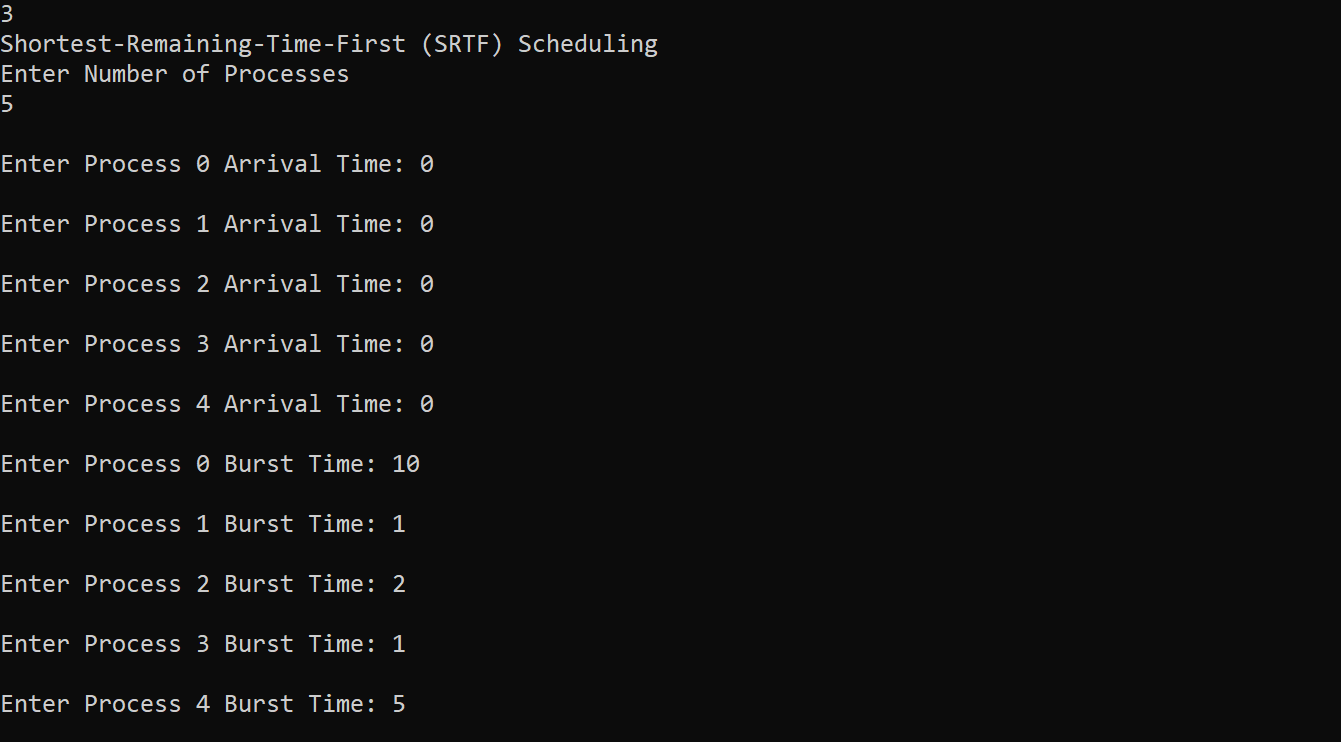
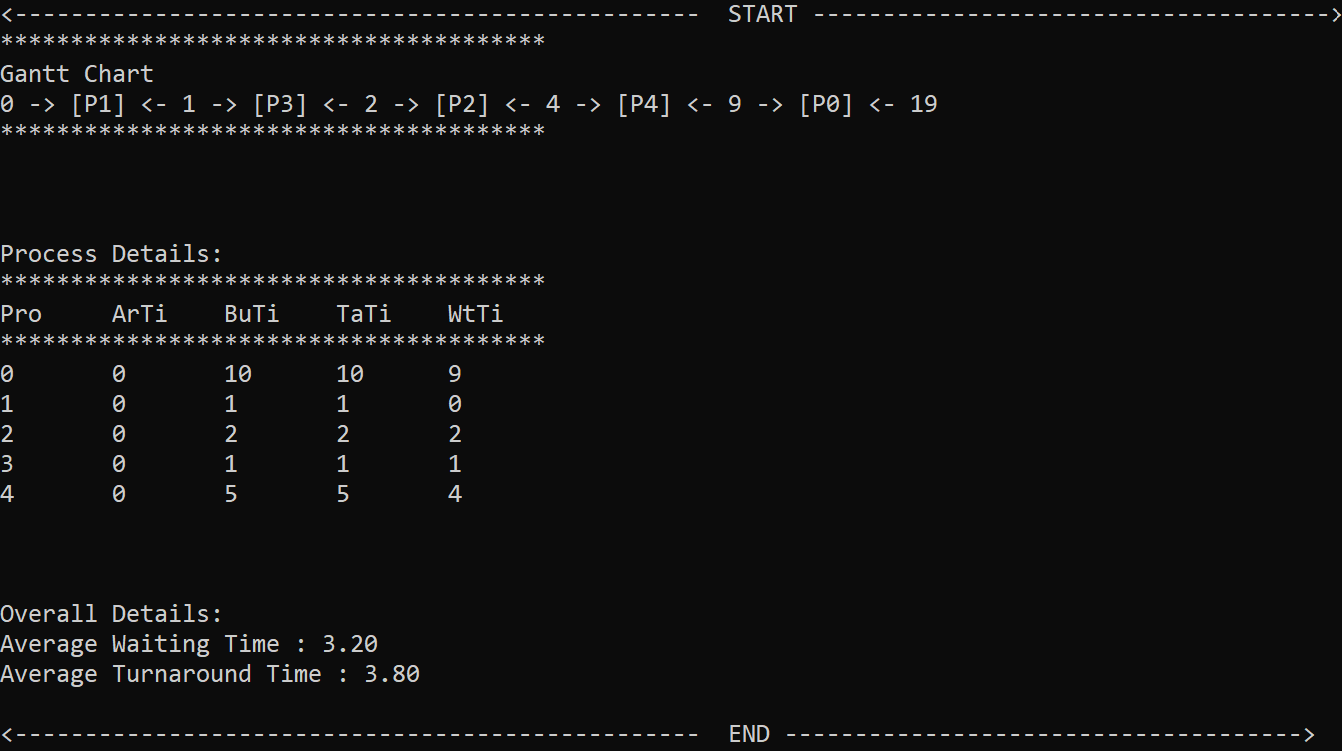


**Test Cases:**

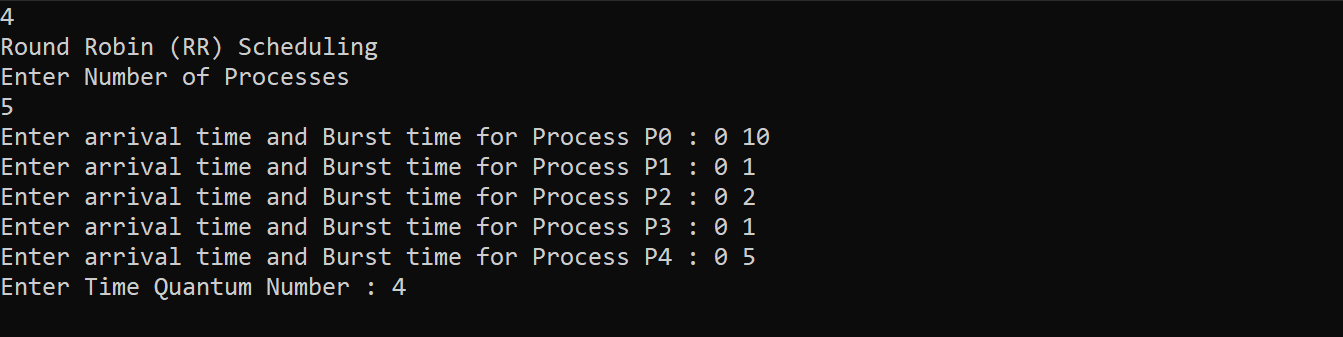
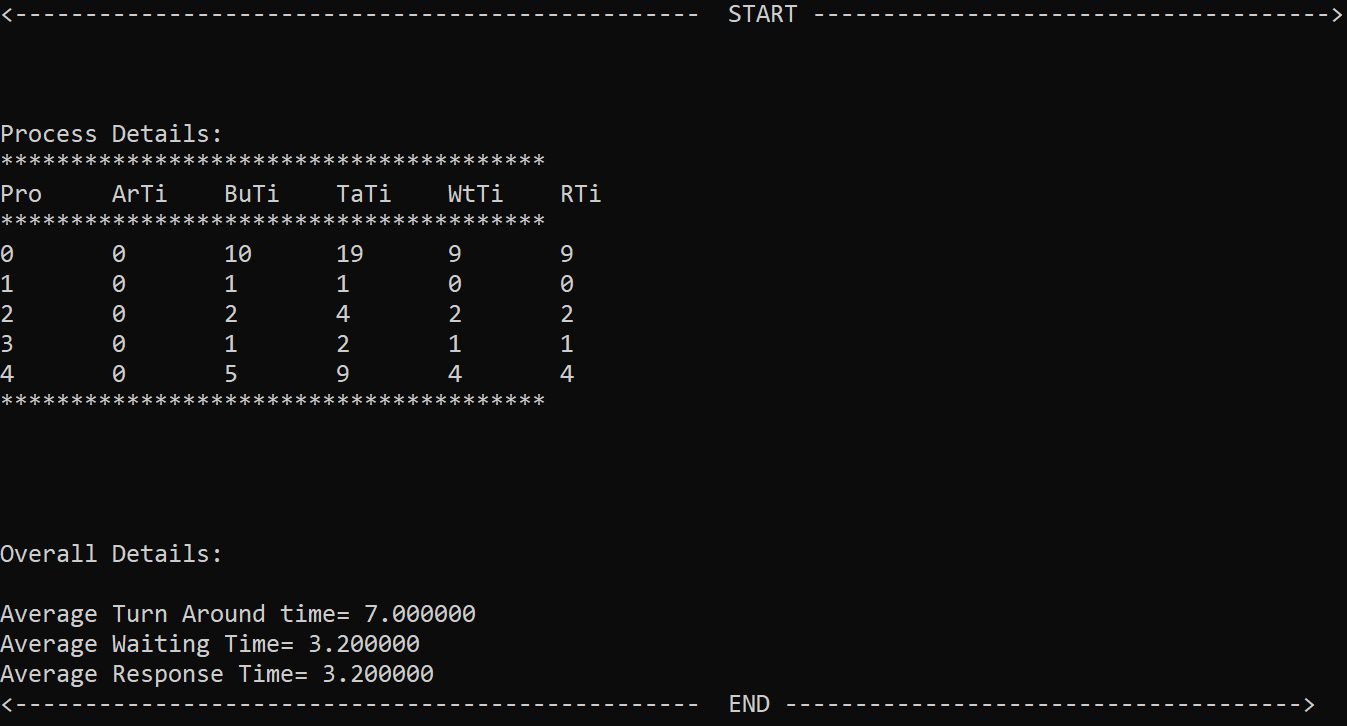
Input Choice 1:



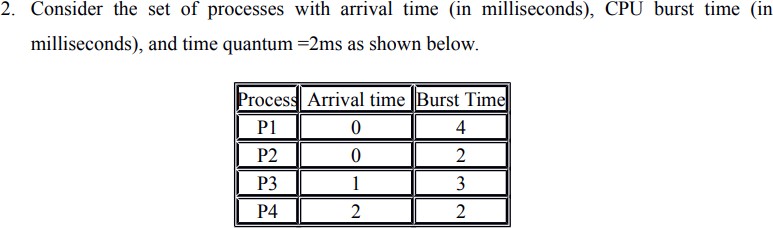
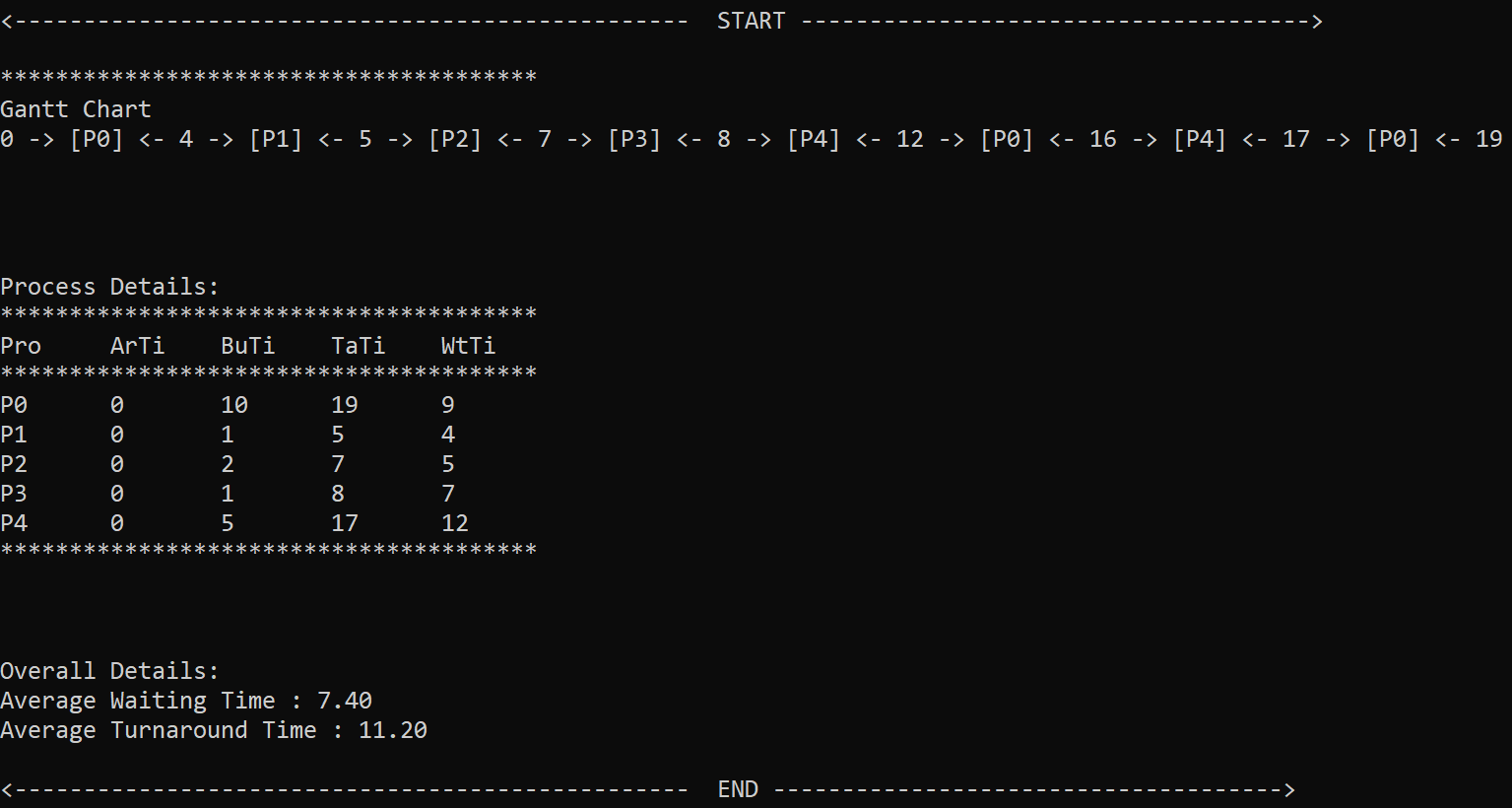
Input Choice 2:



Input Choice 3:

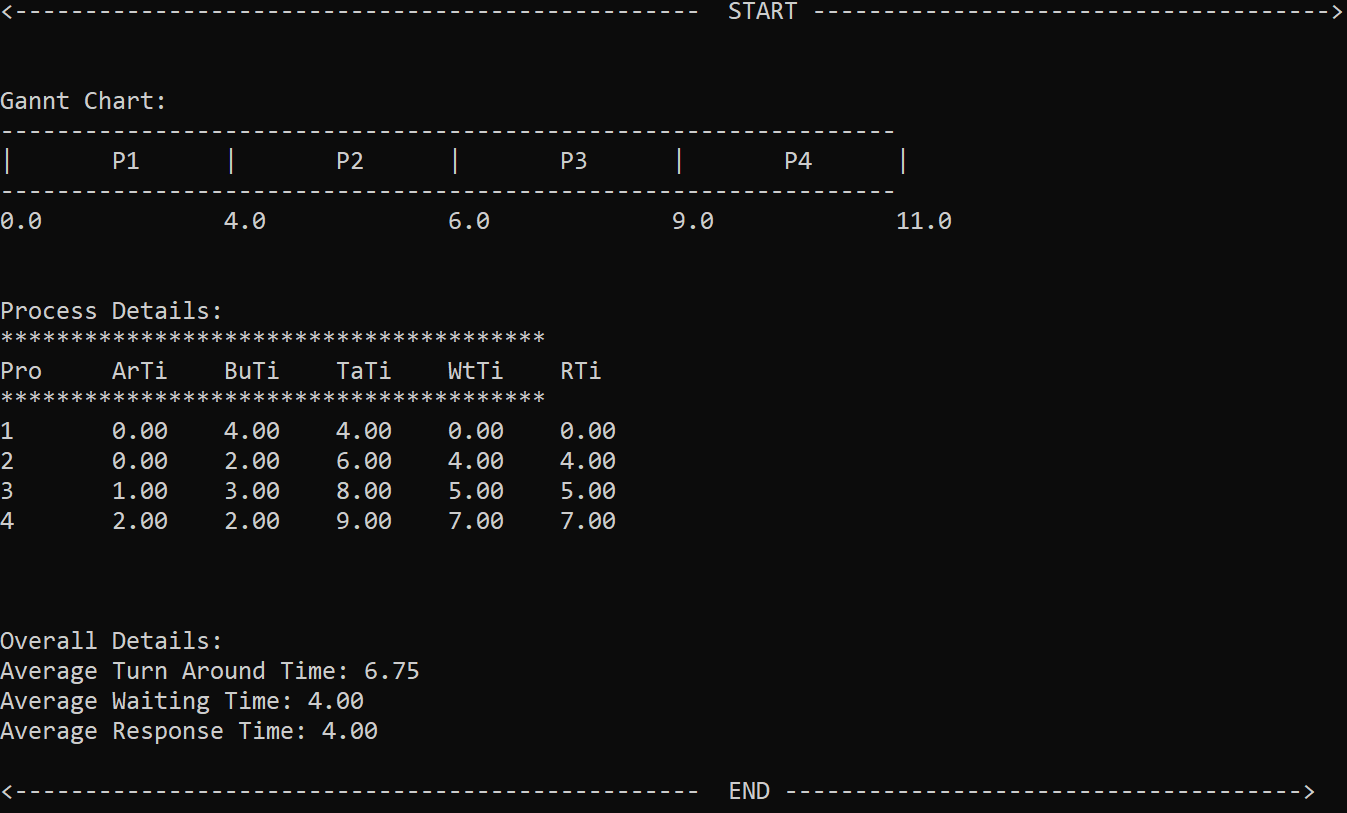
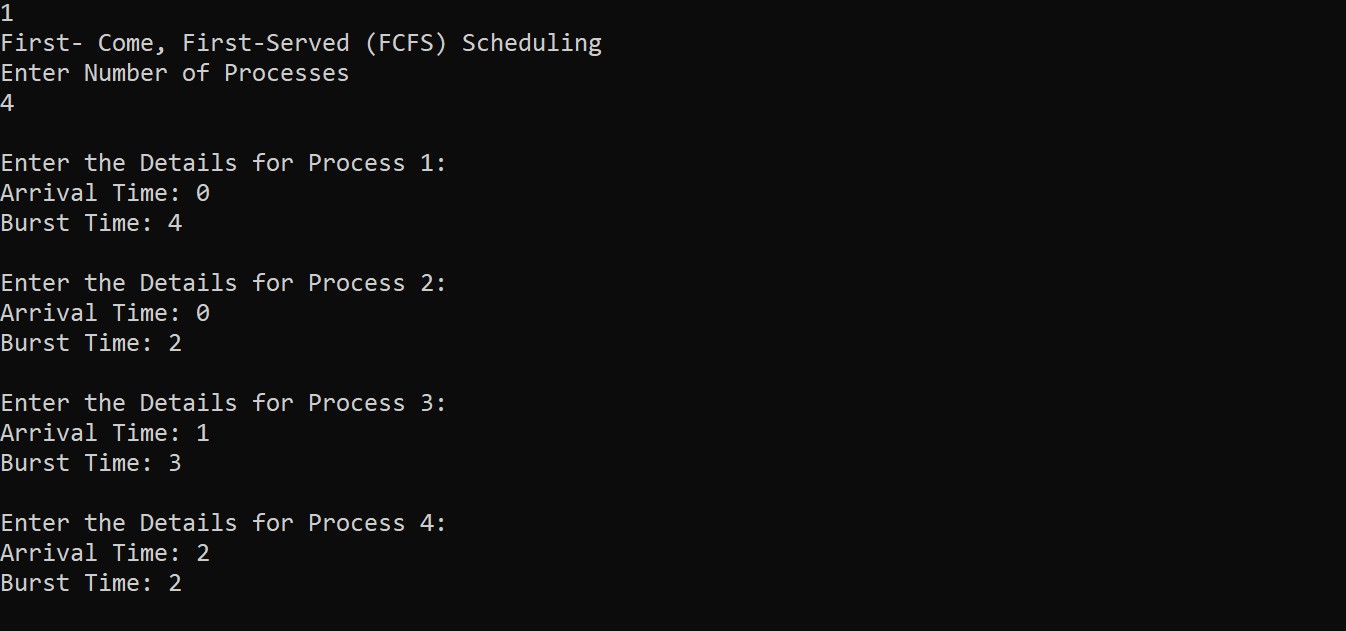


Input Choice 4:

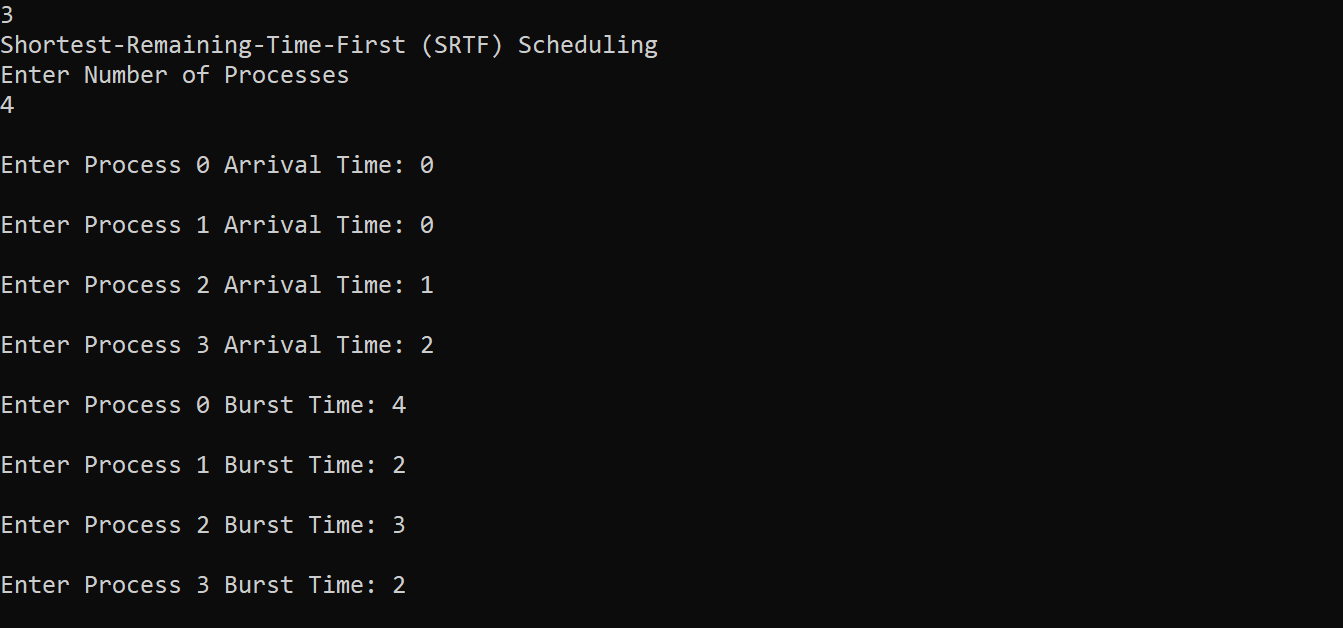
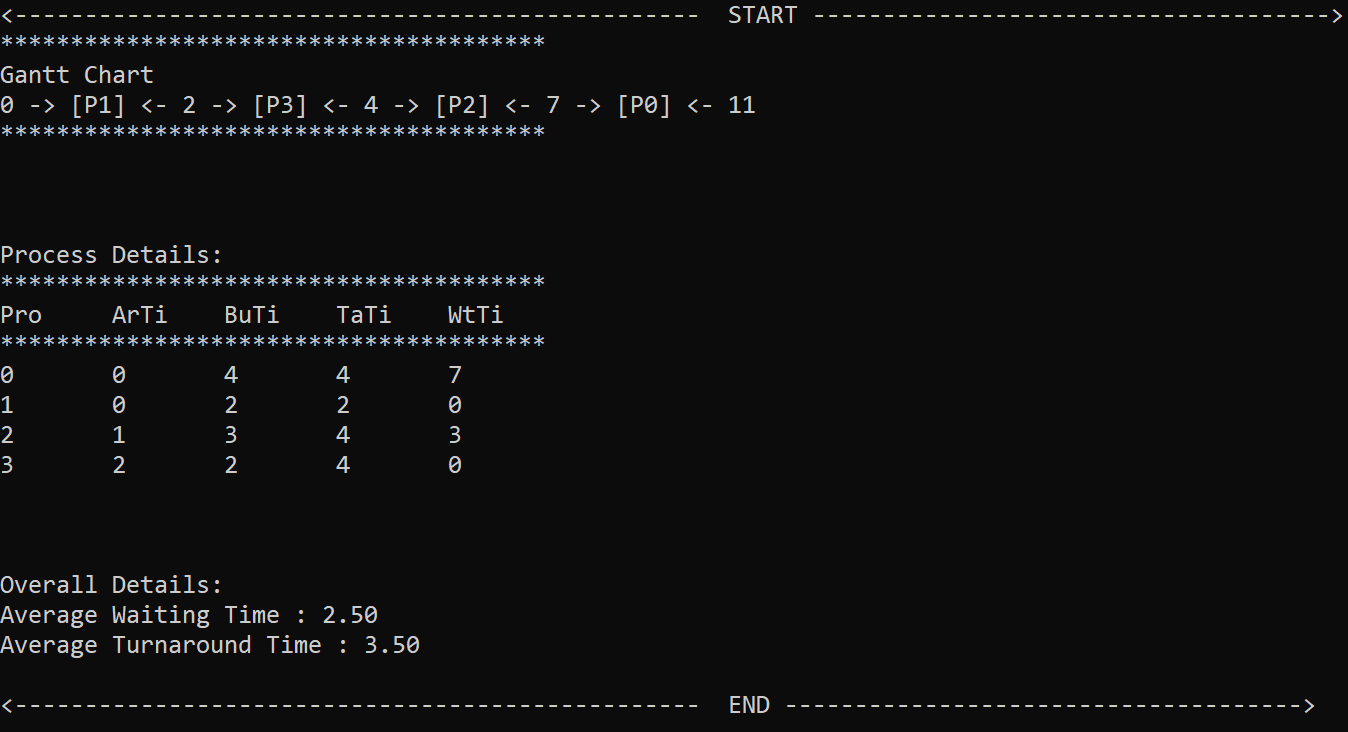
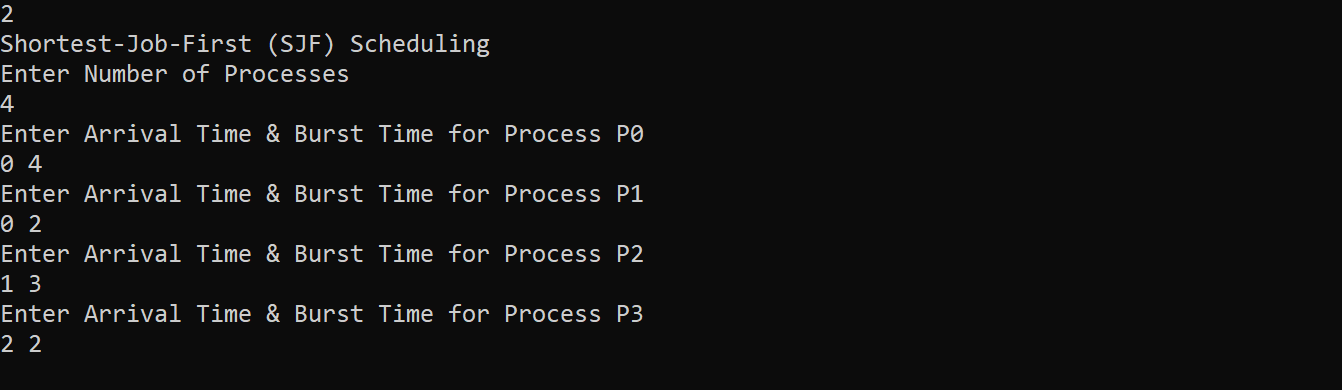


* On analysing the results of the algorithm, the minimum average waiting time is Shortest Job First (SJF) and shortest Remaining Time First (SRTF).

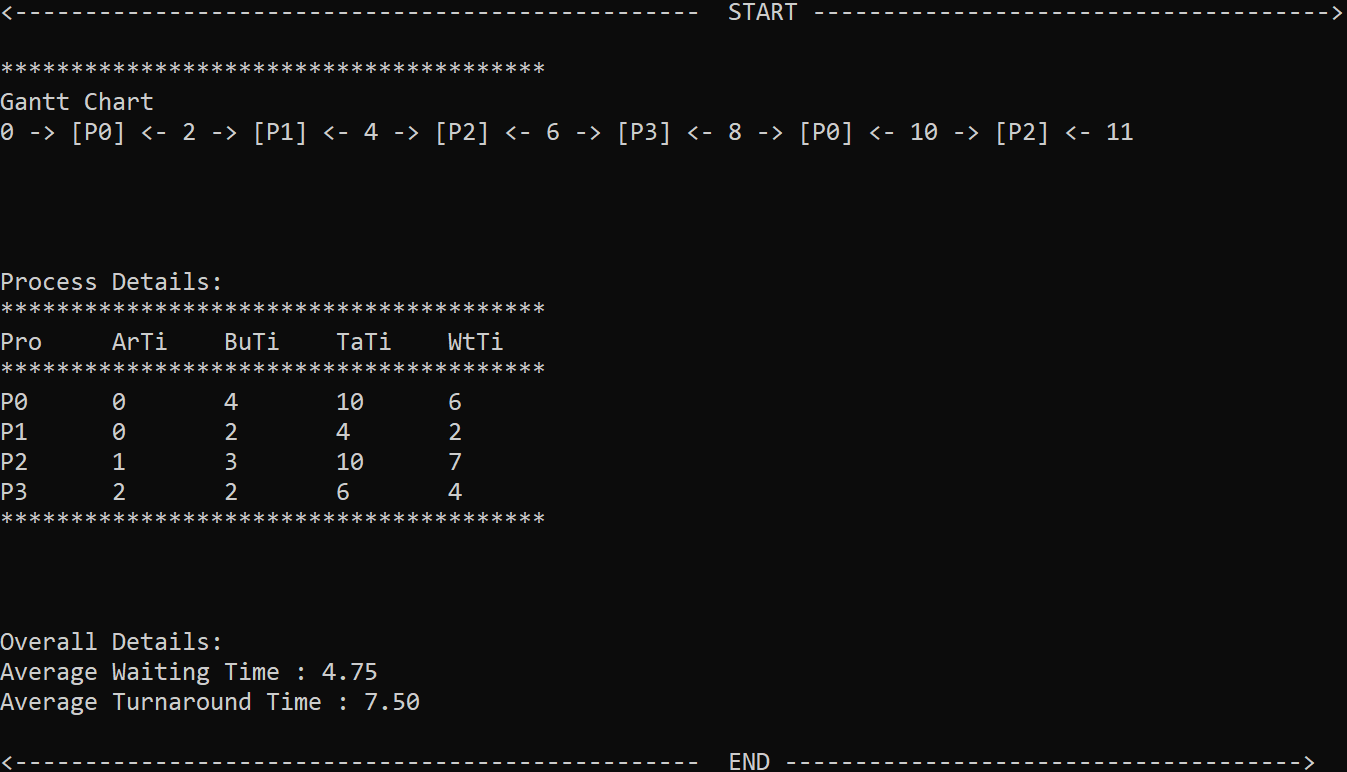
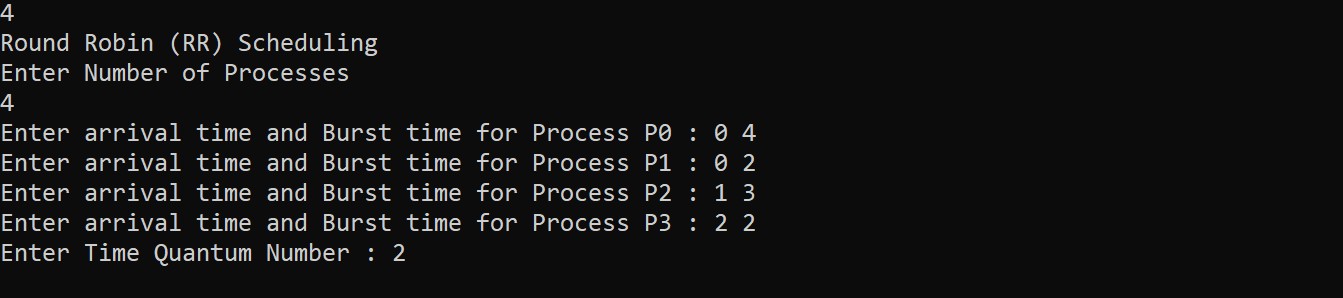
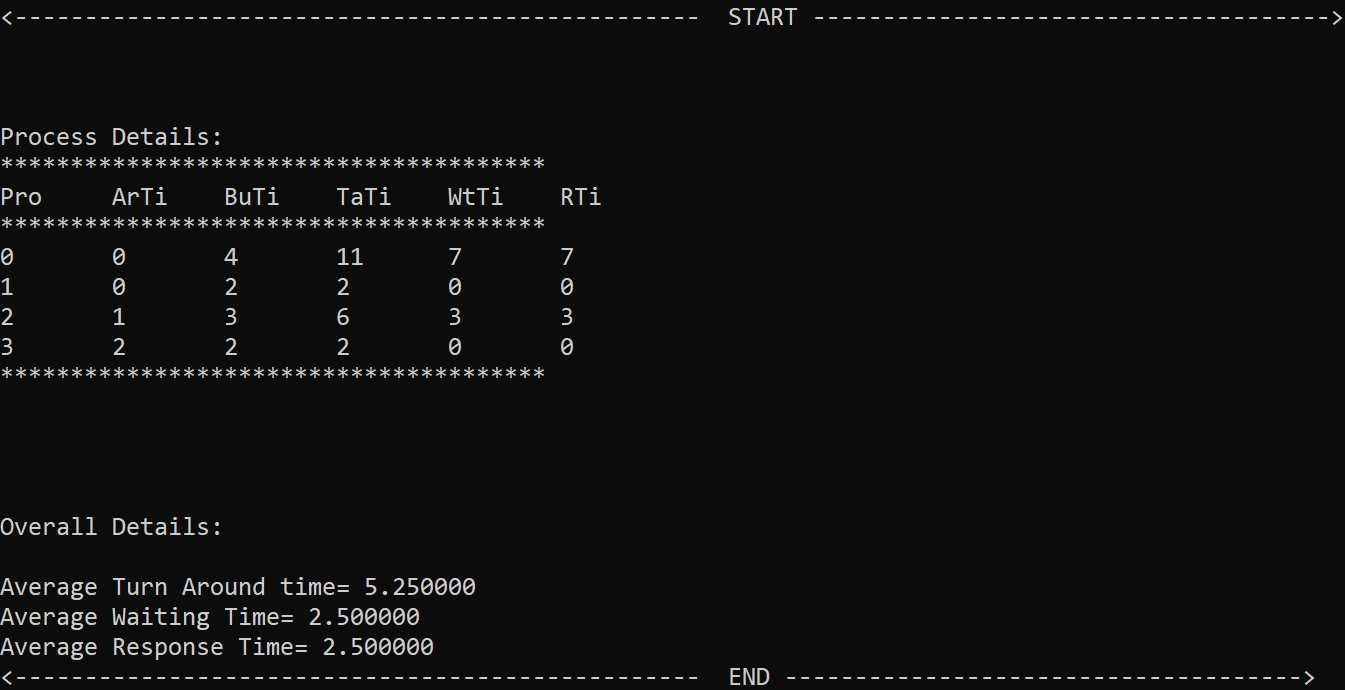
Input Choice 1:



Input Choice 2:



Input Choice 3:



Input Choice 4:

* On analysing the results of the algorithm, the minimum average waiting time is Shortest Job First (SJF) and shortest Remaining Time First (SRTF).