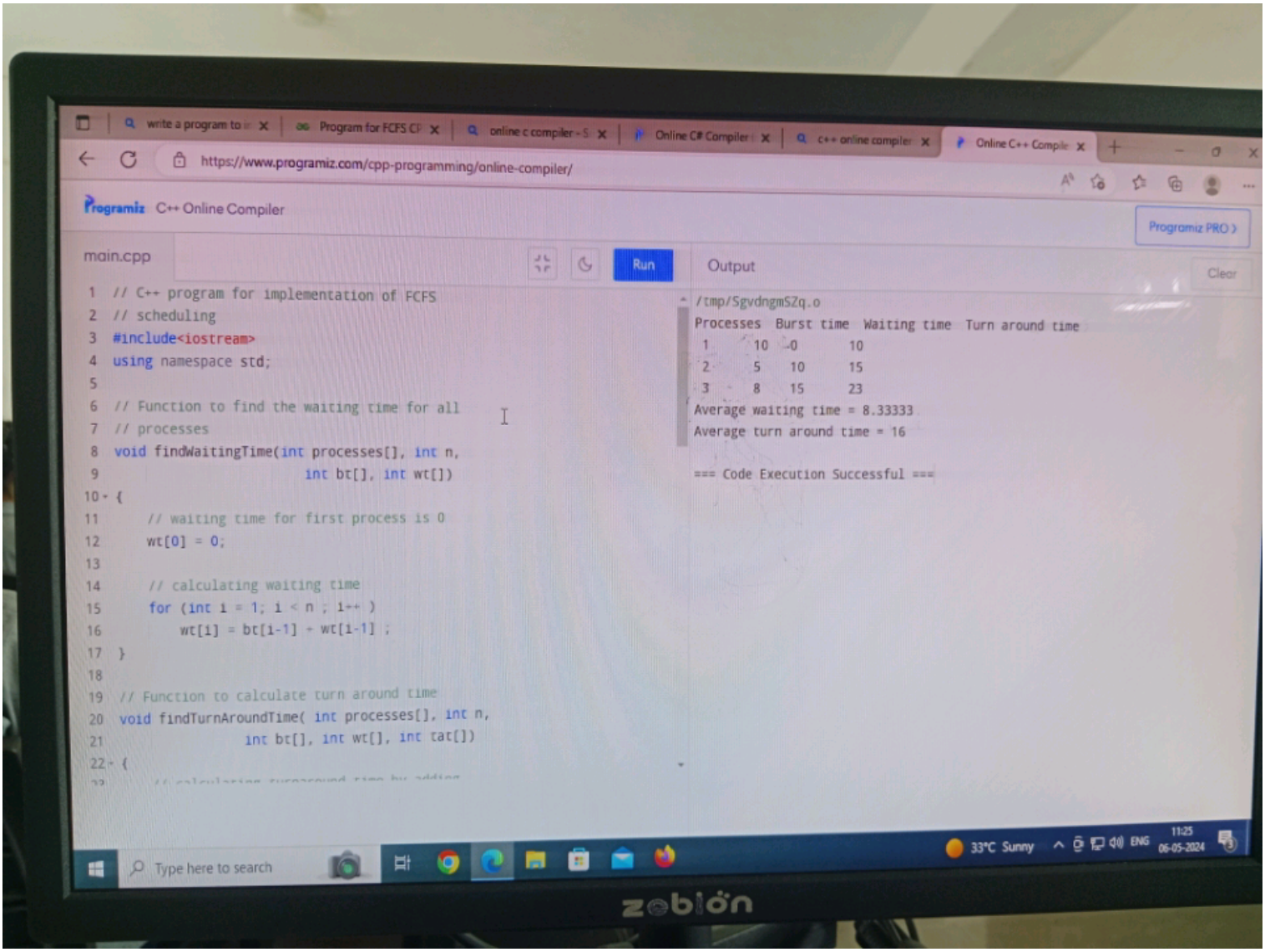
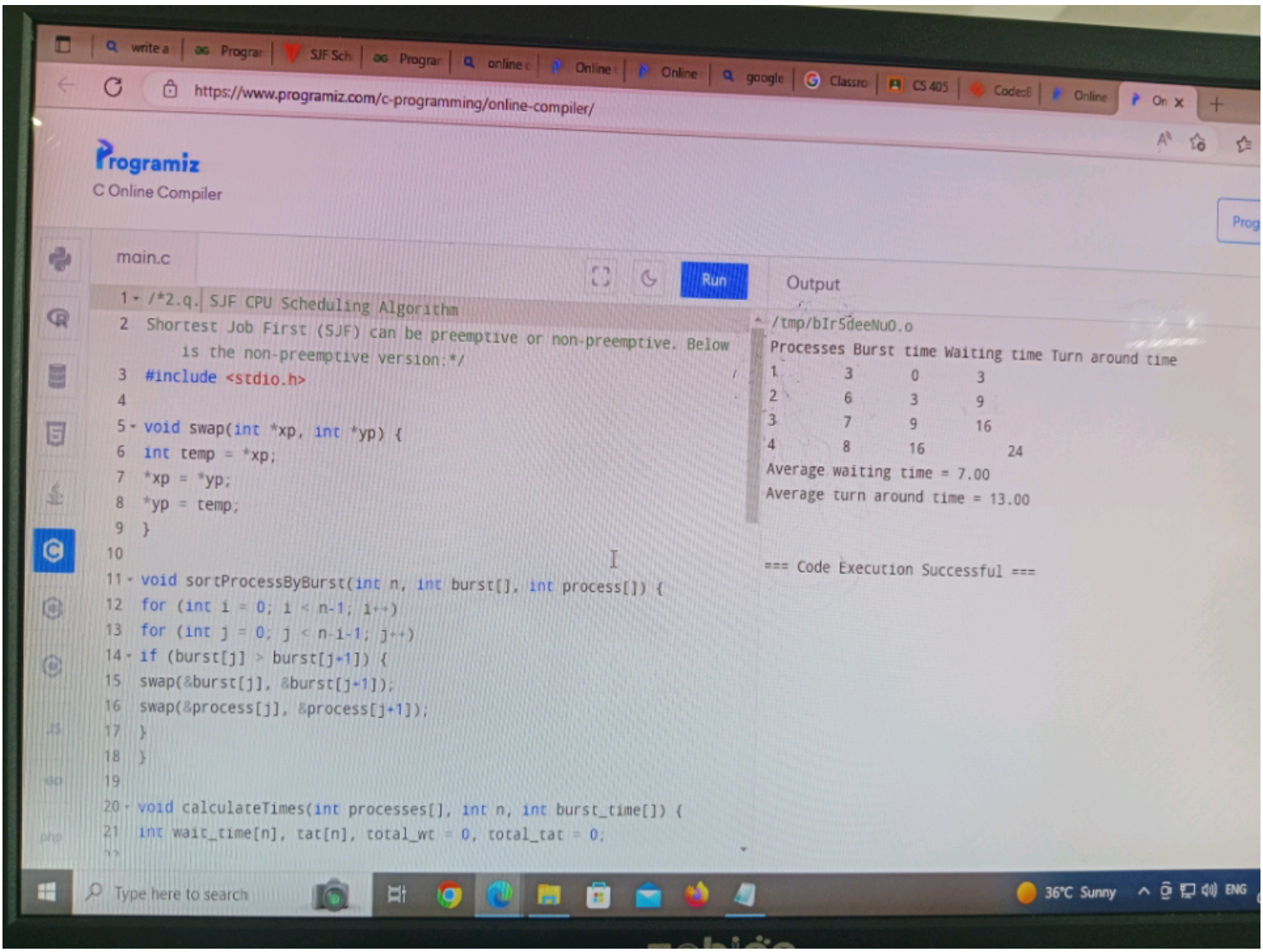
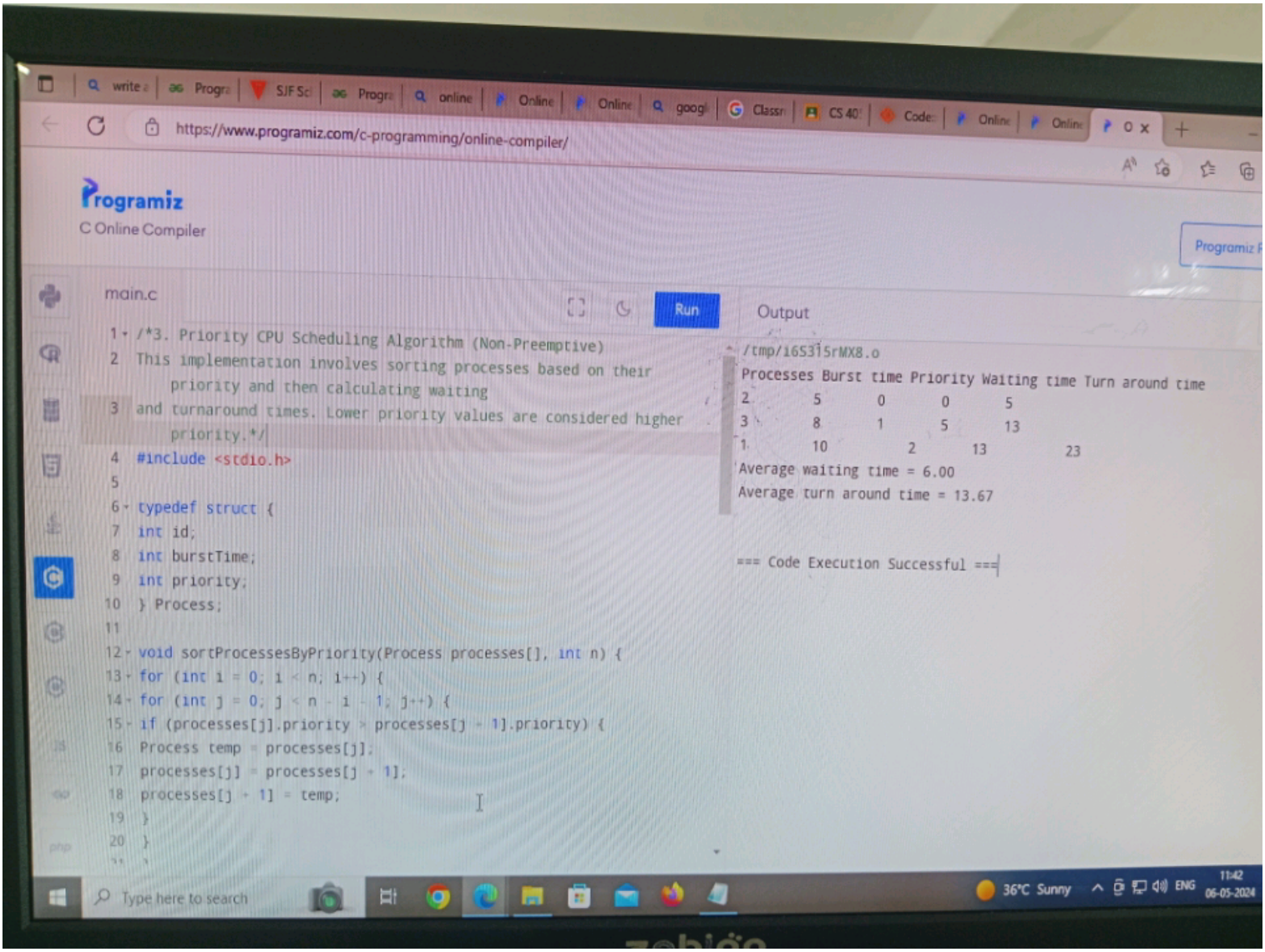
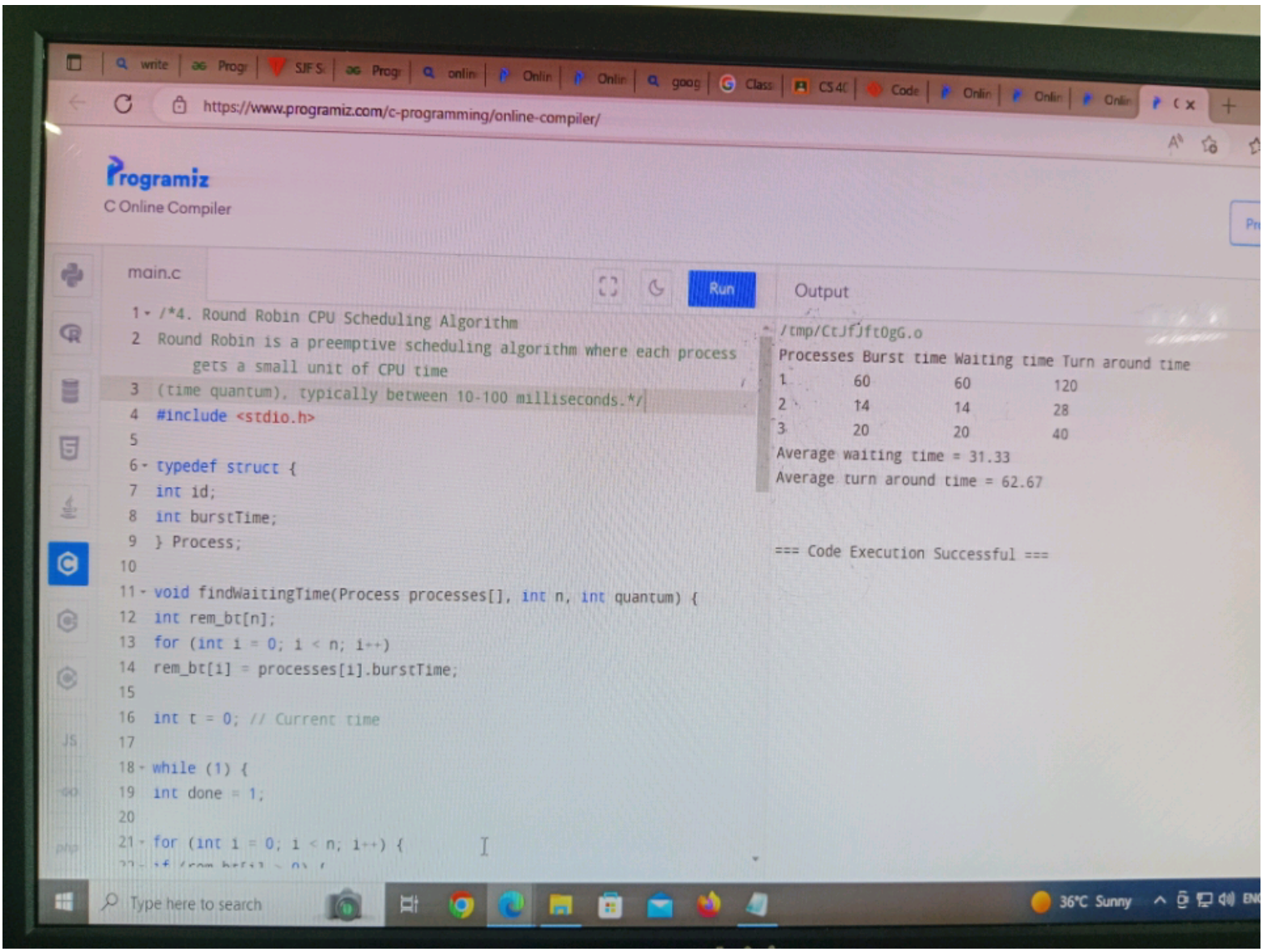
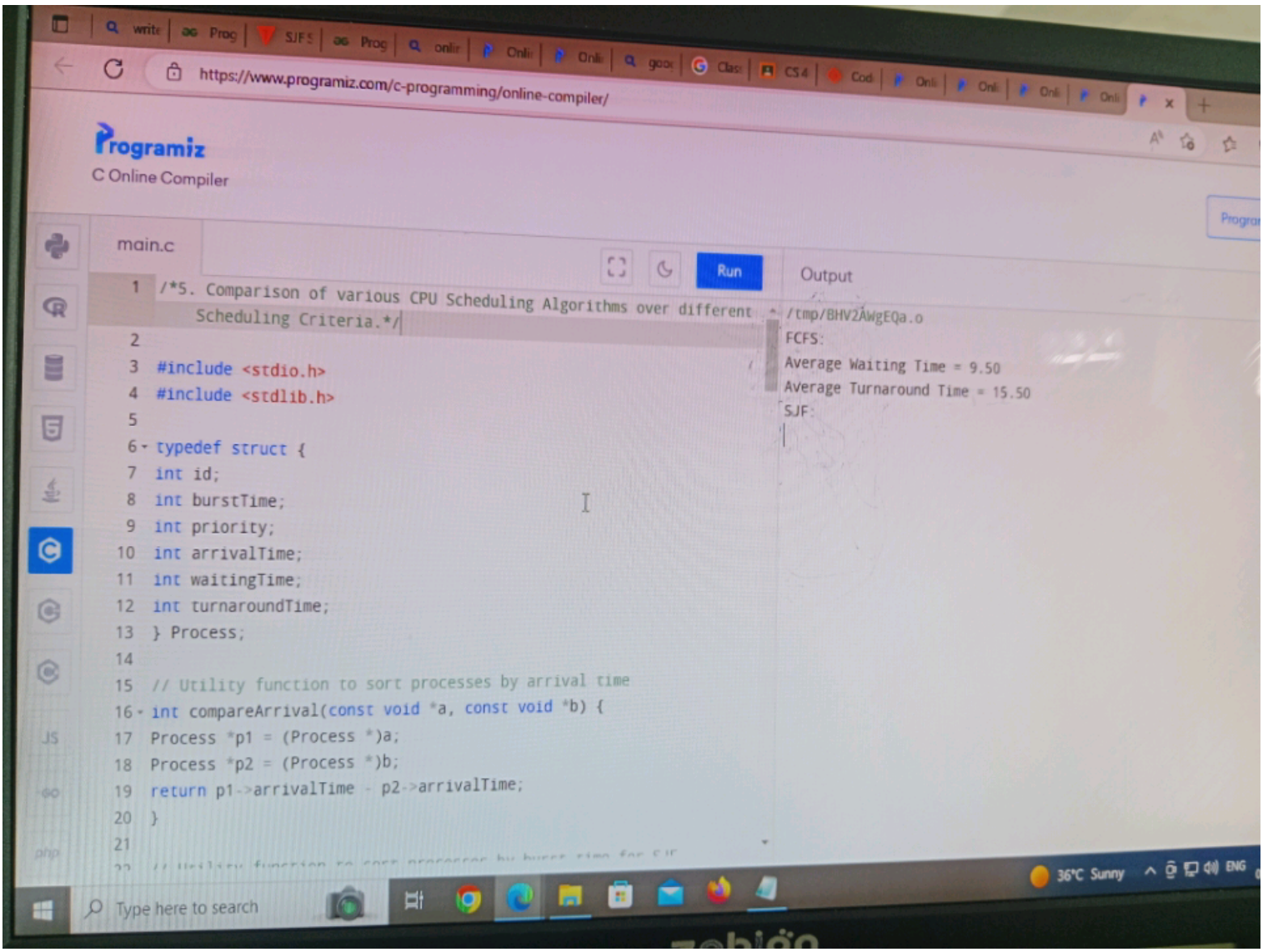

write a program to
Program for FCFS CP
online c compiler - S
Online C# Compiler
c++ online compiler
Online C++ Compile
https://www.programiz.com/cpp-programming/online-compiler/
Programiz C++ Online Compiler
Programiz PRO
main.cpp
Run
Output
Clear
// C++ program for implementation of FCFS
// scheduling
#include<iostream>
using namespace std;
// Function to find the waiting time for all
// processes
void findWaitingTime(int processes[], int n,
int bt[], int wt[])
{
// waiting time for first process is 0
wt[0] = 0;
// calculating waiting time
for (int i = 1; i < n ; i++)
wt[i] = bt[i-1] + wt[i-1] ;
}
// Function to calculate turn around time
void findTurnAroundTime(int processes[], int n,
int bt[], int wt[], int tat[])
{
/tmp/SgvdngmSZq.o
Processes Burst time Waiting time Turn around time
1 10 0 10
2 5 10 15
3 8 15 23
Average waiting time = 8.33333
Average turn around time = 16
=== Code Execution Successful ===
Type here to search
33°C Sunny
ENG
11:25
06-05-2024
zebion


https://www.programiz.com/c-programming/online-compiler/
Programiz
C Online Compiler
main.c
Run
Output
/*2.q. SJF CPU Scheduling Algorithm
Shortest Job First (SJF) can be preemptive or non-preemptive. Below is the non-preemptive version:*/
#include <stdio.h>
void swap(int *xp, int *yp) {
int temp = *xp;
*xp = *yp;
*yp = temp;
}
void sortProcessByBurst(int n, int burst[], int process[]) {
for (int i = 0; i < n-1; i++)
for (int j = 0; j < n-i-1; j++)
if (burst[j] > burst[j+1]) {
swap(&burst[j], &burst[j+1]);
swap(&process[j], &process[j+1]);
}
}
void calculateTimes(int processes[], int n, int burst_time[]) {
int wait_time[n], tat[n], total_wt = 0, total_tat = 0;
/tmp/bIr5deeNuO.o
Processes Burst time Waiting time Turn around time
1 3 0 3
2 6 3 9
3 7 9 16
4 8 16 24
Average waiting time = 7.00
Average turn around time = 13.00
=== Code Execution Successful ===
Type here to search
36°C Sunny
ENG


https://www.programiz.com/c-programming/online-compiler/
Programiz
C Online Compiler
main.c
Run
Output
/*3. Priority CPU Scheduling Algorithm (Non-Preemptive)
This implementation involves sorting processes based on their priority and then calculating waiting and turnaround times. Lower priority values are considered higher priority.*/
#include <stdio.h>
typedef struct {
int id;
int burstTime;
int priority;
} Process;
void sortProcessesByPriority(Process processes[], int n) {
for (int i = 0; i < n; i++) {
for (int j = 0; j < n - i - 1; j++) {
if (processes[j].priority > processes[j + 1].priority) {
Process temp = processes[j];
processes[j] = processes[j + 1];
processes[j + 1] = temp;
}
}
/tmp/i6S3I5rMX8.o
Processes Burst time Priority Waiting time Turn around time
2 5 0 0 5
3 8 1 5 13
1 10 2 13 23
Average waiting time = 6.00
Average turn around time = 13.67
=== Code Execution Successful ===
Type here to search
36°C Sunny
ENG
11:42
06-05-2024


https://www.programiz.com/c-programming/online-compiler/
Programiz
C Online Compiler
main.c
Run
Output
/*4. Round Robin CPU Scheduling Algorithm
Round Robin is a preemptive scheduling algorithm where each process
gets a small unit of CPU time
(time quantum), typically between 10-100 milliseconds.*/
#include <stdio.h>
typedef struct {
int id;
int burstTime;
} Process;
void findWaitingTime(Process processes[], int n, int quantum) {
int rem_bt[n];
for (int i = 0; i < n; i++)
rem_bt[i] = processes[i].burstTime;
int t = 0; // Current time
while (1) {
int done = 1;
for (int i = 0; i < n; i++) {
/tmp/CtJfJft0gG.o
Processes Burst time Waiting time Turn around time
1 60 60 120
2 14 14 28
3 20 20 40
Average waiting time = 31.33
Average turn around time = 62.67
=== Code Execution Successful ===
Type here to search
36°C Sunny


https://www.programiz.com/c-programming/online-compiler/
Programiz
C Online Compiler
main.c
Run
Output
/*5. Comparison of various CPU Scheduling Algorithms over different Scheduling Criteria.*/
#include <stdio.h>
#include <stdlib.h>
typedef struct {
int id;
int burstTime;
int priority;
int arrivalTime;
int waitingTime;
int turnaroundTime;
} Process;
// Utility function to sort processes by arrival time
int compareArrival(const void *a, const void *b) {
Process *p1 = (Process *)a;
Process *p2 = (Process *)b;
return p1->arrivalTime - p2->arrivalTime;
}
/tmp/BHV2AWgEQa.o
FCFS:
Average Waiting Time = 9.50
Average Turnaround Time = 15.50
SJF:
Type here to search
36°C Sunny
ENG