**1.Process Creation**

#include <stdio.h>

#include <unistd.h>

int main() {

pid\_t pid = fork();

if (pid == -1) {

// Fork failed

printf("Process creation failed\n");

return 1;

}

else if (pid == 0) {

// Child process

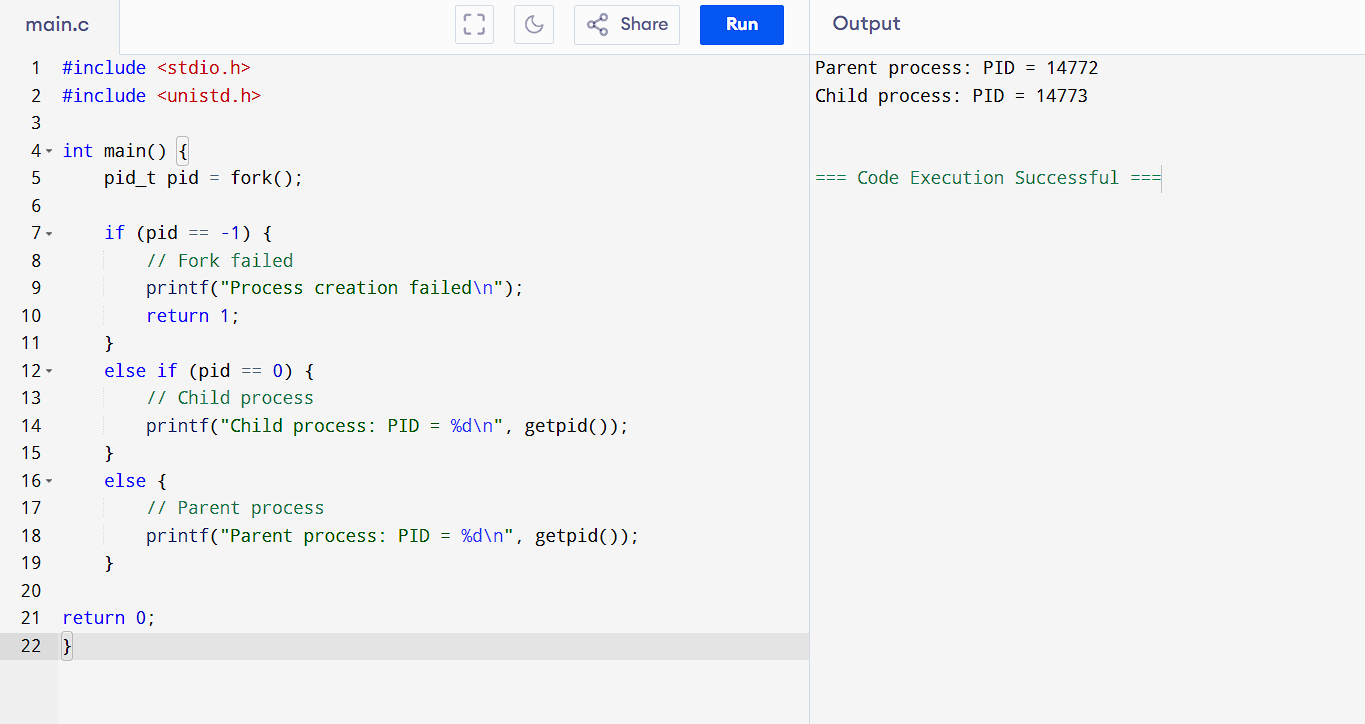
printf("Child process: PID = %d\n", getpid());

}

else {

// Parent process

printf("Parent process: PID = %d\n", getpid());

}

 return 0;

}

**2. File Copying**

#include <stdio.h>

int main() {

FILE \*source, \*destination;

char ch;

source = fopen("source.txt", "r");

if (source == NULL) {

printf("Error opening source file.\n");

return 1;

}

destination = fopen("destination.txt", "w");

if (destination == NULL) {

printf("Error opening destination file.\n");

fclose(source);

return 1;

}

while ((ch = fgetc(source)) != EOF) {

fputc(ch, destination);

}

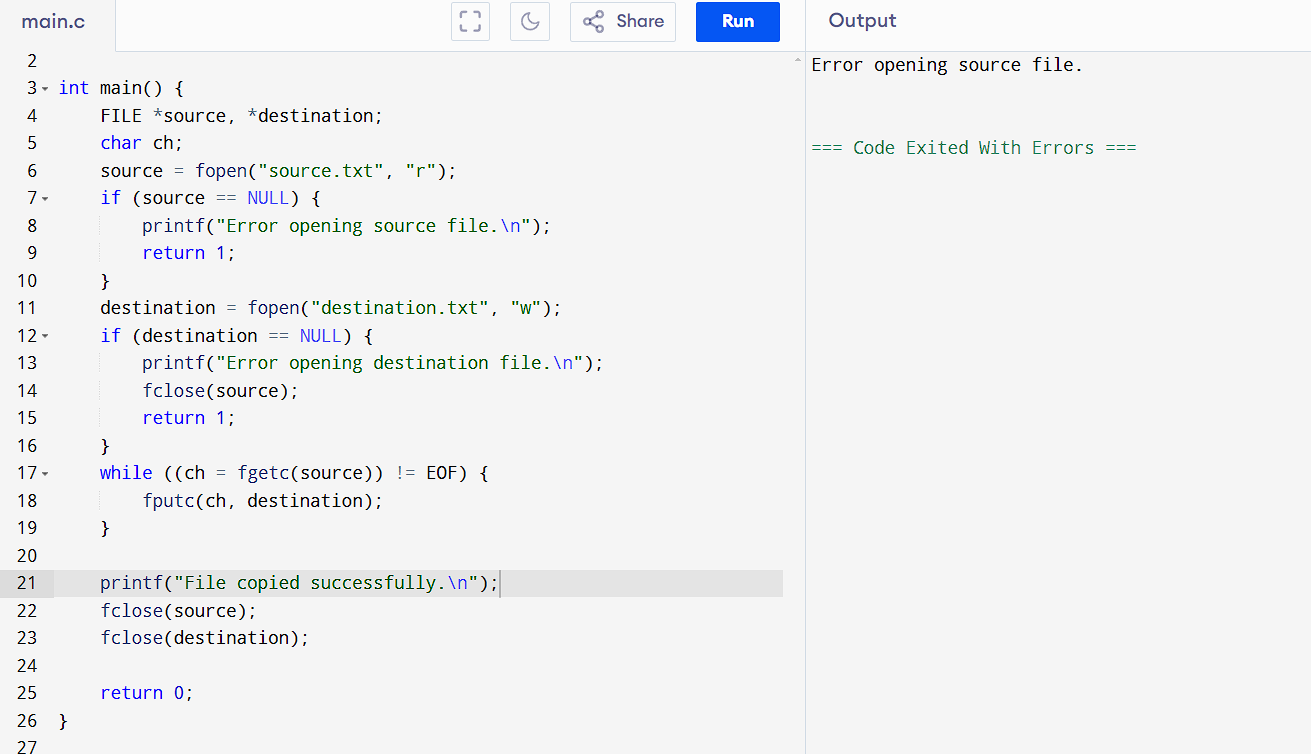
printf("File copied successfully.\n");

fclose(source);

fclose(destination);

return 0;

}

****

**3. FCFS Scheduling**

#include <stdio.h>

int main() {

int n;

printf("Enter number of processes: ");

scanf("%d", &n);

int arrival\_time[n], burst\_time[n], completion\_time[n], waiting\_time[n], turnaround\_time[n];

// Input arrival and burst times

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

printf("Enter arrival time and burst time for process %d: ", i + 1);

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

}

completion\_time[0] = arrival\_time[0] + burst\_time[0];

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

if (arrival\_time[i] > completion\_time[i - 1]) {

completion\_time[i] = arrival\_time[i] + burst\_time[i];

} else {

completion\_time[i] = completion\_time[i - 1] + burst\_time[i];

}

}

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

waiting\_time[i] = completion\_time[i] - arrival\_time[i] - burst\_time[i];

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

}

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

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

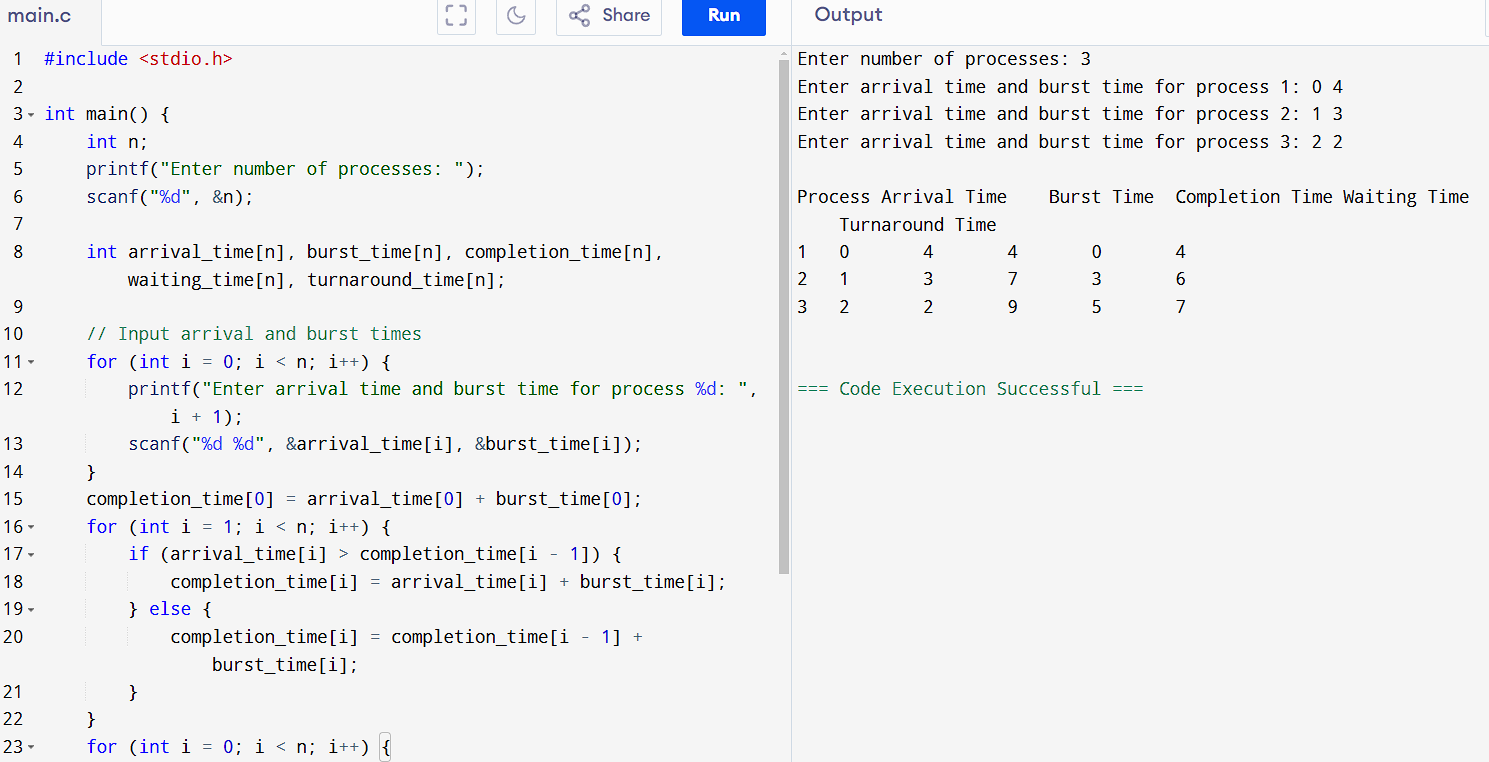
printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", i + 1, arrival\_time[i], burst\_time[i],

completion\_time[i], waiting\_time[i], turnaround\_time[i]);

}

return 0;

}



**4. SJF Scheduling**

#include <stdio.h>

int main() {

int n;

printf("Enter number of processes: ");

scanf("%d", &n);

int bt[n], wt[n], tat[n], ct[n], p[n];

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

p[i] = i + 1; // Process IDs (1 to n)

printf("Enter burst time for process %d: ", p[i]);

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

}

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

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

if (bt[i] > bt[j]) {

// Swap burst times

int temp = bt[i];

bt[i] = bt[j];

bt[j] = temp;

temp = p[i];

p[i] = p[j];

p[j] = temp;

}

}

}

wt[0] = 0; // The first process has no waiting time

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

wt[i] = bt[i - 1] + wt[i - 1]; // Waiting time is sum of burst times of previous processes

}

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

tat[i] = bt[i] + wt[i]; // Turnaround time = Burst time + Waiting time

ct[i] = tat[i]; // Completion time = Turnaround time

}

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

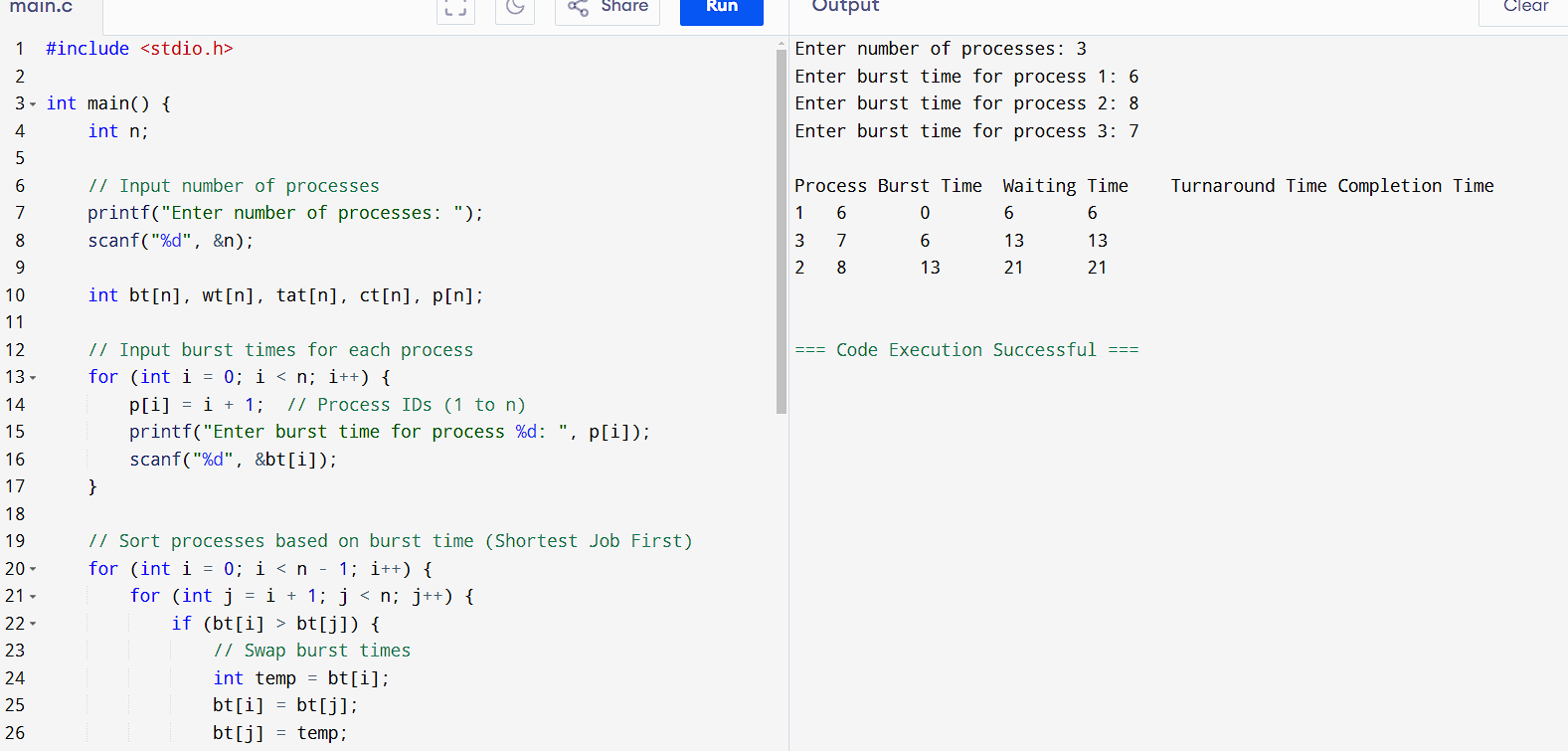
for (int i = 0; i < n; i++) {

printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", p[i], bt[i], wt[i], tat[i], ct[i]);

}

return 0;

}

****

**5. Priority Scheduling**

#include <stdio.h>

typedef struct {

int id;

int burst\_time;

int priority;

int waiting\_time;

int turnaround\_time;

int completion\_time;

} Process;

void sortProcessesByPriority(Process processes[], int n) {

Process temp;

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

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

if (processes[i].priority > processes[j].priority) {

temp = processes[i];

processes[i] = processes[j];

processes[j] = temp;

}

}

}

}

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

processes[0].waiting\_time = 0; // The first process has no waiting time

processes[0].completion\_time = processes[0].burst\_time;

// Calculate waiting time, turnaround time, and completion time

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

processes[i].waiting\_time = processes[i - 1].completion\_time;

processes[i].completion\_time = processes[i].waiting\_time + processes[i].burst\_time;

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

}

}

void printResults(Process processes[], int n) {

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

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

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

printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",

processes[i].id,

processes[i].burst\_time,

processes[i].priority,

processes[i].waiting\_time,

processes[i].turnaround\_time,

processes[i].completion\_time);

total\_waiting\_time += processes[i].waiting\_time;

total\_turnaround\_time += processes[i].turnaround\_time;

}

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

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

}

int main() {

int n;

// Input the number of processes

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

scanf("%d", &n);

Process processes[n];

// Input burst time and priority for each process

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

processes[i].id = i + 1;

printf("Enter burst time and priority for process %d: ", i + 1);

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

}

// Sort processes based on priority (higher priority comes first)

sortProcessesByPriority(processes, n);

// Calculate waiting time, turnaround time, and completion time

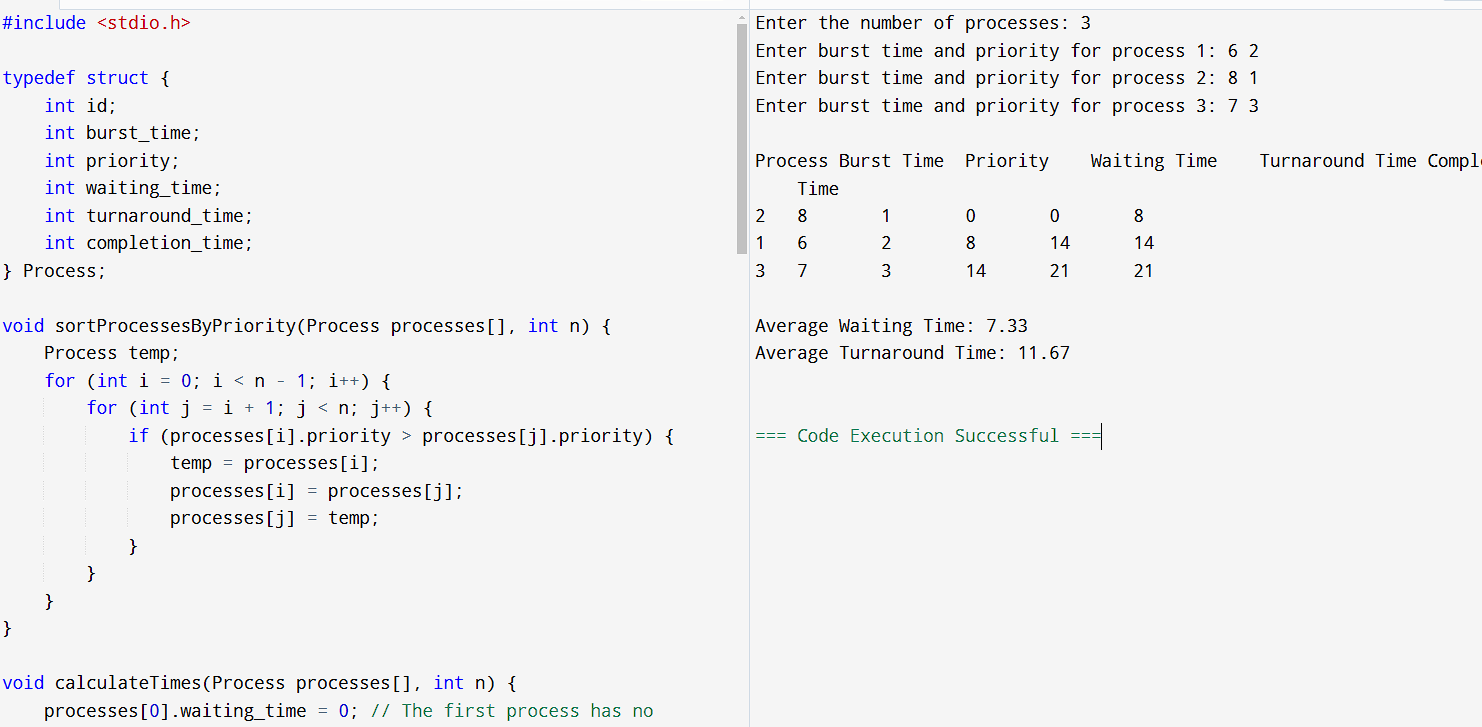
calculateTimes(processes, n);

// Display results

printResults(processes, n);

return 0;

}

****