Operating Systems Lab 07

The First-Come-First-Serve Scheduling Algorithms

|  |
| --- |
| **Objective** |
| * To implement the First-Come-First-Serve (FCFS) short-term scheduling algorithms in C language. |

# Short-Term Scheduling

Short-term scheduling, or *dispatching*, involves scheduling processes to execute on a processor. Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the short-term scheduler. It selects a process from the processes in memory that are ready to execute and allocates the CPU to that process.

There are various algorithms available for scheduling processes. Some of these are discussed in this manual while the rest are left for the students as tasks.

## First Come First Serve (FCFS)

FCFS is a relatively simple algorithm, which treats the processes strictly in order of arrival. Processor is allocated to the process which arrives first. Once the processor is allocated to a process, it is released when that process finishes. When the processor is available, the scheduler selects the next process to be executed.

The codes provided in Example#01 and Example#02 are implementations of the FCFS algorithms. In the code provided in Example#01, it is assumed that all processes are present in the system at the same time; that is, all processes have arrived in the system. However, the processes usually arrive at different times. In order to accommodate processes arriving at different times, the code in Example#01 can be modified as given in Example#02.

**Example #01:**

#include<stdio.h>

void main ()

{

int st[20], wt[20], tat[20], i, n;

float wt\_avg, tat\_avg;

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

scanf("%d", &n);

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

{

printf("\nEnter Burst/Service time for process%d: ", i);

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

}

wt[0] = wt\_avg = 0;

tat[0] = tat\_avg = st[0];

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

{

wt[i] = wt[i-1] + st[i-1];

tat[i] = tat[i-1] + st[i];

wt\_avg = wt\_avg + wt[i];

tat\_avg = tat\_avg + tat[i];

}

printf("\n PROCESS \t SERVICE TIME \t WAITING TIME \t TURNAROUND TIME\n");

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

{

printf("\nP%d \t\t%d \t\t%d \t\t%d\n", i, st[i], wt[i], tat[i]);

}

printf("\nAverage waiting time:%f ", wt\_avg/n);

printf("\nAverage turnaround time:%f \n", tat\_avg/n);

}

**Example #02:**

#include<stdio.h>

void main ()

{

int at[20], st[20], wt[20], ft[20], tat[20], i, n;

float wt\_avg, tat\_avg;

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

scanf("%d", &n);

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

{

printf("\nEnter the Arrival time for process%d: ", i);

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

printf("\nEnter Burst/Service time for process%d: ", i);

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

}

wt[0] = wt\_avg = 0;

tat[0] = tat\_avg = st[0];

ft[0] = st[0];

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

{

if (at[i] <= at[i+1])

{

wt[i] = wt[i-1] + st[i-1];

ft[i] = wt[i] + st[i];

tat[i] = ft[i] - at[i];

wt\_avg = wt\_avg + wt[i];

tat\_avg = tat\_avg + tat[i];

}

}

printf("\n PROCESS \t SERVICE TIME \t WAITING TIME \t FINISH TIME \t TURNAROUND TIME\n");

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

{

printf("\nP%d \t\t%d \t\t%d \t\t%d \t\t%d\n", i, st[i], wt[i], ft[i], tat[i]);

}

printf("\nAverage waiting time:%f ", wt\_avg/n);

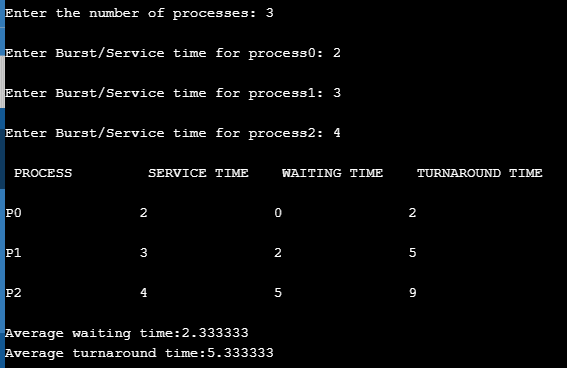
printf("\nAverage turnaround time:%f \n", tat\_avg/n);

}

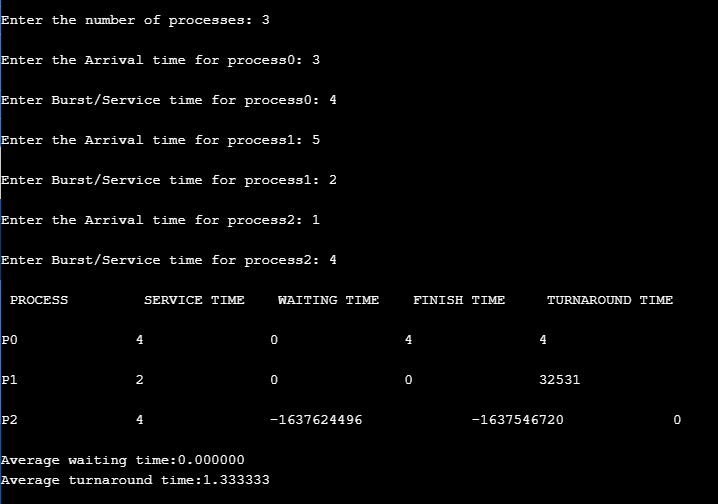
# TASKS

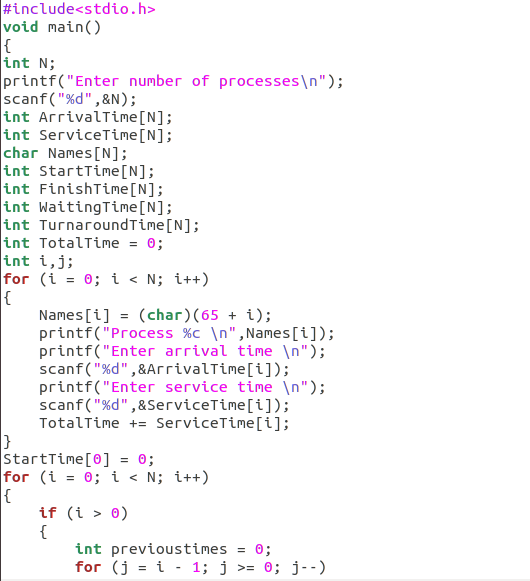
1. Implement the codes of Example#01 and Example#02. Discuss what you have learned after implementing and executing these two codes. Provide snapshots of the generated outputs.

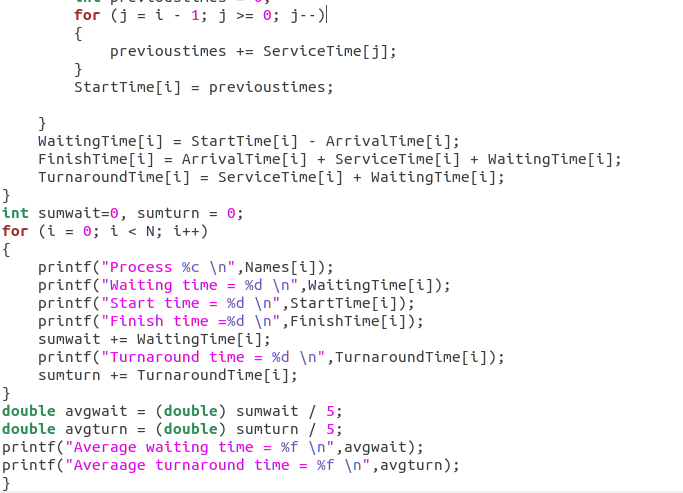
OUTPUT (Example 1):



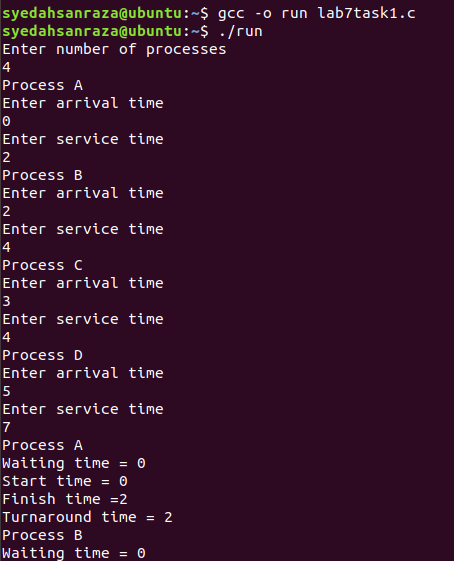
OUTPUT (Example 2):

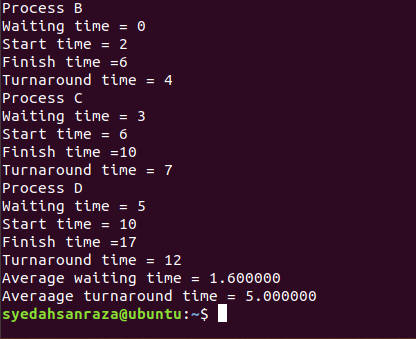


1. The FCFS source code given above assumes that the arrival times for processes are provided in ascending order of time. Modify the code in C language for random process arrival times.

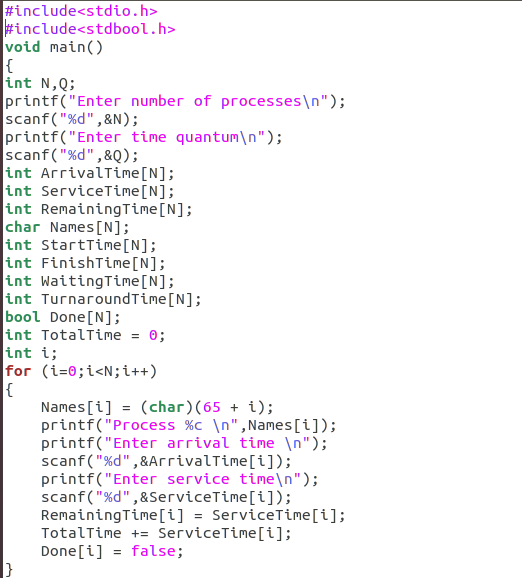


**Output:**

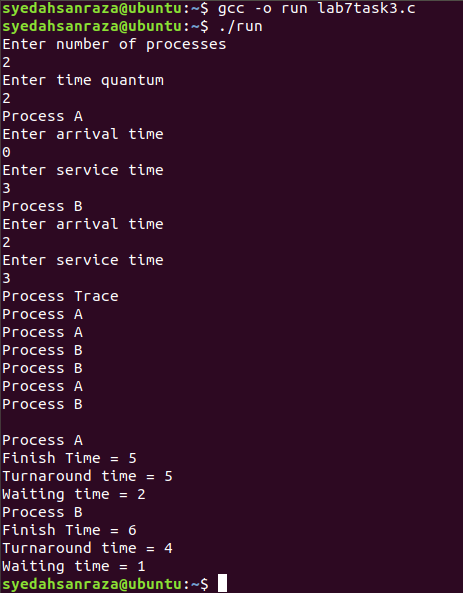




1. Implement Round Robin Algorithm in C language.

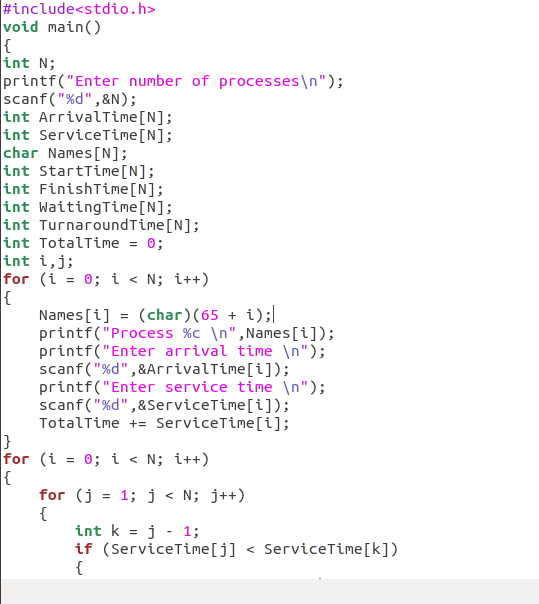


OUTPUT:

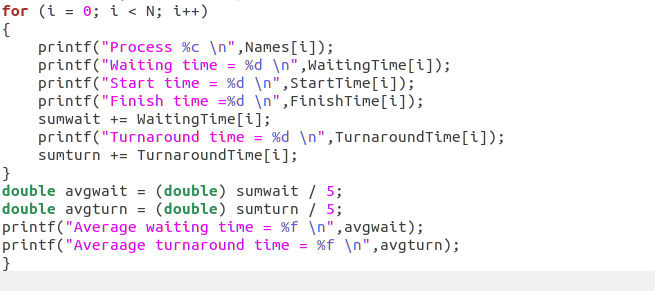


1. Implement SPN algorithm in C.

**Code:**







**Output:**

