CS F372 ASSIGNMENT

T V Chandra Vamsi[1], Vishwas Vasuki Gautam[2], Kaustubh Bhanj[3], Naren Vilva[4], Ruban S[5], Sathvik Bhaskarpandit[6], Shrikrishna Lolla[7] and Suraj S Prakash[8]

[1]2019A7PS0033H
[2]2019A3PS0443H
[3]2019A7PS0009H
[4]2019AAPS0236H
[5]2019A7PS0097H
[6]2019A7PS1200H
[7]2019AAPS0345H
[8]2019AAPS0317H

Table of Contents

Overview

Scheduling is the action of assigning resources to perform tasks. The resources may be processors, network links or expansion cards. The tasks may be threads, processes or data flows.

The scheduling activity is carried out by a process called scheduler. Schedulers are often designed so as to keep all computer resources busy (as in load balancing), allow multiple users to share system resources effectively, or to achieve a target quality-of-service.

Scheduling is fundamental to computation itself, and an intrinsic part of the execution model of a computer system; the concept of scheduling makes it possible to have computer multitasking with a single central processing unit (CPU).

In this project, the resource is a single processor (we have written our program under the assumption that it will run on a uniprocessor system). The tasks in the project are multithreaded processes. Each process has two threads; the monitor thread, which communicates with the main process to monitor the execution of the task thread, which is the thread that does the actual work. The scheduler here is the main process in the program.

CPU Scheduling Algorithms

First Come, First Serve (FCFS)

First Come, First Serve (FCFS), is the simplest scheduling algorithm. FIFO simply queues processes in the order that they arrive in the ready queue. This is commonly used for a task queue.

Since context switches only occur upon process termination, and no reorganization of the process queue is required, scheduling overhead is minimal. Throughput can be low, because long processes can be holding the CPU, causing the short processes to wait for a long time.

Starvation does not occur, because each process gets chance to be executed after a definite time.

Turnaround time, waiting time, response time depend on the order of their arrival and can be high.

Round-Robin (RR)

The scheduler assigns a fixed time quantum per process, and cycles through them. If process completes within that time-slice it gets terminated otherwise it is rescheduled after giving a chance to all other processes.

RR scheduling involves extensive overhead, especially with a small time quantum.

Good average response time, waiting time is dependent on number of processes, and not average process length.

Starvation can never occur, since no priority is given. Order of time unit allocation is based upon process arrival time, similar to FIFO.

Tools and Resources Used

We used the `fork()` system call to spawn C1, C2 and C3, the respective child processes. Multi-threading is done using the POSIX `pthread` API. Shared memory is used as an Inter-Process Communication (IPC) mechanism between the master and child processes. The shared memory is used by the master process to keep track of the execution progress of the child processes, and the child processes will update the shared memory according to its status. Another IPC mechanism employed is the `pipe` , where the child processes send the output of their execution to the master process through a dedicated pipe .To implement the actual scheduling of processes, we use a `semaphore` , where the "shared resource" is the processor itself (assuming the program runs on a uniprocessor system). Finally, to measure the turn around time and waiting time, we use the `gettimeofday()` function, which has an accuracy of 1 microsecond. We have given the time quantum as 20 milliseconds for all tests as we have experimentally found out that it is the optimal time quantum to produce meaningful results.

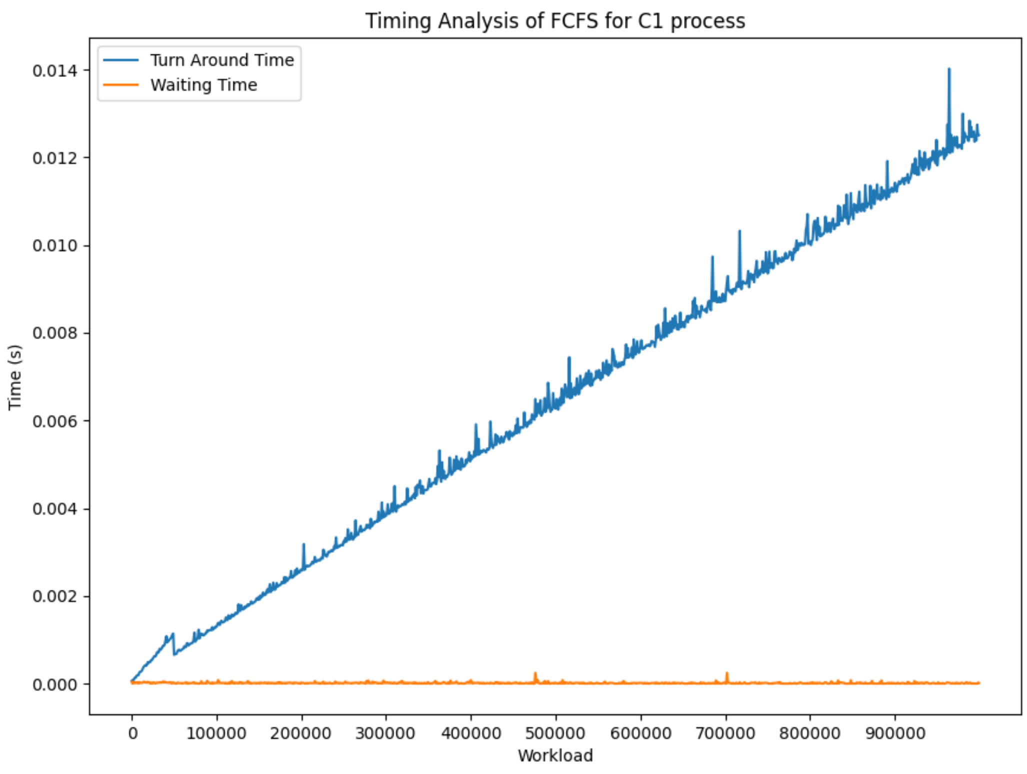
While the coding and testing were done on both Linux Virtual Machines (VMs) and physical machines running a Linux distribution, the time measurements were performed on a system that has (Arch) Linux as the sole operating system. The text file used has all the numbers from 25 to 1,000,000 in a random order (generated by a small Python program). The `gcc` complier is used to compile the program. The output is written to a `.csv` file, which will be read by a Python program that will generate a plot using the `matplotlib` library.

Lastly, we have a private GitHub repository where we keep track of the code and other resources pertaining to this project.

Performance Analysis and Plots

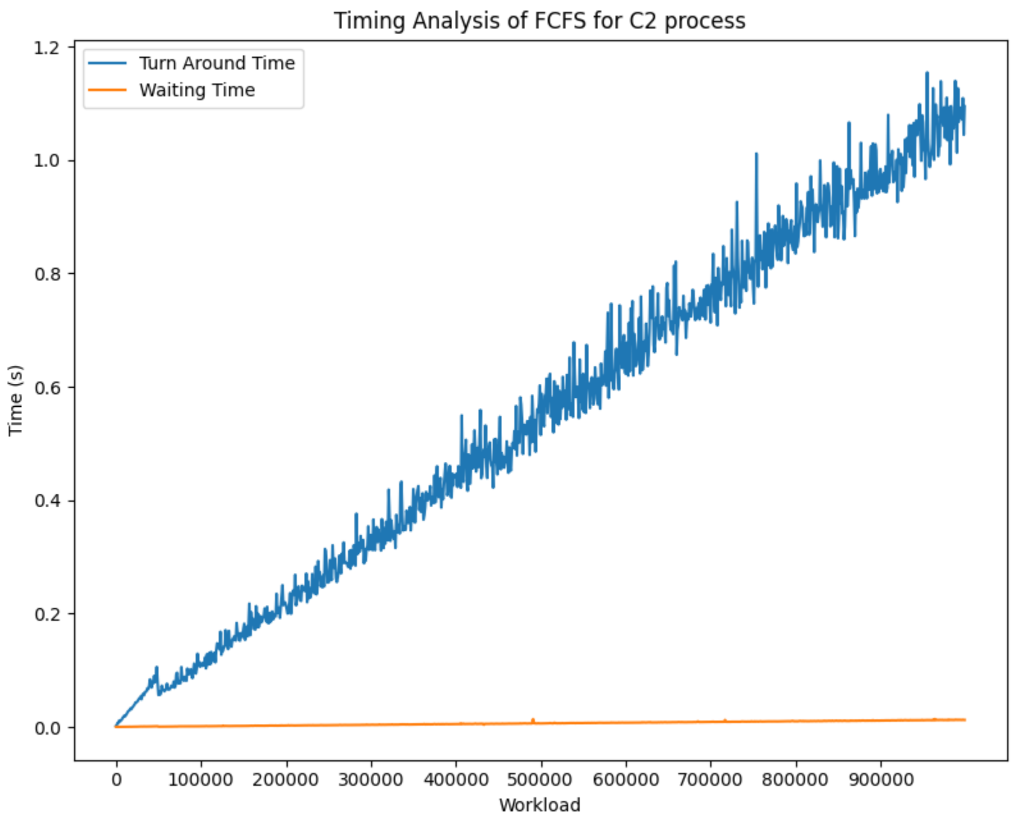
FCFS

C1



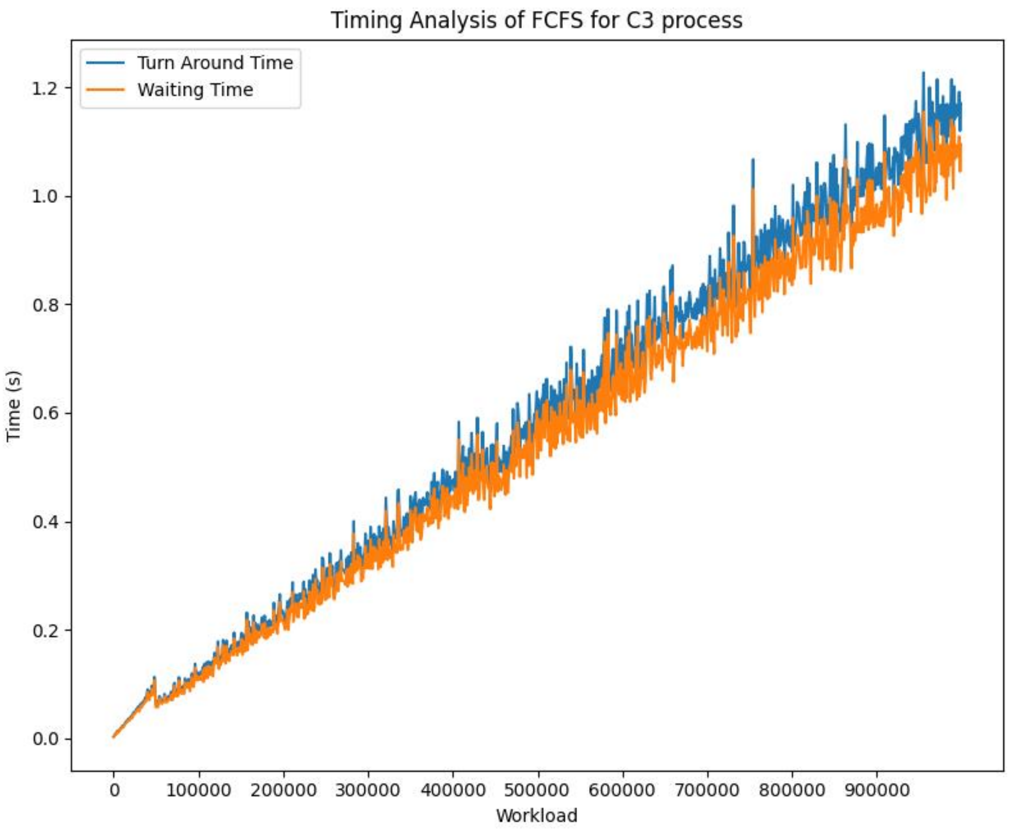
From the above plot, we observe that the turnaround time varies more or less linearly with the workload, while the waiting time seems to remain constant. This is because C1 is the first process that is scheduled; hence the waiting time is around a low time of less than 1 millisecond. The turnaround time varies linearly with workload.

C2



From the above plot, we observe that the turnaround time varies more or less linearly with the workload, while the waiting time seems to remain constant. This is because the waiting time for C2 depends on the waiting time and turnaround time for C1 (should more or less vary linearly with the sum of waiting time and turnaround time of C1), which is a very less value (less than 20 ms, as C1 is a compute-intensive task). The turnaround time varies linearly with workload.

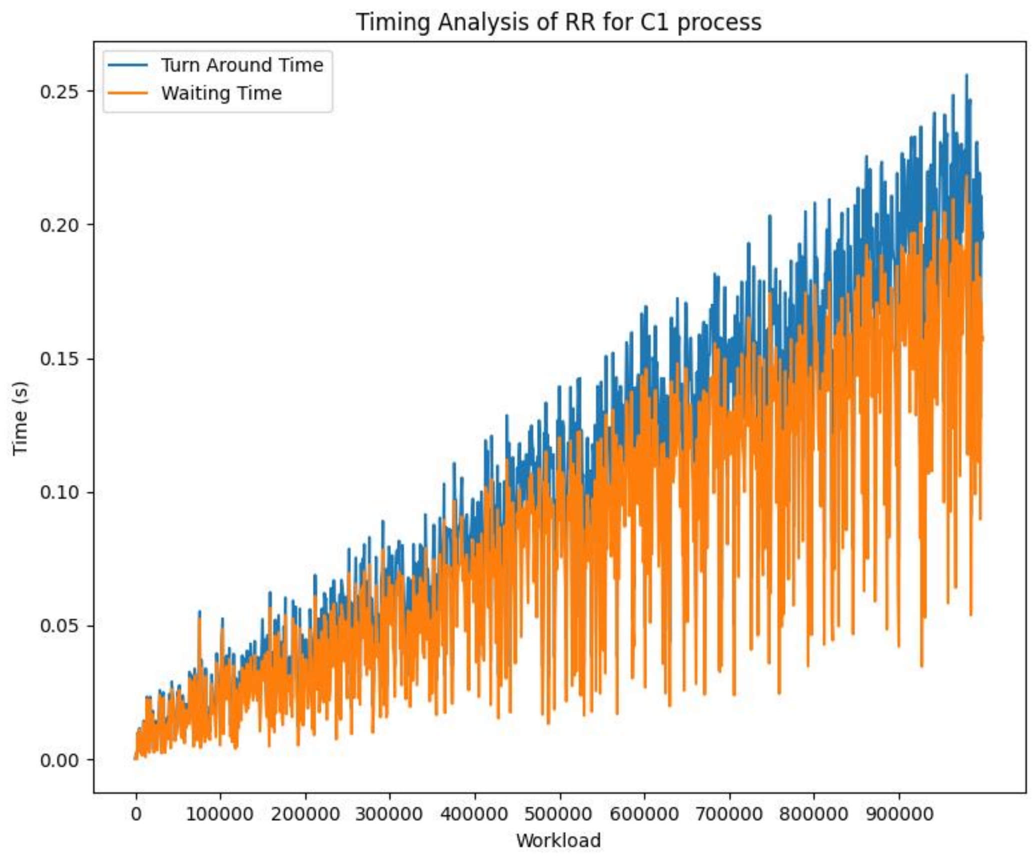
C3



From the above plot, we observe that both the waiting time and turnaround time vary more or less linearly with the workload. The waiting time varies linearly as it depends on the sum of turnaround time and waiting time of both C1 and C2 (as it is the last scheduled process), and the turnaround time for C2 increases by a lot with workload due to the fact that C2 is both a compute-intensive as well as I/O-intensive task. The turnaround time varies linearly with workload.

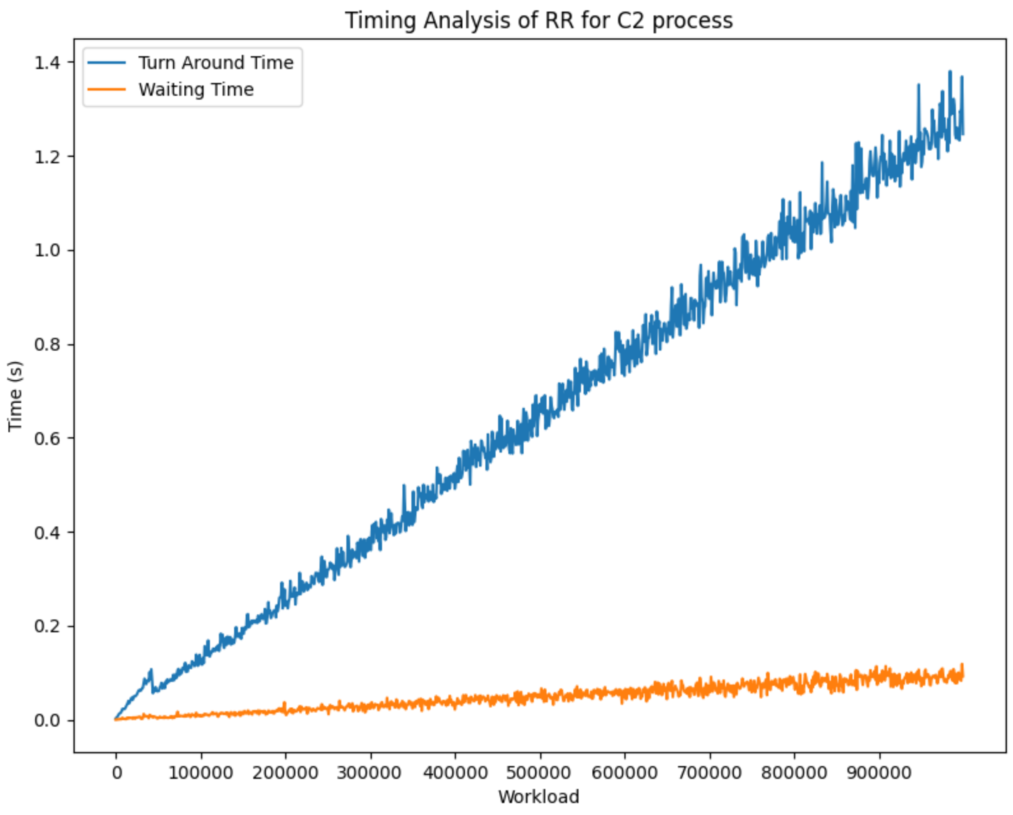
RR

C1



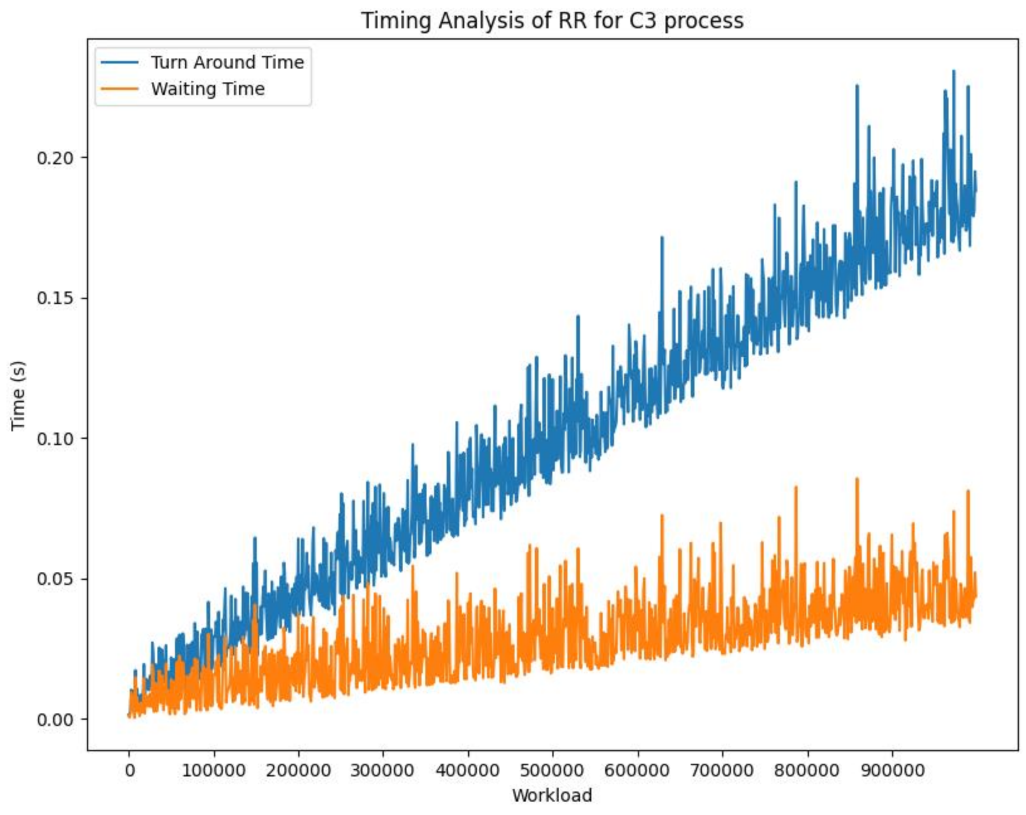
From the above plot, we observe that both turnaround time and waiting time vary somewhat linearly with workload, with the average slope of waiting time being less as compared to turnaround time. One possible explanation is that C3 is a compute and I/O intensive process and C2 is an I/O intensive process, hence the waiting time oscillates due to the frequent context-switching that occurs. The turnaround time also has a linear relationship with workload. The reason for the erratic plot is due to the fact that the scale on y-axis is very less, hence it has a higher tendency to capture the subtle variance in time measurements.

C2



From the above plot, we observe that both turnaround time and waiting time vary linearly with workload, with the slope of waiting time being very less as compared to turnaround time. This is because of the time quantum that we have given, which is 20 milliseconds (as mentioned before). The turnaround time will not change much as compared to FCFS since the workload remains same, just that the task is not run continuously and is exactly resumed from where it has been paused.

C3



From the above plot, we observe that both turnaround time and waiting time vary somewhat linearly with workload, with the average slope of waiting time being less as compared to turnaround time. One possible explanation is that C1 is a compute-intensive process and C2 is an I/O intensive process, hence C1 finishes faster than C2, and the waiting time after C1 finishes will depend mainly on C2. The turnaround time also has a linear relationship with workload since it is both an I/O and compute-intensive task, hence the turnaround time is like a superimposition of the turnaround time for both compute and I/O. The reason for the erratic plot is due to the fact that the scale on y-axis is very less, hence it has a higher tendency to capture the subtle variance in time measurements.

Conclusion

After performing the analysis for both the scheduling algorithms, we have come to the conclusion that FCFS can be used to obtain the results of the processes as soon as possible as it reduces the turnaround time and RR can be used to ensure a fair distribution of the resources to the processes by reducing the waiting time.

References

- Man pages of Linux
- Stackoverflow
- Geeks for Geeks
- Wikipedia
- Tutorial material from the course