Institute of Engineering and Technology

Operating Systems

Submission of Final Project Report – Semester 5

Instructor: Dr. Sanjay Chaudhary

1. **Project Title**

   CPU Scheduling Algorithms

2. **Team Members**

   - Pranali Raval – 131034
   - Rachana Solanki – 131038
   - Shaily Mishra – 131048
   - Stavan Pandya – 131056

3. **Brief Description**

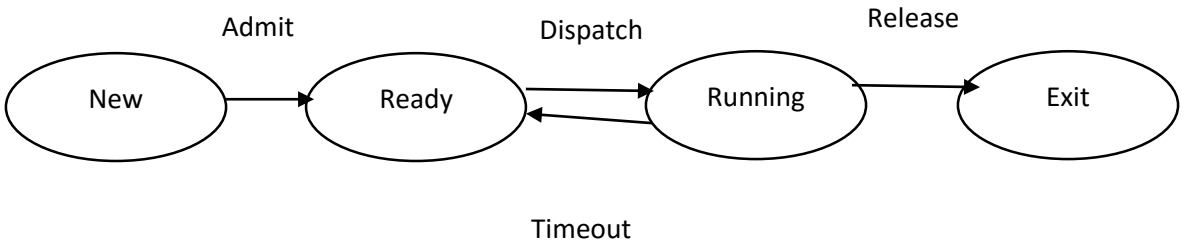
   As the final outcome of this project, we have implemented First Come First Serve (FCFS) and Round Robin (RR) algorithms for CPU scheduling. The program for a particular algorithm takes file containing arithmetic expressions as processes, parses those expressions to scheduler and calculates results. Based on algorithm specified, these results are displayed on terminal as well as are stored in a separate file with file name from which expression was taken, arrival time and time taken to execute the expression. In order to implement scheduler, we have used pthread and signal handling.

   3.1. Terminology used:

   - Turn-around time:
     Indicates time required from the time of submission to the time of completion.
   - Waiting time:
     Indicates sum of times spent in ready queue.
   - Throughput:
     Indicates number of processes completed per unit of time.

4. **Architecture**

   - Process Model



5. Technical Specifications

5.1. Assumptions

Following assumptions were made while implementing this project

- Time quantum for Round Robin algorithm = 1 second
- Parser only evaluates arithmetic expressions
- Time taken by parser to evaluate each expression = 1 second
- Only addition and multiplications are executed as per B.O.D.M.A.S. rule

5.2. Threads

For this project, each process acts as a thread. There are 3 files containing arithmetic expressions. The name of this file is parsed as argument to thread. Then thread method is created and parser is called and file is sent to parser. After that parser reads this files and evaluate arithmetic expression.

5.3. Signal Handling

For each thread being scheduled we have used a signal handler which is triggered on SIGUSR1 and generates call to pause(). As all thread functions start with pause(), threads will be suspended right after creation.

In order to start scheduling, following function is called – pthread_kill(pthread_first, SIGUSR2) which resumes thread to be run first. In order to perform scheduling, an additional thread runs an infinite loop while calling function sleep(scheduling_interval) and calls pthread_kill(pthread_current, SIGUSR1) after that to suspend current thread. Then, next thread is resumed and pthread_current is made pthread_next.

6. Algorithms and test data sets

6.1 Algorithms

- FCFS Scheduling Algorithm
  1. Create Threads
  2. Once Thread in excuetion block all other threads
  3. Thread excueted , unblock next thread to execute
  4. Do this until no threads left

- RR Scheduling Algorithm : This will switch threads every 1 sec [2]
  1. Have a signal handler installed for each thread to be scheduled.
  2. This handler is triggered on say SIGUSR1 and internally does nothing more than to invoked a call to pause().
  3. The thread functions all start with a call to pause(), which suspends all threads immediatly after creation.
  4. Create all threads to be schedules using pthread_create(). Stores the pthreads created into an array pthreads.
  5. Assign the first pthread to be run (from pthread) to pthread_first.
  6. To start scheduling call pthread_kill(pthread_first, SIGUSR2) to resume the thread to be run first (by makeing the pause() its blocking on to return).
  7. Make pthread_current become pthread_first.

8. To actually perform scheduling an additional thread (perhaps the main thread) loops infinitly and calls sleep(SCHEDULING_INTERVALL) and then calls pthread_kill(pthread_current, SIGUSR1) to suspend the current thread (by invoking its signal handler and with this running into pause()).
9. Then call pthread_kill(pthread_next, SIGUSR2) to resume the next thread (by makeing the pause() its blocking on to return). Make pthreat_current become pthread_next, and pthread_next become another entry from the array pthread filled during thread creation.

- Parser Algorithm [1]

```
1 Expr Function
2  Until ; symbol not found
3          Call Term Function
4                  Call Factor Function
5                          Reads interger of expression at that point from the given file
and store in val
6                          returns val to Term Function
7                  if no stored previous operator than
8                          Reads operator of expression at that point from the given file
9                  else
10                         load that operator
11                 If operator is Multipy
12                         Call Factor function and get the next integer and multiply them
and store in val
13                 else
14                         store that operator and val remains as it was before
15                 return the integer val
16         if no stored previous operator than
17                         Reads operator of expression at that point from the given file
18         else
19                 load that operator
20         If operator is addition
21                         Call Term function and add the output to val and store it in val
22                 else
23                         store that operator and val remains as it was before
24         return the integer val
```

- Thread method
  1. File name is passed as paramter to thread
  2. File is open and count the number of lines i.e. count the number of expression
  3. Pass each line into the parser according to the scheduling algorithm
  4. Count the finish time and turn around time

6.2 Test Data Sets

There are 3 files name process 1, process 2 and process 3. Based on the c program run (fcfs.c or rr.c) parser takes expressions from these files.

7. Implementation

List of source codes developed, tested and implemented

- fcfs.c
  Contains c program to run First Come First Serve (FCFS) scheduling algorithm.
- rr.c
  Contains c program to run Round Robin (RR) scheduling algorithm.

8. Test Results

- Files containing expressions

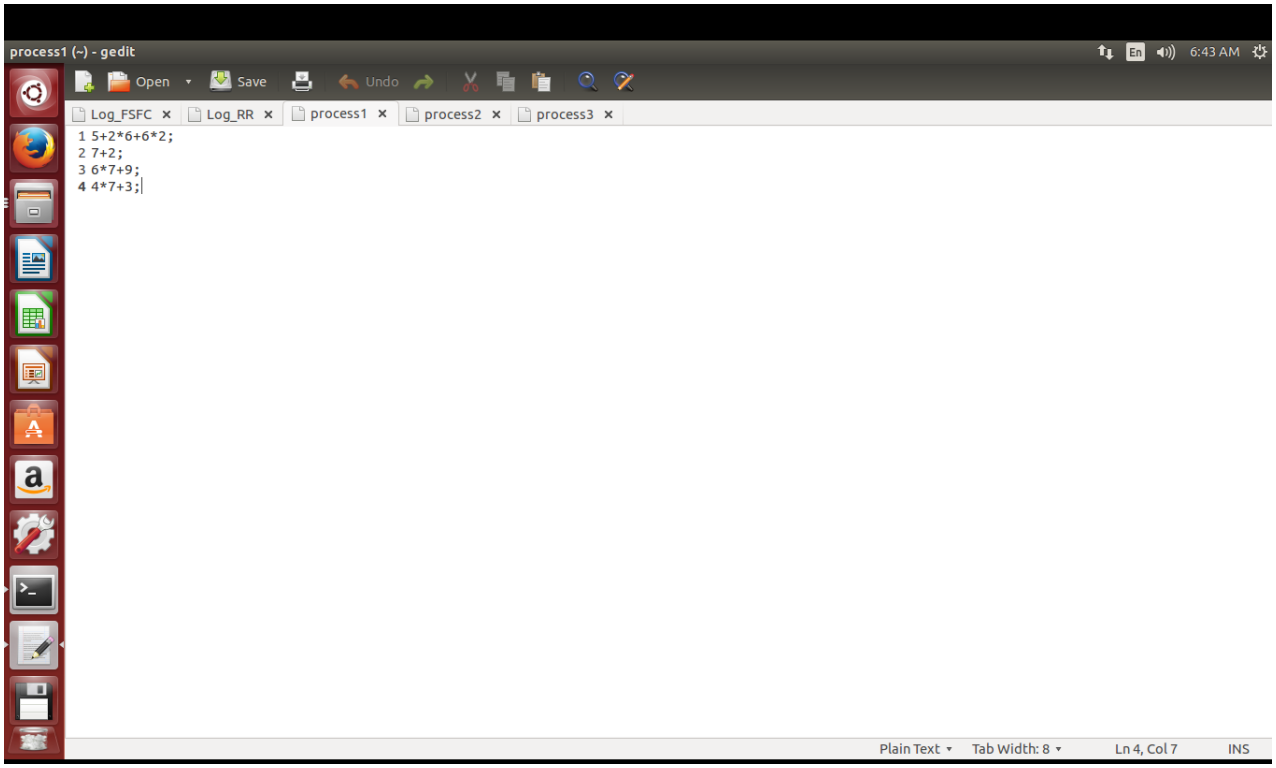


Figure 1 Process 1

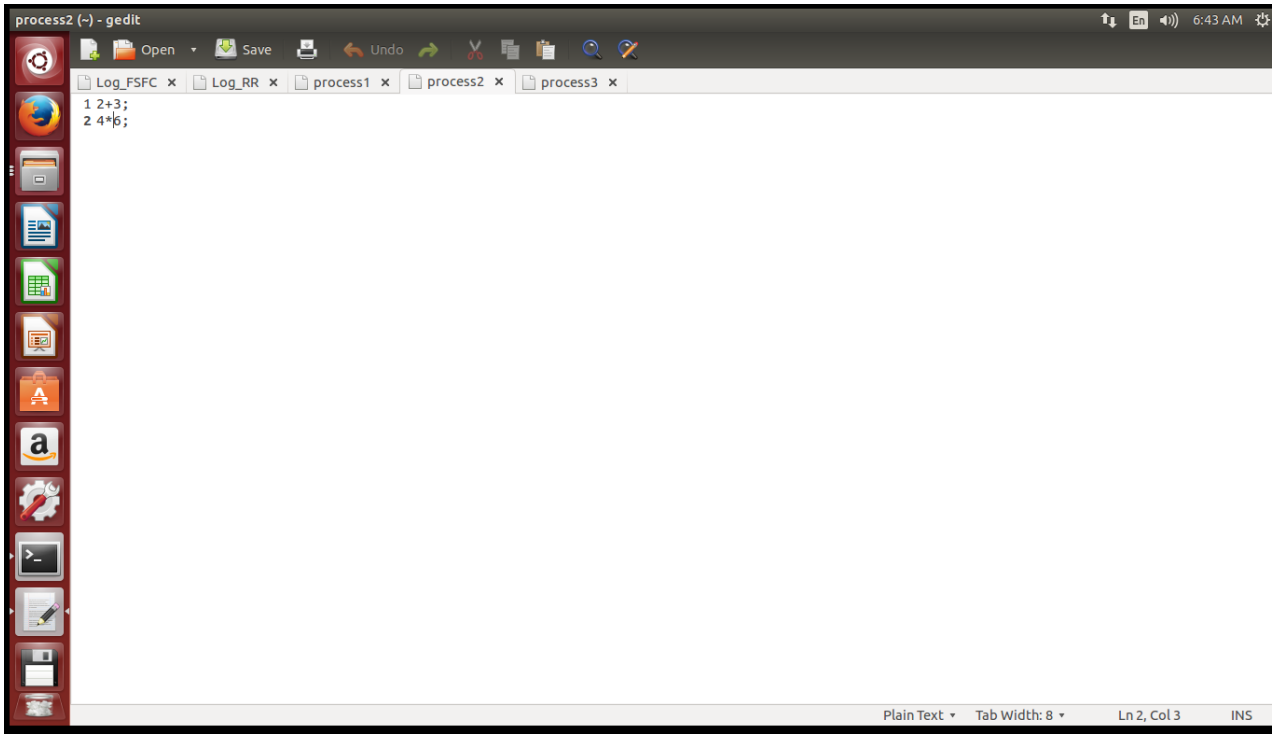


Figure 2 Process 2

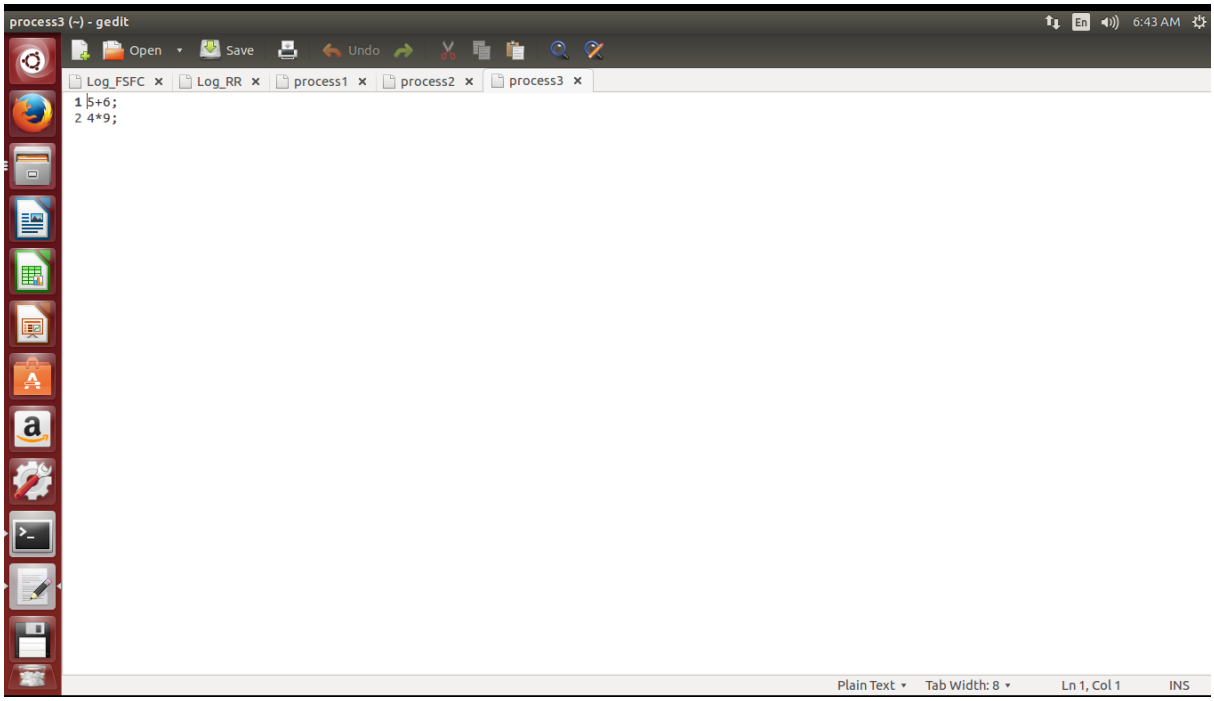


Figure 3 Process 3

- Outputs on terminal

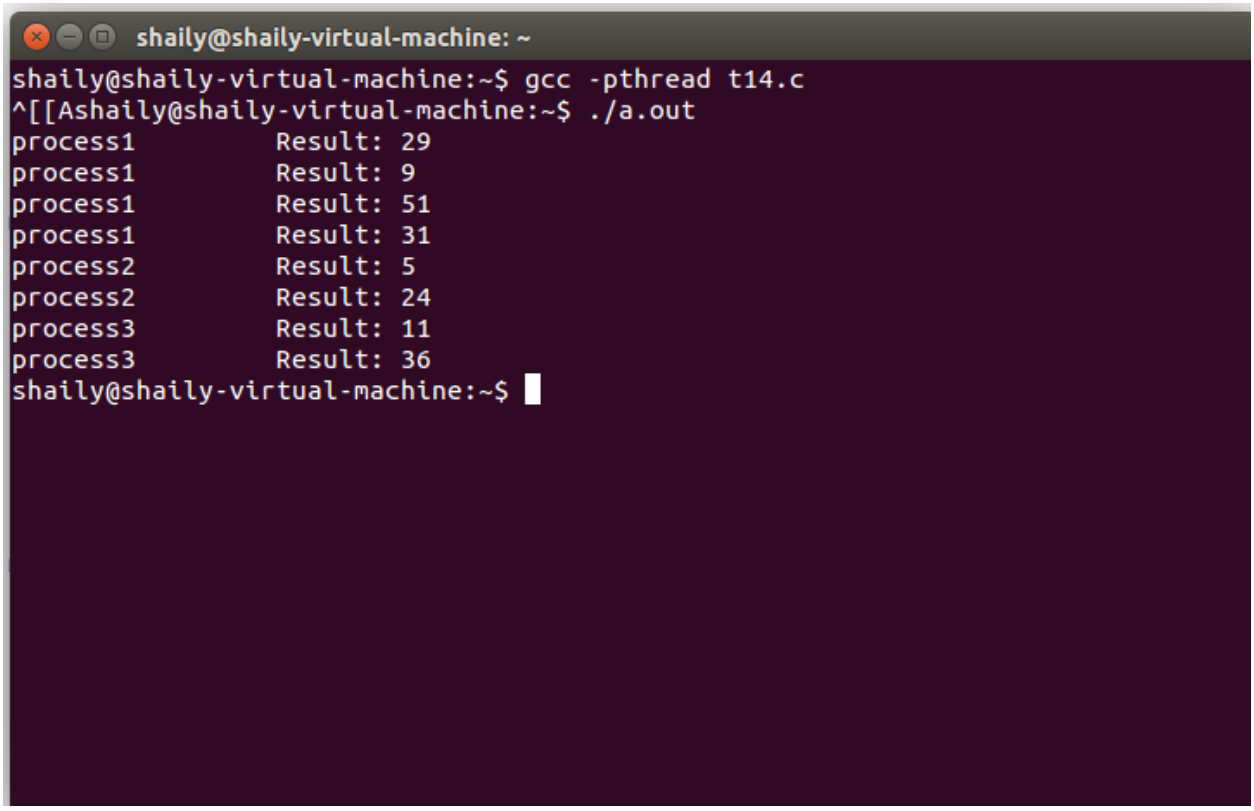


Figure 4 FCFS scheduling

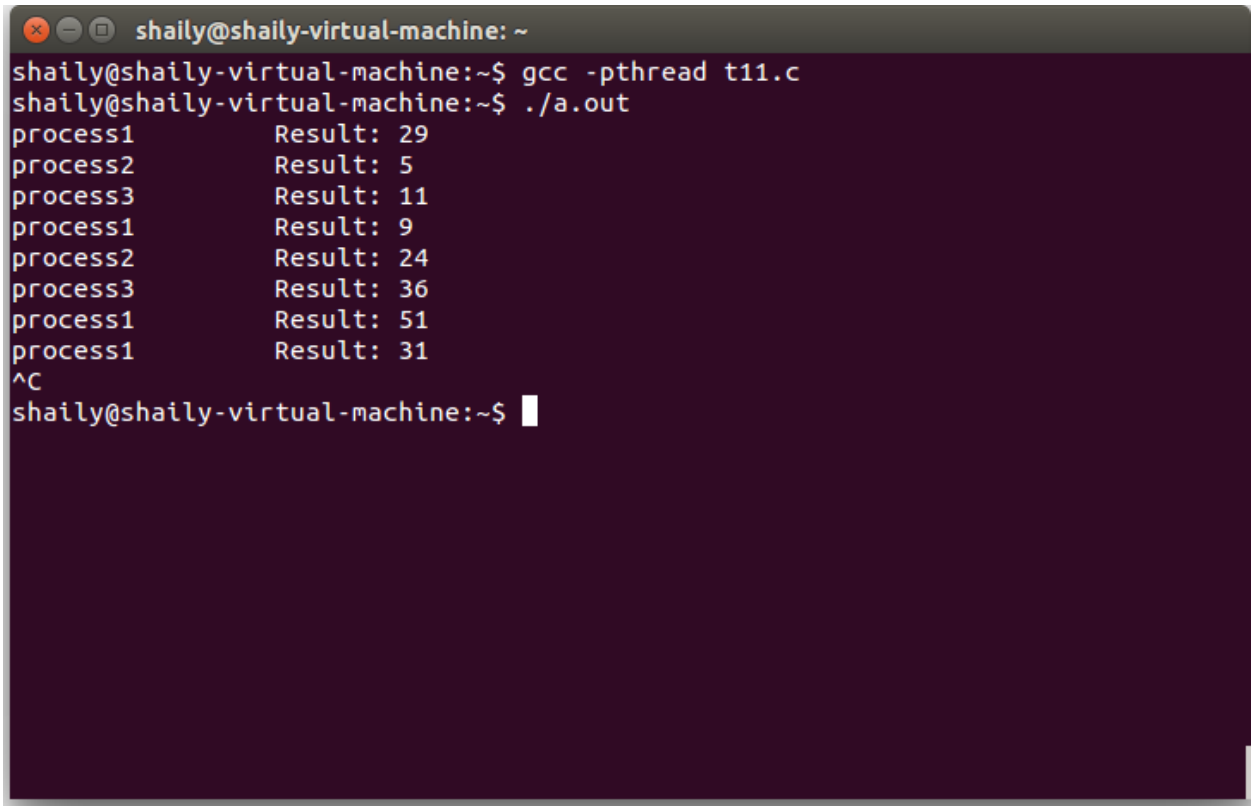


Figure 5 RR Scheduling

- Output in file

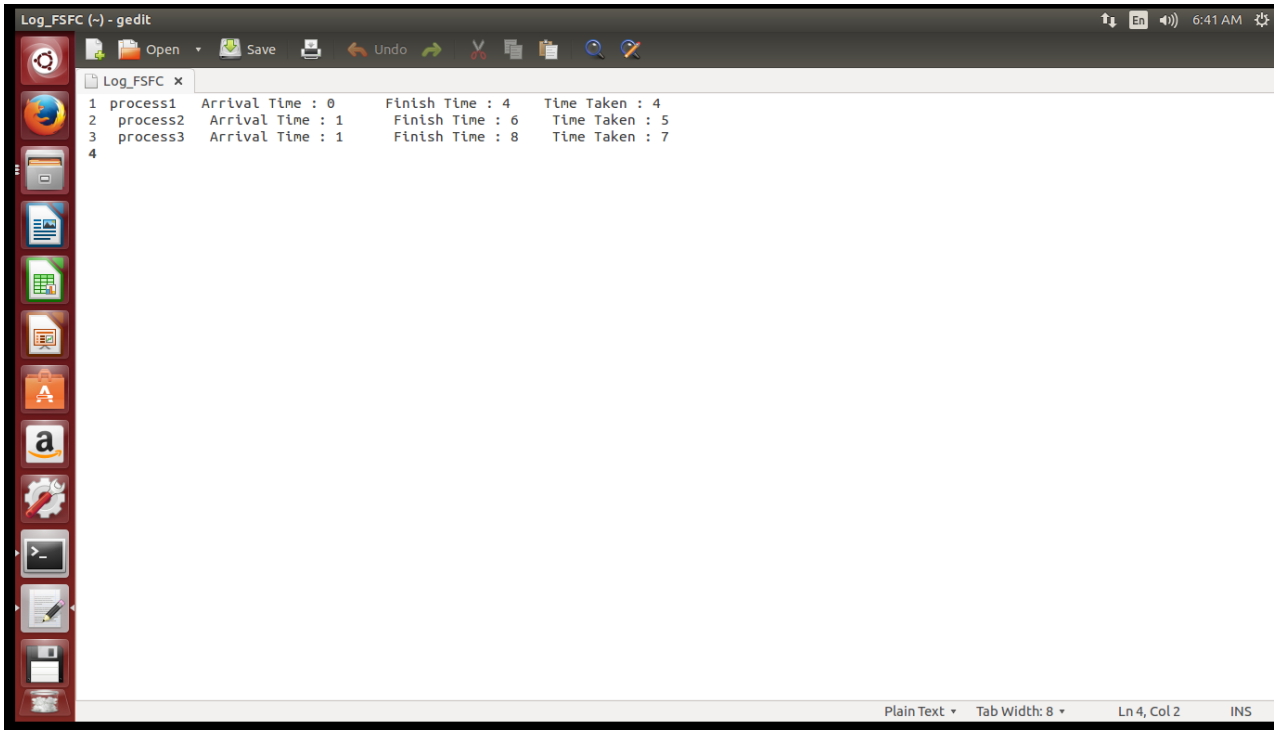


Figure 6 Output logged for FCFS

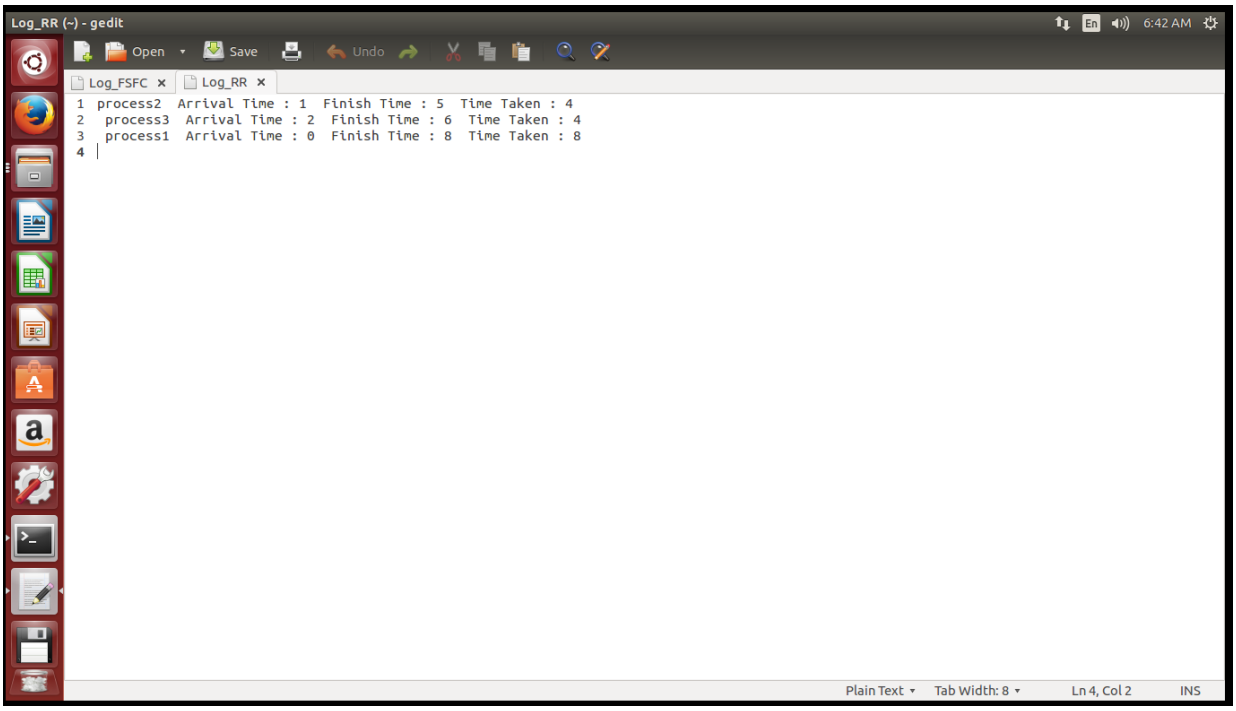


Figure 7 Output logged for RR

9. References

[1] http://2k8618.blogspot.in/2011/03/recursive-predictive-parser.html

[2] https://stackoverflow.com/questions/16368653/how-to-switch-between-posix-threads/16373606#