



AHMEDABAD
UNIVERSITY

Operating Systems Project

Context Switching with CPU Scheduler

Group - 39

Chintan Gandhi (201501019)

Pankti Vadalia (201501068)

Hardik Udeshi (201501113)



Problem Definition

- Implementation of Context Switching, sorting processes based on its arrival time and scheduling according to Round Robin algorithm.
- When the process is switched out, the context is saved and when the process is allocated CPU again, the process is resumed back from its last saved state.
- We have designed a hypothetical processor where we have defined custom instructions and each file is considered as 1 process and 1 thread handles 1 process.

Overview of OS concepts used

- Multithreading

- We have implemented threads where each thread is responsible for handling one process. The thread fetches instructions from the file (process) and executes it till time slice exhausts or process is completed. When the time slice exhausts, the thread is paused and the next thread of next process in ready queue takes over the processor. This way, no processes suffer from starvation.

- Context switching

- We have multiple processes vying for processor time. To not let any process into starvation, we have implemented the Round Robin Scheduling algorithm for scheduling these processes on the processor. When a process is switched out, its variables information and the line no in the file is stored by which the state of the process can be restored the next time it is allocated the processor.



Overview of OS concepts used

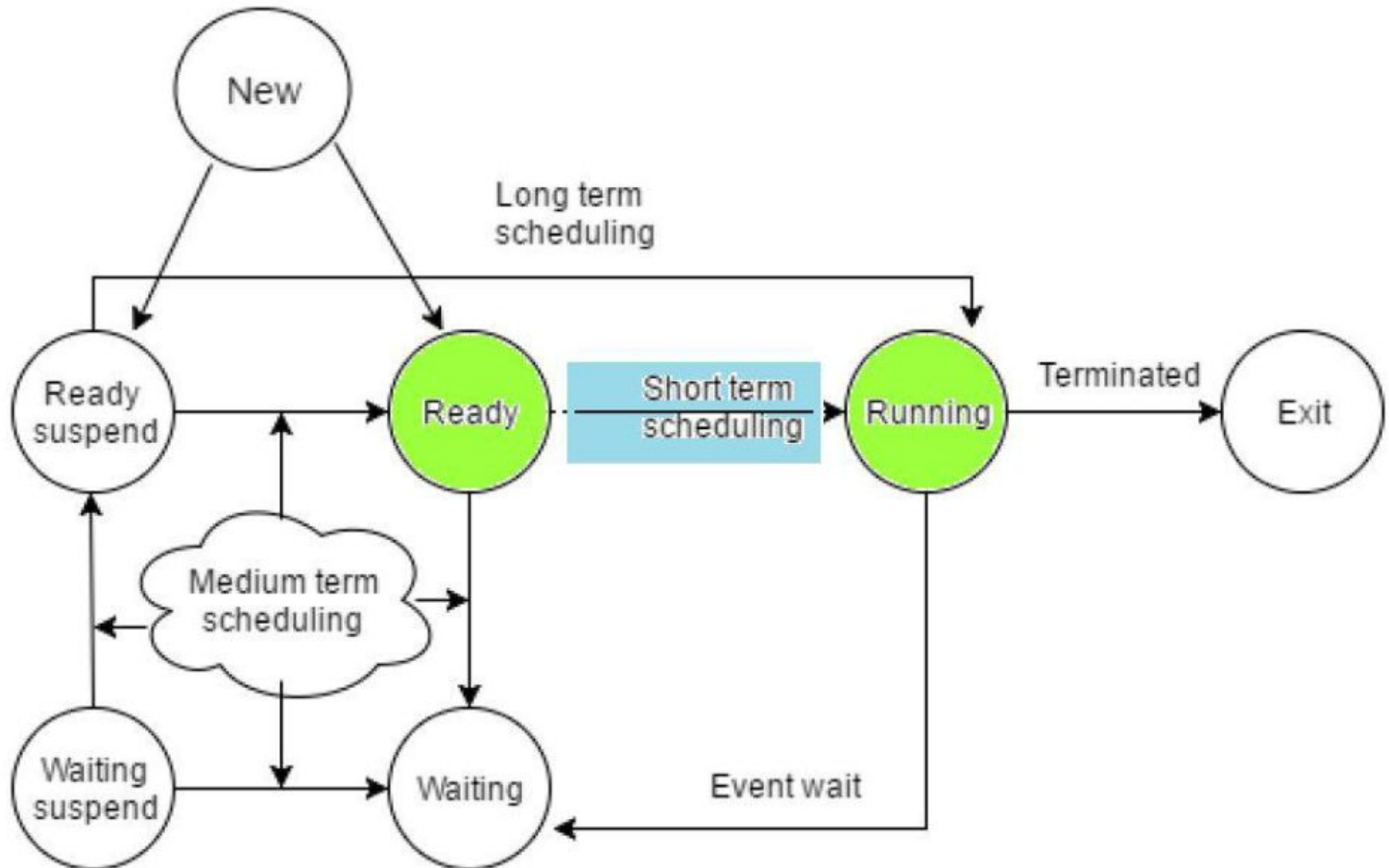
- PCB Management

- Each process in the system has a dedicated Process Control Block (PCB) where its registers information, current state, variables info. is stored. In our case, we store the process id, process name and the no. of instructions executed of the process. For our simulation, we have saved the variables in a separate variables.txt file. We have demonstrated the use of shared variables in our simulation.

- Scheduling

- We have implemented the Short Term scheduler which is responsible for putting the process from the ready queue onto processor and switching the processor time between the processes. There are many scheduling algos. like FCFS, RR, SJF, etc. but in real life, RR is the most widely used algorithm, so we have used Round Robin algorithm for process scheduling.

Model



Project Modules and Files

Main Project modules:

- `launchMain.c`
 - This module provides a menu as to whether the implementation is to be carried out using threads or not. We generate different log files for both the threaded and non-threaded implementation of context switching.
- `mainFunc.c`
 - This module contains the main core logic of the project. This is where the threaded and non-threaded implementation is done.
 - In threaded implementation, 1 thread handles 1 process and it is responsible for reading instructions from the files and executing them.
 - For the non-threaded implementation, we take a process from ready queue and start executing its instructions. When time slice arrives, the process is switched out and next process is resumed from the last saved context.

Project Modules and Files

- pcbInitialize.c
 - This module initializes the parameters of the process control block for each process like process id, process name, arrival time, burst time, priority. The ready queue is also formed here from which the processes are taken one by one and allocated processor time.
- decodeExec.c
 - This module takes an instruction and decodes the instruction by breaking it into different tokens: Opcode_Name Operand1 Operand2 Result_Variable. The operands inputted “Operand1” & “Operand2” values’ are fetched from the “variables.txt” file. In case, direct value is provided in place of operands, that value is typecast into integer and used for further calculations. The opcode for the “Opcode_name” is fetched from the “opcode.txt” file and the operation is performed. The modified values of the variables are stored back in “variables.txt” file.



Project Modules and Files

Files to store opcodes & variables information:

opcode.txt

```
ADD 1  
SUB 2  
MUL 3  
DIV 4  
MOD 5
```

variables.txt

```
a 10  
b 20  
c 30  
d 40  
e 50  
f 60  
g 70  
h 80  
i 90  
j 100  
AddResult 200  
SubResult 300  
MulResult 400  
DivResult 500  
ModResult 600
```


Project Modules and Files

- We have stored the instructions that is to be executed in various files and these files are assumed to be various processes (one file is one process). These files are stored in “processes” directory.
- **Input Instruction Format:**
Opcode_Name Operand1 Operand2 Result_Variable

```
ADD i j sum1  
SUB b a sub2  
MUL c d mul2  
DIV e f div2  
ADD AddResult 1000 MulResult
```

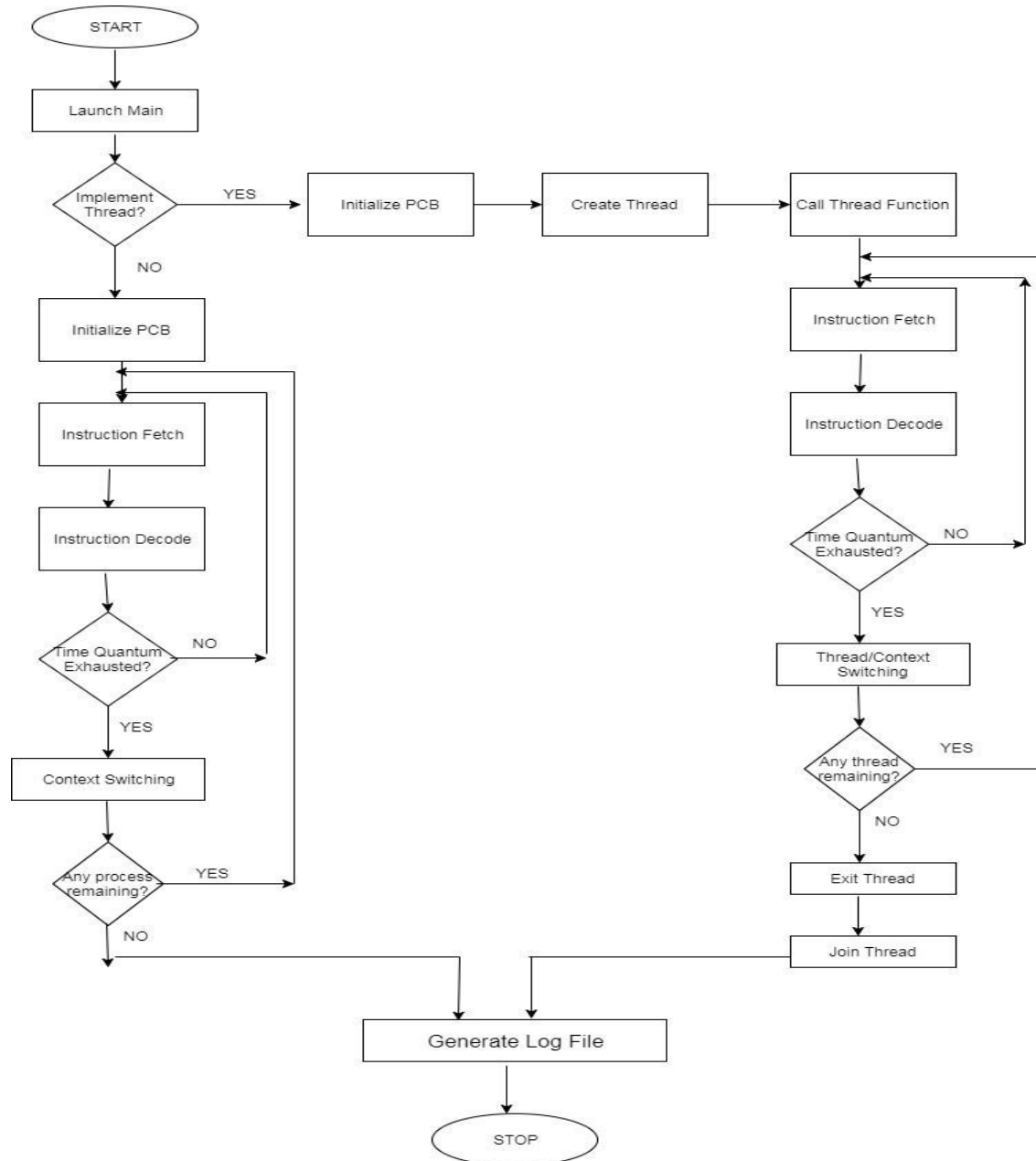
Project Modules and Files

- We have maintained our output in log files.
- There are two log files : logs_rr_without_thread and logs_rr_with_thread.
- In logs_rr_without_thread, output of our “Implementation of Context Switching without threading” is stored.
- While in logs_rr_with_thread, output of our “Implementation of Context Switching with threading” is stored.

Flowchart



AHMEDABAD
UNIVERSITY



Results

“variables.txt” - stores state of variables before & after execution

Before

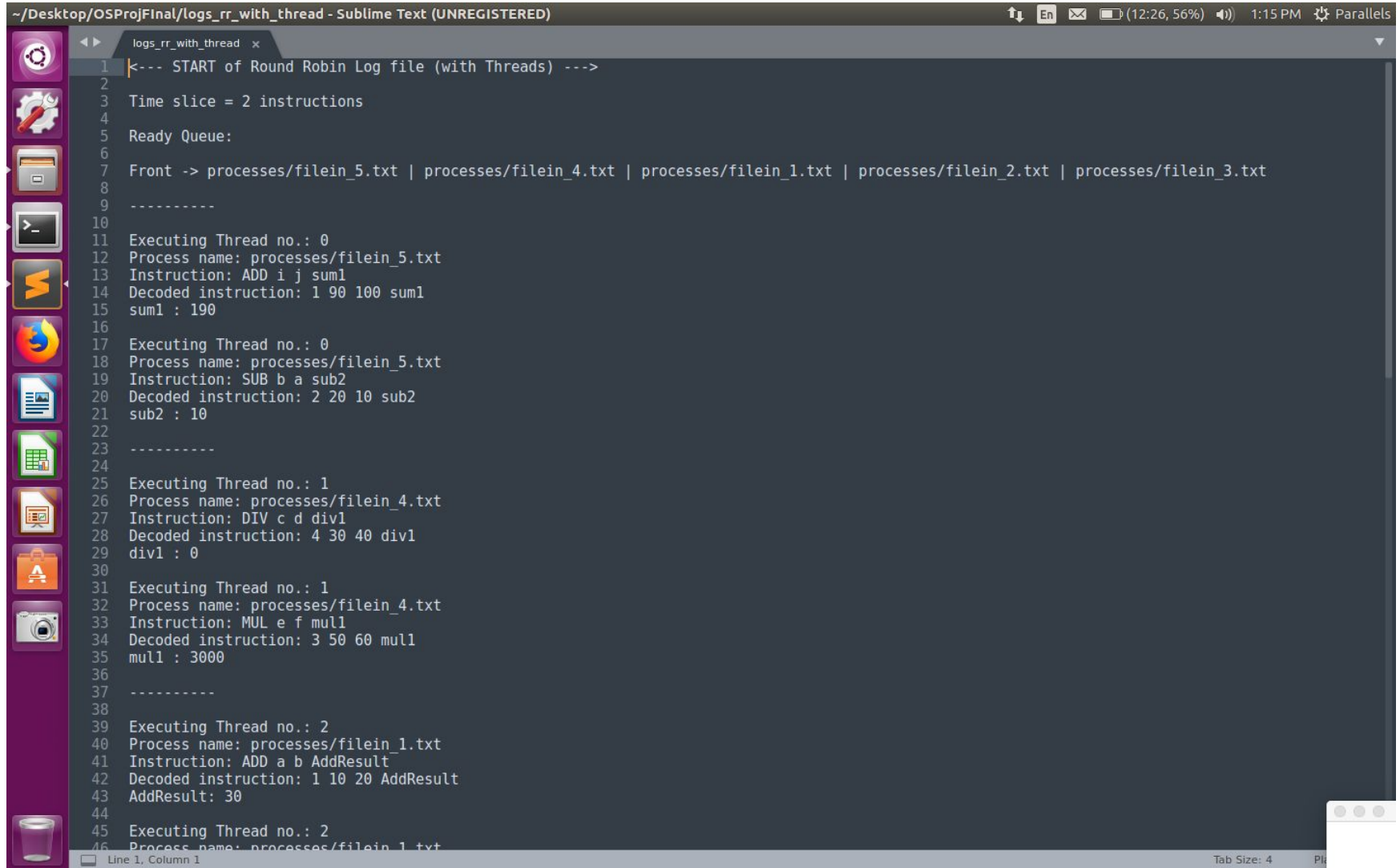
```
a 10
b 20
c 30
d 40
e 50
f 60
g 70
h 80
i 90
j 100
AddResult 200
SubResult 300
MulResult 400
DivResult 500
ModResult 600
```

After

```
a 10
b 20
c 30
d 40
e 50
f 60
g 70
h 80
i 90
j 100
AddResult 30
SubResult -10
MulResult 1030
DivResult 3000
ModResult 90
sum1 190
sub2 10
div1 0
mul1 3000
mul2 1200
div2 0
sub1 -10
mod1 10
```

Results

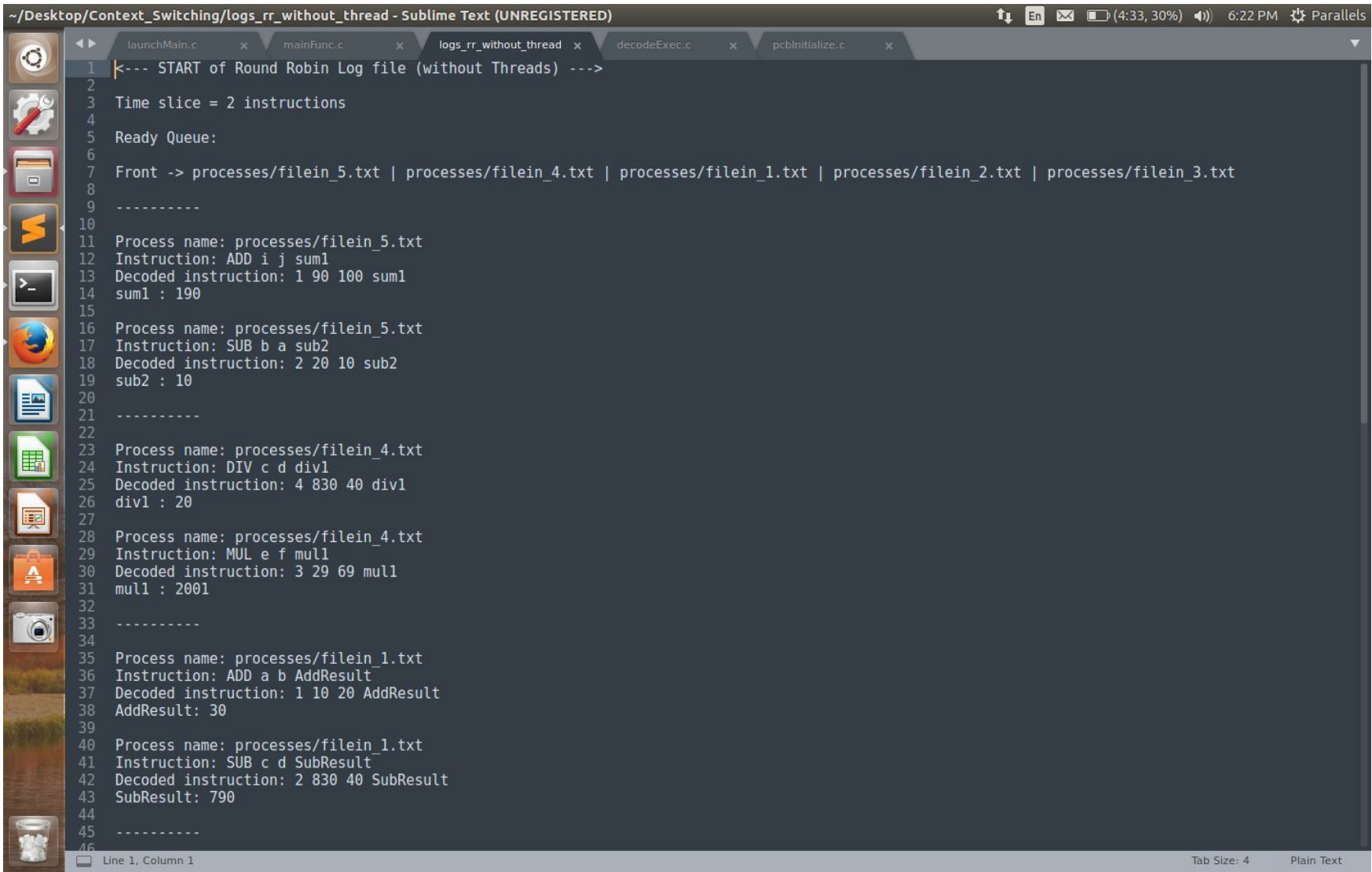
Output stored in file: "logs_rr_with_thread"



```
~/Desktop/OSProjFinal/logs_rr_with_thread - Sublime Text (UNREGISTERED)
1 <--- START of Round Robin Log file (with Threads) --->
2
3 Time slice = 2 instructions
4
5 Ready Queue:
6
7 Front -> processes/filein_5.txt | processes/filein_4.txt | processes/filein_1.txt | processes/filein_2.txt | processes/filein_3.txt
8
9 -----
10
11 Executing Thread no.: 0
12 Process name: processes/filein_5.txt
13 Instruction: ADD i j sum1
14 Decoded instruction: 1 90 100 sum1
15 sum1 : 190
16
17 Executing Thread no.: 0
18 Process name: processes/filein_5.txt
19 Instruction: SUB b a sub2
20 Decoded instruction: 2 20 10 sub2
21 sub2 : 10
22
23 -----
24
25 Executing Thread no.: 1
26 Process name: processes/filein_4.txt
27 Instruction: DIV c d div1
28 Decoded instruction: 4 30 40 div1
29 div1 : 0
30
31 Executing Thread no.: 1
32 Process name: processes/filein_4.txt
33 Instruction: MUL e f mul1
34 Decoded instruction: 3 50 60 mul1
35 mul1 : 3000
36
37 -----
38
39 Executing Thread no.: 2
40 Process name: processes/filein_1.txt
41 Instruction: ADD a b AddResult
42 Decoded instruction: 1 10 20 AddResult
43 AddResult: 30
44
45 Executing Thread no.: 2
46 Process name: processes/filein_1.txt
```

Results

Output stored in file: “logs_rr_without_thread”



```
~/.Desktop/Context_Switching/logs_rr_without_thread - Sublime Text (UNREGISTERED)
1 k--- START of Round Robin Log file (without Threads) --->
2
3 Time slice = 2 instructions
4
5 Ready Queue:
6
7 Front -> processes/filein_5.txt | processes/filein_4.txt | processes/filein_1.txt | processes/filein_2.txt | processes/filein_3.txt
8
9 -----
10
11 Process name: processes/filein_5.txt
12 Instruction: ADD i j sum1
13 Decoded instruction: 1 90 100 sum1
14 sum1 : 190
15
16 Process name: processes/filein_5.txt
17 Instruction: SUB b a sub2
18 Decoded instruction: 2 20 10 sub2
19 sub2 : 10
20
21 -----
22
23 Process name: processes/filein_4.txt
24 Instruction: DIV c d div1
25 Decoded instruction: 4 830 40 div1
26 div1 : 20
27
28 Process name: processes/filein_4.txt
29 Instruction: MUL e f mul1
30 Decoded instruction: 3 29 69 mul1
31 mul1 : 2001
32
33 -----
34
35 Process name: processes/filein_1.txt
36 Instruction: ADD a b AddResult
37 Decoded instruction: 1 10 20 AddResult
38 AddResult: 30
39
40 Process name: processes/filein_1.txt
41 Instruction: SUB c d SubResult
42 Decoded instruction: 2 830 40 SubResult
43 SubResult: 790
44
45 -----
46
```

Conclusions

- Timeslice plays an important role in context switching, because more the timeslice, lesser would be the context switching and vice-versa. As we keep on increasing time slice, we approach towards FCFS kind of algorithm.
- Round Robin (RR) is the best and most suitable scheduling algorithm for context switching.
- There is no need to use First Come First Serve (FCFS) scheduling algorithm for Context Switching. As FCFS, we also don't need to use Priority Scheduling for Context Switching, because in FCFS and Priority Scheduling, whatsoever process is entered in running state is completely executed and once it completes its execution, it is terminated.
- Also, Shortest Remaining Time First (SRTF) and Shortest Job Next (SJN) is of no use as in real life, we don't know the Service time or say, CPU Burst Time of any Process.



THANK YOU

