# **Mholi Mncube**

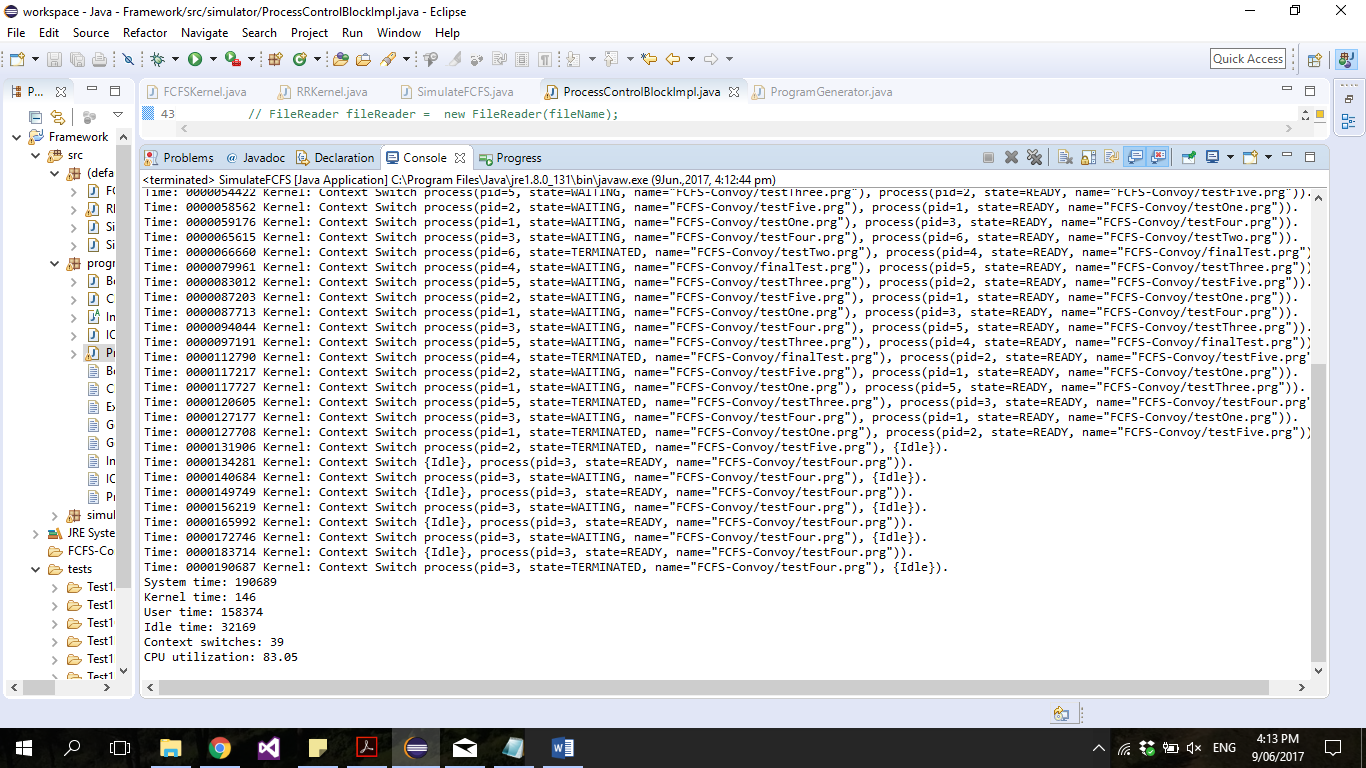
# **Scheduling Short Report on Results**

# **Convoy Effect Demonstration**

The objective of this short report is to show verification as to how the selected workload demonstrates the effect.

In a multiprogramming environment, if multiple processes are waiting for the CPU time for execution in “first come first serve” method. And a slow process is utilizing the CPU keeping the fast process on wait. It will lead to the convoy effect. Unnecessary wait will be done by the fast processes.”

# **FCFS Convoy**



The selected workload was chosen in a way that the processes are IO bound, e.g. have a higher IO burst than that of a CPU burst.

A process with a higher IO burst is loaded and will be loaded into the CPU thus quickly finish its execution on the CPU and wait on the IO queue. Now the CPU-bound process (finalTest.prg) will get hold the CPU and spends lot of time on the CPU. Whilst the CPU bound process is on the CPU, all the other I/O bound processes which came before will finish without interruptions their I/O thus moved into the ready queue, waiting for the CPU to be free. The I/O devices will remain idle while the IO bound processes are still on the ready queue waiting to use the CPU. The CPU-bound process completes its CPU burst and moves to an I/O device and the I/O queue will be empty. The I/O-bound processes will then again quickly complete their execution on the CPU and move back to the I/0 queues and the CPU will be sit idle. The CPU-bound process will again be allocated the CPU and I/O processes end up waiting in the ready. Steps will be iterated until all the instructions have been executed. This illustrates a convoy effect as the I/O bound process must wait long for a big process to complete its execution on the CPU.

**Round-Robin Rule of Thumb**

A test was done that has a well distributed load between IO and CPU-bound processes as seen below:

TEST FILE:

Test1/config.cfg

# RR Thumb Test config file

DEVICE 1 disk

DEVICE 2 CD/ROM

PROGRAM 0 0 Test1/programA.prg

PROGRAM 0 0 Test1/programB.prg

Test1/programA.prg

# Program A for RR Thumb Test

# Balanced workload mix of IO and CPU bound programs

CPU 1000

IO 500 2

CPU 500

IO 2000 1

CPU 1000

Test1/programB.prg

# Program B for RR Thumb Test

# Balanced workload mix of IO and CPU bound programs

CPU 500

IO 1000 1

CPU 1000

IO 2000 2

CPU 150

The test ran in 200ms decrements from 2000 to 200. 2000 was chosen as the maximum as it is the maximum IO/CPU burst time given in the program files. When it is run with a time slice of 2000 it is effectively a FCFS implementation.

Experiment Findings:

An analysis of the findings shows that the longer time slices produced far less variation in CPU utilization, System Time etc.. This is due to the amount that each process time done per time slice is not changing (thus the number of context switches is constant for them).

Shorter time slices produce more variation in System Time and Kernel Time, this is due to the actual amount of switches done has increased substantially, so majority of the processing time is spent switching between processes rather than executing .

An initial deduction would be that the ideal slice time would thus be somewhere in between, such that the context switch overhead does not outweigh the benefits of switching between processes.

All of the results seem to show to best improvement in performance vs number of context switches in the range from 400-600. However the number of context switches required between these values is still relatively high (20 – 15). Therefore I would suggest that the ideal time slice is between 600 and 800.