OPERATING SYSTEMS

# DOCUMENTATION

Assignment - 2

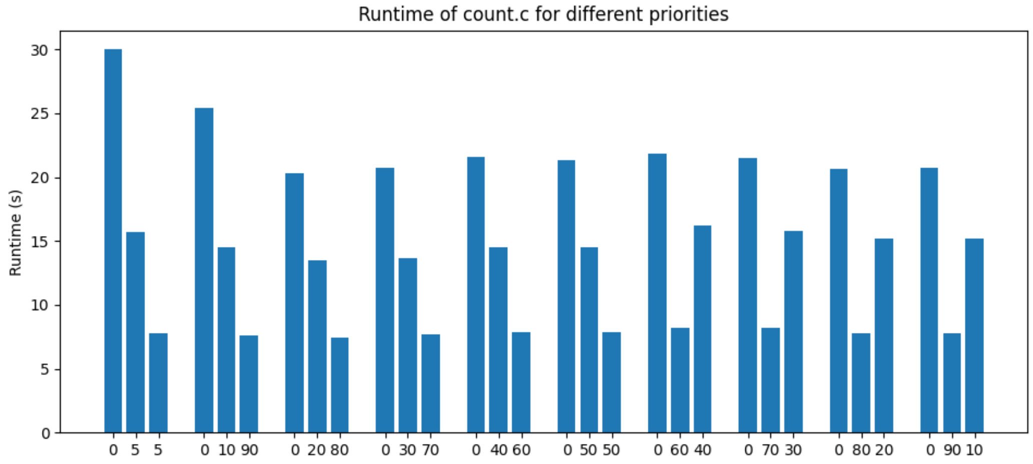
## Introduction

This is the documentation for Assignment - 2. The uploaded folder has 2 folders for question 1 and 2 with all the required programs enclosed inside it.

## Question 1

### Part 1

The file count.c has all the required code to run the program. I have also included a run_count.sh file to run the program 6 times with different priority values as was required. Thus, just calling the bash file should be enough. Also, the code has been executed on a single core to obtain consistent results. Then the python file ‘count_plot.py’ needs to be called to create the graph. Here is an example graph:



The first bar in every set represents the thread with SCHED_OTHER, the second represents SCHED_RR and the last represents SCHED_FIFO.

When the priorities for both RR and FIFO are the same, FIFO finishes earlier as once the process is called, it only stops executing when it's finished. Then we start with higher priority for FIFO, and thus observe lower runtimes. As we decrease the priority for FIFO and increase for RR, the runtimes for RR reduce and increase for FIFO. The trend continues for the remaining runs.

### count.c

This c program has the 3 functions to count from 1 to $2^{32}$ , namely countA, countB and countC. The priorities can be passed as Command line arguments while calling the program.

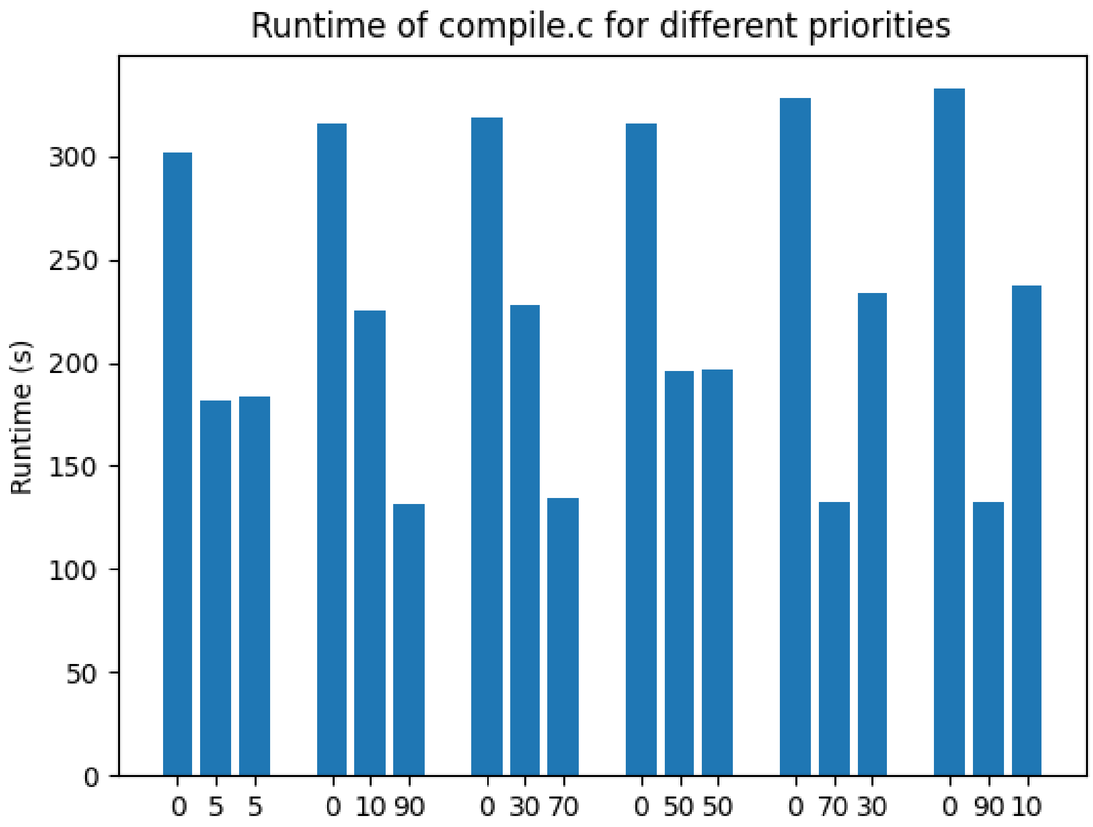
I am using pthread_setschedparam to set the scheduler and priority values for each thread.

Further I am using time.h library function clock_gettime() to time the threads, with real clock as its first argument. The individual values are then stored in an array, from where I calculate the runtime use difference between end and begin times.

The resulting runtimes are then stored in count_time.txt from where they are plotted by the python file.

## Part 2

The file compile.c has all the required code, and the structure is mostly the same. The files compileA.sh, compileB.sh and compileC.sh have the commands to compile the kernel. Also the files cleanA.sh, cleanB.sh and cleanC.sh has commands to remove the executable files from the compiled kernel to compile again, but has was instructed by Dr. Sambuddho Chakravarty, I am no longer using these files in compile.c. The final graph generated is as follows by compile_plot.py:



The first bar in every set represents the thread with SCHED_OTHER, the second represents SCHED_RR and the last represents SCHED_FIFO.

As we can observe in the graph, when the priority for RR and FIFO are the same, the runtimes are virtually identical. Then, I increase the priority of FIFO, and the runtime decreases. As the priorities are then slowly flipped, runtimes also show a similar behavior.

## compile.c

The program uses fork to create child-processes which in turn call the bash scripts.

I am using sched_setscheduler to set the scheduler and priority values of the processes. I am using wait() to return the process ID of the process terminated and store its end time. The runtimes are calculated similarly as above and are stored in the compile_time.txt file.

The bash scripts also use 3 directories kernel_A, kernel_B and kernel_C, where they compile the kernels. I am not attaching them here due to their large size.

## Question - 2

I have compiled the kernel with appropriate changes and have successfully added the syscall kernel_2d_memcpy. The diff file attached shows all the changes made by me. Instead of running diff on the whole kernel, I am running it on the essential files only, as there were many other changes made while compiling the code.

I am also attaching a C program, test_syscall.c to test if the syscall is working properly. For a successful compilation, it should output "LHS = RHS" on the terminal. If the copy made is not correct, it will output  "LHS != RHS". A test array is hard coded in the file, and can be appropriately changed as required.