Lab 2

Multi-Threading, Real Time Tasks, and Simple Synchronization

WEEK-1

Objective

The goals are to:

- Learn how to create multiple threads within a process.
- Appreciate the computational advantage and potential disadvantages of multi-threading.

Prelab

Before coming to the lab, you should list and describe the functions to: *create*, *exit* and *join* a pthread. Include information like the function name, arguments and return values.

Lab Procedure: Threaded Program

1. This first week of lab 2 you will implement a 1D-convolution procedure. You will do that using different number of **pthreads**. Given a text file with a matrix and a 1D-filter of size 1x3, you will need to convolve the filter with the matrix. You can find a few example files named "NxM.txt" on Canvas, under *Modules* > *Additional Lab Files* > *ece4220_lab2.zip*.

   Each of those example files is formatted as follows: the first line contains the number of rows and number of columns of the matrix. The next few lines (equal to row count) contain the actual matrix data, where each number in a row is separated by a space. After that a new line with rows and column number holds the filter information. The last line will be containing the filter data. Consider the following example

```
2 5                 // number of rows and columns of matrix
1  2  3  4  5
11 12 13 14 15
1 3                  // number of rows and columns of filter
1 2 1
```

   Be sure to **check for errors** when accessing the files, such as opening, closing and reading. You could find a short video tutorial on how to convolve nx1 matrix with a 3x1 filter on this youtube link. For the above example, you should get the following results.

```
4   8   12  16  14
34  48  52  56  44
```

2. After the convolution has been completed over entire matrix, your program should display:

   - The results of the convolution
   - Execution time
   - Total number of convolution operations performed

   Perform the convolution with different number of threads. You should try the cases shown in the table-1 (the convolution results and the total number of convolutions operations should be the same, but the execution times may not be the same). For each case, you should run your program multiple times (at least 10) and calculate average convolution times. Fill in the observation table-1.

Case	**Avg. Conv. Time (2x10.txt)**	**Avg. Conv Time (20x10.txt)**
One thread to convolve the filter with the **entire matrix**		
One thread for convolving the filter with **each row** of the matrix		
One thread for convolving **each element** of the matrix		

Table 1: Observation Table

   What is the best number of pthreads for doing the convolution, according to your results? Why do you think that was the case? Answer the questions in your report.

3. Since you are using **pthreads**, first you have to include the header file pthread.h and then you need to add the parameter *pthread* to your linker options when you build your project. On the terminal you could run:

```
gcc Lab2_Part1.c -o Lab2_Part1 -lpthread
```

   To do this in Eclipse, go to the *Properties*, and under where it says *"GCC C Linker"*, click on *Libraries*. On the right, this will bring up places to add libraries to link to, and their paths (if needed). Simply click on the add button under the *libraries* (denoted *–l*) and type in *pthread*. The pthread library is in a known location so we do not need to add a path of where to find the library.

WEEK-2

Objective

The goals are to:

- Learn how to initialize threads as Real Time tasks.
- Notice the advantages/disadvantages of synchronizing task/threads relying on time events alone.

Prelab

Before coming to lab on the second week of lab 2, check the file *"RT_tasks_functions.c"* in *ece4220_lab2.zip* (on *Canvas*). Investigate the different functions and structures shown in the file. Also add a brief description about the following functions along with their input arguments and return values:

- sched_setscheduler()
- clock_gettime()
- clock_nanosleep()

Lab Procedure: Real Time Tasks in User Space

1. In this part of the lab there are two files that contain strings that need to be collated together into one file. An original text was divided into two parts. The odd lines of the text were saved into the file named **first.txt**, and the even lined were saved into the file **second.txt**. Your task is to reconstruct and display the original text. To achieve this you will develop a program that creates three pthreads, and a common buffer (a character array).
2. Two of the pthreads will be used as file readers that read in one line of their file at a time (the first pthread opens and reads first.txt, the second opens and reads second.txt), and save the read line to the common buffer. The third pthread will be used to take information from the common buffer, and store it in a string array that can hold up to 20 strings. Finally, the reconstructed text will be printed in the main thread.

3. After the files have been opened, all pthreads should initialize themselves as real time tasks, and schedule themselves at alternating times so that data can be read in from the files and stored in the array in the correct sequence. You will need to synchronize the tasks by adjusting their periods and starting times. Make sure to try different combinations. You may want to start with relatively long periods, to ensure that the reading and saving into the buffers is completed on time. Then, try reducing the periods. **Can you find the minimum period(s) that still allows you to reconstruct the original file properly?**

4. Remember to you include the appropriate headers in your program (check the *"RT_tasks_functions.c"* file). Since you are creating pthreads, the corresponding header and library need to be included.

5. Make sure you use sudo while running your executable.

6. Add time.h to access all timer related functions and structures. Also, as you are going to use *rt-* functions, you need to link your executable to **rt** library. So, while compiling you need run:

```
gcc Lab2_Part2.c -o Lab2_Part2 -lpthread -lrt
```

7. Make sure to report the experiments that you ran, the times that you used, etc.

References

- Multithreaded Programming
- POSIX Threads Programming
- HOWTO build a simple RT application
- HOWTO build a basic cyclic application