



# SOFE 3950U / CSCI 3020U: Operating Systems

## TUTORIAL #5: POSIX Threads

### Objectives

- Learn the fundamentals of multithreading
- Gain experience using POSIX threads

### Important Notes

- Work in groups of **four** students
- All reports must be submitted as a PDF on blackboard, if source code is included submit everything as an archive (e.g. zip, tar.gz)
- Save the file as <tutorial\_number>\_<first student's id>.pdf (e.g. tutorial5\_100123456.pdf)
- If you cannot submit the document on blackboard then please contact the TA with your submission at [jonathan.gillett@uoit.net](mailto:jonathan.gillett@uoit.net)

## Notice

It is recommended for this lab activity and others that you save/bookmark the following resources as they are very useful for C programming.

- <http://en.cppreference.com/w/c>
- <http://www.cplusplus.com/reference/clibrary/>
- <http://users.ece.utexas.edu/~adnan/c-refcard.pdf>
- <http://gribblelab.org/CBootcamp>

The following resources are helpful as you will need to use pthreads in order to make your program multithreaded.

- <https://computing.llnl.gov/tutorials/pthreads/>
- <http://randu.org/tutorials/threads/>
- [http://pages.cs.wisc.edu/~travitch/pthreads\\_primer.html](http://pages.cs.wisc.edu/~travitch/pthreads_primer.html)

## Conceptual Questions

1. Read the pthread documentation and explain the following three functions: **pthread\_create**, **pthread\_join**, **pthread\_exit**.
2. Explain how the memory of threads work in comparison to processes, do threads share the same memory, can threads access the memory of other threads?
3. Name the differences between **multithreading** and **multiprocessing** (multiple processes). What are the advantages and disadvantages of each?
4. Provide an explanation of **mutual exclusion**, what is a **critical section**?
5. Research the functions used to perform **mutual exclusion** with pthreads and explain the purpose of each function.

## Application Questions

All of your programs for this activity can be completed using the template provided, where you fill in the remaining content. A makefile is not necessary, to compile your

programs use the following command in the terminal. **If you do not have clang then replace clang with gcc, if you use gcc you must use -pthread instead of -lpthread.**

```
clang -Wall -Wextra -std=c99 -lpthread <program name>.c -o  
<program name>
```

### Example:

```
clang -Wall -Wextra -std=c99 -lpthread question1.c -o question1
```

You can then execute and test your program by running it with the following command.

```
./<program name>
```

### Example:

```
./question1
```

### Template

```
#include <stdlib.h>  
#include <stdio.h>  
#include <pthread.h>
```

```
int main(void)  
{  
  
}
```

1. Create a program that does the following, make sure you can complete this before moving to further questions, when compiling add the **-lpthread** argument, if you are using **gcc** use the **-pthread** argument.
  - Creates two threads, the first uses a function **hello\_world()** which prints *hello world*, the second uses a function **goodbye()** which prints *goodbye*.
  - Each function has a random sleep duration before printing the output
  - After running your program a few times you should notice that the order of *hello world* and *goodbye* being printed to the screen is not consistent, as

each thread is executing independently.

2. Create a program that does the following.

- Prompts the professor for **five** student's grades.
- Creates 5 threads, one for each student.
- Each thread uses a function called **bellcurve(grade)** which takes as an argument the grade and bellcurves it by multiplying the grade by **1.50** and then **printing** the bellcurved grade to the terminal.
- The program **must** create the 5 threads and initialize them only after receiving all 5 grades.

3. Create a program that does the following.

- Prompts the professor for **five** student's names, student\_id, and grade.
- Creates **five** threads, one for each student.
- Create a struct named **student** containing three members, **name**, **student\_id**, and **grade**.
- Create a function **bellcurve(student)** which takes a student (the struct type) as an argument and bellcurves the grades by multiplying it by **1.50** and prints the student name, id, and bellcurved grade to the terminal.
- The program **must** create the 5 threads and initialize them only after receiving all 5 grades.

4. Create a program that does the following.

- Prompts the professor for **ten** student's grades,
- Creates **ten** threads, one for each student.
- Create a function **class\_total(grade)** which adds the grade to a **global variable total\_grade** using the operator **+=** to increment **total\_grade**
- You **MUST** use mutual exclusion when incrementing total\_grade
- Print the results of total grade, it should be the correct sum of all ten grades.

5. Create a program that does the following.

- Reads in 10 grades from the file **grades.txt** using one thread with the function called **read\_grades()**
- You must use a **barrier** to wait for grades to be read by the program
- **Create 10** threads, each uses the function **save\_bellcurve(grade)** which
  - Adds the grade to a **global variable total\_grade** using the operator **+=** to increment **total\_grade**

- Bellcurves the grades by multiplying it by **1.50** and adds the grade to a **global variable total\_bellcurve**
- Saves (**appends**) the bellcurved grade to the file **bellcurve.txt**
- After saving all the bellcurved grades to the file, the main program then prints to the terminal the total grade and the class average **before and after the bellcurve**.
- You will need to use a combination of barriers, mutual exclusion, and thread joining to complete this question.