SOFE 3950U
Operating Systems

Tutorial 4

Group: A8

Date: March 6th, 2022

First Name	Last Name	Student Number
David	Fung	100767734
Anish	Patel	100751489
Raphaiel	Halim	100700318

Conceptual Questions

1. **Read the pthread documentation and explain the following three functions: pthread_create, pthread_join, pthread_exit.**
    - pthread_create: function used to start a new thread
    - Pthread_join: function that joins a specified thread when it terminates
    - pthread_exit: function that is used to terminate the calling thread

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?**

   Processes are mostly isolated, meaning they do not share memory and are also unable to share data with other processes. However, threads share memory and can access the memory of other threads.

3. **Name the differences between multithreading and multiprocessing (multiple processes). What are the advantages and disadvantages of each?**

   Multithreading is an execution technique that allows a single process to execute multiple threads/code sections concurrently. Whereas multiprocessing requires the system to have more than two processors.

   Some of the advantages and disadvantages of having multiprocessing are:

   - Allows to get more work done over a short period (advantage)
   - Takes advantage of multiple cores/cpus (advantage)
   - Takes up a larger memory footprint (disadvantage)

   Some of the advantages and disadvantages of multithreading are:

   - Threads sharing the same address space (advantage)
   - Threads being lightweight resulting in a smaller memory footprint (advantage)
   - Code is usually harder to understand compared to multiprocessing (disadvantage)

4. **Provide an explanation of mutual exclusion, what is a critical section?**

   A critical section refers to the segment of code where processes can access shared resources. Mutual exclusion (mutex) is a program object that prevents simultaneous access to a shared resource. It is a property of concurrent access control where one thread of execution never enters a critical section while another thread of execution is already accessing the critical section. This is significant in concurrent programming because if one thread attempts to change the value of a variable that another thread is reading, it can have unpredictable results.

5. **Research the functions used to perform mutual exclusion with pthreads and explain the purpose of each function.**

   - pthread_mutex_init(): initializes a mutex
   - pthread_mutex_lock(): mutex object referenced becomes locked
   - pthread_mutex_unlock(): releases the mutex object referenced
   - pthread_mutex_destroy(): destroys an initialized mutex (should be unlocked, otherwise may have unpredictable behavior)

Application Questions

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 the 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.