

University of Ontario Institute of Technology (UOIT)

Faculty of Engineering and Applied Science (FEAS)

Department Of Electrical, Computer, And Software Engineering (ECSE)

**Operating Systems (SOFE 3950)**

Tutorial #4 Activity | Submission

Yong Deng

Winter 2020

Group Members

Aryan Kukreja (100651838)

Emil Ilnicki (100659072)

Sydney Smith (100654205)

Harasees Singh Gill (100656810)

# Conceptual Questions

## Question 1

**pThread\_create()**: Used to create a new thread, with attributes defined by attr. When created pThread\_create() stores the ID of the created thread in the location referenced by the thread. If successful the pThread\_create() function returns zero. If unsuccessful an error number is returned to indicate the error (EAGAIN, EINVAL, EPERM).

**pThread\_join()**: Suspends execution of the calling thread until the target thread terminates, unless the target thread is already terminated. If successful pThread\_join() returns zero, else it returns an error number to indicate what error occurred (EDEADLK)

**pThread\_exit()**: This function terminates the calling thread and makes the value ptr available to any join with the terminated thread. If successful the process exits with a status of 0.

## Question 2

Threads run in a shared memory space while processes run in separate memory spaces. Threads share all segments except for their stack. These call stacks are independent between threads. However, the memory in other thread stacks are accessible and you could have a pointer pointing to some threads stack frame.

## Question 3

**Multithreading** involves creating multiple threads of a single process on the same CPU running concurrently to increase computation speeds. On the other hand, **multiprocessing** uses multiple different CPU cores, running concurrently and connected by a single bus, to execute that program.

Multithreading and multiprocessing are both different techniques for improving the performance of a system; however, they both do it in different ways:

**Multithreading**:

* Multithreading does not add more processors or cores to improve the performance of executing a program; rather, it creates multiple threads of a single process that execute concurrently to improve the running speed of the program.
* It is more economical; a processor’s usage is maximized to improve speed, rather than just adding more processors. This saves cost and is more time-efficient as well (since startup of multiple processors will take longer).
* However, multithreading *cannot be classified as symmetric or asymmetric.*

**Multiprocessing**:

* Multiprocessing simply adds more processor cores to the execution of a program; adding more CPUs will increase the performance of the program’s execution.
* Adding more CPUs is more resource-heavy; this adds to the cost, and the communication between different CPUs takes longer than multithreading
* However, multiprocessing *can be symmetric or asymmetric*

## Question 4

**Mutual Exclusion** is an aspect of concurrency control that dictates how parallel threads run concurrently. Typically, when parallel-running threads need to access common memory, it could create conflicts that result in one of the threads processing incorrect memory values since another thread accessed it earlier and disrupted its contents.

The **critical section** is the official term given to the shared region of memory that multiple threads access, and needs to be maintained properly throughout all threads’ execution so that no thread modifies a value in memory that another thread will need in the future.

## Question 5

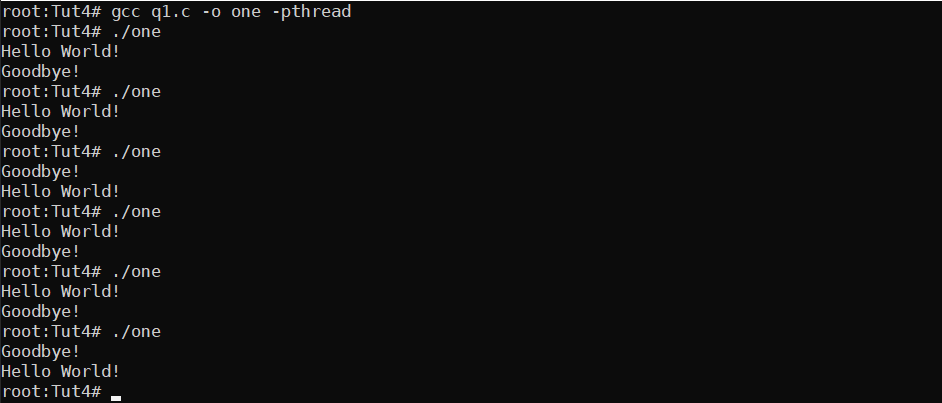
The functions used to perform **mutual exclusion** with pthreads are mutexes. Mutexes block access to variables from other operating threads making the set of variables exclusive to the specific thread.

The most common use of mutex is the lock function, which locks the variables to the current thread. To unlock it there is the unlock function, releasing the other threads access to variables.

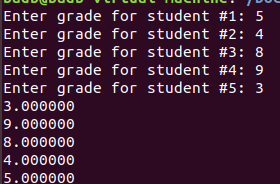
# Application Questions

## Question 1

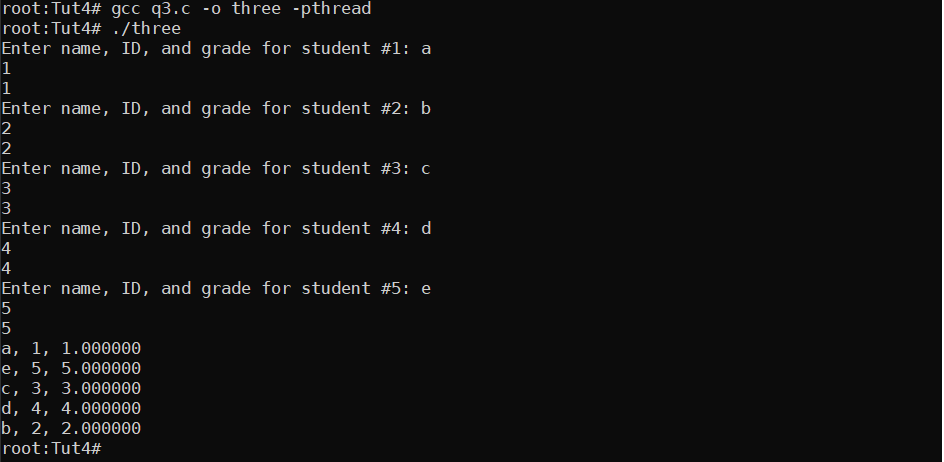




## Question 2

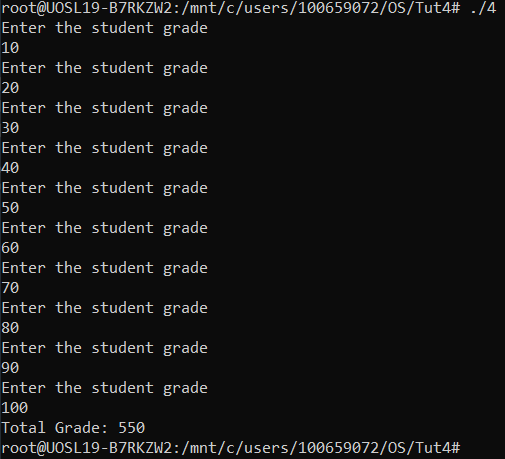


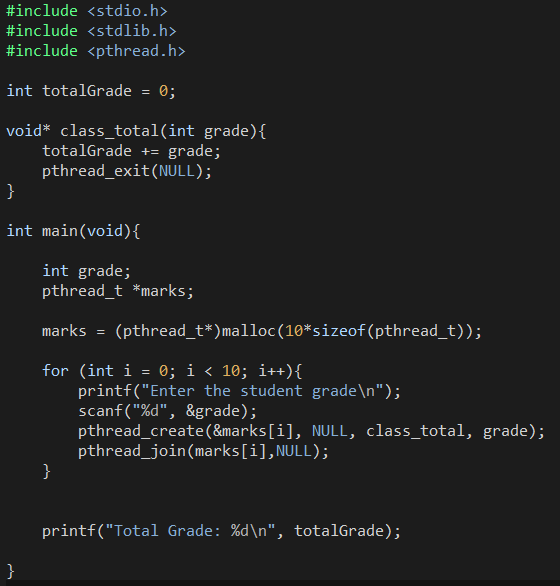
## Question 3



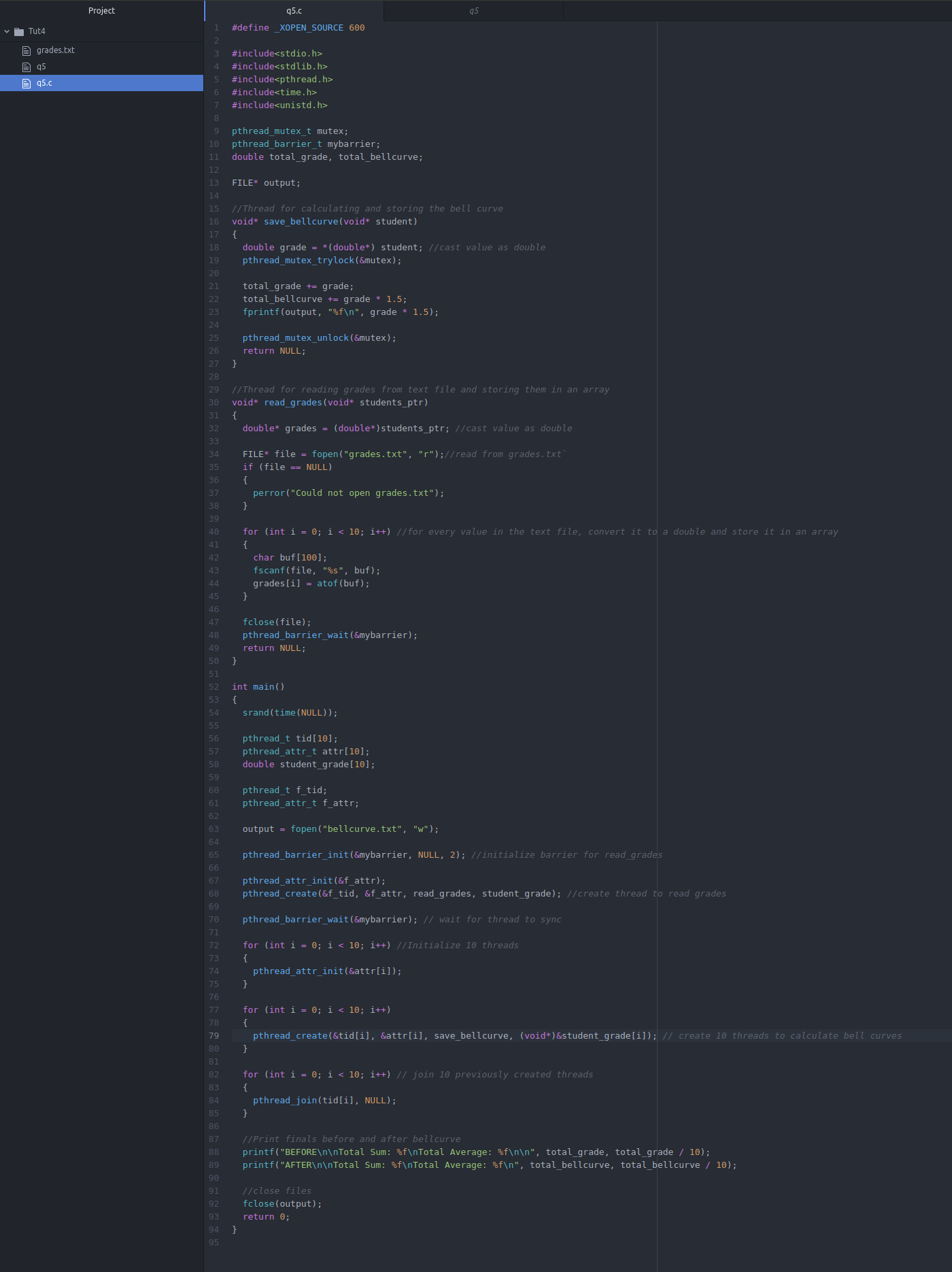
**CODE TOO LARGE TO PASTE IN HERE, SEE q3.c FOR THE FULL CODE.**

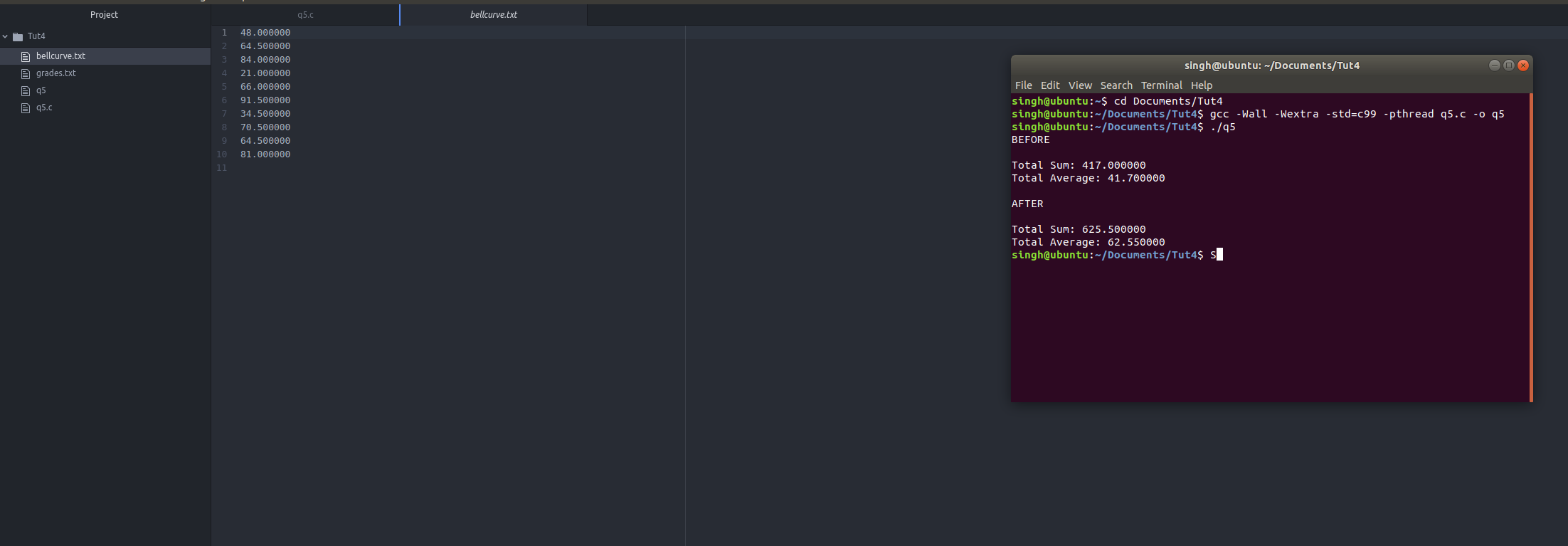
## Question 4





## Question 5





Please note that the output of the bellcurve.txt file may seem incorrect at first but that is because the first integer value from the grades.txt file is represented by the last integer value in the bellcurve.txt file.