EEE3535-01 **Fall 2020**

Assignment 6: Pthread

Due: Tuesday, Dec. 22, 2020, 11:59PM

1 Introduction

- The objective of this assignment to write a multi-threaded program using pthread.
- The program calculates the Mandelbrot set and plots a color-coded image via OpenGL. It is a standalone program and has nothing to do with xv6-riscv.
- The OpenGL part of implementation (i.e., graphics rendering) is provided in the skeleton code. You are only responsible for calculating the Mandelbrot set with pthread.
- Before starting the assignment, install OpenGL graphics libraries using the following commands in Ubuntu. If your work environment is Mac OS, the OpenGL libraries are pre-installed, and you do not have to take any actions.

```
$ sudo apt update
$ sudo apt install libglu1-mesa-dev freeglut3-dev mesa-common-dev
$ sudo apt install mesa-utils libgl1-mesa-glx
```

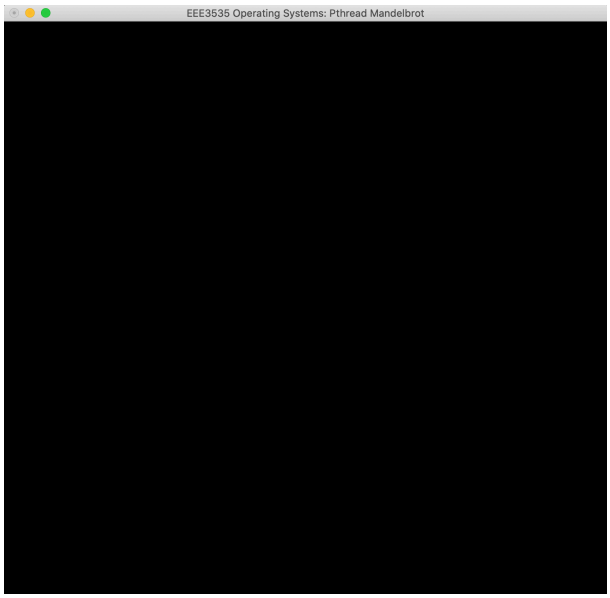
- Download a tar copy of skeleton code, untar it, and compile C++ files. The Makefile automates all compilation steps, so simply putting `make` in the terminal will do the job for you.

```
$ wget https://icsl.yonsei.ac.kr/wp-content/uploads/pthread.tar
$ tar xf pthread.tar
$ cd pthread/
$ make
```

- Try executing the program as follows. Typing in only the program name, `./mandelbrot`, will prompt a usage message that it takes an argument to specify the number of threads, i.e., `num_threads`. Put any number in it such as 1. The number of threads has no effects at this moment.

```
$ ./mandelbrot
Usage: ./mandelbrot [num_threads]
$ ./mandelbrot 1
```

- If the OpenGL libraries are correctly installed, you should see the pop-up of a black window as follows. In case you work in WSL on Windows 10, you must enable graphics forwarding; refer to earlier lecture slides regarding how to use `gedit` GUI editor with WSL.



- If everything works as described, press a `q` key to close the black pop-up window.
- You may clean up C++ build files (i.e., `*.o` object files and executable binary) by typing `make clean`. In case you make some major changes to the C++ files during your assignment, it may a good idea to clean up the project first and then build it again.

```
$ make clean
```

2 Mandelbrot Set

- The *Mandelbrot set* is defined as a set of points satisfying the following condition on the complex plane.

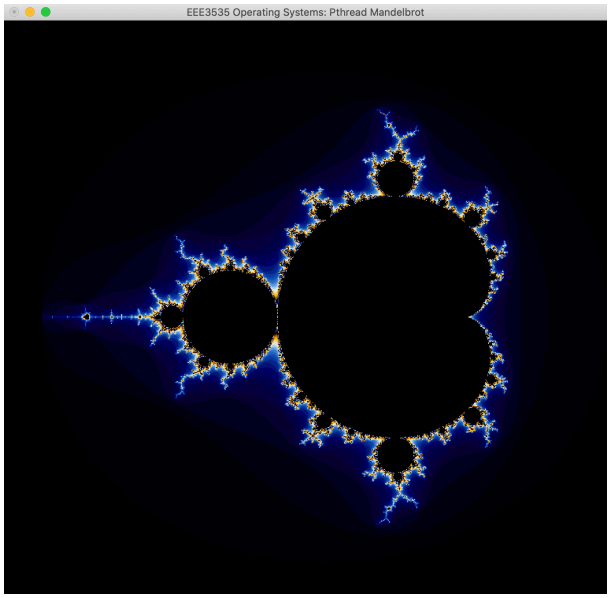
  $M = \{c \in C | \lim_{n \to \infty} Z_n \neq \infty\}$, where

  - M is the set of all complex numbers in the Mandelbrot set.
  - C is the set of all complex numbers on the complex plane.
  - $Z_0 = c$ and $Z_{n+1} = Z_n^2 + c$

- Although the definition of Mandelbrot set states that it requires iterating n over all integer values from 0 to ∞, the following assumptions can help us limit the number of iterations.

  - Assumption 1: If the magnitude of any Z_n is greater than $|2.0|$, it can be proved that Z will eventually reach infinity. Thus, if you ever get a Z_n with magnitude greater than 2.0, you can stop the iterations and claim that the point c is *not* in the Mandelbrot set.
  - Assumption 2: If you iterated 2,048 times but still find that a Z_n has magnitude smaller than 2.0, you can stop the iterations and claim that the point c is in the Mandelbrot set.

- To limit the scope of $c \in C$ on the complex plane, you can constrain the range of c values as (-2.2, -1.5) $< c <$ (1.0, 1.6). These range limits are defined as `minW`, `minH`, `maxW`, and `maxH` in the skeleton code, respectively. The `W` is short for width, and the `H` is for height.
- The size of an OpenGL window is set to 768×768, and the Mandelbrot set will be displayed at this resolution. Since image pixels are discrete elements, continuous values of c must be discretized as well. Thus, complex values in the range of (-2.2, -1.5) and (1.0, 1.6) are discretized into 768×768 units. For instance, a pixel at the left bottom corner of window represents (-2.2, -1.5), and the rest are (-2.2 + 3.2*`w`/768, -1.5 + 3.1*`h`/768). For each discrete c, you can iteration Z_n values based on the rules described above.
- Each pixel of the display window is colored depending on how many iterations are made until its c value is proven to escape from the Mandelbrot set (i.e., $|Z_n| > 2.0$) or belong to it. If the Mandelbrot set is correctly calculated, the result should be displayed as follows.



- If you own an Apple Mac with retina display, you may possibly get a quarter-sized image anchored at the bottom left corner of window. If this happens, try to clean up and re-build with an extra flag as follows.

```
$ make clean
$ make OPT=-DRETINA
```

3 Implementation

- In the provided skeleton code, the only file you have to work on is `mandelbrot.cc`. The `main.cc` file deals with OpenGL to plot the Mandelbrot set, and `stopwatch.*` files define a stopwatch class to measure the calculation time of Mandelbrot set.
- In the `mandelbrot.cc` file, `unsigned mandelbrot[resolution*resolution]` defines an 1-D array, where 768×768 elements are arranged in the row-major fashion.
- The `pthread_mutex_t lock` and `pthread_cond_t cond` are lock and condition variables, respectively.
- Then, you should notice two empty functions, `thread_exit()` and `thread_join()`. Use these functions to replace the POSIX function, `pthread_join()`, that waits for a child thread to complete. In other words, you are not allowed to use `pthread_join()` in this assignment.
- Instead, use a pair of lock and condition variables to implement `thread_exit()` and `thread_join()` to wait for children threads to finish. The `thread_exit()` function is called at the end of thread function to wake up another thread waiting on the condition variable, and the `thread_join()` is called in the `main()` function to check if it can pass the condition or should go into sleep.
- The `thread_mandelbrot()` function is supposed to be executed by individual threads, and this function is invoked via `pthread_create()` in the `calc_mandelbrot()` function called by the `main()`.
- In the thread function, `thread_mandelbrot()`, you need to fill in the `mandelbrot` array with the number of iterations that each entry takes to escape from the Mandelbrot set. For instance, index 0 of the array represents a complex number at (-2.2, -1.6). If this number iterated for 0 times to have $|Z_n| > 2.0$, then put `mandelbrot[0] = 0`.
- The `thread_mandelbrot()` and `calc_mandelbrot()` functions must be able to handle arbitrary number of threads such as the command run of `./mandelbrot 11` to divide 768×768 points into 11 threads. Simply dividing 768×768 by 11 will not work, and you should carefully think about load balancing between threads. Your code will be tested only with a reasonable amount of threads, e.g., up to 32 threads.
- Once the `mandelbrot` array is correctly filled in, you will see a nice visual plotted in the OpenGL window.

4 Submission

- Inside the `pthread/` directory, type `make tar` command. This command will prompt a message to get your student ID and then create a tar file named after your student ID.

```
$ make tar
Enter your 10-digit student ID:
```

- Do not modify the name of created tar file to include other tags such as your name, project6, etc.

5 Grading Rules

- The following is a general guideline for grading the assignment. 30-point scale will be used for this assignment. The minimum score is zero, and negative scores will not be given. Grading rules are subject to change, and a grader may add a few extra rules for fair evaluation of students' efforts.

  -5 points: The tar file is renamed and includes some other tags such as a student name.

  -5 points: Program codes do not have sufficient comments. Comments in the skeleton code do not count. You must make an effort to clearly explain what each part of your implementation intends to do.

  -10 points: The POSIX function, `pthread_join()`, is used to wait for threads.

  -10 points: The program cannot handle uneven number of threads, e.g., 11 threads.

-10 points: The Mandelbrot set is not correctly displayed.

-30 points: The Mandelbrot set is not calculated using pthread.

-30 points: No or late submission.

F grade: A submitted code is copied from someone else. All students involved in the incident will be penalized and given F for final grades irrespective of assignments, attendance, etc.

- Your teaching assistant (TA) will grade your assignments. If you think your assignment score is incorrect for any reasons, discuss your concerns with the TA. Always be courteous when contacting the TA. In case no agreement is made between you and the TA, elevate the case to the instructor to review your assignment. Refer to the course website for the contact information of TA and instructor: `https://icsl.yonsei.ac.kr/eee3535`
- Begging for partial credits for no viable reasons will be treated as a cheating attempt, and thus such a student will lose all scores for the assignment.