

Program 2: Bletchley Park

Due: October 26, 2022 5:30PM

Description

In this assignment you will gain hands-on experience with parallel programming and the difficulty of building correct parallel programs. You are tasked with taking a multi-threaded program and correctly identifying and guarding all critical regions.

In this assignment, a thread receives encrypted messages. These messages are placed in a buffer. Multiple decryption threads read each message and decrypt them. Currently there are no guards against race conditions. You must fix the code such that each message is decrypted correctly, no message waits for more than 2 seconds to be decrypted, and the buffer is not over-filled or drained past empty.

Updating your Container

The container for the course has been updated. You will need this new container for this assignment.

1. Type: `docker system prune`
2. Type: `docker pull tbakker/cse3320`

Setting up the Container

3. Open a command prompt or terminal and change to your Docker VM directory
4. Create the container with:

Linux / macOS

```
docker run -it --name=cse3320 -d -v ${PWD}/Code:/home/cse3320 tbakker/cse3320
```

Windows Command Prompt NOT Power Shell

```
docker run -it --name=cse3320 -d -v "%cd%"/Code:/home/cse3320 tbakker/cse3320
```

Starting the Container

5. Type: `docker start cse3320`

Attaching to the Container

6. Type: `docker attach cse3320`

Getting the Code

In your container:

1. `cd /home/cse3320`
2. `git clone https://www.github.com/CSE3320/Bletchley-Park-Assignment`
3. `cd Bletchley-Park-Assignment`

Building the Code

1. To build your code type: `make`
2. To start clean type: `make clean`

Running the Code

The program takes one parameter which is the number of decryption threads to run.

1. To run the program with 6 description threads type: `./decrypt 6`

Code:

`main.c` - This is the file you will modify for this assignment. You have two thread functions you will need to focus on:

`receiver_thread()` - There is a single receiver thread. It receives messages from the scheduler and places the messages in the buffer using the `insertMessage()` function.

`decryptor_thread()` - There are multiple decryption threads. They read messages from buffer and decrypt them.

- The program will crash if the buffer is overfilled. Do not remove this assert.
- The program will crash if the buffer is drained beyond 0. Do not remove this assert.
- The program will crash if any message waits longer than 3 seconds. Do not remove this assert.

`plaintext` - This directory holds the plaintext of the encrypted messages. You will use these to compare your results. You won't edit this file.

`ciphertext` - This directory holds the encrypted messages. You won't edit these messages.

`schedule.txt` - Each line of this file specifies what second a message appears and which encrypted message it is. You won't edit this file.

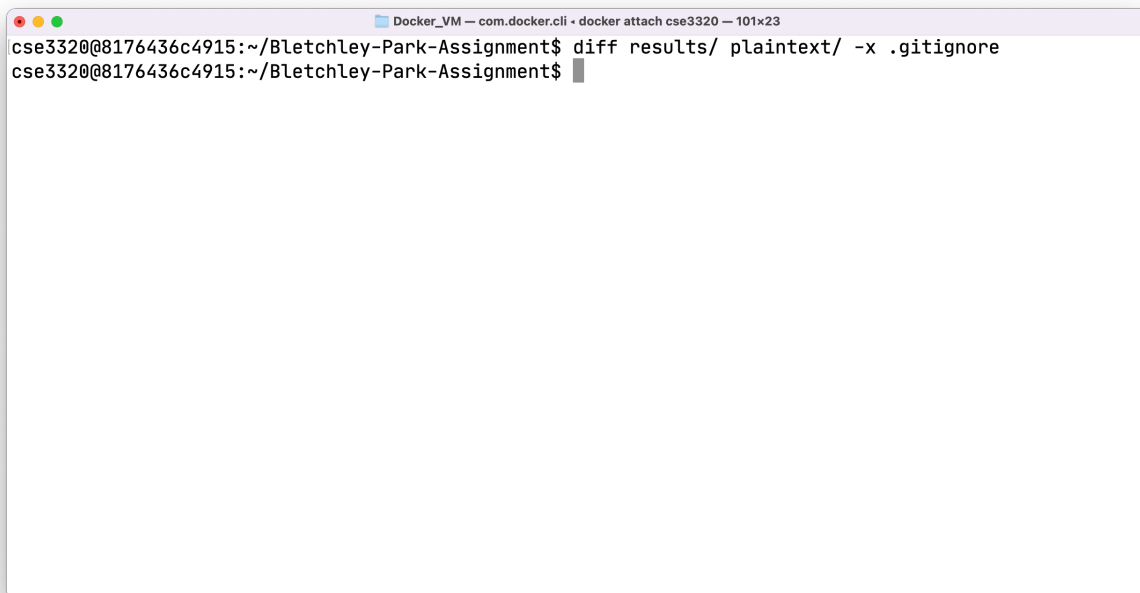
`clock.c` - This file contains the code that keeps the scheduler ticking. You won't edit this file.

`schedule.c` - This file reads the `schedule.txt` data file and serves the messages when they should appear. You won't edit this file.

Requirements

- Learn and understand how the message receiver and description threads work.
- Guard any critical regions touched by the message receiver and description threads.
- Your code must correctly decrypt all messages when run with 10 threads without crashing.
- Each message must be decrypted within 3 seconds of arrival.
- No deadlocks or race conditions.
- No segfaults.
- No memory leaks.
- To verify correct decryption type:

```
diff results/ plaintext/ -x .gitignore
```

A screenshot of a terminal window titled "Docker_VM - com.docker.cli - docker attach cse3320 - 101x23". The terminal shows a prompt "cse3320@8176436c4915:~/Bletchley-Park-Assignment\$" followed by the command "diff results/ plaintext/ -x .gitignore". The command is executed, and the prompt returns to "cse3320@8176436c4915:~/Bletchley-Park-Assignment\$" without any output, indicating a successful decryption.

No output means you correctly decrypted everything.

Administrative

This assignment must be coded in C. Any other language will result in 0 points. Your programs will be compiled and graded on a CSE 3320 container. Please make sure they compile and run in the container before submitting them. Code that does not compile on the container with:

```
make
```

will result in a 0.

Your program, `main.c` is to be turned in via Canvas. Submission time is determined by the Canvas system time. You may submit your programs as often as you wish. Only your last submission will be graded.

There are coding resources and working code you may use on Canvas and in the course github repositories at: <https://github.com/CSE3320> . You are free to use any of that code in your program if needed. You may use no other outside code.

Academic Integrity

This assignment must be 100% your own work. No code may be copied from friends, previous students, books, web pages, etc. All code submitted is automatically checked against a database of previous semester's graded assignments, current student's code and common web sources. By submitting your code on Canvas you are attesting that you have neither given nor received unauthorized assistance on this work. **Code that is copied from an external source or used as inspiration, excluding the course github or Canvas, will result in a 0 for the assignment and referral to the Office of Student Conduct.**