Assignment 2

If you are reading this either the day before or the same day this assignment is due:

- the only way to earn marks now is to write test code for other members in your group
- no marks will be earned for a copy-paste or refactor of someone else's code
- test code submitted will have a 10% penalty applied for not fully participating earlier

The basic work to do for test code:

- check each of the functions of the others group members' code
- if the others have all code in one function, then split into smaller functions yourself
- test the organized functions with enough variety of input to see that they work correctly
- use a text file to log results of executing tests with time and date for each test session

If there is not code submitted from others in your group, then that is your choice to gamble working on assignment last-minute.

Remember, you must have a git log of your progress to earn full marks. Commit to your git at least every other day, and break down your work into small pieces. Check Blackboard group for messages.

Learning Goals

- practice programming POSIX threads, i.e.: pthreads
- practice managing timing of scheduling with lottery algorithm
- practice use of condition and mutex variables to control sharing of resources
- have threads cooperate toward a common goal

You will work in random groups of three or four on this assignment, but this time submission is by group:

- you do not need to wait for anyone else to get started programming;
- using UFV GitLab as much as you can with your group;
- reference any website you use to help you write *small* parts of your code,
  (e.g.: the code used from an outside source should not solve the entire assignment)

- if so, give brief comments describing how you modified something you used and why;
- do not copy an entire program.
- Backup your code separately---code only in the group shared repo can easily be lost by others in the group experimenting with git commands; one git clone folder and a different folder of your own.
- Make a separate copy, IN A DIFFERENT FOLDER, and not a subfolder of the clone.

Together, choose a group leader and have them:

- create a project named `ThreadsProject` on GitLab
- add the others to this project
  (their UFV GitLab username should be on Blackboard discussion)
- set ``Developer'' role for each person in the project in UFV GitLab
  (otherwise they cannot git push)
- post your UFV GitLab username in Blackboard discussion if you have not done so already (make sure to repost if you change it again)

POSIX Threads

Write a program that uses `pthreads` to do the following:

- Schedule 100 threads by lottery algorithm.
- Each thread should have quantum of 100 msec before giving up the CPU again.
- Each thread should print one char to a 2D char array that displays a bar chart:
  - there are 10 bins (like a histogram chart), labelled 0 to 9 on the x-axis;
  - assign thread ID between 0 and 99, inclusive.
- Use the tens digit of the thread ID to increment value to display for corresponding bin when a thread gets its turn to execute; in other words, each bin counts the scheduled executions for its group of ten threads.
- Have a counter for each thread that allows it to execute an iteration two times before exiting (100 threads, means 200 char contributions to the chart total)
- You will be given code to display the chart.

Possible bonus, 2 marks:

- depending on your choice of lottery randomness, your histogram will give you a rough frequency distribution for how often your threads are being scheduled;

- if you care about statistics, this should help you practice the concept of random variable distributions, but otherwise, you do not need to know statistics;
- schedule your threads with a design of your own random distribution based on those used in statistics, and comment which type of distribution you design.

Expect demonstration videos for the following to help with your progress:

- timing
- mutex locking and unlocking
- condition variables
- display of chart

Debugging

With a VirtualBox, we can install GDB, which is the GNU Project Debugger. We do not have this installed on Jupyter Notebooks. Install and then run your program with GDB by compiling your program with the `gcc` option `-g`. GDB can tell you what line segmentation faults (bad memory accesses) happen inside your program.

Various commands for GDB:

• gdb	begin an interactive session of GDB
• quit	1cm exit the interactive GDB terminal prompt
• file <your_program>	load your program
• run	let GDB execute your program

Submission

Submit your individual git repository as a bundle. Create a bundle file on the command line using git:

```
git bundle create ThreadsProject.bundle master
```

inside the `ThreadsProject` folder. Do not worry about adding this file to your project itself. It is meant to compress the project into one file with the records of your commits. You can check what is in your bundle file with the following command:

```
git bundle verify ThreadsProject.bundle
```

Submit the bundle file only to our Blackboard Assignment and Tests section in Assignment 2. You can submit multiple times, but I only mark the last submission before deadline. Submit every couple of days, and then you are always worry free to know your submission has most of your hard work.

Please do not email me to check that your files are submitted to Blackboard. You are capable of checking that yourself.

This assignment is due on Thursday, Oct 20, 11:59 pm.

(rubric on last page)

Marking Rubric

20 marks total

	Excellent	Good	Adequate	Poor
Code style (4 marks)	no mixed whitespace (tabs and spaces), indentation aligns per block of code, meaningful variable names, and no unnecessary lines of code	one or two issues with mixed whitespace, alignment, variable names, or extra unnecessary code	more than two issues with style	many issues with style
Code functionality (16 marks)	project has regular (every other day) git log, no runtime errors, threads scheduled by lottery, each thread runs for the length of its quantum, each thread iterates two times, update to histogram chart displayed so numeric amounts are readable, use of mutex and condition variables to control critical sections of code	project has git log with multiple commits spread out over time, no runtime errors, threads scheduled by lottery, each thread runs for the length of its quantum, each thread iterates two times, use of mutex and condition variables to control critical sections of code	git log with few commits, or a few runtime errors, or variable names are trivial `"a"`, `"b"`, `"c"`, or output is difficult to read; resources protected by mutexes between threads	no git log, there are runtime errors that halt the program, or output is not presented in a readable format, or the program does not compile