CS 252: Systems Programming
Fall 2019
Lab 4: Introduction to Threads

Prof. Turkstra

Goals

The goal of this project is to implement a program that spawns threads that interact with each other. Some skeleton code has been provided, so you will not be starting from scratch.

Deadlines

The deadline for the final submission is **Monday, October 28 11:58pm**.

Setup

Login to a CS department machine (a lab machine or `data.cs.purdue.edu`), navigate to your preferred directory, and run

```
git clone ~cs252/repos/$USER/lab4.git
cd lab4
```

Testing

Your lab will be graded with a mixture of automated tests and hand grading. The automated tests test_part1_count, test_part1_order, and test_part2_count will evaluate the correctness of your code. The hand-grading will ensure that you used threads by code inspection and with test_part2_nondeterminism. If your code is deterministic, you are probably not using threads the way we want you to. However, nondeterminism sometimes looks deterministic. If you fail test_part2_nondeterminism you will not automatically lose points, but your code will be graded with more scrutiny.

To run each of the test cases, make your code and type:

```
./test_cases/test_part1_count
./test_cases/test_part1_order
./test_cases/test_part2_count
./test_cases/test_part2_nondeterminism
```

1 Producer/Consumer Problem

1.1 Problem Statement

Using the pthreads API, implement producer/consumer communication through a bounded buffer. The solution to the bounded buffer problem was described in class. The string to be used is "the greatest teacher, failure is", i.e. the producers will be producing characters from this string. Once a producer has inserted all characters from the string, it should terminate. Once a consumer has consumed **strlen(producer_string)** characters it should also terminate.

Test your solution with a multi-character buffer and with multiple producers and consumers. Of course, with multiple producers or consumers, the output display will be gobbledygook. However, note that a correct solution will not produce arbitrary output!

1.2 Formatting

Make sure your solution can be run using the command line as follows:

```
./part1 <numproducers> <numconsumers>
```

The output should be in the form:

```
Thread X produced H
Thread Z consumed W
```

X and Z in this example are numbers, ranging from 0 to numproducers/numconsumers.

1.3 Suggested Implementation

You should define the following functions:

```
void *producer(void *ptr);
void *consumer(void *ptr);
```

Your **main()** function should use **pthread_create()** to spawn the appropriate number of producer and consumer threads. There are many ways to implement this correctly. You can use semaphores, mutexes, and/or condition variables. It is entirely up to you. Each producer should be numbered 0 to N. The same goes for consumers.

2 Terraforming

2.1 Atmospheric Chemistry

The atmosphere on the planet Kessel consists primarily of the elements Nitrogen and Oxygen and of their compound Nitrogen diOxide (NO_2). Trace quantities of Ozone (O_3) have also been discovered. There are 4 chemical reactions that occur in the atmosphere:

1. $N + N \rightarrow N_2$
2. $O + O \rightarrow O_2$
3. $N_2 + 2O_2 \rightarrow 2NO_2$
4. $3O_2 \rightarrow 2O_3$

Each of the reactions occur when enough molecules of the requisite compounds are present. Remember that the formation of NO_2 and O_3 are *molecular* and not atomic operations. You need one *molecule* of nitrogen and two *molecules* of oxygen to produce two molecules of NO_2. Similarly, you need three *molecules* of oxygen to produce two molecules of O_3.

2.2 Modeling With Threads

The scientists of the Rebel Alliance have asked us to help out on Kessel by creating a computer simulation of the the terraforming process. They require a program that:

a) Injects atoms of Nitrogen and Oxygen into the atmosphere

b) Monitors the formation of the N_2, O_2, NO_2 and O_3 molecules

c) Prints out messages at every stage

In order to solve this problem, you may think of each reaction as a separate thread. Everytime a Nitrogen or Oxygen atom is created, you will have to print out the corresponding message:

```
An atom of nitrogen was created
An atom of oxygen was created
```

Whenever a reaction takes place, print out the corresponding message:

```
Two atoms of nitrogen combined to produce one molecule of N2
Two atoms of oxygen combined to produce one molecule of O2
One molecule of N2 and two molecules of O2 combined to produce two molecules of NO2
Three molecules of O2 combined to produce two molecules of O3
```

2.3 Suggested Implementation

Here is one method of implementing the solution...

1. Each N atom invokes a procedure called n_ready when it's ready to "react" and prints out the appropriate message (see above).
2. Similarly, each O atom invokes o_ready when it's ready to react. Again, print out messages.

3. A procedure called make_nitrogen is called when a molecule of Nitrogen is formed. The corresponding procedure for Oxygen is called make_oxygen.

There are a number of ways of proceeding from this point on. You may think of make_nitrogen and make_oxygen as being both producers and consumers (since they "consume" 2 atoms to "produce" a molecule of N_2 or O_2, and they then "consume" one molecule of N_2 or two molecules of O_2 to "produce" of NO_2). Another procedure called make_NO2 is then the "consumer" for NO_2. You may similarly have a procedure for O_3 called make_ozone. The trick is to synchronize appropriately between all of these procedures. Feel free to use semaphores or mutexes and condition variables to effect the synchronization.

3 Submission

The deadline for final submission is **Monday, October 28 11:58pm**.

To turn in your final submission:

1. Login to a CS department machine
2. Navigate to your `lab4` directory
3. Run `make clean`
4. Run `make` to check that your code builds correctly
5. Run `make submit_final`

These Standard Rules Apply

- Follow the coding standards as specified on the course website.
- Your code must compile and run on the lab Linux machines.
- Do not look at anyone else's source code. Do not work with any other students.

Grading

Grading is subject to change, but will roughly follow:

15% Part 1- Buffer Management (Count)
25% Part 1- Synchronization (Order)
50% Part 2- Correctness (Count)
10% Part 2- Threads (Nondeterminism)