CMPSC 440
Operating Systems
Spring 2014

Laboratory Assignment Four: A Producer-Consumer Model with Semaphores

Introduction

Processes and threads are commonly used by the operating system itself and by the programs that run on the operating system. In the previous laboratory assignment you learned how to implement and evaluate Java programs that use the `synchronized` keyword to create a multi-threaded produced consumer model. In this laboratory assignment, you will download, use, extend, and further investigate with a multi-threaded producer-consumer model that uses semaphores, As in the previous assignment, you will add some features to the model that enable you to study it more effectively. Finally, you will systematically break various parts of the model in order to observe what can go wrong when implementing multi-threaded Java programs that use semaphores.

Accessing the Producer-Consumer Model

As in the previous assignment, you should change into directory for the Git repository that I use to share files with you. Now, you can type the command `git pull` to retrieve all of the files for this laboratory assignment. If this step did not work correctly, then please see the instructor. Finally, you should use your terminal window to browse the files that are in this Git repository. In particular, please look in the `labs/lab4/src/` directory and use Vim to study the Java programs that you find. What files are used to implement the producer-consumer model?

Understanding and Extending the Producer-Consumer Model

In this laboratory session, we will use code segments from Vijay K. Garg's book, *Concurrent and Distributed Computing in Java*, to learn more about how to implement a producer-consumer model with semaphores. Before you start to modify the programs that you received through the Git repository, take some time to learn more about the history of the term semaphore. What is the meaning of this term when it is used in a context not connected to the discipline of computer science? What was the first paper to propose the use of semaphores in computer science? Finally, it is important to note that this program uses both binary and counting semaphores. What are the similarities and differences between these two types of semaphores?

After you have carefully studied the source code of the producer-consumer model, you can compile and execute it. What type of output does this program produce? Will this program halt? If yes, then how long will it take to halt? If no, then why does it not halt? How is this program similar to and different from the producer-consumer model that you used in the last assignment?

As you will see from studying the source code, the current implementation of the `ProducerConsumer` always produces debugging output. Leveraging the JCommander system that you learned how to use in a previous laboratory assignment, you should now implement a command-line interface for the `ProducerConsumer`. First, you should add a parameter that can indicate whether or not the `ProducerConsumer` should produce debugging output. To implement debugging

output correctly, you will need to find each of the classes that execute output-producing statements. Next, you should notice that the current implementation of the `ProducerConsumer` only creates a single producer and consumer. To extend the program, you should add a command-line argument that allows the user to specify the number of consumers.

Finally, you should notice that the `ProducerConsumer` does not does not furnish any information about the specific thread that is producing the debugging output. As such, you should further extend the debugging output so that the name of the thread is also displayed. To add this feature, you may need to change the inheritance hierarchy for the `Producer` and `Consumer` classes. You will also need to determine which methods provided by `java.lang.Thread` return information about the name of the thread that is currently executing.

Understanding Defects in Multi-Threaded Programs

After you have finished writing the source code to support the command-line arguments for the `ProducerConsumer`, you can take steps to explore what happens if you make mistakes when implementing multi-threaded Java programs that use semaphores. Please ensure that you carefully make the following changes, recompile the program, execute the program, record the incorrect program output, and comment on why the output is evident. Once you are done making the required defective implementations of the program, please return the modified code to the correct state!

1. Comment out the first two calls to `P` in both the `deposit` and `fetch` methods. For example, in the `deposit` method you would produce the following new source code statements. What output does the program produce? Why does it produce this output?

```
//isFull.P(); // wait if buffer is full
//mutex.P(); // ensures mutual exclusion
```

2. After returning the source code to the correct state, you should now comment out the calls to `V` inside of the `deposit` method, thus yielding the source code written below. What output does the program now produce? Can you explain why it produces this output?

```
//mutex.V();
//isEmpty.V(); // notify any waiting consumer
```

Reflecting on the Producer-Consumer Models

Now that you have extended your producer-consumer model and taken steps to better understand the types of defects that are evident in multi-threaded code that uses semaphores, you can start to further study the behavior of the model. Try to construct two, four, and six `Consumer`s instead of using a single `Consumer`. Now, execute the program again and let it run for a longer period of time, making sure to record a portion of the output. Can you find any pattern in the output in terms of which `Consumer` thread the operating system kernel schedules? As in the previous laboratory assignment, you should consider the use of operating system tools that you can use to better understand the behavior of the multi-threaded producer-consumer model.

Take some time to compare and contrast the two producer-consumer models that you have implemented in these two laboratory assignments. After studying the source code of both implementations, you should think about the similarities and differences that they exhibit. Which one do you think is easier to implement, test, and maintain? Which one is more likely to lead to a defective implementation? How did you arrive at these conclusions?

Summary of the Required Deliverables

This assignment invites you to submit printed and signed versions of the following deliverables:

1. A clear description of the term semaphore, with comments about its historical origins
2. A revised and extended implementation of the producer-consumer model
3. Output demonstrating the command-line arguments that you added to the `ProducerConsumer`
4. The output from running the producer-consumer model in all defective configurations
5. A comprehensive analysis of the output of each defective multi-threaded Java program
6. A reflection on the characteristics of the two different producer-consumer models

In adherence to the honor code, students should complete this assignment on an individual basis. While it is appropriate for students in this class to have high-level conversations about the assignment, it is necessary to distinguish carefully between the student who discusses the principles underlying a problem with others and the student who produces assignments that are identical to, or merely variations on, someone else's work. As such, deliverables that are nearly identical to the work of others will be taken as evidence of violating the Honor Code.