CS431 — Exercise 2

January 18, 2017

Due: Monday, January 23, 2017 before midnight (50 points)

In this exercise you will practice some multithreaded programming to help gain insight in the complexities of managing multiple threads of execution for a single group of resources.

Multithreaded Program

You will write a Java program that has three threads:

1. User I/O thread: this thread is responsible for requesting commands from the user and acting on those commands.
2. File read thread: given some command from the user, this thread will read the content of a file and store lines of text from the file in a queue.
3. Processing thread: this thread will constantly check the queue populated by the file read thread and remove a line to process it by counting the number of occurrences of lowercase letters, uppercase letters, and digits 0-9 storing these in an array.

User I/O Thread

This thread should prompt the user to enter a command. Acceptable commands are:

1. `read filename.txt`

   This command will pass the file name to the file read thread. I recommend also using a queue for storing each file name so the file read thread can process the queue in the same way as the processing thread.

2. `counts`

   This command should cause the program to print out the current values of the number of occurrences determined by the processing thread.

3. `exit`

   This command should cause the program to exit. Can you find a way to cleanly exit each thread without needing to use `System.exit` to end the program?

File Read Thread

This thread should run a loop processing file names provided by the user I/O thread storing the lines of text read in a queue accessible by the processing thread.

Processing Thread

This thread should run a loop processing strings of text from the queue populated by the file read thread. It should count occurrences of lowercase letters, uppercase letters, and the digits 0-9 storing the total count for all lines read in an array.

The file read and processing threads should *block* when the queues are empty until there is something available for them to process. There are tools in the Java standard library to help with this. Look in the Collections API.

This model of multithreaded programming is known as the *producer-consumer* model.

Submission

1. On `https://codebank.xyz`, create a project named `CS431-EX2`. Follow this naming convention precisely including case.
2. Create your own local repository by the following:
   (a) Navigate to your local directory for this project.
   (b) Run `git init` to initialize the repository.
3. Make sure you add the reference to the remote repository in your local repository with:
   ```
   $ git remote add origin https://codebank.xyz/username/CS431-EX2.git
   ```
4. Your project should have a main class named `ThreadTest` in a file named `ThreadTest.java`. You can have other files or classes, but it should successfully compile and run by simplying using:
   ```
   $ javac ThreadTest.java
   $ java ThreadTest
   ```

I will create additional test input files when grading the assignments.

You will lose points if your submission is not correct (e.g., incorrect repository name, file names, class names, or package declaration that causes the above commands to fail to run).