Lab 9: Actors
Instructor: Mack Joyner

Course Wiki: http://comp322.rice.edu
Staff Email: comp322-staff@mailman.rice.edu

1 Lab Goals

In today’s lab you will use HJlib Actors to approximate Pi using the manager-worker paradigm. This lab can be downloaded from the following GitHub repository: https://classroom.github.com/a/6LBjEVkD

Use the git command-line client or IntelliJ to checkout the project into appropriate directories locally.

In today’s lab, you need to use NOTS to run performance tests. If you need any guidance on how to submit jobs on NOTS manually, please refer to lab 5 or ask a member of the teaching staff.

2 HJlib Actors

HJlib actors were introduced in Lectures 31–32. An actor class is defined by extending the edu.rice.hj.runtime.actors.Actor class. Concrete sub-classes of Actor are required to implement the process() method.

The following code snippet shows the schema for defining an actor class:

```java
import edu.rice.hj.runtime.actors.Actor;
public class EchoActor extends Actor<Object> {
    protected void process(Object aMessage) {
        ...
    }
}
```

Method calls can be invoked on actor objects, and they work just like method calls on any other HJlib objects. However, what distinguishes actors from normal objects is that they can be activated by the start() method, after which the HJlib runtime ensures that the actor’s process() method is called in sequence for each message sent to the actor’s mailbox. The actor can terminate itself by calling exit() in a process() call.

Messages can be sent to actors from actor code or non-actor code by invoking the actor’s send() method using a call as follows, “someActor.send(aMessage)”. A send() operation is non-blocking and asynchronous. The HJlib Actor library preserves the order of messages between a sender and receiver pair, but messages from different senders may be interleaved in an arbitrary order at a single receiver.

As mentioned in the lectures, there are three basic states for an actor:

- **new**: when an instance of an actor is created, it is in the new state. In this state, an HJlib actor will receive messages sent to its mailbox but will not process them.

- **started**: in this state, the actor will process all messages in its mailbox, one at a time. It will keep doing so until it decides to terminate. In HJlib, an actor is started by invoking its start() method: e.g., “myActor.start()”.
terminated: in this state the actor has decided it will no longer process any more messages. Once terminated, an actor cannot be restarted. An actor requests termination by calling its exit() method, which changes the actor’s state to terminated after the process() call containing exit() returns. Note that the exit() call does not itself result in an immediate termination of the process() call; it just ensures that no subsequent messages will be processed.

All async tasks created internally within an actor are registered on the finish scope that contained the actor’s start() operation. The finish scope will block until all actors started within it terminate. This is similar to the finish semantics while dealing with asyncs.

Other examples that were discussed in Lectures 31–32 include Pipeline.java and ThreadRingMain.java.

2.1 Tips and Pitfalls

- Use an actor-first approach when designing programs that use actors i.e., think about which actors need to be created and how they will communicate with each other. This step will also require you to think about the communication objects used as messages.
- If possible, use immutable objects for messages, since doing so avoids data races and simplifies debugging of parallel programs.
- When overriding the start() or exit() methods in actor classes, remember to make the appropriate calls to the parent’s implementation with super.start() or super.exit(), respectively.
- The HJlib actor start() method is not idempotent. Take care to ensure you do not invoke start() on the same actor instance more than once. The exit() method on the other hand is idempotent, invoking exit() multiple times is safe within the same call to process().
- **Always remember to ensure that all started actors terminate using the exit() method.** If an actor that has been started but is not terminated, the enclosing finish will wait forever (deadlock).
- When sending asynchronous messages to actors, be careful to use Actor.send(), not Actor.process(). Calling Actor.process() will do the work synchronously, and not create any parallel work.

3 Pi Computation using Bailey-Borwein-Plouffe Formula

Our first exercise involves computing π to a specified precision using HJlib. The following formula can be used to compute π:

\[
\pi = \sum_{n=0}^{\infty} \left( \frac{4}{8n + 1} - \frac{2}{8n + 4} - \frac{1}{8n + 5} - \frac{1}{8n + 6} \right) \left( \frac{1}{16} \right)^n
\]

The PiSerial.java file contains a sequential algorithm that uses a while loop to compute more and more terms of the series until a desired precision is reached.

We have already provided a version of PiActor.java with TODO comments. For this section, your assignment is to convert the sequential program in PiSerial.java (for computing π to a desired precision) to an actor-based parallel program in PiActor.java by filling in code at the TODO segments. Next, you will need to evaluate the performance of the serial and parallel versions, PiSerial.java and PiActor.java, on a NOTS compute node. The reference implementation achieved over 11× speedup over the sequential implementation on NOTS while using 16 worker threads. You will need to achieve at least 2x speedup on a manual NOTS submission to receive credit for the lab. Remember to edit your slurm file before submitting the NOTS run.
Note that because the template PiActor class has no functionality filled in, running the tests without any changes will cause them to hang.

4 Turning in your lab work

For this lab, you will need to turn in your work before leaving, as follows.

1. Show your work to an instructor or TA to get credit for this lab, including the output of the NOTS performance test for PiActor by Friday, April 5th at 5pm.
2. Check that all the work for today’s lab is in the lab9 directory.