CS-377 Parallel Programming
Assignment 4: Dining Java Philosophers Lab Report

Eliot Cowley

April 15, 2015

Implementing the Dining Philosophers problem in Java was an interesting, though somewhat frustrating experience. From my experience with Java, it doesn't seem particularly suited for concurrency. Programming with UPC was as easy as using single statements like `upc_for_all`, and UPC would take care of the rest. With Java, you really have to know what you're doing.

In principle, object-oriented programming should be a perfect match for concurrency, but until UPC I had never found a programming language that made it easy. Java does have some useful keywords that deal with concurrency. At first it was difficult for me to tell how exactly to use the keyword ”synchronized”– synchronized methods made sense to me, since the whole class would then be like a lock, but synchronized code blocks confused me. What I had to realize was that each object has its own intrinsic lock, and a synchronized code block acts on that object's lock.

Once I figured out how to use ”synchronized”, it started to become clearer to me how to implement the program. I'm always surprised by how versatile Java is. I'm constantly learning new things I can do with it, which speaks to the power of the language.

An issue that I encountered during this assignment was determining whether my program would deadlock or not. I ended up simply running it for several hours and examining my code carefully for anything that might lead to one thread holding a lock and never releasing it. However it made me realize that it's hard to tell for sure if a program won't deadlock, which brought to mind the example that Professor Smith gave of the IBM machine that had been running for many years before it finally deadlocked.

I'm glad that we did this assignment. Java is a language I use every day, and learning how to write concurrent Java programs is something that will likely become useful in the future. As parallel programming becomes more and more necessary, existing languages will either have to adapt or new languages with better concurrency implementations will have to take their place.