Kenta Hasui

CS377

Assignment 6 Report

For this assignment, we used Linda to implement solutions to the Producer-Consumer problem and the Bounded-Buffer problem for a single producer and a single consumer. Ruby's implementation of Linda, Rinda, was surprisingly intuitive and easy to use. This was an exciting assignment for me because I've been wanting to learn Ruby, and this was a perfect opportunity for that.

I started with the bounded-buffer problem, which was a bit harder to figure out than the simple producer-consumer problem. I created a Consumer class and a Producer class, as well as a server.rb file that initialized the semaphores. I realized that since there was only a single producer and a single consumer, we didn't need to keep the variables "rear" and "front" inside the tuple space. Only the consumer uses front, which points to the index of the next item to be consumed. Only the producer uses rear, which points to the next index in the shared buffer for an item to be produced into. Thus I simply kept these as instance variables for individual consumer and producer objects. If we had multiple producers and multiple consumers, keeping these variables in the tuple space would have been necessary. I liberally used ruby's symbols, as their immutable property seemed perfect to use as tags in tuples. Implementing semaphores was shockingly easy, as Rinda has all the synchronizing figured out for you! In the server class I added 20 "Empty" semaphores and the rest fell into place.

After the bounded buffer, I moved on to the simpler producer-consumer problem with a buffer capacity of 1. In this case, we didn't need to implement empty and full as counting

semaphores. Furthermore, the order in which the consumer took elements from the shared buffer didn't matter, as there could only be one item placed there at a time. Thus it was a much simpler exercise, and I didn't have much trouble with it.

Throughout this assignment, it was amazing to see just how powerful Ruby was. It seemed like a very clean and efficient language. Creating a new class was very simple, as well as creating new methods and instances. Being able to declare and initialize multiple variables at a time was a life saver for tuple matching. My favorite part so far was being able to declare the scope of a variable by prepending the values with @, @@, $, _, etc. This made it much easier to distinguish which variables were local, global, or instance variables just by reading through the code. There was much less backtracking involved than in a language such as Java. In all, this was a very fulfilling and exciting assignment and I hope to get more experience with Ruby going forward.