Schuyler Asplin

CSCD 340 Operating Systems

Chapter 31 Semaphores (coding problems 1-5)

1. The first problem is just to implement and test a solution to the **fork/join problem**, as described in the text. Even though this solution is described in the text, the act of typing it in on your own is worthwhile; even Bach would rewrite Vivaldi, allowing one soon-to-be master to learn from an existing one. See fork-join.c for details. Add the call sleep(1) to the child to ensure it is working.
2. Let’s now generalize this a bit by investigating the **rendezvous problem**. The problem is as follows: you have two threads, each of which are about to enter the rendezvous point in the code. Neither should exit this part of the code before the other enters it. Consider using two semaphores for this task, and see rendezvous.c for details.
3. Nowgoonestepfurtherbyimplementingageneralsolutionto**barriersyn- chronization**. Assume there are two points in a sequential piece of code, called P1 and P2. Putting a **barrier** between P1 and P2 guarantees that all threads will execute P1 before any one thread executes P2. Your task: write the code to implement a barrier() function that can be used in this man- ner. It is safe to assume you know N (the total number of threads in the running program) and that all N threads will try to enter the barrier. Again, you should likely use two semaphores to achieve the solution, and some other integers to count things. See barrier.c for details.
4. Now let’s solve the **reader-writer problem**, also as described in the text. In this first take, don’t worry about starvation. See the code in reader-writer.c for details. Add sleep() calls to your code to demonstrate it works as you expect. Can you show the existence of the starvation problem?
5. Let’s look at the reader-writer problem again, but this time, worry about starvation. How can you ensure that all readers and writers eventually make progress? See reader-writer-nostarve.c for details.