Grand Canyon University

Assignment 3: Deadlock Avoidance

Evan Lee and Ryan Scott

CST-315: Operating Systems Lecture and Lab

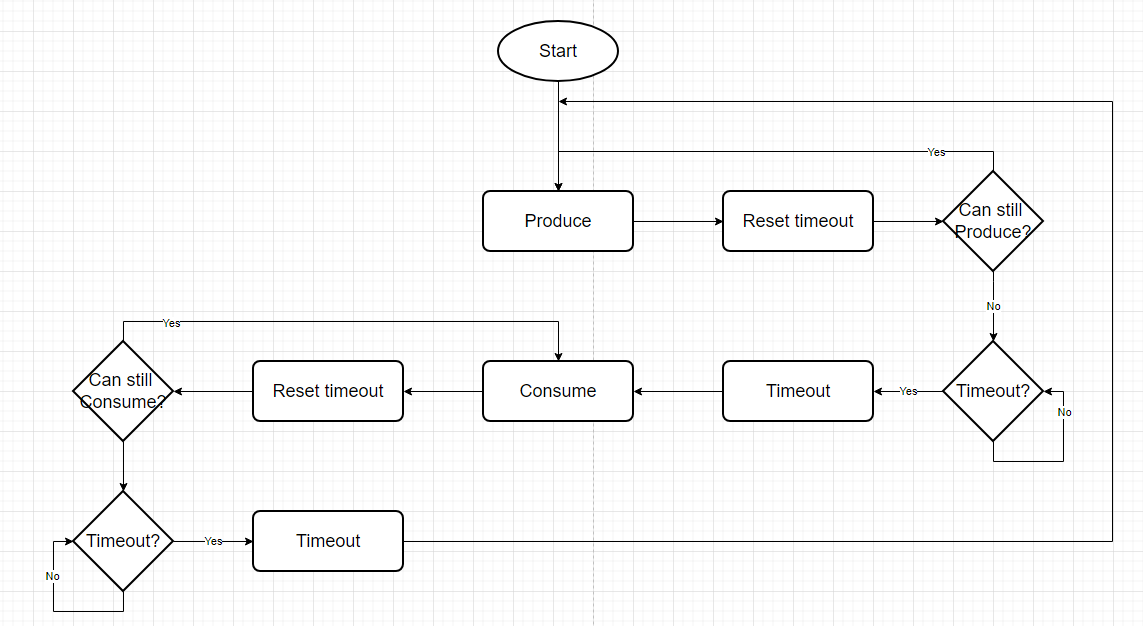
Dr. Ricardo Citro

February 27, 2022

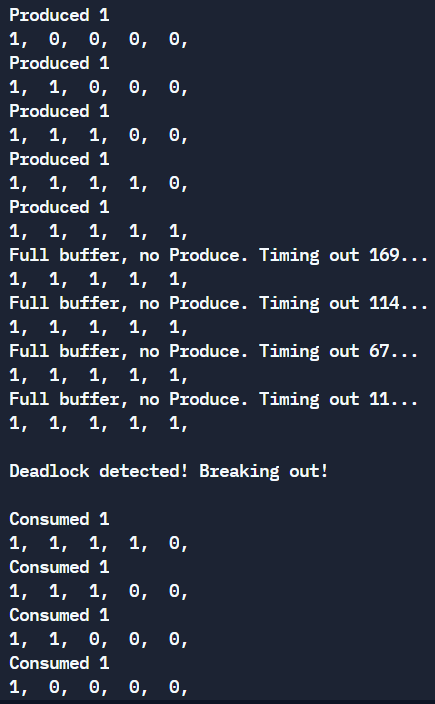
**Implementation of Deadlock Avoidance:**

For our implementation of deadlock avoidance, we used the producer and consumer problem and created a model that would have the system deadlock. We needed to implement a deadlock avoidance system that would check to see if there is a deadlock by seeing if the producer and consumer are stuck, then reset the system if so. We used the clock() function to be able to time each producer and consumer call, and if it reached a certain amount of time, it would reset and unbreak itself.

**Flowchart on Deadlock Avoidance Using a Timing System:**



**Code Output:**



In the red box, the producer function is called, adding the number 1 to an array with buffer size 5. It will continue to produce until the length of the array hits the buffer size. Then in the blue, the buffer is seen as full and it creates a deadlock not allowing the consumer function to work. This will trigger the timer to count down until it hits zero in which case a deadlock is detected and the producer function is broken out of to reset the system. The numbers used for the countdown are shown as a representation for different snapshots in time and would decrease by a set number normally. Lastly in yellow, the system has been reset and the consumer function can now consume the items.

**Analysis on Deadlock Solution:**

The use of a timing system for deadlock avoidance was a clever way to solve a deadlock problem for our producer and consumer code. It worked for the producer and consumer scenario but there could be implementation problems that could arise in a different project setting. Consider a company that needs to process lots of information using complex algorithms that would take a long period of time to complete. In this example let’s say it normally would take 1 hour to compute. Then the deadlock avoidance timer would need to be set to around 1.5 hours to account for any variation in complexity. If a deadlock were to occur in this situation, it would take 1.5 hours to figure out if a deadlock had occurred, and on top of that it could not have deadlocked but just been a slower calculation time. Then all of the progress is erased because the deadlock avoidance system would restart the system even when there was no deadlock. This is a major problem that could cost companies lots of time and money. A way to avoid deadlocks would be to change the conditional statement in the code for both the producer and consumer functions from while(true) statements to while(true && !produce) / while(true && !consume) where produce and consume (initialized on line 14 in deadlock.cpp) are the Boolean values that act as a binary semaphore to allow the producer and consumer functions to operate.