**Deadlock Avoidance**

Mathew Paschall

CST-221

John Zupan

2/4/17

I decided to make a few threads that would attempt to run a function in my program. In order to ensure that only one process could run my function at any time, I created a mutex that is locked when a thread enters and unlocked when the function is complete. In order to simulate a deadlock with just 1 resource, I created an infinite loop, thus barring any other thread from performing the function. The second thread always gets the resource first, and thread one will never be able to access the resource. In this case, waiting for the process to complete will not work, as the second thread will never free up the resource, no matter the wait. I then created a scenario where there was no deadlock. The second thread would eventually finish and the first thread could either run after thread 2 was completely finished, or they could take turns getting the resource and completing the function.

In a situation where there is a deadlock with a single resource, simply using a timer to attempt to get the resource again will not work, as restarting a thread will not affect a deadlock using only one resource. It would be effective if the process being restarted was holding a resource the other process needed, allowing it to run to completion and then using the resource itself.

An alternative approach to this would be to implement a signal to have the running process save its progress and stop running, then attempt to run again after a short sleep. This would allow the other starved processes to get the single resource and run to completion or until others become starved. In this case, processes could continue to run, even if it takes longer.