# Embedded System Design
# Dining Philosophers

Max Thrun — Ian Cathey — Mark Labbato

November 18, 2012

## Description

The purpose of this lab was to implement a solution to the 'Dining Philosophers Problem'. In short, the problem describes 5 philosophers all sitting around a circular table. Each philosopher shares a fork with the person next to him. A philosopher can either be thinking, in which he holds no forks, or eating, in which he holds both. If a fork is not available then he waits until it does become available. To solve this problem we implemented the common resource hierarchy solution [1]. Our solution is based on the convention that each fork is numbered and the philosopher will pick up the fork with the lowest number first. This breaks the deadlock of each person simply picking up the fork to their left. For instance, with 5 philosophers philosopher 5 and philosopher 1 both share fork 1. Fork 1 is the lowest value fork for both these philosophers so they cannot both pick it up at the same time. This fact prevents the deadlock of all the philosophers taking one fork from a particular side.

From our programs perspective, when each thread starts it calculates the value of the fork on its left and right. Then, after a random 'thinking' delay, it checks to see which fork has the lower value. If the fork on the left has the lowest value it tries to pick that one up first and vica versa. Each fork is represented by a mutex which ensures that it can only be touched by one thread (aka philosopher) at a time. After the philosopher has obtained both forks it delays for a random amount of time while it 'eats' and then releases both forks.

Additionally, a pseudo-random number generator, based on a LFSR, was developed and fitted with a mutex lock to make it thread safe. Using random numbers for our delays allowed us to avoid time aliasing which might occur with fixed length delays. Time aliasing can cause conditions such as deadlocks and starvation to remain hidden since the threads are executing the exact same sequence and timing over and over. If that specific time interval happens to be deadlock free you will never see it occur. While the LFSR does not provide truly random delays it allows us to have more confidence in the robustness of our solution.

## References

[1] `http://en.wikipedia.org/wiki/Dining_philosophers_problem#Resource_hierarchy_solution`