TRAFFIC: A Strategy to Fighting Deadlock

Walter B. Gress V

Drexel University

Friday, June 7 2013

This concept of operations details a “Divide and Defend” Firewall. Recently there has been a surge of malware attacks, those attacks typically being orchestrated through botnets. Modern enterprise and personal firewalls are not designed by default to handle massive number of attacks at once like botnets can incur. D&D is designed with that very thought in mind.

# introduction

TRAFFIC is a framework that allows for a minimal, possibly impossible, situation of dead lock or live lock. By using certain flags and timed mutexes, it is not possible to achieve a situation of permanent deadlock.

# AN ANOLOGY

Consider a city scape, with its rapid traveling vehicles and its slow moving trucks and buses. How is it that these vehicles don’t collide, slow down each other, or have a complete traffic jam at every instance. Well, we have developed a system for traffic. And this system is analogous to the problems we have when dealing with multi-threading.

Now consider a system where variables are embedded in streams. Streams are encapsulated in an entity container, and there are multiple streams flowing from here to there at once. This system I propose is similar to the first.

TRAFFIC, I propose, can prevent deadlock and livelock using the following structure. First, there is a global container that contains all of the data streams. Each stream contains a series of variables appropriate to its thread. The streams are similar to streets in a city and the variables in the streams similar to a vehicle. The final component is a system of flags which represent the stop-signs, traffic lights, and yield signs present in everyday driving. Each flag is a special type of mutex. The traffic light type flag is a multi-intersection mutex which is either timed or temporarily disabled until encountered. Upon encounter, it turns green. It then functions like a timed light until all traffic has cleared.

It is possible for two vehicles to reach a traffic light at the same time, this is similar to deadlock. In that case the traffic light, that is the mutex, reverts to a timed light.

Stop sign “flags” are a form of mutex where two or more sections of code alternate one after the other. Yield sign “flags” allow a steady flow of traffic from one side until exhausted, then the other process continues.

So is it possible, if these flags are implemented, for dead lock to occur? When the code reaches a flag the first thing that it does is push the address and variables to a stack, similar to a function call in assembly. It allows the code from the opposing side to execute, time passed and the mutex (flag) dissolve on its own. Once the code on the other side completes, it returns to the code on the stack and pushes the code on the opposing side.