CX 4010 Assignment 3

Sriram Iyer and Omar Wali

13 October 2017

Introduction

The simulation developed simulates the execution of discrete events by implementing a priority queue using a pointer-based heap implementation in the C programming language. The simulation engine is paired with a simulation application that schedules events and the simulation engine advances through the scheduled events at irregular intervals using the timestamps of the scheduled events.

API Description

The API between the simulation engine and application is very similar to the Sample Simulation code in that it includes the RunSim, Schedule, CurrentTime, and EventHandler functions. However, the functions urand and randexp were added to the API in order to allow the simulation application to be able to generate random numbers on a given distribution. These are the only changes of note to the API.

Simulation Application

The initial pseudocode for the simulation application is included in Figures 1 and 2. State variables included are the state of the bridge, the number of cars in the current group and the number of cars in the waiting group. The EventHandler is designed to handle two types of events: Entry and Exit events. Note that the twowaysimulation.c file included with this submission reflects a more sophisticated application model based on revised pseudocode. This updated pseudocode could not be prepared at the time of this report.

|  |
| --- |
|  |
| Figure 1. Preliminary Pseudocode for One-Way Traffic Simulation (1 of 2) |
|  |
| Figure 2. Preliminary Pseudocode for One-Way Traffic Simulation |

Unfortunately, the simulation application was not completed at the time of the writing of this report, so there are no tests to confirm the functionality of the application.

Simulation Engine

The engine for the Discrete Event Simulator uses a priority queue implemented as a binary heap to carry out a simulation. It contains a function to schedule new events and also provides functions to compute random numbers. The implementation of the future event list (FEL) is critical to the performance of the simulator because it performs in O(log n) time on functions that would have yielded O(n) performance given an array or list implementation. This allows us to run simulations on applications that produce a very large number of events efficiently.

The testing procedure for the simulation engine involved creating a simple tree such that all functions must be called. Creating a tree with ordered values would not call the heapify up function. Additionally, the test tree needed to be at least three levels, as a tree with only two represents edge cases of the functions algorithm. The stack trace screenshots shown in Figure 3 demonstrate a tree being built and subsequently queued.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Figure 3. Construction and queuing of tree demonstrating heap implementation. | | |

Results

Due to the failure of completion of the simulation application, no results were obtained for the integrated simulation environment.