Parallelism project

Traffic simulator

TUT	Department of pervasive computing	TIE-02500 Rinnakkaisuus
Authors Ville Juven<240035>Roope Vuorinen<220043>Atte Pellikka<195095>		Printed: 8.5.2014
Distribution: JykeSavia		
Document state: final		Modified: Roope Vuorinen 20:14

VERSION HISTORY

Version	Date	Author	Changes
0.1	May 08, 2014	RV	Initial doc
1.0	May 08, 2014	RV, VJ, AP	Final doc

TABLE OF CONTENTS

1. GETTING STARTED

First we had a face to face meeting with the three of us at Lintula. We discussed about the assignment and started to think about the coming architecture and modules for the project. We agreed to use C++ for the project and decided to use Git as our version control instead of SVN. We established an IRC channel where future communication would be held.

2. ARCHITECTURE

Our program consists of following classes:

Car:

The cars were made to be as stupid as possible. What a car does, it basically asks for a route, then follows it until the end.

Ferry:

The ferry is an independent thread that just goes from point L to point D and back.

Genericsignal:

This interface was created to avoid re-writing the mechanical code required for safe signaling with pthreads.

Genericthread:

This interface was created for easy thread management with pthreads.

Randomizer:

Threadsafe random.

Routemanager:

Generates routes once, provides a “route table” for cars.

Syncthread:

We decided to add a heartbeat signal to the program, this is an abstraction layer that combines threads and the heartbeat.

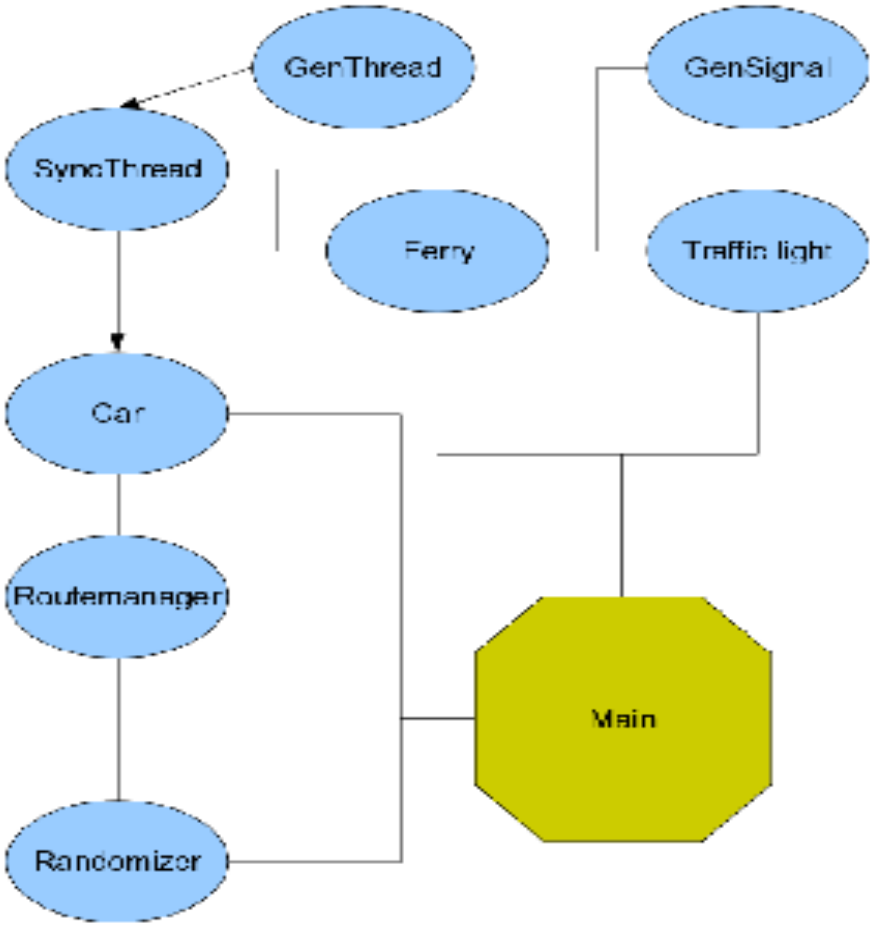
Trafficlight:

Single synchronization structure, that lets a single car through at a time.

We decided to put as much pthread specific mechanical code hidden in classes as we could.

We wanted to keep the mainfile simple and focus on building good classes. This also helped to minimize concurrent editing of same files, so the scope was clear most of the times who was doing what and why.

The main function acts as kind of an observer which implements a ticking mechanism that keeps the system running in sync (all threads like ferry and the cars wait for a heartbeat signal from main after they have finished their task for a tick). Main also keeps track of all the cars created and passed, including reserving and freeing the memory for them.



3. ANSWERING THE REQUIREMENTS

The simulation makes sure that not a single car will get stuck by keeping score how many cars have been spawned and how many cars have reached their final destination. When the car is spawned, it knows where it wants to go. The car also knows where its at; on road, on traffic lights, on ferry.

The traffic lights ensure their functions by letting only one car to the intersection at a time.

When the car first arrives at the traffic light zone, it asks the light that is it red or green, and acts accordingly. If the light was red, it waits until it changes. If it was green, it passes the light and continues to the intersection. At this point, the light is red for other cars. When the car clears the intersection, the light is green again.
From there the car goes either to the ferry, or the other endzone (C).
If the car wants to use the ferry, it first arrives at the waiting zone. The car then signals that its ready to board the ferry when the ferry is ready at the dock. There can be multiple cars at the waiting zone simultaneously. When the ferry arrives at the dock (L), it signals all the cars currently waiting to start loading, one at a time. When a car is told to board, the ferry waits for the boarding to complete, one car at a time. Before departing, the ferry makes a final check, that all cars actually made it onboard. After all cars are on board, the ferry starts its journey to (D). There, once it has arrived, it signals the cars on board that they are free to unload to the dock, one car at a time. From there the unloaded cars continue their way to the endzone. There the ferry also makes sure that all cars on board have left, and it continues its loop to go back and forth.

4. CONSENSUS

All in all, the exercise was very well suited for the course, and it taught us many aspects of threadsafety, how to make signals protected and what signals can be used as is. We also encountered a problem with pthreads itself; some of its methods are 'implementation defined', or 'platform dependant', meaning their functionality varies depending on the platform they are being used on. This caused an anomaly, where some developers got correct functionality, and some developers did not.

APPENDIX 1: ASSIGNMENT

To build a threadsafe environment to simulate cars passing by traffic lights and loading/unloading to a ferry.