|  |
| --- |
| Queensland University of Technology |
| INB365 Assignment 2 |
| Distributed Communication |
|  |
| **Dustin Wilson - n6325157**  **Chris Rhodan - n6862624** |
| **21/10/2014** |

|  |
| --- |
|  |

# Introduction

The task for this assessment was to develop a client/server system for dieticians. This system allows users to query statistics such as name, weight, fat etc for a variety of food items which is stored in a csv file that is retrieved by the server when run. The client program allows the user to both searches for food items, and to add new items to the database.

When searching for food items, the client sends a request to the Server through a socket, and the server performs the search, returning any matching results to the client through the same socket, which is then displayed on the console for the client program.

Similarly, when adding new food items, the client sends the new food item to the server, where it stored temporarily in the internal database. When the server is closed, this data is saved to the csv file so it can be reloaded by the server the next time it is run. Food items are stored in alphabetical order in this file.

This system has been done in a pair’s team of: Dustin Wilson and Chris Rhodan.

# Statement of Completeness

Pairs Team

Student1 Name: Dustin Wilson

Student1 Number: n6325157

Student2 Name: Chris Rhodan

Student2 Number: n6862624

To the best of our knowledge, all tasks have been completed. The only thing of note is that we did not include a makefile as we did not have a need to use one.

We were unable to get pthreads\_join working in a manner we were happy with so we are instead using pthread\_cancel, which is less than ideal but should not impact the behavior of our program due to the way it is implemented.

# Data Structures

For this simulator we used two data structures, a struct for the airplane in order to record its code and its time spent in its hanger. The second data structure was an array of 10 pointers to the aforementioned Airplane structs which formed the “hangar” of the airport.

As C is not an OOP language, a struct was used in order to keep track of multiple characteristics for the airplane. The struct for the Airplane was designed as follows, and contains a char\* which holds the (string) code for the airplane, and a time\_t which holds the time value of arrival in the landing bay.

struct Airplane{

char \*code;

time\_t parkTime;

};

The array of size 10 of pointers to Airplane structs was constructed as follows; it was designed to house pointer to each of the landed airplanes. Pointers were used as it allowed us the option to easily free up the data space when an airplane left the airport, by using the free() function.

struct Airplane \*airport[MAXIMUM\_AIRPORT\_CAPACITY];

# Threads

Four threads were used for the purposes of this assignment;

## Main thread

The main thread is run when the program begins and it initializes the data structures, converts the program arguments into global variables for the Arrival/Departure threads to access and then creates the other three threads for the program. It also is responsible for freeing memory and joining threads when the program is exited by the user.

## Arrival thread

The arrival thread calculates if a plane must be generated, and if it is, it then performs the steps necessary to create a new Airplane. It interacts with other threads mainly through the array of pointers mentioned earlier, whose access is controlled by semaphores, and through the runway which is mutex protected.

## Departure thread

The departure thread calculates if a plane will be departing the airport, and if it is, it then performs the steps necessary to select a random Airplane to depart. It interacts with other threads mainly through the array of pointers mentioned earlier, whose access is controlled by semaphores, and through the runway which is mutex protected.

## Monitor thread

The monitor thread waits for user input, and performs tasks based on what input is given. If a p or P is given, the current status of the Airport is printed, which interacts with other threads though the Airport “hangar” array mentioned earlier. If a q or Q is entered by the user, then it toggles the keep\_running flag, which tells the Arrival and Departure threads to break out of their core loop, ending all three threads and thus allowing the main thread to join the threads and free memory. If anything else is entered, an error message is given.