CMPUT 379, Assignment 3, Winter 2013

Due Monday, April 8 2013 (submit by 11:59pm)
(threads & synchronization & locking)
Note: submit early and submit often. Late assignments will not be accepted
Not knowing how to submit, or not being able to access eclass is not a valid reason for an extension.

Objective

In this assignment you will develop a multithreaded game, `saucer`. Multithreaded software developnent and locking is non-trivial. If you want to complete this assignment, you should start early.

You have been hired into your dream job, video game developer. Sophisticated internet investors have decided that terminal based shoot-em-ups are the next big thing, and you've been hired by a startup company with tons of venture capital developing what is sure to be the next big gaming sensation.

Alas you're not the first person to work here, the initial developer quit and left for greener pastures, and managed to throw away most of the code. All the company has to start from is a game description, below, some sample code, and a partially finished executable without source code. Your job is to start from scratch and make a working game!

The Game Description

In the game, the player must stop alien saucers (which look like this `<--->`) from crossing the sky (the screen) from left to right by positioning a launch site (which looks like this `|`) and firing rockets (which look like this `^`). The player will have a limited supply of rockets, and accuracy (hits on saucers) will be rewarded with additional rockets. Failing to stop the saucers will have dire consequences: if a predetermined number of saucers manage to cross the skyline from left to right without being destroyed, the game will end (because the aliens destroy your home world..)

Creating the game will give you extensive experience with POSIX threads, and importantly, the use of the synchronization functions. you will need to use them correctly to protect critical sections of your code. using them incorrectly could cause you to deadlock.

Getting Started

To implement the game, you will use `ncurses` to animate text in a login window. We start you with a sample program, available at http://bofh.srv.ualberta.ca/beck/c379/tanimate.c (This file was originally obtained from kent.edu, via grant macewan). This source code implements multithreaded animation of strings on the screen using ncurses - a necessary component of this assignment.

A partially completed executable (which is buggy, and lacks many desirable features) that was left by the departing developer is located in the lab machines in `~beck/saucer`. You can run this to get a feel for what the game can look like, although it lacks many necessary features and has bugs that may make it crash! Yours should be much better than this!

POSIX threads

You will need to use the posix threads synchronization functions fo this assignment. I suggest you review the synchronization examples and the Lawrence Livermore posix threads tutorial before you start designing a solution.

Threading requirement

Your program must

- use one thread per shot
- use one thread per saucer
- immediately recognize key presses; i.e. be highly responsive to user input

Design Documentation

Given the nature of working with threads, and the nature of this assignment you have some latitute in how you choose to make your game. Your design therefore needs to be documented. You must in your design documentation describe at a minimum:

- describe the overall structure of your code
- describe the threads used by your code and what their jobs are
- describe the critical sections, i.e. the sharing of data structures between your threads (which data are shared, who accesses them, and how they are protected)
- describe the user interface to your game - including your reasons for making particular decisions about it. Note - your TA should be able to read your design documentation and then know how to read your code and play the game.

Your design documentation should be *Less Than* 2000 words and should be submitted as a text file named `design.txt`, or a pdf formatted document named `design.pdf`.

Feature Checklist

As a baseline, your program should include certain functionality To help you meet that functionality consider the following things

For the initial state of your program:

- The user initially has a reasonably small number of rockets (define a constant)
- Saucers can only occupy the top few lines of the screen (define a constant)

- The user can position a launch site one line up from the bottom line of the screen.
- The bottom line of the screen must display status (score, rockets remaining, escaped saucers, etc. etc. etc)

For the saucers:

- At random intervals, a saucer may appear at the left of the screen in one of the saucer rows.
- Saucers must move from left to right.
- Multiple saucers can exist per row.
- Different saucers can move at different speeds.
- Saucers can pass each other on a row.
- When a saucer leaves the right side of the screen it "escapes"
- After a predetermined number of saucers escape the game is over.

For the rockets:

- Firing a rocket must decrease the available rockets
- When a rocket and saucer occupy the same location, both must disappear. *Exception: if multiple saucers occupy a location you can define the behavior, and document it*
- Destroying a saucer scores a point for the player
- Destroying a saucer rewards the player with additional rockets
- When the user runs out of rockets, the game is over.

To increase player enjoyment of the game, you might consider gradually increasing the rate at which saucers are generated, saving high scores to a file, or any other interesting features you can come up with. Be sure to document any such features in your design document.

Assumptions

Be sure to state any assumptions you make about the game, or the assignment, in your design documentation. If you encounter ambiguities, ask for clarification on eclass.

Marking

This assignment will be marked out of 100 marks.

- Your `saucer` implementation will be worth 75 marks
- Your Design Documentation will be worth 25 marks

Assignments that do not compile or run will receive 0 marks. Design documentation for a nonexistent implementation, or that does not match the implementation will receive 0 marks.

Deliverables

Your programs must be written exclusively in C. C++ idioms will be penalized. Consistent and readable C coding style is very important to spotting bugs and having your code readable by others. Your coding style should comply with the code style standards documented here. Some (but not all) important highlights from this which will be relevant for you are:

- indentation is an 8 space TAB
- NO line may be longer than 80 chars.
- secondary indentation (for long lines) is four spaces.
- Tab should be used in place of all 8 spaces. (i.e. do not indent a block 3 deep with 24 space characters - use three tabs)
- all functions are prototyped
- all variables are declared at the start of a block
- C style comments must be used (not C++ style)

If you find that the 8 space tab and 80 column width means you are constrained and end up with blocks indented to the far right of your screen, that is *actually the point* of this. If you are nested that deep, either your code logic is getting too complex and you're either barking up the wrong tree, or you should refactor some of it (like putting some stuff into a function, etc. This style may be painful at first but it will make you a better C programmer.

Note: as the sample code provided came from elsewhere - it might not actually match these standards. Your code you deliver is expected to match these standards. **you should probably fix the style in this file before using it**

You must deliver the code for your implementation, as well as a `Makefile` to build your implementation where the command `Make all` will build your saucer game. You must also deliver your design documentation (as `design.txt` or `design.pdf`).

The exact way of submitting your solutions will be explained by the TA's in the lab and on eclass. At this stage of your CS education, you are expected to deliver good quality code, which is easy to read and comprehend. The quality of your code will be marked.

****