Assignment 2 - Nordvik Suspension Bridge Problem

Aims

- Revise some C programming fundamentals, including pointers, file I/O, command line arguments and system calls.
- Develop a multi-threaded program that uses mutual exclusion and synchronization to solve a copmlex problem.
- To explore and implement a system of *semaphores*

The problem

This assignment is a variation on the readers and writers synchronization problem. The classic readers and writers problem is that although any number of readers should be able to read shared data at a time, a writer should have exclusive access. This classic problem is elegantly solved by binary semaphores:

```
shared semaphore count_lock

shared int readers_count initially 0
shared semaphore data_lock
shared data

reader algorithm:
lock count_lock readers_count++

if( readers_count == 1 )
  lock data_lock

unlock count_lock
read the data
lock count_lock readers_count--

if( readers_count == 0 )
  unlock data_lock

unlock count_lock
```

That is, the first reader attempting access gains access to the data on behalf of all readers that follow. After reading the data, the last reader out releases access to the data.

```
writer algorithm:
lock data_lock
write to the data
unlock data_lock
```

That is, a writer gets exclusive access to the data, locking out all other writers and readers. Using pthreads, the semaphores can be implemented simply as pthread mutexs. The readers and writers are threads executing different functions as the algorithms.

Now, the ***Nordvik Suspension Bridge*** problem:

The road to Nordvik carries traffic (cars and trucks) in both directions, except at a 1 lane suspension bridge. The bridge is only wide enough to carry traffic in one direction at a time. Of course, if cars entered the bridge from both directions at once, there would be a road block:

```
                    car3 car2 car1 -> <- carA carB carC ...
                             -----------------
                             suspension bridge
```

But it would be all right if more than one car was on the bridge travelling in the same direction:

```
                   car5 car4 car3 car2 car1 -> <- carA carB
                            -----------------
                            suspension bridge
```

So, once cars from one direction are on the bridge, no cars travelling in the other direction should enter. They will have to wait until the bridge is clear. Furthermore, engineers are a bit worried about weight on the bridge, so they have decided that if a truck is crossing the bridge, it should be the only vehicle (car or truck) on the bridge at a time.

Write a program in C that creates different numbers of threads representing cars and trucks crossing the bridge. Some from either direction. Their crossing of the bridge should be synchronized according to the rules of using the bridge described above.

Although the actual order of outputs can’t be guaranteed, the output of the program may look like:

```
$ nordvik
Car 0 going west on the bridge
Car 1 going west on the bridge
Car 0 going west off the bridge
Car 1 going west off the bridge
Truck 0 going east on the bridge
Truck 0 going east off the bridge
Car 2 going east on the bridge
Car 0 going east on the bridge
Car 1 going east on the bridge
Car 2 going east off the bridge
Car 0 going east off the bridge
Car 1 going east off the bridge
Truck 1 going west on the bridge
Truck 1 going west off the bridge
Truck 0 going west on the bridge
Truck 0 going west off the bridge
Truck 1 going east on the bridge
Truck 1 going east off the bridge
Car 2 going west on the bridge
Car 2 going west off the bridge
$
```

Hints:

- This is a variation on the readers and writers synchonization problem.
- In order to get more variety of output, I had threads execute

```
sleep( rand() % MAXWAIT );
```

at the start of their function, with define `MAXWAIT 20`

- Also, I had threads execute:

```
sleep( CROSSINGTIME );
```

while on the bridge, with `#define CROSSINGTIME 4`

Tentative Marking Guide

item	Marks
The makefile	....
targets	[/1]
uses -Wall	[/1]
compiles without warnings etc.	[/1]
The Program	....
Correct Output	[/7]
Correct Access protocol	[/7]
Command line error checking	[/2]
Checks for system call failures	[/2]
Doesn't use hardwired constants	[/2]
Consistent Use of Style	[/1]
Correct number of children	[/1]
Total	25 Marks