COSC 3360/6310-Operating System Fundamentals

Assignment #3: The Bridge

(Due on Monday, May 1, 2017 at 11:59 pm)

OBJECTIVES

This rather short assignment should teach you how to use pthreads, pthread mutexes and pthread condition variables.

THE PROBLEM

A bridge is load-zoned and can only carry a total weight of **`maxweight`** tons. Your solution should ensure that the total weight of all vehicles on the bridge will never exceed its maximum load. Each vehicle will be represented by a separate thread and all inter-thread synchronization must be implemented through pthread mutexes and pthread condition variables. ***You cannot use semaphores***.

Vehicles whose total loaded weight exceeds the maximum load of the bridge should be rejected and a descriptive message containing the serial number of the vehicle printed out.

YOUR PROGRAM

Your program should consist of:

1. A main program,
2. An **`enterBridge(weight)`** method to be performed by each arriving vehicle
3. A **`leaveBridge(weight)`** method to be performed by each vehicle leaving the bridge, and
4. One child thread per vehicle.

The **`maxweight`** constant should be read from the command line as in:

```
./a.out 10
```

All other parameters will be read from the standard input. Each input line will describe one vehicle arriving at the bridge and will contain four parameters:

1. An alphanumeric string without spaces representing the car license plate,
2. A positive integer representing the number of seconds elapsed since the arrival of the previous vehicle (it will be equal to zero for the first vehicle arriving to the bridge);
3. A positive integer representing the total loaded weight of the vehicle rounded up to an integer number of tons,
4. A positive integer representing the number of seconds the vehicle will take to cross the bridge.

A possible set of input could be:

```
HIOFCR   0   1  10
STOL3N   3  10  20
SHKSPR   8   1  30
2DIE4    7   1   5
BYOFCR   2   1  15
```

Your main program should read the input line per line, wait for the appropriate amount of time and fork a different child process for each incoming vehicle. It should print out a descriptive message including the vehicle license number ***and the current bridge load*** every time a vehicle:

1. Arrives at the bridge,
2. Starts crossing the bridge, and
3. Leaves the bridge.

The current bridge load when a vehicle starts crossing the bridge includes the weight of that vehicle but the current bridge load after a vehicle leaves the bridge does not.

HINTS

- Create your mutexes and your condition variables in your main program before you fork any child thread.

This document was updated last on Thursday, April 06, 2017

PTHREADS

1. Don't forget the pthread include:

   ```
   #include <pthread.h>
   ```

2. All variables that will be shared by all threads must be declared `static` as in:

   ```
   static int bridgeLoad;
   ```

3. If you want to pass an integer value to your thread function, you should declare it as in:

   ```
   void *vehicle(void *arg) {
       int serial;
       serial = (int) arg;
       …
   } // vehicle
   ```

   If your compiler rejects the cast of a `void` into an `int` as a fatal error, you must use the flag `-fpermissive`.

4. To start a thread that will execute the customer function and pass to it an integer value use:

```
pthread_t tid;
int i;
…
pthread_create(&tid, NULL,
       vehicle, (void *) i);
```

5. To terminate a thread, use:

```
pthread_exit((void*) 0);
```

6. To wait for the completion of a specific thread, use:

```
pthread_join(tid, NULL);
```

Note that the pthread library has no way to let you wait for an unspecified thread and do the equivalent of:

```
for (i = 0; i < nchildren; i++)
    wait(0);
```

Your main thread will have to keep track of the thread id's of all the threads of all the threads it has created:

```
pthread_t child[maxchildren];
for (i = 0; i < nchildren; i++)
    pthread_join(child[i], NULL);
```

PTHREAD MUTEXES

1. To be accessible from all threads pthread mutexes must be declared `static`:

```
static pthread_mutex_t access;
```

2. To create a mutex use:

```
pthread_mutex_init(&access, NULL);
```

Your mutex will be automatically initialized to one.

3. To acquire the lock for a given resource, do:

```
pthread_mutex_lock(&access);
```

4. To release your lock on the resource, do:

```
pthread_mutex_unlock(&access);
```

PTHREAD CONDITION VARIABLES

1. The easiest way to create a condition variable is:

```
static pthread_cond_t ok =
    PTHREAD_COND_INITIALIZER;
```

2. Your condition waits must be preceded by a successful lock request on the mutex that will be passed to the wait:

```
pthread_mutex_lock(&access);
while (carweight > …)
    pthread_cond_wait(&ok, &access);
…
pthread_mutex_unlock(&access);
```

3. To avoid unpredictable scheduling behavior, the thread calling `pthread_cond_signal()` must ***own*** the mutex that the thread calling `pthread_cond_wait()` had specified in its call:

```
pthread_mutex_lock(&access);
…
pthread_cond_signal(&ok);
pthread_mutex_unlock(&access);
```

All programs passing arguments to a thread must be compiled with the `-fpermissive` ***flag. Without it a cast from a*** `void` ***to anything else will be flagged as an error by some compilers.***