COSC 6360-Operating Systems

Assignment #3: The Tunnel

(Due on Tuesday, November 26, 2019 at 11:59 pm)

OBJECTIVES

This rather short assignment should teach you how to use pthreads, pthread mutexes and pthread condition variables. ***Your program cannot use semaphores***.

THE PROBLEM

The authorities managing a road tunnel that suffers from ventilation issues have decided to limit the number of cars and trucks that can be in tunnel at the same time. They have decided to classify incoming vehicles into three groups:

1. Cars and light trucks using an internal combustion engine,
2. Larger vehicles using an internal combustion engine,
3. Electric vehicles.

They have arbitrarily assumed that vehicles in the group 2 pollute five times more than conventional car and light trucks while electric vehicles do not pollute at all. As a result, a vehicle of the group two will count as five vehicles of the group 1 and electric vehicles will always be allowed in the tunnel.

If the maximum load of the tunnel is below five cars equivalents, all vehicles in the group 2 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 **`enterTunnel(groupNo)`** method to be performed by each arriving vehicle
3. A **`leaveTunnel(groupNo)`** method to be performed by each vehicle exiting the tunnel, and
4. One child thread per vehicle.

All parameters will be read from the standard input. The first input line will contain a single number specifying the maximum number of cars and light trucks equivalents that can be at the same time in the tunnel. The following input lines will each describe a vehicle arriving at one entrance of the tunnel and will contain four parameters:

1. An alphanumeric string without spaces representing the vehicle license plate,
2. A positive integer describing the type of vehicle, that is, group 1, 2, or 3.
3. 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 at an entrance of the tunnel);
4. A positive integer representing the number of seconds the vehicle will take to go through the tunnel.

You can safely assume that all inputs will always be correct. A possible set of input could be:

```
4
HIOFCR  1  0    10
STOL3N  2  3    20
SHKSPR  1 8   30
2DIE4   3 7    5
BYOFCR  1 2    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 vehicle group every time a vehicle:

1. Arrives at the tunnel entrance,
2. Enters the tunnel, and
3. Exits the tunnel.

HINTS

- Create your mutexes and your condition variables in your main program before you fork any child thread.
- Use the **`sleep()`** function to represent delays.
- The best tutorial on pthreads is
  - Blaise Barney, POSIX Threads Programming, https://computing.llnl.gov/tutorials/pthreads/

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 carEquivalents;
   ```

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. If you want to pass an more than one value to your thread function, you should put them in a **struct** as in:

```
struct VehicleDetails {
    char license[8];
    int group;  // 1,2, or 3
    int crossingTime;
}

struct VehicleDetails thisVehicle;
```

Here again you might have to use the **-fpermissive** flag.

5. 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);
```

6. To terminate a thread, use:

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

7. 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 < nVehicles; 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 (vehicleEquivalent > …)
    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 most compilers.***