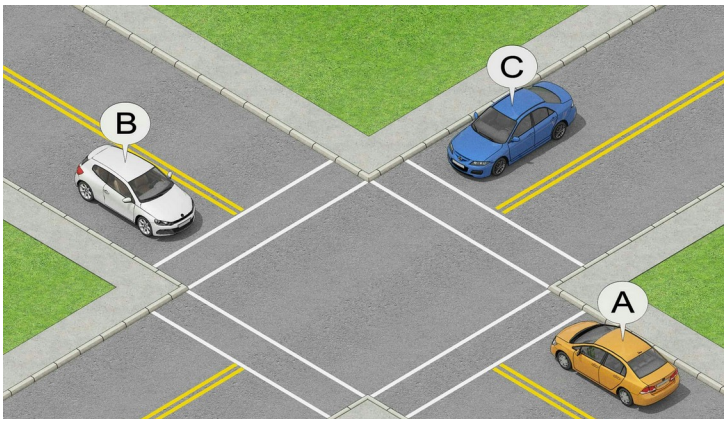
CS 370 – Project 5B, Semaphores

Purpose: Become familiar with multi-threaded programming, semaphores, and deadlock concepts
Points: 200 100 for program and 100 for write-up
Scoring will include functionality, documentation, coding style, and write-up

Introduction

Traffic Light Simulation Project → This project involves creating a model of a traffic intersection where multiple roads converge. Each road has a traffic light that controls the flow of vehicles (represented by threads) across the intersection.

The objective of this project is to use semaphores to manage the timing and sequencing of the traffic lights, ensuring that traffic flows smoothly and safely without collisions or deadlocks.



If I make a C joke, will you get the reference? Or will you need pointers?

Intersection → We will simulate a typical four-way intersection (see diagram). Each road will have a traffic light controlling the entry of vehicles into the intersection. Each traffic light is represented by a semaphore. The semaphore controls whether vehicles on a particular road can proceed (green light) or must wait (red light). Each road can have multiple lanes, and vehicles on each lane will be simulated as individual threads. The intersection can be a simple 4-way crossing or a more complex roundabout, depending on the complexity desired.

Vehicles → Each vehicle will be represented via a thread that simulates the behavior of a car approaching the intersection, waiting for the light to turn green, and then proceeding through the intersection. Vehicles will approach the intersection at random intervals, which will be simulated using sleep functions with random durations. When a vehicle reaches the intersection, it checks the status of the semaphore controlling its traffic light.

Traffic Light States → Implement states for each traffic light (Green, Yellow, Red). The state changes are controlled by semaphores, where:

Green: Semaphore is released, allowing vehicles to pass.

Yellow: After the green light phase, the yellow light phase is introduced for ***n*** seconds. During this phase, no new vehicles should start crossing the intersection, but vehicles already crossing should clear the intersection. The yellow light does not require a semaphore change because the semaphore controls the green/red light, and yellow is simply a transitional phase.

Red: Semaphore is acquired, blocking vehicles from passing.

Traffic Light Timing → Implement a timing mechanism that changes the state of each traffic light based on real-world traffic light timings. For example, a green light might last for 30 seconds, followed by a yellow light for 5 seconds, and then a red light.

Synchronization → Use semaphores to synchronize the traffic lights, ensuring that conflicting directions (e.g., vehicles going straight and vehicles turning left) do not have green lights at the same time.

There are many variations for this problem, so be sure to follow the provided guidance. Solutions downloaded from the internet will be referred to the Office of Student Rights and Responsibilities. Implement this traffic light synchronization problem in **C** (not C++). The program must compile and execute under Ubuntu 22.04 LTS (and will only be tested with Ubuntu). Assignments not executing under Ubuntu will not be scored. See additional guidance on following pages.

Approach Overview

The ***main()*** function should perform the following actions:

- Read and validate command line argument (in a function)
  - Count of in the simulation (-vc <integerValue>)
    - Must be between VEHICLE_CNT_MIN and VEHICLE_CNT_MAX
  - Must be done in a function (not the main)
- Display headers (see example)
- Initializations
  - Initialize semaphores
    - Dynamically create semaphore array (4)
    - All lights should be initially set to red (0)
  - Create array for threads (one per vehicle)
    - The array must be dynamically allocated (i.e., use **`malloc()`**).
- Create traffic light controller thread
- Create ***vehicleCount*** threads and pass a random direction (NORTH, EAST, SOUTH, WEST) to the thread (using **`rand()%4`**).
- Wait for the vehicle threads to complete.
- Cancel the light controller thread (**`pthread_cancel()`**) and join the light controller thread.
- Destroy the semaphores and free the dynamically allocated arrays.
- Display final completion message (see example).

The traffic light controller thread function will perform the following actions

- Loop forever
  - Print green light message for north-south direction
  - Turn north-south lights green-yellow (via semaphore)
  - Keep light green for LIGHT_WAIT seconds (via sleep)
  - Print yellow light message for north-south warning
  - Wait for additional YELLOW_WAIT seconds (via sleep)
  - Turn light red for north-south direction
  - Print green light message for east-west direction
  - Turn north-south lights east-west (via semaphore)
  - Keep light green for LIGHT_WAIT seconds (via sleep)
  - Print yellow light message for east-west warning
  - Wait for additional YELLOW_WAIT seconds (via sleep)
  - Turn light red for east-west direction

The vehicle thread function should perform the following actions.

- Print approaching message (including direction)
- Wait for light to be green (via semaphore)
- Print passing through message (including direction)
- Wait for vehicle to pass through the intersection
  - Simulated via: **`sleep(TRANSIT_TIME + usleep(rand()%5000));`**
- Release, light turns red (via semaphore)
- End thread

The messages appropriate traffic light messages should be printed in green, yellow, and red or include the initial symbol. The below examples should be useful.

```
printf("\033[0;4mCS 370 Project #5B -> Traffic Light ",
        "Simulation Project\033[0m\n");
printf("\033[0;92mGreen light for North-South\033[0m\n");
printf("\033[0;93mYellow light for North-South\033[0m\n");
printf("\033[0;31mRed light for North-South\033[0m\n");
printf("\u2195 Vehicle approaching from %s\n", dir[direction]);
printf("\u2194 Vehicle passing through from %s\n", dir[direction]);
```

Refer to the example execution for examples of the coloring.

Include Files

In order to use threading functions and semaphores in C, you will need the below include files:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <pthread.h>
#include <semaphore.h>
#include <stdbool.h>
#include <errno.h>                // for errno
```

Use the following parameters

```
const unsigned int    VEHICLE_CNT_MIN = 10;
const unsigned int    VEHICLE_CNT_MAX = 500;

const unsigned int    LIGHT_WAIT = 4;
const unsigned int    YELLOW_WAIT = 1;
const unsigned int    TRANSIT_TIME_BASE = 1;
const unsigned int    ARRIVAL_INTERVAL = 2;

const unsigned int    NORTH = 0;
const unsigned int    EAST = 1;
const unsigned int    SOUTH = 2;
const unsigned int    WEST = 3;
```

The semaphores and parameters will be declared globally.

Compilation Options

Use the following compiler options

```
gcc -Wall -pedantic -pthread -o traffic traffic.c
```

Points will be removed for poor coding practices and unresolved warning messages.

Results Write-Up

When the program is working, prepare a write-up summarizing the results and answer the following questions:

- Explain why the example results (below) either match or do not match your results.
- For this specific problem, what is the primary purpose of using semaphores?
- For this specific problem, what is a semaphore's role in synchronization between the light controller and the vehicle threads?
- Describe what happens when the light is green and there are no vehicles.
- Describe what happens when the light is red and there are many vehicles.

Example Execution

Below are some example executions for reference.

```
ed-vm% ./traffic
Usage: ./traffic -vc <vehicleCount>
ed-vm%
ed-vm% ./traffic vc 10
Error, invalid item count specifier.
ed-vm%
ed-vm% ./traffic -vc 9
Error, vehicle count value out of range.
ed-vm%
ed-vm% ./traffic -vc 501
Error, vehicle count value out of range.
ed-vm%
ed-vm% ./traffic -vc 1f
Error, invalid vehicle count value.
ed-vm%
ed-vm%
ed-vm% ./traffic -vc 15
CS 370 Project #5B -> Traffic Light Simulation Project
  Vehicles: 15

Green light for North-South
↕ Vehicle approaching from East
↕ Vehicle approaching from West
Yellow light for North-South
↕ Vehicle approaching from South
↔ Vehicle passing through from South
Red light for North-South
Green light for East-West
↔ Vehicle passing through from East
↔ Vehicle passing through from West
↕ Vehicle approaching from West
↔ Vehicle passing through from West
↕ Vehicle approaching from West
↔ Vehicle passing through from West
Yellow light for East-West
```

```
↕ Vehicle approaching from West
↔ Vehicle passing through from West
Red light for East-West
Green light for North-South
↕ Vehicle approaching from North
↔ Vehicle passing through from North
↕ Vehicle approaching from East
Yellow light for North-South
↕ Vehicle approaching from East
Red light for North-South
Green light for East-West
↔ Vehicle passing through from East
↔ Vehicle passing through from East
↕ Vehicle approaching from West
↔ Vehicle passing through from West
↕ Vehicle approaching from West
↔ Vehicle passing through from West
Yellow light for East-West
Red light for East-West
Green light for North-South
↕ Vehicle approaching from North
↔ Vehicle passing through from North
↕ Vehicle approaching from South
↔ Vehicle passing through from South
Yellow light for North-South
↕ Vehicle approaching from North
↔ Vehicle passing through from North
Red light for North-South
Green light for East-West
↕ Vehicle approaching from North
Yellow light for East-West
Red light for East-West
Green light for North-South
↔ Vehicle passing through from North

All vehicles successfully passed through the intersection.
ed-vm%
ed-vm%
```

Note, actual mileage may vary.

Submission

When complete, submit:

- A copy of the final working **C** source code (not C++) file.
- A PDF of the write-up (as described above).

Submissions received after the due date/time will not be accepted.