Final Project Part I
Synchronization at San Diego Zoo

Project Description:

The San Diego zoo consists of an indoor snake exhibit and a great park full of exotic animals. There are *M* visitors and *N* single-passenger cars at the zoo at any given time. Visitors typically wander around the indoor snake exhibit for a while, then line up to take a ride in a car to see the rest of the zoo. When a car is available, it loads the one visitor it can hold and rides around for a random amount of time, say T. If the *N* cars are out riding other visitors around, then a visitor who wants to ride waits. If a car is ready to load but there are no waiting visitors, then the car waits. The zoo also has a small gas station that has K gas pumps to fill the cars whenever they need gas. When a car comes in, if there is a free pump, the car goes and fills its gas tank. If the car comes in and there is no free pump, then the car waits. Sometimes the gas truck comes in to refill the pumps. If the gas truck is refilling, then all cars should wait and not use the pump. If there are cars at the pumps or waiting in line, then the truck has to wait for its turn. The cars that came before the gas truck can take their turns to go to the pump and the cars that came after the gas truck have to wait till the gas truck finishes refilling the pumps. Write a program to synchronize the visitor, the car, and the gas station processes/threads using mutex and semaphore.

Project Requirements:

You may think of a solution in terms of a two-fold producer-consumer problem implemented with appropriate semaphores considering the following:

- Create a data file and read the M, N, T, and K from this file. The file must have the following format:

  There will be multiple sets of data. Each run will be 4 numbers (M,N,T,K) in one line separated by ",". If the first number is a 0 then there are no more sets of data. Example:

  ```
  35,6,5,2
  19,4,10,3
  0,0,0,0
  ```

  Feel free to add additional rows to this data file and simulate all possible situations. The instructor may use her own set of number to test.

- Assume all cars initially are full of gas and after 5 rides they need to refill

- Assume the gas station is initially full and can fill at most 200 cars (among all its K pumps) before it needs to be refilled by a gas truck. The gas station doesn't have to be completely empty before it is refilled. The gas station attendant will call for a gas truck when at least 150 cars have been filled.

- Assume the *upper bound* on M is 100 visitors, N is 10 cars, K is 3 pumps, and T is 5 units of time.

- Assume it takes 3 units of time to fill a car with gas and 15 units to refill the gas station by a gas truck.

- The car process *produces* rides, the visitor process *consumes* rides
- The gas station process *produces* gas, the car process *consumes* gas

To simplify the problem, use only 4 true processes/threads and display the status of each process/thread on the screen. The four processes/threads are:

- Master: starts a new run, terminates the running, and does other house keeping activities.
- Visitor Agent: assist visitors to consume car rides (each has its fair turn). The display shows the number of visitors waiting for a car.
- Car Agent: produces car rides to be consumed by the visitors and evaluates when gas is needed. Car agent generates Cars on schedule and sends them for fuel at proper intervals. The display shows the number of cars out with riders.
- Gas Station: manages gas station activities and allows production of gas by trucks and consumption by cars (each has its fair turn) and subject to rules mentioned above (i.e. car to go before or after a gas truck, etc.). The display shows number of cars at pumps, number of cars waiting for pumps, presence of fuel trucks.

Your program must report any cases of process starvation or other anomalies that may occur.

Submissions:

- Well-written and well-commented source code
- A report that:
    - contains a diagram describing how the processes/threads communicate/synchronize with each other. Processes/threads, mutexes and semphores, and shared resources should be clearly shown in your diagram.
    - describes how to compile and run your program
    - lists test cases that cover all possible situations
    - includes screenshots of your running program
- Demo your code to the instructor before/on the last day of class