O.S. Lab 2: POSIX Threads and Semaphores

This assignment involves simulating a problem involving mutual exclusion. In the course of solving this problem, you will implement threads and semaphores in C++. When finished submit a copy of your code in a zipped folder to brightspace.

Many amusement parks in the United States have bumper car rides. Each car holds one person. People who want to ride a bumper car line up and wait for a free car. When a car is returned and the previous rider gets out, a new rider gets in the car from the head of the waiting line and rides around on the bumper ride floor, bumping into the other cars. After riding around for a while, the rider in the bumper car decides that enough is enough and returns the car to the area where new rides line up. A new rider gets in and takes a ride in the bumper car. The rider who just got out of a bumper car wanders around the amusement park, perhaps getting something to eat. After wandering for a while, the rider decides to take another bumper car ride, returns to the rider waiting area, and gets in line. If there are more free cars that people wishing to ride, any free car can take the rider in the head of the waiting line.

There are N_{riders} rider threads and N_{cars} bumper car threads. Riders are identified by *rider IDs* (*rid*) from 1 to N_{riders}. Bumper cars are identified by *car IDs* (*cid*) from 1 to N_{cars}. Each car takes a (different) random amount of time between 0 to T_{bump} seconds to bump around the floor (simulated with the Bump() function) each time a rider gets into the car. The bump time is a random number regenerated each time a car goes onto the bumper floor. After getting a ride, each rider wanders around the amusement park for a random amount of time between 0 to T_{wander} seconds before returning to the waiting area to get in line for another ride. The wander time is a random number regenerated each time a rider leaves a car and goes into the amusement park.

Develop code for a multi-thread program simulating the rider and the car threads. Use semaphores for synchronization. Run the simulation for *COUNT_DOWN* number of total bumper car rides. Each of the N_{riders} is simulated with a rider thread, identified by the input parameter *rid*. Each bumper car is simulated with a car thread, identified by the input *cid*. Finally, there is another *displaying thread* which is active all the time printing out the current situation in the amusement park, for example, “Who is taking which car?”, “Who is wandering around the park?”, and “Who is in the waiting line?”, etc. The displaying thread is just for easy debugging for you and easy testing for the TA.

The waiting area where riders get in lines is implemented with a bounded buffer with the size at least N_{riders}. Each rider who finishes wandering around the amusement park gets in line by depositing its *rid* into the buffer and then waits on its slot in an array of semaphores, P(WaitForRideBegin[rid]). When a bumper car is ready to pick up a rider, it fetches an *rid* from the head of the bounded buffer and execute V(WaitForRideBegin[rid]). This releases the rider from the waiting line so that it can take a seat. After taking the seat, the rider executes P(WaitForRideOver[rid]), blocking itself until the ride is over. When the bumper car's bumping time is over, the car thread executes V(WaitForRideOver[rid]) so the rider can wander around the park again. Make sure you have eliminated all the race conditions by protecting the critical sections in other semaphores.

In order to solve this part, you need to develop the codes to simulate the behavior of the cars and the riders. Here is one way to organize the program, as shown below in the pseudo code (*may not be correct*). Each rider thread contains an instance of the rider function Rider(int rid), each with an unique *rid*. Each car thread contains an instance of the car function Car(int cid), each with an unique *cid*. Each time a rider wants to get in line, it calls the GetInLine(rid) function. When it returns, the rider's *rid* has been placed into the bounded buffer. The rider then calls the TakeASeat(rid) to block itself in the corresponding slot of WaitForRideBegin[] semaphore to wait for a car to pick it up. After the rider gets in the car, it calls TakeARide(rid) to wait for the ride over. When the car is ready to load a rider, it calls the function Load(cid) to load the rider in the head of the waiting line. When the ride is over, the car calls the function Unload(cid) to free the rider.

Note that the pseudo code given below is not completely correct. It seems that occasionally a rider in the head of the waiting line is taken at the same time by two or more cars, which is impossible in the real world. Nonetheless, use this as a starting point for writing your simulation of the problem.

Last updated 2.18.2022 by T. O'Neil. Based on a project by E. H.-M. Sha.

```
#define N_CARS 2
#define N_RIDERS 5
#define T_WANDER 100   /* each wandering time is between 0 to T_WANDER */
#define T_BUMP 40      /* each bumping time is between 0 to T_BUMP */

int COUNT_DOWN = 10;   /* Sim time: Total number of bumper car rides */

int finish() {
  if (COUNT_DOWN==0) return TRUE; else return FALSE;
}

/* Rider thread. rid is a number between 1 to N_RIDERS.  */
void Rider(int rid) {
   while (TRUE)
     { /* wander around for a random amount of time */
       Wander(rid, random()%T_WANDER);
       GetInLine(rid);
       TakeASeat(rid);
       TakeARide(rid);
       ...    /* check for the condition to exit the while loop here */
     }
}

void GetInLine(int rid) {
   ...
   printf("Rider %d gets in the waiting line.\n", rid);
}

void TakeASeat(int rid) {
   ...
}

void TakeARide(int rid) {
   printf("Rider %d is taking the ride.\n", rid);
   ...
}

void Wander(int rid, int interval) {
   printf("Rider %d is wandering around the park.\n", rid);
   ...
}
```

```
/* Bumper car thread. cid is a number between 1 to N_CARS. */
void Car(int cid) {
  while (TRUE) {
       Load(cid);
       Bump(cid, random()%T_BUMP);
       Unload(cid);
       ...  /* decrease the COUNT_DOWN */
       ...  /* check for the condition to exit the while loop */
    }
}

void Load(int cid) {
   ...
   this_guy=WaitArea[LineHead];
   LineHead++;
   printf("Car %d takes the rider %d.\n", cid, this_guy);
   ...
}

void Unload(int cid) {
   ...
   printf("This ride of Car %d is over.\n");
   ...
}

/* Displaying thread */
void Display(int dummy) {
  while (!finish())
  {
   printf("The current situation in the park is:\n");
   for (i=1 to N_CARS)
      if (Car[i] is running)
         print "Car i is running. The rider is ???"
      else
         print "Car i is not running."

   for (i=1 to N_RIDERS)
      if (Rider[i] is wandering)
         print "Rider i is wandering"
      else if (Rider[i] is waiting in line)
             print "Rider i is waiting in line"
           else
             print "Rider i is in a car."
  }
}
```