Synchronization / Resource allocation

Exercise - underlying approach useful for Lab3

Operating Systems
Chalmers University of Technology and Gothenburg University

Narrow Bridge problem

A two way east-west road contains a narrow bridge with only one lane. An eastbound (or westbound) car can pass over the bridge only if there is no oncoming car on the bridge. Traffic may only cross the bridge in one direction at a time, and if there are ever more than 3 vehicles on the bridge at one time, it will collapse under their weight. In this system, each car is represented by one thread, which executes the procedure OneVehicle when it arrives at the bridge.

Narrow Bridge problem

A two way east-west road contains a narrow bridge with only one lane. An eastbound (or westbound) car can pass over the bridge only if there is no oncoming car on the bridge. Traffic may only cross the bridge in one direction at a time, and if there are ever more than 3 vehicles on the bridge at one time, it will collapse under their weight. In this system, each car is represented by one thread, which executes the procedure OneVehicle when it arrives at the bridge.

```
OneVehicle(Direction direc) {
    ArriveBridge(direc);
    CrossBridge(direc);
    ExitBridge(direc); }
```

direc gives the direction in which the vehicle will cross the bridge

Write the procedures *ArriveBridge* and *ExitBridge*.

ArriveBridge must not return until it safe for the car to cross the bridge in the given direction (it must guarantee that there will be no head-on collisions or bridge collapses).

ExitBridge is called to indicate that the caller has finished crossing the bridge; *ExitBridge* should take steps to let additional cars cross the bridge.

Solution using Condition Variables [*]

Correctness constraints

1. At most 3 cars are on the bridge at a time
2. All cars on the bridge go in the same direction
3. Whenever the bridge is empty and a car is waiting, that car should get on the bridge
4. Whenever the bridge is not empty or full and a car is waiting to go the same direction as the cars on the bridge, that car should get on the bridge
5. Only one thread accesses shared state at a time

Approach:

- Cars will be waiting to get on the bridge, but in two directions. Use an array of two condition variables, `waitingToGo[2]` and an associated lock `l`
- It is useful to know the number of cars on the bridge (`cars`, initially 0), and the direction of these cars if there are any (call it `CurrentDir`).
- It is also useful to know the number of cars waiting to go in each direction; use an array `waiters[2]`.

[*] check also https://inst.eecs.berkeley.edu/~cs162/fa13/hand-outs/synch-problems.html
https://inst.eecs.berkeley.edu/~cs162/fa13/hand-outs/synch-solutions.html

Solution using Condition Variables Cont.

(PintOS-like syntax(*))

```
ArriveBridge (direction) {
  lock_acquire(&l);
  while ((cars == 3)  || (cars > 0 && CurrentDir != direction)) { // while can't get on the bridge, wait
    waiters[direction]++;
    cond_wait(&waitingToGo[direction], &l);
    waiters[direction]--;
  }
  cars++; // get on the bridge
  CurrentDir = direction;
  lock_release(&l);
}
```

(*) Note: after signal, signaling thread keeps lock, waking thread goes on the queue waiting for the lock.

Check also for perspective:

- Condition Variables slides on this course
- PintOS API-examplesexamples in Lecture Notes by J.Ousterhout https://web.stanford.edu/~ouster/cgi-bin/cs140-spring14/lecture.php?topic=locks
- `Ptreads` API examples eg here https://pages.cs.wisc.edu/~remzi/OSTEP/threads-cv.pdf

Solution using Condition Variables Cont.

(PintOS-like syntax(*))

```
ArriveBridge (direction) {
  lock_acquire(&l);
  while ((cars == 3)  || (cars > 0 && CurrentDir != direction)) { // while can't get on the bridge, wait
    waiters[direction]++;
    cond_wait(&waitingToGo[direction], &l);
    waiters[direction]--;
  }
  cars++; // get on the bridge
  CurrentDir = direction;
  lock_release(&l);
}
```

```
ExitBridge (direction) {
  lock_acquire(&l);
  cars--; // get off the bridge
  if (waiters[CurrentDir] > 0)  // if anybody wants to go the same direction, wake them
    cond_signal(&waitingToGo[CurrentDir], &l);
  else if (cars == 0) // else if empty, try to wake somebody going the other way
    cond_broadcast(&waitingToGo[1-CurrentDir], &l);
  lock_release(&l);
}
```

(*) Note: after signal, signaling thread keeps lock, waking thread goes on the queue waiting for the lock.

Check also for perspective:

- Condition Variables slides on this course
- PintOS API-examplesexamples in Lecture Notes by J.Ousterhout https://web.stanford.edu/~ouster/cgi-bin/cs140-spring14/lecture.php?topic=locks
- Ptreads API examples eg here https://pages.cs.wisc.edu/~remzi/OSTEP/threads-cv.pdf

Solution using Condition Variables Cont.

(PintOS-like syntax[*])

```
ArriveBridge (direction) {
  lock_acquire(&l);
  while ((cars == 3)  || (cars > 0 && CurrentDir != direction)) { // while can't get on the bridge, wait
    waiters[direction]++;
    cond_wait(&waitingToGo[direction], &l);
    waiters[direction]--;
  }
  cars++; // get on the bridge
  CurrentDir = direction;
  lock_release(&l);
}
```

```
ExitBridge (direction) {
  lock_acquire(&l);
  cars--; // get off the bridge
  if (waiters[CurrentDir] > 0)  // if anybody wants to go the same direction, wake them
    cond_signal(&waitingToGo[CurrentDir], &l);
  else if (cars == 0) // else if empty, try to wake somebody going the other way
    cond_broadcast(&waitingToGo[1-CurrentDir], &l);
  lock_release(&l);
}
```

[*] Note: after signal, signaling thread keeps lock, waking thread goes on the queue waiting for the lock.

Check also for perspective:

- Condition Variables slides on this course
- PintOS API-examplesexamples in Lecture Notes by J.Ousterhout https://web.stanford.edu/~ouster/cgi-bin/cs140-spring14/lecture.php?topic=locks
- `Ptreads` API examples eg here https://pages.cs.wisc.edu/~remzi/OSTEP/threads-cv.pdf