Classic Synchronization Problems

Tom Kelliher, CS 311

Mar. 26, 2012

1 Administrivia

Announcements

Assignment

From Last Time

Software solutions to the CS problem.

Outline

1. Dining philosophers.
2. Second readers/writers problem.
3. Sleepy barber problem.

Coming Up

Deadlock.

2 Classic Synchronization Problems

2.1 Dining Philosophers

Five philosophers, five chopsticks, five chairs, a table, a bowl of noodles.

Some "solutions:"

```
while (1)
{
   think;
   pick up left chopstick;
   pick up right chopstick;
   eat;
   put down left chopstick;
   put down right chopstick;
}
```

```
while (1)
{
   think;
   while (haven't obtained both chopsticks)
   {
      pick up left chopstick;
      if (right chopstick is available)
         pick up right chopstick;
      else
         put down left chopstick;
   }
   eat;
   put down left chopstick;
   put down right chopstick;
}
```

```
while (1)
{
   think;
   while (haven't obtained both chopsticks)
   {
      if (both chopsticks available)
```

```
      {
         pick up left chopstick;
         pick up right chopstick;
      }
   }
   eat;
   put down left chopstick;
   put down right chopstick;
}
```

The real solution: put a total order on the chopsticks.

```
while (1)
{
   think;
   pick up lower-numbered chopstick;
   pick up higher-numbered chopstick;
   eat;
   put down lower-numbered chopstick;
   put down higher-numbered chopstick;
}
```

2.2 Second Readers-Writers Problem

Writers have priority.

- Implement using semaphores:

```
Semaphore wm = 1;      // Control access to writeCount.
Semaphore write = 1;   // Needed to write.
Semaphore rq = 1;      // Queue of readers.
Semaphore rm = 1;      // Control access to readCount.
Semaphore read = 1;    // Needed to read.

int writeCount = 0;
int readCount = 0;


// Writer:
```

```
while (1)
{
   wait(wm);
   if (++writeCount == 1)
      wait(read);
   signal(wm);
   wait(write);

   // Write.

   signal(write);
   wait(wm);
   if (--writeCount == 0)
      signal(read);
   signal(wm);
}


// Reader:
while (1)
{
   wait(rq);
   wait(read);
   wait(rm);
   if (++readCount == 1)
      wait(write);
   signal(rm);
   signal(read);
   signal(rq);

   // Read.

   wait(rm);
   if (--readers == 0)
      signal(write);
   signal(rm);
}
```

- Implement using locks and condition variables.

```
lock wl;               // Control access to writeCount.
lock rl;               // Control access to readCount.
lock write;            // Only a single writer.
```

```
condition noReaders;   // Used by writers.
condition noWriters;   // Used by readers.

int readCount = 0;
int writeCount = 0;


// Writer:
while (1)
{
   acquire(wl);
   ++writeCount;
   while (readCount != 0)
      wait(noReaders);
   release(wl);

   // Write.

   acquire(wl);
   if (--writeCount == 0)
      broadcast(noWriters);
   release(wl);
}


// Reader:
while (1)
{
   acquire(rl);
   while (writeCount != 0)
      wait(noWriters);
   ++readCount;
   release(rl);

   // Read.

   acquire(rl);
   if (--readCount == 0)
      signal(noReaders);
   release(rl);
}
```

What's wrong with this?

2.3 The Sleeping Barber Problem

A barbershop consists of a waiting room with `N` chairs, and the barber room containing the barber chair. If there are no customers to be served the barber goes to sleep. If a customer enters the barbershop and all chairs are busy, then the customer leaves the shop. If the barber is busy, then the customer sits in one of the available free chairs. If the barber is asleep, the customer wakes the barber up. Write a program to coordinate the barber and the customers.

What needs to be modeled?

1. Barber: awake/asleep.
2. The barber chair.
3. The customer chairs.
4. Waiting customers.
5. Access to shared data.