

CS414: Operating Systems – Nachos

Assignment 3: Synchronization Part Deux: Revenge of the Semaphores

Due: 12:01 am on April 27th, 2008

The purpose of this assignment is to further understand how to use semaphores to achieve synchronization. The basic ideas and programming environment are similar to Nachos Assignment 2.

Assignment

1. The local Wash-O-Matic laundromat has just entered the computer age. As each customer enters, he or she puts coins into slots at one of two stations and types in the number of washing machines he/she will need. The stations are connected to a central computer that automatically assigns available machines and outputs tokens that identify the machines to be used. The customer puts laundry into the machines and inserts each token into the machine indicated on the token. When a machine finishes its cycle, it informs the computer that it is available again. The computer maintains an array `available[NMACHINES]` whose elements are non-zero if the corresponding machine is available; `NMACHINES` is a constant equal to the number of machines there are in the laundromat, and `nfree` is a semaphore that indicates the number of available machines.

The code to allocate and release machines is as follows:

```
int allocate() // Returns index of available machine.
{
    int i;
    P(nfree); // Wait until a machine is available
    for (i=0; i < NMACHINES; i++)
    {
        if (available[i] != 0)
        {
            available[i] = 0;
            return i;
        }
    }
}

release(int machine) // Releases machine
{
    available[machine] = 1;
    V(nfree);
}
```

The available[] array is initialized to all ones, and semaphore nfree is initialized to NMACHINES.

It seems that if two people make requests at the two stations at the same time, they will occasionally be assigned the same machine. This has resulted in several brawls in the laundromat, and you have been called in by the owner to fix the problem. Assume that one thread handles each customer station. **Explain how the same washing machine can be assigned to two different customers.**

- (a) **Modify the allocate()/release() code to eliminate the problem by using semaphores (or locks or conditions).**
- (b) **Design and write code to test your modified allocate()/release() functions by simulating a sequence of customers using the laundromat:**
- Customers arrive and go to one of the two coin stations, deposit coins, and enter the number of machines they need.
 - Customers wait at the coin station until they are assigned a machine(s) and given the appropriate token(s).
 - Customer loads laundry, and waits for laundry to complete
 - Meanwhile, other customers are getting machines assigned to them and using the machines.
- (c) Notes, Hints, and **Additional Requirements:**
- For purposes of this problem, assume that a customer waits at a coin station until a washing machine(s) has been assigned – they don't have to step back and allow other customers to use the station while they're waiting for a machine(s) to become available. (This approach is simpler, and could reduce confusion and conflict in real life as well.)
 - Also assume that arriving customers form a single line and go to the first available coin station (i.e., like a bank with a single line where customers wait for the first available teller, not like a supermarket where the customers pick a checkout lane and then wait in its line).
 - You can simulate the operation of each washing machine in a fairly simple way; the main thing is to be sure it's released when its load is finished.
 - When a customer is using **multiple machines**, the machines can complete their cycles independently or all at the same time, whichever you choose.
 - **DON'T** worry about the exact amount of time each washer takes to complete a load; you don't need to simulate a specific delay for the machine's load cycles into your code (but it might not finish immediately).
 - **DO** design your code to allow multiple machines to operate simultaneously.
 - **DO** have more than 3 washing machines in the laundromat.
 - **DON'T** have so many machines available that there is no contention for their use. **DON'T** have so many machines that it makes testing your code

difficult.

- **DO** create customers who need more than one washing machine for your tests; this may require further changes to `allocate()` or influence the design of the functions that simulate the customer.
- **DON'T** allow a customer to request more machines than are available.

(d) Add your new & modified code to **threads/threadtest.cc** as for Assignment 2; you should leave all the existing code in place, create an additional `ThreadTestn()` function, and have `ThreadTest()` call the new test function.

2. You've just been hired by Mother Nature to help her out with the chemical reaction to form water, which she doesn't seem to be able to get right due to synchronization problems. The trick is to get two Hydrogen (H) atoms and one Oxygen (O) atom together at the same time to produce H_2O . The atoms are threads. Each H atom invokes a procedure `hReady` when it is ready to react, and each O atom invokes a procedure `oReady` when it's ready.

- (a) For this problem, you are to **write and test the code for `hReady` and `oReady`**.
- (b) The procedures must delay until there are at least two H atoms and one O atom present, and then one of the procedures must call the procedure `makeWater` (which just prints out a message saying that water was made).
- (c) After the `makeWater` call, two instances of `hReady` and one instance of `oReady` should return. **Write the code for `hReady` and `oReady` using semaphores (or locks or conditions) for synchronization. Your solution must avoid starvation and busywaiting.**
- (d) Notes, Hints, & **Additional Requirements**:
- In your test code, **DO** generate a sequence of atoms that is sufficient to exercise `makeWater`; **DON'T** just generate a few atoms and quit.
 - **DO** mix up the types (elements) of atoms that are generated; **DON'T** just create a sequence of one water molecule after another: H_2O , H_2O , H_2O ... (like a dripping water faucet).
- (e) Add your new code to **threads/threadtest.cc** as for Assignment 2; you should leave all the existing code in place, create yet another `ThreadTestn()` function, and have `ThreadTest()` call the new function.

What to Turn In

For both parts of the assignment, you should turn in:

- (a) A description of your implementation. Describe which thread and semaphores you used and how they enable your implementation.
- (b) A well documented listing of code.

- (c) A test plan for your implementation, with output.

How to Submit

No printed code or documents are needed. Every group only needs one copy of the submission. Follow the following instructions to submit your homework:

1. Create a directory called `assign3` under your **group** directory:

`/home/Spring08/cs414gnn/nachos/assign3`
2. Copy your code, description of your implementations and test plans to the **assign3** directory.
3. Use the shell command: **`chmod -R 775 /home/Spring08/cs414gnn/nachos`** to change the permissions of all your group's files to allow the group to read and write and everyone else to read them.
4. Write a `README.cs414gnn` file, including the name of files you modified in the **threads** directory, how to build your code, the answer to the question in Part 1, and your test plans.
5. Send an e-mail to *bursik@cs.virginia.edu* before the deadline, using "cs414 nachos3" as the subject. In the body of the e-mail, please include your group number your group members' names and usernames, and the path to the directory where the submission is located.

Start early on the Assignment – the end of the semester will be here soon!