Operating Systems
ECE344

Classic Synchronization Problems

Ashvin Goel
ECE
University of Toronto
Overview

- Classic synchronization examples
  - Readers and writers
  - Sleeping barber
The Readers and Writers Problem

- Multiple reader and writer threads want to access some shared data

- Synchronization requirements
  - Multiple readers can read concurrently
  - Writers synchronize with readers and other writers to ensure:
    - Only one writer can write the shared data at a time
    - When a writer is writing, no readers must access the data

- Goals: maximize concurrency, prevent starvation
Readers/Writers – Basics

```cpp
int rc = 0;

Reader () {
    rc = rc + 1;
    // Read shared data
    rc = rc - 1;
    // non-critical section
}

Writer () {
    // non-critical section
    // Write shared data
}
```

4
Readers/Writers – Mutex

```c
int rc = 0;
Mutex lock = UNLOCKED;

Writer () {
    // non-critical section
    // Write shared data
}

Reader () {
    lock(lock);
    rc = rc + 1;
    unlock(lock);
    // Read shared data
    lock(lock);
    rc = rc - 1;
    unlock(lock);
    // non-critical section
}
```
Readers/Writers – Synchronization

```c
int rc = 0;
Mutex lock = UNLOCKED;
Semaphore available = 1;

Writer () {
    // non-critical section
    down(available);
    // Write shared data
    up(available);
}

Reader () {
    lock(lock);
    if (rc == 0)
        down(available);
    rc = rc + 1;
    unlock(lock);
    // Read shared data
    lock(lock);
    rc = rc - 1;
    if (rc == 0)
        up(available);
    unlock(lock);
    // non-critical section
}
```

- Any problems with this solution?
The Sleeping Barber Problem
Sleeping Barber Logic

- **Barber**
  - When a customer is sitting on a waiting chair, ask them to come to the barber chair, and cut their hair
  - When done, move to the next customer
  - Else go to sleep, until someone comes in

- **Customer**
  - If barber is asleep, wake them up for a haircut
  - If someone is getting a haircut, wait for the barber to become free by sitting in a waiting chair
  - If all waiting chairs are full, leave the barber shop
How to Solve Sleeping Barber?

- How will we model the barber and customers?
  - As threads

- What shared state variables do we need?
  - Number of waiting customers

- How will we protect the state variables?
  - Mutex lock

- How will the barber wait and wake up?
  - Customer semaphore: number of available customers

- How will the customers wait and wake up?
  - Barber semaphore: barber is available to cut hair
const int CHAIRS = 5; int nr_waiting = 0;

Barber_Thread() {
    while (1) {
        nr_waiting--;
        cut_hair();
    }
}

Customer_Thread() {
    if (nr_waiting < CHAIRS) {
        nr_waiting++;
        get_hair_cut();
    } else { // give up
    }
}
```c
const int CHAIRS = 5; int nr_waiting = 0;
Mutex lock = UNLOCKED; // protect nr_waiting

Barber_Thread() {
    while (1) {
        lock(lock);
        nr_waiting--;
        unlock(lock);
        cut_hair();
    }
}

Customer_Thread() {
    lock(lock);
    if (nr_waiting < CHAIRS) {
        nr_waiting++;
        unlock(lock);
        get_hair_cut();
    } else { // give up
        unlock(lock);
    }
}
```
Sleeping Barber – Synchronize Barber

const int CHAIRS = 5; int nr_waiting = 0;
Mutex lock = UNLOCKED; // protect nr_waiting
Semaphore customers = 0;

Barber_Thread() {
    while (1) {
        down(customers);
        lock(lock);
        nr_waiting--;
        unlock(lock);

        cut_hair();
    }
}

Customer_Thread() {
    lock(lock);
    if (nr_waiting < CHAIRS) {
        nr_waiting++;
        unlock(lock);
        up(customers);

        get_hair_cut();
    } else { // give up
        unlock(lock);
    }
}
const int CHAIRS = 5; int nr_waiting = 0;
Mutex lock = UNLOCKED; // protect nr_waiting
Semaphore customers = 0; Semaphore barber = 0;

Barber_Thread() {
    while (1) {
        down(customers);
        lock(lock);
        nr_waiting--;
        unlock(lock);
        up(barber);
        cut_hair();
    }
}

Customer_Thread() {
    lock(lock);
    if (nr_waiting < CHAIRS) {
        nr_waiting++;
        unlock(lock);
        up(customers);
        down(barber);
        get_hair_cut();
    } else { // give up
        unlock(lock);
    }
}
const int CHAIRS = 5; int nr_waiting = 0;
Mutex lock = UNLOCKED; // protect nr_waiting
Semaphore customers = 0; Semaphore barber = 0;

Barber_Thread() {
    while (1) {
        down(customers);
        lock(lock);
        nr_waiting--;
        unlock(lock);
        up(barber);
        cut_hair();
    }
}

Customer_Thread() {
    lock(lock);
    if (nr_waiting < CHAIRS) {
        nr_waiting++;
        unlock(lock);
        up(customers);
        down(barber);
        get_hair_cut();
    } else { // give up
        unlock(lock);
    }
}
Summary

- Synchronization enables threads to wait on some condition before proceeding
- Monitors and semaphore provide two systematic solutions for implementing synchronization
- Used semaphores to implement
  - Reader-writer problem
  - Sleeping barber problem
- Next lecture: Unix threads and processes
Think Time

- A variant of the reader-writer problem is the reader-writer lock problem. Find out how a reader-writer lock works, and then design a solution for reader-writer locks using the solution we have provided for the reader-writer problem.

- The reader-writer problem suffers from starvation. Design a solution for avoiding starvation.

- Does the sleeping barber problem suffer from starvation?

- What are the similarities and differences between the sleeping barber and the producer-consumer problems?
A variant of the reader-writer problem is the reader-writer lock problem. Find out how a reader-writer lock works, and then design a solution for reader-writer locks using the solution we have provided for the reader-writer problem.

The solution is similar to the reader-writer solution:

- **writer lock**: down(available)
- **writer unlock**: up(available)
- **reader lock**: lock(lock); if (rc == 0) down(available); rc = rc + 1; unlock(lock);
- **reader unlock**: lock(lock); rc = rc - 1; if (rc == 0) up(available); unlock(lock);
The reader-writer problem suffers from starvation. Design a solution for avoiding starvation.

- You need to implement a queue for the reader-writer problem. The queue is similar to a producer-consumer problem: 1) accesses to the queue need to be protected from races, 2) you need to add synchronization to ensure that readers and writers enqueue themselves in the queue and are dequeued in first-come-first-served order.

Does the sleeping barber problem suffer from starvation?

- Customers wait on barber by using down(barber). If the implementations of the down and up operations use queues, then there will be no starvation. Otherwise, starvation is possible.
What are the similarities and differences between the sleeping barber and the producer-consumer problems?

- In the sleeping barber problem, the customers are producers that enqueue themselves in chairs. There are a fixed number of chairs, similar to a bounded buffer. There is only one consumer, which is the barber. The main difference between the two problems is that customers leave when the buffer is full (nr_waiting >= CHAIRS), instead of waiting for the barber to become available. Also, the initialization is different. With the barber problem, both semaphores are initialized to 0, while with the producer-consumer problem, the empty semaphore is initialed to N.