

Lab T Introduction

Don Porter
CS372H - Spring 2007

A break from JOS

- Not related to other labs
- Multithreaded programming one of the most important skills to take away from an OS course!

Barbershop Problem

- 1 Barber
- 1 Chair
- n Chairs for waiting customers

Barbershop, cont.

- If no customers
 - Barber sits down, goes to sleep
- First customer arrives
 - Wakes up barber
 - Sits in chair & gets cut
 - Customer Leaves
 - Barber calls next customer or goes to sleep
- If others arrive while customer in chair
 - Sit in waiting chair (if available)
 - Or leave shop (if no waiting chairs)

High-level code

```
barber(BarberShop bs){
    while(true){
        bs.waitForCustomerArrive(); // Post-condition -- barber out of
        // chair and at least one customer in shop
        bs.waitForCustomerInBarberChair(); // Post-condition -- customer in
        // barber chair

        cutHair();
        bs.doneCutting(); // Post-condition -- customer out of barber chair
    }
}

customer(BarberShop bs, int customerId){
    action = bs.waitForLeave(customerId);
    if(action == ACTION_LEAVE){
        return;
    }
    bs.sitInBarberChair(customerId); // Pre-condition -- barber chair is empty
    // Post-condition -- hair cut and
    // out of chair
    return;
}
```

Multithreaded Programming

- Difficult to get right!
 - Absolutely must read lab handouts!
- Correctness determined by reading code and careful reasoning, not testing!
 - Why?
 - Grading will be done the same way
- Careful pre/post conditions important

Simple Thread Package

- pthread interface messy - we give you a simpler version
- See `sthread.h`
- mutex - one thread can lock at a time
- Condition variables - associated with mutex
 - `scond_wait(cond, mutex)`
 - Must be holding mutex to call
 - Sleep until signaled
 - Wake up holding mutex

Thread Package, cont.

- Condition variables, cont.
 - `scond_broadcast()` - signal all waiters
 - `scond_signal()` - signal one waiter
 - Always wrap waits in while loop on a condition
 - Why?
 - Broadcast should always replace signal safely, but not vice versa
 - Why?

Thread Package, cont.

- Threads
 - `pthread_create(thread, start_routine, arg)`
 - `start_routine` - pointer to function
 - Makes a `pthread_t`
 - `pthread_exit()` - terminate
 - `pthread_sleep(seconds, ns)`
 - Sleep for a fixed time
 - **If you are waiting on anything other than a fixed amount of time, use `cond_wait!!!`**
 - `pthread_yield()`
 - Be nice, and allow others to run
 - May or may not wait

Back to the Barbershop

- Warmup: Shared Counter
- Implement various functions in skeleton code
- Priority Barbershop
 - Some customers are VIP's, serviced before non-VIP's
 - Superset of regular barbershop
 - Word to the wise: don't try to do this first - get regular barbershop correct first