CS 372H
Spring 2010
February 23, 2010

Potentially useful, if convoluted, example, to test your understanding of how to do concurrent programming. [Thanks to Mike Dahlin.]

Example: Sleeping Barber (Midterm 2002)

The shop has a barber, a barber chair, and a waiting room with NCHAIRS chairs. If there are no customers present, the barber sits in the barber chair and falls asleep. When a customer arrives, he wakes the sleeping barber. If an additional customer arrives while the barber is cutting hair, he sits in a waiting room chair if one is available. If no chairs are available, he leaves the shop. When the barber finishes cutting a customer's hair, he tells the customer to leave; then, if there are any customers in the waiting room he announces that the next customer can sit down. Customers in the waiting room get their hair cut in FIFO order.

The barber shop can be modeled as 2 shared objects, a BarberChair with the methods napInChair(), wakeBarber(), sitInChair(), cutHair(), and tellCustomerDone(). The BarberChair must have a state variable with the following states: EMPTY, BARBER_IN_CHAIR, LONG_HAIR_CUSTOMER_IN_CHAIR, SHORT_HAIR_CUSTOMER_IN_CHAIR. Note that neither a customer or barber should sit down until the previous customer is out of the chair (state == EMPTY). Note that cutHair() must not return until the customer is sitting in the chair (LONG_HAIR_CUSTOMER_IN_CHAIR). And note that a customer should not get out of the chair (e.g., return from sit in chair) until his hair is cut (SHORT_HAIR_CUSTOMER_IN_CHAIR). The barber should only get in the chair (BARBER_IN_CHAIR) if no customers are waiting. **You may need additional state variables.**

The WaitingRoom has the methods enter() which immediately returns WR_FULL if the waiting room is full or (immediately or eventually) returns MY_TURN when it is the caller's turn to get his hair cut, and it has the method callNextCustomer() which returns WR_BUSY or WR_EMPTY depending on if there is a customer in the waiting room or not. Customers are served in FIFO order.

Thus, each customer thread executes the code:

```
Customer(WaitingRoom *wr, BarberChair *bc)
{
    status = wr->enter();
    if(status == WR_FULL){
        return;
    }
    bc->wakeBarber();
    bc->sitInChair();  //  Wait for chair to be EMPTY
                       // Make state LONG_HAIR_CUSTOMER_IN_CHAIR
                       // Wait until SHORT_HAIR_CUSTOMER_IN_CHAIR
                       // then make chair  EMPTY and return
    return;
}
```

The barber thread executes the code:

```
Barber(WaitingRoom *wr, BarberChair *bc)
{
    while(1){          // A barber's work is never done
        status = wr->callNextCustomer();
        if(status == WR_EMPTY){
            bc->napInChair(); // Set state to BARBER_IN_CHAIR; return with state EMPTY
```

```
        }
        bc->cutHair(); // Block until LONG_HAIR_CUSTOMER_IN_CHAIR;
                       // Return with SHORT_HAIR_CUSTOMER_IN_CHAIR
        bc->tellCustomerDone(); // Return when EMPTY
    }
}
```

Write the code for the WaitingRoom class and the BarberChair class. Use locks and condition variables for synchronization and follow the coding standards specified in Mike Dahlin's write-up.

Hint and requirement reminder: remember to start by asking for each method "when can a thread wait?" and writing down a synchronization variable for **each** such situation.

List the member variables of class **WaitingRoom** including their type, their name, and their initial value. Then write the methods for WaitingRoom

List the member variables of class **BarberChair** including their type, their name, and their initial value. Then write the methods for BarberChair

(Solutions on next page)

Waitiing Room Solution:

Type	Name	Initial Value (if applicable)
mutex	*lock*	
cond	*cond*	
int	*nfull*	*0*
int	*ticketAvail*	*0*
int	*ticketTurn*	*-1*

```
int WaitingRoom::custEnter()
    lock.acquire();
    int ret;
    if(nfull == NCHAIRS){
        ret = WR_FULL;
    }
    else{
        ret = MY_TURN;
        myTicket = ticketAvail++;
        nfull++;
        while(myTicket > ticketTurn){
                cond.wait(&lock);
        }
        nfull--;
    }
    lock.release();
    return ret;

int WaitingRoom::callNextCustomer()
    lock.acquire();
    ticketTurn++;
    if(nfull == 0){
        ret = EMPTY;
    }
    else{
        ret = BUSY;
        cond.broadcast();
    }
    lock.release();
    return ret;
```

Barber Chair Solution:

Type	Name	Initial Value (if applicable)
mutex	*lock*	
cond	*custUp*	
cond	*barberGetUp*	
cond	*sitDown*	
cond	*seatFree*	
cond	*cutDone*	
int	*state*	*EMPTY*
int	*custWalkedIn*	*0*

```
void BarberChair::napInChair()
      lock.acquire();
     if(custWalkedIn == 0){   // Cust could arrive before I sit down
        state = BARBER_IN_CHAIR;
     }
     while(custWalkedIn == 0){
        barberGetUp.wait(&lock);
     }
     custWalkedIn = 0;
    if(state == BARBER_IN_CHAIR){ // Cust could have beaten us
        state = EMPTY
        seatFree.signal(&lock);
    }
    lock.release();

void BarberChair::wakeBarber()
    lock.acquire();
    custWalkedIn = 1;
    barberGetUp.signal(&lock);
    lock.release()

void BarberChair::sitInChair()
    lock.acquire()
    while(state != EMPTY){
       seatFree.wait(&lock);
    }
    state = LONG_HAIR_CUSTOMER_IN_CHAIR;
    sitDown.signal(&lock);
    while(state != SHORT_HAIR_CUSTOMER_IN_CHAIR){
       cutDone.wait(&lock);
    }
    state = EMPTY;
    custUp.signal(&lock);
    lock.release();
}


void BarberChair::cutHair()
    lock.acquire();
    while(state != LONG_HAIR_CUSTOMER_IN_CHAIR){
        sitDown.wait(&lock);
    }
    state = SHORT_HAIR_CUSTOMER_IN_CHAIR;
    cutDone.signal(&lock);
    lock.release();
```

```
void BarberChair::tellCustomerDone()
    lock.acquire();
    while(state != EMPTY){  // NOTE: No other cust can arrive until I call call_next_cust()
        custUp.wait(&lock);
    }

    lock.release();
```