Synchronization II

CS 241

Oct. 11, 2013

Midterm Exam Information

- **TA Review Sessions:**
  - **Today**, 4:00pm – 6:00pm
    119 MSEB (MatSc, next to Engineering Hall)
    Not recorded
  - **Monday**, 11:00am – 11:50am
    1404 SC (During regular class time!)
    Recorded
- **Midterm Exam: Monday, 7:00 – 9:00pm**

Midterm Exam Information

- **Exam Information:**
  - **Questions / Scoring:**
    - **50%**: Multiple Choice
    - **50%**: Free Response
  - **Required Materials:**
    - i-Card
    - #2 Pencil
    - Nothing else!
  - **There will be assigned seating.**
  - **Room Numbers:**
    - Based on last name:
    - **A-G:** 103 TB
    - **H-Le**: 112 TB
    - **Li-Z**: 1404 SC
  - **We will not answer <u>any</u> questions during exam.**

Review

- Wednesday:
  - Software Solution: **Peterson's Solution**.
    - Implements a lock using only software
    - Requires **busy waiting**

- Today**:**
  - Hardware solution!

Primitive #1: mutex

- A **mutex** is an atomic lock.
  -
  - On a call to **pthread_mutex_lock()**:
    -
    -
  - On a call to **pthread_mutex_unlock()**:
    -

test_and_set()

- Modern hardware provides a CPU operation to implement a **test_and_set()** function.

- **int test_and_set**(**int *atomic**)
  - Atomically sets the value in **atomic** to 1 and returns the previous value in **atomic**.

  - Still busy waiting:

```
while ( test_and_set(&atomic) == 0 ) { }
```

Mutex Implemented

```
int pthread_mutex_lock(pthread_mutex_t *mutex)
{
    /* Mutex is already locked */
   if ( test_and_set(&mutex->lock) == 1 )
      add_to_blocked_queue_on(mutex);

   /* Mutex was not locked, now is locked. */
   else
       return 0;
}
```

```
int ct = 0;
int X = 10000000;


void *up(void *ptr) {
  int i;
  for (i = 0; i < X; i++) {

    ct++;

  }
}

void main() {

   /* ... */

}
```

Primitive #2: conditional variables

- A **conditional variable** is the synchronization needed to implement a monitor.
  - **pthread_cond_wait(pthread_cond_t, pthread_mutex_t)**:
    -
    -
  - **pthread_cond_signal(pthread_cond_t)**:
    -
    -

Creating a monitor

Monitor Example

- Suppose you have a **bounded queue** (a queue with a fixed maximum capacity).

- You should:
  - – Block if the queue is full, wait for an empty spot in the queue before adding.
  - – Otherwise, add the element immediately.

  …this will create a **blocking bounded queue**.

Blocking Bounded Queue

```
void blocking_queue_push(queue_t *q, void *data) {



   /* queue_push() adds the element to the queue;
      queue_push() is not thread-safe */
   queue_push(q, data);


}
```

Blocking Bounded Queue

```
void *blocking_queue_pop(queue_t *q) {



   /* queue_pop() pops the top element;
      queue_pop() is not thread-safe */
   void *data = queue_pop(q);



}
```

cond_signal vs. cond_broadcast

- There are two ways to wake up a cond_wait():
  - **pthread_cond_signal()**
  - **pthread_cond_broadcast()**