CSc 360
Operating Systems
Mutual Exclusion

Jianping Pan

Summer 2015

Review: threads

- Thread vs process
  - easy to share info btw threads in one process
    - share and protect!
- Create and terminate threads
  - start routine, argument passing
    - threads are executed “in parallel”
- Join and detach threads
  - synchronization
    - “wait”

Shared or not shared?

- E.g., increment a counter (shared variable)
  - read the counter (from memory)
  - increment by one (at CPU)
  - write the counter
- How about two threads?
  - *sharing* only one counter
    - non-deterministic result: $R_1W_1R_2W_2$; $R_1R_2W_1W_2$
- "There is something not to be (always) shared"

Critical section

- Critical section
  - code section accessing shared data
  - only one thread executing in critical section
    - only one thread accessing the shared data: serialize
  - choose the right (size of) critical section!
- Approach: exclusion (lock)
  - if locked, wait!
  - if not lock, lock (and later, unlock)

Mutual exclusion

- Mutual exclusion (mutex)
  - only two states
    - unlocked: there is no thread in critical section
    - locked: there is one thread in critical section
  - state change is atomic
    - if it is unlocked, it can be locked by at most one thread when entering the critical section
    - if it is locked, it can “only” be unlocked by the locking thread when leaving the critical section

Mutex: more

- Mutex procedures
  - create a mutex variable (initially unlocked)
  - (some threads) attempt to lock the mutex
    - only one can lock the mutex
      - others may be blocked and waiting
    - the one with the mutex
      - execute the critical section
      - unlock the mutex variable eventually
  - destroy the mutex variable

Mutex with pthread

- Create mutex
  - int pthread_mutex_init (mutex, attributes);
- Attempt to lock
  - int pthread_mutex_lock (mutex);
    - if unlocked, lock and return immediately
    - if locked
      - “fast” lock: blocked until the mutex is unlocked
      - “test” lock: return immediately with error
      - “recursive” lock: “over”-lock
        - » multiple pthread_mutex_unlock() to unlock

Mutex with pthread: more

- Try to lock
  - int pthread_mutex_trylock (mutex);
    - if locked, return immediately with error code
- Unlock
  - int pthread_mutex_unlock (mutex);
    - if “recursive” lock, multiple pthread_mutex_unlock necessary to fully unlock the mutex
- Destroy mutex
  - int pthread_mutex_destroy (mutex);

Condition variable

- Used together with mutex
  - mutex: control access to shared data
  - condition: synchronize by condition “predict”
- Wait for condition
  - pthread_cond_wait (condition, mutex);
    - automatically unlock and wait for signal
    - on signal, wake up and automatically lock
- Signal or broadcast
  - pthread_cond_signal (condition);

Example: mutex and condition

- Main thread
  - global variable
  - create mutex and condition variable

- Wait to be signaled
  - pthread_mutex_lock();
  - pthread_cond_wait();
  - pthread_mutex_unlock();

- Send the signal
  - pthread_mutex_lock();
  - pthread_cond_signal();
  - pthread_mutex_unlock();

This lecture

- Mutex and condition
  - mutex: binary access control
    - locked or unlocked
  - condition: access control by condition
    - used together with mutex
- Explore further
  - multi-thread RSI?
    - compared with multi-process RSI?
  - P2?

Next lecture

- CPU scheduling
  - read OSC7 Chapter 5 (or OSC6 Chapter 6)