SSE3044 Introduction to Operating Systems

Prof. Jinkyu Jeong

Project 3-2. Mutex & CV

2018.05.30 (Wed.)

TAs
이규선(gyusun.lee@csl.skku.edu) /
안민우(minwoo.ahn@csl.skku.edu)

Project Plan

- Total 4 projects
  1) Process management
     - System call
     - Priority scheduler
  2) Virtual memory
  3) Synchronization
     - Thread
     - Mutex & CV (5/30 ~ 6/12 11:59pm)
  4) File system

Locks of Xv6

- Spinlock: busy wait until lock unlocks
  - acquire(struct spinlock *lock)
  - release(struct spinlock *lock)

```
// Mutual exclusion lock.
struct spinlock {
  uint locked;       // Is the lock held?

  // For debugging:
  char *name;        // Name of lock.
  struct cpu *cpu;   // The cpu holding the lock.
  uint pcs[10];      // The call stack (an array of program counters)
                     // that locked the lock.
};
```

- Sleeplock: sleep until lock unlocks
  - acquiresleep
  - releasesleep

```
// Long-term locks for processes
struct sleeplock {
  uint locked;       // Is the lock held?
  struct spinlock lk; // spinlock protecting this sleep lock

  // For debugging:
  char *name;        // Name of lock.
  int pid;           // Process holding lock
};
```

Project 3-2. Mutex & CV

- Implement 6 new system calls
  - int mutex_init(mutex_t *mutex)
  - int mutex_lock(mutex_t *mutex)
  - int mutex_unlock(mutex_t *mutex)
  - int cond_init(cond_t *cond)
  - int cond_wait(cond_t *cond, mutex_t *mutex)
  - int cond_signal(cond_t *cond)
- Make your own mutex & cv structure
  - synch.c / synch.h

<synch.h>

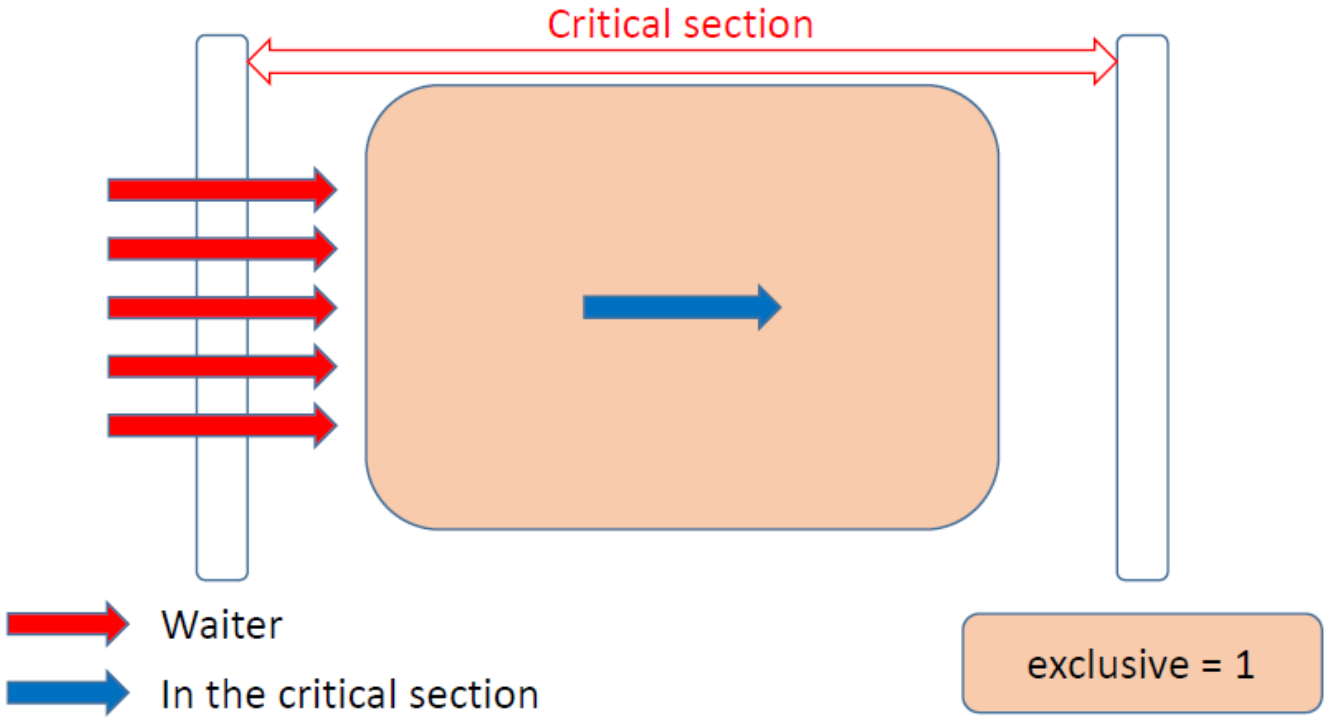
```
struct spinlock;
struct proc;

struct mutex_t{
    int active;
    int value;
    struct spinlock lock;
    struct proc *current;
    struct proc *queue[NTHREAD];
    int qsize;
};

struct cond_t{
    int active;
    struct spinlock lock;
    struct proc *queue[NTHREAD];
    int qsize;
};
```

Supporting Mutex on Xv6

- Only 1 thread can enter the critical section.
- Other threads are blocked until mutex released.



mutex_init()

- Name
  - mutex_init(): initialize the mutex referenced by mutex
- Synopsis
  - int mutex_init(mutex_t *mutex);
- Return value
  - Return 0, at initialization success.
  - Return -1, when the mutex is invalid.
  - Return -2, when attempting to reinitialize an already initialized mutex.

mutex_lock()

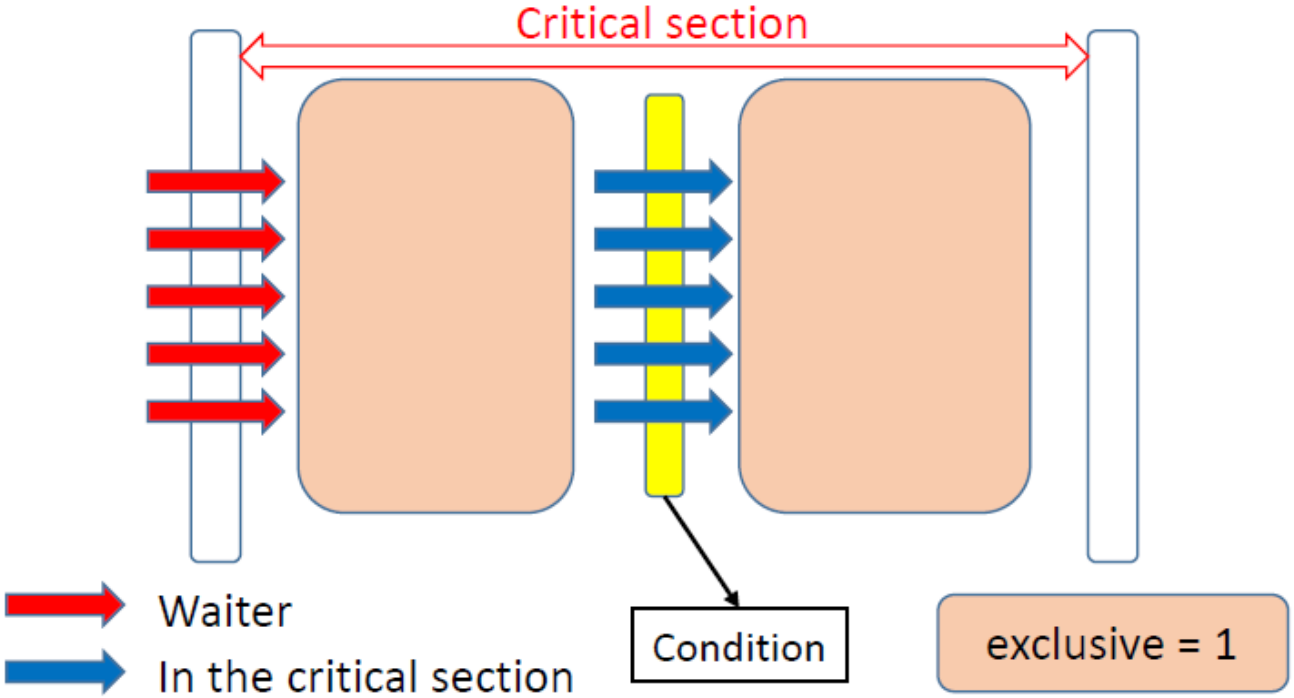
- Name
  - mutex_lock(): lock the mutex object. If the mutex is already locked, the calling thread is blocked until the mutex becomes available.
- Synopsis
  - int mutex_lock(mutex_t *mutex);
- Return value
  - Return 0, when lock is achieved.
  - Return -1, when the mutex is invalid.
  - Return -2, when the mutex is not initialized.
  - Return -3, when calling thread already has the mutex.

mutex_unlock()

- Name
  - mutex_unlock(): release the mutex object. If there are threads blocked on the mutex, unblocks the thread that waiting first for the mutex.
- Synopsis
  - int mutex_unlock(mutex_t *mutex);
- Return value
  - Return 0, when lock released successfully.
  - Return -1, when the mutex is invalid.
  - Return -2, when the mutex is not initialized.
  - Return -3, when calling thread does not have the mutex.

Supporting Condition Variable on Xv6

- Threads wait other threads with condition variable.
- Exiting thread signals to one thread with condition variable for wakeup.



cond_init()

- Name
  - cond_init(): initialize the condition variable
- Synopsis
  - int cond_init(cond_t *cond);
- Return value
  - Return 0, at initialization success.
  - Return -1, when the condition variable is invalid.
  - Return -2, when attempting to reinitialize an already initialized condition variable.

cond_wait()

- Name
  - cond_wait(): wait on the condition variable
- Synopsis
  - int cond_wait(cond_t *cond, mutex_t *mutex);
- Return value
  - Return 0, at success.
  - Return -1, when the condition variable is invalid.
  - Return -2, when the condition variable or mutex is not initialized.
  - Return -3, when calling thread does not have has the mutex.

cond_signal()

- Name
  - cond_signal(): wakeup blocked thread waiting on the condition variable
- Synopsis
  - int cond_signal(cond_t *cond);
- Return value
  - Return 0, at success.
  - Return -1, when the condition variable is invalid.
  - Return -2, when the condition variable is not initialized.

Conditional Variable Example

```
thread 1:
    pthread_mutex_lock(&mutex);
    while (!condition)
        pthread_cond_wait(&cond, &mutex);
    /* do something that requires holding the mutex and condition is true */
    pthread_mutex_unlock(&mutex);

thread2:
    pthread_mutex_lock(&mutex);
    /* do something that might make condition true */
    pthread_cond_signal(&cond);
    pthread_mutex_unlock(&mutex);
```

Tips

- Our condition variable follows Mesa semantic
  - cond_signal() places an unblocked thread on the ready queue, but the signaler continues inside the critical section.
- Resources in the critical section are shared only in the process, not inter-process.

CV Semantics

- Mesa semantic (used in Pthread)
  - signal() places a waiter on the ready queue, but signaler continues inside the critical section.
  - Condition is not necessarily true when waiter runs again.
  - Being woken up is only hint that something has changed.
  - Must recheck the condition.
- Hoare semantic
  - signal() immediately switches from the caller to a waiting thread, blocking the caller.
  - The condition that the waiter was anticipating is guranteed to hold when waiter excutes.

Reference

- Our thread interface is simplified version of POSIX Pthread interface.

- Refer to manual pages on
  - pthread_mutex_init()
  - pthread_mutex_lock()
  - pthread_mutex_unlock()
  - pthread_cond_init()
  - pthread_cond_wait()
  - pthread_cond_signal()

Submission

- Compress your xv6 folder as YourStudentID-3-2.tar.gz
  - tar cvf 2016710580-3-2.tar.gz xv6-SSE3044
- Send your tar.gz file to gyusun.lee@csl.skku.edu
  - Please command $make clean, before submission
  - Please send mail with uniformized title
    - [SSE3044]YourStudentID-3-2
- PLEASE DO NOT COPY
  - YOU WILL GET F GRADE IF YOU COPIED
- Due date: 6/12(Tue.), 23:59:59 PM
  - -25% per day for delayed submission

Grade policy

- If you failed to pass “oral test”, you will got 0
- Decribe how you have implemented to pass each test cases (If not, we consider it as failed to pass test cases)

Questions

- If you have questions, please email to TA
- You can also visit Semiconductor Building #400509
  - Please email TA before visiting