EEE3052: Introduction to Operating Systems

Fall 2017

Project #3

Project Plan

- 4 projects
  0) Install Xv6
  1) Process management
  2) Virtual memory
  3) Synchronization
     - Thread (11/13 ~ 11/21)
     - Mutex & Condition variable (11/20 ~ 12/3)
  4) File system

Locks of Xv6

- Spinlock: busy wait until lock free
  - acquire
  - release
- Sleeplock: sleep until lock free
  - acquiresleep
  - releasesleep

```
void
acquiresleep(struct sleeplock *lk)
{
  acquire(&lk->lk);
  while (lk->locked) {
    sleep(lk, &lk->lk);
  }
  lk->locked = 1;
  lk->pid = myproc()->pid;
  release(&lk->lk);
}

void
releasesleep(struct sleeplock *lk)
{
  acquire(&lk->lk);
  lk->locked = 0;
  lk->pid = 0;
  wakeup(lk);
  release(&lk->lk);
}
```

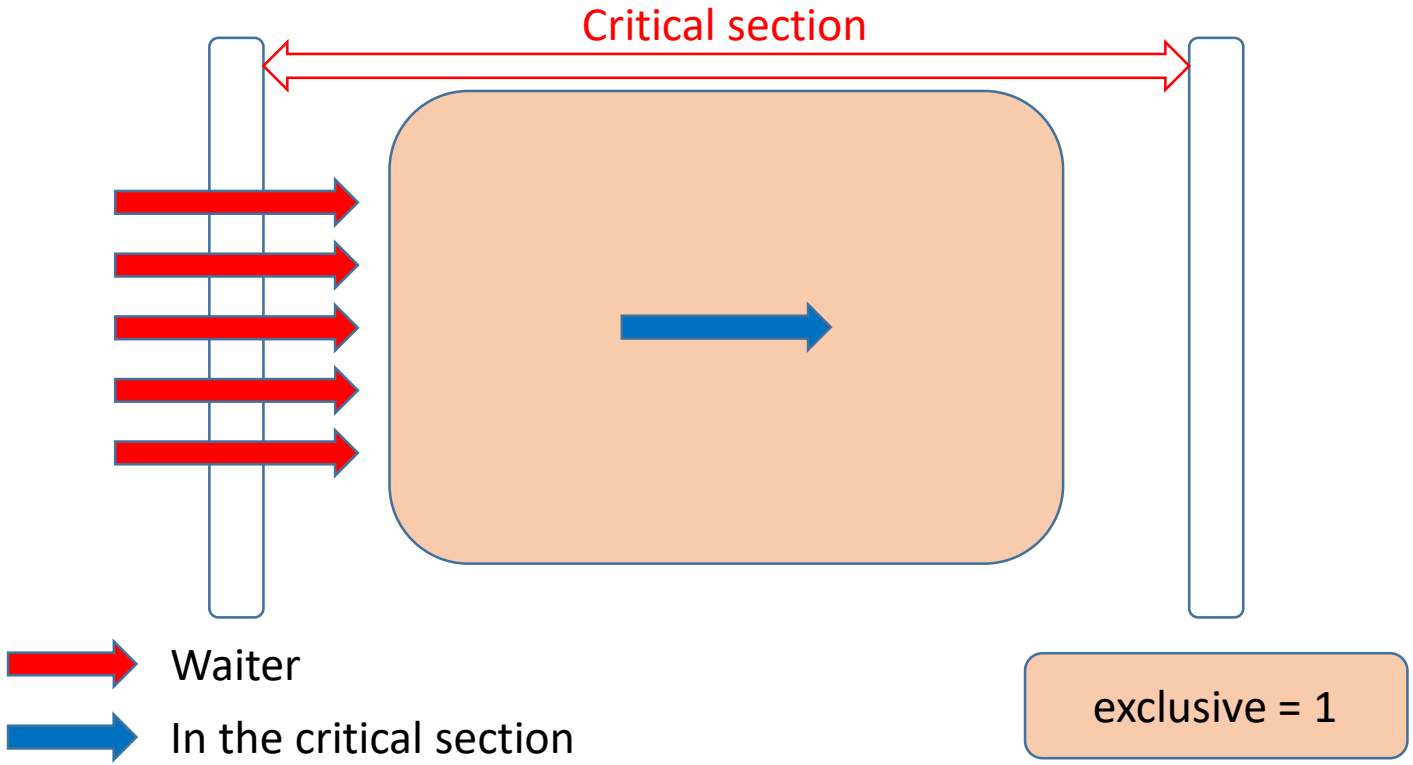
Project #3-2 – Mutex & Cond Variable

- Implement 6 new system call
  - 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)

- Change thread_create
  - Before
    - int thread_create(void *(*function)(void *), void *arg, void *stack)
  - After
    - int thread_create(void *(*function)(void *), int priority, void *arg, void *stack)

Supporting Mutex on Xv6

- Only 1 thread can enter the critical section (mutex).
- 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()

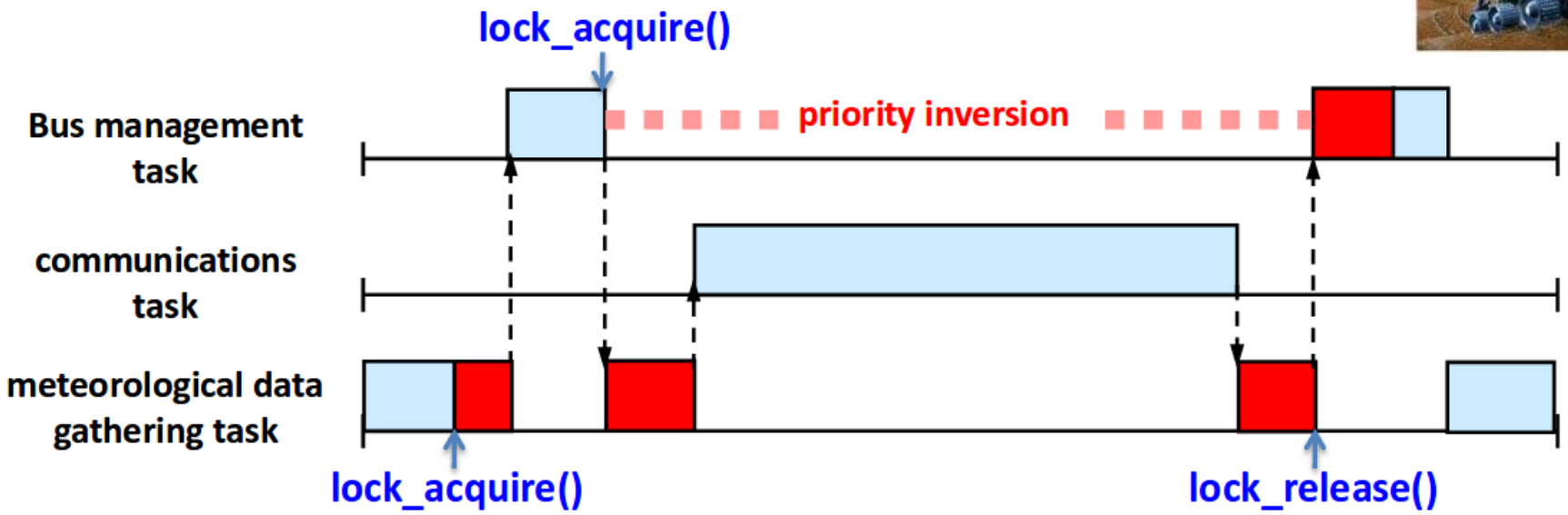
- Name
  - mutex_unlock: release the mutex object. If there are threads blocked on the mutex, the highest priority thread waiting for the mutex should be unblocked.

- 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.

Priority Donation

- Priority inversion problem
  - A situation where a higher-priority thread is unable to run because a lower-priority thread is holding a resource it needs, such as a lock.
  - *What really happened on Mars?*

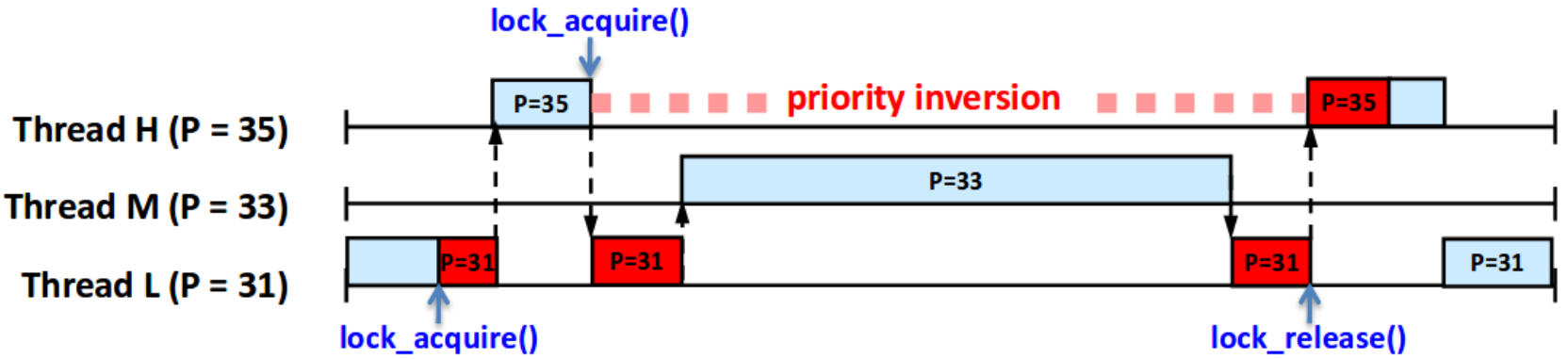


Priority Donation (Cont.)

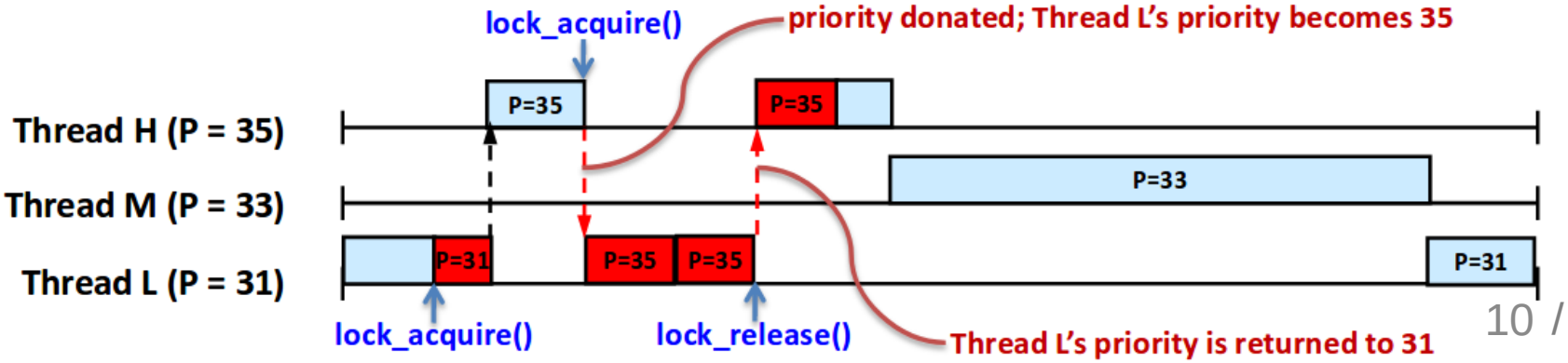
- Priority donation (or priority inheritance)
  - The higher-priority thread (doner) can donate its priority to the lower-priority thread (donee) holding the resource it requires.
  - The donee will get scheduled sooner since its priority is boosted due to donation.
  - When the donee finishes its job and releases the resource, its priority returned to the original priority.

Priority Donation (Cont.)

- Before priority donation

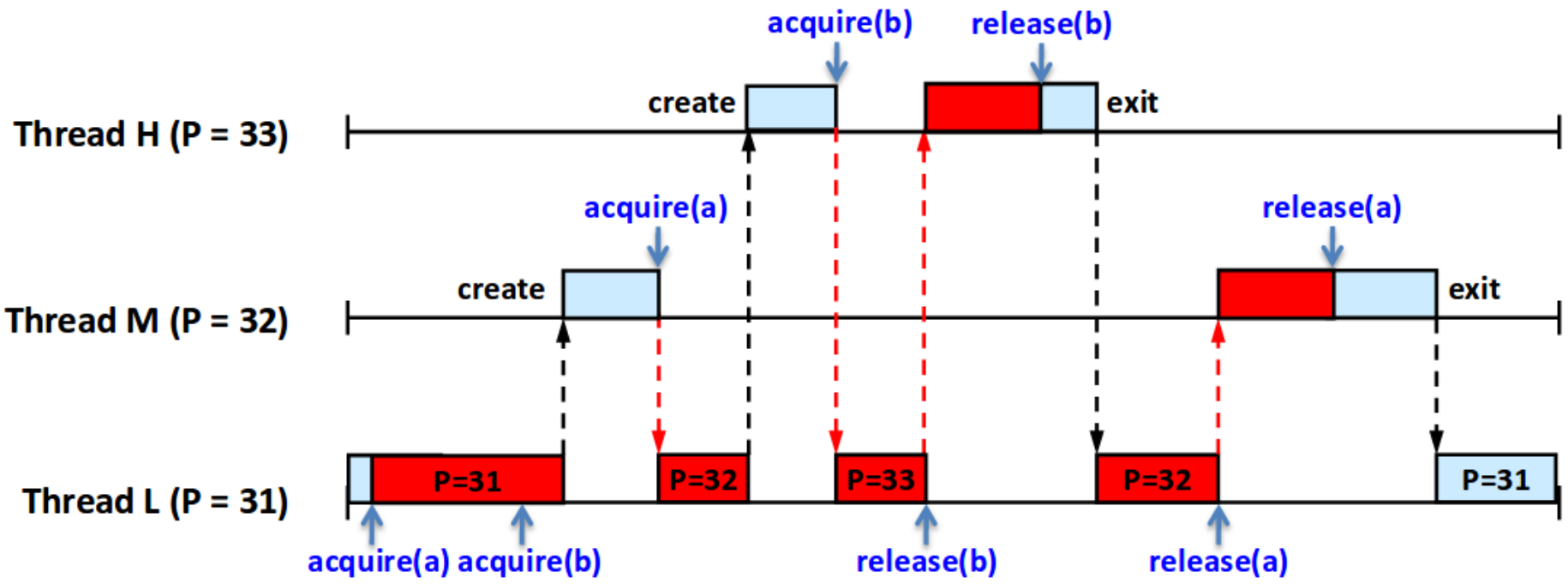


- After priority donation



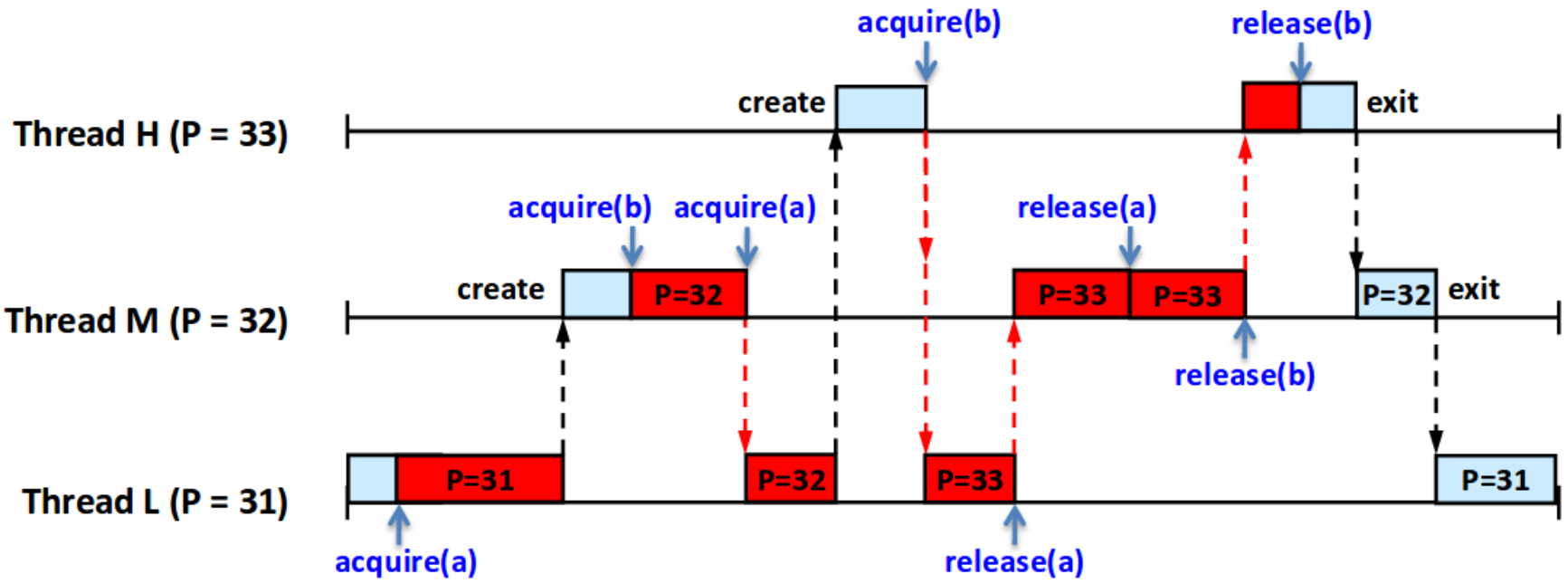
Priority Donation (Cont.)

- Multiple donations
  - Multiple priorities are donated to a single thread.



Priority Donation (Cont.)

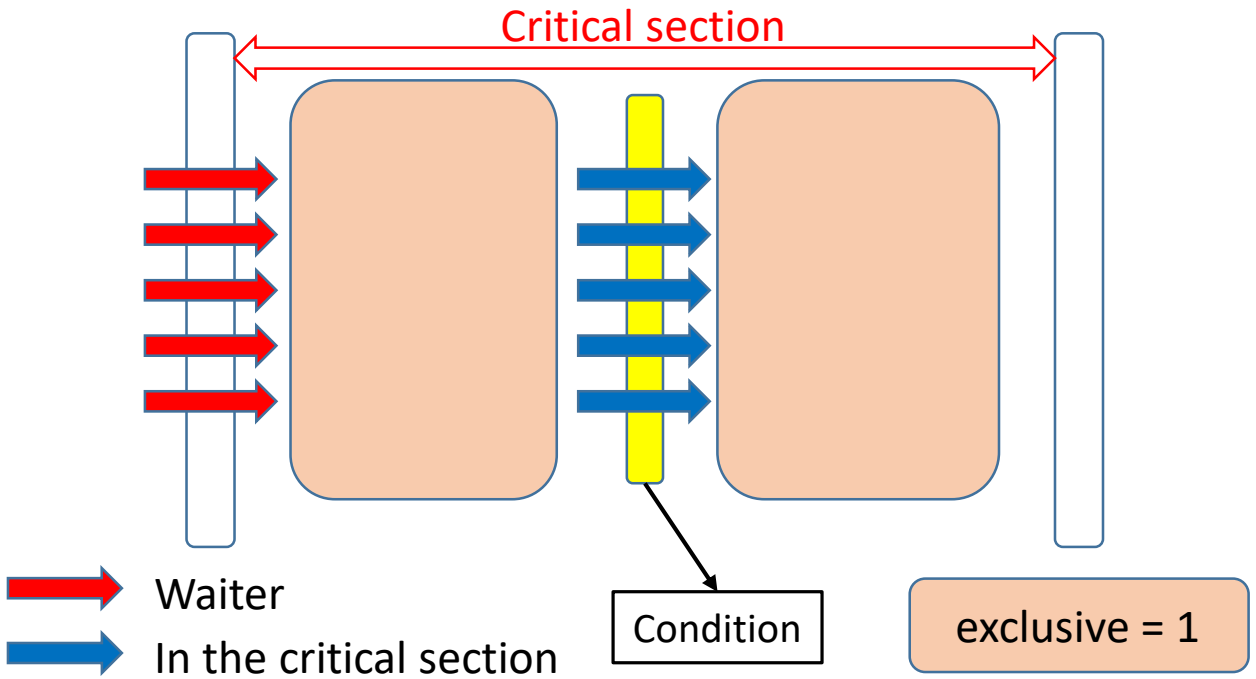
- Nested donation
  - If H is waiting on a lock that M holds and M is waiting on a lock that L holds, then both M and L should be boosted to H's priority.



Supporting Condition Variable on Xv6

- Threads wait other threads in condition variable.

- Exiting thread signals to one thread in 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 in 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 the mutex.

cond_signal()

- Name
  - cond_signal: wake up blocked thread in 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.

Test Case

- 5 test cases.
- Test cases may not have an effect on your grade.
- Just use for guide & test.
- Test4 is producer & consumer example
- Test5 is Fibonacci numbers

Things to Consider

- Our condition variable follows Mesa semantics
  - *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 semantics (used in Pthreads)
  - `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 a hint that something has changed
  - Must recheck the condition

- Hoare semantics
  - `signal()` immediately switches from the caller to a waiting thread, blocking the caller
  - The condition that the waiter was anticipating is guaranteed to hold when waiter executes

Reference

- Our thread interface is a simplified version of POSIX Pthreads interface.

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

Submit

- Send email to jaehyun.song@csl.skku.edu
  - Title: [EEE3052]Project-3_2-studentID-name
  - File name: studentID-3_2.tar.gz

- Due date
  - 2017/12/3 Sunday 23:59:59
  - Penalty 10% of each project score per one day

- TA contact
  - jaehyun.song@csl.skku.edu
  - minwoo.ahn@csl.skku.edu