

Operating Systems

Lab. Class

Project #6

Project Plan

- 6 projects
 0. Install xv6
 1. System call
 2. Scheduling
 3. Virtual memory 1
 4. Virtual memory 2
 5. Concurrency 1
 6. Concurrency 2
- Individual projects

Project #6 Overview

- Support mutex and condition variable on xv6
- Implement six new system calls
 - `mutex_init()`
 - `mutex_lock()`
 - `mutex_unlock()`
 - `cond_init()`
 - `cond_wait()`
 - `cond_signal()`
- Threads are blocked (rather than busy-waiting) on `mutex_lock()` and `cond_wait()`
- Consider thread priority when unblock a waiting thread
 - The highest priority thread waiting for a mutex or condition variable should be unblocked

Supporting Mutex on Xv6 – mutex_init ()

Name

mutex_init – initialize a mutex

Synopsis

```
int mutex_init(mutex_t *mutex)
```

Description

The `mutex_init()` function initializes the mutex referenced by *mutex*. Upon successful initialization, the state of the mutex becomes initialized and unlocked.

Return value

If successful, the `mutex_init()` function returns 0. It returns -1 when the value specified by *mutex* is invalid, -2 when attempting to reinitialize an already initialized mutex, or -3 when the mutex cannot be initialized for other reasons.

Supporting Mutex on Xv6 – mutex_lock ()

Name

mutex_lock – lock a mutex

Synopsis

```
int mutex_lock(mutex_t *mutex)
```

Description

The mutex object referenced by *mutex* is locked by calling mutex_lock(). If the mutex is already locked, the calling thread blocks until the mutex becomes available.

Return value

If successful, the mutex_lock() function returns 0. It returns -1 when the value specified by *mutex* is invalid, -2 when the mutex is not initialized, or -3 when the current thread already owns the mutex.

Supporting Mutex on Xv6 – mutex_unlock ()

Name

mutex_unlock – unlock a mutex

Synopsis

```
int mutex_unlock(mutex_t *mutex)
```

Description

The mutex_unlock() function releases the mutex object referenced by *mutex*. If there are threads blocked on the mutex, **the highest priority thread waiting for the mutex should be unblocked** and put on the list of ready threads.

Return value

If successful, the mutex_unlock() function returns 0. It returns -1 when the value specified by *mutex* is invalid, -2 when the mutex is not initialized, or -3 when the current thread does not own the mutex.

Supporting CV on Xv6 – cond_init ()

Name

cond_init – initialize a condition variable

Synopsis

```
int cond_init(cond_t *cond)
```

Description

The cond_init() function initializes the condition variable referenced by *cond*. Upon successful initialization, the state of the condition variable becomes initialized.

Return value

If successful, the cond_init() function returns 0. It returns -1 when the value specified by *cond* is invalid, -2 when attempting to reinitialize an already initialized condition variable, or -3 when the condition variable cannot be initialized for other reasons.

Supporting CV on Xv6 – cond_wait ()

Name

cond_wait – initialize a condition variable

Synopsis

```
int cond_wait(cond_t *cond, mutex_t *mutex)
```

Description

The cond_wait() function blocks on a condition variable. It should be called with *mutex* locked by the calling thread. It should release *mutex* and cause the calling thread to block on the condition variable *cond*. Upon successful return, the mutex should be locked and owned by the calling thread.

Return value

If successful, the cond_wait() function returns 0. It returns -1 when the value specified by *mutex* or *cond* is invalid, -2 when the mutex or condition variable is not initialized, or -3 when the mutex was not owned by the current thread.

Supporting CV on Xv6 – cond_signal ()

Name

cond_signal – signal a condition

Synopsis

```
int cond_signal(cond_t *cond)
```

Description

The cond_signal() function unblocks a thread blocked on the specified condition variable *cond*. If more than one thread is blocked on the condition variable, **the highest priority thread waiting for the condition variable should be unblocked** and put on the list of ready threads. We follow Mesa semantics

Return value

If successful, the cond_signal() function returns 0. It returns -1 when the value specified by *cond* is invalid, or -2 when the condition variable is not initialized.

PA #6 – Things to Consider

- Our condition variable follows Mesa semantics, i.e. `cond_signal()` places an unblocked thread on the ready queue, but the signaler continues inside the critical section
- As in PA #5, the maximum number of threads per process is limited to 8 (including the main thread). This means that the number of threads that are blocked on a mutex or a condition variable does not exceed 7.

PA #6 Template Code & Test Cases

- `git clone https://github.com/jinsoox/xv6-skku.git -b pa6`
- Modifications
 - Remove debug messages
 - Do not print any messages on screen
 - halt system call
 - Halt xv6 program
 - New system calls for supporting threads (in `synch.c`)
 - `mutex_init()`, `mutex_lock()`, `mutex_unlock()`, `cond_init()`, `cond_wait()`, `cond_signal()`
 - Mutex & CV test program
 - `mutetest1.c`
 - `mutetest2.c`

Project Submission Procedure

- <http://sys.skku.edu>
 - Submit a tarball file made from “**make tarball**”
- Submission deadline
 - 2017-06-11 23:59
- Since 6nd submission, -5% penalty of the project score
 - Up to 5 submissions are free to accept
- Every one day delay, -25% penalty of the project score
 - You can use up to 5 *slip* days