SWE3004: Operating Systems
prof. Euiseong Seo

Project 3. Synchronization

2019.4.17 (Wed.)

TAs
김종석(ks77sj@gmail.com) /
최동규(gmj03003@gmail.com)

Project plan

- Total 7 projects
  0) Starting xv6 operating system (5%)
  1) System call (10%)
  2) Thread (15%)
  3) Syncronization (15%)
  4) Scheduling 1(10%)
  5) Scheduling 2(15%)
  6) Page fault handler (15%)
  7) Copy on Write (15%)

Locks of xv6

- spinlock : busy wait until lock unlocks (spinlock.c)
  - acquire(struct spinlock *lock)
  - release(struct spinlock *lock)
- sleeplock : sleep until lock unlocks (sleeplock.c)
  - acquiresleep(struct sleeplock *lock)
  - releasesleep(struct sleeplock *lock)
  - You don’t have to use sleeplock for project3

Project 3. 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)

Project 3. Mutex & CV

- mutex & cv structure
  - make your own synch.c and synch.h files
  - Example :

```
#define NTHREAD 8
struct mutex_t{
    int valid;
    struct spinlock lock;
    struct proc *current;
    struct proc *queue[NTHREAD-1];
    int qsize;
};

struct cond_t{
    int active;
    struct spinlock lock;
    struct proc *queue[NTHREAD-1];
    int qsize;
};
```

valid : is initialized?
lock : spinlock for mutex
current : current locked thread
queue : queueing thread list
qsize : queue size

You don’t have to use this structure. This is just sample.

mutex_init()

- Synopsis
  - int mutex_init(struct 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
  - return 0 : at initialization success
  - return -1 : when the mutex is invalid
  - return -2 : when attempting to reinitialize an already initialized mutex

mutex_lock()

- Synopsis
  - int mutex_lock(struct 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
  - 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()

- Synopsis
  - int mutex_unlock(struct mutex_t *mutex);
- Description
  - The mutex_unlock() function releases the mutex object referenced by mutex. If there are threads blocked on the mutex, the thread waiting for the mutex should be unblocked in order and put on the list of ready threads.
- 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

cond_init()

- Synopsis
  - int cond_init(struct 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
  - 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()

- Synopsis
  - int cond_wait(struct cond_t *cond, struct 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
  - 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()

- Synopsis
  - int cond_signal(struct 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, thread waiting for the condition variable should be unblocked in order and put on the list of ready threads.
- Return value
  - return 0 : at success.
  - return -1 : when the condition variable is invalid
  - return -2 : when the condition variable is not initialized

CV semantic

- Mesa semantic
  - signal() places a waiter on the ready queue, but signaler continues inside the critical section.
  - You have to use Mesa semantic for this project.

Submission

- You should enter your ID & project no. (this time is 3) on Makefile
- $make tarball
- Then, xv6-project-3-studentID.tar.gz will be created in the parent folder.
- You need to submit a document.

Submission

- Send your code file (xv6-project-3-studentID.tar.gz) and document file to gmj03003@gmail.com
- Please send a mail with tittle including [SWE3004-P3]
  - Ex) [SWE3004-P3] 2014111111-project3
- PLEASE DO NOT COPY
  - YOU WILL GET F GRADE IF YOU COPIED
- Due date: 5/1(Tue.), 23:59:59 PM
  - Delays are allowed only one week from the deadline. And there will be up to -40%(6%*6 day + 4%*1 last day) penalty.

Questions

- If you have questions, please email to TA
- You can also visit #85533. Please email TA before visiting