CS 537: Intro to Operating Systems (Fall 2017)

Worksheet 9 - Semaphore and Deadlocks

Due: Nov 8th 2017 (Wed) in-class OR email Simmi before 11:59 pm

1. Semaphore Implementations

Consider the following implementation of lock and condition variable (CV) with semaphore.

Lock Implementation

```
typedef struct {
    sem_t s;
} lock;

void lock_init(lock* lk) {
    sem_init(&lk->s, 1);
}

void lock_acquire(lock* lk) {
    sem_wait(&lk->s);
}

void lock_release(lock* lk) {
    sem_post(&lk->s);
}
```

CV Implementation

```
typedef struct {
    sem_t s;
} cond;

void cond_init(cond* cv) {
    sem_init(&cv->s, 0);
}

void cond_wait(cond* cv, lock* lk) {
    lock_release(lk);
    sem_wait(&cv->s);
    lock_acquire(lk);
}

void cond_signal(cond* cv) {
    sem_post(&cv->s);
}
```

a. What is the difference between this lock implementation and a standard spinlock?

b. What is the difference between this CV implementation and a standard condition variable?

2. Deadlocks

Assume Thread#1 tries to `acquire(&v1->lock, &v2->lock)` and Thread#2 tries to `acquire(&v2->lock, &v1->lock)`. Four conditions need to hold for a deadlock to occur:

- Mutual exclusion
- Hold-and-wait
- No preemption
- Circular wait

a. The following implementations of `acquire(lock* L1, lock* L2)` try to prevent deadlock by breaking one of the conditions above. Write the corresponding condition (on the side of the three code snippets) that each solution is trying to break.

`if (L1 > L2) {`
`    pthread_mutex_lock(L1);`
`    pthread_mutex_lock(L2);`
`} else {`
`    pthread_mutex_lock(L2);`
`    pthread_mutex_lock(L1);`
`}`	

`pthread_mutex_lock(prevention);`
`pthread_mutex_lock(L1);`
`pthread_mutex_lock(L2);`
`pthread_mutex_unlock(prevention);`	

`top:`
`    pthread_mutex_lock(L1);`
`    if (pthread_mutex_trylock(L2) != 0) {`
`        pthread_mutex_unlock(L1);`
`        goto top;`
`    }`	

b. The three code snippets above prevent a deadlock by breaking one of the four required conditions for a deadlock to happen. How can you break the deadlock condition that is not attacked by the above three solutions?

c. For one of the solutions above, it is still possible that no thread can get both locks and proceed. Which of the above solutions has this issue? What is the problem here? How can you solve it?