Project 5: Sleeplock

Due: Nov, 2nd 2020 11:59 pm (submission: by email)
Submit a report to TA: Joontaek Oh (email: na94jun@gmail.com)
Name your file with “hw5_[your studentid]”. ex: hw5_20201234.c

In this project, you will modify sleeplock of xv6 so that use list structure and guarantee lock ordering. While spinlock of xv6 makes lock waiters repeat loop spending cpu time, sleeplock of xv6 makes waiters sleep process that not spend cpu time until wakeup time. However, when a holder of sleeplock release it, it wake process that have acquired spinlock of sleeplock firstly. Other systems adopt sleeplock mechanism that guarantees order of waiters. We will modify mechanism of sleeplock of xv6 to mechanism guaranteeing ordering waiters.

Part One: Add new system call to check lock ordering

First, we will check whether sleeplock of xv6 guarantees ordering of waiters. Because sleeplock is kernel data structure, we have to add new system call and this system call acquire lock if the calling process is not holding lock, release lock if the calling process is holding lock. We will add new system call, testlock. Add below code to bottom of sysfile.c.

```c
int
sys_testlock(void)
{
    static struct sleeplock lk;

    if (holdingsleep(&lk))
        releasesleep(&lk);
    else
        acquiresleep(&lk);

    return 0;
}
```
The `lk` will be in kernel data area so it will be maintained though the system call has been returned.

To add new system call, we have to modify more thing consist of both part of kernel part and user part.

In the kernel part, we have to define new system call number, add new prototype of system call and define new entry to system call table.

To define new system call number, add below code to `syscall.h`.

```c
#define SYS_testlock 22
```

(If you already using the number 22, define `SYS_testlock` as another number.)

To add new prototype of system call, add below code to `syscall.c`. The part of definition of prototypes of system call is in 85 line approximately.

```c
extern int sys_testlock(void);
```

To add new new entry to system call table, add below code to `syscall.c`. The system call table is defined in 108 line approximately.

```c
[SYS_testlock] sys_testlock,
```

In the user part, we have to add prototype and body of new system call.

Add below code to `user.h` to define prototype of new system call.

```c
int testlock(void);
```

Add below code to `usys.S` to define body of new system call.

```c
SYSCALL(testlock)
```

**Part Two: Check lock ordering**
Below code is user program that check lock order using new system call we added. Save this code as testlock.c.

```c
#include "types.h"
#include "user.h"
#include "fcntl.h"

int main()
{
    int i, pid;

testlock();

    for (i = 0; i<10; i++) {
        pid = fork();
        if (pid) {
            printf(1, "process %d is created\n", i);
            sleep(100);
        }
        else
            break;
    }

    if (pid) {
        sleep(1000);
        testlock();
        for (i = 0; i<10; i++)
            wait();
    }
    else {
        testlock();
        printf(1, "%d have acquired lock\n", i);
        testlock();
    }

exit();
}
```
It acquire sleeplock as call `testlock()` firstly, create 10 child. Each child tries to acquire sleeplock as call `testlock()` but they are blocked because sleeplock is already acquired by parent process, so they are wait for lock sleeping. The parent process release lock as call `testlock()` and wait for exit of all childs. To run this program in xv6, add below code to Makefile. The UPROGS is in 168 line in Makefile approximately.

```
UPROGS=
  ...
  _testlock"
```

Run `testlock` in xv6 and see printed result.

$ ./testlock
process 0 is created
process 1 is created
process 2 is created
process 3 is created
process 4 is created
process 5 is created
process 6 is created
process 7 is created
process 8 is created
process 9 is created
2 have acquired lock
3 have acquired lock
4 have acquired lock
5 have acquired lock
6 have acquired lock
7 have acquired lock
8 have acquired lock
9 have acquired lock
0 have acquired lock
1 have acquired lock

Though process 0 and 1 have created before others, they have acquired lock after others acquire and release lock. Thus, current version of sleeplock of xv6 not guarantee ordering of waiters. In this project, you will modify sleeplock so that `testlock` program always print process number in sequence.
Part Three: Modify sleeplock of xv6

Sleeplock in xv6 just use wait()/sleep() mechanism. When a process call acquire_sleep() to acquire lock, the process first acquire spinlock protecting the sleeplock. Then it check locked flag if it is 0. If it is 0, it modify locked flag to 1, release spinlock and return acquire sleep so that complete acquiring. If the locked flag is 1, it should wait until holder of sleeplock release the sleeplock. It call wait() function setting channel as address of sleeplock data structure. It will go to sleep status after releasing spinlock. There may are so many processes in sleep status waiting for sleeplock. Once the holder of sleeplock calls release_sleep(), the holder release sleeplock and call wakeup(). It will wake processes waiting for the sleeplock up. If multiple processes are woken up, they races to acquire spinlock. The process that have acquired spinlock will acquire sleeplock.

We will adopt list to sleeplock. If a process can’t acquire sleeplock, the process is added to list of sleeplock. When a holder of sleeplock release the sleeplock, the holder wake the process that is in front of the list of sleeplock up.

To adopt list of process to sleeplock, we make process can be consisted of list. Add next pointer to proc structure like below.

```c
struct proc {
    ...
    struct proc *next;
};
```

Sleeplock have to maintain head of list. Modify sleeplock structure like below.

```c
struct sleeplock {
    uint locked;    // Is the lock held?
    struct spinlock lk; // spinlock protecting this sleep lock
    struct proc *head;

    // For debugging:
    char *name;     // Name of lock.
    int pid;        // Process holding lock
```
Below codes are `acquiresleep()` and `releasesleep()` that acquire sleeplock and release sleeplock using `wakeup()` and `sleep()` respectively.

```c
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);
}
```

Bolden codes are core code that let process sleep or wakeup. Replace them code that use list so that wake a process that have tried to acquire very firstly. After you replace them, check whether it guarantee lock ordering correctly as run `testlock` program multiple times.

Hint: How can we wake a specific process up? Please see inline code of `wakeup1()`. How it wake processes up?

Submit: modified `sleeplock.c` file.