CEng 536 Advanced Unix
Fall 2016
Kernel Project 1
Due: 10/01/2016

1 Description

In this project you are going to implement two system calls for 2-D map locking similar to HW2. Caller processes/threads will have reader/writer locks of rectangular regions in a hyphotetical map.

When a rectangular area is locked for writing, no other lock can be given to any other intersecting lock requests. When a rectangular area is locked for reading, only read lock requests to the intersecting regions are given.

When a lock cannot be immediately given to a request, request blocks until the area can be locked. Unlocking of a region can unblock multiple blocked requests if they are compatible. If they are not, the system unblocks any of them.

Different than previous homework, the coordinates are **`unsigned long`** values. The processes use your system calls for synchronization rather than connecting the socket.

For simplicity, there is only one map.

```
#include<maplock536.h>
#include<syscall.h>

....
long lid ;

lid =  map_rdlock(10,10, 1000, 100000); /* may block */
if (lid > 0) {
        /* do critical region stuff */
        map_unlock(lid);
} else {
        /* blocked operation interrupted */
}

lid = map_rdlock_try(10,10, 1000, 10000);  /* does not block */
if (lid > 0) {
        /* do critical region stuff */
        map_unlock(lid);
} else  {
        printf("region already locked, try failed\n");
}

/* similarly writer locks exist */
lid = map_wrlock(10, 10, 1000, 10000);

lid = map_wrlock_try(10, 10, 1000, 10000);
```

All calls above are defined as macros mapped to system calls in `maplock536.h`:

```
#define NR_MAPLOCK 317
#define NR_MAPUNLOCK 318

#define MAP_RDLOCK       0
#define MAP_WRLOCK       1
```

```
#define MAP_TRYLOCK 2

#define map_rdlock( xlt, ylt, xrb, yrb) \
        syscall(NR_MAPLOCK, xlt, ylt, xrb, yrb, MAP_RDLOCK)

#define map_rdlock_try( xlt, ylt, xrb, yrb)      \
        syscall(NR_MAPLOCK, xlt, ylt, xrb, yrb, MAP_RDLOCK | MAP_TRYLOCK)

#define map_wrlock( xlt, ylt, xrb, yrb) \
        syscall(NR_MAPLOCK, xlt, ylt, xrb, yrb, MAP_WRLOCK)

#define map_wrlock_try( xlt, ylt, xrb, yrb)      \
        syscall(NR_MAPLOCK, xlt, ylt, xrb, yrb, MAP_WRLOCK | MAP_TRYLOCK)

#define map_unlock( id) \
        syscall(NR_MAPUNLOCK, id)
```

317 and 318 are replaced by the first available system call slot in your system. All lock requests return an **int** id for the request. Non-zero id reports the lock is successfully acquired. Zero return value reports either system call is interrupted without acquiring lock or lock is not available in case of "try" versions. Returned id uniquely identifies a successful lock request, so that a a following unlock operation can refer to it.

2 Implementation

You need the recompile the kernel in order to add system call. You need the following steps:

1. Get kernel source 3.16.2 (Any other 3.x version should work as long as following steps are valid)
2. Before compilation edit arch/x86/syscalls/syscall_32.tbl. Add following lines:
```
354     i386     map_lock    sys_map_lock
355     i386     map_unlock  sys_map_unlock
```
   Edit arch/x86/syscalls/syscall_64.tbl. Add following in common section:
```
317     common  map_lock    sys_map_lock
318     common  map_unlock  sys_map_unlock
```
3. Under ipc directory (chosen as a standard place, actually it can be anywhere with a proper Makefile), create maplock536.c file with following empty declarations:
```
SYSCALL_DEFINE5(map_lock, unsigned long, xlt, unsigned long, ylt,
        unsigned long , xrb, unsignled long, yrb, short flags) {
        printk("in map_lock flags: %d\n", flags)
        return 0;
}
SYSCALL_DEFINE1(map_unlock, int, lockid) {
        printk("in map_unlock for %d\n", lockid)
        return 0;
}
```
4. Add following line in ipc/Makefile
```
obj-y += maplock536.o
```
5. Copy your systems config as kernel configuration under source top directory:
```
cp /boot/config-`uname -r` .config
```

6. run `make`
7. Now you can install this kernel and boot Linux system. However installation of modules to initial ram disk is not trivial. Scripts and more explanation will be given in newsgroup for booting a virtual machine with this new kernel.

Using this template, develop your project. Note that you need to submit `maplock536.c` and a header file for kernel. You can use any kernel library and synchronization primitive you like. Make sure you protected all critical regions, and do not busy wait.
Use following header files when required:
`semaphore.h, completion.h, slab.h, kernel.h, compiler.h, printk.h, list.h, errno.h, wait`
You may need the following symbols:
`current, cred, task_struct, in_group_p, kuid_t, kgid_t`

3 Submission and External libraries

You need to submit a C source code. C++ is not allowed in kernel. Submit `maplock536.c` kernel code, and a header file you like.
Submission details will be announced later. Please ask all questions to:
`news://news.ceng.metu.edu.tr:2050/metu.ceng.course.536/`
`https://cow.ceng.metu.edu.tr/News/thread.php?group=metu.ceng.course.536`