# Project 3

조교: 박지웅, 김화정, 민철기

os-tas@dcslab.snu.ac.kr

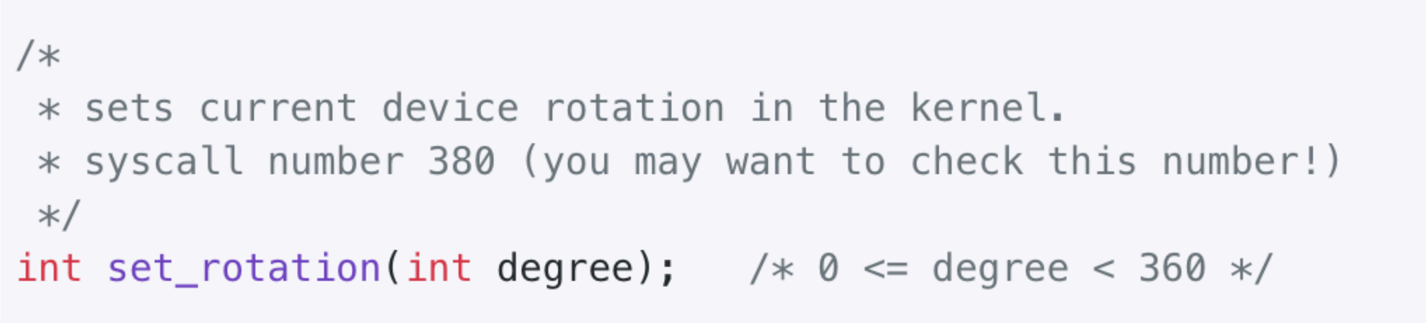
# Project 3

- Rotation Lock
  1. A user-space daemon to pass device rotation into the kernel
  2. Rotation-based reader/writer locks (45 pts.)
  3. Write two C programs to test the system (10 pts.)

- Due: 11.20 (월) 14:00PM

# A user-space daemon to pass device rotation into the kernel

- We will use a fake one-dimensional orientation (device rotation)
- We provide a daemon that updates this fake device rotation information, called *rotd*.
  - in the sequence of 0, 30, 60, … 330, 0, … in a fixed frequency (e.g., 2 seconds).
- You should write *set_rotation* system call by yourself, which updates the rotation information
  - All rotation related functions should be placed in kernel/rotation.c, and in include/linux/rotation.h



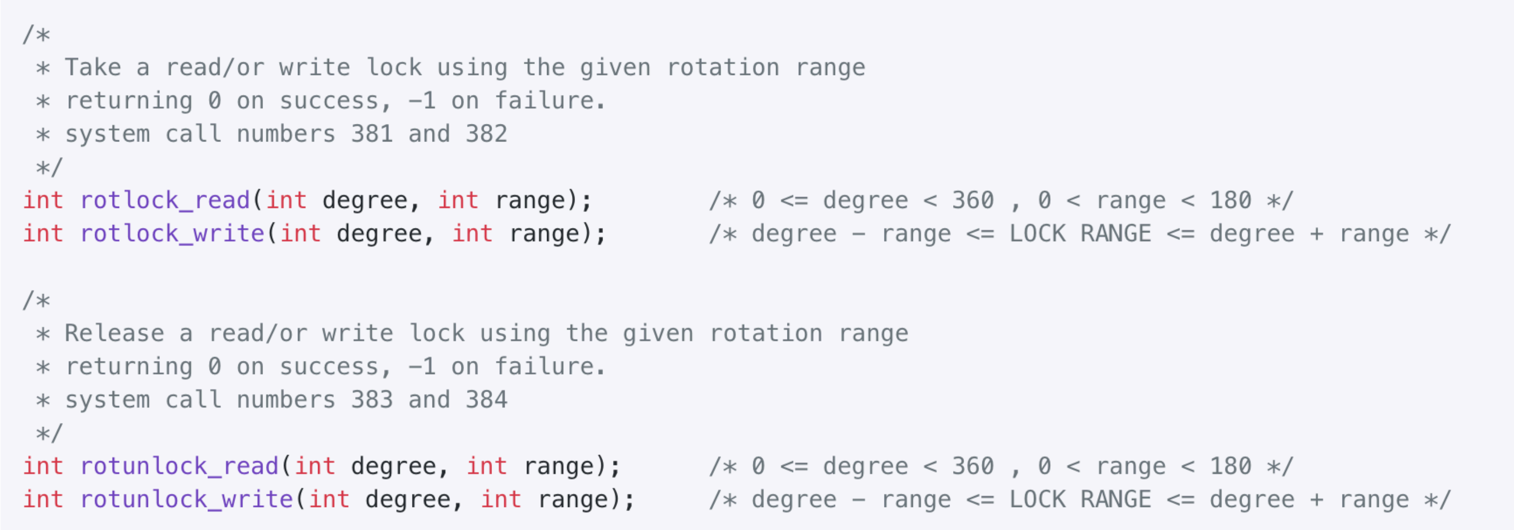
```
/*
 * sets current device rotation in the kernel.
 * syscall number 380 (you may want to check this number!)
 */
int set_rotation(int degree);   /* 0 <= degree < 360 */
```

# Rotation-based reader/writer locks (45 pts.)

- Design and implement a new kernel synchronization primitive that will provide reader-writer locks based on device rotation
  - Several readers can grab the lock at the same time, but only a single writer can grab the lock at any time
  - Make sure not to starve writers
- A lock is defined by a range of the rotation of the device
  - e.g. read_lock [30, 60] Success -> write_lock [20, 40] Fail -> write_lock [70, 80] Success
- If a process wants to grab a lock for a range, which does not cover the current physical device rotation, the process should block until the device is rotated into that range

# Rotation-based reader/writer locks (45 pts.)

- When the device rotation is updated, the processes that are waiting to grab a lock on a range entered by the new rotation should be allowed to grab the lock
  - Modify *set_rotation* system call to unblock processes waiting on a lock that covers the new rotation.
- You should choose to allow either all blocked readers to take the lock or a single blocked writer to take the lock, considering fairness

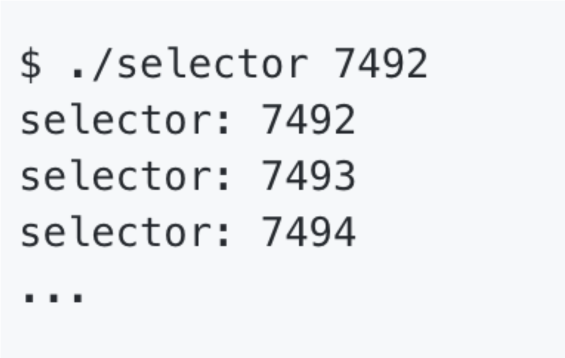


```
/*
 * Take a read/or write lock using the given rotation range
 * returning 0 on success, -1 on failure.
 * system call numbers 381 and 382
 */
int rotlock_read(int degree, int range);        /* 0 <= degree < 360 , 0 < range < 180 */
int rotlock_write(int degree, int range);       /* degree - range <= LOCK RANGE <= degree + range */

/*
 * Release a read/or write lock using the given rotation range
 * returning 0 on success, -1 on failure.
 * system call numbers 383 and 384
 */
int rotunlock_read(int degree, int range);      /* 0 <= degree < 360 , 0 < range < 180 */
int rotunlock_write(int degree, int range);     /* degree - range <= LOCK RANGE <= degree + range */
```

# Write two C programs to test the system (10 pts.)

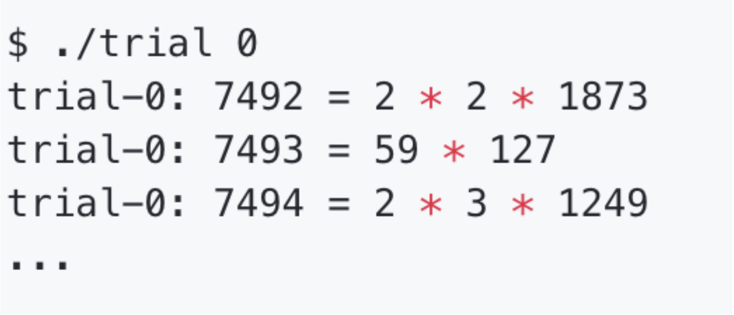
- Write two user programs for testing
  - selector
    - accepts a starting integer as the only argument
    - take write lock [0,180]
    - writes the integer from the argument to a file called integer
    - write the result to stdout
    - release write lock
    - re-acquired before the last integer + 1
    - run until terminated using Ctrl+c



```
$ ./selector 7492
selector: 7492
selector: 7493
selector: 7494
...
```

# Write two C programs to test the system (10 pts.)

- Write two user programs for testing
  - trial
    - accepts an integer identifier as the only argument
    - take read lock [0,180]
    - open the file called integer
    - calculate the prime factorization of the integer
    - write the result to stdout
    - close file and release read lock
    - run until terminated using Ctrl+c



```
$ ./trial 0
trial-0: 7492 = 2 * 2 * 1873
trial-0: 7493 = 59 * 127
trial-0: 7494 = 2 * 3 * 1249
...
```