Q1.

```
xv6...
cpu1: starting 1
lapicid 1: panic: acquire
 8010441d 801057ea 801056dc 80102e3f 80102e5a 705a 0 0 0 0
```

Right Control

```
void
acquire(struct spinlock *lk)
{
  pushcli(); // disable interrupts to avoid deadlock.
  if(holding(lk))
    panic("acquire");

  // The xchg is atomic
```

acquire panic 발생. acquire한상태에서 또 lock을 얻으려고 하면 holding 확인하고 acquire panic 일으키기 때문

Q2.

```
335+1 records out
171984 bytes (172 kB, 168 KiB) copied, 0.00524932 s, 32.8 MB/s
qemu-system-i386 -nographic -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,ind
ex=0,media=disk,format=raw -smp 2 -m 512
Failed to open module: /usr/lib/x86_64-linux-gnu/libcurl-gnutls.so.4: invalid ELF header
xv6...
cpu1: starting 1
cpu0: starting 0
lapicid 1: panic: acquire
 801043fd 80102063 801059a5 801056bc 80100183 801013c5 801014bf 801036c4 801056bf 0_
```

acquire panic이 발생한다.

```
alltraps:
  # Build trap frame.
  pushl %ds
801056a7:       1e                      push   %ds
  pushl %es
801056a8:       06                      push   %es
  pushl %fs
801056a9:       0f a0                   push   %fs
  pushl %gs
801056ab:       0f a8                   push   %gs
  pushal
801056ad:       60                      pusha

  # Set up data segments.
  movw $(SEG_KDATA<<3), %ax
801056ae:       66 b8 10 00             mov    $0x10,%ax
  movw %ax, %ds
801056b2:       8e d8                   mov    %eax,%ds
  movw %ax, %es
801056b4:       8e c0                   mov    %eax,%es

  # Call trap(tf), where tf=%esp
  pushl %esp
801056b6:       54                      push   %esp
  call trap
801056b7:       e8 e4 00 00 00          call   801057a0 <trap>
  addl $4, %esp
801056bc:       83 c4 04                add    $0x4,%esp

801056bf <trapret>:
```

Trapret에 forkret address 쌓는다

```
  if (first) {
    // Some initialization functions must be run in the context
    // of a regular process (e.g., they call sleep), and thus cannot
    // be run from main().
    first = 0;
801036b3:       c7 05 00 a0 10 80 00    movl   $0x0,0x8010a000
801036ba:       00 00 00
    iinit(ROOTDEV);
801036bd:       6a 01                   push   $0x1
801036bf:       e8 ac dd ff ff          call   80101470 <iinit>
    initlog(ROOTDEV);
801036c4:       c7 04 24 01 00 00 00    movl   $0x1,(%esp)
801036cb:       e8 e0 f3 ff ff          call   80102ab0 <initlog>
801036d0:       83 c4 10                add    $0x10,%esp
  }
```

Initlog함수 호출;

```
iinit(int dev)
{
  int i = 0;

  initlock(&icache.lock, "icache");
  for(i = 0; i < NINODE; i++) {
801014a4:       83 c4 10                add    $0x10,%esp
801014a7:       81 fb 40 26 11 80       cmp    $0x80112640,%ebx
801014ad:       75 e1                   jne    80101490 <iinit+0x20>
    initsleeplock(&icache.inode[i].lock, "inode");
  }

  readsb(dev, &sb);
801014af:       83 ec 08                sub    $0x8,%esp
801014b2:       68 c0 09 11 80          push   $0x801109c0
801014b7:       ff 75 08                pushl  0x8(%ebp)
801014ba:       e8 f1 fe ff ff          call   801013b0 <readsb>
  cprintf("sb: size %d nblocks %d ninodes %d nlog %d logstart %d\
801014bf:       ff 35 d8 09 11 80       pushl  0x801109d8
```

iinit에서 readsb호출

```
void
readsb(int dev, struct superblock *sb)
{
801013b0:       55                      push   %ebp
801013b1:       89 e5                   mov    %esp,%ebp
801013b3:       56                      push   %esi
801013b4:       53                      push   %ebx
801013b5:       8b 75 0c                mov    0xc(%ebp),%esi
  struct buf *bp;

  bp = bread(dev, 1);
801013b8:       83 ec 08                sub    $0x8,%esp
801013bb:       6a 01                   push   $0x1
801013bd:       ff 75 08                pushl  0x8(%ebp)
801013c0:       e8 0b ed ff ff          call   801000d0 <bread>
801013c5:       89 c3                   mov    %eax,%ebx
  memmove(sb, bp->data, sizeof(*sb));
```

Bread 함수호출

```
bread(uint dev, uint blockno)
{
  struct buf *b;

  b = bget(dev, blockno);
  if((b->flags & B_VALID) == 0) {
80100175:       f6 03 02                testb  $0x2,(%ebx)
80100178:       75 0c                   jne    80100186 <bread+0xb6>
    iderw(b);
8010017a:       83 ec 0c                sub    $0xc,%esp
8010017d:       53                      push   %ebx
8010017e:       e8 6d 1f 00 00          call   801020f0 <iderw>
80100183:       83 c4 10                add    $0x10,%esp
  }
  return b;
```

bread에서 iderw 호출

```
  call trap
801056b7:       e8 e4 00 00 00          call   801057a0 <trap>
  addl $4, %esp
801056bc:       83 c4 04                add    $0x4,%esp
```

Trap 부르고 esp 사이즈 4만큼 증가

```
1     break;
2   case T_IRQ0 + IRQ_IDE:
3     ideintr();
4 801059a0:       e8 ab c6 ff ff          call   80102050 <ideintr>
5     lapiceoi();
6 801059a5:       e8 66 cd ff ff          call   80102710 <lapiceoi>
7     break;
```

trap에서 lapiceoi 호출

```
7 void
8 ideintr(void)
9 {
0 80102050:       55                      push   %ebp
1 80102051:       89 e5                   mov    %esp,%ebp
2 80102053:       57                      push   %edi
3 80102054:       56                      push   %esi
4 80102055:       53                      push   %ebx
5 80102056:       83 ec 18                sub    $0x18,%esp
6   struct buf *b;
7
8   // First queued buffer is the active request.
9   acquire(&idelock);
0 80102059:       68 80 a5 10 80          push   $0x8010a580
1 8010205e:       e8 ed 22 00 00          call   80104350 <acquire>
2
3   if((b = idequeue) == 0){
4 80102063:       8b 1d 64 a5 10 80       mov    0x8010a564,%ebx
```

```
void
acquire(struct spinlock *lk)
{
  pushcli(); // disable interrupts to avoid deadlock.
  if(holding(lk))
    panic("acquire");
801043f0:       83 ec 0c                sub    $0xc,%esp
801043f3:       68 81 76 10 80          push   $0x80107681
801043f8:       e8 73 bf ff ff          call   80100370 <panic>
801043fd:       8d 76 00                lea    0x0(%esi),%esi
```

Acquire panic 발생

Iderw 함수에서 acquire와 release 사이에서 실행되는 코드를 보면

acquire후에 idequeue가 다음 buf를 가리키고, idestart 함수를 실행하는 코드가 있고 idestart 함수를 따라가보면 interrupt를 generate하는 코드들이 실행된다. acquire후에 interrupt를 허용하는 sti(); 코드를 추가 했기 때문에 lock 도중에 interrupt들이 허용되고 interrupt handler에서 acquire 하려다가 커널패닉이 발생할 것이다.

실제로 stack trace하다보면 패닉이 발생하기전 들어간 함수가 ideintr이고 그 후에 acquire에서 panic이 일어난 것을 알 수 있다.

Q3. 아마 file.c의 filealloc에서 iderw과 다르게 커널 패닉이 발생하지 않은 이유는 아마 acquire과 release 사이에 interrupt를 발생하는 함수가 실행되는지 아닌지의 차이일 것이다. iderw에서는 acquire을하면서 interrupt를 방해했지만 다시 sti(); 로 풀어줌으로써 interrupt들이 다시 acquire하는 상황이 있지만 filealloc에서는 acquire와 release 사이에 interrupt가 발생될 수 있는 함수를 호출하는 코드가 전혀 없다. 아마 f->ref 가 0인 것을 찾는 과정에서 ref가 그 사이에 바뀌는 것을 막기 위해 앞뒤로 lock을 하고 있을 것이다. 아마 acquire다음에 sti();를 넣음으로써 race condition이 발생할 수 있을 것이다.

Q4. 아마 lock을 잡기위해 대기하고 있는 다른 lock들이 그 사이에 lock이 hold된 상태인지 check해서 lock을 해버린다면 pcs와 cpu를 초기화 하지도 않고 lock을 잡아버려서 race condition이 발생할 수 있기 때문에 cpu와 pcs를 확실하게 초기화 한 후에 lock 상태 변수를 0으로 바꾸어 준다.

```
1 // Mutual exclusion lock.
2 struct spinlock {
3   uint locked;       // Is the lock held?
4
5   // For debugging:
6   char *name;        // Name of lock.
7   struct cpu *cpu;   // The cpu holding the lock.
8   uint pcs[10];      // The call stack (an array of program counters)
9                      // that locked the lock.
10 };
11
~
~
```