Homework 7: xv6 locking

2015005205

최홍규

Question 1: 위 코드를 수행했을 때, 어떤 일이 발생하는가?

-Deadlock 이 걸린다. 첫번째 acquire 에서 lock 이 잡힌다. 두번째 acquire 은 그 lock 이 풀릴 때까지 while 문을 계속해서 실행한다. 두번째 acquire 가 끝나고 release 가 되어야 lock 이 풀리는데 두번째 acquire 의 반복문이 끝나지 않으므로 Deadlock 이 발생한다. 즉, 첫번째 락이 풀리지 않고 두번째 락이 계속 loop 를 돌기 때문에 deadlock 이 발생한다.

Question 2: 위 상황에서 커널 패닉이 발생하는 이유가 무엇인가? 커널 패닉 상황에서 출력되는 스택 트레이스와 kernel.asm 파일을 참조하시오.

```
lapicid 0: panic: sched locks
 80103c01 80103d72 8010585b 8010564f 80100183 801013e5 801014df 80103714 80105652 0
```

```
373   if(mycpu()->ncli != 1)
374     panic("sched locks");
```

sched()함수의 373 번째 줄의 if 문에서 panic 이 발생한다는 것을 출력에 의해 알 수 있다.

스택 트레이스의 첫번째 주소 80103c01 는 panic()함수의 return address 이다. 이 앞 주소 함수에서 panic 이 발생 하였을 것이다.

두번째 주소 80103d72 는 sched()함수의 return address 이다. sched()함수 실행 도중 kernel panic 이 발생했다.

세번째 주소 8010585b 는 yield()함수의 return address 이다.

네번째 주소 8010564f 는 trap()함수의 return address 이다.

다섯번째 주소 80100183 는 iderw()함수의 return address 이다.

인터럽트를 활성화 시켜 둔 iderw()의 함수를 실행하기 때문에 iderw()함수 실행 도중 타이머 인터럽트가 발생하여 trap 함수로 빠진다. yield()함수에서 ptable.lock 을 걸고 sched()함수를 실행한다. sched()함수에서 373 번째 줄의 조건문에서 현재 걸려있 lock 의 수가 1 개가 아니면 panic 을 발생시킨다.

iderw()에서 idelock을 걸고 인터럽트를 활성화 시킨 뒤 넘어오기 때문에 sched()함수 실행 할 때 idelock, ptable.lock 두개가 걸려있다. 그렇기 때문에 커널 패닉이 발생한다. 즉, sched()함수를 실행할 때 두 개의 lock이 걸려있기 때문에 커널 패닉이 발생한다.

Question 3: 커널 패닉이 발생하지 않는 이유가 무엇인가? 왜 ftable.lock 과 idelock 이 다르게 동작하는가?

filealloc()과 iderw()의 함수 실행에 걸리는 시간이 다르기 때문이다. filealloc()은 비어있는 파일테이블이 있다면 연결 후 종료한다. 하지만 iderw()는 작업이 끝날 때까지 sleep을 하기 때문에 filealloc()함수보다 시간이 더 걸릴 것이다. 따라서 iderw()함수 실행 중 타이머 인터럽트가 발생하여 panic에 빠지게 되는데 filealloc()함수의 경우는 iderw()함수의 비해 실행하는 시간이 더 짧아서 타이머 인터럽트가 발생하기 전에 끝나기 때문에 panic이 일어나지 않는다. 작은 확률로 panic이 생기는 이유는 가끔 타이머 인터럽트가 filealloc()함수 실행 때 발생하기 때문이다.

Question 4: release () 함수가 lk->locked 를 0 으로 초기화하여 락을 해제하기 전에, lk->pcs[0]와 lk->cpu 를 초기화하는 이유는 무엇인가? 왜 락을 해제한 후, 초기화하지 않는가?

```
107 panic(char *s)
108 {
109   int i;
110   uint pcs[10];
111
112   cli();
113   cons.locking = 0;
114   // use lapiccpunum so that we can call panic from mycpu()
115   cprintf("lapicid %d: panic: ", lapicid());
116   cprintf(s);
117   cprintf("\n");
118   getcallerpcs(&s, pcs);
119   for(i=0; i<10; i++)
120     cprintf(" %p", pcs[i]);
121   panicked = 1; // freeze other CPU
122   for(;;)
123     ;
124 }
```

lock을 해제하고 lk->pcs[0]와 lk->cpu를 초기화하면 다른 프로세서에서 두 변수를 초기화하기 전에 그 락을 가져가 버릴 수 있다. 하지만 다른 프로세스의 acquire()함수에서 그 값들을 새로운 값들로 덮어 쓰기때문에 큰 문제가 되지

않는다. 하지만 만약 새로운 값들로 lk->pcs, cpu 을 덮어쓰기 전에 panic 이 발생한다면 문제가 발생할 수 있다. panic 이 발생하면 panic 함수가 실행되는데 panic 함수에서 출력하는 pcs 값이 panic 이 발생한 프로세스의 값이 아닌 락을 사용하던 그전 프로세스의 pcs 값을 출력하게 된다. 이러한 경우가 발생할 수 있으므로 lock 의 변수들을 안전하게 초기화해주기 위해 lock 을 해제하기 전에 모든 변수들을 초기화해준 후 lock 을 해제해준다.