운영체제 Race Condition and Mutex Lock 보고서

미디어학과 201700000 박성범

1.1. 개요

두 스레드 사이에서 race condition을 발생시키고, 이를 mutex lock으로 해결하는 것을 목적으로 개발한 프로그램이다. 3개의 스레드가 global variable인 counter의 값에 100,000번 접근하고, counter의 값을 1씩 증가시킨다. 실행 로그는 event.log 파일에 남는다.

1.2. 프로그램 설계 및 주요 부분

1.2.1. 스레드 생성

```
for (i = 0; i < 100000; i++) {
    pthread_create(&threads[0], NULL, performThread, (void*)threadNames[0]);
    pthread_create(&threads[1], NULL, performThread, (void*)threadNames[1]);
    pthread_create(&threads[2], NULL, performThread, (void*)threadNames[2]);
}
```

각 스레드를 100,000번 생성해 performThread 함수의 내용을 수행한다.

1.2.2. 스레드 작업

```
void* performThread(void* data) {
    pthread_mutex_lock(&mutex);

    time_t currentTime;
    struct tm* timeInfo;
    char currentTimeString[128];
    char* threadName = (char*)data;

    time(&currentTime);
    timeInfo = localtime(&currentTime);
    strftime(currentTimeString, 128, "%Y-%m-%d %H:%M:%S", timeInfo);

    fprintf(file, "%s\t%s\t%d\n", currentTimeString, threadName, counter);

    counter++;

    pthread_mutex_unlock(&mutex);
}
```

함수 진입에 앞서 mutex lock으로 다른 스레드가 critical section에 진입하지 못하도록 막는다.

만약 lock이 되어있지 않다면 함수 내로 진입해 로그를 남기고 counter 값을 증가시킨다. 함수를 나가기 직전에는 mutex unlock으로 다른 스레드가 진입할 수 있도록 한다.

1.3. 프로그램 실행 조건

1.3.1. 실행 및 테스트 방식

3개의 스레드를 통해 counter를 증가시키는 main 프로그램과 별개로, main 프로그램이 counter 증가를 마치면 로그를 분석해 race condition이 발생했는지 확인하는 checker 프로그램이 실행되도록 했다. checker는 event.log 파일에서 n번 라인의 counter 값에 1을 더한 값이 n + 1번 라인의 counter 값과 같은 지 비교해 race condition 발생 여부를 체크한다. checker는 race condition이 발생할 때까지 main의 실행과 event.log의 삭제를 반복하도록 만들었다.

1.3.2. 실행 환경

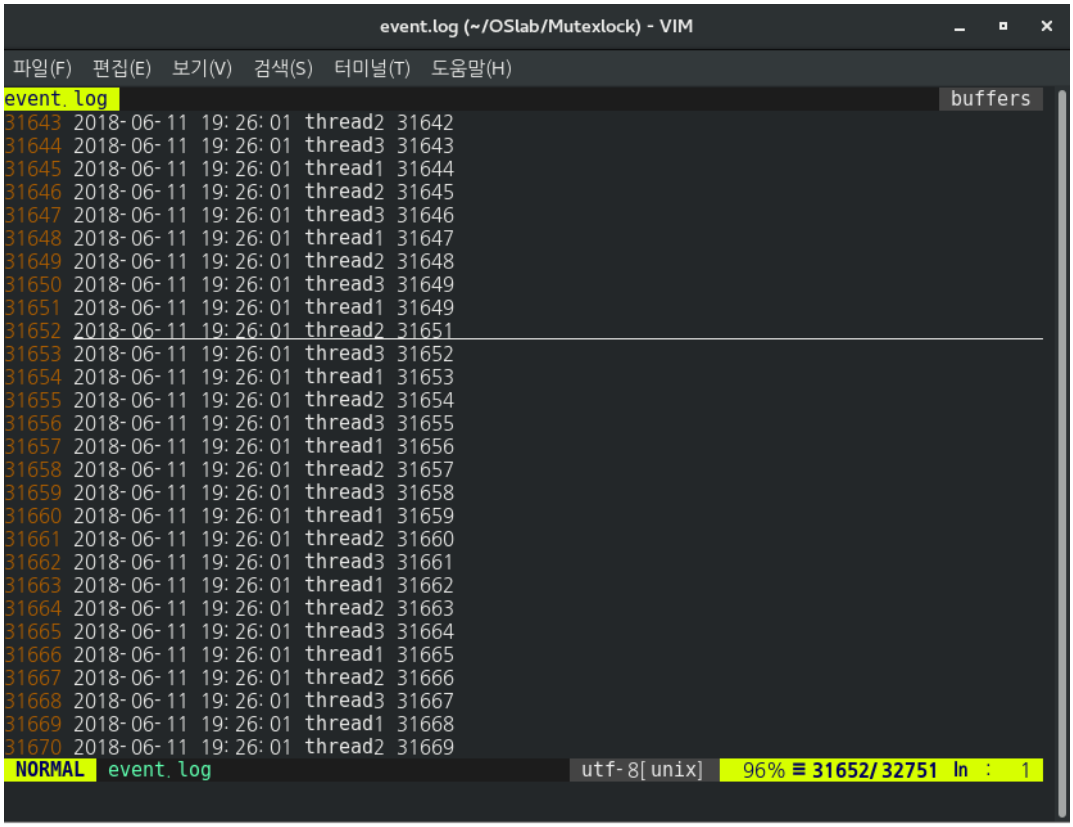
- CentOS 7 on virtual machine. (Oracle VM Virtual Box)
- Windows 10 on physical machine.

1.4. 프로그램 실행 결과 및 비교 분석

1.4.1. mutex lock 미작동 (v1)

```
[parksb@localhost Mutexlock]$ ./debug
Done: 32753
A race condition does not occur.
Makefile  checker  checker.c  debug  main.c  release
Done: 32752
A race condition does not occur.
Makefile  checker  checker.c  debug  main.c  release
Done: 32751
A race condition occurs between 31649 and 31651
[parksb@localhost Mutexlock]$ vim event.log
[parksb@localhost Mutexlock]$ vim event.log
[parksb@localhost Mutexlock]$
```

CentOS에서는 counter 값이 약 32,750 정도까지 증가했으며, 여러 시도 끝에 race condition이 발생했다. counter의 최종 값이 예상 값인 300,000보다 훨씬 낮은 것과 값이 일관적이지 않음에도 race condition이 발견되지 않은 것은 후술할 virtual machine 문제 때문인 것으로 보인다.



event.log 파일을 열어 31650 라인과 31652 라인 사이에서 counter 값이 잘못된 것을 확인했다.

```
2018-06-11 18:06:01 thread3   0
2018-06-11 18:06:01 thread2   1
2018-06-11 18:06:01 thread1   3
2018-06-11 18:06:01 thread2   4
2018-06-11 18:06:01 thread1   5
2018-06-11 18:06:01 thread2   7
2018-06-11 18:06:01 thread3   8
2018-06-11 18:06:01 thread1   9
2018-06-11 18:06:01 thread3   10
2018-06-11 18:06:01 thread2   11
2018-06-11 18:06:01 thread1   15
2018-06-11 18:06:01 thread3   17
2018-06-11 18:06:01 thread1   18
2018-06-11 18:06:01 thread2   19
2018-06-11 18:06:01 thread2   22
2018-06-11 18:06:01 thread3   24
2018-06-11 18:06:01 thread2   25
2018-06-11 18:06:01 thread2   28
2018-06-11 18:06:01 thread1   30
2018-06-11 18:06:01 thread2   31
2018-06-11 18:06:01 thread3   31
2018-06-11 18:06:01 thread1   34
2018-06-11 18:06:01 thread1   38
2018-06-11 18:06:01 thread2   41
2018-06-11 18:06:01 thread2   49
2018-06-11 18:06:01 thread3   50
2018-06-11 18:06:01 thread1   51
2018-06-11 18:06:01 thread3   53
2018-06-11 18:06:01 thread1   54
```

```
2018-06-13 08:37:49thread2      0
2018-06-13 08:37:49thread3      0
2018-06-13 08:37:49thread2      2
2018-06-13 08:37:49thread1      3
2018-06-13 08:37:49thread3      4
2018-06-13 08:37:49thread1      5
2018-06-13 08:37:49thread1      6
2018-06-13 08:37:49thread2      7
2018-06-13 08:37:49thread3      8
2018-06-13 08:37:49thread1      9
2018-06-13 08:37:49thread2      10
2018-06-13 08:37:49thread3      11
2018-06-13 08:37:49thread1      12
```

CentOS에서 테스트했을 때와 달리 Windows에서는 counter 값이 약 300,000까지 증가했다.

좌측 데이터는 checker가 필요하지 않을 정도로 빠르게, 자주 race condition이 발생했으며, 우측 데이터는 테스트 시작부분에서 race condition이 발생했다.

```
for (i = 0; i < 100000; i++) {
    pthread_create(&threads[0], NULL, performThread, (void*)threadNames[0]);
    pthread_create(&threads[1], NULL, performThread, (void*)threadNames[1]);
    pthread_create(&threads[2], NULL, performThread, (void*)threadNames[2]);

    pthread_join(threads[0]);
    pthread_join(threads[1]);
    pthread_join(threads[2]);
}
```

```
24001:  72006
24002:  72009
24003:  72012
24004:  72015
24005:  72018
24006:  72021
24007:  72024
24008:  72027
24009:  72030
```

CentOS와 Windows에서 counter 값의 차이가 생기는 현상은 CentOS에서 프로그램을 구동했을 때 각 스레드는 100,000회 수행을 반복하지만, counter 값이 약 32,750에 도달하면 더 이상 값을 증가시키지 않기 때문에 발생하는 것으로 확인되었다. 확실하지는 않지만, 이는 CentOS를 virtual machine 위에서 구동하기 때문에 일정한 term 없이 스레드 생성을 반복하면 퍼포먼스에 제약이 생기기 때문인 것으로 추정된다. 위 사진처럼 스레드 생성 후 pthread_join 함수를 통해 각 스레드의 종료를 기다린 뒤 반복을 이어서 수행하면 counter 값이 계속 증가했으며, race condition도 빠르게 발생했다. (로그는 “시도 횟수: counter 값” 형식.)

1.4.2. mutex lock 작동 (v2)

2018-06-12 19:26:32	thread1	0	2018-06-12 19:56:09	thread3	299978
2018-06-12 19:26:32	thread2	1	2018-06-12 19:56:09	thread1	299979
2018-06-12 19:26:32	thread3	2	2018-06-12 19:56:09	thread2	299980
2018-06-12 19:26:32	thread1	3	2018-06-12 19:56:09	thread3	299981
2018-06-12 19:26:32	thread2	4	2018-06-12 19:56:09	thread1	299982
2018-06-12 19:26:32	thread3	5	2018-06-12 19:56:09	thread2	299983
2018-06-12 19:26:32	thread1	6	2018-06-12 19:56:09	thread3	299984
2018-06-12 19:26:32	thread2	7	2018-06-12 19:56:09	thread1	299985
2018-06-12 19:26:32	thread3	8	2018-06-12 19:56:09	thread2	299986
2018-06-12 19:26:32	thread1	9	2018-06-12 19:56:09	thread3	299987
2018-06-12 19:26:32	thread2	10	2018-06-12 19:56:09	thread1	299988
2018-06-12 19:26:32	thread3	11	2018-06-12 19:56:09	thread2	299989
2018-06-12 19:26:32	thread1	12	2018-06-12 19:56:09	thread3	299990
2018-06-12 19:26:32	thread2	13	2018-06-12 19:56:09	thread1	299991
2018-06-12 19:26:32	thread3	14	2018-06-12 19:56:09	thread2	299992
2018-06-12 19:26:32	thread1	15	2018-06-12 19:56:09	thread3	299993
2018-06-12 19:26:32	thread2	16	2018-06-12 19:56:09	thread1	299994
2018-06-12 19:26:32	thread3	17	2018-06-12 19:56:09	thread2	299995
2018-06-12 19:26:32	thread1	18	2018-06-12 19:56:09	thread1	299996
2018-06-12 19:26:32	thread2	19	2018-06-12 19:56:09	thread3	299997
2018-06-12 19:26:32	thread3	20	2018-06-12 19:56:10	thread2	299998
2018-06-12 19:26:32	thread1	21	2018-06-12 19:56:10	thread3	299999
2018-06-12 19:26:32	thread2	22			

Windows에서 프로그램을 실행시키고 event.log 파일을 분석한 결과 race condition이 발생하지 않았으며, 스레드가 순서대로 실행되었다. counter 값은 0부터 299999까지 증가해 각 스레

드가 100,000번 실행되었음을 확인할 수 있었다. mutex lock을 작동시키지 않았을 때 곳곳에서 race condition이 발생하던 것과 상반된다.

앞서 Cent OS 테스트에서 발견된 virtual machine 문제 때문에 mutex lock을 이용한 테스트는 Windows에서만 진행했다.