집중교육2- Deferred work (Lab13 보고서)

201521032 한태희

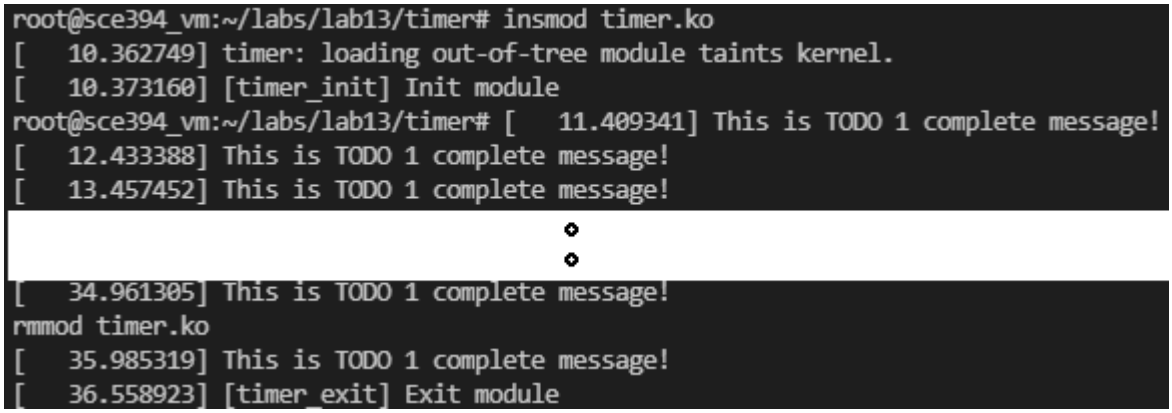
0. 과제 개요

Lab13은 3가지 코드를 완성하는 작업을 진행했다. 첫번째로 timer.c에서 리눅스 인터럽트 타이머, kthread.c에서 커널 스레드, waitqueue를 이용하여 간단한 예제를 만들어 보고, defeered.c에서는 workqueue API까지 이용하여 코드를 완성했다. 과제 테스트 케이스 (목표)가 분할되어 있으므로 각 테스트 케이스를 달성하기 위한 구현과 검증을 따로 기술한다.

1. timer

timer.c는 커널 모듈에서 타이머 인터럽트를 이용하여 주기적으로 프린트문을 출력하는 코드를 작성하였다. 모듈은 타이머 인터럽트를 예약해서 timer_handler가 1초 뒤에 실행되도록 하는데, timer_handler 함수는 타이머 인터럽트를 다시 예약해서 1초 뒤 자신이 재귀적으로 실행되게 해 이를 무한히 반복한다.

아래 사진은 timer/timer를 실행시긴 결과이다. 과제의 TODO 이름을 착각해 잘못된 텍스트를 출력했지만, 의도한 출력은 완성한 상태이다. 정확히 1초는 아니지만 1초에 가까운 시간 주기로 텍스트 출력이 나오는 것을 확인할 수 있다.



2. kthread

kthread.c는 커널 모듈에서 커널 스레드를 생성하고 생성된 커널 스레드가 모듈의 종료를 기다렸다가, 모듈이 종료될 때 ACK를 보내 모듈과 함께 동기적 종료를 하도록 하는 코드를 작성하였다. 이때, 모듈 코드와 커널 스레드의 종료 동기화를 위해서 wait queue api를 사용하였다.

아래 사진은 kthread/kthread.c를 실행한 결과이다. 모듈을 제거하면 khread도 같은 타이밍에 종료되는 것을 확인할 수 있다.

```
root@sce394_vm:~/labs/lab13/kthread# insmod kthread.ko
[   21.054015] kthread: loading out-of-tree module taints kernel.
[   21.063422] [kthread_init] Init module
root@sce394_vm:~/labs/lab13/kthread# [   21.073579] [my_thread_f] Current process id is 236 (mykthread0)
root@sce394_vm:~/labs/lab13/kthread# rmmod kthread.ko
[   26.709024] [my_thread_f] Exiting
[   26.709444] [kthread_exit] Exit module
```

3. Deffered

3.1. TODO 1 (TIMER_TYPE_SET)

Deffered 파일의 첫번째 요구사항은 TIMER_TYPE_SET 기능을 구현하는 것이다. 커널 모듈은 ioctl을 이용해 유저 코드에게서 타이머 길이를 초 단위의 정수로 받은 다음, 해당 초 뒤에 메시지가 출력되도록 한다.

아래 사진은 유저 코드에서 s 인자를 이용해 TIMER_TYPE_SET을 1초뒤, 그리고 2초뒤에 실행시킨 결과이다. 결과 메시지가 각각 약 1초 뒤, 약 2초뒤에 출력되는 것을 확인할 수 있다.

```
root@sce394_vm:~/labs/lab13/deferred# user/test s 1
[   13.324780] [deferred_open] Device opened
Set timer to 1 seconds
[   13.330412] [deferred_ioctl] Command: Set timer
[   13.330757] [deferred_release] Device released
root@sce394_vm:~/labs/lab13/deferred# [   14.391633] TIMER_TYPE_SET Handler is activated.

root@sce394_vm:~/labs/lab13/deferred# user/test s 2
[   18.207151] [deferred_open] Device opened
Set timer to 2 seconds
[   18.207946] [deferred_ioctl] Command: Set timer
[   18.208157] [deferred_release] Device released
root@sce394_vm:~/labs/lab13/deferred# [   20.215382] TIMER_TYPE_SET Handler is activated.
```

3.2. TODO 2 (ERROR OF TIMER_TYPE_ALLOC)

Deffered 파일의 첫번째 요구사항은 TIMER_TYPE_ALLOC을 구현하는 것이다. ALLOC 명령이 들어왔을 때는 SET과 동일하게 타이머 예약이 진행되지만, 단순 문자열을 출력하는 것이 아닌 뼈대 코드로 주어진 alloc_io 함수를 실행하게 된다. alloc_io 함수는 태스크 스케줄링을 통해 약 5초간 정지하는 함수인데, 인터럽트 컨텍스트에서는 이런 작동이 금지되어 있으므로 에러가 발생하게 된다.

```
[   50.809406] bad: scheduling from the idle thread!
[   50.809718] CPU: 0 PID: 0 Comm: swapper/0 Tainted: G        W  O      5.15.4 #7
[   50.810230] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.10.2-1ubuntu1 04/01/2014
[   50.810818] Call Trace:
[   50.811060]  <TASK>
[   50.811239]  dump_stack_lvl+0x33/0x42
[   50.811561]  dequeue_task_idle+0x1b/0x30
[   50.811888]  __schedule+0x3a7/0x6e0
[   50.812173]  schedule_idle+0x17/0x30
[   50.812450]  do_idle+0x184/0x280
[   50.812734]  cpu_startup_entry+0x14/0x20
[   50.813025]  start_kernel+0x652/0x679
[   50.813334]  secondary_startup_64_no_verify+0xc2/0xcb
[   50.813731]  </TASK>
```

3.3. TODO 3 (CORRECTION OF TIMER_TYPE_ALLOC)

3.2. 에서 발생한 문제를 해결하기 위해서는 인터럽트 컨텍스트인 timer_handler가 아닌 커널 권한으로 작동하는 일반 프로세스 컨텍스트에서 alloc_io 함수를 작동시키는 우회책을 사용해야 한다. 이런 인터럽트 컨텍스트의 스케줄링 불가능 속성으로 인해 프로세스가 버그를 일으키거나 성능을 저하시키는 상황을 방지하기 위해 인터럽트는 아주 중요한 최초의 작업만 처리하고, 급하지 않은 작업은 일반 프로세스 컨텍스트에서 진행하도록 만드는 기법을 top half/bottom half 라고 부른다. 이번 과제에서는 bottom half를 work queue API를 이용해 작성했다. 인터럽트 핸들러가 미리 정의된 work queue에 요청을 보내면 프로세스 컨텍스트가 handler에서 alloc_io를 실행시킨다.

아래 사진은 유저 코드에서 a 인자를 이용해 1초 뒤에 alloc_io 함수의 실행을 요청하고 약 6초 (인터럽트 대기 시간 1초 + alloc_io 대기 시간 5초) 뒤에 메시지가 출력되는 결과물이다.

```
root@sce394_vm:~/labs/lab13/deferred# ./user/test a 1
[   29.061181] [deferred_open] Device opened
Allocate memory after 1 seconds
[   29.066458] [deferred_ioctl] Command: Allocate memory
[   29.066763] [deferred_release] Device released
root@sce394_vm:~/labs/lab13/deferred# [   35.448765] Yawn! I've been sleeping for 5 seconds.
```

3.4. TODO 4 (TIMER_TYPE_MON)

MON 명령어는 앞의 두 경우와 달리 유저가 인자로 프로세스의 PID를 입력해야 한다. 커널 모듈은 해당 get_proc 함수를 이용해 해당 pid를 가진 task struct를 가져오고, task struct를 데이터로 가진 LIST에 집어넣는다. 그 뒤에 MON flag를 세팅하고 타이머 인터럽트를 예약한다. MON 명령일 경우, 타이머는 주기적으로 반복되며, list에 들어있는 태스크들이 종료 상태인지 주기적으로 검사한다. 태스크가 종료되었을 경우, 태스크가 종료되었다는 문자열을 출력하고 리스트에서 노드를 삭제한다. 이때, 커널 모듈과 인터럽트 핸들러가 동시에 리스트에 접근하면 동기화 문제가 발생하므로 spin lock을 이용하여 리스트에는 오직 한 컨텍스트만 접근할 수 있도록 제한하였다.

아래 사진은 유저 코드에서 p 인자를 이용하여 MON 명령을 실행시킨 결과이다. 우선 sleep 1을 반복하는 쉘 스크립트인 sleep.sh를 실행하고, 해당 프로세스의 pid를 알아낸 다음 모듈에 ./test p <pid> 입력으로 MON 명령을 실행한다. kill <pid> 명령어를 이용하여 리스트에 등록된 프로세스를 강제적으로 종료한 직후, 주기적 타이머 인터럽트가 태스크의 종료를 감지하고 태스크가 종료되었음을 알린다.

```
root@sce394_vm:~/labs/lab13/deferred# kernel/makenode
crw-r--r--    1 root     root       42,   0 Dec  7 09:25 /dev/deferred
root@sce394_vm:~/labs/lab13/deferred# insmod kernel/deferred.ko
[   29.715598] deferred: loading out-of-tree module taints kernel.
[   29.724847] [deferred_init] Init module
root@sce394_vm:~/labs/lab13/deferred# user/sleep.sh &
root@sce394_vm:~/labs/lab13/deferred#
root@sce394_vm:~/labs/lab13/deferred# ps | grep sleep.sh
  246 root      7836 S    {exe} ash user/sleep.sh
  254 root      7836 S    grep sleep.sh
root@sce394_vm:~/labs/lab13/deferred# user/test p 246
[   41.525614] [deferred_open] Device opened
Monitor PID 246.
[   41.532589] [deferred_ioctl] Command: Monitor pid
[   41.532996] [deferred_release] Device released
root@sce394_vm:~/labs/lab13/deferred# kill 246
[1]+  Terminated                 user/sleep.sh
root@sce394_vm:~/labs/lab13/deferred# [   57.909382] monitored task pid:246 is dead!!!
```