2015-2 System Programing

Kernel Lab

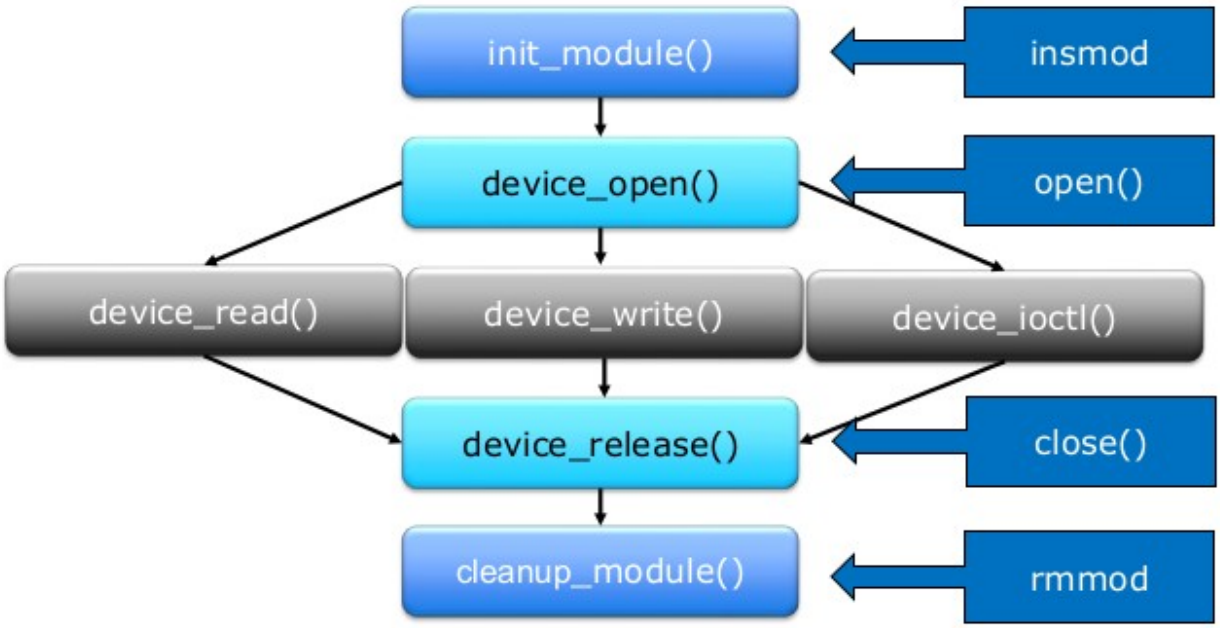
2010-13419
이용재

1. Linux device module

kernel 에 새로운 기능을 구현하려고 할 때마다 커널코드를 새로 짜고 새로 부팅하는 방식은 효율적이지도 않고 비용이 많이 든다. 그래서 linux 에서는 kernel 모드에서 실행할 기능들을 module 화로 구현하여 따로 업로드하고 실행되도록 하고 있다.

Device module 이란 이중 대표적으로 device 와의 입출력을 담당하는 module 이다. driver 를 구현하고 linux/fs.h 에 정의돼있는 register_chrdev 함수를 통해 kernel 에 등록한다. 이때 major number 는 해당 device module 을 인식하기 위한 것으로 unique 한 숫자를 배당해줘야 한다.

■ Device Driver Progress Flow



shell 에서 insmod *module_name.ko* 를 통해 module 을 load 하면 차례로 init_module(), device_open()이 실행되며 사용할수 있는 상황이 된다. device_read(), device_write()를 통해 driver 가 담당하는 device file 에 입출력을 할수 있으며, device_ioctl()을 통해 driver 자체와 정보를 주고받을 수 있다. 우리는 이 ioctl 함수를 통해 kernel level 의 시스템 함수를 활용할 것이다.

2. Ptree & Profiling

1) ptree 그리기

i. [profiler.c]
ioctl(fd, IOCTL_GET_PTREE, &ptree);
호출을 통해 chardev.h 에 정의돼 있는
struct PtreeInfo { char names[20][20]; } ptree; 의 포인터를 chardev 에 전달한다.

ii. [chardev.c]
struct PtreeInfo* ptree = (struct PtreeInfo*)ioctl_param;
로 전달된 ptree 포인터를 받는다.

```
struct task_struct* tmptask = current;
while(tmptask->pid!=0){
```

```
    mystrcpy(ptree->names[i++], tmptask->comm);
    tmptask=tmptask->parent;
}
ptree->names[0][0]=i;
```

task_struct* tmptask 에 kernel 의 글로벌 변수 current 를 통해 현재 task 의 task_struct 포인터를 받고 이후 루프를 돌며 parent 를 계속 찾아서 task_struct.comm 에 저장돼 있는 프로세스 명을 ptree->names 에 차례대로 저장한다. mystrcpy 함수를 구현해서 char* 복사에 활용했다.
마지막으로 프로세스 총 갯수를 names[0][0]에 저장한다.

iii. [profiler.c]
ptree.names 에 저장된 문자열들을 저장한 반대 순서로 차례대로 출력한다.

2)profiling

i. [chardev.c]
```
static long long read_msr(unsigned int ecx) {
  unsigned int edx = 0, eax = 0;
  unsigned long long result = 0;
  __asm__ __volatile__("rdmsr" : "=a"(eax), "=d"(edx) : "c"(ecx));
  result = eax | (unsigned long long)edx << 0x20;

  return result;
}

static void write_msr(int ecx, unsigned int eax, unsigned int edx) {
    __asm__ __volatile__("wrmsr" : : "c"(ecx), "a"(eax), "d"(edx));
}
```

---
inline assembly 를 통해 인자를 받아 msr 에 사용하는 함수를 구현한다.

ii. [profiler.c]
```
ioctl(fd, IOCTL_MSR_CMDS, (long long)msr_init);
ioctl(fd, IOCTL_MSR_CMDS, (long long)msr_set_start);

if((pid=fork())==0){
  execvp(argv[1], argv);
  return -1;
} else {
  waitpid(-1,NULL,0);
}

ioctl(fd, IOCTL_MSR_CMDS, (long long)msr_stop_read);
print_profiling(msr_stop_read[6].value, msr_stop_read[4].value,
msr_stop_read[7].value);
```

---
ioctl 호출을 통해 msr 초기화, PMU profiling 을 시작하는 명령을 전달한 뒤 child process 를 생성해 execvp 에 커맨드라인에 입력받은 파일경로를 전달한다. parent process 는 child process 가 실행을 마치기까지 기다린다. 실행이 끝난 뒤 ioctl 호출을 통해 msr 동작을 멈추고, 그 결과를 msr_stop_read 에 저장한다.

print_profiling 함수에 순서대로 executed instruction, stalled cycles, core cycles 인자를 전달해 형식에 맞게 출력한다.

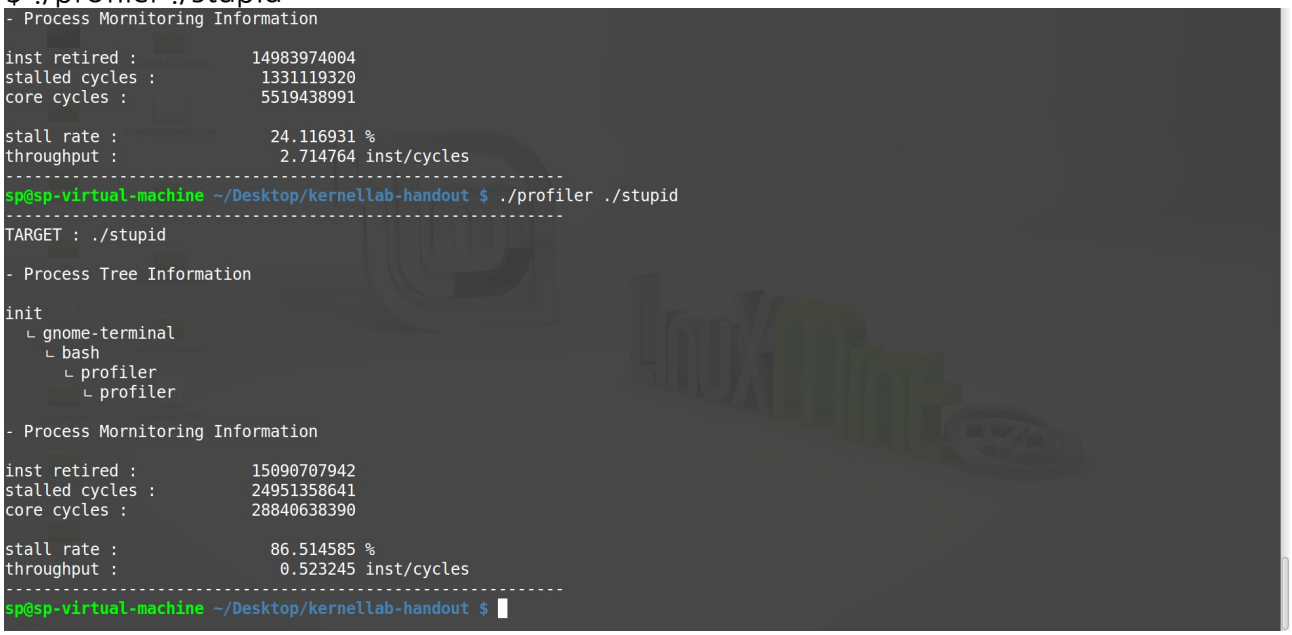
iii. MSR

```
struct MsrInOut {
  unsigned int op;          // MsrOperation
  unsigned int ecx;         // msr identifier
  union {
    struct {
      unsigned int eax;     // low double word
      unsigned int edx;     // high double word
    };
    unsigned long long value; // quad word
  };
};
```

MSR 인자를 전달하는 struct MsrInOut 의 구성이다. 이를 통해 전달된 명령을 chardev 의 read_msr, write_msr 을 통해 실행하는 것이다.

실행화면

$ ./profiler ./stupid



$ ./profiler ./clever

```
sp@sp-virtual-machine ~/Desktop/kernellab-handout $ ls
chardev.c       chardev.h.org  chardev.mod.o  clever_bench.c  modules.order   profiler.c      stupid_bench.c
chardev.c.org   chardev.ko     chardev.o      hello.c         Module.symvers  profiler.c.org  test.c
chardev.h       chardev.mod.c  clever         Makefile        profiler        stupid
sp@sp-virtual-machine ~/Desktop/kernellab-handout $ ./profiler ./clever
------------------------------------------------------------
TARGET : ./clever

- Process Tree Information

init
  └ gnome-terminal
    └ bash
      └ profiler
        └ profiler

- Process Mornitoring Information

inst retired :            14983974004
stalled cycles :           1331119320
core cycles :              5519438991

stall rate :                 24.116931 %
throughput :                  2.714764 inst/cycles
------------------------------------------------------------
sp@sp-virtual-machine ~/Desktop/kernellab-handout $ ./profiler ./stupid
------------------------------------------------------------
TARGET : ./stupid

- Process Tree Information
```

3. Difficulties

getptree 를 구현할때, chardev 에서 시스템 콜을 통해 부모프로세스 이름을 얻는 데에 성공했지만 이것을 user level 의 profiler 로 옮기는 데에서 어려웅믈 겪었다. ioctl 을 통해 struct PtreeInfo*를 전달했지만 PtreeInfo 를 어떻게 구성해서 어떻게 전달할지가 잘 떠오르지 않아서 막혔다. PtreeInfo 자체를 링크드리스트의 노드처럼 구현할지 아니면 PtreeInfo 안에 링크드리스트를 포함시킬지 등을 생각해봤는데 잘 되지 않았고 무엇보다 chardev 에서 malloc 함수를 사용할수 없어서 다른 방법을 생각해봤다. 그냥 char* 배열을 통해 task_struct->comm 의 주소를 받는 방법을 생각해봤지만 주소를 옮길수는 있어도 user level 에서 접근할수 없는 주소값이기 때문에 또 실행이 되지 않았다. 결국 가장 간단하게 적당한 크기(20*20)의 2 차원 char 배열을 만들어서 여기에 char 하나하나를 옮기는 방법을 사용했다. 지금 생각해보면 driver 의 특성을 활용해 file 에 프로세스명을 입출력함으로써 낭비적인 배열 선언을 방지할수 있을것 같다는 생각이 든다.

또 profiler 를 구현할 때에는 msr_init, msr_start, msr_stop 은 어렵지 않게 뼈대코드와 reference 를 참고해 쉽게 만들수 있었지만 입력받은 binary 를 어떻게 실행시켜야 하는지 잘 이해하지 못해서 헤맸다. 코드를 짜고 난 뒤에도 개인 랩탑을 사용할때는 실행이 잘 되지 않아서 디버그를 잘 못하고 있었다. 수업 TA 의 도움을 받아 가상머신에서 제대로 활용할수 있다는 것을 알게 되어 실행해볼수 있었다.

수행하면서 많은 어려움이 있었던 과제였지만 그만큼 C 에 대해서도 kernel 에 대해서도 device 에 대해서도 이해가 넓어지는 과제였다.