**OS (Operating System)\_1**

**Term**

**작업환경: ubuntu-14.04.4-desktop-amd64,**

**VirtualBox-5.0.16-105871-Win**

**컴퓨터학과**

**2014210057 최준명**

**제출 일자: 2016.03.31**

**A. 목적**

리눅스에 새로운Kernel을 컴파일해서 새로운System Call을추가 하는 것이다.

Kernel을 직접 컴파일하고, 이후 기존의 Kernel에 새로운System Call을 추가한 후에 이를 호출할 수 있는 User-Application까지를 구현 하도록 한다.

**B. 세부사항**

**a. System Call(시스템 콜)**

System Call이란 Kernel이 시스템이나 하드웨어에 특권이 있어야만 하는 동작들을 지정해둔 함수로써 User-Space에서 Kernel-Space로 전환하기 위해서 특별히 지정된 Trap Instruction을 거쳐야한다.

**b. Kernel에서 문자열 출력**

Kernel 영역에서는 기존 C에서 사용하던 printf를 사용하지 못한다. 커널영역에서 메시지를 출력 하려면 printk를 사용해야한다. System Call에서printk를 통해 메시지를 출력했다면 최근에 발생한 커널 메시지를 확인하는 dmesg 명령어를 통해 확인할 수 있다.

**c. Interrupt(인터럽트)**🡺 Interrupt 정의

H/W는 Interrupt를 통해 Process에 보내는 신호를 Interrupt라고하고 클럭과 관계없이 발생 가능하다.

🡺 Interrupt Descriptor Table 구성

0x0 ~ 0x1F - Trap Handler

0x20 ~ 0x7F - Interrupt Handler(IRQ)

0x80 - System Call

🡺 Interrupt 과정

1. 하드웨어로부터 나오는 전기적 신호(Interrupt)는 인터럽트 컨트롤러에 전송  
2. 인터럽트 컨트롤러는 프로세서에 신호를 전달  
3. 신호를 감지하면 프로세서는 인터럽트 처리를 위해 현재 실행하던 일을 중단  
4. 운영체제에 인터럽트 발생 사실 전달  
5. 운영체제는 상황에 맞게 정리

**d. Linux Interrupt Handler (인터럽트 핸들러)**

인터럽트를 처리하기 위해 커널이 실행하는 함수, 인터럽트가 발생하였을 때 커널이 호출한다는 점과 인터럽트 컨텍스트라는 특별한 컨텍스트에서 실행된다는 점입니다. 인터럽트는 언제든 발생할 수 있기 때문에 중단된 코드로 다시 돌아가기 위해서는 핸들러의 속도도 빨라야 합니다.

**e. asmlinkage**

gcc 확장 기능으로 assemble + linkage가 합쳐진 단어입니다. 어셈블리와 C언어를 연결하여 C언어로 작성된 함수를 호출 할 수 있게 해주는 기능을 갖고 있습니다.

**f. System Call Handling**

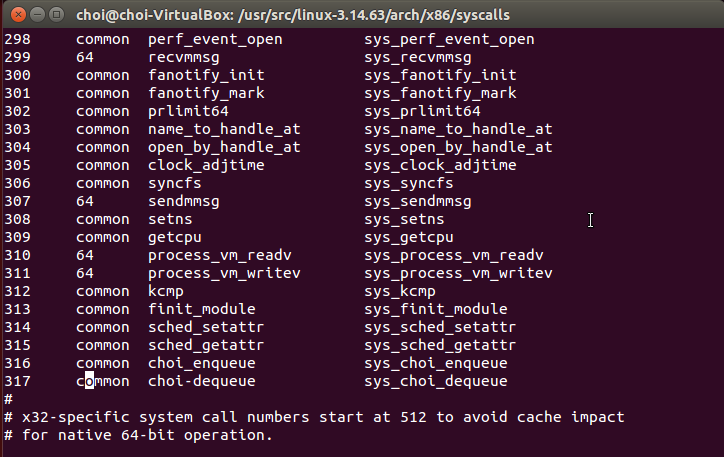
시스템 호출을 처리하기 위해 실행하는 함수를 의미합니다.

**C. 수행 과정**

**a. handout에 명시된 4개의 파일 수정 및 application 생성**

1. **syscall\_64.tbl**

🡺 캡쳐 화면



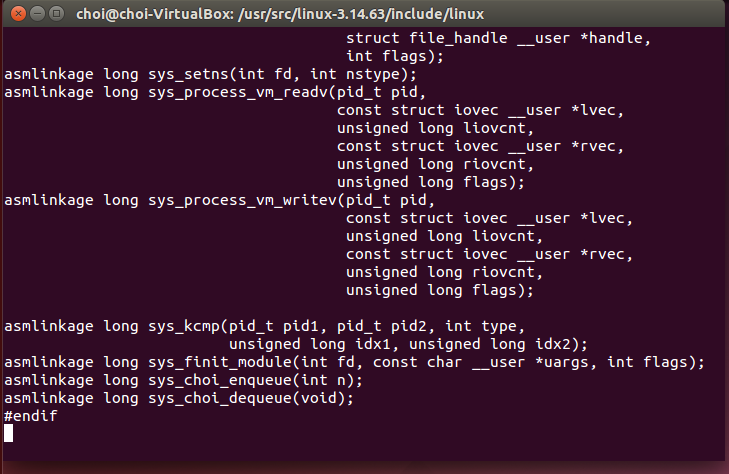
🡺 기능

시스템 콜 함의 고유번호를 저장해 놓은 파일, 추가할 시스템 콜의 번호를 입력해야 함.

🡺 수정한 부분  
316번에 enqueue함수를 317번에 dequeue함수를 지정하였습니다. My\_queue\_syscall.c 파일에서 두 함수의 이름을 ‘sys\_choi\_enqueue’, ‘sys\_choi\_dequeue’로 정하였기에 이름도 같게 하였습니다. 나중에 application에서 사용할 수 있도록 system call에 추가하는 것이 목적입니다.

**2. syscalls.h**

🡺 캡쳐화면



🡺 기능  
 시스템 콜 함수들의 prototype(원형)을 정의해서 Linux에서 사용 할 수 있도록 함

🡺 수정한 부분  
my\_queue\_syscall.c 파일에서 만든 함수의 원형과 이름이 같아야 하고 저는 return 값은 long형으로 하고 넣어주는 n값은 ppt와 같도록 int형으로 만들어주었습니다. Asmlinkage를 사용한 이유는 위에서 설명해주었습니다. c파일에서 쓰인 함수를 정확하게 나타내 주는 것 입니다.

**3. my\_queue\_syscall.c**

🡺 캡쳐화면

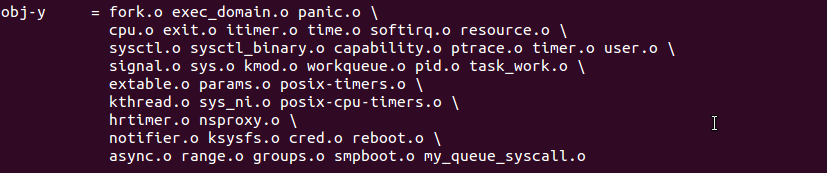


🡺 기능  
syscall에 저장하여 kernel에서 사용할 enqueue, dequeue의 실제 c파일의 기능을 만들어 주는 파일입니다.

🡺 수정한 부분  
형식이 자율이라고 하셨기에 저는 queue를 배열로 만들었습니다. for함수를 통해 중복된 queue가 있는지 확인하여 중복되지 않은 수만 enqueue 하였고 dequeue 부분은 배열의 head부분을 하나 증가시키는 것으로 만들었습니다. 이렇게 만들어진 C파일은 make를 통해서 kernel space에 넣어서 system call에 추가해줄 것 입니다.

**4. Makefile**

🡺 캡쳐화면



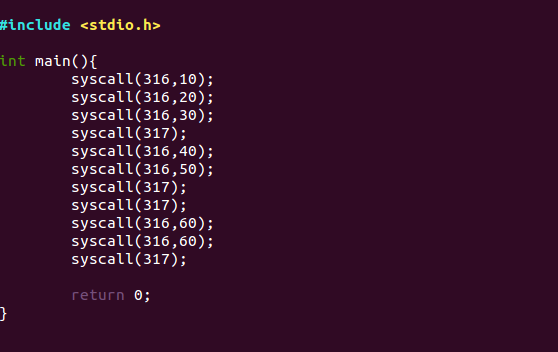
🡺 기능  
c파일을 오브젝트파일로 컴파일 해주는 역할을 합니다.

🡺 수정한 부분

My\_queue\_syscall.c의 오브젝트형인 my\_queue\_syscall.o를 추가해 주었습니다. 이렇게 object형으로 변환하여 우리가 사용할 수 있도록 해 줍니다.

**5. application**

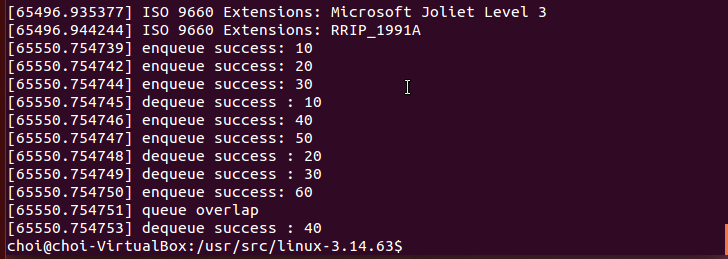
🡺 캡쳐화면



🡺 기능  
추가한 systemcall을 확인하기 위해 실행시키는 application역할을 합니다.enqueue와 dequeue를 번갈아 가면서 실행해 보았고 60이라는 값도 중복해서 enqueue시켜 중복된 값은 저장되지 않는 것을 확인하였습니다. Application은 kernel space에 있는 systemcall을 수행시켜서 user space에서 우리가 볼 수 있도록 해주는 프로그램입니다.

**D. 결과**

🡺 캡쳐화면



**E. 힘들었던 부분**

리눅스를 제대로 다뤄본 적이 없었기 때문에 처음 시작하는 부분이 가장 힘들었던 것 같습니다. 어떻게 시작해야하는지 이게 무엇을 뜻하는지도 잘 모르는 상태에서 PPT만 따라가다보니 오류가 생겼을 때는 물론이고 make를 해줘야 하는 부분같이 PPT에 적혀있지 않는 부분을 스스로 해야 할 때 대처하기가 어려웠습니다. 또 간단한 명령어도 검색해보면서 과제를 해야해서 꽤 시간이 오래 걸렸고 kernel부분을 잘못 건드려 중간에 Ununtu가 뻑가서 다시 해야할 때 힘들었습니다. 또 LAB Session에서도 질문했는데 0차과제 였지만 공유파일 설정에 문제가 생겨서 이것을 해결하는 부분에서 시간을 꽤나 많이 썼던 것 같습니다. (결국 해냈습니다!!!!)

참조 Reference

<http://coffeenix.net/doc/develop/syscall.txt>

<http://blog.secmem.org/605>