**운영체제 1차 과제**

Operating System 1st Assignment Report



• 학과 : 컴퓨터학과

• 학번 : 2015410056

• 이름 : 김지윤

• 제출 일자 : 2017년 4월 4일

• 담당 교수 : 유혁 교수님

**목 차**

1. **개발환경**
2. **시스템 콜과 인터럽트 핸들링**
3. CPU의 두가지 실행모드
4. 인터럽트와 시스템 콜
5. **수정 및 작성한 부분과 그 역할**
6. my\_queue\_syscall.c
7. Makefile
8. syscall\_64.tbl
9. syscalls.h
10. call\_my\_queue.c
11. System Call 동작의

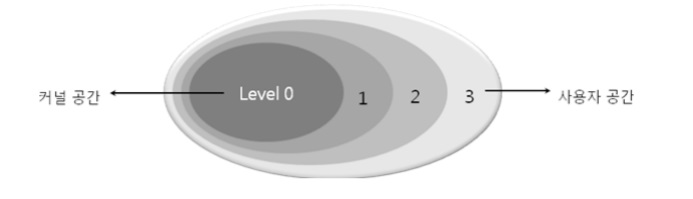
전체적인 이해

1. **실행결과**
2. **수행과정 중 문제점**
3. **개발환경**

* Ubuntu 16.04 LTS
* Linux Kernel Version : 4.4.0

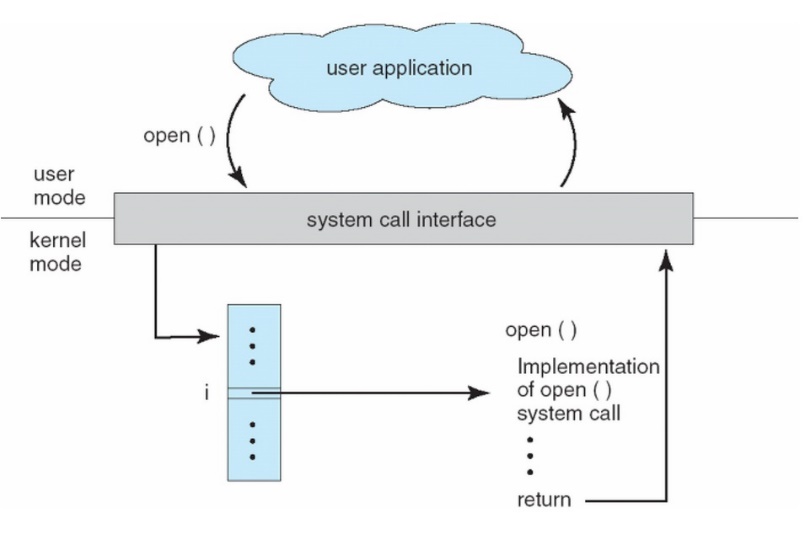
1. **시스템 콜과 인터럽트 핸들링**
2. **CPU의 두가지 실행모드**

운영체제의 효율적인 역할 수행과 시스템 보호를 위해 컴퓨팅 시스템 공간은 User Mode와 Kernel Mode로 나누어지며, 각 Mode별로 권한(Privilege)이 부여된다. 인텔의 경우 4개의 모드, Privilege level을 제공하고 있는데 레벨 0이 커널 모드, 레벨 3이 유저 모드로 지정되어 있으며 나머지 두 레벨은 사용하지 않고 있다.



<그림 1> 커널모드와 유저모드>

운영체제가 실행되는 공간은 모든 권한을 가지고 있는 커널 모드이고, 어플리케이션이 실행되는 공간은 상대적으로 낮은 권한의 유저 모드이다. 유저모드에선 메모리 접근, 명령어에 대해 실행이 일부 제한된다.

메모리의 파일을 읽어오고 저장하는 등의 동작은 커널 모드에서 수행 가능한데, 어플리케이션에서 이러한 동작을 필요로 한다면 어플리케이션은 커널에 호출을 보내고 커널이 사용자 대신 수행하는 방식으로 처리가 가능하다. 이러한 처리 과정을 **시스템 콜(System Call)**이라고 한다.

<그림 2> 시스템 콜

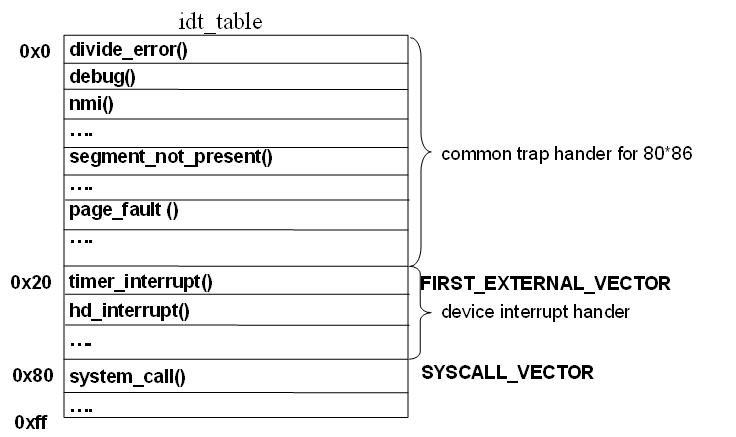
1. **인터럽트와 시스템콜**

**[인터럽트]**

운영체제는 여러 하드웨어를 효율적으로 사용하기위해 각 하드웨어에 대해 작업을 요청하고 다른 일을 수행하다가 요청한 작업이 끝나면 인터럽트를 받는 방식으로 실행된다. **인터럽트(Interrupt)**란 CPU가 프로그램 실행 중 I/O 장치에 사건이 일어나거나 예외 상황이 발생하는 등의 처리가 필요한 경우 자신의 상태를 기억해두고 우선 처리할 수 있도록 하는 것을 말한다. 이는 Source (인터럽트를 요구하는 대상), Vector(인터럽트 발생 시 점프 되는 곳), 그리고, Priority(두가지 이상의 인터럽트 발생 시 허용 우선순위) 의 세가지 요소를 지닌다.

**인터럽트의 처리(Interrupt Handling)** 방식은 다음과 같다.

CPU에 인터럽트가 접수되면 실행중인 코드를 중단하고 인터럽트 소스가 어디인지 어디에 인터럽트가 걸리는지 알기 위한 절차가 진행된다. 이때 소스를 구별할 수단으로 CPU코어에 벡터라는 숫자가 전달되는데, 인터럽트 벡터로 인터럽트 서술자 테이블인IDT에서 적절한 인터럽트 처리를 위한 루틴 ISR (Interrupt Service Routine) 주소 값을 찾아 해당 주소로 점프한다. 리눅스에선 ISR에 대해 일련의 포인터를 사용한다. 이 과정에서 PC값은 자동으로 대피 저장된다. 점프한 후 ISR 코드를 실행하는데, 현재 진행중인 프로그램은 중단된 상태로 저장이 되어야 하므로 ISR 코드는 레지스터를 스택에 대피시키는 부분이 구성 되어있어야 한다. ISR코드에 따라 진행중인 프로그림의 레지스터를 대피하고 인터럽트 처리를 수행한다. 실행이 끝나면 대피시킨 레지스터를 복원하고 인터럽트를 해제시킨 후 대피시킨 PC값을 복원하여 이전 실행 위치로 되돌아간다. 인터럽트의 Priority는 비트들을 켜거나 꺼서 인터럽트를 가능, 불가능하게 만들 수 있는 마스크 레지스터에 의해 제어된다.



<그림 3> IDT

**[시스템 콜]**

**시스템 콜(System Call)**도 일종의 인터럽트이다. 이는 어플리케이션과 커널 간의 인터페이스로 Privilege level을 커널 모드가 지닌 level로 변경시키는 것이다. 이것이 Trap Instruction의 발생을 통해 이루어진다. 리눅스 커널에서 시스템콜은 IDT에 System Call로 예약되어 있는 0x80 인터럽트를 통해 수행된다. 트랩 핸들러가 해당 시스템콜의 번호를 확인해 저장 되어있는 함수를 호출하고 유저 프로세스에 결과값을 return한다.

이번 과제에서 인터럽트의 일종인 시스템콜을 새로 추가하고 이를 Application에서 호출하는 과정을 수행해보았다.

1. **수정 및 작성한 부분과 그 역할**
2. **my\_queue\_syscall.c (kernel 내)**

이번 과제 목표는 내부적으로 FIFO Queue를 갖고 enqueue, dequeue 기능을 하는 시스템 콜을 추가해보는 것이었다. 가장 먼저 해야 할 일은 시스템 콜 함수를 정의하는 것이다.

* 헤더파일 : <kernel.h>

printk() 함수와 같이 커널 수준에서 실행되는 함수들이 매크로로 정의되어 있다.

* 전역변수, 호출함수

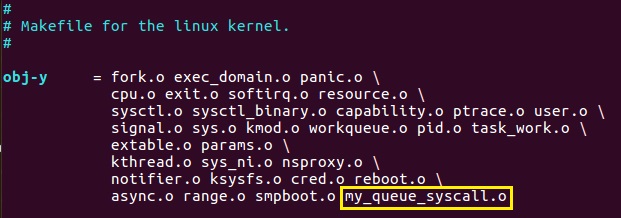
Queue 배열과 큐의 head, tail등을 전역변수로 선언했고 각각 enqueue, dequeue 기능을 하는 sys\_my\_enqueue 와 sys\_my\_dequeue의 호출함수를 갖는다. sys 접두사는 시스템 콜 함수임을 나타내기 위해 관습적으로 붙이는 것이라고 한다. 함수 정의 앞에 붙은 asmlinkage 키워드는 C로 짜인 코드가 Assembly로 구현된 함수로부터 호출될 경우 사용한다.

* 소스코드

크기가 정해진 Int 배열로 최대 (배열 크기 – 1) 만큼 저장 가능한 Circular Queue를 구현했다. 초기 상태는 head와 tail값 모두 0이며 head가 한 칸 앞으로 이동한 후 그 자리에 값을 저장하며 enqueue하고 tail이 한 칸 앞으로 이동한 후 그 자리의 값을 return 하며 dequeue한다. Head와 tail이 배열의 맨 끝에 위치하고 있을 경우 그 다음 index는 0으로 한다. Head 와 tail이 같은 값일 때 Queue는 빈 상태이고 head 다음 칸이 tail 일 경우 Queue는 꽉 찬 상태이며 tail이 head를 앞서는 경우는 존재하지 않는다.

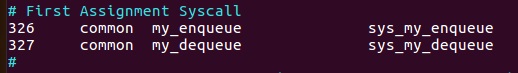
코드 내에 n\_head(next index of head), n\_tail을 계산하는 부분이나 Queue를 출력하는 부분 등 반복되어 사용되는 부분이 많았다. 이 부분을 함수로 정의했으면 코드가 훨씬 간단해질 수 있을 텐데 시스템 콜 번호 등록이나 여러가지 걸리는 것이 많아 그냥 일일이 작성하였다.

1. **Makefile (kernel 내)**



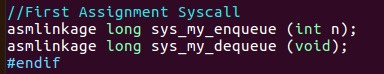
새로 작성한 시스템 콜을 컴파일 하도록 같은 디렉토리 내 Makefile에 같은 파일명으로 오브젝트 파일을 추가한다.

1. **syscall\_64.tbl (linux 내)**



새로운 시스템 콜 함수에 대한 고유번호를 추가한다. 시스템 콜을 요청하는 코드가 포함된 유저 프로그램은 <unistd.h>라는 헤더파일을 포함해야 한다. 이 파일 내부를 살펴보면 시스템 호출 번호 테이블에 고유 번호를 추가함에 따라 구현하고자 하는 시스템 호출을 위해 인터럽트 주소를 부여해주는 내용이 담겨있다.

1. **syscalls.h (kernel 내)**

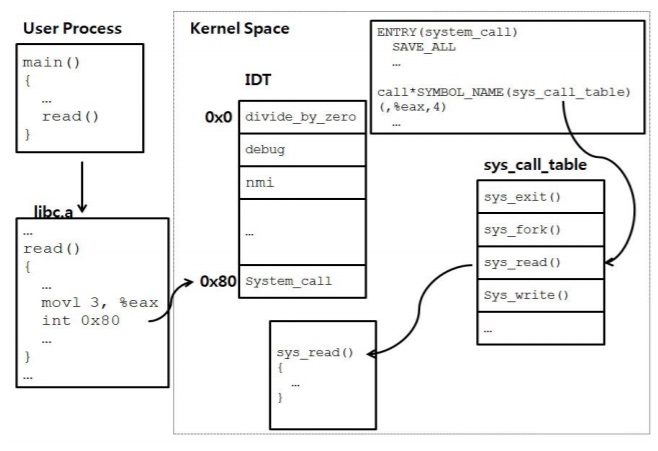


시스템 콜 함수를 등록한다. 이는 고유 호출 번호와 연결시켜주는 부분인데 등록된 콜 함수들은 syscall\_64.tbl 의 테이블과 1대1 대응한다. 테이블 마지막 부분에 +1씩 해서 등록했으니 이 파일에서도 마지막 부분에 추가 등록한다.

1. **call\_my\_queue.c**

새로 추가한 시스템 콜을 호출하는 User Application이다. 시스템 콜 함수 정보가 있는 헤더파일<unistd.h>을 포함하고 있다. 앞서 syscall\_64.tbl 부분에서 언급한 unistd.h 파일과 동일한 헤더파일은 아니다. 그러나 Makefile 내에는 다른 특정 디렉토리에 있는 헤더 파일을 이용할 수 있도록 하는 컴파일 옵션을 포함하고 있고, 헤더 파일들은 서로 참조하게 되어있다.

1. **System Call 동작의 전체적인 이해**



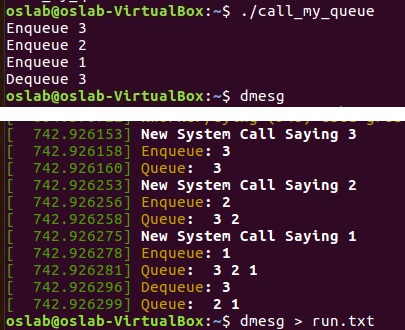
<그림 4> System Call 동작 과정

<그림 4>를 참고해서 설명하자면 시스템 콜의 동작과정은 다음과 같다.

User Application에서 시스템 콜이 호출되면 Wrapper 함수를 통해 IDT 상 System Call 이 예약되어 있는 0x80에 걸리는 Trap Instruction이 추가된다. 이것이 System Call을 발생시키고 User Application 프로그램 내에 시스템 콜 인자로 서술된 시스템 콜 번호를 확인해 syscall\_64.tbl에 저장되어 있는 해당 시스템 콜을 호출한다. 이 연결 (호출 번호와 시스템 콜 함수)은 syscall.h에 명시되어 있고 이에 따라 커널 내 Makefile을 통해 my\_queue\_syscall.c 파일이 컴파일 된 오브젝트 파일 my\_queue\_syscall.o 가 실행되며 해당 시스템 콜을 실행하게 되는 것이다.

my\_dequeue의 경우 int 값을 return하며 어플리케이션 프로그램 상 변수 r이 이를 받아 printf()로 출력하게 된다.

1. **실행결과**



1. **수행과정 중 문제점**

시스템 콜 함수를 작성하고 테스트 Application을 실행시켜보니 출력 형식이 과제에서 요구하는 것과 맞지 않았다. 그래서 수정 후 다시 컴파일, 실행시켰는데 이전 시스템 콜 함수 형식이 계속해서 나왔다 그땐 내가 빠트린 부분이 없다고 생각해서 구글링 하다가 답이 안 나오길래 아예 개발환경을 모두 지우고 다시 깔았다.

지금 생각해보니 컴파일 후 커널 설치 부분(make modules\_install install)을 빠트린 채 재부팅을 해서 그런 것 같다.

\*그림 출처

<http://luckyyowu.tistory.com/133>

<https://wiki.kldp.org/KoreanDoc/html/EmbeddedKernel-KLDP/idt.html>

<ftp://os2.sch.ac.kr/Embedded/reference/embed-ch4-5.pdf>

OS\_1차과제\_문서.pdf