# 운영체제 1차 과제

**학과: 노어노문학과**

**학번: 2018131321**

**이름: 김현우**

**제출일: 2022.10.27**

**Freeday 사용일수: 없음**

목차



**개발환경:**
**Window11, VirtualBox, Linux 18.04.02 LTS, Kernel Version 4.20.11**

# Ⅰ. 리눅스의 시스템 콜에 대한 설명

## 1. System call 정의

시스템 콜은 User mode에서 Kernel mode로 진입하기 위한 통로를 말한다. 이는 커널에서 제공하는 protected 서비스를 이용하기 위하여 필요하다.

운영체제는 크게 본다면 User mode와 Kernel mode로 나뉘어 있다. Kernel mode는 모든 권한을 가진 실행 모드이며, 운영체제가 실행된다. Kernel mode는 사용자가 접근하지 못하는 서비스에 접근이 가능하며 레지스터에도 접근이 가능하다. User mode는 Kernel mode에 비해 낮은 권한을 가진다. 이 모드에서는 user application이 실행된다. User mode에서는 하드웨어에 직접적인 접근은 불가능하다. 이러한 User mode를 일시적으로 Kernel mode로 전환해줘 커널 영역의 기능을 사용자 모드가 접근하게 도움을 주는 것이 시스템 콜(System Call)인 것이다. 즉 시스템 콜은 프로세스가 하드웨어에 직접 접근하게 도와줘 필요한 기능을 사용할 수 있게 한다.

## 2. System Call 쓰는 이유

시스템 콜을 사용하는 이유는 User mode에서 컴퓨터를 사용하고 있는 User가 application으로 운영체제의 데이터에 접근하는 것을 막기 위함이다. User가 운영체제에 제약 없이 접근하게 된다면 운영체제의 데이터 수정 혹은 삭제가 가능할 것이다. 이는 치명적인 오류를 발생시킬 수도 있다. 그러므로 User mode에서 해당 application은 시스템 콜을 통해 커널 모드로 잠시 전환되었다가 다시 User mode로 돌아오게 된다.

## 3. System Call의 종류와 처리방식

- 프로세서 제어
- 파일조작
- 장치관리
- 시스템 정보 및 자원 관리
- 통신 관련

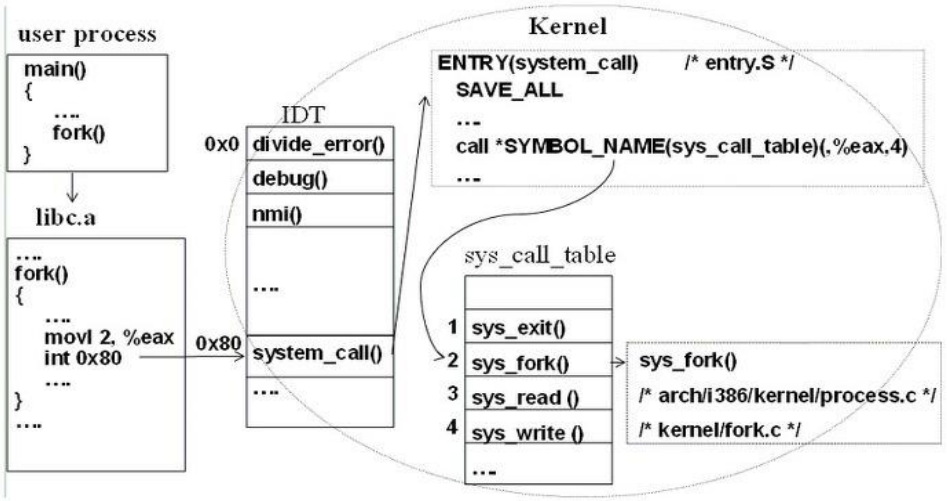
커널은 내부적으로 시스템 콜을 구분하기 위해서 고유의 번호를 할당해준다. 커널은 호출된 시스템 콜에 해당되는 번호를 확인한 후, 번호에 맞는 루틴을 호출하게 된다. 모든 작업을 완료한 후 커널 모드에서 다시 유저 모드로 돌아오게 된다.

### 4. System Call 호출 방식

User mode에서 fork() 시스템 콜을 호출한다면 C 라이브러리(libc.a)에서 시스템 콜 고유 번호인 2번을 eax 레지스터에 저장하고 Wrapper Function을 통해 Trap instruction을 발생시킨다. 0x80 Trap은 IDT에서 System call()로 예약이 되어있다.

리눅스는 System call을 처리하기 위해 IDT를 사용한다. 각 서비스 루틴들은 함수로 구현되어있고 각 함수의 시작점 주소는 IDT에 등록되어 있다. IDT는 메모리에 위치한다. 만약 다음 그림과 같이 0x80 Trap이 발생한다면 IDTR(IDT의 위치를 가리키는 레지스터)을 통해 IDT를 알아낸다. 후에 Trap 발생 번호에 대응되는 핸들러를 찾아 정의된 내용을 처리한다.

Trap 핸들러는 eax(=2)값을 index로 sys_call_table을 탐색한다. 현재 eax 레지스터에 들어있는 2값으로 sys_fork의 포인터 값을 받아 sys_fork() 함수를 호출한다. 다음과 같은 서비스가 종료되면 다시 User Process로 이동한다.




## Ⅱ. 수정 및 작성한 부분과 설명

### 1. syscall_64.tbl

시스템 콜을 추가할 때 처음으로 syscall_64.tbl에 시스템 콜의 고유 번호를 추가했다. 이 테이블에는 리눅스에서 제공하는 모든 시스템 콜의 고유 번호가 저장되어 있다. application에서 시스템 콜을 호출하면 eax 레지스터에 저장된 시스템 콜 번호를 보고 해당 함수가 호출된다. 따라서 시스템 콜 테이블에 새로운 시스템 콜을 추가할 때 등록이 필요하다. 따라서 335번과 336번으로 enqueue와 dequeue 함수를 등록했다. 시스템 콜 번호는 마지막 할당 번호의 다음 번호가 되어야 하므로 마지막 334번 이후의 번호인 335, 336번으로 등록했다.

### 2. syscalls.h

이후 시스템 콜 함수들의 prototype을 syscalls.h에 정의했다. 이때 asmlinkage를 사용했는데 인터럽트 핸들러는 assembly 코드로 작성되므로 asmlinkage를 함수 앞에 선언하여 assembly code에서도 C 함수를 호출할 수 있게 하였다. 함수의 prototype은 다음과 같이 정의하였다.

- void sys_oslab_enqueue(int);
- int sys_oslab_dequeue(void);

enqueue 함수는 새로운 int type의 element를 queue에 추가하므로 파라미터에 int를 넣었고

return 값은 존재하지 않으므로 void를 썼다. Dequeue 함수는 새로운 element를 추가하지는 않기 때문에 파라미터에 아무것도 넣지 않는다. 따라서 파라미터값은 void로 설정하였다. 대신 dequeue는 queue에서 제일 먼저 들어간 element 하나를 queue 자료구조에서 추출해주기 때문에 int를 return 값으로 설정하였다.

### 3. My_queue_syscall.c

*#define MAXSIZE 500*

Queue의 최대 사이즈를 500으로 가지는 선형 큐를 목적으로 작성하였다.

SYSCALL_DEFINEx에서 x는 파라미터의 개수를 의미한다. oslab_enqueue는 x가 1이므로 파라미터의 개수가 1개이고 oslab_dequeue의 파라미터의 개수는 0개이다.

```
SYSCALL_DEFINE1(oslab_enqueue, int, a )
        if (rear > MAXSIZE - 1)
                return -2;
```

rear는 queue에 있는 마지막 element 다음 위치를 가리키는 index이다. rear가 가리키는 index 위치의 queue 배열에는 아직 element가 들어가 있지 않다. 이 함수의 마지막에 rear = rear + 1; 라는 명령어가 있기 때문이다. 따라서 rear가 가리키는 위치가 배열의 최대 인덱스(MAXSIZE – 1)보다 커진다면 queue가 full인 상태라 시스템 콜을 종료한다.

```
        for ( i = front; i<rear ;i++)
                if (queue[i] == a)
                        return a;
```

변수 i는 queue의 제일 처음에 있는 값을 가리키는 front이다. i는 1씩 더해져 새로운 element 값이 들어갈 위치를 가리키는 rear 전까지 커진다. 이때 i는 queue의 index 위치를 가리키는데 queue[i]의 값이 새롭게 추가할 element a와 같은 일이 발생한다면 함수를 그대로 종료하고 queue에 삽입하려는 a 값을 반환한다.

```
        queue[rear] = a ;
```

후에 rear index위치에 파라미터 a의 값을 대입한다.

```
        for (i = front; i <= rear ; i ++)
                printk("%d₩n", queue[i]);
        rear = rear + 1;
```

queue에 새로운 element를 대입한 후 처음 값을 가리키는 index인 front부터 마지막 대입 값의 위치를 가리키는 rear까지 i를 1씩 더해주면서 queue[i]의 값을 출력해준다. 출력 값은 queue에 들어가 있는 모든 element이다. 또한 rear의 값은 1 더해준다. 이제 rear는 queue에 새롭게 추가할 index의 위치를 가리킨다. 마지막으로 삽입하려고 했던 element 값인 a를 return 해주고 종료한다.

```
SYSCALL_DEFINE0(oslab_dequeue)
        if (rear == front)
                return -2;
```

만약 rear의 값이 front랑 같다면 queue는 empty인 것으로 간주하고 함수를 종료 시킨다.

*res = queue[front];*

*front = front + 1;*

front는 queue에 있는 element 중 가장 처음에 있는 값을 가리키는 index이다. 따라서 queue[front]로 제일 처음 값을 추출해 res에 대입해주고, 다음 처음 값을 가리키게 하려고 front에 1을 추가해 주었다. 후에 enqueue와 같은 방식으로 queue의 element 값들을 출력해 주고 res 값을 return 해준다.

마지막으로 **Makefile**에서 작성한 시스템 콜 함수의 오브젝트 생성을 위해 컴파일 추가를 해주었다.

### 4. User application

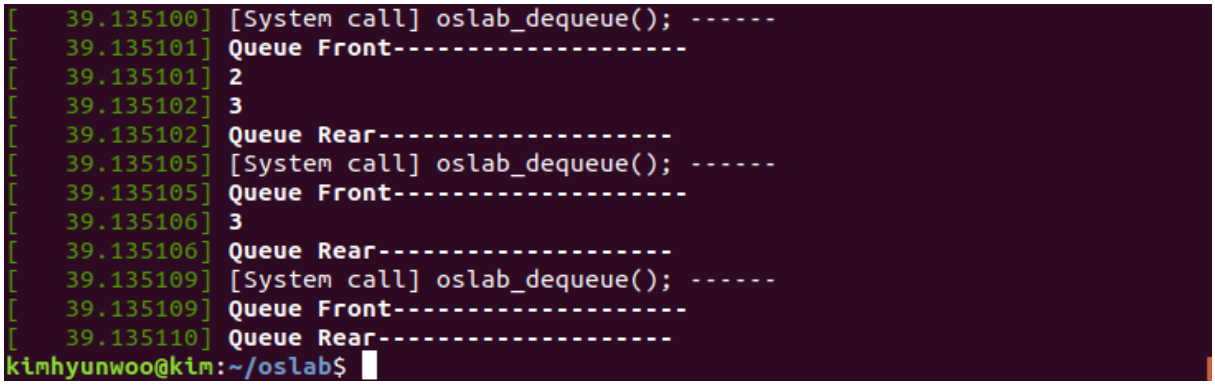
시스템 콜 추가와 수정을 마친 후 User mode에서 system call을 호출하는 user application을 작성하였다. 이 application에서는 syscall()이라는 매크로 함수를 이용하여 시스템 콜을 호출한다.

파일의 처음에 335번과 336번에 있는 oslab_enqueue와 oslab_dequeue 시스템 콜을 my_queue_enqueue와 my_queue_dequeue로 다시 명명하였다. 이러한 define을 통해 원래 syscall(335, 1)과 같은 형식으로 작성했을 코드를 syscall(my_queue_enqueue, 1)과 같이 사용할 수 있게 되었다. 처음에는 syscall()함수를 사용해서 queue에 1, 2, 3, 3 순서대로 enqueue 하였다. syscall()함수의 return 값은 queue에 삽입한 element 값이다. Enqueue를 진행한 후 dequeue를 진행하였다. Dequeue는 파라미터 값이 필요 없으므로 enqueue와는 다르게 따로 파라미터를 주지 않는다. Dequeue는 총 3번 진행하였고 syscall(my_queue_dequeue)의 return 값은 dequeue함수의 return 값인 res이다.

## Ⅲ. 실행 결과 스냅샷

```
kimhyunwoo@kim:~/oslab$ ./call_my_queue
Enqueue : 1
Enqueue : 2
Enqueue : 3
Enqueue : 3
Dequeue : 1
Dequeue : 2
Dequeue : 3
```

```
[   39.134942] [System call] oslab_enqueue(); ------
[   39.134944] Queue Front--------------------
[   39.134945] 1
[   39.134945] Queue Rear--------------------
[   39.135085] [System call] oslab_enqueue(); ------
[   39.135086] Queue Front--------------------
[   39.135087] 1
[   39.135087] 2
[   39.135088] Queue Rear--------------------
[   39.135092] [System call] oslab_enqueue(); ------
[   39.135092] Queue Front--------------------
[   39.135093] 1
[   39.135093] 2
[   39.135094] 3
[   39.135095] Queue Rear--------------------
[   39.135097] [Error] - Already existing value
```




## IV. 숙제 수행 과정 중 발생한 문제점과 해결방법

수행 과정 중 발생한 문제점은 몇 가지가 존재했다. 첫 번째는 시스템 콜 소스를 작성할 때 모든 변수들의 선언을 첫 부분에 해야 한다는 사실을 몰라서 발생했다. front와 rear, i를 제외하고 j라는 변수를 추가로 선언하여 실행해보았지만 sudo make부분에서 오류가 발생하였다. 후에 모든 변수들을 첫 부분에 선언해야 한다는 사실을 알게 되었고 j 변수를 소스 파일의 첫 부분에 추가했다. 두 번째 문제는 SYSCALL_DEFINE 함수 외에 새로운 oslab_enqueue라는 함수를 따로 소스 내에서 만들어야 한다고 생각한 것에 기인한다. SYSCALL_DEFINE 함수가 oslab_enqueue라는 시스템 콜을 이용하기 위해 내부적으로 수정을 해야 한다는 사실을 몰랐기에 새로운 함수를 만들었지만 오류가 발생했다. 하지만 계속된 오류가 발생하였고 여러 정보를 수집한 후 SYSCALL_DEFINE 내부에 코드를 작성해야 함을 깨닫게 되었다.