운영체제 1차 과제 보고서

학과 : 컴퓨터학과

학번 : 2018320225

이름 : 이지수

제출 날짜 : 2022.10.27

Freeday 사용 일수 : 0일

목 차

- 개발 환경

AMD Ryzen 3 PRO 4350G, Windows 10

Oracle VM VirtualBox, Ubuntu (64-bit) 18.04.2, Linux Kernel (ver 4.20.11)

- 리눅스의 시스템 콜(호출 루틴 포함)에 대한 설명

시스템 콜을 이해하기 위해서는 CPU에서의 실행 모드에 대한 이해가 선행되어야 한다. CPU는 시스템의 보호를 위해 2가지 이상의 실행 모드(Execution mode)를 가지고 보통 Kernel mode와 User mode로 구분된다. Kernel mode는 모든 권한을 가진 실행 모드로 OS가 실행되는 모드이다. Kernel mode에서는 모든 시스템 메모리에 접근이 가능하며, 모든 CPU명령어(Privilege 명령어) 실행이 가능하다. 반면 User mode는 Kernel mode에 비해 낮은 권한을 가지며 사용자의 application이 실행되는 모드이다. User mode에서는 Privilege 명령어 실행이 불가능 하기 때문에 User의 process에서 하드웨어에 직접 접근하는 것을 방지하고 이를 통해 시스템을 보호한다. 그렇다면 User의 process가 Kernel에서 제공하는 보호된 서비스를 이용하기 위한 방법이 필요한데 여기서 system call이 사용된다. 따라서 system call은 User mode에서 Kernel mode로 진입하기 위한 통로라고 할 수 있다.

User의 process가 시스템 콜을 요청하면 User mode에서 Kernel mode로 제어권이 넘어온다. 커널에서는 다양한 종류의 시스템 콜을 구분하기 위해 기능별로 고유 번호를 할당하여 table 형태(IDT)로 저장해 놓는다. Kernel은 시스템 콜을 요청받으면 해당하는 기능 번호를 확인하고 그에 맞는 서비스 루틴을 호출하고 모든 처리가 끝나면 반환과 함께 사용자 모드로 다시 전환된다.

- 수정 및 작성한 부분과 설명 (이유)

a) syscall_64.tbl

```
#oslab
335     common  oslab_enqueue           __x64_sys_oslab_enqueue
336     common  oslab_dequeue           __x64_sys_oslab_dequeue
```

위에서 설명한 것처럼 시스템 콜은 기능별로 고유 번호가 할당되어 table 형태로 저장이 되어 있는데 이 중 335번과 336번에 enqueue와 dequeue를 위한 시스템 콜 함수를 지정하였다.

b) syscalls.h

```
/*oslab*/
asmlinkage int sys_oslab_enqueue(int);
asmlinkage int sys_oslab_dequeue(void);
```

시스템 콜 함수들을 c의 형식으로 선언하고 asmlinkage를 사용하였다. 시스템 콜 호출은 int80 인터럽트 핸들러에서 호출하는데 이것이 assembly 코드로 작성되었기 때문에 asmlinkage를 사용하여 assembly 코드에서도 c형식의 함수 호출이 가능하게 하기 위함이다.

c) my_queue_syscall.c

enqueue와 deqeue 시스템 콜 함수에 대한 구체적인 구현 내용이 들어가 있는 파일이다.

```
#include<linux/syscalls.h>
#include<linux/kernel.h>
#include<linux/linkage.h>

#define MAXSIZE 500

int queue[MAXSIZE]; // 큐를 구현하는 배열 선언
int front = 0; // 큐의 가장 앞 원소 위치를 의미하는 index
int rear = 0; // 큐의 가장 마지막 원소 한 칸뒤를 의미하는 index
int i, res = 0; // for문 iteration을 위한 변수 i와 dequeue시에 반환 값을 담을 변수 res 선언
```

큐를 배열을 이용하여 구현하되 최대 사이즈를 500으로 정하였고 구체적인 변수와 코드에 대한 설명은 주석으로 표기하였다.

```
SYSCALL_DEFINE1(oslab_enqueue, int , a){ // enqueue에 대한 system call 정의하는 함수
        if(rear >= MAXSIZE - 1){
                printk(KERN_INFO "[Error] - QUEUE IS FULL---------------\n");
                return -2; // queue에 남은 공간이 있는지 확인하고 없다면 error 메시지 출력과 -2를 반환
        } else { // 남은 공간이 있다면
                for(i = front; i < rear; i++) { // 큐의 앞부터 마지막까지 scan하면서
                        if(queue[i] == a) { // 입력값으로 들어온 a가 이미 큐에 존재한다면
                                printk(KERN_INFO "[Error] - %d is already existing value\n", a);
                                return a; // error 메시지 출력과 이미 존재하는 값 a 반환
                        }
                }
                queue[rear++] = a; // for문을 통과했다는 것은 입력값이 queue에 존재하지 않으므로 정상적인 enqueue가 가
능하므로 rear의 index에 a를 넣고 값을 1 증가시킴
        }
        printk(KERN_INFO "[System call] oslab_enqueue(); -----\n");
        printk("Queue Front---------------------\n");
        for (i = front; i < rear; i++) {
                printk("%d\n", queue[i]);
        } // enqueue 이후 현재 큐의 상태 for문으로 출력
        printk("Queue Rear---------------------\n");

        return a; // enqueue 된 값 반환
}
```

다음 코드는 enqueue에 대한 시스템 콜 함수의 구현으로 enqueue의 경우 queue에 넣을 파라미터를 1개 받아야 하므로 SYSCALL_DEFINE1을 통해 정의하였다.

```
SYSCALL_DEFINE0(oslab_dequeue){ // dequeue에 대한 system call 정의하는 함수
        if(rear == front){
                printk(KERN_INFO "[Error] - EMPTY QUEUE----------------\n");
                return -2; // rear == front로 queue가 비어있는지 확인하고 비어있다면 error 메시지를 출력하고 -2 반환
        }
        res = queue[front]; // queue가 비어있지 않다면 dequeue 가능하므로 dequeue할 값을 res에 저장
        front++; // front가 기존 front의 다음 index를 가리키게 값을 1 증가시킴
        printk(KERN_INFO "[System call] oslab_dequeue(); -----\n");
        printk("Queue Front----------------------\n");
        for (i = front; i < rear; i++) {
                printk("%d\n", queue[i]);
        } // dequeue 이후 현재 큐의 상태 for문으로 출력
        printk("Queue Rear----------------------\n");

        return res; // dequeue 된 값 반환
}
```

다음 코드는 dequeue에 대한 시스템 콜 함수의 구현으로 dequeue의 경우 파라미터를 받을 필요 없이 queue의 front의 값을 반환하기 때문에 SYSCALL_DEFINE0을 통해 정의하였다.

d) Makefile

```
obj-y     = fork.o exec_domain.o panic.o \
            cpu.o exit.o softirq.o resource.o \
            sysctl.o sysctl_binary.o capability.o ptrace.o user.o \
            signal.o sys.o umh.o workqueue.o pid.o task_work.o \
            extable.o params.o \
            kthread.o sys_ni.o nsproxy.o \
            notifier.o ksysfs.o cred.o reboot.o \
            async.o range.o smpboot.o ucount.o my_queue_syscall.o
```

Kernel make 과정에서 my_queue_syscall.c에서 작성한 내역들이 오브젝트 파일로 포함될 수 있게 obj-y 부분에 추가하였다.

e) call_my_queue.c

위에서 정의한 시스템 콜을 사용하는 user applicaion이라 할 수 있다.

```
#include<unistd.h>
#include<stdio.h>

#define my_queue_enqueue 335
#define my_queue_dequeue 336

int main(){
        int a = 0; // 시스템 콜 호출 반환 값을 받을 변수 a 선언
        int i; // iteration을 위한 변수 i 선언

        for (i = 1; i <= 3; i++) {
                a = syscall(my_queue_enqueue, i);
                printf("Enqueue : ");
                printf("%d\n", a);
        } // enqueue를 i에 1,2,3을 넣어서 한 번씩 호출

        a = syscall(my_queue_enqueue, 3);
        printf("Enqueue : ");
        printf("%d\n", a);
        // 이미 queue에 존재하는 값인 3을 한번 더 넣어서 중복 처리에 대한 기능 체크

        for (i = 1; i <= 3; i++) {
                a = syscall(my_queue_dequeue);
                printf("Dequeue : ");
                printf("%d\n", a);
        }
        // dequeue 3번 호출
        return 0;
}
```

Syscall() 이라는 함수를 이용하여 syscall_64.tbl에서 할당한 시스템 콜 함수 번호를 인자로 시스템

콜을 호출한다. 코드에 대한 설명은 주석으로 표기하였다.

- 실행 결과 스냅샷

```
jslee@jslee-VirtualBox:~/test$ gcc call_my_queue.c -o call_my_queue
jslee@jslee-VirtualBox:~/test$ ./call_my_queue
Enqueue : 1
Enqueue : 2
Enqueue : 3
Enqueue : 3
Dequeue : 1
Dequeue : 2
Dequeue : 3
```

```
jslee@jslee-VirtualBox:~/test$ dmesg
[  489.675178] [System call] oslab_enqueue(); -----
[  489.675196] Queue Front---------------------
[  489.675197] 1
[  489.675198] Queue Rear----------------------
[  489.675258] [System call] oslab_enqueue(); -----
[  489.675259] Queue Front---------------------
[  489.675259] 1
[  489.675259] 2
[  489.675259] Queue Rear----------------------
[  489.675261] [System call] oslab_enqueue(); -----
[  489.675261] Queue Front---------------------
[  489.675261] 1
[  489.675262] 2
[  489.675262] 3
[  489.675262] Queue Rear----------------------
[  489.675263] [Error] - 3 is already existing value
[  489.675264] [System call] oslab_dequeue(); -----
[  489.675265] Queue Front---------------------
[  489.675265] 2
[  489.675265] 3
[  489.675265] Queue Rear----------------------
[  489.675266] [System call] oslab_dequeue(); -----
[  489.675266] Queue Front---------------------
[  489.675267] 3
[  489.675267] Queue Rear----------------------
[  489.675268] [System call] oslab_dequeue(); -----
[  489.675269] Queue Front---------------------
[  489.675269] Queue Rear----------------------
```

- 숙제 수행 과정 중 발생한 문제점과 해결방법

코딩을 완료하고 정상적으로 실행 결과가 나오는 것을 확인하고 다음 날 재부팅하여 다시 실행 결과를 확인하려 하는데 아래와 같이 모든 return 값이 -1로 출력되는 것을 확인하였다.

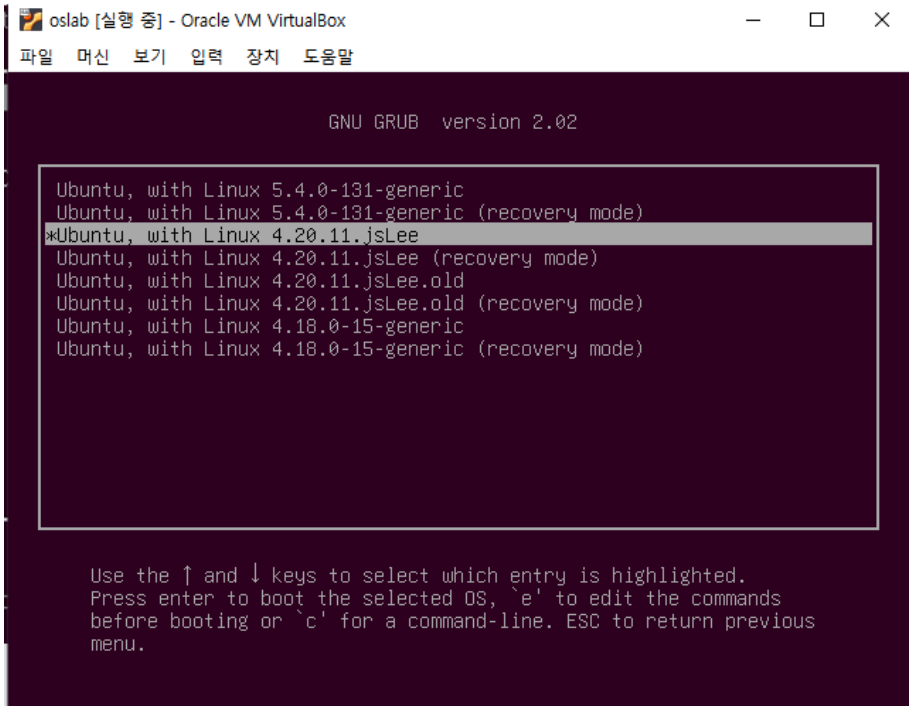
```
jslee@jslee-VirtualBox:~/test$ gcc call_my_queue.c -o call_my_queue
jslee@jslee-VirtualBox:~/test$ ./call_my_queue
Enqueue : -1
Enqueue : -1
Enqueue : -1
Enqueue : -1
Dequeue : -1
Dequeue : -1
Dequeue : -1
```

재부팅을 하고 코딩한 내역들을 확인해보아도 문제점이 발견되지 않아 고민을 하던도중 uname -r로 kernel 버전을 확인해 보았더니

```
jslee@jslee-VirtualBox:/usr/src/linux-4.20.11$ uname -r
5.4.0-131-generic
```

다음과 같은 버전으로 자동적으로 업데이트가 되어있는 것이 문제점이었다는 것을 발견했다.

이를 해결하기 위해 부팅 시에 왼쪽 shift를 눌러서 ubuntu grub 모드로 진입하였고



다음과 같이 4.20.11 version으로 재진입하고 자동적으로 업데이트된 커널 버전을 삭제하였더니 정상적으로 결과가 출력되었다.