

2022 년 2 학기 운영체제 1 차 과제

컴퓨터학과 2015150220 이윤수

지도 교수님	유혁 (hxy@os.korea.ac.kr)
조교	고영훈, 이준석(ta@os.korea.ac.kr)
주제	시스템 콜 추가 및 이해
출제일	2022. 10.06. (목)
제출일	2022. 10. 26. (수) - No Free Day Used

Object

이번 과제의 목적은 일반 배포판 리눅스에 새로운 Kernel 을 컴파일하고 수정해서 새로운 System Call 을 추가하는 것이다. 본 과제에서는 Kernel 을 직접 컴파일하고, 이후 기존의 Kernel 에 새로운 System Call 을 추가한 후에 이를 호출할 수 있는 User-Application 까지를 구현하도록 한다.

새로 추가한 system call 은 내부적으로 Queue 자료구조를 갖도록 한다. 그리고 이 Queue 에 integer 값을 Enqueue 하거나 Dequeue 하는 역할을 수행하도록 작성한다. 단, 중복된 integer 값을 저장할 수 없다. 그 밖에 정해진 Queue 의 구현 방법은 없으며 자유롭게 구현할 수 있다. 기본적인 Enqueue 와 Dequeue 가 가능하면 된다. Queue 에 저장할 수 있는 entry 의 개수 또한 고정적으로 하여도 무관하다.

Environment

Host - MacOS Monterey 12.6, x86_64, virtualBox - 6.1.40

Guest - Ubuntu 18.04, Linux(4.20.11), x86_64

Explanation of System Call

System Call

프로그램이 실행되는 동안 하드웨어 자원에 대해 자유로운 접근 및 이용이 가능해야 한다. 이때 주로 문제가 되는 것이 특정 자원에 대해 여러 프로그램이 동시에 접근하는 경우이다. 이때 프로그래머가 그와 같은 문제 상황을 모두 고려해서 프로그램을 작성하는 것은 매우 어려운 일이다. 이러한 어려움을 해결하고자 고안된 것이 운영체제 커널과 시스템콜이다. 잠재적인 충돌이 예상되는 시스템 자원은 응용 프로그램이 아닌 커널이 특권을 가지고 단독으로 접근 및 사용한다. 만약 어떤 프로그램이 그러한 자원을 이용하고 싶다면, 해당 프로그램은 자원에 직접 접근하는 대신 시스템콜을 호출한다. 시스템콜이 호출되면 특별히 지정된 Trap Instruction 이 발생한다. 트랩이 발생하면 그에 대한 대응으로 커널이 기존 응용프로그램을 대신해서 필요한 작업을 수행하고 그 결과를 해당 프로그램에게 제공한다.

Implementation

syscall_64.tbl

Location : arch/x86/entry/syscalls/syscall_64.tbl

ADD:

```
#oslab
335 common  oslab_enqueue      _x64_sys_oslab_enqueue
336 common  oslab_dequeue      _x64_sys_oslab_dequeue
```

Syscall table 에 새로운 system call 을 추가했다. 335 번이 enqueue, 336 번이 dequeue 이다.

syscalls.h

Location : include/linux/syscalls.h

ADD:

```
/*oslab*/

asmlinkage int sys_oslab_enqueue(int);
asmlinkage int sys_oslab_dequeue(void);
```

실제로 호출할 syscall 함수들을 헤더파일에 추가해주었다.
그 구현은 아래의 my_queue_syscall.c 에서 이루어졌다.

my_queue_syscall.c

```

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

#define MAXSIZE 500

int queue[MAXSIZE];
int front = 0;
int rear = 0;
int i, res = 0;

SYSCALL_DEFINE1(oslab_enqueue, int, a){

    if(rear >= MAXSIZE){ // if queue is full
        printk(KERN_INFO "[Error] - QUEUE IS FULL-----\n");
        return -2;
    }
    // check if the given number is already in the queue.
    for(i=front; i<rear; i++){
        if(queue[i] == a){
            printk(KERN_INFO "[Error] - ALREADY EXISTING VALUE\n");
            return a;
        }
    }
    // insert number in the queue and increase value of rear
    queue[rear] = a;
    rear++;
    printk(KERN_INFO "[System call] oslab_enqueue(); -----\n");
    printk("QUEUE FRONT ----- \n");
    // print queue from front to rear-1
    for(i=front; i<rear; i++){
        printk("%d\n", queue[i]);
    }
    printk("QUEUE REAR ----- \n");
    return a;
}

SYSCALL_DEFINE0(oslab_dequeue){
    if(rear == front){ // if queue is empty
        printk(KERN_INFO "[Error] - EMPTY QUEUE-----\n");
        return -2;
    }
    // queue is not empty, return the front value in the queue.
    res = queue[front];
    // remove the front value by overwriting the rest in order from the front.
    for(i=front+1; i<rear; i++){
        queue[i-1] = queue[i];
    }
    // reduce the rear because the queue size has been decreased.
    rear--;

    printk(KERN_INFO "[System call] oslab_dequeue(); -----\n");
    printk("QUEUE FRONT----- \n");
    // print the queue from front to rear.
    for(i=front; i<rear; i++){
        printk("%d\n", queue[i]);
    }
}

```

```

        printk("QUEUE REAR-----\n");
        return res;
}

```

Makefile

Location : kernel/Makefile

From:

(.....)

async.o range.o smpboot.o ucount.o

To:

(.....)

async.o range.o smpboot.o ucount.o my_queue_syscall.o

my_queue_syscall 의 오브젝트 파일을 기존 커널의 Makefile 에 추가했다.

call_my_queue..c

```

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

#define my_queue_enqueue 335
#define my_queue_dequeue 336

int main(){

    int a = 0;

    a = syscall(my_queue_enqueue, 1);
    printf("Enqueue : ");
    printf("%d\n", a);

    a = syscall(my_queue_enqueue, 2);
    printf("Enqueue : ");
    printf("%d\n", a);

    a = syscall(my_queue_enqueue, 3);
    printf("Enqueue : ");
    printf("%d\n", a);

    a = syscall(my_queue_enqueue, 3);
    printf("Enqueue : ");

```

```

    printf("%d\n", a);

    a = syscall(my_queue_dequeue);
    printf("Dequeue : ");
    printf("%d\n", a);

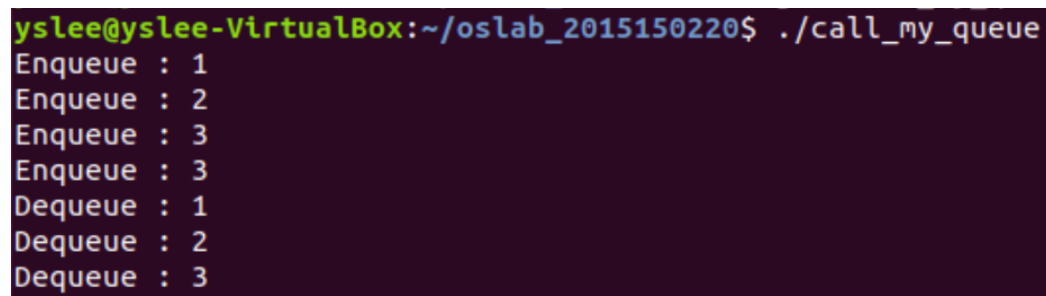
    a = syscall(my_queue_dequeue);
    printf("Dequeue : ");
    printf("%d\n", a);

    a = syscall(my_queue_dequeue);
    printf("Dequeue : ");
    printf("%d\n", a);

    return 0;
}

```

Result



```

yslee@yslee-VirtualBox:~/oslab_2015150220$ ./call_my_queue
Enqueue : 1
Enqueue : 2
Enqueue : 3
Dequeue : 1
Dequeue : 2
Dequeue : 3

```

User application의 실행결과이다. 1 - 2 - 3 -3 의 순서로 enqueue, 1 - 2 - 3의 순서로 dequeue 가 일어났다. 해당 숫자들이 정상적으로 출력된 것을 통해 system call이 실제로 호출되고 있음을 확인할 수 있다.

```

[ 49.571280] [System call] oslab_enqueue(); -----
[ 49.571282] QUEUE FRONT -----
[ 49.571283] 1
[ 49.571284] QUEUE REAR -----
[ 49.571375] [System call] oslab_enqueue(); -----
[ 49.571375] QUEUE FRONT -----
[ 49.571376] 1
[ 49.571377] 2
[ 49.571378] QUEUE REAR -----
[ 49.571382] [System call] oslab_enqueue(); -----
[ 49.571382] QUEUE FRONT -----
[ 49.571383] 1
[ 49.571384] 2
[ 49.571384] 3
[ 49.571385] QUEUE REAR -----
[ 49.571388] [Error] - ALREADY EXISTING VALUE
[ 49.571392] [System call] oslab_dequeue(); -----
[ 49.571392] QUEUE FRONT-----
[ 49.571393] 2
[ 49.571394] 3
[ 49.571394] QUEUE REAR-----
[ 49.571397] [System call] oslab_dequeue(); -----
[ 49.571398] QUEUE FRONT-----
[ 49.571399] 3
[ 49.571399] QUEUE REAR-----
[ 49.571402] [System call] oslab_dequeue(); -----
[ 49.571403] QUEUE FRONT-----
[ 49.571404] QUEUE REAR-----
yslee@yslee-VirtualBox:~/oslab_2015150220$ sudo vim result.txt

```

“dmesg” 명령어의 실행 결과이다.

우선 1, 2, 3 이 front 부터 차례로 queue에 저장되는 것을 확인할 수 있다. 이후 3을 한 번 더 enqueue했는데 이 경우 3은 이미 큐 안에 존재하는 수 이기 때문에 Error 문이 출력되고 큐에는 저장되지 않았다.

다음으로 1, 2, 3 이 차례로 dequeue 되는 것을 확인할 수 있다. 이때 기존 front 값이 dequeue되면, 큐 내부의 나머지 값들이 한 칸 씩 앞으로 이동함을 알 수 있다.

Problem and Solution

git 을 사용한 코드 관리

```

yslee@yslee-VirtualBox:~/os_assignments$ git branch
* main

```

여러 파일과 코드를 작성하고 수정하는 과정에서 이전까지의 작업 내역을 기억하고 확인하는 데 어려움을 겪었다. 또한 코드나 캡처 이미지 등 파일 전체를 host와 guest 간에 주고 받는 과정에서 이를 간편하게 할 방법이 필요했다. 그래서 과제 소스파일들을 git으로 만들어 github에 올린 다음, commit과 fetch 등을 활용하여 위와 같은 어려움을 해소할 수 있었다.