

# OS KERNEL (system call, synchronization) REPORT

201514768

임유탉

- os\_kboard.c-ring\_buffer algorithm

```
int ring[MAX_CLIP] = {0};
long ring_lp = 0;
long ring_cp = 0;
long ring_len = 0;

void ring_push(int data)
{
    ring[ring_lp++] = data;
    ring_len++;

    if (ring_lp == MAX_CLIP) {
        ring_lp = 0;
    }
}

void ring_pop(void)
{
    if (ring_cp == MAX_CLIP) {
        ring_cp = 0;
    }
    ring_len--;
    ring_cp++;
}
```

- os\_kboard.c-enqueue

```
long do_sys_kb_enqueue(int item){
    printk(KERN_DEBUG "os201514768: do_sys_kb_enqueue() CALLED!! item=%d\n", item);

    if(ring_len == 5) {
        printk(KERN_DEBUG "os201514768: do_sys_kb_enqueue() full queue!!\n");
        return -1;
    }
    if(item < 0){
        printk(KERN_DEBUG "os201514768: do_sys_kb_enqueue() input negative!!\n");
        return -2;
    }
    ring_push(item);
    /*printk("1 : %d\n", ring[0]);
    printk("2 : %d\n", ring[1]);
    printk("3 : %d\n", ring[2]);
    printk("4 : %d\n", ring[3]);
    printk("5 : %d\n", ring[4]);*/

    return 0;
}
```

- os\_kboard.c-dequeue

```
long do_sys_kb_dequeue(int *user_buf){
    printk(KERN_DEBUG "os201514768: do_sys_kb_dequeue() CALLED!!\n");

    if(ring_len == 0){
        printk(KERN_DEBUG "os201514768: do_sys_kb_dequeue() hollow queue!!\n");
        return -1;
    }

    ring_pop();
    copy_to_user(user_buf , &ring[ring_cp - 1], sizeof(int));

    /*printk("1 : %d\n", ring[0]);
    printk("2 : %d\n", ring[1]);
    printk("3 : %d\n", ring[2]);
    printk("4 : %d\n", ring[3]);
    printk("5 : %d\n", ring[4]);*/

    return 0;
}
```

- kboard.h

```
#include <stdio.h>

#define sys_enqueue 335
#define sys_dequeue 336

long kboard_copy(int clip)
{
    syscall(sys_enqueue, clip);
}
int kboard_paste(int *clip)
{
    syscall(sys_dequeue, clip);
}
```

- copy.c

```
#include <unistd.h>
#include <sys/syscall.h>
#include "kboard.h"

int main()
{
    int clip;

    scanf("%d", &clip);

    int a = kboard_copy(clip);

    if(a == 0) return printf("Copy Success : %d\n", clip);
    else if(a == -1) return printf("ERROR : full queue\n");
    else if(a == -2) return printf("ERROR : input negative\n");
}
```

- paste.c

```
#include <unistd.h>
#include <sys/syscall.h>
#include "kboard.h"

int main()
{
    int clip;

    int a = kboard_paste(&clip);

    if(a == 0) return printf("Paste Success : %d\n", clip);
    else if(a == -1) return printf("ERROR : hollow queue\n");
}
```

어려웠던 점 및 해결 방안

처음엔 ring\_buffer algorithm을 kernel에 구현한다는 것이 와 닿지 않아 헛갈렸다. 하지만 초반에 system call을 만들다 보니 어떻게 user 모드와 kernel 모드가 communication하는지 감이 왔다. 일단은 과제 ppt에 적혀있는 대로 ring\_buffer algorithm이 되는지 확인하기 위해 user 모드에서 compile하였고 이를 kernel에 이식하였다. kernel에 이식하는 것을 알게 된 과정은 과제 ppt를 다시 정독하고 kernel의 구조와 user모드, kernel모드의 차이점을 인지하고 각각의

역할을 분리하여 적어보았다. 그 결과 kernel안에 ring\_buffer구현을 enqueue, dequeue 부분에 attraction하여 os\_kernel.c를 구성하게 되었다. 전체적인 structure은 user모드에서 system call이 kernel모드로 넘어가고 이에 대한 return value를 이용하여 출력을 하는 방식을 이용했다. 이로써 나온 결과화면을 밑에 캡처하였다.

- result

```
ubuntu@os201514768:~$ ./paste
ERROR : hollow queue
ubuntu@os201514768:~$ sudo dmesg -c
[ 40.356849] os201514768: do_sys_kb_dequeue() CALLED!!
[ 40.356851] os201514768: do_sys_kb_dequeue() hollow queue!!
ubuntu@os201514768:~$ ./copy
111
Copy Success : 111
ubuntu@os201514768:~$ ./copy
222
Copy Success : 222
ubuntu@os201514768:~$ ./copy
333
Copy Success : 333
ubuntu@os201514768:~$ ./paste
Paste Success : 111
ubuntu@os201514768:~$ ./copy
444
Copy Success : 444
ubuntu@os201514768:~$ ./copy
555
Copy Success : 555
ubuntu@os201514768:~$ ./copy
666
Copy Success : 666
ubuntu@os201514768:~$ ./copy
777
ERROR : full queue
ubuntu@os201514768:~$ sudo dmesg -c
[ 57.610225] os201514768: do_sys_kb_enqueue() CALLED!! item=111
[ 61.035186] os201514768: do_sys_kb_enqueue() CALLED!! item=222
[ 62.682444] os201514768: do_sys_kb_enqueue() CALLED!! item=333
[ 67.348537] os201514768: do_sys_kb_dequeue() CALLED!!
[ 73.362442] os201514768: do_sys_kb_enqueue() CALLED!! item=444
[ 75.169770] os201514768: do_sys_kb_enqueue() CALLED!! item=555
[ 76.833533] os201514768: do_sys_kb_enqueue() CALLED!! item=666
[ 80.706136] os201514768: do_sys_kb_enqueue() CALLED!! item=777
[ 80.706138] os201514768: do_sys_kb_dequeue() full queue!!
ubuntu@os201514768:~$ ./paste
Paste Success : 222
ubuntu@os201514768:~$ ./paste
Paste Success : 333
ubuntu@os201514768:~$ ./paste
Paste Success : 444
ubuntu@os201514768:~$ ./paste
Paste Success : 555
ubuntu@os201514768:~$ ./paste
Paste Success : 666
ubuntu@os201514768:~$ ./copy
-1
ERROR : full queue
ubuntu@os201514768:~$ sudo dmesg -c
[ 100.435723] os201514768: do_sys_kb_dequeue() CALLED!!
[ 103.019862] os201514768: do_sys_kb_dequeue() CALLED!!
[ 105.404067] os201514768: do_sys_kb_dequeue() CALLED!!
[ 106.098775] os201514768: do_sys_kb_dequeue() CALLED!!
[ 107.339816] os201514768: do_sys_kb_dequeue() CALLED!!
[ 121.081118] os201514768: do_sys_kb_enqueue() CALLED!! item=-1
[ 121.081120] os201514768: do_sys_kb_dequeue() input negative!!
```

## + synchronization

### - copy.c

```
#include <unistd.h>
#include <sys/syscall.h>
#include "kboard.h"

int main()
{
    /*    int clip;

    scanf("%d", &clip);

    int a = kboard_copy(clip);

    if(a == 0) return printf("Copy Success : %d\n", clip);
    else if(a == -1) return printf("ERROR : full queue\n");
    else if(a == -2) return printf("ERROR : input negative\n");
    */

    int i = 1;
    int a;

    while(1){
        a = kboard_copy(i);
        if(a == 1){
            while(1){
                a = kboard_copy(i);
                if(a != 1) break;
            }
        }
        i++;
    }
    return 0;
}
```

### - paste.c

```
#include <unistd.h>
#include <sys/syscall.h>
#include "kboard.h"

int main()
{
    /*int clip;

    int a = kboard_paste(&clip);

    if(a == 0) return printf("Paste Success : %d\n", clip);
    else if(a == -1) return printf("ERROR : hollow queue\n");*/

    int clip;
    int i = 0;
    int a;
    int count = 0;

    while(1){
        a = kboard_paste(&clip);
        if(a == 1) i--;

        i++;
        count++;

        if(clip != i) { break;}
    }
    printf("real value: %d\n", clip);
    printf("musbe: %d\n", i - 1);
    printf("iteration: %d\n", count - 1);
    return 0;
}
```

- synchronization 문제 (os\_keyboard.c)

```
#define MAX_CLIP 5

int ring[MAX_CLIP] = {-7777};

long ring_lp = 0;
long ring_cp = 0;
long ring_len = 0;

void ring_push(int data)
{
    ring_len++;
    ring[ring_lp++] = data;
    if(ring_lp == MAX_CLIP) {
        ring_lp = 0;
    }
}

void ring_pop(void)
{
    ring_len--;
    ring_cp++;
    if(ring_cp == MAX_CLIP) {
        ring_cp = 0;
    }
}

long do_sys_kb_enqueue(int item){

    if(ring_len == 5) return 1;
    if(item < 0){
        printk(KERN_DEBUG "os201514768: do_sys_kb_enqueue() input negative!!\n");
        return -2;
    }

    ring_push(item);
    return 0;
}

long do_sys_kb_dequeue(int *user_buf){

    if(ring_len == 0) return 1;

    copy_to_user(user_buf, &ring[ring_cp], sizeof(int));
    ring[ring_cp] = -7777;

    ring_pop();

    return 0;
}
```

```
ubuntu@os201514768:~$ ./paste
real value: -7777
musbe: 3925
iteration: 3932
ubuntu@os201514768:~$ █
```

```
ubuntu@os201514768:~$ ./copy
█
```

- spinlock를 이용한 synchronization 문제 해결 (os\_keyboard.c)

```
#include <linux/spinlock.h>

#define MAX_CLIP 5

int ring[MAX_CLIP] = {-7777};

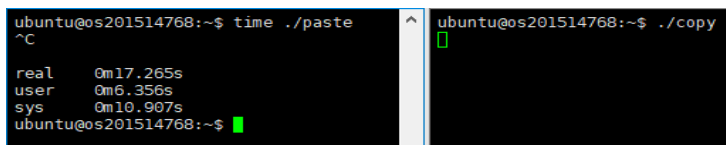
long ring_lp = 0;
long ring_cp = 0;
long ring_len = 0;
int count = 0;
spinlock_t lock;

void ring_push(int data){
    ring_len++;
    ring[ring_lp++] = data;
    if(ring_lp == MAX_CLIP) ring_lp = 0;
}

void ring_pop(void){
    ring_len--;
    ring_cp++;
    if(ring_cp == MAX_CLIP) ring_cp = 0;
}

long do_sys_kb_enqueue(int item){
    if(count == 0) spin_lock_init(&lock);
    if(ring_len == 5) return 1;
    spin_lock(&lock);
    if(item < 0){
        printk(KERN_DEBUG "os201514768: do_sys_kb_enqueue() input negative!!\n");
        return -2;
    }
    ring_push(item);
    spin_unlock(&lock);
    count++;
    return 0;
}

long do_sys_kb_dequeue(int *user_buf){
    if(count == 0) spin_lock_init(&lock);
    if(ring_len == 0) return 1;
    spin_lock(&lock);
    copy_to_user(user_buf, &ring[ring_cp], sizeof(int));
    ring[ring_cp] = -7777;
    ring_pop();
    spin_unlock(&lock);
    count++;
    return 0;
}
```



```
ubuntu@os201514768:~$ time ./paste
^C
real    0m17.265s
user    0m6.356s
sys     0m10.907s
ubuntu@os201514768:~$

ubuntu@os201514768:~$ ./copy
█
```

### 어려웠던 점 및 해결 방안

이번 과제에서 synchronization을 해결하는 과정은 쉬웠지만 synchronization을 일으키는 상황을 생각해내는 부분에서 어려웠다. ppt를 계속 읽어보며 과제의 의도를 파악하려 했고 구글에서 synchronization에 관련된 문제들과 솔루션들을 찾아보았다. 특히 솔루션들을 보며 왜 이렇게 해결하는지를 생각하며 왜 그런 문제가 발생하는지 더 쉽게 생각해 낼 수 있었다. synchronization부분 뿐 아니라 kernel에서 작업하는 부분에 대해 'fork retry resource temporarily unavailable'가 계속 뜨는 문제에 대해 아무런 명령어를 받지 않아 그대로 instance를 지워버린 부분에서 실수가 있었다. 후에 instance를 107번 포트로 재생성하여 작업했고 다시 같은 오류나 아예 들어가지지 않는 상황도 나왔지만 Jcloud에서 instance를 종료했다가 다시 시작하니 문제를 해결 할 수 있었다.