



임베디드 소프트웨어

Assignment_1

2021.4

201611218

이기현

0. 라이브러리에서 사용된 함수의 내부는 모두 커널에 접근하기 위한 ioctl 함수로 구성되어 있습니다. 따라서 커널 내 함수만 설명하도록 하겠습니다.

1. 주요 변수 설명

<code>struct param</code>	ioctl의 arg로 인자들을 통째로 전달하기 위해서 파라미터 struct를 선언하였습니다.
<code>static struct msg_node msg_node_head[10];</code>	10개의 메시지 큐를 배열로 만들었습니다.
<code>int reference_counter[10] = { 0 };</code>	프로세스들의 개수를 관리하는 레퍼런스 카운터를 선언했습니다. 카운트는 사용자가 msgget에 성공할 때 1증가하고 msgclose에 성공할 때 1감소합니다.
<code>wait_queue_head_t my_wq;</code>	중지된 프로세스들을 저장하는 wait_queue를 미리 선언하고 커널이 초기화될 때 같이 초기화 해주었습니다.

2. 함수 설명

<code>int get_queue_length(int key)</code>	Description	key에 해당하는 큐에 저장되어 있는 메시지 개수를 반환해주는 함수입니다.
	Parameters	알고 싶은 큐의 ID
	Return Value	해당 큐에 딸려있는 메시지 개수
<code>int get_queue_size(int key)</code>	Description	key에 해당하는 큐에 저장되어 있는 메시지들의 사이즈 합을 리턴하는 함수입니다.
	Parameters	알고 싶은 큐의 ID
	Return Value	해당 큐에 딸려있는 메시지들의 사이즈 합

<pre>bool isFull(int msqid, int msgsz)</pre>	Description	msqid에 해당하는 큐가 꽉 차있는 상태를 반환합니다. Full의 정의는 (메시지 개수 == MAX길이 메시지 사이즈+더해질 메시지 사이즈>맥스 사이즈) 입니다.
	Parameters	알고 싶은 큐의 ID, 더해질 메시지의 사이즈
	Return Value	큐가 꽉 차있다면 true, 아니면 false

<pre>bool isEmpty(int msqid)</pre>	Description	list_empty를 사용하여 msqid에 해당하는 큐가 비어 있는 상태를 반환합니다.
	Parameters	알고 싶은 큐의 ID
	Return Value	큐가 비어 있다면 true, 아니면 false

<pre>int msgget(int key, int msgflg)</pre>	Description	key에 해당하는 큐를 오픈하는 함수입니다. 누군가 큐를 쓰고 있을 때에도 오픈할지 말지를 msgflg로 설정할 수 있습니다.
	Parameters	오픈할 큐의 key, KU_IPC_CREAT or KU_IPC_EXCL
	Return Value	오픈 성공 시 큐의 ID, 실패 시 -1

<pre>int msgclose(int msqid)</pre>	Description	msqid에 해당하는 큐를 클로즈하는 함수입니다. id가 0~9가 아니거나 큐가 열려있지 않다면 실패합니다.
	Parameters	닫을 큐의 msqid
	Return Value	클로즈 성공 시 0, 실패 시 -1

<pre>int msgsnd(int msqid, struct msgbuf* msgp, int msgsz, int msgflg)</pre>	Description	사용자로부터 받은 메시지를 큐에 추가하는 함수입니다. Full이 아니라면 추가하고 Full이라면 대기할지 말지를 msgflg로 설정합니다.
	Parameters	큐 id, 입력받은 msg, 받은 msg의 사이즈, KU_IPC_NOWAIT or 0
	Return Value	send 성공 시 0, 실패 시 -1

<pre>int msgrcv(int msqid, struct msgbuf* msgp, int msgsz, long msgtyp, int msgflg)</pre>	Description	큐에서 메시지를 꺼내 사용자에게 전달하는 함수입니다. msgtyp에 해당하는 msg를 찾아서 전달합니다. Empty가 아니면 바로 꺼내고 Empty라면 대기할지 말지를 msgflg로 설정합니다.
	Parameters	큐 id, 입력받을 msg, 입력받을 msg의 사이즈, 받을 msg의 type, KU_IPC_NOWAIT KU_MSG_NOERROR
	Return Value	receive 성공 시 길이만큼, 실패 시 -1

3. 실행 결과

우선 메시지가 Full일 때를 체크해 보기 위해 MAXMSG를 4로, MAXVOL을 17*KU_IPC_MAXMSG로 정의해 두었습니다.

open close test

The screenshot shows a code editor on the left with the following C code in `ku_ipc.c`:

```
#include "ku_ipc_lib.c"

int main(){
    ku_msgget(3, KU_IPC_CREAT);
    ku_msgget(3, KU_IPC_CREAT);
    ku_msgget(3, KU_IPC_EXCL);
    ku_msgget(4, KU_IPC_EXCL);

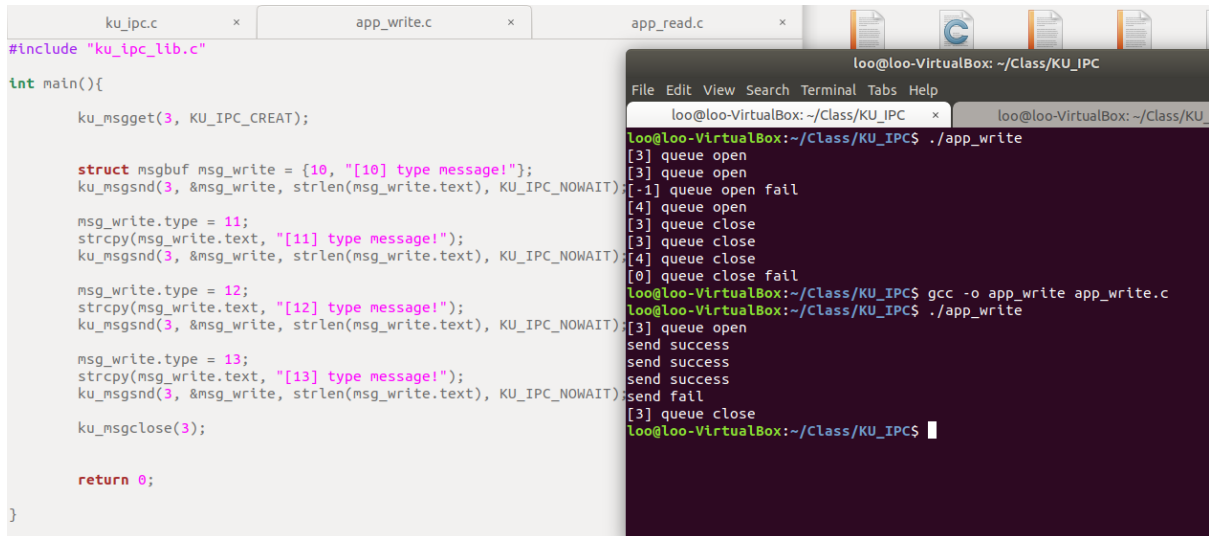
    ku_msgclose(3);
    ku_msgclose(3);
    ku_msgclose(4);
    ku_msgclose(0);
}
```

On the right, a terminal window shows the output of running `./app_write`:

```
loo@loo-VirtualBox: ~/Class/KU_IPC$ ./app_write
[3] queue open
[3] queue open
[-1] queue open fail
[4] queue open
[3] queue close
[3] queue close
[4] queue close
[0] queue close fail
loo@loo-VirtualBox: ~/Class/KU_IPC$
```

3번 큐를 CREAT flag로 두 번 열고 EXCL flg로 열 때 실패합니다. 오픈하지 않은 0번 큐를 close하려하자 실패를 리턴합니다.

send test



```
ku_ipc.c x app_write.c x app_read.c x
#include "ku_ipc_lib.c"
int main(){
    ku_msgget(3, KU_IPC_CREAT);

    struct msgbuf msg_write = {10, "[10] type message!"};
    ku_msgsnd(3, &msg_write, strlen(msg_write.text), KU_IPC_NOWAIT);

    msg_write.type = 11;
    strcpy(msg_write.text, "[11] type message!");
    ku_msgsnd(3, &msg_write, strlen(msg_write.text), KU_IPC_NOWAIT);

    msg_write.type = 12;
    strcpy(msg_write.text, "[12] type message!");
    ku_msgsnd(3, &msg_write, strlen(msg_write.text), KU_IPC_NOWAIT);

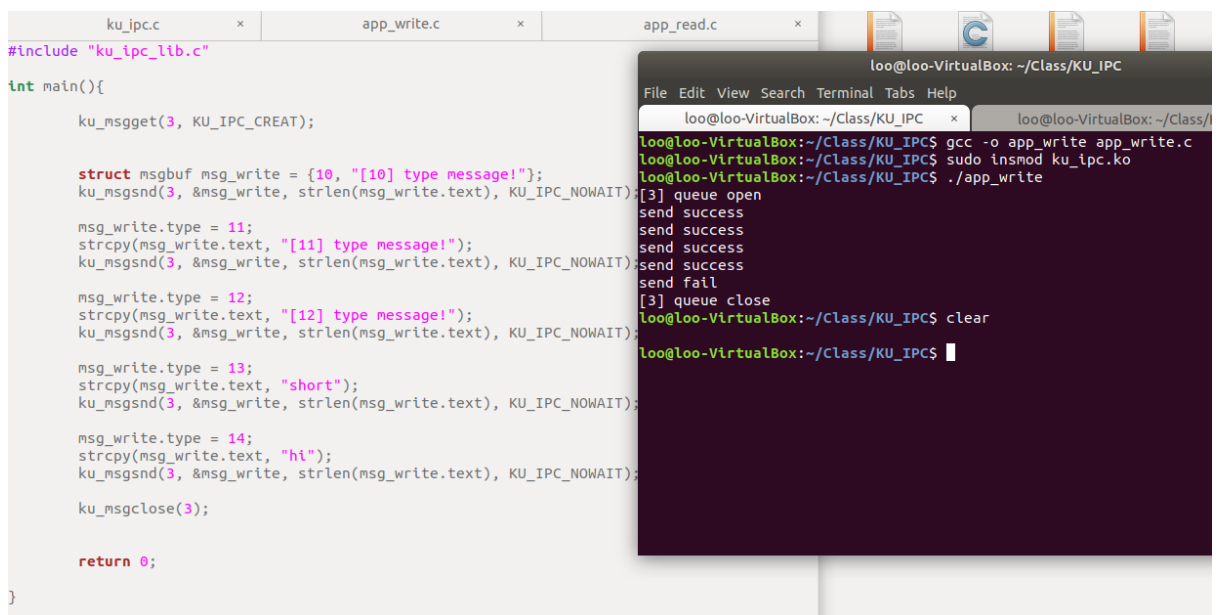
    msg_write.type = 13;
    strcpy(msg_write.text, "[13] type message!");
    ku_msgsnd(3, &msg_write, strlen(msg_write.text), KU_IPC_NOWAIT);

    ku_msgclose(3);

    return 0;
}

loo@loo-VirtualBox: ~/Class/KU_IPC
File Edit View Search Terminal Tabs Help
loo@loo-VirtualBox: ~/Class/KU_IPC x loo@loo-VirtualBox: ~/Class/KU_IPC
loo@loo-VirtualBox:~/Class/KU_IPC$ ./app_write
[3] queue open
[3] queue open
[-1] queue open fail
[4] queue open
[3] queue close
[3] queue close
[4] queue close
[0] queue close fail
loo@loo-VirtualBox:~/Class/KU_IPC$ gcc -o app_write app_write.c
loo@loo-VirtualBox:~/Class/KU_IPC$ ./app_write
[3] queue open
send success
send success
send success
send fail
[3] queue close
loo@loo-VirtualBox:~/Class/KU_IPC$
```

맥스 길이를 4로, 맥스 사이즈를 17*KU_IPC_MAXMSG로 define해 놓았기 때문에 넣으려는 4번째문자열의 사이즈 합이 18*4 로 MAXVOL을 넘게 됩니다. 거기에 msgflg가 NOWAIT이므로 send에 바로 실패합니다.



```
ku_ipc.c x app_write.c x app_read.c x
#include "ku_ipc_lib.c"
int main(){
    ku_msgget(3, KU_IPC_CREAT);

    struct msgbuf msg_write = {10, "[10] type message!"};
    ku_msgsnd(3, &msg_write, strlen(msg_write.text), KU_IPC_NOWAIT);

    msg_write.type = 11;
    strcpy(msg_write.text, "[11] type message!");
    ku_msgsnd(3, &msg_write, strlen(msg_write.text), KU_IPC_NOWAIT);

    msg_write.type = 12;
    strcpy(msg_write.text, "[12] type message!");
    ku_msgsnd(3, &msg_write, strlen(msg_write.text), KU_IPC_NOWAIT);

    msg_write.type = 13;
    strcpy(msg_write.text, "short");
    ku_msgsnd(3, &msg_write, strlen(msg_write.text), KU_IPC_NOWAIT);

    msg_write.type = 14;
    strcpy(msg_write.text, "hi");
    ku_msgsnd(3, &msg_write, strlen(msg_write.text), KU_IPC_NOWAIT);

    ku_msgclose(3);

    return 0;
}

loo@loo-VirtualBox: ~/Class/KU_IPC
File Edit View Search Terminal Tabs Help
loo@loo-VirtualBox: ~/Class/KU_IPC x loo@loo-VirtualBox: ~/Class/KU_IPC
loo@loo-VirtualBox:~/Class/KU_IPC$ gcc -o app_write app_write.c
loo@loo-VirtualBox:~/Class/KU_IPC$ sudo insmod ku_ipc.ko
loo@loo-VirtualBox:~/Class/KU_IPC$ ./app_write
[3] queue open
send success
send success
send success
send success
send fail
[3] queue close
loo@loo-VirtualBox:~/Class/KU_IPC$ clear
loo@loo-VirtualBox:~/Class/KU_IPC$
```

맥스 사이즈는 맞지만 5번 째 메시지가 맥스 길이 4를 넘기므로 send에 실패합니다.

The screenshot shows a code editor with three files: `ku_ipc.c`, `app_write.c`, and `app_read.c`. The `ku_ipc.c` file contains the following code:

```
#include "ku_ipc_lib.c"

int main(){

    ku_msgget(3, KU_IPC_CREAT);

    struct msgbuf msg_write = {10, "[10] type message!"};
    ku_msgsnd(3, &msg_write, strlen(msg_write.text), KU_IPC_NOWAIT);

    msg_write.type = 11;
    strcpy(msg_write.text, "[11] type message!");
    ku_msgsnd(3, &msg_write, strlen(msg_write.text), KU_IPC_NOWAIT);

    msg_write.type = 12;
    strcpy(msg_write.text, "[12] type message!");
    ku_msgsnd(3, &msg_write, strlen(msg_write.text), KU_IPC_NOWAIT);

    msg_write.type = 13;
    strcpy(msg_write.text, "[13] type message!");
    ku_msgsnd(3, &msg_write, strlen(msg_write.text), 0);

    return 0;
}
```

The terminal window shows the following commands and output:

```
loo@loo-VirtualBox: ~/Class/KU_IPC
loo@loo-VirtualBox:~/Class/KU_IPC$ sudo rmmod ku_ipc
loo@loo-VirtualBox:~/Class/KU_IPC$ sudo dmesg -C
loo@loo-VirtualBox:~/Class/KU_IPC$ gcc -o app_write app_write.c
loo@loo-VirtualBox:~/Class/KU_IPC$ gcc -o app_read app_read.c
loo@loo-VirtualBox:~/Class/KU_IPC$ sudo insmod ku_ipc.ko
loo@loo-VirtualBox:~/Class/KU_IPC$ ./app_write
[3] queue open
send success
send success
send success
```

msgflg에 0을 주고 4번째 메시지를 send하면 Full이 되어 프로세스가 wait_queue에 들어갑니다.

The screenshot shows the same code editor as before, but the `ku_ipc.c` file now contains the following code:

```
#include "ku_ipc_lib.c"

int main(){

    struct msgbuf msg_read;
    ku_msgrcv(3, &msg_read, 50, 11, KU_MSG_NOERROR);
    printf("read: %s\n", msg_read.text);

    ku_msgclose(3);
    return 0;
}
```

The terminal window shows the following commands and output:

```
loo@loo-VirtualBox: ~/Class/KU_IPC
loo@loo-VirtualBox:~/Class/KU_IPC$ ./app_read
receive success & msg len [18]
read: [11] type message!
[3] queue close
loo@loo-VirtualBox:~/Class/KU_IPC$
```

write 프로세스가 대기 중일 때 read를 실행시켜 msgtyp이 11인 메시지를 꺼내왔습니다.

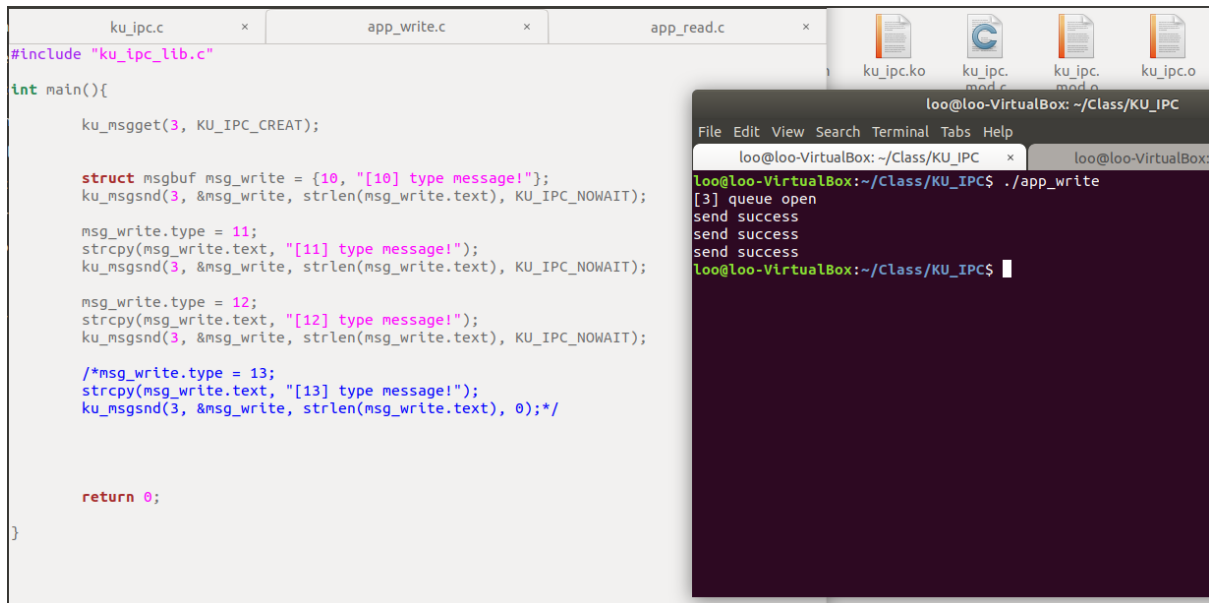
The terminal window shows the following commands and output:

```
loo@loo-VirtualBox:~/Class/KU_IPC$ ./app_write
[3] queue open
send success
send success
send success
send success
loo@loo-VirtualBox:~/Class/KU_IPC$
```

receive한 뒤엔 wait_queue에서 잠자고 있던 프로세스를 깨웁니다. 깨우는 조건에 !isFull을

넣어 Full이 아닐 때만 프로세스가 깰 수 있도록 했습니다. 깨고 나니 receive 덕분에 Full상태에서 벗어 났으므로 send에 성공합니다.

receive test



```
#include "ku_ipc_lib.c"

int main(){
    ku_msgget(3, KU_IPC_CREAT);

    struct msgbuf msg_write = {10, "[10] type message!"};
    ku_msgsnd(3, &msg_write, strlen(msg_write.text), KU_IPC_NOWAIT);

    msg_write.type = 11;
    strcpy(msg_write.text, "[11] type message!");
    ku_msgsnd(3, &msg_write, strlen(msg_write.text), KU_IPC_NOWAIT);

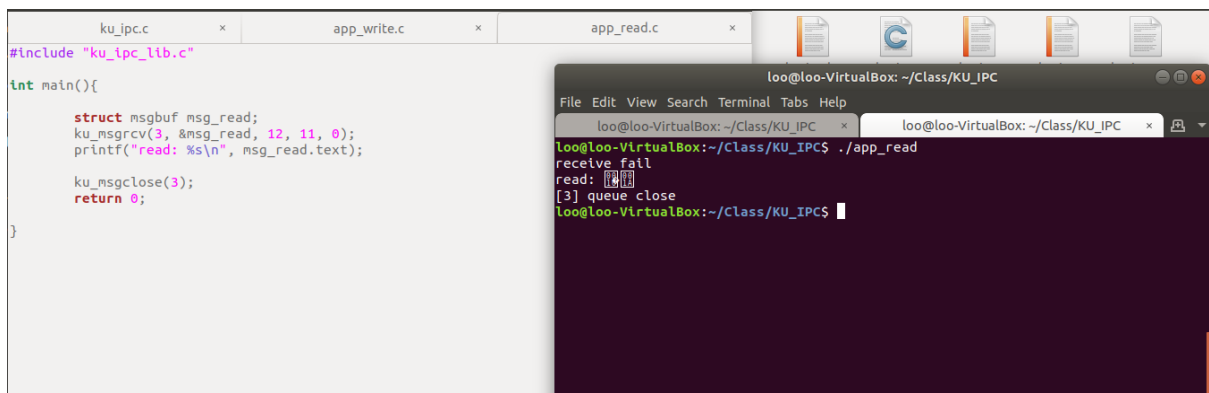
    msg_write.type = 12;
    strcpy(msg_write.text, "[12] type message!");
    ku_msgsnd(3, &msg_write, strlen(msg_write.text), KU_IPC_NOWAIT);

    /*msg_write.type = 13;
    strcpy(msg_write.text, "[13] type message!");
    ku_msgsnd(3, &msg_write, strlen(msg_write.text), 0);*/

    return 0;
}
```

```
loo@loo-VirtualBox: ~/Class/KU_IPC
loo@loo-VirtualBox:~/Class/KU_IPC$ ./app_write
[3] queue open
send success
send success
send success
loo@loo-VirtualBox:~/Class/KU_IPC$
```

먼저 type이 10, 11, 12인 메시지들을 성공적으로 넣습니다.



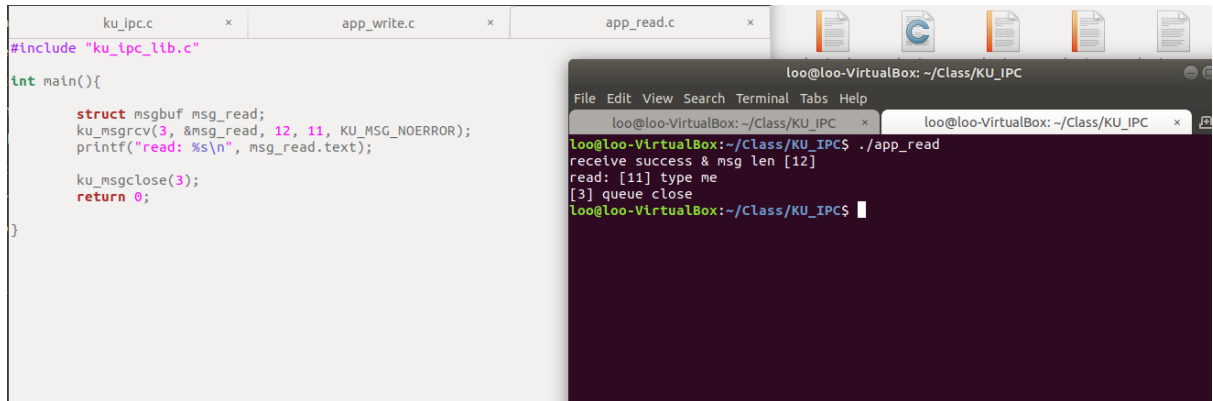
```
#include "ku_ipc_lib.c"

int main(){
    struct msgbuf msg_read;
    ku_msgrcv(3, &msg_read, 12, 11, 0);
    printf("read: %s\n", msg_read.text);

    ku_msgclose(3);
    return 0;
}
```

```
loo@loo-VirtualBox: ~/Class/KU_IPC
loo@loo-VirtualBox:~/Class/KU_IPC$ ./app_read
receive fail
read: [11]
[3] queue close
loo@loo-VirtualBox:~/Class/KU_IPC$
```

type이 11인 메시지를 길이 12만큼 받으려고 했으나 받으려는 메시지의 길이가 18로 12보다 깁니다. msgflg가 0으로 NOERROR가 아니기 때문에 바로 fail을 리턴합니다.



The screenshot shows a code editor with three tabs: `ku_ipc.c`, `app_write.c`, and `app_read.c`. The `ku_ipc.c` tab is active, showing the following code:

```
#include "ku_ipc_lib.c"

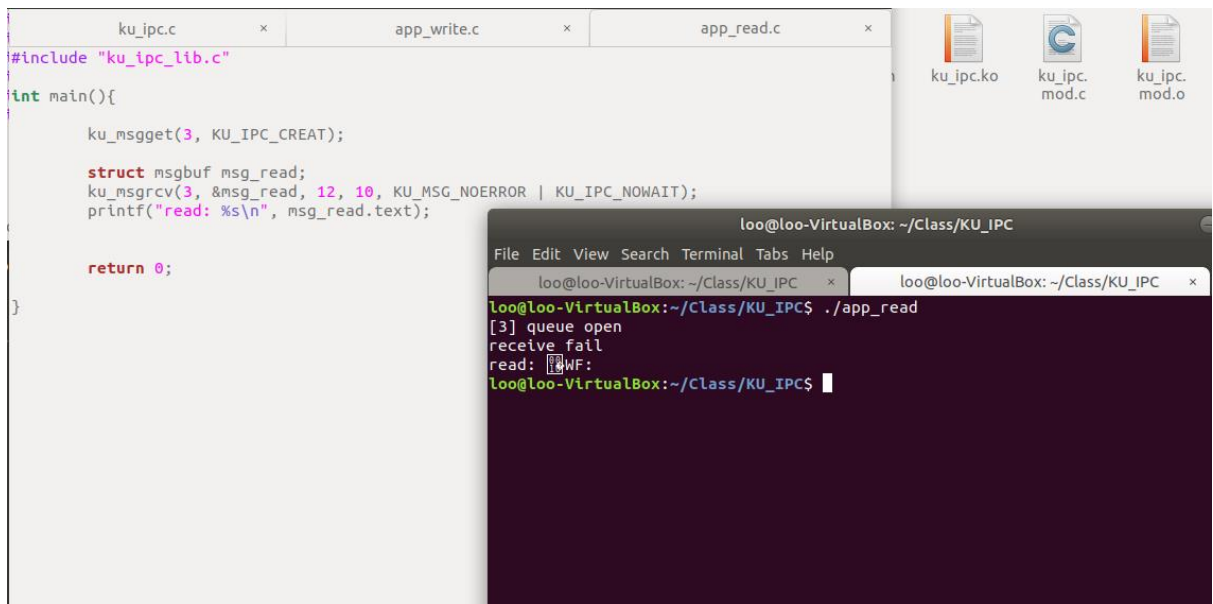
int main(){
    struct msgbuf msg_read;
    ku_msgrcv(3, &msg_read, 12, 11, KU_MSG_NOERROR);
    printf("read: %s\n", msg_read.text);

    ku_msgclose(3);
    return 0;
}
```

Overlaid on the code editor is a terminal window titled `loo@loo-VirtualBox: ~/Class/KU_IPC`. The terminal shows the execution of `./app_read` and the output:

```
loo@loo-VirtualBox:~/Class/KU_IPC$ ./app_read
receive success & msg len [12]
read: [11] type me
[3] queue close
loo@loo-VirtualBox:~/Class/KU_IPC$
```

같은 방식으로 `msgflg`에 `NOERROR`를 줬을 땐 원하는 길이만큼 잘라서 `receive`를 하고 성공을 리턴합니다.



The screenshot shows the same code editor with the `ku_ipc.c` tab active. The code is updated to:

```
#include "ku_ipc_lib.c"

int main(){
    ku_msgget(3, KU_IPC_CREAT);

    struct msgbuf msg_read;
    ku_msgrcv(3, &msg_read, 12, 10, KU_MSG_NOERROR | KU_IPC_NOWAIT);
    printf("read: %s\n", msg_read.text);

    return 0;
}
```

The terminal window shows the execution of `./app_read` and the output:

```
loo@loo-VirtualBox:~/Class/KU_IPC$ ./app_read
[3] queue open
receive fail
read: [10]WF:
loo@loo-VirtualBox:~/Class/KU_IPC$
```

이번엔 커널을 초기화하고 시작과 동시에 `read`를 `NOWAIT`으로 호출한 결과, 큐가 비어있으므로 바로 `fail`을 리턴합니다.

The screenshot shows a code editor with three tabs: `ku_ipc.c`, `app_write.c`, and `app_read.c`. The `ku_ipc.c` file contains the following code:

```
#include "ku_ipc_lib.c"

int main(){

    ku_msgget(3, KU_IPC_CREAT);

    struct msgbuf msg_read;
    ku_msgrcv(3, &msg_read, 12, 10, KU_MSG_NOERROR | 0);
    printf("read: %s\n", msg_read.text);

    return 0;

}
```

Overlaid on the code editor is a terminal window titled `loo@loo-VirtualBox: ~/Class/KU_IPC`. The terminal shows the command `./app_read` being executed, resulting in the output `[3] queue open`.

같은 방법으로 비어 있는 상태에서 `msgflg`가 0인 상태로 `read`한 결과, 새로운 메시지가 들어 올 때 까지 `wait_queue`에 프로세스가 들어갑니다.

The screenshot shows the same code editor and terminal window. The `ku_ipc.c` file now contains more code:

```
#include "ku_ipc_lib.c"

int main(){

    struct msgbuf msg_write = {10, "[10] type message!"};
    ku_msgsnd(3, &msg_write, strlen(msg_write.text), KU_IPC_NOWAIT);

    msg_write.type = 11;
    strcpy(msg_write.text, "[11] type message!");
    ku_msgsnd(3, &msg_write, strlen(msg_write.text), KU_IPC_NOWAIT);

    msg_write.type = 12;
    strcpy(msg_write.text, "[12] type message!");
    ku_msgsnd(3, &msg_write, strlen(msg_write.text), KU_IPC_NOWAIT);

    /*msg_write.type = 13;
    strcpy(msg_write.text, "[13] type message!");
    ku_msgsnd(3, &msg_write, strlen(msg_write.text), 0);*/

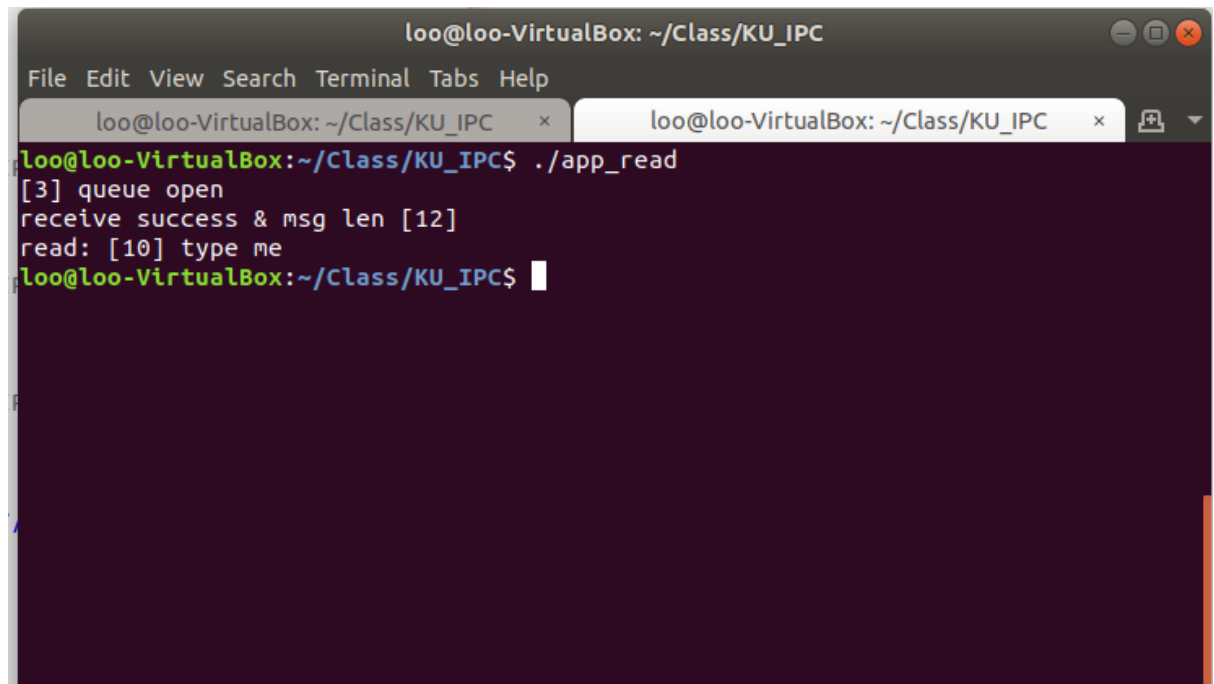
    ku_msgclose(3);

    return 0;

}
```

The terminal window shows the command `./app_write` being executed, resulting in the output `send success` repeated three times, followed by `[3] queue close`.

`read`가 대기 중인 상태에서 `write`를 하여 큐가 비지 않게 만들고 프로세스를 깨우도록 했습니다.



```
loo@loo-VirtualBox: ~/Class/KU_IPC
File Edit View Search Terminal Tabs Help
loo@loo-VirtualBox: ~/Class/KU_IPC x loo@loo-VirtualBox: ~/Class/KU_IPC x
loo@loo-VirtualBox:~/Class/KU_IPC$ ./app_read
[3] queue open
receive success & msg len [12]
read: [10] type me
loo@loo-VirtualBox:~/Class/KU_IPC$
```

receive가 깨는 조건에 `!isEmpty()`를 주어 큐가 비어 있지 않을 때만 깎 수 있도록 했습니다. 깨어난 read 프로세스는 큐가 비어있지 않다는 것을 확인하고 해당 type의 메시지를 찾아 성공적으로 receive합니다.