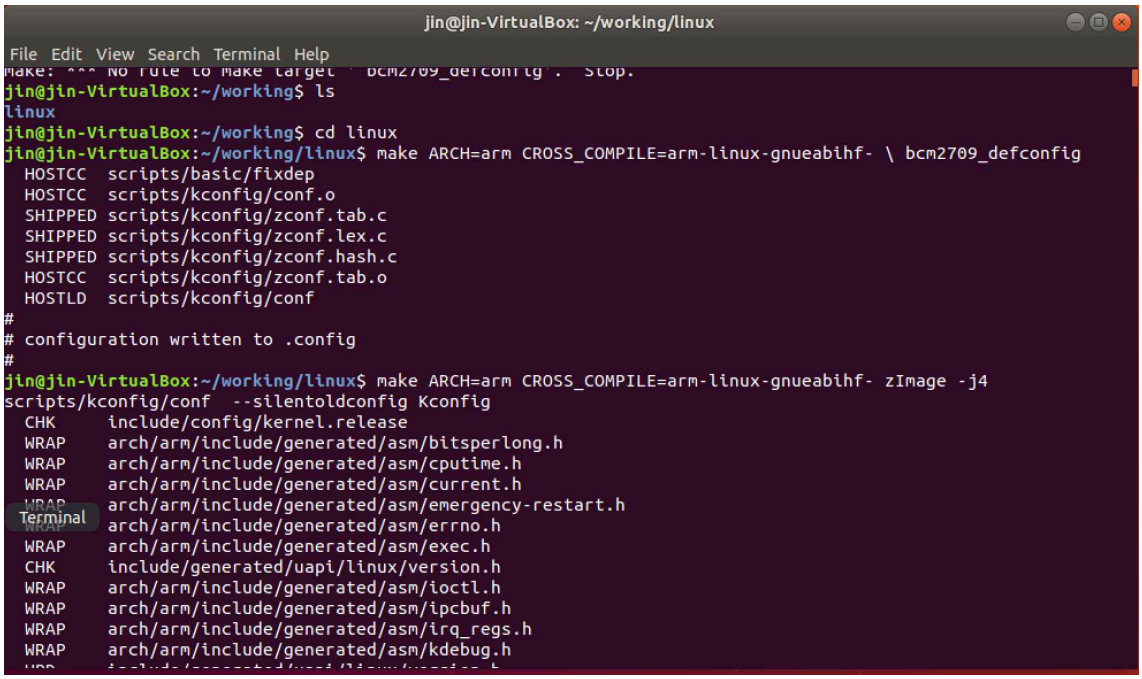
IoT 소프트웨어 과제15

2015104124
전자공학과
진우빈

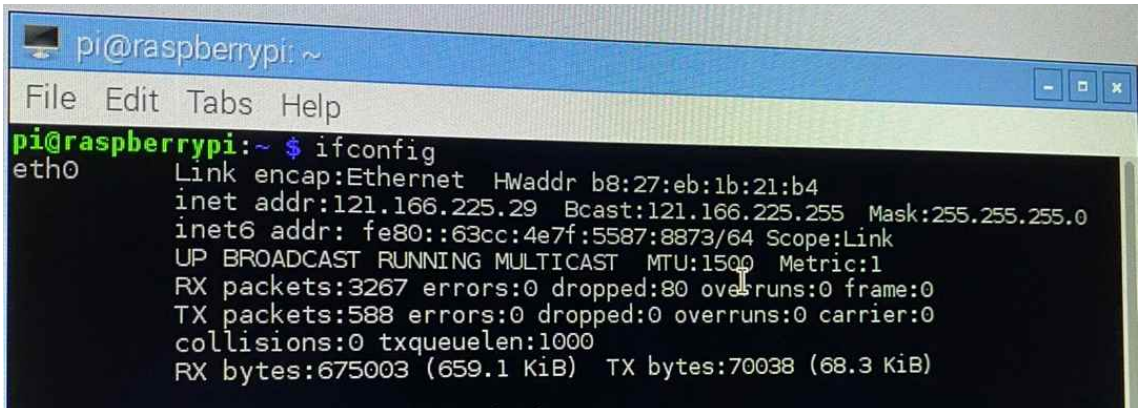
실습예제)

시스템 콜을 리눅스 우분투 환경에서 실습하는 실습입니다. 먼저 라즈베리파이와 우분투 환경 사이에서 크로스 컴파일을 진행할 것이기 때문에 그에 대한 커널 컴파일과 모듈 컴파일을 진행하였습니다. 대략 40분정도의 시간이 걸리는 작업이였습니다.

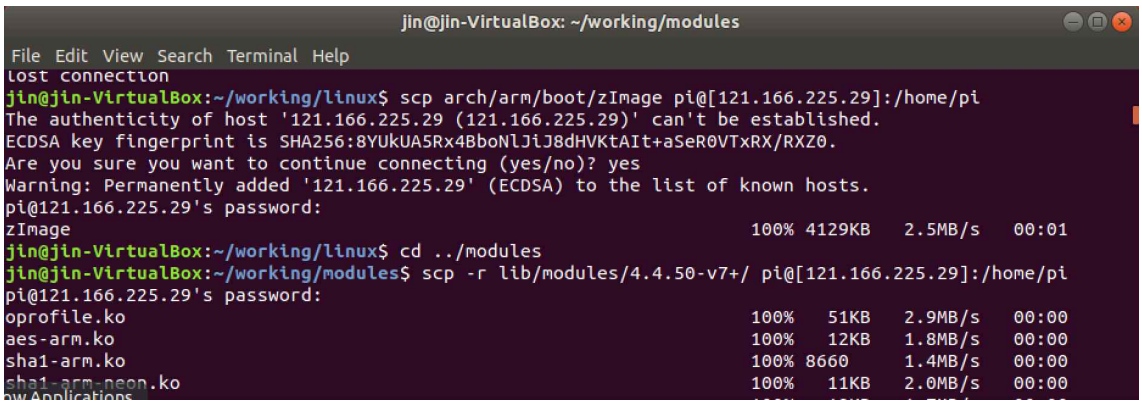


이후에 라즈비안이 설치되어 있는 SD카드를 삽입하여 마운트 시킨 후 해당 os의 안으로 커널 이미지와 모듈을 복사시켜 넣어주었습니다.

SD카드를 마운트 해제한 후 이제 커널 이미지와 모듈 디렉토리들을 라즈비안의 mnt와 filesystem이 아닌 home/pi 영역으로 즉 transmit 하였습니다. 이 때 scp 명령어를 사용하여 파일들을 전송해야 했기 때문에 라즈베리파이의 IP주소가 필요하였습니다. 따라서 라즈비안을 연결하여 IP주소를 확인하였습니다.

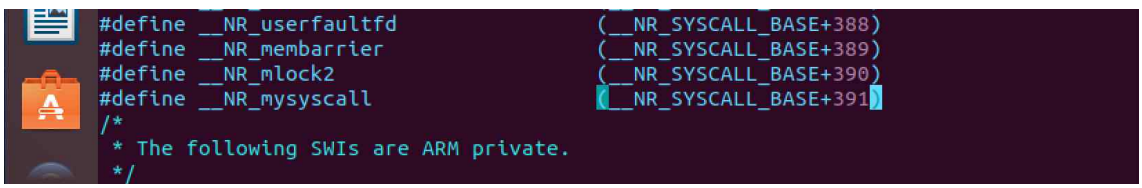


이더넷을 통해 연결하였으므로 라즈베리 파이의 주소는 121.166.225.29 인 것을 확인하였습니다.



이후 IP 주소를 입력하며 커널 이미지와 모듈 디렉토리들을 전송하였습니다.

본격적으로 시스템 콜을 이용해 컴파일을 실습하기 위해 unistd.h 헤더파일 안으로 들어가서 새롭게 지정할 함수를 정의하였습니다. __NR_mysyscall 이라는 이름으로 제일 마지막 함수 이후에 새로운 함수를 선언하였으며 이에 따른 시스템 콜 Table의 페이지 번호를 392번으로 설정해 주었습니다.



System Call을 하기위한 Call number를 추가한 이후에는 __NR_mysyscall이 불렸을 시 수행해야 할 함수를 정의해 주어야 했습니다. 따라서 sys_mysyscall 이라는 함수명으로 시스템 콜 함수를 등록해 주었습니다.

```
/* 385 */       CALL(sys_memfd_create)
                CALL(sys_bpf)
                CALL(sys_execveat)
                CALL(sys_userfaultfd)
                CALL(sys_membarrier)
                CALL(sys_mlock2)
                CALL(sys_mysyscall)
-- INSERT --                                          403,22-36     98%
```

이후 syscall.h 헤더파일에 들어가서 sys_mysyscall 함수를 프로토타입으로 정의하였으며 mysyscall_c 파일에서는 kernel에서의 print함수인 printk 명령어를 통해 HelloWorld를 출력하고 입력 parameter들을 간단하게 계산하는 수식을 넣어 주었습니다. (동작확인)

```
asmlinkage long sys_mlock2(unsigned long start, size_t len, int flags);

asmlinkage int sys_mysyscall(int n, int m);
:
```

```
jin@jin-VirtualBox: ~/working/linux
File  Edit  View  Search  Terminal  Help
#include <linux/unistd.h>
#include <linux/errno.h>
#include <linux/sched.h>

asmlinkage int sys_mysyscall(int n, int m) {
        printk("[Hello world] %d * %d = %d\n", n, m, n*m);
        return n*m;
}
~
```

그리고 나서 컴파일을 하기 위해 Makefile에 들어가서 object 파일을 선언하여 주었고 커널 함수가 하나 추가되었으므로 커널 컴파일을 다시 실행해 주었습니다.
이제 커널 영역에서의 함수 선언 및 컴파일은 끝났고 User영역에서 해당 시스템 콜을 사용할 함수를 만들어 줘야 하였습니다. 따라서 커널에서의 mysyscall을 불러오기 위한 User영역에서의 mysyscall함수를 정의하였습니다. 이 때는 linux 폴더안의 unistd.h 헤더파일을 include 시켜서 SystemCall Number를 불러옵니다.

```
jin@jin-VirtualBox: ~/working/syscall
File  Edit  View  Search  Terminal  Help
#include "/home/ubuntu/working/linux/arch/arm/include/uapi/asm/unistd.h"

int mysyscall(int n, int m) {
        return syscall(__NR_mysyscall, n, m);
}
~
```

이후 정의한 mysyscall.c 함수를 사용하기 위한 main 함수인 syscall_app.c 파일을 생성하였습니다.

```
jin@jin-VirtualBox: ~/working/syscall
File  Edit  View  Search  Terminal  Help
#include <stdio.h>

int main() {
        mysyscall(3,5);

        return 0;
}
```

유저 영역에서도 Makefile을 해줘야 컴파일이 가능합니다. 따라서 컴파일을 할 수 있게 도와주는 Makefile을 선언하였습니다. 이 때는 정의한 함수들의 정보와 사용할 컴파일러 및 라이브러리 등이 선언됩니다.

```
jin@jin-VirtualBox: ~/working/syscall
File  Edit  View  Search  Terminal  Help
LIB_NAME=newsyscall
APP_NAME=syscall_app

all: lib app

lib:
        arm-linux-gnueabihf-gcc -c $(LIB_NAME).c
        ar ruv $(LIB_NAME).a $(LIB_NAME).o

app:
        arm-linux-gnueabihf-gcc -o $(APP_NAME) $(APP_NAME).c $(LIB_NAME).a

clean:
        rm -rf $(LIB_NAME).o
        rm -rf $(LIB_NAME).a
        rm -rf $(LIB_NAME)
        rm -rf $(APP_NAME)
~
```

make를 통해 build한 이후 다시 sd카드를 mount 시켜서 새로 컴파일 한 syscall_app 및 커널 이미지를 copy해 주었습니다. 그리고 나서 scp 명령어를 통해 라즈베리파이의 /home/pi 안으로 syscall_app을 전송해 주었습니다. 이후 rasbian os 안에서 kernel image를 전송받은 kernel7.img 로 바꿔주고 우분투에서 컴파일된 ./systemcall_app 실행파일을 실행시켜 주었습니다.

```
[   11.980542] Bluetooth: HCI UART protocol BCM registered
[   12.140545] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
[   12.140570] Bluetooth: BNEP filters: protocol multicast
[   12.140600] Bluetooth: BNEP socket layer initialized
[   41.355709] [Hello world] 3 * 5 = 15
pi@raspberrypi:~ $
```

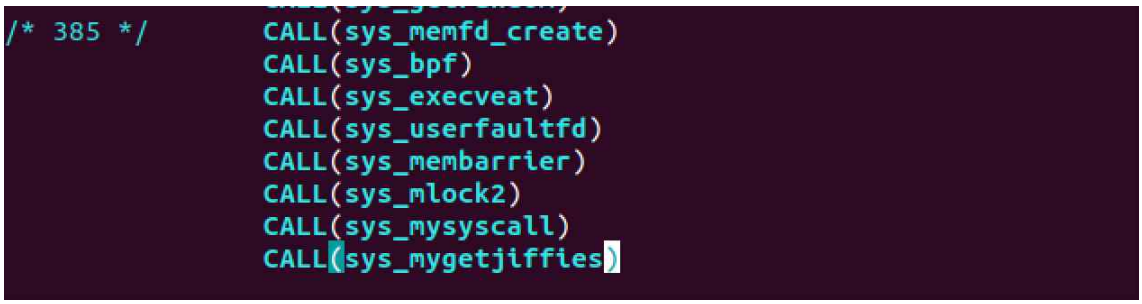
예상대로 출력이 원활히 된 것을 확인 하였습니다.

실습 과제)

시스템이 부팅되고 흐른 시간만큼의 초를 time_tick으로 받아오는 커널 안의 jiffies를 사용자가 확인할 수 있게끔 user영역에서 jiffies를 확인하는 과제입니다. 함수 실행 과정은 이렇습니다.

1. 유저영역에서 mygetjiffies함수로 jiffiy 값을 받고자 하는 변수의 메모리 주소를 넘긴다.
2. 커널영역에서 동작하는 mygetjiffies 함수는 jiffiy 값을 받아온다.
3. copy_to_user 명령어를 사용하여 받아온 jiffiy 값을 매개변수로 받아온 User 영역의 주소 안으로 jiffiy값을 새로 씌워준다.
4. 함수 return 후 User 영역에서 Application 함수를 실행하여 jiffiy 값을 확인한다.

먼저 실습예제에서처럼 SystemCall Number를 정의해 주었고 SystemCall Table에서 새로 정의한 함수 페이지를 호출할 수 있게끔 하였습니다. (__NR_mygetjiffies로 정의)



이후 syscall.h 헤더파일에 들어가서 sys_mygetjiffies 함수를 정의하였으며 커널 영역에서의 함수를 만들었습니다. Linux 우분투 환경의 경우 기본적으로 kernel 영역의 메모리와 user 영역의 메모리가 분리 되어 있습니다. 따라서 유저 영역에 kernel 영역에서 받은 jiffiy 값을 넘기기 위해 copy_to_user()를 사용하였습니다. 먼저 커널영역에서 get_jiffies_64 명령어를 통해 jiffy를 받아왔습니다. 그리고 넘겨온 값을 copy_to_user를 통해 매개변수인 p_jiffies의 포인터 주소 안으로 값을 deep copy 해주었습니다.

copy_to_user 에 대해서 찾아보면 from [kernel] 에서 n만큼의 블록 데이터를 to[user]에 써 넣으며 리턴 값은 복사되지 않은 바이트수를 리턴하고, 정상종료 시 0을 리턴 한다고 되어 있습니다. 따라서 to, from, n에 맞게 끔 매개 변수를 넣어주었습니다.

```
asmlinkage int sys_mysyscall(int n, int m);

asmlinkage int sys_mygetjiffies(unsigned long *p_jiffies);

#endif
                                                              896,1
```

jin@jin-VirtualBox: ~/working/linux

File Edit View Search Terminal Help

```
#include <linux/unistd.h>
#include <linux/errno.h>
#include <linux/sched.h>
#include <linux/jiffies.h>
#include <asm/uaccess.h>

asmlinkage int sys_mygetjiffies(unsigned long *p_jiffies) {
        unsigned long jiffies = (unsigned long)get_jiffies_64();
        printk("[Kernel jiffies] : %lu\n", jiffies);
        copy_to_user(p_jiffies, &jiffies, sizeof(jiffies));
        return 1;
}
~
```

커널에 새 함수가 추가 되었으므로 컴파일을 하기 위해 Makefile에 들어가서 object 파일을 선언하여 주었고 커널 컴파일을 다시 실행해 주었습니다. 이제 커널 영역에서의 함수 선언 및 컴파일은 끝났고 User영역에서 jiffiy를 받아오기 위한 함수를 만들어 줘야 하였습니다.
유저영역에서의 mygetjiffies 함수는 jiffiy의 type인 unsigned long으로 변수를 정의하고 이 때 커널영역에서의 mygetjiffies 함수를 불러와 인자로 정의된 p를 전달합니다. 그러면 system_call에서의 mygetjiffies는 전달받은 p의 메모리 주소에다 jiffiy 값을 copy해 줄 것입니다. 이후 syscall2_app.c 는 main문으로써 전달받은 p의 값을 출력합니다. real_p는 p의 값을 그대로 복사하여 전달받은 jiffiy를 printf를 통해 화면에 출력됩니다.

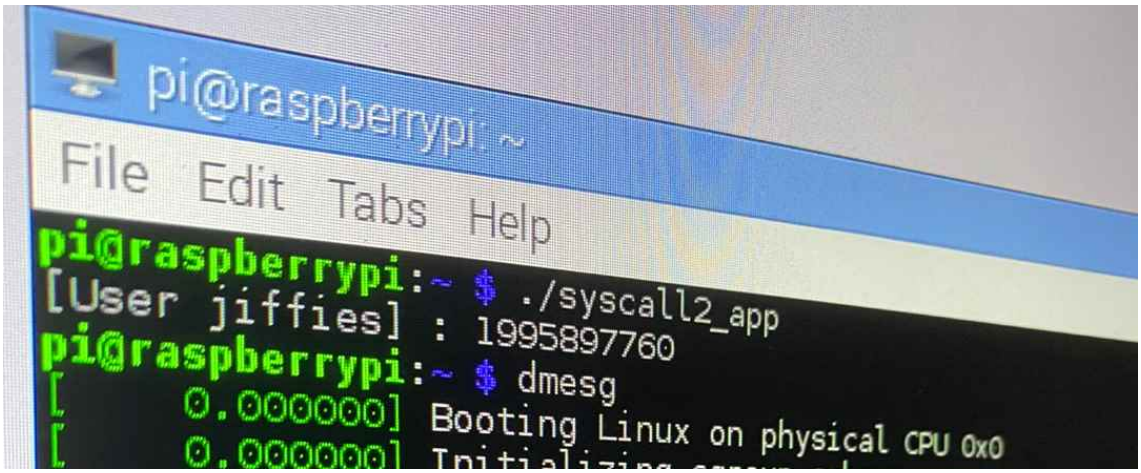
jin@jin-VirtualBox: ~/working/syscall2

File Edit View Search Terminal Help

```
#include "/home/jin/working/linux/arch/arm/include/uapi/asm/unistd.h"

int mygetjiffies() {
        unsigned long p;
        syscall(__NR__mygetjiffies, &p);
        return p;

}
~
~
```

```
jin@jin-VirtualBox: ~/working/syscall2
File  Edit  View  Search  Terminal  Help
#include <stdio.h>

int main() {
        unsigned long real_p;
        real_p = mygetjiffies();
        printf("[User jiffies] : %lu\n", real_p);
        return 0;
}
~
```

이후 sd카드를 마운트 시킨 후 커널 이미지와 App 실행 파일을 모두 라즈베리파이로 전송하였습니다. 아래 사진은 라즈베리파이에서 App 실행 결과입니다.



예상보다 jiffies가 크게 나왔지만 반복적으로 실행한 결과 1초마다 거의 100만큼 증가하는 것을 확인하였습니다. 이는 라즈베리파이의 3B의 기본 Hz가 100이기 때문에 그런 것으로 사료됩니다.