به نام خدا

آزمایشگاه سیستم‌های عامل

نام استاد: دکتر حمید بیگی

تابستان ۱۴۰۴

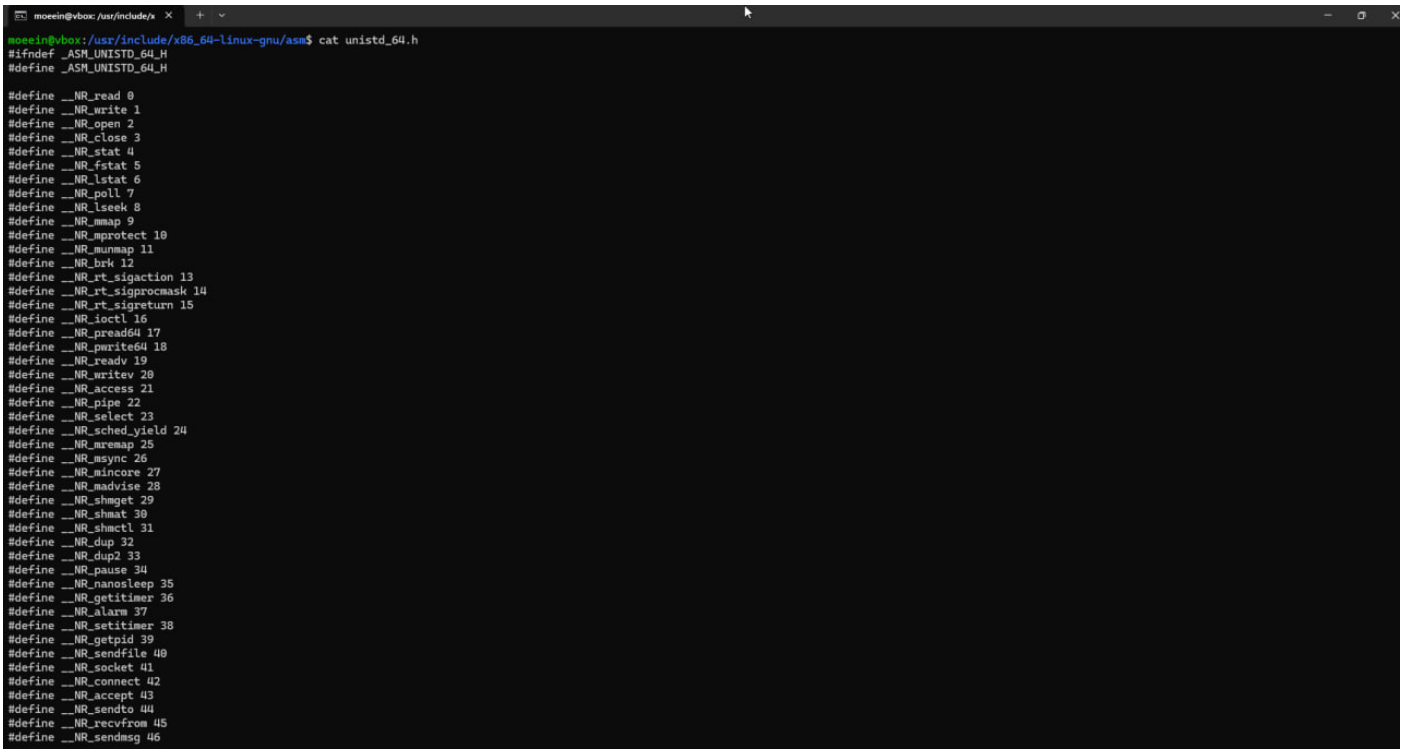
معین آعلی – ۴۰۱۱۰۵۵۶۱

ثمین اکبری – ۴۰۱۱۰۵۵۹۴

ما داخل Debian هستیم پس وارد پوشه زیر می‌شویم:

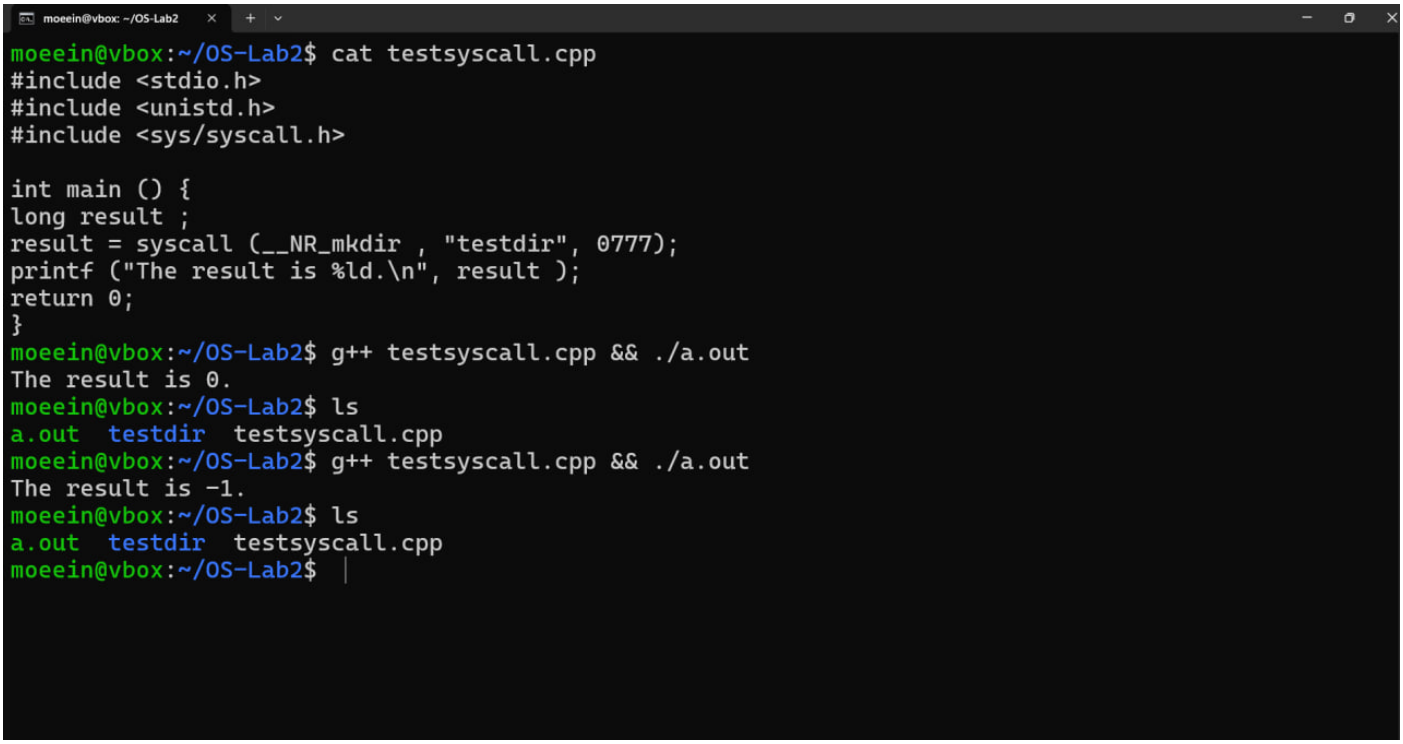
/usr/include/x86_64-linux-gnu/asm

و فایل unistd_64.h را مشاهده می‌کنیم.



```
moeein@vbox:/usr/include/x86_64-linux-gnu/asm$ cat unistd_64.h
#ifndef _ASM_UNISTD_64_H
#define _ASM_UNISTD_64_H

#define __NR_read 0
#define __NR_write 1
#define __NR_open 2
#define __NR_close 3
#define __NR_stat 4
#define __NR_fstat 5
#define __NR_lstat 6
#define __NR_poll 7
#define __NR_lseek 8
#define __NR_mmap 9
#define __NR_mprotect 10
#define __NR_munmap 11
#define __NR_brk 12
#define __NR_rt_sigaction 13
#define __NR_rt_sigprocmask 14
#define __NR_rt_sigreturn 15
#define __NR_ioctl 16
#define __NR_pread64 17
#define __NR_pwrite64 18
#define __NR_readv 19
#define __NR_writev 20
#define __NR_access 21
#define __NR_pipe 22
#define __NR_select 23
#define __NR_sched_yield 24
#define __NR_mremap 25
#define __NR_msync 26
#define __NR_mincore 27
#define __NR_madvise 28
#define __NR_shmget 29
#define __NR_shmat 30
#define __NR_shmctl 31
#define __NR_dup 32
#define __NR_dup2 33
#define __NR_pause 34
#define __NR_nanosleep 35
#define __NR_getitimer 36
#define __NR_alarm 37
#define __NR_setitimer 38
#define __NR_getpid 39
#define __NR_sendfile 40
#define __NR_socket 41
#define __NR_connect 42
#define __NR_accept 43
#define __NR_sendto 44
#define __NR_recvfrom 45
#define __NR_sendmsg 46
```

با استفاده از کد cpp یا c فراخوانی سیستمی mkdir را اجرا می‌کنیم:



```
moeein@vbox:~/OS-Lab2$ cat testsyscall.cpp
#include <stdio.h>
#include <unistd.h>
#include <sys/syscall.h>

int main () {
long result ;
result = syscall (__NR_mkdir , "testdir", 0777);
printf ("The result is %ld.\n", result );
return 0;
}
moeein@vbox:~/OS-Lab2$ g++ testsyscall.cpp && ./a.out
The result is 0.
moeein@vbox:~/OS-Lab2$ ls
a.out  testdir  testsyscall.cpp
moeein@vbox:~/OS-Lab2$ g++ testsyscall.cpp && ./a.out
The result is -1.
moeein@vbox:~/OS-Lab2$ ls
a.out  testdir  testsyscall.cpp
moeein@vbox:~/OS-Lab2$
```

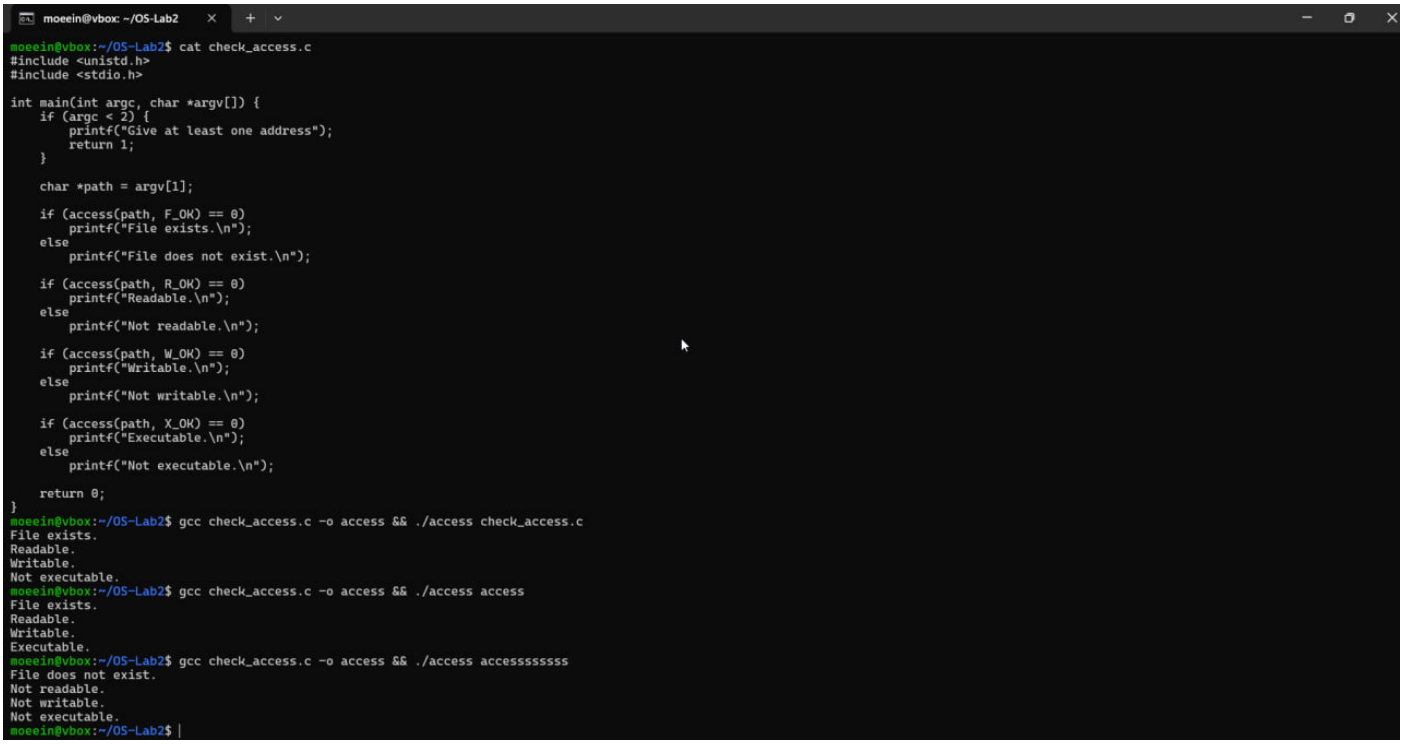
در این کد __NR_mkdir شماره‌ی سیس کال مربوط به mkdir است که در بخش قبل داخل فایل unistd_64.h موجود بود. همچنین 0777 به دسترسی‌های فولدر اشاره دارد.

نحوه استفاده از تابع syscallبه این صورت است که شماره سیس کال را در آرگومان اول به آن می‌دهیم و باقی آرگومان‌ها هم ورودی‌های آن سیس‌کال فرخوانی‌شده هستند.

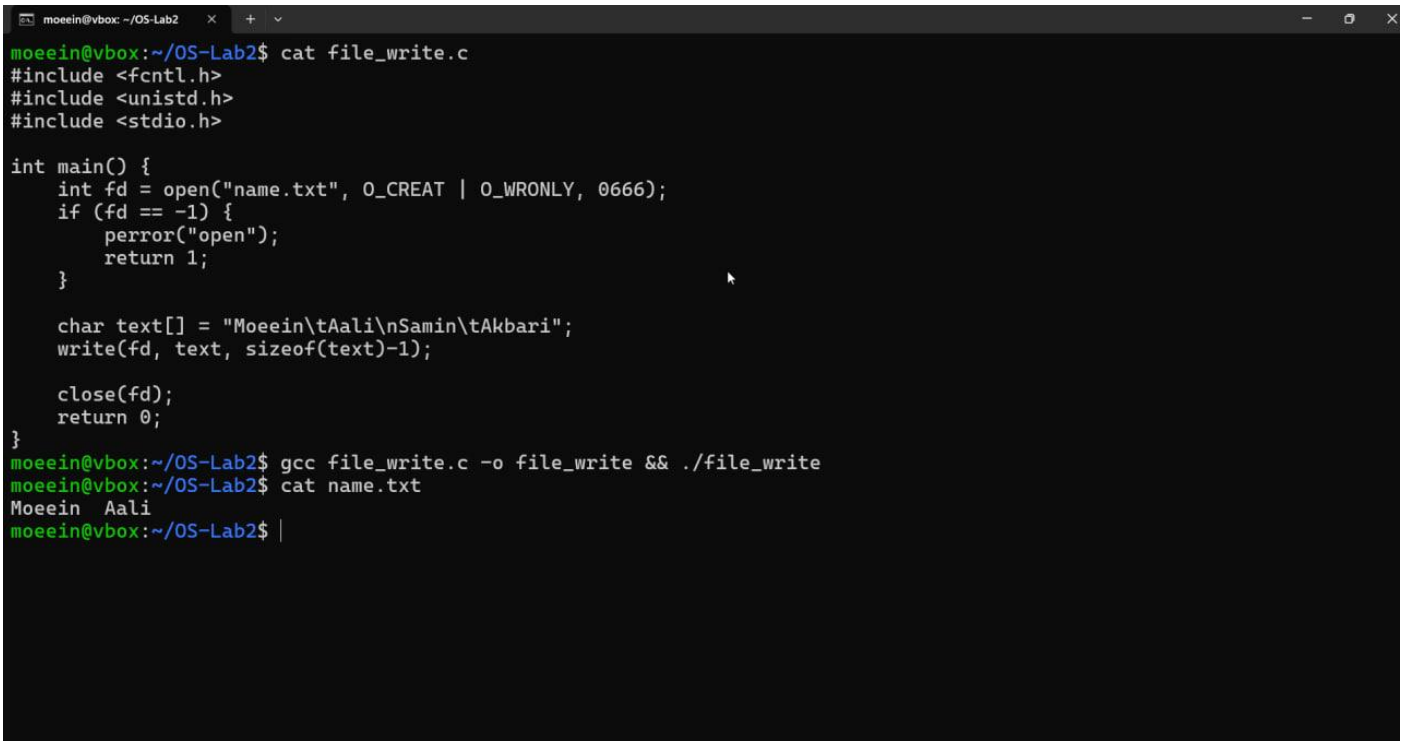
```
moeein@vbox:~/OS-Lab2$ cat testsys2.cpp
#include<stdio.h>
#include<unistd.h>
#include<sys/stat.h>

int main() {
        long result;
        result = mkdir("testdir2", 0777);
        printf("Syscall result : %ld\n", result);
        return 0;
}
moeein@vbox:~/OS-Lab2$ g++ testsys2.cpp && ./a.out
Syscall result : 0
moeein@vbox:~/OS-Lab2$ ls
a.out  testdir  testdir2  testsys2.cpp  testsyscall.cpp
moeein@vbox:~/OS-Lab2$
```

در این کد به جای include کردن از sys/syscall.h از sys/stat.h توابع مورد نظر خود را include می‌کنیم که در اصل wrapperهایی برای سیس‌کال‌های قبلی هستند تا آن‌ها را ساده‌تر اجرا کنیم. پس ما از wrapper مربوط به ساخت پوشه به نام mkdir استفاده می‌کنیم و فولدر مورد نظر را می‌سازیم.

همانند سیس‌کال بخش قبل، اگر فولدر با موفقیت ساخته شود نتیجه ۰ و اگر ساخته نشود نتیجه ۱- دارد.



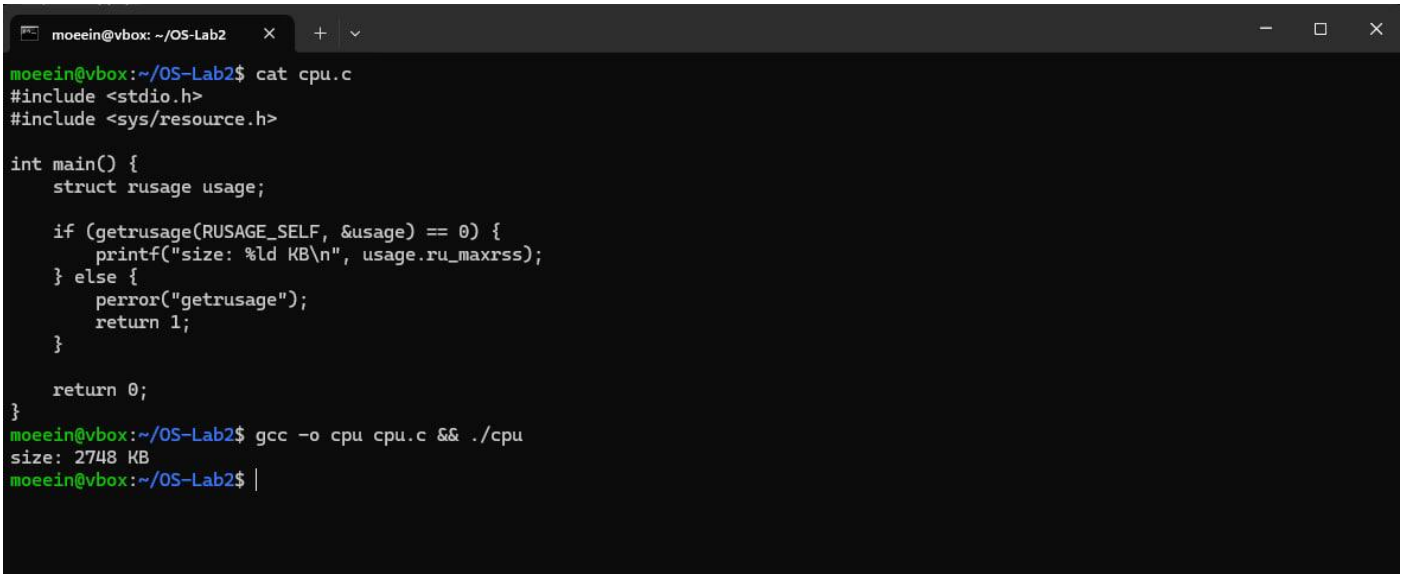
تابع access دارای تعدادی فلگ است. فلگ F_OK چک می‌کند که فایل وجود دارد یا خیر. همینطور فلگ‌های R_OK و W_OK و X_OKبه ترتیب نشان‌دهنده Read و Write و Execute هستند.



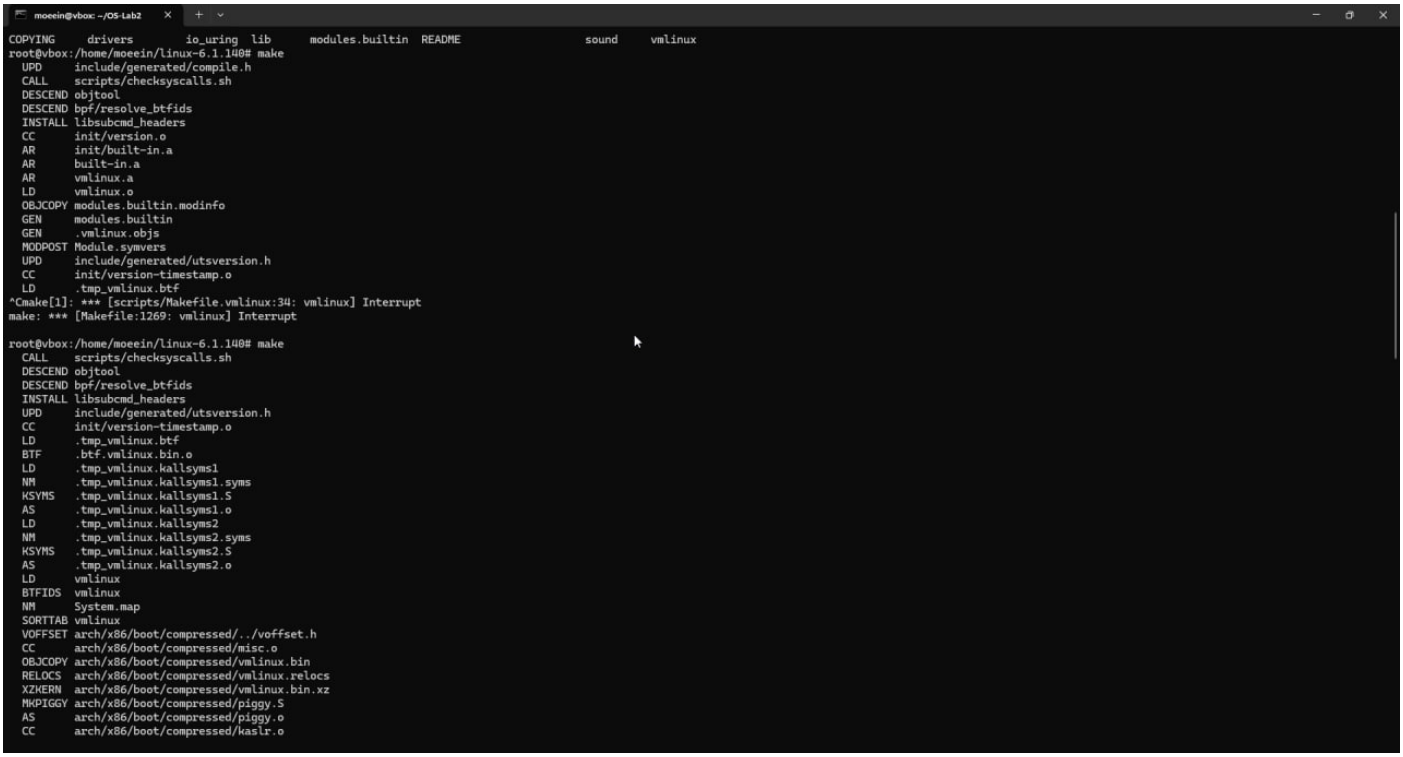
ابتدا fcntl.h که شامل فلگ‌های باز کردن فایل است را include می‌کنیم. سپس با استفاده از تابع open یک فایل با نام name.txt باز می‌کنیم که به این تابع فلگ‌های O_CREAT و O_WRONLY پاس داده شده است. فلگ اول برای ایجاد فایل در صورت عدم وجود و فلگ دوم برای اینکه فایل را برای نوشتن باز کند. در اخر هم سطح دسترسی آن را می‌دهیم.

در صورت موفق بودن عملیات، تابع open یک File Descriptor برمیگرداند. اگر مقدار آن ۱- بود یعنی به ارور خوردیم. در نهایت با تابع write متن مورد نظر را داخل فایل می‌نویسیم. همچنین نیاز است سایز متن را بدهیم. چون در انتهای متن 0\ وجود دارد، از سایز متن ۱ واحد کم می‌کنیم. در نهایت با استفاده از تابع close فایل را می‌بندیم و برنامه به اتمام می‌رسد.

برای استفاده از فراخوانی سیستم sysinfo نیاز است یک پوینتر به struct از نوع sysinfo به آن پاس دهیم (جلوی if) اگر به ارور نخوریم فیلدهایی که داخل sysinfo وجود دارند را می‌توانیم داخل استراکت info مشاهده کنیم.

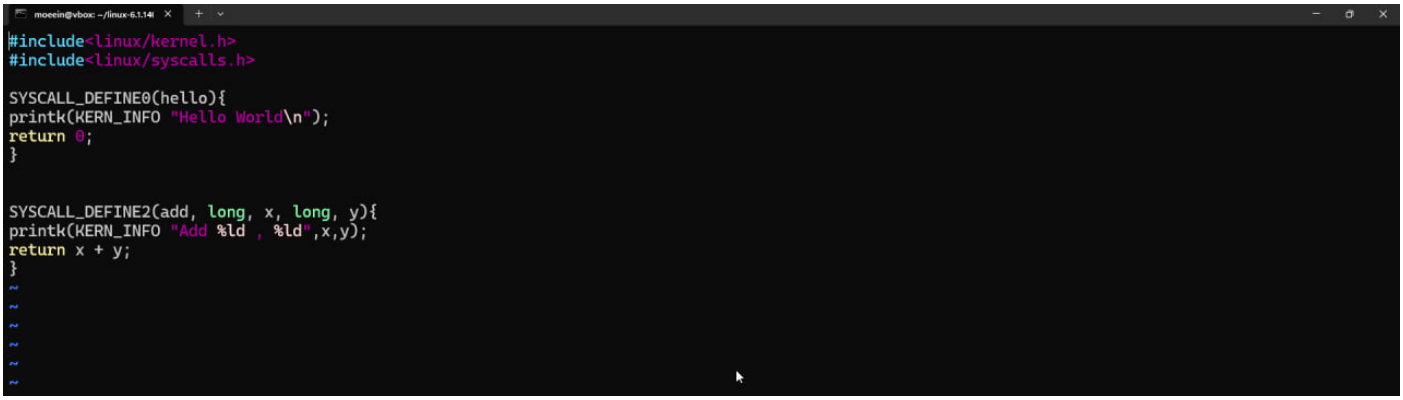


برای استفاده از فراخوانی سیستمی getrusage نیاز است مشابه با قبلی به آن یک پوینتر به استراکت از نوع rusage بدهیم. البته این فراخوانی سیستمی یک فلگ هم می‌گیرد که اینجا ما RUSAGE_SELF را دادیم تا میزان مصرف مناسب پردازه فعلی را به ما بدهد.



قبل از اضافه کردن فراخوانی سیستمی خودمان، یک بار مجدد کرنل را کامپایل کرده.

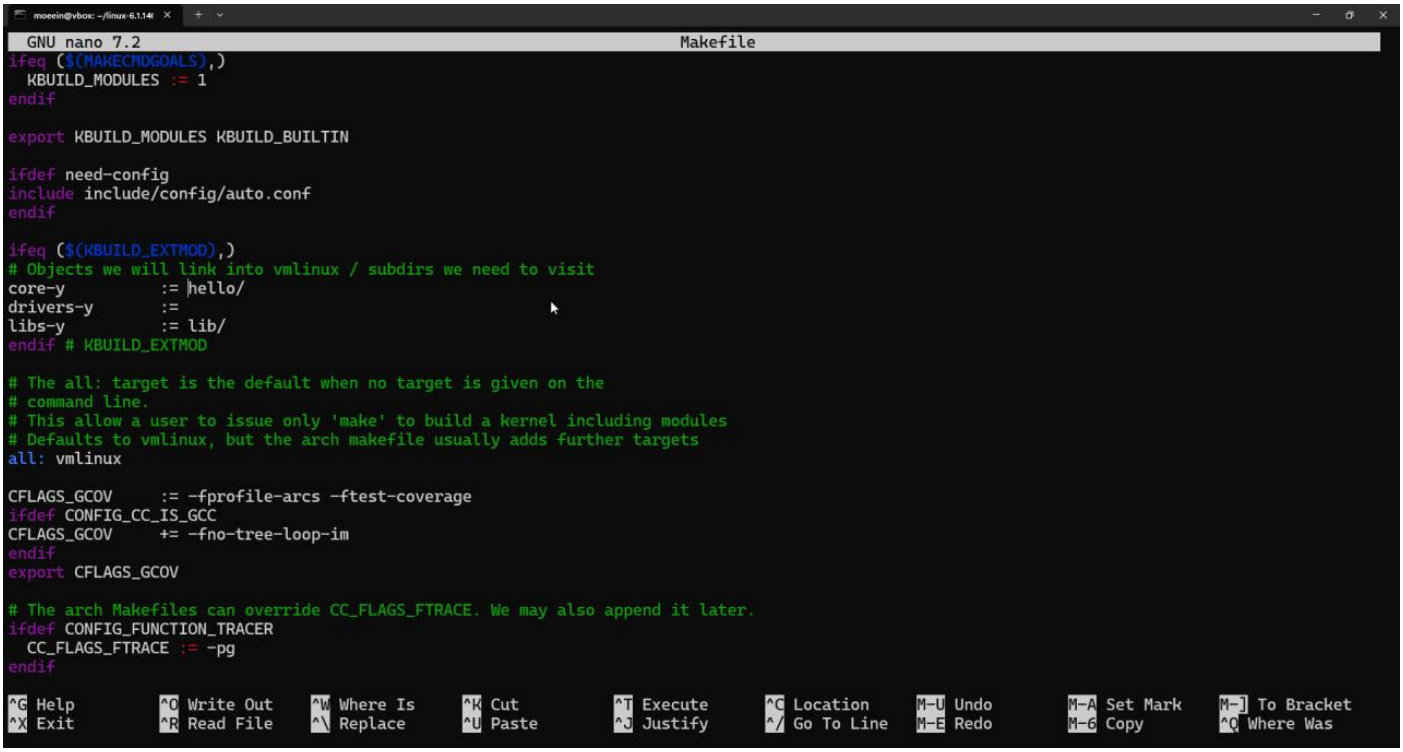
داخل سورس کرنل لینوکس، یک پوشه به نام hello ایجاد کرده و داخل فایل hellosys.cpp فراخوانی‌های سیستمی مورد نظر را اضافه می‌کنیم:



از SYSCALL_DEFINE0 برای تعریف فراخوانی سیستمی با ورودی void و از SYSCALL_DEFINE2 برای فراخوانی سیستمی با ۲ ورودی استفاده می‌کنیم. در آرگومان اول هر دو هم نام فراخوانی سیستمی تعریف شده را وارد می‌کنیم.

سپس داخل همان پوشه Hello d یک Makefile ایجاد کرده و محتوای زیر را در آن قرار می‌دهیم.

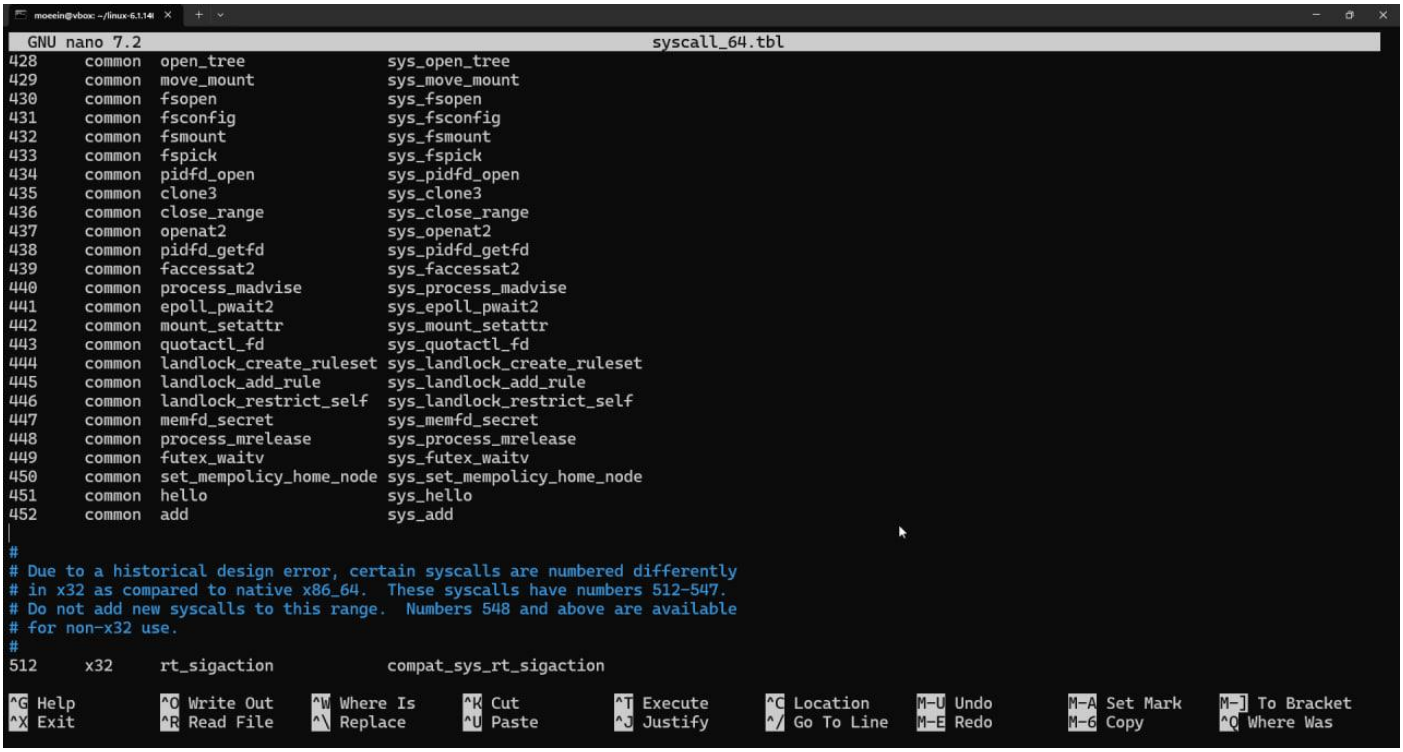
حال باید این را به Makefile اصلی سورس کد هم اضافه کنیم:



حال در گام بعد به فایل جداول فراخوانی‌های سیستمی رفته و ۲ فراخوانی سیستمی جدید را به آن اضافه می‌کنیم.

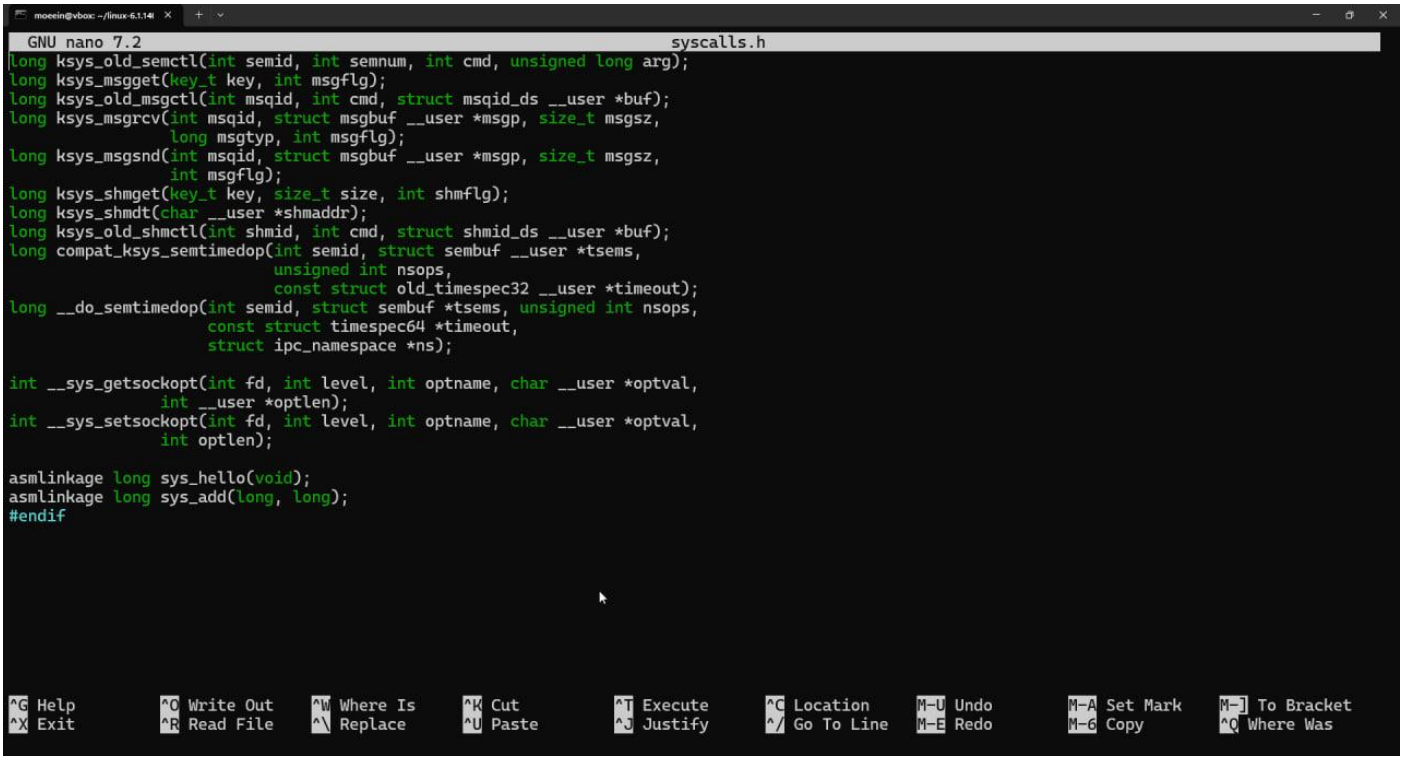
شماره این دو فراخوانی سیستمی ۴۵۱ و ۴۵۲ است.

arch/x86/entry/syscalls/



در نهایت هم ساختار دو فراخوانی سیستمی جدید را به این فایل اضافه می‌کنیم:

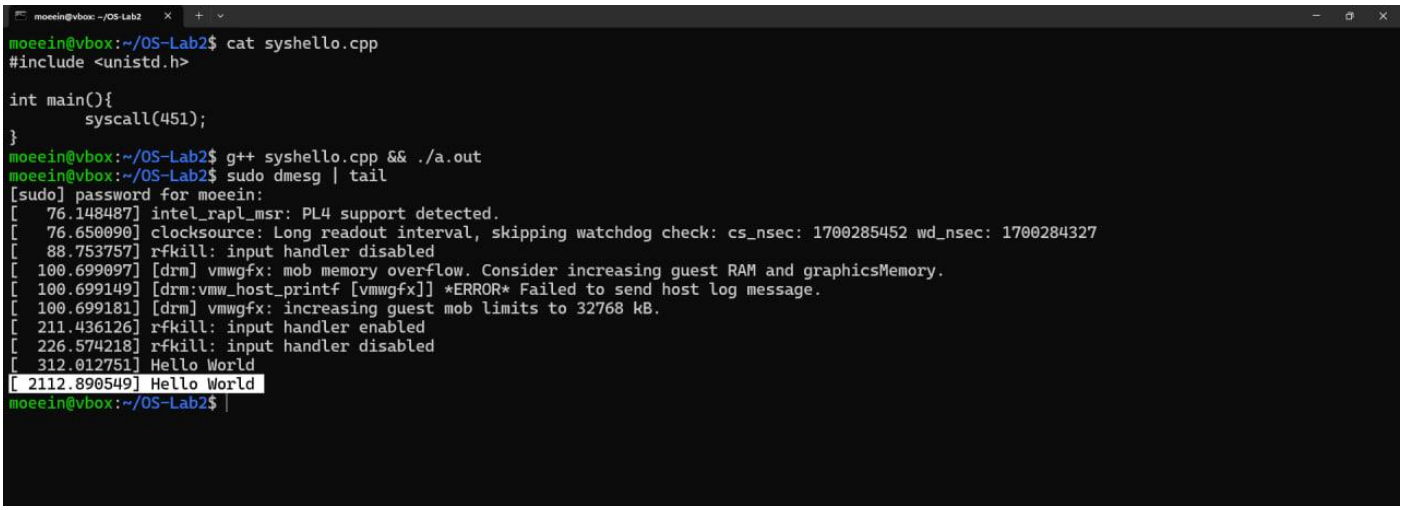
include/linux/Syscall.h



در نهایت هسته را مجدد کامپایل و نصب کرده و grub را اپدیت می‌کنیم. (مراحل آن در آزمایش قبل نشان داده شده است)

حال با هسته‌ی کامپایل شده سیستم را reboot می‌کنیم و با استفاده از کد زیر فراخوانی سیستمی نوشته شده توسط خودمان را اجرا می‌کنیم.

توجه کنید که برای مشاهده پرینت داده شده باید از dmesg استفاده کنیم چون پرینت در سطح کرنل انجام شده (printk)



همچنین برای استفاده از فراخوانی سیستمی add به این صورت عمل می‌کنیم:

توجه: به جهت تنبلی در نوشتن، هر دو فراخوانی سیستمی را داخل فایل hello نوشته و پیاده‌سازی کردیم و هر دو را همزمان به کرنل اضافه کردیم.