

# 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.

```
moeen@vbox: /usr/include/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_shaget 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:

```
moeen@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;
}

moeen@vbox:~/OS-Lab2$ g++ testsyscall.cpp && ./a.out
The result is 0.
moeen@vbox:~/OS-Lab2$ ls
a.out testdir testsyscall.cpp
moeen@vbox:~/OS-Lab2$ g++ testsyscall.cpp && ./a.out
The result is -1.
moeen@vbox:~/OS-Lab2$ ls
a.out testdir testsyscall.cpp
moeen@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 `syscall` به این صورت است که شماره سیس کال را در آرگومان اول به آن می‌دهیم و باقی آرگومان‌ها هم ورودی‌های آن سیس کال فرخوانی شده هستند.

```
moein@vbox: ~/OS-Lab2
moein@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;
}
moein@vbox:~/OS-Lab2$ g++ testsys2.cpp && ./a.out
Syscall result : 0
moein@vbox:~/OS-Lab2$ ls
a.out  testdir  testdir2  testsys2.cpp  testsyscall.cpp
moein@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 `wrapper`هایی برای سیس کال‌های 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 سیس کال بخش قبل، اگر فولدر با موفقیت ساخته شود نتیجه ۰ و اگر ساخته نشود نتیجه ۱- دارد.

```

moeen@vbox: ~/OS-Lab2
moeen@vbox:~/OS-Lab2$ cat check_access.c
#include <unistd.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
    if (argc < 2) {
        printf("Give at least one address");
        return 1;
    }

    char *path = argv[1];

    if (access(path, F_OK) == 0)
        printf("File exists.\n");
    else
        printf("File does not exist.\n");

    if (access(path, R_OK) == 0)
        printf("Readable.\n");
    else
        printf("Not readable.\n");

    if (access(path, W_OK) == 0)
        printf("Writable.\n");
    else
        printf("Not writable.\n");

    if (access(path, X_OK) == 0)
        printf("Executable.\n");
    else
        printf("Not executable.\n");

    return 0;
}
moeen@vbox:~/OS-Lab2$ gcc check_access.c -o access && ./access check_access.c
File exists.
Readable.
Writable.
Not executable.
moeen@vbox:~/OS-Lab2$ gcc check_access.c -o access && ./access access
File exists.
Readable.
Writable.
Executable.
moeen@vbox:~/OS-Lab2$ gcc check_access.c -o access && ./access accessssssss
File does not exist.
Not readable.
Not writable.
Not executable.
moeen@vbox:~/OS-Lab2$

```

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_OK` به ترتیب نشان‌دهنده `Read` و `Write` و `Execute` هستند.

```

moeen@vbox: ~/OS-Lab2
moeen@vbox:~/OS-Lab2$ cat file_write.c
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

int main() {
    int fd = open("name.txt", O_CREAT | O_WRONLY, 0666);
    if (fd == -1) {
        perror("open");
        return 1;
    }

    char text[] = "Moeen\tAali\nSamin\tAkbari";
    write(fd, text, sizeof(text)-1);

    close(fd);
    return 0;
}
moeen@vbox:~/OS-Lab2$ gcc file_write.c -o file_write && ./file_write
moeen@vbox:~/OS-Lab2$ cat name.txt
Moeen Aali
moeen@vbox:~/OS-Lab2$

```

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 آخر 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 سائز متن را بدهیم. چون در انتهای متن 0\ وجود دارد، از سائز متن ۱ واحد کم می‌کنیم. در نهایت با استفاده از تابع `close` فایل را می‌بندیم و برنامه به اتمام می‌رسد.

```
moeen@vbox: ~/OS-Lab2
moeen@vbox:~/OS-Lab2$ cat info.c
#include <stdio.h>
#include <sys/sysinfo.h>

int main() {
    struct sysinfo info;

    if (sysinfo(&info) == 0) {
        long total_ram = info.totalram / (1024 * 1024);
        long free_ram = info.freeram / (1024 * 1024);

        printf("Total RAM: %ld MB\n", total_ram);
        printf("Free RAM: %ld MB", free_ram);
    } else {
        perror("sysinfo");
        return 1;
    }
    return 0;
}

moeen@vbox:~/OS-Lab2$ gcc -o info info.c && ./info
Total RAM: 13972 MB
Free RAM: 12767 MBmoeen@vbox:~/OS-Lab2$ |
```

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.

```
moeen@vbox: ~/OS-Lab2
moeen@vbox:~/OS-Lab2$ cat cpu.c
#include <stdio.h>
#include <sys/resource.h>

int main() {
    struct rusage usage;

    if (getrusage(RUSAGE_SELF, &usage) == 0) {
        printf("size: %ld KB\n", usage.ru_maxrss);
    } else {
        perror("getrusage");
        return 1;
    }

    return 0;
}

moeen@vbox:~/OS-Lab2$ gcc -o cpu cpu.c && ./cpu
size: 2748 KB
moeen@vbox:~/OS-Lab2$ |
```

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 پردازش فعلی را به ما بدهد.

```
moein@vbox: ~/OS-tab2 x + v
COPYING drivers io_uring lib modules.builtin README sound vmlinux
root@vbox:~/home/moein/linux-6.1.140# make
CALL scripts/checksyscalls.sh
DESCEND objtool
DESCEND bpf/resolve_btfids
INSTALL libsubcmd_headers
CC init/version.o
AR init/built-in.a
AR built-in.a
AR vmlinux.a
LD vmlinux.o
OBJCOPY modules.builtin.modinfo
GEN modules.builtin
GEN .vmlinux.objs
MODPOST Module.symvers
UPD include/generated/utsversion.h
CC init/version-timestamp.o
LD .tmp_vmlinux.btf
^Cmake[1]: *** [scripts/Makefile.vmlinux:34: vmlinux] Interrupt
make: *** [Makefile:1269: vmlinux] Interrupt

root@vbox:~/home/moein/linux-6.1.140# make
CALL scripts/checksyscalls.sh
DESCEND objtool
DESCEND bpf/resolve_btfids
INSTALL libsubcmd_headers
UPD include/generated/utsversion.h
CC init/version-timestamp.o
LD .tmp_vmlinux.btf
BTF .btf.vmlinux.bin.o
LD .tmp_vmlinux.kallsyms1
NM .tmp_vmlinux.kallsyms1.syms
KSYMS .tmp_vmlinux.kallsyms1.5
AS .tmp_vmlinux.kallsyms1.o
LD .tmp_vmlinux.kallsyms2
NM .tmp_vmlinux.kallsyms2.syms
KSYMS .tmp_vmlinux.kallsyms2.5
AS .tmp_vmlinux.kallsyms2.o
LD vmlinux
BTFIDS vmlinux
NM System.map
SORTTAB vmlinux
VOFFSET arch/x86/boot/compressed/./voffset.h
CC arch/x86/boot/compressed/misc.o
OBJCOPY arch/x86/boot/compressed/vmlinux.bin
RELOCS arch/x86/boot/compressed/vmlinux.relocs
XZKERN arch/x86/boot/compressed/vmlinux.bin.xz
MKPIGGY arch/x86/boot/compressed/piggy.5
AS arch/x86/boot/compressed/piggy.o
CC arch/x86/boot/compressed/kaslr.o
```

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:

```
moein@vbox: ~/linux-6.1.140 x + v
#include<linux/kernel.h>
#include<linux/syscalls.h>

SYSCALL_DEFINE0(hello){
    printk(KERN_INFO "Hello World\n");
    return 0;
}

SYSCALL_DEFINE2(add, long, x, long, y){
    printk(KERN_INFO "Add %ld , %ld",x,y);
    return x + y;
}
~
~
~
~
~
```

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.

```
moein@vbox: ~/linux-6.1.140 x + v
moein@vbox:~/linux-6.1.140/hello$ cat Makefile
obj-y := hellosys.o

moein@vbox:~/linux-6.1.140/hello$ |
```

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:

```
GNU nano 7.2 Makefile
ifeq ($(MAKECMDGOALS),)
KBUILD_MODULES := 1
endif

export KBUILD_MODULES KBUILD_BUILTIN

ifdef need-config
include include/config/auto.conf
endif

ifeq ($(KBUILD_EXTMOD),)
# Objects we will link into vmlinux / subdirs we need to visit
core-y      := hello/
drivers-y   :=
libs-y      := lib/
endif # KBUILD_EXTMOD

# The all: target is the default when no target is given on the
# command line.
# This allow a user to issue only 'make' to build a kernel including modules
# Defaults to vmlinux, but the arch makefile usually adds further targets
all: vmlinux

CFLAGS_GCOV      := -fprofile-arcs -ftest-coverage
ifdef CONFIG_CC_IS_GCC
CFLAGS_GCOV      += -fno-tree-loop-imb
endif
export CFLAGS_GCOV

# The arch Makefiles can override CC_FLAGS_FTRACE. We may also append it later.
ifdef CONFIG_FUNCTION_TRACER
CC_FLAGS_FTRACE := -pg
endif

^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo      M-A Set Mark   M-] To Bracket
^X Exit      ^R Read File  ^_ Replace    ^U Paste      ^J Justify    ^_/ Go To Line M-E Redo      M-6 Copy      ^_ Where Was
```

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/

```
GNU nano 7.2 syscall_64.tbl
428 common open_tree      sys_open_tree
429 common move_mount     sys_move_mount
430 common fsopen          sys_fsopen
431 common fsconfig        sys_fsconfig
432 common fsmount         sys_fsmount
433 common fspick          sys_fspick
434 common pidfd_open      sys_pidfd_open
435 common clone3          sys_clone3
436 common close_range     sys_close_range
437 common openat2         sys_openat2
438 common pidfd_getfd     sys_pidfd_getfd
439 common faccessat2      sys_faccessat2
440 common process_madvise  sys_process_madvise
441 common epoll_pwait2    sys_epoll_pwait2
442 common mount_setattr   sys_mount_setattr
443 common quotactl_fd     sys_quotactl_fd
444 common landlock_create_ruleset sys_landlock_create_ruleset
445 common landlock_add_rule sys_landlock_add_rule
446 common landlock_restrict_self sys_landlock_restrict_self
447 common memfd_secret     sys_memfd_secret
448 common process_mrelease sys_process_mrelease
449 common futex_waitv     sys_futex_waitv
450 common set_mempolicy_home_node sys_set_mempolicy_home_node
451 common hello          sys_hello
452 common add             sys_add

#
# Due to a historical design error, certain syscalls are numbered differently
# in x32 as compared to native x86_64. These syscalls have numbers 512-547.
# Do not add new syscalls to this range. Numbers 548 and above are available
# for non-x32 use.
#
512 x32 rt_sigaction      compat_sys_rt_sigaction

^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo      M-A Set Mark   M-] To Bracket
^X Exit      ^R Read File  ^_ Replace    ^U Paste      ^J Justify    ^_/ Go To Line M-E Redo      M-6 Copy      ^_ Where Was
```



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

include/linux/Syscall.h

```
GNU nano 7.2 syscalls.h
long ksys_old_semctl(int semid, int semnum, int cmd, unsigned long arg);
long ksys_msgget(key_t key, int msgflg);
long ksys_old_msgctl(int msqid, int cmd, struct msqid_ds __user *buf);
long ksys_msgrcv(int msqid, struct msgbuf __user *msgp, size_t msgsz,
                long msgtyp, int msgflg);
long ksys_msgsnd(int msqid, struct msgbuf __user *msgp, size_t msgsz,
                int msgflg);
long ksys_shmget(key_t key, size_t size, int shmflg);
long ksys_shmctl(char __user *shmaddr);
long ksys_old_shmctl(int shmid, int cmd, struct shmid_ds __user *buf);
long compat_ksys_semtimedop(int semid, struct sembuf __user *tsems,
                            unsigned int nsops,
                            const struct old_timespec32 __user *timeout);
long __do_semtimedop(int semid, struct sembuf *tsems, unsigned int nsops,
                    const struct timespec64 *timeout,
                    struct ipc_namespace *ns);

int __sys_getsockopt(int fd, int level, int optname, char __user *optval,
                    int __user *optlen);
int __sys_setsockopt(int fd, int level, int optname, char __user *optval,
                    int optlen);

asmlinkage long sys_hello(void);
asmlinkage long sys_add(long, long);
#endif

^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo     M-A Set Mark  M-J To Bracket
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^_ Go To Line M-E Redo     M-G Copy      ^Q Where Was
```

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)

```
moeen@vbox: ~/OS-Lab2$ cat syshello.cpp
#include <unistd.h>

int main(){
    syscall(451);
}

moeen@vbox: ~/OS-Lab2$ g++ syshello.cpp && ./a.out
moeen@vbox: ~/OS-Lab2$ sudo dmesg | tail
[sudo] password for moeen:
[ 76.148487] intel_rapl_msr: PL4 support detected.
[ 76.650090] clocksource: Long readout interval, skipping watchdog check: cs_nsec: 1700285452 wd_nsec: 1700284327
[ 88.753757] rfkill: input handler disabled
[ 100.699097] [drm] vmwgfx: mob memory overflow. Consider increasing guest RAM and graphicsMemory.
[ 100.699149] [drm:vmw_host_printf [vmwgfx]] *ERROR* Failed to send host log message.
[ 100.699181] [drm] vmwgfx: increasing guest mob limits to 32768 kB.
[ 211.436126] rfkill: input handler enabled
[ 226.574218] rfkill: input handler disabled
[ 312.012751] Hello World
[ 2112.890549] Hello World
moeen@vbox: ~/OS-Lab2$
```



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

```
moeen@vbox: ~/OS-Lab2$ cat sysadd.cpp
#include <unistd.h>
#include <stdio>

int main(){
    long res = syscall(452,10,20);
    printf("10+20=%ld",res);
}
moeen@vbox:~/OS-Lab2$ g++ sysadd.cpp && ./a.out
10+20=30moeen@vbox:~/OS-Lab2$
```

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.