



دانشگاه صنعتی شریف

دانشکده مهندسی کامپیوتر

گزارش کار آزمایشگاه آزمایشگاه سیستم‌های عامل

گزارش آزمایش شماره ۲
(آشنایی با فراخوانی‌های سیستمی)

۲۰

شماره گروه:

ارشیا یوسف‌نیا (۴۰۱۱۰۴۱۵)

گروه:

محمد عارف زارع زاده (۴۰۱۱۰۶۰۱۷)

استاد درس:

دکتر بیگی

تاریخ:

تابستان ۱۴۰۴

فهرست مطالب

۱	۱	شرح آزمایش
۱	۱,۱	مشاهده فراخوانی‌های سیستمی تعریف شده
۲	۱,۲	اجرای یک فراخوانی سیستمی
۳	۱,۳	اجرای ساده‌تر فراخوانی سیستمی
۴	۱,۴	آشنایی با چند فراخوانی سیستمی پرکاربرد
۱۰	۱,۵	اضافه کردن یک فراخوانی سیستمی به سیستم عامل

لیست تصاویر

۱	خواندن فایل unistd_64.h	۱
۱	محتویات فایل unistd_64.h	۲
۲	کد ساخت پوشه با استفاده از فراخوانی سیستمی	۳
۲	ساخت پوشه با استفاده از فراخوانی سیستمی	۴
۴	استفاده از تابع mkdir به جای syscall	۵
۵	کد برنامه‌ی بررسی فایل و سطوح دسترسی برای پردازه‌ی اجرا شده	۶
۵	اجرای کد بالا	۷
۶	کد ساخت و نوشتن اسم در فایل	۸
۷	اجرای کد بالا	۹
۸	کد برنامه‌ی چاپ اطلاعات رم	۱۰
۸	اجرای برنامه‌ی چاپ اطلاعات رم	۱۱
۹	کد برنامه‌ی چاپ کننده‌ی حافظه‌ی مصرفی	۱۲
۹	اجرای برنامه‌ی چاپ کننده‌ی حافظه‌ی مصرفی	۱۳
۱۰	ورود به پوشه‌ی کد منبع هسته‌ی سیستم عامل	۱۴
۱۱	درستی کامپایل هسته	۱۵
۱۱	نصب هسته‌ی جدید	۱۶
۱۲	reboot کردن سیستم	۱۷
۱۲	ساخت پوشه‌ی hello و فایل hello.c	۱۸
۱۳	محتویات فایل hello.c	۱۹
۱۳	محتویات Makefile موجود در hello	۲۰
۱۴	محتویات Kbuild موجود در ریشه‌ی کد منبع	۲۱
۱۴	اضافه کردن تابع فراخوانی سیستمی ساخته شده به لیست فراخوانی‌های سیستمی	۲۲
۱۵	خط اضافه شده به فایل syscalls.h	۲۳
۱۵	کامپایل و نصب مجدد هسته	۲۴
۱۶	استفاده از فراخوانی سیستمی hello در کد یک برنامه	۲۵
۱۷	اضافه کردن پوشه‌ی adder	۲۶
۱۷	کد فراخوانی سیستمی adder	۲۷
۱۸	اضافه کردن adder به Kbuild	۲۸
۱۸	اضافه کردن adder به لیست فراخوانی‌های سیستم	۲۹
۱۹	اضافه کردن adder به syscalls.h	۳۰
۱۹	کامپایل مجدد هسته و نصب ماژول‌ها	۳۱
۲۰	نصب مجدد هسته	۳۲
۲۰	نوشتن برنامه‌ای که از فراخوانی سیستمی adder استفاده می‌کند	۳۳
۲۱	خروجی برنامه‌ی بالا	۳۴

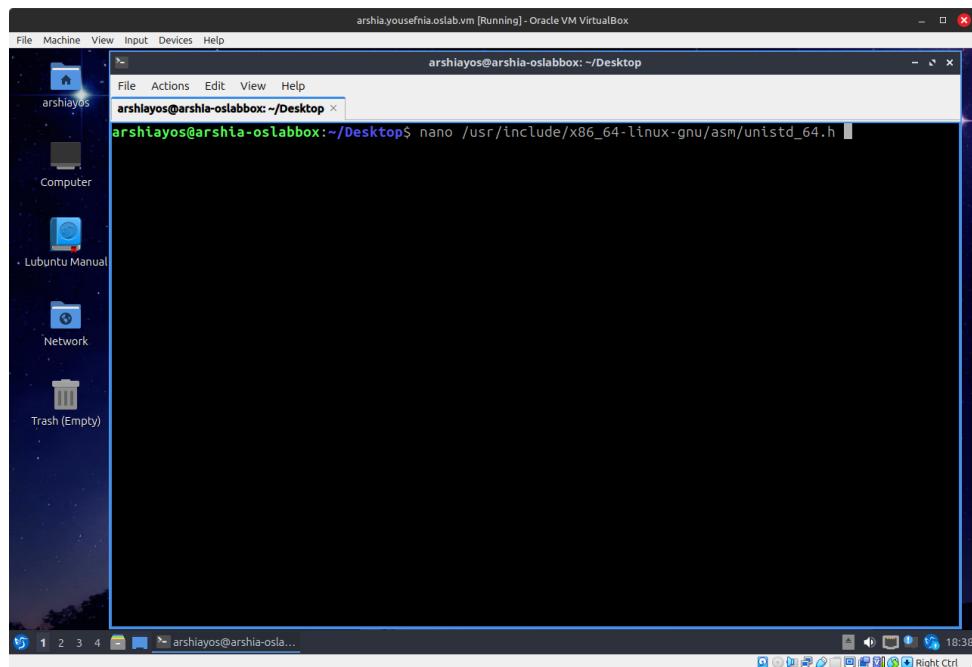
لیست جداول

۴	access Mode های مختلف برای فراخوانی سیستمی	۱
۶	open فلگ های فراخوانی سیستمی	۲

۱ شرح آزمایش

۱.۱ مشاهده فراخوانی‌های سیستمی تعریف شده

مانند شکل ۱ فایل گفته شده را باز کرده و محتویات آن را در شکل ۲ می‌توان مشاهده کرد. همانطور که می‌توان مشاهده کرد، به ازای هر فراخوانی سیستم، یک مacro به فرمت [NR_[syscall name]] وجود دارد تا نیاز نباشد هنگام برنامه‌نویسی با آنها، عدد هر فراخوانی سیستم را حفظ کنیم.



شکل ۱: خواندن فایل 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_readyv 19
#define __NR_writenv 20
#define __NR_access 21
#define __NR_pipe 22

[ File '/usr/include/x86_64-linux-gnu/asm/unistd_64.h' is unwriteable ]
```

شکل ۲: محتویات فایل unistd_64.h

۱،۲ اجرای یک فراخوانی سیستمی

۱. همانطور که از شکل ۳ می‌توان دید، فایل را ساخته و کد گفته شده را در آن می‌گذاریم.

```
arshiyos@arshia-oslabbox:~/Desktop$ nano /usr/include/x86_64-linux-gnu/asm/unistd_64.h
arshiyos@arshia-oslabbox:~/Desktop$ cd ~
arshiyos@arshia-oslabbox:~$ touch testsyscall.cpp
arshiyos@arshia-oslabbox:~$ 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;
}
arshiyos@arshia-oslabbox:~$ 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;
}
```

شکل ۳: کد ساخت پوشه با استفاده از فراخوانی سیستمی

۲. کد را در شکل ۳ می‌توان دید.

۳. خروجی حاصل از اجرای این کد را در شکل زیر می‌توان مشاهده کرد.

```
arshiyos@arshia-oslabbox:~$ gcc testsyscall.cpp
arshiyos@arshia-oslabbox:~$ ls
a.out Desktop Documents Downloads Music Pictures Public Templates testsyscall.cpp Videos
arshiyos@arshia-oslabbox:~$ ./a.out
The result is 0.
arshiyos@arshia-oslabbox:~$ ls -a
. .bash_logout Desktop Music .sudo_as_admin_successful Videos
.. .bashrc Documents Pictures Templates .xsession-errors
a.out .cache Downloads .profile testdir
.bash_history .config .local Public testsyscall.cpp
arshiyos@arshia-oslabbox:~$
```

شکل ۴: ساخت پوشه با استفاده از فراخوانی سیستمی

۴. این کد ابتدا با دستور `syscall` و با کمک شماره‌ی فراخوانی سیستمی `mkdir` آن را اجرا می‌کند. تابع `syscall` در آرگومان اول خود شماره‌ی فراخوانی سیستمی و در آرگومان‌های بعدی خود، آرگومان‌هایی که به آن فراخوانی سیستمی داده می‌شود را می‌دهیم.

آرگومان `testdir` نام پوشه‌ای که ساخته می‌شود است، و عدد ۰۷۷۷ که در مبنای ۸ است، سطوح دسترسی را مشخص می‌کند، که در این نمونه، همه دسترسی خواندن و نوشتمن و اجرا کردن را دارند.

سپس خروجی `syscall` در متغیر `result` ریخته می‌شود. به طور کلی هنگام فراخوانی‌های سیستمی، خروجی اگر ۰ باشد یعنی مشکلی نبوده و اگر ۱- باشد، یعنی به خطأ خورده است.

درنهایت این خروجی چاپ شده و برنامه تمام می‌شود.

درستی کارکرد آن را نیز در شکل بالا می‌توان دید.

پاسخ به تمرین‌های این بخش:

(۱) نقش `NR_mkdir` که یک ماکرو است، دادن شماره‌ی فراخوانی سیستمی `mkdir` به تابع `syscall` است. آن را در شکل ۲ نیز می‌توان مشاهده کرد.

(ب) همانطور که در بالا نیز توضیح داده شد، آرگومان اول تابع `syscall` شماره‌ی فراخوانی سیستمی است، و آرگومان‌های بعدی، ورودی‌ها یا آرگومان‌های مربوط به آن فراخوانی سیستم (برای اجرا کردن) است.

خروجی تابع `syscall` هم عموماً ۰ یا ۱- است. اگر ۰ باشد یعنی فراخوانی سیستمی بدون هیچ مشکلی اجرا شده و عملیات آن موفق بوده، و اگر ۱- باشد یعنی فراخوانی سیستمی به خطأ خورده، و همچنین `errno` با توجه به شماره‌ی خطای خورده شده را می‌گیرد.

۱.۳ اجرای ساده‌تر فراخوانی سیستمی

کد مربوط به `testsySCALL2.cpp` را در شکل زیر می‌توانید مشاهده کنید. این کد با کد بخش قبل تقریباً هیچ تفاوتی ندارد. تنها تفاوت آن این است که به جای استفاده از تابع `syscall` از تابع `mkdir` استفاده شده است. ورودی‌های آن هم به جز ورودی اول (که به وضوح دیگر لازم نیست) مانند ورودی‌های تابع `syscall` است.

```

arshayos@arshia-oslabbox:~$ 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;
}
arshayos@arshia-oslabbox:~$ touch testsyscall2.cpp & cat > testsyscall2.cpp
[1] 1245
#include <stdio.h>
#include <sys/stat.h>
int main() {
    int result;
    result = mkdir("testdir", 0777);
    printf("The result is %d.\n", result);
    return 0;
}
^C[1]+ Done          touch testsyscall2.cpp
arshayos@arshia-oslabbox:~$ rmdir testdir/
arshayos@arshia-oslabbox:~$ gcc testsyscall2.cpp
arshayos@arshia-oslabbox:~$ ./a.out
The result is 0.
arshayos@arshia-oslabbox:~$ ls
a.out  Documents  Music   Public    testdir  testsyscall.cpp  Videos
Desktop  Downloads  Pictures  Templates  testsyscall2.cpp
arshayos@arshia-oslabbox:~$ 

```

شکل ۵: استفاده از تابع mkdir به جای syscall

۱.۴ آشنایی با چند فراخوانی سیستمی پرکاربرد

- فراخوانی سیستمی access دو آرگومان ورودی دارد: اولی نام/آدرس فایل و دومی Mode است. مدهای مختلف آن را در جدول زیر می‌توانید مشاهده کنید [۱]. برای اینکه چند مد مختلف با هم تست شوند، می‌توان مدهای مورد نظر را با هم or محسوباتی (نه منطقی) کرد.

توضیحات	Mode
بررسی اینکه فایل وجود دارد	F_OK
بررسی اینکه فایل قابل خواندن است	R_OK
بررسی اینکه فایل قابل نوشتن است	W_OK
بررسی اینکه فایل قابل اجرا کردن است.	X_OK

جدول ۱: های مختلف برای فراخوانی سیستمی access

طبق این توضیحات، در شکل زیر می‌توانید کد مورد نظر را مشاهده کنید. در این کد ابتدا چک می‌شود که تعداد آرگومان‌ها درست باشد، سپس به ترتیب وجود داشتن، دسترسی خواندن، و دسترسی نوشتن چک می‌شود.

```

arshiyos@arshia-oslabbox:~/syscalls
#include <unistd.h>
int main(int argc, char *argv[]) {
    //check argument
    if (argc == 1) {
        printf("input filename as argument.\n");
        return 1;
    }
    const char *filename = argv[1];
    //access
    if (access(filename, F_OK) == 0) {
        printf("%s exists.\n", filename);
    } else {
        printf("%s doesn't exist.\n", filename);
        return 0;
    }
    //read access
    if (access(filename, R_OK) == 0) {
        printf("has read permission for %s\n", filename);
    } else {
        printf("doesn't have read permission for %s.\n", filename);
    }
    //write access
    if (access(filename, W_OK) == 0) {
        printf("has write permission for %s\n", filename);
    } else {
        printf("doesn't have write permission for %s.\n", filename);
    }
}
"access.cpp" 30L, 720B
29,3-10 Bot
arshiyos@arshia-osla...

```

شکل ۶: کد برنامه‌ی بررسی فایل و سطوح دسترسی برای پردازه‌ی اجرا شده

```

arshiyos@arshia-oslabbox:~/syscalls
arshiyos@arshia-oslabbox:~/syscalls$ vim access.cpp
arshiyos@arshia-oslabbox:~/syscalls$ ls
access.cpp
arshiyos@arshia-oslabbox:~/syscalls$ gcc access.cpp
arshiyos@arshia-oslabbox:~/syscalls$ ./a.out access.cpp
access.cpp exists.
has read permission for access.cpp
has write permission for access.cpp
arshiyos@arshia-oslabbox:~/syscalls$ ./a.out random.txt
random.txt doesn't exist.
arshiyos@arshia-oslabbox:~/syscalls$ touch file.txt
arshiyos@arshia-oslabbox:~/syscalls$ ./a.out file.txt
file.txt exists.
has read permission for file.txt
has write permission for file.txt
arshiyos@arshia-oslabbox:~/syscalls$ 

```

شکل ۷: اجرای کد بالا

- فراخوانی سیستمی open سه آرگومان را به عنوان ورودی می‌گیرد. اولی نام فایلی که می‌خواهیم باز کنیم است.
- دومی فلگ‌های مربوط به آن است. سومی هم سطوح دسترسی، در صورت استفاده از فلگ O_CREAT است. انواع فلگ‌ها را در تصویر زیر می‌توانید مشاهده کنید [۲]. برای استفاده از چندین فلگ می‌توان آنها را با هم or کرد.

خروجی open در صورت موفق بودن عملیات عددی مثبت که بیانگر file descriptor است خواهد بود، و در غیر این صورت -1 خواهد بود.

توضیحات	Flag
برای باز کردن و فقط خواندن	O_RDONLY
برای باز کردن و فقط نوشتن	O_WRONLY
برای باز کردن و خواندن و نوشتن	O_RDWR
برای ساخت فایل در صورت وجود نداشتن (نیاز به آرگومان سطح دسترسی دارد)	O_CREAT
همراه با O_CREAT در صورت وجود داشتن فایل از قبل، خطاب خورد	O_EXCL
با هر بار نوشتن، داده به انتهای فایل اضافه شود	O_APPEND
در صورت وجود فایل، محتويات آن خالی شود	O_TRUNC
باز کردن در حالت non-blocking	O_NONBLOCK
نوشتن به صورت هماهنگ (سنکرون) باشد.	O_SYNC

جدول ۲ : فلگ‌های فراخوانی سیستمی open

فراخوانی سیستمی write سه آرگومان دارد. اولی file descriptor که همان خروجی open است، دومی یک پوینتر به چیزی که می‌خواهیم بنویسیم، و سومی تعداد بایت‌هایی که می‌خواهیم بنویسیم است [۳]. خروجی آن در صورت موفق بودن، تعداد بایت‌هایی که نوشته شده‌اند است، و در صورت خطاب خوردن ۱- است.

فراخوانی سیستمی close فقط یک آرگومان ورودی دارد که همان file descriptor است. خروجی آن هم در صورت موفق بودن ۰ و در غیر این صورت ۱- است [۴].

کد مربوط به خواسته‌ی این بخش را در شکل زیر می‌توانید ببینید. این کد ابتدا با open با کمک فلگ‌های مناسب فایل را در صورت وجود نداشتن ساخته و در صورت وجود داشتن آن را خالی کرده و آن را در حالت فقط نوشتن باز می‌کند. سپس با دستور write محتويات خواسته شده را (که ۱۷ بایت است) را در آن فایل می‌نویسد. درنهایت آن را با دستور close می‌بندد. در صورت خطاب داشتن در هر یک از مراحل، خطای مناسبی را چاپ کرده و خارج می‌شوند.

```

arshiyos@arshia-oslabbox: ~/syscalls
File Actions Edit View Help
arshiyos@arshia-oslabbox: ~/syscalls
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>

int main() {
    const char *name = "Arshia Yousefnia\n";
    int fd;
    ssize_t written_bytes;

    fd = open("oslab2.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd == -1) {
        printf("Error opening file.\n");
        return 1;
    }

    written_bytes = write(fd, name, 17);
    if (written_bytes == -1) {
        printf("Error writing to file.\n");
        close(fd);
        return 1;
    }

    if (close(fd) == -1) {
        printf("Error closing file.\n");
        return 1;
    }

    printf("wrote content to file.\n");
    return 0;
}
'createwrite.cpp' 29L, 534B
16,37-44 All

```

شکل ۸: کد ساخت و نوشتن اسم در فایل

```

arshiyos@arshia-oslabbox:~/syscalls$ vim createwrite.cpp
arshiyos@arshia-oslabbox:~/syscalls$ ls
access.cpp createwrite.cpp
arshiyos@arshia-oslabbox:~/syscalls$ gcc createwrite.cpp
arshiyos@arshia-oslabbox:~/syscalls$ ./a.out
wrote content to file.
arshiyos@arshia-oslabbox:~/syscalls$ ls
access.cpp a.out createwrite.cpp slab2.txt
arshiyos@arshia-oslabbox:~/syscalls$ cat slab2.txt
Arshia Yousefnia
arshiyos@arshia-oslabbox:~/syscalls$ 

```

شکل ۹: اجرای کد بالا

- فراخوانی سیستمی sysinfo فقط یک آرگومان که از نوع پوینتر به struct sysinfo است را ورودی گرفته و خروجی اش در صورت موفق بودن ۰ و در غیر این صورت -۱ است.

همچنین پس از اجرای موفقیت آمیز آن، struct sysinfo که ورودی فراخوانی سیستم به آن اشاره می‌کرد، براساس اطلاعات سیستم پر می‌شود.

در شکل زیر کد مربوط به خواسته این بخش را می‌توان مشاهده کرد. این کد ابتدا فراخوانی سیستمی را انجام می‌دهد، سپس در صورت خطا داشتن، پیغام خطأ چاپ می‌کند و در صورت خطا نداشتن، اطلاعات خواسته شده را چاپ می‌کند.

```

arshiyos@arshia-oslabbox: ~/syscalls
#include <stdio.h>
#include <sys/sysinfo.h>

int main() {
    struct sysinfo info;

    if (sysinfo(&info) == -1) {
        printf("Total RAM: %lu bytes\nFree RAM: %lu\n",
               info.totalram,
               info.freeram);
    } else {
        printf("sysinfo error.\n");
        return 1;
    }
    return 0;
}

```

"sysinfo.cpp" 17L, 270B

2,17 All 23:38 arshiyos@arshia-osla...

شکل ۱۰: کد برنامه‌ی چاپ اطلاعات رم

```

arshiyos@arshia-oslabbox: ~/syscalls
arshiyos@arshia-oslabbox: ~/syscalls $ vim sysinfo.cpp
arshiyos@arshia-oslabbox: ~/syscalls $ ls
access.cpp createwrite.cpp sysinfo.cpp
arshiyos@arshia-oslabbox: ~/syscalls $ gcc sysinfo.cpp
arshiyos@arshia-oslabbox: ~/syscalls $ ./a.out
Total RAM: 8327065600 bytes
Free RAM: 7272329216
arshiyos@arshia-oslabbox: ~/syscalls $

```

arshiyos@arshia-oslabbox: ~/syscalls \$

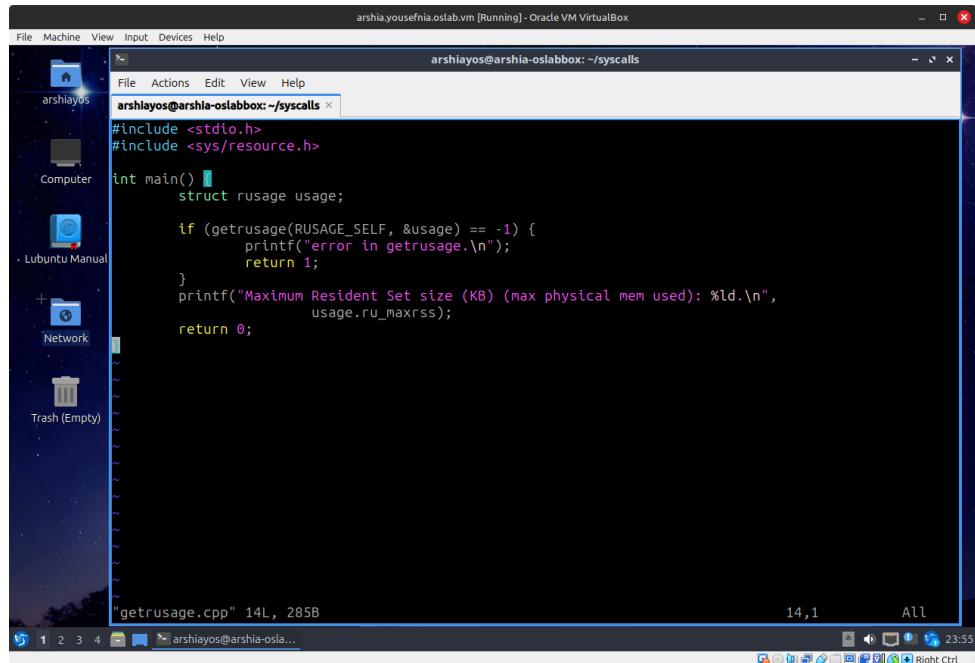
شکل ۱۱: اجرای برنامه‌ی چاپ اطلاعات رم

- فراخوانی سیستمی getrusage دو آرگومان به عنوان ورودی می‌گیرد. اولی مشخص می‌کند که اطلاعات مصرف حافظه برای چه پردازه‌ها یا تردهایی محاسبه شود. از مقادیر پرکارد آن می‌توان RUSAGE_SELF که به پردازه‌ی اجرا کننده‌ی این فراخوانی سیستمی اشاره می‌کند، RUSAGE_CHILDREN که به پردازه‌های فرزند این پردازه که اجراشان تمام شده و در حالت wait هستند اشاره می‌کند، و RUSAGE_THREAD که به تردی که این دستور را اجرا کرده اشاره می‌کند را بیان کرد [۶].

آرگومان ورودی دوم هم یک پوینتر به یک struct rusage است که قرار است پس از اجرای این فراخوانی سیستمی، فیلدهای درون آن به درستی پر شوند.

این استراکت فیلدهای زیادی دارد، اما تنها موردنی که در این بخش به آن نیاز داریم، ru_maxrss است، که میزان حافظه‌ی مصرفی را نشان می‌دهد.

کد آن را در زیر می‌توان مشاهده کرد. این کد ابتدا فراخوانی سیستم را اجرا می‌کند و در صورت خطا نداشتن، اطلاعات خواسته شده را چاپ می‌کند.

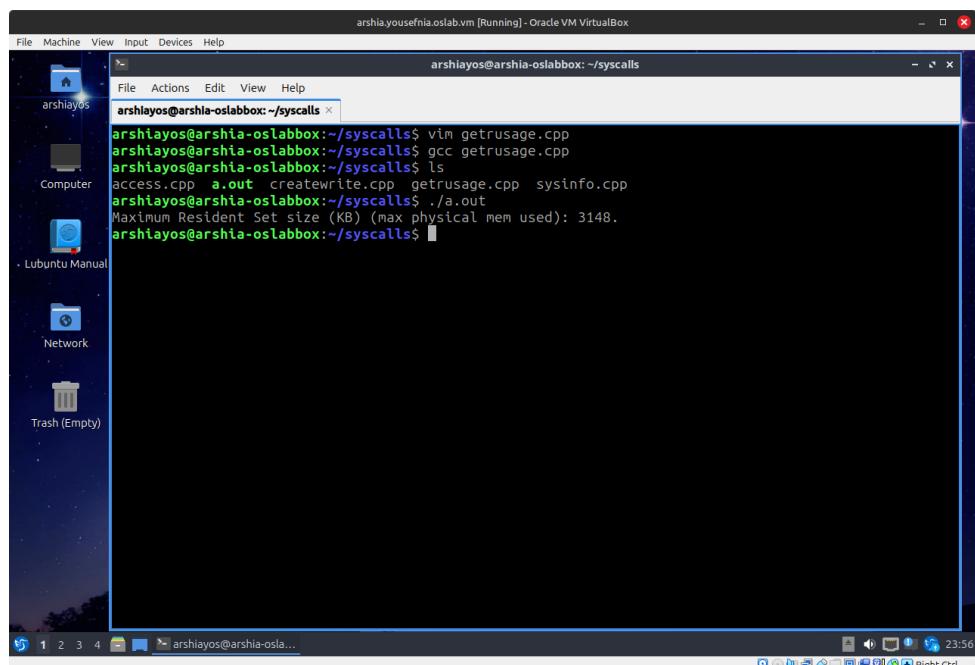


```
#include <stdio.h>
#include <sys/resource.h>

int main() {
    struct rusage usage;

    if (getrusage(RUSAGE_SELF, &usage) == -1) {
        printf("error in getrusage.\n");
        return 1;
    }
    printf("Maximum Resident Set size (KB) (max physical mem used): %ld.\n",
           usage.ru_maxrss);
    return 0;
}
```

شکل ۱۲: کد برنامه‌ی چاپ کننده‌ی حافظه‌ی مصرفی



```
arshiyos@arshia-oslabbox:~/syscalls$ vim getrusage.cpp
arshiyos@arshia-oslabbox:~/syscalls$ gcc getrusage.cpp
arshiyos@arshia-oslabbox:~/syscalls$ ls
access.cpp  a.out  createwrite.cpp  getrusage.cpp  sysinfo.cpp
arshiyos@arshia-oslabbox:~/syscalls$ ./a.out
Maximum Resident Set size (KB) (max physical mem used): 3148.
arshiyos@arshia-oslabbox:~/syscalls$
```

شکل ۱۳: اجرای برنامه‌ی چاپ کننده‌ی حافظه‌ی مصرفی

۱.۵ اضافه کردن یک فراخوانی سیستمی به سیستم عامل

مطابق مراحل دستور کار، هر بخش را توضیح داده و در صورت نیاز، اسکرین شات می‌گذاریم.

۱. از آنجایی که فقط یک کاربر داریم و آن کاربر هم دسترسی root دارد، در این مرحله مشکلی نمی‌تواند رخ بدهد.

۲. مطابق شکل زیر، وارد آن شده‌ایم.

```
arshiyos@arshia-oslabbox: /usr/src/linux-source-6.8.0 $ uname -r
6.11.0-17-generic
arshiyos@arshia-oslabbox: /usr/src $ ls
linux-headers-6.11.0-17-generic  linux-source-6.8.0
linux-hwe-6.11-headers-6.11.0-17  linux-source-6.8.0.tar.bz2
arshiyos@arshia-oslabbox: /usr/src $ cd linux-source-6.8.0/
arshiyos@arshia-oslabbox: /usr/src/linux-source-6.8.0 $ ls
arch          fs           lib           net          Ubuntu.md
block         generic.depmod.log  LICENSES      README       usr
built-in.a    generic.inclusion-list.log  MAINTAINERS  rust        virt
certs          include        Makefile     samples    vmlinux
COPYING       init          mm          security   vmlinux.a
CREDITS      io_uring      modules.builtin  sound      vmlinux.map
crypto        ipc           modules.modinfo  tools
Documentation Kbuild      modules.order   System.map vmlinux.o
drivers       Kconfig      Module.synvers
dropped.txt   kernel
arshiyos@arshia-oslabbox: /usr/src/linux-source-6.8.0 $ make oldconfig
#
# No change to .config
#
arshiyos@arshia-oslabbox: /usr/src/linux-source-6.8.0 $
```

شکل ۱۴: ورود به پوشه‌ی کد منبع هسته‌ی سیستم عامل

۳. در شکل ۱۴ می‌توان خروجی اجرای این دستور را مشاهده کرد.

۴. درستی اجرای این عملیات در شکل زیر قابل مشاهده است.

```

arshayos@arshia-oslabbox: /usr/src/linux-source-6.8.0
drivers      Kconfig          modules.order      tools
dropped.txt  kernel           Module.symvers   ubuntu
#           #
#           # No change to .config
#
#           arshayos@arshia-oslabbox: /usr/src/linux-source-6.8.0$ make oldconfig
#           mkdir -p /usr/src/linux-source-6.8.0/tools/objtool && make O=/usr/src/linux-source-6.8.0 subdir=too
ls/objtool --no-print-directory -C objtool
INSTALL libsubcmd_headers
CALL scripts/checksyscalls.sh
CHK kernel/kheaders_data.tar.xz
Kernel: arch/x86/boot/bzImage is ready (#1)
arshayos@arshia-oslabbox: /usr/src/linux-source-6.8.0$ sudo make modules_install
[sudo] password for arshayos:
INSTALL /lib/modules/6.8.12/modules.order
INSTALL /lib/modules/6.8.12/modules.builtin
INSTALL /lib/modules/6.8.12/modules.builtin.modinfo
SYMLINK /lib/modules/6.8.12/build
INSTALL /lib/modules/6.8.12/kernel/arch/x86/events/amd/amd-uncore.ko
SIGN /lib/modules/6.8.12/kernel/arch/x86/events/amd/amd-uncore.ko
INSTALL /lib/modules/6.8.12/kernel/arch/x86/events/intel/intel-cstate.ko
SIGN /lib/modules/6.8.12/kernel/arch/x86/events/intel/intel-cstate.ko
INSTALL /lib/modules/6.8.12/kernel/arch/x86/events/rapl.ko
SIGN /lib/modules/6.8.12/kernel/arch/x86/events/rapl.ko
INSTALL /lib/modules/6.8.12/kernel/arch/x86/kernel/cpu/mce/mce-inject.ko
SIGN /lib/modules/6.8.12/kernel/arch/x86/kernel/cpu/mce/mce-inject.ko
INSTALL /lib/modules/6.8.12/kernel/arch/x86/kernel/msr.ko
SIGN /lib/modules/6.8.12/kernel/arch/x86/kernel/msr.ko
INSTALL /lib/modules/6.8.12/kernel/arch/x86/kernel/cpuid.ko

```

شکل ۱۵: درستی کامپایل هسته

۵. نصب هسته جدید را دوباره انجام می‌دهیم.

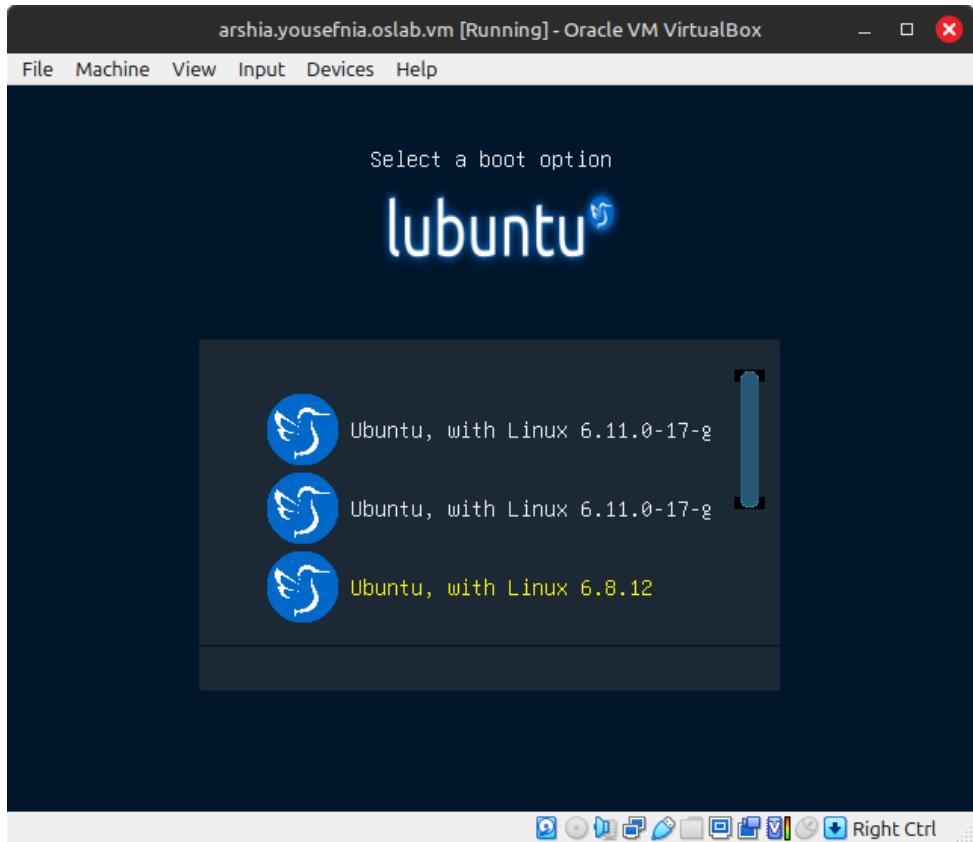
```

arshayos@arshia-oslabbox: /usr/src/linux-source-6.8.0
File Actions Edit View Help
arshayos@arshia-oslabbox: /usr/src/linux-source-6.8.0
SIGN /lib/modules/6.8.12/kernel/net/qrtr/qrtr-mhi.ko
INSTALL /lib/modules/6.8.12/kernel/virt/lib/irqbypass.ko
SIGN /lib/modules/6.8.12/kernel/virt/lib/irqbypass.ko
INSTALL /lib/modules/6.8.12/kernel/ubuntu/ubuntu-host/ubuntu-host.ko
SIGN /lib/modules/6.8.12/kernel/ubuntu/ubuntu-host/ubuntu-host.ko
DEPMOD /lib/modules/6.8.12
arshayos@arshia-oslabbox: /usr/src/linux-source-6.8.0$ sudo make install
INSTALL /boot
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 6.8.12 /boot/vmlinuz-6.8.12
update-initramfs: Generating /boot/initrd.img-6.8.12
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 6.8.12 /boot/vmlinuz-6.8.12
run-parts: executing /etc/kernel/postinst.d/xx-update-initrd-links 6.8.12 /boot/vmlinuz-6.8.12
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 6.8.12 /boot/vmlinuz-6.8.12
Sourcing file `/etc/default/grub'
Sourcing file `/etc/default/grub.d/lubuntu-grub-theme.cfg'
Generating grub configuration file ...
Found theme: /usr/share/grub/themes/lubuntu-grub-theme/theme.txt
Found linux image: /boot/vmlinuz-6.11.0-17-generic
Found initrd image: /boot/initrd.img-6.11.0-17-generic
Found linux image: /boot/vmlinuz-6.8.12
Found initrd image: /boot/initrd.img-6.8.12
Found linux image: /boot/vmlinuz-6.8.12.old
Found initrd image: /boot/initrd.img-6.8.12
Found memtest86+x64 image: /boot/memtest86+x64.bin
Warning: os-prober will not be executed to detect other bootable partitions.
Systems on them will not be added to the GRUB boot configuration.
Check GRUB_DISABLE_OS_PROBER documentation entry.
Adding boot menu entry for UEFI Firmware Settings ...
done
arshayos@arshia-oslabbox: /usr/src/linux-source-6.8.0$ 

```

شکل ۱۶: نصب هسته جدید

۶. سیستم را reboot کرده و سیستم عامل را دوباره اجرا می‌کنیم.



شکل ۱۷ : reboot کردن سیستم

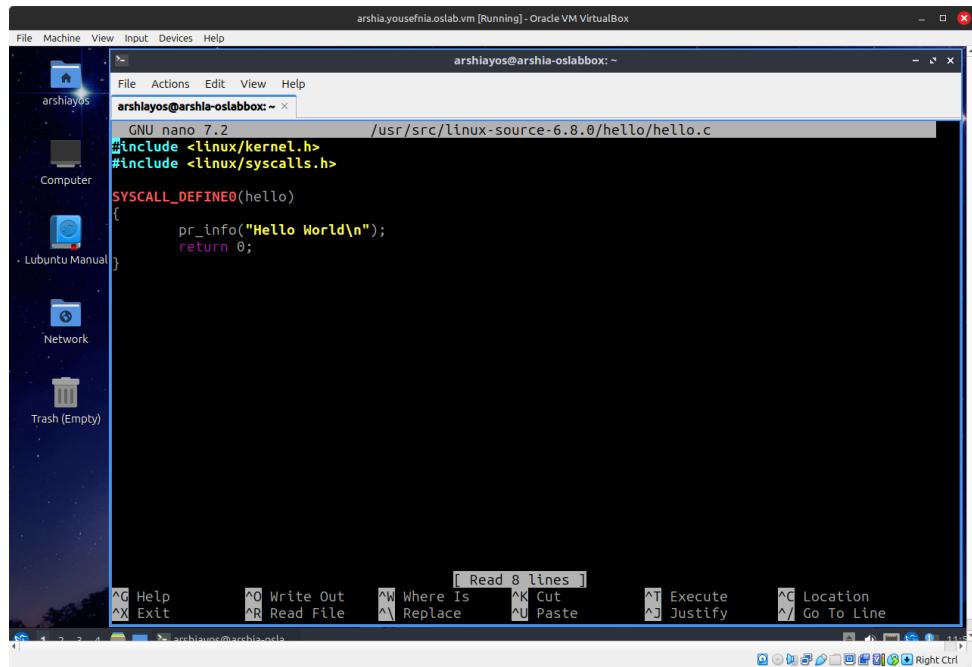
۷. مطابق دستور کار، پوشه‌ی hello را می‌سازیم.

```

arshayos@arshia-oslabbox:~$ uname -r
6.8.12
arshayos@arshia-oslabbox:~$ cd /usr/src/
arshayos@arshia-oslabbox:~$ cd /usr/src/
arshayos@arshia-oslabbox:/usr/src$ ls
linux-headers-6.11.0-17-generic  linux-source-6.8.0
linux-hwe-6.11-headers-6.11.0-17  linux-source-6.8.0.tar.bz2
arshayos@arshia-oslabbox:/usr/src$ cd linux-source-6.8.0/
arshayos@arshia-oslabbox:/usr/src/linux-source-6.8.0$ ls
arch          fs           lib           net           Ubuntu.md
block         generic.depmod.log  LICENSES      README        usr
built-in.a    generic.inclusion-list.log  MAINTAINERS  RUST          virt
certs         include        Makefile     samples        vmlinux
COPYING       int           mm           scripts      vmlinux.a
CREDITS      io_uring      modules.builtin  security    vmlinux-gdb.py
crypto        ipc           modules.builtin.modinfo  sound      vmlinux.map
Documentation Kbuild      modules.order   System.map  vmlinux.o
drivers       Kconfig      Module.synvers  tools
dropped.txt   kernel      modules.order
arshayos@arshia-oslabbox:/usr/src/linux-source-6.8.0$ mkdir hello
arshayos@arshia-oslabbox:/usr/src/linux-source-6.8.0$ cd hello
arshayos@arshia-oslabbox:/usr/src/linux-source-6.8.0/hello$ nano hello.c
arshayos@arshia-oslabbox:/usr/src/linux-source-6.8.0/hello$ 
```

شکل ۱۸ : ساخت پوشه‌ی hello و فایل hello.c

۸. کد داده شده را با کم تغییر (به دلیل ورژن‌های مختلف سیستم عامل) در فایل hello.c در می‌نویسیم.

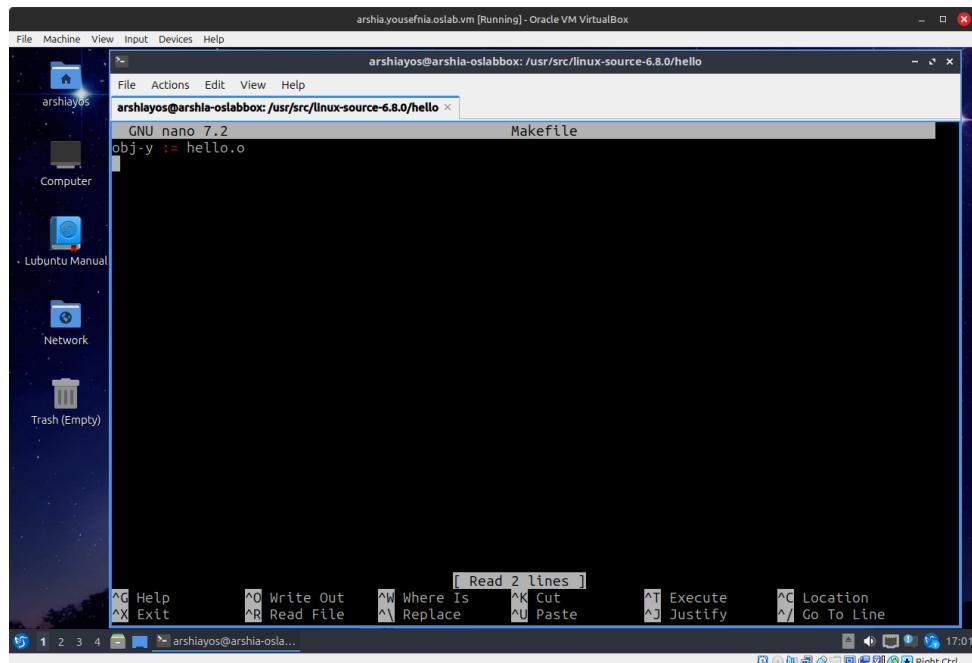


```
GNU nano 7.2 /usr/src/linux-source-6.8.0/hello/hello.c
#include <linux/kernel.h>
#include <linux/syscalls.h>

SYSCALL_DEFINE0(hello)
{
    pr_info("Hello World\n");
    return 0;
}
```

شکل ۱۹ : محتویات فایل hello.c

۹. خواسته شده را در پوشه‌ی hello ساخته و محتوای گفته شده را در آن قرار می‌دهیم.



```
GNU nano 7.2 Makefile
obj-y := hello.o
```

شکل ۲۰ : محتویات Makefile موجود در hello

۱۰. فایل Kbuild موجود در ریشه‌ی کد منبع را باز کرده و hello را در مکان خواسته شده به پ obj-y اضافه می‌کنیم.

```

arshiyos@arshia-oslabbox:/usr/src/linux-source-6.8.0>
GNU nano 7.2
Kbuild

obj-y      += init/
obj-y      += usr/
obj-y      += arch/$(SRCARCH)/
obj-y      += $(ARCH_CORE)
obj-y      += kernel/
obj-y      += certs/
obj-y      += mm/
obj-y      += fs/
obj-y      += ipc/
obj-y      += security/
obj-y      += crypto/
obj-$(CONFIG_BLOCK) += block/
obj-$(CONFIG_IOURING) += io_uring/
obj-$(CONFIG_RUST)   += rust/
obj-y      += $(ARCH_LIB)
obj-y      += drivers/
obj-y      += sound/
obj-$(CONFIG_SAMPLES) += samples/
obj-$(CONFIG_NET)    += net/
obj-y      += virt/
obj-y      += $(ARCH_DRIVERS)
obj-y      += ubuntu/

obj-y += hello/

```

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location
^X Exit ^R Read File ^L Replace ^U Paste ^J Justify ^I Go To Line

شکل ۲۱: محتويات Kbuild موجود در ریشه‌ی کد منبع

۱۱. فایل خواسته شده را باز کرده و خطی مشابه خط گفته شده را به آن اضافه می‌کنیم (به دلیل تفاوت در نسخه‌ی سیستم عامل، کمی نام فایل‌ها و خط اضافه شده با چیزی که در دستور کار گفته شده بود تفاوت دارد).

```

arshiyos@arshia-oslabbox:/usr/src/linux-source-6.8.0/arch/x86/entry/syscalls>
GNU nano 7.2
syscall_64.tbl *

526 x32 timer_create compat_sys_timer_create
527 x32 mq_notify compat_sys_mq_notify
528 x32 kexec_load compat_sys_kexec_load
529 x32 waitid compat_sys_waitid
530 x32 set_robust_list compat_sys_set_robust_list
531 x32 get_robust_list compat_sys_get_robust_list
532 x32 vmsplice sys_vmsplice
533 x32 move_pages sys_move_pages
534 x32 preadv compat_sys_preadv64
535 x32 pwritev compat_sys_pwritev64
536 x32 rt_tgsigqueueinfo compat_sys_rt_tgsigqueueinfo
537 x32 recvmsg compat_sys_recvmsg_time64
538 x32 sendmsg compat_sys_sendmsg
539 x32 process_vm_readv sys_process_vm_readv
540 x32 process_vm_writev sys_process_vm_writev
541 x32 setssockopt sys_setssockopt
542 x32 getsockopt sys_getsockopt
543 x32 io_setup compat_sys_io_setup
544 x32 io_submit compat_sys_io_submit
545 x32 execveat compat_sys_execveat
546 x32 preadv2 compat_sys_preadv64v2
547 x32 pwritev2 compat_sys_pwritev64v2

# This is the end of the legacy x32 range. Numbers 548 and above are
# not special and are not to be used for x32-specific syscalls.

548 common hello sys_hello

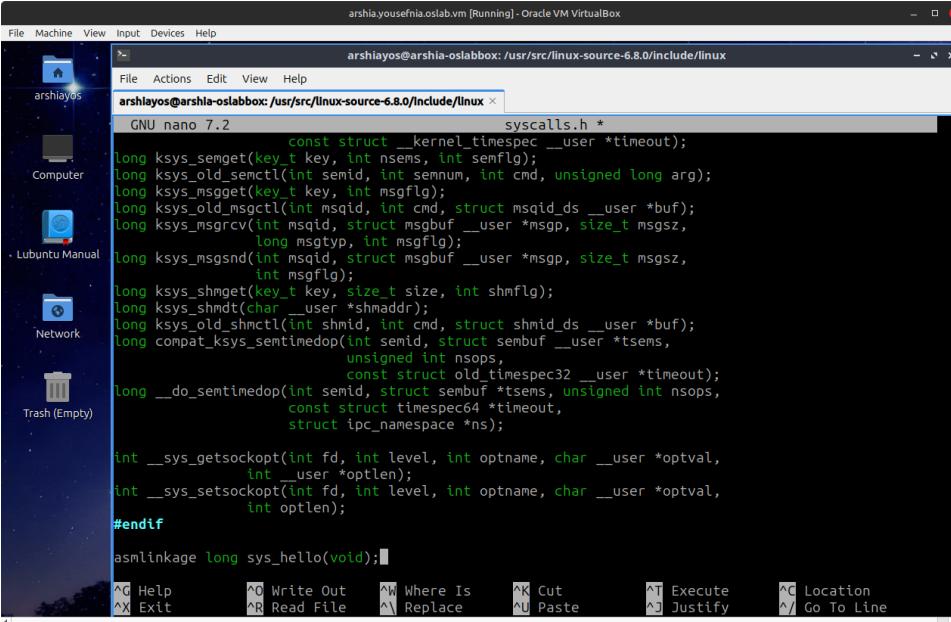
[ Cancelled ]

```

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location
^X Exit ^R Read File ^L Replace ^U Paste ^J Justify ^I Go To Line

شکل ۲۲: اضافه کردن تابع فراخوانی سیستمی ساخته شده به لیست فراخوانی‌های سیستمی

۱۲. در فایل گفته شده در صورت سوال، خط زیر را اضافه می‌کنیم.



The screenshot shows a terminal window titled "arshiyos@arshia-oslabbox: /usr/src/linux-source-6.8.0/include/linux". The terminal displays the content of the file "syscalls.h" from the Linux kernel source code. The code includes various system call definitions such as ksys_semget, ksys_old_semctl, ksys_msget, ksys_old_msctl, ksys_nsget, ksys_old_nsctl, ksys_msnd, ksys_nsnd, ksys_shmget, ksys_shmdt, ksys_old_shmctl, compat_ksys_semtimedop, __do_semtimedop, __sys_getsockopt, __sys_setsockopt, and sys_hello. The terminal interface includes standard Linux command-line tools like nano at the top and a menu bar at the very top.

```
GNU nano 7.2          syscalls.h *
long ksys_semget(key_t key, int nsems, int semflg);
long ksys_old_semctl(int semid, int semnum, int cmd, unsigned long arg);
long ksys_msget(key_t key, int msgflg);
long ksys_old_msctl(int msgid, int cmd, struct msqid_ds __user *buf);
long ksys_nsget(key_t key, int msgflg);
long ksys_nsgetv(int msgid, struct msgbuf __user *msgp, size_t msgsz,
                long msgtyp, int msgflg);
long ksys_msnd(int msgid, struct msgbuf __user *msgp, size_t msgsz,
               int msgflg);
long ksys_shmget(key_t key, size_t size, int shmflg);
long ksys_shmdt(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,
                     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);
#endif

asm linkage long sys_hello(void);
```

شکل ۲۳: خط اضافه شده به فایل syscalls.h

۱۳. هسته را مجدد کامپایل کرده و نصب می‌کنیم.

File Machine View Input Devices Help

arshia.yousefnia.oslab.vm [Running] - Oracle VM VirtualBox

arshiyos@arshia-oslabbox: /usr/src/linux-source-6.8.0

```
CC      arch/x86/boot/compressed/error.o
OBJCOPY arch/x86/boot/compressed/vmlinux.bin
CC      arch/x86/boot/compressed/early_serial_console.o
CC      arch/x86/boot/compressed/kaslr.o
CC      arch/x86/boot/compressed/ident_map_64.o
CC      arch/x86/boot/compressed/idx_64.o
AS      arch/x86/boot/compressed/idx_handlers_64.o
AS      arch/x86/boot/compressed/mem_encrypt.o
CC      arch/x86/boot/compressed/pgtable_64.o
CC      arch/x86/boot/compressed/sev.o
CC      arch/x86/boot/compressed/acpi.o
AS      arch/x86/boot/compressed/tdcall.o
CC      arch/x86/boot/compressed/tdx-shared.o
CC      arch/x86/boot/compressed/nem.o
CC      arch/x86/boot/compressed/efi.o
AS      arch/x86/boot/compressed/efi_mixed.o
CC      arch/x86/boot/compressed/misc.o
ZSTD22 arch/x86/boot/compressed/vmlinux.bin.zst
MKPIGGY arch/x86/boot/compressed/piggy.S
AS      arch/x86/boot/compressed/piggy.o
LD      arch/x86/boot/compressed/vmlinux
ZOFFSET arch/x86/boot/zoffset.h
OBJCOPY arch/x86/boot/vmlinux.bin
AS      arch/x86/boot/header.o
LD      arch/x86/boot/setup.elf
OBJCOPY arch/x86/boot/setup.bin
BUILD arch/x86/boot/bzImage
Kernel: arch/x86/boot/bzImage is ready (#4)
arshiyos@arshia-oslabbox: /usr/src/linux-source-6.8.0$ sudo make -j$(nproc) modules_install & sudo make install
```

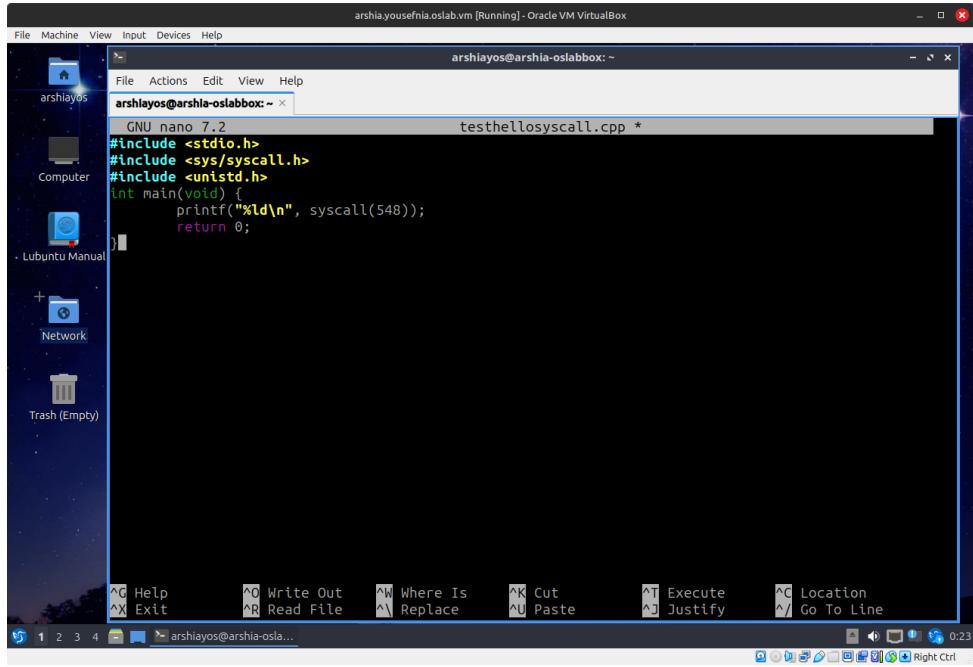
شکل ۲۴: کامپایل و نصب مجدد هسته

حال که فراخوانی سیستم به سیستم عامل اضافه شد، با reboot کردن سیستم می‌توان از آن فراخوانی سیستمی استفاده کرد.

حال تمارین این بخش را انجام می‌دهیم.

- ابتدا برنامه‌ای می‌نویسیم که با دستور `syscall` فراخوانی سیستمی اضافه شده را اجرا کند. می‌دانیم شماره‌ی

این فراخوانی سیستمی 548 است و آرگومان ورودی ندارد. پس برنامه‌ی خواسته شده به صورت شکل زیر می‌شود:



```
GNU nano 7.2
#include <stdio.h>
#include <sys/syscall.h>
#include <unistd.h>
int main(void) {
    printf("%ld\n", syscall(548));
    return 0;
}
```

شکل ۲۵ : استفاده از فراخوانی سیستمی hello در کد یک برنامه

سپس این برنامه را کامپایل و اجرا کرده و با دستور dmesg (صرفا چاپ کردن چند خط آخر آن با دستور tail) می‌توان دید که فراخوانی سیستمی اجرا شده است.

- از آنجایی که قبل مراحل اضافه کردن فراخوانی سیستمی گفته شده، در بخش‌های تکرار توضیحی نداده و صرفا از مراحل اسکرین شات می‌گذاریم.

The screenshot shows a Linux desktop environment with a terminal window open. The terminal window title is "arshiyos@arshia-oslabbox: /usr/src/linux-source-6.8.0". The terminal content shows the following commands being run:

```

arshiyos@arshia-oslabbox:~$ cd /usr/src/linux-source-6.8.0/
arshiyos@arshia-oslabbox:/usr/src/linux-source-6.8.0$ mkdir adder
arshiyos@arshia-oslabbox:/usr/src/linux-source-6.8.0$ cp hello/hello.c adder/adder.c
arshiyos@arshia-oslabbox:/usr/src/linux-source-6.8.0$ nano adder.c

```

شکل ۲۶: اضافه کردن پوشی adder

کد فراخوانی سیستمی، علاوه بر دادن خروجی مناسب، ورودی‌ها و خروجی را چاپ می‌کند تا بررسی آن در آینده راحت‌تر شود.

The screenshot shows a Linux desktop environment with a nano editor window open. The editor title is "GNU nano 7.2" and the file name is "adder/adder.c". The code in the editor is:

```

#include <linux/kernel.h>
#include <linux/syscalls.h>

SYSCALL_DEFINE2(add, int, a, int, b)
{
    pr_info("syscall add(): %d + %d = %d\n", a, b, a + b);
    return a + b;
}

```

شکل ۲۷: کد فراخوانی سیستمی adder

```

arshiyos@arshia-oslabox: /usr/src/linux-source-6.8.0
GNU nano 7.2
Kbuild *
obj-y      += init/
obj-y      += usr/
obj-y      += arch/$(SRCARCH)
obj-y      += $(ARCH_CORE)
obj-y      += kernel/
obj-y      += certs/
obj-y      += mm/
obj-y      += fs/
obj-y      += ipc/
obj-y      += security/
obj-y      += crypto/
obj-$(CONFIG_BLOCK) += block/
obj-$(CONFIG_IOURING) += io_uring/
obj-$(CONFIG_RUST) += rust/
obj-y      += $(ARCH_LIB)
obj-y      += drivers/
obj-y      += sound/
obj-$(CONFIG_SAMPLES) += samples/
obj-$(CONFIG_NET)    += net/
obj-y      += virt/
obj-y      += $(ARCH_DRIVERS)
obj-y      += ubuntu/

obj-y += hello/
obj-y += adder/█

```

File Actions Edit View Help
arshiyos@arshia-oslabox: /usr/src/linux-source-6.8.0
GNU nano 7.2 Kbuild *
Help Write Out Where Is Cut Execute Location
Exit Read File Replace Paste Justify Go To Line
arshiyos@arshia-osl... 10:42

شکل ۲۸: اضافه کردن به adder

```

arshiyos@arshia-oslabox: /usr/src/linux-source-6.8.0
GNU nano 7.2 arch/x86/entry/syscalls/syscall_64.tbl *
538 x32 sendmmsg compat_sys_sendmmsg
539 x32 process_vm_readv sys_process_vm_readv
540 x32 process_vm_writev sys_process_vm_writev
541 x32 setsockopt sys_setsockopt
542 x32 getsockopt sys_getsockopt
543 x32 io_setup compat_sys_io_setup
544 x32 io_submit compat_sys_io_submit
545 x32 execveat compat_sys_execveat
546 x32 preadv2 compat_sys_preadv64v2
547 x32 pwritev2 compat_sys_pwritev64v2
# This is the end of the legacy x32 range. Numbers 548 and above are
# not special and are not to be used for x32-specific syscalls.
548 common hello sys hello
549 common add sys_add█

```

File Actions Edit View Help
arshiyos@arshia-oslabox: /usr/src/linux-source-6.8.0
GNU nano 7.2 arch/x86/entry/syscalls/syscall_64.tbl *
Help Write Out Where Is Cut Execute Location
Exit Read File Replace Paste Justify Go To Line
arshiyos@arshia-osl... 10:42

شکل ۲۹: اضافه کردن adder به لیست فراخوانی‌های سیستم

```

arshayos@arshia-oslabbox: /usr/src/linux-source-6.8.0
include/linux/syscalls.h *
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);
#endif

asmlinkage long sys_hello(void);
asmlinkage long sys_add(int a, int b);

Save modified buffer? [Y/N]
Y Yes
N No      Cancel

```

شکل ۳۰: اضافه کردن به adder syscalls.h

```

arshayos@arshia-oslabbox: /usr/src/linux-source-6.8.0
-W FILE, --what-if=FILE, --new-file=FILE, --assume-new=FILE
Consider FILE to be infinitely new.
--warn-undefined-variables Warn when an undefined variable is referenced.

This program built for x86_64-pc-linux-gnu
Report bugs to <bug-make@gnu.org>
arshayos@arshia-oslabbox: /usr/src/linux-source-6.8.0$ sudo make -j$(nproc) modules_install
INSTALL /lib/modules/6.8.12/modules.order
INSTALL /lib/modules/6.8.12/modules.builtin
INSTALL /lib/modules/6.8.12/build
INSTALL /lib/modules/6.8.12/kernel/arch/x86/events/rapl.ko
INSTALL /lib/modules/6.8.12/kernel/arch/x86/kernel/msr.ko
INSTALL /lib/modules/6.8.12/kernel/arch/x86/crypto/aesni-intel.ko
SIGN /lib/modules/6.8.12/kernel/arch/x86/events/rapl.ko
SIGN /lib/modules/6.8.12/kernel/arch/x86/kernel/msr.ko
SIGN /lib/modules/6.8.12/kernel/arch/x86/crypto/aesni-intel.ko
INSTALL /lib/modules/6.8.12/kernel/arch/x86/crypto/sha1-ssse3.ko
INSTALL /lib/modules/6.8.12/kernel/arch/x86/crypto/ghash-clmulni-intel.ko
SIGN /lib/modules/6.8.12/kernel/arch/x86/crypto/sha1-ssse3.ko
SIGN /lib/modules/6.8.12/kernel/arch/x86/crypto/sha256-ssse3.ko
INSTALL /lib/modules/6.8.12/kernel/arch/x86/crypto/polyval-clmulni.ko
SIGN /lib/modules/6.8.12/kernel/arch/x86/crypto/ghash-clmulni-intel.ko
INSTALL /lib/modules/6.8.12/kernel/arch/x86/crypto/crc32-pclmul.ko
INSTALL /lib/modules/6.8.12/kernel/arch/x86/crypto/crc32-pclmul.ko
SIGN /lib/modules/6.8.12/kernel/arch/x86/crypto/polyval-clmulni.ko
SIGN /lib/modules/6.8.12/kernel/arch/x86/crypto/crc32-pclmul.ko
INSTALL /lib/modules/6.8.12/kernel/kernel/configs.ko
INSTALL /lib/modules/6.8.12/kernel/fs/binfmt_misc.ko

```

شکل ۳۱: کامپایل مجدد هسته و نصب ماژول ها

```

arshiyos@arshia-oslabbox: /usr/src/linux-source-6.8.0
arshiyos@arshia-oslabbox: /usr/src/linux-source-6.8.0$ sudo make install
INSTALL /boot
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 6.8.12 /boot/vmlinuz-6.8.12
update-initramfs: Generating /boot/initrd.img-6.8.12
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 6.8.12 /boot/vmlinuz-6.8.12
run-parts: executing /etc/kernel/postinst.d/xx-update-initrd-links 6.8.12 /boot/vmlinuz-6.8.12
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 6.8.12 /boot/vmlinuz-6.8.12
Sourcing file '/etc/default/grub'
Sourcing file '/etc/default/grub.d/lubuntu-grub-theme.cfg'
Generating grub configuration file ...
Found theme: /usr/share/grub/themes/lubuntu-grub-theme/theme.txt
Found linux image: /boot/vmlinuz-6.11.0-17-generic
Found initrd image: /boot/initrd.img-6.11.0-17-generic
Found linux image: /boot/vmlinuz-6.8.12
Found initrd image: /boot/initrd.img-6.8.12
Found linux image: /boot/vmlinuz-6.8.12.old
Found initrd image: /boot/initrd.img-6.8.12
Found memtest86+x64 image: /boot/memtest86+x64.bin
Warning: os-prober will not be executed to detect other bootable partitions.
Systems on them will not be added to the GRUB boot configuration.
Check GRUB_DISABLE_OS_PROBER documentation entry.
Adding boot menu entry for UEFI Firmware Settings ...
done
arshiyos@arshia-oslabbox: /usr/src/linux-source-6.8.0$ 

```

شکل ۳۲: نصب مجدد هسته

```

arshiyos@arshia-oslabbox: ~
arshiyos@arshia-oslabbox: ~$ gnu nano 7.2 testaddsyscall.cpp *
GNU nano 7.2
#include <stdio.h>
#include <sys/syscall.h>
#include <unistd.h>
int main(void) {
    int a = 14;
    int b = 23;
    printf("%d\n", syscall(549, a, b));
    return 0;
}

```

شکل ۳۳: نوشتن برنامه‌ای که از فراخوانی سیستمی adder استفاده می‌کند

The screenshot shows a terminal window titled "arshayos@arshia-oslabbox: ~" running on a desktop environment. The terminal displays the following command-line session:

```
arshayos@arshia-oslabbox: ~
command 'nano' from deb nano (7.2-2ubuntu0.1)
command 'dan' from deb emboss (6.6.0+dfsg-12ubuntu1)
command 'na6' from deb ipv6 toolkit (2.0+ds.1-2)
command 'pan' from deb pan (0.155-1)
command 'man' from deb man-db (2.12.0-1)
command 'an' from deb an (1.2-7build2)
command 'nn' from deb nn (6.7.3-15)
command 'nad' from deb ncbi-acc-download (0.2.8-1)
Try: sudo apt install <deb name>
arshayos@arshia-oslabbox:~$ nano testaddsyscall.cpp
arshayos@arshia-oslabbox:~$ gcc testaddsyscall.cpp
arshayos@arshia-oslabbox:~$ ./a.out
37
arshayos@arshia-oslabbox:~$ sudo dmesg | tail
[sudo] password for arshayos:
[ 16.966916] Console: switching to colour frame buffer device 160x50
[ 16.966926] vmwgfx 0000:00:02.0: [drm] fb0: vmwgfxdrmfb frame buffer device
[ 18.896446] snd_intel8x0 0000:00:05.0: allow list rate for 1028:0177 is 48000
[ 21.255740] kaudited_printk_skb: 112 callbacks suppressed
[ 21.255744] audit: type=1400 audit(1753172138.312:124): apparmor="STATUS" operation="profile_replace" info="same as current profile, skipping" profile="unconfined" name="rsyslogd" pid=646 comm="apparmor_parser"
[ 27.891464] NET: Registered PF_QIPCRTR protocol family
[ 32.612335] e1000: enp0s3 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: RX
[ 36.374987] audit: type=1400 audit(1753172153.431:125): apparmor="DENIED" operation="capable" class="cap" profiles="/usr/sbin/cupsd" pid=794 comm="cupsd" capability=12 capname="net_admin"
[ 45.616721] systemd-journald[263]: /var/log/journal/3d12e25e5f274c70ad7a2204b32abe1e/user-1000.journal: Journal file uses a different sequence number ID, rotating.
[ 341.016315] syscall add(): 14 + 23 = 37
arshayos@arshia-oslabbox:~$
```

شکل ۳۴: خروجی برنامه‌ی بالا

مراجع

- [١] Linux man-pages project. access(2) – Linux manual page. Accessed: -2025-07-28 .2025 URL: <https://man7.org/linux/man-pages/man2/access.2.html>.
- [٢] Linux man-pages project. open(2), openat(2) – Linux manual page. Accessed: .2025-07-28 .2025 URL: <https://man7.org/linux/man-pages/man2/open.2.html>.
- [٣] Linux man-pages project. write(2) – Linux manual page. Accessed: -2025-07-28 .2025 URL: <https://man7.org/linux/man-pages/man2/write.2.html>.
- [٤] Linux man-pages project. close(2) – Linux manual page. Accessed: -2025-07-28 .2025 URL: <https://man7.org/linux/man-pages/man2/close.2.html>.
- [٥] Linux man-pages project. sysinfo(2) – Linux manual page. Accessed: -2025-07-28 .2025 URL: <https://man7.org/linux/man-pages/man2/sysinfo.2.html>.
- [٦] Linux man-pages project. getrusage(2) – Linux manual page. Accessed: -2025-07-28 .2025 URL: <https://man7.org/linux/man-pages/man2/getrusage.2.html>.