



دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

فاز صفر پروژه درس سیستم‌های عامل

گردآورندگان

نیمیا جمالی ۹۶۱۰۵۶۶۱

علیرضا دقیق ۹۶۱۰۵۷۲۳

سینا کاظمی ۹۶۱۰۶۰۱۱

فهرست مطالب

مقدمه

طراحی راه‌انداز

تابع task_init

تابع task_exit

تابع device_llseek

تابع device_write

تابع copy_from_user

بازسازی PID

تابع device_read

یافتن پردازنده بر اساس PID

حالت پردازنده

اطلاعات حسابداری

اطلاعات مربوط به فایل‌ها

تابع copy_to_user

کامپایل راه‌انداز

طراحی برنامه‌ی سطح کاربر

اجرای برنامه‌ی سطح کاربر

منابع

مقدمه

در این فاز از پروژه، با ورودی گرفتن یک PID و یک period از کاربر، باید اطلاعات مربوط به آن پردازش را در خروجی به کاربر نمایش دهیم. به این منظور یک راه‌انداز به نام `kernelmodule.c` تعریف شده است که در آن با استفاده از PID ورودی کاربر، اطلاعات مربوط به آن پردازش به کاربر برگردانده می‌شود. رابط کاربری نیز به وسیله‌ی برنامه‌ی `api.c` تعریف شده است. در قسمت‌های بعدی به شرح نحوه‌ی طراحی راه‌انداز و رابط کاربری می‌پردازیم.

طراحی راه‌انداز

برای طراحی یک `char driver` علاوه بر دو تابع `init` و `exit`، باید توابعی از `struct` ای به نام `file_operations` که در آدرس `linux/fs.h` موجود است، مجدداً تعریف کنیم. `open`، `release`، `read`، `llseek` و `write` توابع مد نظر هستند که از این بین توابع `open` و `release` هم به همان شکلی که در حالت `default` هستند، درست کار می‌کنند و لذا نیازی به تعریف مجدد آنها نیست. در نهایت با تعریف `owner` برای نمونه `struct` ساخته شده و تناظر ایجاد کردن بین توابع تعریف شده و توابع مورد نیاز `struct`، طراحی این ماژول به پایان می‌رسد. تمامی توابع تعریف شده در این بخش ایستا (`static`) هستند و از بیرون ماژول به آنها دسترسی نداریم. حال به توضیح عملکرد توابع تعریف شده می‌پردازیم:

تابع `task_init`

تابع `init` امکاناتی را که توسط ماژول ارائه شده است ثبت می‌کند. این امکانات می‌تواند یک `driver` باشد یا یک چکیده از یک نرم‌افزار که توسط یک برنامه قابل دسترسی است. برای فراخوانی این تابع استفاده از `module_init` الزامی می‌باشد. این ماکرو بخش ویژه‌ای را به `object` `code` اضافه می‌کند که مشخص می‌کند مقداردهی اولیه ماژول یافت شده است. بدون این تعریف `initialization` هرگز انجام نمی‌شود. یکی از اولین کارهایی که `driver` هنگام تنظیم یک `char device` باید انجام دهد، به‌دست آوردن یک یا چند شماره دستگاه برای کار کردن با آن می‌باشد. تابع مورد نظر برای این کار `register_chrdev_region` می‌باشد که در `linux/fs.h` قرار دارد.

اگر از قبل بدانیم دقیقاً کدام شماره دستگاه را نیاز داریم، تابع `register_chrdev_region` به خوبی عمل می‌کند. اما معمولاً `major number` دستگاه مورد استفاده را نمی‌دانیم. لینوکس به صورت پویا `device number` ها را نگه می‌دارد و اختصاص می‌دهد و در نتیجه باید از راهکار دیگری آن را به دست آورد. به این منظور از تابع `alloc_chrdev_region` استفاده کردیم که تعریف آن به صورت زیر است:

```
int alloc_chrdev_region(dev_t *dev, unsigned int firstminor, unsigned int count, char *name);
```

اولین ورودی شروع `device number` از محدوده‌ای است که می‌خواهیم (معمولاً ۰ در نظر گرفته می‌شود). `firstminor` باید اولین شماره درخواست شده برای استفاده باشد (۰). `count` تعداد کل دستگاه‌های درخواست شده است. اگر این عدد بزرگ باشد دامنه درخواستی می‌تواند به `major number` بعدی برود اما تا زمانی که محدوده شماره درخواستی در دسترس باشد، همه چیز درست کار می‌کند. ورودی آخر هم نام دستگاهی است که مربوط به محدوده‌ی مربوطه است.

مقدار منفی در خروجی تابع نشان‌دهنده وجود خطا می‌باشد. پس از آن نوبت به `cdev` می‌رسد. هسته از ساختارهایی از نوع ساختار `cdev` استفاده می‌کند تا `char device` های داخلی را نمایش دهد. قبل از این که هسته توابع `device` را فراخوانی کند، باید یک یا چندتا از این ساختارها را اختصاص داده و ثبت کند.

پس از تنظیم ساختار `cdev` با استفاده از تابع `cdev_add` به `kernel` اطلاع می‌دهیم. با دو روش می‌توانیم `device` مورد نظر را بسازیم: ۱- در ترمینال و با استفاده از دستور `sudo mknod` عمل کنیم که نیازمند این است که `major number` را بدانیم ۲- در خود کد از تابع `device_create` استفاده کنیم.

در کد `kernelmodule.c` از راه دوم و تابع `device_create` استفاده کرده‌ایم. در نتیجه با اجرای `task_init`، در پوشه‌ی `/dev` پرونده‌ی ارتباطی مورد نظر ساخته می‌شود. نام این پرونده `first_phase` است.

در نهایت در صورت موفقیت عملیات ایجاد پرونده‌ی ارتباطی، پیغام “`module loaded`” مشاهده می‌شود.

تابع `task_exit`

در این تابع تمامی منابع اختصاص داده شده، آزاد می‌شوند و در نهایت پیغام “`module unloaded`” در ترمینال قابل مشاهده خواهد بود.

تابع `device_lseek`

تابع `lseek` یک `system call` است که برای تغییر دادن موقعیت پوینتر فایل استفاده می‌شود. این موقعیت هم می‌تواند مطلق و هم می‌تواند نسبی باشد. این تابع سه ورودی دریافت می‌کند که به شرح هر کدام می‌پردازیم.

`static loff_t lseek(struct file *file, loff_t position, int whence)`
ورودی اول فایل مورد نظر است. ورودی دوم `position` است که از جنس `long offset` و ۶۴ بیتی است و خروجی تابع نیز از این جنس است. سومین ورودی اگر مقدار ۰ داشته باشد مقدار `offset` باید مطلق محاسبه گردد. اگر مقدار ۱ داشته باشد مقدار `offset` به طور نسبی از موقعیت کنونی اشاره گر حساب می‌شود. و اگر مقدار ۲ داشته باشد، `offset` به طور نسبی از موقعیت انتهای فایل محاسبه می‌گردد. در این برنامه تنها حالت ۰ بودن `whence` (`SEEK_SET`) مد نظر است. در انتها `f_pos` را که یکی از پارامترهای `file` است، به مقدار `position` تغییر داده و خروجی می‌دهیم.

تابع `device_write`

تابع `write` یک `system call` است که برای نوشتن داده‌ها در فایل مورد استفاده قرار می‌گیرد، به این صورت که کاراکترهای موجود در آرایه `buffer` را در `device_file` می‌نویسد. ساختار این تابع به صورت زیر است:

`static ssize_t read(struct file *file, char *buf, size_t count, loff_t *offset)`

که در آن `count` نشان‌دهنده‌ی تعداد بایت‌های قابل نوشتن و `buf` نیز آرایه کاربر است. بدنه‌ی این تابع شامل دو قسمت عمده است که به شرح هر کدام می‌پردازیم:

تابع `copy_from_user`

با استفاده از این تابع، می‌توان آرایه‌ای از کاراکترها را از حالت `user` گرفت و در حالت `kernel` کپی کرد:
`unsigned long copy_from_user(char *destination, char *user_buffer, long count)`

مطابق ساختار بالا، این تابع به تعداد `count`، عناصر آرایه `user_buffer` را در آرایه `destination` کپی می‌کند. در صورت موفقیت، این تابع مقدار ۰ را باز می‌گرداند و در حالت کلی خروجی این تابع برابر با تعداد بایت‌های کپی نشده است. در کد `kernelmodule.c` ساختاری به نام `char_arr` تعریف شده که آرایه‌ای از جنس `char` به نام `array` دارد و عناصر `buf` که در واقع رقم‌های `PID` هستند که در سطح

کاربر به کاراکتر تبدیل شده‌اند، در `char_arr.array` کپی می‌شوند و از این پس ارقام PID در این آرایه موجود خواهند بود.

بازسازی PID

در این قسمت با داشتن ارقام PID به صورت کاراکتر، قصد داریم PID را به صورت عددی (integer) بازسازی کنیم. ارقام PID به صورت برعکس در آرایه `char_arr.array` ذخیره شده‌اند، به این صورت که کم‌ارزش‌ترین رقم در `char_arr.array[0]` و پر ارزش‌ترین رقم در `char_arr.array[count-1]` ذخیره شده‌است. لذا متغیری به اسم `num` با مقدار اولیه صفر در نظر می‌گیریم و در یک حلقه با پیمایش از $i = \text{count} - 1$ تا 0، ابتدا متغیر `num` را ۱۰ برابر کرده و سپس کاراکتر موجود در خانه‌ی `i` ام را به رقم تبدیل نموده و با `num` جمع می‌کنیم. در نتیجه پس از اتمام حلقه، PID ورودی در متغیر `num` ذخیره می‌شود.

تابع device_read

تابع `read` یک system call است که داده را می‌خواند و آن را در `buffer` می‌ریزد.

ساختار کلی این تابع به صورت زیر است:

```
static ssize_t read(struct file *file, char *user_buffer, size_t size, loff_t *offset)
```

که در آن `*offset` تعداد بایت‌هایی است که خوانده شده و در آرایه `user_buffer` ذخیره می‌شود. در سطح هسته این تابع باید مجدداً به گونه‌ای تعریف گردد که بتوانیم با استفاده از آن اطلاعات مربوط به پردازش‌های که PID آن برابر با متغیر `num` - که در قسمت `write` محاسبه گردید - است را در آرایه کاربر بریزیم. در نتیجه تابع `device_read` را می‌توان متشکل از چند قسمت اصلی دانست که در ادامه به آنها می‌پردازیم:

یافتن پردازش بر اساس PID

به طور کلی در لینوکس اطلاعات مربوط به پردازش در `task_struct` نگهداری می‌شود.

برای پیدا کردن پردازش بر اساس PID از تابع `pid_task` که در پکیج `rcupdate.h` وجود دارد استفاده می‌کنیم، خروجی تابع یک `*task_struct` از نوع `struct` است که در آن اطلاعات مربوط به پردازش‌ها

(حالت پردازش، آدرس فایل های باز مربوط به آن پردازش و ...) نگهداری می شود، ورودی تابع pid_task یک pid *pid struct است و یک pid_type:

pid_task(struct pid *pid, enum pid_type)

حال برای این که عددی که کاربر به عنوان ورودی وارد کرده است را به * pid struct تبدیل کنیم، می بایست از تابع find_vpid استفاده کنیم که یک int (همان PID ورودی کاربر) را به عنوان ورودی می گیرد و خروجی * pid می دهد. سپس این خروجی به عنوان ورودی اول تابع pid_task مورد استفاده قرار می گیرد و ورودی دوم آن یک enum است که نوع PID را مشخص می کند، آن را در حالت PIDTYPE_PID قرار می دهیم.

پس از این که task_struct مربوط به پردازشی مد نظر کاربر را پیدا کردیم، می توانیم اطلاعات مربوط به آن را با استفاده از عملگر >- استخراج کنیم.

حالت پردازش

برای فهمیدن حالت پردازش ابتدا به وسیله state->tsk مقدار state را که یک عدد صحیح (int) است، می خوانیم و در یک متغیر به نام state می ریزیم. برای ذخیره کردن در فایل ابتدا باید آن را در user_buff ذخیره کنیم که این آرایه، یک آرایه از جنس char است. در یک حلقه و طی دو مرحله، ابتدا هر رقم state را به وسیله دستور 10 % state + '0' به char تبدیل می کنیم و سپس آن را در آرایه user_buff ذخیره می کنیم.

اطلاعات حسابداری

با توجه به توضیحات quera، ذکر دو مورد از اطلاعات حسابداری کافی است که از میان آنها nvcsw و nivcsw انتخاب شده اند.

nvcsw (Number of Voluntary Context Switches): تعداد context switch های داوطلبانه را مشخص می کند.

nivcsw (Number of Involuntary Context Switches): تعداد context switch های غیر داوطلبانه را مشخص می کند.

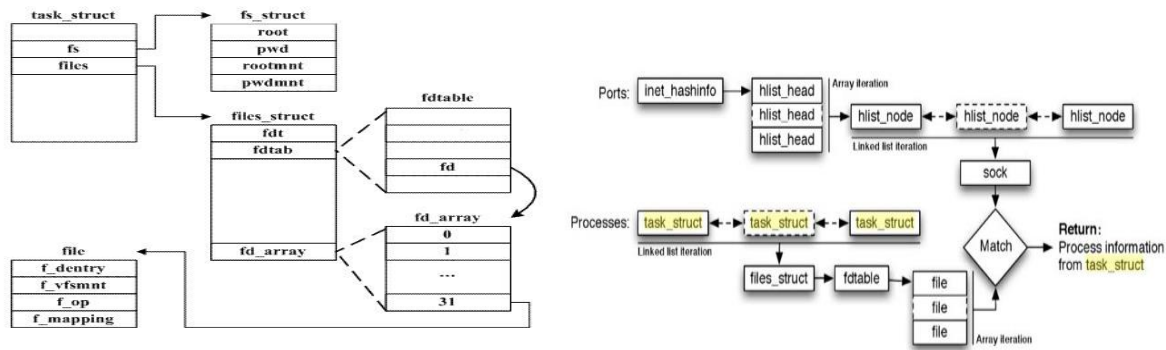
برای خواندن این دو مقدار، مشابه با state عمل می کنیم، یعنی ابتدا به وسیله دو دستور tsk->nvcsw و tsk->nivcsw این مقادیر را می خوانیم و سپس در دو متغیر از جنس unsigned long می ریزیم. سپس

مشابه state این دو مقدار را به آرایه‌ای از کاراکترها تبدیل می‌کنیم و در انتهای کار آن را در آرایه user_buff و بعد از مقادیر ذخیره شده برای حالت پردازش قرار می‌دهیم.

اطلاعات مربوط به فایل‌ها

برای به دست آوردن اطلاعات فایل‌ها باید ابتدا از task_struct به دست آمده، files_struct مربوط به آن را بگیریم. در files_struct کل اطلاعات مربوط به فایل‌های آن پردازش نگهداری می‌شود (مثلاً لیستی از فایل‌ها و ...). سپس یک fd_table به نام *files_table تعریف می‌کنیم که در آن کل اطلاعات مربوط به یک فایل ذخیره می‌شود و ما برای به دست آوردن اطلاعات مربوط به کل فایل‌ها، باید روی آنها for بزنیم و تا زمانی که یکی از آن fd_table‌ها برابر null شود، حلقه را ادامه داده. اطلاعات مربوط به هر فایل را به دست آوریم.

سپس از روی fdtable هر فایل f_path را می‌گیریم و برای به دست آوردن آدرس کامل از یک api که مرتبط با file system است، استفاده می‌کنیم و f_path را به عنوان ورودی به آن می‌دهیم و آدرس کامل فایل‌های باز را خروجی می‌گیریم و در آرایه user_buff و به صورت کاراکتری ذخیره می‌کنیم. همچنین عبارت kmalloc برای گرفتن فضای heap در حالت kernel می‌باشد.



تابع copy_to_user

در این قسمت آرایه user_buff که شامل کاراکترهای خروجی است، در آرایه‌ای که کاربر در تابع read استفاده کرده است، کپی می‌شود. فقط برای آن که در حالت کاربر بدانیم خروجی تا کجا باید چاپ شود، پس از اطلاعات مربوط به فایل‌ها، یک کاراکتر '\0' به آرایه user_buff اضافه می‌کنیم و آن را برابر با اندیس '\0' در نظر می‌گیریم، سپس از تابع copy_to_user به صورت زیر استفاده می‌کنیم:

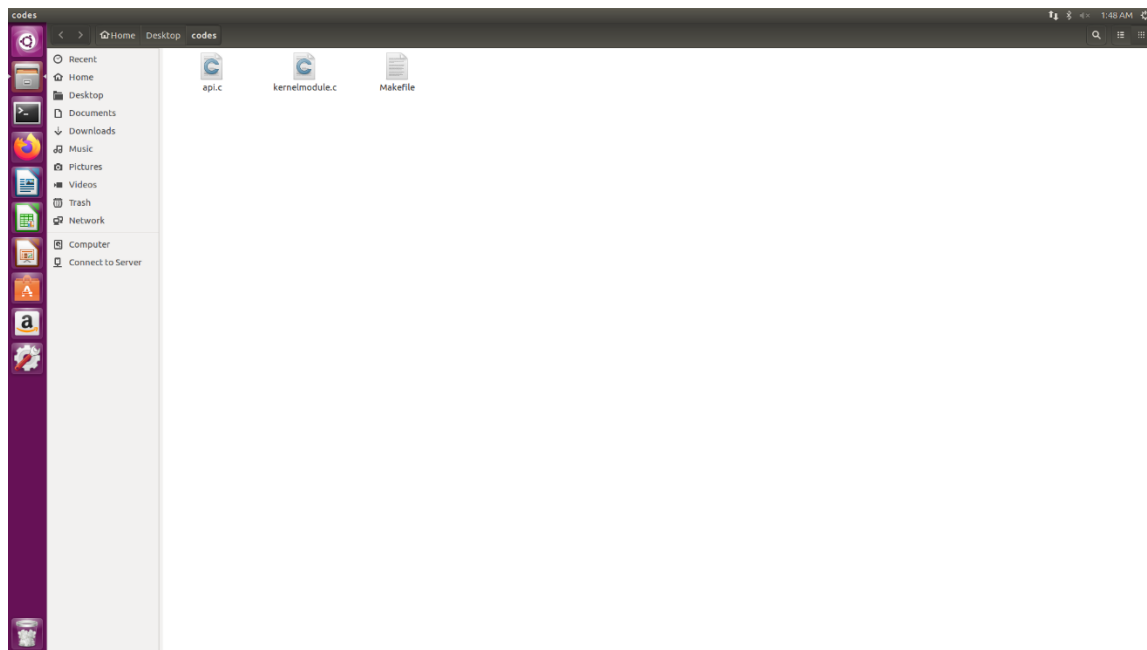
copy_to_user(buf, user_buff, count)

(عملکرد این تابع مشابه `copy_from_user` است که پیش تر توضیح داده شد)

کامپایل راه انداز

برای کامپایل کردن ماژول راه انداز، از آنجا که در کدها در سطح `kernel` تعریف می شوند، بر خلاف کدهایی که در سطح کاربر با دستور `gcc` کامپایل می شوند، باید یک `Makefile` بنویسیم و سپس با دستور `make` آن را کامپایل کنیم. `Makefile` غالباً نحوه ی کامپایل و لینک های برنامه را مشخص می کند. به عنوان `target` فایل `kernelmodule.o` در `Makefile` عنوان گردیده است.

ابتدا فایل های موجود در پوشه `codes` را قبل از اجرای دستور `make` می بینیم.

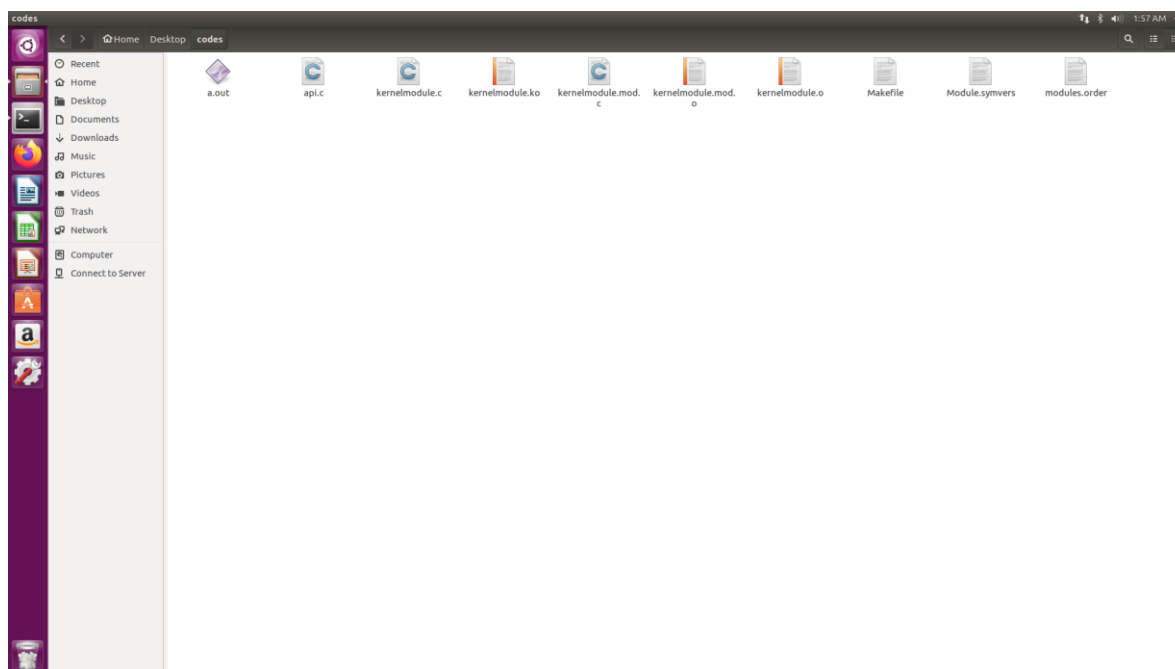


سپس دستور make را در ترمینال وارد می کنیم.

```
ninajan41@ubuntu: ~/Desktop/codes$ make
make -C /lib/modules/4.15.0-101-generic/build M=/home/ninajan41/Desktop/codes modules
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-101-generic'
CC [M] /home/ninajan41/Desktop/codes/kernelmodule.o
/home/ninajan41/Desktop/codes/kernelmodule.c:18:9: warning: "MODULE" redefined
#define MODULE
^
<command-line>:0:0: note: this is the location of the previous definition
/home/ninajan41/Desktop/codes/kernelmodule.c:20:8: warning: "__KERNEL__" redefined
#define __KERNEL__
^
<command-line>:0:0: note: this is the location of the previous definition
/home/ninajan41/Desktop/codes/kernelmodule.c: In function 'device_read':
/home/ninajan41/Desktop/codes/kernelmodule.c:43:5: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
int state = tsk->state;
^
/home/ninajan41/Desktop/codes/kernelmodule.c:62:5: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
char nvcsaRev[lenOfNvcsw];
^
/home/ninajan41/Desktop/codes/kernelmodule.c:71:5: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
int j = 0;
^
/home/ninajan41/Desktop/codes/kernelmodule.c:88:5: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
char states[len];
^
/home/ninajan41/Desktop/codes/kernelmodule.c:95:5: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
int cu = j;
^
/home/ninajan41/Desktop/codes/kernelmodule.c:108:5: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
struct files_struct *current_files;
^
/home/ninajan41/Desktop/codes/kernelmodule.c:117:5: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
size_t li;
^
/home/ninajan41/Desktop/codes/kernelmodule.c:110:9: warning: unused variable 'fds' [-Wunused-variable]
unsigned int *fds;
^
/home/ninajan41/Desktop/codes/kernelmodule.c:88:10: warning: unused variable 'states' [-Wunused-variable]
char states[len];
^
/home/ninajan41/Desktop/codes/kernelmodule.c:64:9: warning: unused variable 'k' [-Wunused-variable]
int k = 0;
^
/home/ninajan41/Desktop/codes/kernelmodule.c:38:9: warning: unused variable 'count' [-Wunused-variable]
int count = *offset;
^
/home/ninajan41/Desktop/codes/kernelmodule.c: In function 'device_write':
/home/ninajan41/Desktop/codes/kernelmodule.c:158:5: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
int x;
^
/home/ninajan41/Desktop/codes/kernelmodule.c: In function 'device_read':
/home/ninajan41/Desktop/codes/kernelmodule.c:135:1: warning: the frame size of 10048 bytes is larger than 1024 bytes [-Wframe-larger-than=]
}

Building modules, stage 2.
MODPOST 1 modules
CC /home/ninajan41/Desktop/codes/kernelmodule.mod.o
LD [M] /home/ninajan41/Desktop/codes/kernelmodule.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.15.0-101-generic'
ninajan41@ubuntu: ~/Desktop/codes$
```

و پس از آن فایل های مربوط به این دستور در پوشه make قابل مشاهده خواهند بود.



طراحی برنامه‌ی سطح کاربر

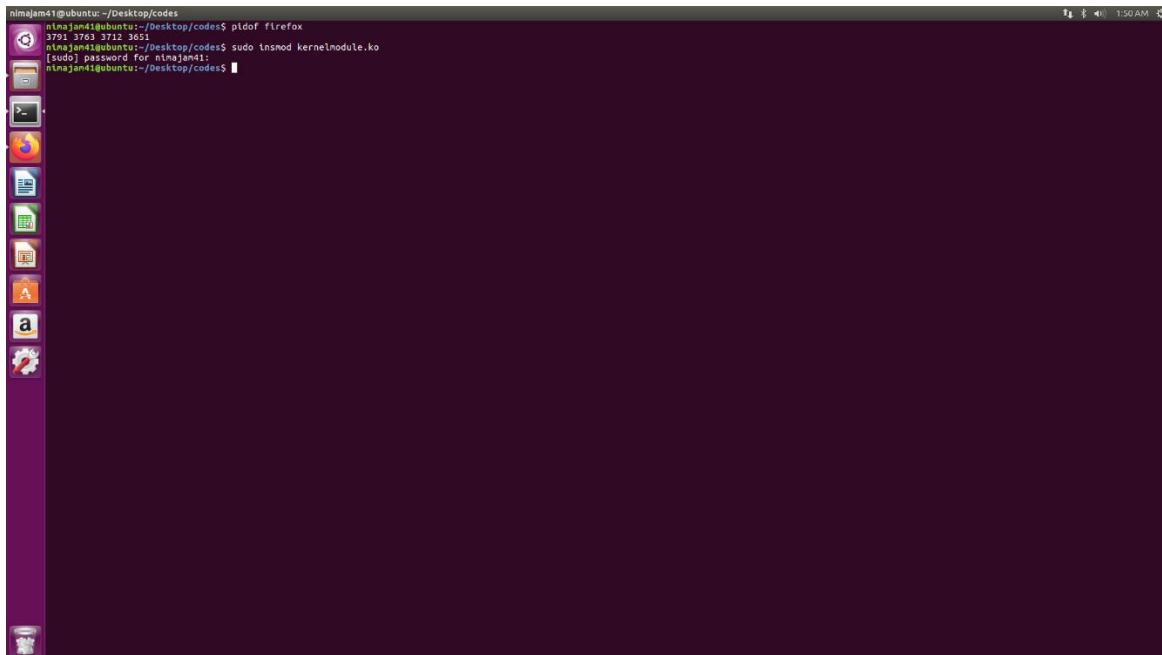
در برنامه‌ی سطح کاربر، ابتدا دو عدد PID و period ورودی گرفته می‌شوند، سپس device_file مورد استفاده یعنی first_phase باز می‌شود. پس از آن برای نوشتن PID در آن فایل باید آن را به صورت مجموعه‌ای از کاراکترها درآورد و برای این منظور، در یک حلقه ابتدا باقی‌مانده PID ورودی کاربر بر ۱۰ گرفته شده و در آرایه‌ای از کاراکترها ذخیره می‌شود (برای تبدیل به کاراکتر با '0' جمع می‌شود) و سپس مقدار PID به ۱۰ تقسیم شده و در مرحله بعدی حلقه استفاده می‌گردد. در نتیجه پس از اتمام حلقه، ارقام PID در آرایه‌ای از کاراکترها به نام pid ذخیره می‌شود. با توجه به آن که PID عددی حداکثر ۵ رقمی است، طول آرایه pid باید حداقل برابر با ۵ باشد که در این جا مقدار ۷ به صورت اختیاری برای طول آن انتخاب شده است. پس از این مرحله با فراخوانی تابع write، آرایه pid در first_phase نوشته می‌شود و می‌تواند توسط راه‌انداز مورد استفاده قرار بگیرد.

پس از آن باید در هر بازه‌ی زمانی به طول period ثانیه، اطلاعات مربوط به آن پردازش چاپ شود. در این کد فقط ۱۰ بازه‌ی زمانی اول خروجی داده می‌شود و با تغییر آن می‌توان تعداد بازه‌ها را بیشتر یا کمتر نمود. به این منظور در هر دور اجرای حلقه، period ثانیه sleep انجام داده و منتظر می‌مانیم. این کار با استفاده از دستور $usleep(period * 1000000)$ انجام می‌گیرد. (زیرا ورودی تابع usleep به میکروثانیه داده می‌شود)

در اولین بازه زمانی offset فایل را در نقطه‌ی $i + 10000$ قرار می‌دهیم که در آن i برابر با طول PID و 10000 نیز برابر با اندازه‌ی آرایه‌ی buf است که در آن خروجی را نگه می‌داریم. حال از این نقطه تابع read را صدا می‌زنیم و اطلاعات مربوطه در آرایه buf قرار می‌گیرند. آرایه buf را از ابتدا پیمایش کرده و تا زمانی که به کاراکتر '\0' برسیم، آنها را چاپ می‌کنیم. در نتیجه در هر بازه‌ی زمانی، عدد اول خروجی برابر با حالت پردازش (state)، عدد دوم برابر با nvcs و عدد سوم برابر با nivcs است و خطوط بعدی خروجی دربرگیرنده‌ی اطلاعات مربوط به فایل‌های پردازش مد نظر هستند. (توجه شود که با توجه به پرسش‌ها و پاسخ‌های موجود در quera، چاپ دو مورد از اطلاعات حسابداری کافی بوده که از میان آنها nvcs و nivcs انتخاب شده‌اند)

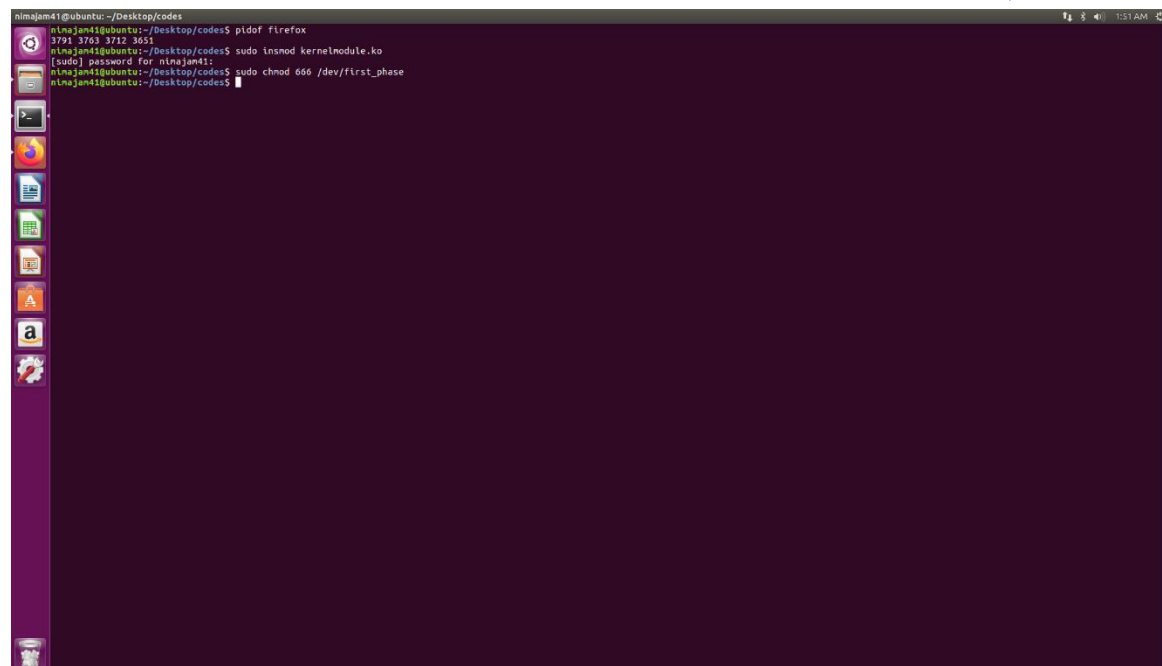
اجرای برنامه‌ی سطح کاربر

پس از طراحی برنامه‌ی سطح کاربر و `make` کردن راه‌انداز، می‌توان برنامه‌ی سطح کاربر را اجرا نمود. ابتدا با استفاده از دستور `sudo insmod kernelmodule.ko`، راه‌انداز اجرا شده و تابع `task_init` صدا زده می‌شود.

A terminal window with a dark purple background. The prompt is 'nimajam41@ubuntu: ~/Desktop/codes'. The user enters 'pidof firefox', which returns '3791 3703 3712 3651'. Then, the user enters 'sudo insmod kernelmodule.ko'. A password prompt '[sudo] password for nimajam41:' is shown, followed by the user's input. The prompt returns to 'nimajam41@ubuntu: ~/Desktop/codes\$'.

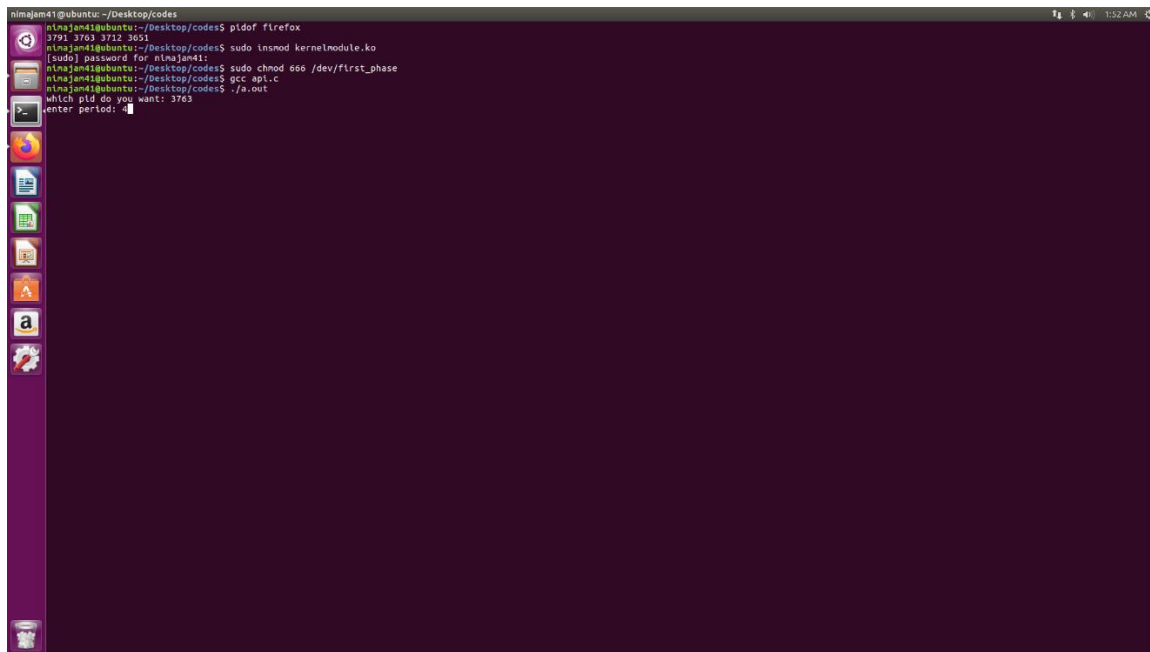
```
nimajam41@ubuntu: ~/Desktop/codes$ pidof firefox
nimajam41@ubuntu: ~/Desktop/codes$ sudo insmod kernelmodule.ko
[sudo] password for nimajam41:
nimajam41@ubuntu: ~/Desktop/codes$
```

برای آن که بتوان به پرونده‌ی ارتباطی دسترسی داشت، از `sudo chmod 666 /dev/first_phase` استفاده می‌کنیم.

A terminal window with a dark purple background. The prompt is 'nimajam41@ubuntu: ~/Desktop/codes'. The user enters 'pidof firefox', which returns '3791 3703 3712 3651'. Then, the user enters 'sudo insmod kernelmodule.ko'. A password prompt '[sudo] password for nimajam41:' is shown, followed by the user's input. Then, the user enters 'sudo chmod 666 /dev/first_phase'. The prompt returns to 'nimajam41@ubuntu: ~/Desktop/codes\$'.

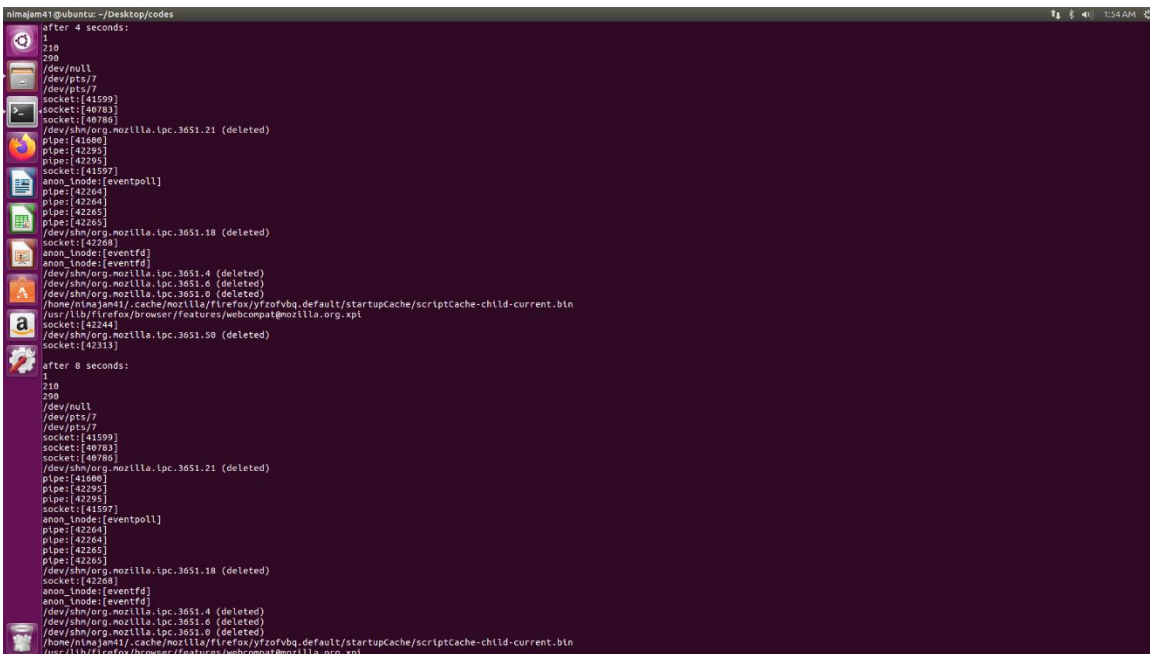
```
nimajam41@ubuntu: ~/Desktop/codes$ pidof firefox
nimajam41@ubuntu: ~/Desktop/codes$ sudo insmod kernelmodule.ko
[sudo] password for nimajam41:
nimajam41@ubuntu: ~/Desktop/codes$ sudo chmod 666 /dev/first_phase
nimajam41@ubuntu: ~/Desktop/codes$
```

پس از آن، نوبت به کامپایل و اجرای رابط کاربری می‌رسد. PID برابر با 3763 (مربوط به مرورگر firefox) و دوره‌ی زمانی برابر با ۴ ثانیه انتخاب شده است.



```
ninajam41@ubuntu: ~/Desktop/codes
ninajam41@ubuntu:~/Desktop/codes$ pldof firefox
3791 3763 3712 3651
ninajam41@ubuntu:~/Desktop/codes$ sudo insmod kernelmodule.ko
[sudo] password for ninajam41:
ninajam41@ubuntu:~/Desktop/codes$ sudo chmod 666 /dev/first_phase
ninajam41@ubuntu:~/Desktop/codes$ gcc api.c
ninajam41@ubuntu:~/Desktop/codes$ ./a.out
which pid do you want: 3763
enter period: 
```

در خروجی، در بازه‌های زمانی ۴ ثانیه‌ای، اطلاعات مربوط به پردازش چاپ می‌شود. در هر بازه در خط اول حالت پردازش، خط دوم nvcs، خط سوم nivcs و در باقی خطوط اطلاعات مربوط به آدرس فایل‌ها چاپ می‌شود. دو تصویر به صورت نمونه از خروجی داده شده است.



```
ninajam41@ubuntu: ~/Desktop/codes
after 4 seconds:
1
210
290
/dev/null
/dev/pts/7
/dev/pts/7
socket:[41599]
socket:[40783]
socket:[40786]
/dev/shm/org.mozilla.ipc.3651.21 (deleted)
pipe:[41600]
pipe:[42295]
pipe:[42295]
socket:[41597]
anon_inode:[eventpoll]
pipe:[42264]
pipe:[42264]
pipe:[42265]
pipe:[42265]
/dev/shm/org.mozilla.ipc.3651.18 (deleted)
socket:[42268]
anon_inode:[eventfd]
anon_inode:[eventfd]
/dev/shm/org.mozilla.ipc.3651.4 (deleted)
/dev/shm/org.mozilla.ipc.3651.6 (deleted)
/dev/shm/org.mozilla.ipc.3651.0 (deleted)
/home/ninajam41/.cache/mozilla/firefox/yfzofvbg.default/startupCache/scriptcache-child-current.bin
/usr/lib/firefox/browser/features/webcompat@mozilla.org.spt
socket:[42244]
/dev/shm/org.mozilla.ipc.3651.50 (deleted)
socket:[42313]

after 8 seconds:
1
210
290
/dev/null
/dev/pts/7
/dev/pts/7
socket:[41599]
socket:[40783]
socket:[40786]
/dev/shm/org.mozilla.ipc.3651.21 (deleted)
pipe:[41600]
pipe:[42295]
pipe:[42295]
socket:[41597]
anon_inode:[eventpoll]
pipe:[42264]
pipe:[42264]
pipe:[42265]
pipe:[42265]
/dev/shm/org.mozilla.ipc.3651.18 (deleted)
socket:[42268]
anon_inode:[eventfd]
anon_inode:[eventfd]
/dev/shm/org.mozilla.ipc.3651.4 (deleted)
/dev/shm/org.mozilla.ipc.3651.6 (deleted)
/dev/shm/org.mozilla.ipc.3651.0 (deleted)
/home/ninajam41/.cache/mozilla/firefox/yfzofvbg.default/startupCache/scriptcache-child-current.bin
/usr/lib/firefox/browser/features/webcompat@mozilla.org.spt
```

```
nimaJan41@ubuntu: ~/Desktop/codes
after 36 seconds:
1
228
290
/dev/null
/dev/pts/7
/dev/pts/7
socket:[41599]
socket:[40783]
socket:[40786]
/dev/shm/org.mozilla.lpc.3651.21 (deleted)
pipe:[41600]
pipe:[42295]
pipe:[42295]
socket:[41597]
anon_inode:[eventpoll]
pipe:[42264]
pipe:[42264]
pipe:[42265]
pipe:[42265]
/dev/shm/org.mozilla.lpc.3651.18 (deleted)
socket:[42268]
anon_inode:[eventfd]
anon_inode:[eventfd]
/dev/shm/org.mozilla.lpc.3651.4 (deleted)
/dev/shm/org.mozilla.lpc.3651.0 (deleted)
/dev/shm/org.mozilla.lpc.3651.0 (deleted)
/home/nimaJan41/.cache/mozilla/ffzofvbq.default/startupCache/scriptCache-chlid-current.bin
/usr/lib/ffmpeg/browser/features/webcompat@mozilla.org.xpi
socket:[42244]
/dev/shm/org.mozilla.lpc.3651.50 (deleted)
socket:[42313]

after 40 seconds:
1
231
290
/dev/null
/dev/pts/7
/dev/pts/7
socket:[41599]
socket:[40783]
socket:[40786]
/dev/shm/org.mozilla.lpc.3651.21 (deleted)
pipe:[41600]
pipe:[42295]
socket:[41597]
anon_inode:[eventpoll]
pipe:[42264]
pipe:[42264]
pipe:[42265]
pipe:[42265]
/dev/shm/org.mozilla.lpc.3651.18 (deleted)
socket:[42268]
anon_inode:[eventfd]
anon_inode:[eventfd]
/dev/shm/org.mozilla.lpc.3651.4 (deleted)
/dev/shm/org.mozilla.lpc.3651.0 (deleted)
/dev/shm/org.mozilla.lpc.3651.0 (deleted)
/home/nimaJan41/.cache/mozilla/ffzofvbq.default/startupCache/scriptCache-chlid-current.bin
/usr/lib/ffmpeg/browser/features/webcompat@mozilla.org.xpi
```

و پس از اجرای برنامه، با استفاده از دستور `sudo rmmod kernelmodule.ko` تابع `task_exit` صدا زده شده و پرونده‌ی ارتباطی نیز از بین می‌رود.

```
nimaJan41@ubuntu: ~/Desktop/codes
nimaJan41@ubuntu:~/Desktop/codes$ sudo rmmod kernelmodule.ko
[sudo] password for nimaJan41:
nimaJan41@ubuntu:~/Desktop/codes$
```

<https://stackoverflow.com/questions/9047950/code-for-writing-and-reading-on-a-device-file-from-a-kernel-module>
<https://stackoverflow.com/questions/8547332/efficient-way-to-find-task-struct-by-pid>
<https://tuxthink.blogspot.com/2012/05/module-to-print-open-files-of-process.html>