**گزارش دستورکار دوم آزمایشگاه سیستم­های عامل**

نگار موقتیان، 9831062

1. **بارگذاری ماژول هستۀ سخت افزاری:**

در این قسمت از آزمایش ماژول هستۀ nvmem-rave-sp-eeprom که یک EEPROM درایور برای Rave SP می­باشد را بر روی هستۀ سیستم عامل بارگذاری کرده و سپس آن را حذف می­کنیم.

برای این کار ابتدا فایل با پسوند .ko مربوط به این ماژول هسته را از سایت زیر دانلود کردم:

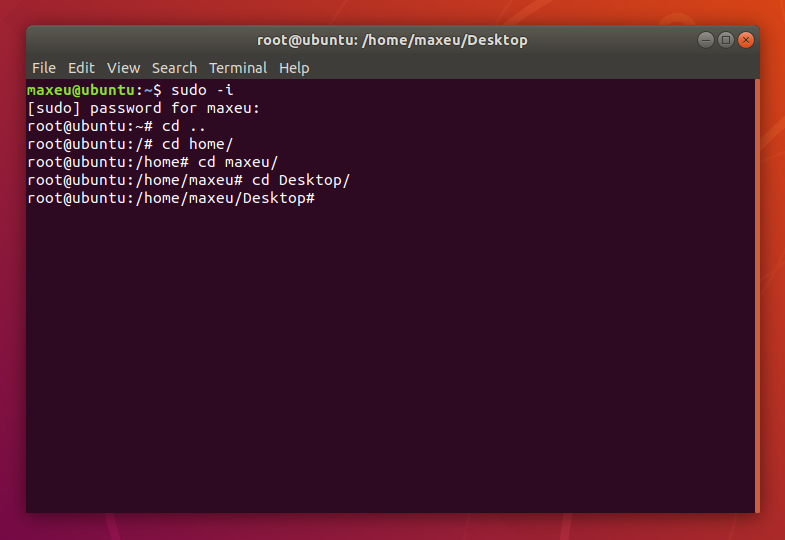
https://www.candelatech.com/downloads/j/lib/modules/5.15.17+/kernel/drivers/nvmem/

اما متاسفانه با وجود استفاده از نسخۀ 18.04 سیستم عامل Ubuntu با خطای Invalid module format مواجه شدم. لذا فایل .c این ماژول را از ریپازیتوری زیر دانلود کرده و آن را make کردم:

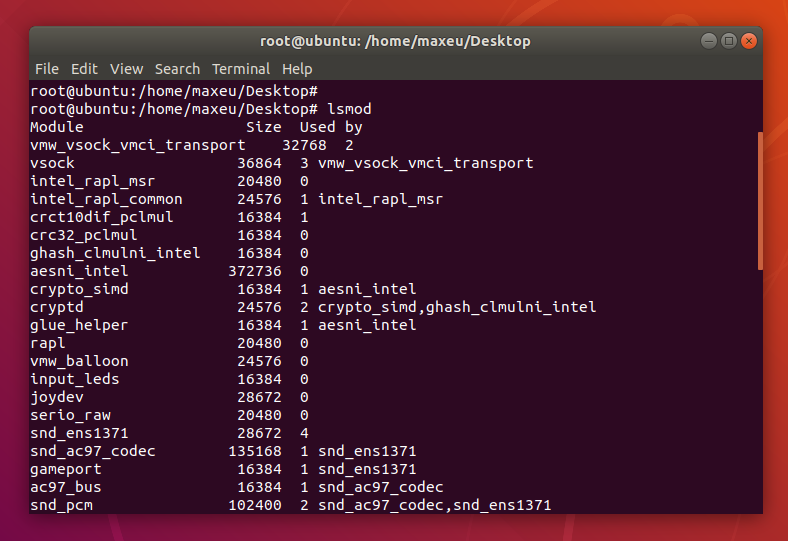
<https://github.com/torvalds/linux/blob/master/drivers/nvmem/rave-sp-eeprom.c>

و این بار در بارگذاری فایل خروجی .ko مرحلۀ قبل بر روی هسته مشکلی وجود نداشت (طریقۀ make کردن فایل­های زبان C در قسمت­های بعدی آزمایش به طور کامل توضیح داده خواهد شد).

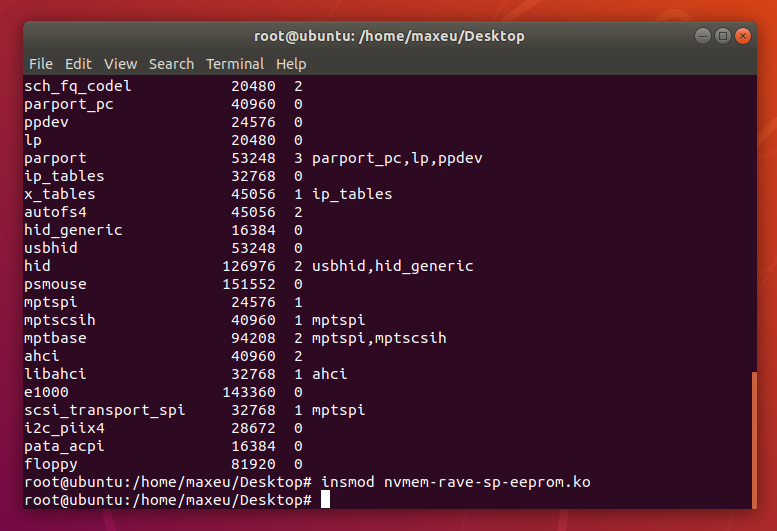
پس از بدست آوردن فایل .ko منطبق با هستۀ سیستم عامل مراحل زیر انجام شد.



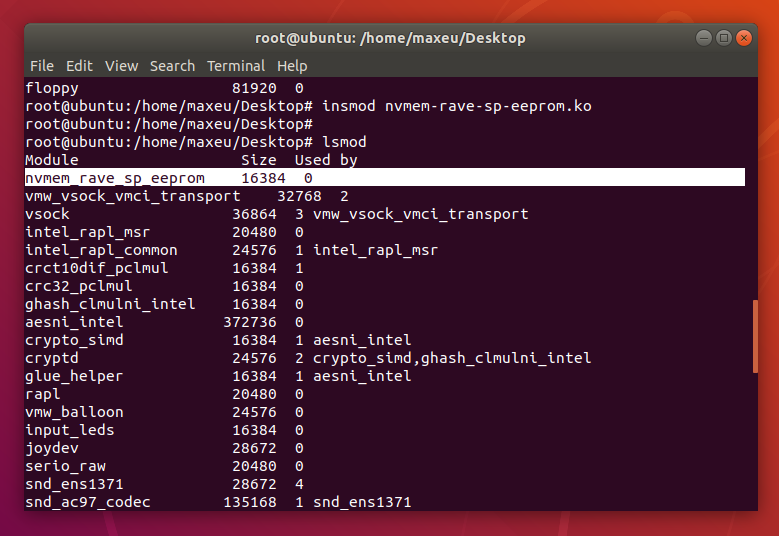
به دلیل این که می­خواهیم ماژولی را بر روی هستۀ سیستم عامل بارگذاری کنیم نیاز به دسترسی root داریم. بنابراین در این مرحله با استفاده از دستور sudo –i سطح دسترسی را به root تغییر داده و به فولدر Desktop (که فایل nvmem-rave-sp-eeprom.ko در آن قرار دارد) می­رویم.



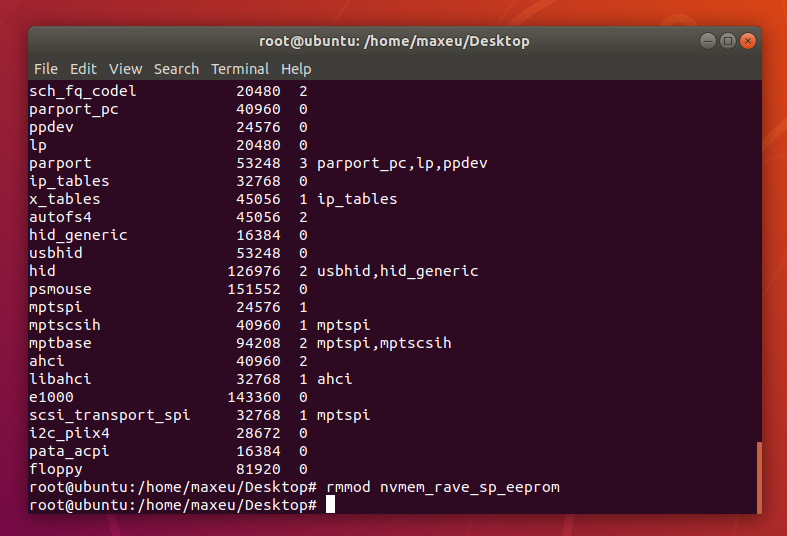
حال ابتدا با استفاده از دستور lsmod لیست ماژول­هایی که در حال حاضر بر روی هستۀ سیستم هستند را مشاهده می کنیم.



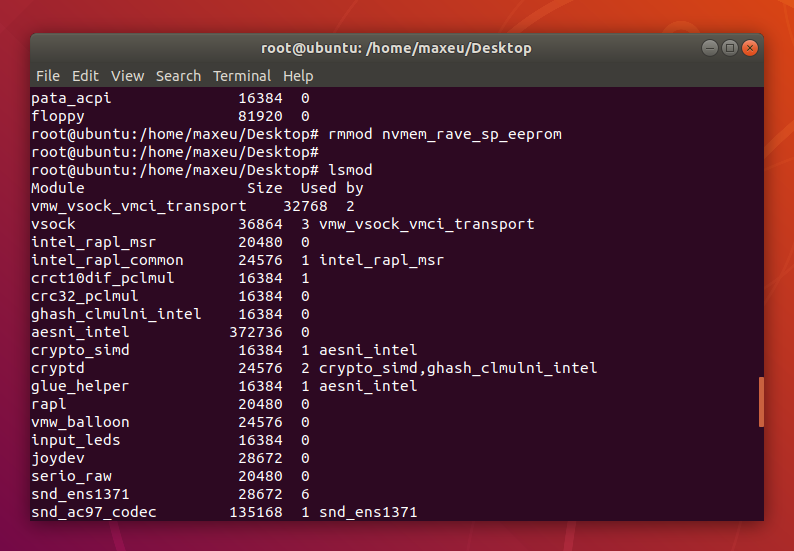
در این مرحله به سادگی با استفاده از دستور insmod nvmem-rave-sp-eeprom.ko ماژول مورد نظر را بر روی هستۀ سیستم عامل باگذاری می­کنیم.



بار دیگر دستور lsmod را اجرا می­کنیم. مشاهده می­شود که ماژول فوق به لیست ماژول­های هسته اضافه شده­است.



برای حذف این ماژول نیز از دستور rmmod (بر خلاف insmod) استفاده می­کنیم.



با اجرای دوبارۀ دستور lsmod مطمئن می­شویم که ماژول فوق با موفقیت حذف شده و دیگر میان لیست ماژول­های هسته وجود ندارد.

1. **بخش 1 – ایجاد ماژول­های هسته:**

در این قسمت از آزمایش ابتدا یک ماژول هسته به زبان C و مطابق با دستورکار می­نویسیم. سپس این ماژول را make می­کنیم تا از فایل C نوشته شده یک خروجی قابل بارگذاری بر هستۀ سیستم عامل با پسوند .ko ایجاد شود. سپس مانند قسمت قبلی آزمایش این فایل .ko را بر روی هسته بارگذاری کرده و در انتهای آزمایش آن را حذف می­کنیم. همچنین در این مراحل با استفاده از دستور dmesg بافر هسته را بررسی می­کنیم تا از بارگذاری و حذف صحیح ماژول فوق اطمینان حاصل کنیم.

کد مربوط به این ماژول در ادامه آمده­ و توضیحات مربوطه داده شده است.

#include <linux/init.h>

#include <linux/kernel.h>

#include <linux/module.h>

/\* this function is called when the module is loaded \*/

int simple\_init(void) {

printk(KERN\_INFO "Loading Module...\n");

return 0;

}

/\* this function is called when the module is removed \*/

void simple\_exit(void) {

printk(KERN\_INFO "Removing Module...\n");

}

/\* Macros for registering module entry and exit points \*/

module\_init(simple\_init);

module\_exit(simple\_exit);

MODULE\_LICENSE("GPL");

MODULE\_DESCRIPTION("A simple module written to learn about Linux kernel modules");

MODULE\_AUTHOR("Negar Movaghatian - 9831062");

در خطوط ابتدایی کتابخانه­های لازم برای ایجاد ماژول هسته را اضافه کرده­ایم.

توابع simple\_init و simple\_exit نقاط شروع و پایان برنامۀ ماژول هستند که با استفاده از ماکروهای moduel\_init و module\_exit که در خطوط پایانی برنامه نوشته شده­اند این دو تابع را به عنوان تابع شروع و پایان معرفی می­کنیم. در این ماژول ساده تنها کاری که این دو تابع انجام می­دهند نوشتن پیغامی هنگام بارگذاری و حذف ماژول بر روی بافر هسته است تا از طریق آن بتوانیم از موفقیت­آمیز بودن روند کار اطمینان حاصل کنیم.

در انتهای برنامه نیز توضیحاتی دربارۀ ماژول فوق را در قالب ماکروهای از پیش تعریف شده نوشته­ایم.

و اما برای بارگذاری این ماژول بر روی هسته نیاز به یک فایل خروجی با پسوند .ko داریم. برای build کردن برنامه از یک makefile و دستور make استفاده برای اجرای آن استفاده می­کنیم. برای استفاده از این دستور و کامپایل کردن برنامۀ نوشته شده به زبان C ابتدا با استفاده از دستورات زیر make و gcc را بر روی سیستم عامل نصب می­کنیم:

apt get update

apt install make

apt install gcc

به علاوه از یک makefile با محتوای زیر استفاده می­کنیم. این فایل را در فولدر برنامۀ نوشته شده قرار می­دهیم تا make بتواند دستورالعمل­های آن را اجرا کند.

KERNELDIR=/lib/modules/`uname -r`/build

#ARCH=i386

#KERNELDIR=/usr/src/kernels/`uname -r`-i686

MODULES = simple-module.ko

obj-m += simple-module.o

all:

make -C $(KERNELDIR) M=$(PWD) modules

clean:

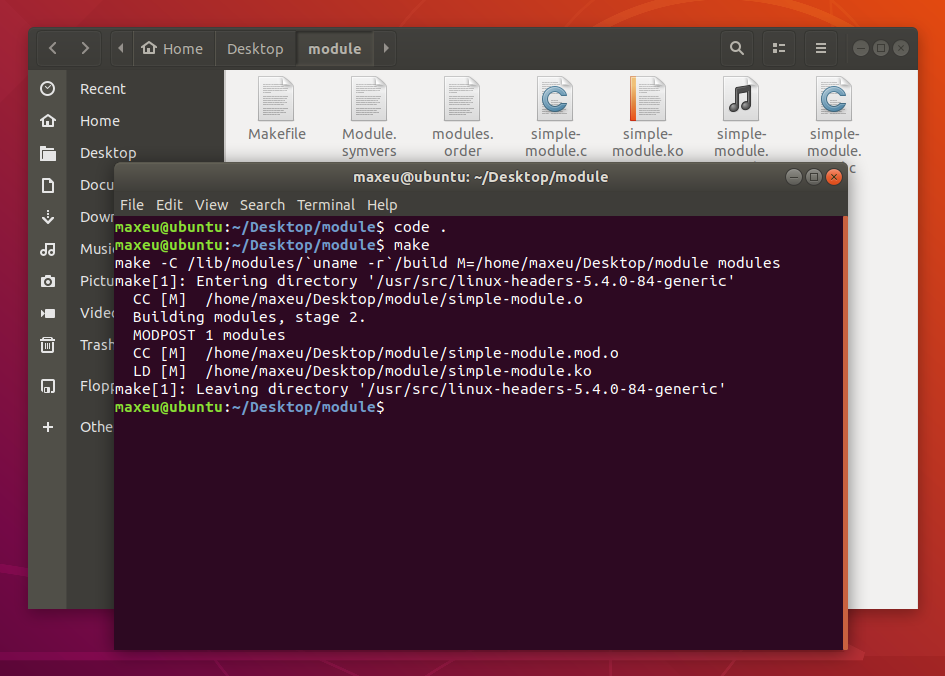
make -C $(KERNELDIR) M=$(PWD) clean

برای مشخص کردن محتوای این فایل از سایت زیر استفاده شده­است:

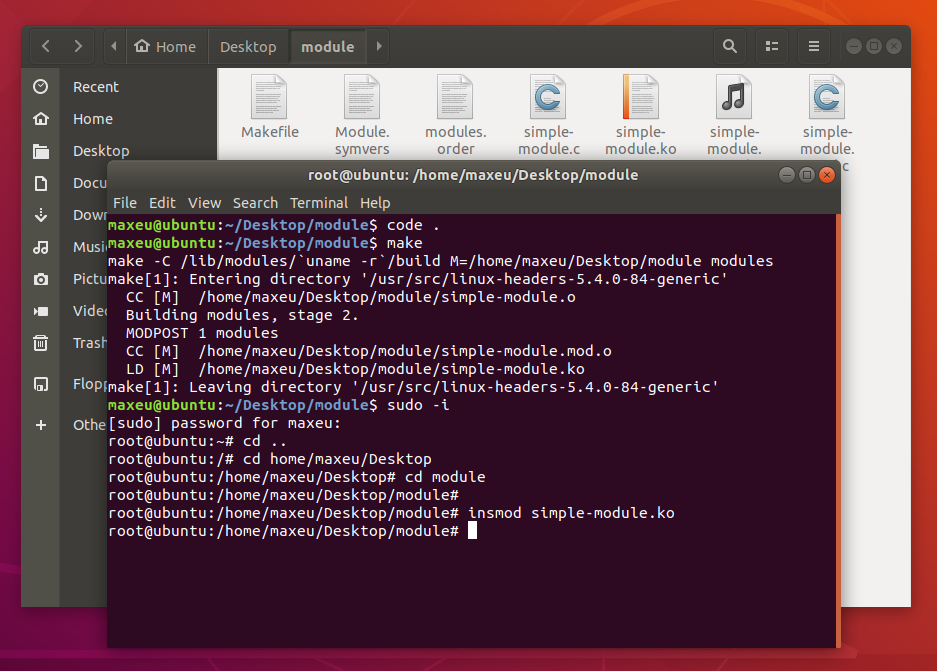
https://www.cs.bham.ac.uk/~exr/lectures/systems/07\_08/kernelProgramming.php

برای جلوگیری از خطای Invalid module format و تطابق ماژول build شده با نسخۀ kernel مسیر هستۀ سیستم عامل (تحت عنوان KERNELDIR) به عنوان آرگومان به make داده شده­است. در خط 4 و 5 نیز نام فایل­هایی خروجی ماژول (با پسوند .ko) و object ها (با پسوند .o) مشخص شده­اند.

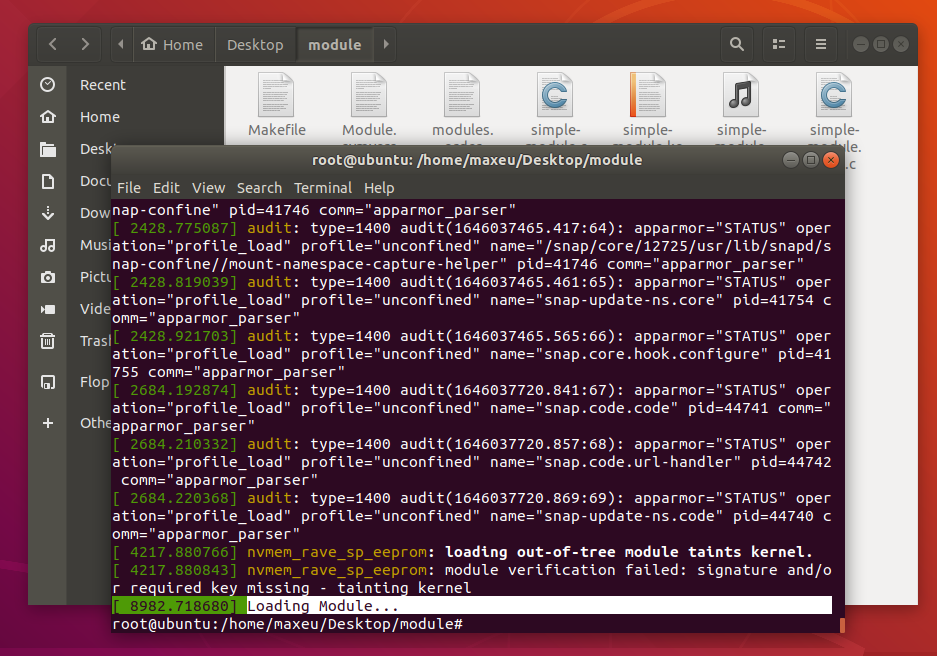
با قرار دادن این فایل در فولدر برنامه­ای که نوشته­ایم و اجرای دستور make می­توان فایل­های خروجی را بدست آورد.



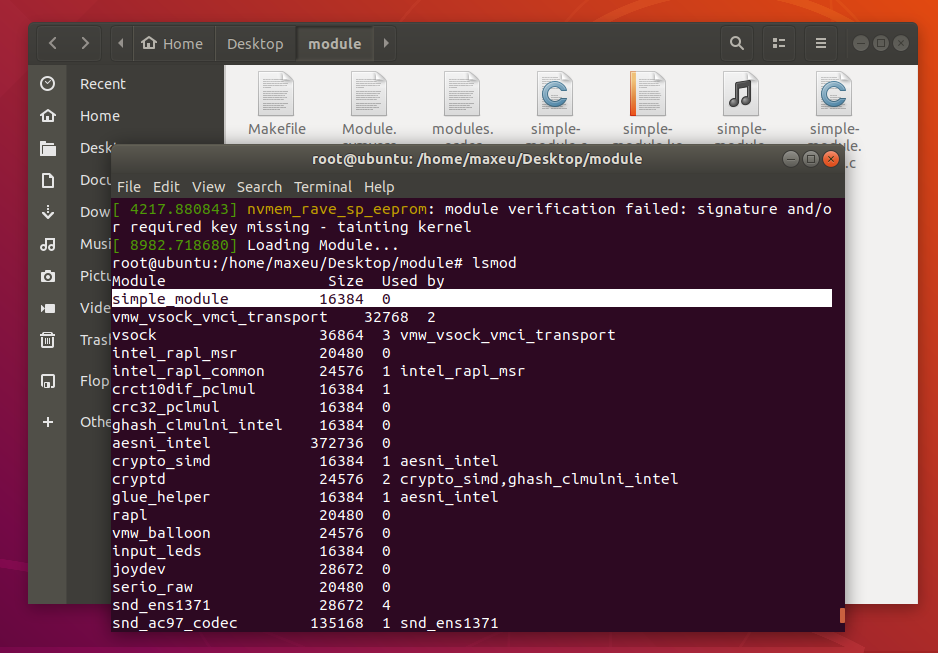
پس از طی این مراحل و بدست آوردن فایل خروجی کافیست مانند بخش اول آزمایش سطح دسترسی را به root تغییر داده و دستور insmod را اجرا کنیم.



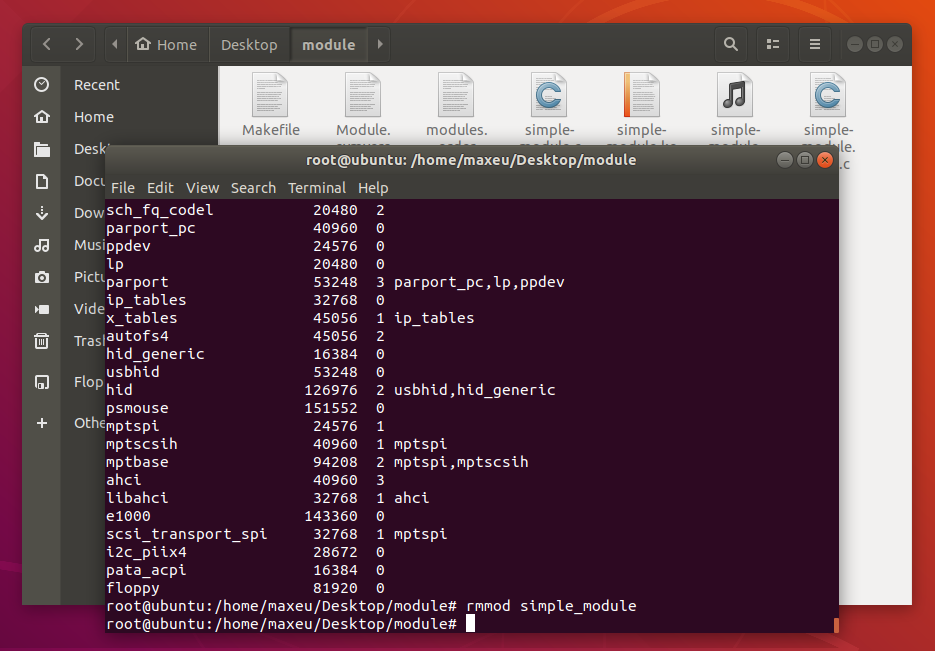
طبق دستورکار با استفاده از دستور dmesg بافر هسته را چک می­کنیم تا بررسی کنیم پیغامی که انتظار داشتیم زمان بارگذاری ماژول هسته چاپ شود در بافر نوشته شده یا خیر.



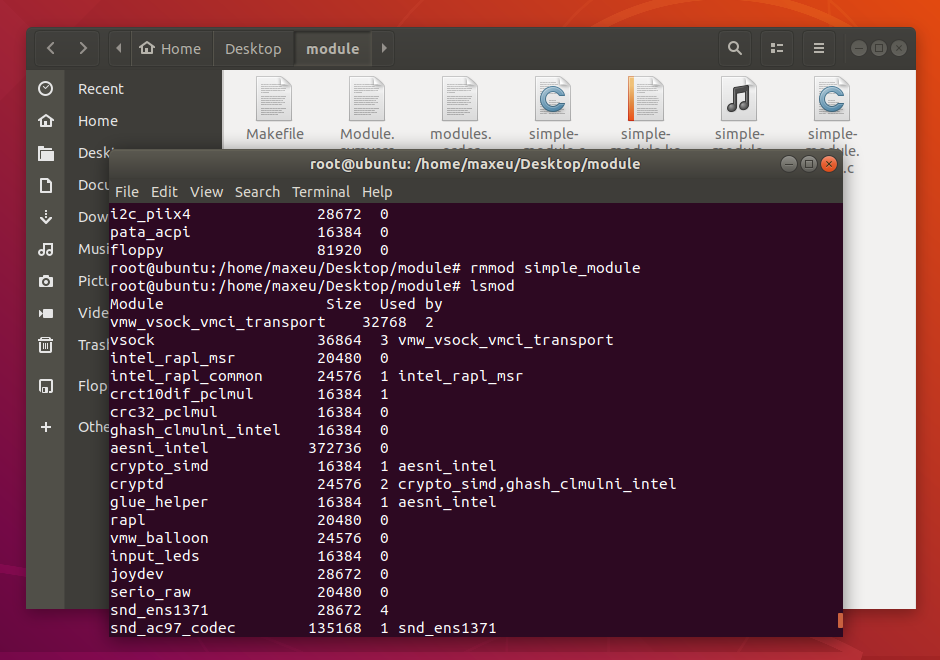
همانطور که انتظار داشتیم عبارت “Loading Module…” چاپ شده­است. به علاوه با استفاده از دستور lsmod لیست ماژول­های هسته را بررسی می­کنیم.



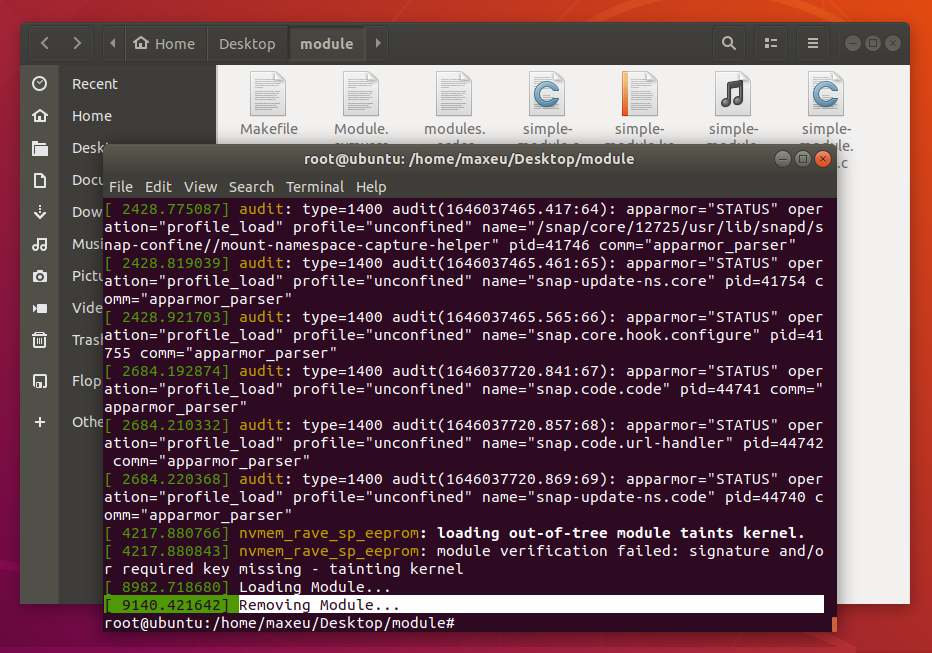
ماژول simple\_module در این لیست قرار دارد، بنابراین ماژول هسته­ای که نوشته بودیم با موفقیت بارگذاری شده­است. حال مانند قسمت اول آزمایش ماژول نوشته شده را با استفاده از دستور rmmod حذف می­کنیم.



بار دیگر از دستور lsmod استفاده می­کنیم. این بار ماژول مورد نظر در لیست ماژول­های هسته وجود ندارد.



و با استفاده از دستور dmesg بافر هسته را چک می­کنیم تا بررسی کنیم پیغامی که انتظار داشتیم زمان حذف ماژول هسته چاپ شود در بافر نوشته شده یا خیر. عبارت “Removing Module…” در بافر چاپ شده­است، بنابراین حذف ماژول نیز با موفقیت انجام شده.



1. **بخش 2 – ساختمان داده­های هسته:**

در این قسمت نیز مانند قسمت قبل آزمایش با استفاده از زبان C یک ماژول هسته نوشته، آن را بر روی هسته بارگذاری کرده و سپس حذف می­کنیم. کد مربوط به هر بخش از این ماژول در ادامه آمده­ و توضیحات مربوطه داده شده است.

#include <linux/init.h>

#include <linux/kernel.h>

#include <linux/module.h>

#include <linux/list.h>

#include <linux/slab.h>

در ابتدا کتابخانه­های مورد نیاز را به ماژول مورد نظر اضافه می­کنیم. سه کتابخانۀ اول برای نوشتن ماژول­های هسته، کتابخانۀ بعدی برای استفاده از ساختمان دادۀ Linked List موجود در هستۀ لینوکس و کتابخانۀ آخر برای استفاده از دستور kmalloc برای تخصیص حافظۀ هسته استفاده می­شود.

static LIST\_HEAD(birthday\_list);

struct birthday {

int day;

int month;

int year;

struct list\_head list;

};

در خطوط بعدی برنامه ابتدا با استفاده از ماکروی LIST\_HEAD متغیر birthday\_list را به عنوان ابتدای linked list ای که در ادامه از آن استفاده می­کنیم تعریف و مقداردهی می­کنیم. سپس یک struct به نام birthday می­سازیم که یک تاریخ را در خود نگهداری کرده و به دلیل داشتن یک عضو از جنس list\_head قابلیت استفاده به عنوان یک node در linked list هسته را دارد.

در تابع simple\_init که در صفحۀ بعد آمده (و در انتهای برنامه آن را به عنوان نقطۀ شروع ماژول معرفی می­کنیم)، ابتدا یک پیغام بر روی بافر هسته قرار می­دهیم. زمان بارگذاری ماژول هسته این پیغام به ما کمک می­کند تا مطمئن باشیم ماژول به درستی بر روی هستۀ سیستم عامل بارگذاری شده­است. در ادامه 5 نمونه از  
struct birthday را ساخته، به آن حافظه تخصیص داده، فیلدهای آن را مقداردهی کرده و در آخر آن را به انتهای linked list ای که ساخته بودیم اضافه می­کنیم. پس از اضافه شدن تمام افراد با استفاده از دستور list\_for\_each\_entry بر روی تمام اعضاء linked list ساخته شده پیمایش کرده و اطلاعات مربوط به آن را چاپ می­کنیم. در انتها نیز پیغامی بر روی بافر هسته چاپ می­کنیم تا مطمئن شویم تمام node ها با موفقیت اضافه شده­اند.

/\* this function is called when the module is loaded \*/

int simple\_init(void) {

printk(KERN\_INFO "Loading Birthday Module...\n");

struct birthday \*person1;

person1 = kmalloc(sizeof(person1), GFP\_KERNEL);

person1->day = 12;

person1->month = 10;

person1->year = 2001;

INIT\_LIST\_HEAD(&person1->list);

list\_add\_tail(&person1->list, &birthday\_list);

.

.

.

struct birthday \*ptr;

int i = 1;

list\_for\_each\_entry(ptr, &birthday\_list, list)

printk(KERN\_INFO "%d. %d/%d/%d\n", i++, ptr->year, ptr->month, ptr->day);

printk(KERN\_INFO "End of init function. Added all nodes successfully!\n");

return 0;

}

در تابع simple\_exit نیز (که در انتهای برنامه آن را به عنوان نقطۀ پایان ماژول معرفی می­کنیم)، ابتدا یک پیغام بر روی بافر هسته قرار می­دهیم. زمان حذف ماژول هسته این پیغام به ما کمک می­کند تا مطمئن باشیم ماژول به درستی از روی هستۀ سیستم عامل حذف شده­است.

/\* this function is called when the module is removed \*/

void simple\_exit(void) {

printk(KERN\_INFO "Removing Birthday Module...\n");

int i = 1;

struct list\_head \*ptr, \*tmp;

list\_for\_each\_safe(ptr, tmp, &birthday\_list){

printk(KERN\_INFO "Freeing node %d\n", i++);

list\_del(ptr);

kfree(ptr);

}

printk(KERN\_INFO "End of exit function. removed all nodes successfully!\n");

}

به دلیل این که زمانی که این تابع در حال اجرا است می­خواهیم ماژول را از روی هسته برداریم باید حافظۀ هسته که در تابع simple\_init به متغیرهای این ماژول اختصاص داده بودیم را به هسته برگردانیم، بنابراین نیاز داریم که بر روی تمام عناصر linked list پیمایش شده حرکت کنیم، آن­ها را از linked list حذف کرده و با استفاده از دستور kfree حافظۀ تخصیص داده شده به آن را آزاد کنیم. در این زمان به ازای هر node پیغامی نیز بر روی بافر هسته چاپ می­کنیم تا مطمئن شویم تمام node ها حذف شده­اند.

اما برای پیمایش بر روی تمام اعضاء این بار از دستور list\_for\_each\_safe استفاده می­کنیم، زیرا می­خواهیم همزمان با پیمایش لیست node هایی را از آن حذف کنیم و استفاده از for\_each ساده می­تواند باعث مشکلاتی شود. برای فراگیری طریقۀ استفاده از این دستور از سایت­­های زیر استفاده شده­است:

https://stackoverflow.com/questions/9207850/why-do-we-need-list-for-each-safe-in-for-deleting-nodes-in-kernel-linked-list

<https://www.kernel.org/doc/htmldocs/kernel-api/API-list-for-each-safe.html>

/\* Macros for registering module entry and exit points \*/

module\_init(simple\_init);

module\_exit(simple\_exit);

MODULE\_LICENSE("GPL");

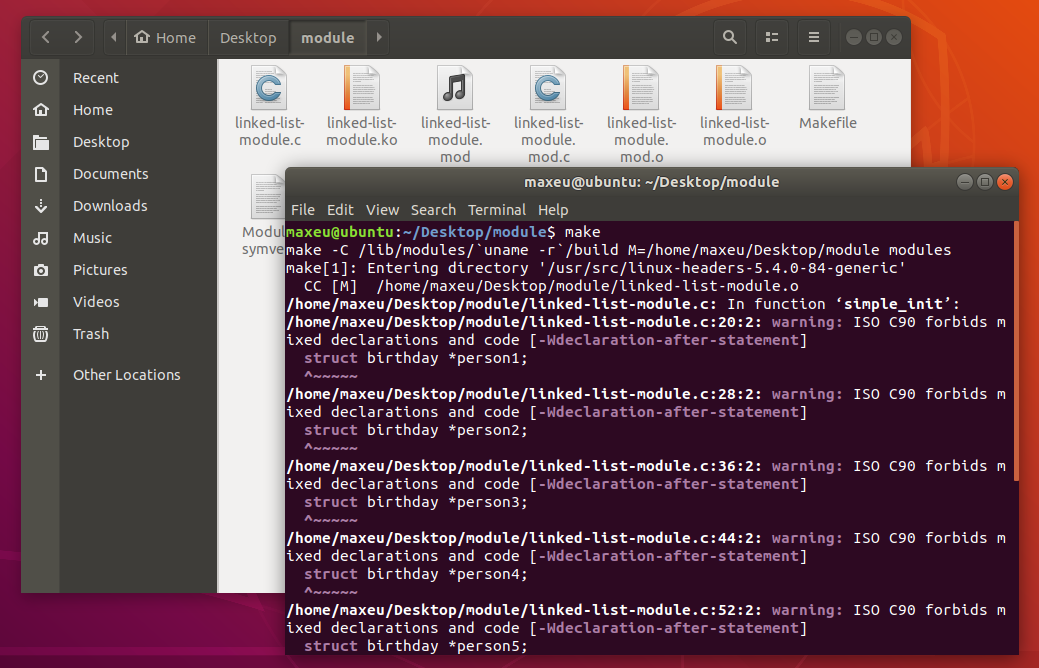
MODULE\_DESCRIPTION("A module written to work with Linux kernel data structures");

MODULE\_AUTHOR("Negar Movaghatian - 9831062");

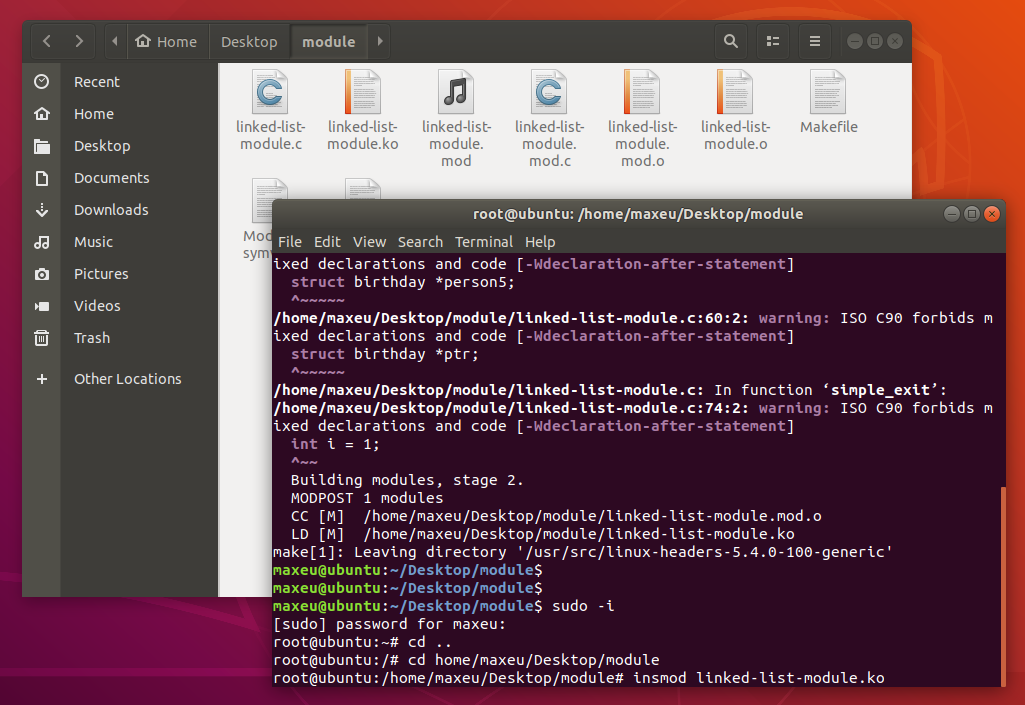
در انتها نیز با استفاده از ماکروهای moduel\_init و module\_exit دو تابع simple­\_init و simple\_exit را به عنوان توابع شروع و پایان ماژول معرفی می­کنیم. به علاوه توضیحاتی دربارۀ ماژول فوق را در قالب ماکروهای از پیش تعریف شده اضافه­ می­کنیم.

حال برای بارگذاری این ماژول بر روی هسته نیاز به یک فایل خروجی با پسوند .ko داریم. برای build کردن برنامه مانند آزمایش قبل از یک makefile و دستور make استفاده برای اجرای آن استفاده می­کنیم. توضیحات مربوط به این فایل در آزمایش قبل آمده­است. تنها کافیست نام فایل­های خروجی را به نام این ماژول تغییر دهیم.

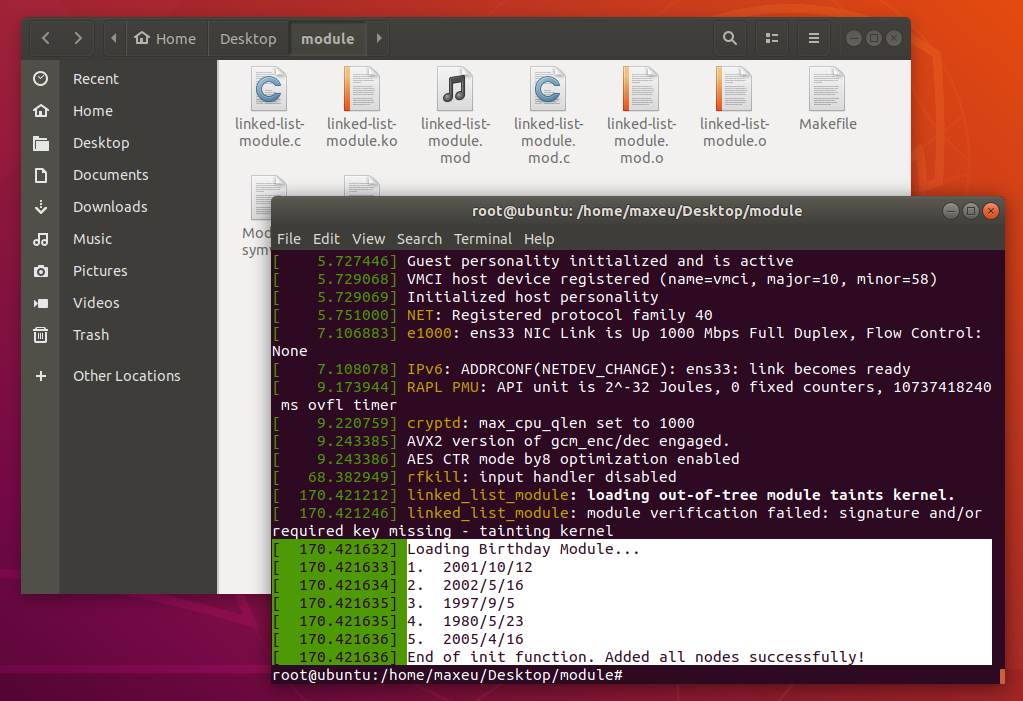
با قرار دادن این فایل در فولدر برنامه­ای که نوشته­ایم و اجرای دستور make می­توان فایل­های خروجی را بدست آورد.



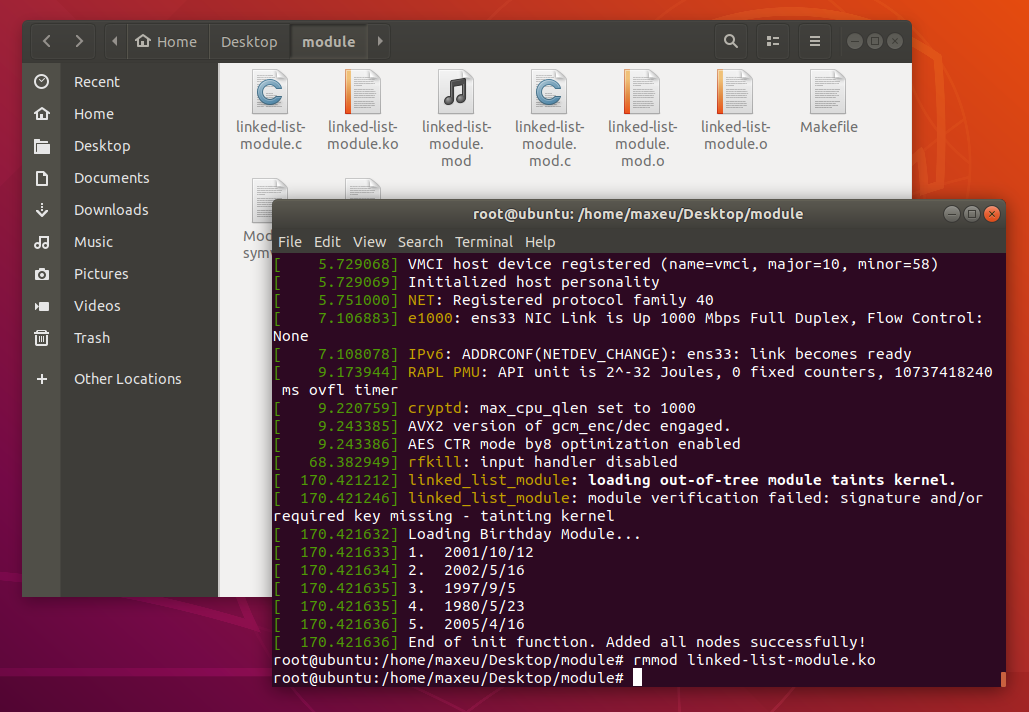
پس از طی این مراحل و بدست آوردن فایل خروجی کافیست مانند بخش­های قبلی آزمایش سطح دسترسی را به root تغییر داده و دستور insmod را اجرا کنیم.



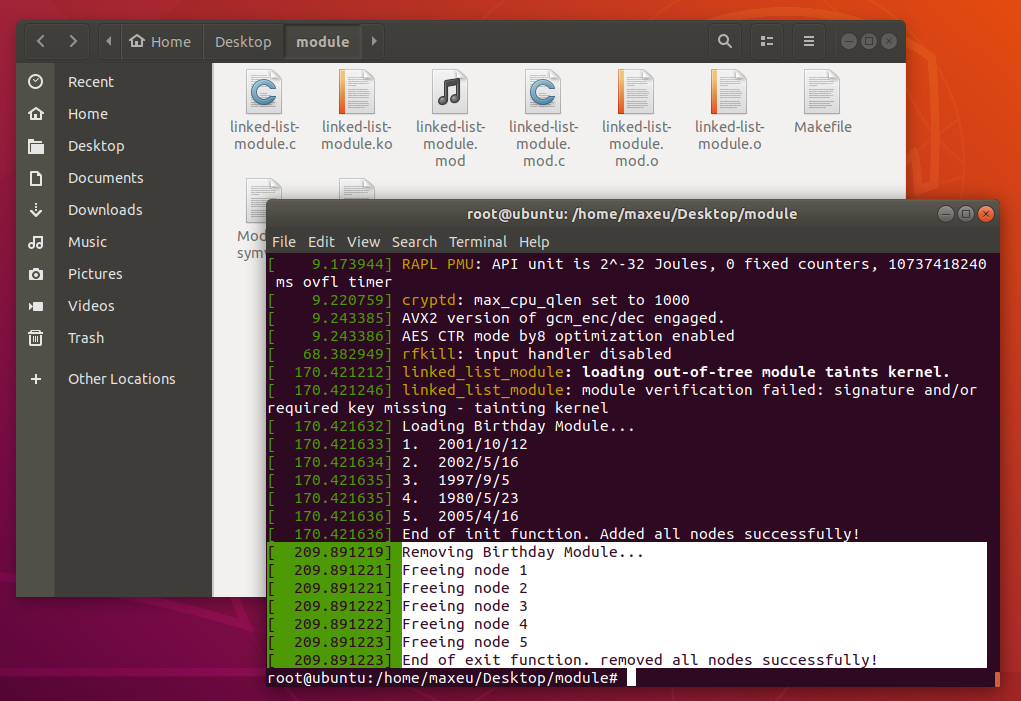
با استفاده از دستور dmesg بافر هسته را چک می­کنیم تا بررسی کنیم پیغامی که انتظار داشتیم زمان بارگذاری ماژول هسته چاپ شود در بافر نوشته شده یا خیر.



همانطور که انتظار داشتیم عبارت “Loading Birthday Module…” و لیست تمامی node ها چاپ شده­است. حال مانند قسمت­های قبلی آزمایش ماژول نوشته شده را با استفاده از دستور rmmod حذف می­کنیم.



بار دیگر با استفاده از دستور dmesg بافر هسته را چک می­کنیم تا بررسی کنیم پیغامی که انتظار داشتیم زمان حذف ماژول هسته چاپ شود در بافر نوشته شده یا خیر.



عبارت “Removing Birthday Module…” و “Freeing node” به ازای تمام node ها در بافر چاپ شده­است، بنابراین حذف ماژول نیز با موفقیت انجام شده.