a3020008 · Follow
Last edited by a3020008 on Dec 6, 2023
Contributed by

Expand all
Back to top
Go to bottom

Assignment 3: System Information Fetching Kernel Module

Have you ever used the `neofetch` tool? neofetch is a command-line utility that displays system information such as the OS distribution and CPU model.

In this assignment, you are going to implement a kernel module that fetches the system information from the kernel.

```
                   os.staque.xyz
       .-.         -------------
      (.. |        Kernel:   5.19.12-os-0816164
      <>  |        CPU:      Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz
     / --- \       CPUs:     4 / 4
    ( |   | |      Mem:      3017 MB / 3928 MB
  |\\_)___/\)/\    Procs:    188
 <__)------(__/    Uptime:   523 mins
```

Linux Kernel Module

A kernel module is a piece of code that can be loaded and unloaded into the kernel dynamically at runtime. This allows the kernel to be extended without the need to recompile the entire kernel.

You can run `lsmod` to list all loaded modules on the system:

```
$ lsmod
Module                  Size  Used by
tls                   110592  0
binfmt_misc            24576  1
intel_rapl_msr         20480  0
intel_rapl_common      40960  1 intel_rapl_msr
snd_hda_codec_generic   102400  1
ledtrig_audio          16384  1 snd_hda_codec_generic
kvm_intel             421888  0
...
```

And you can use `modinfo` to show information about a module. For example, show the information of the module `tls` :

```
$ modinfo tls
filename:       /lib/modules/5.19.12-os-0816164/kernel/net/tls/tls.ko
alias:          tcp-ulp-tls
alias:          tls
license:        Dual BSD/GPL
description:    Transport Layer Security Support
author:         Mellanox Technologies
srcversion:     CA655CA00B96B66949E2221
...
```

One thing to keep in mind is that a kernel module exists in kernel space and cannot be written in the same way as a normal C program. This is because functions from the C standard library, such as `printf` and `fopen`, do not exist in the kernel. Likewise, structures like `FILE` and `wchar_t` are not available in the kernel either.

To learn how to write a kernel module, we recommend reading the book The Linux Kernel Module Programming Guide. This book has been rewritten by jserv and other contributors to support recent kernel versions (v5.x) and provides a comprehensive guide to writing kernel modules.

Descriptions

In this assignment, you are required to implement a kernel module `kfetch_mod` . `kfetch_mod` is a character device driver that creates a device called `/dev/kfetch` The user-space program `kfetch` can retrieve the system information by reading from this device.

```
$ sudo ./kfetch
                   os.staque.xyz
       .-.         -------------
      (.. |        Kernel:   5.19.12-os-0816164
      <>  |        CPU:      Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz
     / --- \       CPUs:     4 / 4
    ( |   | |      Mem:      3017 MB / 3928 MB
  |\\_)___/\)/\    Procs:    188
 <__)------(__/    Uptime:   523 mins
```

Here is a list of the information that your kernel module should retrieve:

- **Kernel**: The kernel release
- **CPU**: The CPU model name
- **CPUs**: The number of CPU cores, in the format `<# of online CPUs> / <# of total CPUs>`
- **Mem**: The memory information, in the format `<free memory> / <total memory>` (in MB)
- **Procs**: The number of processes
- **Uptime**: How long the system has been running, in minutes.

Kernel Module: `kfetch_mod`

The kernel module `kfetch_mod` is responsible for retrieving all necessary information and providing it when the device is read. Additionally, users can customize the information that kfetch displays by writing a *kfetch information mask* to the device. For example, a user could specify that only the CPU model name and memory information should be shown.

Kfetch information mask

A *kfetch information mask* is a bitmask that determines which information to show. Each piece of information is assigned a number, which corresponds to a bit in a specific position.

```
#define KFETCH_NUM_INFO 6

#define KFETCH_RELEASE   (1 << 0)
#define KFETCH_NUM_CPUS  (1 << 1)
#define KFETCH_CPU_MODEL (1 << 2)
#define KFETCH_MEM       (1 << 3)
#define KFETCH_UPTIME    (1 << 4)
#define KFETCH_NUM_PROCS (1 << 5)

#define KFETCH_FULL_INFO ((1 << KFETCH_NUM_INFO) - 1);
```

The mask is set by using bitwise OR operations on the relevant bits. For example, to show the CPU model name and memory information, one would set the mask like this: `mask = KFETCH_CPU_MODEL | KFETCH_MEM`.

Device operations

Your device driver must support four operations: open, release, read and write.

```
const static struct file_operations kfetch_ops = {
    .owner   = THIS_MODULE,
    .read    = kfetch_read,
    .write   = kfetch_write,
    .open    = kfetch_open,
    .release = kfetch_release,
};
```

For the `read` operation, you need to return a buffer that contains the content of the logo and information to the user space. This allows the user to access and use the data from the device.

```
static ssize_t kfetch_read(struct file *filp,
                           char __user *buffer,
                           size_t length,
                           loff_t *offset)
{
    /* fetching the information */

    if (copy_to_user(buffer, kfetch_buf, len)) {
        pr_alert("Failed to copy data to user");
        return 0;
    }

    /* cleaning up */
}
```

For the `write` operation, a single integer representing the information mask that the user wants to set is passed to the device driver. Subsequent `read` operations will use this mask to determine which information to return to the user. This allows the user to specify which information they want to receive, and the device driver can use the mask to ensure that only the specified information is returned.

```
static ssize_t kfetch_write(struct file *filp,
                            const char __user *buffer,
                            size_t length,
                            loff_t *offset)
{
    int mask_info;

    if (copy_from_user(&mask_info, buffer, length)) {
        pr_alert("Failed to copy data from user");
        return 0;
    }

    /* setting the information mask */
}
```

For the `open` and `release` operations, you need to set up and clean up protections, since in a multi-threaded environment, concurrent access to the same memory can lead to race conditions. These protections can take the form of locks, semaphores, or other synchronization mechanisms that ensure that only one thread can access the variables at a time.

Requirements

- The `open` and `release` operations should set up and clean up protections properly.
- The `write` operation should set the information mask in the module, which determines what data is returned by the `read` operation.
- The `read` operation should return data that includes:
  - A logo of your choosing (or you can use the one provided below if you prefer)
  - Information
    - The first line is the machine hostname, which is mandatory and cannot be disabled
    - The next line is a separator line with a length equal to the hostname
    - The remaining lines depend on the information mask.
  - Note that color support is optional.
- Note that you must release resources such as allocated memory and device major/minor number when the module is removed.

Default logo

```
       .-.
      (.. |
      <>  |
     / --- \
    ( |   | |
  |\\_)___/\)/\
 <__)------(__/
```

Hint

- Linux uses the proc file system to export information about the system, located in the `/proc` directory.
  - For example, memory information can be found in `/proc/meminfo` .
  - The source code can be found at fs/proc. You can view the source code to see how the information is retrieved.
- For the hostname and the release, you might want to see uts_namespaces(7).
- For the number of threads, you might want to see `nr_threads` variable.

Test

We have prepared the user-space program `kfetch` for you. You can download the program source code kfetch.c and the header file (shared with the kernel module) kfetch.h to test your module.

To test, compile it and run it with the option `-h` as root:

```
$ cc kfetch.c -o kfetch
$ sudo ./kfetch -h
```

It will show the program usage:

```
Usage:
        ./kfetch [options]
Options:
        -a  Show all information
        -c  Show CPU model name
        -m  Show memory information
        -n  Show the number of CPU cores
        -p  Show the number of processes
        -r  Show the kernel release information
        -u  Show how long the system has been running
```

Initially, when the module is loaded, the first invocation without any options will display all the information. If the options `-c` and `m` are specified, only the information about the CPU model name and the memory will be displayed.

```
$ sudo ./kfetch -c -m
                   os.staque.xyz
       .-.         -------------
      (.. |        CPU:      Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz
      <>  |        Mem:      3005 MB / 3928 MB
     / --- \
    ( |   | |
  |\\_)___/\)/\
 <__)------(__/
```

Further invocations without any options will print the same information that was specified in the previous call.

```
$ sudo ./kfetch
                   os.staque.xyz
       .-.         -------------
      (.. |        CPU:      Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz
      <>  |        Mem:      3005 MB / 3928 MB
     / --- \
    ( |   | |
  |\\_)___/\)/\
 <__)------(__/
```

Score

- The kernel module can be compiled, loaded, unloaded succesfully. (15%)
- The user-space program `kfetch` can open the device created by the module successfully. (25%)
- The user-space program `kfetch` can set the information mask successfully. (25%):
- The user-space program `kfetch` can display the following information correctly. (35%):
  - Hostname and the seperation line (5%)
  - Kernel (5%)
  - CPU (5%)
  - CPUs (5%)
  - Mem (5%)
  - Proc (5%)
  - Uptime (5%)

For MAC M1/M2, the CPU model name may not be fetched successfully (you can check by `neofetch` or `cat /proc/cpuinfo`). If you target ARM CPU model name, we will use program correctness to replace the displayed information for scoring this part.

Submission

Please submit a **zip** file to E3, which contains your program sources.

- The program must implemented using C.
- Make sure your code can be compiled with `make` command on **Ubuntu 22.04 AMD64 / ARM64**.
  - `make` should compile the kernel module sources.
  - `make load` should insert the kernel module.
  - `make unload` should remove the kernel module.

The name of the zip file should be `<student_id>.zip`, and the structure of the file should be as the following:

```
<stduent_id>.zip
    |- <student_id>/
        |- Makefile
        |- kfetch_mod_<student_id>.c
        |- other files (if you have any)
```

⚠️ **Attention**. You will get *NO POINT* when

- do not follow the submission rule including file name and format.
- cheating including any suspected plagiarism in source code.

The deadline is on **12/29 23:59**.