CSIE 5374 Assignment 4 (Due on 06/05 23:59)

Unlike regular filesystems in Linux (ex: ext4) a **psuedo filesystem** resides in memory entirely and consumes no storage space.
It provides the filesystem hierarchy (e.g. folders and files) like normal filesystem, to store files and directory entries that expose kernel or other information or support system configuration.

The most well-known example of a pseudo file system is procfs (process file system), which is traditionally mounted at the path /proc.
Most of it is read-only, but some files allow kernel variables and behavior to be modified (more details).
In addition to procfs, Linux supports other pseudo file systems such as sysfs (mounted in /sys), ramfs, devpts, debugfs, tmpfs, etc.

For this assignment, you are to implement a pseudo file system called ***SeccompFS***, which exposes seccomp system call trace of the currently running processes and allows users to install seccomp filter for a targeted process.
This assignment will demonstrate Linux's filesystems behavior and teach you how to construct a filesystem that can be mounted to anywere using the mount command.

This is a group assignment, so you should collaborate with your group members on this assignment.

Development Tips

In the following sections, we assume you already know how to compile the Linux kernel and create a root file system image and run it with `qemu-system-aarch64`. You could refer to the assignment 1 specification for these steps.

Compile Linux

To compile the kernel module, we assume you have downloaded or installed the following prerequisites.

- Prerequisite
  - Linux v5.4 source code
    - `git clone git@github.com:torvalds/linux.git`
    - `git checkout tags/v5.4`
  - Cross compiler
    - gcc-aarch64-linux-gnu (4:9.3.0-1ubuntu2)
    - `sudo apt update && sudo apt install gcc-aarch64-linux-gnu`

NOTE 1: You have to make sure the kernel you use in your VM is compiled from the source you specified above. You should use the provided **defconfig** to compile your Linux kernel.

QEMU shared folder (optional)

Frequently updating kernel module binary to filesystem image can be annoying. So it's nice to have a shared folder.

You can follow the instructions below to have a shared folder in QEMU VM.

First, append these two lines to `run-vm.sh`

```
@@ -9,6 +9,7 @@ FS=cloud.img
 CMDLINE="earlycon=pl011,0x09000000"
 DUMPDTB=""
 DTB=""
+SHARED_DIR=./shared

 usage() {
        U=""
@@ -98,3 +99,4 @@ qemu-system-aarch64 -nographic -machine virt,gic-version=2 -m 1024 -
cpu cortex-a
        -append "console=ttyAMA0 root=/dev/vda rw $CMDLINE" \
        -netdev user,id=net0,hostfwd=tcp::2222-:22 \
         -device virtio-net-pci,netdev=net0,mac=de:ad:be:ef:41:49 \
+       -virtfs
local,path=$SHARED_DIR,mount_tag=shared,security_model=passthrough,readonly \
```

Then create a corresponding directory in host.

```
$ mkdir ./shared
```

Finally, boot your QEMU VM up and mount the shared folder somewhere.

```
$ ./run-vm.sh
...

Ubuntu 18.04.6 LTS ubuntu ttyAMA0

ubuntu login: root

...

root@ubuntu:~# mount -t 9p -o trans=virtio shared /mnt
```

Requirements

Implementing SeccompFS (80%)

You should be able to mount your filesystem at /mnt using:

```
mount -t seccompfs none /mnt
```

The mounted SecommpFS should provide the following directory hierarchy and the corresponding attributes of each files.

Directory hierarchy

```
.
├── config
├── begin
├── PID1
│   └── log
├── PID2
│   └── log
└── ...
    └── log
```

File attributes

```
permission  uid     gid     name

/
dr-x------  root    root    PIDs
--w-------  root    root    begin
--w-------  root    root    config

/PIDs/
-r--------  root    root    log
```

The contents of the root of the filesystem should contain two files: `config`, `begin`, as well as a set directories named after different process identifiers (PIDs), the PIDs should be the pid of the currently running process in the system. Each of the PID directory should include a file called `log`, which store the seccomp filtering log of the process respective to the PID.

The following is a simple example of `log` content.

```
63, 7fff0000
64, 7fff0000
29, 0
```

Where `63`, `64`, `29` are the system call numbers called by the target process, which are `__NR_read`, `__NR_write`, `__NR_ioctl` respectively. And `7fff0000` and `0` are the action numbers after evaluating the filter, which mean `SECCOMP_RET_ALLOW` and `SECCOMP_RET_KILL_THREAD` respectively.

Your SeccompFS implementation should support file/directory operations needed for it to function correctly just as other filesystems support by Linux. The filesystem operations in your implementation should use locks and/or mutexs properly to protect concurrent accesses.

Furthermore, your SeccompFS should support the following features:

- The user writes to the file `config` for the PID of the process to attach seccomp filter and the syscall numbers of the whitelisted system calls (i.e. the system calls that a process can make).
  - Your filesystem implementation should accept input string written to `config`, which includes pid and a list of system call numbers separated by commas.

    ```
    PID,LEN,NR1,NR2,NR3,NR4...
    ```

    ```
    48763,5,1,2,45,200,201
    ```

    This means attaching a filter that allows 5 system calls (1, 2, 45, 200, 20) to pid 48763.
  - You should check if the input are valid or not, If any of the input is invalid, you should abort this operation and report an error.
- Allow users to write to the file `begin` to attach a seccomp filter to the process (specified by `PID` in `config`). The Seccomp filter should be constructed using the filtering rule that you specified in the file `config`.
  - You should check if the pid and system call numbers stored in `config` are valid or not. If any configuration is invalid, you should abort this operation and report an error.
  - If the target process has a filter attached by the SeccompFS, you should abort this operation and report an error.
- You should implement SeccompFS at fs/seccompfs and properly create or modify the Makefile to build your code.

More detailed requirements

- If a task is a process that has multiple child threads, you only need to attach the seccomp filter to the process. You don't need to attach the filter to each of its child threads.

Tips

- You are encouraged to refer to the existing kernel code for psuedo filesystem implementations. For instance, you can find the code for ram filesystem from `fs/ramfs` in your kernel source tree, a simple debug filesystem from `fs/debugfs`, and the process filesystem from `fs/procfs`. All of the implementations here make use of the API provided by the generic filesystem library in `fs/libfs.c`.
- You are free to modify existing files in Linux to implement the required functionality.
- You might want to check out **seccomp_set_mode_filter** in `kernel/seccomp.c` - it includes most of the stuff you need to attach a seccomp filter to a target process. For logging filtering information, check out **__seccomp_filter** in `kernel/seccomp.c`.

Write-up (20%)

Each of the groups are required to provide a write-up about the assignment. The write-up should include the explanation of your source code (e.g. description for how your filesystem implementation works), how you test it to make sure it fulfills the required functionality, and contribution from each of your group members.

Homework submission

You should submit the assignment via NTU Cool.

Submission Format

You are required to submit the kernel patch for Linux v5.4 and a write-up file. Compress all the files in hw4.zip, and upload the zip file to NTU Cool.

Your Makefile does not have to deal with copying or deploying the test program to your VM.

```
.
├── write-up.md
├── YOUR_5.4_KERNEL.patch
├── any extra file
```

Each of the groups only need to submit **one copy** of the assignment.

Grading criteria

Please run checkpatch against the changes you made to the Linux kernel. You will lose points if checkpatch reports either warnings or errors. Note that you are not asked to run checkpatch against your userspace programs.

In your system call implementation, you are required to properly protect shared memory accesses, manage resources (free allocated memory), handle errors, and return error codes to userspace.

We will also apply your submitted kernel patch to the Linux v5.4 kernel and test the patched kernel in our Ubuntu 20.04 environment. If we fail to apply your patch to the kernel source, or the patched Linux fails to compile, you will get **Zero** points.

Plagiarism policy

You are allowed to reference sources from the internet. If you do, please attach all of your references in the write-up. If we find that your code is similar to other's from the internet, we will count it as plagiarism if we could not find the corresponding references. You will then get zero points for the assignment automatically.

Late Policy

We do not accept late submissions for this assignment. Please start the assignment early.