Homework 6

W4118 Fall 2014

UPDATED Wednesday 11/26/2014 at 4:45pm EST
DUE: Monday, 12/8/2014 at 11:59pm EST

All non-programming, written problems in this assignment are to be done by yourself. Group collaboration is permitted only on the kernel programming problems. All homework submissions (both individual and group) are to be made via **Git**. Git repository access will use the same public/private key-pair you used for previous homeworks (see these **Git** instructions).

You must submit a detailed list of references as part your homework submission indicating clearly what sources you referenced for each homework problem. You do not need to cite the course textbooks and instructional staff. All other sources must be cited. Be aware that commits pushed after the deadline will not be considered. Refer to the homework policy section on the **class web site** for further details.

Individual Written Problems:

The Git repository you will use for the individual, written portion of this assignment can cloned using:

```
git clone https://os1.cs.columbia.edu/UNI/hmwk6-writ.git
```

(replace `UNI` with your own UNI). This repository will be accessible only by you.

Exercise numbers refer to the course textbook, *Operating Systems Principles and Practice*. Each problem is worth 5 points.

1. Exercise 9.9 (Chapter 9, Exercise 9)
2. Exercise 11.2
3. Exercise 12.8
4. Exercise 12.15
5. Exercise 13.5
6. Exercise 13.9

Group Programming Problems:

Group programming problems are to be done in your assigned **groups**. The Git repository your entire group will use to submit the group programming problems can be cloned using:

```
git clone https://os1.cs.columbia.edu/TEAM/hmwk6-prog.git
```

(Replace `TEAM` with the name of your team, e.g. `team1`). This repository will be accessible to all members of your team, and all team members are expected to commit (local) and push (update the server) changes / contributions to the repository equally. You should become familiar with team-based shared repository Git commands such as **git-pull**, **git-merge**, **git-fetch**. One important thing to remember is to pull any changes into your local repository before trying to push any commits to the class server. Some useful screen casts / tutorials can be found on **gitcasts.com**.

All team members should make at least *five* commits to the team's Git repository. The point is to make incremental changes and use an iterative development cycle. Please follow the **Linux kernel coding**

style. Use the `checkpatch.pl` script to ensure your code conforms to this standard. *Points will be deducted for non-compliance!* It's a good idea to clone your repository after your final push (but before the deadline) and test the cloned code to be sure it behaves as expected and that there are no missing files.

All kernel programming assignments in this year's class are done on the Android operating system and targeting the ARM architecture. The kernel programming for this assignment will be done using a Google **Nexus 7 (2013)** The Android platform can run on many different architectures, but the specific platform we will be targeting is the ARM CPU family. The Google Nexus 7 uses an ARMv7 CPU. The Android emulator can also be used and is built on a system called QEMU which emulates an ARMv7 processor.

You can use Homework 2's VM for this assignment, which can be downloaded from here [www.cs.columbia.edu/~nieh/teaching/w4118/homeworks/w4118.f14.v4.linux.zip] .

Geo-tagged File System

A key feature of modern mobile devices is the availability of various forms of physical sensor input such as accelerometer / orientation input, compass / directional input, and GPS / location input. In particular, mobile computing is increasingly taking advantage of location information to do things like geo-tag photos, provide turn-by-turn directions, locate friends / family, find restaurants, etc. Many of these "location aware" applications use the location information in an application specifc way, and the code which uses this data must be re-written every time a new "location aware" application is developed.

In this homework, you will develop a new kernel mechanism for embedding GPS location information into the filesystem, so that this information can be used by any application. Please review Chapters 13 (The Virtual Filesystem) in your *Linux Kernel Development* book.

1. **Kernel Device Location**

   Write a new system call which updates the kernel with the device's current location then write a user space daemon called `gpsd` which reads the GPS sensor values from a file produced by a custom Android application. The daemon should then update the kernel with the GPS location.

   The new system call number should be **`378`** and the prototype of new system call should be:

   ```
   int set_gps_location(struct gps_location __user *loc);
   ```

   Where `struct gps_location` should be defined as:

   ```
   struct gps_location {
           double latitude;
           double longitude;
           float  accuracy;  /* in meters */
   };
   ```

   Only a system administrator should be able to set the GPS location. All GPS related functions should be put in `kernel/gps.c` and `include/linux/gps.h`.

   In your homework git repository you will find a template daemon project in the `userspace/gpsd` directory. A custom Android application, `GPSLocator.apk`, has been provided in the `userspace` directory. This application will access the GPS on your device and write the values of latitude, longitude, and accuracy (in that order) to your screen and to the file

`/data/media/0/gps_location.txt`. The three values are separated by a newline, note that accuracy is of type float and not double. In your daemon, read the values written to that file and set the GPS location in the kernel using your system call. You should read the values once every second. To install GPSLocator.apk (**device only**, this will not work for the emulator) run this command under your userspace directory in your VM:

```
adb install GPSLocator.apk
```

The application can then be accessed by going to the menu on your device

IMPORTANT!: For the application to work on the device you need to enable the use of location services in `Settings`

2. Ext3 GPS File System Modification

Modify the Linux inode operations interface to include GPS location manipulation functionality, then implement the new operations in the `ext3` file system.

To modify the inode operations interface, add the following two function pointers to the `struct inode_operations` structure in `include/linux/fs.h`:

```
int (*set_gps_location)(struct inode *);
int (*get_gps_location)(struct inode *, struct gps_location *);
```

Note that the `set_gps_location` function pointer does *not* have an input gps location structure - it should use the latest GPS data available in the kernel as set by the `gpsd` you wrote for **Problem 1**.

You only need to implement this GPS location feature for the `ext3` file system. You should update a file's GPS coordinates whenever the file is created *or* modified. Look in the `flo-kernel/fs/` directory for all the file system code, and in the `flo-kernel/fs/ext3/` directory for ext3 specific code. You will need to change the physical representation of an inode on disk by adding the following four fields *in order* to the *end* of the appropriate ext3 structure:

```
        i_latitude (64-bits)
        i_longitude (64-bits)
        i_accuracy (32-bits)
        i_coord_age (32-bits)
```

The first three fields correspond to the `gps_location` structure fields, and the `i_coord_age` field should be set to the number of seconds (32-bits) since the last GPS location was last updated i.e. the *age* of the data.

HINT: You will need to pay close attention to endian-ness of the fields you add to the ext3 physical inode structure. This data is intended to be read by both big *and* little endian CPUs. Also note that the kernel does not have any floating point or double precision support.

Because you are changing the physical representation of the ext3 inode structure, the resulting file system is not technically ext3 anymore. Therefore, you will need to change the **`EXT3`** magic super block number in `flo-kernel/include/linux/magic.h` to be **`0xEF54`**.

Now that you have modified the ext3 file system's physical representation on disk, you will

also need to modify the user space tool which creates such a file system. In your git repository you will find the ext3 file system utilities in the userspace. `userspace/e2fsprogs/` directory. Modify the file in `lib/ext2fs/ext2_fs.h` appropriately to match the new physical layout. (For the purpose of this assignment, it does not matter if your change break e2fsprogs' ability to create ext2 file system.) The modification to an ext2 structure is due to the fact that ext3 is merely an extension of ext2 with journaling enabled. Compile the tool set using:

```
w4118@osvm:~/hmwk6/userspace/src/e2fsprogs$ ./configure
w4118@osvm:~/hmwk6/userspace/src/e2fsprogs$ make
```

The tool you will be most interested in is `misc/mke2fs`. This tool should now create an ext3 file system with your custom modifications. You will use this tool *on your VM* in **Problem 3** to test the geo-tagged file system.

3. User Space Testing

Create a modified ext3 file system using the mke2fs program from **Problem 2**, and write a user space utility named `file_loc` which will output a file's embedded location information including the GPS coordinates, the data age and a Google Maps link.

In order to retrieve a file's location information, write a new system call numbered **379** with the following prototype:

```
int get_gps_location(const char __user *pathname,
                     struct gps_location __user *loc);
```

On success, the system call should return the `i_coord_age` value of the inode associated with the path. The `get_gps_location` system call should be successful only if the file is readable by the current user. It should return -1 on failure setting an appropriate error code which must include **`ENODEV`** if no GPS coordinates are embedded in the file.

The user space utility, `file_loc`, should take exactly one argument which is the path to a file or directory. It should then print out the GPS coordinates / accuracy and data age of the specified file or directory, and also format a Google Maps URL which can be pasted into a web browser to view the location.

In order to use this utility, you will need to create a modified ext3 file system using the `mke2fs` utility you modified in **Problem 2**. Create the ext3 file system *on your VM* using the loop back device:

```
w4118@osvm:~/hmwk6$ dd if=/dev/zero of=hmwk6.fs bs=1M count=2
w4118@osvm:~/hmwk6$ sudo losetup /dev/loop0 hmwk6.fs
w4118@osvm:~/hmwk6$ sudo ./userspace/e2fsprogs/misc/mke2fs -I 256 -t ext3 -L w4118.hmwk6
/dev/loop0
w4118@osvm:~/hmwk6$ sudo losetup -d /dev/loop0
```

The file `hmwk6.fs` should now contain a modified ext3 file system. Before using the file on your device, you will probably need to setup a couple of symlinks in the `/dev/` directory in order to properly mount/umount a file system contained in a regular file (this is referred to as "loopback" mode which allows the kernel to interact with a regular file as a block device). By default, Android puts all the loop back device files in `/dev/block/`, but the mount utilities look for these files in `/dev/`. Run the following series of commands on either the emulator or the device to fix the problem (you will have to do this every time you reboot, so wrapping it in a shell script may be userful):

```
w4118@osvm:~$ adb shell ln -s /dev/block/loop0 /dev/loop0
w4118@osvm:~$ adb shell ln -s /dev/block/loop1 /dev/loop1
w4118@osvm:~$ adb shell ln -s /dev/block/loop2 /dev/loop2
w4118@osvm:~$ adb shell ln -s /dev/block/loop3 /dev/loop3
w4118@osvm:~$ adb shell ln -s /dev/block/loop4 /dev/loop4
w4118@osvm:~$ adb shell ln -s /dev/block/loop5 /dev/loop5
w4118@osvm:~$ adb shell ln -s /dev/block/loop6 /dev/loop6
w4118@osvm:~$ adb shell ln -s /dev/block/loop7 /dev/loop7
```

You can now push the file to either your device or your emulator and mount it:

```
w4118@osvm:~/hmwk6$ adb push hmwk6.fs /data/misc
w4118@osvm:~/hmwk6$ adb shell
# mkdir /data/misc/hmwk6
# mount -o loop -t ext3 /data/misc/hmwk6.fs /data/misc/hmwk6
```

You can now create files and directories in /data/misc/hmwk6 which should be geo-tagged. You need to include the `hmwk6.fs` file in you final git repository submission, and the filesystem must contain at least 1 directory and 2 files all of which must have unique GPS coordinates. Show the output of `file_loc` in your README when called on each of the files / directories you create. **NOTE**: be sure to un-mount the filesystem before pulling the file off of the device, otherwise you risk corruption:

```
w4118@osvm:~/hmwk6$ adb shell umount /data/misc/hmwk6
w4118@osvm:~/hmwk6$ adb pull /data/misc/hmwk6.fs
```

HINT: Wait until your device is fully booted before mounting the loopback device. Otherwise you may receive errors from the losetup utility.

Additional Hints/Tips

- You must always run the custom Android application on the device to update the current location
- Consider using the command `touch` to quickly create files.