Operating System Project 1 Report

Zihan Wang Student ID: 517021911179

Project Files

The complete list of project files are listed below:

```
Prj1+517021911179
    report.pdf                  # The PDF version of report
    report.md                   # The text version of report
    process_tree                # Problem 1-3: Android Process Tree
        ptree.h                 # Common included header of kernel module and test programs
        ptree_mod               # Problem 1: Kernel module for acquiring the process tree
            ptree_mod.c         # 'ptree' kernel module is implemented in this file
            Makefile            # Unix makefile of ptree_mod
        ptree                   # Problem 2: Test program that prints the process tree
            jni                 # JNI directory required by NDK
                ptree.c         # 'ptree' program is implemented in this file
                Android.mk      # Android build configuration of NDK
        ptree_modtest           # Additional tests that shows the robustness of kernel
module
            jni                 # JNI directory required by NDK
                ptree_modtest.c # Additional tests is implemented in this file
                Android.mk      # Android build configuration of NDK
        ptree_exec              # Problem 3: Forks a child process and executes 'ptree'
            jni                 # JNI directory required by NDK
                ptree_exec.c    # ptree_exec is implemented in this file
                Android.mk      # Android build configuration of NDK
    encrytion                   # Problem 4: Caesar Encryption Service
        encrypt.h               # Common header for both server and client program
        server.c                # Multithreaded server program of the encryption service
        client.c                # Client program of the encryption service
```

Note that since problem 1-3 has shared header (which facilitates code reuse), I didn't place their respective source files in the same folder. Instead, I group them in a folder called `process_tree`.

Experiments

The process of experiments and results are stated below. They are arranged in the order of the problems.

Problem 1: Write a new system call

In this part I write a system call with prototype `int ptree(struct prinfo *buf, int *nr)` and register the function as a system call with kernel module. The system call traverses the process struct and copies the relevant information from `task_struct`. Besides, the number of copied items is returned in `nr`.

I made two modifications to the `prinfo` struct. First, I add a member `depth` to record the depth of node in the process tree. This information will help the printing of process tree. Second, I decrease the length of `comm`, because the `comm` member in `task_struct` is not so long.

From my experience, the system call number is a tricky thing. Number 391 will not work in my environment, as the system always issues a `SIGSYS` error. Also the number 356 is not a good choice, as it may be a another preexisting call in the system. From time to time, it's called by another process, and since the system call has changed, it will cause kernel panic. After trail and error, I choose number 375 as the number, which is less erroneous.

Since this part alone has no running result, I'll skip the result demonstration.

Problem 2: Test the system call

This part I test the system call with a program named `ptree`. The program allocates a buffer space for the incoming process information, and specify the capacity of the buffer in `nr`. Then it calls the system call 375 and prints the process tree with information provided in `buf`. The whole process tree is listed above:

```
swapper, 0, 0, 0, 1, 0, 0
    init, 1, 1, 0, 43, 2, 0
        ueventd, 43, 1, 1, 1, 60, 0
        logd, 60, 1, 1, 1, 61, 1036
        vold, 61, 1, 1, 1, 66, 0
        healthd, 66, 1, 1, 1, 67, 0
        lmkd, 67, 1, 1, 1, 68, 0
        servicemanager, 68, 1, 1, 1, 69, 1000
        surfaceflinger, 69, 1, 1, 1, 71, 1000
        qemud, 71, 1, 1, 1, 74, 0
        sh, 74, 1, 1, 1, 75, 2000
        adbd, 75, 1, 1, 251, 77, 0
            sh, 251, 1, 75, 1127, 1, 0
                ptree, 1127, 0, 251, 1, 1, 0
        netd, 77, 1, 1, 1, 78, 0
        debuggerd, 78, 1, 1, 1, 79, 0
        rild, 79, 1, 1, 1, 80, 1001
        drmserver, 80, 1, 1, 1, 81, 1019
        mediaserver, 81, 1, 1, 1, 82, 1013
        installd, 82, 1, 1, 1, 83, 0
        keystore, 83, 1, 1, 1, 85, 1017
        main, 85, 1, 1, 231, 86, 0
            system_server, 231, 1, 85, 0, 661, 1000
            putmethod.latin, 661, 1, 85, 0, 675, 10032
            m.android.phone, 675, 1, 85, 0, 690, 1001
            d.process.acore, 690, 1, 85, 0, 742, 10002
            .quicksearchbox, 742, 1, 85, 0, 759, 10042
            ndroid.systemui, 759, 1, 85, 0, 779, 10013
            m.android.music, 779, 1, 85, 0, 806, 10035
            d.process.media, 806, 1, 85, 0, 835, 10005
            id.printspooler, 835, 1, 85, 0, 891, 10040
            ndroid.keychain, 891, 1, 85, 0, 922, 1000
            .android.dialer, 922, 1, 85, 0, 956, 10004
            viders.calendar, 956, 1, 85, 0, 986, 10001
            gedprovisioning, 986, 1, 85, 0, 1004, 10008
            ndroid.calendar, 1004, 1, 85, 0, 1022, 10019
            droid.deskclock, 1022, 1, 85, 0, 1042, 10023
            m.android.email, 1042, 1, 85, 0, 1057, 10027
            ndroid.exchange, 1057, 1, 85, 0, 1077, 10029
            ndroid.settings, 1077, 1, 85, 0, 1109, 1000
            droid.launcher3, 1109, 0, 85, 0, 1, 10007
        gatekeeperd, 86, 1, 1, 1, 87, 1000
        perfprofd, 87, 1, 1, 1, 89, 0
        fingerprintd, 89, 1, 1, 1, 1, 1000
    kthreadd, 2, 1, 0, 3, 0, 0
        ksoftirqd/0, 3, 1, 2, 0, 5, 0
        kworker/u:0, 5, 1, 2, 0, 6, 0
        khelper, 6, 1, 2, 0, 7, 0
        sync_supers, 7, 1, 2, 0, 8, 0
        bdi-default, 8, 1, 2, 0, 9, 0
        kblockd, 9, 1, 2, 0, 10, 0
        rpciod, 10, 1, 2, 0, 11, 0
        kworker/0:1, 11, 1, 2, 0, 12, 0
        kswapd0, 12, 1, 2, 0, 13, 0
        fsnotify_mark, 13, 1, 2, 0, 14, 0
        crypto, 14, 1, 2, 0, 25, 0
        kworker/u:1, 25, 1, 2, 0, 30, 0
        mtdblock0, 30, 1, 2, 0, 35, 0
        mtdblock1, 35, 1, 2, 0, 40, 0
        mtdblock2, 40, 1, 2, 0, 41, 0
        binder, 41, 1, 2, 0, 42, 0
        deferwq, 42, 1, 2, 0, 45, 0
        kworker/0:2, 45, 1, 2, 0, 46, 0
        jbd2/mtdblock0-, 46, 1, 2, 0, 47, 0
        ext4-dio-unwrit, 47, 1, 2, 0, 50, 0
        flush-31:1, 50, 1, 2, 0, 52, 0
        jbd2/mtdblock1-, 52, 1, 2, 0, 53, 0
        ext4-dio-unwrit, 53, 1, 2, 0, 56, 0
        flush-31:2, 56, 1, 2, 0, 58, 0
        jbd2/mtdblock2-, 58, 1, 2, 0, 59, 0
        ext4-dio-unwrit, 59, 1, 2, 0, 99, 0
        kauditd, 99, 1, 2, 0, 905, 0
        kworker/0:0, 905, 1, 2, 0, 0, 0
```

I make additional tests to demonstrate that this system call can exit gracefully on a error input. Three cases are tested: null buffer pointer, null size number pointer, invalid buffer size. My system call can identify these error input can exit before it's too late (kernel panic).

Besides, I test the case when the buffer size is fewer than the total process number. In the test program, it can be seen that only a limited number of process is printed. Also note that in kernel console, after the limit has reached, although the system call still traverses current level of tree, it doesn't copy more blocks to the buffer.

- Test program:

```
Call with null buffer pointer.
Error: Buffer pointer is null!
Call with null size pointer.
Error: Buffer size pointer is null!
Call with invalid size.
Error: Invalid buffer size!
Call with limited size.
swapper, 0, 0, 0, 1, 0, 0
init, 1, 1, 0, 43, 2, 0
    ueventd, 43, 1, 1, 1, 60, 0
    logd, 60, 1, 1, 1, 61, 1036
    vold, 61, 1, 1, 1, 66, 0
    healthd, 66, 1, 1, 1, 67, 0
    lmkd, 67, 1, 1, 1, 68, 0
    servicemanager, 68, 1, 1, 1, 69, 1000
    surfaceflinger, 69, 1, 1, 1, 71, 1000
    qemud, 71, 1, 1, 1, 74, 0
```

- Kernel info:

```
Module loaded. System call number: 375.
Build Apr  6 2019, 23:44:23
ptree() called.
buf:   (null), nr: acf73000
Buffer pointer is null!
ptree() called.
buf: acf73008, nr:   (null)
Buffer size pointer is null!
ptree() called.
buf: acf73008, nr: acf73000
Invalid buffer size!
ptree() called.
buf: acf73008, nr: acf73000
Ready to acquire task list lock.
Start traversing task tree.
At level 0.
Task number: 0.
Task: swapper, 0, 0, 0, 1, 0, 0.
Traverse next level.
At level 1.
Task number: 1.
Task: init, 1, 1, 0, 43, 2, 0.
Traverse next level.
At level 2.
Task number: 2.
Task: ueventd, 43, 1, 1, 1, 59, 0.
Traverse next level.
Exit level 2.
At level 2.
Task number: 3.
```

```
Task: logd, 59, 1, 1, 1, 60, 1036.
Traverse next level.
Exit level 2.
At level 2.
Task number: 4.
Task: vold, 60, 1, 1, 1, 64, 0.
Traverse next level.
Exit level 2.
At level 2.
Task number: 5.
Task: healthd, 64, 1, 1, 1, 65, 0.
Traverse next level.
Exit level 2.
At level 2.
Task number: 6.
Task: lmkd, 65, 1, 1, 1, 66, 0.
Traverse next level.
Exit level 2.
At level 2.
Task number: 7.
Task: servicemanager, 66, 1, 1, 1, 67, 1000.
Traverse next level.
Exit level 2.
At level 2.
Task number: 8.
Task: surfaceflinger, 67, 2, 1, 1, 69, 1000.
Traverse next level.
Exit level 2.
At level 2.
Task number: 9.
Task: qemud, 69, 1, 1, 1, 72, 0.
Traverse next level.
Exit level 2.
At level 2.
At level 2.
At level 2.
At level 2.
At level 2.
At level 2.
At level 2.
At level 2.
At level 2.
At level 2.
At level 2.
At level 2.
At level 2.
At level 2.
Exit level 1.
At level 1.
Exit level 0.
Task tree traversed.
Task list lock released.
```

Problem 3: Fork child and execute `ptree`

In this problem, I start a process names `ptree_exec`, forks a child, and use `execlp` to execute `ptree`, to show the relationship of parent and child process. In the demonstration, redundant output are omitted. Only critical information is shown. The parent process has `pid` 1133 and the child has 1134, this corresponds to the process tree, where the process `ptree_exec` has `pid` 1133 and `ptree` 1134.

```
517021911179Parent, PID: 1133.
517021911179Child, PID: 1134.
swapper, 0, 0, 0, 1, 0, 0
    init, 1, 1, 0, 43, 2, 0
        ...
        adbd, 75, 1, 1, 251, 77, 0
            sh, 251, 1, 75, 1133, 1, 0
                ptree_exec, 1133, 1, 251, 1134, 1, 0
                    ptree, 1134, 0, 1133, 1, 1, 0
        ...
    kthreadd, 2, 1, 0, 3, 0, 0
        ...
```

Problem 4: Caesar Encryption Service

I implement both server and client programs of Caesar Encryption Service, using network infrastructure provided by Linux. To impose the limit of two clients served at a time, I use a semaphore, which can indicate the availability of resources. I first start the server program, then several clients. To demonstrate the order of the sending and receiving messages, I print current time in both programs. Therefore, I don't need to mention their order here. Note that the third client can be served only after the previous two has quitted. Also I add a feature to to the server that it can quit itself by typing `:q` in its console, by creating a daemon thread for the server program.

- Server:

```
Server initialized at 09:45:47.
Server thread initialized at 09:45:55.
Message received at 09:46:09: Hello world!
Server thread initialized at 09:46:16.
Message received at 09:46:21: Operating System
Message received at 09:46:29: :qw
Message received at 09:47:14:
Server thread closed at 09:47:18.
Server thread initialized at 09:47:18.
Message received at 09:47:18: CS356
Message received at 09:47:29: Goodbye!
Message received at 09:47:39: Nice to meet you.
Server thread closed at 09:47:41.
Server thread closed at 09:47:46.
:q
```

```
Server quit at 09:47:48.
```

- Client 1:

```
Connected to socket at 09:45:55.
Please enter the message:
Hello world!
From server at 09:46:09: Khoor zruog!
Please enter the message:
:qw
From server at 09:46:29: :tz
Please enter the message:
:q
Client closed at 09:47:18.
```

- Client 2:

```
Connected to socket at 09:46:16.
Please enter the message:
Operating System
From server at 09:46:21: Rshudwlqj Vbvwhp
Please enter the message:

From server at 09:47:14: 
Please enter the message:
Goodbye!
From server at 09:47:29: Jrrgebh!
Please enter the message:
:q
Client closed at 09:47:41.
```

- Client 3:

```
Connected to socket at 09:46:38.
Please enter the message:
CS356
From server at 09:47:18: FV356
Please enter the message:
Nice to meet you.
From server at 09:47:39: Qlfh wr phhw brx.
Please enter the message:
:q
Client closed at 09:47:46.
```