システムコール実装の手順書

2018/4/4

高橋 桃花

1　はじめに

本手順書では，システムコール実装の手順を述べる．このシステムコールは，カーネルのメッセージバッファに任意の文字列を出力する機能を持つ．本手順書の読者は，コンソールの基本的な操作を習得している者を想定する．また，本手順書の工程で必要なパッケージは全てインストールされているものとする．以下に章立てを示す．

1 章 はじめに
2 章 実装環境
3 章 Linux カーネルの取得
4 章 システムコールの実装
5 章 テスト
6 章 おわりに

2　実装環境

実装環境を表 1 に示す．

表 1　実装環境

OS	Debian 7.10
カーネル	Linux カーネル 3.15.0
CPU	Intel Core i7 870
メモリ	2.0GB
インストールしたパッケージ	git build-essential libncurses5-dev bc

3 Linux カーネルの取得

3.1 Linux のソースコードの取得

Linux のソースコードを取得する．Linux のソースコードは Git で管理されている．Git とはオープンソースの分散型バージョン管理システムである．下記の Git リポジトリからクローンし，Linux のソースコードを取得する．

```
git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git
```

リポジトリとはファイルやディレクトリを保存する場所のことであり，クローンとはリポジトリの内容を任意のディレクトリに複製することである．本手順書では， /home/takahashi/git 以下でソースコードを管理する．/home/takahashi で以下のコマンドを実行する．

```
$ mkdir git
$ cd git
$ git clone git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git
```

コマンド実行後，mkdir コマンドにより /home/takahashi 以下に git ディレクトリが作成される．そして， cd コマンドにより， git ディレクトリに移動する．git clone コマンドにより，/home/takahashi/git 以下に linux-stable ディレクトリが作成される．linux-stable 以下に Linux のソースコードが格納されている．

3.2 ブランチの作成と切り替え

Linux のソースコードのバージョンを切り替えるため，ブランチの作成と切り替えを行う．ブランチとは開発の履歴を管理するための分岐である．/home/takahashi/git/linux-stable で以下のコマンドを実行する．

```
$ git checkout -b 3.15 v3.15
```

実行後，v3.15 というタグが示すコミットからブランチ 3.15 が作成され，カレントブランチが 3.15 に切り替わる．コミットとはある時点における開発の状態を記録したものである．タグとは，コミットを識別するためにつける印である．以降はブランチ 3.15 において作業を行う．

4 システムコールの実装

4.1 ソースコードの編集

本節では以下の手順でソースコードを編集する．本手順書では，既存ファイルの内容変更を示す際，書き加えた行の先頭には + を，削除した行の先頭には - を付与する．

(1) システムコールの作成
カーネルのメッセージバッファに任意の文字列を出力するシステムコールを作成するため，新しくソースファイルを作成する．本手順書では，ファイル名を mysyscall.c とし，/home/takahashi/git/linux-stable/kernel 以下に作成する．また，システムコールの関数名は sys_mysyscall() とする．以下に詳細を示す．

【形式】 asmlinkage void sys_mysyscall(char* msg)

【引数】 char* msg: 任意の文字列

【戻り値】 なし

【機能】 カーネルのメッセージバッファに任意の文字列を出力する．

(2) システムコールのプロトタイプ宣言
/home/takahashi/git/linux-stable/include/linux 以下の syscalls.h にプロトタイプ宣言を追加する．以下に編集例を示す．

```
 245 asmlinkage long sys_setsid(void);
 246 asmlinkage long sys_setgroups(int gidsetsize, git_t __user *grouplist);
+247 asmlinkage void sys_mysyscall(char* msg);
```

(3) システムコール番号の定義
実装するシステムコールのシステムコール番号を定義する．
/home/takahashi/git/linux-stable/arch/x86/syscalls 以下の syscall_64.tbl を編集する．この番号は，システムコールを呼び出す際の引数として使用する．以下に編集例を示す．

```
 315 common sched_getattr sys_sched_getattr
 316 common renameat2     sys_renameat2
+317 common mysyscall     sys_mysyscall
```

新しく作成するシステムコールの番号を登録する際，正しくシステムコールを呼び出すために既存のシステムコールの番号と重複しないように割り当てる必要がある．本手順書では，システムコール sys_mysyscall() のシステムコール番号を 317 と定義する．

(4) Makefile の作成
作成したファイル mysyscall.c のコンパイルをカーネルのコンパイルに含めるため，/home/takahashi/git/linux-stable/kernel 以下の Makefile を編集する．以下，編集例

を示す.

```
  5 obj-y  = fork.o exec_domain.o panic.o \
  6          cpu.o exit.o itimer.o time.o softirq.o resource.o \
  7          sysctl.o sysctl_binary.o capability.o ptrace.o timer.o user.o \
  8          signal.o sys.o kmod.o workqueue.o pid.o task_work.o \
  9          extable.o params.o posix-timers.o \
 10          kthread.o sys_ni,o posix-cpu-timers.o \
 11          hrtimer.o nsproxy.o \
 12          notifier.o ksysfs.o cred.o reboot.o \
-13          async.o range.o groups.o smpboot.o
+13          async.o range.o groups.o smpboot.o mysyscall.o
```

4.2 カーネルの再構築

次に,以下の手順でカーネルの再構築を行う．各手順のコマンドは/home/takahashi/git/linux-stable 以下で実行する.

(1) .config ファイルの作成
.config ファイルを作成する．.config ファイルとはカーネルの設定を記述したコンフィギュレーションファイルである．以下のコマンドを実行し，x86_64_defconfig ファイルを基にカーネルの設定を行う．x86_64_defconfig ファイルにはデフォルトの設定が記述されている.

```
$ make defconfig
```

実行後，/home/takahashi/git/linux-stable 以下に.config ファイルが作成される.

(2) カーネルのコンパイル
次に，カーネルのコンパイルを行う．以下のコマンドを実行する.

```
$ make bzImage -j8
```

コマンド実行後，/home/takahashi/git/linux-stable/arch/x86/boot 以下に
bzImage という名前の新しいカーネルイメージが作成される．カーネルイメージとは Linux カーネルを格納して圧縮したファイルである．カーネルイメージ作成と同時に，/home/takahashi/git/linux-stable 以下に全てのカーネルシンボルのアドレスを記述した System.map が作成される．カーネルシンボルとは，カーネルのプログラムが格納されたメモリアドレスと対応付けられた文字列のことである.
make コマンドにおける-j [N] オプションは，一度に実行できるジョブの数を指定するものである．[N] はジョブ数を示す．適切なジョブ数を指定することにより，コンパイルを高速化することができる．ジョブ数を指定せずに実行した場合は，可能な限りジョブ数を増やすことになる

が，不用意に数を増やすとメモリ不足となり速度が低下するため，適切な数を指定する必要がある．ここでは，CPU のコア数 4 に対して 2 倍となる 8 を指定した．

(3) カーネルのインストール
コンパイルしたカーネルをインストールする．以下のコマンドを実行する．

```
$ sudo cp /home/takahashi/git/linux-stable/arch/x86/boot/bzImage \
          /boot/vmlinuz-3.15.0-linux
$ sudo cp /home/takahashi/git/linux-stable/System.map \
          /boot/System.map-3.15.0-linux
```

実行後，bzImage と System.map がそれぞれ/boot 以下に vmlinuz-3.15.0-linux と System.map-3.15.0-linux という名前でコピーされる．コピーする際のファイル名は，vmlinuz-(バージョン)-(任意の文字列)，System.map-(バージョン)-(任意の文字列) の形式で設定する．

(4) カーネルモジュールのコンパイル
カーネルモジュールをコンパイルする．カーネルモジュールとはカーネルの機能を拡張するためのバイナリファイルである．以下のコマンドを実行する．

```
$ make modules
```

(5) カーネルモジュールのインストール
コンパイルしたカーネルモジュールのインストールを行う．以下のコマンドを実行する．

```
$ sudo make modules_install
```

上記コマンドの実行結果の最後の行は以下のように出力される．

```
DEPMOD 3.15.0
```

これはカーネルモジュールをインストールしたディレクトリ名を示している．上記の例では，/lib/modules/3.15.0 にカーネルモジュールがインストールされている．

(6) 初期 RAM ディスクイメージの作成
初期 RAM ディスクイメージを作成する．初期 RAM ディスクとは，実際のルートファイルシステムが使用できるようになる前にマウントされる初期ルートファイルシステムである．以下のコマンドを実行する．

```
$ sudo update-initramfs -c -k 3.15.0
```

手順 (5) の実行結果の最後に表示されたディレクトリ名をコマンドの引数として与える．実行後，/boot 以下に初期 RAM ディスクイメージ initrd.img-3.15.0 が作成される．

(7) ブートローダの設定
システムコールを実装したカーネルをブートローダから起動可能にするために，ブートローダの

設定を行う．ブートローダとは，カーネルイメージを RAM へ読み込むために，BIOS が呼び出すプログラムである．BIOS(Basic Input/Output System) とは，基本的な入出力機能の制御を行うプログラムである [1].

Debian7.10 で使用されているブートローダは GRUB2 と呼ばれるものである．ブートローダの設定ファイルは/boot/grub/grub.cfg である．GRUB2 でカーネルのエントリを追加する際，設定ファイルを直接編集せず，/etc/grub.d 以下にエントリ追加用のスクリプトを作成し，そのスクリプトを実行することでエントリを追加する．以下にブートローダの設定の手順を示す.

（A）エントリ追加用のスクリプトの作成

カーネルのエントリを追加するため，エントリ追加用のスクリプトを作成する．本手順書では，既存のファイル名に倣い作成するスクリプトのファイル名は 11_linux-3.15.0 とする．スクリプトの記述例を以下に示す.

```
1 #!/bin/sh -e
2 echo "Adding my custom Linux to GRUB2"
3 cat << EOF
4 menuentry "My custom Linux" {
5   set root=(hd0,1)
6   linux /vmlinuz-3.15.0-linux ro root=/dev/sda2 quiet
7   initrd /initrd.img-3.15.0
8 }
9 EOF
```

スクリプトに記述してある各項目について以下に示す.

(a) menuentry ＜表示名＞

＜表示名＞: カーネル選択画面に表示される名前

(b) set root=(＜ HDD 番号＞, ＜パーティション番号＞)

＜ HDD 番号＞: カーネルが保存されている HDD の番号

＜パーティション番号＞: HDD の/boot が割り当てられたパーティション番号

(c) linux ＜カーネルイメージのファイル名＞

＜カーネルイメージのファイル名＞: 起動するカーネルのカーネルイメージ

(d) ro ＜ root デバイス＞

＜ root デバイス＞: 起動時に読み込み専用でマウントするデバイス

(e) root= ＜ルートファイルシステム＞ ＜その他のブートオプション＞

＜ルートファイルシステム＞: /root を割り当てたパーティション

＜その他のブートオプション＞: quiet はカーネル起動時に出力するメッセージを省略する

(f) initrd ＜初期 RAM ディスク名＞

＜初期 RAM ディスク名＞: 起動時にマウントする初期 RAM ディスク名

（B）実行権限の付与

/etc/grub.d で以下のコマンドを実行し，作成したスクリプトに実行権限を付与する．

```
$ sudo chmod +x 11_linux-3.15.0
```

（C）エントリ追加用のスクリプトの実行

以下のコマンドを実行し，作成したスクリプトを実行する．

```
$ sudo update-grub
```

実行後，/boot/grub/grub.cfg にシステムコールを実装したカーネルのエントリが追加される．

(8) 再起動

任意のディレクトリで以下のコマンドを実行し，計算機を再起動させる．

```
$ sudo reboot
```

再起動した際，GRUB2 のカーネル選択画面に新しく登録したエントリが追加されている．手順 (7) の (A) のスクリプトを用いた場合，カーネル選択画面で My custom Linux を選択し，起動する．

5 テスト

5.1 テストの概要

システムコールが実装されているか否かを確認するため，システムコールを実行してテストする．テストの手順は以下の通りである．

(1) テストプログラムを作成し，カーネルのメッセージバッファに任意の文字列を出力するシステムコールを実行する．
(2) カーネルのメッセージバッファに書き込まれた内容を確認する．

5.2 テストプログラムの作成

システムコールを実行するテストプログラムを作成する．本手順書では，テストプログラムの名前は test.c とし，/home/takahashi/workspace 以下に作成する．このテストプログラムの処理の流れは以下の通りである．

(1) 任意の文字列を定義する
(2) 文字列とシステムコール番号を引数に取り，sys_mysyscall() を呼び出す．
(3) カーネルのメッセージバッファに指定した文字列が格納される．

テストプログラムの作成例を以下に示す．

```
1  #define _GNU_SOURCE
```

```
 2  #include <unistd.h>
 3  #include <sys/syscall.h>
 4
 5  #define SYS_mysyscall 317
 6
 7  int main(void)
 8  {
 9    char *str = "this is test"
10    syscall(SYS_mysyscall, str);
11    return 0;
12 }
```

5 行目では，sys_mysyscall() のシステムコール番号 317 を定義している．

5.3 テストプログラムの実行

5.2 節で作成したプログラムをコンパイルし，実行する．実行後，dmesg コマンドを実行しカーネルのメッセージバッファを確認する．

```
$ dmesg
```

実行後，システムコールが実装されていれば以下のような結果が得られる．

```
[  106.445008] mysyscall: this is test
```

6 おわりに

本手順書では，カーネルのメッセージバッファに任意の文字列を出力するシステムコールの実装手順を示した．また，実装ができているか否かを確認するためのテスト方法について示した．

参考文献

[1] Bovet, D. P. and Cesati, M.: 詳解 Linux カーネル第 3 版，株式会社オライリー・ジャパン (2007).