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書

2018/4/12

高橋 桃花

1　はじめに

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を実装する手順について述べる．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すべてインストールされているものとする．7 章には，付録として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ソースコード例を添付する．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
7 章 付録

2　実装環境

実装環境を表 1 に示す．表 1 の導入済みパッケージの git は，3 章の Linux のソースコードの取得の際に用いる．また，gcc, make, bc は 4 章のカーネルの再構築に用いる．

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 4 コア
メモリ	2.0GB
導入済みパッケージ	git, gcc, make, 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．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．

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

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．

3.2 ブランチの作成と切り替え

Linux カーネルのバージョン 3.15.0 にシステムコールを追加するため，ソースコードのバージョンを切り替える．従って，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の変更を示す際，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．また，以降記載するソースコードにおいて，左端の数字は行番号を示す．

(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．

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ソースコード例を 7 章に添付する．

(2) システムコールのプロトタイプ宣言
/home/takahashi/git/linux-stable/include/linux/syscalls.h に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
/home/takahashi/git/linux-stable/arch/x86/syscalls/syscall_64.tbl を編集し，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．新しくシステムコール番号を定義する際は，既存のシステムコール番号と重複しないようにする．システムコール番号は，システムコールを呼び出すための関数，syscall() の引数として使用するためのものである．以下に編集例を示す．

```
 324 315 common sched_getattr sys_sched_getattr
 325 316 common renameat2     sys_renameat2
+326 317 common mysyscall     sys_mysyscall
```

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．また，syscall_64.tbl 内におけるフォーマットは，左から順に

```
<number> <abi> <name> <entry point>
```

である．各項目について，以下に詳細を示す．

（A）number: システムコール番号

（B）abi: アプリケーションのバイナリファイルとカーネル間のインタフェース
64，x32，または common を指定する．

（C）name: システムコールの名前

（D）entry point: システムコールを呼び出す際に用いる関数の名前

(4) Makefile の編集

カーネルのコンパイル時に，mysyscall.c をコンパイルするため，
/home/takahashi/git/linux-stable/kernel/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
```

上記の例では，13 行目の末尾に mysyscall.o を追加した．

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．以下にコマンドの実行例を示す．

```
$ nproc
8
$ make bzImage -j8
```

nproc コマンドにより利用可能な CPU 数を確認する．次に，make コマンドの-j [N] オプションに確認した CPU 数を指定し，実行する．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り，[N] はジョブ数を示す．ジョブ数を指定しない場合，make コマンドは同時に実行できるジョブの数を制限しない．

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 以下にすべ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．

(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
```

上記の出力結果は，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．本手順書では，/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．上記のコマンドを実行すると，/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].

本環境で使用されているブートローダは GRUB2 である．また，GRUB2 の設定ファイルは /boot/grub/grub.cfg である．GRUB2 でカーネル選択画面に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 GRUB2 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＞: 起動時に選択できるブートオプション

(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 (7A) 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できているか否かを確認するため，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してテストする．テストの手順は以下のとおりである．

(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
(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

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．テストプログラムの処理の流れは以下のとおりである．

(1) 任意の文字列を定義
(2) システムコール番号を指定し，`sys_mysyscall()` の呼び出し

この結果，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  #include <unistd.h> // syscall() が宣言されている
2
3  int main(void)
4  {
5    char *str = "this is test"
6    syscall(317,str); // syscall(sys_mysyscall(),str);
7    return 0;
8  }
```

6 行目において，317 は `sys_mysyscall()` のシステムコール番号を表す．

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．その後，`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] this is test
```

上記の結果において，角括弧内の数字はカーネル起動開始からの経過時間を表す．

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を実装する手順を示した．また，実装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．

7 付録

以下に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ソースコード例を添付する．/home/takahashi/git/linux-stable/kernel/mysyscall.c にソースコードを記述する．

```
1 #include <linux/kernel.h>
2 #include <linux/syscalls.h>
3
4 asmlinkage void sys_mysyscall(char* msg)
5 {
6   printk("%s\n", msg);
7 }
```

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).