Linux カーネルへのシステムコール追加の手順書

2019/04/19

浜本 時空

1　はじめに

本手順書では，Linux カーネルへシステムコールを追加する方法を記す．カーネルへシステムコールを追加するには，Linux のソースコードを取得してシステムコールのソースコードを追加した後，カーネルの再構築を行う．また，本手順書で想定する読者はコンソールの基本的な操作を習得しているものとする．以下に本手順書の章立てを示す．

1 章　はじめに
2 章　実装環境
3 章　追加したシステムコールの概要
4 章　システムコール追加の手順
5 章　テスト
6 章　おわりに
7 章　付録

2　実装環境

システムコールを追加した環境を表 1 に示す．導入済みパッケージのうち git は linux の取得に，gcc，make，bc，libncurses5-dev はカーネルの再構築に用いる．

表 1　実装環境

項目	環境
OS	Debian GNU/Linux 8.11
カーネル	Linux 3.16.0
CPU	Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz
メモリ	16GB

3　追加したシステムコールの概要

今回は，実装のしやすさと実装されているか否かの確認のしやすさの観点から，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追加した．ソースコードは付録 A に添付して

ある．

システムコールの関数名は sys_my_syscall とした．システムコールの概要を以下に示す．

【形式】 `asmlinkage void sys_my_syscall(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．

4 システムコール追加の手順

4.1 概要

システムコール追加手順の構成を以下に示す．

(1) Linux カーネルの取得
　（A）Linux のソースコードの取得
　（B）ブランチの作成と切り替え
(2) ソースコードの編集
　（A）システムコールの作成
　（B）Makefile の編集
　（C）システムコール番号の設定
　（D）システムコールのプロトタイプ宣言
(3) カーネルの再構築
　（A）.config ファイルの作成
　（B）カーネルのコンパイル
　（C）カーネルのインストール
　（D）カーネルモジュールのコンパイル
　（E）カーネルモジュールのインストール
　（F）初期 RAM ディスクの作成
　（G）ブートローダーの設定
　（H）再起動

以降では上記の手順について述べる．(1) については第 4.2 節，(2) については第 4.3 節，(3) については第 4.4 節で述べる．

4.2 Linux カーネルの取得

(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することで，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の状態を記録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/hamamoto/git 以下のディレクトリでソースコードを管理する．/home/hamamoto 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
```

実行すると，mkdir コマンドにより git ディレクトリが作成され，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/hamamoto/git 以下に linux-stable ディレクトリが作成される．この linux-stable ディレクトリ以下に Linux のソースコードが格納されている．

(2) ブランチの作成と切り替え

Linux カーネル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/hamamoto/git/linux-stable で以下のコマンドを実行する．

```
$ git checkout -b 3.16 v3.16
```

実行後，v3.16 というタグが示すコミットからブランチ 3.16 が作成され，ブランチ 3.16 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．

4.3 ソースコードの編集

ソースコードの左端に書かれている数字は行数を表し，追加した行には + を，削除した行には - をつけている．

(1) システムコールの作成

/home/hamamoto/git/linux-stable/kernel 以下に第 3 章で示したシステムコールを作成する．本手順書ではファイル名は my_syscall.c とした．

(2) Makefile の編集

/home/hamamoto/git/linux-stable/kernel 以下にある Makefile を編集して，カーネルのコンパイル時に my_syscall.c がコンパイルされるように設定する．具体的には，Makefile 内にある obj-y = の右辺に my_syscall.o を追加する．編集例を以下に示す．

```
-       13              async.o range.o groups.o smpboot.o
+       13              async.o range.o groups.o smpboot.o my_syscall.o
```

(3) システムコール番号の設定

/home/hamamoto/git/linux-table/arch/x86/syscalls/syscall_64.tbl を編集して，追加するシステムコールのシステムコール番号を定義する．追加する際には既存のシステムコール番号と重複しないようにする．今回作成したシステムコールの番号を 317 とした．このシステムコール番号は，システムコールを呼び出す際に必要になる．編集例を以下に示す．

```
        325  316     common  renameat2               sys_remaneat2
+       326  317     common  my_syscall              sys_my_syscall
```

(4) システムコールのプロトタイプ宣言

/home/hamamoto/git/linux-stable/include/linux/syscalls.h を編集して，追加するシステムコールのプロトタイプ宣言を記述する．編集例を以下に示す．

```
        868  amslinkage long sys_finit_module(int fd, const char __user *uargs, int flags);
+       869  amslinkage void sys_my_syscall(char *msg);
```

4.4 カーネルの再構築

手順 (1) から (6) のコマンドは/home/hamamoto/git/linux-stable 以下で実行する．

(1) .config ファイルのコピー

.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，インストールした Debian のコンフィギュレーションファイルをコピーする．

```
$ cp /boot/config-3.16.0-6-amd64 .config
```

実行後，/home/hamamoto/git/linux-stable 以下に.config ファイルがコピーされる．

(2) カーネルのコンパイル

Linux カーネルをコンパイルする．以下のコマンドを実行する．

```
$ make bzImage -j8
```

上記コマンドにつけた-jN オプションは，同時に実行できるジョブ数を N 個に指定する．ジョブ数が多すぎるとメモリ不足により実行速度が低下する場合があるため，適切なジョブ数を指定す

る必要がある．ジョブ数は CPU のコア数の 2 倍の数が効果的であるため，ジョブ数は 4 コアの 2 倍の 8 とした．コマンド実行後，/home/hamamoto/git/linux-stable/arch/x86/boot 以下に bzImage という名の圧縮カーネルイメージが作成される．カーネルイメージとは実行可能形式のカーネルを含むファイルである．同時に，/home/hamamoto/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/hamamoto/git/linux-stable/arch/x86/boot/bzImage /boot/vmlinuz-3.16.0-linux
$ sudo cp /home/hamamoto/git/linux-stable/System.map /boot/System.map-3.16.0-linux
```

実行後，bzImage と System.map が boot 以下にそれぞれ vmlinuz-3.16.0-linux と System.map-3.16.0-linux という名前でコピーされる．

(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．このコマンドも-j オプションによって高速化できるので，-j8 をコマンドに付与して実行する．

```
$ make modules -j8
```

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

```
$ sudo make modules_install
```

実行結果の最後の行は以下のように表示される．

```
DEPMOD 3.16.0+
```

上記の出力結果のうち，3.16.0+ 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表している．このディレクトリ名は手順 (6) で必要になるため，控えておく．上記の例では/lib/modules/3.16.0+ ディレクトリ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 ディスクの作成
初期 RAM ディスクを作成する．初期 RAM ディスクとは初期ルートファイルシステムのことである．これ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．以下のコマンドを実行する．

```
$ sudo update-initramfs -c -k 3.16.0+
```

手順 (5) で控えておい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 initrd.img-3.16.0+ が作成される．

(7) ブートローダの設定

システムコールを追加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，ブートローダを設定する．ブートローダの設定ファイルは/boot/grub/grub.cfg である．Debian8.11 で使用されているブートローダは GRUB2 である．GRUB2 でカーネルのエントリを追加する際は，この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．ブートローダを設定する手順を以下に示す．

(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-3.16.0+ 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-3.16.0-linux ro root=/dev/sda2 quiet
7 initrd /initrd.img-3.16.0+
8 }
9 EOF
```

スクリプトに記述された各項目名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 はカーネルの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-3.16.0+
```

(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
```

エントリ追加用のスクリプトの実行後，/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
```

再起動後，GRUB2 のカーネル選択画面に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，カーネル選択場面で My custom Linux を選択し，起動する．

5 テスト

5.1 概要

本手順書で追加したシステムコールが実装されているか確認するために，以下の手順で，実際に追加したシステムコールを実行してテストする．

(1) テストプログラムを作成
(2) テストプログラムの実行
(3) システムコールの確認

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を call_my_syscall.c とし，/home/hamamoto 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する
(2) sys_my_syscall のシステムコール番号で syscall() を呼び出す

このプログラムを実行すると，追加したシステムコールを使用して，指定した文字列がカーネルのメッセージバッファに出力される．作成したテストプログラムのソースコードを付録 B に添付してある．

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．以下のコマンドを実行する．

```
$ gcc call_my_syscall.c
$ ./a.out
```

5.4　システムコールの確認

カーネルのメッセージバッファを確認する．以下のコマンドを実行する．

```
$ dmesg
```

実行後，システムコールが追加できていれば以下のような結果が得られる．なお，角括弧内の数字はカーネル起動開始時からの経過時間を表す．

```
[  154.239201] My syscall
```

6　おわりに

本手順書では，カーネルへシステムコールを追加する手順を示した．また，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追加し，機能が正しく動作し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．

付録 A　システムコールのソースコード

以下にシステムコールのソースコードを示す．

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

付録 B　テストプログラムのソースコード

以下にテストプログラムのソースコードを示す．

```
1 #include <unistd.h>
2 #define SYS_my_syscall 317
3
4 int main(void)
5 {
```

```
 6 char *buf = "My syscall";
 7
 8 syscall(SYS_my_syscall,buf);
 9
10 return 0;
11 }
```