Linuxカーネルへのシステムコール追加の手順書

2018/4/20

吉田 修太郎

1　はじめに

本手順書では，Linux カーネルへ新たにシステムコールを追加するための手順について述べる．本手順書で追加するシステムコールは，任意の文字列をカーネルのメッセージバッファに書き込むシステムコールである．なお，本手順書はコンソールの基本操作と `Git` の基礎知識を習得している者を対象としている．本手順書の章立てを以下に示す．

第 1 章　はじめに
第 2 章　システムコールを追加する実装環境
第 3 章　システムコールの追加手順
第 4 章　システムコールの動作テスト
第 5 章　おわりに

2　システムコールを追加する実装環境

本手順書においてシステムコールを追加する実装環境を表 1 に示す．

表 1　実装環境

OS	Debian 7.11
カーネル	Linux カーネル 3.15.0
CPU	Intel(R) Core(TM) i7-4770
メモリ	16GB

3　システムコールの追加手順

3.1　概要

本章では，Linux カーネルへ新たにシステムコールを追加する手順について述べる．以降では，環境設定，Linux カーネルの入手，ソースファイルの作成，システムコールのプロトタイプ宣言，システムコール番号の定義，Makefile の編集，およびカーネルの再構築について，それぞれに節を設けて説明する．また，本章においてファイルの内容を編集する際には，追加した行の先頭に + を，削除した行の先頭に-を記して編集内容を示す．なお，以降で現れる `username` は，システムコールを追加する作業を

行うユーザ名である．

3.2 環境設定

(1) sudo 権限の付与

システムコールを追加する作業を行うユーザに sudo 権限を付与する．以下のコマンドを実行する．

```
$ su
# visudo
```

実行後，エディタが起動し，sudoers.tmp を編集できるようになる．ここで，sudoers.tmp を以下のように編集する．

```
 19 # User privilege specification
 20 root    ALL=(ALL:ALL) ALL
+21 username ALL=(ALL) ALL
```

編集により，username という名前のユーザに sudo 権限を付与する．

(2) ライブラリとコマンドのインストール

カーネルの再構築に必要なライブラリとコマンドをインストールする．以下のコマンドを実行する．

```
$ sudo apt-get update
$ sudo apt-get install git
$ sudo apt-get install gcc
$ sudo apt-get install bc
$ sudo apt-get install make
```

実行により，git，gcc，bc，および make をインストールする．これらは，以降の操作で用いるコマンドの実行に必要となる．

3.3 Linux カーネルの入手

(1) ソースコードの入手

Linux カーネルのソースコードを入手する．このため，Linux カーネルのソースコードが管理されている Git のリポジトリをクローンする．本手順書では，/home/username 以下に git ディレクトリを作成し，/home/username/git 以下でクローンする．以下のコマンドを実行する．

```
$ cd /home/username
$ mkdir git
$ cd git
```

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

実行により，/home/username/git 以下に linux-stable ディレクトリを作成し，Linux カーネルのソースコードは/home/username/git/linux-stable 以下に格納される．

(2) ブランチの作成と切替え

ブランチの作成と切替えを行う．本手順書で使用する Linux カーネルのバージョンは 3.15 であるため，HEAD をバージョン 3.15 のコミットへ移動する．そこで，新規のブランチを作成した後，新規のブランチへカレントブランチを変更する．また，以降で示される操作は，すべて/home/username/git/linux-stable 以下で行う．以下のコマンドを実行する．

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

実行により，v3.15 というタグの示すコミットから新規のブランチ 3.15 を作成し，カレントブランチをブランチ 3.15 に切り替える．

3.4 ソースファイルの作成

追加するシステムコールのソースファイルを作成する．本手順書で追加するシステムコールの概要を以下に示す．

形式 :asmlinkage int sys_prt_to_rbuf(char *s)
引数 :char *s: 任意の文字列の先頭を指すポインタ
戻り値
成功 : カーネルのメッセージバッファに書き込んだ文字数
失敗 : -1
機能 :任意の文字列をカーネルのメッセージバッファに書き込む

本手順書で追加するシステムコールのソースコードを以下に示す．

```
1 #include <linux/kernel.h>
2 #include <linux/syscalls.h>
3
4 asmlinkage int sys_prt_to_rbuf(char *s){
5  int ret;
6  ret = printk(KERN_INFO "%s\n",s);
7  return ret;
8 }
```

上記のソースコードにおいて，6 行目で呼び出している printk() は，引数として与えられた文字列をカーネルのメッセージバッファに書き込む．printk() の戻り値は，書込み成功時はカーネルのメッセージバッファに書き込んだ文字数であり，書込み失敗時は-1 である．上記のソースコードを記した

ソースファイルを/home/username/git/linux-stable/kernel/prt_to_rbuf.c として保存する．

3.5 システムコールのプロトタイプ宣言

システムコールのプロトタイプ宣言を行う．本手順書では，/home/username/git/linux-stable/include/linux/syscalls.h を以下のように編集する．

```
 866 asmlinkage long sys_kcmp(pid_t pid1, pid_t pid2, int type,
 867                          unsigned long idx1, unsigned long idx2)
 868 asmlinkage long sys_finit_module(int fd, const char __user *uargs, int flags);
+869 asmlinkage int sys_prt_to_rbuf(char *s);
```

編集により，sys_prt_to_rbuf() をプロトタイプ宣言する．

3.6 システムコール番号の定義

システムコール番号を定義する．本手順書では，/home/username/git/linux-stable/arch/x86/syscalls/syscall_64.tbl を以下のように編集し，317 番を sys_prt_to_rbuf のシステムコール番号として定義する．

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

3.7 Makefile の編集

Makefile を編集する．編集により，prt_to_rbuf.c を，カーネルのコンパイルに含める．本手順書では，/home/username/git/linux-stable/kernel/Makefile を以下のように編集する．

```
  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 prt_to_rbuf.o
```

3.8 カーネルの再構築

3.8.1 .config ファイルの作成

.config ファイルを作成する．これは，カーネルの設定ファイルである．以下のコマンドを実行する．

```
$ make defconfig
```

実行により，x86_64_defconfig ファイルに基づき，.config ファイルを/home/username/git/linux-stable 以下に生成する．

3.8.2 カーネルのコンパイル

システムコールを追加した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
```

上記コマンドは，実行により/home/username/git/linux-stable/arch/x86/boot 以下に bzImage という圧縮カーネルイメージを生成し，/home/username/git/linux-stable 以下に System.map というシンボルテーブルを生成する．圧縮カーネルイメージとはカーネルを格納して圧縮したファイルであり，シンボルテーブルとはシンボル名とアドレスの対応関係を示すものである．また，上記コマンドに設定されている-j オプションは，同時に実行できるジョブ数を後ろの数字で指定している．

3.8.3 カーネルのインストール

コンパイルしたカーネルのインストールを行う．以下のコマンドを実行する．

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

実行により，カーネルをコンパイルした際に生成された，bzImage と System.map を，vmlinuz-3.15.0-linux と System.map-3.15.0-linux という名前で/boot 以下にコピーする．

3.8.4 カーネルモジュールのコンパイル

以下のコマンドを実行し，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，カーネルの拡張的な機能をモジュール化したものである．

```
$ make modules
```

3.8.5 カーネルモジュールのインストール

以下のコマンドを実行し，コンパイルしたカーネルモジュールをインストールする．

```
$ sudo make modules_install
```

実行後，端末に実行結果が出力され，その最下行には以下のように出力される．

```
DEPMOD 3.15.0
```

この，DEPMOD 以後に続く 3.15.0 という部分は，カーネルモジュールのインストールされたディレクトリ名であり，環境によって異なる．このディレクトリ名は次の操作で用いるため，控えておく．

3.8.6 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、実際のルートファイルシステムを使用できるようになる前にマウントされる一時的なルートファイルシステムのことである．本手順書では，以下のコマンドを実行する．このとき，3.8.5 項で控えておいたディレクトリ名を引数として与える．

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

3.8.7 ブートローダの設定

ブートローダを設定し，カーネル選択画面から，システムコールを追加したカーネルを選択できるようにする．本手順書における環境では，ブートローダは GRUB2 が用いられている．GRUB2 の設定ファイルは/boot/grub/grub.cfg であるが，設定を行う際，この設定ファイルは直接編集しない．/etc/grub.d 以下にあるスクリプトを基に，この設定ファイルが生成される．このため，このディレクトリにスクリプトを追加した後，コマンドを用いて設定ファイルを書き換える．手順を以下に述べる．

(1) スクリプトの追加

/etc/grub.d 以下にスクリプトを追加する．スクリプトのファイル名は，本手順書では /etc/grub.d 以下にある README ファイルの説明に沿うように 11_linux-3.15.0 とする．このスクリプトの内容を以下に示す．

```
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
```

上記スクリプトの内容に関する補足を以下に示す.

(A) `mementry` **<表示名>**

<表示名>: ブートローダのカーネル選択画面に表示される名前. 本手順書では My custom Linux としている.

(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) `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

(D) `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

(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

(2) 実行権限の付与

以下のコマンドを実行し，`/etc/grub.d/11_linux-3.15.0` に実行権限を付与する.

```
$ sudo chmod +x /etc/grub.d/11_linux-3.15.0
```

(3) ブートローダ設定ファイルの更新

以下のコマンドを実行し，ブートローダの設定ファイルを更新する.

```
$ sudo update-grub
```

実行により，起動時に，システムコールを追加したカーネルを選択可能になる.

3.8.8 再起動

再起動を行う．以下のコマンドを実行する.

```
$ sudo reboot
```

実行により，計算機を再起動する．再起動後，カーネル選択画面に 3.8.7 項のスクリプトで設定した**<表示名>**がエントリとして追加されているため，それを選択する．本手順書の場合は，`My custom Linux` を選択する.

4　システムコールの動作テスト

4.1　概要

本章では，追加したシステムコールの動作テストについて述べる．動作テスト用のプログラムを用いて，追加したシステムコールを呼び出し，その振舞いが意図したものであるか否か確認する．以降では，動作テスト用プログラムの作成，動作テスト用プログラムのコンパイル，および動作テストの手順について，順に節を設け述べる．

4.2　動作テスト用プログラムの作成

sys_prt_to_rbuf() の動作テストに用いるプログラムを作成する．本手順書における動作テスト用プログラムについて，ソースコードを以下に示し，説明する．

```
 1 #include<stdio.h>
 2 #include<unistd.h>
 3
 4 int main(){
 5  char buf[128];
 6  long sys_num = 317;
 7  int ret;
 8  scanf("%[^\r\n]",buf);
 9  ret = syscall(sys_num,buf);
10  printf("ret:%d\n",ret);
11  return 0;
12 }
```

上記の動作テスト用プログラムは，任意の文字列の入力を要求し，入力された文字列を syscall() に第 2 引数として渡す．なお，syscall() には 317 という値が第 1 引数として渡されているが，これは sys_prt_to_rbuf() のシステムコール番号である．syscall() によって sys_prt_to_rbuf() を呼び出し，戻り値を端末に出力して終了する．本手順書では，このソースコードを記述したファイルを test_sys.c として保存する．

4.3　動作テスト用プログラムのコンパイル

動作テスト用プログラムをコンパイルする．以下のコマンドを実行する．

```
$ gcc test_sys.c
```

実行により，test_sys.c をコンパイルし，a.out という実行形式ファイルを生成する．

4.4 動作テストの手順

追加したシステムコールの動作テストの手順を以下に示し，説明する．

(1) 動作テスト用プログラムの実行
動作テストプログラムを実行する．以下のコマンドを実行する．

```
$ ./a.out
```

この後，実行された動作テスト用プログラムは，実行時に任意の文字列の入力を要求する．ここで，以下の文字列を入力する．

```
under the water
```

入力後，追加したシステムコールが正常に動作している場合，カーネルのメッセージバッファに上記の文字列“under the water”が書き込まれる．また，このとき端末には以下の以下のような出力がある．

```
ret:15
```

上記の出力は，カーネルのメッセージバッファに 15 文字の文字列が書き込まれたことを意味している．

(2) 書き込みがなされたか否か確認
本手順書では，dmesg コマンドを用いて，カーネルのメッセージバッファに正しく文字列が書き込まれたか否か確認する．dmesg は，カーネルのメッセージバッファの内容を端末に出力するコマンドである．dmesg コマンドを実行したときの出力結果の一部を以下に示す．

```
[12300.934433] under the water
```

上記の出力結果において，左に表示されている数字は，計算機が起動してから何秒後にその行の内容がメッセージバッファに書き込まれたかを表している．上記の出力結果から，指定した文字列“under the water”がカーネルのメッセージバッファに書き込まれていることがわかる．これにより，追加したシステムコールは，意図した動作をしていると言える．

5 おわりに

本手順書では，Linux カーネルへ新たにシステムコールを追加する手順について述べた．さらに，追加したシステムコールが，意図した動作をしているかどうかを調べるテストの手順について述べた．