Linuxカーネルに対するシステムコール追加のための手順

2018/4/10

吉田 修太郎

1 はじめに

本手順書では，Linux カーネルに対して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．本手順書において追加するのは，任意の文字列をカーネルバッファに書き込む機能を持つ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を習得している者に向けてかかれており，本手順書において示される一連の操作の中で，必要となるあらゆるパッケージは，すでにインストールされているものとする．以降では，追加したシステムコールの実装環境，追加したシステムコールの概要，追加の手順，動作テストについて，順に章立ててそれぞれの詳細を述べる．

2 追加するシステムコールの実装環境

本手順書において追加したシステムコールの，実装環境を下表に示す．

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 追加するシステムコールの概要

本章では，本手順書において Linux カーネルに追加するシステムコールの概要について述べる．

形式 asmlinkage int sys_prt_to_rbuf(char *s)
引数 char *s: 出力する文字列のポインタ
戻り値 カーネルバッファに書き込んだ文字数
機能 引数として受け取った文字列をカーネルバッファに書き込む

4 追加の手順

4.1 概要

本章では，新たな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，ソースコードの作成，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ついて節を設けて詳細に述べる．

4.2 ソースコードの作成

ここでは，追加するシステムコールのソースコードの作成について述べる．具体的な記述例として，本手順書 3 章において示したシステムコールの，ソースコードを以下に示す．なお，このソースコードは C 言語で記述されている．

```
#include <linux/kernel.h>
#include <linux/syscalls.h>

asmlinkage int sys_prt_to_rbuf(char *s){
  int ret;
  ret = printk(KERN_INFO "%s\n",s);
  printk("%d charactor(s) outputed\n",ret);
  return ret;
}
```

4.3 プロトタイプ宣言

本節では，プロトタイプ宣言の手順について述べる．システムコール関数のプロトタイプ宣言が，まとめて書かれているヘッダファイルを探し，編集する．本手順書では，以下のファイルを編集する．

- /home/git/linux-stable/include/linux/syscalls.h

本手順書では，このヘッダファイルの末尾に以下の行を追加する．

- /home/git/asmlinkage int prt_to_rbuf(char *s);

4.4 システムコール番号の定義

本節では，システムコール番号を定義する手順について述べる．システムコール関数と，システムコール番号との対応づけが書かれているファイルを探し，編集する．本手順書では，以下のファイルを

編集する．

- /home/git/linux-stable/arch/x86/syscalls/syscall_64.tbl

このファイルの内容の一部を抜粋し，以下に示す．

```
#
# 64-bit system call numbers and entry vectors
#
# The format is:
# <number> <abi> <name> <entry point>
#
# The abi is "common", "64" or "x32" for this file.
#
0 common read sys_read
1 common write sys_write
2 common open sys_open
3 common close sys_close
               ~ (中略) ~
314 common sched_setattr sys_sched_setattr
315 common sched_getattr sys_sched_getattr
316 common renameat2 sys_renameat2

#
# x32-specific system call numbers start at 512 to avoid cache impact
# for native 64-bit operation.
#
512 x32 rt_sigaction compat_sys_rt_sigaction
513 x32 rt_sigreturn stub_x32_rt_sigreturn
514 x32 ioctl compat_sys_ioctl
               ~ (中略) ~
540 x32 process_vm_writev compat_sys_process_vm_writev
541 x32 setsockopt compat_sys_setsockopt
542 x32 getsockopt compat_sys_getsockopt
(EOF)
```

上記のファイル内容の先頭付近に，このファイルのフォーマットは< number >< abi >< name >< entry point >であるという旨が記述されている．それぞれの要素についての簡単な説明を以下に示す．

number　システムコール番号
abi　Applicatioin Binary Interface
name　関数名
entry point　この関数のエントリポイント

このフォーマットに従い，追加したいシステムコールをこのファイルに追記する．ただし，システムコール番号は，システムコール呼出しの際に関数の特定に使用されるため，他の関数と重複してはならない．本手順書では，以下のように設定する．

number　317
abi　common
name　sys_prt_to_rbuf
entry point　sys_prt_to_rbuf

上記の場合におけるファイルへの記入例 (変更した部位とその前後数行のみ抜粋) を以下に示す．先頭に + を置いて示した行が，追加された行である．

```
 314 common sched_setattr sys_sched_setattr
 315 common sched_getattr sys_sched_getattr
 316 common renameat2 sys_renameat2
+317     commoc  sys_prt_to_rbuf         sys_prt_to_rbuf


#
# x32-specific system call numbers start at 512 to avoid cache impact
# for native 64-bit operation.
#
```

4.5 Makefile 編集

ここでは，Makefile の編集について述べる．今回編集する Makefile を以下に示す．

- /home/git/linux-stable/kernel/Makefile

make コマンドは，このファイルの内容に基づいて実行されるため，今回追加したシステムコール関数をコンパイルするためには，ここにその処理を追記する必要がある．具体的には，Makefile の先頭付近に記述されている，各システムコール関数のオブジェクトファイルが代入される obj-y という変数に対して，新たに作成したシステムコールのオブジェクトファイルも代入されるように追記する．本手順書における，この Makefile の編集内容 (編集した部分のみ抜粋) を以下に示す．なお，ここでは削除した行の先頭に-を，追加した行の先頭に＋を挿入している．

```
 #
 # Makefile for the linux kernel
 #

obj-y     = fork.o exec_domain.o panic.o \
            cpu.o exit.o itimer.o time.o softirq.o resource.o \
            sysctl.o sysctl_binary.o capability.o ptrace.o timer.o user.o \
            signal.o sys.o kmod.o workqueue.o pid.o task_work.o \
            extable.o params.o posix-timers.o \
            kthread.o sys_ni.o posix-cpu-timers.o \
            hrtimer.o nsproxy.o \
            notifier.o ksysfs.o cred.o reboot.o \
-           async.o range.o groups.o smpboot.o
+           async.o range.o groups.o smpboot.o   prt_to_rbuf.o
```

4.6 カーネルの再構築

4.6.1 概要

本節では，カーネル再構築の手順について述べる．本節において示される操作は，とくに断りがない限り，すべて/home/username/git/linux-stable 以下で行う．なお，このパスの中の username は，本手順書の操作を行う計算機にログインしているユーザ名に置き換える．また，以降で現れる username もすべて同様である．

4.6.2 .config ファイルの作成

カーネルを再構築するにあたり，はじめに，.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
```

上記コマンドの実行により，x86_64_defconfign ファイルに基づき，.config ファイルが/home/username/git/linux-stable 以下に生成される．

4.6.3 カーネルのコンパイル

システムコールを追加する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
```

上記コマンドの実行により，bzImage という名前の圧縮カーネルイメージと，System.map というファイルが生成され，前者は home/username/git/linux-stable/arch/x86/boot 以下，後者は

/home/username/git/linux-stable 以下に配置される．上記コマンドに付与されている-j オプションは，同時に実行できるジョブ数を指定するものである．これはジョブ数の増加による実行速度の低下を防止するための措置である．また，このときのジョブ数は，”CPU のコア数*2 が効果的である”[1] とされる．

4.6.4 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
```

上記コマンドの実行により，カーネルをコンパイルした際に生成された，bzImage と System.map が/boot 以下にコピーされ，カーネルのインストールが完了する．

4.6.5 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．以下のコマンドを実行する．

```
$ make modules
```

上記コマンドの実行により，カーネルモジュールがコンパイルされる．

4.6.6 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する．以下のコマンドを実行する．

```
$ make modules_install
```

上記コマンドの実行により，コンパイルしたカーネルモジュールがインストールされる．このとき，端末に実行結果が出力されるが，その最下行には以下のような表記がある．

```
DEPMOD 3.15.0+
```

この，DEPMOD 以後に続く 3.15.0 ＋という部分が，カーネルモジュールがインストールされたディレクトリ名である．なお，このディレクトリ名は環境によって異なる．

4.6.7 初期 RAM ディスクの作成

カーネルの再構築において，本項以降の操作は，対象となるカーネルの初回の構築時のみ必要な手順である．初期 RAM ディスクを作成する．本指示書では，以下のコマンドを実行するが，このとき，6 項でカーネルがインストールされたディレクトリ名を引数として与えている．本節第 6 項で述べたように，このディレクトリ名は環境によって変わるため，カーネルモジュールをインストールした際の実行結果をみて，適宜書き換える．

$ sudo update-initramfs -c -k 3.15.0+

上記コマンドの実行により，初期 RAM ディスクが作成される．

4.6.8 ブートローダの設定と再起動

ブートローダの設定と再起動を行う．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．このブートローダの設定ファイルは/boot/grub/grub.cfg であるが，このファイルは gurb のコマンドによって自動生成され，直接編集してはいけないことになっている．/etc/grub.d 以下にあるファイルが，この設定ファイルが生成されるときに用いられるテンプレートとなるので，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．その後 1 度シャットダウンし，システムコールを追加したカーネルで再起動する．その手順を以下に述べる．

(1) 追加するスクリプトの作成
柄するスクリプトを作成する．スクリプトのファイル名であるが，本手順書では/etc/grub.d 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
```

(2) 実行権限の付与
作成したスクリプトに実行権限を付与する．どのディレクトリでもよいので，以下のコマンドを実行する．
$ sudo chmod +x /etc/grub.d/11_linux-3.15.0
上記コマンドの実行により，11_linux-3.15.0 に実行権限が付与される．

(3) ブートローダ設定ファイルの再生成
ブートローダの設定ファイルを再生成する．どのディレクトリでもよいので，以下のコマンドを実行する．
$ sudo update-grub
上記コマンドの実行により，ブートローダの設定ファイルが再生成される．そして，システム

コールを追加したカーネルで起動できるようになる．

(4) 再起動

再起動を行う．以下のコマンドを実行する．

$ sudo reboot

上記コマンドの実行により，計算機が再起動する．このとき，カーネル選択画面で My custom Linux がエントリに追加されているので，それを選択し，起動する．

上記スクリプトを/etc/grub.d 以下に作成することで，システムコールを追加したカーネルがブートローダから起動可能になる．

5 動作テスト

5.1 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．簡単なプログラムを用いて，追加したシステムコールの呼び出しを行い，その振る舞いが意図したものであるか確認する．以降では，動作テスト用プログラムの準備と，動作テストの手順について，順に節を設け述べる．

5.2 動作テスト用プログラムの準備

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を以下に示す．

```
#include<stdio.h>
#include<unistd.h>

int main(){
  char buf[128];
  long sys_num = 317;
  int ret;
  scanf("%[^\r\n]",buf);
  ret = syscall(sys_num,buf);
  printf("ret:%d\n",ret);
  return 0;
}
```

5.3 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す.

(1) 動作テスト用プログラムの実行
(2) dmesg コマンドの実行

上記のとおり，まず動作テストプログラムを実行する．本手順書における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．今回は，” under the water ” と入力する．このとき，追加したシステムコールが適切に動作していれば，カーネルバッファに先ほどのの文字列 ” under the water ” が書き込まれているはずである．そこで，カーネルバッファの内容を端末に出力するため，dmesg コマンドを実行する．このときの出力結果を以下に示す.

```
[12300.934433] under the water
[12300.934434] 18 charactor(s) outputed
```

上記の結果により，指定した文字列”nuder the water”がカーネルバッファに書き込まれていることがわかる．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.

6 おわりに

本手順書では，Linux カーネルに対して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.

参考文献

[1] 小倉 伊織，新人研修資料，https://github.com/nomlab/BootCamp/blob/master/2018/README.org