Linux カーネルへのシステムコール追加の手順書

2020/6/24

野村 優文

1 はじめに

本手順書では，Linux カーネルへのシステムコールの追加手順について記述する．本手順書で追加するシステムコールは，カーネルのメッセージバッファに指定した文字列を出力する機能をもつ．また，本手順書では，読者はコンソールの基本的な操作を習得していることを想定している．

2 実装環境

本システムコールの実装環境を表 1 に示す．

表 1　実装環境

項目	内容
OS	Debian GNU/Linux 10
カーネル	Linux カーネル 4.19.0
CPU	Intel(R) Core(TM) i7 860 CPU @ 2.90GHz
メモリ	2.0GB

3 システムコール追加の手順

3.1 Linux カーネルのソースコードの取得

Linux カーネルのソースコードを取得する．このソースコードは，分散型バージョン管理システムである Git で管理されている．以下に示す Git リポジトリから，Linux カーネルのソースコードをクローンする．

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

リポジトリとはディレクトリを保存する場所であり，クローンとはリポジトリの内容を指定したディレクトリに複製することである．本手順書では，`/home/masafumi/git` 以下でソースコードを管理する．`/home/masafumi` 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
```

上記のコマンドを実行すると，mkdir コマンドで/home/masafumi に git ディレクトリが生成され，cd コマンドで git ディレクトリに移動する．また，git clone コマンドで/home/masafumi/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 カーネルのソースコードが格納されている．

3.2 ブランチの切り替え

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

```
$ git checkout -b 4.19 v4.19
```

上記のコマンドを実行すると，v4.19 というタグが示すコミットからブランチ 4.19 が生成され，ブランチ 4.19 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．また，タグとはコミットを識別するために付ける印である．

3.3 ソースコードの編集

以下の手順でソースコードを編集する．本手順書では，既存のファイルを編集する際，書き加えたソースコードの行の左端には + を付けている．

(1) 自作のシステムコール関数の追加
/home/masafumi/git/linux-stable/kernel にある sys.c に，システムコール関数を記述する．この関数は，カーネルのメッセージバッファに指定した文字列を出力する機能をもつ．表 2 に作成したシステムコール関数の概要を記述する．なお，追加したシステムコール関数のソースコードを付録 A に示す．

表 2　追加したシステムコール関数の概要

項目	内容
形式	SYSCALL_DEFINE1(original_syscall, char __user * msg)
引数	char __user * msg: 指定した文字列の先頭アドレス
戻り値	0
機能	カーネルのメッセージバッファに指定した文字列を出力する．

(2) システムコール番号の追加
/home/masafumi/git/linux-stable/arch/x86/entry/syscalls にある syscall_64.tbl を編集して，追加したシステムコール関数のシステムコール番号を書き加える．システムコール

番号を追加する際は，既存のシステムコール番号と重複しないようにする．本手順書では，新たに追加したシステムコール関数のシステムコール番号を 335 とした．このシステムコール番号は，4 章 (1) において，追加したシステムコール関数のテストプログラムを作成する際に必要なので控えておく．syscall_64.tbl を以下のように書き加える．

```
  343 333     common  io_pgetevents         __64_sys_io_pgetevents
  344 334     common  rseq                  __x64_sys_rseq
+ 345 335     common  original_syscall      __x64_sys_original_syscall
```

このソースコードを入力する際，各列の間はスペースではなくタブを用いて入力する．

(3) システムコール関数のプロトタイプ宣言の追加
/home/masafumi/git/linux-stable/include/linux にある syscalls.h を以下のように書き加える．

```
  1421 long compat_ksys_semtimedop(int semid, struct sembuf __user *tsems,
  1422     unsigned int nsops,
  1423     const struct old_timespec32 __user *timeout);
  1424
  1425 #endif
+ 1426 asmlinkage long sys_original_syscall( char __user *msg );
```

3.4 Linux カーネルの再構築

Linux カーネルの再構築を以下の手順で行う．

(1) .config ファイルの生成
.config ファイルを生成する．ここで，.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．/home/masafumi/git/linux-stable で以下のコマンドを実行する．

```
$ make defconfig
```

上記のコマンドを実行すると，/home/masafumi/git/linux-stable 以下に.config ファイルが生成される．

(2) Linux カーネルのコンパイル
/home/masafumi/git/linux-stable で以下のコマンドを用いて，Linux カーネルをコンパイルする．

```
$ make bzImage -j8
```

(3) コンパイルした Linux カーネルのインストール

コンパイルしたLinuxカーネルをインストールする．/home/masafumi/git/linux-stableで以下のコマンドを実行する．

```
$ sudo cp arch/x86/boot/bzImage /boot/vmlinuz-4.19.0-linux
$ sudo cp System.map /boot/System.map-4.19.0-linux
```

上記のコマンドを実行すると，bzImage と System.map が，/boot 以下にそれぞれvmlinuz-4.19.0-linuxとSystem.map-4.19.0-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．/home/masafumi/git/linux-stableで以下のコマンドを実行する．

```
$ make modules
```

上記のコマンドを実行すると，カーネルモジュールをコンパイルできる．

(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をインストールする．
/home/masafumi/git/linux-stableで以下のコマンドを実行する．

```
$ sudo make modules_install
```

上記のコマンドを実行すると，カーネルモジュールをインストールできる．また，実行結果の最終行には，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が表示される．以下にこの例を示す．

```
DEPMOD 4.19.0
```

上記の例では，/lib/modules/4.19.0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ことを示している．このディレクトリ名は(6)で必要となるため，控えておく．

(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．このディスクは，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．/home/masafumi/git/linux-stableで以下のコマンドを実行する．

```
$ sudo update-initramfs -c -k 4.19.0
```

上記のコマンドでは，(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．このコマンドを実行すると，/boot 以下に初期RAMディスクの initrd.img-4.19.0 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Linux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にあるgrub.cfgである．エン

トリを追加するためには，このファイルを編集する必要がある．Debian10.3 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，設定ファイルを直接編集しない．このため，/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記述する．

（A）エントリ追加用のスクリプトの生成

Linux 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-4.19.0 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-4.19.0-linux ro root=/dev/sda3 quiet
7 initrd /initrd.img-4.19.0
8 }
9 EOF
```

スクリプトに記述された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 >: Linux カーネルが保存されている HDD の番号

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

(c) linux < カーネルのイメージファイル名 >

< カーネルのイメージファイル名 >: 起動する Linux 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

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

```
$ sudo chmod +x 11_linux-4.19.0
```

上記のコマンドを実行すると，作成したスクリプトに実行権限が付与される．

（C）エントリ追加用のスクリプトの実行

/etc/grub.d で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
```

上記のコマンドを実行することにより，/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

以下のコマンドを実行し，計算機を再起動させる．

```
$ sudo reboot
```

再起動した後，GRUB2 のカーネル選択画面にエントリが追加されている．(7) のスクリプトを用いた場合，“My Custom Linux” という名前のエントリが追加される．このエントリを選択し，起動する．

4 追加したシステムコールのテスト

Linux カーネルのメッセージバッファに指定した文字列を出力するシステムコール関数が追加できているか，実際に実行してテストする．テストの手順を以下に記述する．

(1) テストプログラムの作成

システムコールを実行するための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ファイル名は call_original_syscall.c とする．このテストプログラムは，3.3 節 (2) で追加したシステムコール番号と，指定した文字列を引数に与えて，syscall 命令を呼び出す．なお，作成したテストプログラムを付録 B に示す．

(2) テストプログラムのコンパイル

作成したテストプログラムを以下のコマンドを実行し，コンパイルする．

```
$ gcc -o call_original_syscall call_original_syscall.c
```

上記のコマンドを実行することにより，call_original_syscall.c が実行され，実行ファイルである call_original_syscall が生成される．

(3) テストプログラムの実行

テストプログラムを実行する．call_original_syscall を以下のコマンドで実行する．

```
$ ./call_original_syscall Hello!
```

上記のコマンドの第 2 引数には，Linux カーネルのメッセージバッファに出力させる文字列を入力する．本手順書では例として，第 2 引数に Hello!を入力している．また，call_original_syscall を起動し，追加したシステムコール関数を呼び出したとき，以下の文

字列を出力する．

```
Outputted
```

追加したシステムコール関数を呼び出すことができなかったとき，以下の文字列を出力する．

```
Error : -1
```

(4) Linux カーネルのメッセージバッファの内容を確認
以下のコマンドを実行し，Linux カーネルのメッセージバッファを確認する．

```
$ dmesg
```

上記のコマンドを実行したとき，追加したシステムコール関数が呼び出されていれば，以下のような結果を出力する．

```
[  122.938506] <ORIGINAL_SYSCALL>: Hello!
```

上記の結果より，(3) に示すコマンドの第 2 引数で指定した文字列である `Hello!`が，Linux カーネルのメッセージバッファに出力されていることを示している．なお，`[]` 内に示される番号は Linux カーネル起動開始時からの経過時間を表している．

5 おわりに

本手順書では，Linux カーネルのメッセージバッファに，指定した文字列を出力するシステムコールの追加手順を示した．また，このシステムコールが追加できているかどうかを確認する手法を示した．

付録 A 追加したシステムコール関数 (sys.c)

```
152 SYSCALL_DEFINE1(original_syscall, char __user *, msg)
153 {
154     printk(KERN_INFO "<ORIGINAL_SYSCALL>: %s\n", msg);
155     return 0;
156 }
```

付録 B テストプログラム (call_original_syscall.c)

```
 1 #include<stdio.h>
 2 #include<unistd.h>
 3 #include<sys/syscall.h>
```

```
 4 #define SYS_original_syscall 335
 5 int main(int argc, char *argv[])
 6 {
 7    long ret;
 8    ret = syscall(SYS_original_syscall, argv[1], sizeof(argv[1]));
 9    if(ret < 0){
10       fprintf(stderr, "Error : %ld\n", ret);
11    }else{
12       printf("Outputted\n");
13    }
14    return 0;
15 }
```