Linuxカーネルにシステムコールを追加するための手順

2018/4/9
吉田 修太郎

1 はじめに

この文書は，Linux カーネルに対して新たにシステムコールを追加するときの手順を記したものである．この文書を作成するにあたり，筆者が実際に Linux カーネルに新たなシステムコールを追加し，その動作を確認した．以降では，実装環境，追加したシステムコールの概要，実装の手順，動作のテストについて，順に章立ててその詳細を述べる．

2 実装環境

ここでは，今回新たにシステムコールを追加した環境について述べる．追加するシステムコールを実行可能にするためには，そのシステムコール関数の実体を記したファイルおよび同関数のプロトタイプ宣言の他，すべてのシステムコールを管理するテーブルへ，その関数を登録することが必要である．よって，これらを司るファイルが一体どこに存在するのか，ということが明らかでなければならないが，その場所は OS およびカーネルの環境によって異なる．よって，新たなシステムコールを追加するにあたっては，OS とそのカーネルの，それぞれの環境がどのようなものであるかということは非常に重要である．今回，筆者がシステムコールを追加で実装した環境は以下のとおりである．

OS　Debian GNU/Linux 7
カーネルバージョン　3.15.0
カーネルモジュールが置かれている場所　/home/user/git/linux-stable/以下

次章では，この環境において，新たなシステムコールを追加するための手順について述べる．

3 実装の手順

ここでは，前章で示した環境において，新たなシステムコールを追加する手順について述べる．大まかな手順は以下のとおりである．

(1) 追加するシステムコール関数のプロトタイプ宣言を行う．
(2) 追加するシステムコール関数の実体を記述する．
(3) 作成したシステムコール関数を管理テーブルに加える．
(4) Makefile を編集し，作成したシステムコール関数のオブジェクトファイルが含まれるようにする．
(5) カーネルを再構築する．

以降では，上の各項目について，より詳しく説明する．

3.1 プロトタイプ宣言

ここでは，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は，以下のファイルにまとめて書かれている．

パス linux-stable/include/linux/syscalls.h

このヘッダファイルに，今回新たに作成したシステムコール関数の宣言を追加する．今回の環境では，システムコール関数は C 言語によって記述されているため，プロトタイプ宣言の方法も通常の C 言語によるユーザプログラムと同様である．このヘッダファイルに書かれているほかのプロトタイプ宣言に倣って記述すれば良い．

3.2 追加するシステムコール関数の作成

ここでは，追加するシステムコール関数の作成について述べる．プロトタイプ宣言と同様に，システムコール関数の記述も，C 言語を用いて行う．システムコール関数の内部にて，システムコール関数の呼び出しを行っても構わない．今回の環境において，作成したシステムコール関数の実体は，以下の場所に保存する．

パス linux-stable/kernel/

なお，システムコール関数をコーディングするにあたっては，以下のヘッダファイルを include する．

パス include/linux/kernel.h

パス include/linux/syscalls.h

3.3 管理テーブルへの追加

ここでは，システムコール関数全体を管理するテーブルに対し，作成したシステムコールを追加する作業について述べる．今回の環境においては，以下に示すファイルによってテーブルが定義されている．

パス linux-stable/arch/x86/syscalls/syscall_64.tbl

このファイルを編集し，作成したシステムコールを新たなエントリとして追加する．このテーブルのすべてのエントリには，シンボルと呼ばれる番号が上から昇順に振られている．システムコール呼び出しはこのシンボルによって関数が指定されるため，今回新たに追加するシステムコールにも，この番号を定義する必要がある．なお，シンボルと関数は 1 対 1 対応なので，どの関数にも使われていない番号を割り当てなければならない．

3.4 Makefile の編集

ここでは，Makefile の編集について述べる．Makefile とは，make コマンドを使用するためのスクリプトのようなもので，今回編集する Makefile は，カーネルを再コンパイルする際に必要となるものである．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，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．

3.5 カーネルの再構築

ここではカーネルの再構築について述べる．新たなシステムコールを追加する場合，カーネルの再構築が必要である．なぜなら，カーネルモジュールの構築には，システムコール関数の実体が用いられるからである．これは，前節にて編集した Makefile をみれば明らかである．具体的な手順を以下に示す．

(1) .config ファイルの作成
(2) カーネルのコンパイル
(3) カーネルのインストール
(4) カーネルモジュールのコンパイル
(5) カーネルモジュールのインストール

筆者の環境において，上記の手順を実行する場合のコマンドを以下に示す．これらの操作は全て/linux-stable(カーネルモジュールが置かれているディレクトリ) 以下で実行する．

```
$ make defconfig
$ make bzImage -j8
$ sudo cp /home/user/git/linux-stable/arch/x86/boot/bzImage /boot/vmlinuz-3.15.0-linux
$ sudo cp /home/user/git/linux-stable/System.map /boot/System.map-3.15.0-linux
$ make modules
$ sudo make modules_install
```

4 追加したシステムコールの概要

ここでは，今回新たに追加したシステムコールの概要について述べる．今回実装したシステムコールは，任意の文字列をカーネルバッファに書き込むという機能を持つ．関数名は prt_to_rbuf とした．これより，いくつかの観点から，それぞれに節を設け説明する．

4.1 形式

この *prt_to_rbuf* 関数は，任意の文字列をポインタで受け取り，システムコール関数である printk 関数にそのポインタを渡すことで，カーネルバッファに文字列を書き込む．

4.2 引数

この *prt_to_rbuf* 関数は，引数に char 型のポインタをとる．この関数では，引数の char 型ポインタをそのまま printk 関数に渡しているが，入力文字列のサイズによって，複数回に分けて printk 関数に渡した方が安全である．

4.3 戻り値

この *prt_to_rbuf* 関数の返り値は，特に利用を想定しておらず，常に 0 である．今後これを利用する場合，カーネルバッファへの出力文字数を返すという用法が考えられる．

4.4 ソースコード

prt_to_rbuf 関数のソースコードを以下に掲載する．

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

asmlinkage long sys_prt_to_rbuf(char *s){
   printk(KERN_INFO "%s",s);
   return 0;
}
```

5 動作テスト

ここでは，今回追加したシステムコールの動作確認について述べる．まず，作成したシステムコールを呼び出すための適当なプログラムを作成する．そして，そのプログラムを実行した後，dmesg コマンドを用いてカーネルバッファの内容を端末に表示する．正しく動作していれば，端末に設定した文字列を含むログが現れるはずである．目当てのログが，他のログに紛れてしまい見つけにくい場合は，-c オプションをつけると良い．

6 おわりに

ここまで，Linux カーネルに対し，新たに作成したシステムコールを追加する手順について述べた．大まかな手順については記述したが，細かな点について書ききれていないことも多い．Makefile の書式についてや，その他カーネルに関する込み入った内容などに関する不明な点などは，別途お調べ願いたい．

参考文献

[1] 乃村研究室 資料　https://github.com/nomlab/BootCamp/blob/master/2018/README.org