システムコール追加の手順書

2020/6/24

氏名：三宅 貴義

1 概要

本資料では 2020 年度新人研修課題のシステムコールの実装に関して，システムコールを追加する手順，追加したシステムコールの概要，及びテスト方法を示す．

2 追加環境

システムコールの追加を行った環境を表 1 に示す．

表 1　システムコールの追加環境

項目	内容
OS	Debian GNU/Linux 10 (buster)
CPU	Intel(R) Core(TM) i7 870
メモリ	2.0GB
カーネル	Linux 4.19

3 システムコールの概要

この章では追加した 2 つのシステムコールについて概要を示す．

3.1 システムコール 1

追加した 1 つ目のシステムコールの概要を以下に示す．

システムコール名　`my_syscall_1`
システムコール番号　335
形式　64bit 命令
引数　任意の文字列 `char *str`
戻り値　正常終了時は 0，文字列の代入ができなかった場合は-1
機能　任意の文字列をカーネルバファに格納する．ただし，256 文字以上 (終端記号含めず) の場合，255 文字目まで格納される．

3.2 **システムコール** 2

追加した 2 つ目のシステムコールの概要を以下に示す.

システムコール名 `my_syscall_2`
システムコール番号 336
形式 64bit 命令
引数 任意の文字列 (`char *str`)，文字列を格納するメモリの先頭アドレス (`char *addr`).
戻り値 正常終了時は 0，文字列の代入ができなかった場合は-1
機能 任意の文字列を指定したメモリに格納する．ただし，256 文字以上 (終端記号含めず) の場合，255 文字目まで格納される.

4 追加手順

この章ではシステムコールの追加手順を示す.
システムコールを追加するには以下の手順に従って操作を行う.

(1) カーネルソースコードの取得
(2) システムコールの記述
(3) カーネルの構築

4.1 **カーネルソースコードの取得**

GitHub 上で管理されているカーネルソースコードを，以下のコマンドを実行し，自身の端末にクローンする.

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

以下のコマンドを実行し，作業ディレクトリをクローンしたレポジトリに変更する.

```
$ cd linux-stable
```

また，以下のコマンドでブランチを変更する.

```
$ git branch 4.19
```

4.2 **システムコールの記述**

この節ではシステムコールの追加の方法について説明する．システムコールを追加するには以下の操作を行う.

- システムコール番号の登録
- システムコールのコードの記述

- システムコール関数のヘッダへの登録

4.2.1 システムコール表への登録

追加するシステムコールを以下のシステムコール表ファイルへ登録する．

`linux_stable/arch/x86/entry/syscalls/syscall_64.tbl`

システムコール表はシステムコールの番号 (naumber)，形式 (abi)，システムコール名 (name)，エントリーポイント (entry_point) を対応付けるファイルである．このファイルに追加するシステムコールに関するパラメータを新たに記述する．なおエントリーポイントは，形式的にシステムコール名に`__x86_sys_`を接頭したものである．

本課題では作成した 2 つのシステムコールについて，以下の'+' がついている行を追記した．

```
 345 334 common rseq           __x64_sys_rseq
+346 335 common my_syscall_1   __x64_sys_my_syscall_1
+347 336 common my_syscall_2   __x64_sys_my_syscall_2
 348
```

4.2.2 システムコールのコードを記述

システムコールの実体となるコードを以下の C 言語ファイルに記述する．

`linux_stable/kernel/sys.c`

このファイルは比較的小さなシステムコールをまとめて記述したファイルであり，追加するシステムコールもこのファイルに記述する．

本課題で記述したコードについて，システムコール 1 を付録 A に，システムコール 2 を付録 B に記載する．

4.2.3 システムコール関数のヘッダへの登録

システムコールのプロトタイプ宣言を以下のヘッダファイルへ登録する．

`linux_stable/include/linux/syscalls.h`

システムコールの関数名はシステムコール名に `sys_`を接頭したものであり，本来のプロトタイプ宣言の直前に `asmlinkage` をつける．`asmlikage` とは全ての引数がスタック経由で渡されることを定義したマクロである．

本課題では作成したシステムコールのプロトタイプ宣言を以下のように記述した．

```
 290
+294 smlinkage int my_syscall_1(char *buf);
+295 smlinkage int my_syscall_2(char *buf, int mem_addr);
 296
```

4.3 カーネルの構築

前節でシステムコールを追加したカーネルソースコードを，新たに起動可能なカーネルとして構築する．

(1) カーネルのコンパイル

カーネルソースコードをコンパイルし，イメージファイルを作成する．

```
$ make bzImage
```

(2) ファイルのコピー

カーネルの起動に必要なファイルをブートディレクトリにコピーする．

```
$ sudo cp arch/x86/boot/bzImage /boot/vmlinuz-4.19.0-linux
$ sudo cp System.map /boot/System.map-4.19.0-linux
$ cp /boot/config-4.19.0-6-amd64 .config
```

(3) カーネルモジュールのインストール

カーネルモジュールをコンパイルし，インストールする．

```
$ make modules
$ sudo make modules_install
```

このとき表示されたディレクトリ名は後の操作で必要になるため，控えておく．

(4) 初期 RAM ディスクの作成

初期 RAM ディスクとは，実際のファイルシステムが使用できるようになる前に読み込まれる，一時的なファイルシステムである．

<Directry name>を前の操作で控えたディレクトリ名に置き換え，以下のコマンドを実行する．

```
$ sudo update-initramfs -c -k <Directry name>
```

(5) ブートローダの設定

インストールしたカーネルでの起動を選択できるよう，ブートローダの設定を行う．ブートローダを直接操作することは危険なため，シェルスクリプトを用いて設定を行う．

（A）スクリプトの作成

カーネルのエントリ追加用のシェルスクリプトを作成する．ファイル名を 11_linux-4.19.0 とし，付録 C のスクリプトを記述したファイルを作成する．

（B）スクリプトの配置

以下のコマンドを実行し，スクリプトを GRUB の設定ファイルとして配置する．

```
$ cp  11_linux-4.19.0 /etc/grub.d/
```

（C）実行権限の付与

以下のコマンドを実行し，配置したスクリプトに実行権限を付与する．

```
$ sudo chmod +x /etc/grub.d/11_linux-4.19.0
```

（D）スクリプトの実行

以下のコマンドを実行し，スクリプトを実行する．

```
$ sudo update-grub
```

以上の設定が完了すれば起動時に”My custom syscall”というカーネルで起動ができるようになっている．このカーネルを選択し，起動することで追加したシステムコールが使用できる．

5 テスト

この章では追加したシステムコールに行ったテストの手法と，その結果を示す．

5.1 システムコール 1

5.1.1 テスト手法

システムコール 1 が正しく動作しているかを確認するために，システムコールを実行し，正しい結果が出力されるかを確認する．

このときシステムコールを呼び出す関数として，ライブラリ関数の syscall 関数を使用する．この関数は引数に，システムコール番号と，そのシステムコール本来の引数をとり，指定したシステムコールを呼び出す．返り値として，呼び出しに成功した場合は呼び出したシステムコールの返り値を，失敗した場合は-1 を返す．

テストのために作成したプログラムを付録 D に示す．このテストプログラムは，以下のような処理を行っている．

(1) 文字列をコマンドライン引数として受け取る．
(2) 文字列を引数としてシステムコールを実行する．
(3) システムコールの呼び出し結果を表示する．

5.1.2 テスト結果

テストを行った際のシェルの内容を以下に示す．

```
1  $ ./test1 test_mesage
2  syscall returned 0
3  $ dmesg | tail -n 1
4  [   114.813300] my_syscall_1:test_message
```

1 行目では，テストプログラムに”test_message”という文字列を与えて実行している．3 行目では，

dmesg コマンドでカーネルバッファの内容を表示している．カーネルバッファを全て表示させると本プログラムと関係ない内容が多く表示される．そのため，tail コマンドを用いて最終行のみを表示させている．

結果を確認すると，2 行目で返り値が 0 であることからシステムコールが正しく呼び出されていることがわかる．また，4 行目でカーネルバッファに指定した文字列が正しく格納されていることがわかる．このことから，システムコール 1 は正しく動作していることがわかる．

5.2 システムコール 2

5.2.1 テスト手法

システムコール 2 が正しく動作しているかを確認するために，システムコールを実行し，正しい結果が出力されるかを確認する．

テストのために作成したプログラムを付録 E に示す．このテストプログラムは，以下のような処理を行っている．

(1) 文字列をコマンドライン引数として受け取る．
(2) 文字列を格納するメモリを確保する．
(3) 文字列と，確保したメモリの先頭アドレスを引数としてシステムコールを実行する．
(4) システムコールの呼び出し結果を表示する．
(5) 確保したメモリの内容を表示する．

5.2.2 テスト結果

テストを行った際のシェルの内容を以下に示す．

```
1  $ ./test2.c test_message
2  syscall returned 0
3  Address(0x5584ea994260):test_message
```

1 行目では，テストプログラムに “ test_message ” という文字列を与えて実行している．

結果を確認すると，2 行目で返り値が 0 であることから，システムコールが正しく呼び出されていることがわかる．3 行目ではメモリに指定した文字列が格納されている．このことからシステムコール 2 は正しく動作していることがわかる．

6 まとめ

この資料では新人研修課題のシステムコールの追加に関する説明を行った．

付録 A　システムコール 1 のソースコード

```
 197
+196 SYSCALL_DEFINE1(my_syscall_1, char*,msg)
+197 {
+198 char buf[256];
+199 int i;
+200 int length;
+201 for(i=0 ; msg[i]!='\0'; i++){
+202 if(i == 255){
+203 break;
+204 }
+205 }
+206 length = i;
+207 if(copy_from_user(buf,msg,length) != 0){
+208 return -1;
+209 }
+210 buf[length] = '\0';
+211 printk("custom_syscall_1:%s\n",buf);
+212
+213 return 0;
+214 }
+215
```

付録 B　システムコール 2 のソースコード

```
+215
+216 SYSCALL_DEFINE2(my_syscall_2, char*,msg, char*,addr)
+217 {
+218 char buf[256];
+219 int i;
+220 int length;
+221 char *p;
+222
+223 for(i=0; msg[i]!='\0'; i++){
+224 if(i == 255){
```

```
+225 break;
+226 }
+227 }
+228 length = i;
+229
+230 if(copy_from_user(buf, msg, length) != 0){
+231 return -1;
+232 }
+233 buf[length] = '\0';
+234 p = addr;
+235 for(i=0; i<length; i++){
+236 *(p+i) = buf[i];
+237 }
+238
+239 return 0;
+240 }
 241
```

付録 C　カーネルエントリ追加用のスクリプト

```
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 }
```

付録 D　システムコール 1 のテストプログラム

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <sys/syscall.h>
```

```
 6
 7 #define syscall_num 335
 8
 9 int main(int argc, char* argv[])
10 {
11   int res;
12
13   if(argc >= 2){
14     res = syscall(syscall_num, argv[1]);
15     printf("syscall returned %ld\n", res);
16     free(p);
17   }else{
18     printf("Input argue\n");
19     res = 0;
20   }
21   return res;
22 }
```

付録 E　システムコール 2 のテストプログラム

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <string.h>
 4 #include <unistd.h>
 5 #include <sys/syscall.h>
 6
 7 #define syscall_num 336
 8
 9 int main(int argc, char* argv[])
10 {
11   int res;
12   char* p;
13
14   if(argc >= 3){
15     p = (char*)malloc(strle(srg[1])*sizeof(char)+1);
16     res = syscall(syscall_num, argv[1], p);
```

```
17      printf("syscall returned %ld\n", res);
18      printf("Address(%p):%s\n", p, p);
19      free(p);
20    }else{
21      printf("Input argue\n");
22      res = 0;
23    }
24    return res;
25  }
```