IPIを送信するシステムコール実装の手順書

2017/5/8

小倉 伊織

1 はじめに

本手順書では, IPI(InterProcessor Interrupt) を送信するシステムコールの実装手順を説明する．IPIとはプロセッサ間割り込みである．本手順書では，読者としてコンソールの基本的な操作を習得している者を想定する．また，本手順書の工程で必要なパッケージはすべてインストールされているものとする．以下に本手順書の章立てを示す.

1 章 はじめに
2 章 実装環境
3 章 Linux カーネルの取得
4 章 システムコールの実装
5 章 テスト
6 章 おわりに

2 実装環境

実装環境を表 1 に示す.

表 1 実装環境

項目名	環境
OS	Debian7.10
カーネル	Linux カーネル 3.15.0
CPU	Intel Core i7-860 Processor
メモリ	2.0GB

3 Linux カーネルの取得

3.1 Linux のソースコードの取得

Linux のソースコードを取得する．Linux のソースコードは Git で管理されている．Git とは，オープンソースの分散型バージョン管理システムである．下記の Git リポジトリをクローンし，Linux のソースコードを取得する.

git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git
リポジトリとは，ファイルやディレクトリの状態を記録する場所のことであり，クローンとは，リポジトリの内容を任意のディレクトリ以下に複製することである．本手順書では，/home/ogura-i/git 以下でソースコードを管理する．/home/ogura-i 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
```

まず，mkdir コマンドにより/home/ogura-i 以下に git ディレクトリが作成される．次に，cd コマンドにより git ディレクトリに移動する．最後に，git clone コマンドにより/home/ogura-i/git 以下に linux-stable ディレクトリが作成される．linux-stable 以下に Linux 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と作成

Linux のソースコード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．ブランチとは，ソースコードの編集履歴を管理するための分岐であり，ブランチごとに異なる編集履歴を管理できる．/home/ogura-i/git/linux-stable 以下でコマンドを実行する．

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

コマンドの実行後，タグ v3.15 の示すバージョンに 3.15 ブランチが生成され，カレントブランチが 3.15 に切り替わる．以降，本手順書では，ブランチ 3.15 で作業する．

4 システムコールの実装

4.1 ソースコードの編集

システムコールを実装するために，以下の手順でソースコードを編集する．本手順書では，既存ファイルの内容を変更を示す際，書き加えた行の先頭には + を，削除した行の先頭には − を付与する．

(1) システムコール本体の作成
linux-stable/kernel 以下に IPI を送信するシステムコールのソースファイルを作成する．本手順書では，ファイル名を syscalls_ipi.c，システムコール名を send_ipi() とする．作成したシステムコールについて以下に示す．

【形式】asmlinkage void send_ipi(int core_id, int vector)
【引数】int core_id: IPI の送信先コア ID
int vector : 割り込みベクタ
【戻り値】無し

【機能】core_id で指定するコア ID の示すコアに，vector で指定する割り込みベクタを通知する．

(2) システムコールのプロトタイプ宣言

include/linux/syscalls.h を以下のように編集し，作成したシステムコールのプロトタイプ宣言を行う．

```
  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 void send_ipi(int core_id, int vector);
```

(3) システムコール番号の定義

arch/x86/syscalls/syscall_64.tbl を以下のように編集し，作成したシステムコールのシステムコール番号を定義する．システムコール番号とは，システムコールと対応付けた番号であり，システムコールを呼び出す引数となる．

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

この際，他のシステムコールの番号とは異なる番号を割り当てる．これは，システムコール番号が重複していると，システムコールを正しく呼び出せないためである．上記の例では，システムコール send_ipi のシステムコール番号を 317 と定義する．

(4) Makefile の編集

kernel/Makefile を以下のように編集し，追加したファイル kernel/syscalls_ipi.c のコンパイルをカーネルのコンパイルに含める．

```
   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 ipi_syscall.o
```

4.2 カーネルの再構築

以下の手順でカーネルの再構築を行う．各手順で実行するコマンドは/home/ogura-i/git/linux-stable 以下で実行する．

(1) .config ファイルの作成
.config ファイルを作成する．.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．以下のコマンドを実行し，x86_64_defconfig ファイルを基にカーネルの設定を行う．x86_64_defconfig ファイルにはデフォルトの設定が記述されている．

```
$ make defconfig
```

実行後，/home/ogura-i/git/linux-stable 以下に.config ファイルが作成される．

(2) カーネルのコンパイル
システムコールを実装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
```

実行後，/home/ogura-i/git/linux-stable/arch/x86/boot 以下に bzImage という名前の圧縮カーネルイメージが作成される．カーネルイメージとは実行可能形式のカーネルを含むファイルである．同時に，/home/ogura-i/git/linux-stable 以下にすべてのカーネルシンボルのアドレスを記述した System.map が作成される．カーネルシンボルとはカーネルのプログラムが格納されたメモリアドレスと対応付けられた文字列のことである．

(3) カーネルのインストール
コンパイルしたカーネルをインストールする．以下のコマンドを実行する．

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

実行後，bzImage と System.map が/boot 以下にそれぞれ vmlinuz-3.15.0-linux と System.map-3.15.0-linux という名前でコピ複写される．

(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機能を拡張するためのバイナリファイルである．

```
$ make modules
```

(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
```

上記コマンドの実行結果の最後の行は以下のように表示される．これ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を表している．

```
DEPMOD 3.15.0
```

以上の例では，/lib/modules/3.15.0 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．これは手順 (6) で必要となるため，ディレクトリ名を控える．

(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．これ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．以下のコマンドを実行する．

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

上記のコマンドでは，手順 (5) で控えたディレクトリ名をコマンドの引数として与える．実行後，/boot 以下に初期 RAM ディスク initrd.img-3.15.0 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カーネルをブートローダから起動可能にするため，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/grub.cfg である．カーネルのエントリを追加するにはこのファイルを編集する必要がある．Debian7.10 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，設定ファイルを直接編集しない．/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カーネルの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示す．

（A）エントリ追加用のスクリプトの作成

システムコールを実装した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ならいスクリプトのファイル名は 11_linux-3.15.0 とする．スクリプトの記述例を以下に示す．

```
#!/bin/sh -e
echo "Adding my custom Linux to GRUB2"
cat << EOF
menuentry "My custom Linux" {
set root=(hd0,1)
linux /vmlinuz-3.15.0-linux ro root=/dev/sda2 quiet
initrd /initrd.img-3.15.0
}
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 >: 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) 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

(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ィション

< 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

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

```
$ sudo chmod +x 11_linux-3.15.0
```

(C) エントリ追加用のスクリプトの実行

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
```

実行後，/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

システムコールを追加したカーネルで起動するために以下のコマンドを実行し，計算機を再起動させる.

```
$ sudo reboot
```

再起動後，GRUB2 のカーネル選択画面にエントリが追加されている．手順 (7) のスクリプトを用いた場合，カーネル選択画面で My custom Linux を選択し，起動する.

5 テスト

5.1 概要

IPI を送信するシステムコールが実装できているか否かをシステムコールを実行してテストする．しかし，IPI の送信だけでは IPI の送信の成功を確認できない．このため，IPI 受信時に実行される割り込みを新たに実装しそれを用いてテストする．具体的には，まず，特定の割り込みベクタを持つ IPI の受信時に実行される割り込みハンドラと，割り込みハンドラを登録するシステムコールを作成する．作成する割り込みハンドラは，カーネルのメッセージバッファにメッセージを格納するものである．次に，割り込みハンドラを登録するシステムコールを実行する．その後，IPI を送信するシステムコールを実行する．最後に，カーネルのメッセージバッファに書き込まれた内容を確認する．上記の流れでテストを行う．割り込みハンドラを登録する処理の流れを以下に示す．

(1) 割り込みハンドラを IRQ(Interrupt ReQuest) 表の未登録のエントリに登録する．IRQ 表とは，各コアにおける割り込み処理を管理するものである．この表の各エントリは IRQ 番号によって識別される．
(2) 割り込みハンドラを登録したエントリの IRQ 番号をベクタ管理表の割り込みベクタのエントリに追加する．ベクタ管理表とは，割り込みベクタと IRQ 番号の対応を管理するものである．

本手順書では，割り込みハンドラの名前を `test_handler()`，割り込みハンドラを登録するシステムコールの名前を `request_ipi_irq()` とする．

5.2 システムコール request_ipi_irq() の実装

4.1 節と同様に，システムコール `request_ipi_irq()` のソースファイルを追加し，プロトタイプ宣言し，システムコール番号を追加する．新たに実装する割り込みハンドラを登録するシステムコール `request_ipi_irq()`，および割り込みハンドラ `test_handler()` について，以下で説明する．

(1) 割り込みハンドラを登録するシステムコール
【形式】`asmlinkage int request_ipi_irq(int core_id, int vector)`
【引数】int core_id: IPI の送信先コア ID
int vector: 割り込みベクタ
【戻り値】成功: `test_handler()` を登録した irq 番号
失敗: -1
【機能】IRQ 表における未登録の IRQ 番号のエントリに割り込みハンドラ `test_handler()` を登録する．その後，この IRQ 番号をベクタ管理表の vector で指定されたエントリに登録する．
(2) 割り込みハンドラ
【形式】`irqremturn_t test_handler(void)`
【引数】無し

【戻り値】成功: IRQ_HANDLED
失敗: IRQ_NONE
【機能】カーネルのメッセージバッファに “ Recieved IPI(core:x) ” というメッセージを格納する．
格納されるメッセージ中の “ core:x ” の x には IPI を受信したコア ID が出力される．

また，4.2 節の手順でカーネルを再構築する．

5.3 テストプログラムの作成

5.1 節で作成したシステムコールを用いて，IPI を送信するシステムコールのテストを行う．テストを行うプログラムを/home/ogura-i/git/linux-stable/test 以下に作成する．本手順書では，テストプログラムの名前は test.c である．

このテストプログラムの処理の流れは以下のとおりである．

(1) システムコール request_ipi_irq() を呼び出す．
(2) IRQ 表に test_handler() が登録され，登録したエントリの IRQ 番号がベクタ表の特定の割り込みベクタ（本手順書では 48 とする）のエントリに追加される．
(3) システムコール send_ipi() が呼び出される．
(4) 送信先のコア（本手順書ではコア ID が 2 のコアを指定する．）に割り込みベクタ 48 の IPI が送信される．
(5) IPI を受信したコアが test_handler() を実行する．
(6) カーネルのメッセージバッファに “ Recieved IPI(core:2) ” が格納される．

このプログラムの具体的な記述例を以下に示す．

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/types.h>
4
5  int main(void){
6          syscall(318,2,48);
7          syscall(317,2,48);
8          return 0;
9  }
```

プログラム中の 6 行目の 318 は，request_ipi_irq() のシステムコール番号であり，7 行目の 317 は send_ipi() のシステムコール番号である．

5.4 テストプログラムの実行

5.2 節で作成したテストプログラムを実行し，IPI を送信するシステムコールが実装できていることを確認する．以下のコマンドを実行し，テストプログラムのコンパイルと実行を行う．

```
$ cd /home/ogura-i/git/linux-stable/test
$ gcc test.c
```

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

その後，任意のディレクトリ下で以下のコマンドを実行し，メッセージバッファの内容を確認する．

```
$ dmesg
```

IPI を送信するシステムコールが実装できていれば，以下のような実行結果が得られる．

```
[ 235.937522] Received IPI(core:2)
```

6 おわりに

本手順書では，IPI を送信するシステムコールを例にシステムコールの実装手順を示した．また，実装できているか否かの確認を行うために確認手法を示した．

本手順書の環境である Linux カーネル 3.15.0 では，割り込みベクタを 48 としている．これは，割り込みベクタ 48 のエントリが未登録であったためである．しかし，本手順書の環境以外ではこのエントリが未登録であるとは限らないため割り込みベクタの値をテスト実行時に探せる方法を検討している．