IPIを送信するシステムコールの実装手順書 (改訂版)

2017/6/5

橋本 鉄平

改訂内容

本手順書は，<No.325-02> で提出した手順書の改訂版である．具体的な変更点は，7 章の割り込みハンドラを登録するシステムコールにおいて，IRQ 番号の設定可能な範囲を検討した．

1　はじめに

本手順書では，IPI(Inter-Processor Interrupt) を送信するシステムコールを実装する手順を述べる．IPI とはプロセッサ間の割り込みである．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．システムコールの実装対象カーネルは，本手順書の工程の中で取得するため，事前に用意する必要はない．7 章には，付録として IPI を送信するシステムコールとテスト用システムコールのソースコード例を添付する．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
7 章 付録

2　実装環境

実装環境を表 1 に示す．

表 1　実装環境

OS	Debian 7.10
システムコールを実装するカーネル	Linux カーネル 3.15.0
CPU	Intel Core i7-870
メモリ	2.0GB

3 Linux カーネルの取得

(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 リポジトリからクローンし，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を保存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に複製することである．本手順書では/home/hashimoto/git 以下でソースコードを管理する．/home/hashimoto 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/hashimoto 以下に git ディレクトリが作成される．そして，cd コマンドにより git ディレクトリに移動する．git clone コマンドにより /home/hashimoto/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．

(2) ブランチの作成と切り替え

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，開発の履歴を管理するための分岐である．/home/hashimoto/git/linux-stable で以下のコマンドを実行する．

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

実行後，タグ v3.15 が示すコミットからブランチ 3.15 が作成され，作業するブランチが 3.15 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．タグとは，コミットを識別するためにつける印である．

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，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変更を示す際，追加した行の行頭には + を，削除した行の行頭には − を付与する．

(1) IPI を送信するシステムコールの作成

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．ソースファイルは /home/hashimoto/git/linux-stable/kernel 以下に作成する．本手順書では，作成するソースファイル名を ipi_syscall.c，システムコールの関数名を sys_send_ipi() とする．sys_send_ipi() について，以下で説明する．また，IPI を送信するシステムコールのソースコード例を 7 章に添付する．

【形式】 asmlinkage void sys_send_ipi(int core_id,int vector)

【引数】 int core_id: IPI の送信先のコア ID

int vector: 割り込みベクタ表のベクタ番号

【戻り値】 なし

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

(2) Makefile の編集

作成したソースファイル ipi_syscall.c のコンパイルをカーネルのコンパイルに含めるため，/home/hashimoto/git/linux-stable/kernel 以下の Makefile を編集する．本手順書では以下のように編集する．

```
 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
```

(3) システムコール番号の定義

実装するシステムコールのシステムコール番号を定義する．この番号はシステムコールを呼び出す際の引数として使用される．このため, 定義するシステムコール番号は, 既存のシステムコール番号と重複しないように定義する．/home/hashimoto/git/linux-stable/arch/x86/syscalls 以下の syscall_64.tbl を編集し，システムコール番号を定義する．本手順書では以下のように編集する．

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

上記のように編集することで sys_send_ipi() にシステムコール番号 317 を定義する．

(4) システムコールのプロトタイプ宣言
実装するシステムコールのプロトタイプ宣言を行う．このため，/home/hashimoto/git/linux-stable/include/linux 以下の syscalls.h を編集し，プロトタイプ宣言を追加する．本手順書では以下のように編集する．

```
 868 asmlinkage long sys_finit_module(int fd, \
                        const char __user *uargs, int flags);
+869 asmlinkage void sys_send_ipi(int core_id, int vector);
```

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．本節で実行するコマンドはすべて/home/hashimoto/git/linux-stable 以下で実行する．

(1) .config ファイルの作成
.config ファイル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し，x86_64_defconfig ファイルを基にカーネルの設定を行う．

```
$ make defconfig
```

実行後，linux-stable 以下にデフォルト設定の.config ファイルが作成される．

(2) カーネルのコンパイル
以下のコマンドを実行してカーネルをコンパイルする．コマンド中のオプション-j8 は，同時に実行できるジョブ (コマンド) の数を指定するオプションである．このオプションを付けることにより，コンパイルが高速化される．

```
$ make bzImage -j8
```

実行後，/home/hashimoto/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，計算機で実行可能な形式のカーネルを格納しているファイルである．同時に，/home/hashimoto/git/linux-stable 以下に全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
以下のコマンドを実行し，コンパイルしたカーネルをインストールする．

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

手順 (3) では，bzImage と System.map を/boot 以下にそれぞれ vmlinuz-3.15.0-linux と System.map-3.15.0-linux という名前でコピーする．これにより，コンパイルしたカーネルがインストールされる．コピー先のファイル名は，vmlinuz-(バージョン)-(任意の文字列)，System.map-(バージョン)-(任意の文字列) の形式で設定する．

(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，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．以下のコマンドを実行する．

```
$ 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てカーネルモジュールをインストールする．

```
$ sudo make modules_install
```

実行結果の最後の行は以下のように出力される．

```
DEPMOD 3.15.0
```

DEPMOD の後ろに表示されている文字列は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を表している．上記の例では，/lib/modules/3.15.0 にカーネルモジュールがインストールされている．このディレクトリ名は後述の手順 (6) で必要になるため，控えておく．

(6) 初期 RAM ディスクイメージの作成
初期 RAM ディスクイメージを作成する．初期 RAM ディスクとは，実際のルートファイルシステムが使用可能になる前にマウントされる初期ルートファイルシステムのことである．以下のコマンドを実行する．

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

手順 (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．実行後，/boot 以下に初期 RAM ディスクイメージ 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．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エントリを追加する．設定手順を以下に示す．

（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．

```
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
```

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のファイル名

(d) ro

ルートファイルシステムを読み込み専用でマウントするブートオプション [2][3]

(e) root=< ルートデバイス >

< ルートデバイス >: 起動時にルートファイルシステムとして使うデバイス [2][3]

(f) quiet

カーネルの起動時に出力するメッセージを省略するブートオプション

(g) 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 ディスクイメージのファイル名

6 行目の < カーネルイメージのファイル名 > と 7 行目の < 初期 RAM ディスク名 > は, それぞれ手順 (3) と手順 (6) で作成したカーネルイメージと初期 RAM ディスクイメージの名前に相当する.

(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
```

(C) エントリ追加用スクリプトの実行

任意のディレクトリ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
```

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

システムコールを実装したカーネルで起動するため，任意のディレクトリで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でエントリが追加されていることを確認できる．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 テストの概要

システムコールを実行して，システムコールが実装できているか否かをテストする．しかし，IPI の送信だけでは，IPI の送信成功を確認できない．このため，IPI 受信時に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割り込みハンドラを実装する．また，この割り込みハンドラを登録するシステムコールも併せて実装する．その後，割り込みハンドラを登録するシステムコールを実行し，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．上記の手順で 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テストする．実装する割り込みハンドラとシステムコールについて，以下にそれぞれの概要を示す．

(1) IPI 受信時に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割り込みハンドラ

この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．その後，割り込みを処理したことを通知し，処理を終了する．また，この割り込みハンドラのメッセージから，指定したコアに IPI を送信できたか否かの確認も行う．本手順書では，実装環境の CPU のコア数が 4 であるため，格納するメッセージの異なる割り込みハンドラを 4 つ作成し，各コアに各割り込みハンドラを対応付ける．これにより IPI を受信したコアを確認する．割り込みハンドラの名前は，それぞれ `message_handler0()`，`message_handler1()`，`message_handler2()`，`message_handler3()` とする．

(2) 割り込みハンドラを登録するシステムコール

このシステムコールは，割り込みハンドラを割り込み要求 (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一括して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．本手順書では異なる 4 つの割り込みハンドラを作成するため，各割り込みハンドラを別のエントリに登録する．その後，ベクタ管理表の割り込みベクタのエントリに各 IRQ 番号を追加する．ベクタ管理表とは，コアごとに存在し，割り込みベクタと IRQ 番号の対応を管理するものである．システムコールの関数名は，`sys_request_ipi_irq()` とし，システムコール番号は 318 に対応付ける．

5.2 テスト用システムコールの説明

割り込みハンドラ message_handler0() と割り込みハンドラを登録するシステムコール sys_request_ipi_irq() について以下で説明する．message_handler1()，message_handler2()，message_handler3() については，message_handler0() の格納するメッセージを変更するだけであるため説明を省略する．また，テスト用システムコールのソースコード例を 7 章に添付する．

(1) 割り込みハンドラ message_handler0()

【形式】 irqreturn_t message_handler0(int irq,void *dev)

【引数】 int irq: IRQ 番号
void *dev: デバイス ID

【戻り値】 IRQ_HANDLED: 割り込みを処理したことを通知する値

【機能】 カーネルのメッセージバッファに"Recieved IPI 0"というメッセージを格納する．メッセージ末尾の数字はコア ID を表している．

この割り込みハンドラは，通常の割り込みハンドラの形式に従って作成するため，引数が 2 つある．しかし，本手順書において，これらの引数は使用しない．

(2) 割り込みハンドラを登録するシステムコール sys_request_ipi_irq()

【形式】 asmlinkage void sys_request_ipi_irq(int vector)

【引数】 int vector: 割り込みベクタ

【戻り値】 成功: 割り込みハンドラを登録した IRQ 番号
失敗: -1

【機能】 IRQ 表の未登録なエントリを探し，割り込みハンドラを登録する．その後，登録したエントリの番号を各コアのベクタ表の vector で指定されたエントリに追加する．

5.3 テストプログラムの作成

システムコールを実行する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名前は test.c とし，/home/hashimoto/test 以下に作成する．

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 sys_request_ipi_irq() を呼び出す．
(2) IRQ 表の未登録なエントリを探し，割り込みハンドラを登録する．
(3) 各コアの割り込みベクタ表の 100 番目のエントリに，対応した IRQ 番号を登録する．
(4) システムコール sys_send_ipi() を呼び出す．
(5) 送信先のコアに割り込みベクタ 100 の IPI が送信される．本手順書では，コア ID が 1 のコアを指定する．

(6) IPI を受信したコアが割り込みハンドラを実行する．本手順書では，messae_handler1() が実行される．

(7) カーネルのメッセージバッファに"Recieved IPI 1"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(318,100);     // syscall(sys_request_ipi_irq(),vector);
 7   syscall(317,1,100);   // syscall(sys_send_ipi(),core_id,vecotr);
 8   return 0;
 9 }
```

本手順書において，vector の値は 32-238 の範囲のどのベクタにでも割り当てることが可能である．ただし，ベクタ 128 は Linux のシステムコール実装に利用しているため割り当てることができない [1]．6 行目の 318 は sys_request_ipi_irq() のシステムコール番号であり，7 行目の 317 は sys_send_ipi() のシステムコール番号である．また，各コアが別の割り込みハンドラに対応していることを確認するには，7 行目の core_id の値を 0 から 3 までの 4 パターンの値で指定し，テストプログラムを作成する．

5.4 テストプログラムの実行

5.3 節で作成したプログラムをコンパイルし実行する．実行後，以下のコマンドを実行することでカーネルのメッセージバッファを確認する．

```
$ dmesg
```

実行後に表示されるメッセージの一部抜粋したものを以下に示す．

```
[   70.439337] request_ipi_start
[   70.439342] genirq: Flags mismatch irq 16. 00000000 \
(message_handler0) vs. 00000080 (ehci_hcd:usb1)
[   70.439503] request_ipi_start
[   70.439504] genirq: Flags mismatch irq 16. 00000000 \
(message_handler1) vs. 00000080 (ehci_hcd:usb1)
[   70.439651] genirq: Flags mismatch irq 17. 00000000 \
(message_handler1) vs. 00000000 (message_handler1)
[   70.439805] genirq: Flags mismatch irq 18. 00000000 \
(message_handler1) vs. 00000080 (ahci)
```

(中略)

```
[   70.442076] genirq: Flags mismatch irq 42. 00000000 \
```

```
(message_handler3) vs. 00014a00 (hpet4)
[   70.442222] genirq: Flags mismatch irq 43. 00000000 \
(message_handler3) vs. 00014a00 (hpet5)
```

上記のメッセージは，割り込みハンドラを登録する処理の際に出力されるメッセージである．また，IPI を送信するシステムコールが実装されていれば，割り込みハンドラの登録処理終了後に，以下のようなメッセージが得られる．

```
[   70.442408] Recieved IPI 1
```

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，実装が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．さらに，本手順書では，指定したコアに IPI を送信できたか否かを確認するため，各コアに別の IRQ 番号を対応させ，異なる割り込みハンドラを登録する方法を示した．

7 付録

以下に IPI を送信するシステムコールとテスト用システムコールのソースコード例を添付する．テスト用システムコールは, IPI を送信するシステムコールと同様, /home/hashimoto/git/linux-stable/kernel 以下の ipi_syscall.c にソースコードを記述する．

(1) IPI を送信するシステムコール

```
102 asmlinkage void sys_send_ipi(int core_id, int vector)
103 {
104   int phys_apicid;
105
106   phys_apicid = per_cpu(x86_bios_cpu_apicid, core_id); // コアを指定する
107
108   apic_icr_write(APIC_INT_ASSERT | APIC_DEST_PHYSICAL |
109           vector, phys_apicid); // vector で指定する割り込みベクタを通知する
110
111   apic_wait_icr_idle(); // IPI 送信が受け入れられるまで待ち状態に入る
112 }
```

(2) 割り込みハンドラ

```
 19 /* 割り込みハンドラ message_handler0() */
 20 irqreturn_t message_handler0(int irq,void *dev){
 21   printk("Recieved IPI 0\n");
 22   return IRQ_HANDLED;
 23 }
 24
 25 /* 割り込みハンドラ message_handler1() */
 26 irqreturn_t message_handler1(int irq,void *dev){
 27   printk("Recieved IPI 1\n");
 28   return IRQ_HANDLED;
 29 }
 30
 31 /* 割り込みハンドラ message_handler2() */
```

```
32 irqreturn_t message_handler2(int irq,void *dev){
33   printk("Recieved IPI 2\n");
34   return IRQ_HANDLED;
35 }
36
37 /* 割り込みハンドラ message_handler3() */
38 irqreturn_t message_handler3(int irq,void *dev){
39   printk("Recieved IPI 3\n");
40   return IRQ_HANDLED;
41 }
```

(3) 割り込みハンドラを登録するシステムコール

本手順書の実装環境において，IRQ 番号は 16-45 の範囲で設定可能である．このため，割り込みハンドラは IRQ 番号 16-45 の範囲の未登録なエントリに登録する．

```
48 asmlinkage int sys_request_ipi_irq(int vector)
49 {
50   int irq, rec[4];
51   int ret;
52   int i;
53
54   printk("request_ipi_start\n"); // message_handler0() 登録処理
55   for(irq = 16; irq < 46; irq++){
56     ret = request_irq(irq, message_handler0,
57      0x00000000, "message_handler0", NULL);
58     if(ret == 0) break;
59   }
60   if(ret != 0) return ret;
61   rec[0] = irq;
62
63   printk("request_ipi_start\n"); // message_handler1() 登録処理
64   for(irq = 16; irq < 46; irq++){
65     ret = request_irq(irq, message_handler1,
66      0x00000000, "message_handler1", NULL);
67     if(ret == 0) break;
68   }
69   if(ret != 0) return ret;
70   rec[1] = irq;
71
72   printk("request_ipi_start\n"); // message_handler2() 登録処理
73   for(irq = 16; irq < 46; irq++){
74     ret = request_irq(irq, message_handler2,
75      0x00000000, "message_handler2", NULL);
76     if(ret == 0) break;
77   }
78   if(ret != 0) return ret;
79   rec[2] = irq;
80
81   printk("request_ipi_start\n"); // message_handler3() 登録処理
82   for(irq = 16; irq < 46; irq++){
83     ret = request_irq(irq, message_handler3,
84      0x00000000, "message_handler3", NULL);
85     if(ret == 0) break;
86   }
87   if(ret != 0) return ret;
88   rec[3] = irq;
89
90   for(i = 0; i < NUM_OF_CORE; i++) // 各コアのベクタ表の指定したエントリに，
91     per_cpu(vector_irq, i)[vector] = rec[i]; // 対応した IRQ 番号を登録する
92
93   return irq;
```

```
94 }
```

参考文献

[1] Daniel P. Bovet and Marco Cesati, 詳解 Linux カーネル第 3 版, 株式会社オライリー・ジャパン, Feb 2017.

[2] Linus Torvalds, linux/kernel-parameters.txt at master torvalds/linux, Linux Foundation, available from <https://github.com/torvalds/linux/blob/master/Documentation/admin-guide/kernel-parameters.txt> (accessed 2017-05-24).

[3] Richard Verhoeven and Michael Hamilton and Andries Brouwer and Federico Lucifredi, Man page of BOOTPARAM, Technische Universiteit Eindhoven, available from <https://linuxjm.osdn.jp/html/LDP_man-pages/man7/bootparam.7.html> (accessed 2017-05-24)