日曜研究室

技術的な観点から日常を綴ります

[xv6 #24] Chapter 2 – Traps, interrupts, and drivers – Code: The first system call

テキストの33〜34ページ

本文

前の章は、initcode.Sがシステムコールを呼び出すところで終わった。
もう一回そこを見てよう。
（initcode.Sの最初のint $T_SYSCALLのところ）
プロセスは、execを呼ぶための引数をそのプロセスのスタックにプッシュし、%eaxにシステムコール番号をセットする。
システムコール番号は、関数ポインタのテーブルであるsyscallの配列のエントリのどれかを指す。
（syscall.hとsyscall.cの関数ポインタテーブルを参照）
我々は、int命令がユーザモードからカーネルモードへ切り替え、カーネルが正しいカーネルの関数（例えばsys_exec）を呼び出し、カーネルがsys_execの引数を回復出来る手はずを整える必要がある。
次のいくつかの節で、システムコールのためのそれをxv6がどうやって手はずを整えるか説明する。
そうしたら割り込みと例外のために同じコードが再利用出来ることを我々は発見するだろう。

initcode.S

```
01 # Initial process execs /init.
02
03 #include "syscall.h"
04 #include "traps.h"
05
06
07 # exec(init, argv)
08 .globl start
09 start:
10   pushl $argv
11   pushl $init
12   pushl $0  // where caller pc would be
13   movl $SYS_exec, %eax
14   int $T_SYSCALL
15
```

```
16 # for(;;) exit();
17 exit:
18   movl $SYS_exit, %eax
19   int $T_SYSCALL
20   jmp exit
21
22 # char init[] = "/init\0";
23 init:
24   .string "/init\0"
25
26 # char *argv[] = { init, 0 };
27 .p2align 2
28 argv:
29   .long init
30   .long 0
```

syscall.h

```
01 // System call numbers
02 #define SYS_fork    1
03 #define SYS_exit    2
04 #define SYS_wait    3
05 #define SYS_pipe    4
06 #define SYS_read    5
07 #define SYS_kill    6
08 #define SYS_exec    7
09 #define SYS_fstat   8
10 #define SYS_chdir   9
11 #define SYS_dup    10
12 #define SYS_getpid 11
13 #define SYS_sbrk   12
14 #define SYS_sleep  13
15 #define SYS_uptime 14
16
17 #define SYS_open   15
18 #define SYS_write  16
19 #define SYS_mknod  17
20 #define SYS_unlink 18
21 #define SYS_link   19
22 #define SYS_mkdir  20
23 #define SYS_close  21
```

syscall.cの関数ポインタテーブル

```
01 static int (*syscalls[])(void) = {
02 [SYS_fork]    sys_fork,
03 [SYS_exit]    sys_exit,
04 [SYS_wait]    sys_wait,
05 [SYS_pipe]    sys_pipe,
06 [SYS_read]    sys_read,
07 [SYS_kill]    sys_kill,
08 [SYS_exec]    sys_exec,
09 [SYS_fstat]   sys_fstat,
10 [SYS_chdir]   sys_chdir,
11 [SYS_dup]     sys_dup,
12 [SYS_getpid]  sys_getpid,
13 [SYS_sbrk]    sys_sbrk,
14 [SYS_sleep]   sys_sleep,
15 [SYS_uptime]  sys_uptime,
16 [SYS_open]    sys_open,
17 [SYS_write]   sys_write,
18 [SYS_mknod]   sys_mknod,
```

```
19 [SYS_unlink]  sys_unlink,
20 [SYS_link]    sys_link,
21 [SYS_mkdir]   sys_mkdir,
22 [SYS_close]   sys_close,
23 };
```

感想

前々回感想に「例外でも割り込みが発生するし、システムコールでも発生するし、デバイスからも発生するということみたいです。」と書きました。
システムコールの呼び出しの処理を見ると、割り込みと例外にも同じような処理が流用出来る風な事がここでも書いてあり、やはりそのあたりは関連してるみたいですね。

カテゴリー: 技術 | タグ: xv6 | 投稿日: 2012/2/29 水曜日 [http://peta.okechan.net/blog/archives/1378] |