System Call Implementation on Minix 3

Felipe Caballero

Universidad Adolfo Ibáñez, Santiago de Chile
Sistemas Operativos 2006-Segundo Semestre

(Note: English is not my primary language, there may be some errors)
(Any observations please right to loco.loop+minix3(at)gmail(dot)com)

Table Of Contents

Introduction

System Calls (SC) are “The invocation of an operating system routine. Operating systems contain sets of routines for performing various low-level operations.”[1]. Implementing a System Call isn’t a very complex task but it requires some knowledge of how the Operating System works.

Implementing the System Call per se

Minix 3 is based on servers, two of them are important: FS and PM. FS (File System) takes care about file handling (create, delete, etc.) and PM (Process Manager) takes care about everything to do with system’s processes’.

This paper is about implementing a System Call (SC) on Minix 3 that returns the PID (Process ID) and the PPID (Parent Process ID).

Once the SC has been implemented, there has to be some way to verify the new SC’s functionality. For this we’re going to implement a program called sc_test.c that compares our SC’s result to the original SC’s result, for this we use getpid() and getppid() native functions. We use a method called assert to compare results.

Steps to Follow

These are the steps to follow to implement our SC:

1- Edit the file /usr/src/include/minix/callnr.h:
-Increase the total number of SC on the system (from 95 to 96)
-Create a line at the bottom of the file with:

```
#define GETALLPID 95
```

2- Edit the file /usr/src/servers/pm/proto.h:
-Below /* getset.c */ we define our SC’s prototype:

```
_PROTOTYPE( int do_getpids, (void)                    );
```

3-Edit the file /usr/src/servers/pm/table.c:
-insert the line do_getpids, /* 95 = get parent pid and own pid */
(It has to be on the right position, in our case position 95, it’s the same specified in step 1)

[1]Definition from http://www.angelfire.com/anime3/internet/opersys.htm

4-Edit /usr/src/servers/fs/table.c

-insert the line do_getpids, /* 95 = get parent pid and own pid */

(It has to be on the right position, in our case position 95, it’s the same specified in step 1)

5-Edit /usr/src/servers/pm/getset.c:

-insert the following function at the end:

```
/*===========================================================================*
 *                                do_getpids                                 *
 *===========================================================================*/
PUBLIC int do_getallpid()
{
/* This method gets the process' pid and its parents'.
 */
        register struct mproc *rmp = mp;
        /* mp is a pointer to current process */
        int proc;

        if(pm_isokendpt(m_in.endpt, &proc) == OK && proc >= 0)
        {
                rmp->mp_reply.reply_res3 = mproc[who_p].mp_pid;
                /* mproc[who_p] represents current process */

                rmp->mp_reply.reply_res2 = mproc[rmp->mp_parent].mp_pid;
                /* mproc[rmp->mp_parent] represents parent process */

                return 0;
                /* If nothing fails 0 is returned */
        }
        else
        return 1;
        /* if some error occurred */
}
```

6-Create the file /usr/src/include/getpids.h with:

```
#include <lib.h>

PUBLIC int getpids(pid_t *current, pid_t *parent)
{
  message m;

  *current = 999;
  *parent = 999;
/* pids' pointers are asigned with arbitrary values initially */

  _syscall(MM, GETALLPID, &m);
  /* actual system call gets executed */

   *current = m.m2_i2;
/* current process' pid is assigned to pointer of current process information pointer */

  *parent= m.m2_i1;
/* parent process' pid is assigned to pointer of parent process information pointer */

   return 0;
}
```

7- Create necessary libraries (to include getpids.h in /usr/include/)

```
cd /usr/src/tools
make includes
```

8-Recompile kernel

```
	cd /usr/src/tools/
	make hdboot /* this makes compiled image to start by default */
	            /* in Minix' initial menu, option 3 must be selected */
```

10-Create sc_test.c:

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <getpids.h>

#include <assert.h>
#include <stdio.h>

#define N 4

void assert_getpids_works(void) {
  pid_t expected_me = getpid();
  pid_t expected_parent = getppid();
  pid_t actual_me, actual_parent;

  int result = getpids(&actual_me, &actual_parent);
  assert(0 == result);
  assert(actual_me == expected_me);
  assert(actual_parent == expected_parent);
}

main(void) {
  int i;

  assert_getpids_works();

  for (i=0; i<N; i++) {
    if (0 == fork()) {
      assert_getpids_works();
      return 0;
    } else {
      int status;
      wait(&status);
    }
  }
  printf("OK\n");
  return 0;
}
```

Conclusion

To create a System Call on Minix is a fairly easy task, but it requieres deep knowledge of the system's file structure.

References

Istruzioni per Minix - http://matteo.vaccari.name/so/minix

Adding System Calls in Minix - http://wwwcsif.cs.ucdavis.edu/~engle/ta/ecs150-f03/syscall.html

Agregar un System Call a Minix - http://www.midnightsoret.com.ar/personales/alejandrovaldez/minix/minixSystemCall.html

Minix Group on Google - http://groups.google.com/group/comp.os.minix?lnk=li