Introducción a los Sistemas Operativos

Introducción – IV

Anexo llamadas al Sistema

Objetivo

- Programar un llamado a una “System Call” de manera directa. Sin utilizar ninguna librería.
- Considerar distintos aspectos al intentar realizar lo mismo en las siguientes arquitecturas:
  - 32 bits
  - 64 bits

Hello World!!

- Para programar el clasico “hello world” se necesitan mínimo realizar hacer 2 llamadas al sistema:
  - Una para escribir en pantalla un mensaje
    SYSCALL WRITE
  - Otra para terminar la ejecución de un proceso
    SYSCALL EXIT

Hello World!!

- Para obtener información sobre estas SYSCALLs podemos utilizar los manuales del sistema.
- El comando man permite acceder a distintos tipos de documentación, en particular a información referida a systemcalls
  - write (man 2 write)
  - exit (man exit)

Hello World!!!

- Los manuales de las system calls permiten saber cuales son los parámetros

```
NAME
       write - write to a file descriptor

SYNOPSIS
       #include <unistd.h>

       ssize_t write(int fd, const void *buf, size_t count);

DESCRIPTION
       write()  writes  up  to  count bytes from the buffer pointed buf to the
       file referred to by the file descriptor fd.
```

```
NAME
       exit - cause normal process termination

SYNOPSIS
       #include <stdlib.h>

       void exit(int status);

DESCRIPTION
       The  exit() function causes normal process termination and the value of
       status & 0377 is returned to the parent (see wait(2)).
```

Número de syscalls a utilizar

- Para indicarle al sistema operativo lo que queremos hacer (write o exit), es necesario saber cuál es el número asociado que tiene cada una de las syscalls
- Puede ser distinto en distintas arquitucturas

Del github de Linus Torvald

- https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_32.tbl
- https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_64.tbl

Hello World en x86 32bit

https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_32.tbl

```
# 32-bit system call numbers and entry vectors
#
# The format is:
# <number> <abi> <name> <entry point> <compat entry point>
#
# The abi is always "i386" for this file.
#
0       i386    restart_syscall         sys_restart_syscall
1       i386    exit                    sys_exit
2       i386    fork                    sys_fork                        sys_fork
3       i386    read                    sys_read
4       i386    write                   sys_write
5       i386    open                    sys_open                        compat_sys_open
6       i386    close                   sys_close
```

En x86 32bit las sistem calls tienen los siguientes números:

- write → syscall número 4
- exit → syscall número 1

Hello World en x86 64bit

https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_64.tbl

```
# 64-bit system call numbers and entry vectors
#
# The format is:
# <number> <abi> <name> <entry point>
#
# The abi is "common", "64" or "x32" for this file.
#
0       common  read                    sys_read
1       common  write                   sys_write
2       common  open                    sys_open
3       common  close                   sys_close
```

```
57      common  fork                    sys_fork/ptregs
58      common  vfork                   sys_vfork/ptregs
59      64      execve                  sys_execve/ptregs
60      common  exit                    sys_exit
61      common  wait4                   sys_wait4
62      common  kill                    sys_kill
```

En x86 64bit las sistem calls tienen los siguientes números:

- write → syscall número 1
- exit → syscall número 60

Pasaje de parámetros en x86 32bit

- https://syscalls.kernelgrok.com/
  - EAX lleva el numero de syscall que se desea ejecutar
  - EBX lleva el primer parámetro
  - ECX lleva el segundo parámetro
  - EDX …
  - ESI
  - EDI

Instrucción que inicia la system call: **int 80h**

Pasaje de parámetros en x86 64bit

- http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/
  - EAX lleva el numero de syscall que se desea ejecutar
  - RDI lleva el primer parámetro
  - RSI lleva el segundo parámetro
  - RDX …
  - R10
  - R8
  - R9

Instrucción que inicia la system call: **syscall**

Hello world en x86 32 bit

```
NAME
       write - write to a file descriptor

SYNOPSIS
       #include <unistd.h>

       ssize_t write(int fd, const void *buf, size_t count);

DESCRIPTION
       write()  writes  up  to  count b
       file referred to by the file des
```

```
# 32-bit system call numbers and entr
#
# The format is:
# <number> <abi> <name> <entry point>
#
# The abi is always "i386" for this f
#
0       i386    restart_syscall
1       i386    exit
2       i386    fork
3       i386    read
4       i386    write
5       i386    open
6       i386    close
```

```
NAME
       exit - cause normal process term

SYNOPSIS
       #include <stdlib.h>

       void exit(int status);

DESCRIPTION
       The  exit() function causes normal process termination and the value of
       status & 0377 is returned to the parent (see wait(2)).
```

```
_start:

    ; sys_write(stdout, message, length)

    mov eax, 4          ; sys_write syscall
    mov ebx, 1          ; stdout
    mov ecx, message    ; message address
    mov edx, 14         ; message string length
    int 80h

    ; sys_exit(return_code)

    mov eax, 1          ; sys_exit syscall
    mov ebx, 0          ; return 0 (success)
    int 80h

section .data
    message: db 'Hello, world!',0x0A    ; message and newline
```

Hello world en x86 64 bit

```
NAME
       write - write to a file descriptor

SYNOPSIS
       #include <unistd.h>

       ssize_t write(int fd, const void *buf, size_t count);

DESCRIPTION
       write()  writes  up  to  count bytes from the buffer pointed buf to the
       file referred to by the file descriptor fd.
```

- write → syscall número 1
- exit → syscall número 60

```
; sys_write(stdout, message, length)

mov rax, 1  ; sys_write
mov rdi, 1  ; stdout
mov rsi, message ; message address
mov rdx, length ; message string length
syscall

; sys_exit(return_code)

mov rax, 60  ; sys_exit
mov rdi, 0  ; return 0 (success)
syscall

section .data
 message: db 'Hello, world!',0x0A ; message and newline
 length: equ 14 ;
```

```
NAME
       exit - cause normal process term

SYNOPSIS
       #include <stdlib.h>

       void exit(int status);

DESCRIPTION
       The  exit() function causes normal process termination and the value of
       status & 0377 is returned to the parent (see wait(2)).
```

Resumen

- Los manuales del sistema indican los parámetros necesarios para activar una system call

```
NAME
       write - write to a file descriptor

SYNOPSIS
       #include <unistd.h>

       ssize_t write(int fd, const void *buf, size_t count);

DESCRIPTION
       write()  writes  up  to  count bytes from the buffer pointed buf to the
       file referred to by the file descriptor fd.
```

- Dependiendo la arquitectura, cambiará:
  - el número de system call utilizado para realizar una función determinada
  - La forma de pasar los parámetros al kenel

Resumen

- Los procesadores 32 bit y 64 bits usan un esquema de registros diferentes.

- Los procesadores 32 bit y 64 bits usan una instrucción distinta para activar las systemcalls:
  - 32 bits: **int 80h**
  - 64 bits: **syscall**

Referencias

Como programar un “hello world” en x86 32bit y 64bit

- http://shmaxgoods.blogspot.com.ar/2013/09/assembly-hello-world-in-linux.html
- https://stackoverflow.com/questions/19743373/linux-x86-64-hello-world-and-register-usage-for-parameters

Mas información sobre formas de pasar parametros a una syscall

- https://github.com/torvalds/linux
- https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_32.tbl
- https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_64.tbl
- https://syscalls.kernelgrok.com/
- http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/
- http://www.int80h.org/bsdasm/#system-calls