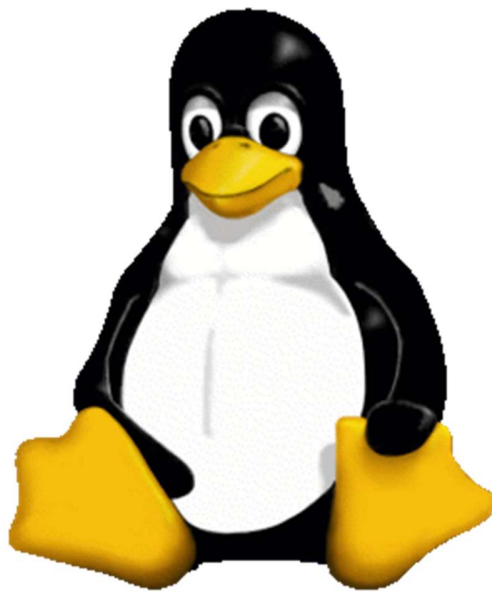

AÑADIR UNA NUEVA LLAMADA AL SISTEMA EN LINUX



Memoria realizada por:

Marcel Andrei Voicu

Carlos Javier Hellín Asensio

Grupo: GII Tarde

Se instala Ubuntu 64-bit en una máquina virtual configurada con el número de núcleos del procesador para compilar el kernel más rápido.

Desde el terminal de Ubuntu se instalan los siguientes paquetes:

```
$ sudo apt-get install build-essential linux-source libncurses5-dev libssl-dev
```

build-essential – Paquetes esenciales para poder compilar (gcc, make, etc..).

linux-source – El código fuente del núcleo de Linux.

libncurses5-dev – Ficheros de desarrollos para ncurses5. Usado en el menú de configuración antes de compilar el núcleo.

libssl-dev – dependencia requerida por el núcleo.

Se descomprime el código fuente y se modifica para añadir una nueva llamada al sistema.

```
$ tar xvf /usr/src/linux-source-*.tar.bz2
```

```
$ cd linux-source-*/
```

Se agrega la función prototipo de la nueva llamada al sistema al fichero **syscalls.h** que contiene las llamadas sin arquitectura específica.

```
$ vi include/linux/syscalls.h
```

al final, antes del **#endif** se añade:

```
asmlinkage long sys_holamundo(void);
```

sys_holamundo será el punto de entrada a la nueva llamada del sistema.

Se edita **sys.c** para implementar la nueva función, aunque existen otros ficheros para llamadas de sistemas o incluso se puede crear un nuevo fichero .c teniendo en cuenta las modificaciones apropiadas al Makefile.

```
$ vi kernel/sys.c
```

al final del todo se añade:

```
SYSCALL_DEFINE0(holamundo)
{
    printk("Hola mundo!\n");
    return 0;
}
```

Se utiliza la macro SYSCALL_DEFINE n () como la manera estándar de definir una llamada del sistema en el código del núcleo, donde n indica que el número de argumentos que se le pasan a la llamada del sistema. Con printk se muestra un mensaje del núcleo para comprobar que realmente sys_holamundo funciona.

Por último, se edita **syscall_64.tbl** debido a que algunas arquitecturas, como por ejemplo x86, tienen sus propias tablas de llamadas al sistema para cada arquitectura específica. Esto también hace que ejecutar las llamadas del sistema sean más rápidas al acceder por índice a un array de punteros con funciones y no con if o switch.

```
$ vi arch/x86/entry/syscalls/syscall_64.tbl
```

después de:

```
# <number> <abi> <name> <entry point>
```

```
...
```

```
...
```

```
332          common          statx          sys_statx
```

```
333          64              holamundo      sys_holamundo
```

Se añade a la última entrada disponible y que en este caso tiene como número el 333. Es un número importante que se usará en los últimos pasos.

Se compila y se instala el núcleo de Linux modificado con la nueva llamada del sistema y después se reinicia. Con -j\$(nproc) nos aseguramos de usar los núcleos del procesador que se ha configurado anteriormente en la máquina virtual.

```
$ sudo make menuconfig
$ sudo make -j$(nproc)
$ sudo make modules_install install
$ sudo reboot
```

Al reiniciar se entra al GRUB dejando pulsado Shift durante el arranque de la máquina virtual y se elige la versión del núcleo que se ha compilado e instalado.

Para comprobar que la llamada del sistema funciona se programa en C lo siguiente:

```
$ vi prueba.c
```

```
#include <unistd.h>
```

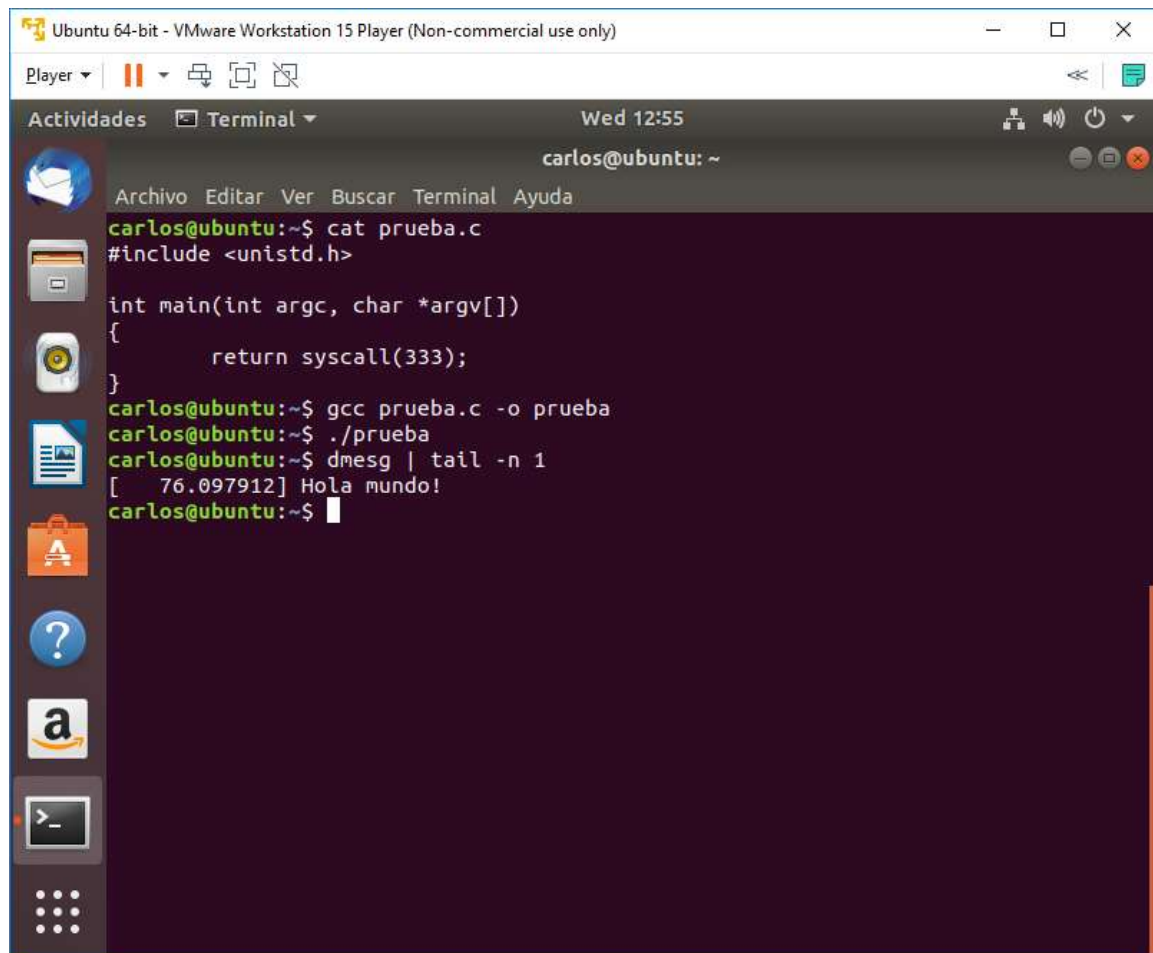
```
int main(int argc, char *argv[])
{
    return syscall(333);
}
```

Debido a que no hay una rutina “wrapper” en la librería de C de la nueva llamada del sistema se utiliza la función `syscall()` con el número 333 que se había especificado anteriormente en la tabla `syscall_64.tbl`

```
$ gcc prueba.c -o prueba
$ ./prueba
```

```
$ dmesg | tail -n 1
```

```
[ 212.771583] Hola mundo!
```



The screenshot shows a terminal window titled "Ubuntu 64-bit - VMware Workstation 15 Player (Non-commercial use only)". The terminal is running on a system named "carlos@ubuntu: ~". The user has executed the following commands:

```
carlos@ubuntu:~$ cat prueba.c
#include <unistd.h>

int main(int argc, char *argv[])
{
    return syscall(333);
}
carlos@ubuntu:~$ gcc prueba.c -o prueba
carlos@ubuntu:~$ ./prueba
carlos@ubuntu:~$ dmesg | tail -n 1
[ 76.097912] Hola mundo!
carlos@ubuntu:~$
```

The terminal output shows the execution of the program, which returns the value 333. The user then uses the `dmesg` command to view the kernel log, showing the message "[76.097912] Hola mundo!".

Para entender el funcionamiento de las llamadas del sistema a bajo nivel se ha realizado esto como parte opcional.

Se programa en ensamblador lo siguiente:

```
$ vi holamundo.S
```

```
.intel_syntax noprefix
```

```
.text
```

```
.globl holamundo
```

```
holamundo:
```

```
    mov rax, 333 // número de índice de sys_holamundo en el registro rax
```

```
    syscall // invoca a la llamada del sistema
```

```
    mov rdi, rax // el retorno de sys_holamundo al primer parámetro de sys_exit
```

```
    mov rax, 60 // número de índice sys_exit en rax
```

```
    syscall // invoca a sys_exit para que termine el proceso
```

```
$ vi holamundo.h
```

```
void holamundo(void);
```

```
$ vi prueba.c
```

```
#include "holamundo.h"
```

```
void _start()
```

```
{
```

```
    holamundo();
```

```
}
```

Se compila sin enlazar con libc:

```
$ gcc -nostdlib holamundo.S prueba.c -o prueba
```

```
$ ./prueba
```

```
$ dmesg | tail -n 1
```

```
[ 1547.987619] Hola mundo!
```

```
$ strace ./prueba
```

```
...
```

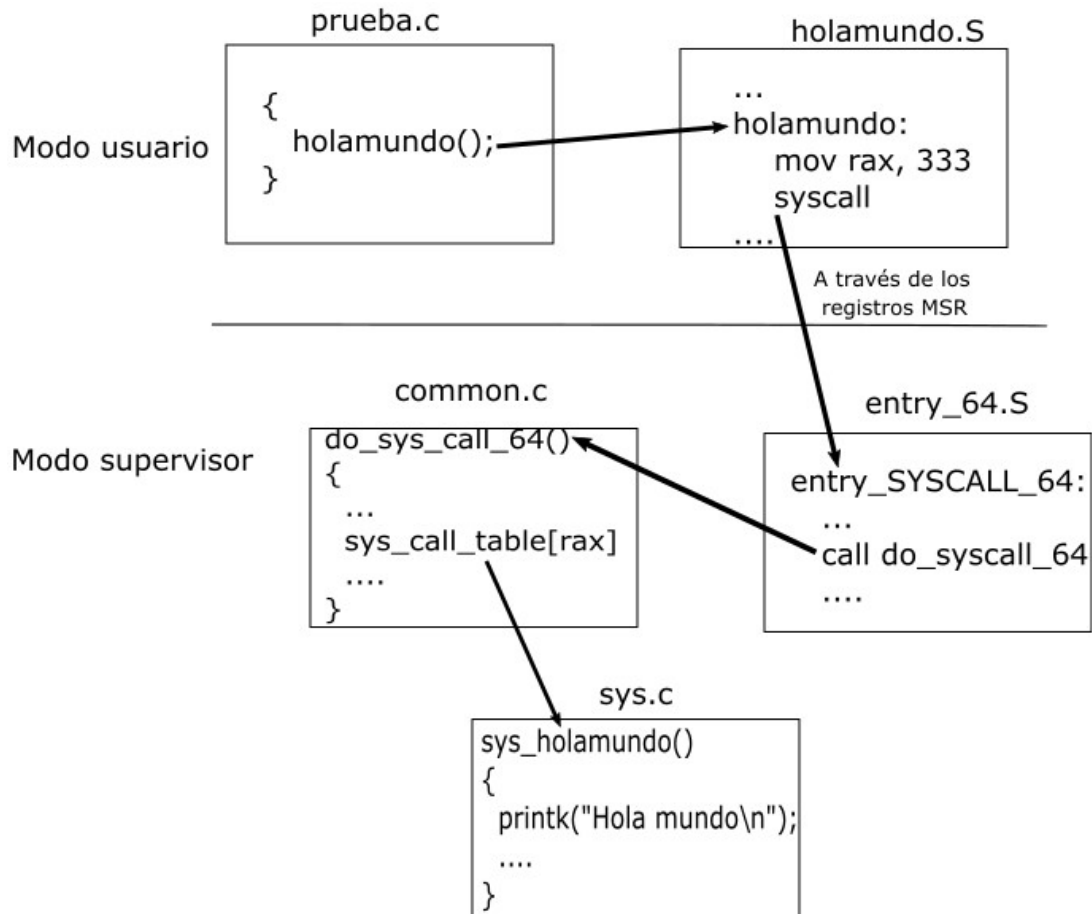
```
syscall_0x14d(0x7f3ae65db170, 0x7f3ae65db700, 0x7f3ae63c29a0, 0x7f3ae65db170, 0, 0) = 0
```

```
exit(0)                                = ?
```

```
+++ exited with 0 +++
```

En definitiva, una llamada al sistema en x86-64 hace los siguientes pasos:

- El número de la llamada del sistema se prepara en el registro rax y si hubiese parámetros se guardan en los registros específicos de la CPU (rdi, rsi, rdx...). Después se ejecuta la instrucción syscall.
- La instrucción syscall cambia del modo usuario al modo supervisor.
- Se lee el registro MSR_LSTAR de los registros MSR (model-specific register) que almacena la dirección de memoria a entry_SYSCALL_64 previamente referenciado en syscall_init() en arch/x86/kernel/cpu/common.c durante la ejecución de arranque del kernel.
- entry_SYSCALL_64 en arch/x86/entry/entry_64.S se encarga, entre otras cosas, de llamar al manejador de la llamada al sistema do_syscall_64()
- do_syscall_64() en arch/x86/entry/common.c lee la tabla con sys_call_table[rax] usando el registro rax como índice
- Ejecuta la función encontrada en ese índice, que en nuestro caso sería sys_holamundo()



Fuentes:

Adding a New System Call <https://www.kernel.org/doc/html/latest/process/adding-syscalls.html>

Anatomy of a system call, part 1 <https://lwn.net/Articles/604287/>

Searchable Linux Syscall Table for x86 and x86_64 <https://filippo.io/linux-syscall-table/>

System V Application Binary Interface

<https://software.intel.com/sites/default/files/article/402129/mpx-linux64-abi.pdf>

Intel 64 and IA-32 Architectures Software Developer's Manual

<https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-instruction-set-reference-manual-325383.pdf>

Linux system calls <http://man7.org/linux/man-pages/man2/syscalls.2.html>

syscall – indirect system call <http://man7.org/linux/man-pages/man2/syscall.2.html>

Linux kernel source tree <https://github.com/torvalds/linux/>