PRÁCTICA 4: LLAMADAS AL SISTEMA OPERATIVO:

ÍNDICE

SALIDA.C:

```
#gcc -m32 -g -o salida salida.c
#include <stdlib.h>
void main (void)
{
        exit (0xFF);
}
```

SALIDA.C:

```
##gcc -m32 -g -o salida salida.c
/* Llamada al sistema desde C
   Prototipo: int syscall(int number, ...);
   man syscall
*/
#define _GNU_SOURCE
#include <unistd.h>
#include <sys/syscall.h>
void main (void)
{
        syscall (__NR_exit,0xFF);
}
```

Los siguientes dos programas hacen llamadas al sistema operativo devolviendo el número 0xFF en hexadecimal. El primero de ellos, a través de la función exit, y el segundo a través de syscall. Como se puede observar, ejecutando el programa salida y ejecutando el comando echo $?, se muestra el valor que se ha devuelto al sistema operativo.

0xFF =0b 1111 1111 = 255 (decimal) = 15*16 + 15 * 1

```
eduardo@eduardo-Victus-by-HP-Laptop-16-e0xxx:~/Escritorio/Eduardo Ezponda/ESTRUCTURA DE COMPUTADORES/P4$ ./salida0
eduardo@eduardo-Victus-by-HP-Laptop-16-e0xxx:~/Escritorio/Eduardo Ezponda/ESTRUCTURA DE COMPUTADORES/P4$ echo $?
255
eduardo@eduardo-Victus-by-HP-Laptop-16-e0xxx:~/Escritorio/Eduardo Ezponda/ESTRUCTURA DE COMPUTADORES/P4$
```

SALIDA.S:

```
##Compilación: gcc -m32 -g -nostartfiles -o salida salida.s
        .global _start
        .section .text
_start:
        push $0xFF #return code
        call exit #libc library
.end
```

SALIDA.S:

```
##gcc -m32 -g -nostartfiles -o salida salida.s
        .global _start
        .section .text
_start:
        push $0xFF #return code
        push $1 # exit syscall code
        call syscall #libc library
.end
```

Como se puede observar, hacemos en primer lugar un examine tras realizar un primer push de 0xFF, para mostrar en tamaño word (2bytes) el valor de la cima de la pila (%esp).
Con lo cual, se imprime 255 en decimal.
En siguiente lugar, se ejecuta un next (n) para realizar el siguiente push (apilar) de 1 a la pila.
Por lo tanto, se imprime los valores 1, 255 en tamaño word en ese orden tras tratarse de una estructura LIFO (Last IN, First OUT).

```
(gdb) x /w $esp
0xffffd15c:     255
(gdb) n
(gdb) x /2w $esp
0xffffd158:     1       255
(gdb)
```

SALIDA.S:

```
##gcc -m32 -g -nostartfiles -o salida salida.s
        .global _start
        .section .text
_start:
        mov $1,%eax #exit
        mov $0xFF,%ebx #argument
        int $0x80 #system call
        .end
```

Los siguientes tres modelos de salida.s, realizan al ejecutarse lo mismo que el salida.c solo que de tres maneras distintas. En primer lugar, haciendo un “push” de 0xFF a la pila y más tarde llamando a exit.
En segundo lugar, se hace un “push” de 0xFF y de 1 para llamar a la función syscall.
Por último, se mueve al registro %eax el 1 para realizar el exit, y más tarde se carga 0xFF en el registro %ebx, el cual se utiliza por convención para devolver su valor al sistema operativo, para finalmente realizar una llamada al sistema operativo con int $0x80.

IMPRIMIR.S: (Compilación gcc -m32 -g -nostartfiles -o imprimir imprimir.s)

```
        .section .data
planet:
        .long 9 # variable planet
        .section .rodata
mensaje:
        .asciz "El número de planetas es %d \n" #string con formato de la función printf
        .global _start
        .section .text
_start:
        ## imprimir en la pantalla
        push planet # 2º argumento de la función printf
        push $mensaje # 1º argumento de la función printf: dirección del string
        call printf
        ## salir al sistema
        push $0
        call exit
```

En el siguiente programa se declara como variable planet, cuyo valor es 9 en tamaño long. Además, se declara como constante mensaje, que es una string. Su declaración se realiza a traves de .asciz, con lo cual tendrá un 0 al final de la string.

En el desarrollo del programa, se hace dos “push” del valor 9 procedente de planet y de la dirección de la string mensaje, cuyos parámetros son necesarios para más adelante llamar a la función printf y así mostrar por pantalla el resultado.
Como se puede observar en la parte superior del programa con la declaración de mensaje, se visualiza %d que indica que se va a mostrar en esa posición un entero, que en este caso se realiza con un casteo del long 9 de planet.

SYSCALL_WRITE_PUTS.C:

```c
/*
Programa syscall_write_puts.c
Descripción: Realiza la llamada al sistema operativo para imprimir en la
pantalla
Realiza la llamada de tres formas diferentes: puts,write,syscall.
Compilación: gcc -m32 -g -o puts_gets puts_gets.c
*/
// Cabeceras de librerías
#include <stdio.h> // prototipo de la función puts()
#include <unistd.h> //declaración de las macros STDOUT_FILENO, STDIN_FILENO
#include <syscall.h> //declaración de la función syscall
#include <sys/syscall.h> // declaración de la macro __NR_write y __NR_exit
#include <stdlib.h> //declaración de exit()
// Macros
#define LON_BUF 5 // Tamaño del string
void main (void)
{
        char buffer[LON_BUF]="Hola\n";
        puts("\n***************** Práctica : LLAMADAS AL SISTEMA ******************\n"); // función puts() de la librería libc
        puts("\n***************** Imprimo el mensaje de bienvenida mediante la función write(): ");
        write(STDOUT_FILENO, buffer,LON_BUF); // wrapper de la llamada al sistema write.
        // ya que write() incluye un syscall(), llama indirectamente al sistema
        puts("\n***************** Imprimo el mensaje de bienvenida mediante la llamada al sistema syscall(): ");
        syscall(__NR_write,STDOUT_FILENO,buffer,LON_BUF); // función syscall de llamada directa al sistema.
        exit(0xAA); //Salir al sistema enviando el código 0xAA. No es lo mismo que retornar.
}
```

En el siguiente programa se muestra por pantalla la string buffer a través de la llamada de la función write o a través de syscall.

SYSCALL_WRITE_PUTS.S:

```
        .section .data
mensaje:
        .asciz  "\n***************** Práctica : LLAMADAS AL SISTEMA ******************\n"
mensaje2:
        .asciz "\n***************** Imprimo el mensaje de bienvenida mediante la función write(): "
mensaje3:
        .asciz "\n***************** Imprimo el mensaje de bienvenida mediante la llamada al sistema syscall(): "
hola:
        .asciz "Hola\n"

        .global _start
        .section .text


# llamada a la función puts de la librería libc. Es necesrio linkar con libc.
_start:
#pila <-argumento
#call puts
        push $mensaje
        call puts

        push $mensaje2
        call puts
# llamada a la función write de la librería libc. Es necesario linkar con libc.
#pila <- 3º argumento
#pila <- 2º argumento
#pila <- 1º argumento
#call write
        push $5
        push $hola
        push $1
        call write

        push $mensaje3
        call puts

# LLamada al sistema operativo para ejecutar la operación write
#EAX<-4
#EBX<-1
#ECX<-etiqueta que apunta al string a imprimir
#EDX<-5
        mov $4, %eax
        mov $1, %ebx
        mov $hola, %ecx
        mov $5, %edx
        int $0x80
#call sistema_operativo
# Llamada al sistema operativo para ejecutar la operación exit
#EAX<-1
#EBX<-0
        mov $1,%eax
        mov $0,%ebx
        int $0x80
        .end
```

En primer lugar, se declaran en la parte superior del archivo las cuatro strings que se desean imprimir por pantalla. El resultado de la ejecución tiene que ser el mismo que tras ejecutar el archivo .c.

Para ello, se imprime por pantalla la string mensaje para introducir el programa. Se realiza se apila la dirección de mensaje para posteriormente llamar a la función puts que imprimirá por pantalla dicha string.

A partir de ahí, se quiere imprimir por pantalla a través de la función write los tres argumentos de la pila. Por lo tanto, se realizan tres push para pasar a la pila los tres argumentos necesarios para llamar a la función. Como los argumentos de la funcion write son fd (descriptor fichero, STDOUT_FILENO), string (buffer), y tamaño string (LON_BUF, y la pila es una estructura LIFO, se tendrán que apilar a la pila en el orden inverso. En primer lugar, el tamaño (5), en segundo lugar buffer, y por último STDOUT_FILENO (1, imprimir pantalla).

```
        char buffer[LON_BUF]="Hola\n";
        puts("\n****************** Práctica : LLAMADAS AL SISTEMA ******************\n"); // función puts() de la librería libc
        puts("\n****************** Imprimo el mensaje de bienvenida mediante la función write(): ");
        write(STDOUT_FILENO, buffer,LON_BUF); // wrapper de la llamada al sistema write.
        // ya que write() incluye un syscall(), llama indirectamente al sistema
        puts("\n****************** Imprimo el mensaje de bienvenida mediante la llamada al sistema syscall(): ");
        syscall(__NR_write,STDOUT_FILENO,buffer,LON_BUF); // función syscall de llamada directa al sistema.
        exit(0xAA); //Salir al sistema enviando el código 0xAA. No es lo mismo que retornar.
```

Sin embargo, al finalizar el programa, se quiere imprimir por pantalla de una manera diferente. En este caso, se cargan los valores 4 (tamaño entero, int 0x80), 1 (STDOUT_FILENO, pantalla), hola (dirección string) y 5 (tamaño string) a los registros eax, ebx, ecx y edx, respectivamente. Este orden se debe otra vez a la estructura LIFO que invierte el orden de los argumentos introducidos en la pila.

A través de int 0x80 se llama al sistema operativo, y se imprime la string hola.

Para finalizar el programa, se hace un exit pasando 1 al registro eax, y se devuelve al sistema operativo el valor 0 con el registro ebx para comprobar el correcto funcionamiento del programa.
Se llama al sistema operativo de nuevo a partir de int 0x80.

```
eduardo@eduardo-Victus-by-HP-Laptop-16-e0xxx:~/Escritorio/Eduardo Ezponda/ESTRUCTURA DE COMPUTADORES/P4$ ./syscall_write_puts

****************** Práctica : LLAMADAS AL SISTEMA ******************


****************** Imprimo el mensaje de bienvenida mediante la función write():
Hola


****************** Imprimo el mensaje de bienvenida mediante la llamada al sistema syscall():
Hola
eduardo@eduardo-Victus-by-HP-Laptop-16-e0xxx:~/Escritorio/Eduardo Ezponda/ESTRUCTURA DE COMPUTADORES/P4$
```

COMPILACIÓN:

gcc -m32 -nostartfiles -g -o syscall_write_puts syscall_write_puts.s

gcc -m32 -g -o syscall_write_puts syscall syscall_write_puts syscall.c

Cabe destacar el uso de -m32 para usar una máquina de 32 bits, y -g para cargar la tabla de símbolos. Además se ha añadido el -nostartfiles porque en este caso se utiliza la etiqueta _start.

CONCLUSIÓN:

Durante ésta práctica, se ha hecho uso de llamadas al sistema a través de call o a través de int $0x80. Para ello, se tendrán que cargar los valores correspondientes a los registros o hacer diferentes “push” o apilar a la pila para posteriormente llamar al sistema operativo. En el caso de cargar valores en los registros, dará igual el orden de las instrucciones si luego se quiere llamar al sistema operativo para imprimir la cadena de strings correspondientes.
En el caso de la pila, se deberá tener en cuenta que invierte el orden de los argumentos, y por ello se debe primero visualizar los argumentos de la función, para luego ir apilándolos en el orden inverso a su declaración.

El valor de los valores de la pila se puede conocer a través de %esp que es un puntero que apunta a la cima de la pila (stack pointer).