x86-write

¿Por qué se le resta 1 al resultado de sizeof?

Porque sizeof devuelve la cantidad en bytes de la cadena que en este caso sería 15 porque se cuenta el '\0' implícito del final el cual, para no imprimirlo, se resta el 1 al resultado de sizeof.

¿Funcionaría el programa si se declarase msg como const char *msg = "...";? ¿Por qué?

No, porque al declararlo como const char *msg el sizeof toma la variable como si fuera un puntero que, al ser en x86/32 bits, devuelve 4 y al restarle 1 el resultado es 3 haciendo que sólo imprima los primeros 3 caracteres (Hel).

Examinar, con objdump -S libc_hello, el código máquina generado e indicar el valor de len en la primera instrucción push. Explicar el efecto del operador . en la línea .set len, . - msg.

El valor de len en el primer push es 0xE (14 en decimal). El . indica la dirección de memoria actual (la cual es justo después de que termina msg) a la cual restada la posición de memoria inicial de msg se obtiene el largo de este.

Mostrar un hex dump de la salida del programa en assembler. Cambiar la directiva .ascii por .asciz y mostrar el hex dump resultante con el nuevo código. ¿Qué está ocurriendo? ¿Qué ocurre con el valor de len?

```
.ascii
0000000  48  65  6c  6c  6f  2c  20  77  6f  72  6c  64  21  0a
          H   e   l   l   o   ,       w   o   r   l   d   !  \n
0000016

.asciz
0000000  48  65  6c  6c  6f  2c  20  77  6f  72  6c  64  21  0a  00
          H   e   l   l   o   ,       w   o   r   l   d   !  \n  \0
0000017
```

En el caso de asciz se toma en cuenta el '\0' y se aumenta en 1 el valor de len.

x86-call

Mostrar en una sesión de GDB cómo imprimir las mismas instrucciones usando la directiva x $pc y el modificador adecuado.

```
(gdb) x/9i $pc
=> 0x8049196 <main>:    push   $0x804c024
   0x804919b <main+5>:  call   0x8049050 <strlen@plt>
   0x80491a0 <main+10>: mov    %eax,%esi
   0x80491a2 <main+12>: push   %esi
   0x80491a3 <main+13>: push   $0x804c024
   0x80491a8 <main+18>: push   $0x1
   0x80491aa <main+20>: call   0x8049070 <write@plt>
   0x80491af <main+25>: push   $0x7
   0x80491b1 <main+27>: call   0x8049040 <_exit@plt>
```

Después, usar el comando stepi (step instruction) para avanzar la ejecución hasta la llamada a write. En ese momento, mostrar los primeros cuatro valores de la pila justo antes e inmediatamente después de ejecutar la instrucción call, y explicar cada uno de ellos.

Antes del write:

```
0xffffcfcc: 0x00000001  0x0804c024  0x0000000e  0x0804c024
```

Los 3 primeros valores corresponden a los 3 argumentos de write a los que se les hizo push:

```
      push %esi //  0x0000000e
      push $msg //  0x0804c024
      push $1 // 0x00000001
```

En el caso del 4to valor (0x0804c024) corresponde al mensaje que se le hizo push antes de call strlen para calcular su longitud.

Al momento de ejecutar write:
0xffffcfc8: 0x080491af 0x00000001 0x0804c024 0x0000000e
Los últimos 3 valores corresponden a los ya mencionados previamente mientras que el nuevo valor (0x0804c024) corresponde a la dirección de retorno.

x86-libc
Mostrar la salida de nm --undefined para este nuevo binario (int80_hi.S).

```
      w __gmon_start__
      U __libc_start_main@@GLIBC_2.0
```

¿Qué significa que un registro sea callee-saved en lugar de caller-saved?
Significa que un registo callee-saved será usado por la función invocada y que su valor va a ser guardado en el stack antes de ser modificado por la función y se les reestablecerá su valor al volver a la función invocante.

En x86 ¿de qué tipo, caller-saved o callee-saved, es cada registro según la convención de llamadas de GCC?

caller-saved: %eax, %ecx, %edx.

callee-saved: %ebx, %esi, %edi, %ebp

En el archivo sys_strlen.S del punto anterior, renombrar la función main a _start, y realizar las siguientes cuatro pruebas de compilación:
int80_hi.S con -nodefaultlibs, luego con -nostartfiles
sys_strlen.S con -nodefaultlibs, luego con -nostartfiles
Mostrar el resultado (compila o no) en una tabla 2 × 2, así como los errores de compilación.

	int80_hi.S	*sys_strlen.S*
-nodefaultlibs	/usr/bin/ld: /usr/lib/gcc/x86_64-linux-gnu/9/../../../../lib32/Scrt1.o: in function `_start': (.text+0x1c): undefined reference to `__libc_csu_fini' /usr/bin/ld: (.text+0x23): undefined reference to `__libc_csu_init' /usr/bin/ld: (.text+0x31): undefined reference to `__libc_start_main' collect2: error: ld returned 1 exit status	/usr/bin/ld: /tmp/cc4Vk7tf.o: in function `_start': (.text+0x0): multiple definition of `_start'; /usr/lib/gcc/x86_64-linux-gnu/9/../../../../lib32/Scrt1.o:(.text+0x0): first defined here /usr/bin/ld: /usr/lib/gcc/x86_64-linux-gnu/9/../../../../lib32/Scrt1.o: in function `_start': (.text+0x1c): undefined reference to `__libc_csu_fini' /usr/bin/ld: (.text+0x23): undefined reference to `__libc_csu_init' /usr/bin/ld: (.text+0x2c): undefined reference to `main' /usr/bin/ld: (.text+0x31): undefined reference to `__libc_start_main' /usr/bin/ld: /tmp/cc4Vk7tf.o: in function `_start': (.text+0x6): undefined reference to `strlen' collect2: error: ld returned 1

		exit status
-nostartfiles	/usr/bin/ld: warning: cannot find entry symbol _start; defaulting to 0000000000001000	Compila sin errores.

¿Alguno de los dos archivos compila con -nostdlib?
No.

Mostrar la salida de nm --undefined para el binario sys_strlen, y explicar las diferencias respecto a int80_hi.

```
         U strlen@@GLIBC_2.0
```

La diferencia con int80_hi es que en sys_strlen sólo hay una llamada directa a strlen, mientras que en int80_hi la llamada se hace a _start.

x86-ret
Se pide ahora modificar int80_hi.S para que, en lugar de invocar a _exit(), la ejecución finalice sencillamente con una instrucción ret. ¿Cómo se pasa en este caso el valor de retorno?
Se pasa por el registro %eax.

x86-ebp
¿Qué valor sobreescribió GCC cuando usó mov $7, (%esp) en lugar de push $7 para la llamada a _exit?
Sobreescribió el return address.

La versión C no restaura el valor original de los registros *%esp* y *%ebp*. Cambiar la llamada a _exit(7) por return 7, y mostrar en qué cambia el código generado. ¿Se restaura ahora el valor original de *%ebp*?

```
Original con _exit(7):
7         _exit(7);
   0x080491d9 <+35>:    movl   $0x7,(%esp)
   0x080491e0 <+42>:    call   0x8049070 <_exit@plt>

Con return 7:
7         return 7;
   0x080491b9 <+35>:    mov    $0x7,%eax
   0x080491be <+40>:    mov    -0x4(%ebp),%ecx
   0x080491c1 <+43>:    leave
   0x080491c2 <+44>:    lea    -0x4(%ecx),%esp
   0x080491c5 <+47>:    ret
```

El valor de %ebp es restaurado en la instrucción leave que es equivalente a:
mov %ebp, %esp
pop %ebp

Crear un archivo llamado lib/exit.c, ¿Qué ocurre con %ebp?
El valor de %ebp es restaurado al final de la función.

En hello.c, cambiar la declaración de my_exit a:
extern void __attribute__((noreturn)) my_exit(int status); y verificar qué ocurre con %ebp, relacionándolo con el significado del atributo noreturn.
El valor de %ebp no es restablecido en main dado que al llamar a my_exit esta función no devolverá ningún valor.

x86-argv
En sys_argv, ¿qué hay en 4(%esp)?

Está argv[0] que es el string ./sys_argv

En libc_argv, ¿qué hay en (%esp)?
Return address.

x86-frames
¿Dónde se encuentra (de haberlo) el primer argumento de f?
Se encuentra en 8(%ebp)

¿Dónde se encuentra la dirección a la que retorna f cuando ejecute ret?
Se encuentra en 4(%ebp).

¿Dónde se encuentra el valor de %ebp de la función anterior, que invocó a f?
Se encuentra en %ebp.

¿Dónde se encuentra la dirección a la que retornará la función que invocó a f?
Se encuentra en 4(%ebp).

Una sesión de GDB en la que se muestre la equivalencia entre el comando bt de GDB y el código implementado; en particular, se debe incluir:

La salida del comando bt al entrar en la función backtrace:

```
(gdb) b backtrace
Breakpoint 1 at 0x80491f1: file backtrace.c, line 10.
(gdb) r
Starting program: /home/ben/Documents/Organización del computador/Lab2/backtrace

Breakpoint 1, backtrace () at backtrace.c:10
10    void backtrace() {
(gdb) bt
#0  backtrace () at backtrace.c:10
#1  0x0804925a in my_write (fd=2, msg=0x804a03c, count=15) at backtrace.c:23
#2  0x080492bb in recurse (level=0) at backtrace.c:32
#3  0x080492a5 in recurse (level=1) at backtrace.c:30
#4  0x080492a5 in recurse (level=2) at backtrace.c:30
#5  0x080492a5 in recurse (level=3) at backtrace.c:30
#6  0x080492a5 in recurse (level=4) at backtrace.c:30
#7  0x080492a5 in recurse (level=5) at backtrace.c:30
#8  0x080492d1 in start_call_tree () at backtrace.c:36
#9  0x080492e5 in main () at backtrace.c:40
```

La salida del programa al ejecutarse la función backtrace (el número de frames y sus direcciones de retorno deberían coincidir con la salida de bt):

```
(gdb) until 19
#1 [0xffffcee8] 0x804925a ( 0x2 0x804a03c 0xf )
#2 [0xffffcf08] 0x80492bb ( 0 0xf63d4e2e 0xf7ffdafc )
#3 [0xffffcf28] 0x80492a5 ( 0x1 0x1 0xf7fcd410 )
#4 [0xffffcf48] 0x80492a5 ( 0x2 0xc10000 0x1 )
#5 [0xffffcf68] 0x80492a5 ( 0x3 0xffffd074 0xf7fb5000 )
#6 [0xffffcf88] 0x80492a5 ( 0x4 0xf7fe39e0 0 )
#7 [0xffffcfa8] 0x80492a5 ( 0x5 0xffffd074 0xffffd07c )
#8 [0xffffcfc8] 0x80492d1 ( 0xf7fb5000 0xf7fb5000 0 )
#9 [0xffffcfd8] 0x80492e5 ( 0x1 0xffffd074 0xffffd07c )
```

Usando los comandos de selección de frames, y antes de salir de la función backtrace, el valor de %ebp en cada marco de ejecución detectado por GDB (valores que también deberían coincidir):

```
backtrace () at backtrace.c:19
19        print_newline();
(gdb) up
#1  0x0804925a in my_write (fd=2, msg=0x804a03c, count=15) at backtrace.c:23
23        backtrace();
(gdb) p/x $ebp
$1 = 0xffffcee8
(gdb) up
#2  0x080492bb in recurse (level=0) at backtrace.c:32
32            my_write(2, "Hello, world!\n", 15);
(gdb) p/x $ebp
$2 = 0xffffcf08
(gdb) up
#3  0x080492a5 in recurse (level=1) at backtrace.c:30
30            recurse(level - 1);
(gdb) p/x $ebp
$3 = 0xffffcf28
(gdb) up
#4  0x080492a5 in recurse (level=2) at backtrace.c:30
30            recurse(level - 1);
(gdb) p/x $ebp
$4 = 0xffffcf48
(gdb) up
#5  0x080492a5 in recurse (level=3) at backtrace.c:30
30            recurse(level - 1);
(gdb) p/x $ebp
$5 = 0xffffcf68
(gdb) up
#6  0x080492a5 in recurse (level=4) at backtrace.c:30
30            recurse(level - 1);
(gdb) p/x $ebp
$6 = 0xffffcf88
(gdb) up
#7  0x080492a5 in recurse (level=5) at backtrace.c:30
30            recurse(level - 1);
(gdb) p/x $ebp
$7 = 0xffffcfa8
(gdb) up
#8  0x080492d1 in start_call_tree () at backtrace.c:36
36        recurse(5);
(gdb) p/x $ebp
$8 = 0xffffcfc8
(gdb) up
#9  0x080492e5 in main () at backtrace.c:40
40        start_call_tree();
(gdb) p/x $ebp
$9 = 0xffffcfd8
(gdb) up
Initial frame selected; you cannot go up.
(gdb) p/x $ebp
$10 = 0xffffcfd8
(gdb) frame 0
#0  backtrace () at backtrace.c:19
```