Trabajo Práctico 0

[7542/9508] Taller de Programación I
Cátedra Veiga
Segundo cuatrimestre de 2020

Jonathan David Rosenblatt - 104105

Índice

1. PASO 0: Entorno de Trabajo **2**
1.1. Output del programa "main.c"sin y con valgrind respectivamente. 2
1.2. b) Utilidades de Valgrind y opciones más comunes. 2
1.3. c) Función sizeof() 3
1.4. d) Sizeof() meets structs 3
1.5. e) STDIN, STDOUT y STDERR 3

1. PASO 0: Entorno de Trabajo

1.1. Output del programa "main.c"sin y con valgrind respectivamente.

```
niyo: /Taller-de-Programacion-9508/tp0$ gcc main.c -o tp
niyo: /Taller-de-Programacion-9508/tp0$ ./tp
Hola mundo!
niyo: /Taller-de-Programacion-9508/tp0$ valgrind ./tp
==7545== Memcheck, a memory error detector
==7545== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==7545== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==7545== Command: ./tp
==7545==
Hola mundo!
==7545==
==7545== HEAP SUMMARY:
==7545== in use at exit: 0 bytes in 0 blocks
==7545== total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==7545==
==7545== All heap blocks were freed - no leaks are possible
==7545==
==7545== For lists of detected and suppressed errors, rerun with: -s
==7545== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
niyo: /Taller-de-Programacion-9508/tp0$
```

1.2. b) Utilidades de Valgrind y opciones más comunes.

Valgrind es un software cuya finalidad está mayoritariamente en facilitar el debugging de un programa, permitiéndote saber cuando tenés pérdidas de memoria en tu código o rastrear facilmente variables no inicializadas, entre otros casos.

Un típico ejemplo donde Valgrind puede ser muy útil, es cuando usamos memoria dinámica y nos estamos olvidando de liberarla. Si tuvieramos fugas de memoria en el heap, esto estaría indicado en lo que vemos por salida estándar.

En el ejemplo antes dado vemos el mensaje
`All heap blocks were freed - no leaks are possible` lo que significa que no estamos dejando memoria en el heap sin liberar.

Las opciones más comunes de este son:

- `-leak-check=full` especifica por la salida estándar del programa todas las pérdidas de memoria que haya tenido el programa;.
- `-track-origins=yes` muestra el origen de las variables no inicializadas.
- `-track-fds=yes` muestra que archivos quedaron abiertos;.
- `-show-reachable=yes` muestra que bloques de memoria no liberados aun son alcanzables.

1.3. c) Función sizeof()

La función `sizeof()` recibe una variable y devuelve el tamaño del mismo en bytes. El valor que devuelve depende de cada arquitectura. En el caso de `sizeof(char)` el resultado es 1 ya que codificamos todo caracter con 8 bits y luego `sizeof(int)` es 4. Pero en casos como `sizeof(void*)` puede ocurrir que si estamos en una arquitectura de 32 bits el resultado sea 4 y en 64 bits el resultado sea 8 (es decir, la cantidad de bits necesarios para representar un puntero cambió entre arquitecturas).

1.4. d) Sizeof() meets structs

Sea el siguiente ejemplo de código en C en una arquitectura x86 de 64 bits:

```
struct tp0{
char c;
void* ptr;
};
```

Si tomamos `sizeof(struct tp0)` veremos que el resultado será de 16. Esto es ya que el offset del char c es de 0 y ocupa 1 byte, haciendo que void* ptr tenga que empezar con offset 1 en el struct, pero esto no ocurre ya que esta posición de memoria no sería múltiplo de su tamaño (es decir, no es múltiplo de 8). En consecuencia el compilador agrega 7 bytes extras entre los elementos del struct). Demostrando así que el valor esperado no es `sizeof(char) + sizeof(void*)`.

Queda entonces demostrado que no siempre ocurre que el `sizeof(struct` s`)` es igual a la suma de sus elementos. Esto ocurre por el padding.

$$\texttt{struct s\{type } s_1\texttt{; ...; type } s_n\texttt{\}} \not\Longrightarrow \texttt{sizeof(struct s)} = \sum_{i=1}^{n} \texttt{sizeof(}s_i\texttt{)}$$

1.5. e) STDIN, STDOUT y STDERR

Estos 3 archivos son la entrada estándar de datos, salida estándar de datos y salida de errores respectivamente.Son archivos que, dentro de los sistemas operativos UNIX, permanecen abiertos por cada proceso para comunicar información, ya sea entre procesos o hacia el usuario.

También se puede concatenar estas salidas y entradas. Por ejemplo, si ejecuto en la términal:

```
ls > archivos.txt
```

estaría escribiendo en mi archivo archivos.txt un listado de todos los documentos, programas y directorios que tenga almacenado en mi posición actual. Si en cambio ejecuto:

```
cat < archivos.txt
```

estaría haciendo que el ejecutable de la izquiera reciba una lista de archivos por stdin y finalmente los imprima. También tengo al pipe que concatena el stdout del ejecutable izquierdo con el stdin del ejecutable del lado derecho del mismo. Por ejemplo si ejecuto:

```
cat archivos.txt | grep .txt
```

estaría conectando lo que lanza por stdout el término izquierdo (que sería una lista de los documentos del directorio donde estoy parado) al stdin del término derecho (que recibe mi lista de

documentos y escribe por stdout los que tienen ".txt.^en el nombre).