Metodología de la Programación

DGIM

Curso 2020/2021

Guion de prácticas

DrMemory y memoria dinamica

Abril de 2021

Índice

1. Descripción **5**

2. Chequeo de memoria con DrMemory **5**

3. Ejemplos de uso de DrMemory **5**
3.1. Uso de memoria no inicializada 6
3.2. Lectura y/o escritura en memoria liberada 9
3.3. Sobrepasar los límites de un array en operación de lectura de memoria 11
3.4. Sobrepasar los límites de un array en operación de escritura en memoria 12
3.5. Problemas con delete sobre arrays 14

4. Integración de DrMemory en NetBeans **14**
4.1. Desde las propiedades del proyecto 14
4.2. Desde la consola del proyecto 15

1. Descripción

En esta práctica aprenderemos a utilizar la herramienta `drmemory` para diagnosticar varios problemas, entre otros, los problemas de mala gestión de memoria dinámica en un programa.

2. Chequeo de memoria con DrMemory

DrMemory es una plataforma de análisis del código. Contiene un conjunto de herramientas que permiten detectar problemas de memoria y también obtener datos detallados de la forma de funcionamiento (rendimiento) de un programa. Es una herramienta de libre distribución que puede obtenerse en `https://www.drmemory.org/`, está disponible para Linux, Windows y Mac e incorpora muchas herramientas de monitorización de la ejecución. Nosotros trabajaremos básicamente con la opción de chequeo de problemas de memoria

Principales opciones de **DrMemory**

- **-light**: Ejecuta un chequeo rápido de problemas de memoria, no todos, pero sí algunos importantes.
- **-check_leaks**: Analiza posibles pérdidas de memoria dinámica.
- Sin opciones. Se controla el uso de zonas de memoria que no habían primero el valor por defecto). El valor de este argumento debe estar en concordancia inicializadas.

La forma habitual de lanzar la ejecución de esta herramienta es la siguiente:

```
drmemory -- ./programa
```

Esta forma de ejecución ofrece información detallada sobre los posibles problemas en el uso de la memoria dinámica requerida por el programa. Iremos considerando algunos ejemplos para ver la salida obtenida en varios escenarios. En algunos programas, sobre todo cuando tenemos decenas de miles de operaciones de reserva y liberación de memoria, la ejecución puede ralentizarse un poco. En estos casos se puede ejecutar una versión más sencilla

```
drmemory -light -check_leaks -- ./programa
```

El cual funciona como la anterior llamada sólo que no hace el test de uso de zonas de memoria no inicializadas

Nota: Es preciso compilar los programas con la opción **-g** para que se incluya información de depuración en el ejecutable.

3. Ejemplos de uso de DrMemory

Se consideran a continuación algunos ejemplos de código con problemas usuales con gestión dinámica de memoria. Se analizan para ver qué

mensajes de aviso nos muestra esta herramienta en cada caso (marcados en rojo). En concreto, vamos a identificar los siguientes escenarios, todos ellos relacionados con el acceso a memoria para leer y/o escribir un valor en una variable: uso de memoria no inicializada, lectura y/o escritura en memoria ya liberada, sobrepasar los límites de un array en operación de lectura, sobrepasar los límites de un array en operación de lectura o escritura, problemas con delete sobre arrays, aviso sobre el uso masivo de memoria no inicializada.

3.1. Uso de memoria no inicializada

Imaginemos que el siguiente código se encuentra en un archivo llamado **ejemplo1.cpp**.

Listing 1: ejemplo1.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main(){
5    int array[5];
6    cout << array[3] << endl; // Memoria NO inicializada
7 }
```

Compilamos mediante la sentencia:

```
g++ -g -o ejemplo1 ejemplo1.cpp
```

Si ejecutamos ahora **drmemory** como hemos indicado antes:

```
drmemory -- ./ejemplo1
```

se obtiene un informe bastante detallado de la forma en que el programa usa la memoria dinámica. Parte del informe generado es:

```
~~Dr.M~~ Dr. Memory version 2.3.0
~~Dr.M~~
~~Dr.M~~ Error #1: UNINITIALIZED READ: reading register r13
~~Dr.M~~ # 0 libstdc++.so.6!std::num_put<>::_M_insert_int<>+0x77
~~Dr.M~~ # 1 libstdc++.so.6!std::ostream::_M_insert<>     +0x84
~~Dr.M~~ # 2 main [ejemplo1.cpp:6]
~~Dr.M~~ Note: @0:00:00.955 in thread 32481
~~Dr.M~~ Note: instruction: test   %r13 %r13
...
~~Dr.M~~
~~Dr.M~~ ERRORS FOUND:
~~Dr.M~~       0 unique,     0 total unaddressable access(es)
~~Dr.M~~     3 unique,   11 total uninitialized access(es)
~~Dr.M~~       0 unique,     0 total invalid heap argument(s)
~~Dr.M~~       0 unique,     0 total warning(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of leak(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of possible leak(s)
~~Dr.M~~ ERRORS IGNORED:
~~Dr.M~~      16 unique,    34 total,  90499 byte(s) of still-reachable allocation(s)
~~Dr.M~~          (re-run with "-show_reachable" for details)
~~Dr.M~~ Details: /tmp/Dr. Memory/DrMemory-ejemplo1.32481.000/results.txt
```

Si nos fijamos en los mensajes que aparecen en el informe anterior (**UNINITIALIZED READ:**) se alude a que los valores del array no han sido inicializados y que se pretende usar el contenido de una posición no inicializada. Incluso nos informa de la línea en la que aparece el error: **main [ejemplo1.cpp:6]**. Si arreglamos el código de forma conveniente y ejecutamos de nuevo, veremos que desaparecen los mensajes de error:

Listing 2: ejemplo1–ok.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main(){
5    int array[5]={1,2,3,4,5};
6    cout << array[3] << endl; // Memoria SI inicializada
7 }
```

El informe de que todo ha ido bien es el siguiente:

```
~~Dr.M~~ Dr. Memory version 2.3.0
4
~~Dr.M~~
~~Dr.M~~ NO ERRORS FOUND:
~~Dr.M~~       0 unique,     0 total unaddressable access(es)
~~Dr.M~~       0 unique,     0 total uninitialized access(es)
~~Dr.M~~       0 unique,     0 total invalid heap argument(s)
~~Dr.M~~       0 unique,     0 total warning(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of leak(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of possible leak(s)
~~Dr.M~~ ERRORS IGNORED:
~~Dr.M~~      16 unique,    34 total,  90499 byte(s) of still-reachable allocation(s)
~~Dr.M~~          (re-run with "-show_reachable" for details)
~~Dr.M~~ Details: /tmp/Dr. Memory/DrMemory-ejemplo1-ok.3176.000/results.txt
```

3.2. Lectura y/o escritura en memoria liberada

Supongamos ahora que el código analizado es el siguiente:

Listing 3: usoMemoriaLiberada.cpp

```
1  #include <cstdio>
2  #include <cstdlib>
3  #include <iostream>
4
5  using namespace std;
6
7  int main(void){
8      // Se reserva espacio para p
9      char *p = new char;
10
11     // Se da valor
12     *p = 'a';
13
14     // Se copia el caracter en c
15     char c = *p;
16
17     // Se muestra
18     cout << "Caracter c: " << c;
19
20     // Se libera el espacio
21     delete p;
22
23     // Se copia el contenido de p (YA LIBERADO) en c
24     c = *p;
25     return 0;
26 }
```

El análisis de este código ofrece el siguiente informe drmemory

```
~~Dr.M~~ Dr. Memory version 2.3.0
~~Dr.M~~
~~Dr.M~~ Error #1: UNADDRESSABLE ACCESS of freed memory: reading 0x0 1 byte(s)
~~Dr.M~~ # 0 main            [usoMemoriaLiberada.cpp:24]
~~Dr.M~~ Note: @0:00:00.918 in thread 3387
~~Dr.M~~ Note: next higher malloc: 0x00007f876e86f1a0-0x00007f876e86f5a0
Caracter c: a~~Dr.M~~
~~Dr.M~~ ERRORS FOUND:
~~Dr.M~~     1 unique,    1 total unaddressable access(es)
~~Dr.M~~        0 unique,     0 total uninitialized access(es)
~~Dr.M~~        0 unique,     0 total invalid heap argument(s)
~~Dr.M~~        0 unique,     0 total warning(s)
~~Dr.M~~        0 unique,     0 total,       0 byte(s) of leak(s)
~~Dr.M~~        0 unique,     0 total,       0 byte(s) of possible leak(s)
~~Dr.M~~ ERRORS IGNORED:
~~Dr.M~~       16 unique,    34 total,  90475 byte(s) of still-reachable allocation(s)
~~Dr.M~~           (re-run with "-show_reachable" for details)
~~Dr.M~~ Details: /tmp/Dr. Memory/DrMemory-usoMemoriaL.3387.000/results.txt
```

El informe indica explícitamente la línea del código en que se produce el error (línea 24): lectura inválida (sobre memoria ya liberada).

3.3. Sobrepasar los límites de un array en operación de lectura de memoria

Veremos ahora el mensaje obtenido cuando se sobrepasan los límites de un array:

Listing 4: ejemplo2.cpp

```
1 #include<iostream>
2 using namespace std;
3
4 int main(){
5     int *array=new int[5];
6     cout << array[10] << endl; // Lectura de memoria fuera
7 }                              // de limites validos
```

Entre los mensajes de error aparece ahora el siguiente texto (parte del informe completo generado):

```
~~Dr.M~~ Dr. Memory version 2.3.0
~~Dr.M~~
~~Dr.M~~ Error #1: UNADDRESSABLE ACCESS beyond heap bounds: reading 0x00007fde1c11c198-0x00007fde1c11c19c 4 byte(s)
~~Dr.M~~ # 0 main          [ejemplo2.cpp:6]
~~Dr.M~~ Note: @0:00:00.904 in thread 5428
~~Dr.M~~ Note: prev lower malloc:  0x00007fde1c11c170-0x00007fde1c11c184
~~Dr.M~~ Note: instruction: mov    (%rax) -> %eax
0
~~Dr.M~~
~~Dr.M~~ Error #2: LEAK 20 direct bytes 0x00007fde1c11c170-0x00007fde1c11c184 + 0 indirect bytes
~~Dr.M~~ # 0 replace_operator_new_array               [/drmemory_package/common/alloc_replace.c:2929]
~~Dr.M~~ # 1 main                    [ejemplo2.cpp:5]
~~Dr.M~~
~~Dr.M~~ ERRORS FOUND:
~~Dr.M~~     1 unique,    1 total unaddressable access(es)
~~Dr.M~~       0 unique,     0 total uninitialized access(es)
~~Dr.M~~       0 unique,     0 total invalid heap argument(s)
~~Dr.M~~       0 unique,     0 total warning(s)
~~Dr.M~~     1 unique,    1 total,    20 byte(s) of leak(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of possible leak(s)
~~Dr.M~~ ERRORS IGNORED:
~~Dr.M~~      16 unique,    34 total,  90475 byte(s) of still-reachable allocation(s)
~~Dr.M~~          (re-run with "-show_reachable" for details)
~~Dr.M~~ Details: /tmp/Dr. Memory/DrMemory-ejemplo2.5428.000/results.txt
```

Observad los dos errores que se reportan

1. Línea 6. **UNADDRESSABLE ACCESS beyond heap bounds: reading Ox.. 4 byte(s)** ocurrido en relación al espacio de memoria reservado anteriormente. Se ha excedido el límite de memoria reservada en 4 bytes, lo que equivale a un entero.

2. Observad también el error relativo a memoria reservada pero no liberada (20 bytes, que equivalen a 5 enteros) **LEAK 20 direct bytes**. En este caso, nos indica la línea número 5 en la que aparece la reserva de memoria que no se ha liberado.

3.4. Sobrepasar los límites de un array en operación de escritura en memoria

También es frecuente exceder los límites del array para escribir en una posición que ya no le pertenece (alta probabilidad de generación de **core**):

Listing 5: ejemplo3.cpp

```
1 #include<iostream>
2 using namespace std;
3
4 int main(){
5    int *array=new int[5];
6    for(int i=0; i <= 5; i++){
7       array[i]=i; // Escritura en memoria fuera de
8    }              // limite valido cuando i == 5
9 }
```

Además del problema mencionado con anterioridad, en este programa no se libera el espacio reservado al finalizar. La información ofrecida por **drmemory** ayudará a solucionar todos los problemas mencionados:

```
~~Dr.M~~ Dr. Memory version 2.3.0
~~Dr.M~~
~~Dr.M~~ Error #1: UNADDRESSABLE ACCESS beyond heap bounds: writing 0x00007fc50f6f5184-0x00007fc50f6f5188 4 byte(s)
~~Dr.M~~ # 0 main          [ejemplo3.cpp:7]
~~Dr.M~~ Note: @0:00:01.540 in thread 7814
~~Dr.M~~ Note: refers to 0 byte(s) beyond last valid byte in prior malloc
~~Dr.M~~ Note: prev lower malloc:  0x00007fc50f6f5170-0x00007fc50f6f5184
~~Dr.M~~ Note: instruction: mov    %eax -> (%rdx)
~~Dr.M~~
~~Dr.M~~ Error #2: LEAK 20 direct bytes 0x00007fc50f6f5170-0x00007fc50f6f5184 + 0 indirect bytes
~~Dr.M~~ # 0 replace_operator_new_array               [/drmemory_package/common/alloc_replace.c:2929]
~~Dr.M~~ # 1 main                    [ejemplo3.cpp:5]
~~Dr.M~~
~~Dr.M~~ ERRORS FOUND:
~~Dr.M~~     1 unique,    1 total unaddressable access(es)
~~Dr.M~~       0 unique,     0 total uninitialized access(es)
~~Dr.M~~       0 unique,     0 total invalid heap argument(s)
~~Dr.M~~       0 unique,     0 total warning(s)
~~Dr.M~~     1 unique,    1 total,   20 byte(s) of leak(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of possible leak(s)
~~Dr.M~~ ERRORS IGNORED:
~~Dr.M~~      15 unique,    33 total,  89451 byte(s) of still-reachable allocation(s)
~~Dr.M~~          (re-run with "-show_reachable" for details)
~~Dr.M~~ Details: /tmp/Dr. Memory/DrMemory-ejemplo3.7814.000/results.txt
```

Observad los dos mensajes de error indicados: escritura inválida de tamaño 4 (tamaño asociado al entero) y en el resumen de memoria usada (leak) se indica que hay memoria perdida (20 bytes formando parte de un bloque: 5×4 bytes).

Con esta información es fácil arreglar estos problemas:

Listing 6: ejemplo4.cpp

```
1  #include<iostream>
2  using namespace std;
3
4  int main(){
5      int *array=new int[5];
6      for(int i=0; i < 5; i++){
7          array[i]=i;
8      }
9      delete [] array;
10 }
```

Y de esta forma desaparecen todos los mensajes de error previos:

3.5. Problemas con delete sobre arrays

Otro problema habitual al liberar el espacio de memoria usado por un array suele consistir en olvidar el uso de los corchetes. Esto genera un problema de uso de memoria, ya que no se indica que debe eliminarse un array. El código y los mensajes correspondientes de **drmemory** aparecen a continuación:

Listing 7: ejemplo5.cpp

```
1  #include<iostream>
2  using namespace std;
3
4  int main(){
5      int *array=new int[5];
6      for(int i=0; i < 5; i++){
7          array[i]=i;
8      }
9      delete array; // Liberacion de memoria incorrecta
10 }
```

```
~~Dr.M~~ Dr. Memory version 2.3.0
~~Dr.M~~
~~Dr.M~~ Error #1: INVALID HEAP ARGUMENT: allocated with operator new[], freed with operator delete
~~Dr.M~~ # 0 replace_operator_delete              [/drmemory_package/common/alloc_replace.c:2975]
~~Dr.M~~ # 1 main                  [ejemplo5.cpp:9]
~~Dr.M~~ Note: @0:00:01.619 in thread 8763
~~Dr.M~~ Note: memory was allocated here:
~~Dr.M~~ Note: # 0 replace_operator_new_array              [/drmemory_package/common/alloc_replace.c:2929]
~~Dr.M~~ Note: # 1 main                  [ejemplo5.cpp:5]
~~Dr.M~~
~~Dr.M~~ ERRORS FOUND:
~~Dr.M~~       0 unique,     0 total unaddressable access(es)
~~Dr.M~~       0 unique,     0 total uninitialized access(es)
~~Dr.M~~     1 unique,    1 total invalid heap argument(s)
~~Dr.M~~       0 unique,     0 total warning(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of leak(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of possible leak(s)
~~Dr.M~~ ERRORS IGNORED:
~~Dr.M~~      15 unique,    33 total,  89451 byte(s) of still-reachable allocation(s)
~~Dr.M~~          (re-run with "-show_reachable" for details)
~~Dr.M~~ Details: /tmp/Dr. Memory/DrMemory-ejemplo5.8763.000/results.txt
```

El mensaje relevante aquí es **INVALID HEAP ARGUMENT**. Indica que no hizo la liberación de espacio (reservado en la línea 5) de forma correcta.

4. Integración de DrMemory en NetBeans

Igual que Doxygen, DrMemory no tiene soporte de integración en NetBeans a través de sus menús o botones, pero existen varias formas de hacerlo.

4.1. Desde las propiedades del proyecto

Igual que se pueden configurar las ejecuciones del programa para incluir parámetros en la llamada, se puede configurar la llamada a drmemory. Propiedades del proyecto - Run - Run Command (Figura 1)

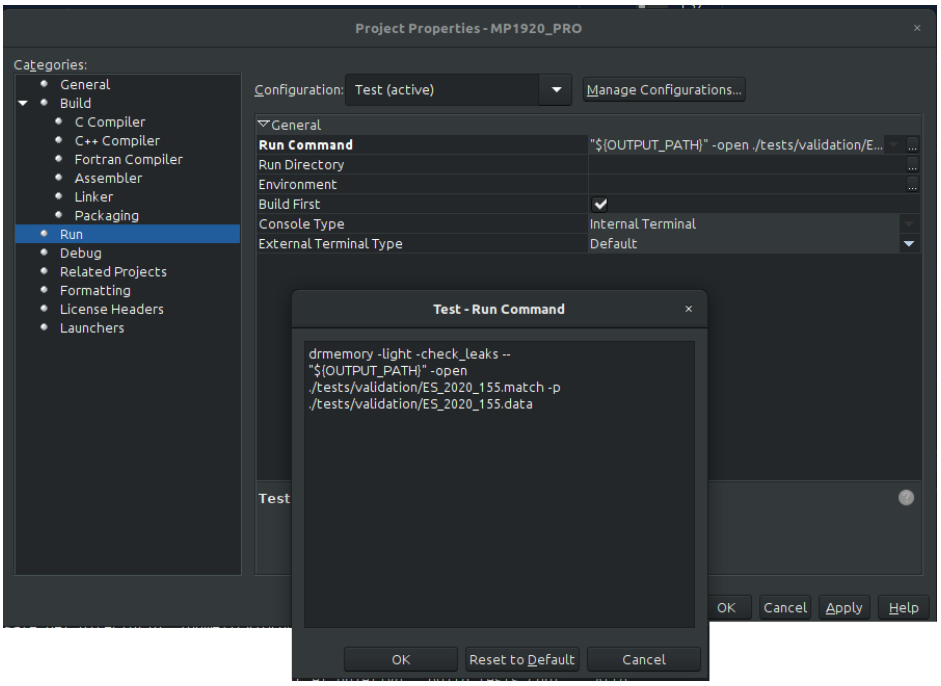

Figura 1: Incluir la llamada a DrMemory en las propiedades del proyecto

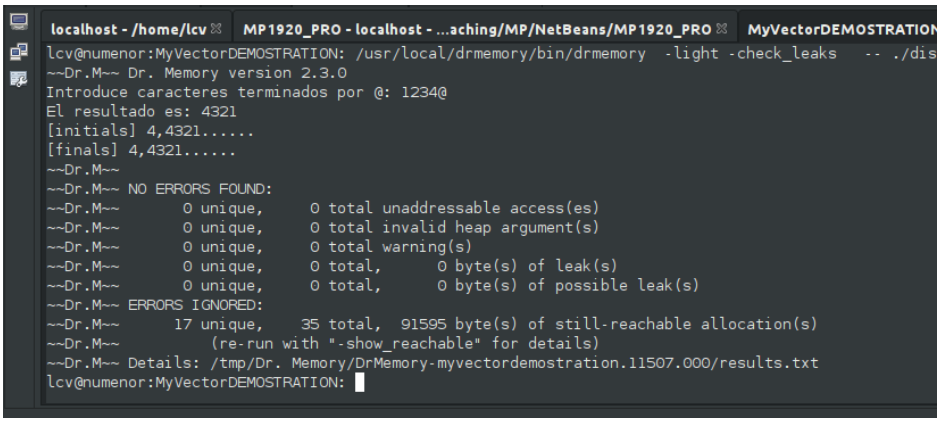

Figura 2: Ejecutar drmemory desde la consola de órdenes del proyecto

4.2. Desde la consola del proyecto

Ejecutar drmemory desde la línea de comandos integrada en el proyecto (Figura 2).