Valgrind.

Valgrind ofrece herramientas de código abierto para la depuración y optimización de programas en Linux. Aunque esté pensado para funcionar sobre todo con programas escritos en C y C++, puede usarse con programas escritos en cualquier lenguaje de programación. Valgrind dispone de varias herramientas, sin embargo, durante la práctica haremos uso sólo de Memcheck.

Memcheck detecta automáticamente los errores de gestión de memoria tales como:

- *Acceso indebido a memoria.* Se producen cuando se intenta acceder a zonas de memoria del heap no reservadas previamente.
- *Uso de variables no inicializadas.*
- *Lagunas de memoria (memory leaks)*. Se producen cuando no se libera la memoria.
- *Liberar memoria de forma incorrecta* (*free* incorrectos).

Estos tipos de errores no son siempre visibles y suelen ser difíciles de encontrar por otros medios, por ello, el uso de **Memcheck** es importante. Es imprescindible para que un programa **sea válido** que no tenga ninguno de los errores antes mencionados.

¿Cómo instalarlo?

En la mayoría de las distribuciones de Linux, se puede ejecutar desde la terminal:

sudo apt-get install valgrind

¿Como usar Memcheck?

1. Se compila el programa utilizando la opción ***-g,*** esto incluye información de depuración en el ejecutable para que los mensajes de error de Memcheck indiquen los números de línea exactos. Si el código fuente está guardado en un fichero con el nombre "test.c", entonces podemos compilarlo como:

   gcc -g test.c -o test

2. Si antes para ejecutar el programa utilizábamos:

   ./test arg1 arg2

   dentro de Valgrind, utilizaremos:

   valgrind --leak-check=full ./test arg1 arg2

donde **arg1** y **arg2** son los argumentos de entrada del programa.

3. Se interpreta la salida que contiene:

    1. Una cabecera;
    2. Para cada error se indica el tipo del error más la pila de ejecución donde se ha producido.
    3. Un resumen de la ejecución que contiene el número de errores más las reservas realizadas y la memoria que no se ha liberado.

EJEMPLO 1. Acceso indebido a memoria.

El siguiente código produce un error al intentar *acceder a una zona de memoria no reservada*:

```
1   #include <stdlib.h>
2   #define NMAX 12
3
4   void test(int * a) {
5
6           a[NMAX] = 0; //ERROR
7
8   }
9
10  int main(void) {
11
12      int* array = malloc(NMAX * sizeof (int));
13
14      test(array);
15
16      free(array);
17
18      return 0;
19  }
```

Al ejecutar el programa con Valgrind se mostrará la salida:

```
alessia@rio-PORTEGE-Z930:~/NetBeansProjects/Tests$ gcc -g main.c
alessia@rio-PORTEGE-Z930:~/NetBeansProjects/Tests$ valgrind --leak-check=full ./a.out
==14862== Memcheck, a memory error detector
==14862== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
==14862== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==14862== Command: ./a.out
==14862==
==14862== Invalid write of size 4
==14862==    at 0x400554: test (main.c:6)
==14862==    by 0x40057D: main (main.c:14)
==14862==  Address 0x51f2070 is 0 bytes after a block of size 48 alloc'd
==14862==    at 0x4C2B6CD: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==14862==    by 0x40056D: main (main.c:12)
==14862==
==14862==
==14862== HEAP SUMMARY:
==14862==     in use at exit: 0 bytes in 0 blocks
==14862==   total heap usage: 1 allocs, 1 frees, 48 bytes allocated
==14862==
==14862== All heap blocks were freed -- no leaks are possible
==14862==
==14862== For counts of detected and suppressed errors, rerun with: -v
==14862== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 2 from 2)
```

El error:

==14862== Invalid write of size 4
==14862== at 0x400554: test (main.c:6)
==14862== by 0x40057D: main (main.c:14)
==14862== Address 0x51f2070 is 0 bytes after a block of size 48 alloc'd
==14862== at 0x4C2B6CD: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==14862== by 0x40056D: main (main.c:12)

nos indica que estamos escribiendo 4 bytes en una zona de memoria no reservada. El error se ha producido por una llamada desde **main** (línea 12) de la función **test** que se encuentra en nuestro fichero main.c en la línea 6. Esta zona de memoria no reservada se encuentra después de una reserva de memoria que se ha hecho desde main (línea 12).

El código corregido:

```c
#include <stdlib.h>
#include <stdio.h>

#define NMAX 12

void test(int * a) {

    int i;
    for (i = 0; i< NMAX; i++)
        a[i] = 0;

}

int main(void) {

    int* array = malloc(NMAX * sizeof (int));
    if (!array) return -1;

    test(array);

    free(array);

    return 0;
}
```

EJEMPLO 2. Uso de variables no inicializadas.

El siguiente código produce errores al intentar utilizar *una variable no inicializada*:

```c
#include <stdlib.h>
#include <stdio.h>

#define NMAX 12

int main(int argc, char** argv) {


    int* array = malloc(NMAX * sizeof (int));

    if (!array) return -1;

    printf("%d ", array[0]); // ERROR

    free(array);

    return (EXIT_SUCCESS);
}
```

Los errores que se generan al ejecutar con Valgrind ocurren al llamar la función **printf** desde el programa **main** de main.c en la línea 13:

==17286== Conditional jump or move depends on uninitialised value(s)
==17286== at 0x4E7C4F1: vfprintf (vfprintf.c:1629)
==17286== by 0x4E858D8: printf (printf.c:35)
==17286== by 0x4005C8: main (main.c:13)
==17286==
==17286== Use of uninitialised value of size 8
==17286== at 0x4E7A7EB: _itoa_word (_itoa.c:195)
==17286== by 0x4E7C837: vfprintf (vfprintf.c:1629)
==17286== by 0x4E858D8: printf (printf.c:35)
==17286== by 0x4005C8: main (main.c:13)
==17286==
==17286== Conditional jump or move depends on uninitialised value(s)
==17286== at 0x4E7A7F5: _itoa_word (_itoa.c:195)
==17286== by 0x4E7C837: vfprintf (vfprintf.c:1629)
==17286== by 0x4E858D8: printf (printf.c:35)
==17286== by 0x4005C8: main (main.c:13)

La salida de Valgrind completa:

```
alessia@rio-PORTEGE-Z930:~/NetBeansProjects/Tests$ gcc -g main.c
alessia@rio-PORTEGE-Z930:~/NetBeansProjects/Tests$ valgrind --leak-check=full ./a.out
==18615== Memcheck, a memory error detector
==18615== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
==18615== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==18615== Command: ./a.out
==18615==
==18615== Conditional jump or move depends on uninitialised value(s)
==18615==    at 0x4E7C4F1: vfprintf (vfprintf.c:1629)
==18615==    by 0x4E858D8: printf (printf.c:35)
==18615==    by 0x4005C8: main (main.c:13)
==18615==
==18615== Use of uninitialised value of size 8
==18615==    at 0x4E7A7EB: _itoa_word (_itoa.c:195)
==18615==    by 0x4E7C837: vfprintf (vfprintf.c:1629)
==18615==    by 0x4E858D8: printf (printf.c:35)
==18615==    by 0x4005C8: main (main.c:13)
==18615==
==18615== Conditional jump or move depends on uninitialised value(s)
==18615==    at 0x4E7A7F5: _itoa_word (_itoa.c:195)
==18615==    by 0x4E7C837: vfprintf (vfprintf.c:1629)
==18615==    by 0x4E858D8: printf (printf.c:35)
==18615==    by 0x4005C8: main (main.c:13)
==18615==
0 ==18615==
==18615== HEAP SUMMARY:
==18615==     in use at exit: 0 bytes in 0 blocks
==18615==   total heap usage: 1 allocs, 1 frees, 48 bytes allocated
==18615==
==18615== All heap blocks were freed -- no leaks are possible
==18615==
==18615== For counts of detected and suppressed errors, rerun with: -v
==18615== Use --track-origins=yes to see where uninitialised values come from
==18615== ERROR SUMMARY: 3 errors from 3 contexts (suppressed: 2 from 2)
```

El código corregido:

```c
#include <stdlib.h>
#include <stdio.h>

#define NMAX 12

void test(int * a) {

    int i;
    for (i = 0; i< NMAX; i++)
        a[i] = 0;

}

int main(void) {

    int* array = malloc(NMAX * sizeof (int));
    if (!array) return -1;

    test(array);
    printf("%d ", array[0]);

    free(array);

    return 0;
}
```

EJEMPLO 3. Lagunas de memoria.

Las lagunas de memoria se producen cuando no se libera la zona de memoria que antes ha sido reservada:

```c
#include <stdlib.h>
#include <stdio.h>

#define NMAX 12

void test(int * a) {

    int i;
    for (i = 0; i< NMAX; i++)
        a[i] = 0;

}

int main(void) {

    int* array = malloc(NMAX * sizeof (int));
    if (!array) return -1;

    test(array);

    return 0; // ERROR: memoria sin liberar
}
```

La salida de Valgrind es:

```
alessia@rio-PORTEGE-Z930:~/NetBeansProjects/Tests$ gcc -g main.c
alessia@rio-PORTEGE-Z930:~/NetBeansProjects/Tests$ valgrind --leak-check=full ./a.out
==22291== Memcheck, a memory error detector
==22291== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
==22291== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==22291== Command: ./a.out
==22291==
==22291==
==22291== HEAP SUMMARY:
==22291==     in use at exit: 48 bytes in 1 blocks
==22291==   total heap usage: 1 allocs, 0 frees, 48 bytes allocated
==22291==
==22291== 48 bytes in 1 blocks are definitely lost in loss record 1 of 1
==22291==    at 0x4C2B6CD: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==22291==    by 0x400535: main (main.c:16)
==22291==
==22291== LEAK SUMMARY:
==22291==    definitely lost: 48 bytes in 1 blocks
==22291==    indirectly lost: 0 bytes in 0 blocks
==22291==      possibly lost: 0 bytes in 0 blocks
==22291==    still reachable: 0 bytes in 0 blocks
==22291==         suppressed: 0 bytes in 0 blocks
==22291==
==22291== For counts of detected and suppressed errors, rerun with: -v
==22291== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 2 from 2)
```

El error generado:

==22886== HEAP SUMMARY:
==22886== in use at exit: 48 bytes in 1 blocks
==22886== total heap usage: 1 allocs, 0 frees, 48 bytes allocated
==22886==
==22886== 48 bytes in 1 blocks are definitely lost in loss record 1 of 1
==22886== at 0x4C2B6CD: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==22886== by 0x400535: main (main.c:16)
==22886==
==22886== LEAK SUMMARY:
==22886== definitely lost: 48 bytes in 1 blocks
==22886== indirectly lost: 0 bytes in 0 blocks
==22886== possibly lost: 0 bytes in 0 blocks
==22886== still reachable: 0 bytes in 0 blocks
==22886== suppressed: 0 bytes in 0 blocks

nos indica que hemos hecho una reserva de memoria con **malloc** en el programa **main** en la línea **16**, pero no la hemos librado con lo cual tenemos 48 bytes (NMAX x sizeof (int)) de memoria perdida.

El código corregido:

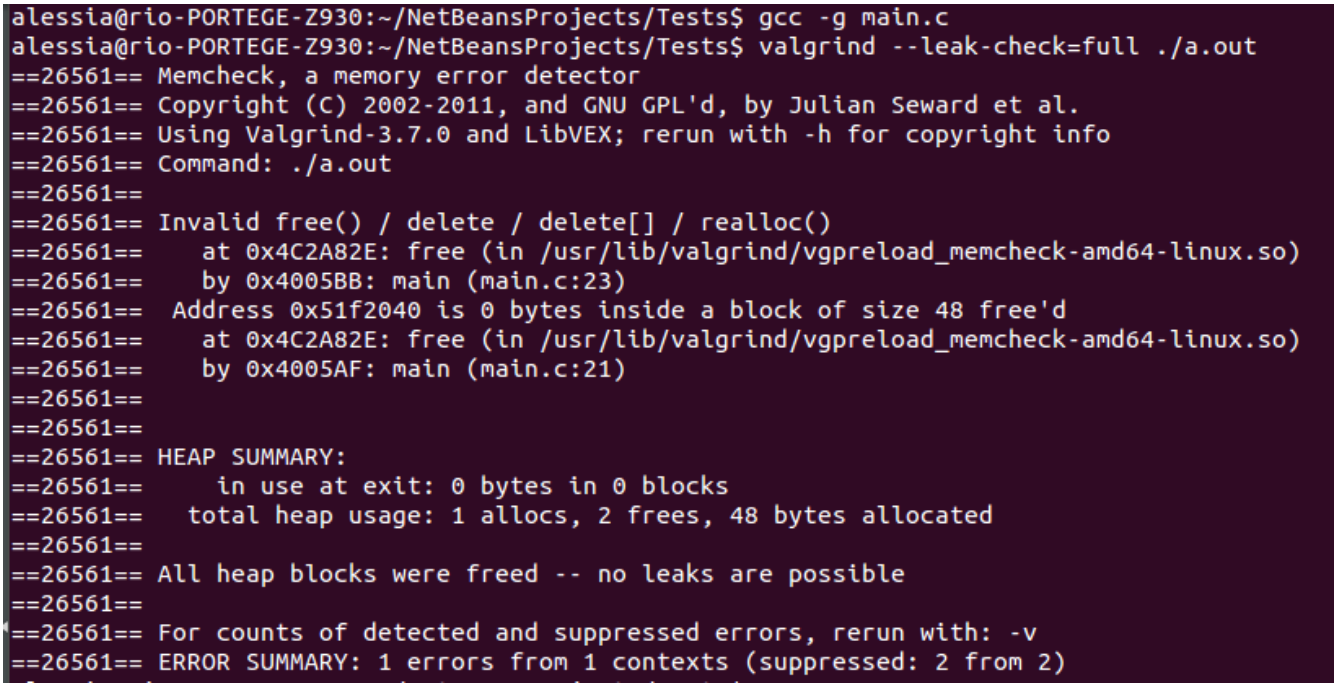
```c
#include <stdlib.h>
#include <stdio.h>

#define NMAX 12

void test(int * a) {...7 lines }

int main(void) {

    int* array = malloc(NMAX * sizeof (int));
    if (!array) return -1;

    test(array);

    free(array);

    return 0;
}
```

EJEMPLO 4. Liberar memoria de forma incorrecta.

Este error se puede producir al intentar liberar dos veces la misma zona de memoria:

```c
#include <stdlib.h>
#include <stdio.h>

#define NMAX 12

void test(int * a) {...7 lines }

int main(void) {

    int* array = malloc(NMAX * sizeof (int));
    if (!array) return -1;

    test(array);

    free(array);

    free(array); //ERROR

    return 0;
}
```

La salida de Valgrind:



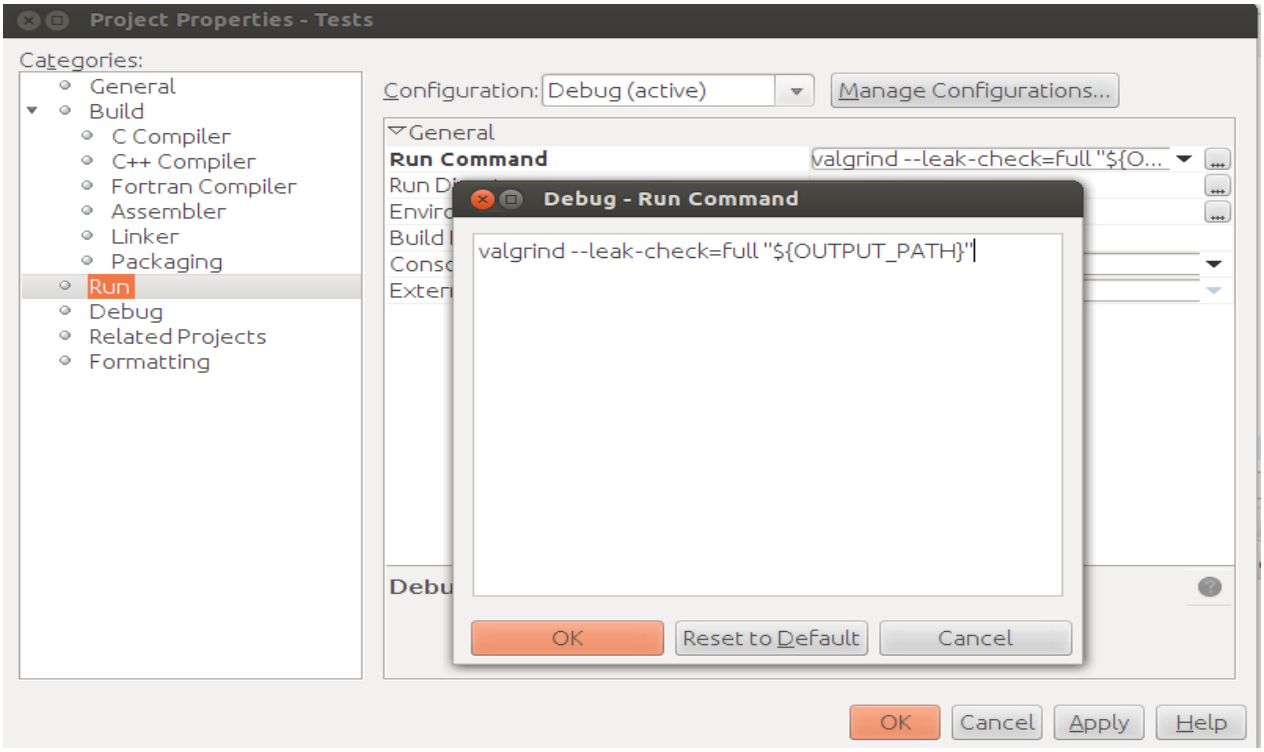
```
alessia@rio-PORTEGE-Z930:~/NetBeansProjects/Tests$ gcc -g main.c
alessia@rio-PORTEGE-Z930:~/NetBeansProjects/Tests$ valgrind --leak-check=full ./a.out
==26561== Memcheck, a memory error detector
==26561== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
==26561== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==26561== Command: ./a.out
==26561==
==26561== Invalid free() / delete / delete[] / realloc()
==26561==    at 0x4C2A82E: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==26561==    by 0x4005BB: main (main.c:23)
==26561==  Address 0x51f2040 is 0 bytes inside a block of size 48 free'd
==26561==    at 0x4C2A82E: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==26561==    by 0x4005AF: main (main.c:21)
==26561==
==26561==
==26561== HEAP SUMMARY:
==26561==     in use at exit: 0 bytes in 0 blocks
==26561==   total heap usage: 1 allocs, 2 frees, 48 bytes allocated
==26561==
==26561== All heap blocks were freed -- no leaks are possible
==26561==
==26561== For counts of detected and suppressed errors, rerun with: -v
==26561== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 2 from 2)
```

nos indica que el error se ha producido en el programa **main** en la línea **21** por intentar librar dos veces la misma zona de memoria:

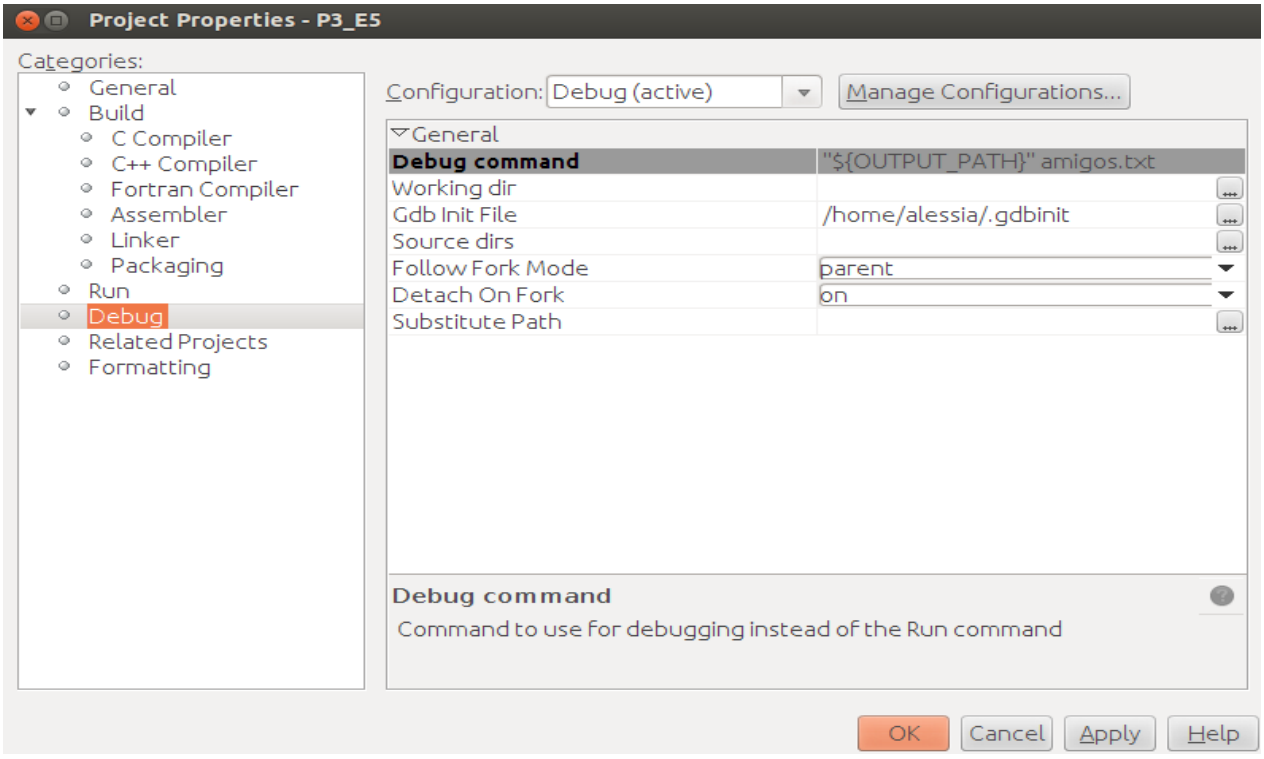
==29453== Invalid free() / delete / delete[] / realloc()
==29453== at 0x4C2A82E: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==29453== by 0x4005BB: main (main.c:23)
==29453== Address 0x51f2040 is 0 bytes inside a block of size 48 free'd
==29453== at 0x4C2A82E: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==29453== by 0x4005AF: main (main.c:21)
==29453==
==29453==
==29453== HEAP SUMMARY:
==29453== in use at exit: 0 bytes in 0 blocks
==29453== total heap usage: 1 allocs, 2 frees, 48 bytes allocated

Valgrind y Netbeans:

Project Properties → Run → Run Command



Debug → Debug command



Referencias

[1] Valgrind. http://valgrind.org/
[2] man valgrind