Memory Leaks And Debugging With Valgrind

Jula Menck

Working group scientific computing
Department of informatics
Faculty of mathematics, informatics and natural sciences
University of Hamburg

2014-06-19

UHH Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

informatik
die zukunft

Content

1 Memory Leaks

2 Valgrind

3 Memcheck

4 Summary

5 Appendix

6 Literature

Memory Leaks

- Memory Leaks are mismanaged memory allocations
  - Caused by heap areas that can no longer be freed up, due to a lost pointer
- Occur because C doesn't clean up after itself, unlike Java or C#
- Program might work for a while and then crash without apparent reason
- Are hard to find

Troublemakers I

Uninitialized memory

- Example: Allocating a pointer to a certain amount of bytes, possibly containing garbage data

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

int main(void)
{
char *p;

char c = *p;

printf("\n␣[%c]\n",c);

return 0;
}
```

Troublemakers II

Memory overwrite

- Writing more into a pointer than allocated bytes

Memory overread

- Reading more from a pointer than allocated bytes

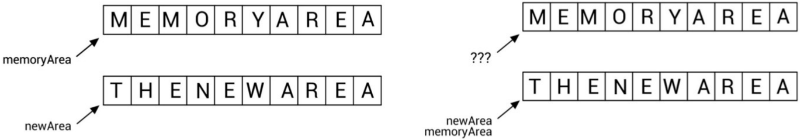
```
#include <stdlib.h>

int main()
{
char *ptr = (char *)malloc(10);
char name[12] ;
memcpy ( name,ptr,12);
}
```

Causes

Losing Pointer through Reassignment

- Reassigning the pointer "memoryArea" to point towards "newArea"



Improper handling of return values

- If a function returns a reference to a dynamically allocated memory it is the job of the calling function to keep track of the memory location. If it fails to do so you lose the address

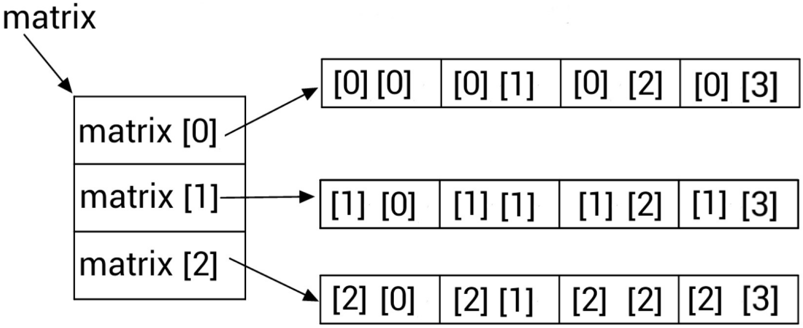
Causes II

Freeing the Parent Block of a Pointer

- Freeing the parent block of a pointer first causes you to lose the address of the pointer within

```
#include <stdlib.h>
int main(int argc, char* argv[])
{
    int z=3;
    int s=z+1;
    int **matrix ;
    int i,k;
    matrix=(int**)malloc(z*sizeof(void*));
    for (i=0;i<z;i++)
    {
     matrix[i]=(int*)malloc(s*sizeof(int));
 }
 free(matrix);
 return 0;
}
```

Matrix



matrix

matrix [0]
matrix [1]
matrix [2]

[0] [0] [0] [1] [0] [2] [0] [3]

[1] [0] [1] [1] [1] [2] [1] [3]

[2] [0] [2] [1] [2] [2] [2] [3]

Tips

- Remember to use free() after malloc()
- Use a copy of a pointer to avoid changing the original accidentally
- Don't orphan memory locations when (re)assigning pointers
- Free structured pointers from child to parent
- Don't access null pointers
- Handle returned references properly

What is Valgrind?

- Instrumentation framework for building dynamic analysis tools
- Open source software
- Licensed under GNU General Public License
- OS: Linux, Mac, Android, Not Windows
- Program Language: Any
- Modular Architecture
  - Several tools are included by default:
    - Memcheck, Cachegrind, Helgrind,...
- Simulates every single instruction (including libraries, suppressions)
  - Is done on a synthetic core
  - Need to start the program with Valgrind attached

Memcheck

- Memory error detector
- Standard tool of Valgrind
- Especially made for C, C++, Fortran
- Makes program run 10-50 times slower while in use
- Reports errors before executing the code
- Allows for suppressing errors (Suppressing system library code)

Checks performed

- Tracks addressability per byte and initialization per bit, enabling it to detect the use of single uninitialized bits
- Tracks heap blocks allocated with malloc(), thus can detect false or missing frees
- Checks all reads and writes of memory if they overlap
- Performs definedness check, allowing it to detect undefined value errors with bit-precision (via shadow bits)

Commands

- Invoke the tools via:
  valgrind [valgrind-options] your-program [your-program-options]
- As Memcheck is the default you can omit the −−tool=[name] option
- Careful with compiler optimization flags
  - "-o2" or "-o1", sometimes report (missing) uninitialized value errors
  - Use -o while using Memcheck (other Valgrind tools unaffected)
- Use "-g" upon compiling to let Valgrind use line numbers in its error messages

Options

commands	description
-v	Adds more detail to error message description (another -v adds yet more detail)
−−leak-check=full	Gives details on definitely lost and possibly lost blocks
−−num-callers	Makes stack traces longer
−−track-origins=yes	See the origins of uninitialized values
−−gen-suppressions=yes	Writes a suppression for each error message which you can copy into a suppression file
−−read-var-info	gives more detailed description of any illegal addresses (may run slower)

Commentary

- Error message written into commentary
- Commentary is send to a specific location. Options being:
  - Default: stderr
  - Specify file: −−log-file=filename
  - Send to network socket: −−log-socket="IP-Address"
    - Use Valgrind-Listener to listen in on that end

Error Messages

- Errors are reported before the associated operation happens
  Possible errors messages are:
- Illegal read / Illegal write errors
  - "Invalid read of size [number]"
  - Happens when Memcheck thinks it shouldn't be accessed at that point
  - Also tells you if that block might have been free()'d already
  - Informs you if it is off by one (heap block)

Error Messages II

- Use of uninitialized values
  - ”Conditional jump or move depends on uninitialized value(s)”
  - If program uses a value which hasn’t been initialized (is undefined)
  - Keeps track if you copy undefined values, but only comments on it, if it causes issues

Leak Error Messages I

Leak Error Messages are summarized in a leak summary with number of bytes and blocks. These messages being:

- Definitely lost
  - Program is leaking memory. Fix that!
- Indirectly lost
  - Program is leaking memory in a pointer-based structure. (e.g. you have a parent node which is "definitely lost", making all the children "indirectly lost")
  - Fixing the "definitely lost" will most likely fix the "indirectly lost"

Leak Error Messages II

- Possibly lost
  - Program is leaking memory, unless you did something inventive with pointers, thus causing them to point to the middle of allocated blocks
  - Use: '−−show-possibly-lost=no' if you don't want to see these
- Still reachable
  - Program didn't free some memory it could have, but is probably ok (quite common and often reasonable)
  - Use '−−show-reachable=yes' if you want to see these reports
- Suppressed
  - A leak error has been suppressed
  - Some are already in the default suppression files and can be ignored

Summary

- Memory leaks occur by forgetting to free() space or by losing pointer to memory area before freeing
- Valgrind is a tool for debugging your program
- Memcheck is the standard Valgrind tool, used for debugging memory errors/leaks

Valgrind Error Message Parentblock Matrix:

```
==4152== HEAP SUMMARY:
==4152== in use at exit: 48 bytes in 3 blocks
==4152== total heap usage: 4 allocs, 1 frees, 60 bytes allocated

==4152== LEAK SUMMARY:
==4152== definitely lost: 48 bytes in 3 blocks
==4152== indirectly lost: 0 bytes in 0 blocks
==4152== possibly lost: 0 bytes in 0 blocks
==4152== still reachable: 0 bytes in 0 blocks
==4152== suppressed: 0 bytes in 0 blocks

==4152== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Valgrind Error Message Overread:

```
==4148== Invalid read of size 4
==4148== at 0x80484A3: main (Overread.c:7)
==4148== Address 0x4204030 is 8 bytes inside a
    block of size 10 alloc'd
==4148==␣at␣0x40299D8:␣malloc(in/usr/lib/
    valgrind/vgpreload_memcheck-x86-linux.so)
==4148==␣by␣0x804848D:␣main␣(Overread.c:5)
==4148==␣HEAP␣SUMMARY:
==4148==␣in␣use␣at␣exit:␣10␣bytes␣in␣1␣blocks
==4148==␣total␣heap␣usage:␣1␣allocs,␣0␣frees,␣10
    ␣bytes␣allocated
==4148==␣LEAK␣SUMMARY:
==4148==␣definitely␣lost:␣10␣bytes␣in␣1␣blocks
==4148==␣indirectly␣lost:␣0␣bytes␣in␣0␣blocks
==4148==␣possibly␣lost:␣0␣bytes␣in␣0␣blocks
==4148==␣still␣reachable:␣0␣bytes␣in␣0␣blocks
==4148==␣suppressed:␣0␣bytes␣in␣0␣blocks
==4148==␣ERROR␣SUMMARY:␣2␣errors␣from␣2␣contexts
    ␣(suppressed:␣0␣from␣0)
```

Literature I

- Valgrind Manual
  http://valgrind.org/docs/manual/manual.html
- Pointers and memory leaks in C
  http://www.ibm.com/developerworks/aix/library/au-toughgame/
- Simple rules to avoid Memory Leaks in C
  http://mousomer.wordpress.com/2010/11/03/simple-rules-to-avoid-memory-leaks-in-c/
- Seward, Julian; Nethercote, Nicholas.
  Using Valgrind to detect undefined value errors with bit-precision.
  Proceedings of the USENIX'05 Annual Technical Conference.
  Anaheim, California, USA. (April, 2005).

Literature II

How to Detect Memory Leaks Using memcheck Tool for C or C++
http://www.thegeekstuff.com/2011/11/valgrind-memcheck/

Using Valgrind to Find Memory Leaks and Invalid Memory Use
http://www.cprogramming.com/debugging/valgrind.html

Speicherverwaltung und fortgeschrittene Pointer-Themen
http://www.fh-kl.de/ guenter.biehl/lehrgebiete/c/c08.html