Valgrind

Frédéric Magniette - LLR

What is Valgrind ?

- A suite of open-source debugging and profiling tools
- For C and C++ programs
- Compatible with plenty of architectures
- Used world-wide for unix development
- Check memory and cache usage, but also function calls and thread conflicts
- Nicely documented http://valgrind.org/docs/

X86/Linux
AMD64/Linux
ARM/Linux
ARM64/Linux
PPC32/Linux
PC64/Linux
PPC64LE/Linux
S390X/Linux
MIPS32/Linux
MIPS64/Linux
X86/Solaris
AMD64/Solaris
ARM/Android (2.3.x and later)
ARM64/Android,
X86/Android (4.0 and later)
MIPS32/Android,
X86/Darwin
AMD64/Darwin (Mac OS X 10.12)

Mini-HOWTO

- Step 1 : compile your code
  - with debugging symbols (-g)
  - at level 1 optimization (avoid -o2 or above)
- Step 2 : use valgrind at execution
  - `valgrind --leak-check=yes prog <args>`
  - Memcheck is the default tool
  - For other tools use –tool
  - `valgrind --tool=callgrind prog <args>`

Memcheck

- Hook any memory operations
- Add sanity check for size
- Keep track of all allocation operations
- Check use of allocated block (unitialized values)

Hello World

```
#include<stdio.h>

int main(void){
    printf("Hello, world !\n");
    return 0;
}
```

```
$ gcc -g -o hello hello.c
$ valgrind ./hello
```

```
==6207== Memcheck, a memory error detector
==6207== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==6207== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==6207== Command: ./hello
==6207==
Hello, world !
==6207==
==6207== HEAP SUMMARY:
==6207==     in use at exit: 0 bytes in 0 blocks
==6207==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==6207==
==6207== All heap blocks were freed -- no leaks are possible
==6207==
==6207== For counts of detected and suppressed errors, rerun with: -v
==6207== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Dereferencing NULL pointer

```
1 #include<stdlib.h>
2
3 int main(void) {
4     int *p = NULL;
5     *p = 0;
6     return 0;
7 }
```

```
==6709== Invalid write of size 4
==6709==    at 0x10860A: main (deref_null.c:5)
==6709==  Address 0x0 is not stack'd, malloc'd or (recently) free'd
==6709==
==6709==
==6709== Process terminating with default action of signal 11 (SIGSEGV)
==6709==  Access not within mapped region at address 0x0
==6709==    at 0x10860A: main (deref_null.c:5)
==6709==  If you believe this happened as a result of a stack
==6709==  overflow in your program's main thread (unlikely but
==6709==  possible), you can try to increase the size of the
==6709==  main thread stack using the --main-stacksize= flag.
==6709==  The main thread stack size used in this run was 8388608.
==6709==
==6709== HEAP SUMMARY:
==6709==     in use at exit: 0 bytes in 0 blocks
==6709==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==6709==
==6709== All heap blocks were freed -- no leaks are possible
==6709==
==6709== For counts of detected and suppressed errors, rerun with: -v
==6709== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
Erreur de segmentation (core dumped)
```

Dereferencing NULL pointer with reading

```
1 #include<stdlib.h>
2
3  int main(void) {
4       int i;
5       int *p = NULL;
6       i = *p;
7       return 0;
8  }
```

```
==7196== Invalid read of size 4
==7196==    at 0x10860A: main (deref_null2.c:6)
==7196==  Address 0x0 is not stack'd, malloc'd or (recently) free'd
==7196==
==7196==
==7196== Process terminating with default action of signal 11 (SIGSEGV)
==7196==  Access not within mapped region at address 0x0
==7196==    at 0x10860A: main (deref_null2.c:6)
==7196==  If you believe this happened as a result of a stack
==7196==  overflow in your program's main thread (unlikely but
==7196==  possible), you can try to increase the size of the
==7196==  main thread stack using the --main-stacksize= flag.
==7196==  The main thread stack size used in this run was 8388608.
```

Double Free

```
1 #include <stdlib.h>
2 int main(void) {
3     char *p = (char*)malloc(1);
4     *p = 'a';
5     free(p);
6     free(p);
7     return 0;
8 }
```

```
==13937== Invalid free() / delete / delete[] / realloc()
==13937==    at 0x4C30D3B: free (in /usr/lib/valgrind/vgpreload_memcheck-
amd64-linux.so)
==13937==    by 0x1086BE: main (double_free.c:6)
==13937==  Address 0x5857040 is 0 bytes inside a block of size 1 free'd
==13937==    at 0x4C30D3B: free (in /usr/lib/valgrind/vgpreload_memcheck-
amd64-linux.so)
==13937==    by 0x1086B2: main (double_free.c:5)
==13937==  Block was alloc'd at
==13937==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-
amd64-linux.so)
==13937==    by 0x10869B: main (double_free.c:3)
```

Overflow

```
1 #include <stdlib.h>
2 int main(void) {
3   int *p = malloc(3 * sizeof(int));
4   if (p != NULL) {
5     p[3] = 0;
6     free(p);
7   }
8   return 0;
9 }
```

```
==7603== Invalid write of size 4
==7603==    at 0x1086AF: main (depasse.c:5)
==7603==  Address 0x585704c is 0 bytes after a block of size 12 alloc'd
==7603==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck
-amd64-linux.so)
==7603==    by 0x10869B: main (depasse.c:3)
```

Underflow

```
1 #include <stdlib.h>
2 int main(void) {
3   int *p = malloc(3 * sizeof(int));
4   if (p != NULL) {
5     p[-1] = 0;
6     free(p);
7   }
8   return 0;
9 }
```

```
==7979== Invalid write of size 4
==7979==    at 0x1086AF: main (underflow.c:5)
==7979==  Address 0x585703c is 4 bytes before a block of size 12 alloc'd
==7979==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-
amd64-linux.so)
==7979==    by 0x10869B: main (underflow.c:3)
```

Freed pointer

```
1 #include <stdlib.h>
2 int main(void) {
3   int *p = malloc(sizeof(int));
4   if (p != NULL) {
5     free(p);
6     *p = 1;
7   }
8   return 0;
9 }
```

```
==8225== Invalid write of size 4
==8225==    at 0x1086B7: main (freed.c:6)
==8225==  Address 0x5857040 is 0 bytes inside a block of size 4 free'd
==8225==    at 0x4C30D3B: free (in /usr/lib/valgrind/vgpreload_memcheck-
amd64-linux.so)
==8225==    by 0x1086B2: main (freed.c:5)
==8225==  Block was alloc'd at
==8225==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-
amd64-linux.so)
==8225==    by 0x10869B: main (freed.c:3)
```

Memory Leak

```
1 #include<stdlib.h>
2 int main(void) {
3   int *p = malloc(sizeof(int));
4   return 0;
5 }
```

```
==12875== HEAP SUMMARY:
==12875==     in use at exit: 4 bytes in 1 blocks
==12875==   total heap usage: 1 allocs, 0 frees, 4 bytes allocated
==12875==
==12875== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==12875==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-
amd64-linux.so)
==12875==    by 0x10865B: main (leak1.c:3)
==12875==
==12875== LEAK SUMMARY:
==12875==    definitely lost: 4 bytes in 1 blocks
==12875==    indirectly lost: 0 bytes in 0 blocks
==12875==      possibly lost: 0 bytes in 0 blocks
==12875==    still reachable: 0 bytes in 0 blocks
==12875==         suppressed: 0 bytes in 0 blocks
```

Definitely lost : no more pointer to this chunk (after quitting main)

Other kind of leak

- Still reachable : there exists a pointer to the block → just free it
- Definitely lost : no more pointer to the block : free it before the pointer disappear (pointer in the stack)
- Indirectly lost : pointer to the pointer of the block are lost. Example : the root of a binary tree is lost. The leafs are indirectly lost even if a pointer to it still exists.
- Possibly lost : the block is pointed by an interior pointer (inside the block but not at its beginning)

Jump on uninitialized variable

```
1 #include<stdlib.h>
2 #include<stdio.h>
3 int main(void) {
4   int j;
5   if (j==0) {
6     printf("Hello, world !\n");
7   }
8   return 0;
9 }
```

```
==13119== Conditional jump or move depends on uninitialised value(s)
==13119==    at 0x108646: main (undef_jump.c:5)
==13119==
Hello, world !
```

By chance, j=0 but this can be different randomly

C++ malloc/delete mismatch

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include<iostream>
4   int main(void) {
5       char *p = (char*)malloc(1);
6       *p = 'a';
7       char c = *p;
8       printf("\n [%c]\n",c);
9       delete p;
10      return 0;
11  }
```

```
$ g++ -g -o malloc_delete malloc_delete.cc
$ valgrind ./malloc_delete
```

```
 [a]
==13524== Mismatched free() / delete / delete []
==13524==    at 0x4C3123B: operator delete(void*) (in /usr/lib/valgrind/
vgpreload_memcheck-amd64-linux.so)
==13524==    by 0x108888: main (malloc_delete.cc:9)
==13524==  Address 0x61a7c80 is 0 bytes inside a block of size 1 alloc'd
==13524==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/
vgpreload_memcheck-amd64-linux.so)
==13524==    by 0x10884B: main (malloc_delete.cc:5)
```

C++ overflow

```
1 #include <iostream>
2 int main() {
3   int *p1 = new int[10];
4   p1[10]=1;
5   std::cout<<p1[10]<<std::endl;
6   delete [] p1;
7 }
```

```
==14147== Invalid write of size 4
==14147==    at 0x108938: main (overflow.cc:4)
==14147==  Address 0x61a7ca8 is 0 bytes after a block of size 40 alloc'd
==14147==    at 0x4C3089F: operator new[](unsigned long)
(in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==14147==    by 0x10892B: main (overflow.cc:3)
==14147==
==14147== Invalid read of size 4
==14147==    at 0x108946: main (overflow.cc:5)
==14147==  Address 0x61a7ca8 is 0 bytes after a block of size 40 alloc'd
==14147==    at 0x4C3089F: operator new[](unsigned long)
(in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==14147==    by 0x10892B: main (overflow.cc:3)
==14147==
1
```

No segmentation fault but memory corruption → harder to detect

Other modules

- Cachegrind : cache profiler. Pinpoint source of cache-miss
- Callgrind : provides call graph
- Helgrind : thread debugger, find data races
- Massif : Heap profiler
- ...

```
int main() {
  int **tab=malloc(linesize*sizeof(int *));
  int i,j;
  for (i=0;i<linesize;i++) {
    tab[i]=malloc(linesize*sizeof(int));
  }
  for (i=0;i<linesize;i++)
    for (j=0;j<linesize;j++)
      tab[i][j]=0;
  for (i=0;i<linesize;i++) {
    free(tab[i]);
  }
  free(tab);
}
```

Cachegrind example

```
$ g++ -g -o testcache testcache.c
$ valgrind --tool=cachegrind ./testcache
$ cg_annotate cachegrind.out.26804 testcache.c
```

```
==28258== I   refs:      1,404,857,032
==28258== I1  misses:            1,112
==28258== LLi misses:            1,094
==28258== I1  miss rate:          0.00%
==28258== LLi miss rate:          0.00%
==28258==
==28258== D   refs:        701,853,513  (601,184,908 rd   + 100,668,605 wr)
==28258== D1  misses:        6,282,709  (     16,810 rd   +   6,265,899 wr)
==28258== LLd misses:        6,281,800  (     16,075 rd   +   6,265,725 wr)
==28258== D1  miss rate:           0.9% (        0.0%     +         6.2%  )
==28258== LLd miss rate:           0.9% (        0.0%     +         6.2%  )
==28258==
==28258== LL refs:           6,283,821  (     17,922 rd   +   6,265,899 wr)
==28258== LL misses:         6,282,894  (     17,169 rd   +   6,265,725 wr)
==28258== LL miss rate:            0.3% (        0.0%     +         6.2%  )
```

Cachegrind example (2)

```
$ cg_annotate cachegrind.out.26804 testcache.c
```

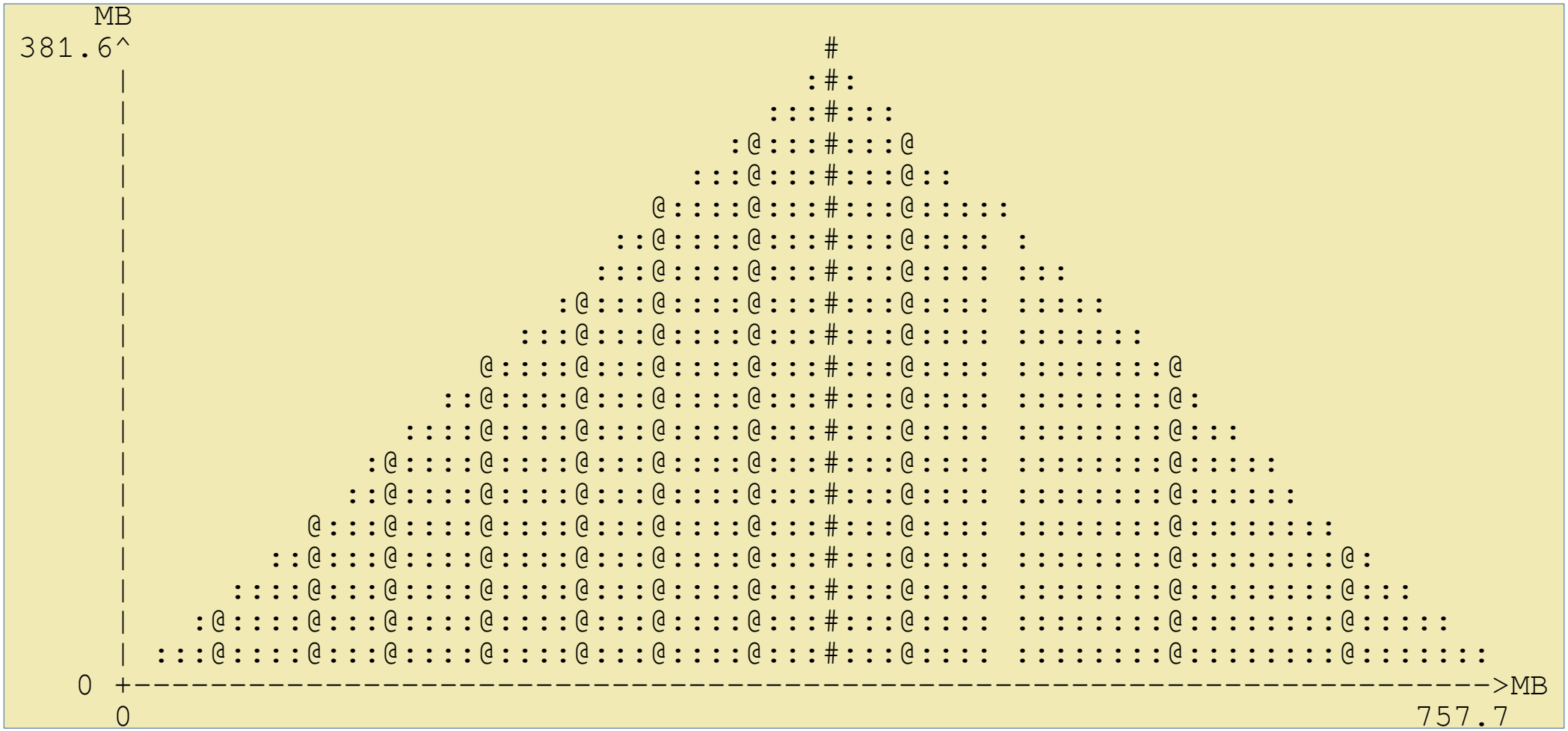
```
--------------------------------------------------------------------------------
           Ir  I1mr  ILmr           Dr   D1mr   DLmr           Dw      D1mw      DLmw
--------------------------------------------------------------------------------
1,404,857,032 1,112 1,094 601,184,908 16,810 16,075 100,668,605 6,265,899 6,265,725

--------------------------------------------------------------------------------
           Ir I1mr ILmr           Dr   D1mr   DLmr           Dw      D1mw      DLmw
--------------------------------------------------------------------------------
1,400,290,027    3    3 600,120,007  2,503  2,478 100,040,008 6,253,938 6,253,836
                        /root/valgrind/testcache.c:main
    1,604,810   37   37     225,359      2      1     282,644     9,573     9,573
                        /build/glibc-OTsEL5/glibc-2.27/malloc/malloc.c:_int_malloc
    1,492,208   20   20     427,399 11,152 11,147     189,260         3         3
                        /build/glibc-OTsEL5/glibc-2.27/malloc/malloc.c:free
```

Simulate the cache miss on probed cache hierarchy

Massif example

```
$ g++ -g -o testcache testcache.c
$ valgrind --tool=massif --time-unit=B ./testcache
$ ms_print  massif.out.27888
```



Monitor the heap usage along the execution time (in instructions or in bytes)

Massif example(2)

Detailed measure points :

```
Number of snapshots: 78
 Detailed snapshots: [5, 10, 15, 20, 25, 30, 35, 39 (peak), 44, 58, 68]

--------------------------------------------------------------------------------
  n        time(B)         total(B)   useful-heap(B) extra-heap(B)    stacks(B)
--------------------------------------------------------------------------------
  0              0                0              0              0            0
  1     10,242,040       10,242,040     10,240,000          2,040            0
  2     20,484,088       20,484,088     20,480,000          4,088            0
  3     30,726,136       30,726,136     30,720,000          6,136            0
  4     40,968,184       40,968,184     40,960,000          8,184            0
  5     51,210,232       51,210,232     51,200,000         10,232            0

...

 71    733,026,568       67,293,448     67,280,000         13,448            0
 72    743,268,616       57,051,400     57,040,000         11,400            0
 73    753,510,664       46,809,352     46,800,000          9,352            0
 74    763,752,712       36,567,304     36,560,000          7,304            0
 75    773,994,760       26,325,256     26,320,000          5,256            0
 76    784,236,808       16,083,208     16,080,000          3,208            0
 77    794,478,856        5,841,160      5,840,000          1,160            0
```

Helgrind example

```
#include <pthread.h>
#include <stdio.h>
int var = 0;

void* child_fn ( void* arg ) {
  int i;
  for(i=0;i<1000000;i++)
    var++;
   return NULL;
}

int main ( void ) {
  int i;
  pthread_t child;
  pthread_create(&child, NULL, child_fn, NULL);
  for(i=0;i<1000000;i++)
    var++;
  pthread_join(child, NULL);
  printf("var=%d, should be 2000000\n",var);
  return 0;
}
```

- Access conflict on a shared variable
- Should be protected by a mutex

```
$ gcc -g -o threads threads.c -lpthread
$ ./threads
var=1191063, should be 2000000
```

Helgrind example (2)

```
08   for(i=0;i<1000000;i++)
09     var++;
10    return NULL;

17   for(i=0;i<1000000;i++)
18     var++;
19   pthread_join(child, NULL);
```

```
$ valgrind --tool=helgrind ./threads
```

```
==29253== Possible data race during write of size 4 at 0x309014 by thread #1
==29253== Locks held: none
==29253==    at 0x1087F4: main (threads.c:18)
==29253==
==29253== This conflicts with a previous write of size 4 by thread #2
==29253== Locks held: none
==29253==    at 0x108794: child_fn (threads.c:9)
==29253==    by 0x4C36C26: ??? (in /usr/lib/valgrind/vgpreload_helgrind-
==29253==    by 0x50506DA: start_thread (pthread_create.c:463)
==29253==    by 0x538988E: clone (clone.S:95)
==29253==  Address 0x309014 is 0 bytes inside data symbol "var"
```

Cons

- Very slow → unable to debug real-time applications
- Very very talkative
- Not working for Windows → give a try to the main concurrent : DrMemory

  https://drmemory.org/

Pros

- Very efficient : monitor all kind of bad memory usage
- Well documented
- Not working for Windows → move to Linux