Valgrind

Concurrent Programming

Introduction

- What is Valgrind?
- Installing Valgrind
- How to use?
- Practice

What is Valgrind?

- A framework for heavyweight dynamic binary instrumentation(DBI)
- Detecting memory management and thread bugs, and profile program
  - a memory error detector (Memcheck)
  - two thread error detectors (Helgrind)
  - a cache and branch-prediction profiler (Cachegrind)
  - a call-graph generating cache and branch-prediction profiler (Callgrind)
  - a heap profiler (Massif)
  - …
- Supporting various system platform
- Open Source / Freeware Software GNU GPL V2

What is Valgrind? - Memcheck

- What can we detect with Memcheck?
  - Accessing memory you shouldn’t
  - Using undefined values
  - Incorrect freeing of heap memory
  - Overlapping src and dst pointers in memcpy and related functions
  - Passing a fishy value to the size parameter of a memory allocation function
  - Memory leaks

What is Valgrind? - Helgrind

- What can we detect with Helgrind?
  - Misuses of the POSIX pthreads API
  - Potential deadlocks arising from lock ordering problems
  - Data races
    - Accessing memory without adequate locking or synchronization

Installing Valgrind

```
$ sudo apt-get install valgrind
```

How to use?

- valgrind --tool=name [options] prog-and-args
  - tool = [memcheck, cachegrind, callgrind, massif, helgrind, drd, nulgrind, etc..]

- Memcheck

```
$ valgrind --tool=memcheck --leak-check=yes ./a.out
```

- Helgrind

```
$ valgrind --tool=helgrind ./a.out
```

* Compile with -g option, if you want to see line numbers that occur problem

Test code for Memcheck

< memcheck_test.cpp >

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

int main(void) {
    int p, t;
    char *pc;
    if (p == 5) {
        t = p + 1;
    }

    printf("%d is not initialized\n", p);
    pc = (char *)malloc(sizeof(char) * 10);

    return 0;
}
```

Memcheck result

```
$ g++ -g -o memcheck_test memcheck_test.cpp
```

```
$ valgrind --tool=memcheck --leak-check=full ./memcheck_test
```

```
==6079== Use of uninitialised value of size 8
==6079==    at 0x578876B: _itoa_word (_itoa.c:179)
==6079==    by 0x578C12C: vfprintf (vfprintf.c:1631)
==6079==    by 0x5793898: printf (printf.c:33)
==6079==    by 0x400581: main (memcheck_test.cpp:11)
==6079==
```

```
==6079== HEAP SUMMARY:
==6079==     in use at exit: 72,714 bytes in 2 blocks
==6079==   total heap usage: 3 allocs, 1 frees, 73,738 bytes allocated
==6079==
==6079== 10 bytes in 1 blocks are definitely lost in loss record 1 of 2
==6079==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==6079==    by 0x40058B: main (memcheck_test.cpp:12)
==6079==
==6079== LEAK SUMMARY:
==6079==    definitely lost: 10 bytes in 1 blocks
==6079==    indirectly lost: 0 bytes in 0 blocks
==6079==      possibly lost: 0 bytes in 0 blocks
==6079==    still reachable: 72,704 bytes in 1 blocks
==6079==         suppressed: 0 bytes in 0 blocks
==6079== Reachable blocks (those to which a pointer was found) are not shown.
==6079== To see them, rerun with: --leak-check=full --show-leak-kinds=all
```

Memcheck - memory leak type

- “Definitely lost”
  - Your program is leaking meory – fix those leaks!
- ”Indirectly lost”
  - Your program is leaking memory in a pointer-based structure.
- ”Possibly lost”
  - your program is leaking memory, unless you're doing unusual things with pointers that could cause them to point into the middle of an allocated block
- ”Still reachable”
  - your program is probably ok
- “Suppressed”
  - Those are leaks outside of your code

Test code for Helgrind

< helgrind_test.cpp >

```
 1 #include <pthread.h>
 2
 3 int var = 0;
 4
 5 void *thread_func(void *arg) {
 6     var++;
 7     return NULL;
 8 }
 9
10 int main(void) {
11     pthread_t child;
12     pthread_create(&child, NULL, thread_func, NULL);
13     var++;
14     pthread_join(child, NULL);
15     return 0;
16 }
```

Test code for Helgrind

```
$ g++ -g -o helgrind_test helgrind_test.cpp -lpthread
```

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

```
==6134== Possible data race during write of size 4 at 0x601044 by thread #1
==6134== Locks held: none
==6134==    at 0x4005E1: main (helgrind_test.cpp:13)
==6134==
==6134== This conflicts with a previous write of size 4 by thread #2
==6134== Locks held: none
==6134==    at 0x4005A8: thread_func(void*) (helgrind_test.cpp:6)
==6134==    by 0x4C34DB6: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==6134==    by 0x4E476B9: start_thread (pthread_create.c:333)
==6134==    by 0x5A6841C: clone (clone.S:109)
==6134==  Address 0x601044 is 0 bytes inside data symbol "var"
```

Practice

- Practice on the prime_mt_valgrind.cpp uploaded at Piazza page

- With valgrind, find some problems in it
  - X memory problems - use memcheck
  - X race condition problems - use helgrind
  - (The number of problems could be different for each environment)

Thank You