Valgrind: Memory Checker Tool

- Use Valgrind to check your program for memory errors and memory leaks. Valgrind can do many more things but we will focus on the memory checking part.
- Use Valgrind for testing your singly linked list as follows.

```
valgrind --leak-check=yes SimpleTest <n>
```

- Run it on the `SimpleTest.c` program without the `freeList` function and then run it again after adding the function.
- Valgrind is installed in the lab on all systems. Install it on your Fedora Linux system with the command:
  `sudo dnf install valgrind`

Valgrind: Overview

- Extremely useful tool for any C/C++ programmer
- Mostly known for Memcheck module, which helps find many common memory related problems in C/C++ code
- Supports X86/Linux, AMD64/Linux, PPC32/Linux, PPC64/Linux and X86/Darwin (Mac OS X)
- Available for X86/Linux since ~2003, actively developed

Sample (Bad) Code

- We will use the following code for demonstration purposes:: The code can be found in the examples at C-examples/debug/valgrind.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define SIZE 100
4  int main() {
5      int i, sum = 0;
6      int *a = malloc(SIZE);
7      for (i = 0; i < SIZE; ++i)
8          sum += a[i];
9      a[26] = 1;
10     a = NULL;
11     if (sum > 0) printf("Hi!\n");
12     return 0;
13 }
```

- Contains many bugs. Compiles without warnings or errors. Run it under Valgrind as shown below:

```
valgrind --leak-check=yes sample
```

Invalid Read

Example

```
==17175== Invalid read of size 4
==17175== at 0x4005C0: main (sample.c:9)
==17175== Address 0x51f60a4 is 0 bytes after a block of size 100 alloc'd
==17175== at 0x4C28C50: malloc (in ...)
==17175== by 0x40059E: main (sample.c:7)
```

- We read past the end of the allocated array
- Trying to read from area which we are not allowed to access
- Could result in a SEGFAULT and surely doesn't do what we want
- Valgrind provides enough details to find the problem

Invalid Write

Example

```
==17175== Invalid write of size 4
==17175== at 0x4005D7: main (sample.c:10)
==17175== Address 0x51f60a8 is 4 bytes after a block of size 100 alloc'd
==17175== at 0x4C28C50: malloc (in ...)
==17175== by 0x40059E: main (sample.c:7)
==17175==
```

- Similar to invalid read
- Details provided by Valgrind
  - Location of fault (addresses, line number if debug-flag used with the compiler)
  - Stack-trace to fault (you can get more using $--num-callers=30$)
  - Relevant blocks details and allocation/de-allocation stack-trace

Memory Leaks

- At the end of the run, Valgrind does "Garbage Collection"
- Unreferenced memory in C/C++ $\Rightarrow$ memory leak

Example

==17175== 100 bytes in 1 blocks are definitely lost in loss record 1 of 1
==17175== at 0x4C28C50: malloc (in ...)
==17175== by 0x40059E: main (sample.c:7)

- Valgrind provides stack-trace for the allocation point
- 3 kinds:
  - Definitely lost (no pointers to allocation)
  - Probably lost (pointers only to the middle of the allocation)
  - Still reachable (block hasn't been free'd before exit, but pointers to it still exists)

Suppression Files

- Valgrind tends to be very noisy
- Most of the times it is indicating bugs that should be fixed
  - But not always the one we want to fix right now
- Sometimes it is correct code, which Valgrind failed to understand
  - Mostly in sophisticated/extremely optimized library code
  - Also possible when having unusual interactions with the kernel
- Valgrind includes a mechanism to silence a specific error
  - Works with all tools that report errors
  - Simple file format, see documentation for details
  - Valgrind includes suppression for many common libs

References

- http://valgrind.org
- http://haifux.org/lectures/239/