Valgrind

Computing Laboratory

http://www.isical.ac.in/~dfslab/2023

Overview

Reference: `https://valgrind.org/docs/manual/index.html`

- Set of tools for automatically detecting memory related bugs / errors

  (also other bugs, profiling)
- For this course, focus on `memcheck`
  - reads / writes beyond array bounds
  - memory leaks

Getting started

- Use `-g` or `-g3` to add debugging information when compiling (as for gdb)

  ```
  $ gcc -g<option_number> -o progx progx.c
  ```

  - `<option_number>` determines amount of debugging information collected
    (1 = min, 2 = default, 3 = max)
- Run memcheck tool of `valgrind`

  ```
  $ valgrind -tool=memcheck -leak-check=full \
     -log-file=<logfile-name> ./progx
  ```

  OR

  ```
  $ valgrind -leak-check=full ./progx
  ```
- Examine logfile / output

Example – pg. I

Input file

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

int main() {
  char *source = "computing"; // Allocates 10 bytes
  char *copy = (char *) malloc(strlen(source));
  strcpy(copy, source); // source is larger than copy
  printf("%s\n", copy);
  // free(copy);
  return 0;
}
```

Example – pg. II

Output

```
==122001== Memcheck, a memory error detector
==122001== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et
    al.
==122001== Using Valgrind-3.21.0 and LibVEX; rerun with -h for copyright
     info
==122001== Command: a.out
==122001== Parent PID: 1458
==122001==
==122001== Invalid write of size 1
==122001==    at 0x484803E: strcpy (vg_replace_strmem.c:560)
==122001==    by 0x1091A6: main (valgrind-basic.c:8)
==122001==  Address 0x4a5d049 is 0 bytes after a block of size 9 alloc'd
==122001==    at 0x4841848: malloc (vg_replace_malloc.c:431)
==122001==    by 0x10918F: main (valgrind-basic.c:7)
==122001==
==122001== Invalid read of size 1
==122001==    at 0x4847ED4: strlen (vg_replace_strmem.c:501)
```

Example – pg. III

```
==122001==    by 0x48F1C07: puts (ioputs.c:35)
==122001==    by 0x1091B2: main (valgrind-basic.c:9)
==122001==  Address 0x4a5d049 is 0 bytes after a block of size 9 alloc'd
==122001==    at 0x4841848: malloc (vg_replace_malloc.c:431)
==122001==    by 0x10918F: main (valgrind-basic.c:7)
==122001==
==122001==
==122001== HEAP SUMMARY:
==122001==     in use at exit: 9 bytes in 1 blocks
==122001==   total heap usage: 2 allocs, 1 frees, 1,033 bytes allocated
==122001==
==122001== 9 bytes in 1 blocks are definitely lost in loss record 1 of 1
==122001==    at 0x4841848: malloc (vg_replace_malloc.c:431)
==122001==    by 0x10918F: main (valgrind-basic.c:7)
==122001==
==122001== LEAK SUMMARY:
==122001==    definitely lost: 9 bytes in 1 blocks
==122001==    indirectly lost: 0 bytes in 0 blocks
```

Example – pg. IV

```
==122001==      possibly lost: 0 bytes in 0 blocks
==122001==    still reachable: 0 bytes in 0 blocks
==122001==         suppressed: 0 bytes in 0 blocks
==122001==
==122001== For lists of detected and suppressed errors, rerun with: -s
==122001== ERROR SUMMARY: 3 errors from 3 contexts (suppressed: 0 from
    0)
```

Types of leaks

- **definitely lost** – memory is lost for sure (should be fixed!!!)
- **indirectly lost** – memory for the root node of a tree is free but not the rest of it (should be fixed!!!)
- **possibly lost** – memory is actually lost due to reasons like function allocates a buffer and returns it but the caller never frees the memory or a lot of allocated memory is actually not in need (should be fixed!!!)
- **still reachable** – memory is probably still in use at the end of the program (not a serious issue!!!)

Exercises

For each of `Test1.c`, ..., `Test6.c` (see 06 Oct, 2023)

1. compile using `-g` ;
2. run under `valgrind` / `memcheck` tool;
3. make sure you can understand the output;
4. fix the memory-related errors.