Debugging with GDB + Valgrind

Jack Ogaja
jack.ogaja@gwdg.de

18. April 2023

Session's Objectives

Learn how to monitor and study the state of your program in a steb-by-step basis using open-source debugging and profiling tools.

GDB: The GNU Source-Level Debugger

Some Useful Commands

- `gdb` :launch gdb
- `file` :load the executable
- `help cmd` :show help for the command `cmd`
- `run` :start program execution
- `break xl` :creates a break-point near the label `xl`
- `break *address` :creates a break-point at a specified address
- `stepi` or `si` :step by one instruction
- `gdb -tui` :launch gdb with Text User Interface
- `quit` :quit gdb

Example GDB Session

- Compile your code with the `-g` option, e.g.

```
$ gcc -Wall -g -c yourProgram.c
$ gcc -Wall -g -o yourBinary yourProgram.o
$ gdb
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
(gdb) file yourBinary
```

Valgrind

Valgrind's Memcheck

- Valgrind is an instrumentation framework with simulation-based debugging and profiling tools for building dynamic analysis.
- Memcheck is the default tool in Valgrind. It detects memory-management problems e.g. it can detect if your program:
  - Leaks memory
  - Missuses uninitialized values
  - Accesses memory incorrectly
  - Does illegal frees of heap blocks
  - Does illegal reads and writes (e.g. reads and writes overlapping memory blocks)

Valgrind's Memcheck - Example Code
Use of uninitialised values in system calls (Valgrind's documentation)

```
1   $ cat > uninit_syscall.c <<'EOF'
2   #include <stdlib.h>
3   #include <unistd.h>
4   int main( void )
5   {
6     char* arr  = malloc(10);
7     int*  arr2 = malloc(sizeof(int));
8     write( 1 /* stdout */, arr, 10 );
9     exit(arr2[0]);
10  }
11  EOF
```

Valgrind's Memcheck
Memcheck's result (Valgrind's documentation)

```
$ gcc -o uninit_syscall uninit_syscall.c -Wall -g
$ module load valgrind
$ valgrind ./uninstall_syscall
...
Syscall param write(buf) points to uninitialised byte(s)
    at 0x25A48723: __write_nocancel (in /lib/tls/libc-2.3.3.so)
    by 0x259AFAD3: __libc_start_main (in /lib/tls/libc-2.3.3.so)
Address 0x25AB8028 is 0 bytes inside a block of size 10 alloc'd
    at 0x259852B0: malloc (vg_replace_malloc.c:130)
    by 0x80483F1: main (a.c:5)
Syscall param exit(error_code) contains uninitialised byte(s)
    ...
```

Tutorial

- Download the tutorial materials
- Compile the programs
- Use GDB and Valgrind to study the execution states and memory access patterns.

Note: Use slurm to start an interactive session in a compute node.