Operation Systems Course: Ex4

Memory leak detection tools: Valgrind/Helgrind
Graphs Data Structure: Finding Eulerian Graph
POSIX Mutex: Creating a Singleton abstract base class that uses POSIX mutex

---

1-3: Run `$ make program` and run `./program [vertices] [edges] [seed]`

4. Run make all and it will output the following reports:
    Valgrind report > **valgrind_report.txt**
    Gprof > **gprof_report.txt**
    Gcov > **code_coverage_report.txt** (also in out/ will contain lcov html visualization)

5. In the folder /hello/**valgrind_report_for_hello_code.txt**

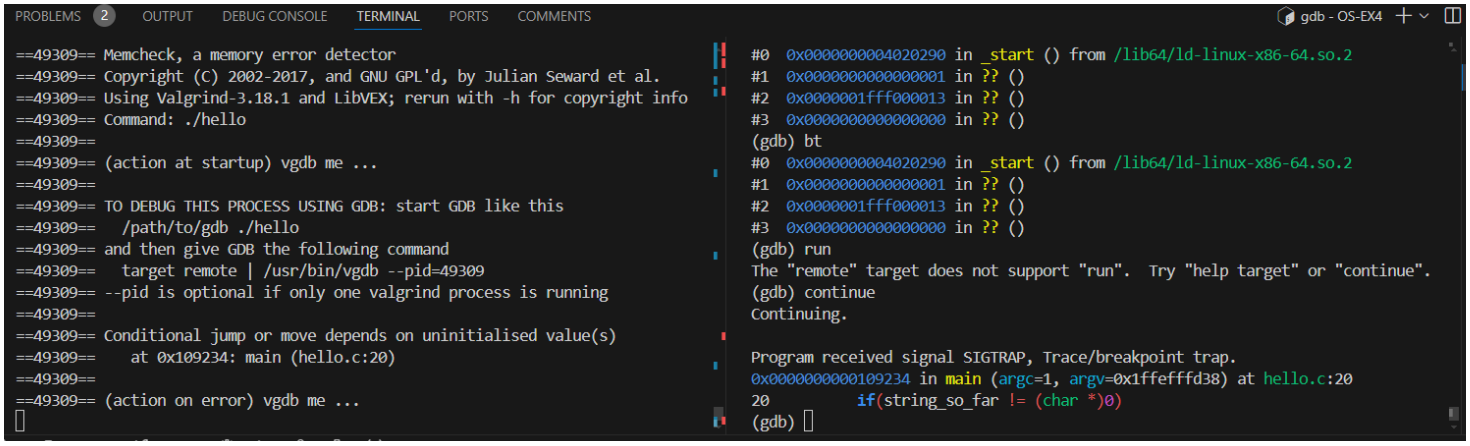
6. Demonstration of Valgrind attached to debugger (gdb):

Left Terminal:

```
$ gcc -g -o hello hello.c
$ valgrind --vgdb=yes --vgdb-error=0 ./hello
```

Right Terminal:

```
$ gdb ./hello
$ (gdb) target remote | vgdb
$ (gdb) continue
```



```
==49309== Memcheck, a memory error detector
==49309== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==49309== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==49309== Command: ./hello
==49309==
==49309== (action at startup) vgdb me ...
==49309==
==49309== TO DEBUG THIS PROCESS USING GDB: start GDB like this
==49309==   /path/to/gdb ./hello
==49309== and then give GDB the following command
==49309==   target remote | /usr/bin/vgdb --pid=49309
==49309== --pid is optional if only one valgrind process is running
==49309==
==49309== Conditional jump or move depends on uninitialised value(s)
==49309==    at 0x109234: main (hello.c:20)
==49309==
==49309== (action on error) vgdb me ...
```

```
#0  0x0000000004020290 in _start () from /lib64/ld-linux-x86-64.so.2
#1  0x0000000000000001 in ?? ()
#2  0x0000001fff000013 in ?? ()
#3  0x0000000000000000 in ?? ()
(gdb) bt
#0  0x0000000004020290 in _start () from /lib64/ld-linux-x86-64.so.2
#1  0x0000000000000001 in ?? ()
#2  0x0000001fff000013 in ?? ()
#3  0x0000000000000000 in ?? ()
(gdb) run
The "remote" target does not support "run".  Try "help target" or "continue".
(gdb) continue
Continuing.

Program received signal SIGTRAP, Trace/breakpoint trap.
0x0000000000109234 in main (argc=1, argv=0x1ffefffd38) at hello.c:20
20          if(string_so_far != (char *)0)
(gdb)
```

7. Race condition detected using Valgrind/Helgrind:

```
$ gcc -g -o race race.c -pthread
$ valgrind --tool=helgrind ./race > race_condition_report.txt 2>&1
```

*Full output is in the folder **/race/race_condition_report.txt.**

8. My Mutex, inside folder myMutex/:
    myMutex: Abstract base class defining the interface for mutex operations.
    myMutexLock: Concrete derived class implementing the lock function specific to myMutexLock.
    myMutex.cpp: manages Singleton instantiation and pthread mutex usage.
    myMutexLock.cpp: provides the actual implementation of the lock function for myMutexLock.

    To demonstrate the usage of the lock mechanism there are 2 main programs: `mainwithoutlock.cpp` and `mainwithlock.cpp`.

    Run `$ make all` to generate **results_comparison.txt** that shows the difference between the programs.