Operating Systems Assignment #4 - Submission

Question 4

Code coverage (GCOV):

All the GCOV files and stdouts are located at `/gcov_outputs`

By using `make coverage`, we recompile the program with coverage flags, run a few tests, each time directing the output of the gcov to stdoutN.txt inside the mentioned folder above. At the end, we move all the gcov related files (.gcda, .gcno, .gcov) to their folder to keep our root folder clean

```
samurai@DESKTOP-O04V85G:~/cs/Operating-Systems/Operating-Systems-Ex4/q1-4$ make coverage
make clean
make[1]: Entering directory '/home/samurai/cs/Operating-Systems/Operating-Systems-Ex4/q1-4'
rm -f main.o euler_circuit.o euler_circuit
make[1]: Leaving directory '/home/samurai/cs/Operating-Systems/Operating-Systems-Ex4/q1-4'
make all COVERAGE=1 DEBUG=0
make[1]: Entering directory '/home/samurai/cs/Operating-Systems/Operating-Systems-Ex4/q1-4'
g++ -std=c++17 -Wall -Wextra -fprofile-arcs -ftest-coverage -c main.cpp
g++ -std=c++17 -Wall -Wextra -fprofile-arcs -ftest-coverage -c euler_circuit.cpp
g++ -std=c++17 -Wall -Wextra -fprofile-arcs -ftest-coverage main.o euler_circuit.o -o euler_circuit
make[1]: Leaving directory '/home/samurai/cs/Operating-Systems/Operating-Systems-Ex4/q1-4'
 Valid input
./euler_circuit -n 100 -e 300 -s 1 > /dev/null || true
gcov main.cpp euler_circuit.cpp > gcov_outputs/stdout1.txt
 Invalid argument
./euler_circuit -x 5 > /dev/null || true
./euler_circuit: invalid option -- 'x'
Usage: ./euler_circuit -n num_vertices -e num_edges -s seed
gcov main.cpp euler_circuit.cpp > gcov_outputs/stdout2.txt
 Invalid argument values
./euler_circuit -n -1 -e 300 -s 1 > /dev/null || true
Number of vertices must be positive.
Usage: ./euler_circuit -n num_vertices -e num_edges -s seed
./euler_circuit -n 100 -e -1 -s 1 > /dev/null || true
Number of edges cannot be negative.
Usage: ./euler_circuit -n num_vertices -e num_edges -s seed
./euler_circuit -n 2 -e 300 -s 1 > /dev/null || true
Too many edges for the number of vertices.
Usage: ./euler_circuit -n num_vertices -e num_edges -s seed
gcov main.cpp euler_circuit.cpp > gcov_outputs/stdout3.txt
 0 edges
./euler_circuit -n 100 -e 0 -s 1 > /dev/null || true
/home/samurai/cs/Operating-Systems/Operating-Systems-Ex4/q1-4/show_graph.py:8: UserWarning: loadtxt: input contained no data: "graph.txt"
  data = np.loadtxt('graph.txt', skiprows=1)
gcov main.cpp euler_circuit.cpp > gcov_outputs/stdoutFinal.txt
mv *.gcov gcov_outputs || true
mv *.gcda gcov_outputs || true
mv *.gcno gcov_outputs || true
samurai@DESKTOP-O04V85G:~/cs/Operating-Systems/Operating-Systems-Ex4/q1-4$
```

We can see that after the first run (stdout1.txt)
main.cpp had 83.1% coverage
Euler_circuit.cpp had 100% coverage (that's because we included many tests in main.cpp that cover all the Graph class code in one run)

q1-4 > gcov_outputs > stdout1.txt

```
1   File 'main.cpp'
2   Lines executed:83.10% of 71
3   Creating 'main.cpp.gcov'
```

q1-4 > gcov_outputs > stdout1.txt

```
48
49  File 'euler_circuit.cpp'
50  Lines executed:100.00% of 91
51  Creating 'euler_circuit.cpp.gcov'
```

After the last run (stdoutFinal.txt)
Main.cpp and Euler_circuit.cpp both have 100% coverage

q1-4 > gcov_outputs > stdoutFinal.txt

```
1   File 'main.cpp'
2   Lines executed:100.00% of 71
3   Creating 'main.cpp.gcov'
```

q1-4 > gcov_outputs > stdoutFinal.txt

```
49  File 'euler_circuit.cpp'
50  Lines executed:100.00% of 91
51  Creating 'euler_circuit.cpp.gcov'
```

Profiling (GPROF):

All the GPROF files and stdouts are located at `/gprof_outputs`

By using `make profile` we get the profiling data, by recompiling the program with profiling flags and running it. We don't have much to test because we are not trying to compare different implementations so we just ran gprof with a very large number of edges to generate and saw it's performance:

For 10,000 vertices and 300,000 edges:

```
297    0.00      0.50      0.00        1     0.00   303.78  Graph::generateRandomEdges(int, unsigned int)
```

It took a total of 0.5 cumulative time in seconds, to generate the graph and search for the euler cycle in it.

Pretty good :))

We also notice that the main time consuming function is the generation of edges, because its second in the list and is 60% of all cumulated time

```
348   -----------------------------------------------
349                  0.00    0.30       1/1           main [1]
350   [2]     60.8   0.00    0.30       1         Graph::generateRandomEdges(int, unsigned int) [2]
351                  0.00    0.21  300000/300019      Graph::addEdge(int, int) [5]
352                  0.00    0.06  300006/300006      std::_Rb_tree_const_iterator<int> std::find<std::_Rb
353                  0.00    0.03  300006/499788      std::multiset<int, std::less<int>, std::allocator<in
354                  0.00    0.00  600012/799789      std::multiset<int, std::less<int>, std::allocator<in
355                  0.00    0.00  900018/1745202     std::vector<std::multiset<int, std::less<int>, std::
356                  0.00    0.00  300006/300006      std::operator==(std::_Rb_tree_const_iterator<int> co
357   -----------------------------------------------
```

Surprisingly, the find euler was only 14% of the time

```
440 -----------------------------------------------
441                  0.00    0.07       6/6           main [1]
442 [17]    14.0     0.00    0.07       6         Graph::findEulerCircuit() [17]
443                  0.00    0.05       6/6           Graph::hasEulerCircuit() [28]
444                  0.00    0.02       2/8           std::vector<std::multiset<int, std::less<int>, std::allocator<int> >, std::all
445                  0.00    0.00       2/8           std::vector<std::multiset<int, std::less<int>, std::allocator<int> >, std::all
446                  0.00    0.00      10/600048      std::multiset<int, std::less<int>, std::allocator<int> >::insert(int const&) [
447                  0.00    0.00      10/10          std::multiset<int, std::less<int>, std::allocator<int> >::find(int const&) [65
448                  0.00    0.00      15/499788      std::multiset<int, std::less<int>, std::allocator<int> >::begin() const [25]
449                  0.00    0.00      10/10          std::multiset<int, std::less<int>, std::allocator<int> >::erase[abi:cxx11](std
450                  0.00    0.00      15/2100454     std::_Rb_tree_const_iterator<int>::operator*() const [41]
451                  0.00    0.00       7/7           std::stack<int, std::deque<int, std::allocator<int> > >::push(int const&) [73]
452                  0.00    0.00       7/7           std::vector<int, std::allocator<int> >::push_back(int const&) [74]
453                  0.00    0.00      10/2100453     std::_Rb_tree_const_iterator<int>::operator++() [44]
454                  0.00    0.00      10/799789      std::multiset<int, std::less<int>, std::allocator<int> >::end() const [56]
455                  0.00    0.00      63/1745202     std::vector<std::multiset<int, std::less<int>, std::allocator<int> >, std::all
456                  0.00    0.00      20/2600226     std::operator!=(std::_Rb_tree_const_iterator<int> const&, std::_Rb_tree_const_
457                  0.00    0.00      18/45321       std::multiset<int, std::less<int>, std::allocator<int> >::empty() const [130]
458                  0.00    0.00      14/14          std::stack<int, std::deque<int, std::allocator<int> > >::empty() const [156]
459                  0.00    0.00       7/7           std::stack<int, std::deque<int, std::allocator<int> > >::top() [222]
460                  0.00    0.00       7/7           std::stack<int, std::deque<int, std::allocator<int> > >::pop() [221]
461                  0.00    0.00       6/6           std::vector<int, std::allocator<int> >::vector() [256]
462                  0.00    0.00       2/8           std::allocator<std::multiset<int, std::less<int>, std::allocator<int> > >::all
463                  0.00    0.00       2/24          std::allocator<std::multiset<int, std::less<int>, std::allocator<int> > >::~al
464                  0.00    0.00       2/2           std::stack<int, std::deque<int, std::allocator<int> > >::stack<std::deque<int,
465                  0.00    0.00       2/2           std::stack<int, std::deque<int, std::allocator<int> > >::~stack() [297]
466 -----------------------------------------------
```

Valgrind (Memory check):

The test results are located at `/valgrind_outputs`

By using `make valgrind`, the makefile runs

```
valgrind: $(TARGET)
    valgrind --tool=memcheck $(VALGRIND_FLAGS) ./$(TARGET) -n 10000 -e 30000 -s 1 2> valgrind_output.txt
```

Results:

```
==88978== HEAP SUMMARY:
==88978==     in use at exit: 0 bytes in 0 blocks
==88978==   total heap usage: 60,073 allocs, 60,073 frees, 2,959,840 bytes allocated
==88978==
==88978== All heap blocks were freed -- no leaks are possible
==88978==
==88978== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

No leaks :))

Callgrind (using kcachegrind to visualize):

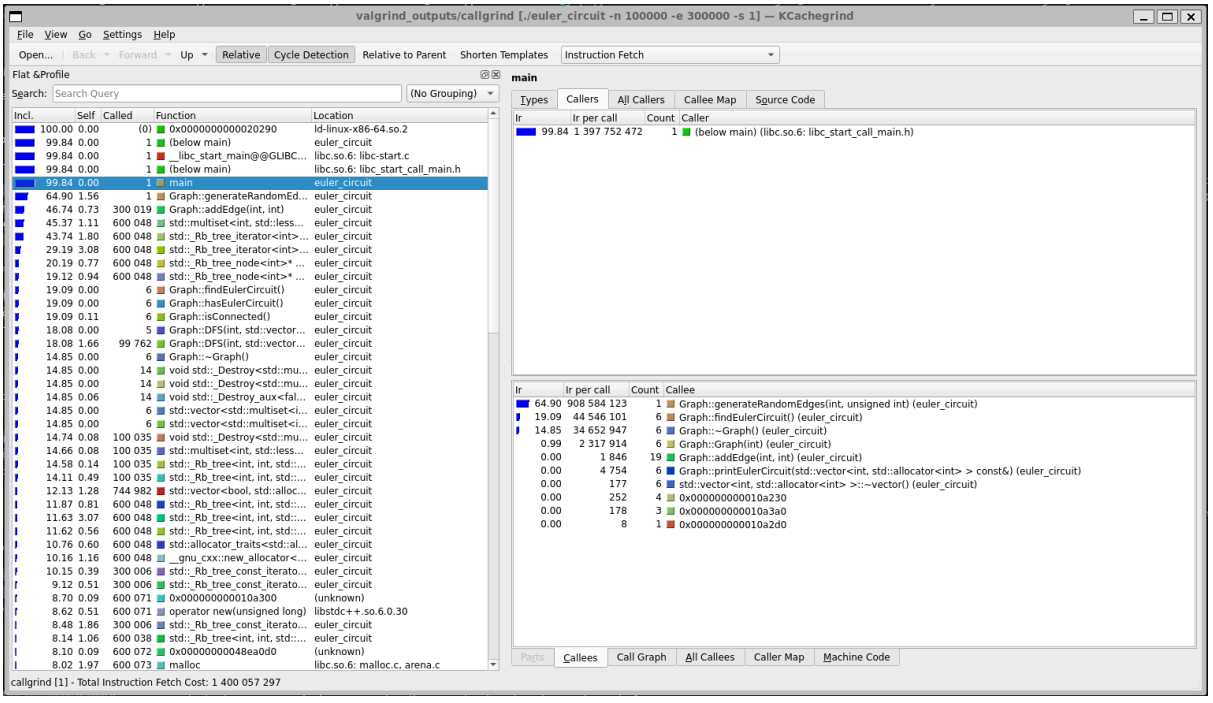
We ran callgrind, again using `make callgrind` which runs

```
81  callgrind: $(TARGET)
82      valgrind --tool=callgrind --callgrind-out-file=$(VALGRIND_OUTPUTS)/callgrind.out ./$(TARGET) -n 100000 -e 300000 -s 1 2> $(VALGRIND_OUTPUTS)/callgrind_stdout.txt
83      callgrind_annotate $(VALGRIND_OUTPUTS)/callgrind.out > $(VALGRIND_OUTPUTS)/callgrind_annotate.txt
84      kcachegrind $(VALGRIND_OUTPUTS)/callgrind
85
```

All outputs, including the stdout of the run are in `/valgrind_outputs`

```
==95818== Callgrind, a call-graph generating cache profiler
==95818== Copyright (C) 2002-2017, and GNU GPL'd, by Josef Weidendorfer et al.
==95818== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==95818== Command: ./euler_circuit -n 100000 -e 300000 -s 1
==95818==
==95818== For interactive control, run 'callgrind_control -h'.
==95818== brk segment overflow in thread #1: can't grow to 0x485b000
==95818== (see section Limitations in user manual)
==95818== NOTE: further instances of this message will not be shown
==95818==
==95818== Events    : Ir
==95818== Collected : 1400057297
==95818==
==95818== I   refs:      1,400,057,297
```

And then ran kcachegrind to visualize the graph



Questions 5+6

First run, simply

`valgrind --tool=memcheck -v --leak-check=full --show-leak-kinds=all --error-exitcode=99 ./hello`

With debug on (-g flag)

```
You entered: ./hello
--102763-- REDIR: 0x49083e0 (libc.so.6:free) redirected to 0x484b210 (free)
==102763==
==102763== HEAP SUMMARY:
==102763==     in use at exit: 9 bytes in 1 blocks
==102763==   total heap usage: 2 allocs, 1 frees, 1,033 bytes allocated
==102763==
==102763== Searching for pointers to 1 not-freed blocks
==102763== Checked 107,856 bytes
==102763==
==102763== 9 bytes in 1 blocks are definitely lost in loss record 1 of 1
==102763==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==102763==    by 0x10922A: main (in /home/samurai/cs/Operating-Systems/Operating-Systems-Ex4/q5/hello)
==102763==
==102763== LEAK SUMMARY:
==102763==    definitely lost: 9 bytes in 1 blocks
==102763==    indirectly lost: 0 bytes in 0 blocks
==102763==      possibly lost: 0 bytes in 0 blocks
==102763==    still reachable: 0 bytes in 0 blocks
==102763==         suppressed: 0 bytes in 0 blocks
==102763==
==102763== Use --track-origins=yes to see where uninitialised values come from
==102763== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
==102763==
==102763== 1 errors in context 1 of 2:
==102763== Conditional jump or move depends on uninitialised value(s)
==102763==    at 0x109234: main (in /home/samurai/cs/Operating-Systems/Operating-Systems-Ex4/q5/hello)
==102763==
==102763== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Add `--track-origins=yes` to see where uninitialised values come from

```
==103939==  Uninitialised value was created by a stack allocation
==103939==    at 0x1091C9: main (in /home/samurai/cs/Operating-Systems/Operating-Systems-Ex4/q5/hello)
```

Problem is with stack allocation, we can't get much from this so we added `--vgdb-error=0` to run gdb hooked with valgrind

Ran valgrind, got this:

```
==109982== TO DEBUG THIS PROCESS USING GDB: start GDB like this
==109982==   /path/to/gdb ./hello
==109982== and then give GDB the following command
==109982==   target remote | /usr/bin/vgdb --pid=109982
==109982== --pid is optional if only one valgrind process is running
```

Ran gdb as said

```
samurai@DESKTOP-004V85G:~/cs/Operating-Systems/Operating-Systems-Ex4/q5$ gdb hello
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 12.1
Copyright (C) 2022 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-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from hello...
(gdb)
```

```
Reading symbols from hello...
(gdb) target remote | /usr/bin/vgdb --pid=109982
```

Break on main and c for continue

```
(gdb) break main
Breakpoint 1 at 0x1091dc: file hello.c, line 14.
(gdb) c
Continuing.

Breakpoint 1, main (argc=1, argv=0x1ffefffc68) at hello.c:14
14          int i, length;      length = 0;
```

We stepped a few lines and found that

```
--109982-- REDIR: 0x49080a0 (libc.so.6:malloc) redirected to 0x4848820 (malloc)
==109982== Conditional jump or move depends on uninitialised value(s)
==109982==    at 0x109234: main (in /home/samurai/cs/Operating-Systems/Operating-Systems-Ex4/q5/hel
lo)
==109982==  Uninitialised value was created by a stack allocation
==109982==    at 0x1091C9: main (in /home/samurai/cs/Operating-Systems/Operating-Systems-Ex4/q5/hel
lo)
==109982==
==109982== (action on error) vgdb me ...
==109982== Continuing ...
```

```
(gdb) s
15          for(i=0; i<argc; i++) {
(gdb) s
16              length += strlen(argv[i])+1;
(gdb) s
17              string = malloc(length+1);
(gdb) s
20              if(string_so_far != (char *)0)
(gdb) s
22              else *string = '\0';
(gdb)
```

Inside the if, we try to compare string_so_far to NULL but string_so_far has no value at all
We should have done `char *string_so_far = NULL;` at the beginning.
We added it, ran again and saw no problem at this exact location

```
--116906-- REDIR: 0x490cac0 (libc.so.6:strncasecmp) redirected to 0x483f220 (_vgnU_ifunc_wrapper)
--116906-- REDIR: 0x490df80 (libc.so.6:rawmemchr) redirected to 0x483f220 (_vgnU_ifunc_wrapper)
--116906-- REDIR: 0x4a00610 (libc.so.6:__strrchr_avx2) redirected to 0x484e810 (rindex)
--116906-- REDIR: 0x4a007e0 (libc.so.6:__strlen_avx2) redirected to 0x484ed10 (strlen)
--116906-- REDIR: 0x49080a0 (libc.so.6:malloc) redirected to 0x4848820 (malloc)
```

```
20              if(string_so_far != (char *)0)
(gdb) n
22              else *string = '\0';
(gdb) n
23              strcat(string, argv[i]);
(gdb)
```

We continued to the end, but still had memory leakage because we didn't deal with it

```
==116906== LEAK SUMMARY:
==116906==    definitely lost: 9 bytes in 1 blocks
==116906==    indirectly lost: 0 bytes in 0 blocks
==116906==      possibly lost: 0 bytes in 0 blocks
==116906==    still reachable: 0 bytes in 0 blocks
==116906==         suppressed: 0 bytes in 0 blocks
==116906==
==116906== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
make: *** [makefile:10: valgrind] Error 99
samurai@DESKTOP-O04V85G:~/cs/Operating-Systems/Operating-Systems-Ex4/q5$
```

```
(gdb) n
20              if(string_so_far != (char *)0)
(gdb) n
22              else *string = '\0';
(gdb) n
23              strcat(string, argv[i]);
(gdb) c
Continuing.
[Inferior 1 (Remote target) exited normally]
(gdb)
```

We ran again, saw that string_so_far wasn't freed at the end
Adding `free(string_so_far);` fixed it

```
You entered: ./hello
--119861-- REDIR: 0x49083e0 (libc.so.6:free) redirected to 0x484b210 (free)
==119861==
==119861== HEAP SUMMARY:
==119861==     in use at exit: 0 bytes in 0 blocks
==119861==   total heap usage: 2 allocs, 2 frees, 1,033 bytes allocated
==119861==
==119861== All heap blocks were freed -- no leaks are possible
==119861==
==119861== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

```
relaying data between gdb and process 119861
warning: remote target does not support file transfer, attempting to access files from local filesy
stem.
Reading symbols from /lib64/ld-linux-x86-64.so.2...
Reading symbols from /usr/lib/debug/.build-id/41/86944c50f8a32b47d74931e3f512b811813b64.debug...
0x0000000000402090 in _start () from /lib64/ld-linux-x86-64.so.2
(gdb) c
Continuing.
[Inferior 1 (Remote target) exited normally]
(gdb)
```

Question 7

The problem with this code is that we create multiple threads that run a function that `pthread_create(&threads[t], NULL, square, (void *)params[t]);`

The square function that is being ran by the threads is modifying the global variable `accum` without and sync mechanism to prevent race condition.
This might create unexpected behaviour because without locking, multiple threads can try to read, modify, and write to `accum` at the same time, which might give us incorrect results for the square.

We ran `valgrind --tool=helgrind ./race` to check for the race condition variable and got

```
==130321== ---Thread-Announcement------------------------------------------
==130321==
==130321== Thread #3 was created
==130321==    at 0x498F9F3: clone (clone.S:76)
==130321==    by 0x49908EE: __clone_internal (clone-internal.c:83)
==130321==    by 0x48FE6D8: create_thread (pthread_create.c:295)
==130321==    by 0x48FF1FF: pthread_create@@GLIBC_2.34 (pthread_create.c:828)
==130321==    by 0x4853767: ??? (in /usr/libexec/valgrind/vgpreload_helgrind-amd64-linux.so)
==130321==    by 0x1092C3: main (race.c:31)
==130321==
==130321== ----------------------------------------------------------------
==130321==
==130321== Possible data race during read of size 8 at 0x10C018 by thread #4
==130321== Locks held: none
==130321==    at 0x10920B: square (race.c:19)
==130321==    by 0x485396A: ??? (in /usr/libexec/valgrind/vgpreload_helgrind-amd64-linux.so)
==130321==    by 0x48FEAC2: start_thread (pthread_create.c:442)
==130321==    by 0x498FA03: clone (clone.S:100)
==130321==
==130321== This conflicts with a previous write of size 8 by thread #3
==130321== Locks held: none
==130321==    at 0x109215: square (race.c:19)
==130321==    by 0x485396A: ??? (in /usr/libexec/valgrind/vgpreload_helgrind-amd64-linux.so)
==130321==    by 0x48FEAC2: start_thread (pthread_create.c:442)
==130321==    by 0x498FA03: clone (clone.S:100)
==130321==  Address 0x10c018 is 0 bytes inside data symbol "accum"
==130321==
==130321== ----------------------------------------------------------------
==130321==
==130321== Possible data race during write of size 8 at 0x10C018 by thread #4
==130321== Locks held: none
==130321==    at 0x109215: square (race.c:19)
==130321==    by 0x485396A: ??? (in /usr/libexec/valgrind/vgpreload_helgrind-amd64-linux.so)
==130321==    by 0x48FEAC2: start_thread (pthread_create.c:442)
==130321==    by 0x498FA03: clone (clone.S:100)
==130321==
==130321== This conflicts with a previous write of size 8 by thread #3
==130321== Locks held: none
==130321==    at 0x109215: square (race.c:19)
==130321==    by 0x485396A: ??? (in /usr/libexec/valgrind/vgpreload_helgrind-amd64-linux.so)
==130321==    by 0x48FEAC2: start_thread (pthread_create.c:442)
==130321==    by 0x498FA03: clone (clone.S:100)
==130321==  Address 0x10c018 is 0 bytes inside data symbol "accum"
==130321==
2870
==130321==
==130321== Use --history-level=approx or =none to gain increased speed, at
==130321== the cost of reduced accuracy of conflicting-access information
==130321== For lists of detected and suppressed errors, rerun with: -s
==130321== ERROR SUMMARY: 36 errors from 2 contexts (suppressed: 344 from 20)
```

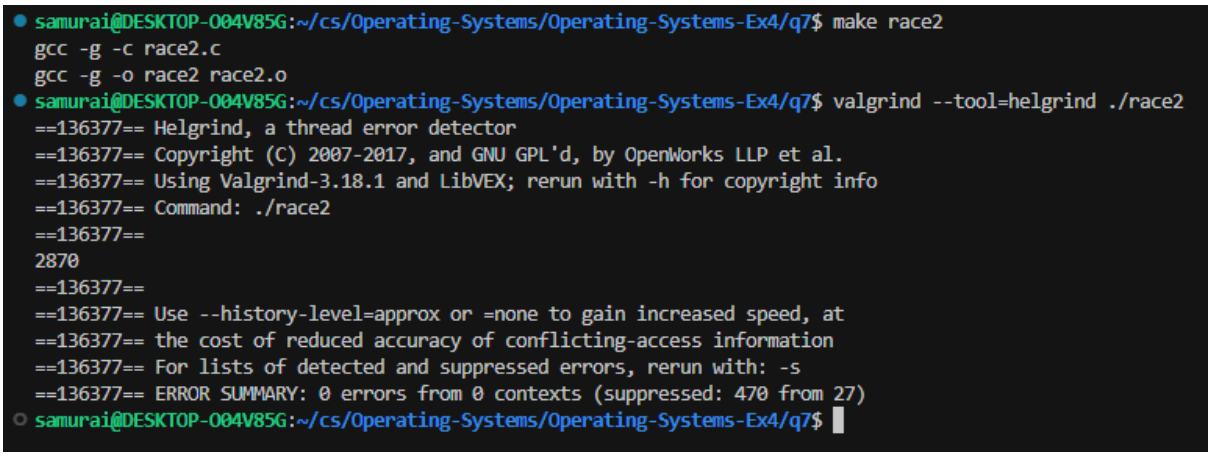
Helgrind detected
“==130321== Possible data race during read of size 8 at ***0x10C018*** by thread #4”
“==130321== Possible data race during *write* of size 8 at ***0x10C018*** by thread #4”
Meaning the problem is at the mentioned memory address.
We also see this memory address is associated with accum:
“==130321== Address ***0x10c018*** is 0 bytes inside data symbol "***accum***"”
So accum is the possible condition variable, with conflicts during reading and writing.

Helgrind also tells us the problem is at square() race.c:19, which is “accum += x * x;” - accessing accum.

To fix this we will add a mutex to protect the access to `accum` inside pthreads, lock the mutex before changing accum and unlock after. At the end destory the mutex to free memory.

Fixed code is inside `race2.c`

Just to make sure, we ran helgrind on our code and saw that the problem is really fixed



Question 8

We implemented a Singleton and Guard (Scope Mutex) and added it to race.c from previous question to see if it work.

We notice the singletons are the instance same because we got 1 in the print - meaning the singleton is implemented correctly.

And we notice that by running helgrind, our Guard works well as a scope mutex.