<u>Operation system assigntment 4</u>

Valgrind, Graph Data Structure, and Euler

Submitted by: Wasim shebalny : 323944280

Shifaa khatib : 324095702

Question 1-3: run $ make and then run

$ ./graph -e <num_edges> -v <num_vertices> -s <seed>

Replace <num_edges>, <num_vertices>, and <seed> with your specific values.

Question 4: Code Coverage and Profiling Reports, valgrind and Callgraph Report Generation.

<u>-coverage:</u>

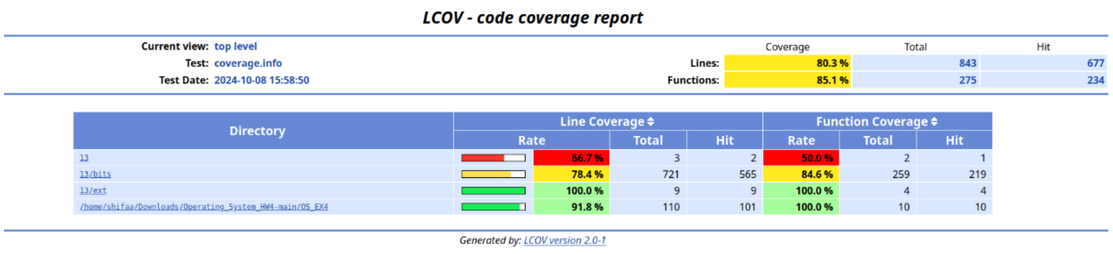
Run $ make coverage , After running make coverage, the following line will be executed : ./graph -e 1000 -v 100 -s 42

After executing the program, the coverage data will be collected, and a coverage report will be generated.

The coverage tool used is lcov, which produces an HTML report.

To open the HTML coverage report, use the following command:

firefox coverage/index.html



LCOV - code coverage report

			Coverage	Total	Hit
Current view:	top level				
Test:	coverage.info	Lines:	80.3 %	843	677
Test Date:	2024-10-08 15:58:50	Functions:	85.1 %	275	234

Directory	Line Coverage				Function Coverage		
		Rate	Total	Hit	Rate	Total	Hit
13	🟥	66.7 %	3	2	50.0 %	2	1
13/bits	🟨	78.4 %	721	565	84.6 %	259	219
13/ext	🟩	100.0 %	9	9	100.0 %	4	4
/home/shifaa/Downloads/Operating_System_HW4-main/OS_EX4	🟩	91.8 %	110	101	100.0 %	10	10

Generated by: LCOV version 2.0-1

<u>-profiling:</u>

Run $ make profile , After running make coverage, the following line will be executed : ./graph -e 1000 -v 100 -s 42

And It will output gprof_report.txt.

Valgrind:

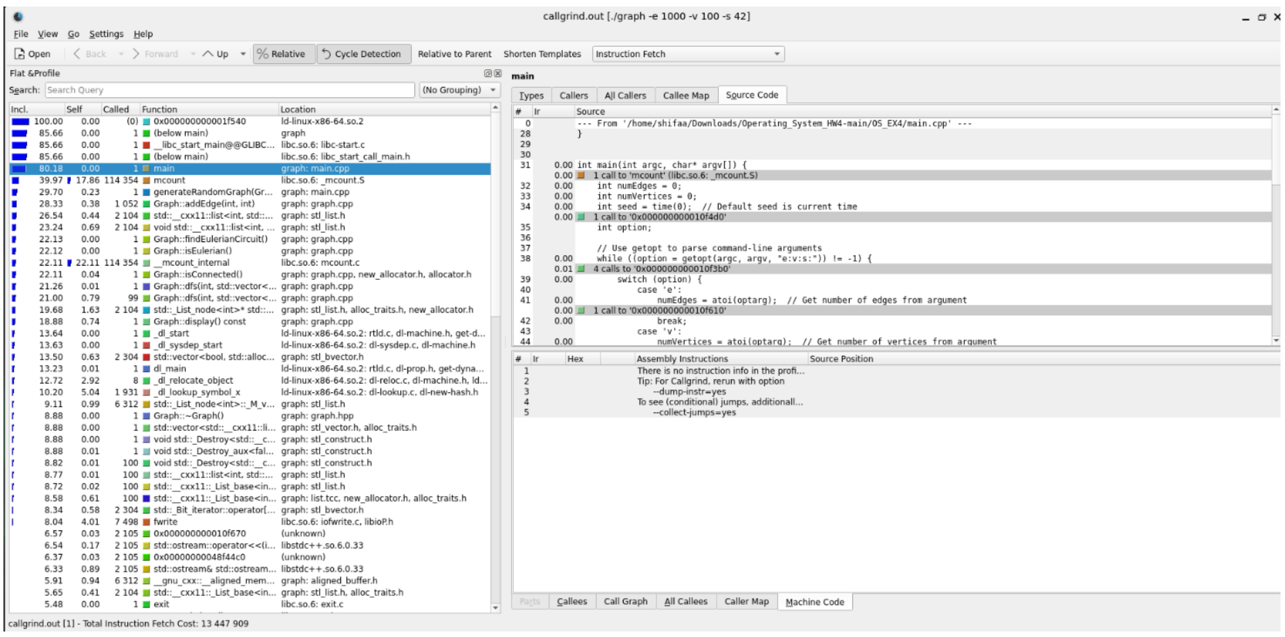
Run $ make valgrind , After running make coverage, the following line will be executed : ./graph -e 1000 -v 100 -s 42

And It will output valgrind_report.txt.

Callgraph:

we generate a **call graph** to visualize function calls within the program. This helps to understand the program flow and identify performance bottlenecks.

Run $ make callgraph it will output callgrind.out report , also:



callgrind.out [./graph -e 1000 -v 100 -s 42]

Flat &Profile — Search: Search Query — (No Grouping)

Incl.	Self	Called	Function	Location
100.00	0.00	(0)	0x000000000001f540	ld-linux-x86-64.so.2
85.66	0.00	1	(below main)	graph
85.66	0.00	1	__libc_start_main@@GLIBC...	libc.so.6: libc-start.c
85.66	0.00	1	(below main)	libc.so.6: libc_start_call_main.h
80.18	0.00	1	main	graph: main.cpp
39.97	17.86	114 354	mcount	libc.so.6: _mcount.S
29.70	0.23	1	generateRandomGraph(Gr...	graph: main.cpp
28.33	0.38	1 052	Graph::addEdge(int, int)	graph: graph.cpp
26.54	0.44	2 104	std::__cxx11::list<int, std::...	graph: stl_list.h
23.24	0.69	2 104	void std::__cxx11::list<int, ...	graph: stl_list.h
22.13	0.00	1	Graph::findEulerianCircuit()	graph: graph.cpp
22.12	0.00	1	Graph::isEulerian()	graph: graph.cpp
22.11	22.11	114 354	__mcount_internal	libc.so.6: mcount.c
22.11	0.04	1	Graph::isConnected()	graph: graph.cpp, new_allocator.h, allocator.h
21.26	0.01	1	Graph::dfs(int, std::vector<...	graph: graph.cpp
21.00	0.79	99	Graph::dfs(int, std::vector<...	graph: graph.cpp
19.68	1.63	2 104	std::_List_node<int>* std::...	graph: stl_list.h, alloc_traits.h, new_allocator.h
18.88	0.74	1	Graph::display() const	graph: graph.cpp
13.64	0.00	1	_dl_start	ld-linux-x86-64.so.2: rtld.c, dl-machine.h, get-d...
13.63	0.00	1	_dl_sysdep_start	ld-linux-x86-64.so.2: dl-sysdep.c, dl-machine.h
13.50	0.63	2 304	std::vector<bool, std::alloc...	graph: stl_bvector.h
13.23	0.01	1	dl_main	ld-linux-x86-64.so.2: rtld.c, dl-prop.h, get-dyna...
12.72	2.92	8	_dl_relocate_object	ld-linux-x86-64.so.2: dl-reloc.c, dl-machine.h, ld...
10.20	5.04	1 931	_dl_lookup_symbol_x	ld-linux-x86-64.so.2: dl-lookup.c, dl-new-hash.h
9.11	0.99	6 312	std::_List_node<int>::_M_v...	graph: stl_list.h
8.88	0.00	1	Graph::~Graph()	graph: graph.hpp
8.88	0.00	1	std::vector<std::__cxx11::li...	graph: stl_vector.h, alloc_traits.h
8.88	0.00	1	void std::_Destroy<std::__c...	graph: stl_construct.h
8.88	0.01	1	void std::_Destroy_aux<fal...	graph: stl_construct.h
8.82	0.01	100	void std::_Destroy<std::__c...	graph: stl_construct.h
8.77	0.01	100	std::__cxx11::list<int, std::...	graph: stl_list.h
8.72	0.02	100	std::__cxx11::_List_base<in...	graph: stl_list.h
8.58	0.61	100	std::__cxx11::_List_base<in...	graph: list.tcc, new_allocator.h, alloc_traits.h
8.34	0.58	2 304	std::_Bit_iterator::operator[...	graph: stl_bvector.h
8.04	4.01	7 498	fwrite	libc.so.6: iofwrite.c, libioP.h
6.57	0.03	2 105	0x000000000010f670	(unknown)
6.54	0.17	2 105	std::ostream::operator<<(i...	libstdc++.so.6.0.33
6.37	0.03	2 105	0x00000000048f44c0	(unknown)
6.33	0.89	2 105	std::ostream& std::ostream...	libstdc++.so.6.0.33
5.91	0.94	6 312	__gnu_cxx::__aligned_mem...	graph: aligned_buffer.h
5.65	0.41	2 104	std::__cxx11::_List_base<in...	graph: stl_list.h, alloc_traits.h
5.48	0.00	1	exit	libc.so.6: exit.c

main — Source Code

```
#   Ir    Source
 0        --- From '/home/shifaa/Downloads/Operating_System_HW4-main/OS_EX4/main.cpp' ---
28        }
29
30
31  0.00  int main(int argc, char* argv[]) {
    0.00    1 call to 'mcount' (libc.so.6: _mcount.S)
32  0.00      int numEdges = 0;
33  0.00      int numVertices = 0;
34  0.00      int seed = time(0);  // Default seed is current time
    0.00    1 call to '0x000000000010f4d0'
35            int option;
36
37            // Use getopt to parse command-line arguments
38  0.00      while ((option = getopt(argc, argv, "e:v:s:")) != -1) {
    0.01    4 calls to '0x000000000010f3b0'
39  0.00          switch (option) {
40                    case 'e':
41  0.00                  numEdges = atoi(optarg);  // Get number of edges from argument
    0.00    1 call to '0x000000000010f610'
42  0.00                  break;
43                    case 'v':
44  0.00                  numVertices = atoi(optarg);  // Get number of vertices from argument
```

```
#  Ir  Hex  Assembly Instructions                        Source Position
1           There is no instruction info in the profi...
2           Tip: For Callgrind, rerun with option
                --dump-instr=yes
4           To see (conditional) jumps, additionall...
5               --collect-jumps=yes
```

callgrind.out [1] - Total Instruction Fetch Cost: 13 447 909

This picture shows a **Call Graph report** generated by Valgrind's **Callgrind** tool and visualized using **KCacheGrind**. It displays how much processing time each function in the program took during execution. Key functions, such as main, are listed alongside their **inclusive cost** (time spent in the function and functions it calls) and **self cost** (time spent within the function itself). The source code on the right shows how often each line of code was executed, allowing for detailed performance analysis and identifying bottlenecks in the code execution.

Question 5: detect and report memory leaks using Valgrind on the hello.c program.

Run $ make valgrind_hello And It will output valgrind_hello_report.txt.

Question 6: demonstrate Valgrind attached to a debugger (such as gdb), you can follow these steps: Steps to Attach Valgrind to GDB:

Open two terminals:

Left terminal : enter this commands:

1) Compile the program**:**

```
 $  gcc -g -o hello hello.c
```

2) Run Valgrind with GDB server:

```
 $ valgrind --vgdb=yes --vgdb-error=0 ./hello
```

--vgdb=yes tells Valgrind to run with the gdb server.

--vgdb-error=0 tells Valgrind to pause on the first error and wait for gdb to attach.

Second terminal: enter this commands:

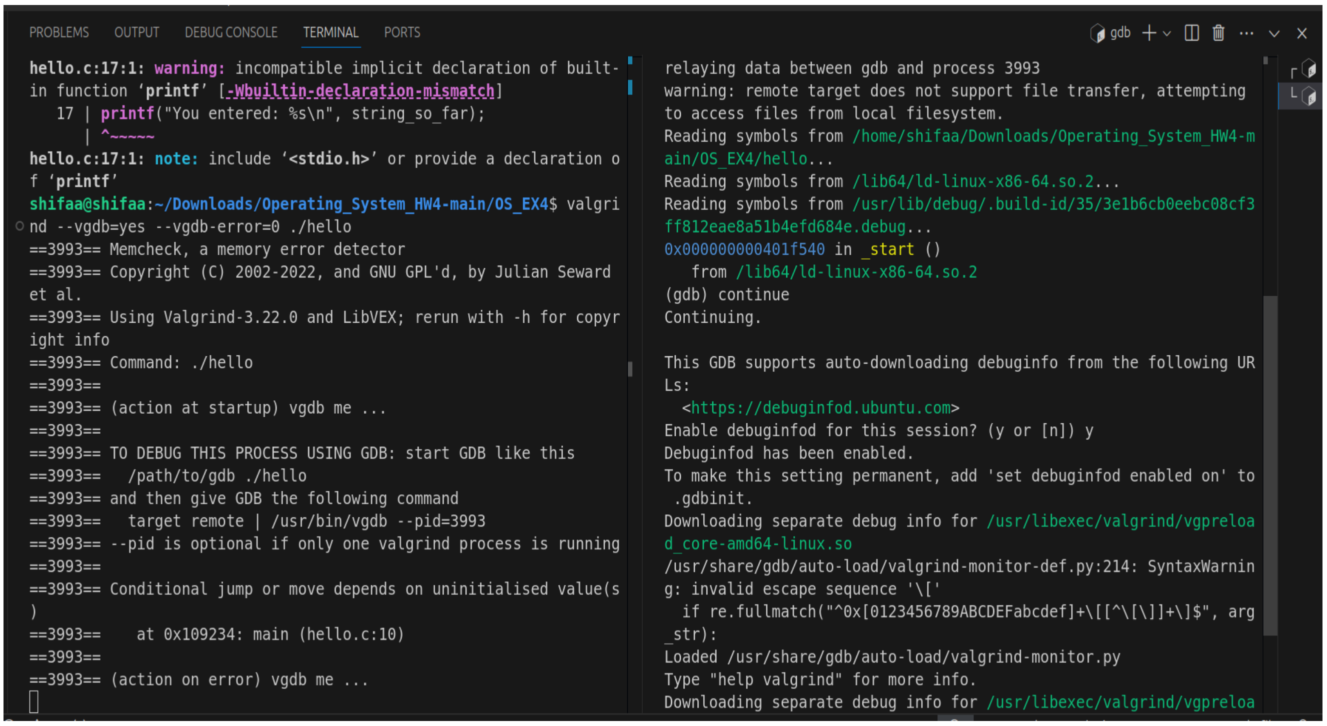
1) Attach GDB to Valgrind: $ gdb ./hello

2) connect to Valgrind by: $ target remote | vgdb

Once gdb is connected to Valgrind, it will allow you to debug and investigate the issue.

Example commands:

```
bt            # Get a backtrace of the current call stack
info locals    # Get information about local variables
step          # Step into the current function
next          # Move to the next line of code
continue       # Continue running the program
```



```
hello.c:17:1: warning: incompatible implicit declaration of built-
in function ‘printf’ [-Wbuiltin-declaration-mismatch]
   17 | printf("You entered: %s\n", string_so_far);
      | ^~~~~~
hello.c:17:1: note: include ‘<stdio.h>’ or provide a declaration o
f ‘printf’
shifaa@shifaa:~/Downloads/Operating_System_HW4-main/OS_EX4$ valgri
nd --vgdb=yes --vgdb-error=0 ./hello
==3993== Memcheck, a memory error detector
==3993== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward
et al.
==3993== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyr
ight info
==3993== Command: ./hello
==3993==
==3993== (action at startup) vgdb me ...
==3993==
==3993== TO DEBUG THIS PROCESS USING GDB: start GDB like this
==3993==   /path/to/gdb ./hello
==3993== and then give GDB the following command
==3993==   target remote | /usr/bin/vgdb --pid=3993
==3993== --pid is optional if only one valgrind process is running
==3993==
==3993== Conditional jump or move depends on uninitialised value(s
)
==3993==    at 0x109234: main (hello.c:10)
==3993==
==3993== (action on error) vgdb me ...
```

```
relaying data between gdb and process 3993
warning: remote target does not support file transfer, attempting
to access files from local filesystem.
Reading symbols from /home/shifaa/Downloads/Operating_System_HW4-m
ain/OS_EX4/hello...
Reading symbols from /lib64/ld-linux-x86-64.so.2...
Reading symbols from /usr/lib/debug/.build-id/35/3e1b6cb0eebc08cf3
ff812eae8a51b4efd684e.debug...
0x000000000401f540 in _start ()
   from /lib64/ld-linux-x86-64.so.2
(gdb) continue
Continuing.

This GDB supports auto-downloading debuginfo from the following UR
Ls:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to
 .gdbinit.
Downloading separate debug info for /usr/libexec/valgrind/vgpreloa
d_core-amd64-linux.so
/usr/share/gdb/auto-load/valgrind-monitor-def.py:214: SyntaxWarnin
g: invalid escape sequence '\['
  if re.fullmatch("^0x[0123456789ABCDEFabcdef]+\[[^\[\]]+\]$", arg
_str):
Loaded /usr/share/gdb/auto-load/valgrind-monitor.py
Type "help valgrind" for more info.
Downloading separate debug info for /usr/libexec/valgrind/vgpreloa
```

On the left side,we are running the command valgrind --vgdb=yes --vgdb-error=0 ./hello to launch **Valgrind** and start the hello program. Valgrind is waiting for us to attach GDB for debugging.

On the right side, **GDB** is started and attached to the process using the instructions provided by Valgrind. we executed gdb ./hello and followed the instructions from Valgrind to attach **GDB** to the running process via target remote | /usr/bin/vgdb --pid=3993.

GDB has successfully connected to Valgrind and is ready to debug the program.

Question 7 : Detect race conditions using Valgrind/Helgrind.

Run $ make valgrind_race And It will output valgrind_race_report.txt.

Question 8 : Singleton Mutex and Guard Class Implementation

Files:

MutexBase:

- Defines an abstract base class for mutex operations (lock and unlock), which are implemented by the derived class.

MutexImpl:

- Implements the singleton pattern using pthread_mutex_t from POSIX. This file defines how the mutex is locked and unlocked, ensuring that only one instance of the mutex is used in the entire application.

Guard:

- A scope guard class that locks the mutex in its constructor and unlocks it in the destructor, ensuring the mutex is properly locked and unlocked within a specific scope.

main.cpp:

- The main file that tests and demonstrates the use of the Singleton Mutex and Guard classes. It simulates a multi-threaded environment where threads safely access shared resources.