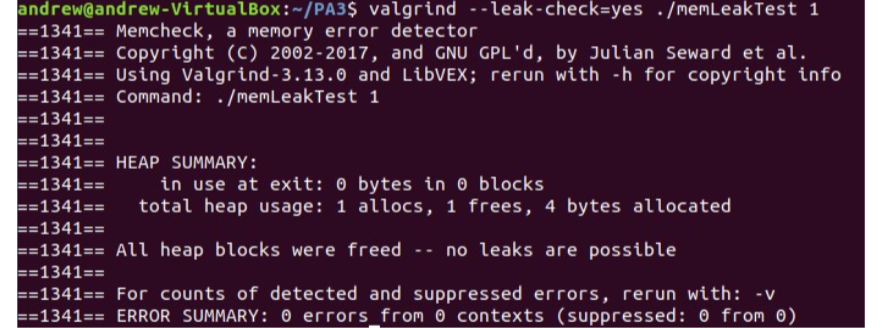
**Detecting Memory Leaks w/ gdb & valgrind: Test Data**

*How was the test data generated?*

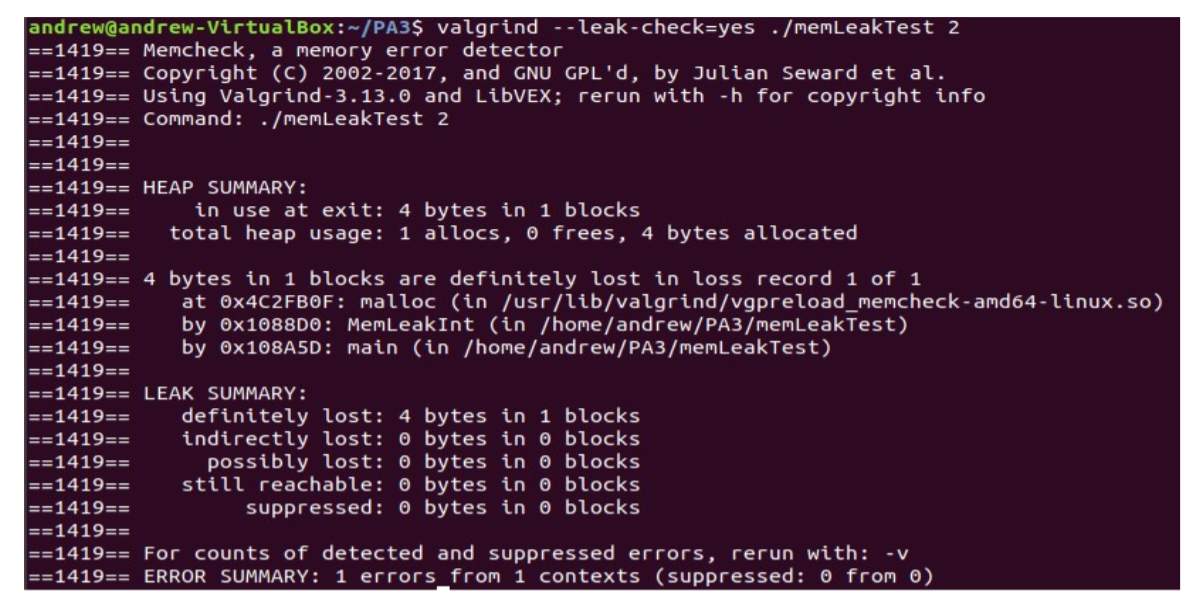
→ A memLeakTest.c file was created in order to aid with detecting memory leaks using *gdb* and *valgrind*. Each test function in this program is designed to test a certain variation of memory leakage. The purpose of each of these, along with test data, is presented in this document. First, we will discuss the results obtained using *valgrind*.

**Valgrind**

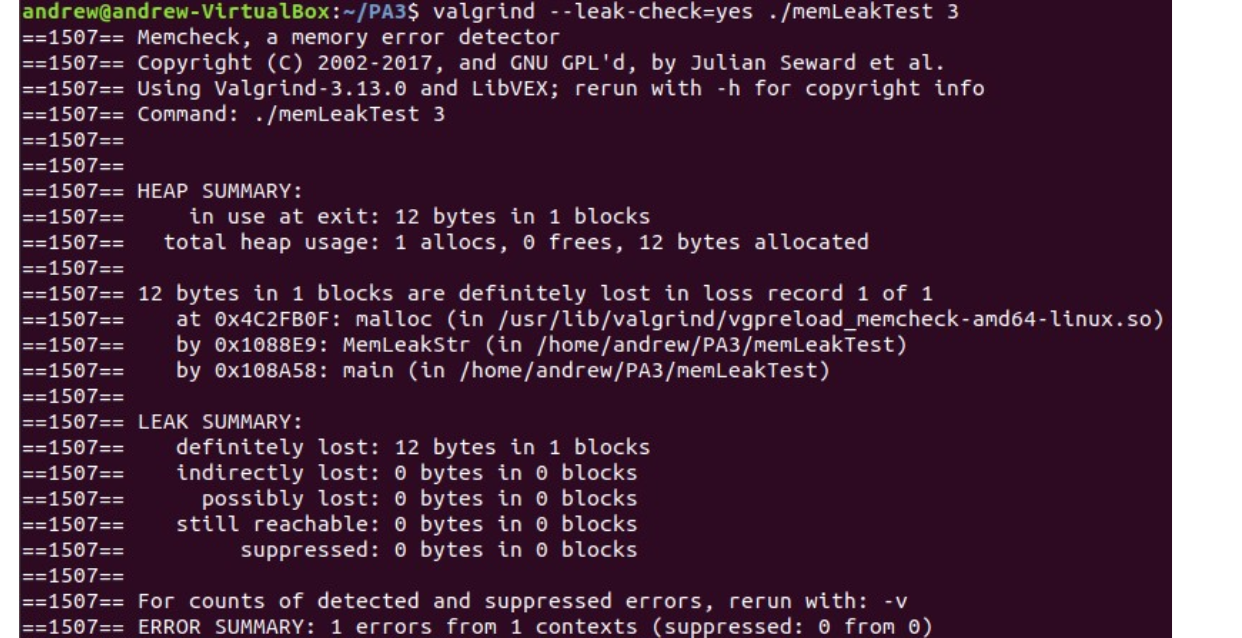
*Test #1 →* This test is just a reference function where memory for an integer is allocated and freed correctly. Accordingly, no memory leaks were expected, which was accurate.



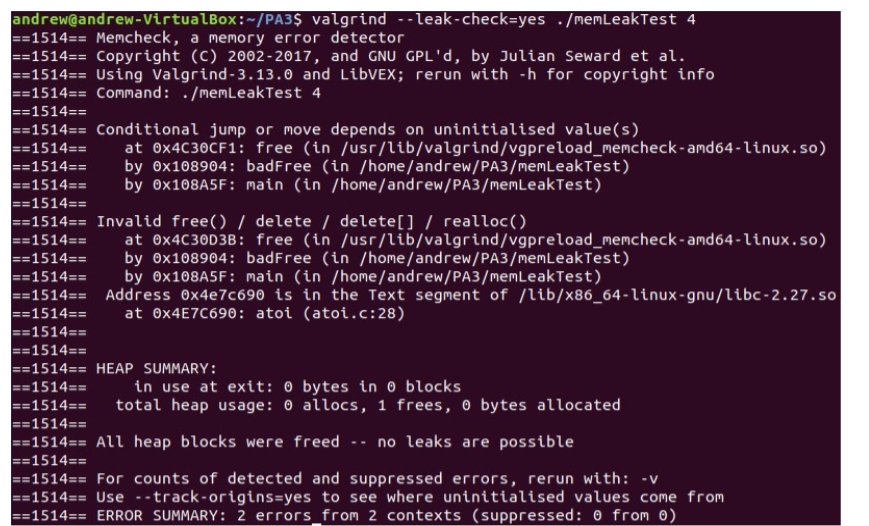
*Test #2 →* Here, memory for an integer (4 bytes) is allocated but not freed. The valgrind report does show a leak of 4 bytes as expected, which can be seen below:



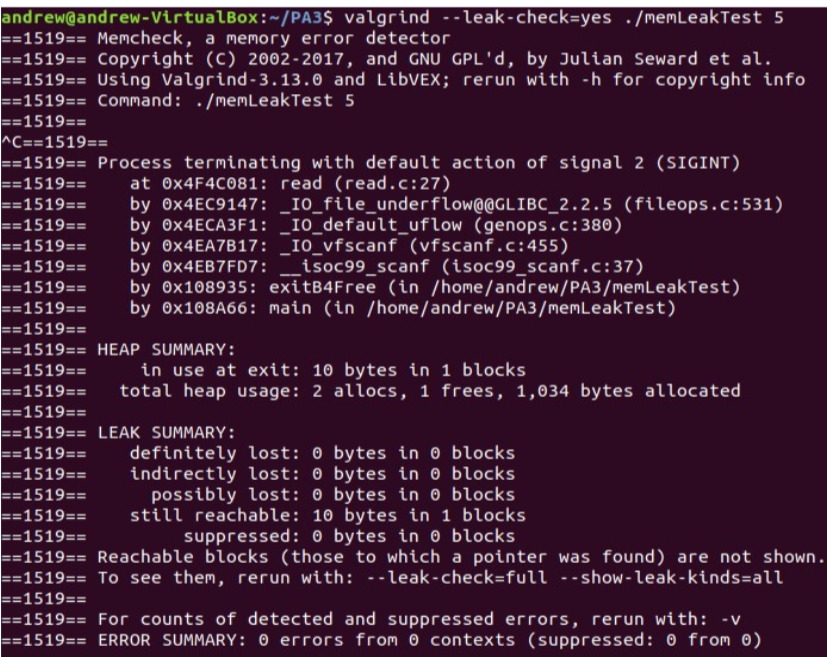
*Test #3 →* Here, memory for a string ("Hello World") is allocated but not freed. Accordingly, we would expect a memory leak of 11 + 1 (end of string character) = 12 bytes. The valgrind report does, in fact, show a memory leak of 12 bytes as expected, which can be seen below:



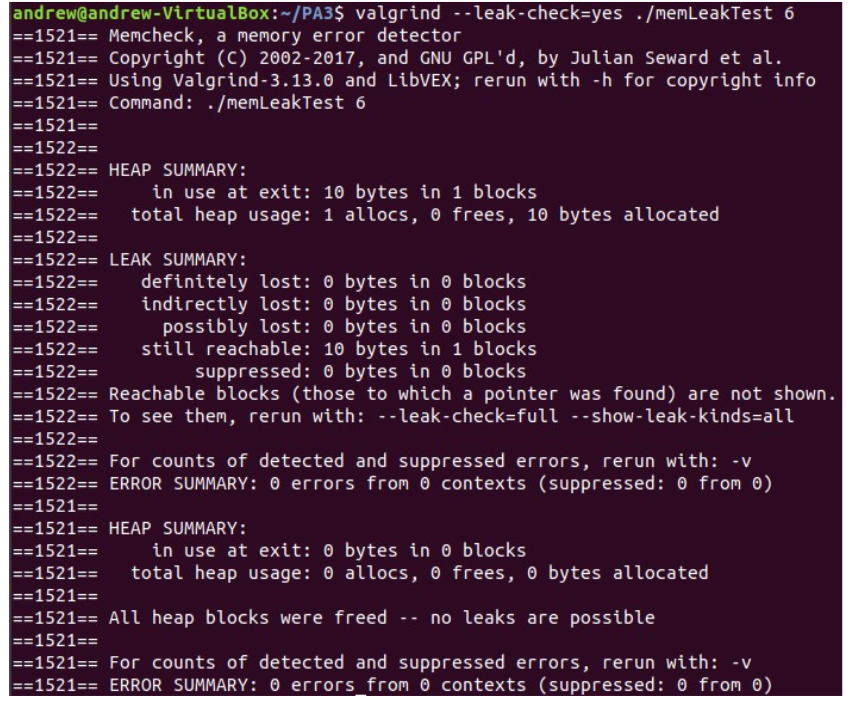
*Test #4 →* This test is used to show what would happen on an invalid free (trying to free memory location referred by a pointer that has not been allocated any memory). We would expect valgrind to display some sort of error message. This is exactly what happened, and the valgrind report for the same was as follows:



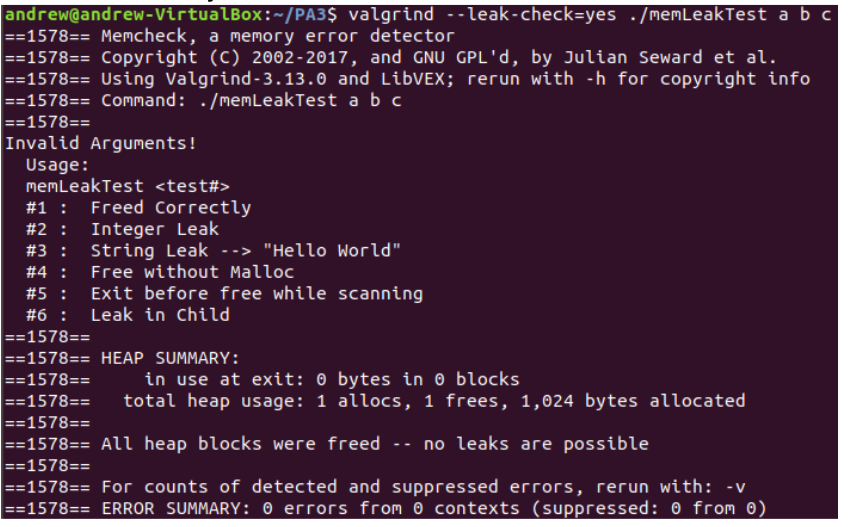
*Test #5 →* This test is used to show what would happen if a program allocates some memory (10 bytes in this case), but the user terminates the program prematurely using the Ctrl+C/Ctrl+D command. Even though there is a free() statement at the end of the code, the program was terminated before reaching that line of code and we would, therefore, expect a memory leak for the same. This valgrind report indeed showed that 10 bytes of memory was "still reachable" (because it wasn't freed properly, but the program terminated).



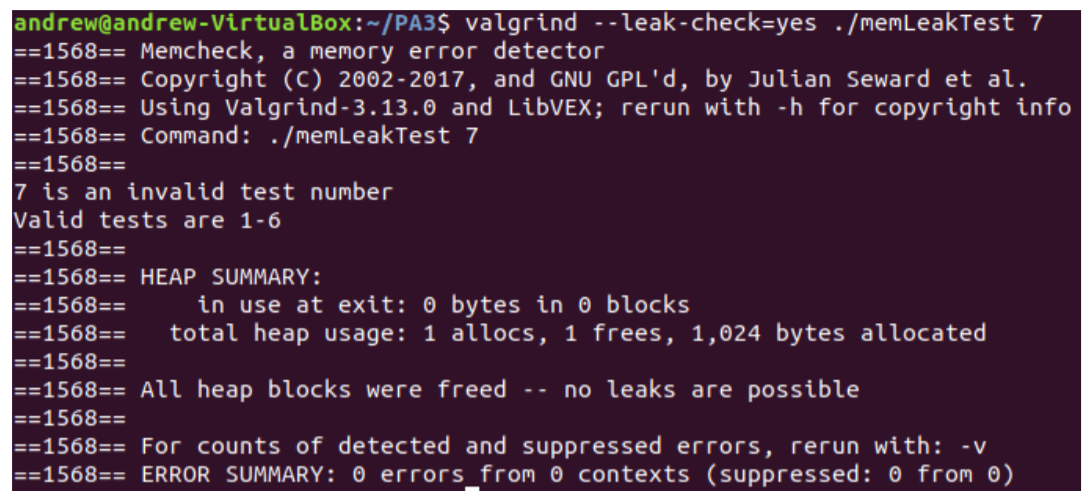
*Test #6 →* In this final test, we want to see what would happen if a process that causes a memory leak is reaped. We would expect a similar result here as in the previous case (because earlier, the shell did reap the process that prematurely terminated). Since the child process allocated 10 bytes of memory and exited without freeing it, and the parent process waited for the child to terminate & reaped it, there would be a memory leak of 10 bytes ("still reachable" memory). This was seen in the valgrind report shown below:



Invalid Arguments → When incorrect arguments were passed to this test program, the usage message was outputted to the screen, which can be seen below:



If the number of arguments were correct, but an invalid test number was inputted, an error message was displayed, as can be seen below:



**GDB**

When these tests were run using gdb, no significant detail suggesting a memory leak of any sort was observed. This was expected because gdb is technically not a memory leak detection tool (to use it for such a purpose, one may need to combine it with other tools or commands). Sample gdb output for one of test cases is shown below. There were no significant differences in the outputs for any other test case.

