PA3 Part 1 Questions

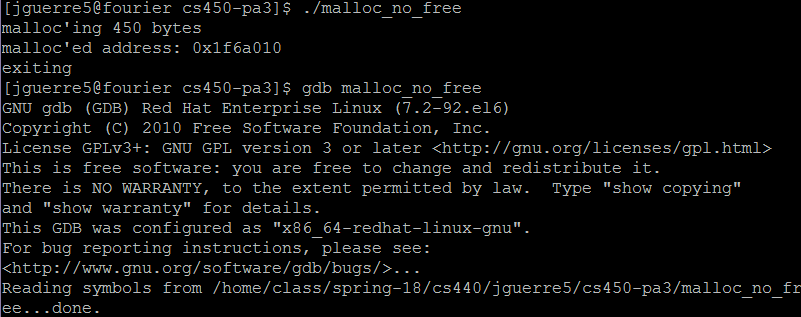
1. Running the program prints a message indicating the memory address of the allocated memory. Since it is never freed there is a memory leak, which means that the space is never deallocated. This can be found in gdb by using breakpoints and checking stack frames. Checking the local variables after the program has returned reveals that the variable’s memory address is still valid. Using Valgrind it is immediately apparent that there is a problem with the program. It easily finds that 450 bytes were lost due to a memory leak and it can point out the allocation that caused the problem.

Commands used: gdb “program name”, break “line number”, run, continue, info locals, bt, valgrind –leak-check=yes ./“program name”

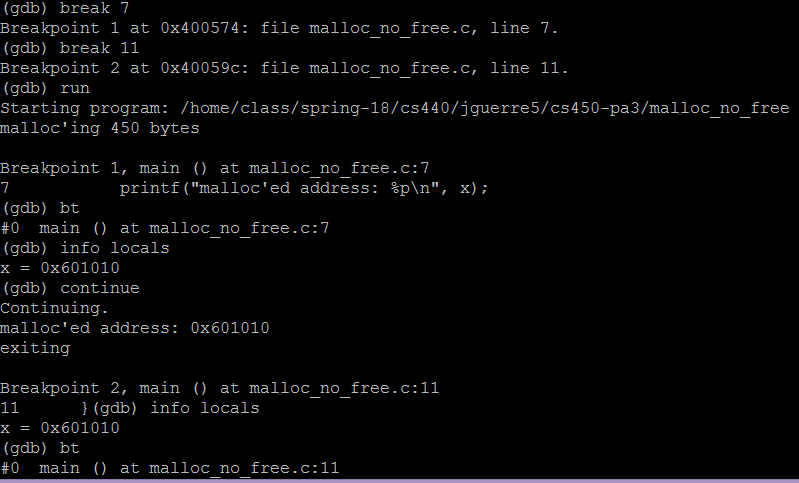
1. The program runs and returns the same value that it did before the data was freed. Valgrind acknowledges that the data was freed and that there are no memory leaks. However, Valgrind complains that there was an invalid read at the line where the data is printed after it is freed. The message states that the data being accessed is within a block of data that was freed in a prior line of code. Valgrind says that there is 1 error from 1 context referring to the invalid read.

Comands used: valgrind –leak-check=yes ./“program name”

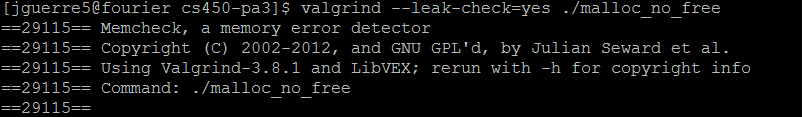
Screenshots



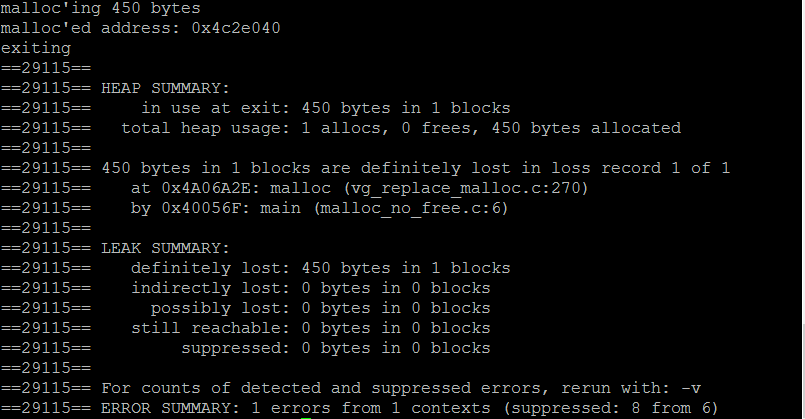
Running the program which allocates memory without freeing and then running gdb on it.



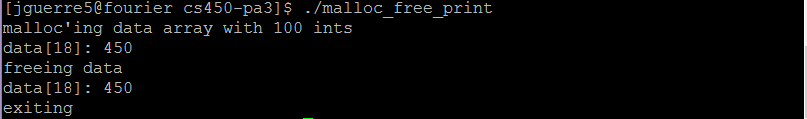
Setting breakpoints in gdb based on line number then running the program through gdb. The local variables are checked using “info locals” at each break point.



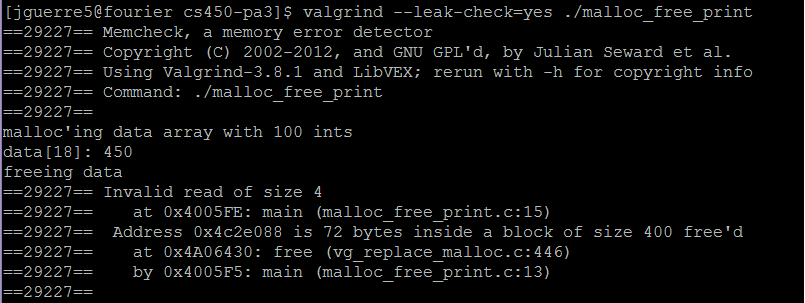
Running Valgrind on the program to check for memory leaks

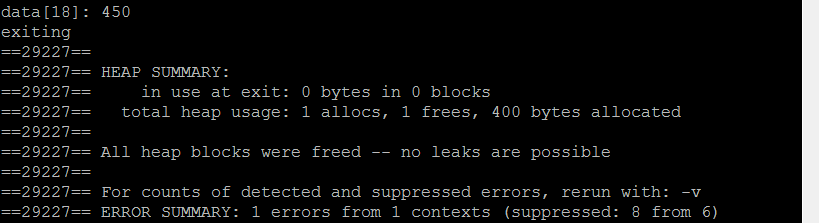


Valgrind showing the memory leak in the program.

1. 

Running the program which allocates and frees an array.



 Running Valgrind on the program, and the beginning of its output where Valgrind detects an invalid read.

The rest of Valgrind’s output where it reports one error in 1 context.