Detecting Memory Errors Using Address Sanitizer

October 2022

Rev. October 2023

Rudranshu Datta

This guide will take you through running your code on the ECE Linux server and explain how to check for memory errors.

Table of Contents

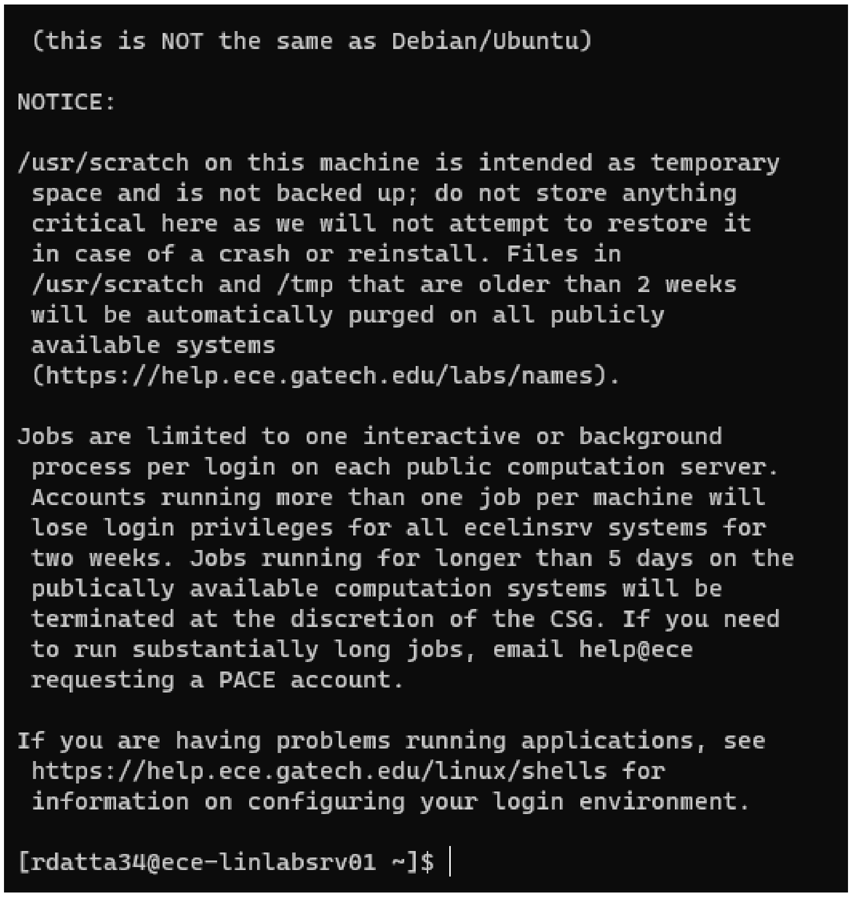
Set-Up

As part of your project requirements, you are required to resolve memory errors, such as memory leaks and referencing something that has previously been freed (dangling pointer). You can check for memory errors by using an open-source tool called Address Sanitizer.

This tool is available for you to run on the ECE Linux servers. These servers are pre-configured with most of the tools you will need for all of your classes at Georgia Tech.

For this class, we will be using the remote linux server linlabsrv01. This guide was written using a Windows machine running WSL2 with Ubuntu and remotely accessing this ECE Linux server.

Follow the steps you have been following so far in the class to get the entire directory onto linlabrv01.



```
 (this is NOT the same as Debian/Ubuntu)

NOTICE:

/usr/scratch on this machine is intended as temporary
 space and is not backed up; do not store anything
 critical here as we will not attempt to restore it
 in case of a crash or reinstall. Files in
 /usr/scratch and /tmp that are older than 2 weeks
 will be automatically purged on all publicly
 available systems
 (https://help.ece.gatech.edu/labs/names).

Jobs are limited to one interactive or background
 process per login on each public computation server.
 Accounts running more than one job per machine will
 lose login privileges for all ecelinsrv systems for
 two weeks. Jobs running for longer than 5 days on the
 publically available computation systems will be
 terminated at the discretion of the CSG. If you need
 to run substantially long jobs, email help@ece
 requesting a PACE account.

If you are having problems running applications, see
 https://help.ece.gatech.edu/linux/shells for
 information on configuring your login environment.

[rdatta34@ece-linlabsrv01 ~]$
```

Running your project with leak checker

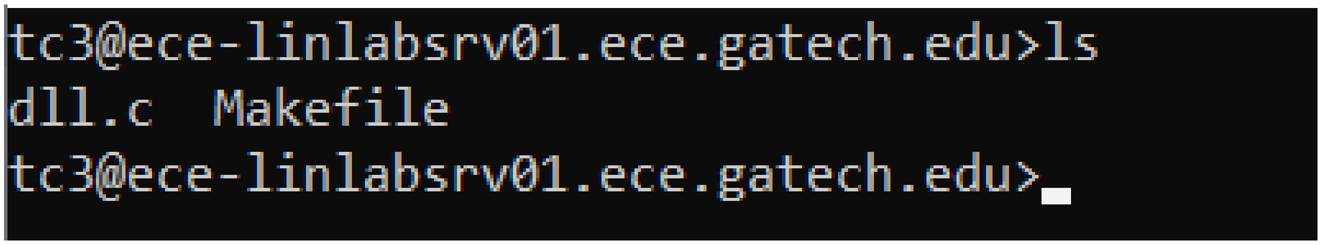
For your project we will be utilizing Address Sanitizer. It is a great built in tool in gcc for checking for memory and stack/heap errors. Address sanitizer libraries should already be preinstalled in gcc.

Running your code on the Linux server

Once you have moved your project folder over using your preferred method, we can continue with the rest of this process.

Step 1: Navigate to the appropriate directory (where your .c and Makefile are).

Step 2: You can use ‘ls’ to list files and cd to navigate. For me the file path looks like this:



```
tc3@ece-linlabsrv01.ece.gatech.edu>ls
dll.c   Makefile
tc3@ece-linlabsrv01.ece.gatech.edu>
```

Your path may be different, depending on your file structure.

Step 3: Do a ‘make clean’. This will remove all the extra files that you may had and all the object files. It is a good practice to do ‘make clean’ after you change something before you run make.

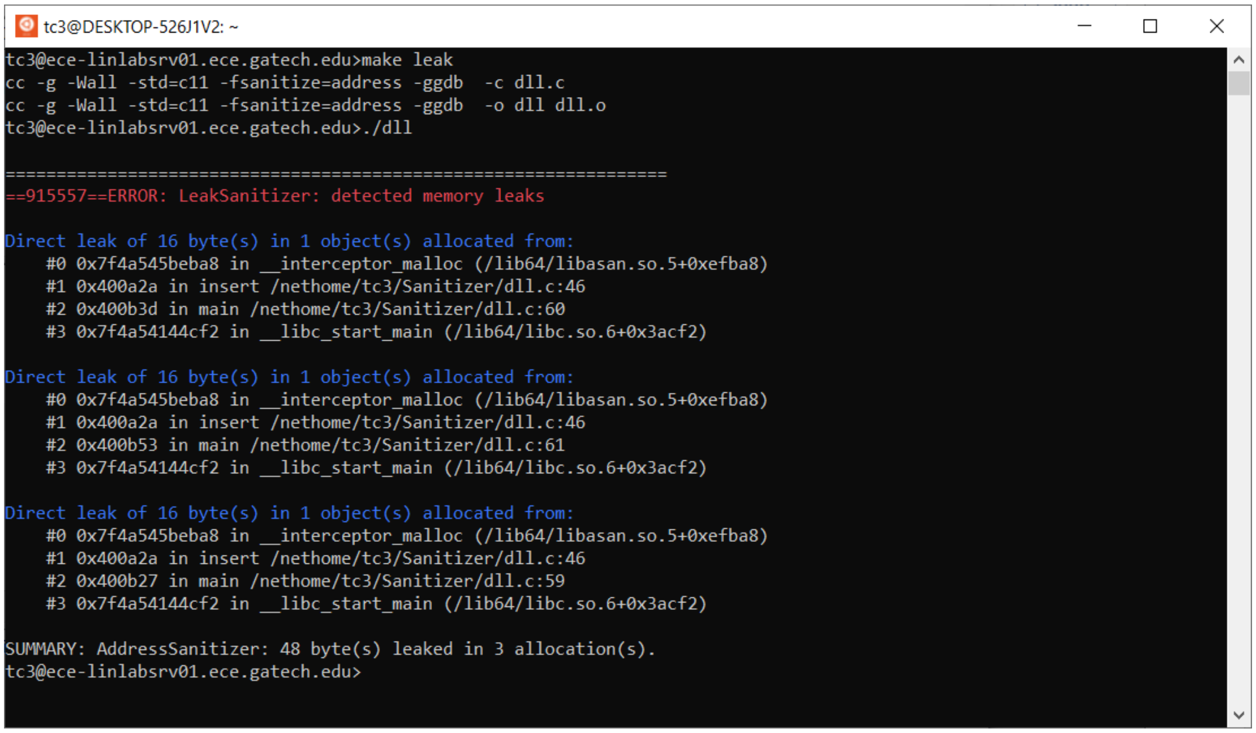
Step 4: Once make clean is done. We can test our code.

First, run ‘make’. This code has a bug (as seen in the relevant part of the lecture), but it causes no compiler errors, and you can even run it successfully:

>> ./dll

We are also using the Google Test framework to detect errors such as the one in this code, and it would always be good practice to fix all tests before you do a memory check. But this code does not have any tests built in, so we will go straight to a memory leak test using Address Sanitizer.

To do so, first do a ‘make clean’, and then do a ‘make leak’, and finally run the new executable program by entering ‘./dll’. This enables the address sanitizer and performs a memory check as the program executes.



```
tc3@ece-linlabsrv01.ece.gatech.edu>make leak
cc -g -Wall -std=c11 -fsanitize=address -ggdb  -c dll.c
cc -g -Wall -std=c11 -fsanitize=address -ggdb  -o dll dll.o
tc3@ece-linlabsrv01.ece.gatech.edu>./dll

=================================================================
==915557==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 16 byte(s) in 1 object(s) allocated from:
    #0 0x7f4a545beba8 in __interceptor_malloc (/lib64/libasan.so.5+0xefba8)
    #1 0x400a2a in insert /nethome/tc3/Sanitizer/dll.c:46
    #2 0x400b3d in main /nethome/tc3/Sanitizer/dll.c:60
    #3 0x7f4a54144cf2 in __libc_start_main (/lib64/libc.so.6+0x3acf2)

Direct leak of 16 byte(s) in 1 object(s) allocated from:
    #0 0x7f4a545beba8 in __interceptor_malloc (/lib64/libasan.so.5+0xefba8)
    #1 0x400a2a in insert /nethome/tc3/Sanitizer/dll.c:46
    #2 0x400b53 in main /nethome/tc3/Sanitizer/dll.c:61
    #3 0x7f4a54144cf2 in __libc_start_main (/lib64/libc.so.6+0x3acf2)

Direct leak of 16 byte(s) in 1 object(s) allocated from:
    #0 0x7f4a545beba8 in __interceptor_malloc (/lib64/libasan.so.5+0xefba8)
    #1 0x400a2a in insert /nethome/tc3/Sanitizer/dll.c:46
    #2 0x400b27 in main /nethome/tc3/Sanitizer/dll.c:59
    #3 0x7f4a54144cf2 in __libc_start_main (/lib64/libc.so.6+0x3acf2)

SUMMARY: AddressSanitizer: 48 byte(s) leaked in 3 allocation(s).
tc3@ece-linlabsrv01.ece.gatech.edu>
```

You can see that three errors are detected, related to each insertion of a node, and resulting from the error discussed in the lecture.

Step 5: Resolve the memory leak

This step is the toughest bit. To find the cause behind your memory leak, you have to go through the stack trace. This becomes more complicated with larger programs, especially with the Gtest framework adding more information to the stack traces. But here, it is relatively simple.

In the stack trace, the last called function is at the top of the log. Reading the stack trace, we know that the leak was triggered by line 46 in dll.cc, where malloc was called by the insert function. That was where memory was allocated and never freed.

Notes

1. Alternatively, you can run tests on your local machine as well. If you are using WSL2, it should work the same way as running it on the linux server. On MacOS, some changes may be required before you can run leak checker on your local machine. Everything else should still work on your local machines on all platforms if you do not run address sanitizer using the ‘make leak’ command. Note: MacOS does not support the full AddressSanitizer library so you should always check your code on the linux server before submission.
2. I recommend that you test on your local machine first before uploading to the linux servers if possible. This will save you a lot of time and frustration.
3. Going through stack traces can be difficult and the line numbers may not always be totally accurate but they put you in the ballpark to start debugging your code.