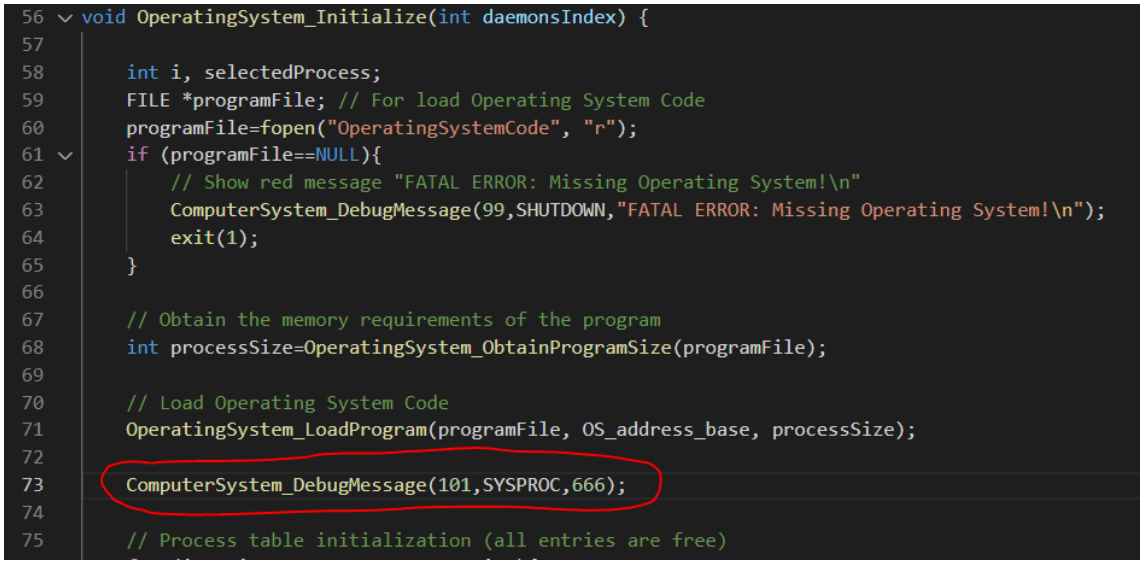
Finding the source code line producing a “Segmentation fault” problem

I have deliberately entered a bug in my code. Here it is the function containing the bug:

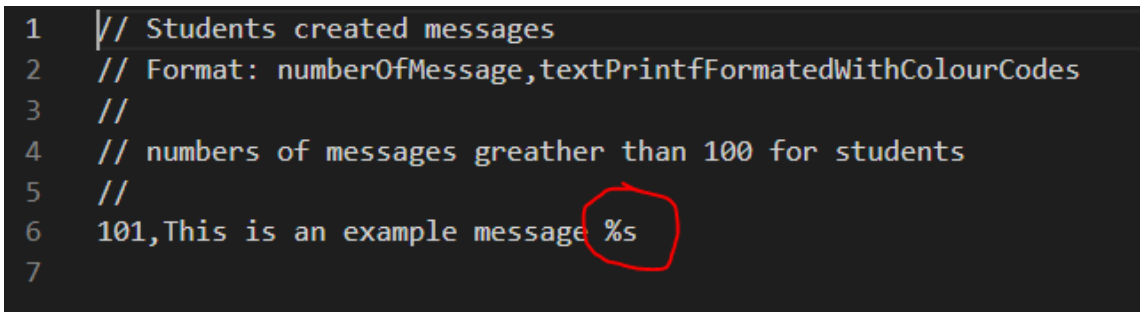
- The `ComputerSystem_DebugMessage()` invocation tries to print 666



```
56  void OperatingSystem_Initialize(int daemonsIndex) {
57
58      int i, selectedProcess;
59      FILE *programFile; // For load Operating System Code
60      programFile=fopen("OperatingSystemCode", "r");
61      if (programFile==NULL){
62          // Show red message "FATAL ERROR: Missing Operating System!\n"
63          ComputerSystem_DebugMessage(99,SHUTDOWN,"FATAL ERROR: Missing Operating System!\n");
64          exit(1);
65      }
66
67      // Obtain the memory requirements of the program
68      int processSize=OperatingSystem_ObtainProgramSize(programFile);
69
70      // Load Operating System Code
71      OperatingSystem_LoadProgram(programFile, OS_address_base, processSize);
72
73      ComputerSystem_DebugMessage(101,SYSPROC,666);
74
75      // Process table initialization (all entries are free)
```

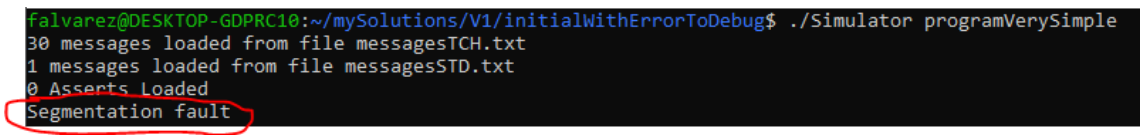
And here it is my `messagesSTD.txt` file:

- Message 101 expects a string, not an integer



```
1  // Students created messages
2  // Format: numberOfMessage,textPrintfFormatedWithColourCodes
3  //
4  // numbers of messages greather than 100 for students
5  //
6  101,This is an example message %s
7
```

So, when I run the simulator, I get the “Segmentation fault” error message and the simulator execution is aborted:

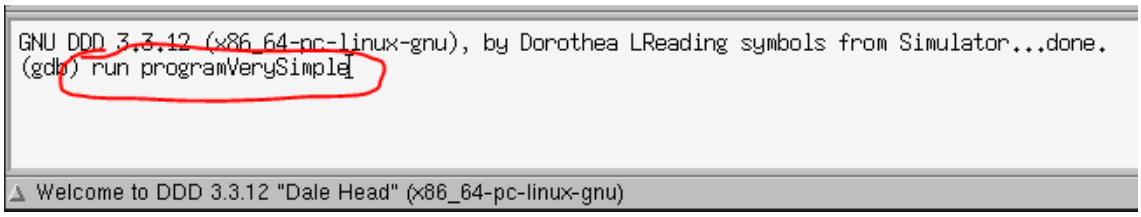


```
falvarez@DESKTOP-GDPRC10:~/mySolutions/V1/initialWithErrorToDebug$ ./Simulator programVerySimple
30 messages loaded from file messagesTCH.txt
1 messages loaded from file messagesSTD.txt
0 Assents Loaded
Segmentation fault
```

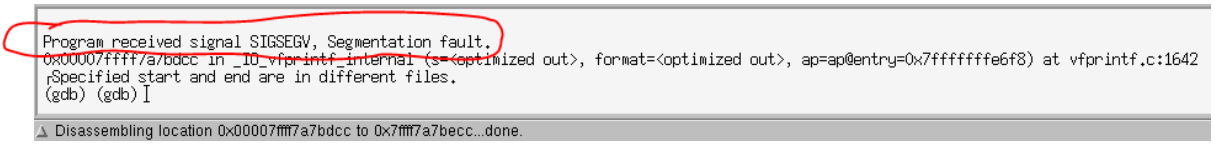
To find the source code line responsible of the problem, we must execute the simulator from inside the debugger.

Using “ddd”

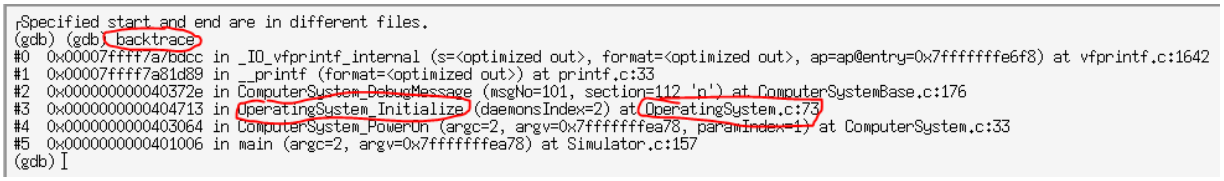
- Do not set any breakpoint
- Directly enter the command to run the simulator



- The debugger tells us that a “Segmentation fault” problem has arisen. The simulator execution has been aborted.

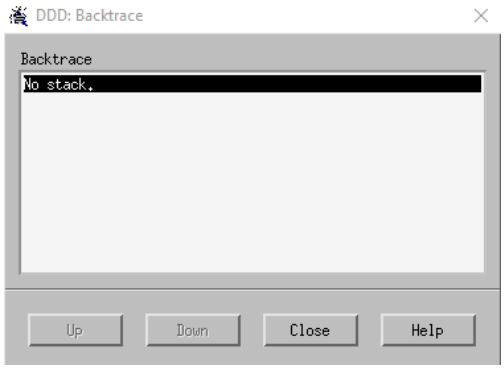


- Run the “backtrace” command. It will show you the function invocation stack (similar to what Java does when a `NullPointerException` is raised):

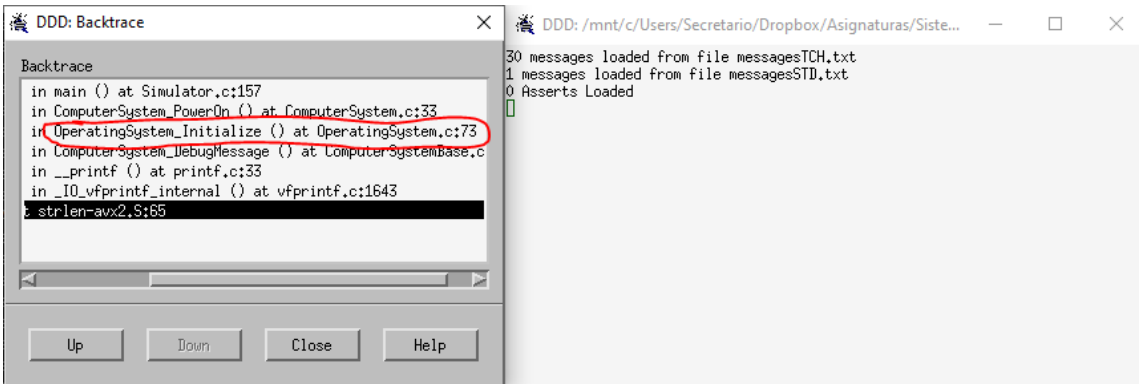


- In the stack, try to find the line closest to the top corresponding to a function inside one of the files you usually modify. There it is the problem:
    - Line 73 of `OperatingSystem.c`, inside `OperatingSystem_Initialize()`.

- Alternatively, the function invocation stack could be always visible, by using the “`Status > Backtrace`” menu option. For example, if you show it before you start the simulator execution:



- Or, if you show it after the “segmentation fault” problem appears:



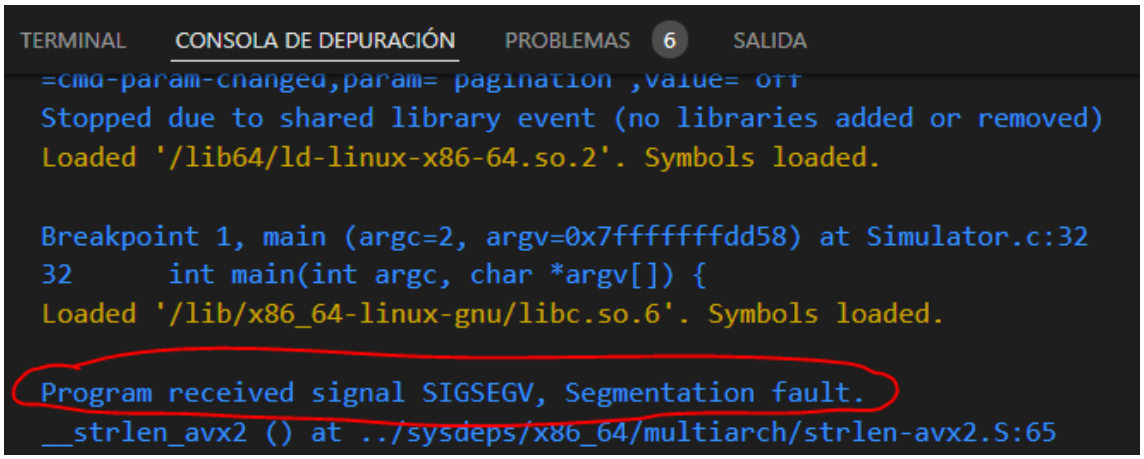
After you have found the code line presenting the problem, **it is time to determine the nature of the bug**. This is done as in any other programming language (debugging skills are always important).

Using "VSCode"

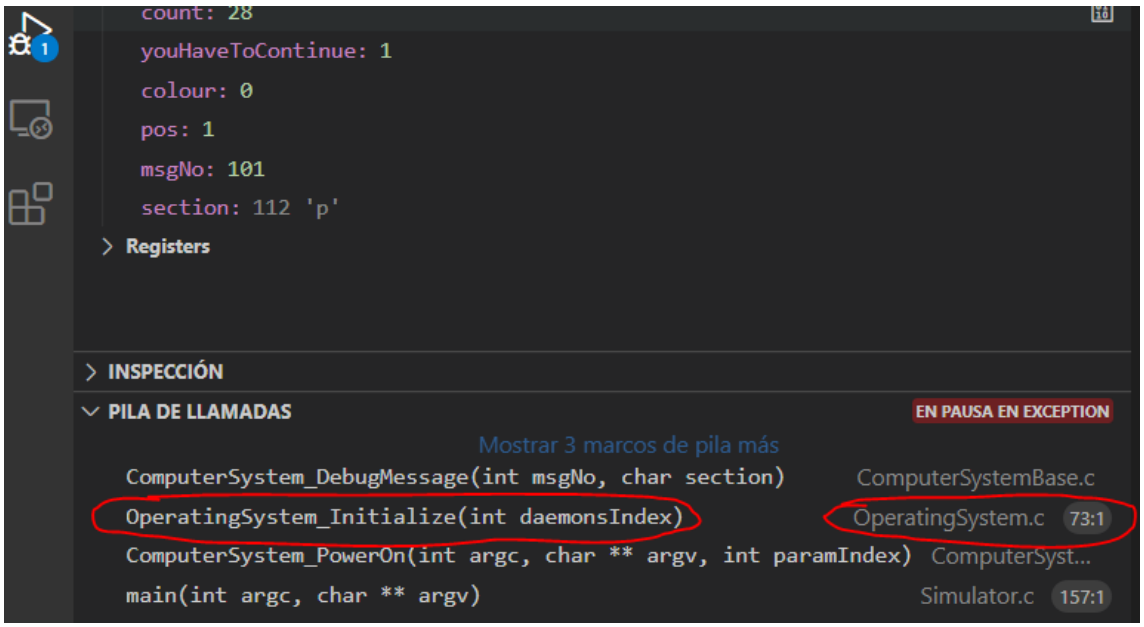
- Before anything else, setup your VSCode to debug the simulator as it is explained in a document in the eCampus.
- Prepare the "`.vscode/launch.json`" file to execute the simulator in the desired conditions:

```
5     "version": "0.2.0",
6     "configurations": [
7         {
8             "name": "Launch Simulator",
9             "type": "cppdbg",
10            "request": "launch",
11            "program": "${workspaceFolder}/Simulator",
12            "args": ["programVerySimple"],
13            "stopAtEntry": false,
14            "cwd": "${workspaceFolder}",
15            "environment": [],
```

- Start debugging (F5). The debugging console shows the "Segmentation fault" problem:



- You can also see the function invocation stack:



After you have found the code line presenting the problem, it is time to determine the nature of the bug. This is done as in any other programming language (debugging skills).