EE2003 Debugging Session

Himanshu Rajnish Borkar (EE22B070) and Deeptansh Nagrale (EE22B067)

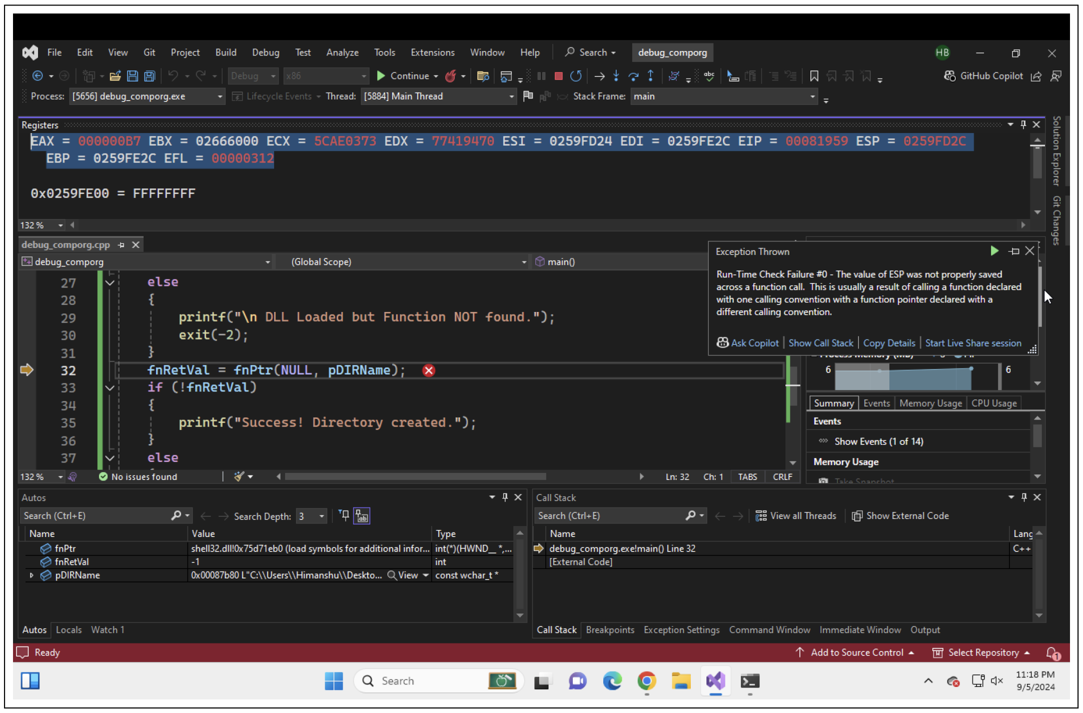
4th September 2024

1 Problem Statement

For the given code, we are supposed to identify the erroneous line/part, and debug accordingly.

2 Debugging on MSVC

First, we load the code on a fresh `.cpp` file, build the solution, add the breakpoint at `int main()` and start the debugging process. Upon stepping over the breakpoints we observe:



```
Registers
EAX = 000000B7 EBX = 02666000 ECX = 5CAE0373 EDX = 77419470 ESI = 0259FD24 EDI = 0259FE2C EIP = 00081959 ESP = 0259FD2C
  EBP = 0259FE2C EFL = 00000312

0x0259FE00 = FFFFFFFF
```

```
27      else
28      {
29          printf("\n DLL Loaded but Function NOT found.");
30          exit(-2);
31      }
32      fnRetVal = fnPtr(NULL, pDIRName);
33      if (!fnRetVal)
34      {
35          printf("Success! Directory created.");
36      }
37      else
```

```
Exception Thrown
Run-Time Check Failure #0 - The value of ESP was not properly saved across a function call.  This is usually a result of calling a function declared with one calling convention with a function pointer declared with a different calling convention.
```

Autos

Name	Value	Type
fnPtr	shell32.dll!0x75d71eb0 (load symbols for additional infor...	int(*)(HWND__ *,...
fnRetVal	-1	int
pDIRName	0x00087b80 L"C:\\Users\\Himanshu\\Deskto...	const wchar_t *

Call Stack

Name	Lang
debug_comporg.exe!main() Line 32	C++
[External Code]	

Figure 1: Error thrown by MSVC

2.1 Register picture on the lines 29, 30 and 32:

Line 29:

```
EAX = 00000020 EBX = 02666000 ECX = 0259FB18 EDX = 00000000 ESI = 0259FD24
EDI = 0259FE2C EIP = 00081923 ESP = 0259FD24 EBP = 0259FE2C EFL = 00000206
```

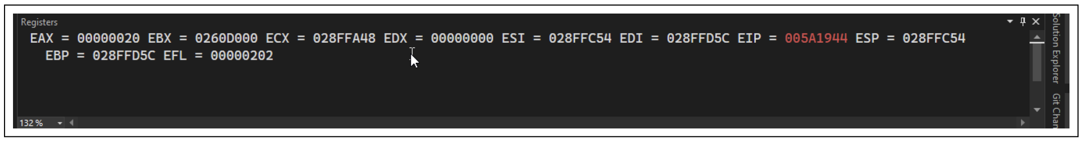
Line 30:

```
EAX = 00000020 EBX = 02666000 ECX = 0259FB18 EDX = 00000000 ESI = 0259FD24
EDI = 0259FE2C EIP = 00081944 ESP = 0259FD24 EBP = 0259FE2C EFL = 00000206
```

Line 32:

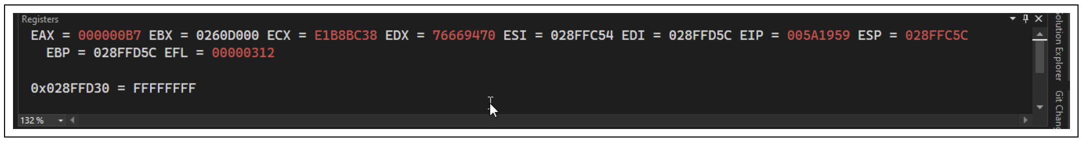
```
EAX = 000000B7 EBX = 02666000 ECX = 5CAE0373 EDX = 77419470 ESI = 0259FD24
EDI = 0259FE2C EIP = 00081959 ESP = 0259FD2C EBP = 0259FE2C EFL = 00000312
```

On another iteration:



```
EAX = 00000020 EBX = 0260D000 ECX = 028FFA48 EDX = 00000000 ESI = 028FFC54 EDI = 028FFD5C EIP = 005A1944 ESP = 028FFC54
  EBP = 028FFD5C EFL = 00000202
```

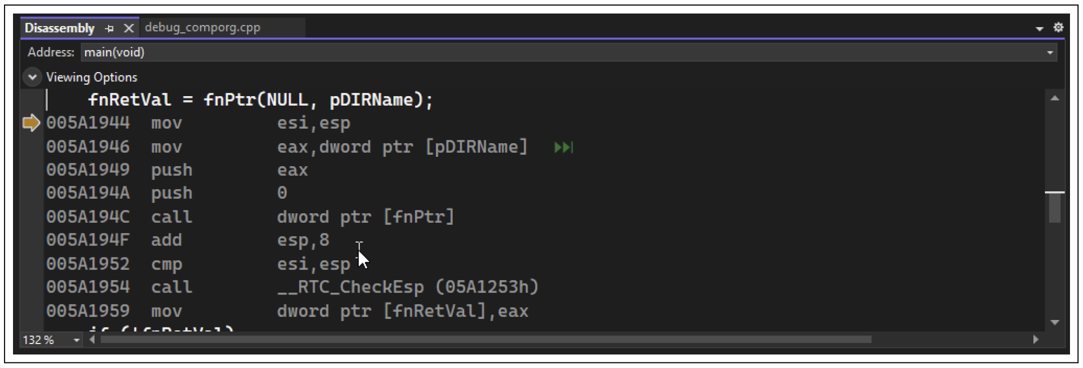
Figure 2: Line 30



```
EAX = 000000B7 EBX = 0260D000 ECX = E1B8BC38 EDX = 76669470 ESI = 028FFC54 EDI = 028FFD5C EIP = 005A1959 ESP = 028FFC5C
  EBP = 028FFD5C EFL = 00000312

0x028FFD30 = FFFFFFFF
```

Figure 3: Line 32

The ESP has an offset of exactly 8 bytes from the original ESP in both the cases, this is because:



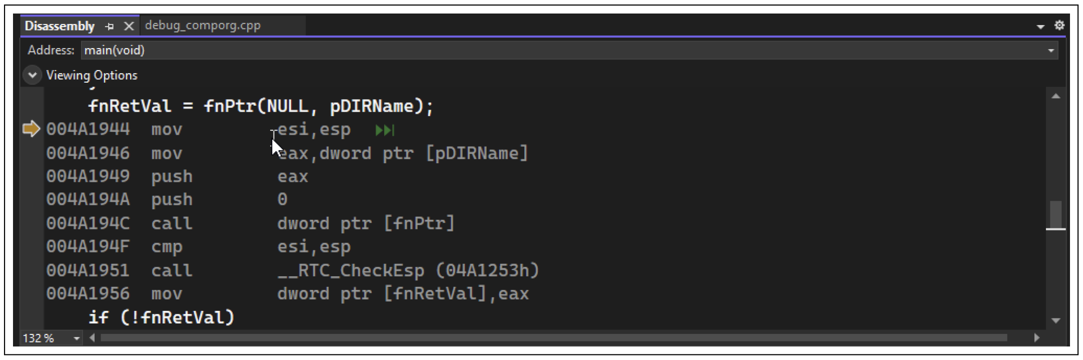
```
    fnRetVal = fnPtr(NULL, pDIRName);
005A1944  mov         esi,esp
005A1946  mov         eax,dword ptr [pDIRName]
005A1949  push        eax
005A194A  push        0
005A194C  call        dword ptr [fnPtr]
005A194F  add         esp,8
005A1952  cmp         esi,esp
005A1954  call        __RTC_CheckEsp (05A1253h)
005A1959  mov         dword ptr [fnRetVal],eax
```

Figure 4: The `ADD ESP, 8` causes the stack pointer to move by 8 bytes from the original ESP

Clearly, the stack is not being cleaned up upon calling the function `fnPtr`.

3 Diagnosis

The reason for the uncleaned Stack at the end of `fnPtr` is that the calling convention used is different from the Declaration made at the top of the program. In this case, it is `__cdecl`.



```
    fnRetVal = fnPtr(NULL, pDIRName);
004A1944  mov          esi,esp
004A1946  mov          eax,dword ptr [pDIRName]
004A1949  push         eax
004A194A  push         0
004A194C  call         dword ptr [fnPtr]
004A194F  cmp          esi,esp
004A1951  call         __RTC_CheckEsp (04A1253h)
004A1956  mov          dword ptr [fnRetVal],eax
    if (!fnRetVal)
```

Figure 5: What the assembly code *should* look like

4 Debugging

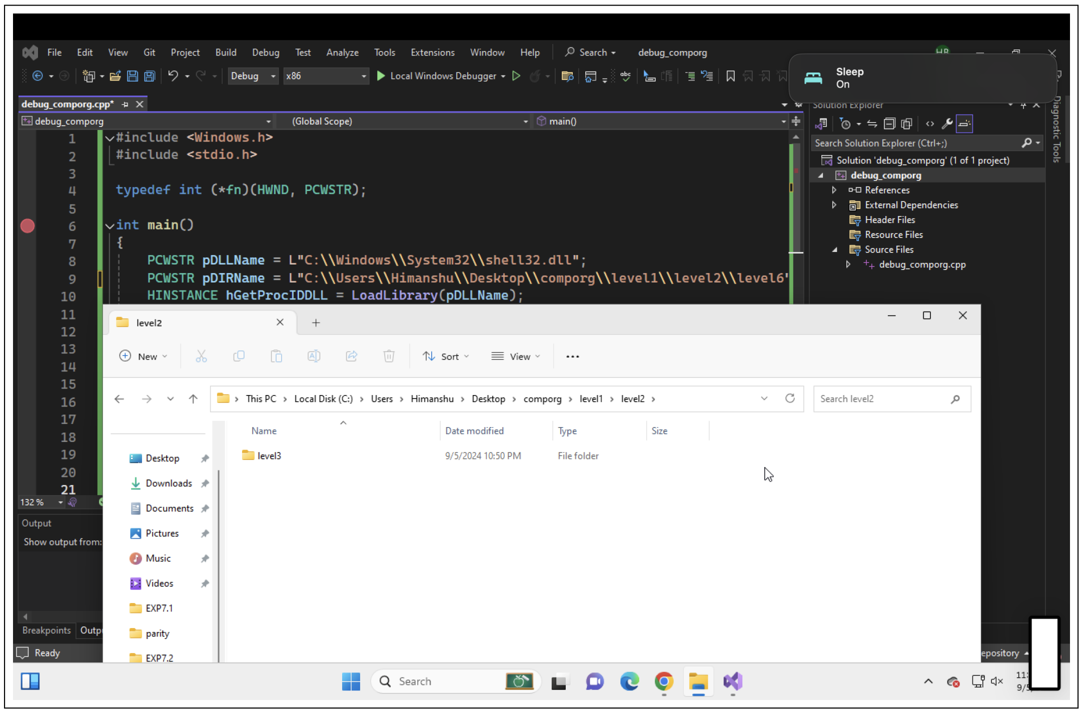
To debug this issue, we simply modify the declaration to enforce the `__stdcall` so that for all the referencing/dereferencing for the given declaration occurs with the same calling convention, i.e., `__stdcall`.

5 Modified Code:

```c
#include <Windows.h>
#include <stdio.h>

typedef int (__stdcall *fn)(HWND, PCWSTR);

int main()
{
    PCWSTR pDLLName = L"C:\\Windows\\System32\\shell32.dll";
    PCWSTR pDIRName = L"C:\\Users\\Himanshu\\Desktop\\comporg\\level1
        \\level2\\level6";
    HINSTANCE hGetProcIDDLL = LoadLibrary(pDLLName);
    int fnRetVal = -1;
    if (hGetProcIDDLL)
    {
        printf("\n DLL Loaded.");
    }
    else
    {
        printf("\n DLL NOT Loaded.");
        exit(-1);
    }

    fn fnPtr = (fn)GetProcAddress(hGetProcIDDLL, "SHCreateDirectory");
    if (fnPtr)
    {
        printf("\n DLL Loaded and Function found.");
    }
    else
    {
        printf("\n DLL Loaded but Function NOT found.");
        exit(-2);
    }
    fnRetVal = fnPtr(NULL, pDIRName);
    if (!fnRetVal)
    {
        printf("Success! Directory created.");
    }
    else
    {
        printf("Failure! Directory not created");
    }
    return 0;
}
```

6 Inference

Even though there is an error in the code, we see an interesting output when we run the original code:

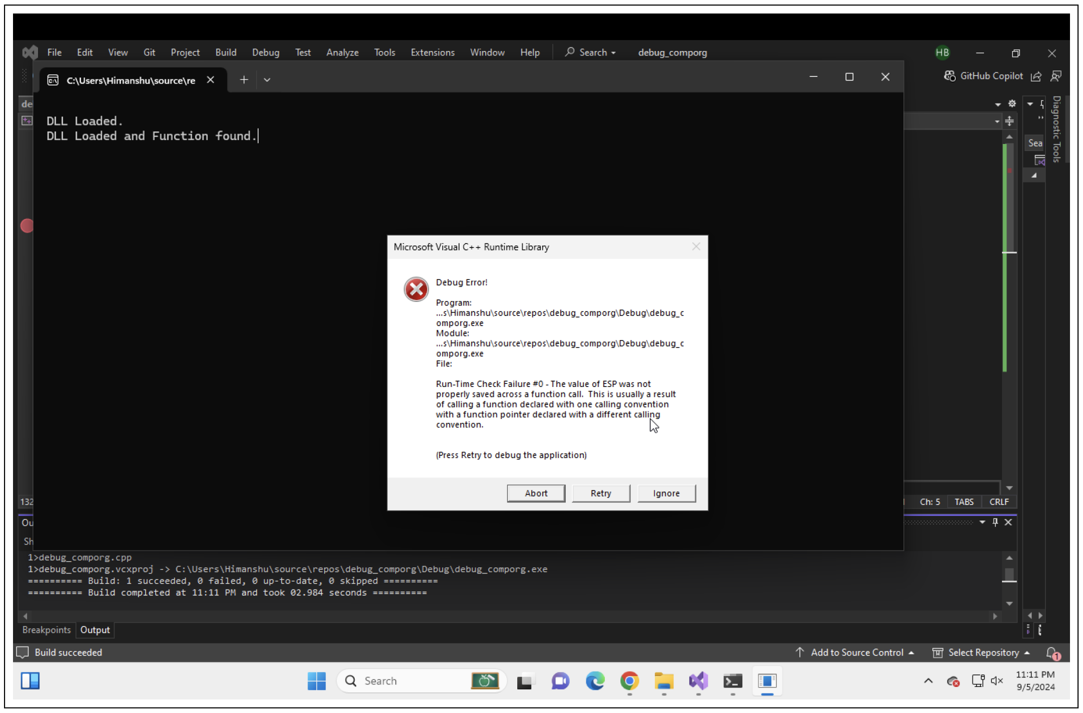


```
#include <Windows.h>
#include <stdio.h>

typedef int (*fn)(HWND, PCWSTR);

int main()
{
    PCWSTR pDLLName = L"C:\\Windows\\System32\\shell32.dll";
    PCWSTR pDIRName = L"C:\\Users\\Himanshu\\Desktop\\comporg\\level1\\level2\\level6'
    HINSTANCE hGetProcIDDLL = LoadLibrary(pDLLName);
```

This PC › Local Disk (C:) › Users › Himanshu › Desktop › comporg › level1 › level2 ›

Name	Date modified	Type	Size
level3	9/5/2024 10:50 PM	File folder	

Figure 6: Level2 only has level3, we want to add level6



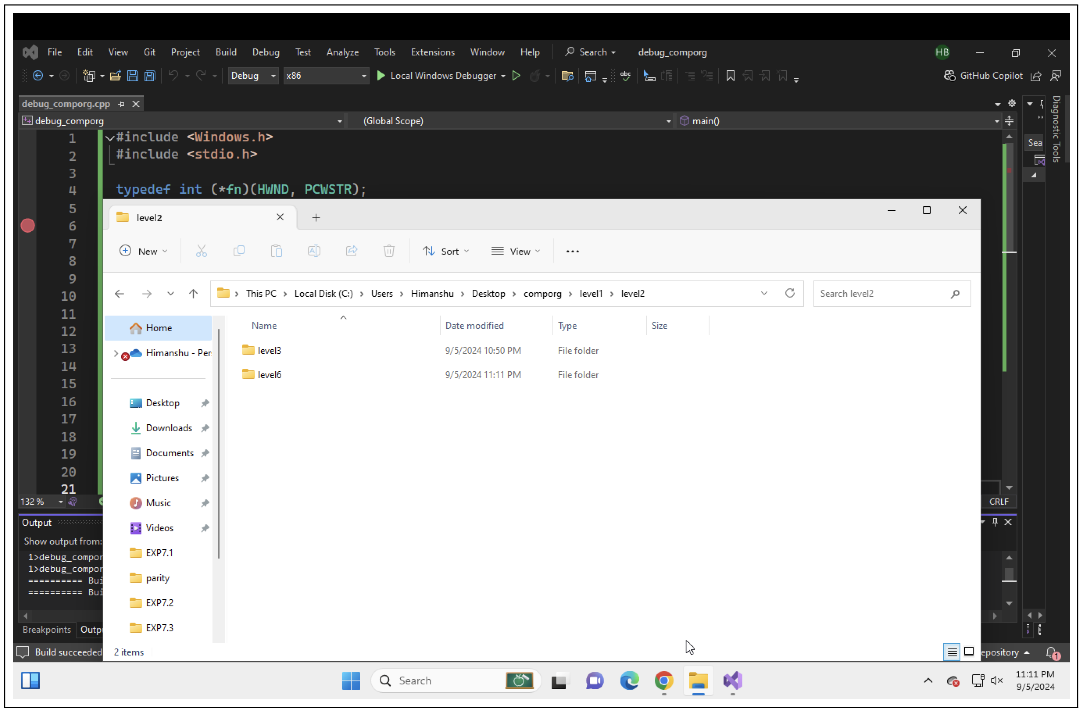
```
DLL Loaded.
DLL Loaded and Function found.
```

Microsoft Visual C++ Runtime Library

```
Debug Error!

Program:
...s\Himanshu\source\repos\debug_comporg\Debug\debug_c
omporg.exe
Module:
...s\Himanshu\source\repos\debug_comporg\Debug\debug_c
omporg.exe
File:

Run-Time Check Failure #0 - The value of ESP was not
properly saved across a function call.  This is usually a result
of calling a function declared with one calling convention
with a function pointer declared with a different calling
convention.


(Press Retry to debug the application)
```

```
1>debug_comporg.cpp
1>debug_comporg.vcxproj -> C:\Users\Himanshu\source\repos\debug_comporg\Debug\debug_comporg.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
========== Build completed at 11:11 PM and took 02.984 seconds ==========
```

Figure 7: Error thrown by compiler



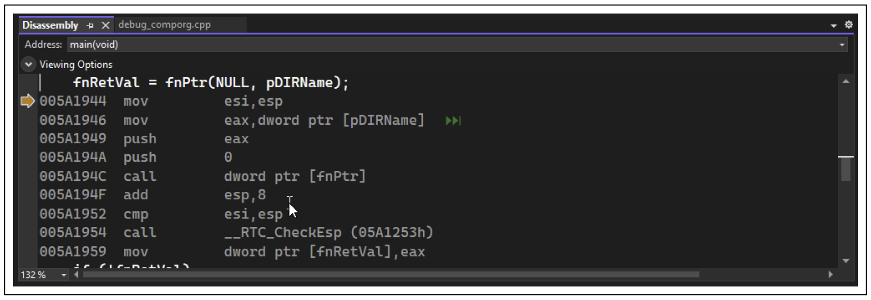
```
#include <Windows.h>
#include <stdio.h>

typedef int (*fn)(HWND, PCWSTR);
```

This PC › Local Disk (C:) › Users › Himanshu › Desktop › comporg › level1 › level2

Name	Date modified	Type	Size
level3	9/5/2024 10:50 PM	File folder	
level6	9/5/2024 11:11 PM	File folder	

Figure 8: The directory is still created!

The reason why the file is created lies in the assembly code for the function and the vanilla epilogue of the code.



```
    fnRetVal = fnPtr(NULL, pDIRName);
005A1944  mov         esi,esp
005A1946  mov         eax,dword ptr [pDIRName]
005A1949  push        eax
005A194A  push        0
005A194C  call        dword ptr [fnPtr]
005A194F  add         esp,8
005A1952  cmp         esi,esp
005A1954  call        __RTC_CheckEsp (05A1253h)
005A1959  mov         dword ptr [fnRetVal],eax
```

Figure 9: ADD ESP, 8 is present implying the `__cdecl` function

But the lines above the `ADD ESP,8` line do the job of creating the directory, this implies that the directory is created **before** the abortion message is thrown for the erroneous stack cleanup.

7 Conclusion

In this debugging session, we successfully identified and resolved the issue related to the stack cleanup by modifying the calling convention. The changes made to the function declaration ensured that the code was executed correctly. This exercise highlighted the importance of proper calling conventions and their effect on system-level programming.

Thank you.