Building User Mode Debugger for Windows

Lab + Home

Objectives

- Write debugger code
- Attach debugged process to debugger
- Get CPU register state
- Handle debug events
- Use breakpoints

Table of Contents

1.

Background Reading

Textbook:

Grey Hat Python, chapter 3

Other resources:
https://docs.python.org/3/library/pdb.html
http://www.gnu.org/software/gdb/documentation/
http://sourceware.org/gdb/current/onlinedocs/gdb.pdf.gz
https://docs.python.org/2/extending/extending.html

Notes common to all lab and home assignment problems

For every lab and home assignment, all work should go into your personal repository, subdirectory named mXX, where XX stands for the module number. For each problem, carefully name the program as described. The programs are extracted from your repository by a Python script, and errors in the program name will result in the instructor never seeing your program, and your mark for it will be ZERO!

There are always many ways how to solve a programming problem, and usually one or two ways which are fast, compact and elegant.

Make sure to push your work to the server often, and have pushed the working version of the program by the deadline specified. The script extracting your programs from your repository will be run at any time after the deadline.

Lab specific intro

In this lab we will create several debugging scenarios. First we write debugger which will start the process to be debugged, then we will expore ways to attach a debugger to process already running. We will get register state, handle debug events, set breakpoints and inspect register state at breakpoints.

We will build a simple debugger in this class. Each successive problem builds up on the code from the previous problem and adds additional functionality.

Problem 1

Create the file **`my_debugger_defines.py`**, and **`m06runcalc.py`**. This is a simple class which will load an executable and then exit.

Write simple test harness, **`m06runcalc_test.py`**, which will test the functionality of the debugger class.

Problem 2

Create the file **m06attach.py**, and **m06attach_test.py**.

The **m06attach.py** will include steps to attach to process specified by PID, wait for debug event, then wait for a keystroke, and finally detach and exit.

Problem 3

Create the file **m06cpureg.py**, and associated test harness **m06cpureg_test.py** from the files used in previous problem. The program will prompt for a process PID to attach to, and once the PID is entered from the keyboard, the program will dump registers for each thread of the process. The output should look similar to the following example:

```
Process PID: 1234
Dumping regs for thread ID: 0x00000c7c
EIP: 0x772670b4
ESP: 0x001aef20
EBP: 0x001aef3c
EAX: 0x00000001
EBX: 0x00000000
ECX: 0x00375ab0
EDX: 0x00000030
```

Problem 4

Create the file **m06events.py** and associated test harness **m06events_test.py**, again from the files used in the previous problem. The program will prompt for PID, then show all debugging events as they arrive. The program will gracefully detach from the process and exit when the letter Q (case insensitive) is hit on the keyboard. Utilize module **msvcrt** to get the keystrokes from the keyboard.

Problem 5

Write short python script called **printfloop.py**, which will print a line of text every 2 seconds, using the **msvcrt** dll library function **printf()**. The script will first print its own PID.

Create the file **m06softbp.py** and associated test harness **m06softbp_test.py**, again from the files used in the previous problem. The program will prompt for PID, then show all debugging events as they arrive. The program will gracefully detach from the process and exit when the letter Q (case insensitive) is hit on the keyboard. Utilize module **msvcrt** to get the keystrokes from the keyboard. Run the program, and use the PID of the **printfloop.py** script. The test harness will set soft breakpoint at the prinf() address. The debugger will report the breakpoint, then reset the breakpoint and continue.

Problem 6

Repeat problem 5, but create file **m06hardbp.py** and associated test harness

`m06hardbp_test.py`. The functionality is the same, but the breakpoint will be hard, i.e. use processor breakpoint registers.

Problem 7

Create the file **`m06membreak.py`** and associated test harness **`m06membreak_test.py`**, again from the files used in the previous problem. The program will prompt for PID, then show all debugging events as they arrive. The program will gracefully detach from the process and exit when the letter Q (case insensitive) is hit on the keyboard. Utilize module **`msvcrt`** to get the keystrokes from the keyboard.