++it Italian C++ Community

Time Travel Debugging

Paolo Severini

Italian C++ Conference 2018

June 23, Milan

Thanks to the sponsors!

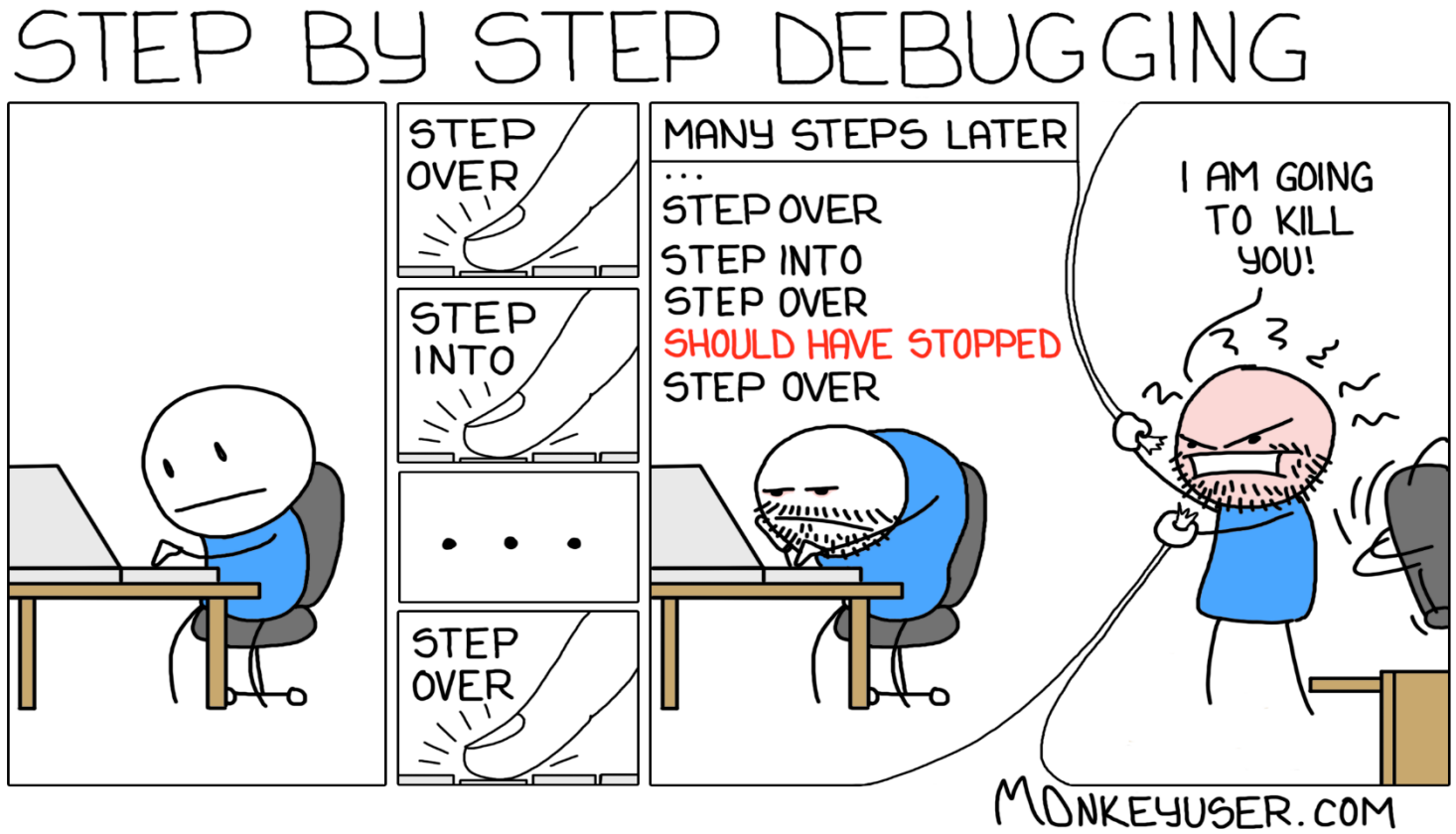
Debugging is Hard

"Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?" [Brian Kernighan]

Debugging may be difficult and time consuming:

- Bugs can be difficult to reproduce
  - Ex: memory corruptions, race conditions, intermittent bugs, …
- It may be difficult to determine the cause of a bug from log files or crash dumps
  - Ex: memory leaks, buffer overruns, …
- Debugging stepping through code may require multiple restarts

Debugging is Painful



Reverse Debugging

A reverse debugger provides the ability to answer the question: "HOW DID THAT HAPPEN?"

- Records the program execution (with the granularity of single CPU instructions)
- Replays the program execution backwards/forwards to uncover the reason for the failure

There are many reverse-debuggers for native code:

- Linux: rr, gdb, UndoDB
- Windows: **Time Travel Debugging**

and for other languages:

- Java (Chronon), .Net/C# (RevDeBug), Python (RevPDB), Elm (Elm TTD),
- JavaScript (ChakraCore)

Time Travel Debugging

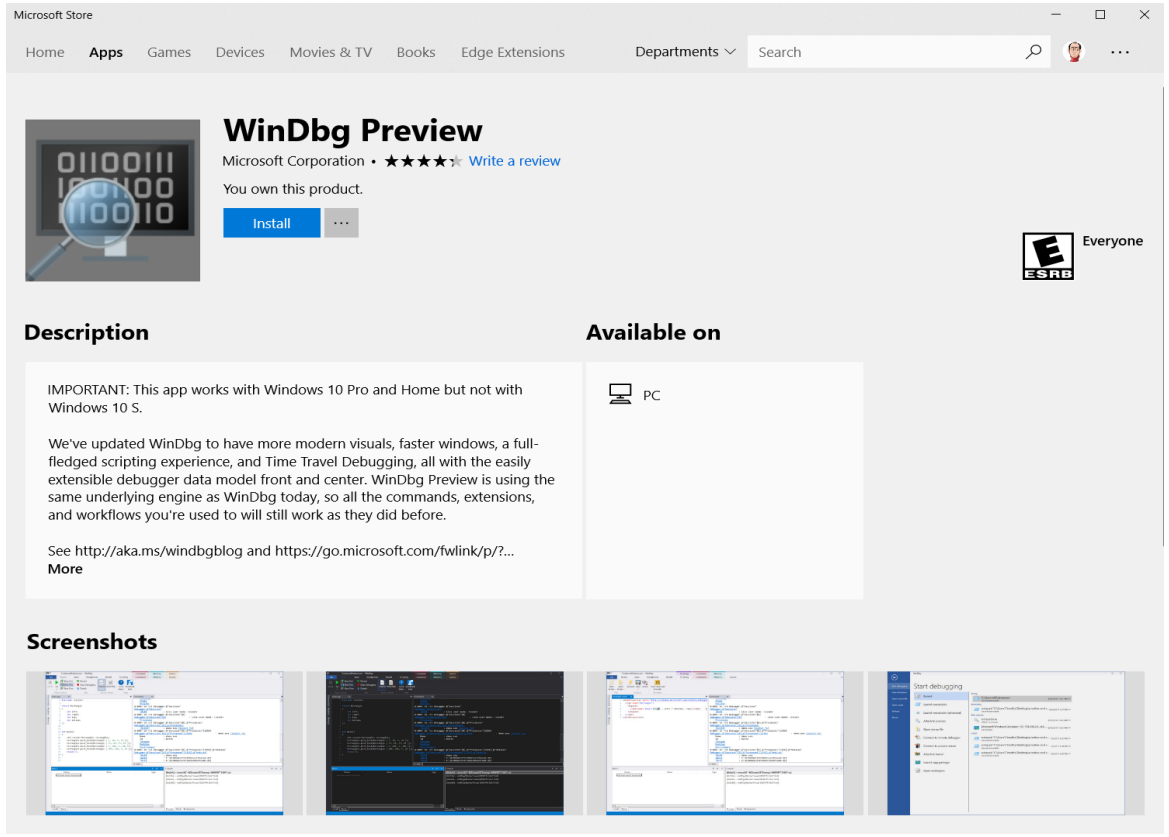
TTD: Microsoft's reverse debugging solution integrated with WinDbg

1. Use WinDbg Preview to **record** a process into a trace file (.run)
   - Works with any user-mode process (does not require code instrumentation)
   - Supports multiple threads on multiple cores
   - Recorder still relatively slow
   - Trace file size: 5 MB/sec to 50 MB/sec
2. Open the trace file in WinDbg Preview and **replay** the code execution backward and forward
   - Supports code/data breakpoints, PDB symbols, …
   - Player: very fast
3. Run queries to **analyze** the trace data

e

WinDbg Preview

- New version of WinDbg available in the Microsoft Store
- Same debugging engine of the “old” WinDbg, with a new user interface
- TTD requires Windows 10 Anniversary Update (RS1)



Demo

Debugger Data Model

Use the TTD data model to **query** time-travel traces to answer questions like:

- What exceptions are in the trace?
- When was a thread created/terminated?
- When was a DLL loaded?
- Which function calls occurred in the trace? What were the arguments and the return values?
- When was memory allocated/freed?

Trace data is queryable with LINQ-like expressions

Example: list the sequence of calls to `GetLastError()` that returned a failure code:

```
dx -g @$cursession.TTD.Calls("kernelbase!GetLastError").Where(x => x.ReturnValue != 0)
```

Resources

To learn more:

- https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/time-travel-debugging-overview
- https://aka.ms/WinDbgBlog

- Old research paper: https://www.usenix.org/legacy/events/vee06/full_papers/p154-bhansali.pdf

Questions?