DrHook: Revitalisation of a Tracing and Performance Profiling Tool

Andrew Beggs, Olivier Marsden, Ioan Hadade

ECMWF: Andrew.Beggs@ecmwf.int, Olivier.Marsden@ecmwf.int, Ioan.Hadade@ecmwf.int

1. Introduction

DrHook is a well established and widely used **library within the IFS** and beyond. It's mainly used by defining **calliper regions** and **tracking performance characteristics** within these regions. Additionally, DrHook "hooks" into the signal handler to **provide richer information than** is available by the **default signal handler**. Recently DrHook has undergone an overhaul in both documenting & refactoring existing features and adding new ones - such as **PAPI & NVTX integrations**, and **memory leak identification**. This poster explores these features and the workflows needed to exploit them efficiently.

2. Documentation Efforts

- Documentation is intended **for both users and developers**
- Currently flags and environment variables are documented
- **Documentation is built by sphinx** and can produce **latex pdf, html, and markdown** build artefacts
  - **All built from a single source** – no synchronisation necessary
  - Github CD pipeline to keep docs up to date
  - CI actions can be used to enforce docs being updated when DrHook is changed

1.3 DR_HOOK_CATCH_SIGNALS

1.3.1 Valid Values

valid_value ::= <signal> | <valid_value> <delim> <signal>
delim ::= ‘,’ | ‘ ‘ | ‘t’ | ‘/’
signal ::= ‘-1’ | <number>
number ::= <digit> | <number> <digit> | <number> ‘0’
digit ::= ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’

1.3.2 Purpose

Specifies a list of signals to be caught and handled by drhook.

1.3.3 Notes

If $1 \leq$ `signal` $\leq$ `NSIG`, then it is registered to be caught and handled by drhook - unless it has been set to ignored by *DR_HOOK_IGNORE_SIGNALS*. `NSIG` is defined in `signal.h` and is system dependant.

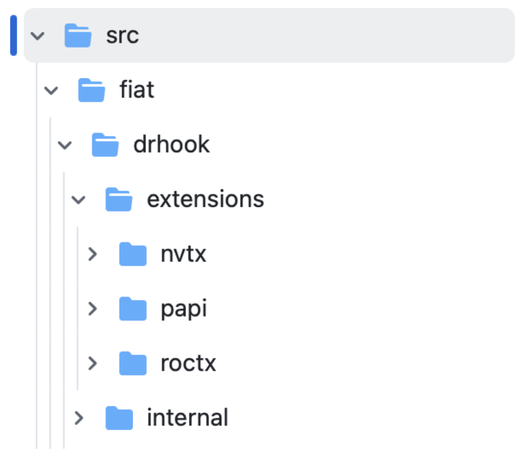
If `signal` is set to `-1`, then all available catchable signals are registered to be caught and handled by drhook. Any value after `-1` will be discarded.

All other values will be silently discarded.

An example entry for documentation on an environment variable

3. NVTX & a new approach to third-party extensions

- DrHook **callipers are already in key locations** in various codebases
- These locations are often chosen because of their impact on performance
- The introduction of Nvidia’s NVTX library provided the opportunity to add a framework for adding third-party extensions
  - Individual folder to contain code & only conditionally compiled
    - Keeps core DrHook code and third-party code separate - **Performance minimally impacted**
  - **Minimal exposure to core DrHook code**, just entry exit functions
    - Prevents the core DrHook code from becoming polluted
- **Contributors** from AMD **have already used this approach** & added their equivalent library, ROC-TX



Extensions are neatly contained in their own folder. Most only need an entry and exit function in the internal DrHook code

4. Memory leak identification

- Traditional tools, such as **valgrind**, can be **too verbose** & find **too many false positives** for the IFS
- DrHook reduces the granularity by grouping readings by region
- **DrHook redefines malloc & free** functions with a shim allocator
  - Internal DrHook mallocs & frees are not intercepted
    - Prevents infinite loops and incorrect tracking
  - Each **allocation is tracked** via its pointer **& associated with the DrHook region** where it was allocated
- Output can be either
  - Table
    - **Lists** DrHook regions in descending order of **number of bytes leaked**
    - Also records the **number of allocations leaked**
    - Helps identifying a large single allocation leak vs many small leaks
    - Good for identifying leaky DrHook regions, **enabling more targeted investigation**
  - Tree
    - Also shows the bytes leaked & number of leaked allocations
    - **Shows the information on the call tree**, so allows for **specific leaky call paths** to be **identified**
    - Includes the memory leaked by the child regions too – helps to quickly identify leaky call paths
    - Not grouped by DrHook region, but **can be filtered to only track** allocations from **“suspect” DrHook regions** identified in Tabel mode
    - Not suitable initially as it can be very large, depending on the application

```
2 150 [0x0]
  2 150 /.../bin/demo[0x41702e]
    2 150 /lib64/libc.so.6(__libc_start_main+0xe5)[0x14ef40110d85]
      2 150 /.../bin/demo(main+0x32)[0x417122]
        2 150 /.../bin/demolib.so(MAIN__+0x7b4)[0x4178f4]
          0 0 /.../bin/demolib.so(foo+0xb3cf)[0x14efbe9bb3ef]
            0 0 /.../bin/demolib.so(bar+0x123e)[0x14efbe9bcd2e]
          1 100 /.../bin/demolib.so(biz+0xb3df)[0x14efbe9bb3ff]
            1 100 /.../bin/demolib.so(bar+0x123e)[0x14efbe9bcd2e]
          1 50 /.../bin/demolib.so(bar+0x123e)[0x14efbe9bcd2e]
```

Example of Tree mode output.
Columns are: allocations leaked, bytes leaked, and location of the leaked malloc call

5. PAPI integration

- **PAPI counters** can be used **from within DrHook**
  - No need for wrappers around applications
- Allows for easy per region statistics of DrHook instrumented code bases
  - No additional work needed from the user
- User specified PAPI **counters can be requested at runtime**
  - Implicitly enables platform native events

Work for the future

- Expand documentation coverage
  - Initial emphasis has been put on the user orientated documentation, developer documentation needs to be expanded
- Add link time extension ABI for more ad-hoc extensions
  - Allows users to simply define their own entry & exit functions, no need to change the DrHook core code
  - Functions are already in the DrHook core – compiled conditionally for security reasons
- Refine memory leak identification to make it more user friendly
  - Currently still in development & user testing
- Allow for custom number of PAPI counters
  - Currently hard limit of 4 due to technical complexity