Monitoring and Debugging

Simon Kuenzer  
CTO at Unikraft UG (haftungsbeschränkt)  
Senior Researcher at NEC Laboratories Europe GmbH  
simon@unikraft.io

Dr.-Ing. Marc Rittinghaus  
Head of Engineering at Unikraft UG (haftungsbeschränkt)  
Post-Doc and Researcher at KIT  
marc@unikraft.io
Unikraft Unikernel

- One application → Flat and single address space
- Single monolithic binary with only necessary kernel components
- Advantages from specialization
  - Performance and efficiency
  - Small TCB and memory footprint
  - Fast boot times

Platform (e.g., hypervisor, bare-metal)
Design Principles

- Specialization as main driving design principle
  - Highly customizable: KPI-driven specialization

- Philosophy: “Everything is a (micro-)library”
  - Decomposed OS primitives
    - Schedulers, memory allocators, VFS, network stacks, …
  - Architectures, platform support, and drivers
    - Virtualization environments, bare-metal
  - Application interfaces
    - POSIX, Linux system call ABI, language runtimes

- Widespread targets
  - Microservices, FaaS, NFV, Edge Computing, (Industrial) IoT and automotive, …
Monitoring and Debugging Features

- **ukdebug**
  - Logging/Print system
  - Assertions
  - Tracepoints
    - GDB server

- **uktest**
  - Unit Testing

- **ukstore**
  - Directory of library getters and setters

- **ubsan**
  - Detect run-time memory bugs

- **Uniprof (tool)**
  - Performance analysis with stack snapshots
1

Monitoring with ukstore
Requirements

- Re-use (micro-)library instrumentation
- ukstore is optional: Remove unneeded instrumentation at compile-time
- Allow retrieving of data and setting of values
- Getter/Setter interface defined by library
  - Name and data type (e.g., int, string)
- Pull-oriented design
  - Minimal overhead: Compute/parse only when requested
- Enable integration into common visualization/alerting systems
  - e.g., Prometheus, Grafana
Architecture

Micro-libraries
- ukalloc
- uknetdev
- uksched

Directory
- ukstore

Access backend
- REST
- Pseudo FS
- Shell command(s)

Provide entries → Look-up entries → Call getter/setter
An ukstore entry

- Name, data type, and function pointer to getter and/or setter

- **Static** entries (always available)
  - Compile-time, no run-time registration

  
<table>
<thead>
<tr>
<th>Library ID</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>uknetdev</td>
<td>interfaces_count (r-)</td>
</tr>
<tr>
<td>ukboot</td>
<td>request_shutdown (w)</td>
</tr>
</tbody>
</table>

- **Dynamic** entries
  - Created and removed at runtime
  - Entries per instance/object (e.g., thread, allocator, network interface)

  
<table>
<thead>
<tr>
<th>Library ID</th>
<th>Object ID</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>uknetdev</td>
<td>1</td>
<td>sent_bytes (r-)</td>
</tr>
<tr>
<td>ukalloc</td>
<td>0</td>
<td>avail_mem (r-)</td>
</tr>
</tbody>
</table>
Example: Grafana/Prometheus with ukstore
Current State and Future Work

- Currently upstreaming ukstore
  - https://github.com/unikraft/unikraft/pull/202

- Next
  - Provide set of initial instrumentation
    - Memory utilization: ukalloc
    - CPU utilization: uksched
    - Network utilization: lwip, uknetdev
    - Storage utilization: vfscore, ukblkdev
    - ...
  - Access backends
    - e.g., Prometheus/REST, pseudo-FS, shell
New Debugging Features in Unikraft (ukdebug)
New Debugging Features in Unikraft

Integrated GDB Stub

Uniform Crash Screen

0.180707 CRIT: Unikraft crash - Dione (0.6.0-2925462)
0.181395 CRIT: RIP: 00008:00000000001e0e8
0.181461 CRIT: RSP: 0010:000000001fffe20 EFLAGS: 00000002 ORIG RAX: 0000000000000000
0.181545 CRIT: RAX: 000000001fffe20 RBX: 0000000000000000 RCX: 0000000001fffe0
0.182428 CRIT: RDX: 0000000000000000 RSI: 0000000000000100
0.183351 CRIT: RSB: 0000000000000000 R11: 0000000000000000
0.183833 CRIT: R12: 0000000000000000 R13: 0000000000000000
0.183833 CRIT: R14: 0000000000000000 R15: 0000000000000000
0.184317 CRIT: Stack:
0.185118 CRIT: 0000000001fffe20 00 00 00 00 00 00 00 00 |........|
0.185483 CRIT: 00000001fffe28 00 00 00 00 00 00 00 00 |........|
0.185868 CRIT: 00000001fffe30 00 00 00 00 00 00 00 00 |........|
0.186245 CRIT: 00000001fffe38 00 00 00 00 00 00 00 00 |........|
0.186638 CRIT: 00000001fffe40 00 00 00 00 00 00 00 00 |........|
0.187086 CRIT: 00000001fffe48 00 00 00 00 00 00 00 00 |........|
0.187391 CRIT: 00000001fffe50 00 00 00 00 00 00 00 00 |........|
0.187778 CRIT: 00000001fffe60 00 00 00 00 00 00 00 00 |........|
0.188177 CRIT: Call Trace:
0.188394 CRIT: 00000001fffe85 _ukplat_entry+Sc4
0.188941 CRIT: 00000001fffe31 _ukplat_entry_argp+Rb
0.189391 CRIT: 000000000082c7 _libkvmplat_entry2+29
0.189623 CRIT: 0000000106405 _libkvmplat_newstack+F
0.189965 CRIT: Could not initialize the scheduler
0.110273 Info: [libkvmplat] <shutdown.c @ 35> Unikraft halted
GDB Debugger Support

- QEMU/KVM provides GDB debugger stub
  - Source-level guest debugging
  - Single-stepping, breakpoints, etc.

- BUT:
  - No debugging support on other platforms (e.g., Hyper-V, bare metal, cloud)
  - Semantic gap (e.g., no thread-level debugging)
  - No debugger integration in crash handling (e.g., failed asserts, kernel crash)

Want: Guest-level debugger support
GDB is obvious choice
Needed Components

- Communication channel
  - Should be available early in the boot phase

- GDB stub
  - Processing of GDB commands

- Means to react to debugging events
  - Architectural events (traps, breakpoints)

- Debugger invocation in error conditions
  - Failed assert, kernel crash

- But: Adhere to Unikraft philosophy
  - Implement as optional / replaceable micro-library
Communication Channel

- Unikraft has very short boot phase
  - System far into the boot when network is available

- Serial device
  - No requirements on other subsystems (e.g., memory allocator)
  - Available on most platforms
  - Quick to setup
  - Simple to use

- But: Already used for kernel messages
  - Share with serial console
  - Dedicate to debugger
GDB Stub – Protocol Handling

- Responsible for communication with GDB
  - Packet-based protocol

- Packet Data
  - Command + parameters
  - Payload encoded depending on packet type (e.g., as hex string)

- Checksum
  - Two-digit hexadecimal sum of all characters in packet data modulo 256

- GDB stub is mostly parsing of packets
  - 70% (~1000 LoC) for protocol handling
  - 10 commands needed for basic operation
GDB Stub – Architecture Integration

■ Responsibilities
  - Save and restore CPU context
  - Read and write memory
  - Set up single-stepping (i.e., set trace flag in EFLAGS register)
  - Implement trap handlers (e.g., breakpoint)

■ Setting / unsetting of breakpoints done by GDB client 😊
  - Replace instructions with debug break (memory read/write commands)
  - BUT: HW breakpoints (watchpoints) need support by stub (not yet)

■ Unikraft supports: x86-64 (450 LoC) and ARM64 (250 LoC)
Trap Handling

- Debugger must react to traps (e.g., breakpoint, single step)
- Could manually invoke debugger in trap handling code
- BUT: Would create dependency in platform to optional GDB stub library

Better: Extensible trap interface

```
void debug_trap(...) {
    gdb_stub();
}

void pagefault_trap(...) {
    /* unhandled */
    gdb_stub();
    CRASH();
}
```
Extensible Trap Interface

- Event-based interface
  - Platform defines and raises events
- Any library can define handlers
  - Link-time handler registration
  - Handler priorities

```
EVENT_HANDLER(TRAP_DBG, gdb_dbg_trap);
```

```
void debug_trap(...) {
    if (uk_raise_event(TRAP_DBG, ctx))
        return;
}
```

Unikraft Guest

Crash — GDB Stub

Platform support code

Traps

KVM x86_64

KVM ARM64
GDB Stub

- Problem: GDB might access invalid memory addresses
  - Tries to interpret integers as pointers (e.g., in ASM TUI mode)
  - Tries to read from invalid pointer (e.g., during backtrace)
  - User command leads to unintentional invalid memory access

- GDB stub may crash system when performing illegal access

Need way to catch invalid memory accesses
Non-Faulting Memory Accesses

- Want: Try memory access and return error on illegal access (i.e., no crash)
  - Flexible approach: Just try and catch illegal memory accesses

```c
int uk_memcpy_nofault(char *dst, const char *src, size_t len) {
    nf_copy_loop(dst, src, len, fault); /* just try memcpy */
    return len;
}
```

- Register low-priority pagefault handler
  - Maps trapped IP to entry in exception table
  - Each `nofault()`-call receives entry in table
  - Entry provides continuation IP for exception handler
Overview

Debug Library
- GDB Stub
- Assert() Stub
- Crash() Stub
- Unhandled Trap Events

Nofault Library
- Memory Access Table
- Exception Handler

Platform Support Code
- Serial Driver

GDB
- Serial Connection

Traps
- KVM x86_64
- KVM ARM64

Traps
- Trap Events
- Raised

KVM x86_64
- KVM ARM64
Uniform Crash Experience

- Previously: Every architecture had its own crash handling code
  - Dumping registers, halting system, etc.

- Want: Uniform experience on all architectures
  - Invoke debugger if available

---

<table>
<thead>
<tr>
<th>CPU</th>
<th>CRIT</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>x86-64</td>
<td>Unikraft crash - Dione (0.6.0-2925462)</td>
<td>Unikraft halted</td>
</tr>
<tr>
<td>arm64</td>
<td>Unikraft crash - Dione (0.6.0-2925462)</td>
<td>Unikraft halted</td>
</tr>
</tbody>
</table>

---

5. February 2022  Debugging and Monitoring in Unikraft  Simon Kuenzer, Marc Rittinghaus
Uniform Crash Experience

- Symbol resolution
  - Uses <IP, string>-like table
  - Link unikernel three times
    1. Link without table and extract debug symbols
    2. Link with table (might change symbol addresses!). Extract debug symbols again
    3. Link with updated table
  - API to resolve symbols at runtime

```c
int uk_resolve_symbol(unsigned long addr, struct uk_symbol *sym);
```
Current State and Future Work

- Currently upstreaming features

- Next:
  - Thread and SMP support
  - Hardware watchpoints
  - Custom commands
    - Crash dump
    - Inspect state (IRQ, paging, …)
  - Debugging over network connection
3

The Project, the Company
Join us!

- **OSS project**
  unikraft.org

- **Code & Contributing**
  github.com/unikraft

- **Documentation**
  docs.unikraft.org

- **Follow us on**
  - Twitter: [@UnikraftSDK](https://twitter.com/UnikraftSDK)
  - LinkedIn: [https://linkedin.com/company/unikraft-sdk](https://linkedin.com/company/unikraft-sdk)
Want to try out Unikraft at your Company?

- Please, contact us!
  - simon@unikraft.io
  - felipe@unikraft.io
  - alex@unikraft.io

- Unikraft UG (haftungsbeschränkt)
  Im Neuenheimer Feld 582
  69120 Heidelberg
  GERMANY

https://unikraft.io

Save your free trial!
Thank you!