Debugging in Unikraft

Hugo Lefeuvre
The University of Manchester

Lyon Unikraft Tutorial @ENS, May *14th*

Debugging in Unikraft: Content

- Debugging in Unikraft is not that hard
- Checklist to debug a Unikraft unikernel
- Debugging Tips and Tricks
- Many commands to copy and paste :-)

Debugging Unikraft is not that hard

Debugging in Unikraft is simpler than debugging your mainstream OS

- Everything is a single binary in a single trust domain
- You debug the app and the kernel at the same time seamlessly
- Unikraft is small

Debugging Checklist

Don't forget to:

- Uncheck `Drop unused functions and data` in `Build Options`
  - We poor humans have difficulties reasoning about this
- Use `make V=1` to debug build system issues
- Toggle optimizations
- Enable maximum debug information level in `Build Options`
- Enable assertions under `ukdebug`
- Enable more print output from the kernel under `ukdebug`
- Switch the memory allocator: will change the memory layout and maybe provide your with more information about your bug
- Enable ASan/UBSan to debug any sort of memory corruption issue
- Networking related bugs: enable lwip debug output
- Etc.

Debugging Checklist

Non exhaustive

Don't forget to:

- Uncheck `Drop unused functions and data` in `Build Options`
  - We poor humans have difficulties reasoning about this
- Use `make V=1` to debug build system issues
- Toggle optimizations
- Enable maximum debug information level in `Build Options`
- Enable assertions under `ukdebug`
- Enable more print output from the kernel under `ukdebug`
- Switch the memory allocator: will change the memory layout and maybe provide your with more information about your bug
- Enable ASan/UBSan to debug any sort of memory corruption issue
- Networking related bugs: enable lwip debug output
- Etc.

Debugging Checklist

Non exhaustive

Don't forget to:

- Uncheck `Drop unused functions and data` in `Build Options`
  - We poor humans have difficulties reasoning about this
- Use `make V=1` to debug build system issues
- Toggle optimizations
- Enable maximum debug information level in `Build Options`
- Enable assertions under `ukdebug`
- Enable more print output from the kernel under `ukdebug`
- Switch the memory allocator: will change the memory layout and maybe provide your with more information about your bug
- Enable ASan/UBSan to debug any sort of memory corruption issue
- Network gs: enable lwip debug output
- Ask on Discord

GDB v.s. Printing

Printing is good for quick and dirty checks, but it has its limits:

- Can get unhandy with complex problems
- Can influence your bug because of performance impact

GDB v.s. Printing

Printing is good for quick and dirty checks, but it has its limits:

- Can get unhandy with complex problems
- Can influence your bug because of performance impact

GDB works seamlessly with Unikraft

- Remember to use the .gdb image generated by the build system
- Slightly different approach depending on the platform
  - Here: linuxu and QEMU/KVM

GDB copy & paste snippets

Debugging with linuxu is very simple:

```
$ gdb build/app-helloworld_linuxu-x86_64.dbg
```

This is your usual userland process.

GDB copy & paste snippets

Debugging with KVM happens in server/client fashion

GDB copy & paste snippets

Debugging with KVM happens in server/client fashion

1. Start your unikernel image in paused state

```
$ qemu-system-x86_64 -s -S -cpu host -enable-kvm -m 128 -nodefaults -no-acpi
-display none -serial stdio -device isa-debug-exit -kernel build/app-
helloworld_kvm-x86_64.dbg -append verbose
```

GDB copy & paste snippets

Debugging with KVM happens in server/client fashion

1. Start your unikernel image in paused state

```
$ qemu-system-x86_64 -s -S -cpu host -enable-kvm -m 128 -nodefaults -no-acpi
-display none -serial stdio -device isa-debug-exit -kernel build/app-
helloworld_kvm-x86_64.dbg -append verbose
```

(or with qemu-guest)

```
$ qemu-guest -P -g 1234 -k build/app-helloworld_kvm-x86_64.dbg
```

GDB copy & paste snippets

Debugging with KVM happens in server/client fashion

1. Start your unikernel image in paused state

```
$ qemu-system-x86_64 -s -S -cpu host -enable-kvm -m 128 -nodefaults -no-acpi
-display none -serial stdio -device isa-debug-exit -kernel build/app-
helloworld_kvm-x86_64.dbg -append verbose
```

(or with qemu-guest)

```
$ qemu-guest -P -g 1234 -k build/app-helloworld_kvm-x86_64.dbg
```

Port of the GDB server

GDB copy & paste snippets

Debugging with KVM happens in server/client fashion

1. Start your unikernel image in paused state
2. **Connect to your image's GDB server**

```
$ gdb --eval-command="target remote :1234" -ex "set confirm off" -ex "set pagination off" \
        -ex "disconnect" -ex "set arch i386:x86-64:intel" \
        -ex "tar remote localhost:1234" build/app-helloworld_kvm-x86_64.dbg
```

You can now run `continue` and debug as usual.

GDB copy & paste snippets

Debugging with KVM happens in server/client fashion

1. Start your unikernel image in paused state
2. **Connect to your image's GDB server**

Make sure ports match with your GDB server

```
$ gdb --eval-command="target remote :1234" -ex "set confirm off" -ex "set pagination off" \
        -ex "disconnect" -ex "set arch i386:x86-64:intel" \
        -ex "tar remote localhost:1234" build/app-helloworld_kvm-x86_64.dbg
```

You can now run `continue` and debug as usual.

GDB copy & paste snippets

Debugging with KVM happens in server/client fashion

1. Start your unikernel image in paused state
2. Connect to your image's GDB server

```
$ gdb --eval-command="target remote :1234" -ex "set confirm off" -ex "set pagination off" \
        -ex "disconnect" -ex "set arch i386:x86-64:intel" \
        -ex "tar remote localhost:1234" build/app-helloworld_kvm-x86_64.dbg
```

You can now run `continue` and debug as usual.

Note: if you want to debug early code, you will need to use hardware breakpoints

GDB copy & paste snippets

Debugging with KVM happens in server/client fashion

1. Start your unikernel image in paused state
2. **Connect to your image's GDB server**

Note: if you want to debug early code, you will need to use hardware breakpoints

```
$ gdb --eval-command="target remote :1234" -ex "set confirm off" -ex "set pagination off" \
        -ex "hbreak myfunc" \
        -ex "disconnect" -ex "set arch i386:x86-64:intel" \
        -ex "tar remote localhost:1234" build/app-helloworld_kvm-x86_64.dbg
```

Debugging Tips and Tricks

Remember that:

- `printf` and `uk_pr_*` are not the same
  - On KVM, `uk_pr_*` go through I/O ports while `printf` goes through stdio device
  - This can impact your bug
- You can run on linuxu, this might give you different insights on what is going on
- You are running with a cooperative scheduler :-)

Debugging in Unikraft

Work items :-)