Inside Unikraft: Building, configuring, using different libraries

Hugo Lefeuvre
The University of Manchester

Lyon Unikraft Tutorial @ENS, May *14th*

Inside Unikraft: Content

- Broad overview of Unikraft's architecture
- Broad overview of Unikraft's build system
- Building and configuring apps and libs in Unikraft
- Running Unikraft on different platforms

Architecture of Unikraft

3 main components of the Unikraft core:

- Platform code
- Architecture code
- Internal libraries

Architecture of Unikraft

3 main components of the Unikraft core:

- **Platform code**
- Architecture code
- Internal libraries

- Hardware-specific code
- Drivers
- Depending on whether a hypervisor is present, it will do different things
  - KVM: behavior like bare metal
  - Linuxu: do system calls to emulate hardware
  - Xen: hypercalls...

Architecture of Unikraft

3 main components of the Unikraft core:

- Platform code
- **Architecture code**
- Internal libraries

- Architecture-specific code
  - Supported: x86 and ARM-64, ongoing RISC-V
- Example on x86:
  - Usable registers & hardware limits (page size, etc.)
  - How to use Thread-Local Storage

Architecture of Unikraft

3 main components of the Unikraft core:

- Platform code
- Architecture code
- **Internal libraries**

- Behavior independent of the hardware
  - Examples: scheduling, memory management, file systems, synchronization, etc.
- Rely on underlying platform and arch code that abstract the HW
- Generally:
  - Internal libraries = kernel functionalities
  - External libraries = user functionalities
  - Though not always a clear cut (unikernel...)

-> uknetdev versus lwip

Platforms in Unikraft

Supported platforms, among others:

Platforms in Unikraft

Supported platforms, among others:

- Virtualized
    - KVM
    - Xen
    - Firecracker
    - ...

VM Emulated or paravirtualized

Platforms in Unikraft

Supported platforms, among others:

- Virtualized
  - KVM
  - Xen
  - Firecracker
  - ...
- Userland
  - linuxu

VM Emulated or paravirtualized

Running as a userland process

Linuxu = development and debugging, NOT for performance evaluation/prototyping

LibCs in Unikraft

Standard libraries:

- Nolibc (internal)
- Isrlib (internal)
- Newlibc (external)
- Musl (external)

LibCs in Unikraft

Standard libraries:

- **Nolibc** (internal)
    - Minimum service lib for the kernel (strings, sort, etc.)
    - Homebaked, content cherry picked from other libcs
- Isrlib (internal)
- Newlibc (external)
- Musl (external)

LibCs in Unikraft

Standard libraries:

- Nolibc (internal)
- **Isrlib** (internal)
  - Interrupt-context variant of nolibc
- Newlibc (external)
- Musl (external)

LibCs in Unikraft

Standard libraries:

- Nolibc (internal)
- Isrlib (internal)
- **Newlibc** (external)
    - Current default for the Unikraft userland
    - Initially thought for the embedded world
    - *Somewhat* complete libc
        - Lacks multithreading features, among others
    - Does the trick for many programs but not all
- Musl (external)

LibCs in Unikraft

Standard libraries:

- Nolibc (internal)
- Isrlib (internal)
- Newlibc (external)
- **Musl** (external)
  - Full-featured libc
  - Really good compatibility, implements some glibc quirks as well
  - Undergoing port, will be the future default

Unikraft's Build System

Unikraft is meant for specialization

- Understand = it's super configurable

Unikraft's Build System

Unikraft is meant for specialization

- Understand = it's super configurable

Configuration is done using KConfig (like in Linux)

Unikraft's Build System

Unikraft is meant for specialization

- Understand = it's super configurable

Configuration is done using KConfig (like in Linux)

At the center: `Config.uk` files

- Enable libs, apps, internal and external
- Declare dependencies between libraries

Unikraft's Build System

Unikraft is meant for specialization

- Understand = it's super configurable

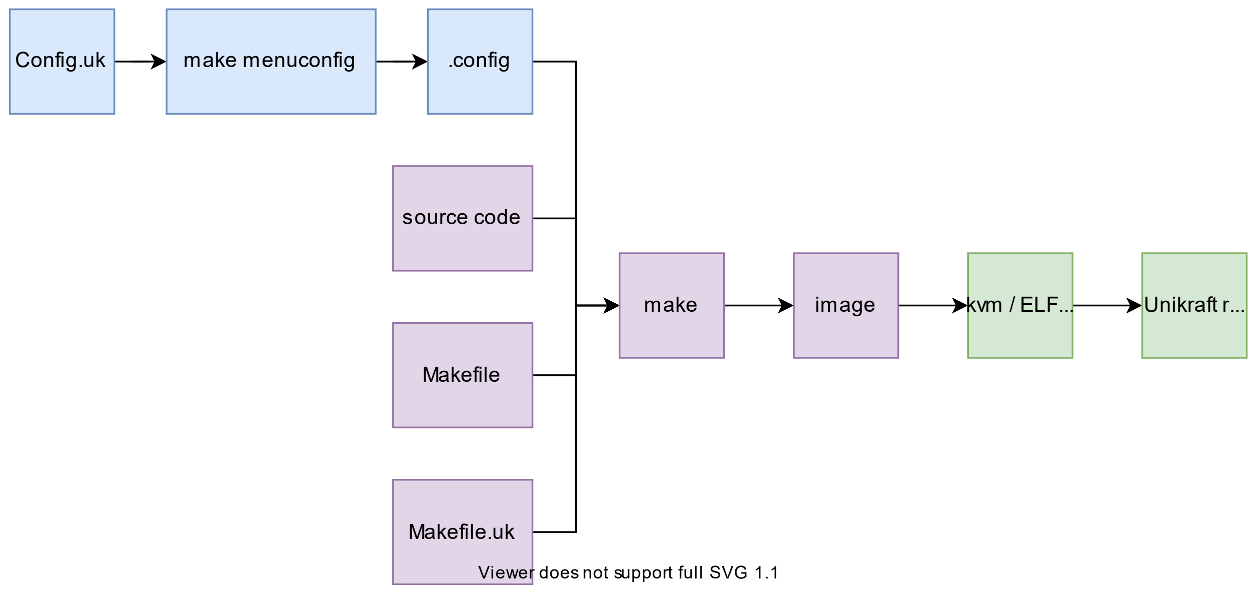
Configuration is done using KConfig (like in Linux)

At the center: `Config.uk` files

- Enable libs, apps, internal and external
- Declare dependencies between libraries

`make menuconfig` takes `Config.uk` and turns it into a `.config` that the build system understands

Unikraft's Build System



`make menuconfig` takes `Config.uk` and turns it into a `.config` that the build system understands

Using the Build System

3 main ways of configuring Unikraft (from low to high level)

Using the Build System

3 main ways of configuring Unikraft (from low to high level)

1. Run `make menuconfig` and edit config manually

Using the Build System

3 main ways of configuring Unikraft (from low to high level)

1. Run `make menuconfig` and edit config manually
2. Edit `Config.uk` using `depends on` and `select` to let KConfig edit the configuration automatically (and detect conflicts)

Using the Build System

3 main ways of configuring Unikraft (from low to high level)

1. Run `make menuconfig` and edit config manually
2. Edit `Config.uk` using `depends on` and `select` to let KConfig edit the configuration automatically (and detect conflicts)
3. Declare dependencies in `Kraft.yaml` and let Kraft configure automatically

Using the Build System

3 main ways of configuring Unikraft (from low to high level)

1. Run **`make menuconfig`** and edit config manually
2. Edit `Config.uk` using `depends on` and `select` to let KConfig edit the configuration automatically (and detect conflicts)
3. Declare dependencies in **`Kraft.yaml`** and let Kraft configure automatically

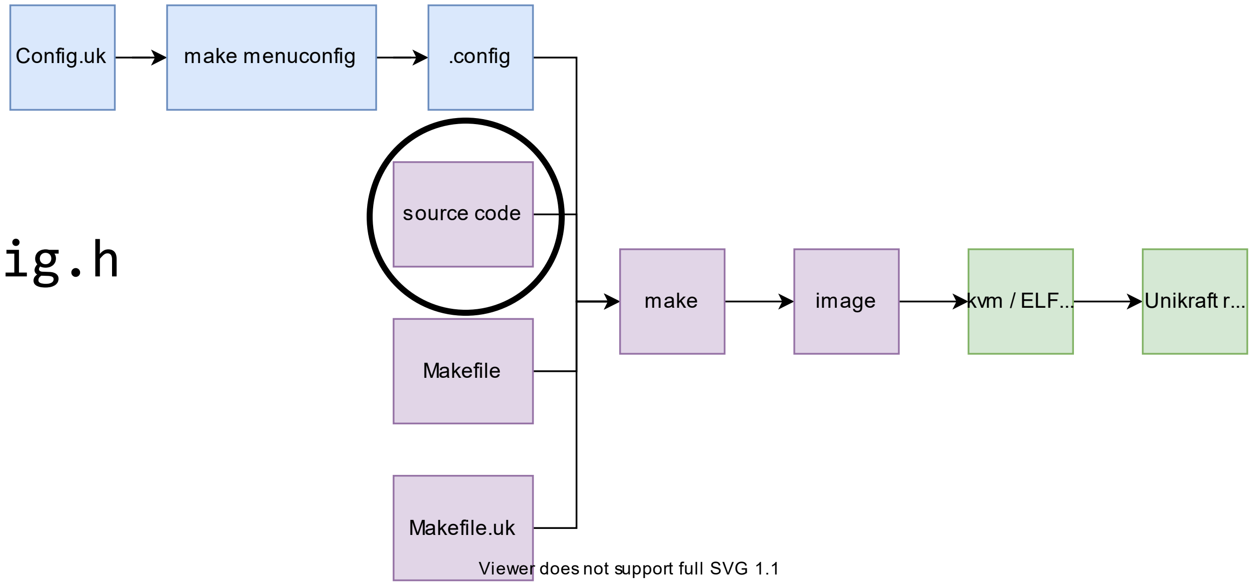
Focus on 1 & 3 in this session

Using the Build System

Symbols are defined according to `.config`'s content, e.g.,

- `CONFIG_ARCH_X86_64`
- `CONFIG_LIBTLSF_INCLUDED`
- `CONFIG_HAVE_SCHED`

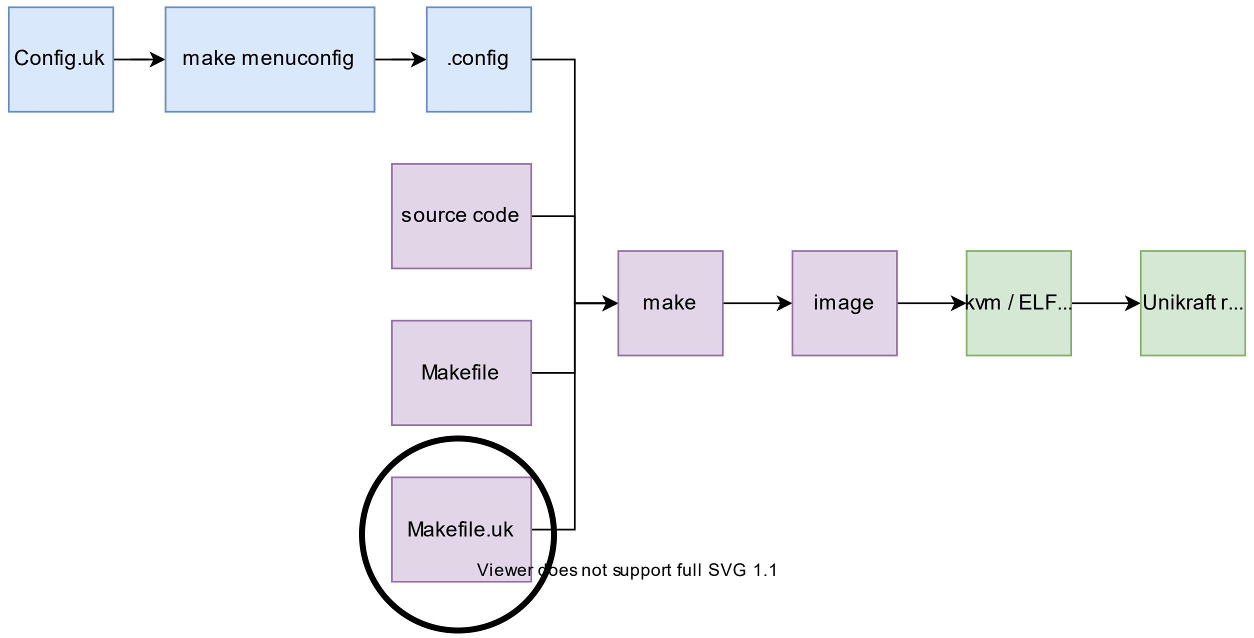
You can use these symbols in your C code, provided you include `uk/config.h`



Using the Build System

Configuration options can also be used in `Makefile.uk`

- Governs behavior of the build system
- Which file gets included, which file doesn't
- If external data must be fetched and uncompressed, define how



Inside Unikraft

Work items :-)