Spelunking Fuchsia

An overview

Google is in NO WAY associated with this talk. Information presented is based on information that is publicly available.

Why a talk on Fuchsia?

No agenda, let’s explore…

Overview

- What is it?
- Examine elements of the design / code
- How to get / build

Google is in NO WAY associated with this talk.
Information presented is publicly available.

What is it?

- Fuchsia is :
  - “A modular capability based operating system”
  - An open project, with irc, gerrit for code submissions, etc
  - Zircon - kernel - it is not monolinic as per Linux, it’s a microkernel more akin to Mach
  - Zircon core - things around the kernel (more about this in a bit)
  - Frameworks - application frameworks
  - Networking, Graphics, Storage
- It’s an operating system

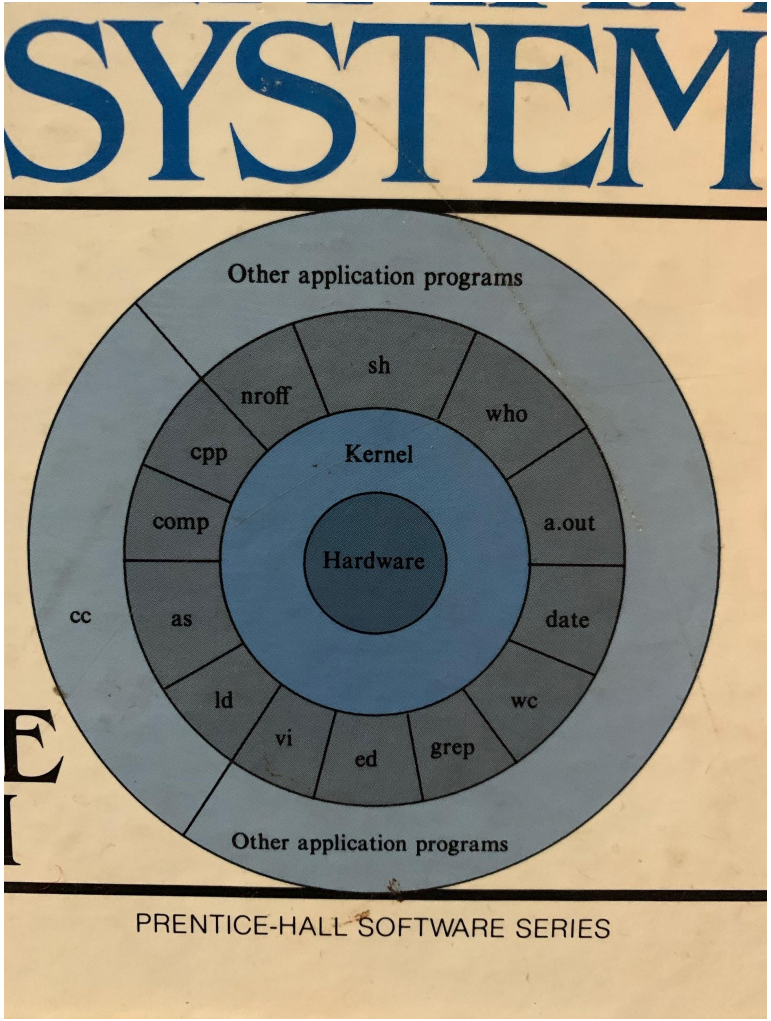
What is it for?

- While Linux followed in footsteps of UNIX tradition, it was easy to say it’s an X86 Unix that is free.
- Fuchsia as project hasn’t answered this question.
  - It’s different, provides a subset of POSIX and isn’t trying run any random open source package.
  - From the source there are hints...

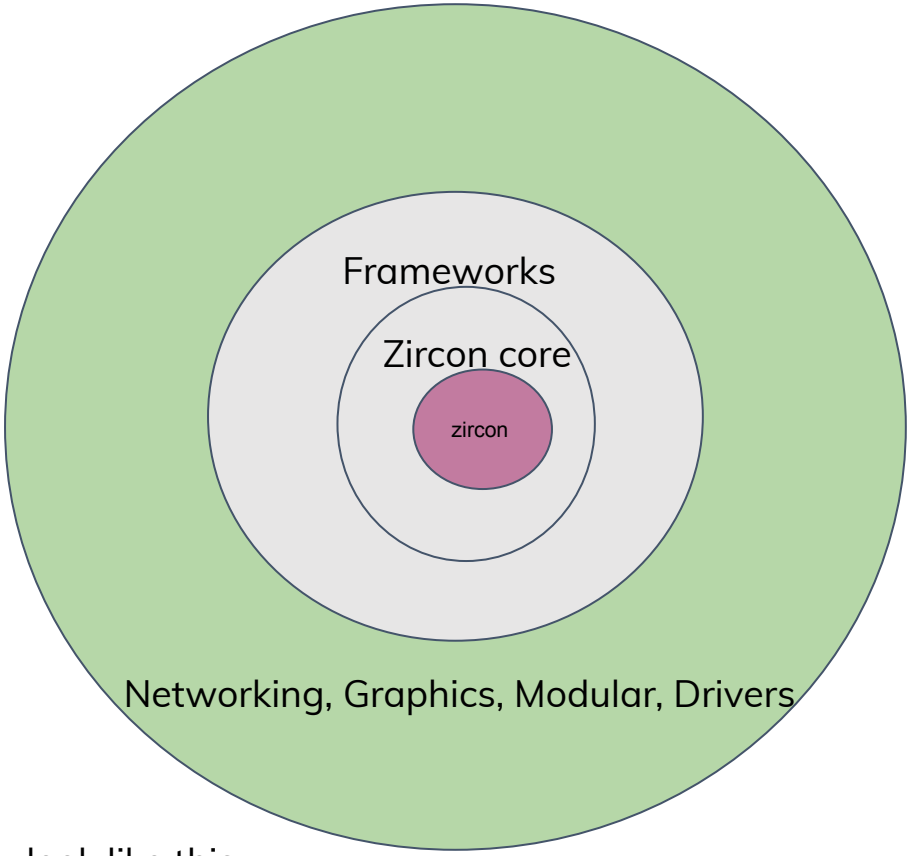
Zircon & LK

- Zircon started as a branch of LK
  - LK used in Android Trusty TEE
  - LK as Android bootloader …
- Zircon is 64 bit only, LK supports 32 bit
- Zircon has user mode support, LK does not
- Zircon is capability based security, LK all code is trusted

So… they've devolved from each other…



If you remember Bach...



Fuchsia might broadly look like this...

System Calls

- User space interacts with the kernel via system calls
- Generally non-blocking
- There are roughly 3 types

1. Call has no limitations, anyone can use Ex: zx_clock_get, zx_nanosleep
2. Calls that take a handle as first parameter. Handle indicates which kernel object is the target Ex: zx_port_queue(zx_handle_t handle, const zx_port_packet_t* packet);
3. Calls which create a new kernel object Ex: zx_event_create(uint32_t options, zx_handle_t* out0, zx_handle_t* out1)

From user space libzircon.so makes the syscalls available. It's a vDSO. (virtual Shared Dynamic Object)

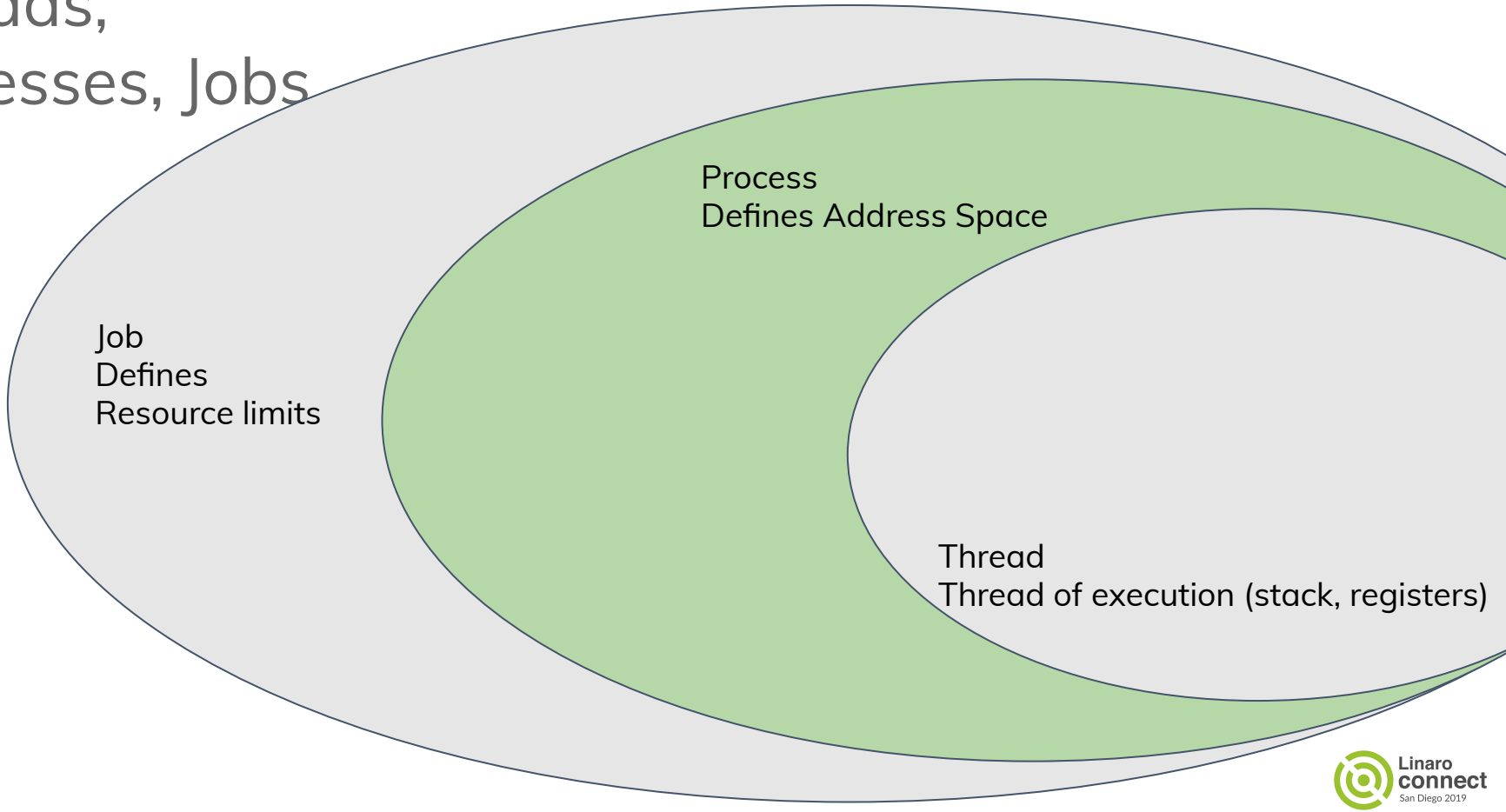
Handles - User space references to Kernel Objects

- A kernel object may have multiple handles in the same or multiple processes
- When the last open handle is closed, the kernel object is generally destroyed or put into a final state
- Handles can be passed between processes using a Channel or as part of new process creation
- A Handle contains rights. Two handles to the same kernel object could have different rights
- Actions that may be taken against an object are governed by the rights in the handle

Kernel Object IDs (koids)

- Each Kernel Object has a unique 64 bit unsigned int associated with it.
- Used to uniquely identify an object
- Unique for the lifetime of the running system
  - No reuse
- Kernel generated koids only use 63 bits
  - Leaves space for artificial koids that can then use the most significant bit for artificial objects
- There are no rules for how koid values are generated

Threads, Processes, Jobs



Other Zircon Kernel Features

- Sockets (stream) & Channels (datagram) for Message passing
- Signals - single waitable bit exposed by a kernel object - meant to imply status info, ZX_CHANNEL_READABLE means a channel endpoint has a message to read
- Event - Simple object that is a collection of active signals
- Event Pairs - Events that might signal each other. (PEER_CLOSED)
- Virtual Memory Objects (VMOs) - represent a set of physical pages OR potential for pages
  - zx_vmo_write - create a VMO, map it, write a dataset into it, pass it along
- Address Space Management - VMAR - abstraction for managing process address space
- Futexes - zx_futex_wait(), zx_futex_wake(), zx_futex_requeue() - sync primitives

Zircon and c++

- -std=c++17
- No : Exceptions, RTTI and dynamic_cast, Operator overloading, Virtual inheritance, Statically constructed objects, Trailing return type syntax (Exception: when necessary for lambdas with otherwise unutterable return types), Initializer lists, thread_local in kernel code
- No std::, but instead ktl::
- The rules : https://fuchsia.dev/fuchsia-src/zircon/cxx
- FBL - Fuchsia Base Library - shared between kernel and user space with very strict deps
  - FBL can't dep on syscalls because syscalls aren't in the kernel
  - FBL can't dep on C lib features that aren't available in kernel
  - Provides, allocators, basic algs, arrays, lists, hashtables, smart pointers, raii
- Hermetic C++ - C++ in use, however ABI compatibility is paramount, so pure C API for externally facing

Rights

- Associated with Handles - grants privilege to do a thing
  - Ex: ZX_RIGHT_READ, ZX_RIGHT_EXECUTE
- ZX_RIGHTS_BASIC : ZX_RIGHT_DUPLICATE, ZX_RIGHT_TRANSFER, ZX_RIGHT_WAIT, and ZX_RIGHT_INSPECT
- `<zircon/rights.h>` Defines default rights for Objects
- Copying a handle or passing a handle can reduce the rights the next handle will have!

Booting userspace

- ZBI - Zircon Boot Image, ramdisk, command line options
  - Ramdisk contains a trivial bootfs format filesystem
  - w/ drivers, utilities, drivers for better storage
- Userboot
  - First user space process
  - Elf binary built as an RODSO
    - RODSO is more straightforward, map elf PT_LOAD segments, that's it, no relocations
  - Userboot is built into the kernel at compile time.
- First process start
  - Kernel maps userboot and the syscall vDSO into memory
    - For userboot ONLY syscall vDSO is right after userboot in memory and thus always at a fixed offset
    - For every other user app, syscall vDSO is located at a random address
  - Control transferred, userboot reads bootstrap messages and uses handle containing a pointer to the BOOTFS image, uncompresses if needed and loads either bin/bootsvc or userboot=file (ie whatever was passed to the kernel)
  - Userboot has a full elf program loader
  - If userboot.shutdown is on, when userboot exits, system shuts down, otherwise, it is the responsibility of bootsvc or akin to do it.

Zircon Device Drivers

- DDs are elf shared libs (DSOs), loaded into devhost Process
- devmgr contains Device Coordinator which
  - tracks drivers and devices
  - manages the discovery of drivers
  - the creation and direction of Device Host processes
  - maintains devfs, the Device Filesystem
- Devfs
  - User space services and apps gain access via devfs
  - Device Coordinator views all devices as a single unified tree
  - System decides how to sub-divide the tree for security, stability, performance or other reasons
- Device Drivers
  - init() - called when driver loaded, allows for global initialization
  - bind() - called when a driver is given a device to bind to, must create a child node
  - create() - called for platform, system bus drivers or proxy drivers, otherwise not used
  - release() - called before a driver is unloaded, (currently unused)

DD

- A new operating system means new device drivers, right?
  - Well, seems a few are copied from netbsd and bolted in
- Device Drivers device their compatibility through bindings:

```
ZIRCON_DRIVER_BEGIN(ahci, ahci_driver_ops, "zircon", "0.1", 4)
    BI_ABORT_IF(NE, BIND_PROTOCOL, ZX_PROTOCOL_PCI),
    BI_ABORT_IF(NE, BIND_PCI_CLASS, 0x01),
    BI_ABORT_IF(NE, BIND_PCI_SUBCLASS, 0x06),
    BI_MATCH_IF(EQ, BIND_PCI_INTERFACE, 0x01),
ZIRCON_DRIVER_END(ahci)
```

Getting started with Fuchsia

- Source
  - curl -s "https://fuchsia.googlesource.com/fuchsia/+/master/scripts/bootstrap?format=TEXT" | base64 --decode | bash
- Jiri
  - Like repo in Android, Fuchsia has a tool for working with lots of git repos at once
  - https://fuchsia.googlesource.com/jiri
- > fx list-products
  - Bringup, core, router, speaker, terminal, workstation
- > fx list-boards
  - HiKey960, qemu, kirin970, msm8x-53, mt8167s, x86…
- fx set <product.board>
  - Fx set core.arm64
- > fx build
- > fx run (runs in qemu)

Fuchsia community

- Irc - Freenode - #fuchsia
- Docs / website - http://fuchsia.dev
- Code submissions via Gerrit
  - Must sign Google CLA to submit code

Summary

- Fuchsia is under development
- Fuchsia looks to be doing its best to follow open development
- Fuchsia.dev is a good website to explore as I've left out much!
- Worthwhile project depending on what your goals are

Google is in NO WAY associated with this talk.
Information presented is publicly available.

Thank you

Join Linaro to accelerate deployment of your Arm-based solutions through collaboration

contactus@linaro.org

Linaro connect
San Diego 2019