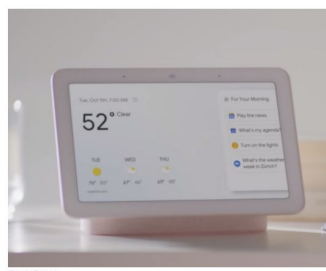




# Building an Operating System from Scratch with LLVM



Zdroj: Google



Google již nějakou dobu pracuje na novém operační OS. Původně uváděl, že se jedná jen o experiment a testuje nejruznější postupy. Poměrně nedávno ale do že se dostane do běžných produktů. V tomto roce se že se testuje Fuchsia OS na zařízení do domácnosti. oficiální oznámení.

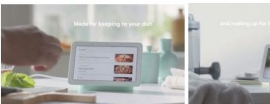
Google Home Hub – chytrý asistent s displejem c

## Fuchsia OS

Asi byste očekávali, že s novým systémem přijde i no Google se rozhodl pro jinou cestu. Oficiálně vydává F zařízení Google Home Hub, dnes známý pod označeř Toto zařízení do domácnosti pochází z roku 2018 a p uvedení dostává nový operační systém.

Samsung také spolupracuje na Fuchsia OS c

Fuchsia OS nahradí původní Cast OS založený na Lir zajímavé, tak samotní uživatel asi vůbec nepostřehne aktualizaci a že jejich Nest Hub běží na zcela novém Fuchsia OS. Google neupravil design, ani nepřináší n



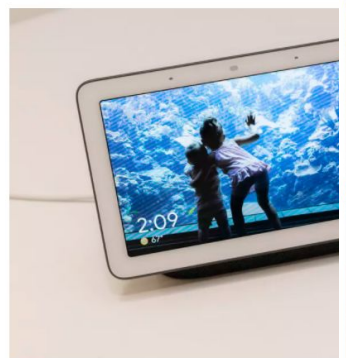
Zdroj: Google

Si se může zdát, že se jedná o poněkud zbytečnou vzhledem k vývoji se jedná o dobrou zprávu. Pokud r zhoršení výkonu a ani se neomezí funkcionalita, tak s považovat za krok kupředu. S největší pravděpodobn aktualizace na další zařízení postupem času.

# Google starts rolling original Nest Hub

Google's newest OS is starting to app

Eli Blumenthal May 25, 2021 10:06 a.m. PT



The first-generation Nest Hub, left, will be among the first James Martin/CNET

Google's Fuchsia OS has been in developm onto some devices. First up: the first-generati

The search giant on Tuesday started rolling o device. The change, which was earlier reporte existing Cast OS with Google's new Fuchsia s over the next few months and there won't be Hub owners, according to Google.

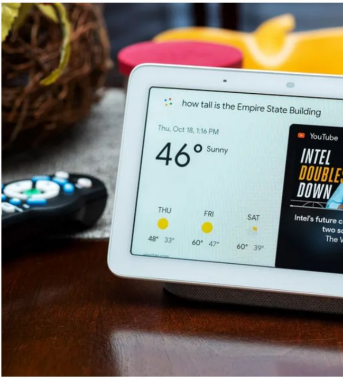
Unlike Android, Fuchsia was developed by Gc run Android apps, however, which should mak Android or Chrome OS a bit easier.

Google previously described the Fuchsia proj phones and modern personal computers" sugh than just smart home or internet-of-things dev meant to be a general purpose OS, though it's software might land on.

# Google's new Fuchsia OS arrives i Hub

Years after becoming public, Google's new OS has arrived

By Jon Porter | @JonPorter1 | May 25, 2021, 4:47am EDT



The Google Nest Hub. | Photo by Dan Seltzer / The Verge

Google's long-awaited Fuchsia OS is starting to quietly roll out on its first consum reports. Google's work on Fuchsia OS first emerged in 2016, and the open-sourc on a Linux kernel, instead using a microkernel called Zircon. "You don't ship a ne day," tweeted a Google technical lead on the Fuchsia OS project, Petr Hosek.

While the rollout on the Nest Hub (which originally released as the Google Home release process will take several months. It'll come to users in the Preview Progra known for a while that the operating system has been tested on the Nest Hub, an emerged thanks to a Bluetooth SIG listing that showed the Nest Hub running Fuc



Although the Nest Hub will swap its current Cast OS for Fuchsia OS, 9to5Google identical, and most users are unlikely to even notice the switch.

All of this raises the question of what exactly Fuchsia OS is meant to achieve. Goc that is secure, updatable, inclusive, and pragmatic." We know that the OS could t was spotted testing it on the Pixelbook back in 2018, and more recently it propose apps), but Fuchsia is not meant to be a one-for-one replacement of Android or Ch

"Fuchsia is about just pushing the state of the art in terms of operating systems a incorporate into other products," Android and Chrome chief Hiroshi Lockheimer s unlikely to be the last device or even form-factor to receive an update to Fuchsia take longer to emerge.

# Google is officially releasing w/ first-gen Nest Hub

Kyle Bradshaw May 25th 2021 12:01 am PT @jshubbs



Google's long-in-development, from-scratch operating sys Google devices, namely, the first-generation Nest Hub.

Google has told us that as of today, an update is beginnin out to owners of the first-generation Nest Hub, first releas 2018. For all intents and purposes, this update will not ch of the functionality of the Nest Hub, but under the hood, th display will be running Fuchsia OS instead of the Linux-ba OS" it used before. In fact, your experience with the Nest H be essentially identical. This is possible because Google's display experience is built with Flutter, which is designe consistently bring apps to multiple platforms, Fuchsia inc

We've been tracking the development of Fuchsia since 20 starting from an ambitious experimental UI, to running on many internal testing devices for Fuchsia, ranging the full Google's smart home and Chromebook lineup. In the time and recently even begun a steady release schedule.

Earlier this month, we spotted the first-generation Google release as it received new approval from the Bluetooth SIG from their internal testing process to something more pu could be intended as a way for developers to easily try the company's developer-focused event — now behind us, it s

The Fuchsia-based update for the Nest Hub will roll out on the Preview Program, before eventually becoming more b and experience will be unchanged, it's likely that Nest Hub switched over to Fuchsia OS. That said, Google appears to moving over the course of months, as switching operat



# Google's Nest Hub is getting a new OS you probably won't be able to spell

Fooshya-OS-Fuchysa-OS-Fushcia-OS-Fuchsia OS is finally ready for prime time

Prasham Parikh 7 hours ago

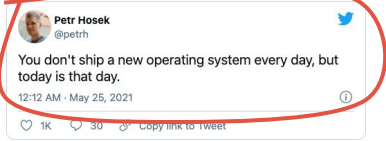


DEVICE UPDATES GOOGLE NEST HUB 9to5 NEWS



After being in development for years, Google's Fuchsia OS is finally ready to be used in a consumer product. Over the coming months, it will be rolled out to the first-gen Nest Hub. You needn't fret or get excited about the update since you'll barely be able to notice any difference when your device transitions to it.

The made-from-scratch Fuchsia OS will replace the Linux-based Cast OS that the smart display currently runs. Google confirms (via 9to5Google) that the update won't change any functionality on the Hub. This is possible because the smart display experience is built using Flutter, which is designed to allow apps to run consistently across multiple platforms, including Fuchsia OS.



The update will first arrive for those who are a part of the preview program and then be rolled out to others. Since the update involves switching the operating system, Google will be cautious with the rollout, making sure it doesn't anger too many customers with its software shenanigans. Google hasn't confirmed if or when the update will be brought to other smart displays.



I am Petr, Technical Lead of the Fuchsia toolchain team.


I joined Google and the Fuchsia project in 2015 and have been responsible for Fuchsia's Clang toolchain since 2016.

I am presenting a work of a large group of people.

Differential > D25116

## Add triple for Fuchsia

Closed  Public

 Authored by **phosek** on Sep 30 2016, 12:12 PM.

### Details

Reviewers  **davide**



Commits [rGe023d62e7691: \[Triple\] Add triple for Fuchsia](#)  
[rL283419: \[Triple\] Add triple for Fuchsia](#)

### ☰ SUMMARY

Fuchsia is a new operating system.

### Diff Detail

Repository [rLLVM](#)

  **phosek** updated this revision to **Diff 73110**.



## [Driver] Add driver support for Fuchsia

Closed

Public



Authored by **phosek** on Sep 30 2016, 12:14 PM.

### Details

Reviewers rsmith

Commits [rG62e1d2398669](#): [Driver] Add driver support for Fuchsia  
[rC283420](#): [Driver] Add driver support for Fuchsia  
[rL283420](#): [Driver] Add driver support for Fuchsia

### ☰ SUMMARY

Provide toolchain and tool support for Fuchsia operating system. Fuchsia uses compiler-rt as the runtime library and libc++, libc++abi and libunwind as the C++ standard library. lld is used as a default linker.

### Diff Detail

Repository [rL-LLVM](#)

## Initially, Clang toolchain was only used for Fuchsia userspace.

We had to improve the linker script support in LLD to support linking Zircon kernel and other low-level libraries.

We also implemented several new features to better match GNU linkers and addressed a number of issues.

We had to build entirely new tools as replacements for GNU counterparts such as `llvm-objcopy` and `llvm-strip`.

Fuchsia was  
open sourced

Clang Fuchsia  
driver landed

LLD can link  
Zircon kernel

All GNU Binutils  
replaced

Clang became a  
default compiler

Fuchsia  
public release

Jun  
2016

Oct  
2016

Apr  
2017

Dec  
2017

Jun  
2019

May  
2021

## It wasn't just technical reasons, LLVM had to prove itself.

GCC has been around for decades and developers have a lot of trust in it, especially for embedded.

There was evidence that Clang is ready, but there were other aspects like assembly and linker script support, binary tools, compiler runtimes and so on.

We had to do a lot of work behind the scenes to convince everyone that LLVM is really serious about embedded.



# Agenda

## Introduction

A short history of the project.

## What is Fuchsia?

A brief overview of the new operating system.

## How we built Fuchsia?

The role LLVM played in Fuchsia's development.

## What's next?

Problems we want to tackle in the future.

# What is Fuchsia?

Fuchsia is an open source operating system that prioritizes security, updatability, and performance.

Fuchsia was built from the ground up and uses a completely new system design which lets us explore ideas for a more efficient and secure design.

## Fuchsia in a nutshell.

Fuchsia is an internet-first operating system that's always up-to-date enabling very long support horizons.

Fuchsia is focused on minimal resource usage, but we aren't targeting embedded and provide a full user experience targeting 64-bit CPUs and Vulkan-capable GPUs.

After 5 years of development, we started shipping Fuchsia to the public for the first time on Google Nest Hub devices earlier this year.



## We are trying to address challenges of the current operating system paradigm.

More connected, always-on devices in private settings have greater need for security.

Security updates require ongoing investment reducing the product lifespan.

Every form factor requires a fork of the operating system which is costly to maintain.

# Principles

Security

Updatability

Inclusivity

Pragmatism

## Software has the least privilege it needs to perform its job.

No ambient authority—components can interact only with the objects to which they have been granted access.

Software isolation is enforced by the kernel so there's no need for additional security systems.

## Software components are independently updatable.

Much like the web, software on Fuchsia is designed to come and go as needed and be always up-to-date.

The system has mechanisms to ensure updatability including hermetic packaging, versioning, evolution affordances for protocols, etc.

**Inclusive throughout, from the  
architecture to the community.**

Fuchsia is runtime and language agnostic—developers can use runtime and language that's best for their product.

Fuchsia is an open source project developed under permissive license with a public governance model.



## Not a playground for experimental operating system concepts.

Fuchsia is designed to power consumer devices and must adhere to fundamentals, like performance.

Fuchsia's roadmap is driven by practical use cases arising from partner and product needs.

# Architecture

Zircon

FIDL

Components

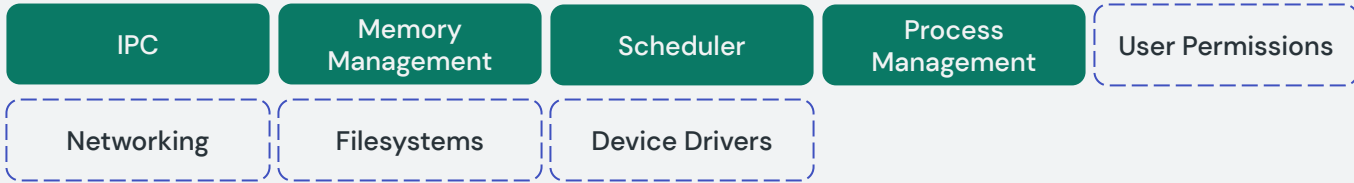
Packages

## Zircon is Fuchsia's capability-based, object-oriented kernel.

Zircon is a pragmatic, message-passing kernel—not a microkernel.

No global namespace, capabilities and resources are passed by handles rather than names.

Binary-stable driver interface allows kernel to be updated independently from drivers.



## FIDL is an IPC protocol enabling loose coupling between components.

FIDL allows language-agnostic interactions between software components written in different languages.

FIDL bindings are available for many languages including C++, Rust, Dart, and more.

Capabilities are expressed by FIDL protocols which promotes interchangeability and reusability.

# FIDL is the Fuchsia Interface Definition Language.

`fidlc` and  
`fidl-format` are  
tools for processing  
FIDL files.

```
library fuchsia.debugdata;

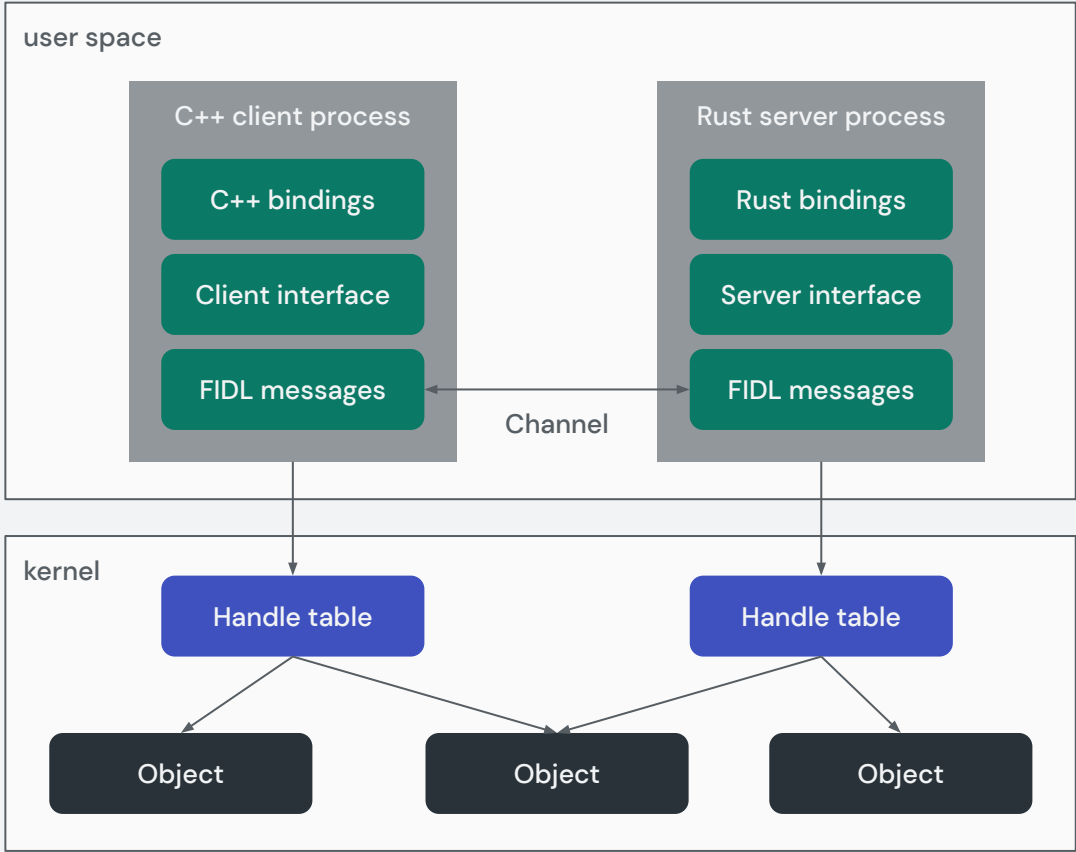
using zx;

const MAX_NAME uint64 = 1024;

protocol DebugDataVmoToken {};

@discoverable
protocol DebugData {
  /// Instrumentation runtime can publish VMO
  /// containing `data` identified by `data_sink`.
  Publish(resource struct {
    data_sink string:MAX_NAME;
    data zx.handle:VMO;
    vmo_token server_end:DebugDataVmoToken;
  });
  ...
};
```

debugdata.fidl



## Components are fundamental units of execution on Fuchsia.

Components are isolated containers for software which must explicitly declare all used and exposed capabilities.

All user-space software is a component, from drivers and system services to applications.

Components have the least privilege and access only to the information they need to do their jobs.

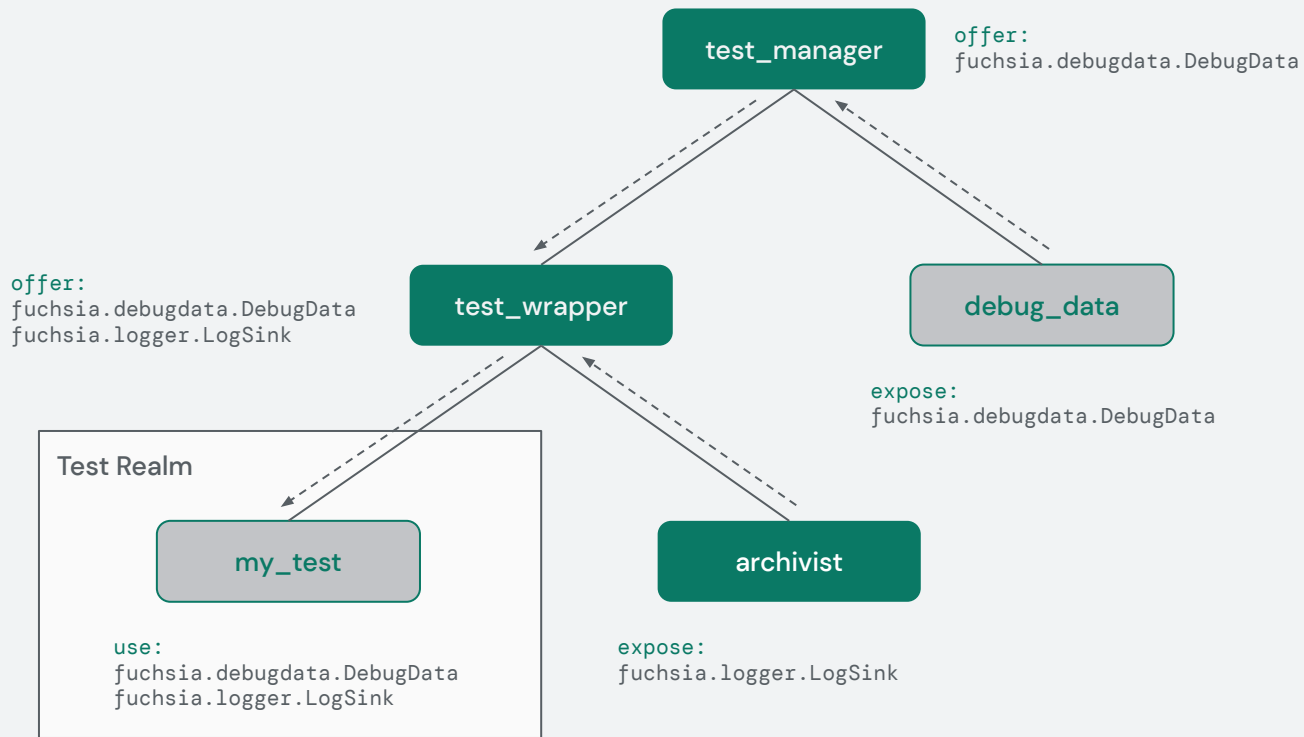


# CML is the Component Manifest Language.

cmc is a tool for  
processing CML  
files.

```
program: {  
  runner: "elf",  
  binary: "bin/my_test",  
  args: [ "--gtest_filter=..." ],  
},  
capabilities: [{  
  protocol: "fuchsia.test.Suite",  
}],  
use: [{  
  protocol: [  
    "fuchsia.debugdata.DebugData",  
    "fuchsia.logger.LogSink",  
  ]  
}],  
expose: [{  
  protocol: "fuchsia.test.Suite",  
  from: "self",  
}]
```

my\_test.cml

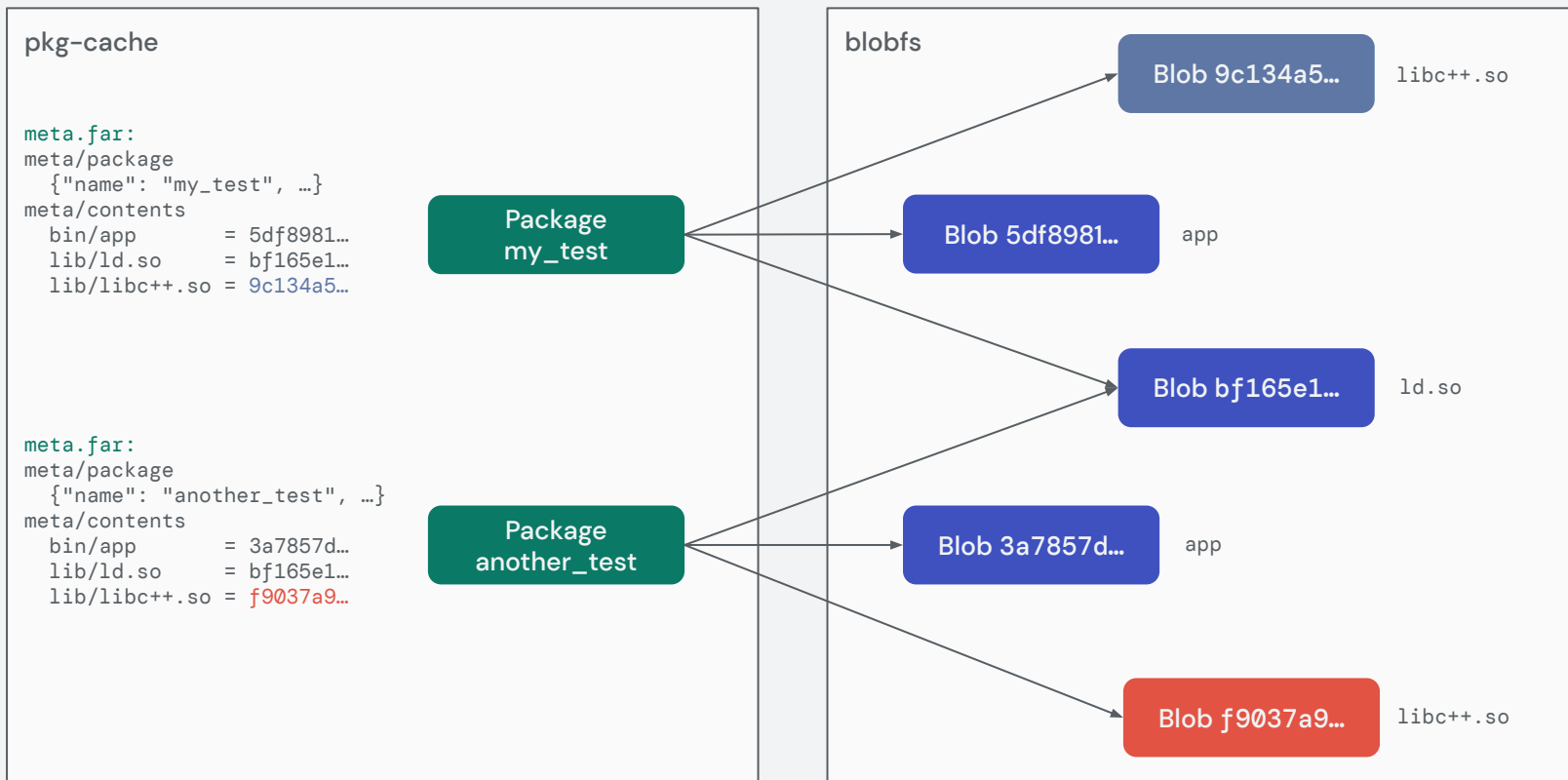


## Packages are units of distribution for software on Fuchsia.

Packages are hermetically sealed bundles of components and related asset files.

There's no global file system—each component only has visibility into its own package.

Packages are identified by URLs and can be downloaded, executed, and updated on demand as needed.



## Visit [fuchsia.dev](https://fuchsia.dev) to learn more.

*Fuchsia Fundamentals* section goes into far greater details including practical examples.

You can follow the platform evolution by tracking Fuchsia RFCs which inform the technical direction of the project.

Reach out via [discuss@fuchsia.dev](mailto:discuss@fuchsia.dev) if you have questions.

# How we built Fuchsia?

Developing an operating system from the ground up requires a significant engineering effort.

We decided to develop Fuchsia in the open from the very beginning, which meant we could not reuse a lot of the internal infrastructure and tooling Google has and had to build it ourselves.

## Fuchsia in numbers.

Most of the code lives in a single Git repository which has 115k+ commits and 700+ contributors to date totalling 5M+ LOC.

Fuchsia gets close to 500 commits from 150 contributors per week.

We rely on a number of third party dependencies, and the full checkout has over 100 Git repositories.

## Fuchsia uses a number of languages.

C++ is used for bootloader, kernel, and drivers.

Rust is used for most of the user-space components.

Dart is used primarily for UI (through Flutter).

Infrastructure uses primarily Go and Python.

Fuchsia SDK supports C++ and Dart.



## Every Fuchsia change has to be reviewed by a code owner.

Fuchsia uses Gerrit for code reviews with a set of plugins to aid the development.

We use both pre-submit and post-submit testing running a combination of unit, integration and end-to-end tests.

We perform a number of automated static checks upon change upload, including linting and formatting.

src/connectivity/network/tun/network-tun/device_adapter.cc		+113	-61	-	-	▼
src/connectivity/network/tun/network-tun/tun_device.cc		+5	-5	-	-	▼
src/connectivity/ppp/drivers/serial-ppp/serial-ppp.cc		+34	-22	-	-	▼
src/connectivity/ppp/drivers/serial-ppp/serial-ppp-test.cc		+69	-60	-	-	▼
		+1285	-848			

Tricum [Run 4749644677513216](#)

[FILE BUG](#)

Tryjobs Showing jobs from patchset 39. Refreshed at 2:17:44 PM.

[SHOW EXPERIMENTAL TRYJOBS](#)

- bringup.arm64-asan
- bringup.arm64-debug-build\_default
- bringup.arm64-debug-enable\_lock\_dep
- bringup.arm64-debug-no\_kernel\_debug
- bringup.arm64-gcc
- bringup.arm64-lto
- bringup.arm64-thinlto
- bringup.vim3-debug-build\_only
- bringup.x64-asan
- bringup.x64-debug
- bringup.x64-debug-enable\_lock\_dep
- bringup.x64-debug-no\_kernel\_debug
- bringup.x64-gcc
- bringup.x64-kasan\_sancov-build\_only
- bringup.x64-thinlto
- core.arm64-asan
- core.arm64-debug-build\_default
- core.arm64-debug-no\_opt-build\_only
- core.arm64-release
- core.x64-asan
- core.x64-debug-build\_default
- core.x64-debug-no\_opt-build\_only
- core.x64-fuzz\_asan-build\_only
- core.x64-fuzz\_ubsan-build\_only
- core.x64-host\_test\_only-mac
- core.x64-release
- core.x64-sdk-modular
- sdk-core-linux
- static-checks
- terminal.x64-release
- tricum
- workstation.x64-release-build\_only
- workstation.x64-release-e2e-isolated
- zbi\_tests-arm64
- zbi\_tests-arm64-asan
- zbi\_tests-arm64-gcc
- zbi\_tests-x64
- zbi\_tests-x64-asan
- zbi\_tests-x64-gcc

## Change Log

Show all entries (87 hidden)

[EXPAND ALL](#)

 **Nathan Mulcahey** Change destination moved from master to main Migrating to `main` branch

Patchset 13 | May 02 12:23 ▼

 **brunodalbo**  2 comments Ping for reviewz 😊

Patchset 17 | May 05 14:51 ▼

## Fuchsia makes use of static analysis.

We use `-Werror` `-Wall` `-Wextra` and we try adopt all new warnings as they are being introduced to Clang.

We use Clang-Tidy and have built a number of checks to enforce Fuchsia-specific C/C++ coding guidelines.

We have implemented a new Clang Static Analyzer checker to detect handle leaks/double closes.

```
ervice/session.h 1 comment || +15 -8 100% 100% MARK UNCHECKED ^ o` can immediately be reused.
153 bool CompleteRx(const RxFrameInfo& frame_info) __TA_REQUIRES_SHARED(parent_>control_lock())
154     __TA_REQUIRES(parent_>rx_lock());
om another session into one of this session's available rx
155 // Completes rx by copying the data from another session into one of this session's available rx
156 // buffers.
er, uint16_t owner_index, const rx_buffer_t* buff)
157 void CompleteRxWith(const Session& owner, const RxFrameInfo& frame_info)
158     __TA_REQUIRES_SHARED(parent_>control_lock()) __TA_REQUIRES(parent_>rx_lock());
ro_llock() __TA_REQUIRES(parent_>rx_lock());
159 // Copies data from a tx frame from another session into one of this session's available rx
other session into one of this session's available rx
160 // buffers.
, uint16_t owner_index) __TA_REQUIRES(parent_>rx_lock());
161 bool ListenFromTx(const Session& owner, uint16_t owner_index) __TA_REQUIRES(parent_>rx_lock());
them back to the session client.
162 // Commits pending rx buffers, sending them back to the session client.
>rx_lock());
163 void CommitRx() __TA_REQUIRES(parent_>rx_lock());
scribed to frame_type on port.
164 // Returns true iff the session is subscribed to frame_type on port.
ort, uint8_t frame_type)
165 bool IsSubscribedToFrameType(uint8_t port, uint8_t frame_type)
ro_llock());
166     __TA_REQUIRES_SHARED(parent_>control_lock());
167
+; }
168 inline void TxTaken() { in_flight_tx++; }
n_flight_tx-- != 0); }
169 inline void TxReturned() { ZX_ASSERT(in_flight_tx-- != 0); }
+; }
170 inline void RxTaken() { in_flight_rx++; }
171 inline bool RxReturned(size_t count = 1) {
```

Lint/ClangTidy [Run Details](#) May 07 ^

Tricium

readability-convert-member-functions-to-static: method 'RxReturned' can be made static

**NOT USEFUL PLEASE FIX**

```
172 ZX_ASSERT(in_flight_rx_.fetch_sub(count) >= count);
173     return rx_valid_;
174 }
175 inline void StopRx() { rx_valid_ = false; }
176 inline bool rx_valid() const { return rx_valid_; }
y() {
177 [[nodiscard]] inline bool ShouldDestroy() {
tx_ == 0) {
178     if (in_flight_rx_ == 0 && in_flight_tx_ == 0) {
179         bool expect = false;
180         // Only ever return true for ShouldDestroy once so the caller can schedule destruction
181         // asynchronously after ShouldDestroy returns true and have a guarantee that it won't be
182         // possible to schedule destruction for the same object twice.
183         return scheduled_destruction_.compare_exchange_strong(expect, true);
184     }
185     return false;
dDestroy once so the caller can schedule destruction
roy returns true and have a guarantee that it won't be
n for the same object twice.
are_exchange_strong(expect, true);
```

☆ Starred by 1 user

Owner: haowei@google.com
CC: phosek@google.com, mcgrathr@google.com, leonardchan@google.com, gulfem@google.com

Status: Fixed (Closed)

Components: Toolchain>Clang

Modified: Oct 8, 2021

ETA: ---

NextAction: ---

Pri: ---

Progress: ---

Related-Issues: ---

Severe: ---

StoryPoints: ---

type: Bug

Restrict-View-Google
Restrict-FlagSpam-CommunityManager
Restrict-FlagSpam-Committer

Your Hotlists: Update your hotlists

Issue 85893: Fuchsia clang canary builders failed with "error: use of bitwise '&' with boolean operands"

Reported by haowei@google.com on Mon, Oct 4, 2021, 10:10 AM PDT Project Member

Code Markdown

Only users with Google permission or issue reporter may view.

Upstream recently landed change https://reviews.llvm.org/D108003 when bitwise operators are used on bool operands.

There are currently cases like this in Fuchsia source code and they should be addressed.

Comment 1 by Git Watcher on Mon, Oct 4, 2021, 12:10 PM PDT Project Member

The following revision refers to this bug:
https://fuchsia.googlesource.com/third\_party/Vulkan-ValidationLayers/+7a19160bef49c43c0a7209a16512a10b3325354d

commit 7a19160bef49c43c0a7209a16512a10b3325354d
Author: Haowei Wu <haowei@google.com>
Date: Mon Oct 04 18:56:18 2021

Fix bitwise operators warning on bool operands

Clang upstream warns when bitwise operators are used on boolean operands. This patch fixes this issue.

Bug: 85893
Change-Id: I231aebf22fa3d679c23a6ca79f6a1f8a05878b21
Reviewed-on: https://fuchsia-review.googlesource.com/c/third\_party/Vulkan-ValidationLayers/+589409
Reviewed-by: Shai Barack <shayba@google.com>

[modify] https://fuchsia.googlesource.com/third\_party/Vulkan-ValidationLayers/+7a19160bef49c43c0a7209a16512a10b3325354d/layers/state\_tracker.c

Comment 2 by Git Watcher on Mon, Oct 4, 2021, 1:00 PM PDT Project Member

The following revision refers to this bug:
https://fuchsia.googlesource.com/fuchsia/+2a856d60bb27a520159dc0c6e51db59e681310cd

## Fuchsia uses source-based code coverage.

We collect source-based code coverage for C++ and Rust in both pre-submit and post-submit testing.

We make incremental and absolute coverage available to developers directly in Gerrit and in code search.

We warn developers when the coverage of their changes is too low.



```
4
5 #include "backtrace.h"
6
7 #include "threads_impl.h"
8
9 namespace __libc_sanitizer {
10
11 size_t BacktraceByFramePointer(cpp20::span<uintptr_t> pcs) {
12     struct FramePointer {
13         const FramePointer* fp;
14         uintptr_t pc;
15     };
16
17     auto on_stack = [&stack = __pthread_self()->safe_stack](const FramePointer* fp) -> bool {
18         uintptr_t address = reinterpret_cast<uintptr_t>(fp);
19         return address >= reinterpret_cast<uintptr_t>(stack.iov_base) &&
20             address < reinterpret_cast<uintptr_t>(stack.iov_base) + stack.iov_len;
21     };
22
23     uintptr_t ra = reinterpret_cast<uintptr_t>(__builtin_return_address(0));
24     auto fp = reinterpret_cast<const FramePointer*>(__builtin_frame_address(0));
25     size_t i = 0;
26     while (i < pcs.size() && on_stack(fp) && fp->pc != 0) {
27         if (i == 0 && fp->pc != ra) {
28             pcs[i++] = ra;
29         } else {
30             pcs[i++] = fp->pc;
31             fp = fp->fp;
32         }
33     }
34     if (i == 0 && i < pcs.size()) {
35         pcs[i++] = ra;
36     }
37
38     return i;
39 }
40
41 #if __has_feature(shadow_call_stack)
42
43 namespace {
44
```

## Fuchsia sanitizers.

We sanitize everything including the kernel and low-level parts like the C library, and even our host tools.

All pre-submit and post-submit tests include (K)ASan, LSan and UBSan; we're bringing up HWASan now.

We encourage the use of coverage-guided fuzz testing and we continuously fuzz the kernel using syzkaller.



## Fuchsia eagerly adopts new security hardening features.

We default to PIE and use (K)ASLR everywhere.

We use automatic variable initialization for all C/C++ to prevent the of use undefined memory.

We use SafeStack (on x86-64) and ShadowCallStack (on AArch64) to protect against return address overwrites.

We use Scudo as the default system allocator to protect against heap based vulnerabilities.

**Behind the  
scenes**

## We chose LLVM because it matches our goals.

We aim to provide a modern, permissively licensed, self-contained toolchain with a complete set of tools.

We want to support a broad range of host and target platforms within a single toolchain.

We want to leverage the ecosystem of tooling like static analysis, linting, formatting, code coverage, sanitizers, etc.

## We maintain our own Clang and Rust toolchains for Fuchsia.

We want to provide the same exact experience on every supported host platform.

We follow the "live at HEAD" model and release new toolchains on a ~weekly cadence.

We don't carry any downstream patches, we develop new features in upstream and adopt them once available.

# Fuchsia Clang Toolchain

A complete C/C++ toolchain distribution that includes a number of LLVM tools and runtime libraries.

We use it to build all of Fuchsia—everything from bootloader to kernel, system libraries, user applications and even host tools—as well as other related projects such as Pigweed, Dart and Flutter.

It is used by hundreds of developers—both inside and outside of Google—and thousands of automated builders.

See "[Fuchsia Clang Toolchain](#)" by Petr Hosek

## LLVM wasn't a complete toolchain when Fuchsia started.

There are tools and runtimes developers take for granted but we had to build those, sometimes from scratch.

From the beginning, we were co-developing the OS and the toolchain, and often adapting the platform to the toolchain.

This made it easier for us to adopt LLVM components even before they were ready for broader adoption.

## Support for building the complete cross-compiling toolchain

We needed a way to build runtimes in the right order with the just-built Clang and LLVM tools.

We needed a way to install runtimes for all supported targets side-by-side.

Solving this required a large a number of changes to the CMake build and Clang driver to support cross-compiling runtimes for multiple targets (and multilibs).

See "[LLVM Runtimes Build](#)" by Petr Hosek

## Fuchsia is unlike existing operating systems.

Many runtimes make assumptions about the operating system, like always having a filesystem as a way to export collected data.

This may not work for programs like filesystems.

```
OutFile = fopen(OutName, "ab"); // returns NULL
if (!OutFile) return -1;
...
for (I = 0; I < NumIOVecs; I++)
    ...
    fwrite(IOVecs[I].Data,
           IOVecs[I].ElmSize,
           IOVecs[I].NumElm,
           OutFile);
...
fclose(OutFile);
```

InstrProfilingFile.c

```
$ clang ... -lfdio # use fuchsia.io implementation
```



## Fuchsia's debugdata protocol as the native solution.

This is a simple  
protocol used solely  
for collecting  
runtime data.

This protocol is used  
by the profile,  
sanitizer coverage  
and XRay runtimes  
(and likely more in  
the future).

```
library fuchsia.debugdata;

using zx;

const MAX_NAME uint64 = 1024;

protocol DebugDataVmoToken {};

@discoverable
protocol DebugData {
  /// Instrumentation runtime can publish VMO
  /// containing `data` identified by `data_sink`.
  Publish(resource struct {
    data_sink string:MAX_NAME;
    data zx.handle:VMO;
    vmo_token server_end:DebugDataVmoToken;
  });
  ...
};
```

debugdata.fidl

## Porting the existing compiler-rt runtimes to Fuchsia.

We had to factor out the filesystem related code to allow the use of mechanisms like `fuchsia.debugdata`.

In addition to implementing the OS abstractions, we also had to do a number of refactorings.

We had to refactor the sanitizer runtime to implement support for offline symbolization.

## Implementing sanitizer support in Fuchsia's C library.

We provided internal API which simplified the sanitizer implementation and obviated the need for interceptors.

This allows instrumenting of the C library itself as well as other runtime libraries like libc++.

The SafeStack and ShadowCallStack is supported directly by Fuchsia's C library.

## We are adopting LLVM libc for Fuchsia.

We have been involved in the LLVM libc proposal and have been working closely with the LLVM libc team.

We are evolving our C library and incrementally replacing existing parts with LLVM libc counterparts.

## We care about resource usage, primarily binary size and memory.

We use `--icf=all` and `--gc-sections` and we addressed a number of related issues in LLVM and LLD.

We have adopted RELR relocation packing shortly after it became supported in LLVM. We also use REL relocations (instead of RELA) on all architectures.

We are experimenting with the ML inliner, using a model trained on the Fuchsia codebase to further reduce size.

## Relative VTables C++ ABI

We replaced the 64-bit function pointers in vtables with 32-bit PC-relative offsets to these function pointers.

This allows vtables to be moved into read-only data sections, which allows them to be shared between processes.

This became the default C++ ABI on Fuchsia but it can be used on any system that supports Itanium C++ ABI.

See "Relative VTables in C++" by  
Leonard Chan

# LLVM IFS

The LLVM IFS tool produces textual description of a shared library ABI and a linkable ELF shared object stubs.

The textual description makes ABI breaking changes easier to spot and review.

The use of ELF stubs reduces the incremental build time as we can avoid relinking binaries if the ABI of their dependents has not changed.

See "Introduction to LLVM IFS and its usages in Fuchsia build system" by Haowei Wu

## There's a lot more...

### **llvm-objcopy and llvm-strip**

drop-in replacement for GNU objcopy and strip

### **address\_space and noderef attributes**

support for Sparse checker warnings in Clang

### **crtbegin.o and crtend.o**

compiler-rt replacements for libgcc counterparts

### **clang-tidy**

Fuchsia-specific checks, some of which were later generalized



**...with more coming.**

**Fixed-Point Arithmetic C99 extension**

support for precise decimal calculations

**clang-doc**

C and C++ documentation generator

**llvm-debuginfod**

LLVM based implementation of the debuginfod protocol

**clang-misexpect**

verifying `__builtin_expect` annotations

# What's next?

We have shipped the first version of Fuchsia to millions of devices but we are not nearly done yet.

In the future, we want to evolve Fuchsia into a more generally usable operating system.

## Support for running LLVM runtime tests on Fuchsia.

We do not yet support running the existing LLVM tests on Fuchsia and rely solely on end-to-end testing.

We exercise the code generated by LLVM and we check binaries that include LLVM runtime libraries when testing Fuchsia.

This requires building the entire system and running all our tests which increases cycle time.

## Support for cross-target testing in lit.

We need a way to support executing tests on targets other than the host in lit to start running tests on Fuchsia.

Some of the runtimes support remotely executing tests, but it is implemented in an ad-hoc way, it is inefficient and difficult to generalize.

## Many lit tests use complex logic.

`%run` substitution can be used to offload test to another target.

```
// RUN: %clangxx_asan -O0 %s -o %t
// RUN: not %run %t 2>&1 | \
// RUN:   FileCheck --check-prefix=CHECK-CRASH %s

// RUN: echo "interceptor_name:strlen" > %t.sup
// RUN: %env_asan_opts=suppressions="'%t.sup' \
// RUN:   %run %t 2>&1 | \
// RUN:   FileCheck --check-prefix=CHECK-IGNORE %s

#include <stdlib.h>
#include <string.h>

int main() {
    char *a = (char *)malloc(6);
    free(a);
    return strlen(a); // BOOM
}
```

suppressions-interceptor.cpp

## Make Fuchsia an officially supported LLVM platform.

LLVM libraries support UNIX and Windows, but Fuchsia is neither of those and emulation may not be the most optimal solution.

Some interfaces make assumption about the underlying system, most commonly the use of file descriptors and sockets.

## Building software on Fuchsia.

We want to eventually support software development on Fuchsia, including compilation and linking.

This may require changes to the compiler organization in order to achieve optimal performance on Fuchsia.

See "[Optimizing builds on Windows: some practical considerations](#)" by Alexandre Ganea

## Designing a new architecture for building software.

The traditional UNIX model for building software is not the best fit for Fuchsia which prefers a service architecture.

Using a service architecture will be a better fit for Fuchsia, but it may also improve performance on other platforms.

This may require significant refactoring throughout LLVM and introduction of new abstractions. We may also need a new generation of build systems.

See "[A New Architecture for Building Software](#)" by Daniel Dunbar



## Compiler as a service

Using Clang as a service through a FIDL protocol as a Fuchsia native solution.

```
using llvm.clang;

type CompileInput = struct { ... };
type CompileOutput = struct { ... };

@discoverable
protocol Compiler {
    Compile(struct {
        input CompileInput;
    }) -> (struct {
        output CompileOutput;
    });
};
```

clang.fidl

# Q&A

Thanks to everyone who contributed to this effort over the years, we wouldn't have made it without your help!

Aaron Green, Annie Cherkaev, Farah Hariri, Farid Molazem Tabrizi, Gulfem Savrun Yeniceri, Gábor Horváth, Haowei Wu, Jake Ehrlich, Jayson Yan, Julie Hockett, Kareem Khazem, Leonard Chan, Marco Vanotti, Noah Shutty, Paul Kirth, Roland McGrath, and many others...

