# An OS for the Next 100 Years

## Introduction

Turn your computer off and put it in the closet. 100 years from now someone finds it and powers it up.

Does it boot? Probably.

Do your applications still run? Maybe..

Can you install new software on it? Not a chance.

## Introduction

What are the limitations of current operating systems and software in general that make this impossible?

What would an operating system have to look like to make this (admittedly pathological) scenario possible?

## Limitations

Software is stable when its dependencies are stable.

- What dependencies?
  - Dynamically linked (shared) libraries
  - Operating system interface, (like system calls)
  - Library/OS ABI

## Limitations: Libraries

If we can statically link libraries, why do we dynamically link them and introduce possible instability?

# Limitations: Libraries

A static library that is used by multiple processes is duplicated in memory.

One copy of a dynamically linked library is shared by all processes that use it.

If a static library changes, applications that use it must be recompiled to use the new version.

One copy of a dynamically linked library is shared by all processes that use it.

One copy of a dynamically linked library exists on the system. This leads to smaller binaries.

Updates to a shared library can be used without recompiling the application that uses it.

# Limitations: Libraries

Sounds great! What's the problem exactly?

- Updates to libraries are often not *backwards compatible*.
- For an application that depends on a shared library to be "stable", the **exact** version of the library **must** exist on the system in the ***expected location*** **at all times**.

## Limitations: OS Interface, ABI

- Changing an OS ABI after the fact is an almost intractable problem. Backwards compatibility is important (mandatory?)
- How does the ABI definition impact programming language interface?
- What decisions are you locked into? Performance implications?
- How about sharing memory? Security? (meltdown)

## Examples

## Windows

| | |
|---|---|
| Strong history of backwards compatibility [1] | : ) |
| DLL-hell | : ( |
| Microsoft | : ( |

[1] http://ptgmedia.pearsoncmg.com/images/9780321440303/samplechapter/Chen_bonus_ch01.pdf

## Linux (Traditional)

## NixOS

Great ideas:

- Isolating build environments
- Identifying packages by hash to ensure that upgrading / installing one package doesn't break another package

Not immune to dynamic linking issues:

*https://www.johnbcoughlin.com/posts/nix-dynamic-linking*

It can only go so far when building upon the traditional systems with dynamically linked dependencies.

## Microkernel Design

- Smaller, more stable and priveleged kernel
- Fault tolerance
- Rapid development/research of system-level components
- But microkernels need way more system calls and contexts are slow...

## IPC

- Message-based IPC can be accomplished without context switching into kernel space.
  - Exception: setting up shared buffers
- If device drivers are userspace applications, then operating on those devices can remain outside of the kernel completely.
  - (Depending on how the device is commanded)

# Message Types:

- Direct:
  - Similar to UNIX pipe; FIFO in shared memory.
- Unknown Recipient:
  - Message is sent to *someone* who claims to handle a certain message type
- Broadcast:
  - Send to all recipients waiting on message type

```
                      --------            ---------             -------            --------
              (0) - > | INIT | - > (0) -> | SHELL | -> (1)      | CAT |  -> (2) -> | GREP | -> (1)
                      --------            ---------     \       -------            --------     \
                         |                                |                                       |
                         v                                |                                       |
                        (1) <-----------------------------/---------------------------------------/
```

Lane Tables (lane ID, shared memory buffer ID)

| INIT | SHELL | CAT | GREP |
|---|---|---|---|
| 0, 0 | 0, 0 | 0, - | 0, 2 |
| 1, 1 | 1, 1 | 1, 2 | 1, 1 |
| 2, - | 2, - | 2, - | 2, - |

```
# direct messaging
# cat "file" | grep "pattern"
S := me();
C := pstart("cat");
G := pstart("grep");
lanemap(C:1, G:0);
lanemap(G:1, S:1);
```

```
#unknown recipient: producer
lane := getlane("notify");
msg_send(lane, "notification");
```

```
#unknown recipient: consumer
lane := getlane("notify");
msg := msg_receive(lane);
```

## Non-Hierarchical Filesystem

- Tags
- Query based
- `open`-like system call can return multiple results
- File copies to "another universe" or an archive?