HPC Operating System Design

Rolf Riesen

26 January 2016

Legal Disclaimer

Introduction

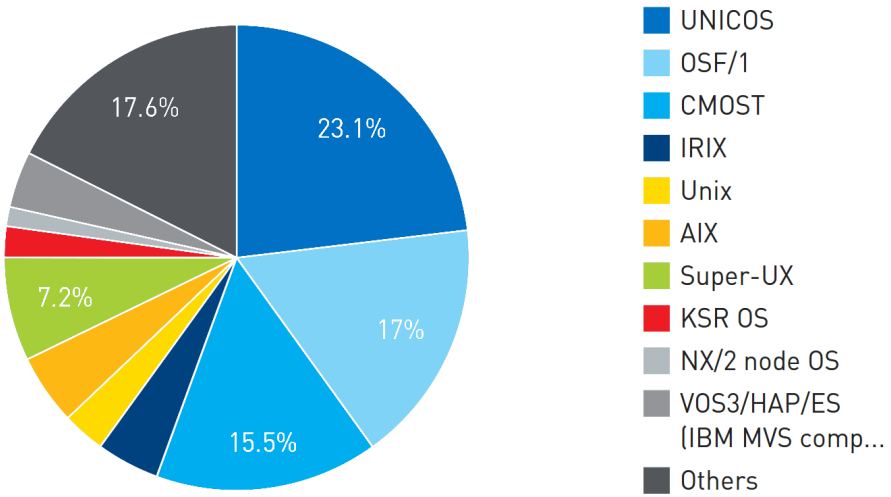
Overview

- *mOS* (Multi-OS) is a research project at Intel
- Aimed at the very top-end of HPC machines
  - Extreme scale systems: tens to hundreds millions of threads
- Goal is to provide a solution beyond exa-scale
- Also, an OS that can be easily adapted to new types of hardware
  - Try out hardware ideas and quickly support them in *mOS*
- An OS that lets us provide support for new runtimes quickly
  - Future runtimes may want more control of the hardware

OS diversity is vanishing

Used to have lightweight kernels (LWK)

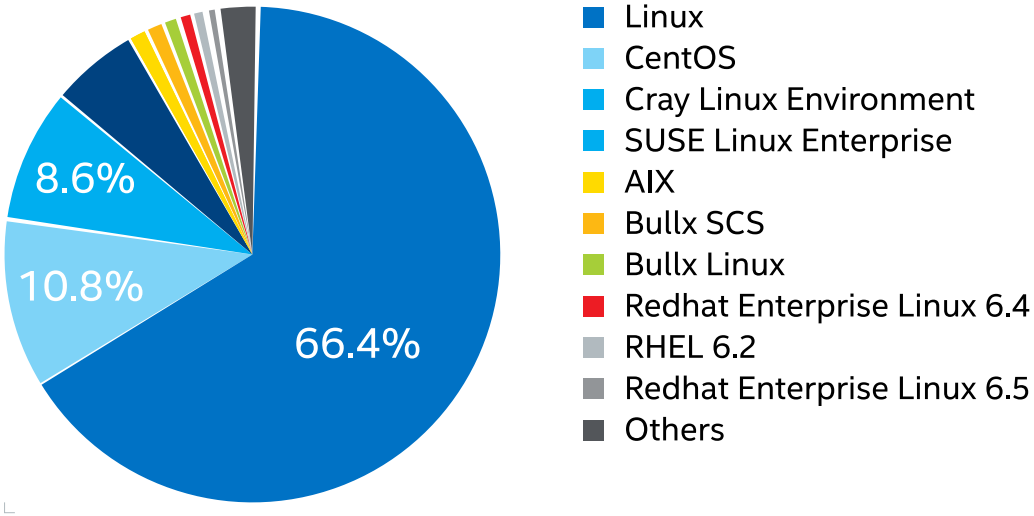
- Special purpose OSes with limited functionality
  - E.g., SUNMOS, Puma, Cougar, CNK*, Catamount*
  - Scaled well, high perf., but difficult to use; not compatible
  - Standard tools did not work



OS diversity on the 1994 Top 500 list

Linux dominates Top 500 list

- Familiar to developers from their laptops and desktops
- Has the features requested by users and tool makers
  - New runtime systems and tools target Linux
  - Not just Linux system calls, also `/proc`, `/sys`, `sched_setaffinity()`, ...

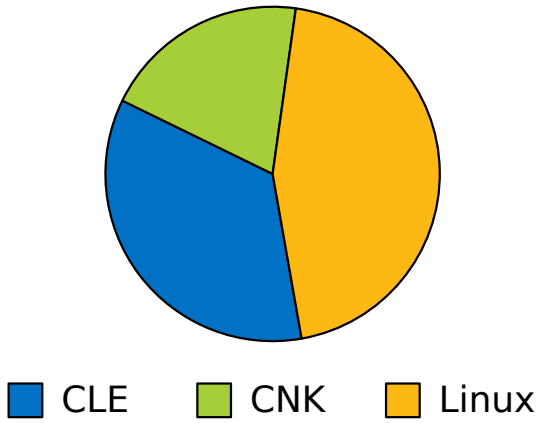


OS diversity on the Nov. 2015 Top 500 list (% of all systems)

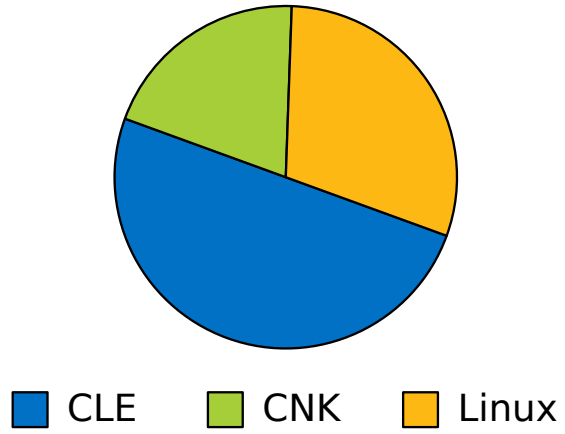
Why not just Linux?

Linux is important, but we also need OS specialization!

- 55% of OSes in top 20 are specialized
- 70% of OSes in top 10 are specialized



Top 20 OSes in Nov 2015



Top 10 OSes in Nov 2015

Why an LWK?

LWK properties are important and beneficial

- Nimbleness
  - Adapt to new, novel hardware features
  - Quickly implement new memory management strategies
  - Adapt and specialize to new programming models
- Get OS overhead out of the way; provide what hardware can do
- Simplify
  - App developers should concentrate on performance and scalability; not OS quirks and unpredictability
- Make OS research on a real system easy
  - Not a toy OS for experimentation
  - Don't have to learn all of Linux to experiment

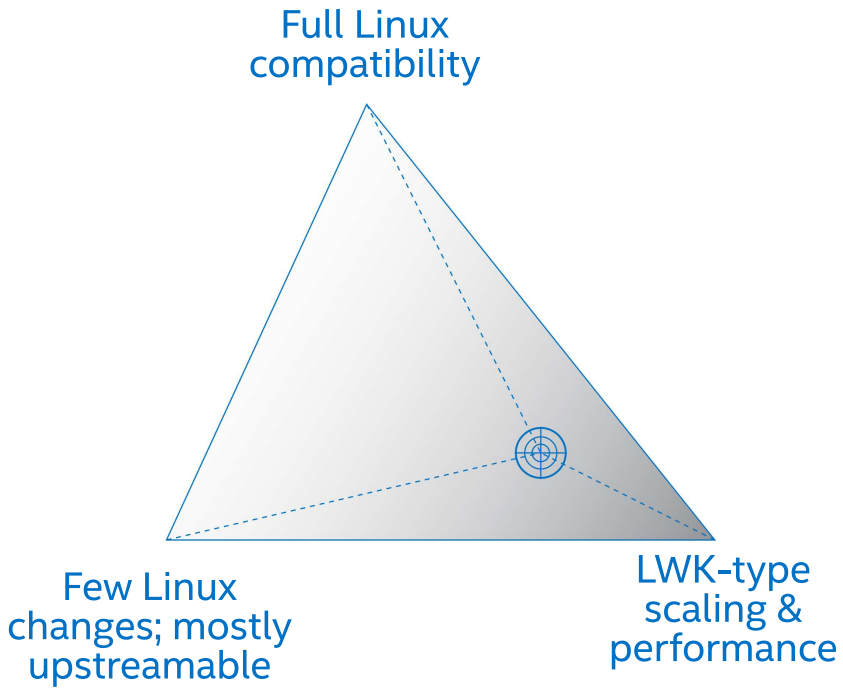
Extreme simplification

- Process management
  - Cooperative, non-preemptive task scheduling
  - Single, or few, task per logical CPU
- Memory management
  - Limited paging, no swap, “pinned” memory
  - physically contiguous memory
- Omit functionality; rely on FWK
- Space sharing
  - Use massive hardware parallelism, not time sharing
- Code and binary size
  - One person can understand and remember the entire LWK

Architecture and Design

Top-level requirements

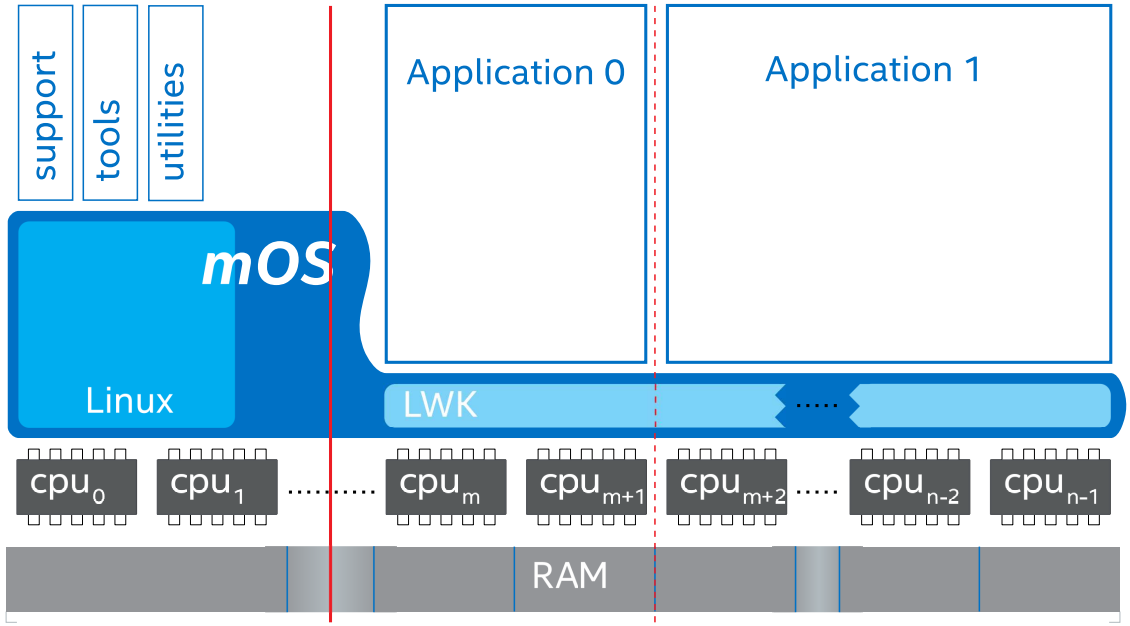
1. Foremost is for *mOS* to scale and deliver the parallel performance needed in an extreme-scale system.
2. *mOS* cannot exist unless we can implement and maintain it.
3. Linux compatibility is also important, but comes in after the performance and scalability goals have been met.



High-level architecture

To get the best of both worlds, run both OSes!

- Dedicate a few cores in a many-core system to Linux
- The remaining cores run compute intensive processes on LWK

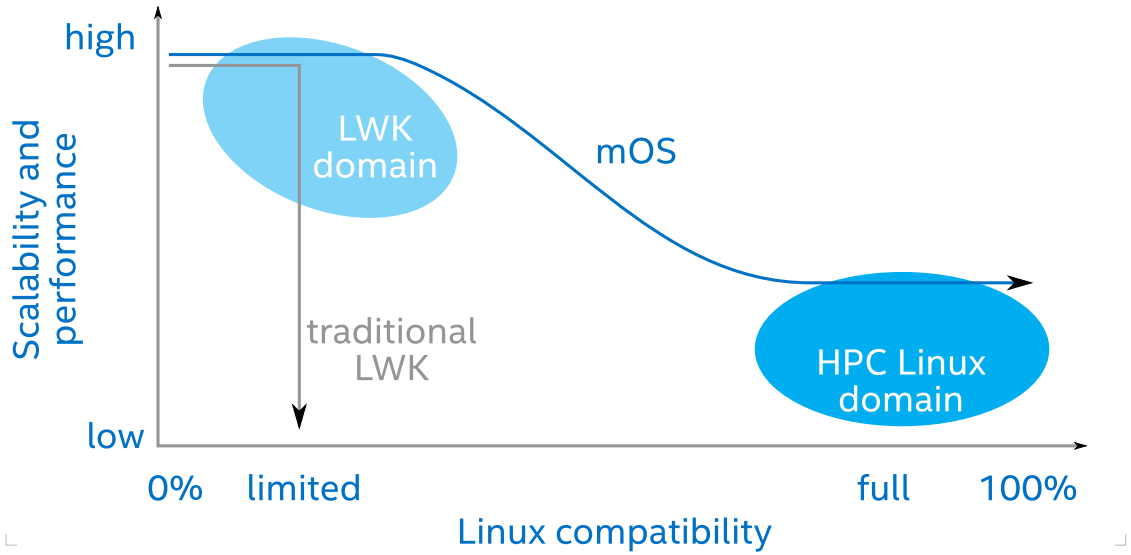


Lightweight kernels and Linux

In the past it was possible to achieve performance and sclability. Or, one could run Linux. But not both.

With an architecture like *mOS*, it is possible to have a more gradual path from the upper left LWK corner to the lower right FWK corner.

An application's choice of which features it wants to use, influences the overall performance and scalability.



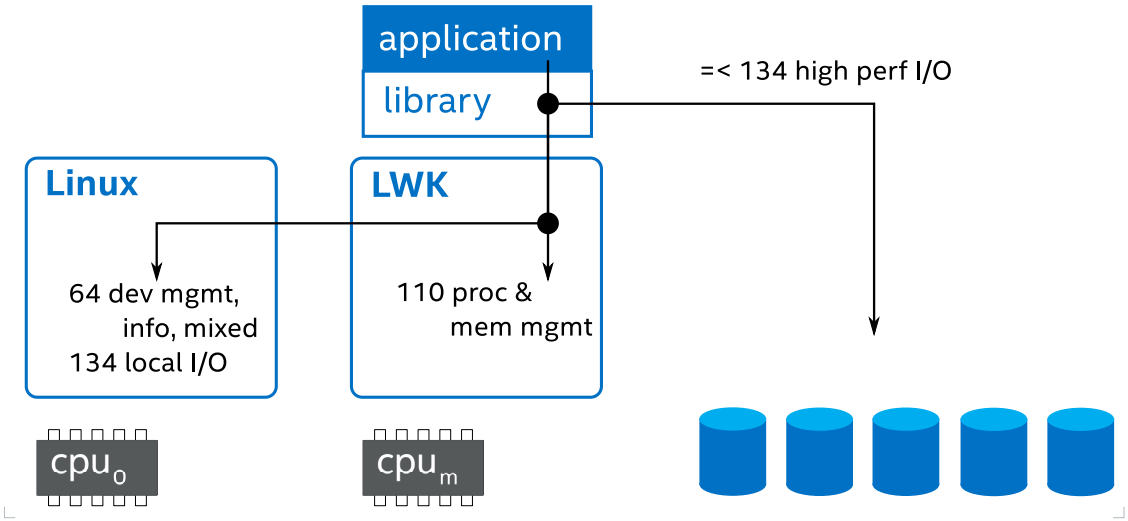
Different design goals

mOS

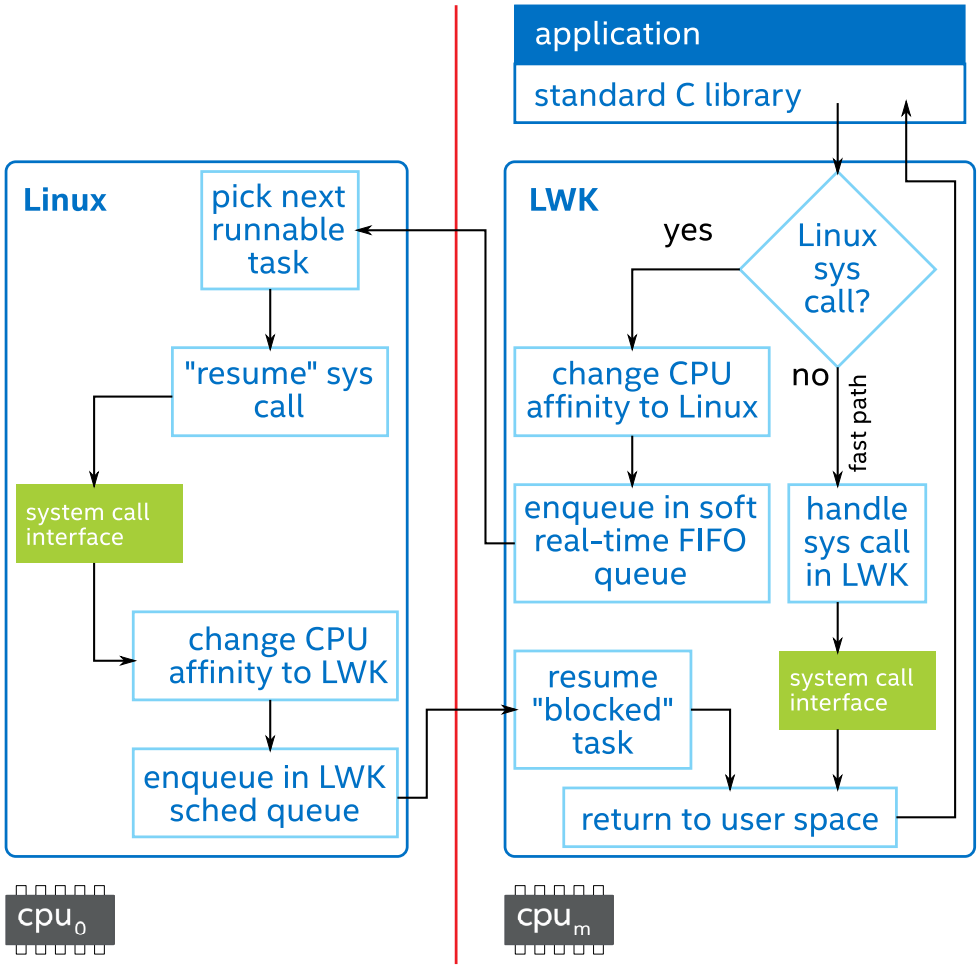
	LWK	FWK
Target	massively parallel systems	laptops, desktops, servers
Support	scalable applications	everything under the sun
Development environment for	parallel applications	business, games, commerce, etc.
Emphasis	efficiency	functionality
Resources	maximize use	fair sharing, QoS
Time to completion	minimal	when needed

System call triage

- Three “places” to handle system calls:
  - Linux, LWK, off-node
- LWK handles proc. and mem. mgmt: 110 calls!
- Excluding local I/O, Linux only handles 64 calls



Direct model implementation



System call path

- Who handles sys call?
- If Linux
  - migrate task to Linux CPU
  - only for duration of sys call
- If LWK
  - handle on local CPU

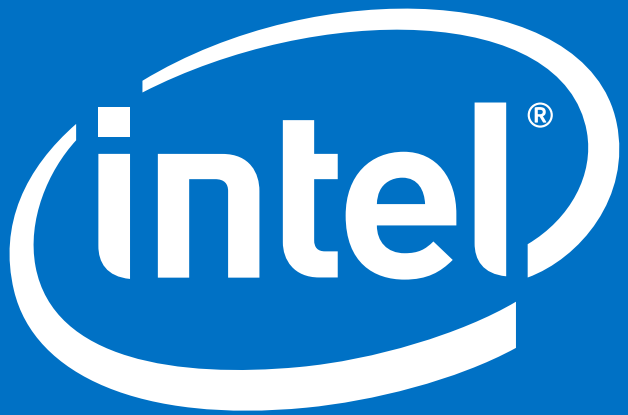
People involved

Architecture, design, implementation, and testing:

- Rolf Riesen
- Tom Musta
- John Attinella
- David Van Dresser
- Andrew Tauferner
- Evan Powers

Vision, management, and guidance:

- Robert W. Wisniewski
- Lance Shuler
- Todd Inglett
- Pardo Keppel
- Thomas Spelce


intel