University of Glasgow | School of Computing Science

Message Passing

Advanced Operating Systems
Tutorial 5

Tutorial Outline

- Review of Lectured Material
- Discussion: Barrelfish and multi-kernel systems
- Programming exercise

Review of Lectured Material

- Implications of multicore systems
  - Hardware trends; NUMA and heterogeneity in multicore systems
  - Challenges of NUMA systems – is a shared memory model appropriate?
  - Multi-kernel systems – distributed operating systems for multicore
- Message passing systems
  - Limitations of threads and lock-based concurrency
  - Multicore memory models; composition of lock-based code
  - Concepts of message passing systems
    - Interaction models; communication and the type system; naming communications
    - Message handling; immutability; linear types; use of an exchange heap
    - Pattern matching and state machines
    - Error handling; let-it-crash philosophy; supervision hierarchies; case study
  - Erlang and Scala+Akka as examples

Key Points

- Understand problems of scaling multicore systems while maintaining a shared memory programming model
  - The multi-kernel operating system model
  - The message passing programming model
- Reflect on the suitability of message passing as a concurrency primitive for future systems
  - Advantages and disadvantages compared to lock-based concurrency with shared mutable state

Discussion: Barrelfish

- A. Baumann *et al*, “The Multikernel: A new OS architecture for scalable multicore systems”, Proc. ACM SOSP 2009. DOI:10.1145/1629575.1629579
- Is the premise that messages are more suitable than shared memory for future systems reasonable?
- Does it make sense to run a distributed operating system on the cores of a single hardware device?
- Where is the boundary for a Barrelfish-like system?
  - Distinction between a distributed multi-kernel and a distributed system of networked computers?
- Barrelfish is clearly an extreme: a shared-nothing system implemented on a hardware platform that permits some efficient sharing
  - Is it a desirable extreme?
  - Current systems sit at the opposite extreme – shared everything, despite increasingly separate hardware resources



The Multikernel: A new OS architecture for scalable multicore systems

Andrew Baumann*, Paul Barham†, Pierre-Evariste Dagand‡, Tim Harris†, Rebecca Isaacs†, Simon Peter*, Timothy Roscoe*, Adrian Schüpbach*, and Akhilesh Singhania*

*Systems Group, ETH Zurich

†Microsoft Research, Cambridge ‡ENS Cachan Bretagne

Abstract

Commodity computer systems contain more and more processor cores and exhibit increasingly diverse architectural tradeoffs, including memory hierarchies, interconnects, instruction sets and variants, and IO configurations. Previous high-performance computing systems have scaled in specific cases, but the dynamic nature of modern client and server workloads, coupled with the impossibility of statically optimizing an OS for all workloads and hardware variants pose serious challenges for operating system structures.

We argue that the challenge of future multicore hardware is best met by embracing the networked nature of the machine, rethinking OS architecture using ideas from distributed systems. We investigate a new OS structure, the *multikernel*, that treats the machine as a network of independent cores, assumes no inter-core sharing at the lowest level, and moves traditional OS functionality to a distributed system of processes that communicate via message-passing.

We have implemented a multikernel OS to show that the approach is promising, and we describe how traditional scalability problems for operating systems (such as memory management) can be effectively recast using messages and can exploit insights from distributed systems and networking. An evaluation of our prototype on multicore systems shows that, even on present-day machines, the performance of a multikernel is comparable with a conventional OS, and can scale better to support future hardware.

Figure 1: The multikernel model.

1 Introduction

Computer hardware is changing and diversifying faster than system software. A diverse mix of cores, caches, interconnect links, IO devices and accelerators, combined with increasing core counts, leads to substantial scalability and correctness challenges for OS designers.

Such hardware, while in some regards similar to earlier parallel systems, is new in the general-purpose computing domain. We increasingly find multicore systems in a variety of environments ranging from personal computing platforms to data centers, with workloads that are less predictable, and often more OS-intensive, than traditional high-performance computing applications. It is no longer acceptable (or useful) to tune a general-purpose OS design for a particular hardware model: the deployed hardware varies wildly, and optimizations become obsolete after a few years when new hardware arrives.

Moreover, these optimizations involve tradeoffs specific to hardware parameters such as the cache hierarchy, the memory consistency model, and relative costs of local and remote cache access, and so are not portable between different hardware types. Often, they are not even applicable to future generations of the same architecture. Typically, because of these difficulties, a scalability problem must affect a substantial group of users before it will receive developer attention.

We attribute these engineering difficulties to the basic structure of a shared-memory kernel with data structures protected by locks, and in this paper we argue for rethinking the structure of the OS as a distributed system of functional units communicating via explicit mes-

1

Programming Exercise

- Exercise 3 now available
  - Aim – to explore the ease of use of message passing programming for non-expert programmers