CPS 510: Grad OS

Jeff Chase
January 8, 2015
About CPS 510

• **CPS 510 is about operating systems research**
  • You will read a lot of old and new papers
  • You will perform a semester-long research project

• **What CPS 510 is not about**
  • **Learning** basic operating systems concepts
  • You should know this already

• **Who should take it?**
  • Graduate students and smart undergrads
About this offering of CPS 510

• **Syllabus and reading list are online**
  • 12-14 papers for you to read and digest...and some others
  • Reading list is subject to “adjustment”
  • “Less than one paper per class”

• **Grading**
  • Exams (50%): one midterm and a final
  • Projects and labs (50%)
  • “Other factors”: adjustments up to half a letter grade
    • Participation and engagement
    • Bring questions and opinions on the reading to class
    • What will you say if I ask you?
    • Read one “classic paper” of your choice and post a short report
About exams

• Exams focus on concepts in the papers, as discussed in class.
  • motivation and lessons
  • what, why, how
  • tools, new concepts, philosophy
  • background concepts are “fair game”
Over the past two decades, there has been a huge amount of innovation in both the principles and practice of operating systems. Over the same period, the core ideas in a modern operating system — protection, concurrency, virtualization, resource allocation, and reliable storage — have become widely applied throughout computer science. Whether you get a job at Facebook, Google, Microsoft, or any other leading-edge technology company, it is impossible to build resilient, secure, and flexible computer systems without the ability to apply operating systems concepts in a variety of settings. ...Because operating systems concepts are among the most difficult in computer science, a top to bottom approach is the only way to really understand and master this important material.
About reading papers

• Older papers: introduce the “canon”
  • “A collection or list of sacred works”

• Newer papers: advances and research topics
  • It’s OK to disagree!

• Papers are different from textbooks. They:
  • Report contributions and set them context.
  • Explain methodology in sufficient detail for the reader to accept and even reproduce the results.
  • Present multiple levels of detail: some that matter, and some that don’t.
  • Are dressed to impress.
  • Are products of their time.

• It is not necessary to learn every detail!
Abstractions, hardware reality

Applications

Threads
Virtual Memory
Files, web

Hardware

Atomic Test/Set
Page Tables
Disk, NIC

Hardware

Programs

OS
An Introduction to Programming with C# Threads

Andrew D. Birrell

This paper provides an introduction to writing concurrent programs with “threads”. A threads facility allows you to write programs with multiple simultaneous points of execution, synchronizing through shared memory. The paper describes the basic thread and synchronization primitives, then for each primitive provides a tutorial on how to use it. The tutorial sections provide advice on the best ways to use the primitives, give warnings about what can go wrong and offer hints about how to avoid these pitfalls. The paper is aimed at experienced programmers who want to acquire practical expertise in writing concurrent programs. The programming language used is C#, but most of the tutorial applies equally well to other languages with thread support, such as Java.
public void AcquireExclusive() {
    lock (this) {
        if (wQueue == null) wQueue = new CV(this);
        writeWaiters++;
        while (i != 0) wQueue.Wait();
        writeWaiters--;
        i = -1;
    }
}

public void AcquireShared() {
    lock (this) {
        readWaiters++;
        if (writeWaiters > 0) {
            wQueue.Pulse();
            Monitor.Wait(this);
        }
        while (i < 0) Monitor.Wait(this);
        readWaiters--;
        i++;
    }
}
Performance of Firefly RPC

MICHAEL D. SCHROEDER and MICHAEL BURROWS
Digital Equipment Corporation

In this paper we report on the performance of the remote procedure call (RPC) implementation for the Firefly multiprocessor and analyze the implementation to account precisely for all measured latency. From the analysis and measurements, we estimate how much faster RPC could be if certain improvements were made. The elapsed time for an intermachine call to a remote procedure that accepts no arguments and produces no results is 2.66 ms. The elapsed time for an RPC that has a single 1440-byte result (the maximum result that will fit in a single packet) is 6.35 ms. Maximum intermachine throughput of application program data using RPC is 4.65 Mbits/s, achieved with four threads making parallel RPCs that return the maximum-size result that fits in a single RPC result packet. CPU utilization at maximum throughput is about 1.2 CPU seconds per second on the calling machine and a little less on the server. These measurements are for RPCs from user space on one machine to user space on another, using the installed system and a 10 Mbit/s Ethernet. The RPC packet exchange protocol is built on IP/UDP, and the times include calculating and verifying UDP checksums. The Fireflies used in the tests had 5 MicroVAX II processors and a DEQNA Ethernet controller.
RPC
First, a little philosophy

• **Structure and Interpretation of Computer Programs**
  • Harold Abelson and Gerald Jay Sussman
  • Longtime book for MIT’s first course in CS

“Underlying our approach to this subject is our conviction that ‘computer science’ is not a science and that its significance has little to do with computers...

Mathematics provides a framework for dealing precisely with notions of ‘what is.’ Computation provides a framework for dealing precisely with notions of ‘how to.’”
Some Challenges

• Many common, important problems
  • Fault tolerance
  • Coordination of concurrent activities
  • Geo. separated but linked data
  • Large-scale data sets
  • Protection from mistakes and attacks
  • Interactions with many people

• All of these problems lead to complexity
Complexity

• How do we control complexity?
  • Build abstractions that hide unimportant details
  • Establish interfaces and boundaries
  • Enable composition of simple, well-defined components

• None of this is specific to computer systems
  • Just principles of good engineering
  • Applies to other human structures as well

• But computer systems must establish a foundation and framework for extension/evolution: a platform.
Key questions for semester

• What are the right abstractions?
• How should we enforce modularity/isolation?
• How do we ensure fair, efficient allocation?
• What is the basis for trust?

• We will read a lot of papers this semester
  • Useful to think about them in terms of these questions
  • Sometimes goals are in tension (e.g., modularity vs. efficiency)
  • Good papers explain the trade-offs

• This semester will emphasize trustworthy computing topics.

Today, in the developed world, we do not worry about electricity and water services being available. ...Computing falls well short of this...

The events of last year - from September's terrorist attacks to a number of malicious and highly publicized computer viruses - reminded every one of us how important it is to ensure the integrity and security of our critical infrastructure, whether it's the airlines or computer systems.

...Microsoft and the computer industry will only succeed in that world if CIOs, consumers and everyone else sees that Microsoft has created a platform for Trustworthy Computing.

Every week there are reports of newly discovered security problems in all kinds of software, from individual applications and services to Windows, Linux, Unix and other platforms. .... Our new design approaches need to dramatically reduce the number of such issues that come up in the software that Microsoft, its partners and its customers create. ...Eventually, our software should be so fundamentally secure that customers never even worry about it.

No Trustworthy Computing platform exists today.
- **CodeRed (2001)**
  - Exploited an overflow in the MS-IIS server
  - 300,000 machines infected in 14 hours

- **SQL Slammer (2003)**
  - Exploited an overflow in the MS-SQL server
  - 75,000 machines infected in 10 minutes

[This content courtesy of Dave Levin]
OS Structure
Singularity: Rethinking the Software Stack

Galen C. Hunt and James R. Larus
Microsoft Research Redmond

galenh@microsoft.com

ABSTRACT
Every operating system embodies a collection of design decisions. Many of the decisions behind today’s most popular operating systems have remained unchanged, even as hardware and software have evolved. Operating systems form the foundation of almost every software stack, so inadequacies in present systems have a pervasive impact. This paper describes the efforts of the Singularity project to re-examine these design choices in light of advances in programming languages and verification tools. Singularity systems incorporate three key architectural features: software-isolated processes for protection of programs and system services, contract-based channels for communication, and manifest-based programs for verification of system properties. We describe this foundation in detail and sketch the ongoing research in experimental systems that build upon it.
Figure 4a. Micro-kernel configuration (like MINIX 3). Dotted lines mark protection domains; dark domains are user-level light are kernel-level.

Figure 4b. Monolithic kernel and monolithic application configuration.

Figure 4c. Configuration with distinct policies for signed and unsigned code.
Rethinking the Library OS from the Top Down

Donald E. Porter†, Silas Boyd-Wickizer‡, Jon Howell, Reuben Olinsky, Galen C. Hunt
Microsoft Research
One Microsoft Way
Redmond, WA 98052

† Department of Computer Science
Stony Brook University
Stony Brook, NY 11794

‡ MIT CSAIL
32 Vassar Street
Cambridge, MA 02139

This paper revisits an old approach to operating system construction, the library OS, in a new context. The idea of the library OS is that the personality of the OS on which an application depends runs in the address space of the application. A small, fixed set of abstractions connects the library OS to the host OS kernel, offering the promise of better system security and more rapid independent evolution of OS components.

We describe a working prototype of a Windows 7 library OS that runs the latest releases of major applications such as Microsoft Excel, PowerPoint, and Internet Explorer. We demonstrate that desktop sharing across independent, securely isolated, library OS instances can be achieved through the pragmatic reuse of networking protocols. Each instance has significantly lower overhead than a full VM bundled with an application: a typical application adds just 16MB of working set and 64MB of disk footprint. We contribute a new ABI below the library OS that enables application mobility. We also show that our library OS can address many of the current uses of hardware virtual machines at a fraction of the overheads. This paper describes the first working prototype of a full commercial OS redesigned as a library OS capable of running significant applications. Our experience shows that the long-promised benefits of the library OS approach—better protection of system integrity and rapid system evolution—are readily obtainable.
Before

After

Figure 1. Windows 7 OS Architecture.

Figure 2. Drawbridge Architecture.
Protection and trust
Dune: Safe User-level Access to Privileged CPU Features

Adam Belay, Andrea Bittau, Ali Mashtizadeh, David Terei, David Mazières, Christos Kozyrakis
Stanford University

Abstract

Dune is a system that provides applications with direct but safe access to hardware features such as ring protection, page tables, and tagged TLBs, while preserving the existing OS interfaces for processes. Dune uses the virtualization hardware in modern processors to provide a process, rather than a machine abstraction. It consists of a small kernel module that initializes virtualization hardware and mediates interactions with the kernel, and a user-level library that helps applications manage privileged hardware features. We present the implementation of Dune for 64-bit x86 Linux. We use Dune to implement three user-level applications that can benefit from access to privileged hardware: a sandbox for untrusted code, a privilege separation facility, and a garbage collector. The use of Dune greatly simplifies the implementation of these applications and provides significant performance advantages.

Figure 1: The Dune system architecture.
Figure 1: The Dune system architecture.
Shielding applications from an untrusted cloud with Haven

Andrew Baumann  Marcus Peinado  Galen Hunt

*Microsoft Research*

**Abstract**

Today’s cloud computing infrastructure requires substantial trust. Cloud users rely on both the provider’s staff and its globally-distributed software/hardware platform not to expose any of their private data.

We introduce the notion of shielded execution, which protects the confidentiality and integrity of a program and its data from the platform on which it runs (i.e., the cloud operator’s OS, VM and firmware). Our prototype, Haven, is the first system to achieve shielded execution of unmodified legacy applications, including SQL Server and Apache, on a commodity OS (Windows) and commodity hardware. Haven leverages the hardware protection of Intel SGX to defend against privileged code and physical attacks such as memory probes, but also addresses the dual challenges of executing unmodified legacy binaries and protecting them from a malicious host. This work motivated recent changes in the SGX specification.
Windows 8 Platform Integrity Architecture

1. Secure boot (UEFI) prevents running an unknown OS loader
2. The kernel launches Early Launch Anti-Malware (ELAM) they enforce 3rd party drivers and apps
3. Measurements of the system start state were recorded in the TPM during boot
4. To prove a client is healthy, the anti-malware software can quote TPM measurements to a remote verifier
Information Flow Control for Standard OS Abstractions

Maxwell Krohn        Alexander Yip       Micah Brodsky       Natan Cliffer
M. Frans Kaashoek    Eddie Kohler†       Robert Morris
MIT CSAIL             †UCLA
http://flume.csail.mit.edu/
Storage layers

User program

Database (x-action begin, commit)

File system (open, close, read, write)

Hardware (Block read/write)
The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google*

We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.

While sharing many of the same goals as previous distributed file systems, our design has been driven by observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system assumptions. This has led us to reexamine traditional choices and explore radically different design points.

The file system has successfully met our storage needs. It is widely deployed within Google as the storage platform for the generation and processing of data used by our service as well as research and development efforts that require large data sets. The largest cluster to date provides hundreds of terabytes of storage across thousands of disks on over a thousand machines, and it is concurrently accessed by hundreds of clients.

In this paper, we present file system interface extensions designed to support distributed applications, discuss many aspects of our design, and report measurements from both micro-benchmarks and real world use.
Spanner: Google’s Globally-Distributed Database


Google, Inc.

Abstract

Spanner is Google’s scalable, multi-version, globally-distributed, and synchronously-replicated database. It is the first system to distribute data at global scale and support externally-consistent distributed transactions. This paper describes how Spanner is structured, its feature set, the rationale underlying various design decisions, and a novel time API that exposes clock uncertainty. This API and its implementation are critical to supporting external consistency and a variety of powerful features: non-blocking reads in the past, lock-free read-only transactions, and atomic schema changes, across all of Spanner.

Spanner’s main focus is managing cross-datacenter replicated data, but we have also spent a great deal of time in designing and implementing important database features on top of our distributed-systems infrastructure. Even though many projects happily use Bigtable [9], we have also consistently received complaints from users that Bigtable can be difficult to use for some kinds of applications: those that have complex, evolving schemas, or those that want strong consistency in the presence of wide-area replication. (Similar claims have been made by other authors [37].) Many applications at Google have chosen to use Megastore [5] because of its semi-
Remus: High Availability via Asynchronous Virtual Machine Replication

Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norm Hutchinson, and Andrew Warfield*

Department of Computer Science
The University of British Columbia
{brendan, geoffrey, dmeyer, feeley, norm, andy}@cs.ubc.ca

represents a severe barrier to improving the dependability of large or legacy applications. We describe the construction of a general and transparent high availability service that allows existing, unmodified software to be protected from the failure of the physical machine on which it runs. Remus provides an extremely high degree of fault tolerance, to the point that a running system can transparently continue execution on an alternate physical host in the face of failure with only seconds of downtime, while completely preserving host state such as active network connections. Our approach encapsulates protected software in a virtual machine, asynchronously propagates changed state to a backup host at frequencies as high as forty times a second, and uses speculative execution to concurrently run the active VM slightly ahead of the replicated system state.
Programming Labs

• Done in groups of one or two
  • Email me groups by January 17

• Two labs
  1. Concurrency and synchronization (50%)
  2. File systems /distributed storage (50%)
Research Projects

- **Done in groups of three**
- **Five phases**
  1. Form groups (due after concurrency project)
  2. Write proposal (20% of project grade)
  3. Write status report (10% of pg)
  4. Write final report (60% of pg)
  5. Give presentation (10% of pg)
Two roles of the OS
OS as illusionist
OS as government
Main government functions

• **Resource manager** (*who gets what and when*)
  • Lock acquisition
  • Processes
  • Disk requests
  • Page eviction

• **Isolation and security** (*law and order*)
  • Access control
  • Kernel bit
  • Authentication
Two roles of the OS

**Abstractions**
- Modularity
- Simplicity
- Hide messy reality

**Government**
- Law and order
- Fair, efficient allocation
- Source of trust
Two roles of the OS

Abstractions

Modularity
Simplicity
Hide messy reality

Government

Law and order
Fair, efficient allocation
Source of trust

How does OS enforce modularity?
Two roles of the OS

Abstractions
- Modularity
- Simplicity
- Hide messy reality

Government
- Law and order
- Fair, efficient allocation
- Source of trust

How does OS ensure fair allocation?
Two roles of the OS

Abstractions
- Modularity
- Simplicity
- Hide messy reality

Government
- Law and order
- Fair, efficient allocation
- Source of trust

What is the basis for trust?