Introduction

Dongyoon Lee
About me

- Dongyoon Lee
- Assistant Professor in CS @ SBU (since Fall 2019)
- Assistant Professor in CS @ Virginia Tech (for five years)
- Ph.D in CSE @ U of Michigan, Ann Arbor (2013)
- Email: dongyoon@cs.stonybrook.edu
- Office: 339 New Computer Science (NCS) Building
- Web: https://www3.cs.stonybrook.edu/~dongyoon/
My research interests: HW + OS + SE for ...

- Reliability
  - Concurrency bug [CGO’18][EuroSys’17][ASPLOS’17][ASPLOS’16]
  - Regular expression [FSE’19][ASE’19][ASE’19]
  - Transient fault (soft error) tolerance [MICRO’16][SC’16][LCTES’15]

- Security
  - Linux kernel permissions [USENIX Security’19]
  - Denial of service [USENIX Security’18][FSE’18][EuroSec’17]
  - Memory safety [ASPLOS’19][MICRO’18]
My research interests: HW + OS + SE for ...

- Performance
  - Edge stream processing [ATC’19]
  - Distributed key value stores [SC’18]
About this course

• CSE 506: Operating Systems
• This semester: **Linux Kernel Programming**
• Goals
  • Understand core subsystems of the Linux kernel in depth
  • Design, implement, and modify Linux kernel code and modules for these subsystems
  • Test, debug, and evaluate the performance of systems software in kernel or user space, using debugging, monitoring and tracing tools
What is the *Linux Kernel*?

- One of operating system kernel
  - e.g., Windows, FreeBSD, OSX, etc.
- What does an OS do for you?
  - **Abstract** the hardware for convenience and portability
  - **Multiplex** the hardware among multiple applications
  - **Isolate** applications to contain bugs
  - Allow **sharing** among applications
View: layered organization

- **User**: applications (e.g., vi and gcc)
- **Kernel**: file system, process, etc.
- **Hardware**: CPU, mem, disk, etc.

→ **Interface** between layers
View: core services

- Processes
- Memory
- File contents
- Directories and file names
- Security
- Many others: users, IPC, network, time, terminals, etc.

→ Abstraction for applications
Example: system calls

- **Interface**: applications talk to an OS via system calls
- **Abstraction**: process and file descriptor

```c
fd = open("out", 1);
write(fd, "hello\n", 6);
pid = fork();
```
Why is Linux kernel interesting?

- OS design deals with conflicting goals and trade-offs
  - Efficient yet portable
  - Powerful yet simple
  - Isolated yet interactable
  - General yet performant
- Open problems: multi-core and security
- How does a state-of-the-art OS deal with above issues?
  - Hack the Linux kernel!
Why is Linux kernel interesting?

- Extremely large software project
  - more than 25 22 million lines of code
  - 7,500 4,600 lines of code are added every day!
Why is Linux kernel interesting?

• Very fast development cycles
  • release about every 70 days
  • 13,000 patches / release
  • 273 companies / release (or 1,600 developers / release)
• One of the most well-written/designed/maintained C code
• Ref: Linux Foundation Kernel Report 2017
Linux is eating the World

- 68% of smartphones and tables run Linux
  - iOS: 24%
- 98% of top 1 million web servers run Linux
- 99% of super computers run Linux

Ref: Usage share of OS
It is good for your job search

• Contributions from unpaid developers had been in slow decline
• Why?
  • “There are many possible reasons for this decline, but, arguably, the most plausible of those is quite simple: Kernel developers are in short supply, so anybody who demonstrates an ability to get code into the mainline tends not to have trouble finding job offers.”
• Ref: Linux Foundation Kernel Report 2017
Who should take this course?

- Anyone wants to work on the above problems
- Anyone cares about what’s going on under the hood
- Anyone has to build high-performance systems
- Anyone needs to diagnose bugs or security problems
Prerequisite

- Undergraduate Operating Systems (CSE 306)
- C programming
- Linux command line
Text book

- Robert Love, Linux Kernel Development, Addison-Wesley
Other useful sources

• Understanding the Linux Kernel, O’Reilly Media
• Professional Linux Kernel Architecture, Wrox
• Linux Device Drivers, O’Reilly Media
• Understanding Linux Network Internals, O’Reilly Media
• Operating Systems: Three Easy Pieces
• Intel 64 and IA-32 Architectures Software Developer Manuals
Communication

- **Course website**
  - Syllabus, lecture slides, schedule, notes, etc.
  - Primary way materials are *distributed*.

- **Blackboard**
  - Project submission
  - Grades posted

- **Piazza**
  - Use it to ask and answer questions
Office hours

- Dongyoon Lee
  - Tues and Thurs 2:30-3:30pm, 339 NCS
- GTA
  - The detail of GTA office hours will be announced when GTAs are assigned
Grades

- Project 1 (10%) - by end of Sept.
- Project 2 (30%) - by end of Oct.
- Project 3 (30%) - by end of semester
- Midterm exam (30%) - Nov. 14th (tentative)
- No final exam (project 3 will be considered to be a final exam)
About projects (subject to change)

- Each student will have a Linux VM (Details will follow).
- P1: Adding new system calls
- P2: S2DSM (Super simple distributed shared memory)
- P3: S2FS (Super simple file system)
Policies

• All programming projects are **individual** assignments. You may discuss the assignment details, designs, debugging techniques, or anything else with anyone you like in general terms, but you may not provide, receive, or take code to or from anyone.

• Late penalty: There will be a **deduction equal to 25%** of the assignment’s maximum score per day.

• The midterm exam will be in-class, closed-book exam.
Student Accessibility Support Center

If you have a physical, psychological, medical or learning disability that may impact your course work, please contact Student Accessibility Support Center, ECC (Educational Communications Center) Building, room 128, (631) 632-6748. They will determine with you what accommodations, if any, are necessary and appropriate. All information and documentation is confidential.
Academic Integrity

Each student must pursue his or her academic goals honestly and be personally accountable for all submitted work. Representing another person’s work as your own is always wrong. Faculty are required to report any suspected instances of academic dishonesty to the Academic Judiciary. For more comprehensive information on academic integrity, including categories of academic dishonesty, please refer to the Academic Integrity website.
Critical Incident Management

Stony Brook University expects students to respect the rights, privileges, and property of other people. Faculty are required to report to the Office of Judicial Affairs any disruptive behavior that interrupts their ability to teach, compromises the safety of the learning environment, or inhibits students’ ability to learn.
Questions?
Today’s agenda

• The history of Linux
• Linux open source model and community
• High level overview of the Linux kernel
Beginning of Linux

From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Message-ID: <1991Aug25.205708.9541@klaava.Helsinki.FI>
Date: 25 Aug 91 20:57:08 GMT
Organization: University of Helsinki

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them 😞

Linus (torvalds@kruuna.helsinki.fi)

PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT portable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.
Linux History

- 1991: First apparition, author: Linus Torvalds
- 1992: GPL License, first Linux distributions
- 1994: v1.0 - Single CPU for i386, then ported to Alpha, Sparc, MIPS
- 1996: v2.0 - Symmetric multiprocessing (SMP) support
- 1999: v2.2 - Big Kernel Lock removed
- 2001: v2.4 - USB, RAID, Bluetooth, etc.
- 2003: v2.6 - Physical Address Expansion (PAE), new architectures, etc.
- 2011: v3.0 - Incremental release of v2.6
Linux open source model

- Linux is licensed under GPLv2

  You may copy, distribute and modify the software as long as you track changes/dates in source files. Any modifications to or software including (via compiler) GPL-licensed code must also be made available under the GPL along with build & install instructions.

- Source code is freely available at https://www.kernel.org/

- Ref: td;lrLegal, GPLv2
Benefit of open source model

“Given enough eyeballs, all bugs are shallow

“Given a large enough beta-test and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone.

• Linus’s Law
  • The Cathedral & the Bazaar by Eric S. Raymond
• Security, stability, quality, speed of innovation, education, research, etc
Kernel release cycle

- (major).(minor).(stable) → E.g., 4.12.8

- Prepatch or “RC” kernel release → for testing before the mainline release
- Mainline release → maintained by Linus with all new features
- Stable release → additional bug fixes after the mainline kernel release
- Long term support (LTS) for a subset of releases → e.g., 4.14, 4.9, 4.4
Overview of operating systems
User space vs. kernel space

- A CPU is executing in either of **user space** or in **kernel space**
- Only the kernel is allowed to perform **privileged operations** such as controlling CPU and IO devices
  - E.g., protection ring in x86 architecture
  - ring 3: user-space application
  - ring 0: operating system kernel
- An user-space application talks to the kernel space through **system call** interface
  - E.g., `open()`, `read()`, `write()`, `close()`
User space vs. kernel space

Example: simplified path in the kernel for reading data into a file on disk
Linux is a *monolithic kernel*

- A traditional design: all of the OS runs in kernel, privileged mode
  - share the same address space
- Kernel interface ~= system call interface
- Good: easy for subsystems to cooperate
  - one cache shared by file system and virtual memory
- Bad: interactions are complex
  - leads to bugs, no isolation within kernel
Alternative: *microkernel design*

- Many OS services run as ordinary user programs
  - e.g., file system in a file server
- Kernel implements minimal mechanism to run services in user space
  - IPC, virtual memory, threads
- Kernel interface ≠ system call interface
  - applications talk to servers via IPCs
- Good: more isolation
- Bad: IPCs may be slow
Kernel & course map

User space

Kernel space

- Processing
- Memory Management

Hardware
- CPU
- Main Memory
Kernel & course map

User space

Kernel space

- Processing
- Memory Management
- Interrupt Management

System Call Interface

- Human Interface & Various Devices
- Storage
- Networking

Devices drivers

Hardware

- CPU
- Main Memory
- Mouse, Kbd., etc.
- HDD / SSD
- Network Interface
Kernel & course map

User space

Kernel space

System Call Interface

Processing
Processes & Threads
Scheduling
Time Mgt.

Interrupt Management

Memory Management

Human Interface & Various Devices
Storage
Networking

Devices drivers

Hardware

CPU
Main Memory
Mouse, Kbd., etc.
HDD / SSD
Network Interface
Kernel & course map

- **Kernel space**
  - Processing
    - Processes & Threads
    - Scheduling
    - Time Mgmt.
  - Interrupts Mgmt.
    - SoftIrq
    - Tasklet
    - Work queues
    - Interrupt handling

- **System Call Interface**
  - Memory Management
  - Human Interface & Various Devices
  - Storage
  - Networking

- **Devices drivers**

- **Hardware**
  - CPU
  - Main Memory
  - Mouse, Kbd., etc.
  - HDD / SSD
  - Network Interface
Kernel & course map

User space

Kernel space

System Call Interface

Processing
- Processes & Threads
- Scheduling
- Time Mgt.

Memory Management
- Physical / Virtual Memory Management
- Process Address Space
- Memory Allocation
- Page Cache

Human Interface & Various Devices

Storage

Networking

Devices drivers

Interrupts Mgt.
- SoftIrq
- Tasklet
- Work queues
- Interrupt handling

Hardware

CPU

Main Memory

Mouse, Kbd., etc.

HDD / SSD

Network Interface
Kernel & course map

User space

Kernel space

System Call Interface

Hardware

Kernel space

- Processing
  - Processes & Threads
  - Scheduling
  - Time Mgt.

- Interrupts Mgt.
  - SoftIrq
  - Tasklet
  - Work queues
  - Interrupt handling

System Call Interface

- Memory Management
  - Physical / Virtual Memory Management
  - Process Address Space
  - Memory Allocation
  - Page Cache

- Human Interface & Various Devices
  - Storage
    - Virtual File System
  - File System
  - Block layer
  - Network
    - Sockets
    - TCP/UDP
    - IP
    - Ethernet

Hardware

- CPU
- Main Memory
- Mouse, Kbd., etc.
- HDD / SSD
- Network Interface
Kernel & course map
Kernel & course map

User space
- Kernel debugging
- Development tools
- Static code exploration
- Performance eval.

Kernel space
- Data structures
- Synchronization
- Debugging
- Tracing
- Perf. Evaluation

System Call Interface
- Processing
  - Processes & Threads
  - Scheduling
  - Time Mgt.
- Memory Management
  - Physical / Virtual Memory Management
  - Process Address Space
- Memory Allocation
- Page Cache
- Human Interface & Various Devices
- Storage
  - Virtual File System
  - File System
  - Block layer
- Networking
  - Sockets
  - TCP/UDP
  - IP
  - Ethernet

Hardware
- CPU
- Main Memory
- Mouse, Kbd., etc.
- HDD / SSD
- Network Interface
Kernel & course map

• Let’s check course schedule
Next lecture

- No class on Thursday, August 29th
- Next Tuesday – What Software Expects of the OS
- Next Thursday – What Hardware Provides to the OS