T-401-ICYB
Operating System Basics

Stephan Schiffel
stephans@ru.is
Reykjavik University, Iceland

26.11.2025

Outline

What is an OS?

The Operating System (Kernel)

The Operating System (Kernel)

- Event-driven program that manages the hardware state.
- Acts as a bridge between applications and hardware.

The Operating System (Kernel)

- Event-driven program that manages the hardware state.
- Acts as a bridge between applications and hardware.
- **The Kernel:**
  - The core component of the OS.
  - Runs in **Privileged Mode (Kernel Mode)**.
  - Has direct access to hardware instructions and memory.
- **User Space:**
  - Where your applications run.
  - Runs in **Restricted Mode (User Mode)**.

Abstraction

Problem: Raw hardware interfaces (registers, disk controllers) are complex and inconsistent.

Solution: System Calls

- The OS wraps hardware instructions in clean software APIs.
- API is consistent across platforms, independent of the actual hardware.
- When you call `printf()` or `fopen()`, you trigger a **System Call**.
- The OS switches to Kernel Mode, performs the hardware task, and returns the result.

Arbitration / Resource Management

Problem: Resources (CPU cycles, RAM, I/O) are finite; multiple programs want them simultaneously.

Solution: Multiplexing

- **Time Multiplexing:** Scheduling different processes on the CPU over time.
- **Space Multiplexing:** dividing RAM and Disk space among processes and users.
- **Isolation:** Ensuring a bug in Process A does not crash or starve Process B.

Key Functionalities

Process Management (Scheduling)

- **Program vs. Process:**
  - *Program:* Static instructions on disk.
  - *Process:* A program in execution (loaded into RAM with a specific program counter, stack pointer, and registers).

Process Management (Scheduling)

- **Program vs. Process:**
  - *Program:* Static instructions on disk.
  - *Process:* A program in execution (loaded into RAM with a specific program counter, stack pointer, and registers).
- **Context Switching:**
  - The OS halts the current process.
  - Saves the CPU register state to a **Process Control Block (PCB)** in RAM.
  - Loads the saved state of the next process.
- **Result:** The illusion of parallelism on a single core.

Memory Management (Virtualization)

The OS provides the abstraction of **Virtual Memory**.

- **Address Spaces:** Every process believes it has access to a contiguous map of memory (e.g., 0x0000 to 0xFFFF).
- **Translation:** The OS + Hardware (MMU) map virtual addresses to physical RAM addresses.
- **Protection:**
  - Process A cannot access Process B's physical memory pages (or segments).
  - A user process cannot access kernel memory.
  - *Segfault:* Occurs when a process tries to access an address that has no mapping to a physical address.

I/O and Interrupt Management

Architecture context: I/O can be millions of times slower than a CPU cycle. We cannot use busy-waiting.

1. **Request:** User program requests I/O (System Call).
2. **Sleep:** OS puts the process in a "Waiting" state and yields the CPU to another process.
3. **Interrupt:** When hardware is ready, it sends an electrical signal (Interrupt) to the CPU.
4. **ISR:** CPU jumps to the OS's **Interrupt Service Routine** (essentially a function in the kernel).
5. **Wake:** OS moves the original process back to the "Ready" queue.

File Systems / Storage Abstraction

- **Physical View:** A hard drive is an array of millions of generic blocks (sectors).
- **Logical View:** The OS creates the concept of Files, Directories, and Paths.
- **Responsibility:**
  - Mapping filenames to physical block addresses.
  - Managing metadata (permissions, timestamps).
  - ensuring consistency (journaling) in case of power loss.

Protection (Ring 0 vs Ring 3)

Hardware architectures (like x86) support privilege levels.

Ring 0 (Kernel Mode)

- Full access to all hardware instructions.
- Can manipulate memory maps and disable interrupts.

Ring 3 (User Mode)

- Restricted subset of instructions.
- Cannot directly access hardware.

Traps: If user code attempts a privileged instruction, the CPU hardware "traps" the attempt and hands control to the OS to handle the violation (usually by killing the process).

Common Operating Systems

Unix / Linux

Origins: Developed at AT&T Bell Labs (late 60s).

The Unix Philosophy

"Everything is a file."

- Hardware, sockets, and data are accessed via **file descriptors**.
- Focus on small, modular tools piped together.

- **Linux:**
  - Technically just the **Kernel** (Linus Torvalds, 1991).
  - **Distro:** Kernel + GNU Tools + Package Manager + Desktop.
  - Architecture: Monolithic Kernel (drivers in kernel space).
- **macOS:**
  - Based on **Darwin** (Hybrid XNU kernel: Mach + BSD).
  - POSIX compliant (compatible with standard Unix APIs).

Windows (compared to Unix)

- **Architecture: Hybrid Kernel**
  - Performance critical parts (filesystem, networking) run in Kernel Mode.
  - Other subsystems run in User Mode (Microkernel-like influence).
- **Configuration: The Registry**
  - *Unix:* Uses text files (e.g., `/etc/config`).
  - *Windows:* Uses a centralized hierarchical database (The Registry).
- **Resource Model:**
  - Resources are treated as abstract **Objects** managed via **Handles**.

Side note: Windows since XP is based on the **Windows NT** architecture, unlike Windows 95, 98, Me which were based on MS-DOS.

Android

Often misunderstood as "Just Linux." It uses the Linux Kernel, but the user-space is unique.

Android

Often misunderstood as "Just Linux." It uses the Linux Kernel, but the user-space is unique.

The Android Stack

1. **Top:** Android Runtime (ART). Apps compile to Bytecode (DEX), not native machine code.
2. **Middle:** Native Libraries & Hardware Abstraction Layer (HAL).
3. **Bottom:** Linux Kernel (Memory, Scheduling, Drivers).

Android

Often misunderstood as "Just Linux." It uses the Linux Kernel, but the user-space is unique.

The Android Stack

1. **Top:** Android Runtime (ART). Apps compile to Bytecode (DEX), not native machine code.
2. **Middle:** Native Libraries & Hardware Abstraction Layer (HAL).
3. **Bottom:** Linux Kernel (Memory, Scheduling, Drivers).

Key Mechanism: Binder IPC

- Apps are strictly sandboxed (security).
- They communicate via **Binder** (Inter-Process Communication) to talk to the OS or other apps.

Embedded and RTOS (Real-Time Operating Systems)

General Purpose OS (Windows/Linux) Goal: Throughput.
RTOS Goal: **Determinism / Consistency**.

Embedded and RTOS (Real-Time Operating Systems)

General Purpose OS (Windows/Linux) Goal: Throughput.
RTOS Goal: **Determinism / Consistency**.

- **RTOS (Real-Time Operating System):**
  - Examples: FreeRTOS, VxWorks, QNX.
  - **Hard Real-Time:** A delayed calculation is a system failure (e.g., car brakes, pacemaker).
  - **Features:** Tiny kernels, fast interrupt handling, often no virtual memory (to avoid paging latency).

Embedded and RTOS (Real-Time Operating Systems)

General Purpose OS (Windows/Linux) Goal: Throughput.
RTOS Goal: **Determinism / Consistency**.

- **RTOS (Real-Time Operating System):**
  - Examples: FreeRTOS, VxWorks, QNX.
  - **Hard Real-Time:** A delayed calculation is a system failure (e.g., car brakes, pacemaker).
  - **Features:** Tiny kernels, fast interrupt handling, often no virtual memory (to avoid paging latency).

- **Embedded Linux:**
  - Standard Linux stripped down (e.g., Raspberry Pi).
  - Used when complex networking or GUIs are needed, but strict microsecond-timing isn't required.

What is POSIX?

POSIX = **P**ortable **O**perating **S**ystem **I**nterface (the **X** is for Unix).

What is POSIX?

POSIX = **P**ortable **O**perating **S**ystem **I**nterface (the **X** is for Unix).

The Concept: An API Contract

POSIX is an IEEE standard that defines an **interface** between User Space applications and the OS Kernel.

- It does **not** tell the OS how to implement a feature.
- It tells the OS: "If you want to be Unix-compatible, you must support these function names and behaviors."

What is POSIX?

POSIX = **P**ortable **O**perating **S**ystem **I**nterface (the **X** is for Unix).

The Concept: An API Contract

POSIX is an IEEE standard that defines an **interface** between User Space applications and the OS Kernel.

- It does **not** tell the OS how to implement a feature.
- It tells the OS: "If you want to be Unix-compatible, you must support these function names and behaviors."

The Motivation (The "Unix Wars")

- In the 80s, Unix fragmented (BSD, System V, etc.). Code written for one machine wouldn't compile on another.
- **Goal:** Source code portability. A C program written using POSIX standards will compile on Linux, macOS, and BSD without changes (mostly).

Components and Adoption

What does POSIX actually define?

- **System Call APIs (C Headers):**
  - File I/O: `open()`, `read()`, `write()` (vs Windows `CreateFile`).
  - Process Control: `fork()`, `exec()`, `wait()`.
  - Threading: **pthreads** (`<pthread.h>`).
- **Shell & Utilities:**
  - Standardizes behavior of CLI tools (`ls`, `grep`, `awk`) so scripts are portable.

Components and Adoption

What does POSIX actually define?

- **System Call APIs (C Headers):**
  - File I/O: `open()`, `read()`, `write()` (vs Windows `CreateFile`).
  - Process Control: `fork()`, `exec()`, `wait()`.
  - Threading: **pthreads** (`<pthread.h>`).
- **Shell & Utilities:**
  - Standardizes behavior of CLI tools (`ls`, `grep`, `awk`) so scripts are portable.

Who is Compliant?

OS	Status
macOS	**Certified.** Fully POSIX compliant.
Linux	**De Facto.** Mostly compliant, but rarely pays for certification.
Windows	**No.** Uses Win32 API. Needs WSL (Subsystem for Linux).

Operating Systems and Security

The OS and the Trusted Computing Base

Trusted Computing Base (TCB)

... is the totality of protection mechanisms within a computer system – including hardware, firmware, and software – that is responsible for enforcing a security policy. TCB is all components that must work correctly for the system to be secure. If any part is broken, the security of the entire system is broken.

The OS and the Trusted Computing Base

Trusted Computing Base (TCB)

... is the totality of protection mechanisms within a computer system – including hardware, firmware, and software – that is responsible for enforcing a security policy. TCB is all components that must work correctly for the system to be secure. If any part is broken, the security of the entire system is broken.

The Reference Monitor

- The OS acts as a gatekeeper between **Subjects** (Users, Processes) and **Objects** (Files, Hardware).
- Every system call (e.g., `open()`) is intercepted to check:
  1. **Authentication:** Who are you?
  2. **Authorization:** Are you allowed to do this?

Authentication: "Who are you?"

Identification vs. Verification

- *Identification:* Claiming an identity (Username).
- *Verification:* Proving it (Password, Biometric, Key).

Authentication: "Who are you?"

Identification vs. Verification

- *Identification:* Claiming an identity (Username).
- *Verification:* Proving it (Password, Biometric, Key).

Secure Storage (Hashing)

- **Rule #1:** The OS **never** stores passwords in plain text.
- **Hashing:** The OS stores a cryptographic hash (e.g., SHA-256) of the password.
  - On login: $InputHash = Hash(UserInput)$.
  - If $InputHash == StoredHash$, access is granted.
- **Salting:**
  - To prevent "Rainbow Table" attacks (pre-computed hash databases), the OS adds a random string (salt) before hashing.
  - $StoredValue = Hash(Password + Salt)$.

Access Control: "What can you do?"

Once authenticated, the OS uses a **User ID (UID)** to enforce permissions.

- **DAC (Discretionary Access Control):**
  - The **Owner** of the file decides permissions.
  - *Example:* Linux `chmod`. You can make your file readable by everyone.
  - *Risk:* Malware running as "You" can change your file permissions.
- **MAC (Mandatory Access Control):**
  - The **System Policy** decides permissions. Users cannot override this.
  - Used in High Security (SELinux) and Mobile (iOS/Android).
  - *Example:* A web-server process cannot read the `/home` directory, even if run by root.

The Principle of Least Privilege

Golden Rule of OS Architecture

A process should only have the **absolute minimum** privileges necessary to do its job.

Why?

The Principle of Least Privilege

Golden Rule of OS Architecture

A process should only have the **absolute minimum** privileges necessary to do its job.

Why?

- It limits the "Blast Radius."
- If a calculator app is hacked, it shouldn't have permission to read the network driver or kernel memory.
- This is enforced via CPU modes (User Mode vs. Kernel Mode).

System Logs

the "Flight Recorder" of the OS.

- **What:** High-level textual records of events.
- **Content:** (Failed) logins, `sudo` usage, Service crashes.
- **Locations:**
  - Linux: `/var/log/syslog`, `/var/log/auth.log`
  - Windows: Event Viewer
- **Forensics:** The first place admins look after a breach.

File Integrity & System Auditing

Going deeper than standard logs:

- **System Auditing (Kernel Level):**
  - Hooks into the kernel to track **System Calls**.
  - *Example:* Not just "User logged in," but "Process 402 attempted `open()` on `/etc/shadow` and was denied."
  - Required for strict compliance to various standards (PCI-DSS, HIPAA).
- **File Integrity Monitoring (FIM):**
  - *Threat:* Attackers replacing system binaries (like `/bin/login`) with Trojan horses.
  - *Defense:* Calculate checksums (hashes) of critical files.
  - If the hash of `/bin/login` changes, the OS triggers an alarm immediately.

Enable System Auditing: Windows

System Auditing is a granular policy managed via the **Local Security Policy** tool.

- **GUI Method:**
  1. Run `secpol.msc` as Administrator.
  2. Navigate to: *Local Policies* → *Audit Policy*.
  3. Double-click a category (e.g., "Audit logon events" or "Audit object access").
  4. Check **Success** (log when it works) and/or **Failure** (log when it is denied).
- **Command Line Method (PowerShell/CMD):** Using the `auditpol` tool.

Example Command

```
> auditpol /set /subcategory:"Logon" /success:enable
/failure:enable
```

*Logs are viewed in the **Event Viewer** or using `Get-EventLog` in PS.*

Enable System Auditing: Linux

Linux uses the **Linux Audit Framework**. It listens to the kernel for specific system calls and file changes.

- **Installation:**
  - Install: `sudo apt install auditd` (Debian/Ubuntu) or `yum install audit` (RHEL/CentOS).
  - Start: `sudo systemctl start auditd`.
- **Defining Rules:** You use `auditctl` to add rules to the kernel in real-time.

Example: Watch for changes to passwords

```
$ sudo auditctl -w /etc/shadow -p wa -k password-change
```

- `-w`: Watch this file path.
- `-p wa`: Trigger on **w**rite or **a**ttribute change.
- `-k`: Tag log entries with a search key.

Logs are stored in `/var/log/audit/audit.log`, queried via `ausearch` or `aureport`.

Enable System Auditing: macOS

macOS uses the **Basic Security Module (BSM)**, an advanced auditing system derived from TrustedBSD.

- **Configuration:**
  - Config file is located at `/etc/security/audit_control`.
  - Controls "flags" for events (e.g., `lo` for login, `ad` for administrative actions).
- **Activation:** The audit daemon (`auditd`) usually runs by default. To reload configuration changes:
  `$ sudo audit -s`
- **Binary Logs:** Unlike Linux, macOS audit logs are **binary** (to prevent tampering). You cannot read them with `cat`.

Viewing Logs

Use the `praudit` tool to parse the binary log to text:
`$ sudo praudit /var/audit/current`

Common Vulnerabilities and Exploits

Memory Safety: Buffer Overflows

stems from C/C++ allowing direct memory manipulation without bounds checking

Memory Safety: Buffer Overflows

stems from C/C++ allowing direct memory manipulation without bounds checking

- **The Flaw:**
  - A program allocates a fixed buffer.
  - The OS/Program fails to check input size, allowing writes *past* the buffer end.

Memory Safety: Buffer Overflows

stems from C/C++ allowing direct memory manipulation without bounds checking

- **The Flaw:**
  - A program allocates a fixed buffer.
  - The OS/Program fails to check input size, allowing writes *past* the buffer end.
- **The Exploit (Stack Smashing):**
  - Attacker overflows the stack to overwrite the **Return Address**.
  - CPU jumps to malicious code (shellcode) instead of returning to the main function.

Memory Safety: Buffer Overflows

stems from C/C++ allowing direct memory manipulation without bounds checking

- **The Flaw:**
  - A program allocates a fixed buffer.
  - The OS/Program fails to check input size, allowing writes *past* the buffer end.
- **The Exploit (Stack Smashing):**
  - Attacker overflows the stack to overwrite the **Return Address**.
  - CPU jumps to malicious code (shellcode) instead of returning to the main function.
- **Mitigation:**
  - **ASLR:** Randomizes memory layout so addresses are hard to guess.
  - **DEP/NX:** Marks stack memory as "Non-Executable."

Race Conditions

Example: TOCTOU (Time-of-Check to Time-of-Use)

Because OSs multitask, a gap exists between checking the file permissions and using it.

1. **Check:** OS confirms User A can write to `temp.txt`.
2. **The Gap (Context Switch):** Attacker quickly swaps `temp.txt` with a symbolic link to `/etc/shadow` (password file).
3. **Use:** OS resumes the process and writes to the target, overwriting the password file because the check already passed.

Command Injection

Occurs when the Shell is tricked into executing unintended commands.

- **The Flaw:** Passing unsanitized user input directly to a system command.
- **The Exploit:**
  - Script expects a filename: `rm $filename`
  - Attacker inputs: `file.txt; rm -rf /`
  - The OS interprets `;` as a command separator and executes both.

Privilege Escalation

The attacker's goal: Move from Ring 3 (User) to Ring 0 (Kernel).

- **Vertical Escalation:**
  - Exploiting bugs in "SetUID" programs (programs that run as root, like `sudo` or `passwd`).
- **Kernel Exploits:**
  - Crashing or manipulating the Kernel itself.
  - **Weak Link:** Third-party **Drivers**. They run with full privileges but often have less rigorous code quality than the core kernel.

Rootkits and Zero-Days

Rootkits

- Malware that modifies the system to hide its own existence.
- e.g., by intercepting system calls.
- Example: When you run `ls` or `ps`, the rootkit filters the output of the underlying system calls to hide the hacker's files and processes. The OS lies to the user.

Rootkits and Zero-Days

Rootkits

- Malware that modifies the system to hide its own existence.
- e.g., by intercepting system calls.
- Example: When you run `ls` or `ps`, the rootkit filters the output of the underlying system calls to hide the hacker's files and processes. The OS lies to the user.

Zero-Day Vulnerabilities

- A flaw known to the attacker but unknown to the vendor (0 days to fix).
- **Defense:** "Defense in Depth." You cannot patch it, so you must rely on firewalls, strict access control, procedures, etc to contain it.

Commands for Sysadmins

Privileges & Permissions

Action	Linux Command	PowerShell / Windows
Execute as root / Admin	`sudo [cmd]`	`Start-Process -Verb RunAs` (Or open Terminal as Admin)
Change File Permissions	`chmod 755 file`	`icacls file /grant ...` (Or `Set-Acl`)
View Permissions	`ls -l`	`icacls [file]` (Or `Get-Acl`)
Change Ownership	`chown user file`	`takeown /f file` (Or `Set-Acl`)
Change Password	`passwd`	`Set-LocalUser`

File Permissions: Linux vs. Windows

Linux (POSIX Model)

- **Simple Structure:** Permissions apply to three rigid categories:
  1. **U**ser (Owner)
  2. **G**roup
  3. **O**thers (World)
- **The Bits:** Read (r), Write (w), Execute (x).
- **Representation:** Octal (e.g., `755`) or Symbolic (`rwxr-xr-x`).

Windows (ACL Model)

- **Complex Structure:** Uses **Access Control Lists (ACLs)**.
- **Granular:** You can assign distinct permissions to 50 different users individually.
- **Inheritance:** Files heavily inherit rules from parent folders.
- **The Rights:** Full Control, Modify, Read & Execute, List, Read, Write.

Advanced Permissions: SUID & SGID

Concept: SUID (Set User ID) allows a user to execute a file with the permissions of the **file owner**, rather than their own.

Use Case: `passwd`

- **Problem:** Standard users cannot write to the password database (`/etc/shadow`).
- **Solution:** The `passwd` binary is owned by **root** and has SUID set.
- **Result:** When run, the process promotes to root temporarily to save changes.

Danger

SUID is a major vector for **Privilege Escalation**.

- If an SUID root binary has a bug (e.g., buffer overflow), an attacker can exploit it to gain a **Root Shell**.
- Attackers scan for these immediately:
  `find / -perm -4000`

Command: `chmod u+s file`
Indicator: `-rwsr-xr-x 1 root root 64152 maí 30 2024 /usr/bin/passwd`

Process & Service Management

Action	Linux Command	PowerShell / Windows
Start/Stop Services	`systemctl start [svc]`	`Start-Service [svc]` (`Get-`, `Stop-`, `Set-`)
List Processes (Snapshot)	`ps aux`	`Get-Process`
Real-time	`top` / `htop`	`Get-Process | Sort CPU` (Or GUI `taskmgr`)
Kill Process	`kill -9 [PID]`	`Stop-Process -Id [PID]`

Disk & File System Management

Action	Linux Command	PowerShell / Windows
Free Space	`df -h` (Disk Free)	`Get-Volume` (or `Get-PSDrive`)
Disk Usage	`du -sh [folder]`	`gci -Recurse | Measure -Property Length -Sum`
Search	`find /path -name x`	`Get-ChildItem -Recurse -Filter "x"`
Archive	`tar -czvf x.tar.gz` (or `zip`, `rar`, etc)	`Compress-Archive` (Win10+ has `tar.exe`)

Logs

Action	Linux Command	PowerShell / Windows
Follow changes live	`tail -f log.txt`	`Get-Content log.txt -Wait` (Alias: `cat -Wait`)
Search	`grep "Error"`	`Select-String "Error"` (Alias: `sls`)
... or both	`tail -f log.txt | grep "Error"`	`Get-Content log.txt -Wait | Select-String "Error"`

Scheduling & Maintenance

Action	**Linux Command**	**PowerShell / Windows**
Schedule Tasks	`crontab -e` (opens the cron table in an editor)	`Register-ScheduledTask`
Install Updates	`apt update && apt upgrade` (Depending on Distro)	`winget upgrade --all` (Only for apps, not the OS itself!)

Up Next ..

Further Studies

- Find examples of attacks that used root kits and discuss what role the rootkits played in the attacks.
- Side-Channel Attacks (Meltdown & Spectre): These famous vulnerabilities exploited Speculative Execution in the CPU hardware. How did they allow a User Mode process to read Kernel Mode memory without ever technically triggering a privilege violation?
- DMA (Direct Memory Access) Attacks: Peripheral devices (like a GPU or Network Card) use DMA to access RAM without asking the CPU. How can a malicious device (e.g., a compromised USB drive) use DMA to read system memory, bypassing the OS entirely?
- Which mitigation means exist to avoid the problem of SUID opening the system up for privilege escalation attacks?

Lab today and Guest Lecture tomorrow

- Lab 4: Use auditing (turn it on!) to find "malicious" processes modifying the file system.
- Guest Lecture tomorrow by **Giovanni Apruzzese** on Phishing