Operating Systems Principles and Programming

Principes et programmation des systèmes d'exploitation

Albert Cohen
`Albert.Cohen@inria.fr`

Fabrice Le Fessant
`Fabrice.Le_Fessant@inria.fr`

École Polytechnique — Majeure 2 d'Informatique — INF552

2006–2007

About the Course

About Us

Albert Cohen: researcher at INRIA Futurs Saclay (Orsay)

ALCHEMY group: architecture and compilation for high-performance and embedded computing

Fabrice Le Fessant: researcher at INRIA Futurs Saclay (Orsay)

ASAP group: foundations of large-scale dynamic distributed systems

Goals

- Understand how the operating system works
- Learn to program applications which interact with the operating system
- Expose (some) design goals and principles
- Abstract and simplify when necessary
- Show practical examples

`http://www.enseignement.polytechnique.fr/profs/informatique/Albert.Cohen/os`

Organization

Practical Information

- 9 lectures and 9 labs
- Questions (during or after the course) are welcome
- If you are lost, do not wait for getting help (after the course or during labs)
- One term exam (principles, algorithms and programmer interface)
- One project: extended labs (pick one among all courses of the *Majeure)*

Prerequisites

- C programming language and standard library
- Attending courses and labs
- Programming or reading code after lab hours

`http://www.enseignement.polytechnique.fr/profs/informatique/Albert.Cohen/os`

1. Why an Operating System?

Outline

1. Historical perspective
2. Technical survey
   - Resource sharing and management (time and hardware)
   - Abstraction (of the hardware, of low-level software layers)
   - Naming framework
   - Synchronization and communication services
   - Enforcing security policies
   - Virtualization (of the hardware, of specific software layers)
3. Design trends (research and industrial)

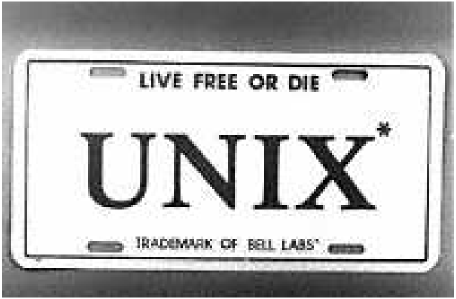
Historical Perspective

1964: IBM System/360
Integrated circuits, family of 6 compatible computers and 40 peripherals
OS: millions of line of assembly code

http://en.wikipedia.org/wiki/Operating_system
http://www.computerhistory.org/timeline http://www.osdata.com/kind/history.htm

Historical Perspective

1969: UNIX — Ken Thompson and Dennis Ritchie
UNiplexed Information and Computing Service (economical redesign of MULTICS)
Rewritten in C with Brian Kernighan in 1973



http://en.wikipedia.org/wiki/Operating_system
http://www.computerhistory.org/timeline   http://www.osdata.com/kind/history.htm

Historical Perspective

1974: Xerox PARC — Alto

First interactive window system, menus, icons

http://en.wikipedia.org/wiki/Operating_system

http://www.computerhistory.org/timeline   http://www.osdata.com/kind/history.htm

Historical Perspective

1983: GNU
Free operating system (and free software in general)

http://en.wikipedia.org/wiki/Operating_system
http://www.computerhistory.org/timeline http://www.osdata.com/kind/history.htm

Historical Perspective

1991: Linux

Free kernel, inspired by Minix, provides a complete OS with GNU

Now runs on mobile phones to 1024 processor NUMA

http://en.wikipedia.org/wiki/Operating_system

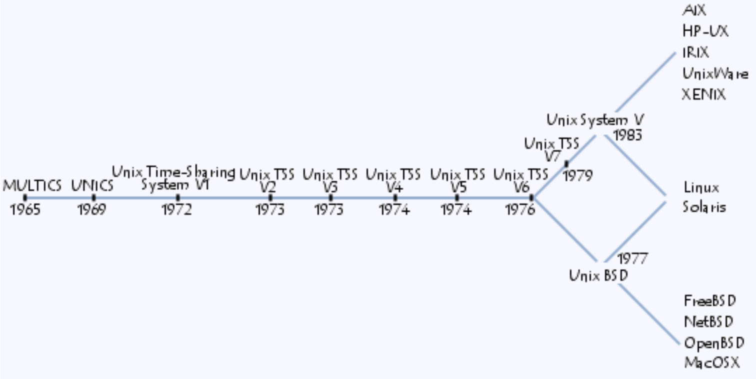
http://www.computerhistory.org/timeline   http://www.osdata.com/kind/history.htm

The UNIX Family

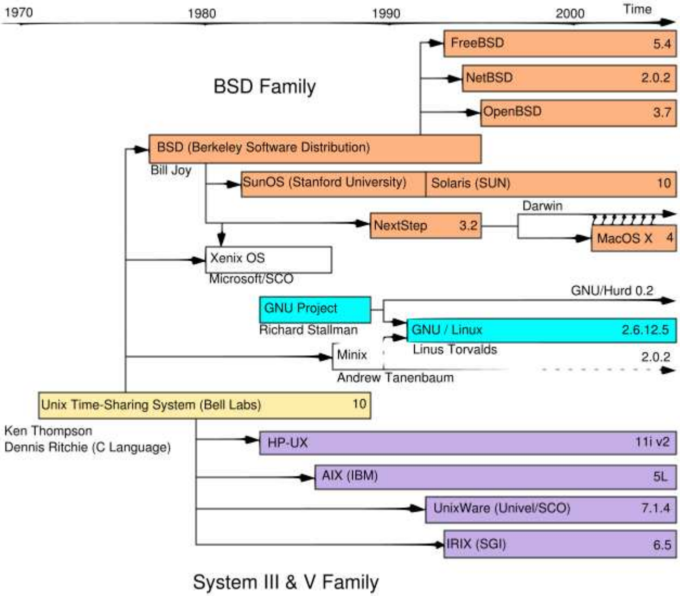
- Multi-user, multi-tasking, general-purpose operating system
- Sources available to a large public (but not free)
- Partly compatible family
  Standards: POSIX, X/Open, BSD, System V, SVr4, LSB...

The UNIX Family

- Multi-user, multi-tasking, general-purpose operating system
- Sources available to a large public (but not free)
- Partly compatible family
  Standards: POSIX, X/Open, BSD, System V, SVr4, LSB...



The UNIX Family



The UNIX Family

- Multi-user, multi-tasking, general-purpose operating system
- Sources available to a large public (but not free)
- Partly compatible family
  Standards: POSIX, X/Open, BSD, System V, SVr4, LSB...

GNU/Linux

- Free software (open source)
- Robust and modern flavor of UNIX
- Most portable and largest range of supported devices
- Highly compatible with other OSes
- Modular and customizable, excellent code quality
- Lightweight: can be downsized for embedded devices
- Benefits from most OS innovations

The UNIX Family

- Multi-user, multi-tasking, general-purpose operating system
- Sources available to a large public (but not free)
- Partly compatible family
  Standards: POSIX, X/Open, BSD, System V, SVr4, LSB...

GNU/Linux

- Free software (open source)
- Robust and modern flavor of UNIX
- Most portable and largest range of supported devices
- Highly compatible with other OSes
- Modular and customizable, excellent code quality
- Lightweight: can be downsized for embedded devices
- Benefits from most OS innovations

Technical Survey: Resource Management

Control

- Bootstrap the whole machine
  Firmware, BIOS, EFI, boot devices, initialization sequence
- Configure I/O devices and service low-level programmable components
  I/O ports and Memory-mapped I/O, interrupts
- Isolate and report errors or improper use of protected resources
  Kernel versus user mode, memory protection, exceptions (software-triggered interrupts)

Technical Survey: Resource Management

Control

- Bootstrap the whole machine
  Firmware, BIOS, EFI, boot devices, initialization sequence
- Configure I/O devices and service low-level programmable components
  I/O ports and Memory-mapped I/O, interrupts
- Isolate and report errors or improper use of protected resources
  Kernel versus user mode, memory protection, exceptions (software-triggered interrupts)

Allocate

- Distribute processing, storage, communications, in time and space
  Process/task, multiprocessing, preemption, virtual memory, file system, socket (port)
- Multi-user environment
  Session, identification, authorization, spying prevention, fairness, terminal
- Fair resource use
  Scheduling, priority, resource limits

Technical Survey: the Kernel

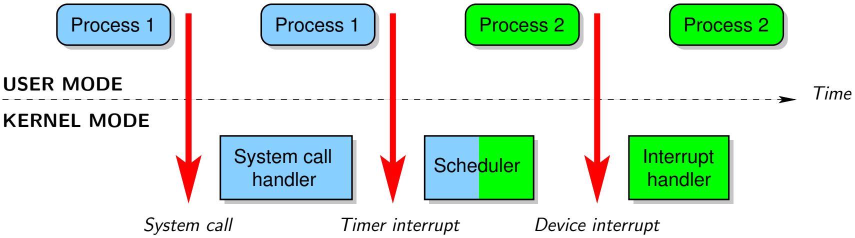
The Kernel

- The kernel is a *process manager*, not a process
- Processors provide instructions to switch between user and kernel modes
  - *Kernel mode*: no restriction
  - *User mode*: restricted instructions and memory regions
- User processes switch to kernel mode when requesting a service provided by the kernel: *system call*

Technical Survey: the Kernel

The Kernel

- The kernel is a *process manager*, not a process
- Processors provide instructions to switch between user and kernel modes
  - *Kernel mode*: no restriction
  - *User mode*: restricted instructions and memory regions
- User processes switch to kernel mode when requesting a service provided by the kernel: *system call*



Technical Survey: Abstraction

Goal

- Simplify, uniformize and standardize
  - Kernel portability
  - Facilitate device driver development
  - Stable execution environment for the user programs

Main Abstractions

1. Process
2. File and file system
3. Device interface and device driver
4. Virtual memory

Process Abstraction

Overview

- Process: *execution context of a running program*
- Multiprocessing, private address space
  - *Segments*: *text* (code), *static data*, *dynamic data* (stack and heap)
  - Code may be shared: multiple instances of a program, e.g., dynamic libraries
  - Data may be shared: IPC shared memory object
- Multiple execution flows in the same address space: *threads*
- There may also be kernel processes/threads (e.g., Solaris and Linux)

Process Abstraction

Overview

- Process: *execution context of a running program*
- Multiprocessing, private address space
  - *Segments*: *text* (code), *static data*, *dynamic data* (stack and heap)
  - Code may be shared: multiple instances of a program, e.g., dynamic libraries
  - Data may be shared: IPC shared memory object
- Multiple execution flows in the same address space: *threads*
- There may also be kernel processes/threads (e.g., Solaris and Linux)

Process State

- Internal descriptor designated by its identifier (PID)
  - State with respect to the scheduler's queue(s) (preemptive or not)
  - File descriptors, IPC (shared memory, semaphores, message queues)
  - Process and thread tree
- Processor registers: program counter (PC), stack pointer (SP), processor control, general-purpose, floating point
- Memory map (private and shared pages)

File and File System Abstractions

Overview

- File: *storage* and *naming* in UNIX
- Directory tree, absolute and relative pathnames
  `/` `.` `..` `/dev/hda1` `/bin/ls` `/etc/passwd`
- File types
  - Regular file or hard link (file name alias within a single file system)
    `$ ln pathname alias_pathname`
  - Soft link: short file containing a pathname
    `$ ln -s pathname alias_pathname`
  - Directory: list of file names (a.k.a. hard links)
  - Block-oriented device: buffered, random access to data
  - Character-oriented device: unbuffered stream of data
  - Pipe (also called FIFO)
  - Socket (UNIX and INET)
- Assemble multiple file systems through *mount points*
  Typical example: `/home` `/usr/local` `/proc`
- Common set system calls, independent of the target file system

Device Abstraction

Device Files

- Special files
  - *Block*-oriented device
    Disks, file systems: `/dev/hda` `/dev/sdb2` `/dev/md1`
  - *Character*-oriented device
    Serial ports, console terminals, audio: `/dev/tty0` `/dev/pts/0` `/dev/usb/hiddev0` `/dev/mixer` `/dev/null`
  - *Major* and *minor* numbers to (logically) connect device files and drivers
    Assigned dynamically (and/or at boot) in modern systems (e.g., Linux's `udev`)

Device Abstraction

Device Files

- Special files
  - *Block*-oriented device
    Disks, file systems: `/dev/hda` `/dev/sdb2` `/dev/md1`
  - *Character*-oriented device
    Serial ports, console terminals, audio: `/dev/tty0` `/dev/pts/0` `/dev/usb/hiddev0` `/dev/mixer` `/dev/null`
  - *Major* and *minor* numbers to (logically) connect device files and drivers
    Assigned dynamically (and/or at boot) in modern systems (e.g., Linux's `udev`)

Device Drivers

- Abstracted by system calls or kernel processes
- Manage buffering between device and local buffer
- Control devices through memory-mapped I/O or I/O ports
- Devices trigger interrupts (end of request, buffer full, etc.)
- Many concurrency challenges (precise synchronization required)
- Multiple layers for portability and reactivity (low-overhead reactions)

Virtual Memory Abstraction

Purpose

- Processes access memory through *virtual* addresses
  - Simulates a large *interval* of memory addresses
  - Expressive and efficient address-space protection and separation
  - Hides kernel and other processes' memory
  - Automatic *translation* to *physical* addresses by the CPU (MMU/TLB circuits)
- *Paging* mechanism
  - Provide a protection mechanism for memory regions, called *pages*
  - The kernel implements a *mapping* of physical pages to virtual ones, different for every process
- *Swap* memory and file system
  - The ability to suspend a process and virtualize its memory allows to store its pages to disk, saving (expensive) RAM for more urgent matters
  - Some systems use swap files rather than partitions (slower but more flexible)
  - Same mechanism to migrate processes on NUMA multi-processors

Technical Survey: Naming

Naming Resources and Abstractions

- Hard problem in operating systems
  - Processes are separated (logically and physically)
  - Need to access persistent and/or foreign resources
  - Resource identification determines large parts of the programming interface
  - Hard to get it right, general and flexible enough
- Good example: filenames and pathnames
  Uniform across complex directory trees, across storage devices (mount points), pipes, UNIX sockets, POSIX IPC
- Could be better: INET addresses (e.g., 129.104.247.5), TCP/UDP ports
- Bad examples: device numbers, System V IPC

Technical Survey: Synchronization

- Kernel primitives
  - Atomic instructions
  - Critical sections
  - Spin-lock and variants
  - Semaphores
  - Interrupt disabling
  - Kernel preemption disabling
- Interprocess (or threads) synchronization programming interface
  - Waiting for a process status change
  - Waiting for a signal
  - IPC Semaphore
  - Reading from or writing to a file (e.g., a pipe)

Technical Survey: Communication

- Interprocess communication programming interface
  - Synchronous or asynchronous signal notification
  - IPC message queue
  - IPC shared memory
  - Pipe (or FIFO)
  - UNIX Socket
- OS interface to network communications
  - INET Socket

Technical Survey: Security

Basic Mechanisms

- Identification
  `/etc/passwd` and `/etc/shadow`, sessions (login)
  UID, GID, effective UID, effective GID
- Encryption, signature and key management
- Access control models
  - Discretionary (DAC), it is the default
  - Mandatory (MAC), systematic controls
  - Role-based (RBAC) and Rule-Based (RB-RBAC)
    - Linux has *capabilities*: e.g., a process not owned by `root` may be granted permission to change ownership of an other user's file (`man 7 capabilities`)
    - Another example: network routing tables
- Logging: `/var/log` and `syslogd` daemon

Enhanced Security

- SELinux: `http://www.nsa.gov/selinux/papers/policy-abs.cfm`
- Defining a security policy
- Enforcing a security policy

Technical Survey: Virtualization

"Every problem can be solved with an additional level of indirection"

Standardization Purposes

- Common, portable interface
- Software engineering benefits (code reuse)
- Example: Virtual File System in Linux
  - Software layer below POSIX I/O system calls
  - Superset API for the features found in UNIX file systems
  - Also supports pseudo file systems (`/proc`, `/sys`, `/dev`, `/dev/shm`...)
  - Also supports foreign and legacy file systems (FAT, NTFS, ISO9660)
- Another example: drivers with SCSI emulation (USB mass storage)

Technical Survey: Virtualization

"Every problem can be solved with an additional level of indirection"

Compatibility Purposes

- Binary-level compatibility
  - Processor and full-system virtualization: emulation, binary translation (*subject of the last chapter*)
  - Protocol virtualization: IPv4 on top of IPv6
- API-level compatibility
  - POSIX (even Windows is more or less POSIX compliant)
  - Relative binary compatibility across some UNIX flavors (e.g., FreeBSD)

Operating System Trends

Design

- Modularity
  - Linux kernel modules (`/lib/modules/*.ko`) and Windows kernel DLLs
  - Run specific functions on behalf of the kernel or a process
- Beyond modularity: microkernel
  - Execute most of the OS code in user mode (debug, safety, adaptiveness)
  - The kernel only implements synchronization, communication, scheduling and low-level paging
  - User mode system processes implement memory management, device drivers and system call handlers (through specific access authorizations)
  - Examples: MACH (MacOSX), Chorus
  - Drawbacks
    - Message passing overhead (across processes and layers)
    - Most of the advantages can be achieved through modularization

Operating System Trends

Adaptation for Performance and Security

- Better support for NUMA
  - Affinity to a core/processor/node
  - Paging and scheduling aware of physical distribution of memory
  - Linux 2.6.18 is already quite sophisticated (thanks to the SGI Altix port)
- Tuning of kernel policies
  - Custom process and I/O scheduling, paging, migration...
    E.g., IBM Research's K42 linux-compatible kernel
  - Access control policies
    E.g., SELinux (sponsored by the NSA)