Imperial College London

Operating Systems

Introduction

Course 211
Spring Term 2016-2017

http://www.imperial.ac.uk/computing/current-students/courses/211/calendar/

Peter Pietzuch

prp@doc.ic.ac.uk
http://www.doc.ic.ac.uk/~prp

Course Objectives

What is an operating system, what defines it and distinguishes it from other features of the computer?

Common concepts that underlie operating systems

Study characteristics of operating systems

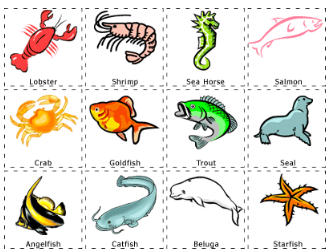
Detailed investigation of the implementation of some key features of the major operating systems

Concurrent programming concepts; concurrency versus parallelism and synchronisation

Outline (covered by Peter Pietzuch)

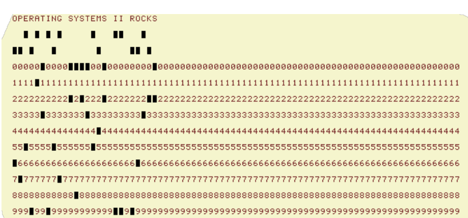
Overview and Revision

- OS structure
- Case studies



Memory Management

- Memory allocation
- Virtual memory
- Paging



Device Management (I/O)

- Types of device I/O
- Device drivers

Outline (covered by Peter Pietzuch)

File Systems

- Files and directories
- Implementation of file systems (FAT, ext2)

Security

- Access Control
- Attacks and defenses

Guest Lecture:
TBA

Outline (covered by Eno Thereska)

Processes and Threads
- Basic concepts
- Scheduling

Inter-process Communication (IPC)
- Main concepts and problems
- Synchronization primitives

Disk Management (I/O)
- Scheduling and caching
- RAID

Course Structure

- 3 combined lectures + tutorial per week (Weeks 2 – 10)
  - Lectures:
    - Mondays 10am-11am LT308 + Tuesdays 2pm-4pm LT311
    - Pintos OS Lab

- Course content what is presented verbally during lectures
- All handouts available before LT from course web page
  - http://www.imperial.ac.uk/computing/current-students/courses/211/calendar/
  - Solution notes for tutorials available following week

- Please ask questions!

Acknowledgement:
Slides based on material by Julie McCann and Daniel Rueckert

Recommended Books

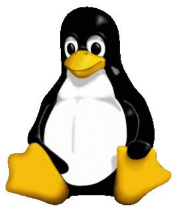
Recommended Books:

1. **Modern Operating Systems**, Tanenbaum, 3nd edition, Prentice Hall, 2008
2. **Operating Systems, Design and Implementation**, Tanenbaum & Woodhull, 2nd edition, Prentice Hal, 1997
3. **Operating Systems**, Deitel, Deitel, Choffnes, 3rd edition, Pearson/Prentice-Hall, 2004
4. **Operating Systems – Internals and Design Principles**, W. Stallings, 5th Edition, Prentice Hall, 2005
5. **Operating System Concepts**, A. Silberschatz, P. Galvin, G. Gagne, 7th Edition, John Wiley & Sons, 2005

☛ Important: Don't just rely on these slides!

Further Reading

Will discuss details of OS design/implementation using examples based on Linux



GNU/Linux:

- **Understanding the Linux kernel** – Bovet & Cesati, O'Reilly, 3rd edition, 2005
- `man` command (every Linux machine)

Windows NT/XP/Vista/7:

- **Inside Windows NT & 2000** – D. A. Solomon, 2nd & 3rd edition, Microsoft Press, 1998 & 2000
- **Operating System Projects using Windows NT** – G. Nutt, Addison Wesley, 1999

OS Overview

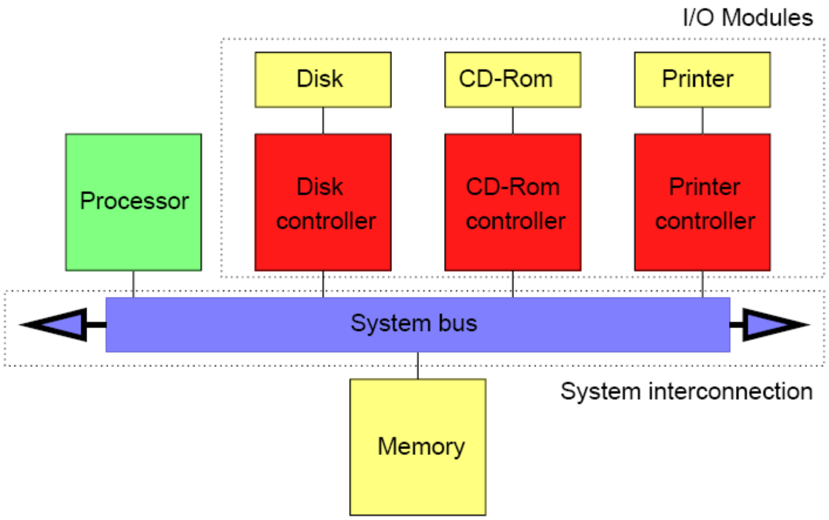
Computer Architecture Overview

Processor

- Controls computer hardware
  Executes instructions and programs

Memory

- Stores data and programs



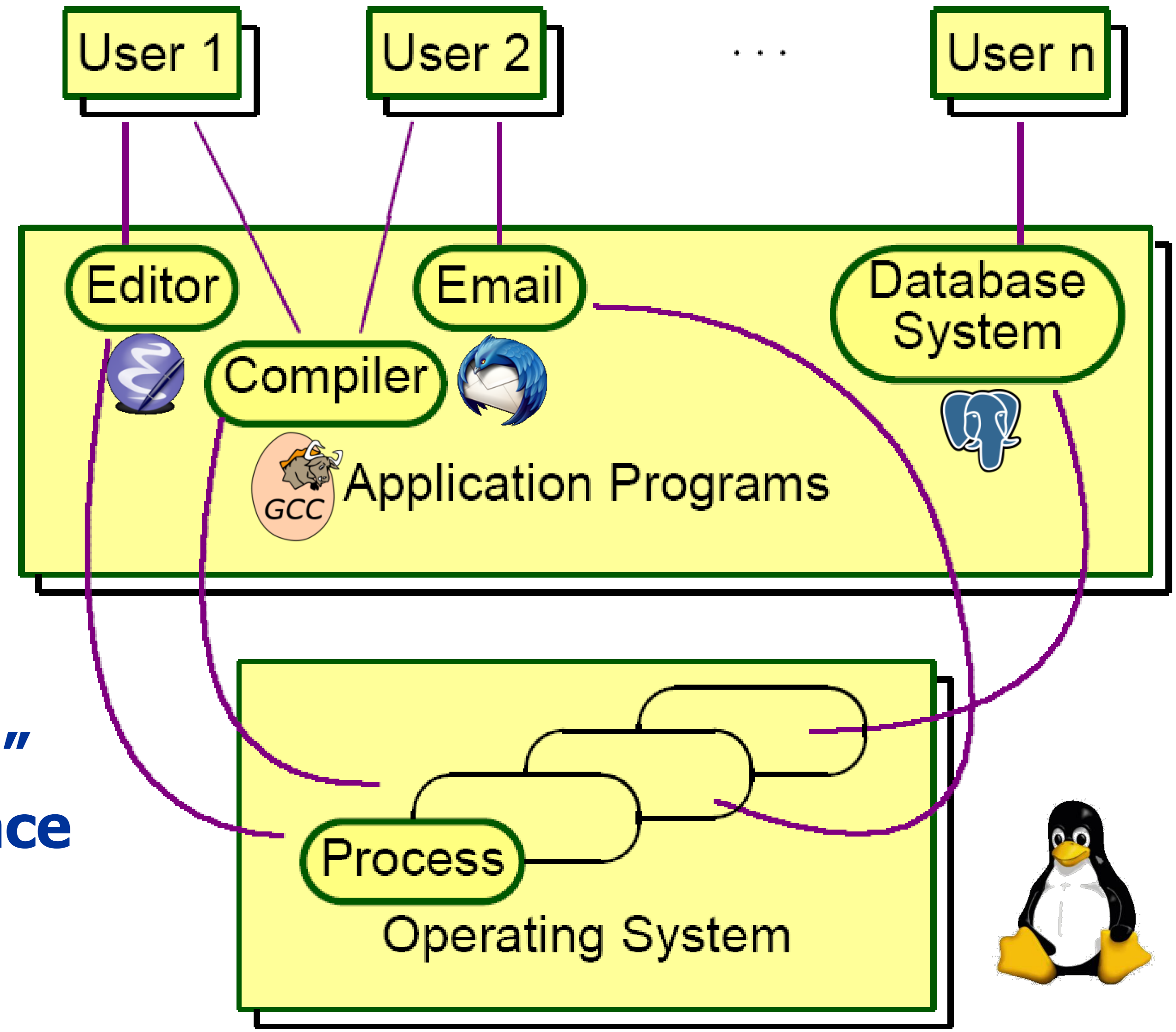
I/O components

- Read and write from
  I/O devices
- Intelligence in I/O controller

System bus

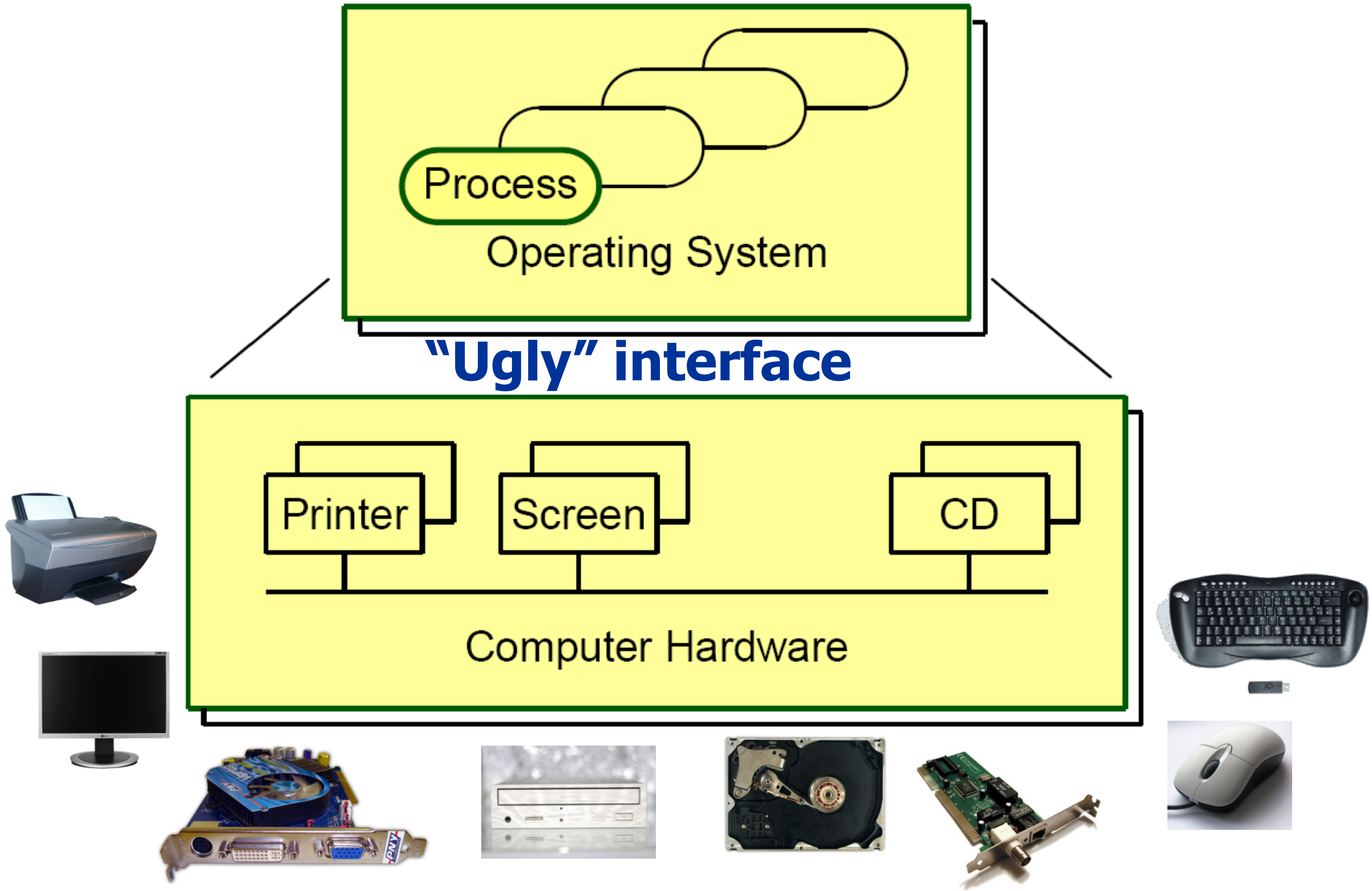
- Interconnects different hardware components
- Provides communication between components

Operating Systems – Top Level View



"Clean" interface

Operating Systems – Bottom Level View



1. Resource Management

Making efficient use of (limited) available resources

- Time, space, money, ...

Sharing resources among multiple users

- Schedule access, fair allocation
- Prevent interference

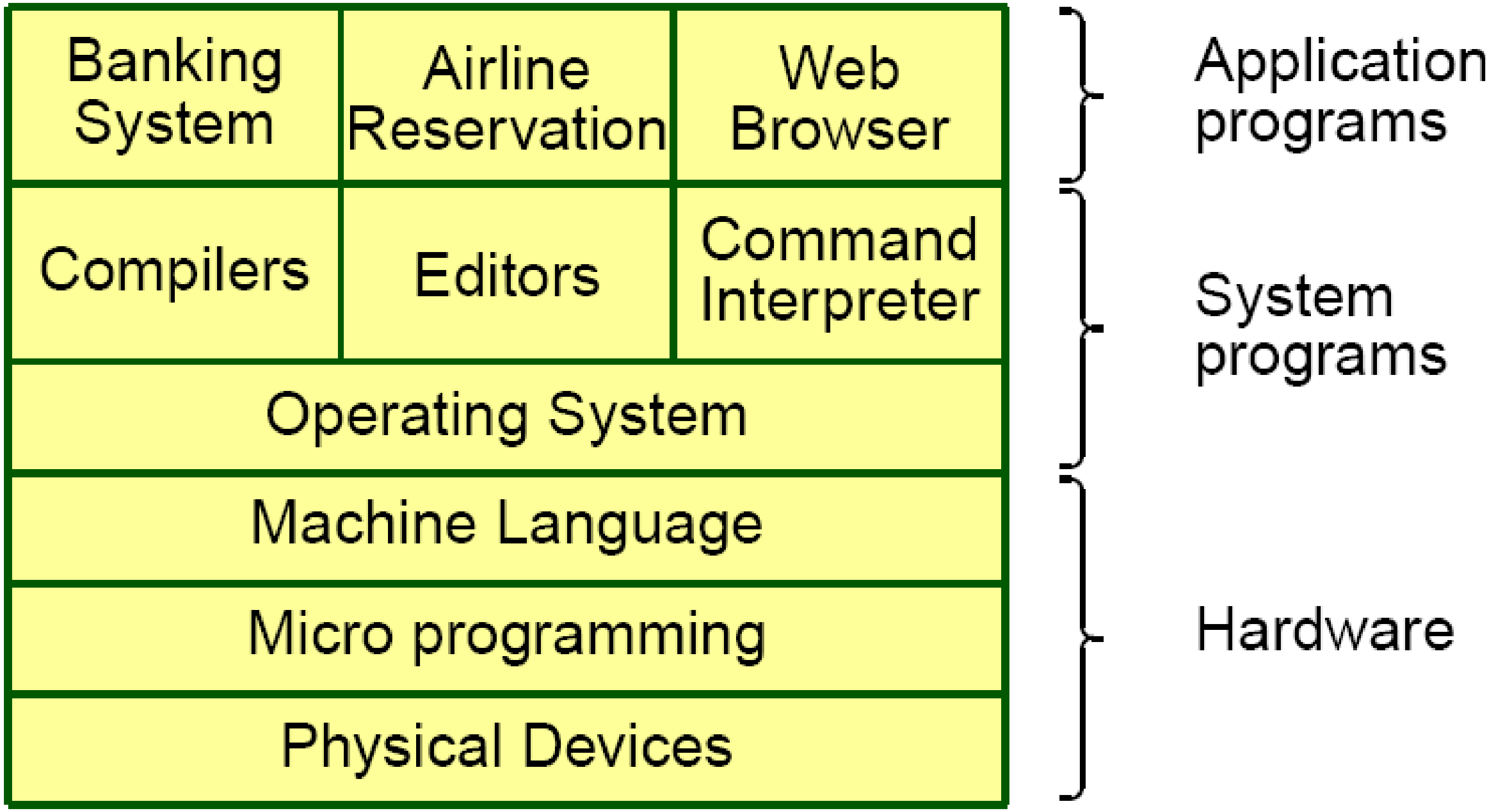
Resources

- Processors
  - Divide number and/or time
- Memory
  - RAM, cache, disks, ...
- Input/Output devices
  - Screens, printers, network cards, ...
- Internal devices
  - Clocks, timers, ...
- Long-term storage (files)
  - Disks, DVD, CDs, tapes, ...
- Software
  - Browsers, editors, e-mail clients, databases

2. Providing Clean Interfaces

OS converts raw hardware into usable computer system

- Hides complexity of lower levels from higher levels



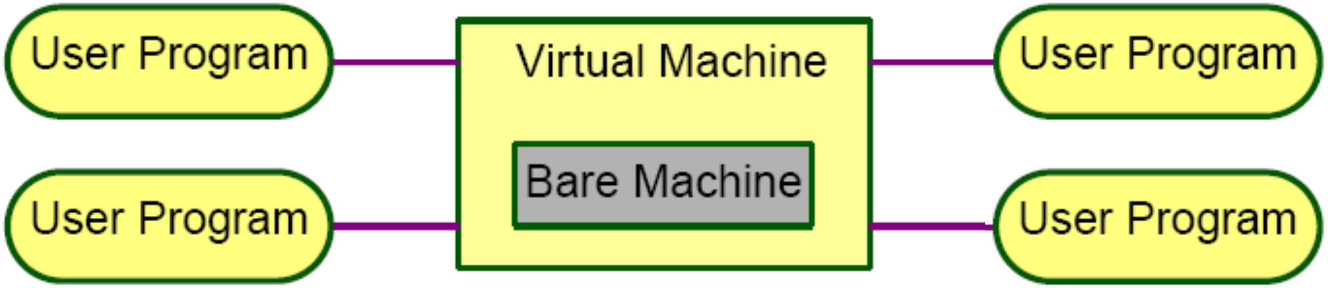
Virtual Machine Abstraction

Details of hardware kept hidden from programs

Only OS can allow access to hardware resources

User request should be abstract

– e.g. no need to know how files stored on disk



OS Characteristics: Sharing

Sharing of data, programs and hardware

- Time multiplexing and space multiplexing

Resource allocation

- Efficient and fair use of memory, CPU time, disk space, ...
- Simultaneous access to resources
  - Disks, RAM, code, network, CPU, ...
- Mutual exclusion
  - Guarantees that risky operations are protected
  - Multiple writes to same file?
- Protection against corruption
  - Accidental or malicious

OS Characteristics: Concurrency I

Several simultaneous parallel activities

- Overlapped I/O & computation
- Multiple users and programs run in parallel

Switch activities at arbitrary times

- Guarantee fairness and prompt replies

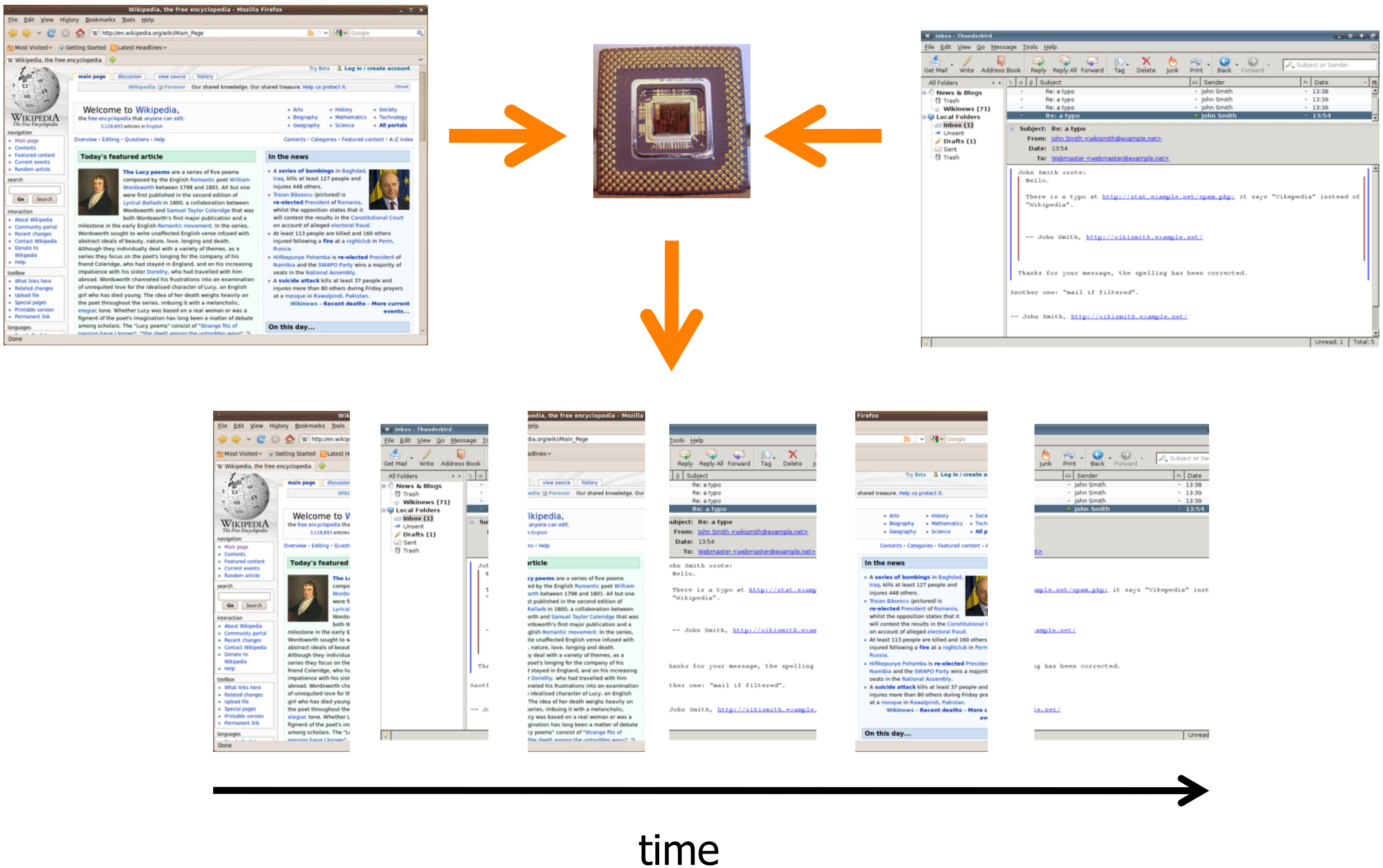
Safe concurrency

- Synchronisation of actions
  - Avoids long waiting cycles; gives accurate error handling
- Protection from interference
  - Each process has its own space

OS Characteristics: Concurrency II

Time-slicing

– Switch application running on physical CPU every 50ms



OS Characteristics: Non-determinism

Non-determinism

- Results from events occurring in unpredictable order
  - e.g. timer interrupts, user input, program error, network packet loss, disk errors, . . .
- Makes programming OS hard!

OS Characteristics: Storing Data

Long term storage: File systems for disks, DVDs, ...

- Easy access to files through user-defined names
  - Directory structure, links, shared disks
- Access controls
  - Read, write, remove, execute or copy permissions
- Protection against failure (backups)
  - Daily/weekly/monthly, partial/complete
- Storage management for easy expansion
  - Add disks without need for re-compilation of OS

OS Facilities I

Simplified I/O
- Access to disk, DVD or remote file server

Virtual memory
- Larger than physical memory and partitioned

File systems
- Long term storage on disk accessed by names

Program interaction and communication
- Messages, pipes, sockets, shared memory

Network communication
- Sending/receiving data on network

OS Facilities II

Security

- Prevent programs from accessing resources not allocated to them

Human-computer interface

- User interaction with programs, command language, shells

Administration, management & accounting

Operating System Zoo

Multiprocessor (eg Windows, Mac OS X, Linux)

- Many cores and CPUs (multiprocessor)
- Today's desktops/laptops - you all use this

Server OS (e.g. Solaris, FreeBSD, Linux and Windows Server 200x)

- Share hardware/software resources e.g. internet servers

Mainframes, supercomputers (e.g. OS/390)

- Bespoke hardware, limited workloads

Smartphones (e.g. iOS, Android)

- Simpler CPUs, starting to be sophisticated

Embedded OS (e.g. QNX, VXWorks)

- Home utilities
- Only trusted software

Real-time OS

- Time oriented (not performance or I/O)

Sensor Network OS (e.g. TinyOS)

- Resource/energy conscious

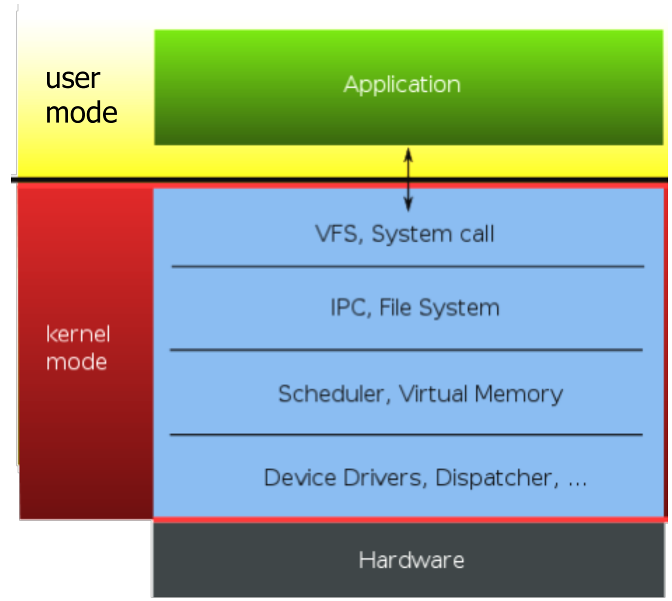
OS Structure

Monolithic OS kernels (e.g. Linux, BSD, Solaris, ...)
- Single black box

Microkernels (e.g. Symbian, L4, Mach, ...)
- Little as possible in kernel (fewer bugs)

Hybrid kernels (e.g. Windows NT, Mac OS X, ...)
- Take a guess... ☺

Monolithic Kernels



Kernel is single executable with own address space

- Structure implied through pushing parameters to stack and trap (systems calls)
- Most popular kernel style

Advantages

- Efficient calls within kernel
- Easier to write kernel components due to shared memory

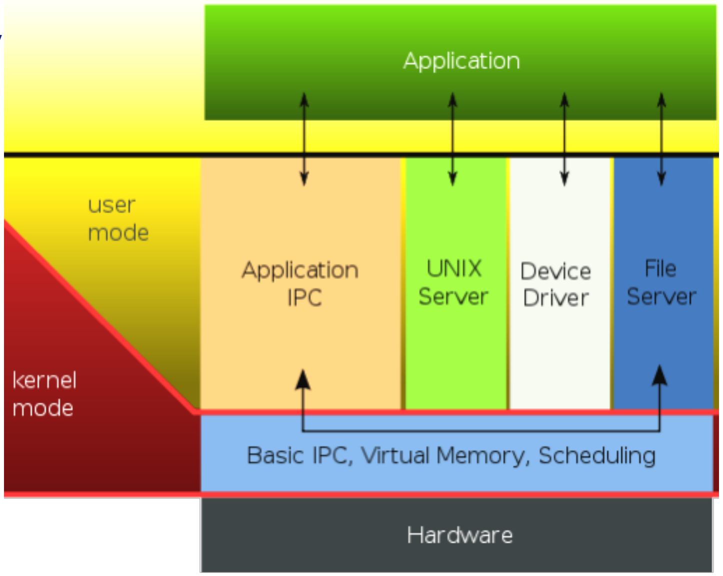
Disadvantages

- Complex design with lots of interactions
- No protection between kernel components

Microkernels

Minimal kernel with functionality in user-level servers

- Kernel does IPC (message-passing) between servers
- Servers for device I/O, file access, process scheduling, ...



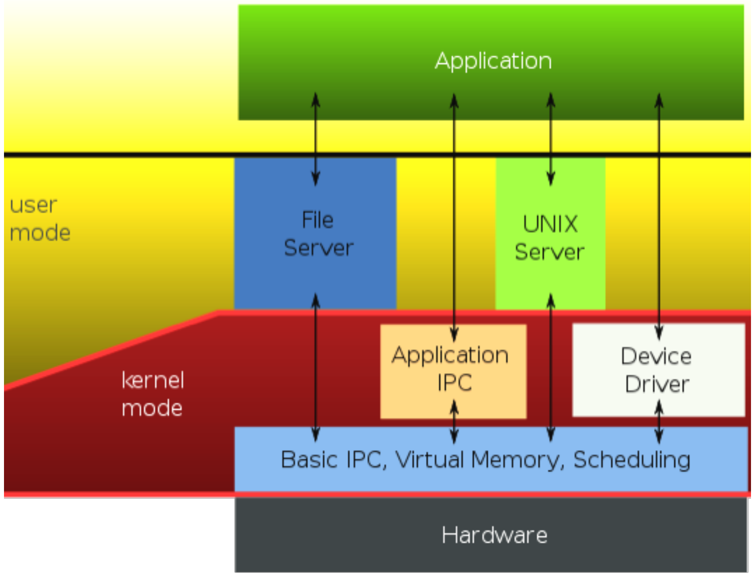
Advantages

- Kernel itself not complex ➔ less error-prone
- Servers have clean interfaces
- Servers can crash & restart without bringing kernel down

Disadvantages

- Overhead of IPC within kernel high

Hybrid Kernels



Combines features of both monolithic and microkernels

- Often just a design philosophy

Advantages

- More structured design

Disadvantages

- Performance penalty for user-level servers

Introduction to Linux

Linux History and Motivation

Variant of Unix like FreeBSD, System V, Solaris etc.

- Ken Thomson left Multics (Bell Labs) – Uniplexed information and computing service
- Dennis Ritchie got interested

Late 80's: 4.3 BSD and System V r3 dominant

- Systems call libraries reconciliation POSIX

1987 Tanenbaum released MINIX microkernel

- Tractable by single person (student)

Linus Torvalds, frustrated, built fully-featured yet monolithic version

- Major goal was interactivity, multiple processes and users
- Code contributed by world-wide community

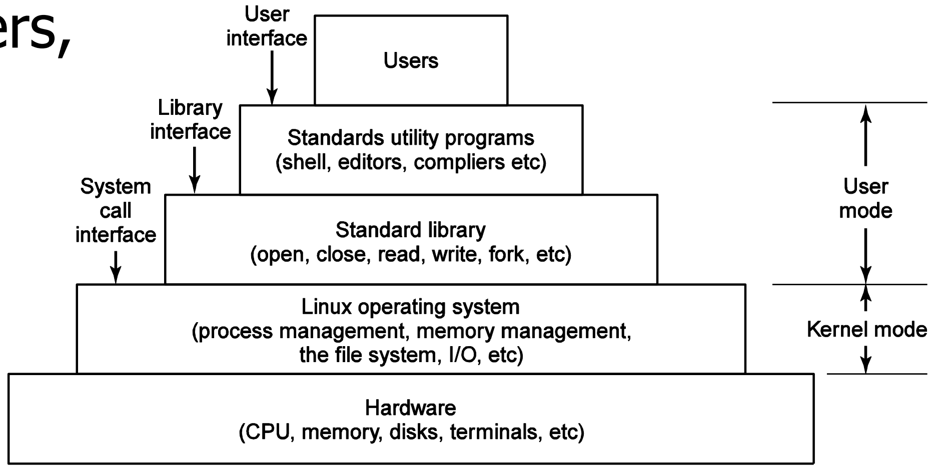
Structure and Interfaces

Systems calls

- Implemented by putting arguments in registers (or stack)
- Issue trap to switch from user to kernel

Rich set of programs (through GNU project)

- e.g. shells (bash, ksh, ...), compilers, editors, ...
- Desktop environments: GNOME, KDE, ...
- Utility programs: file, filters, editors, compilers, text processing, sys admin, etc



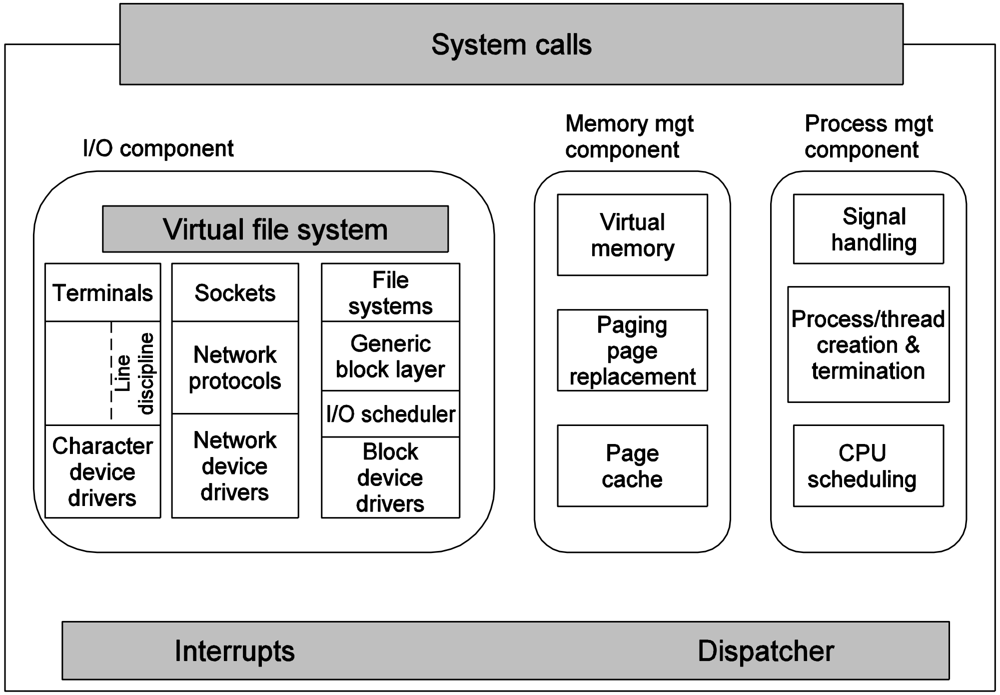
Kernel Structure

Interrupt handlers primary means to interact with devices

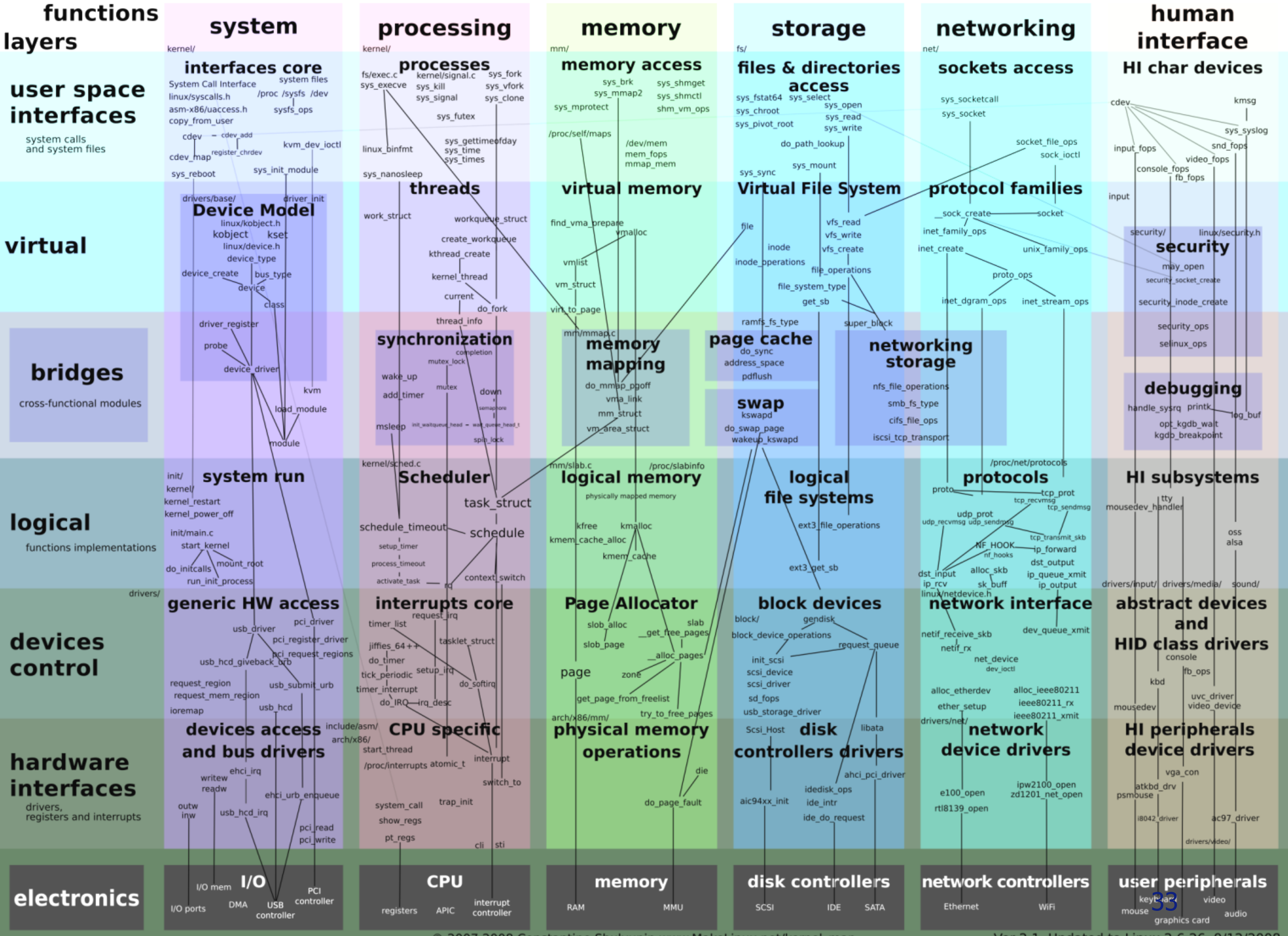
- Kicks off dispatching
  - Stop process, save state and start driver and return
- Dispatcher written in assembler

IO scheduler orders disk operation

Static in-kernel components and dynamically loadable modules



Linux kernel map



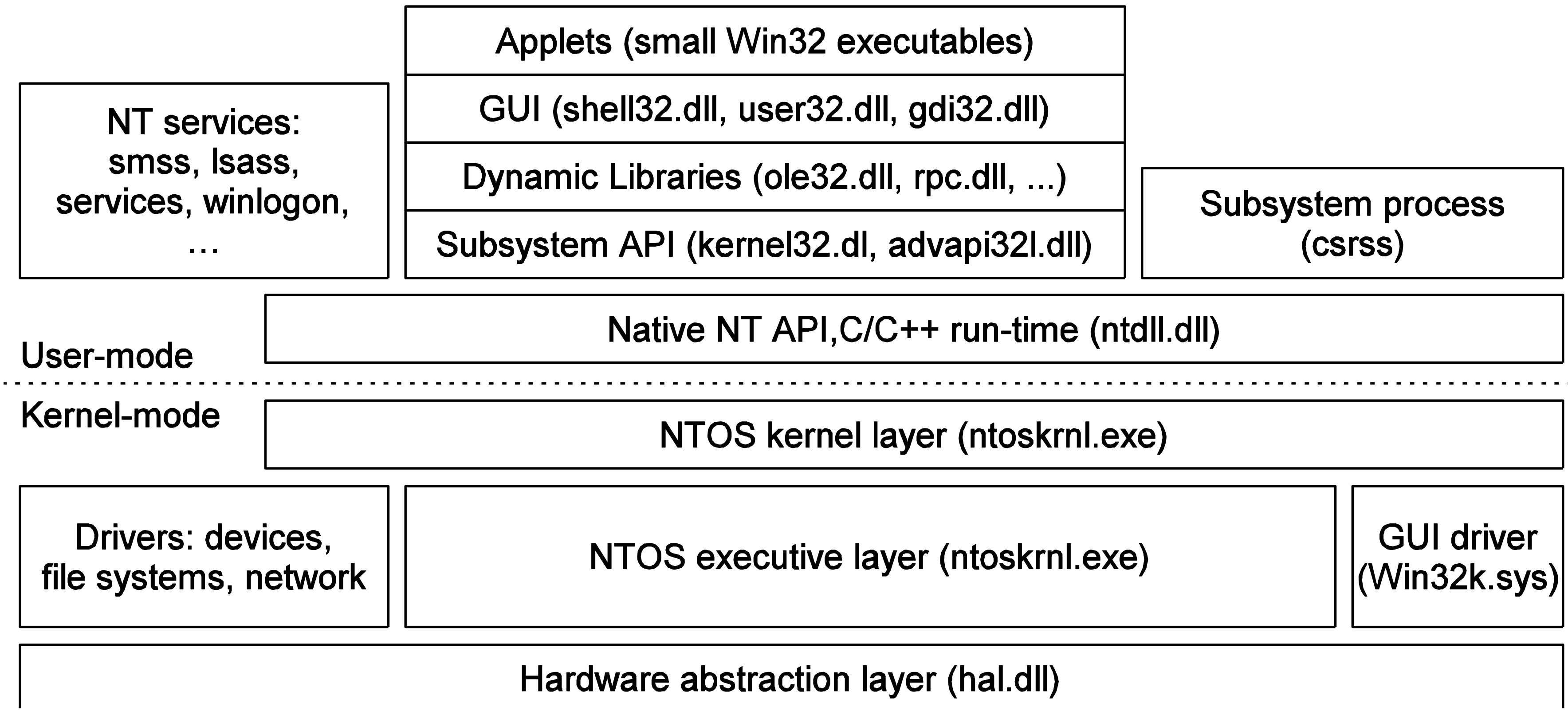
Introduction to Windows

History and Motivation

- 1980s – Microsoft bought CP/M OS for IBM PC
  - Called it MS-DOS and added Windows in the 90's
- 1993 - Windows NT
  - Inspired by VMS: 32-bit OS, to be more portable
- 2000 - Windows 2000
  - Plug-and-play, network directory services, power management, GUI
- 2001 - Windows XP
  - Split into client/server releases
- 2006 - Windows Vista
  - GUI design, security
- 2009 – Windows 7
  - Focus on stability, reliability, hardware support
- 2012 – Windows 8.x, 10
  - Better support for touch screen devices, integration with cloud services

Structure and Interfaces

- NTOS provides system calls
- Programs build on top of dynamic code libraries (DLLs)
  - Implement OS services in modular fashion



NT Kernel Structure I

NTOS is loaded from ntoskrnl.exe at boot

Two layers:

- Executive: most of the services
- Kernel: thread scheduling and synchronisation, trap, interrupts and CPU management

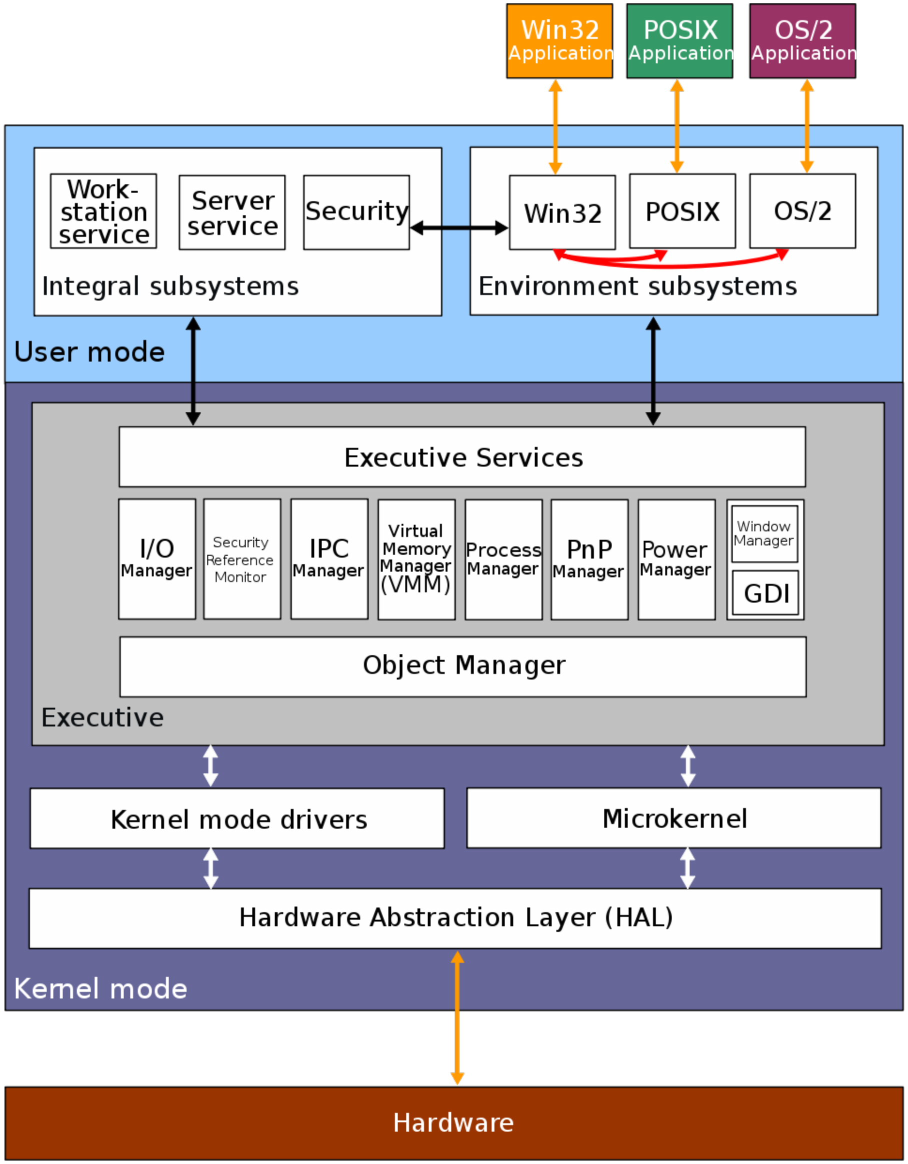
Hardware Abstraction Layer (HAL)

- Abstracts out DMA operations, BIOS config, CPU types

IO/VM load device drivers into kernel memory

- Dynamically links to NTOS and HAL layers

NT Kernel Structure II



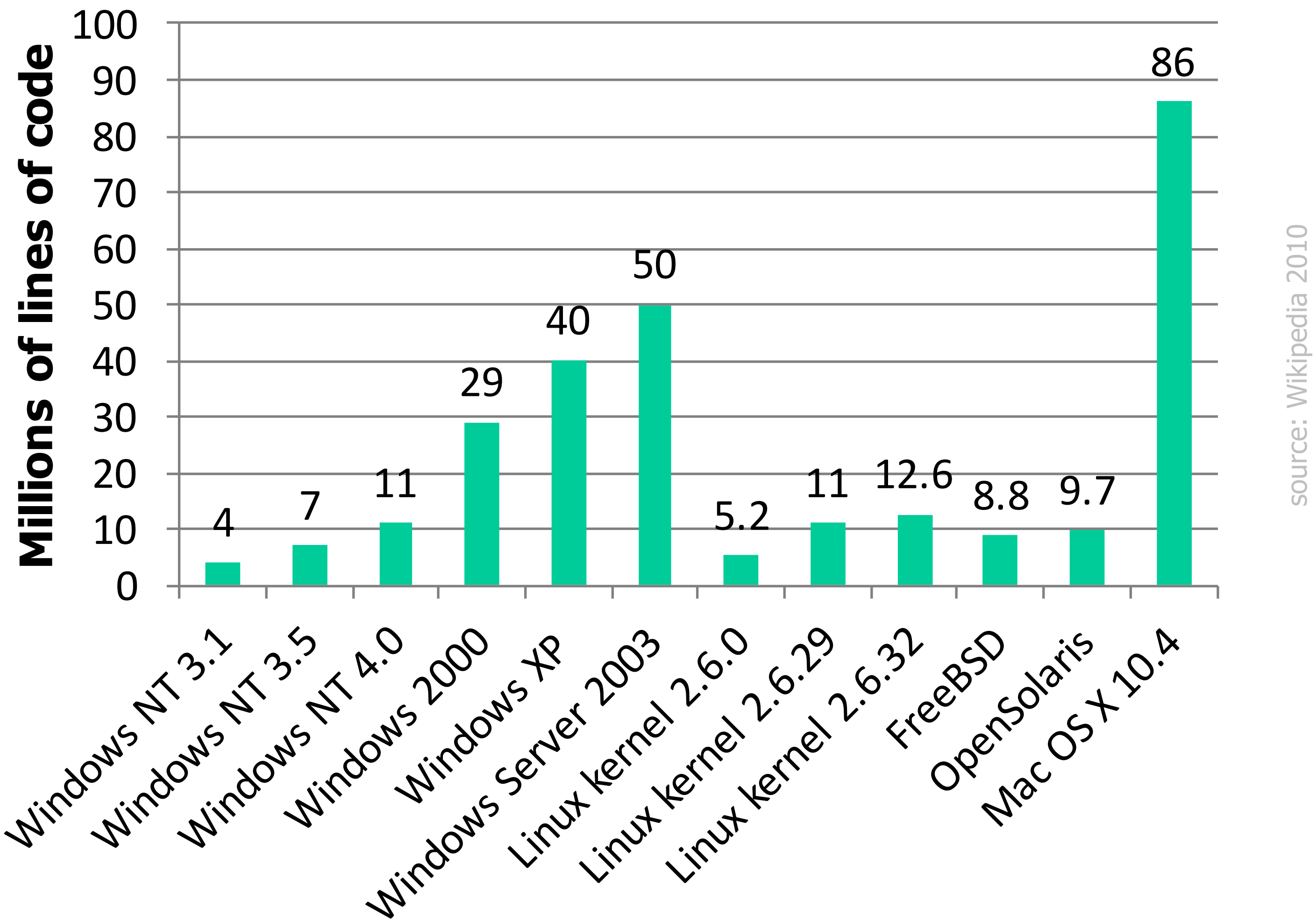
Structured like microkernel

- Individual servers provide functionality
- Emulation subsystems run in user-space

Most components run in same kernel address space

- More efficient implementation
- Bugs can bring down whole kernel

Evolution of OS Code Sizes



Code bloat

– Is lines of code useful comparison for complexity?