

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

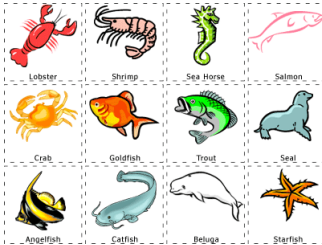
Outline (covered by Peter Pietzuch)

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!

Recommended Books

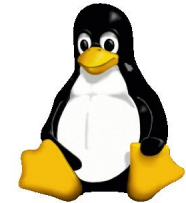
Recommended Books:

1. **Modern Operating Systems**, Tanenbaum, 3rd 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

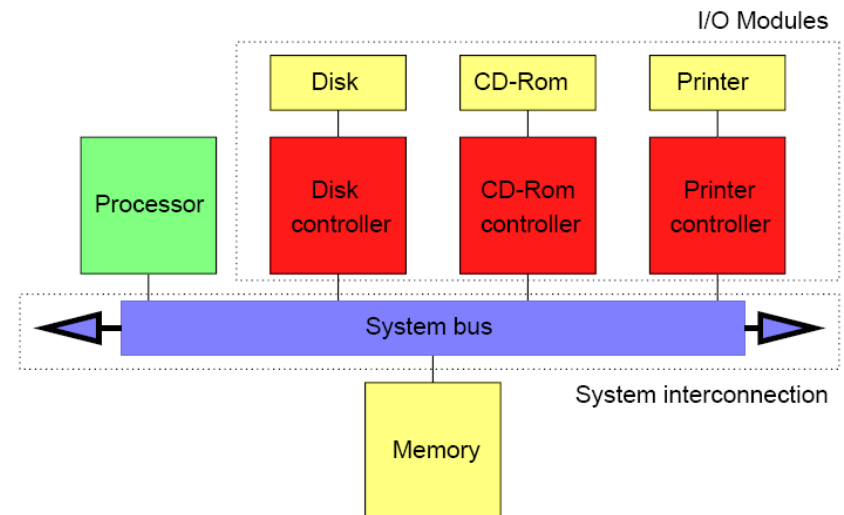
- Stores data and programs

I/O components

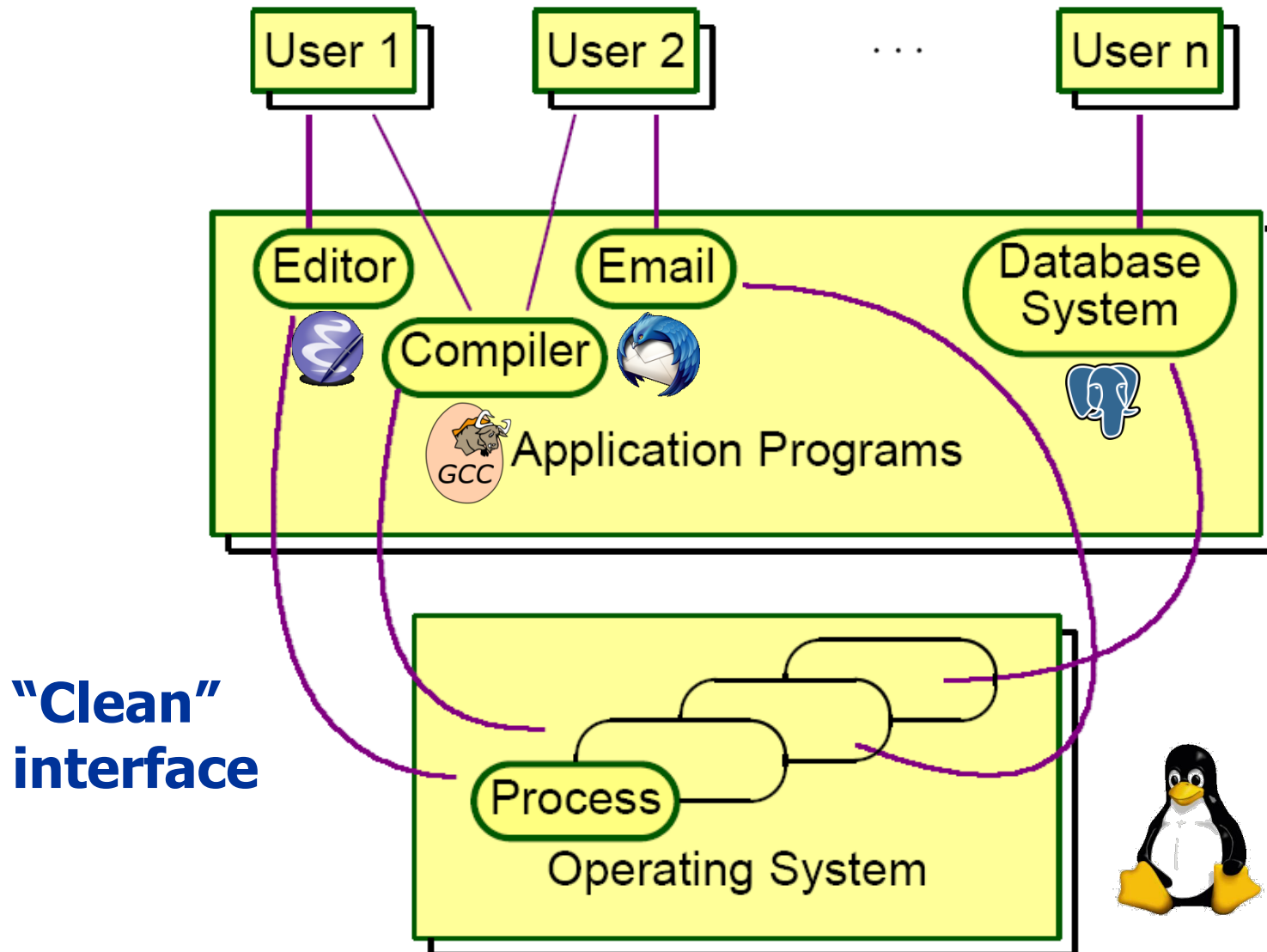
- Read and write from I/O devices
- Intelligence in I/O controller

System bus

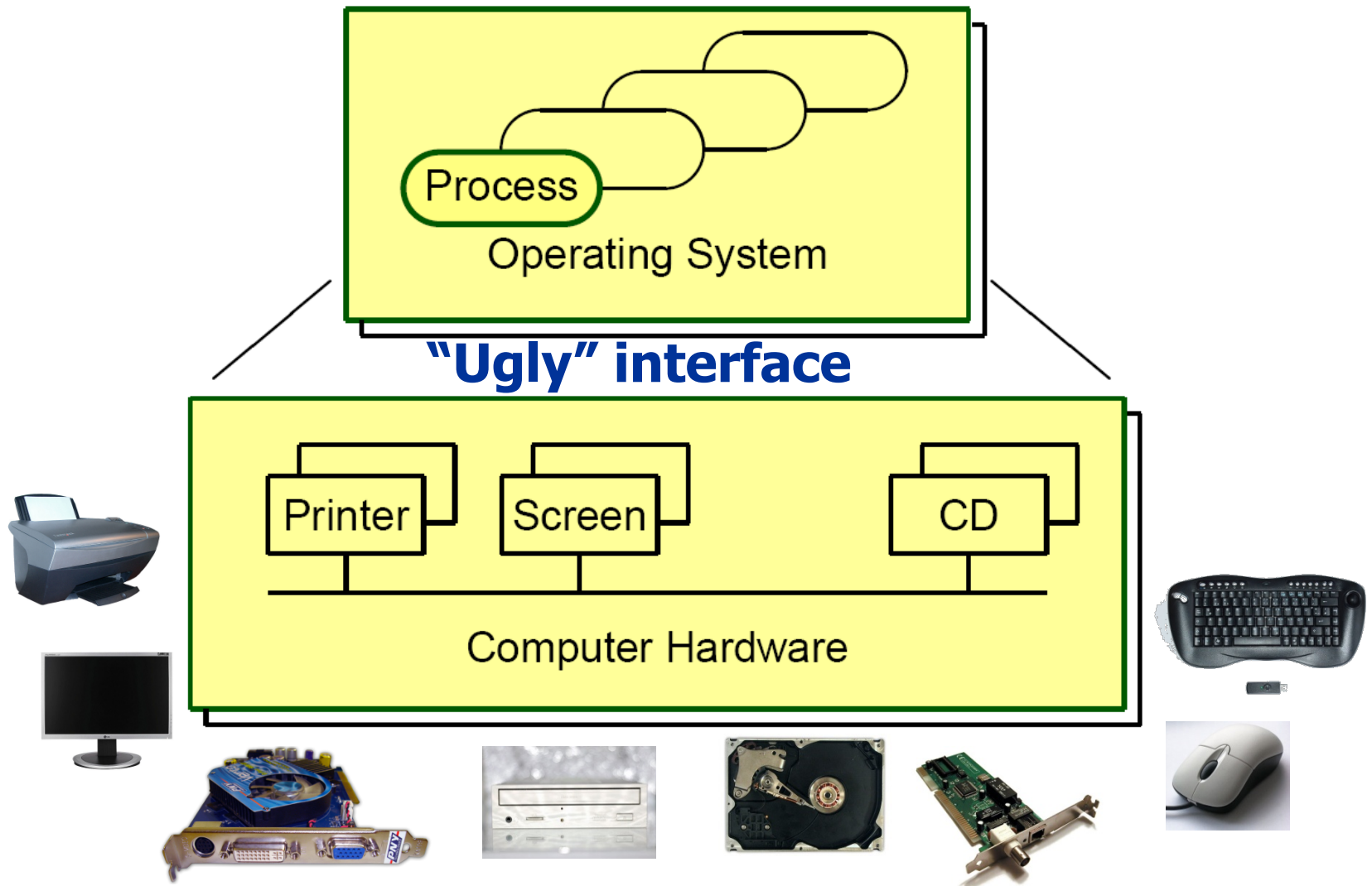
- Interconnects different hardware components
- Provides communication between components



Operating Systems – Top Level View



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, ...

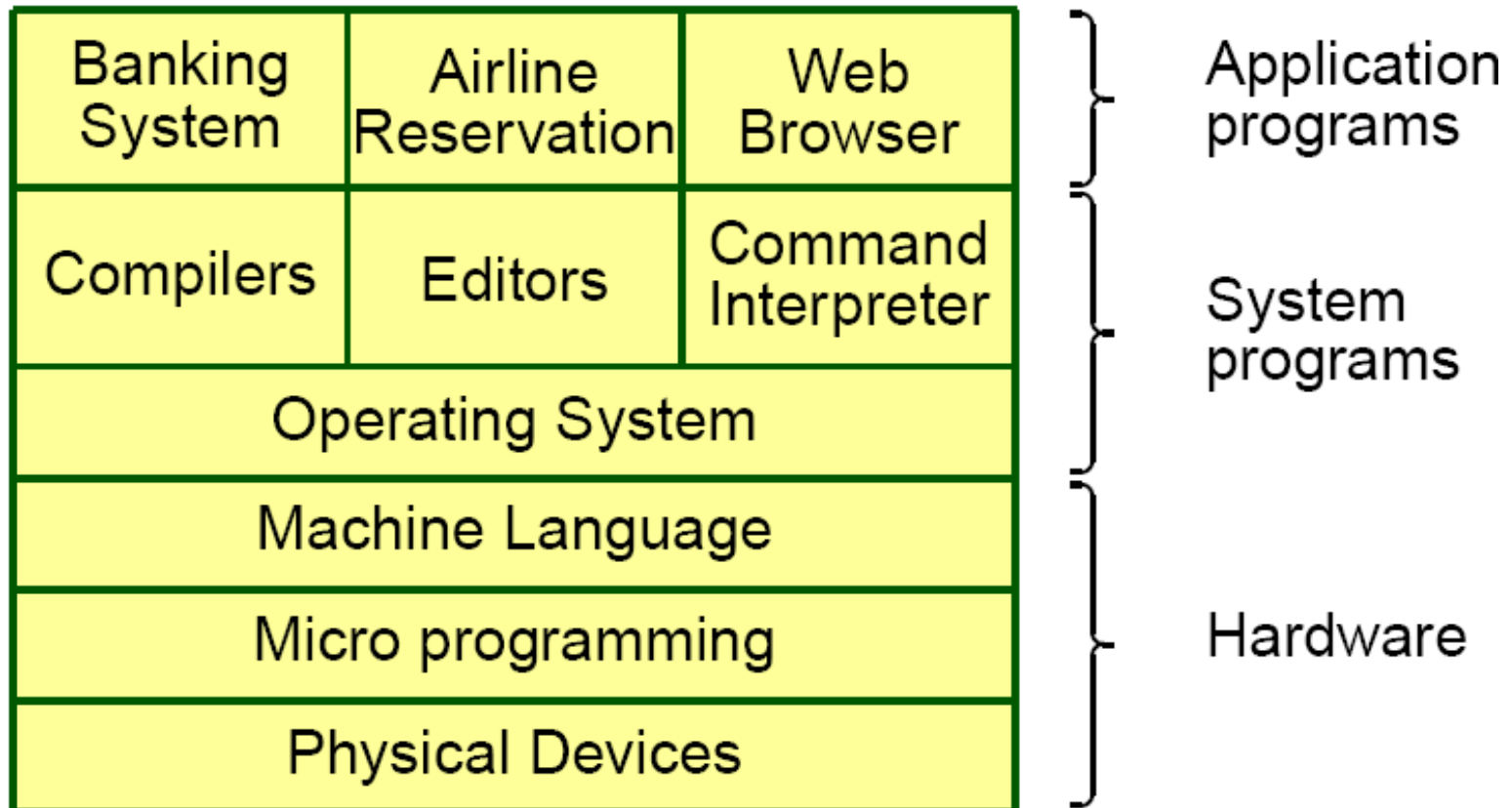
Software

- 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



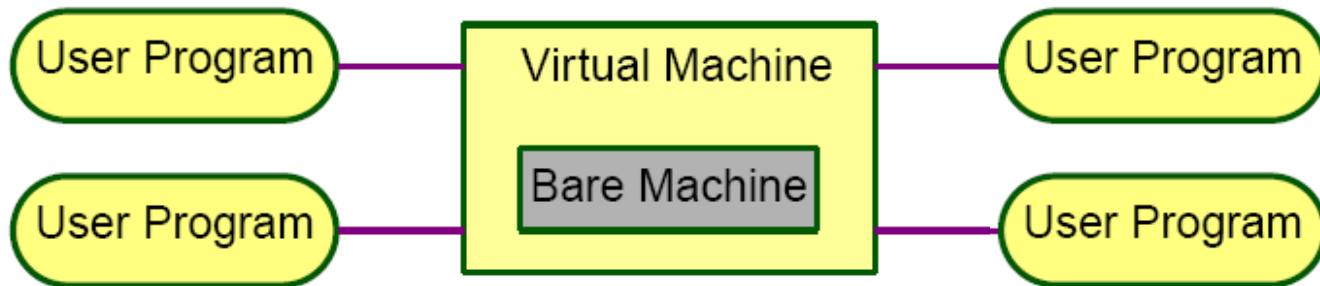
Virtual Machine Abstraction

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

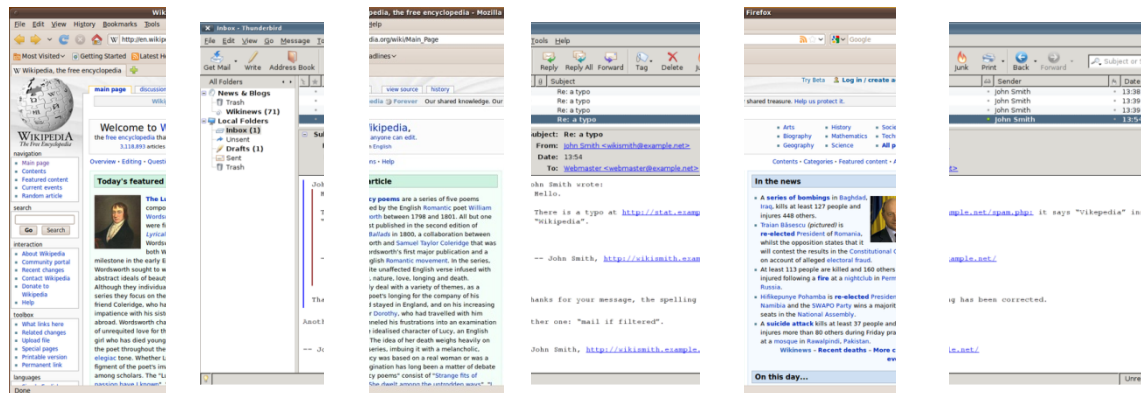
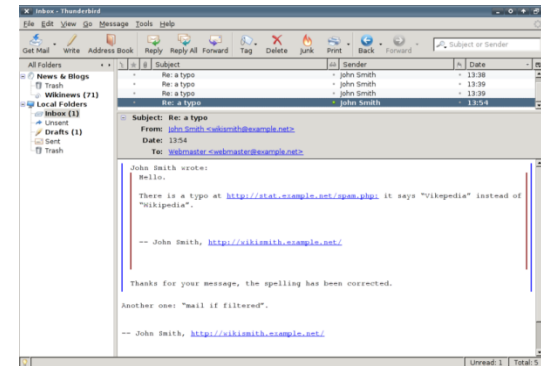
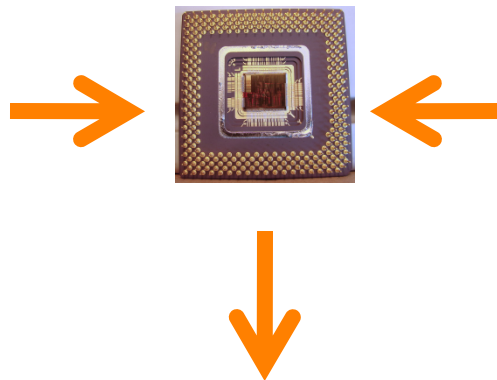
Safe concurrency

- 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



time

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

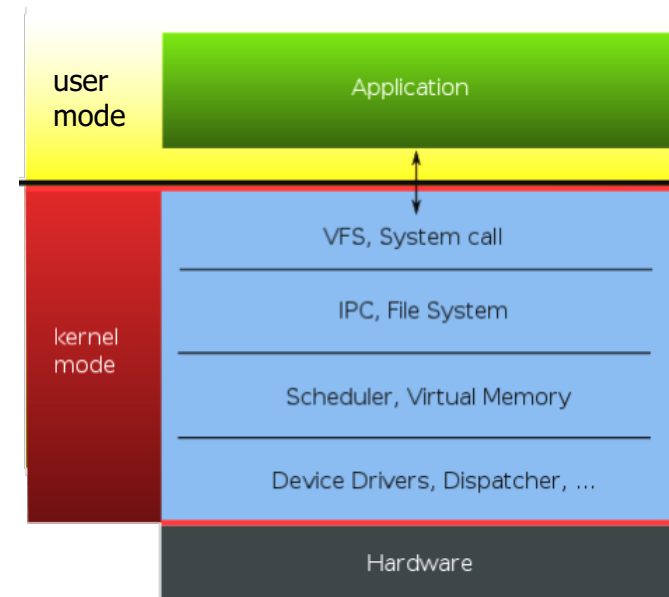
- Structure implied through pushing parameters to stack and trap (systems calls)
- Most popular kernel style

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

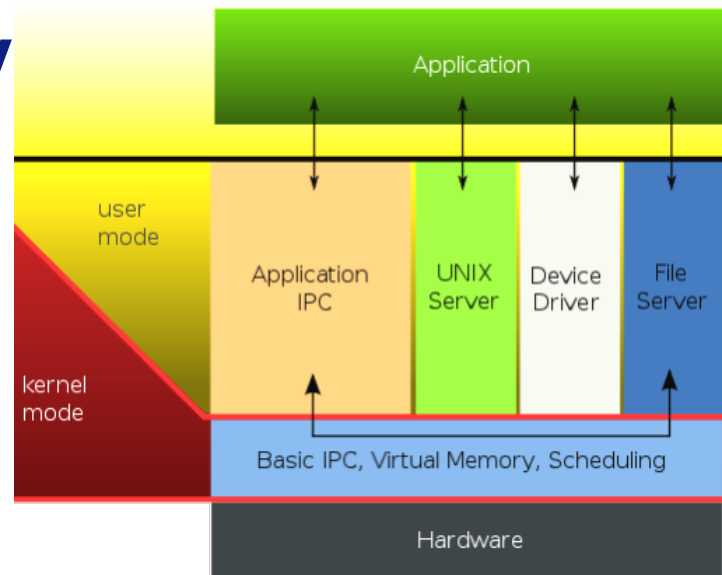
- Kernel does IPC (message-passing) between servers
- Servers for device I/O, file access, process scheduling, ...

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

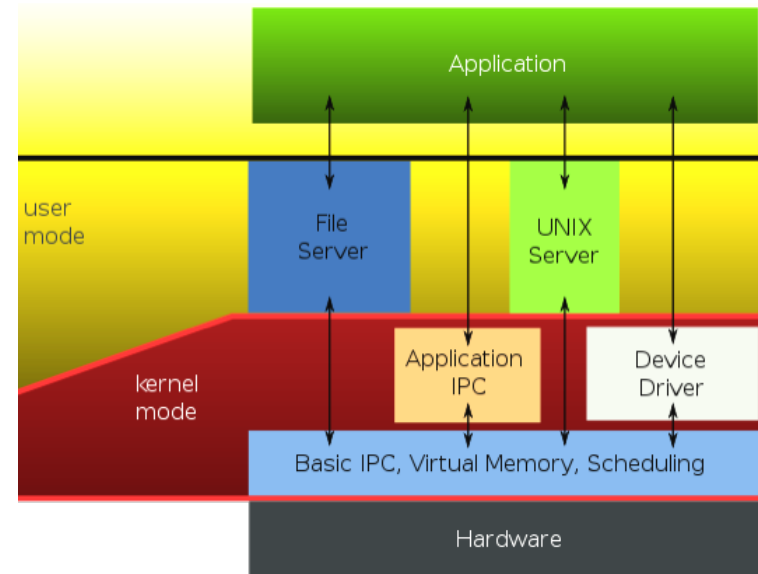
- Often just a design philosophy

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

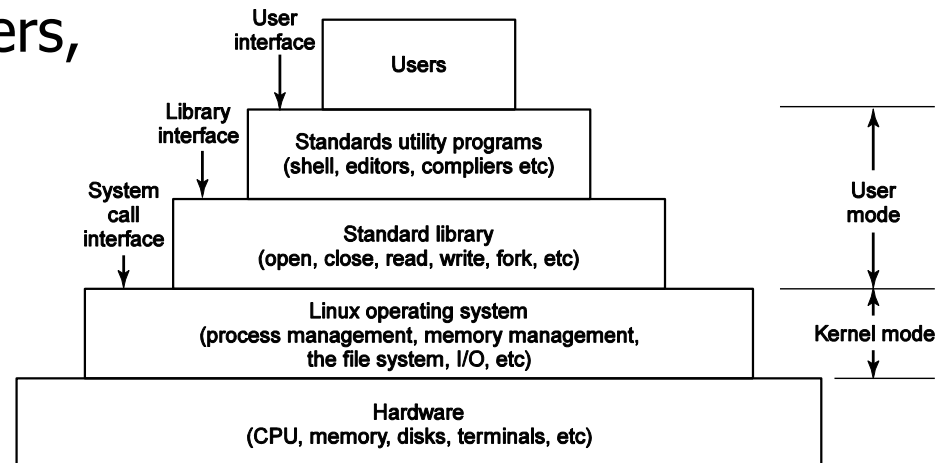
Structure and Interfaces

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



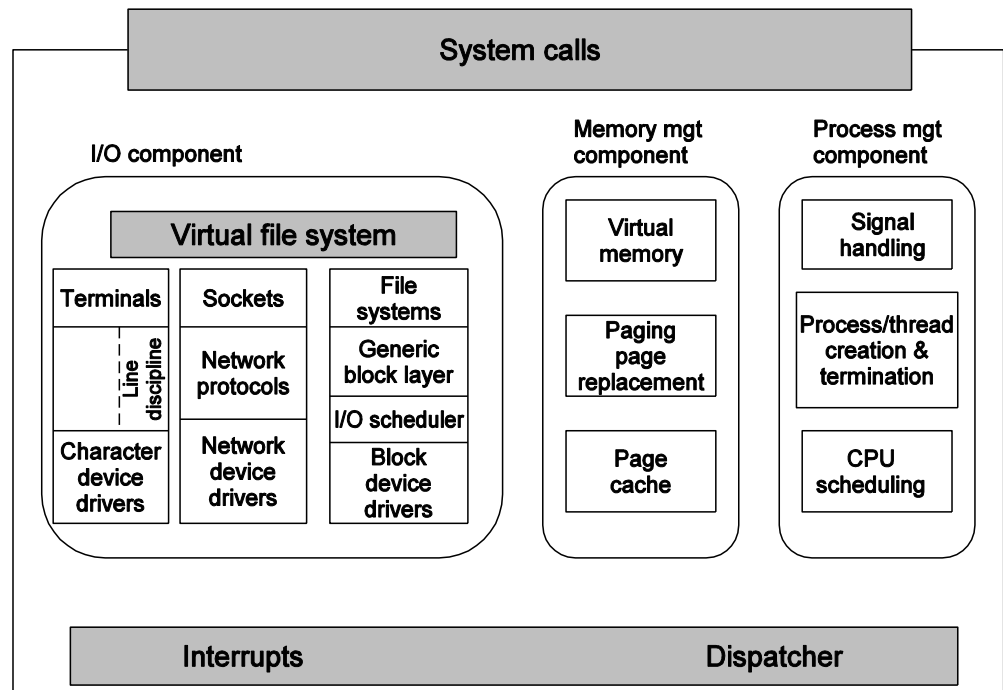
Kernel Structure

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

functions
layers

user space
interfaces

virtual

bridges

logical

devices
control

hardware
interfaces

electronics

system

kernel/
interfaces core
System Call Interface
linux/syscalls.h /proc /sysfs /dev
asm-x86/uaccess.h sysfs_ops
copy_from_user
cdev → cdev_add
cdev_map register_chrdev
sys_reboot
kvm_dev_ioctl
sys_init_module

Device Model

drivers/base/
linux/kobject.h kobject kset
linux/device.h device_type
device_create bus_type
device class
driver_register
probe
device driver
kvm
load_module
module

system run

init/
kernel/
kernel_restart
kernel_power_off
init/main.c
start_kernel
do_initcalls
run_init_process
mount_root

generic HW access

drivers/
usb_driver pci_driver
pci_register_driver
pci_request_regions
usb_hcd_giveback_irq
usb_submit_urb
request_region
request_mem_region
ioremap
usb_hcd

**devices access
and bus drivers**

ehci_irq
ehci_urb_enqueue
usb_hcd_irq
pci_read
pci_write
outw
inw
I/O mem
DMA
USB controller
PCI controller
I/O ports

I/O

I/O ports DMA USB controller PCI controller

processing

kernel/
processes
fs/exec.c sys_execve
kernel/signal.c sys_kill
sys_signal
sys_futex
sys_gettimeofday
sys_time
sys_times
sys_nanosleep
linux/binfmt
sys_fork
sys_vfork
sys_clone

threads

work_struct
workqueue_struct
create_workqueue
kthread_create
kernel_thread
current
thread_info
do_fork
thread_info
wake_up
add_timer
mutex_lock
completion
down
init_waitqueue_head
wait_queue_head_t
spin_lock
msleep

synchronization

kernel/sched.c
Scheduler
schedule_timeout
setup_timer
process_timeout
activate_task
task_struct
schedule
context_switch
timer_list
request_irq
tasklet_struct
do_timer
tick_periodic
timer_interrupt
do_IRQ
irq_desc

interrupts core

timer_list
request_irq
tasklet_struct
do_timer
tick_periodic
timer_interrupt
do_IRQ
irq_desc
atomic_t
interrupt
switch_to
cli
sti

CPU specific

include/asm/
arch/x86/
start_thread
/proc/interrupts
atomic_t
interrupt
switch_to
cli
sti

CPU

system_call
show_regs
pt_regs
trap_init
cli
sti

CPU

registers APIC interrupt controller

memory

mm/
memory access
sys_brk
sys_mmap2
sys_shmget
sys_shmctl
shm_vm_ops
sys_mprotect
/proc/self/maps
/dev/mem
mem_fops
mmap_mem

virtual memory

find_vma_prepare
vmalloc
vmlist
vm_struct
vm_to_page
mm/mmap.c
memory mapping
do_mmap_pgoff
vma_link
mm_struct
vm_area_struct

logical memory

mm/slab.c
physically mapped memory
kfree
kmallocc
kmem_cache_alloc
kmem_cache
slob_alloc
slob_page
_get_free_pages
_alloc_pages
page
zone
get_page_from_freelist
try_to_free_pages
arch/x86/mm/
die
do_page_fault

Page Allocator

physically mapped memory
kfree
kmallocc
kmem_cache_alloc
kmem_cache
slob_alloc
slob_page
_get_free_pages
_alloc_pages
page
zone
get_page_from_freelist
try_to_free_pages
arch/x86/mm/
die
do_page_fault

**physical memory
operations**

arch/x86/mm/
die
do_page_fault

memory

RAM MMU

memory

RAM MMU

storage

fs/
**files & directories
access**
sys_fstat64 sys_select
sys_chroot
sys_pivot_root
do_path_lookup
sys_sync
sys_mount
sys_open
sys_read
sys_write

Virtual File System

file
inode
inode_operations
file_operations
file_system_type
get_sb
super_block
ramfs_fs_type
do_sync
address_space
pdflush

page cache

do_sync
address_space
pdflush

swap

do_swap_page
wakeuptcp_kswapd

**logical
file systems**

ext3_file_operations
ext3_get_sb
gendisk
block_device_operations
request_queue
init_scsi
scsi_device
scsi_driver
sd_fops
usb_storage_driver
Scsi_Host
libata
ahci_pci_driver
ideisk_ops
ide_intr
ide_do_request
aic94xx_init

block devices

gendisk
block_device_operations
request_queue
init_scsi
scsi_device
scsi_driver
sd_fops
usb_storage_driver
Scsi_Host
libata
ahci_pci_driver
ideisk_ops
ide_intr
ide_do_request
aic94xx_init

**disk
controllers drivers**

ahci_pci_driver
ideisk_ops
ide_intr
ide_do_request
aic94xx_init

disk controllers

SCSI IDE SATA

networking

net/
sockets access
sys_socketcall
sys_socket
socket_file_ops
sock_ioctls

protocol families

_sock_create
inet_family_ops
inet_create
proto_ops
inet_dgram_ops
inet_stream_ops
socket
unix_family_ops

**networking
storage**

nfs_file_operations
smb_fs_type
cifs_file_ops
iscsi_tcp_transport

protocols

proto
udp_prot
udp_rcvmsg
udp_sendmsg
tcp_prot
tcp_rcvmsg
tcp_sendmsg
tcp_transmit_skb
ip_forward
dst_output
ip_queue_xmit
ip_output
NF_HOOK
nf_hooks
dst_input
ip_rcv
linux/netdevice.h
alloc_skb
sk_buff
netif_receive_skb
netif_rx
net_device
dev_ioctl
dev_queue_xmit

network interface

netif_receive_skb
netif_rx
net_device
dev_ioctl
dev_queue_xmit
alloc_etherdev
ether_setup
drivers/net/
alloc_ieee80211
ieee80211_rx
ieee80211_xmit

**network
device drivers**

e100_open
rt8139_open
ipw2100_open
zd1201_net_open

network controllers

Ethernet WiFi

human
interface

HI char devices

cdev
input_fops
console_fops
fb_fops
video_fops
snd_fops
sys_syslog
kmsg

security

security/
linux/security.h
may_open
security_socket_create
security_inode_create
security_ops
selinux_ops

debugging

handle_sysrq
printk
opt_kgdb_wait
kgdb_breakpoint
log_buf

HI subsystems

mousedev handler
tty
oss
alsa

**abstract devices
and
HID class drivers**

console
kbd
mousedev
uvc_driver
video_device
fb_ops

**HI peripherals
device drivers**

atkbd_drv
psmouse
i8042_driver
vga_con
ac97_driver
drivers/video/

user peripherals

keybdrv mouse video audio graphics card



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

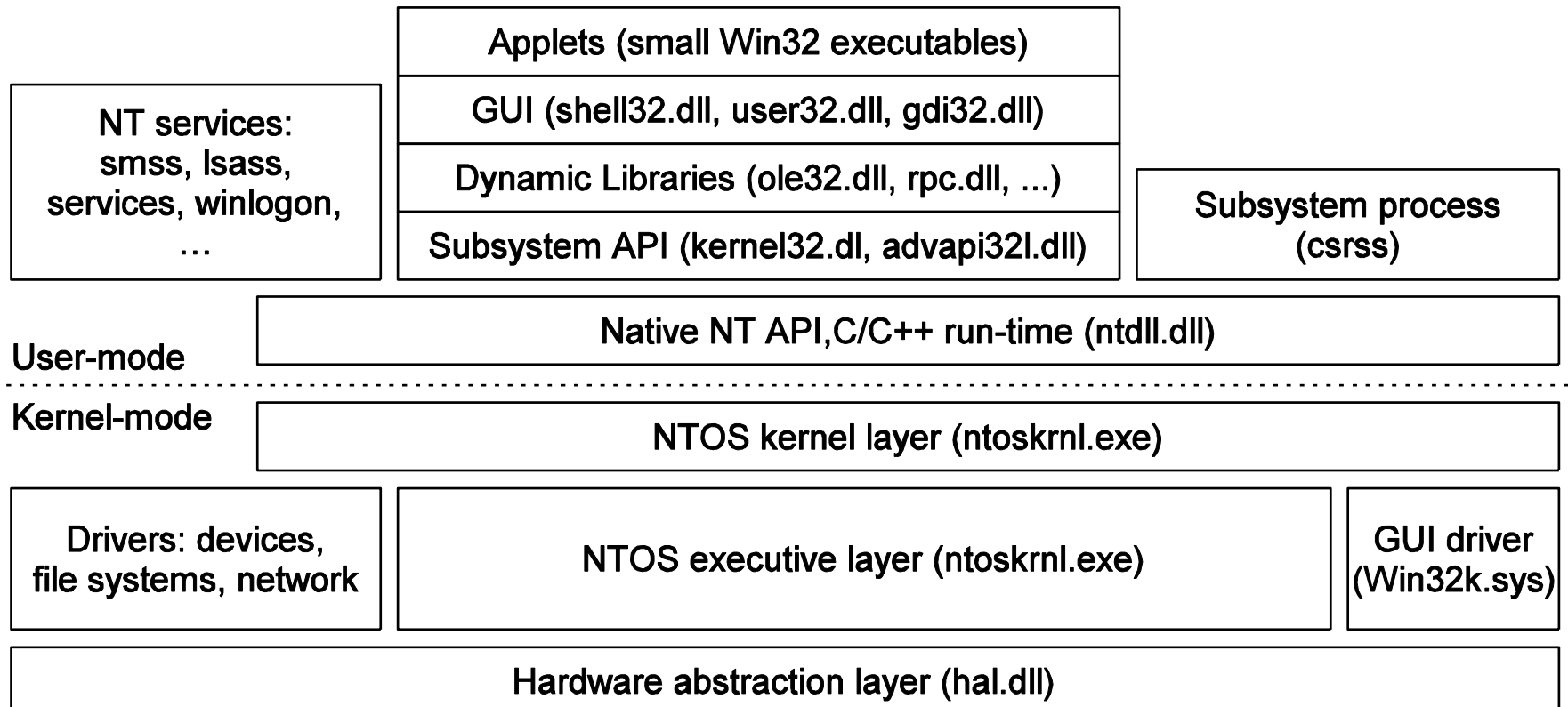
- Better support for touch screen devices, integration with cloud services

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

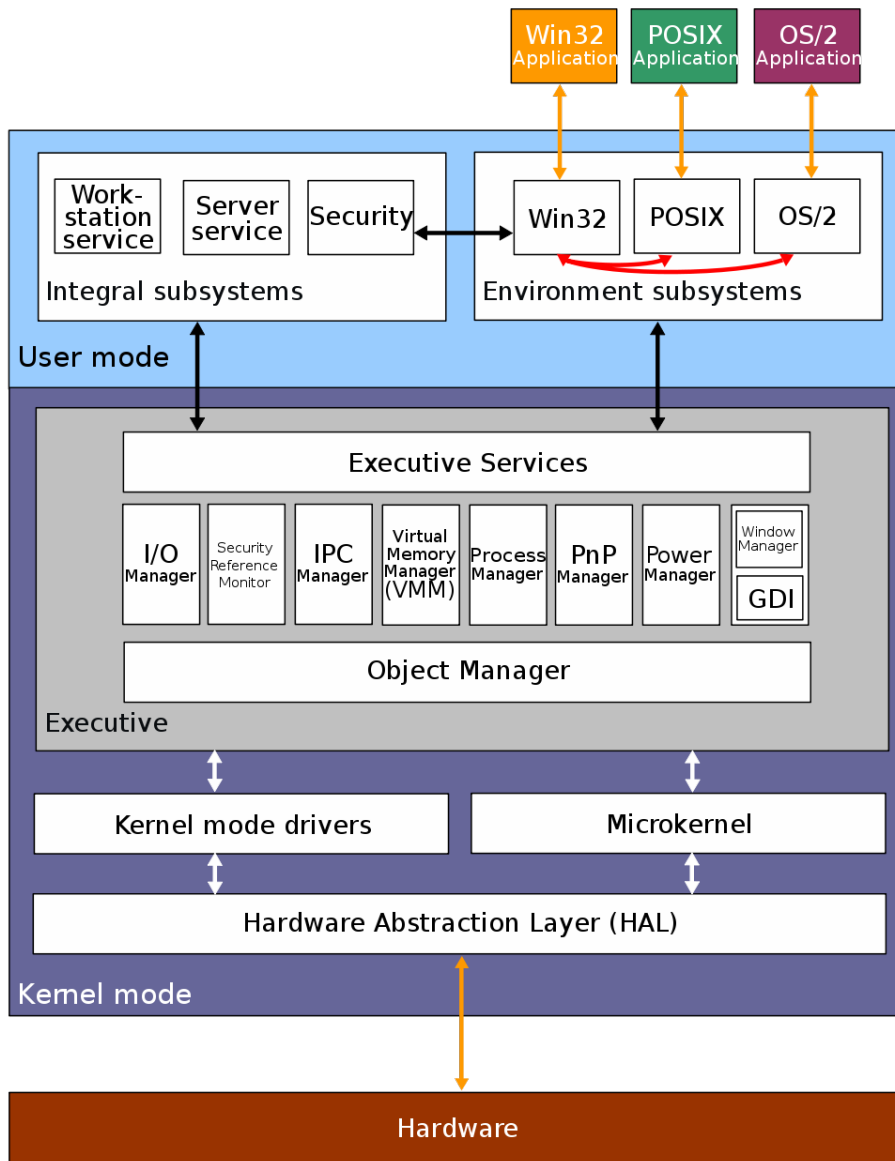
Hardware Abstraction Layer (HAL)

- 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



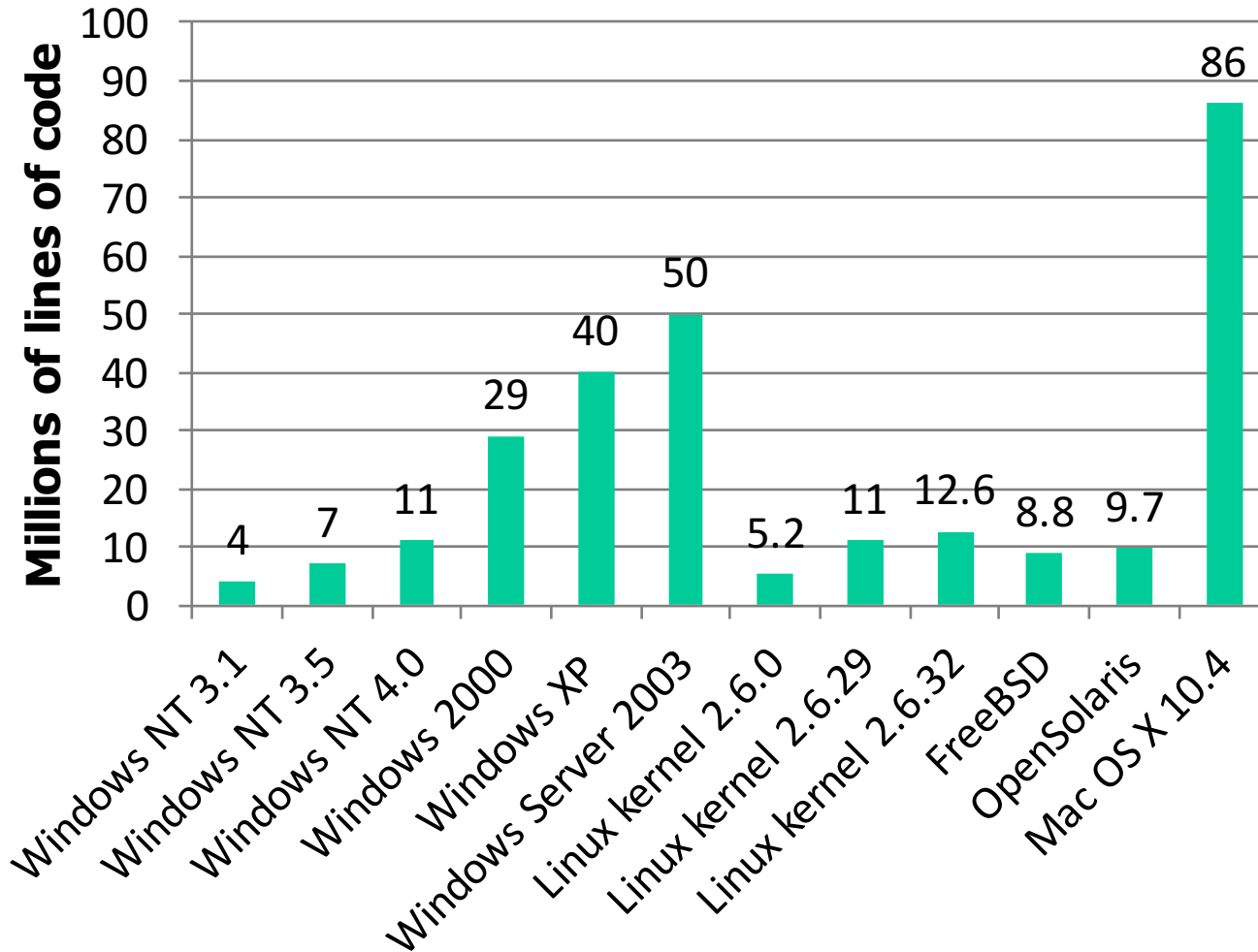
Structured like microkernel

- 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



source: Wikipedia 2010

Code bloat

- Is lines of code useful comparison for complexity?