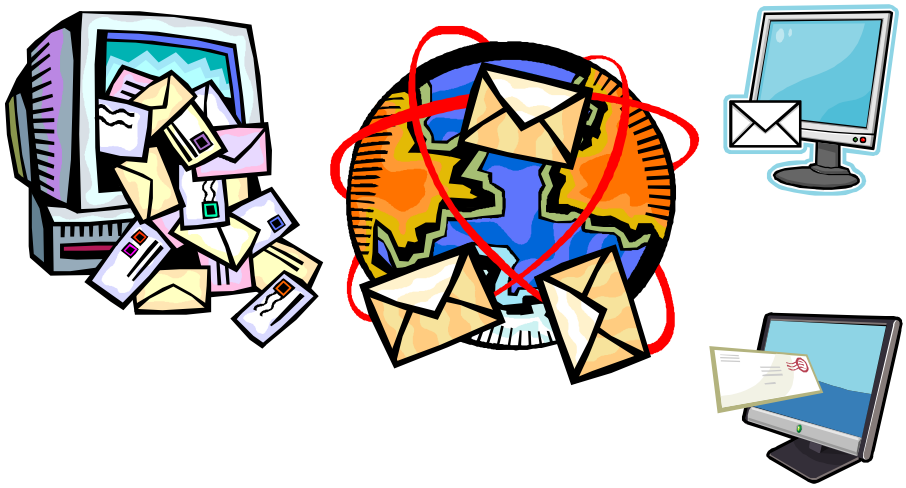
Operating Systems

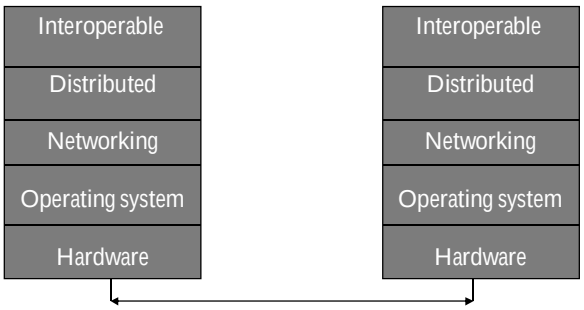
Fall 2008

Tiina Niklander

Distributed Systems and Networking specialization area



Distributed Systems and Networking



Master level: Operating systems, distributed systems, networking,

Bachelor level: basic communication & protocols, concurrency concepts, security, computer organisation

Material

- Course book: A.S. Tanenbaum, *Modern Operating Systems, 3rd. ed.*, Prentice-Hall, 2007
  - Any other large OS book, f.e. Stallings, Silbershatz, or Deitel, should be feasible alternative
- Lectures & exercises on the course web page *http://www.cs.helsinki.fi/u/niklande/opetus/kj/*

Course structure

- 1: Overview
- 2: Processes and Threads
- 3: Virtual Memory and Paging
- 4: Page Replacement
- 5: Segmentation
- 6: File Systems, part 1
- 7: File Systems, part 2
- 8: I/O Management
- 9: Multiple Processor Systems
- 10: Example: Linux, Windows, Symbian
- 11: Design Issues
- 12: Recapitulation, hints for exam

Tanenbaum: Sections 1,2,3,4 ,5,8,10,11,12,13,14

Schedule

- Lectures (in Finnish) twice a week
  - Read the book carefully,
  - Focus on mastering the points given in slides
- Exercises once a week
  - Problems available one week before
  - Must be solved in advance, that is BEFORE the meeting (not during it!)
- One weekly problem to be submitted on paper
- EXAM: Wed 15.10. 16.00

Operating System Overview

Introduction

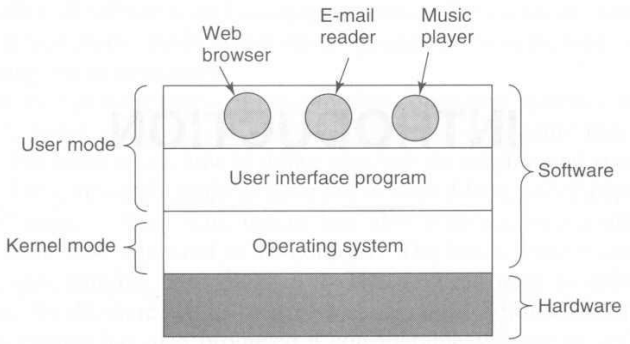
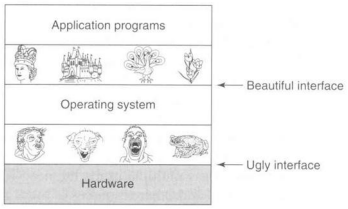


Figure 1-1. Where the operating system fits in.

- A computer system consists of
  - hardware
  - system programs
  - application programs

8

Main Tasks of Operating System



- Provide abstractions
  - Abstractions (like file, address space) to user programs
  - Implement and manage the abstract objects
  - Hide the ugly and messy details in these
- Manage resources
  - Multiplexing in time and in space!
  - Each program gets time with the resource
  - Each program gets space on the resource
  - Optimize the hardware usage to improve performance

9

Concepts (Abstractions)

- Process
  - Is a container that holds all information needed to run a program
  - Process table entry contains all process related info
- Address Spaces
  - Memory locations 0..MAX available for one process
  - Protected from each other
- File
  - Directories to contain files: working directory, path
  - File descriptor
  - File system

OS structure

User

Applications Shell System programs

System calls

Process management Resource management

protection

File system Memory management

blocks

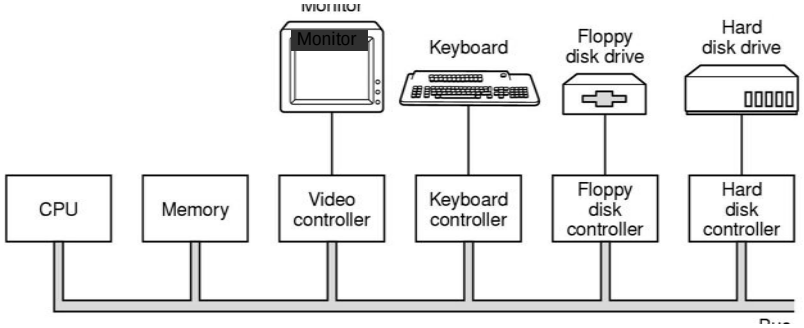
I/O management

I/O module (Device controller) Interrupt handling

Devices and device drivers

Hardware

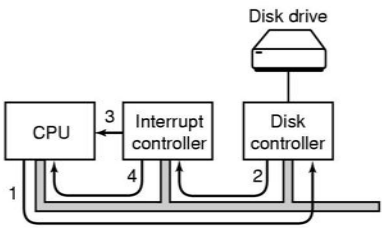
Computer Hardware Review

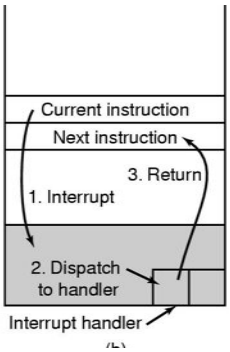


- Set of instructions
- Registers
  - Program counter
  - Stack pointer
  - Process Status Word (PSW)
- Kernel mode vs User mode
- System call (service call, trap)

13

Interrupts



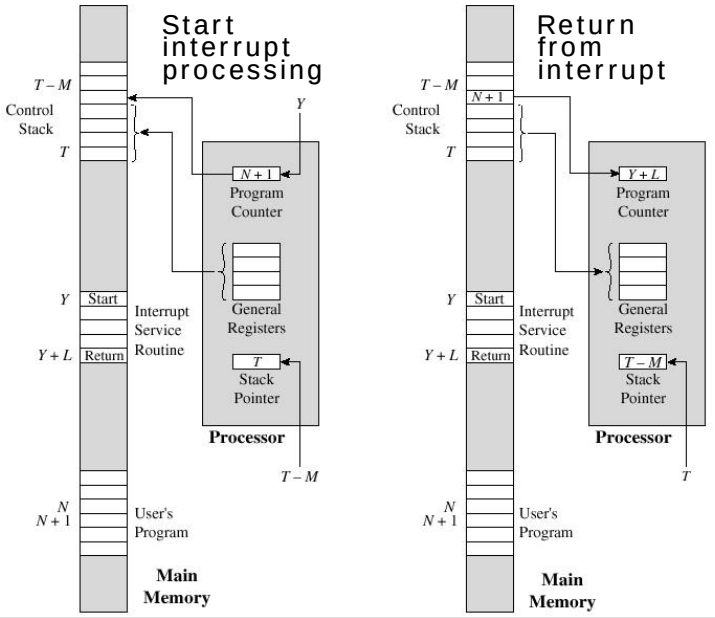


- I/O device controller provides a clean interface to the device
- Device driver is the software that communicates with controller
- Interrupt is the way for controller to inform driver about status change

14

Interrupt in details

Sta Fig 1.11



Memory hierarchy

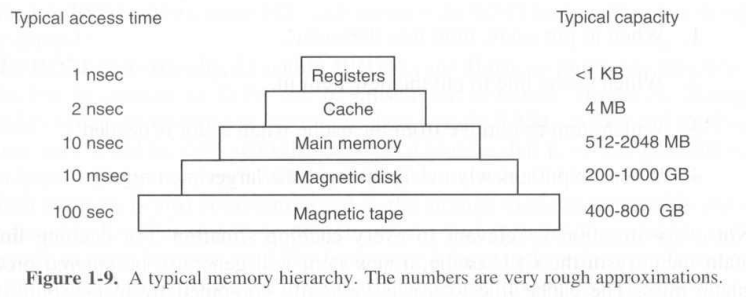


Figure 1-9. A typical memory hierarchy. The numbers are very rough approximations.

Pentium 4 cache:

8 KB data, 12 KB code/text, external 256 KB

nano = 10^{-9}, micro = 10^{-6}, milli = 10^{-3}

Locality of reference

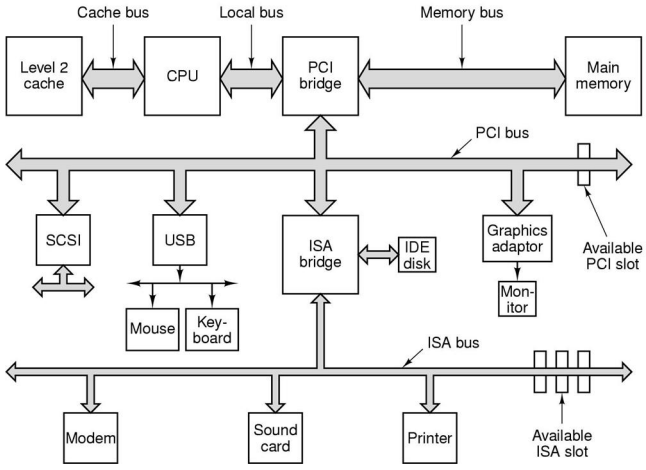
Spatial and temporal locality:

- For example in a loop a small set of instructions is executed several times
- Certain part of code only uses a small set of variables (data)

➔ When program makes a memory reference (data or instructions), it is likely to refer again to the same location or a location near by

- This is the principle behind the usage of caches

Computer Hardware Review



Structure of a large Pentium system: multiple buses

18

System calls

Systems calls

- Interface between user programs and OS dealing with abstractions
- Special kind of procedure call: switch between user and kernel mode
- Trap into kernel and invoke OS

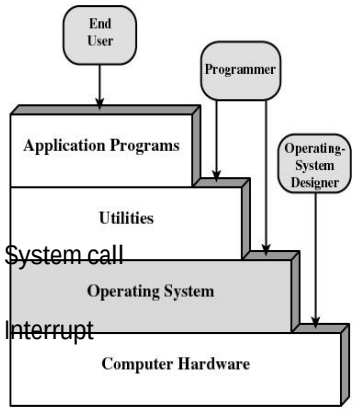
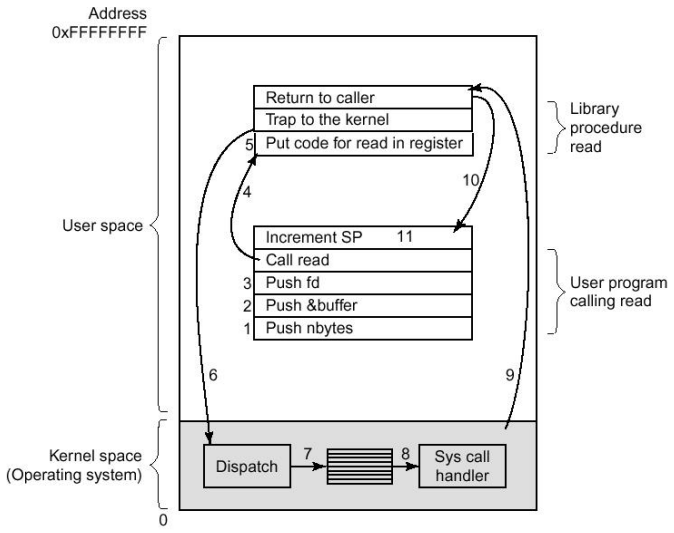


Figure 2.1 Layers and Views of a Computer System

read(fd, buffer, nbytes) Tan08 1-17



POSIX: Some System Calls For Process and File Management

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

22

POSIX: Some System Calls For Directory Management and Misc

Directory and file system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

23

A stripped down shell **Tan08 F.1-19**

```
#define TRUE 1

while (TRUE) {                                  /* repeat forever */
     type_prompt( );                            /* display prompt on screen */
     read_command(command, parameters);         /* read input from terminal */

     if (fork( ) != 0) {                        /* fork off child process */
          /* Parent code. */
          waitpid(-1, &status, 0);              /* wait for child to exit */
     } else {
          /* Child code. */
          execve(command, parameters, 0);       /* execute command */
     }
}
```

"Riisuttu" komentotulkki

Some Win32 API Calls

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

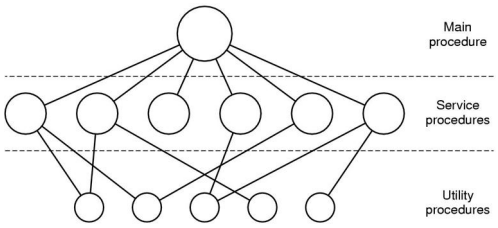
25

Operating System Structures

Some examples

- Monolithic System
- Layered System
- Microkernel
- Client-Server Model
- Virtual Machines
- Exokernels

Monolithic Systems



- Whole OS as one program in kernel mode
  - Collection of procedures, linked together
- Basic structure of a monolithic system
  - Main program invokes requested service procedure
  - Service procedures carry out the system calls
  - Utility procedures help service procedures
- There might be loadable extensions

Layered systems

Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

Structure of the THE operating system

- Generalization of the procedure approach
- Each layer has specific tasks
- Example: THE system
  - The layers mainly a design aid

Micro-kernel

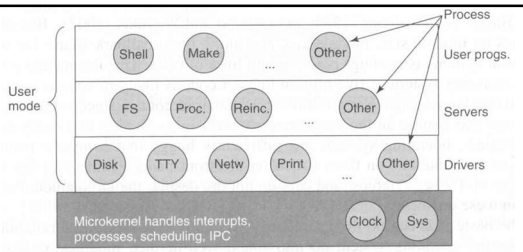
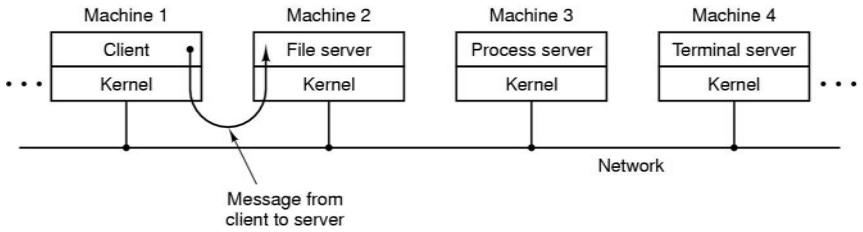


Figure 1-26. Structure of the MINIX 3 system.

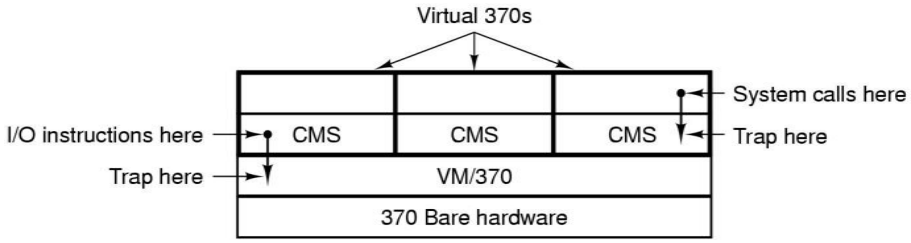
- Split the OS to small, well-defined modules
- Only the microkernel module runs in kernel mode
- Other modules run in user mode
- Goal: increase availability
  - A buggy driver cannot crash the whole computer
  - MINIX has reincarnation server to automatically replace failed module
- Disadvantage:
  - A lot of mode switches within OS itself

Client-Server Model



- Client-server model is an abstraction that can be used on single computer or networked computers
- Clients send message saying what it wants over network or e.g. using microkernel

Virtual machines



Structure of VM/370 with CMS

- Virtual machine abstraction was designed 40 years ago
- Basic idea:
  - Virtual machine monitor does multiplexing of the virtual machines
  - Each virtual machine is 'exact' copy of the bare hardware and has its own OS and applications

Virtualization today

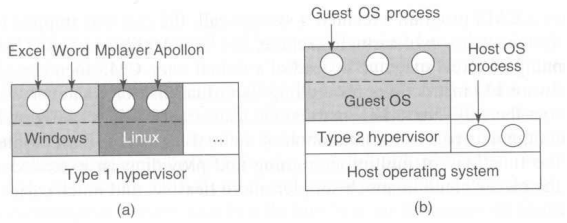


Figure 1-29. (a) A type 1 hypervisor. (b) A type 2 hypervisor.

(a) Type 1 hypervisor
on top of hardware
multiplex
all OS in parallel

(b) Type 2 hypervisor
on top of host OS
multiplex
only guest OSes

- Virtual machine monitor renamed as hypervisor
- Java virtual machine uses the virtualization abstraction for code portability

Exokernels

- Partitioning of the machine
- Each virtual machine gets only a subset of the exiting resources (not the image of the whole machine)

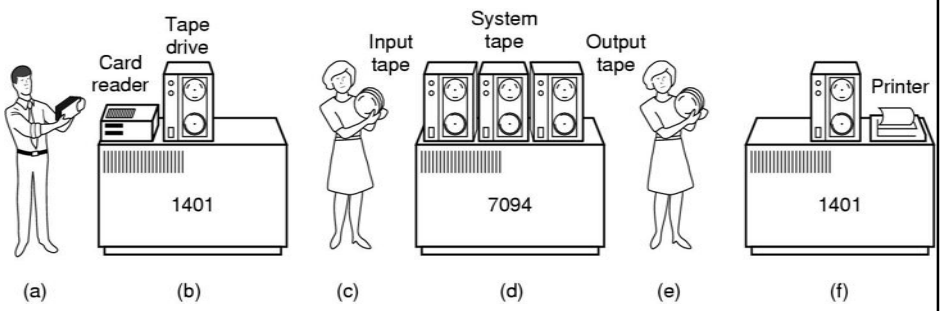
C and Metric Units

- Must be able to use:
  - Reading C-code examples
  - Unit conversions

Exp.	Explicit	Prefix	Exp.	Explicit	Prefix
10^{-3}	0.001	milli	10^{3}	1,000	Kilo
10^{-6}	0.000001	micro	10^{6}	1,000,000	Mega
10^{-9}	0.000000001	nano	10^{9}	1,000,000,000	Giga
10^{-12}	0.000000000001	pico	10^{12}	1,000,000,000,000	Tera
10^{-15}	0.000000000000001	femto	10^{15}	1,000,000,000,000,000	Peta
10^{-18}	0.000000000000000001	atto	10^{18}	1,000,000,000,000,000,000	Exa
10^{-21}	0.000000000000000000001	zepto	10^{21}	1,000,000,000,000,000,000,000	Zetta
10^{-24}	0.000000000000000000000001	yocto	10^{24}	1,000,000,000,000,000,000,000,000	Yotta

History of Operating Systems

History of Operating Systems (1)



Early batch system
- bring cards to 1401
- read cards to tape
- put tape on 7094 which does computing
- put tape on 1401 which prints output

37

History of Operating Systems (2)

- First generation 1945 - 1955
  - vacuum tubes, plug boards
- Second generation 1955 - 1965
  - transistors, batch systems
- Third generation 1965 – 1980
  - ICs and multiprogramming
- Fourth generation 1980 – present
  - personal computers

38