# Chapter 1
# Introduction
## General Aspects Regarding Operating Systems

Print Version of Lectures Notes of *Operating Systems*

Technical University of Cluj-Napoca (UTCN)
Computer Science Department

Adrian Coleșa

February 24th, 2021

1.1

## Part I
## Course Presentation and Administrivia

1.2

## 1 Presentation

### Useful Information

1. Lecture classes (2 hours / week): Adrian Coleșa
2. Lab classes (2 hours / week / semigroup)
   - Balint Szabo for group 30421/1
   - Andrei Seicean for group 30421/2
   - Lazslo Ciople for group 30422/1
   - Adrian Coleșa for group 30422/2
   - David Acs for group 30423/1
   - Istvan Csaszar for group 30423/2
   - Andrei Crișan for group 30424/1
   - Adrian Buda for group 30424/2
3. OS Course's Web page: `https://moodle.cs.utcluj.ro/course/view.php?id=354`
   - create user account, if not having one yet
     - pay attention at what **firstname** and **surname** (familyname) are
   - register for the "*Operating Systems, Spring 2021*" course
   - enrollment keys
     - for semi-group 3042N-M: `os-2021-GR-N-M`, $N \in 1,4$, $M \in 1,2$
     - for students that re-attend the course: `os-2021-Retaken`

1.3

### Purpose and Objectives

- general purpose
  - make students understand the fundamental concepts and functionality of modern OSes
- specific objectives

1. understand the OS's role and its components' functionality
2. be familiar with OS' major services (system calls)
3. have general knowledge about the internal mechanisms of an OS

- methods
  - presents the most important components of an OS from two points of view: **external** (interface) and internal (design)
  - problem solving
  - practice (write C programs) on a real OS: ***Linux***

1.4

### Contents

1. Introduction. General Aspects Related to Operating Systems
2. The Command Interpreter
3. The File System (2 parts)
4. Process Management
5. Thread Management
6. Synchronization (2 parts)
7. Memory Management (3 parts)
8. Inter-Processes Communication (IPC) Mechanisms
9. Protection and Security

1.5

### Docs sources

**Books**

- **A. Tanenbaum, *Modern Operating Systems*, 4nd Edition, Pearson, 2014**
- A. Silberschatz, P. Galvin, G. Gagne, *Operating Systems Concepts*, 9th Edition, Wiley, 2012
- Michael Kerrisk, *The Linux Programming Interface*, No Starch Press, 2010
- M. Mitchell, J. Oldham, A. Samuel, *Advanced Linux Programming*, New Riders Publishing, 2001

**On-line**

- Remzi H. Arpaci-Dusseau, Andrea C. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*, online available at `http://pages.cs.wisc.edu/~remzi/OSTEP/`

1.6

### Acquired Students' Competences and Skills

1. be able to explain the role of the OS in a computing system
2. be able to define and explain fundamental OS concepts
3. be able to use basic Linux commands
4. be able to write C programs to use Linux system calls

1.7

## 2 Requirements and Policies

### Prerequisites

1. basic knowledge about computing systems
2. C programming

1.8

## Students' and Teacher's Responsibilities

1. interest for the subject
    - **trust me**: it is fundamental and really interesting
2. receptiveness to the other side challenges
    - **ask** (*use the forum, chat!*), **answer**, **propose**
3. **perseverance** during the entire semester
4. **balance the effort**, be in time to scheduled deadlines
    - use the night before the deadline / exam for sleeping!

1.9

## Attendance and Recovery Policy

1. **this is not about planning "*what and how much to miss (skip)*"**
    - our activities are meetings, based on communication, not possible without the presence of both parties
    - all that I and my colleagues prepare(d) is **for you**
    - a meeting and a discussion is a living action: you are given not just information, but also personal experience
    - reading at home is not communication and could by far be not so effective!
    - though, reading and learning by him/herself is required and important, obviously
2. attendance policy's terms and rules are really very, very, very strict
    - exceptional cases must be discussed in time, not just at the end of semester

1.10

## Attendance and Recovery Policy (cont.)

1. attending lecture classes
    - minimum 7 to be allowed to take examinations
    - *below 7 $\Rightarrow$ must retake the course next year!*
    - **there is no possibility to recover missed classes**
2. attending lab classes
    - maximum 2 missing labs to be allowed to take the (summer) lab examination
    - maximum 4 missing labs to be allowed to take the lab re-examinations
    - *more than 4 $\Rightarrow$ must retake the course next year!*

1.11

## Attendance and Recovery Policy (cont.)

1. recovery policy for labs
    - **absences are not removed!**
    - missed classes should be recovered in lab rooms (extra time with other groups)
        - send solutions to proposed problems to your lab teacher
    - maximum 2 per exam session, i.e.
        - 2 missed labs could be recovered until one week before the end of semester
        - 2 extra missed labs could be recovered in any re-examination session this year

1.12

## Assignment Lateness and Recovery Policy

- 3 lab assignments
- two deadlines for each assignment
    - 1st one is soft and optional (on Tuesday at 1 am)
    - about one week between them
    - assignment could be resubmitted on the 2nd deadline, if not please with received grade
    - 2nd one is hard (on Sunday at 23:55)
- assignment recovery
    - only in re-examination sessions

1.13

## Plagiarism and Cheating Policy

- **cheating is not allowed and not accepted!**
- if **anyone** found guilty of something like this will not be allowed to take the exam in any exam session!
- we use an application that **checks against plagiarism** your submitted solutions
  - between them
  - against submissions from all previous years
- this is really very, very, . . . strict
- if you need help, ask it in time from us (your teachers)
- **never show, give or make public your code**
  - excepting to your teachers
- read more details in the “Examination and Plagiarism Policy” on the course page

1.14

## What I Like

- work with enthusiastic people (students)
- see interested, friendly faces
- have a feedback from those I work with (teach)
- be asked good questions
- learn about interesting (useful) thinks
- . . .

1.15

## What I Do Not Like

- teach someone things he/she thinks from the beginning (having a prejudice) is of no use or interest
- teach someone things he/she thinks from the beginning he/she knows better and everything about the subject
- on-site classes
  - talk to someone who is not looking at me
    - but in his/her laptop, phone etc.
  - be disturbed during my talk (presentation)
    - have question? ask them!
    - something not clear? ask about!
    - interesting idea related to OS? just say it loudly!
    - anything else? delay it for later!
- on-line classes
  - talk to someone not paying attention
  - ???
- . . . not so many though!

1.16

# 3 Evaluation and Grading

## Lab Evaluation

- quiz tests each lab class (about 10 minutes)
  - $\boxed{Lab\_QT_i \in [1,10], i=\overline{2,13}}$
- bonuses: $\boxed{Lab\_B_i \in [8,10], i=\overline{1,13})}$ for
  - lab activity at each class
  - extra (special) problems

- $\boxed{Lab\_B = \sum_{i=0}^{13} \frac{max(0;6+(Lab\_B_i-8)*2)}{10}}$

- 3 lab assignments: $\boxed{Lab\_A_i \in [0,10], i = \overline{1,3}}$
  - one each 3-4 weeks (see the moodle page for deadlines)
  - consists in one problem needing about working 4-5 hours for being solved
- final test (last week): $\boxed{Lab\_P \in [0,10]}$
  - 2-3 problems to be solved on the computer

1.17

## Lab Evaluation (cont.)

- Lab grade formula $\boxed{Lab = 0.10 * Avg(Lab\_QT_i) + 0.60 * Avg(Lab\_A_i) + 0.30 * (Lab\_P + Lab\_B)}$
- Conditions to pass
  - $Avg(Lab\_QT) >= 5$
  - $Avg(Lab\_A) >= 5$
  - $Lab\_P >= 5$
  - $Lab >= 5$

1.18

## Lecture Evaluation

- short quiz tests AFTER each class (sort of homework)
  - $\boxed{Lect\_T_i \in [0,10], i = \overline{1,14}}$
  - available ONLY to students that attended the corresponding lecture class (based on a password)
  - during one week after the class
- final written examination in the summer session
  - $\boxed{Lect\_E \in [0,10]}$
  - closed-book exam
  - check *understanding* of fundamental concepts
  - three quiz tests
    - basic-level (63%): must take minimum 6 to pass and minimum 7 to have access to next
    - medium-level (20%): must take minimum 7 to have access to next
    - high-level (17%): essay subjects
- Lecture grade formula
  - $\boxed{Lecture = \frac{1}{6}Avg(Lect\_T_i) + \frac{5}{6}Lect\_E}$
- Conditions to pass
  - $Lect\_E >= 5$
  - $Lecture >= 5$

1.19

## Final Grade

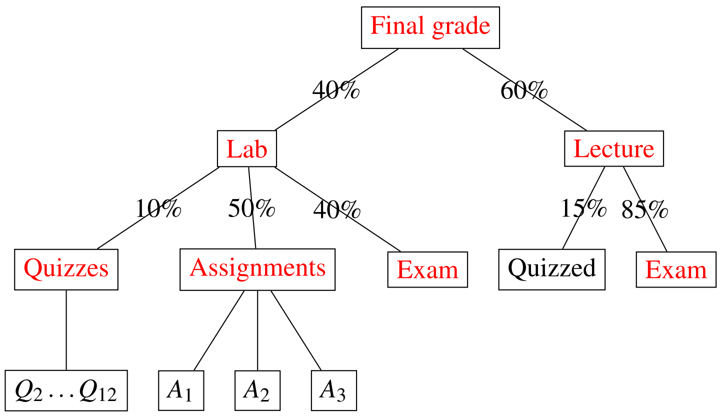
**Final grade formula**
**Final_Grade** $= 0,4 * Lab + 0.6 * Lecture$

1.20

## Evaluation and Grading Illustration

The red items should be $\geq 5$ in order to pass.

1.21



### Optional: OS Club

- optional meetings for OS-related subject presentations
  - presentations and hands-on exercises
- OS Club channel on the OS course on Teams
- open to anyone
- every two weeks
- on Monday, from 18:00-20:00
- first meeting on Monday, March 1st 2021
- proposed subjects
  - Python scripting
  - Linux Shell (Bash) scripting
  - debugging with GDB
  - Windows API for OS services
  - remote communication using sockets
  - basic C++ resource management
  - OS-related vulnerabilities

1.22

# Part II
# OS Definition. Hardware Aspects Review. OS Structure

1.23

## Purpose and Contents

### The purpose of this part

- Define the OS and some of its basic concepts
- Review few fundamental aspects of computer hardware
- Discuss about two possible OS structures: monolith and micro-kernel

1.24

### Bibliography

- A. Tanenbaum, *Modern Operating Systems*, 2nd Edition, 2001, Chapter 1, pg. 1 – 34
- A. Tanenbaum, *Modern Operating Systems*, 2nd Edition, 2001, Chapter 1, pg. 34 – 62

1.25

## Contents

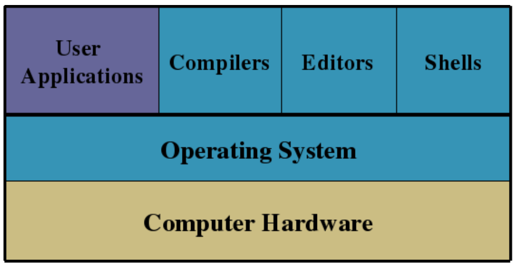


Figure 1: Computing System Structure

1.26

## 4 OS Definition

### What Is An OS?

- A **software**, i.e. a program, an application
  - though, it is a **system** software
  - special from some perspective
- A collection of functions that
  - manage the hardware resources
  - provide the users the environment in which they can
    - run their own applications, in order to
    - use the system resources

1.27

### Where Is Placed The OS?

- *Logical perspective*: **between** user applications and hardware ⇒ it is an **inter-mediator**
- *Physical perspective*: in the system's memory (RAM)
  - a (system) program **among other** (user) programs

1.28

## Roles of An OS

- **Provider** of the *virtual or extended machine* interface
  - Hides the complexity of using the hardware devices
  - Provides **abstractions**: a more convenient view of the system
  - Purpose: **Convenience**
  - *The external perspective*: the user point of view
  - Example: files and directory for HDD, applications for processors
- **Manager** of the *hardware resources*
  - Brings the hardware resources in a functional state
  - Provides each program with time and space for using resources
  - Purpose: **Efficiency**
  - *The internal perspective*: the designer point of view
  - Example: multiplex one processor among more competing applications

1.29

## OS's Interfaces

1. with hardware
   - the kernel and device drivers
2. **with user applications**
   - system calls (special functions)
3. with (human) user
   - shell, usually a user application

1.30

## Do We Need An OS?

- **theoretically NO**
  - any application could be placed directly on the hardware
  - managing it in the way it wants
- yet, in practice **we NEED an OS as a helper**
  - we need help, as hardware interaction is difficult
  - we usually need someone specialized
  - we need to be efficient in terms of application development
  - we need to concentrate on our application logic
  - we need portability for our applications, as hardware is variable
- also **we NEED an OS as a mediator**
  - mediate between multiple applications running on the same computer
  - competing for the (same) limited resources
- ⇒ OS is a **common body of software** for all applications
- ⇒ OS is the **central mediation authority** for all applications

1.31

### Desired OS Characteristics

- be **helpful** and **general**
  - provide a rich functionality
  - provide anything needed by any application
- provide **protection**!
  - protect the HW from user applications
  - protect itself from user applications
  - protect applications from one another
- be **invisible**
  - efficient: “no” performance overhead
  - light: “no” resource usage
  - flexible: “no” restrictions in terms of interface and abstractions
  - general: supporting perfectly any type of application

1.32

**Though ... there is no “one-size-fits-all” OS!**

1.33

### Different Types of OSes

- Mainframe operating systems: *OS/390*, *z/OS*
- Server operating systems: *UNIX*, *Windows Server*, *Linux*
- PC (workstation) operating systems: *Windows 10*, *Mac OS*, *Linux*
- Real-time operating systems: VxWorks, QNX
- Mobile and embedded operating systems: *Android*, *PalmOS*, *Windows CE*, *Windows Mobile*, *Symbian*
- Virtualization operating systems (hypervisors): *VMware ESXi*, *MS Hyper-V*, *Xen*

1.34

### A History of OSes

- the OS evolution was closely related to hardware evolution (computer generations)
- the more complex the hardware, the more complex and more responsible the OS
  - hides the increased complexity of hardware
  - uses it efficiently
- $\Rightarrow$ **an OS is closely tied to the hardware it runs on**

1.35

## 5 Computer Hardware Review

### 5.1 General View

#### Hardware Components

- CPU (mono- vs. multi-processor)
- Memory
- I/O Devices: *monitor*, *keyboard*, *storage devices* etc.
- BUS-es

1.36

#### Hardware Organization

1.37

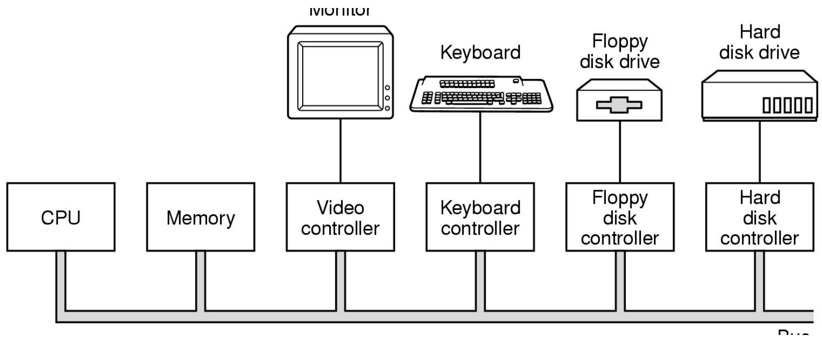


Figure 2: Computing Hardware

## 5.2 Main Hardware Components' Role and Characteristics

### The Processor (CPU)

- Functionality: **executes programs**
  - fetch instructions from memory, decode and execute them
  - execute one instruction flow (i.e. program) at one moment
- Instruction set architecture (ISA)
  - has a specific set of instructions that can be executed
  - ⇒ specific executable programs each processor supports and runs
- Registers
  - Program counter
  - Stack pointer
  - Many others – architecture dependent
  - Compose the *machine state* that is saved at *context switch*

1.38

### CPU's "Intelligence"

- from a technical point of view, the CPU
  - is very complex
  - is very fast
  - does a good job executing programs
- from a logical (semantical) point of view, the CPU
  - is very "simple" and has no "intelligence"
  - provides support (and work), yet
  - does not understand what it executes
  - does not take (logical) decisions
  - does not manage the system
  - does not know what to run and when

1.39

### OS – CPU Relationship (Questions to Answer)

- general question
  - how does the OS control and manage the system?
- specific question
  - how does the OS protect itself and the hardware from user applications?
    - ∗ how does the CPU avoid letting an application execute something that can harm the system?
  - when is the OS executed (by CPU)?
    - ∗ when does the CPU switch between user applications and OS?

1.40

## Different Processor's Modes of Execution

- CPU could execute code in different privilege modes
- OS uses them for protecting itself and the hardware from user applications!
- **Privileged Mode**
  1. access allowed to the complete set of instructions
  2. the OS (kernel) runs in it $\rightarrow$ ***kernel* mode**
- **Non-Privileged Mode (Less-Privileged Modes)**
  1. restrict the possibility to execute some instructions from the instruction set
  2. does not allow direct access to hardware resources
  3. user applications are run in it $\rightarrow$ ***user* mode**

1.41

## OS Usage of CPU's Execution Modes for Protection

1. when the system starts (boots) it runs in privileged mode
2. first software run is the OS, which "takes control of the system"
3. OS configures the CPU such that to trap into OS code at certain events (see below)
   - such configurations could only by done in privileged mode
4. when giving the processor to an application, the OS switches the CPU to user mode
   - there is no way to switch the CPU to privileged mode and still run the user application
5. CPU is automatically switched back to kernel mode when
   - the application generates exceptions, like illegal instruction, division by zero etc.
   - the application calls the OS (system calls), actually generating a sort of exception
   - hardware interrupts occur

1.42

## When does the OS run? Switching From User Applications to OS (Part 1)

- **when the application calls explicitly the OS**
- through a special software mechanism
- called **system call**, which
- is a sort of program exception
- synchronous mechanism
  - triggered voluntarily by applications
  - the application waits for a result
- based on dedicated CPU instructions
  - examples on Intel: *int*, *sysenter / sysexit*, *syscall / sysret*

1.43

## System Call Functionality

- the `read` system call takes about 11 steps

1.44

## Memory

- functionality: **store programs** that are run by CPU
- should be extremely fast, large and cheap
- hierarchy of layers
  - registers: fastest, no delay, but limited size
  - cache
  - **main memory**: **RAM** (Random Access Memory)
  - external storage: HDDs, SSDs, magnetic tapes
    - few orders of magnitude cheaper and larger then RAM, but many orders of magnitude slower

1.45

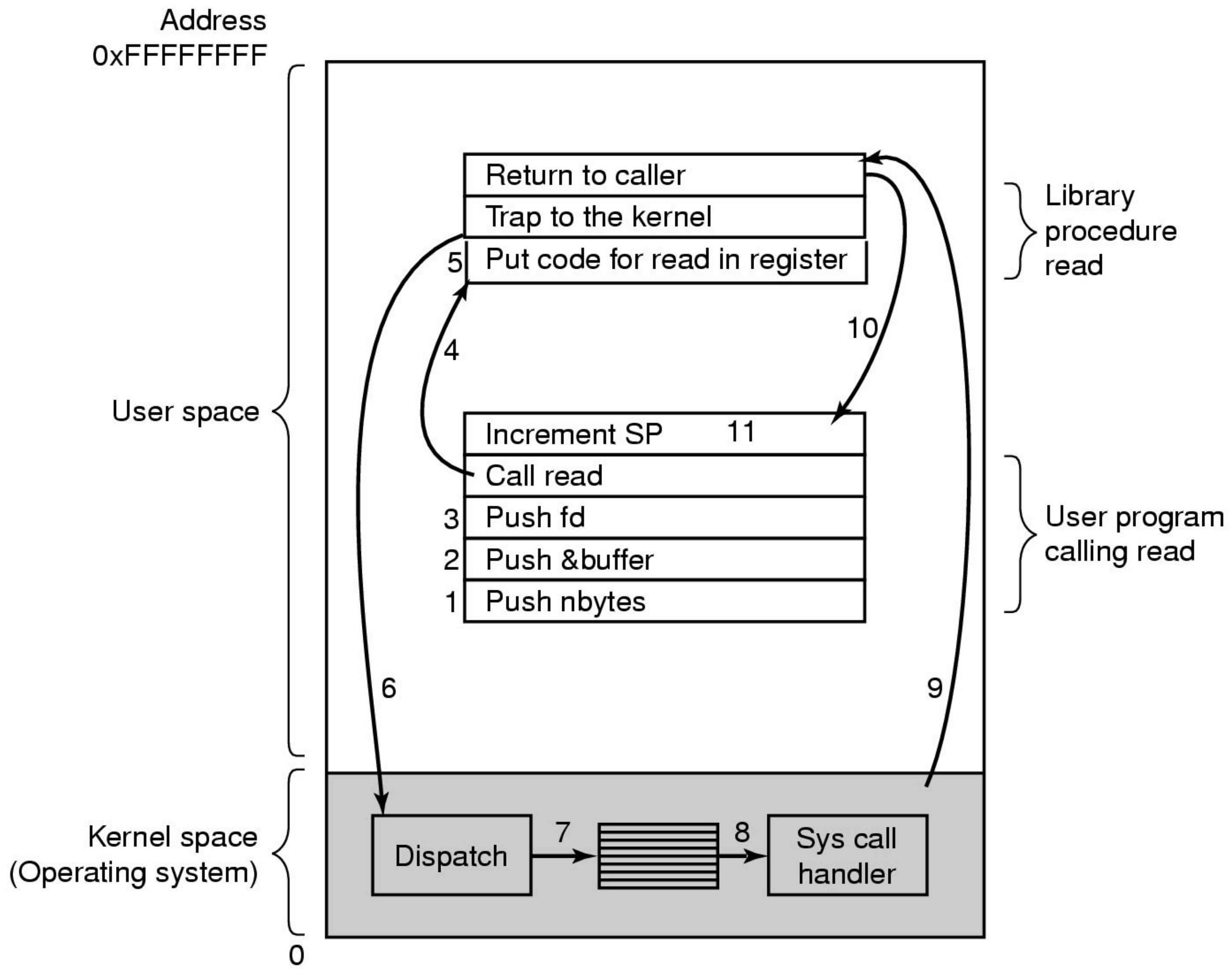


Figure 3: Taken from A. Tanenebaum, Modern OS, p. 46

## I/O Devices

- components
  - controller and the device itself
- controller
  - directly controls the physical device
  - receives commands from the OS
- **device driver**
  - (normally) supplied by controller manufacturer
  - inserted into the OS ⇒ **part of the OS** ⇒ runs in kernel mode
- possible way to work
  - busy waiting
  - **interrupts**

1.46

## When does the OS run? Switching to OS When Interrupts Occur (Part 2)

- **when external interrupts occur**
- it is a hardware mechanism → generated by hardware devices
  - e.g. timer interrupt, disk interrupts
- **asynchronous** mechanism
  - not triggered by applications
  - occurs at unpredictable moments
  - transparent for the applications
- the CPU is configured to switch to predefined memory locations
- where OS code is placed
- CPU automatically switches to privileged mode
- OS handles the interrupt, i.e. the hardware operations
- when finished, returns to the user interrupted application

1.47

## Classes of Interrupts (based on Stallings, *Operating Systems*, p. 34)

**Program Exceptions** Occur as a result of an instruction execution, such as arithmetic overflow, division by zero, illegal instruction, illegal memory address.

**Timer Interrupt** Generated by a timer within the processor. This allows the OS to perform certain functions on a regular basis.

**I/O Interrupt** Generated by an I/O controller, to signal normal completion of an operation or various errors.

**Hardware Failures** Generated by a failure, such as power failure, memory parity error.

1.48

## Do Not Forget!

**OS does not run all the time! It is an event-triggered software.**

1.49

## 6 Conclusions

### Conclusions

- defined the OS
- identified the OS' roles
  - hide hardware complexity (provide a virtual machine)
  - manage hardware
- reviewed main hardware components: CPU, memory (RAM), I/O devices
- CPU's modes of execution: privileged (kernel) vs. non-privileged (user)
- CPU switch between OS and user applications
  - system calls

1.50

### Lessons Learned

- OS is dependent on the hardware it runs on
- OS cooperates with the hardware (use hardware support) to provide its functionality
- there is no “one-size-fits-all” OS

1.51

### Optional Sections

**The following sections contain optional subjects, related to the presented ones! They are optional, though recommended for reading.**

1.52

## 7 Main OS Concepts and Terms (Optional)

### User

- the human being using a computing system
- *multiuser* system
  1. authentication mechanism: identify the user
  2. protection mechanisms: e.g. permission rights
  3. accounting mechanisms
- user groups
- *administrator*

1.53

### Processes (User Applications)

- Definition
  - a program in execution
  - consists of executable code, data, stack, CPU registers value, and other information
- Processes can be created by
  - the OS: special system processes
  - other processes ⇒ process-child relationship and process hierarchy
- *Multiprogramming* and *timesharing* ⇒ *scheduling*
- Process *synchronization*
- Inter-Process *Communication* (IPC)

1.54

### Files and Directories

- *file*
  - user’s basic unit of data allocation
  - a collection of related data
- *directory* (*folder*)
  - used for file organization
  - results in a file hierarchy (tree, graph)
- permanent (physical) data space is viewed as a *file system*

1.55

### Address Space

- the set of addresses accessible to a process
- virtual vs. physical space
- memory management techniques: segmentation, pagination, swapping

1.56

### System Calls

- OS's services (interface)
- involve a special trapping mechanism to "jump" from user space (application's code) to kernel space (OS's code)
- available through normal library functions
- relationship between system call set and API
  - e.g. POSIX is an API, not a system call specification
- Linux examples: `open`, `read`, `fork`, `wait`, `exit`
- Windows examples: `CreateFile`, `ReadFile`, `CreateProcess`, `WaitForSingleObject`

1.57

# 8 Most Common OS Structures (Optional)

## 8.1 Monolithic OS Structure

### Monolithic OS Structure. Description

- all OS parts linked together in a single piece of code
- the OS code is obtained by
  - compiling all the source files and then
  - linking all the object files in a single final executable
- OS is a collection of functions, each one being
  - visible to all the other ones (no information hiding)
  - able to call any other one, if necessary
  - able to access any data, if necessary
- apparently there is "no structure"
  - though, there is a logical one
- entire OS (all its components and code) runs in kernel mode
- examples: most *UNIX* versions, *Linux*, MS Windows

1.58

### Monolithic OS Structure. Advantages vs Disadvantages

- advantages
  - performance $\rightarrow$ very fast
- disadvantages
  - non-modular: could be very big
  - non-modular: could be difficult to extend
  - non-portable: difficult to port on other systems
  - unreliability: a bug in a module can freeze / crash the entire OS
  - unreliability: a vulnerability in a module can compromise the entire OS

1.59

### Monolithic OS Structure. Illustration

1.60

### Monolithic Structure of Linux

1.61

### Monolithic Structure of Windows

1.62

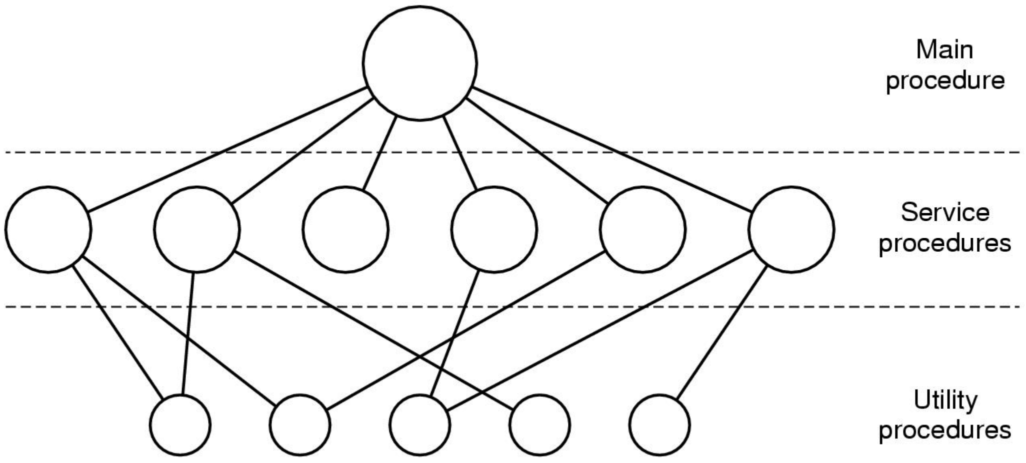


Figure 4: from Tanenbaum, Modern OS, p. 57

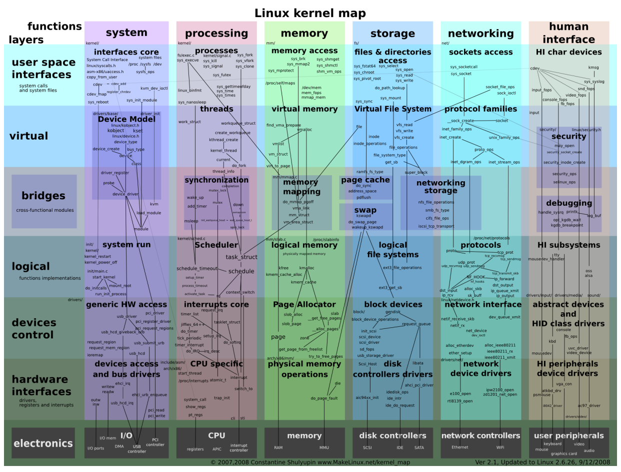


Figure 5: from `http://www.makelinux.net/kernel_map`

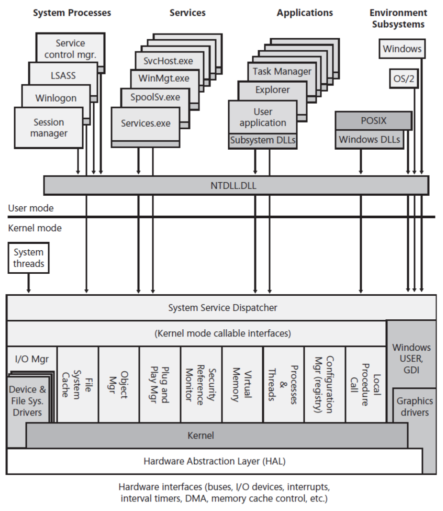


Figure 6: from `MS Windows Internals`

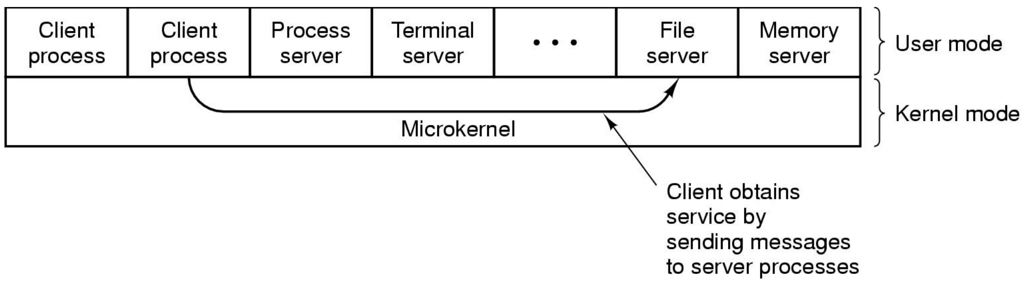


Figure 7: from Tanenbaum, Modern OS, p. 62

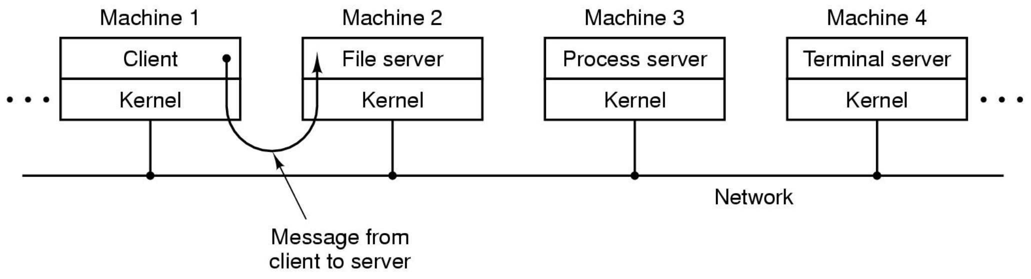


Figure 8: from Tanenbaum, Modern OS, p. 63

## 8.2 Microkernel OS Structure

### Microkernel OS Structure. Description

- based on
  - modularization
  - clear and strict separation of duties and privileges
  - client / server architecture
- a small OS part (the kernel) runs in kernel mode
  - provides basic I/O operation, synchronization and communication between other OS modules
- other OS parts (system servers) run in user space
- examples: *Mach*, *GNU Hurd*, *Minix*, *L3/L4 family*, *seL4*

1.63

### Microkernel OS Structure. Advantages vs. Disadvantages

- advantages
  - modularity
  - reliability, protection
  - portability
  - adaptability to distributed environments
- disadvantages
  - performance penalties due to inter-modules communications

1.64

### Microkernel OS Structure. Illustration

1.65

### Distributed OS Structure

1.66

### Modules in Linux

- modules can be added (loaded) to and removed (unloaded) from kernel at run time
- takes advantages of micro-kernel architecture (modularity, efficient memory utilization etc.)
- Linux is still a monolithic OS

1.67

# 9 OS's Components (Optional)

## Main OS's Components

- Process manager
- Memory manager
- Data manager (File system)
- I/O devices manager
- Communication system
- Protection system
- Shell

1.68

## Process Manager

- manages user applications (processes and threads): creates, suspends, blocks, terminates, schedules processes
- POSIX system calls: `fork`, `execv`, `wait`, `exit`, `getpid` etc.

1.69

## Memory Manager

- manage the system memory, allocate memory to processes, provide virtual memory
- system calls: `mmap`, `munmap`, `shmget`, `shmat`, `shmctl` etc.

1.70

## Data Manager

- storage manager: physical data allocation and retrieval
- data presentation (visualization and access) manager: the file system
- POSIX system calls: `creat`, `open`, `lseek`, `read`, `write`, `close`, `link`, `unlink`, `stat`, `ioctl`, `fcntl` etc.

1.71

## Shell

- it is a special user applications
  - do not belong to SO; runs in user space
  - each OS has its own shell
- provides the user the interface to interact with the OS
  - use the system
  - launch other applications
- two types of shells
  - text interface – *command interpreter*
  - graphical interface

1.72