

# Operating System Structures

Meelis ROOS  
mroos@ut.ee

Institute of Computer Science  
Tartu University

---

fall 2009

## Outline of the course

- Structure of computer and operating system
- Input-output devices
  - Storage systems, network structure, ...
- File systems
  - File system interfaces and implementation
- Memory management
  - Main memory management, virtual memory
- Process management
  - Processes and threads
  - Process scheduling, synchronization, deadlocks
- ...

## Literature

- A. S. Tanenbaum. Modern Operating Systems. 2nd ed. Prentice Hall. 2001.
- G. Nutt. Operating Systems: A Modern Perspective. 3rd ed. Addison-Wesley. 2003.
- A. Silberschatz, P. B. Galvin, G. Gagne. Operating System Concepts. 6th ed. or later, John Wiley & Sons. 2002.
- Course webpage:  
<http://www.cs.ut.ee/~mroos/os/>
- Course mailing list:  
[ati.osehitus@lists.ut.ee](mailto:ati.osehitus@lists.ut.ee)

## Operating System Components

- Process management
- Main memory management
- File management
- I/O system management
- Secondary storage management
- Networking
- Protection and security
- Shell

## Process Management

- Process is a running program
- Process needs resources to run (CPU time, main memory, files, I/O devices)
- Operating system is responsible for the following duties:
  - Process creation and deletion
  - Stopping and resuming of processes
  - Process synchronization mechanisms
  - Inter-process communication mechanisms
  - Deadlock management mechanisms

## File Management

- File is a batch of information about related subjects
- Structure of any specific file is defined by the creator
- Programs (in both source and executable form) and data are mostly stored in files
- Operating system is responsible for the following tasks in file management:
  - File creation and deletion
  - Directory creation and deletion
  - Support for manipulating files and directories
  - Storing files on secondary storage
  - Managing backups of files

## Memory Management

- Memory can be seen as a large array of words or bytes, each having own address; can be accessed quickly by CPU and I/O devices
- Main memory is volatile (lost on crash or power-off)
- Operating system is responsible for the following tasks:
  - Memory allocation and freeing when asked by processes
  - Accounting of memory areas — are they used, by whom?
  - Decisions about what process to bring into memory next when some memory is freed

## Input-Output System

- I/O system consists of:
  - I/O buffering system
  - Cache management system
  - General interfaces for device drivers
  - Specific device drivers

## Secondary Storage Management

- Main memory is lost after power outage, neither do all necessary programs and data fit into memory
- Thus we need to use secondary storage to hold information
- Disks, tapes, ...
- Operating system is responsible for the following tasks in secondary storage management:
  - Space allocation
  - Free space management
  - Disk access scheduling

## Shell

- Many commands to operating system are control commands of the system itself:
  - Process creation
  - I/O operations
  - Main memory management
  - File system manipulation
  - Protection mechanisms
  - Network operations
- Shell — a program that interacts with the user and lets him control the system
- Examples: UNIX shell, MS-DOS COMMAND.COM, MacOS GUI+Finder, Windows GUI+Explorer

## Protection Mechanisms

- Protection mechanisms control access to system and user resources by program, processes or users
- A protection mechanism must:
  - Tell apart authorized and unauthorized users
  - Provide access rules
  - Enforce these rules

## Operating System Services

- Services that operating system provides to programs
- Program execution (loading into memory and starting)
- I/O operations — because programs can not usually access peripheral devices directly, operating system must offer services for it
- File system manipulation — file creation, deletion, reading, writing, directory operations
- Communication — exchange of data between processes (in the same or in different computers), implemented by message passing or shared memory
- Fault detection — discovery of faults with CPU, memory, I/O devices and user programs and appropriate reactions to them

## Additional functions of the OS

- Not user-oriented but for better functioning of the system
- Resource allocation and freeing
- Accounting — who used what resources and how much (for payment and statistics)
- Protection and security

## Most Common Types of System Calls

- Process management
- File management
- Device management
- General information
- Communication

## System Calls

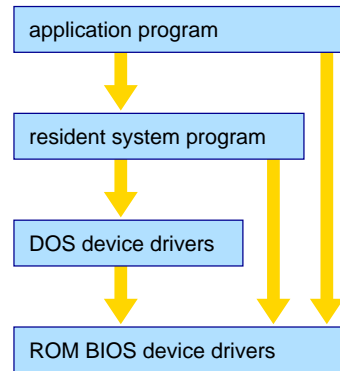
- System calls are the interface between running program and operating system kernel
- Usually implemented in assembly language
- Often also directly available in system programming languages (C, C++)
- Three ways of parameter passing:
  - Parameters in CPU registers
  - Parameters in memory table, fixed register contains address of the table
  - Parameters in the stack
- API — Application Programming Interface
- ABI — Application Binary Interface

## System Programs

- ... help to create an environment for application program execution and development
- ... give access to system calls and services in user-accessible way
- User view of the system is usually defined by system programs
- From the point of view of operating system kernel these programs look like any user program
  - File manipulation
  - Status information retrieval
  - ...
- Some system programs are often implemented as parts of shell

## Structure: MS-DOS

- Designed to get maximal functionality with scarce memory
- Not divided into modules
- Some internal structure exists but interfaces and functionality are not separated
- Applications extend operating system as they find convenient

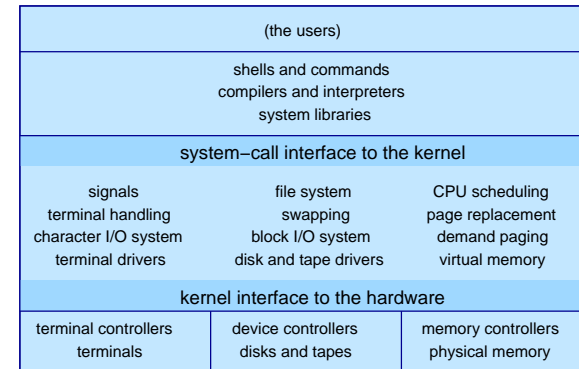


## Layered Approach

- Operating system is divided into layers
- Each layer uses services of lower layers
- The lowest layer is hardware
- The highest layer is application program with user interface
- Modularity: each layer only uses operations and services from lower layers
- Example: OS/2

## Structure: original UNIX

- UNIX was also written for computers with modest resources
- Somewhat structured but still monolithic
- Two separated layers: system programs and kernel
- Kernel is everything below system call interface



## Microkernel

- As much modules as possible are moved out from kernel to user level (as processes)
- The modules communicate using message passing
- Benefits:
  - Easier to extend
  - Easier to port to a new architecture
  - More robust (less code works in kernel mode)
  - More secure
- Exemplified: Tru64 UNIX, MacOS X, QNX, also somewhat Windows NT

## Virtual Machines

- Logical next step from layered system
- Any layers below are seen as hardware that is usable for current layer
- Virtual machine gives client machines an interface that is identical to actual hardware
- Creates an illusion of many processes that use their own processor and run its own operating system there along with its own (virtual) memory
- This client operating system can be very simple (even single-user and single-tasking)
- IBM has been the main maker of virtualized architectures

## Virtual Machines: pros and cons

- Perfect protection of system resources because of virtual machine isolation
- No direct sharing of resources because of this
- Theoretically a powerful means for operating systems research and development
- More efficient use of hardware
- Implementation is complex, especially when architecture is not fully virtualized and needs special tricks to achieve near-100% emulation of hardware; fully virtualizable hardware is a big bonus

## Virtual Machines

- The resources of physical computer are divided to create virtual machines
- The sharing of processor(s) gives the impression that users have their own processors
- File system and spooling give virtual card readers, disks and printers
- Normal time-shared terminal acts as the console of virtual machine
- Shared disks (*minidisk*)
- Virtual network interfaces for communicating between virtual machines
- Paravirtualization as a way to compensate for not fully virtualizable architecture

## Exokernels

- Advancement of microkernel idea
- Device access is not translated (each client system no more has its own zero address)
- Different processes are given different address ranges
- These address ranges are enforced on hardware and kernel level
- Bonus is eliminated address translation overhead