MICROPROCESSOR SYSTEMS (IAS0430)

Department of Computer Systems
Tallinn University of Technology

THE OPERATING SYSTEM (OS)

- **What is the OS?**

THE OPERATING SYSTEM

- **What is the OS?**
  - The OS at its core, is a **program**… a very controlling and powerful program.
    - As we explained in last class, the OS is responsible of managing access to resources (memory, devices, …)
    - It **optimizes the utilization of resources** in the most efficient way.
    - It **protects the resources** from abuse by different programs
    - It also **allows communication between software and hardware.**
    - It **simplifies the complexity of the kernel mode** functions into more comprehensible services.
      - It moderates the access of user mode programs to the kernel mode.
      - Hides kernel and hardware details from the user
      - Creates an additional layer of protection for the kernel mode from users.
      - Prevents tampering of kernel mode by the user.
    - The OS exists based on the assumption that user access to kernel mode is bad.

THE OPERATING SYSTEM

- **What is the OS?**
  - The OS at its core, is a **program**… a very controlling and powerful program.
    - It has a list of services that **emulates the operations of hardware** for the software to access and use.
    - Once the software tries to access these services, the **OS translates these emulated operations** on the software level, to **real operations** on the hardware level. This is what a **syscall** is.
    - The OS manages processes and schedules tasks
      - We will talk about that a bit later

THE OPERATING SYSTEM

- **What is the OS?**
  - The OS at its core, is a **program**… a very controlling and powerful program.
    - It has a list of services that **emulates the operations of hardware** for the software to access and use.
    - Once the software tries to access these services, the **OS translates these emulated operations** on the software level, to **real operations** on the hardware level. This is what a **syscall** is.
    - The OS manages processes and schedules tasks
      - We will talk about that a bit later

    - **But, how does all this happen?!**

THE OPERATING SYSTEM

- **What is the OS?**
  - The OS at its core, is a **program**… a very controlling and powerful program.
    - It has a list of services that **emulates the operations of hardware** for the software to access and use.
    - Once the software tries to access these services, the **OS translates these emulated operations** on the software level, to **real operations** on the hardware level. This is what a **syscall** is.
    - The OS manages processes and schedules tasks
      - We will talk about that a bit later

    - **But, how does all this happen?!**
    - First, the OS needs to communicate with the user … or, more accurately, the user programs needs to communicate with the OS.
      - This is done using the **API**

THE API

- **What is the API?**
    - Stands for **Application Program Interface.**
        - The API is the bridge between the user mode programs and the OS.
        - It separates the user mode from the OS moderated kernel mode functions.
        - The API is a layer of different functionalities and program commands that allow the user or a user mode program to access some of the hardware operations through the OS.
        - There are two major parts that make up the API:
            - **System calls**
                - Those are operations that user mode programs can request the OS to do in kernel mode.
                - A user mode will need to make a system call to run in kernel mode.
            - **The OS libraries**
                - Those are functions that are natively performed and provided to the user mode programs by the OS.
                - This includes file system access, data encoding, GUI functionalities, etc..

THE API

- **What is the API?**
  - It is basically the kernel shell that user programs can communicate with the do different operations provided by the **OS services** or **provided by the kernel mode**.
    - In Unix based systems, it is called the **POSIX**
    - In windows, it is a mesh of DOS, Win32, and other windows APIs
      - Collectively known as the Windows API or WinAPI.

THE API

- **What is the API?**
    - It is basically the kernel shell that user programs can communicate with the do different operations provided by the **OS services** or **provided by the kernel mode**.
        - In Unix based systems, it is called the **POSIX**
        - In windows, it is a mesh of DOS, Win32, and other windows APIs
            - Collectively known as the Windows API or WinAPI.

    - **So, why do we need an API?**

THE API

- **What is the API?**
  - It is basically the kernel shell that user programs can communicate with the do different operations provided by the **OS services** or **provided by the kernel mode**.
    - In Unix based systems, it is called the **POSIX**
    - In windows, it is a mesh of DOS, Win32, and other windows APIs
      - Collectively known as the Windows API or WinAPI.

  - **So, why do we need an API?**

    - The API allows different programs to be ran on different machines that use the same architecture and the same API
      - Code that is compiled to work on **computer 1** that runs **Win XP**, can be ran on any other computer (with the same architecture) using **Win XP**
      - Almost all Unix based systems can run the same binaries compiled on other Unix based systems! Again, noted that they have the same architecture!
      - Can we run **Win XP** programs on **Win 10**?

THE API

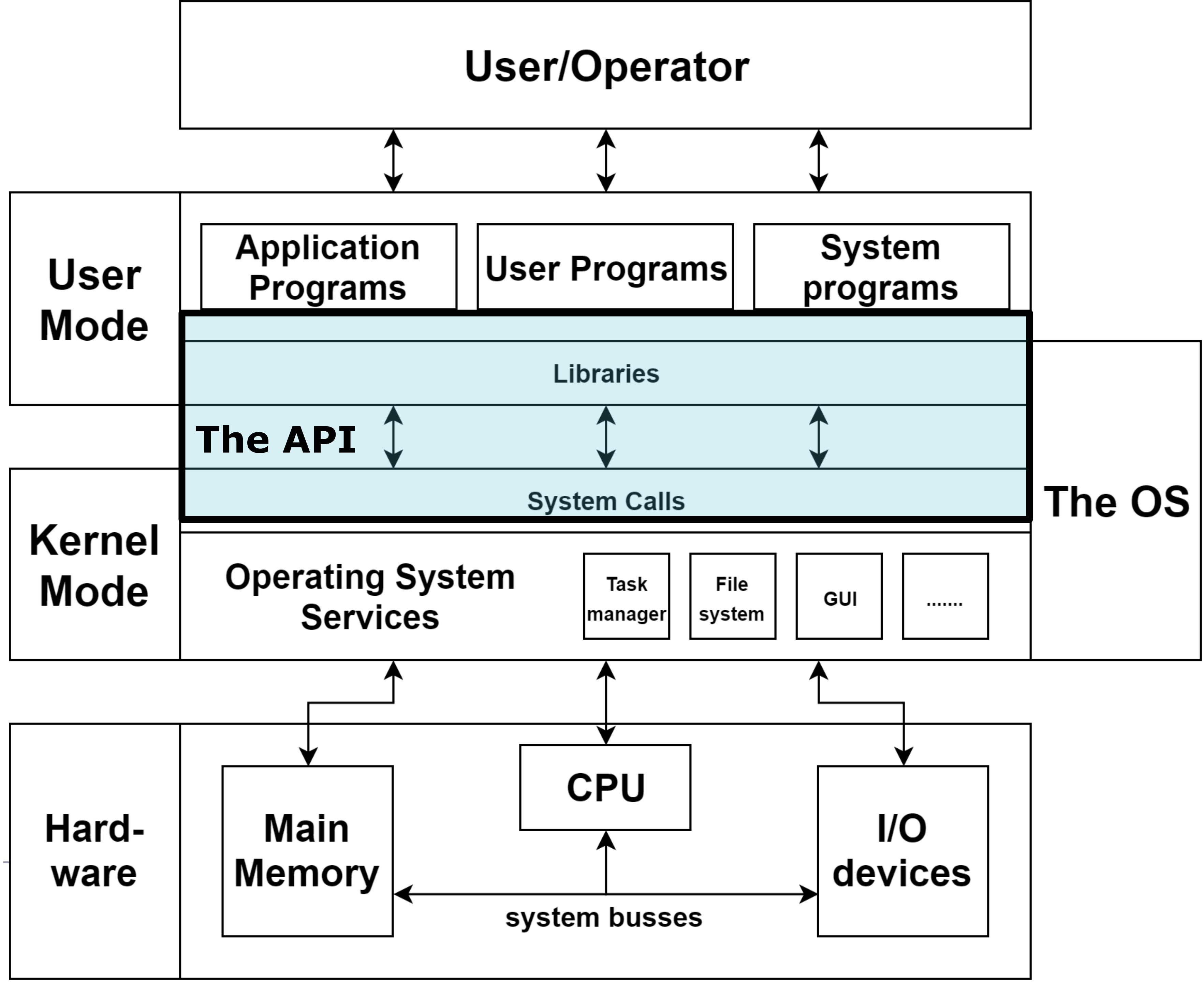
- **What is the API?**
  - **Since the API is the same... YES**
    - Win XP programs can run on Win 10!
    - This can vary depending on the functionalities that the program requires, but in reality it should be possible!

THE API

- **What is the API?**
  - **Since the API is the same... YES**
    - Win XP programs can run on Win 10!
    - This can vary depending on the functionalities that the program requires, but in reality it should be possible!
    - **Why is that possible?!**

THE API

- OSes that have the same API, share similar Libraries and system calls (traps).
- This allows programs to be compiled with the similar libraries and issue similar system calls
- Being so, programs compiled on the same API can easily communicate with OSes that has the same API.
- **The API is like a link.**



- Link to UNIX-LINUX-POSIX
  - https://www.youtube.com/watch?v=hy4OeVCLGZ4
- Link to Windows API
  - https://www.youtube.com/watch?v=S4lQwJawOzI

OS TYPES

- Since the OS stands at the center of communication between software and hardware, there should be different OSes that can handle different types of SW and HW.
- There are 5 main types of Oses:
    - **The Personal Computer Operating System (PC OS)**
        - Are the most common type of OSes.
        - They are designed to handle one or a small number of users at a time.
        - They feature Graphical User Interface (GUI) to ease user experience.
        - Can feature different types of environments based on the user need.
        - Can run software that is compiled using common API.
        - Examples:
            - OS X
            - Windows
            - Linux

OS TYPES

- Since the OS stands at the center of communication between software and hardware, there should be different OSes that can handle different types of SW and HW.
- There are 5 main types of Oses:
  - **Embedded Operating Systems:**
    - Simple operating systems that has little to no software that comes with an embedded OS.
    - It is used mainly to control hardware operations that require no user input.
    - Has little to no software applications.
    - They are designed to operate with little to no human interaction
    - EXAMPLES:
      - Smart TVs – AppleTV, ChromeCast, etc…
      - Smart Appliances – Microwave, fridge, monitor, etc…
      - Complex Machinery – cars, assembly robots, etc …

OS TYPES

- Since the OS stands at the center of communication between software and hardware, there should be different OSes that can handle different types of SW and HW.
- There are 5 main types of Oses:
  - **Real-time Operating Systems:**
    - Those are special types of systems that are designed to respond to input with low latency.
    - Require little to no user interaction.
    - Used in devices that has sensory equipment and requires immediate reaction to sensor input
    - An embedded system can be a real-time OS.
    - EXAMPLES:
      - QNX – A microkernel that is used in devices that work in real time.
      - RTX – a microkernel that converts windows functionalities to support real time operations

OS TYPES

- Since the OS stands at the center of communication between software and hardware, there should be different OSes that can handle different types of SW and HW.
- There are 5 main types of Oses:
  - **Server Operating Systems:**
    - Run on servers and they are designed to provide services (serve) many user at a time at high efficiency.
    - Services include streaming, storage, file management, network operations, etc.
    - They aim for both low user response time and computer utilization.
    - They are general-purpose OSes, meaning they aim to do whatever need to be done with little to no modification possible.
    - EXAMPLES:
      - Windows 2000 Server
      - Linux Server
      - Solaris Server

OS TYPES

- Since the OS stands at the center of communication between software and hardware, there should be different OSes that can handle different types of SW and HW.
- There are 5 main types of Oses:
  - **Mainframe Operating Systems:**
    - Those OSes run on mainframes.
    - They designed to handle immense number of I/O operations as well as provide availability for a large number of users.
    - They are meant to handle even more amounts of users than servers.
    - They run specialized software, although not always.
    - Are used mainly for special use cases.
    - EXAMPLES:
      - **IBM mainframes**

WHAT IS THE OS MADE OF

- The OS is a collection of complex services.
- These services manage resources and protects them from overuse and abuse.
- **Processes:** We will talk about this more in the future.
  - The process is a program being executed.
    - It is in the main memory.
    - It takes in input.
    - Manipulates that input
    - Produces an output.
  - In order for a process to exist, it needs an **address space**.
    - **Address space** is where all the instructions are located in the memory and no other processes can access except the process assigned to that address space.
  - It also needs dedicated CPU registers to perform quick operations (addition, subtraction, etc..)
  - Processes will also invoke system calls to obtain services and operation in kernel mode.

WHAT IS THE OS MADE OF

- **The OS is a collection of complex services.**
- These services manage resources and protects them from overuse and abuse.
- **Memory manager:**
  - Once a process is created, **the operating system designates an address space** for that process.
  - Many processes at time can **cause the memory to be overflowed.** Memory manager assigns a number (binary) to each possible address and divides it fairly across the processes.
  - The memory manager makes sure **than no process accesses memory locations that does not belong to it.**
  - The memory manager also specifies **how to allocate memory space and when**.
  - It keeps track of **what addresses are available and are being used by different processes**.
  - It also **reclaims address spaces once processes are finished**.
    - This will be discussed in detail in the future.

WHAT IS THE OS MADE OF

- The OS is a collection of complex services.
- These services manage resources and protects them from overuse and abuse.
- **File systems:**
  - Files are all types of data that **do not disappear once a process is done**.
  - **Programs, source codes, data, documents, media files, etc..**
  - OSes must provide means to **create, open, modify, close, erase, write, and read files.**
  - The file system provides **a map of where each files is being stored on permanent storage** such as disks, USBs, and flash memory devices.
  - The file system also provides **permissions to users/processes**. Some files can only be accessed by the **OS or authorized users**. Such permissions are maintained by the file system.
    - Remember the Kernel and User modes

WHAT IS THE OS MADE OF

- File system concept
    - https://www.youtube.com/watch?v=mzUyMy7Ihk0

WHAT IS THE OS MADE OF

- The OS is a collection of complex services.
- These services manage resources and protects them from overuse and abuse.
- **The GUI**
  - The GUI is an essential part of many operating systems.
  - It **allows easier access to system functions** as well as **allowing the user to communicate more efficiently with the OS**.
  - It works as a **mapping tool for the different windows presented to the user** by the different applications.
  - This map **exists as a collection of physical display areas** provided by the **video card** and **computer monitor**.
  - The OS, on its part, **connects each of those widows** with the input and output devices. This is called the **input focus**.

WHAT IS THE OS MADE OF

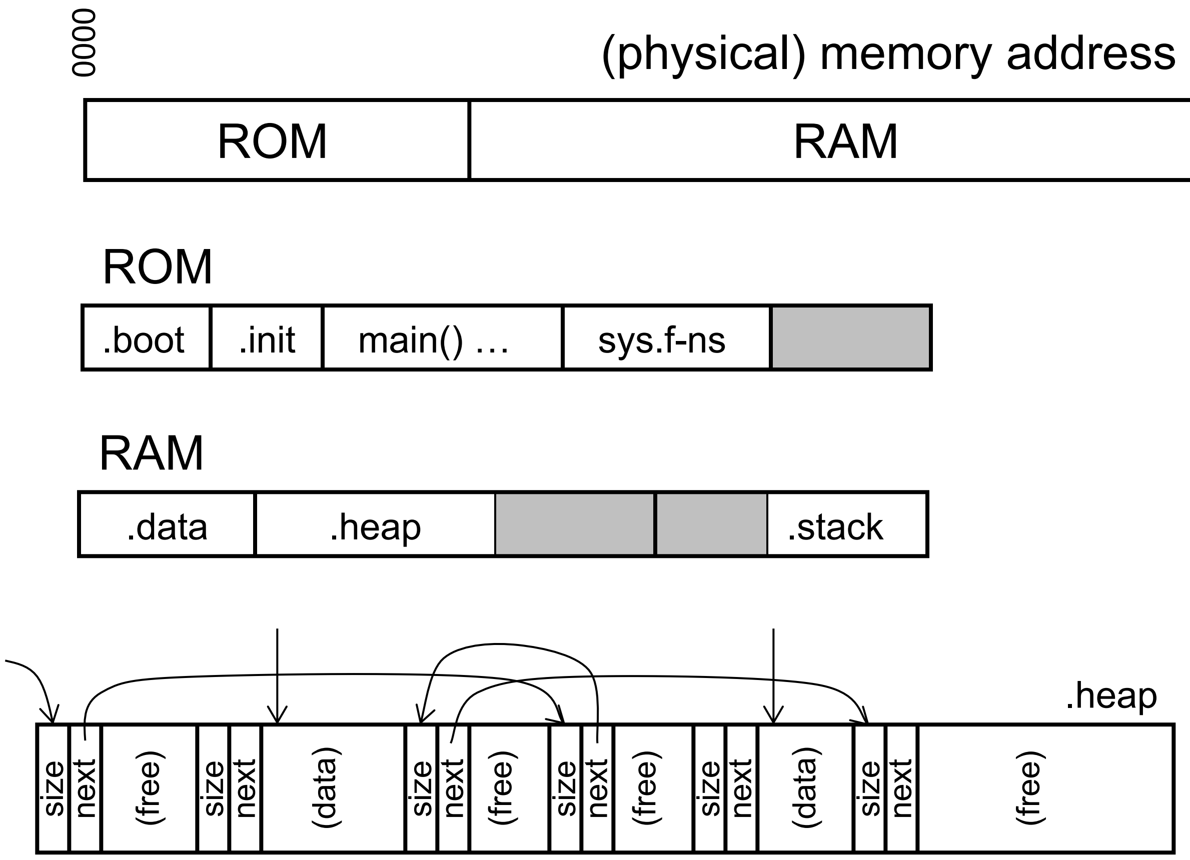
- The OS is a collection of complex services.
- These services manage resources and protects them from overuse and abuse.
- **User management**
  - Since the OS is responsible of protecting the computer resources from the user and user programs, a mechanism is put in place for user mode programs to access kernel mode in a way that that does not violate the rules set forth by the OS.
  - Of course, the memory manager participates in this mechanism by preventing any program from accessing memory that was not allocated to it.
  - **What happens when a user mode program tries to access memory outside its address space?**
    - In most cases, The Memory manager calls an ***Exception***.
    - Once the *Exception* is detected by the CPU, the **CPU switches to kernel mode immediately**. WHY? So that only the **OS running in Kernel mode can see and access the exception**.
    - Once the OS reads the exception, the **exception handler is kicked in** and it handles the exception by either **ending the process that the program is running** or **blocking the program entirely**.

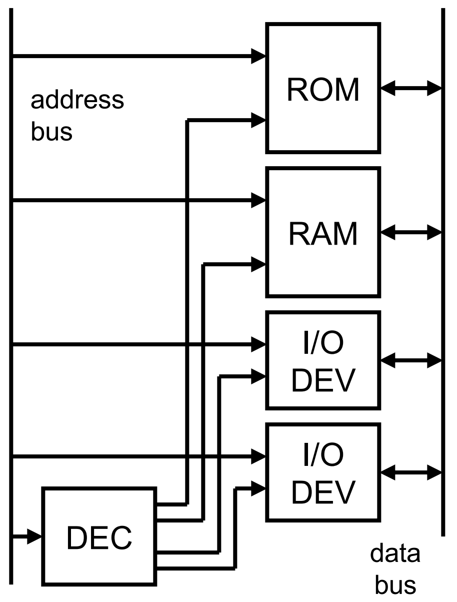
WHAT IS THE OS MADE OF

- Link to user managment:
  - https://jumpcloud.com/blog/what-is-user-management

OTHER THINGS ON THE OS

- MicroKernel: MINIX
- nanoKernels: Micro controllers – single task controllers

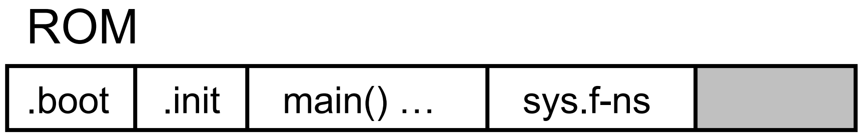




The physical module is selected/activated by decoder(s)

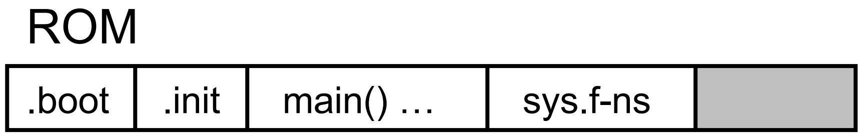
MEMORY LAYOUT

- **ROM** (Read-Only Memory)
        - RO segment (multiple programs/users)
    - **.boot** area
        - CPU initialization
        - Interrupt vectors
        - Addresses of stack and other memory areas
        - … or assigned by (RT)OS
    - **.init** area
        - Parameters for the program / Result for the OS
        - Initializing object (e.g., C++)



MEMORY LAYOUT

- **ROM** (cont.)
  - main() …
    - Applications / programs
    - Compiled or from libraries (as .o-fails)
  - sys.f-ns (system functions)
    - Standard subroutines
      - Compiler specific
    - System’s subroutines
      - OS specific
      - I/O drivers etc.

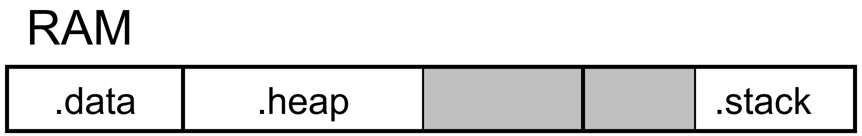


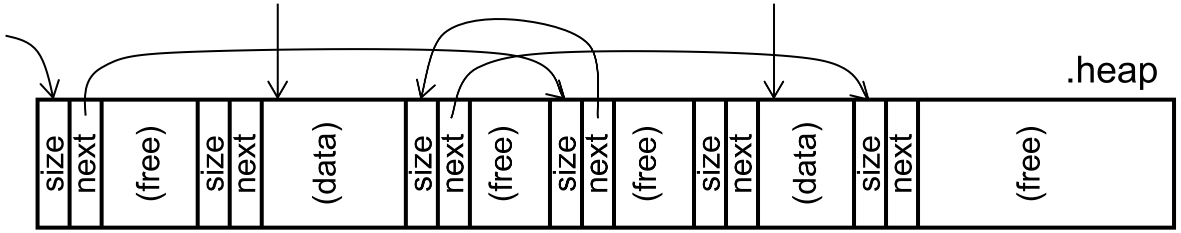
ROM

.boot	.init	main() …	sys.f-ns	

MEMORY LAYOUT

- **RAM** (Random Access Memory)
    - RW segment (multiple programs/users)
  - **.data** area
    - Static (global) variables/data
  - **.heap** area
    - Dynamic (global) variables/data
    - List of used/free blocks
      - Garbage collection may be needed (now and then)
      - Enlarged by OS when needed
  - **.stack** area
    - Local variables/data (stack)





MEMORY IMAGE

- **a.out** / **.exe** / **.com**
    - Image of a program, to be loaded into memory by OS
  - Contains
    - Program initialization & content (code)
    - Data with initial value (variables & constants)
      - Some of the global variables
  - Done by the OS
    - Preparing the memory layout
      - Allocating areas (MMU), pointer into registers, …
    - Controlling the program
      - Starting, interrupting, OS accesses, …