# Introduction to Programming

Labor 01

# Basic Information

- **Type of course:** Labor
- **Subject code:** INBPA0104-17
- **Credit:** 3
- https://elearning.unideb.hu/course/view.php?id=9468
- **Password/Enrollment key:** IntroProg2022
- **Lecturers:**
  - Piroska Biró, PhD
  - Anikó Apró

# Contact & Office Hours

- **Piroska Biró, PhD**
  - Office Hours: IK-227 or Online -> MS Teams
    - Tuesday 13:00-14:00
    - Wednesday 14:00-15:00
  - E-mail:
    - biro.piroska@inf.unideb.hu
- **Anikó Apró**
  - Office Hours: IK-229 or Online -> MS Teams
    - Monday  11:00-12:00
    - Tuesday 11:30-12:30
  - E-mail:
    - aniko.apro@inf.unideb.hu

# Requirements

- **Attendance and Participation:**
  - In every labor there will be an attendance sheet.
  - **Maximum three absences** are allowed in labor.
  - **Maximum 15 minutes late arrival** is accepted in labor.

**Assessment and grading:**

  - Midterm max. 100 points must be achieved **min. 50 points**
  - Endterm max. 100 points must be achieved **min. 50 points**
  - Midterm + Endterm max. 200 points must be achieved **min. 100 points**

- **Assessment: Practical mark**
  - To calculate the Final Grade the following formula and table should be used.
  - Final Point= (Midterm + Endterm)/2

| Grade | Final Point |
|---|---|
| 5 | 90 – 100 |
| 4 | 80 – 89 |
| 3 | 65 – 79 |
| 2 | 50 – 64 |
| 1 | 0 – 49 |

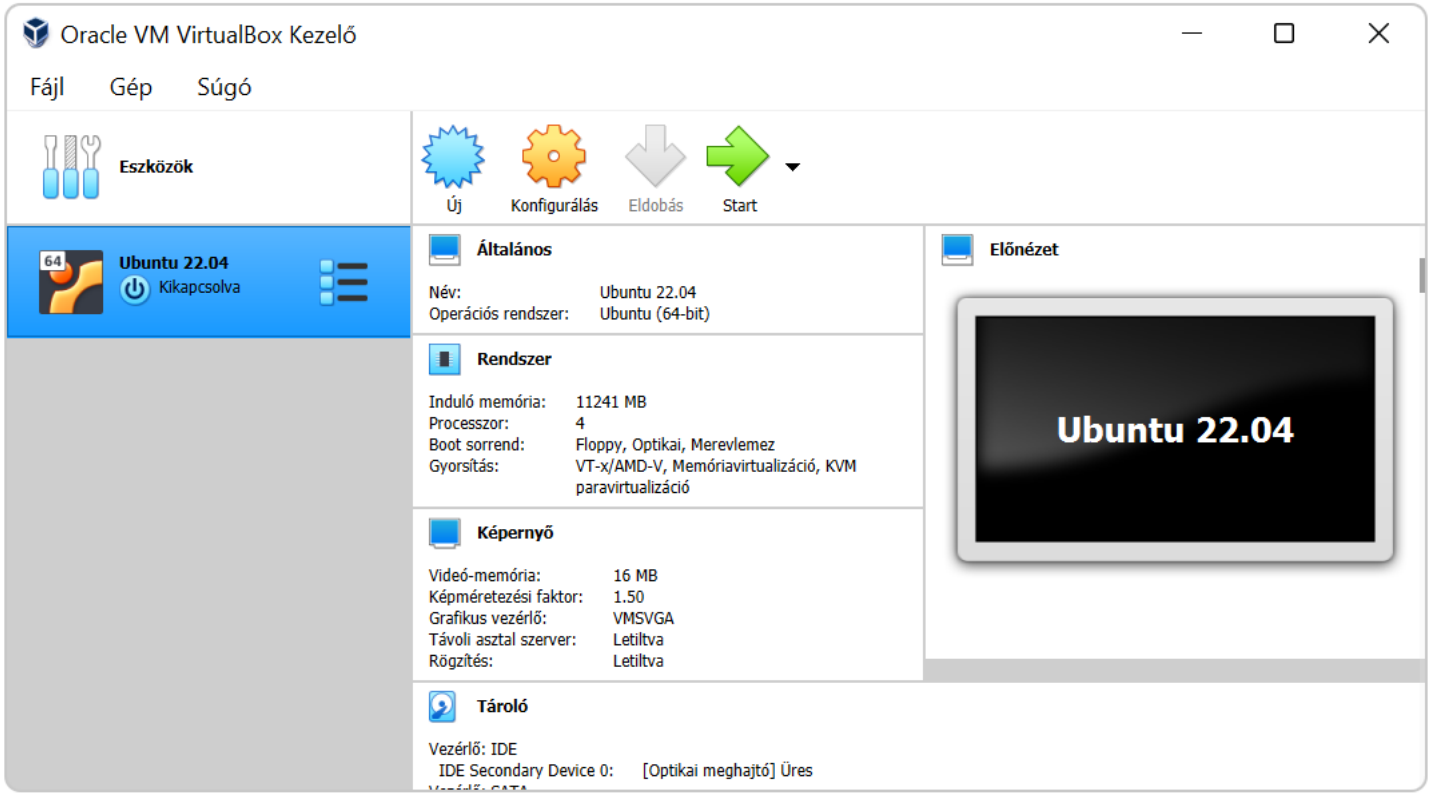
# References


- Ivor Horton: Beginning C, 5th edition, Apress, 2013, ISBN-13: 978-1430248811.
- **Brian W. Kernighan, Dennis M. Ritchie**: C Programming Language, 2nd edition, Prentice Hall, 1988, ISBN-13: 978-0131103627.
- Narasimha Karumanchi: Data Structures and Algorithmic Thinking with Python, CareerMonk Publications, 2015, ISBN-13: 978-819210759
- Robert Sedgewick: Algorithms in C, Parts 1-5 (Bundle): Fundamentals, Data Structures, Sorting, Searching, and Graph Algorithms, 3rd edition, Addison-Wesley Professional, 2001, ISBN-13: 978-0201756081.

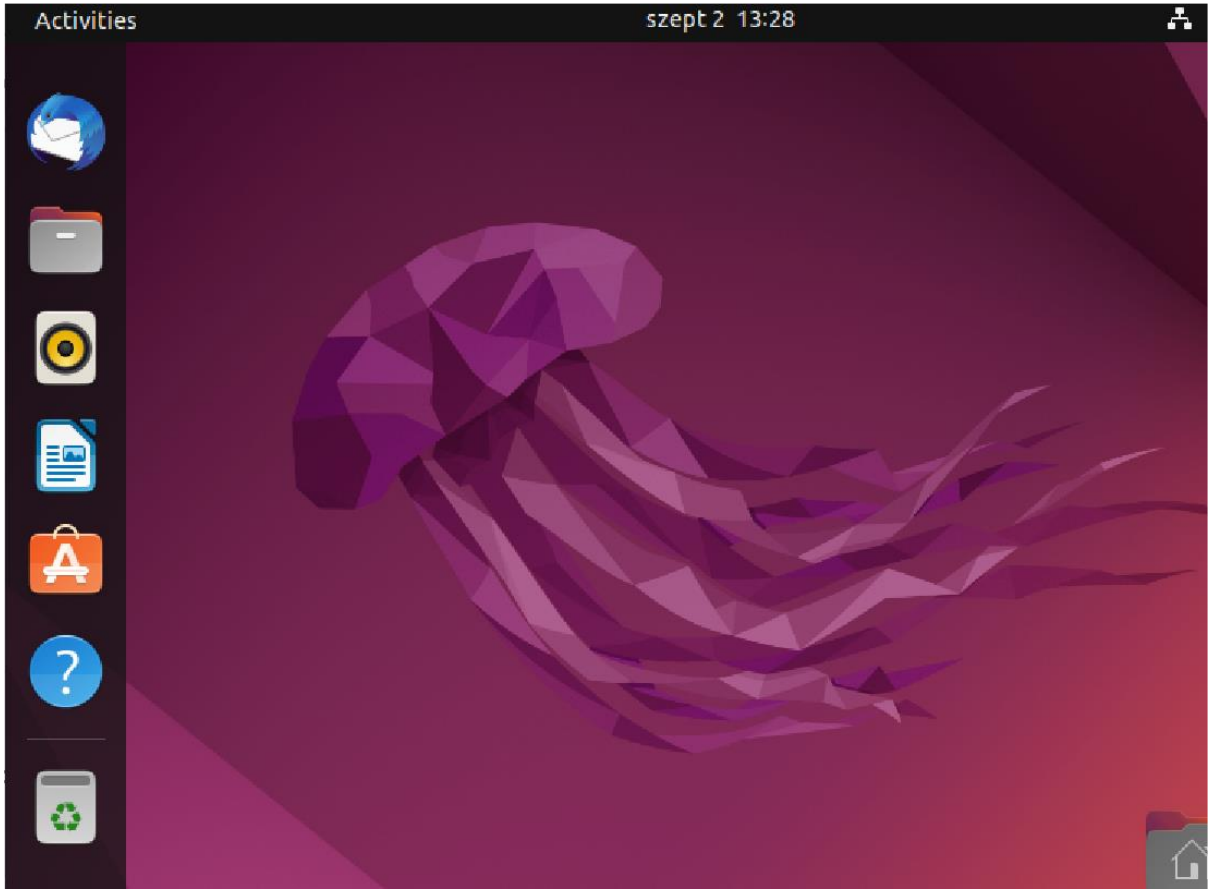
# Softver, environments

- Code::Blocks, Eclipse IDE for C/C++, NetBeans IDE C/C++, Visual Studio Code, CLion, Codelite, Atom, etc.
- **Linux environment**
- **Installing Virtual Machine – HOMEWORK!!!**
  - **VirtualBox:** https://www.virtualbox.org/wiki/Downloads
  - **Ubuntu 22.04.1 LTS:** https://www.ubuntu.com/download/desktop
  - **Help:** https://www.youtube.com/watch?v=v1JVqd8M3Yc
- **Online:**
  - **IDEONE** – https://ideone.com/
  - **Codingground** – http://www.tutorialspoint.com/compile_c_online.php
  - **Code()** – https://codeboard.io/
  - **Codepad** – http://codepad.org/

# VirtualBox


Oracle VM VirtualBox Kezelő
Fájl
Gép
Súgó
Eszközök
Új
Konfigurálás
Eldobás
Start
Ubuntu 22.04
Kikapcsolva
Általános
Név: Ubuntu 22.04
Operációs rendszer: Ubuntu (64-bit)
Rendszer
Induló memória: 11241 MB
Processzor: 4
Boot sorrend: Floppy, Optikai, Merevlemez
Gyorsítás: VT-x/AMD-V, Memóriavirtualizáció, KVM paravirtualizáció
Képernyő
Videó-memória: 16 MB
Képméretezési faktor: 1.50
Grafikus vezérlő: VMSVGA
Távoli asztal szerver: Letiltva
Rögzítés: Letiltva
Tároló
Vezérlő: IDE
IDE Secondary Device 0: [Optikai meghajtó] Üres
Előnézet
Ubuntu 22.04

# Ubuntu – 22.04.1 LTS




Latest release:

Ubuntu 22.04.1 LTS, April 21, 2022

# Install – Ubuntu

Terminal window

- sudo apt **udpate**
- sudo apt install **gcc**
- sudo apt install **indent**
- sudo apt install **mc** (Midnight Commander)

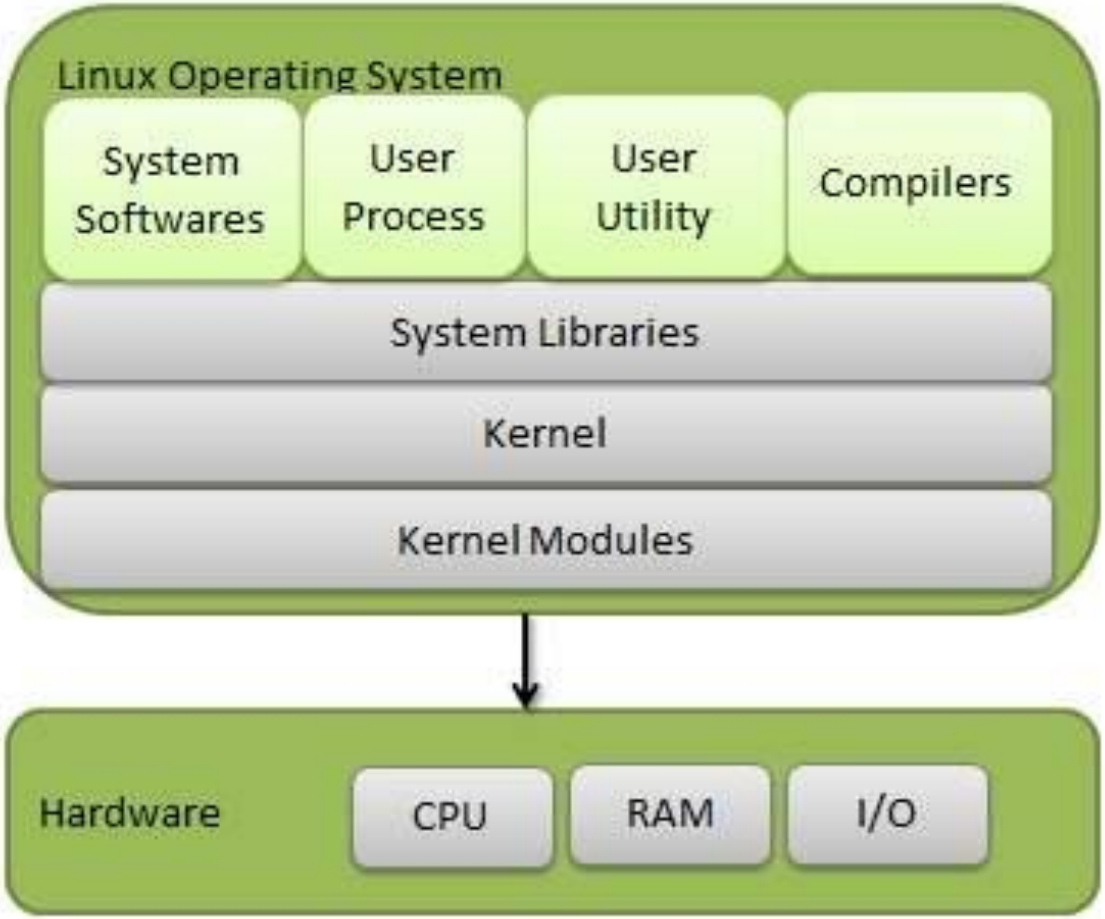
# Linux – Operating System

- **Linus Torvalds**
- Linux is one of popular version of UNIX operating System.
- It is **open source** as its source code is freely available.
- It is **free** to use.
- Linux was designed considering **UNIX** compatibility.
- Its functionality list is quite similar to that of UNIX.

# Components of Linux System

- **Kernel**
  - Kernel is the core part of Linux.
  - It is responsible for all major activities of this operating system.
  - It consists of various modules and it interacts directly with the underlying hardware.
  - Provides the required abstraction to hide low level hardware details to system or application programs.
- **System Library**
  - Are special functions or programs using them application programs or system utilities accesses Kernel's features.
  - These libraries implement most of the functionalities of the operating system and do not require kernel module's code access rights.
- **System Utility**
  - Programs are responsible for doing specialized, individual level tasks.

# Components of Linux System



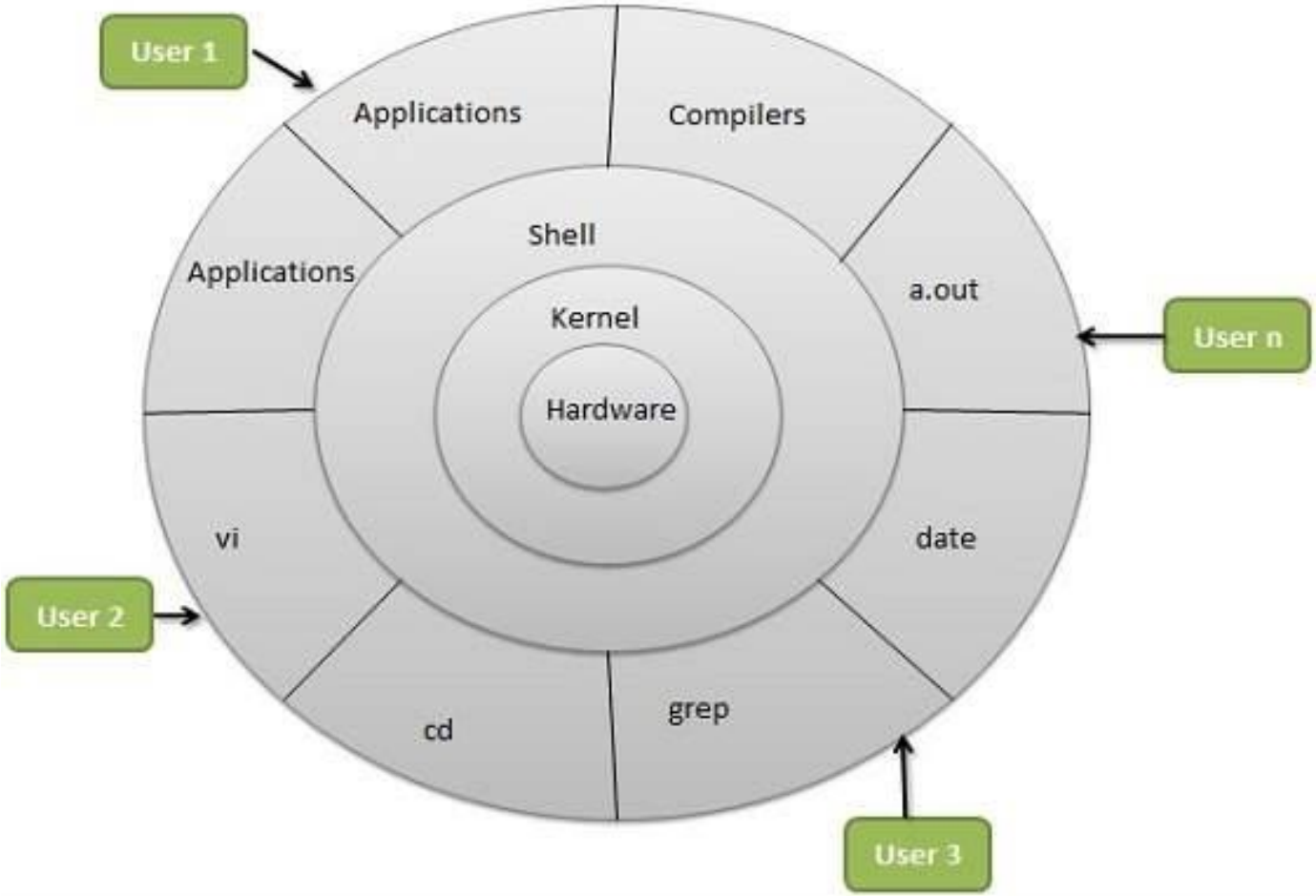
# Basic Features

- **Portable**
  - Portability means software can works on different types of hardware in same way.
  - Linux kernel and application programs support their installation on any kind of hardware platform.
- **Open Source**
  - Linux source code is freely available and it is community based development project.
  - Multiple teams work in collaboration to enhance the capability of Linux operating system and it is continuously evolving.
- **Multi-User**
  - Linux is a multiuser system means multiple users can access system resources like memory/ ram/ application programs at same time.

# Basic Features

- **Multiprogramming**
  - Linux is a multiprogramming system means multiple applications can run at same time.
- **Hierarchical File System**
  - Linux provides a standard file structure in which system files/ user files are arranged.
- **Shell**
  - Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs. etc.
- **Security**
  - Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

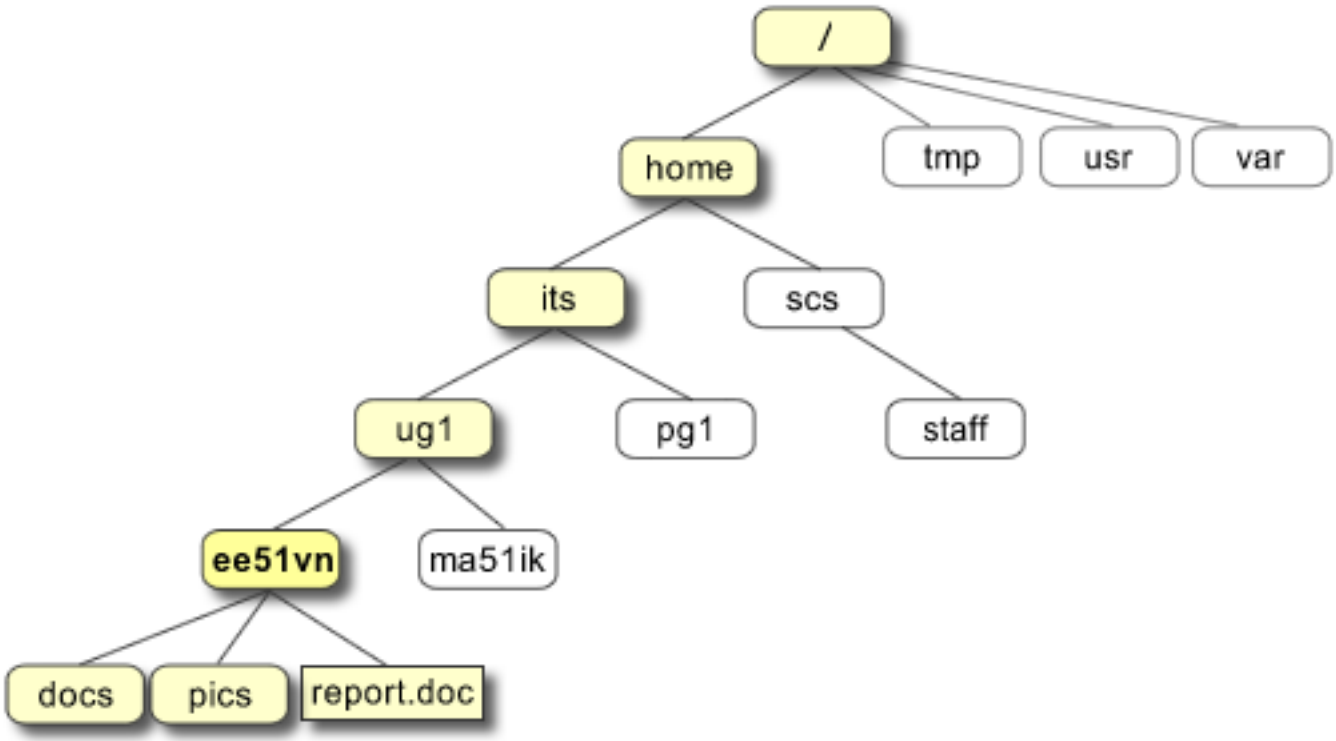
# Architecture



# Architecture

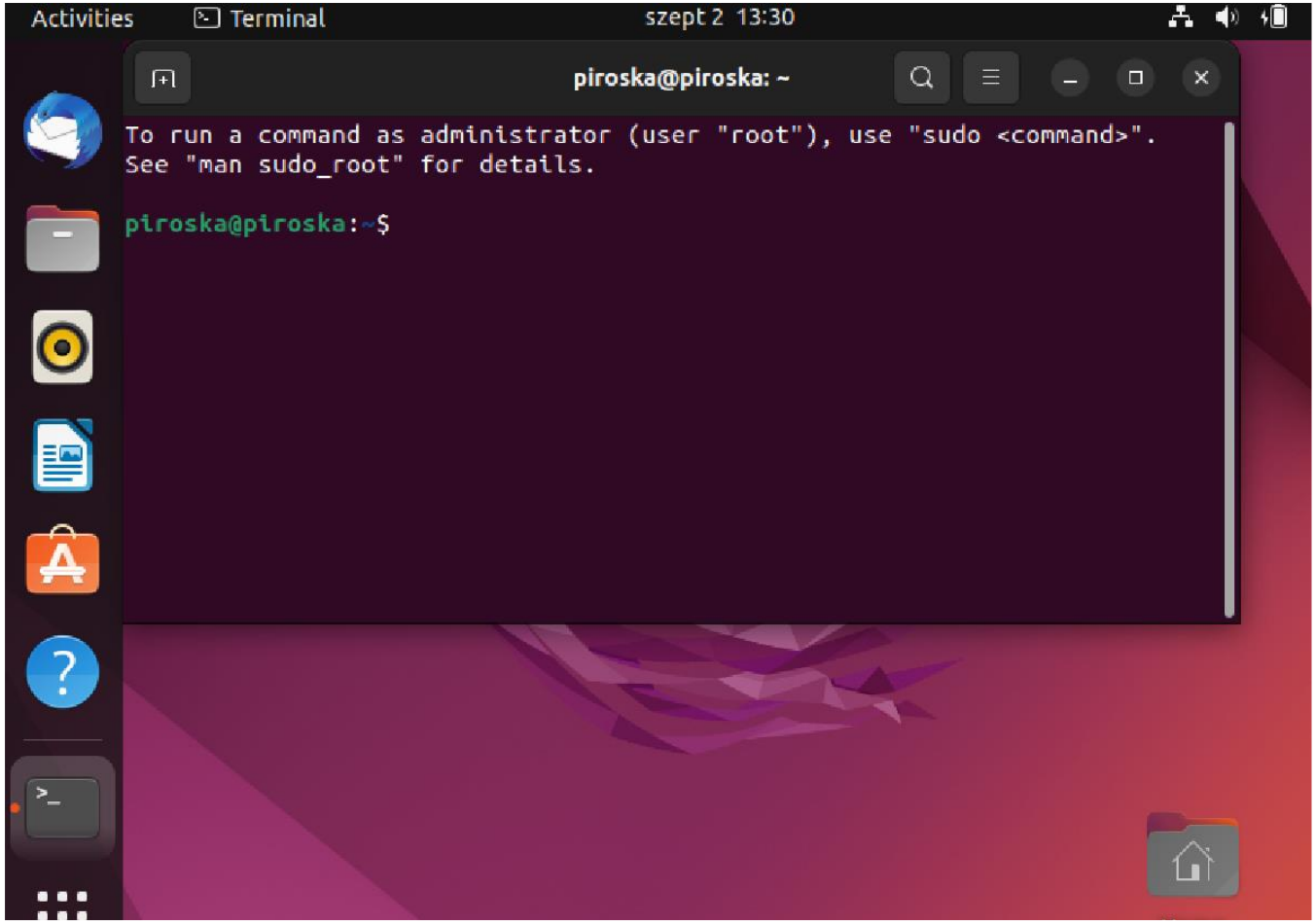
- **Hardware layer**
  - Hardware consists of all peripheral devices (RAM/HDD/CPU etc).
- **Kernel**
  - It is the core component of Operating System, interacts directly with hardware, provides low level services to upper layer components.
- **Shell**
  - An interface to kernel, hiding complexity of kernel's functions from users.
  - The shell takes commands from the user and executes kernel's functions.
- **Utilities**
  - Utility programs that provide the user most of the functionalities of an operating systems.

# The Directory Structure
## Hierarchical File System




/home/its/ug1/ee51vn/report.doc

# Starting an UNIX Terminal



# Basic Unix/Linux Commands

- **man**: an interface to the online reference manuals
- for more details, use MAN

  $**man** command
- For example:

  $**man** mkdir

# Basic Unix/Linux Commands

- **cat**: displays File Contents
  - cat hello.c
- **cd**: change  Directory
- **cd** *dirname*: changes Directory to *dirname*
- **cd** .. : move to the parent directory of the current directory.
- **cd** /: move to the root directory

# Basic Unix/Linux Commands

- **ls**: directory listing
  - ls
- **pwd**: show current directory
- **head**: displays the first 10 lines of a file
  - head hello.c
- **more**: display the contents of file
  - more hello.c

# Basic Unix/Linux Commands

- **mkdir** *dirname*: create a directory *dirname*
  - mkdir lab01
- **rm**: delete files or directory
  - rm hello.c
- **cp**: copies source file into destination
  - cp source.c destination.c
- **mv** *file1 file2* – rename or move *file1* to *file2,* if *file2* is an existing directory, moves *file1* into directory *file2*

# Practice

- Make a new directory and enter it
- Creating a new file: hello.c
  - **mkdir** lab01
  - **cd** lab01

- Make a new file:
  - **gedit** hello.c  or use a **Text Editor**

- Word processors:
  - gedit, joe, mcedit, nano, vi, kate

# hello.c – The first C program

**$gedit hello.c or open a Text Editor**

```
#include <stdio.h>
int main() {
        printf(”Hello!”);
        return 0;
}
```

# hello.c

- the **#include <stdio.h>** is a preprocessor command. This command tells compiler to include the contents of stdio.h (standard input and output) file in the program.
- the **stdio.h** file contains functions such as scanf() and printf() to take input and display output respectively.
- if you use **printf()** function without writing #include <stdio.h>, the program will not be compiled.
- the execution of a C program starts from the main() function.
- the **printf()** is a library function to send formatted output to the screen. In this program, the printf() displays **Hello!** text on the screen.
- the **return 0**; statement is the "Exit status" of the program. In simple terms, program ends with this statement.

# Compiling C Program

- **Regular source code files.** These files contain function definitions, and have names which end in **".c"** by convention.
- **Header files.** These files contain function declarations and various preprocessor statements. They are used to allow source code files to access externally-defined functions. Header files end in **".h"** by convention.
- **Object files.** These files are produced as the output of the compiler. They consist of function definitions in binary form, but they are not executable by themselves. Object files end in **".o"** by convention, although on some operating systems (Windows), they often end in ".obj".
- **Binary executables.** These are produced as the output of a program called a "linker". The linker links together a number of object files to produce a binary file which can be directly executed. Binary executables have no special suffix on Unix operating systems, although they generally end in ".exe" on Windows.

# Making the object file: the compiler

- After the C **preprocessor** has included all the header files and expanded out all the **#define** and **#include** statements (as well as any other preprocessor commands that may be in the original file), the compiler can compile the program.
- It does this by turning the C source code into an **object code** file, which is a file ending in ".o" which contains the binary version of the source code. Object code is not directly executable, though.
- In order to make an executable, you also have to add code for all of the library functions that were #included into the file (this is not the same as including the declarations, which is what #include does). This is the job of the **linker**.

# Making the object file: the compiler

- In general, the compiler is invoked as follows:

  **$ gcc –c hello.c**

- where $ is the unix prompt.
- This tells the compiler to run the preprocessor on the file hello.c and then compile it into the object code file **hello.o**.
- The **–c** option means to compile the source code file into an object file but not to invoke the linker.

# Making the object file: the compiler

- If your entire program is in one source code file, you can instead do this:

  **$ gcc hello.c -o hello**

- This tells the compiler to run the preprocessor on hello.c, compile it and then link it to create an executable called **hello**.

  **$ ./hello**

- The -o option states that the next word on the line is **the name of the binary executable file** (program).
- If you don't specify the -o, i.e. if you just type **gcc hello.c**, the executable will be named **a.out** for silly historical reasons.
- Note also that the name of the compiler we are using is gcc, which stands for "GNU C compiler" or "GNU compiler collection" depending on who you listen to. Other C compilers exist; many of them have the name cc, for "C compiler". On Linux systems cc is an alias for gcc.

# Putting it all together: the linker

- The job of the **linker** is to link together a bunch of object files (.o files) into a binary executable. This includes both the object files that the compiler created from your source code files as well as object files that have been pre-compiled for you and collected into library files.
- Like the preprocessor, the linker is a separate program called ld. Also like the preprocessor, the linker is invoked automatically for you when you use the compiler. The normal way of using the linker is as follows:

```
$ gcc hello.o first.o -o myprog
```

- This line tells the compiler to link together three object files (hello.o, and first.o) into a binary executable file named. Now you have a file called **myprog** that you can run and which will hopefully do something cool and/or useful.
- This is all you need to know to begin compiling your own C programs.

# Recommend command-line options

- Generally, we also recommend that you use the following command-line option:

```
$ gcc -Wall hello.c -o hello
```

- The **-Wall** option causes the compiler to warn you about legal but dubious code constructs, and will help you catch a lot of bugs very early.

```
$ gcc -Wall -ansi -pedantic-errors hello.c -o hello
```

- The **-ansi** and **-pedantic** options cause the compiler to warn about any non-portable construct (e.g. constructs that may be legal in gcc but not in all standard C compilers; such features should usually be avoided).

# Homework

- Install an Ubuntu Virtual Machine!
- Try and learn the basic Unix/Linux Commands!
- Write a C program that should write a single line to the standard output containing the string „Have a nice day!".