# **Laboratorio # 1 Sistemas Operativos Grupo 80**

# **2242232-2724, Orejuela Albornoz**

# **2340816-2724, Albornoz Suárez**

# **2343025-2724, Silva Zuñiga**

# 

# **Operating Systems**

# **Jefferson A.**

# **Cali, 2024-II**

# **Laboratorio # 1 Sistemas Operativos Gr. 80.**

## **General Description**

The following outlines the guidelines for a theoretical-practical activity for students in the Operating Systems course. The activity specifically assesses their mastery of some basic concepts, definitions related to CPU management, processes, scheduling, and the implementation of a simulator using C or C++.

**Objectives** During the development of the activities, students will achieve the following:

● Enumerate the milestones in the evolution of computer systems.

● Describe the objectives and functions of modern operating systems.

● Simulate the behavior of the CPU and process management.

● Apply the principles of scheduling.

### **Before You Begin**

Read the following:

* *Operating System Concepts* (9th Edition) by Silberschatz and Galvin: Chapters 1, 2, and 3
* *Modern Operating Systems* by Tanenbaum: Chapters 1 and 2

Create a git repository and add the link:

|  |
| --- |

Use this document as template [here]

## **Activity No. 1: Conceptualization [40%]**

After completing the recommended readings in this document, answer the following questions in your own words, concisely and accurately.

1. **Enumerate the milestones in the evolution of computer systems**

**HISTORY OF OPERATING SYSTEMS**

**The First Generation (1945–55): Vacuum Tubes**

These early computers were primitive, using vacuum tubes or relays, and required direct programming in machine language or by wiring plugboards. By the early 1950s, the process improved slightly with the introduction of punched cards for programming.

**The Second Generation (1955–65): Transistors and Batch Systems**

The introduction of the transistor in the mid-1950s revolutionized computing by making computers more reliable and allowing them to be sold to customers.

**The Third Generation (1965–1980): ICs and Multiprogramming**

In the early 1960s, IBM introduced the System/360, a series of compatible computers for both scientific and commercial use, simplifying product lines and ensuring program compatibility across models. This approach continued with the 370 and zSeries.

**The Fourth Generation (1980–Present): Personal Computers**

The development of LSI circuits led to personal computers. Intel's 8080 microprocessor in 1974 used CP/M, an OS developed by Gary Kildall. IBM missed out on CP/M and instead bought DOS from Seattle Computer Products through Bill Gates. MS-DOS became dominant, evolving with features from UNIX.

**The Fifth Generation (1990–Present): Mobile Computers**

The idea of portable communication devices has been around since the 1940s, starting with large mobile phones and evolving to the first handheld "brick" phones in the 1970s. Today, mobile phones are ubiquitous, used for much more than calling.

1. **What are the four components of a computer system? Describe each one.**

**Processors**

The ‘‘brain’’ of the computer is the CPU. It fetches instructions from memory and executes them. The basic cycle of every CPU is to fetch the first instruction from memory, decode it to determine its type and operands, execute it, and then fetch, decode, and execute subsequent instructions. The cycle is repeated until the program finishes. In this way, programs are carried out.

**Memory**

The second major component in anycomputer is the memory.Ideally,amemory should be extremely fast (faster than executing an instruction so that the CPU is not held up by the memory), abundantly large, and dirt cheap. No current technology satisfies all of these goals, so a different approach is taken. The memory system is constructed as a hierarchyof layers, Thetop layers have higher speed, smaller capacity, and greater cost per bit than the lower ones, often by factors of a billion or more.

**Disk**

Next in the hierarchy is magnetic disk (hard disk). Disk storage is two orders of magnitude cheaper than RAM per bit and often two orders of magnitude larger as well. The only problem is that the time to randomly access data on it is close to three orders of magnitude slower. The reason is that a disk is a mechanical device,

**I/O Devices**

These devices consist of a controller and the device itself. The controller, often a chip or set of chips, handles commands from the operating system, like reading data, and performs complex tasks such as converting data formats and managing device operations. The actual device has a simpler interface but relies on the controller to interact with the operating system. Device drivers are specialized software needed to manage each type of controller and are typically integrated into the operating system kernel, though some systems allow drivers to run in user space.

1. **What is the difference between a monolithic kernel and a microkernel?**

In a monolithic kernel, the entire operating system operates as a single program in kernel mode. This approach allows for efficient inter-procedure calls but can lead to a complex and potentially unstable system, as a bug in any part of the kernel can crash the entire system. The system is organized into a main program, service procedures, and utility procedures, all of which run in kernel mode.

In contrast, a microkernel architecture splits the operating system into a small kernel that runs in kernel mode and many other modules (like device drivers and file systems) that run in user mode. This separation improves system reliability because a bug in one of these user-mode components does not affect the entire system. The microkernel handles essential functions such as interprocess communication and basic scheduling, while other functionalities are managed by separate user processes. This design makes the system more modular and resilient to crashes.

1. **Define an Operating System from two different perspectives.**

An operating system (OS) serves two main functions: abstracting hardware complexity and managing hardware resources.

**Operating System as an Extended Machine:**

The OS provides an abstraction layer that simplifies the interaction with hardware. For example, it hides the complexities of hardware like SATA disks behind simpler interfaces like file systems, enabling programs to perform tasks (e.g., reading/writing files) without dealing with intricate hardware details. This abstraction makes complex hardware interactions manageable and efficient.

**Operating System as a Resource Manager:**

The OS manages and allocates system resources such as CPU, memory, and I/O devices among multiple programs and users. It ensures orderly access to these resources to prevent conflicts and inefficiencies. For instance, it handles tasks like time and space multiplexing—time multiplexing allows multiple programs to share a single resource like the CPU or printer in turns, while space multiplexing involves sharing resources like memory and disk space simultaneously among different programs.

1. **What is the purpose of system calls?**

***Answer/*** System calls act as a bridge between user programs and the operating system, allowing programs to access services and resources that are typically managed by the operating system. When a program needs to perform a task that requires system-level permissions, it makes a system call, triggering a special instruction that shifts control to the operating system's kernel. The kernel, which operates in a secure and privileged mode, takes over, interprets the request, checks the parameters, and carries out the required operation. Once the task is completed, control is handed back to the user program.

1. **What is a multiprogrammed operating system?**

***Answer//*** Is a type of operating system designed to maximize CPU utilization by allowing multiple programs to be loaded and executed concurrently.

### Four main characteristics:

1. **Concurrency:** several processes are kept in memory at the same time, allowing the operating system to switch between processes and execute them in an interleaved manner, which helps in efficiently utilizing the CPU.
2. **Process Management:** using techniques like scheduling and context switching, processes are selected for execution based on a scheduling algorithm, and the state of a process is saved and restored during context switches.
3. **Memory Management:** using memory management techniques such as partitioning (dividing memory into sections) or paging (dividing memory into fixed-size pages) to allocate memory to multiple processes.
4. **I/O and CPU Overlap:** When one process is waiting for I/O operations (like reading from a disk), the CPU can be allocated to another process that is ready to execute. This overlap of I/O operations and CPU execution helps in maintaining high CPU utilization.
5. **What is a process?**

***Answer/*** A process is a program that has been loaded into memory and is currently being executed by the CPU. In other words a process could be considered as a running instance.

1. **What are the states of a process?**

***Answer /*** “each process is an independent entity, with its own program counter and internal state, processes often need to interact with other processes”

Andrew S. Tanenbaum. Modern operating systems . pearson, 1992.

The state of a process is determined by current activity of the process. According to Tanunbaum there is three states that a process can be:

* Running (actually using the CPU at that instant).
* Ready (runnable; temporarily stopped to let another process run).
* Blocked (unable to run until some external event happens).

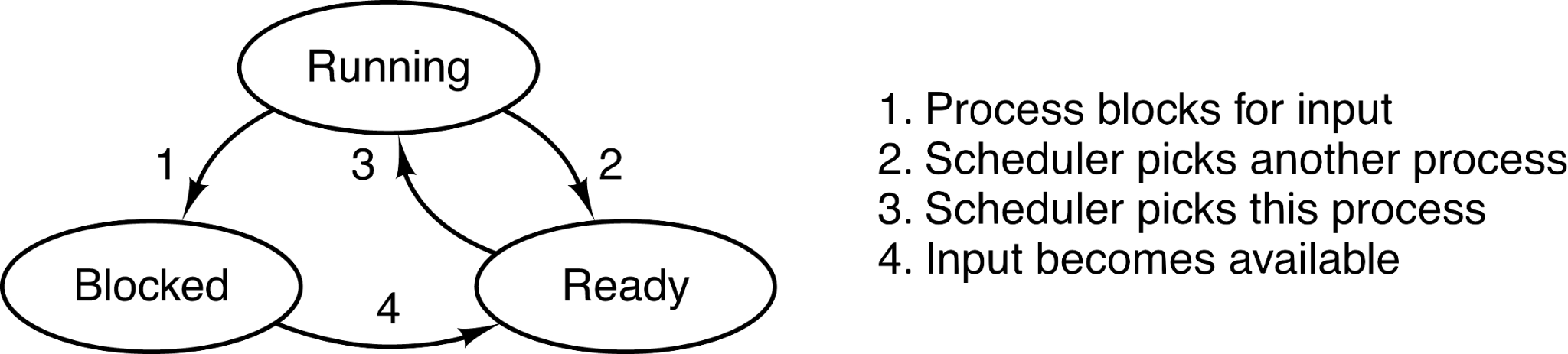


image from “Andrew S. Tanenbaum. Modern operating systems . pearson, 1992.”

1. **What information is stored in the Process Control Block (PCB) associated with a process?**

***Answer/*** The PCB stored process tables, it means arrays of structures with information like process state, process ID, program counter, cpu registers, memory management information, scheduling information, accounting information, i/o status information, list of open files and process hierarchy information.

1. **What are the main activities of an operating system in relation to process management?**

**Answer/** In the context of process management, here are the main activities:

1. Scheduling processes and threads on the cpus
2. Creating and deleting both user and system processes
3. Suspending and resuming processes
4. Providing mechanisms for process synchronization
5. Providing mechanisms for process communication.

## **Activity No. 2: Remembering the C and C++ languages [50%]**

1. Implements the programs and functions below:

| 1 | Write a program in c to find the leap year.  #include <stdio.h>  int main(void) {  //2.1. Write a program in c to find the leap year.  int year=1900;  if (year % 400 == 0) {  printf("%d is a leap year\n", year);  } else if (year % 100 == 0) {  printf("%d is not a leap year\n", year);  } else if (year % 4 == 0) {  printf("%d is a leap year\n", year);  } else {  printf("%d is not a leap year\n", year);  }    }    here declaring the variable “year” as 1900(is not a leap year) |
| --- | --- |
| 2 | Write a program in c to calculate factorial of a number  #include <stdio.h>  int main() {  int number = 3; // Example input  int acum = 1; // Initialize the accumulator to 1  // Factorial calculation directly in main  if (number == 0) {  acum = 1; // Base case: 0! = 1  } else {  for (int i = 1; i <= number; i++) {  acum = acum \* i; // Multiply acum by each number from 1 to 'number'  }  }  // Print the result  printf("The factorial of %d is %d\n", number, acum);  return 0;  } |
| 3 | Write a program in c to calculate power using recursion.  #include <stdio.h>  //2.3 Write a program in c to calculate power using recursion.  int power(int base, int exponent) {  if (exponent == 0) {  return 1; // Base case: any number to the power of 0 is 1  } else {  return base \* power(base, exponent - 1); // Recursive case  }  }  int main() {  int base = 2; // Example base  int exponent = 3; // Example exponent  int result;  result = power(base, exponent); // Calculate power using recursion  printf("%d to the power of %d is %d\n", base, exponent, result);  return 0;  } |
| 4 | Write a program in c to find even or odd numbers.  #include <stdio.h>  int main()  {  int number;    printf("Ingrese un número entero. ");  scanf("%d", &number);    if (number % 2 == 0)  {  printf("%d es par ", number);  }  else{  printf("%d es impar ", number);  }    return 0;  } |
| 5 | Write a program in c to print fabonnaci series.  #include <stdio.h>  int main()  {  int n, primero = 0, segundo = 1, siguiente;    printf("Digite el numero de terminos: ");  scanf("%d", &n);    printf("La serie de Fibonacci: \n");    for (int i = 0; i < n; i++)  {  if (i <= 1){  siguiente = i;  }  else {  siguiente = primero + segundo;  primero = segundo;  segundo = siguiente;  }    printf("%d\n", siguiente);  }    return 0;  } |
| 6 | Write a function to check uppercase letter  #include <stdio.h>  #include <ctype.h>  int esMayuscula(char caracter)  {    // Verifica si en la tabla ASCII la variable esta entre los valroes de las mayusculas    if (caracter >= 'A' && caracter <= 'Z')  {  return 1; // si es mayúscula retornará uno  }  else  {  return 0;  }  }  int main()  {  char caracter;  printf("Ingrese el caracter:");  scanf("%c", &caracter);    // se llama a la funcion    if (esMayuscula(caracter))  {  printf("%c es una letra mayúscula. \n", caracter);  }else{  printf("%c no es una letra mayúscula. \n", caracter);  }    return 0;  } |
| 7 | Write a program in c to function to check lowercase letter.  #include <stdio.h>  #include <ctype.h>  int esMinuscula(char caracter)  {    // Verifica si en la tabla ASCII la variable esta entre los valroes de las minusculas    if (caracter >= 'a' && caracter <= 'z')  {  return 1; // si es minuscula retornará uno  }  else  {  return 0;  }  }  int main()  {  char caracter;  printf("Ingrese el caracter:");  scanf("%c", &caracter);    // se llama a la funcion    if (esMinuscula(caracter))  {  printf("%c es una letra minuscula. \n", caracter);  }else{  printf("%c no es una letra minuscula. \n", caracter);  }    return 0;  } |
| 8 | **Find the greater of the three numbers.**  #include <stdio.h> ***//Biblioteca estándar de entrada y salida en C.***  int main() {  int numero1, numero2, numero3; ***// Definición de variables.***  printf("Digitar el primer número: "); ***// Ingreso de los números.***  scanf("%d", &numero1);  printf("Digitar el segundo número: ");  scanf("%d", &numero2);  printf("Digitar el tercer número: ");  scanf("%d", &numero3);  if(numero1 >= numero2 && numero1 >= numero3){ ***//EStructura condicional para determinar el numero mayor***  printf("El mayor número es: %d\n", numero1);  } else if(numero2 >= numero1 && numero2 >= numero3){  printf("El mayor número es: %d\n", numero2);  } else{  printf("El mayor número es: %d\n", numero3);  }  return 0;  } |
| 9 | Write a program in c to type casting implicit explicit.  ***//Biblioteca estándar para ingreso y salida de datos***  #include <stdio.h>  ***//Definicion de variables a transformar***  int main() {  int intVar;  float floatVar;  double doubleVar;  ***//Ingreso número entero***  printf("Ingrese un número entero: ");  scanf("%d", &intVar);  ***//Ingreso dato tipo float***  printf("Ingrese un número tipo float: ");  scanf("%f", &floatVar);  ***//Conversión implícita de int a double***  doubleVar= intVar;  printf("Conversión implícita (int a double): %d a %.2f\n", intVar, doubleVar);  ***//Conversión implícita de float a double***  doubleVar= floatVar;  printf("Conversión implícita (float a double): %.2f a %.2f\n", floatVar, doubleVar);  ***//Conversión explícita de float a int***  intVar= (int)floatVar;  printf("Conversión explícita (float a int): %.2f a %d\n", floatVar, intVar);  return 0;  } |
| 10 | Write a program to display number 1 to 10 in octal, decimal and hexadecimal system.  #include <stdio.h>  int main() {  ***//Encabezado***  printf("Número\t Decimal Octal\t Hexadecimal\n");  printf("------------------------------------------\n");  ***//Bucle para iterar de 1 a 10***  for (int i = 1; i <= 10; i++) {  ***//Imprimir el número en diferentes formatos***  ***//%d imprime el número en formato decimal***  ***//%o imprime el número en formato octal***  ***//%x imprime el número en formato hexadecimal***  printf("\t%d\t\t%d\t\t%o\t\t%X\n", i, i, i, i);  }  return 0;  } |

1. Write a C program that receives and processes grades for the Operating Systems (SO) course using structures.

***// Biblioteca estándar de entrada y salida en C***

#include <stdio.h>

***// Estructura para almacenar la información del estudiante***

struct Estudiante {

char nombre[30];

int nota;

};

int main() {

int cantidad, i;

float promedio = 0.0;

int notaMaxima = -1; ***//Variable inicializada con un valor menor que cualquier calificación posible como referencia.***

int notaMinima = 6; ***//Variable inicializada con un valor mayor que cualquier calificación posible como referencia.***

***//Solicitar al usuario el número de estudiantes***

printf("Ingrese el número de estudiantes en el curso: ");

scanf("%d", &cantidad);

***// Array de estructuras Estudiante***

struct Estudiante estudiantes[cantidad];

***// Ciclo for para ingresar información de los estudiantes***

for(i = 0; i < cantidad; i++) {

printf("Ingrese el nombre del estudiante %d: ", i + 1);

scanf("%s", estudiantes[i].nombre);

printf("Ingrese la calificación de %s: ", estudiantes[i].nombre);

scanf("%d", &estudiantes[i].nota);

***//Suma de calificaciones para el promedio***

promedio += estudiantes[i].nota;

***//Calificación más alta***

if(estudiantes[i].nota > notaMaxima) {

notaMaxima = estudiantes[i].nota;

}

***//Calificación más baja***

if(estudiantes[i].nota < notaMinima) {

notaMinima = estudiantes[i].nota;

}

}

/***/Cálculo del promedio de las calificaciones***

promedio /= cantidad;

***//Resultados***

printf("\nReporte de calificaciones:\n");

printf("Promedio de calificaciones: %.2f\n", promedio);

***//Ciclo for para mostrar las calificaciones de los estudiantes***

for (i = 0; i < cantidad; i++) {

printf("Estudiante: %s, Calificación: %d\n", estudiantes[i].nombre, estudiantes[i].nota);

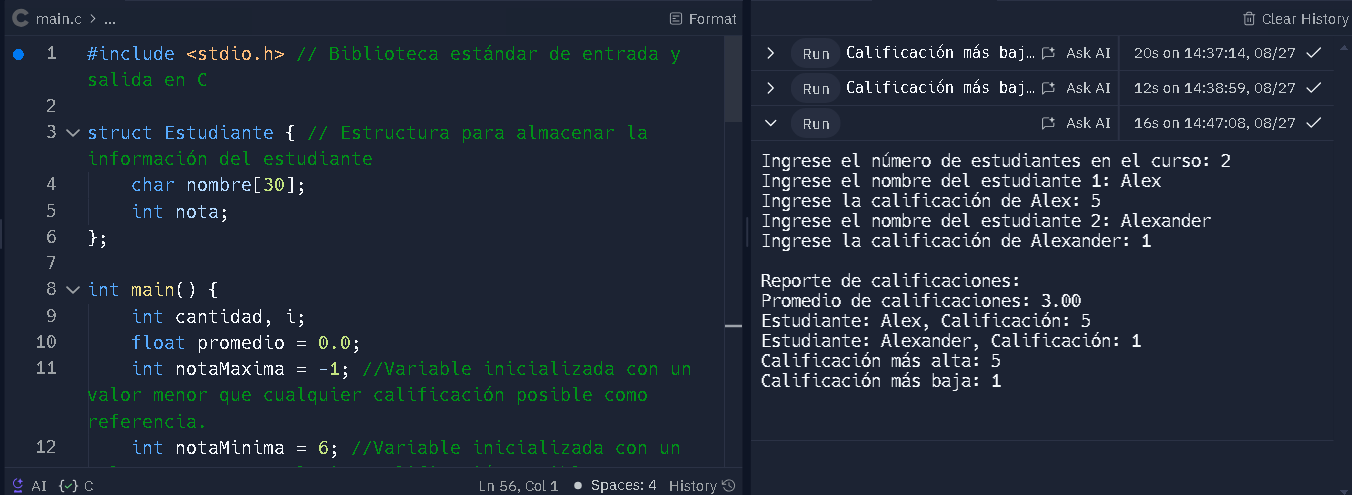
}

printf("Calificación más alta: %d\n", notaMaxima);

printf("Calificación más baja: %d\n", notaMinima);

return 0;

}



1. Write a C++ program that performs the following tasks: First, the program should receive a numeric value and determine whether it is a prime number, displaying the result. Second, the program should accept a list of numbers and identify the prime numbers within that list. Lastly, the program should allow the user to input a numeric range and display all prime numbers within that specified range.

Answer: #include <iostream>

#include <vector>

/\*Write a C++ program that performs the following tasks: First, the program should receive a numeric value and determine whether it is a prime number, displaying the result. Second, the program should accept a list of numbers and identify the prime numbers within that list. Lastly, the program should allow the user to input a numeric range and display all prime numbers within that specified range.\*/

// Function to check if a number is prime

bool isPrime(int num) {

if (num <= 1) return false;

for (int i = 2; i \* i <= num; ++i) {

if (num % i == 0) return false;

}

return true;

}

// Function to display primes in a list or range

void displayPrimes(const std::vector<int>& numbers) {

std::cout << "Prime numbers: ";

for (int num : numbers) {

if (isPrime(num)) {

std::cout << num << " ";

}

}

std::cout << "\n";

}

int main() {

int choice;

std::cout << "Select an option:\n";

std::cout << "1. Check if a single number is prime\n";

std::cout << "2. Identify prime numbers in a list\n";

std::cout << "3. Display prime numbers within a range\n";

std::cout << "Enter your choice: ";

std::cin >> choice;

if (choice == 1) {

int num;

std::cout << "Enter a number: ";

std::cin >> num;

if (isPrime(num)) {

std::cout << num << " is a prime number.\n";

} else {

std::cout << num << " is not a prime number.\n";

}

} else if (choice == 2) {

int n;

std::cout << "Enter the number of elements: ";

std::cin >> n;

std::vector<int> numbers(n);

std::cout << "Enter the numbers:\n";

for (int i = 0; i < n; ++i) {

std::cin >> numbers[i];

}

displayPrimes(numbers);

} else if (choice == 3) {

int start, end;

std::cout << "Enter the start of the range: ";

std::cin >> start;

std::cout << "Enter the end of the range: ";

std::cin >> end;

std::vector<int> numbers;

for (int i = start; i <= end; ++i) {

numbers.push\_back(i);

}

displayPrimes(numbers);

} else {

std::cout << "Invalid choice.\n";

}

return 0;

}

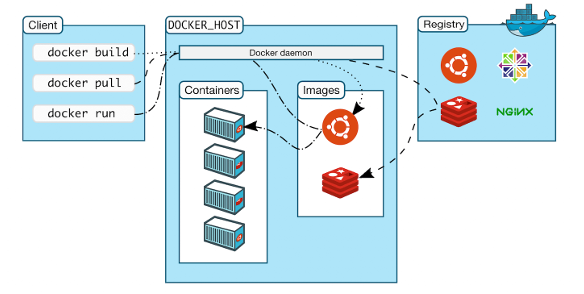
1. Create a program with C++ to model geometric shapes and perform area, perimeter, and color operations. Rectangle, square, triangle, etc. shapes must be considered. Tip. Use classes and inheritance. //this program is in folder 4 of activity 2

**Activity No. 3: Docker practical session [10%]**

Docker is a command line-based software allowing users to manipulate images and create application containers.

As presented in the course, Docker consists of two elements:

* a client, to receive commands from the user
* a server, to execute commands and manage images and containers



### **Docker commands architecture**

Typing this command will give the Client and Server versions available on your computer. ***Paste a screenshot of result***

| ***docker version*** |
| --- |
|  |

Usage, options and a full list of available commands can be accessed through the command line in a terminal.Type the following command

| docker --help |
| --- |

The general usage of a Docker command line is as follows:

| docker [OPTIONS] COMMAND [arg...] |
| --- |

**Questions**

1. **How many arguments are absolutely required by the command ‘docker pull’ ?**

Answer/

According to docker docs there’s two ways to write the command:

This command is for download an image from a registry

Usage: 

Aliases: 

Docker.Docs. <https://docs.docker.com/reference/cli/docker/image/pull/>.

We will focus on the shorthand of this command “docker pull”

and his basic syntax is: **docker pull [OPTIONS] NAME[:TAG|@DIGEST].**

**NAME[:TAG|@DIGEST] (required):**

* **NAME: The name of the image to pull.**
* **TAG: An optional tag to specify a version of the image. If omitted, Docker defaults to latest.**
* **DIGEST: An optional digest for a specific version of the image.**

**Options (optional):**

* **--all-tags, -a: Pull all tags of an image.**
* **--platform: Specify the platform to pull the image for (e.g., linux/amd64, linux/arm64).**

Answering the main question the command primarily takes a single argument which is the name of the image that is needed to pull. Add an optional arguments are helpful to refine the pull operation.

1. **Do you remember what a registry is?**

Answer/

Yes, I do, an image registry is a location that hosts storing and sharing container images. In this case the registry by default is Docker hub, this register can be public or private.

### **Download a predefined image available on the DockerHub**

In a web browser, navigate to the DockerHub :<https://hub.docker.com/>

In the top search bar, type : japeto/pujgcc and paste a screenshot

|  |
| --- |

Our course has images available for the development of practical sessions. **Questions**

1. **How many times was the *japeto* image downloaded ?**

Answer/ in the moment shows that was pull **228 times and 139 downloads.**

Execute the command inside a terminal.

#### **Paste a screenshot of result**

| docker pull japeto/pujgcc:v0.12 |
| --- |
|  |

You will get an error as this image has no default tag (“latest”). So we need to specify one in the command line.

Go to the “Tags” tab and copy the pull command of version latest

#### **Paste a screenshot of result**

| docker pull japeto/pujgcc:v0.12 |
| --- |
|  |

**Question:**

1. How many times do you see ‘Pull complete’ displayed ? Why ?

Answer/ is displayed 11 times, pull complete appears many times, because the images are made by layers, when an image is pulled each layer is pulled separately. The message "Pull complete" indicates that a specific layer has been fully downloaded.

Now, to be sure that the image was correctly pulled, let’s see the list of all available downloaded images inside our workspace.

***Paste a screenshot of result***

| ***docker image*** |
| --- |
|  |

**Question:**

1. What is the size of the japeto/pujgcc image ?

Answer/ Japeto/pujgcc size is 1.39 GB

**Perform a task using a pulled image**

Among the Docker commands, we will now use the ‘run’ command. **Question:**

1. What are the options and parameters of the ‘run’ command ?

Answer/ Japeto

| docker run --help |
| --- |

As displayed in the terminal, the description of the command is ‘Run a command in a new container’.

**Question :**

1. What is the difference between an image and a container ?

Answer

Now, to run the application, execute the following command:

***Paste a screenshot of result***

| ***docker run japeto/pujgcc bash --help*** |
| --- |
|  |

### **Congratulations!**

**You just successfully downloaded and used your first Docker image!**

Running *PUJGCC* without parameters was interesting as a demonstration of Docker’s features. But if we want to really run *PUJGCC*, we also need to provide parameters and, most importantly, input files.

### **Find the paths to bind**

To bind our current folder to the /data/ folder located inside a container, we first need the absolute path of the current folder, obtained through the unix pwd command.

#### **Paste a screenshot of result**

| pwd |
| --- |
|  |

This path will be used in further commands through ${PWD}.

Instead of running ls command to /home/ files, we will now just list the content of the /data/ folder inside the container but bind with the host.

#### **Paste a screenshot of result**

| docker run japeto/pujgcc:latest ls /data |
| --- |
|  |

If nothing appears, it is normal: the folder is empty and only serves as a “*branching point*”.

We now have the paths of the two folders we want to bind together.

### **Bind a local folder into a container**

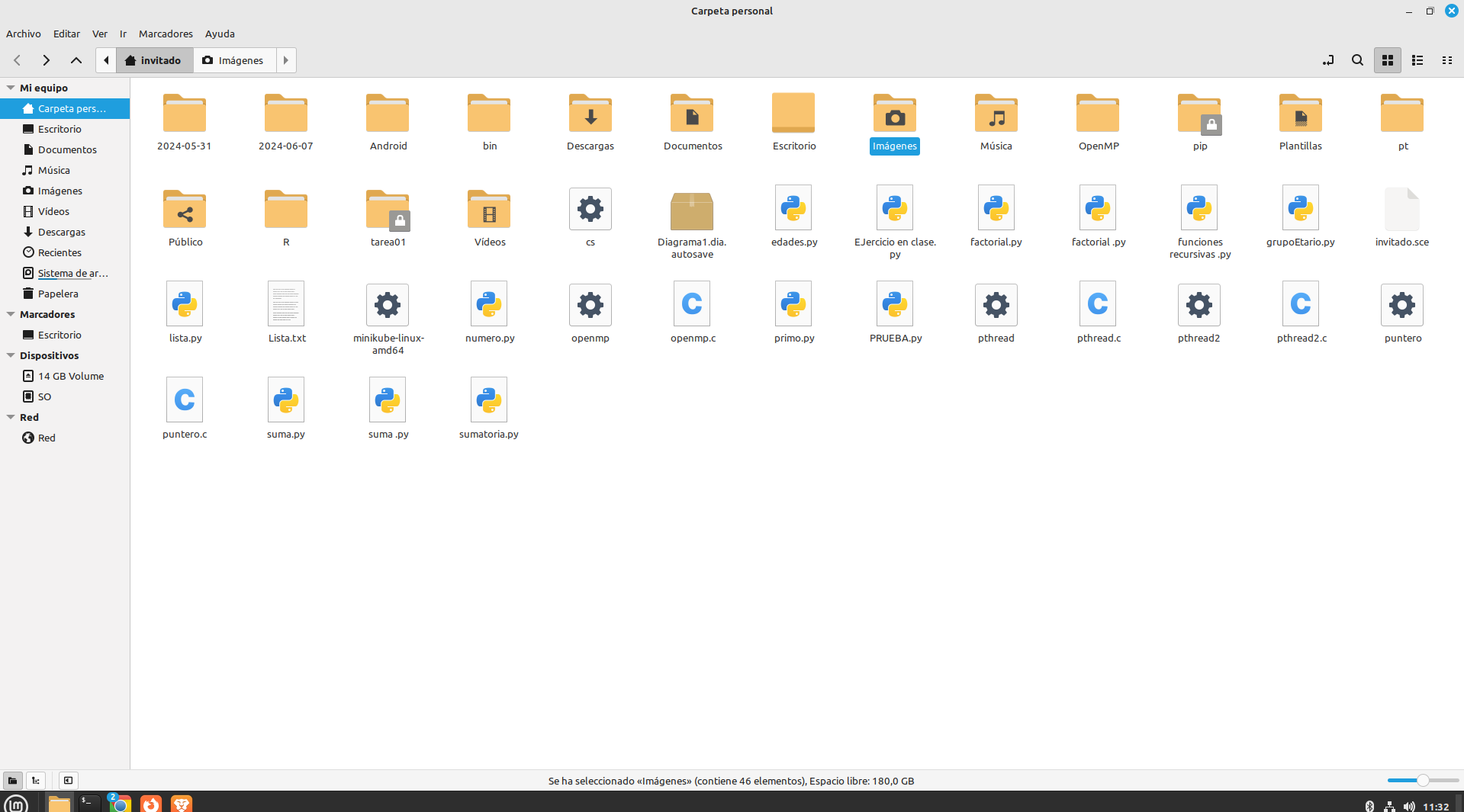
To perform the folder mapping between the current folder and /data inside the image, the syntax is simple. ***Paste a screenshot of result***

| ***docker run -v ${PWD}:/data/ japeto/pujgcc:latest ls /data/*** |
| --- |
|  |

**Question:**

1. **Is the displayed list the same as what is in your current folder?**

Answer/ yes, it is



Finally, we can run C on a C or C++ file located in the Data folder. Change the name of the file to any of the provided files.

***Paste a screenshot of result***

| ***docker run -v ${PWD}:/data/ japeto/pujgcc:latest gcc /data/helloworld.c / our version (docker run -v ${PWD}:/data/ japeto/pujgcc:latest g++ /data/lab.cpp)*** |
| --- |
|  |

**Restart and detach a container**

Learn how to re-use a container where you installed something

Use the start command to restart the container created in the last exercise

**Paste a screenshot of result**

| **docker start -ti mycontainer /bin/bash** |
| --- |
|  |

Go back to the container using the exec command instead of the run command.

#### **Paste a screenshot of result**

| docker exec -ti mycontainer /bin/bash |
| --- |
|  |

#### 

**Question :**

1. **What happens now when you exit the container? Is it stopped?**

Answer

*Check with and* ***Paste a screenshot of result***

| docker ps -l |
| --- |
|  |

In fact, the container keeps on running. This is because re-starting a container turns it into a “detached process” running in the background. Alternatively, we could have added the -d option to the first docker run command, creating directly a detached container.

Finally, you can stop the container.

| docker stop mycontainer |
| --- |
|  |

**This is the end of the practical session. We hope you enjoyed it. Don’t hesitate to ask any questions and feel free to contact us any time after the session!**

**List of commands**

Search the available versions of an image in the Docker registry:

| docker search |
| --- |

Pulling an image:

| docker pull |
| --- |

Starting a container on a given image running a single command:

| docker run -ti |
| --- |

Starting a container on a given image running a single command (detached):

| docker run -d |
| --- |

List all containers and their status

| docker ps -l |
| --- |

List all pulled images

| docker images |
| --- |

Removing one local container

| docker rm |
| --- |

Removing one local image

| docker rmi |
| --- |

Clean all containers

| docker rm $(docker ps -aq) |
| --- |

Clean all images (after cleaning the containers)

| docker rmi $(docker images -aq) |
| --- |

### 

### **Observations**

* Deliveries must be made in teams of 4. Using a public repository on github and a pdf report
* If you do not understand the instructions for any of the activities, do not hesitate to jefferson.amado.pena@correounivalle.edu.co