

**UNIVERSIDAD MAYOR DE SAN ANDRÉS**  
**FACULTAD DE CIENCIAS PURAS Y NATURALES**  
**CARRERA DE INFORMÁTICA**



## **Programación Distr. y Paralela - INF 317**

### **EXAMEN 1**

**ESTUDIANTE:**

**ALDUNATE DE LA BARRA MAURICIO - 9194103 LP**

**REGISTRO UNIVERSITARIO:**

**1778600**

**GESTIÓN II/2024**

**LA PAZ - BOLIVIA**

### 1. Describa de manera teórica los siguientes conceptos: SISD, SIMD, MISD y MIMD. Indique además que lenguajes aplican a estos.

#### Respuesta:

Al hablar de SISD, SIMD, MISD y MIMD es referirnos a la Taxonomía de Flynn, el cual consultando en wikipedia nos indica que es una clasificación de arquitecturas de computadores, propuesta por Michael J. Flynn en 1966 y ampliada en 1972. Este sistema ha perdurado y fue usado como herramienta de diseño de procesadores modernos y sus funcionalidades. Se nos habla también que el contexto de la multiprogramación ha evolucionado como una extensión del sistema de clasificación, todo debido a la aparición de unidades centrales de multiprocesamiento.

**Taxonomía de Flynn**

	Una instrucción	Múltiples instrucciones
Un dato	SISD	MISD
Múltiples datos	SIMD	MIMD

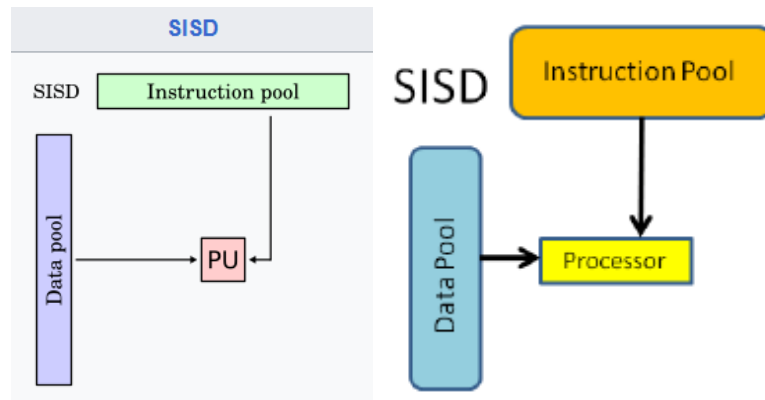
Entrando a las clasificaciones, estas se basan en el número de instrucciones concurrentes (control) y en los flujos de datos disponibles en la arquitectura.

#### 1. SISD (Single Instruction, Single Data o Una instrucción, un dato)

Se refiere a una arquitectura computacional en la que un único procesador ejecuta un solo flujo de instrucciones para operar sobre datos almacenados en una única memoria.

En una configuración de SISD, un solo procesador realiza operaciones secuencialmente en datos de memoria. La secuencia de operaciones es lineal e implica que el procesador reciba una instrucción de memoria, la aplique a un punto de datos y luego se proceda a la siguiente instrucción. La secuencia es lineal, sin que se produzcan tareas simultáneas

Si bien los sistemas SISD proporcionan un procesamiento directo y predecible, no pueden ejecutar múltiples instrucciones simultáneamente, una característica conocida como paralelismo. Esta limitación resulta en velocidades de procesamiento más lentas que modelos más sofisticados, los cuales pueden ejecutar múltiples instrucciones o procesar múltiples puntos de datos simultáneamente.



Lenguajes que aplican SISD:

- C
- C++
- Python
- Java
- Assembly

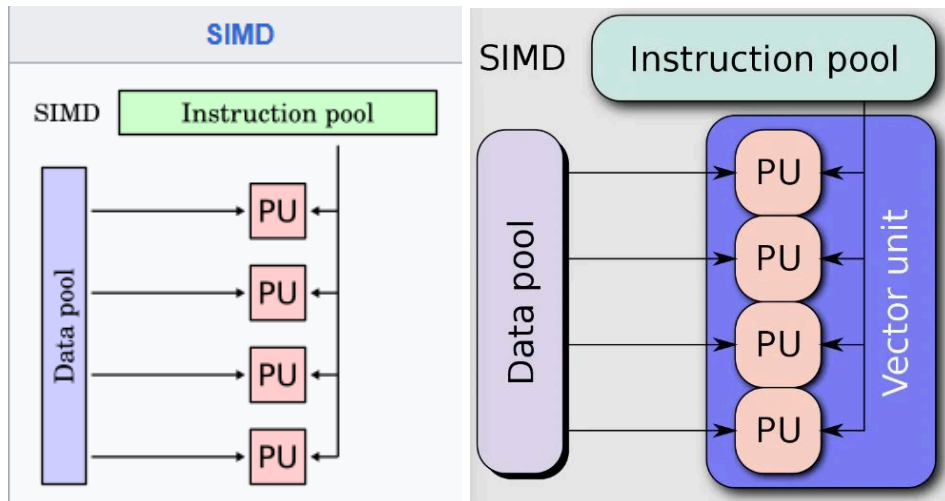
## 2. SIMD (Single Instruction, Multiple Data o Una instrucción, múltiples datos)

Es una técnica de computación que permite operar con múltiples elementos de datos al mismo tiempo. Es una forma de paralelización de software. Es decir, la misma instrucción se aplica a varios elementos de datos en paralelo, lo que permite al procesador trabajar con varios elementos de datos al mismo tiempo, lo que permite mejorar significativamente el rendimiento de los programas aprovechando las capacidades de los procesadores modernos.

Dentro de la arquitectura SIMD, el procesador puede procesar varios elementos de datos en paralelo usando registros vectoriales. Los registros vectoriales se utilizan para almacenar los elementos de datos que se van a procesar en paralelo. Luego, el procesador ejecuta la instrucción en

cada uno de los elementos de datos almacenados en los registros vectoriales. Esto permite al procesador trabajar con varios elementos de datos al mismo tiempo, lo que aumenta el rendimiento del programa.

Algunas aplicaciones donde es útil SIMD es en Gráficos 3D, Procesamiento de señales, Minería de datos, etc.



Lenguajes que aplican SIMD:

- C
- C++
- Python
- Rust
- Java
- .NET (C#)
- Fortran
- Julia

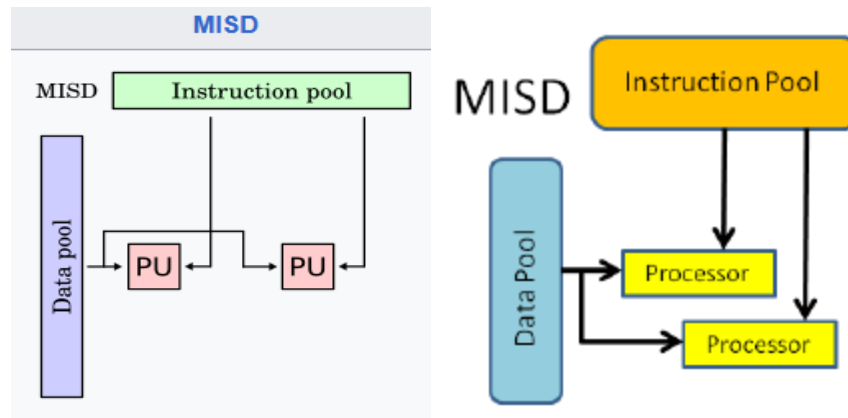
### 3. MISD (Multiple Instruction, Single Data o Múltiples instrucciones, un dato)

Se refiere a que múltiples unidades funcionales ejecutan diferentes instrucciones sobre los mismos datos. Entrando con mayor profundidad, varias unidades de procesamiento realizan distintas operaciones sobre un único flujo de datos. Al no ser una arquitectura comúnmente usada por sus limitaciones y complejidad, esta tiene aplicaciones específicas, como en los sistemas tolerantes a fallos, donde se ejecutan múltiples versiones

de un programa para detectar y corregir errores. Las arquitecturas segmentadas pertenecen a este tipo, aunque en un extremo se podría llegar a decir que los datos son diferentes después de ser procesados por cada etapa en el pipeline, con lo cual no entraría en esta categoría.

En síntesis, MISD se caracteriza por:

- Múltiples Instrucciones
- Un solo dato
- Usado principalmente en aplicaciones especializadas (sistemas donde la tolerancia de fallos es crucial)



Lenguajes que aplican MISD:

- C++
- Ada
- Verilog y VHDL

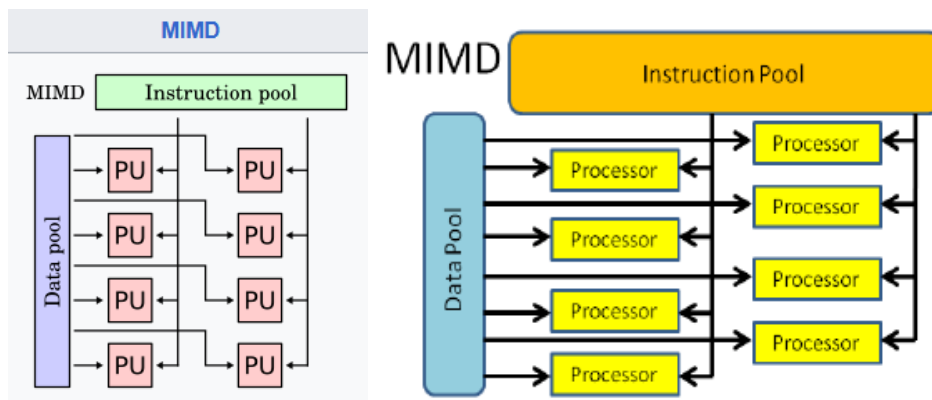
#### 4. MIMD (Multiple Instruction, Multiple Data o Múltiples instrucciones, múltiples datos)

Se refiere a una técnica empleada para lograr el paralelismo. Dicho de otra manera, es una arquitectura de computación paralela en la que múltiples procesadores ejecutan diferentes instrucciones sobre diferentes conjuntos de datos de manera simultánea. Las máquinas que utilizan MIMD tienen un número de procesadores que funcionan de manera asíncrona e independiente. En cualquier momento, cualquier procesador puede ejecutar diferentes instrucciones sobre distintos datos. Los sistemas distribuidos suelen clasificarse como arquitecturas MIMD, bien

sea explotando un único espacio compartido de memoria, o uno distribuido.

Las computadoras MIMD pueden categorizarse por tener memoria compartida o distribuida, clasificación que se basa en como el procesador MIMD accede a la memoria. La memoria compartida de las máquinas puede estar basada en buses, extensiones, o de tipo jerárquico. Las máquinas con memoria distribuida pueden tener esquemas de interconexión en hipercubo o malla.

La arquitectura MIMD puede usarse en varias aplicaciones como ser el diseño asistido, modelado, simulación e interruptores.



Lenguajes que aplican MIMD:

- C/C++ con MPI
- CUDA, cuando se combinan múltiples kernels
- Python con MPI4Py
- Java con Fork/Join Framework
- Podríamos mencionar también tal vez la API "OpenMP"

**2. Realice un programa que tenga los métodos suma, resta, multiplicación, división en lenguaje c. Programe los mismos sin el uso de punteros.**

**Enlace Github:**

<https://github.com/maurice920/INF317-P1-MAURICIO-ALDUNATE-DE-LA-BARRA/tree/main/Ejercicio2>

3. Realice un programa que tenga los métodos suma, resta, multiplicación, división en lenguaje c. Programe los mismo con el uso de punteros.

Enlace Github:

<https://github.com/maurice920/INF317-P1-MAURICIO-ALDUNATE-DE-LA-BARRA/tree/main/Ejercicio3>

4. Realice el cálculo de pi secuencial con el uso de punteros, hágalo iterativamente y recursivamente.

Enlace Github:

<https://github.com/maurice920/INF317-P1-MAURICIO-ALDUNATE-DE-LA-BARRA/tree/main/Ejercicio4>

5. Con multiprocessing calculo pi hasta el término 1 millón, utilice al menos 3 procesadores.

Enlace Github:

<https://github.com/maurice920/INF317-P1-MAURICIO-ALDUNATE-D E-LA-BARRA/tree/main/Ejercicio5>

6. Realice el algoritmo de fibonacci, utilizando solo los términos iniciales en cada vector, conservando la forma de cálculo convencional.

Enlace Github:

<https://github.com/maurice920/INF317-P1-MAURICIO-ALDUNATE-D E-LA-BARRA/tree/main/Ejercicio6>

7. En c# utilizando biblioteca de clases realice la calculadora con expresiones infijas o prefijas.

Enlace Github:

<https://github.com/maurice920/INF317-P1-MAURICIO-ALDUNATE-D E-LA-BARRA/tree/main/Ejercicio7/CalculatorLibrary>

## PROGRAMACIÓN DISTRIB. Y PARALELA

8. En c# utilizando servicios web realice la calculadora con expresiones infijas o prefijas.

Enlace Github:

<https://github.com/maurice920/INF317-P1-MAURICIO-ALDUNATE-D-E-LA-BARRA/blob/main/Ejercicio8.rar>

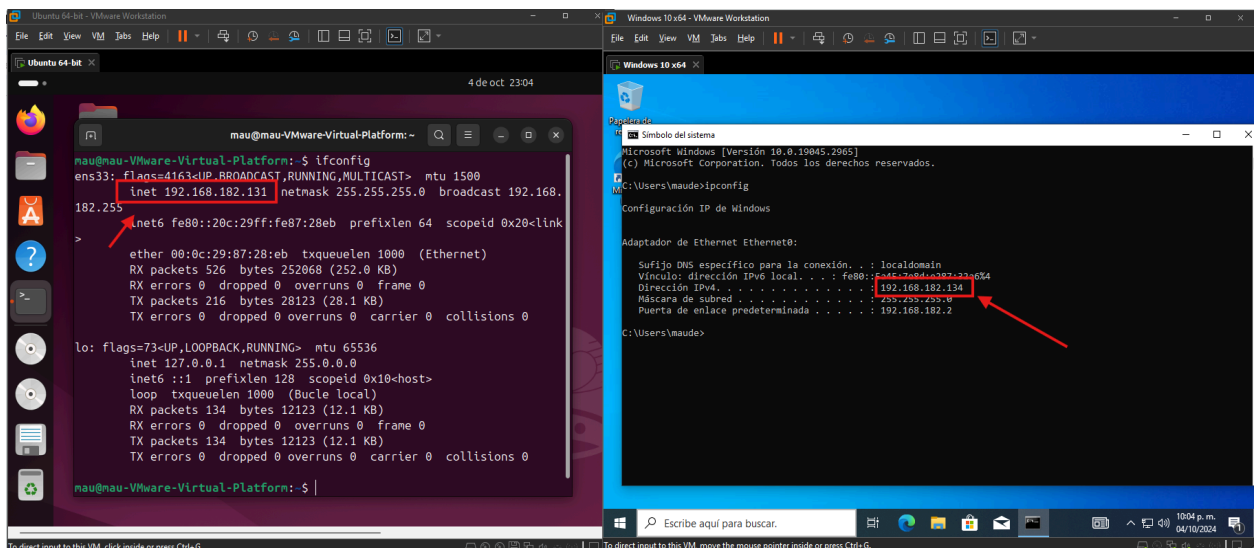
9. En visual c# realice la detección de bordes.

Enlace Github:

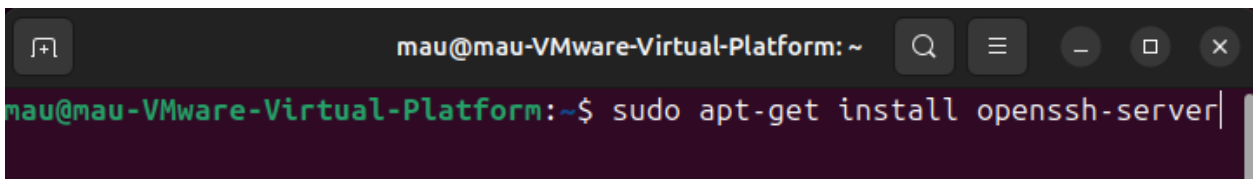
<https://github.com/maurice920/INF317-P1-MAURICIO-ALDUNATE-D-E-LA-BARRA/tree/main/Ejercicio9>

10. Con el uso de sus máquinas virtuales, realice la comunicación ssh entre la máquina Windows y Linux.

Lo primero es verificar que ambas máquinas virtuales estén en la misma red:



Luego debemos instalar SSH en nuestro UBUNTU:



Verificamos que SSH esté activo:



```
mau@mau-VMware-Virtual-Platform: ~  
mau@mau-VMware-Virtual-Platform:~$ sudo systemctl status ssh  
● ssh.service - OpenBSD Secure Shell server  
   Loaded: loaded (/usr/lib/systemd/system/ssh.service; disabled; pr  
   Active: inactive (dead)  
TriggeredBy: ● ssh.socket  
   Docs: man:sshd(8)  
         man:sshd_config(5)  
lines 1-6/6 (END)
```

Por parte de Windows, instalamos el Cliente de OpenSSH y el Servidor de OpenSSH.

### Capturas de la Solución (Windows a Linux):

Ahora solo colocamos el comando SSH, usuario y luego IP de la máquina linux, posterior a eso nos preguntará la contraseña del usuario linux, le ingresamos y listo, estamos conectados a nuestro linux desde windows.

```
Selecciónar símbolo del sistema - ssh mau@192.168.182.131  
Microsoft Windows [Versión 10.0.19045.2965]  
(c) Microsoft Corporation. Todos los derechos reservados.  
  
C:\Users\maude>ipconfig  
  
Configuración IP de Windows  
  
Adaptador de Ethernet Ethernet0:  
  
    Sufijo DNS específico para la conexión. . . : localdomain  
    Vínculo: dirección IPv6 local. . . : fe80::5a45:7e8d:e287:32a6%4  
    Dirección IPv4. . . . . : 192.168.182.134  
    Máscara de subred . . . . . : 255.255.255.0  
    Puerta de enlace predeterminada . . . . . : 192.168.182.2  
  
C:\Users\maude>ssh mau@192.168.182.131  
ssh: connect to host 192.168.182.131 port 22: Connection refused  
  
C:\Users\maude>ssh mau@192.168.182.131  
The authenticity of host '192.168.182.131 (192.168.182.131)' can't be established.  
ECDSA key fingerprint is SHA256:IMJ634E092MTj5jHM8HbxMXoP1puNtUz56y7MAbsdZQ.  
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes  
Warning: Permanently added '192.168.182.131' (ECDSA) to the list of known hosts.  
mau@192.168.182.131's password:
```

# PROGRAMACIÓN DISTRIB. Y PARALELA

```
C:\Users\maude>ssh mau@192.168.182.131
mau@192.168.182.131's password:
Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 6.8.0-45-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

El mantenimiento de seguridad expandido para Applications está desactivado

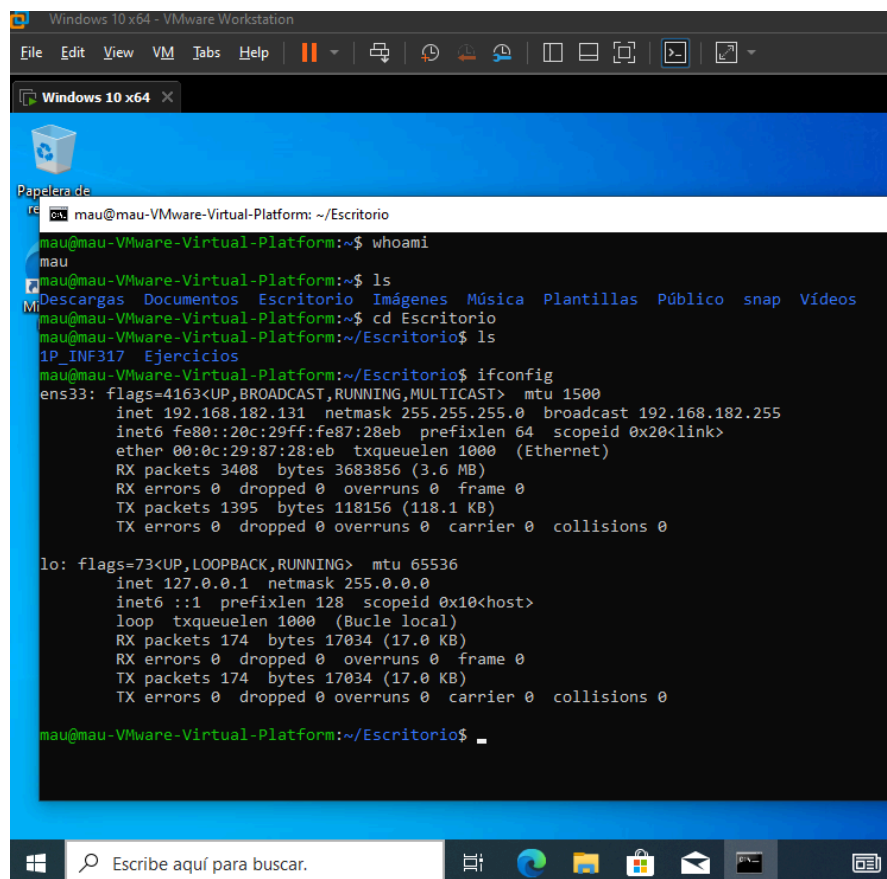
Se pueden aplicar 32 actualizaciones de forma inmediata.
4 de estas son actualizaciones de seguridad estándares.
Para ver estas actualizaciones adicionales, ejecute: apt list --upgradable

Active ESM Apps para recibir futuras actualizaciones de seguridad adicionales.
Vea https://ubuntu.com/esm o ejecute «sudo pro status»

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

mau@mau-VMware-Virtual-Platform:~$
```



```
Windows 10 x64 - VMware Workstation
File Edit View VM Tabs Help
Windows 10 x64 x
Papelera de reciclaje
mau@mau-VMware-Virtual-Platform: ~/Escritorio
mau@mau-VMware-Virtual-Platform:~$ whoami
mau
mau@mau-VMware-Virtual-Platform:~$ ls
Descargas Documentos Escritorio Imágenes Música Plantillas Público snap Vídeos
mau@mau-VMware-Virtual-Platform:~$ cd Escritorio
mau@mau-VMware-Virtual-Platform:~/Escritorio$ ls
1P_INF317 Ejercicios
mau@mau-VMware-Virtual-Platform:~/Escritorio$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.182.131 netmask 255.255.255.0 broadcast 192.168.182.255
    inet6 fe80::20c:29ff:fe87:28eb prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:87:28:eb txqueuelen 1000 (Ethernet)
    RX packets 3408 bytes 3683856 (3.6 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1395 bytes 118156 (118.1 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

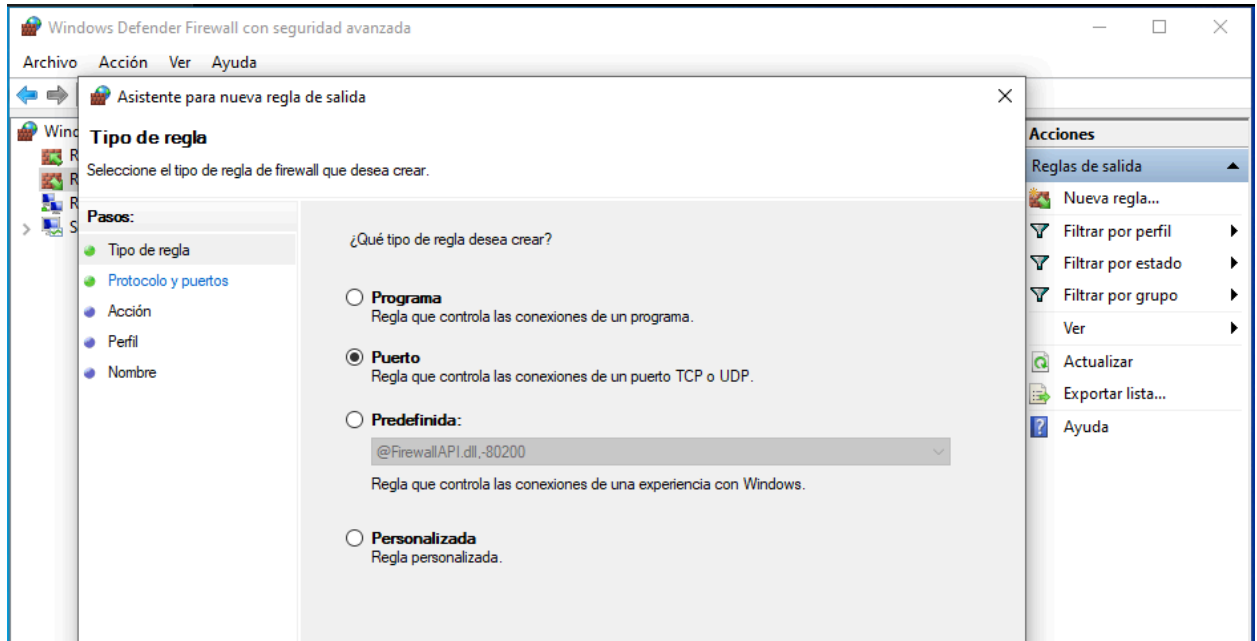
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Bucle local)
    RX packets 174 bytes 17034 (17.0 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 174 bytes 17034 (17.0 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

mau@mau-VMware-Virtual-Platform:~/Escritorio$
```

## Capturas de la Solución (Linux a Windows):


Por este lado hacemos lo siguiente:

Agregamos una nueva regla en el windows virtualizado



Habilitamos el puerto 22 para conexión SSH:

# PROGRAMACIÓN DISTRIB. Y PARALELA

 Asistente para nueva regla de salida

**Protocolo y puertos**

Especifique los puertos y protocolos a los que se aplica esta regla.

**Pasos:**

- Tipo de regla
- Protocolo y puertos**
- Acción
- Perfil
- Nombre


¿Se aplica esta regla a TCP o UDP?

☒ TCP  
☐ UDP

¿Se aplica esta regla a todos los puertos remotos o a unos puertos remotos específicos?

☐ Todos los puertos remotos  
☒ Puertos remotos específicos:

Ejemplo: 80, 443, 5000-5010

 Asistente para nueva regla de salida

**Acción**

Especifique la acción que debe llevarse a cabo cuando una conexión coincide con las condiciones especificadas en la regla.

**Pasos:**

- Tipo de regla
- Protocolo y puertos
- Acción**
- Perfil
- Nombre


¿Qué medida debe tomarse si una conexión coincide con las condiciones especificadas?

☒ **Permitir la conexión**  
Esto incluye las conexiones protegidas mediante IPsec y las que no lo están.

☐ **Permitir la conexión si es segura**  
Esto incluye solamente las conexiones autenticadas mediante IPsec. Éstas se protegerán mediante la configuración de reglas y propiedades de IPsec del nodo Regla de seguridad de conexión.

☐ **Bloquear la conexión**

## PROGRAMACIÓN DISTRIB. Y PARALELA

 Asistente para nueva regla de salida



### Perfil

Especifique los perfiles en los que se va a aplicar esta regla.

#### Pasos:

- Tipo de regla
- Protocolo y puertos
- Acción
- Perfil**
- Nombre

¿Cuándo se aplica esta regla?

☒ **Dominio**


Se aplica cuando un equipo está conectado a su dominio corporativo.

☒ **Privado**

Se aplica cuando un equipo está conectado a una ubicación de red privada, como una red doméstica o del lugar de trabajo.

☒ **Público**

Se aplica cuando un equipo está conectado a una ubicación de redes públicas.

 Asistente para nueva regla de salida



### Nombre

Especifique el nombre y la descripción de esta regla.

#### Pasos:

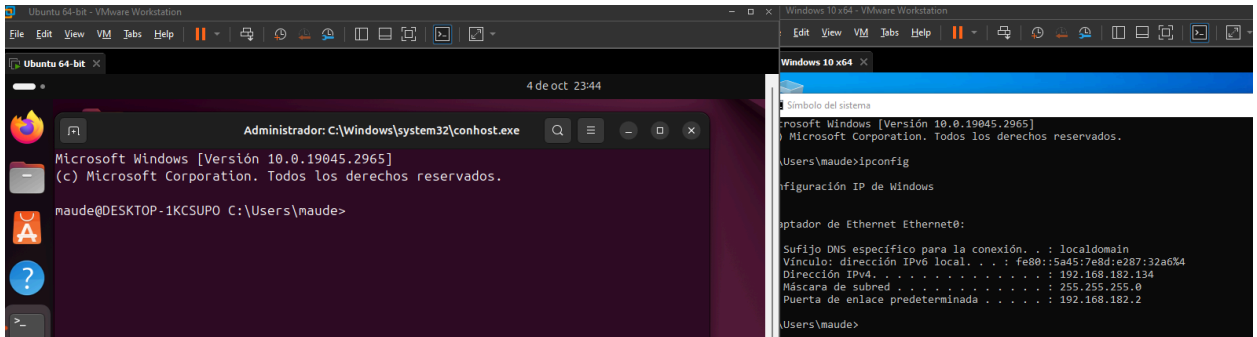
- Tipo de regla
- Protocolo y puertos
- Acción
- Perfil
- Nombre**

Nombre:

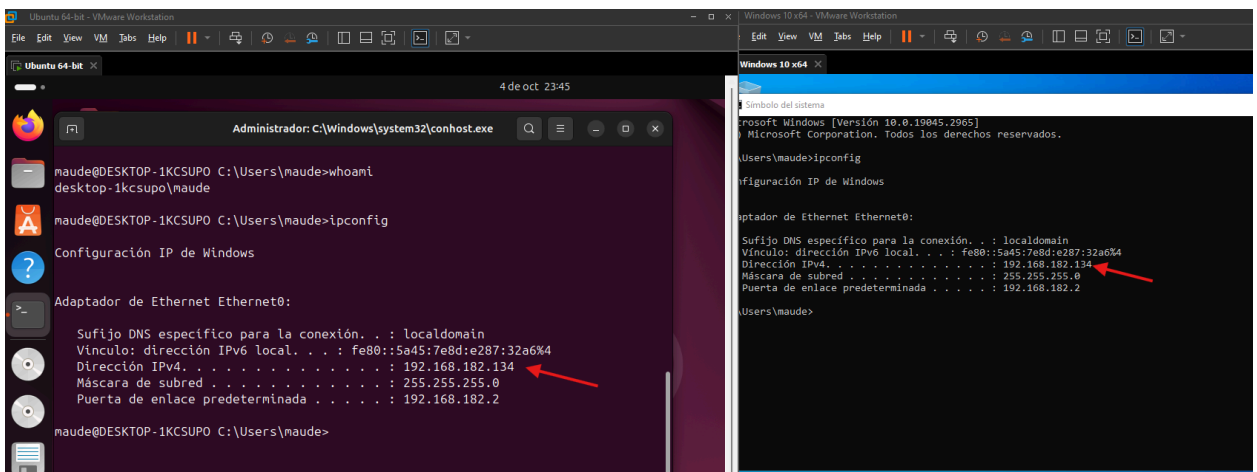
Descripción (opcional):

Luego volvemos al UBUNTU y colocamos el SSH, el nombre de usuario windows, la ip y la contraseña:

# PROGRAMACIÓN DISTRIB. Y PARALELA



Con eso ya nos conectamos por medio de UBUNTU a nuestra máquina WINDOWS:



11. Con MPI sume dos vectores, siendo el procesador 0 el maestro, el procesador 1 suma posiciones impares y el 2 posiciones suma pares.

Enlace Github:

<https://github.com/maurice920/INF317-P1-MAURICIO-ALDUNATE-D-E-LA-BARRA/tree/main/Ejercicio11>

12. Con el uso de OPENMP utilizando regiones con variables shared y private, realice el despliegue de la serie Fibonacci.

Enlace Github:

<https://github.com/maurice920/INF317-P1-MAURICIO-ALDUNATE-D-E-LA-BARRA/tree/main/Ejercicio12>

13. Utilizando MPI realice el despliegue de un vector de datos tipo cadena, el procesador 0 será el distribuidor, el 1 desplegará posiciones pares y el 2 posiciones impares.

Enlace Github:

<https://github.com/maurice920/INF317-P1-MAURICIO-ALDUNATE-D-E-LA-BARRA/tree/main/Ejercicio13>

14. Con MPI utilizando MPI\_Send y MPI\_Recv multiplique dos matrices.

Enlace Github:

<https://github.com/maurice920/INF317-P1-MAURICIO-ALDUNATE-D-E-LA-BARRA/tree/main/Ejercicio14>