

Nombre: Luis Alberto Vargas González.

Fecha: 22/02/2025

Actividad y unidad: U7 A1 Envío de archivos

Clase: Sistemas Distribuidos.

Maestría en Ciberseguridad.

Descripción de programas.

En esta práctica se hace la programación nuevamente de 2 nodos usando el sistema peer to peer para la comunicación de ambos nodos que adicionalmente fueron modificados ambos programas para que cada uno sea un emisor de archivos (servidor) y receptor de archivos(cliente) por lo que los códigos de ambos varían ligeramente, adicional a esto, se programa un tercer código que nos permitirá poder generar un archivo lo suficientemente grande en MB para cumplir con los requisitos de la tarea.

Nota: Se intentó usar el cifrado AES para los programas, sin embargo, fue complicado su manejo para el cifrado y descifrado en movimiento por lo que generó muchos problemas su uso, entre ellos: los nodos no eran capaces de mantener la conexión abierta durante más de 10 segundos por lo que era imposible poder enviar los archivos completos ya que el nodo 2 cerraba la conexión con el 1 debido a la configuración del padding, los datos además llegaban corrompidos. Es por ello por lo que me vi obligado a usar el cifrado César de nuevo.

Código Nodo 1.

```
import socket
import threading
import os
import time

# Configuración de nodos (debes cambiar los puertos en cada nodo)
NODOS = [
    ("127.0.0.1", 56432), # Nodo 1
    ("127.0.0.1", 56433), # Nodo 2
```

```

]

# Tamaño del paquete (bytes)
TAMANO_PAQUETE = 1024

# Función para cifrar el texto
def cifrar (texto, desplazamiento):
    resultado = []
    for char in texto:
        if char.isalpha():
            # Desplazar solo las letras
            ascii_offset = 65 if char.isupper() else 97
            resultado.append(chr((ord(char) - ascii_offset + desplazamiento) % 26 +
            ascii_offset))
        else:
            resultado.append(char) # No cambiar caracteres no alfabéticos
    return ''.join(resultado)

# Función para descifrar el texto
def descifrar(texto, desplazamiento):
    return cifrar(texto, -desplazamiento)

# Función para enviar un archivo en fragmentos (con cifrado)
def enviar_archivo(mi_ip, mi_puerto, ruta_archivo, desplazamiento_cesar=3):
    if not os.path.exists(ruta_archivo):
        print("El archivo no existe.")
    return

nombre_archivo = os.path.basename(ruta_archivo)
tamano_archivo = os.path.getsize(ruta_archivo)

```

```
print(f"Tamaño del archivo a enviar: {tamano_archivo} bytes")

for ip, puerto in NODOS:
    if (ip, puerto) == (mi_ip, mi_puerto):
        continue # No enviarse a sí mismo

    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
            client_socket.connect((ip, puerto))
            print(f"Conectado a {ip}:{puerto}, enviando archivo...")

            # Enviar metadatos del archivo (nombre y tamaño) como una línea separada
            metadatos = f"{nombre_archivo}|{tamano_archivo}\n"
            client_socket.sendall(metadatos.encode())

            # Leer y cifrar el archivo en fragmentos antes de enviarlo
            with open(ruta_archivo, "rb") as archivo:
                while fragmento := archivo.read(TAMANO_PAQUETE):
                    # Cifrar cada fragmento (convertir bytes a texto cifrado)
                    fragmento_texto = fragmento.decode(errors='ignore') # Convertir a texto
                    fragmento_cifrado = cifrar_cesar(fragmento_texto, desplazamiento_cesar)
                    client_socket.sendall(fragmento_cifrado.encode())

            print(f"Archivo {nombre_archivo} enviado con éxito a {ip}:{puerto}")

    except ConnectionRefusedError:
        print(f"No se pudo conectar a {ip}:{puerto}. Reintentando en 2s...")
        time.sleep(2)
```

```
# Servidor TCP para recibir archivos (con descifrado)

def iniciar_servidor_tcp(mi_ip, mi_puerto, carpeta_destino,
desplazamiento_cesar=3):

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind((mi_ip, mi_puerto))
server_socket.listen()

print(f"Servidor TCP escuchando en {mi_ip}:{mi_puerto}")

while True:

conn, addr = server_socket.accept()

print(f"Conexión establecida con {addr}")

try:

# Primero, recibir los metadatos del archivo

metadata = conn.recv(1024).decode().strip()

if "|" not in metadata:

raise ValueError(f"Formato de metadatos incorrecto: {metadata}")

nombre_archivo, tamaño_archivo = metadata.split("|", 1)

tamaño_archivo = int(tamaño_archivo.strip())

print(f"Tamaño del archivo recibido: {tamaño_archivo} bytes")

# Guardar el archivo en la carpeta de destino

ruta_completa = os.path.join(carpeta_destino, nombre_archivo)

with open(ruta_completa, "wb") as archivo:

bytes_recibidos = 0

while bytes_recibidos < tamaño_archivo:

fragmento = conn.recv(TAMANO_PAQUETE)
```

```
if not fragmento:
    break

# Descifrar el fragmento antes de escribirlo en el archivo
fragmento_texto = fragmento.decode(errors='ignore')
fragmento_descifrado = descifrar_cesar(fragmento_texto,
desplazamiento_cesar)
archivo.write(fragmento_descifrado.encode())
bytes_recibidos += len(fragmento)

print(f"Archivo {nombre_archivo} recibido y guardado en {ruta_completa}")

except Exception as e:
    print(f"Error al recibir archivo: {e}")

# Configuración y ejecución de los nodos
if __name__ == "__main__":
    # Define si este nodo es el emisor o receptor
    mi_ip = "127.0.0.1"
    mi_puerto = 56432 # Cambia este puerto a 56433 en el otro nodo

    # Carpeta donde se guardarán los archivos recibidos
    carpeta_destino = "./archivos_recibidos"
    os.makedirs(carpeta_destino, exist_ok=True)

    # Iniciar el servidor en un hilo
    servidor = threading.Thread(target=iniciar_servidor_tcp, args=(mi_ip,
mi_puerto, carpeta_destino))
    servidor.start()

    # Simular envío de archivo desde un nodo
```

```
time.sleep(3) # Esperar a que el servidor se inicie
if mi_puerto == 56432: # Solo el nodo 1 enviará un archivo de prueba
    enviar_archivo(mi_ip, mi_puerto, "archivo_prueba.txt")
```

En este programa se generan las funciones que permitirán poder enviar archivos desde el emisor al cliente receptor el cuales el nodo 2, por lo que aquí la función más importante es la de enviar_archivos la cual envía en fragmentos (paquetes) de 10245 bytes la información, y la de iniciar servidor, por lo que dichas funciones son vitales para el correcto funcionamiento del emisor.

Código Nodo 2.

```
import socket
import threading
import os
import time
```

```
# Configuración de nodos (debes cambiar los puertos en cada nodo)
NODOS = [
    ("127.0.0.1", 56432), # Nodo 1
    ("127.0.0.1", 56433), # Nodo 2
]

# Tamaño del paquete (bytes)
TAMANO_PAQUETE = 1024

# Función para cifrar
def cifrar(texto, desplazamiento):
    resultado = []
    for char in texto:
        if char.isalpha():
            # Desplazar solo las letras
            ascii_offset = 65 if char.isupper() else 97
            resultado.append(chr((ord(char) - ascii_offset + desplazamiento) % 26 +
                                ascii_offset))
        else:
            resultado.append(char) # No cambiar caracteres no alfabéticos
    return ''.join(resultado)

# Función para descifrar el texto
def descifrar(texto, desplazamiento):
    return cifrar(texto, -desplazamiento)

# Función para enviar un archivo en fragmentos (con cifrado)
def enviar_archivo(mi_ip, mi_puerto, ruta_archivo, desplazamiento_cesar=3):
    if not os.path.exists(ruta_archivo):
```



```
print("El archivo no existe.")
return

nombre_archivo = os.path.basename(ruta_archivo)
tamano_archivo = os.path.getsize(ruta_archivo)

print(f"Tamaño del archivo a enviar: {tamano_archivo} bytes")

for ip, puerto in NODOS:
    if (ip, puerto) == (mi_ip, mi_puerto):
        continue # No enviarse a sí mismo

    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
            client_socket.connect((ip, puerto))
            print(f"Conectado a {ip}:{puerto}, enviando archivo...")

            # Enviar metadatos del archivo (nombre y tamaño) como una línea separada
            metadatos = f"{nombre_archivo}|{tamano_archivo}\n"
            client_socket.sendall(metadatos.encode())

            # Leer y cifrar el archivo en fragmentos antes de enviarlo
            with open(ruta_archivo, "rb") as archivo:
                while fragmento := archivo.read(TAMANO_PAQUETE):
                    # Cifrar cada fragmento (convertir bytes a texto cifrado)
                    fragmento_texto = fragmento.decode(errors='ignore') # Convertir a texto
                    fragmento_cifrado = cifrar_cesar(fragmento_texto, desplazamiento_cesar)
                    client_socket.sendall(fragmento_cifrado.encode())

            print(f"Archivo {nombre_archivo} enviado con éxito a {ip}:{puerto}")
```

```
except ConnectionRefusedError:
    print(f"No se pudo conectar a {ip}:{puerto}. Reintentando en 2s...")
    time.sleep(2)

# Servidor TCP para recibir archivos (con descifrado)
def iniciar_servidor_tcp(mi_ip, mi_puerto, carpeta_destino,
    desplazamiento_cesar=3):
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server_socket.bind((mi_ip, mi_puerto))
    server_socket.listen()
    print(f"Servidor TCP escuchando en {mi_ip}:{mi_puerto}")

    while True:
        conn, addr = server_socket.accept()
        print(f"Conexión establecida con {addr}")

        try:
            # Primero, recibir los metadatos del archivo
            metadata = conn.recv(1024).decode().strip()
            if "|" not in metadata:
                raise ValueError(f"Formato de metadatos incorrecto: {metadata}")

            nombre_archivo, tamaño_archivo = metadata.split("|", 1)
            tamaño_archivo = int(tamaño_archivo.strip())

            print(f"Tamaño del archivo recibido: {tamaño_archivo} bytes")

            # Guardar el archivo en la carpeta de destino
```

```

ruta_completa = os.path.join(carpeta_destino, nombre_archivo)
with open(ruta_completa, "wb") as archivo:
    bytes_recibidos = 0
    while bytes_recibidos < tamaño_archivo:
        fragmento = conn.recv(TAMANO_PAQUETE)
        if not fragmento:
            break
        # Descifrar el fragmento antes de escribirlo en el archivo
        fragmento_texto = fragmento.decode(errors='ignore')
        fragmento_descifrado = descifrar_cesar(fragmento_texto,
        desplazamiento_cesar)
        archivo.write(fragmento_descifrado.encode())
        bytes_recibidos += len(fragmento)

    print(f"Archivo {nombre_archivo} recibido y guardado en {ruta_completa}")

except Exception as e:
    print(f"Error al recibir archivo: {e}")

# Configuración y ejecución de los nodos
if __name__ == "__main__":
    # Define si este nodo es el emisor o receptor
    mi_ip = "127.0.0.1"
    mi_puerto = 56433 # Cambia este puerto a 56433 en el otro nodo

    # Carpeta donde se guardarán los archivos recibidos
    carpeta_destino = "./archivos_recibidos"
    os.makedirs(carpeta_destino, exist_ok=True)

    # Iniciar el servidor en un hilo

```

```
servidor = threading.Thread(target=iniciar_servidor_tcp, args= (mi_ip,
mi_puerto, carpeta_destino))
servidor.start()

# Simular envío de archivo desde un nodo
time.sleep(3) # Esperar a que el servidor se inicie
if mi_puerto == 56432: # Solo el nodo 1 enviará un archivo de prueba
    enviar_archivo(mi_ip, mi_puerto, "archivo_prueba.txt")
```

Como podemos ver el código es prácticamente igual para ambos nodos a excepción de del uso que se le dé en la ejecución del método main en el cual definimos el puerto que será usado por cada nodo, esto es clave ya que dependiendo del nodo a ser ejecutado se usan unas u otras funciones, es decir; en el nodo 1 se usarán las funciones de enviar archivo y la de iniciar servidor, y en este programa se usará las funciones de iniciar_servidor_tcp que es la cual nos brinda la función de recibir todos los paquetes. Adicionalmente es importante denotar que la función de cifrado es usada por el nodo 1 y la de descifrado por el nodo 1.

Código Generar.py

```
import random
import string

# Función para generar una cadena aleatoria de una longitud específica
def generar_texto_aleatorio(longitud):
    return ''.join(random.choices(string.ascii_letters + string.digits +
string.punctuation + ' ', k=longitud))

# Generar un archivo de texto de 50 MB o más
tamano_objetivo = 50 * 1024 * 1024 # 50 MB en bytes
tamano_actual = 0

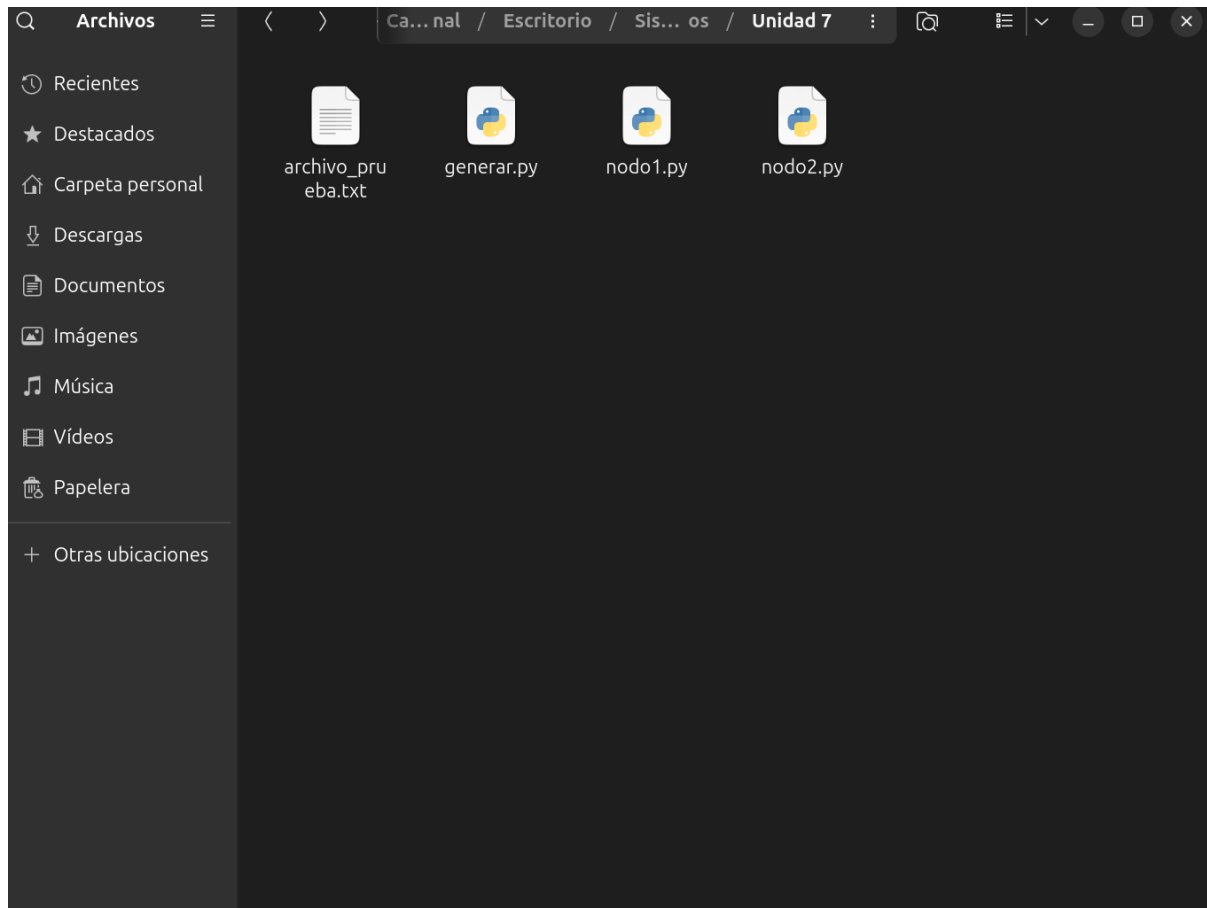
ruta_archivo = "archivo_prueba.txt"

# Abrir el archivo para escribir
with open(ruta_archivo, "w") as archivo:
    while tamano_actual < tamano_objetivo:
        # Generar un bloque de texto aleatorio de tamaño 1 MB
        bloque_aleatorio = generar_texto_aleatorio(1024 * 1024) # 1 MB de texto
        archivo.write(bloque_aleatorio)
        tamano_actual += len(bloque_aleatorio)
```

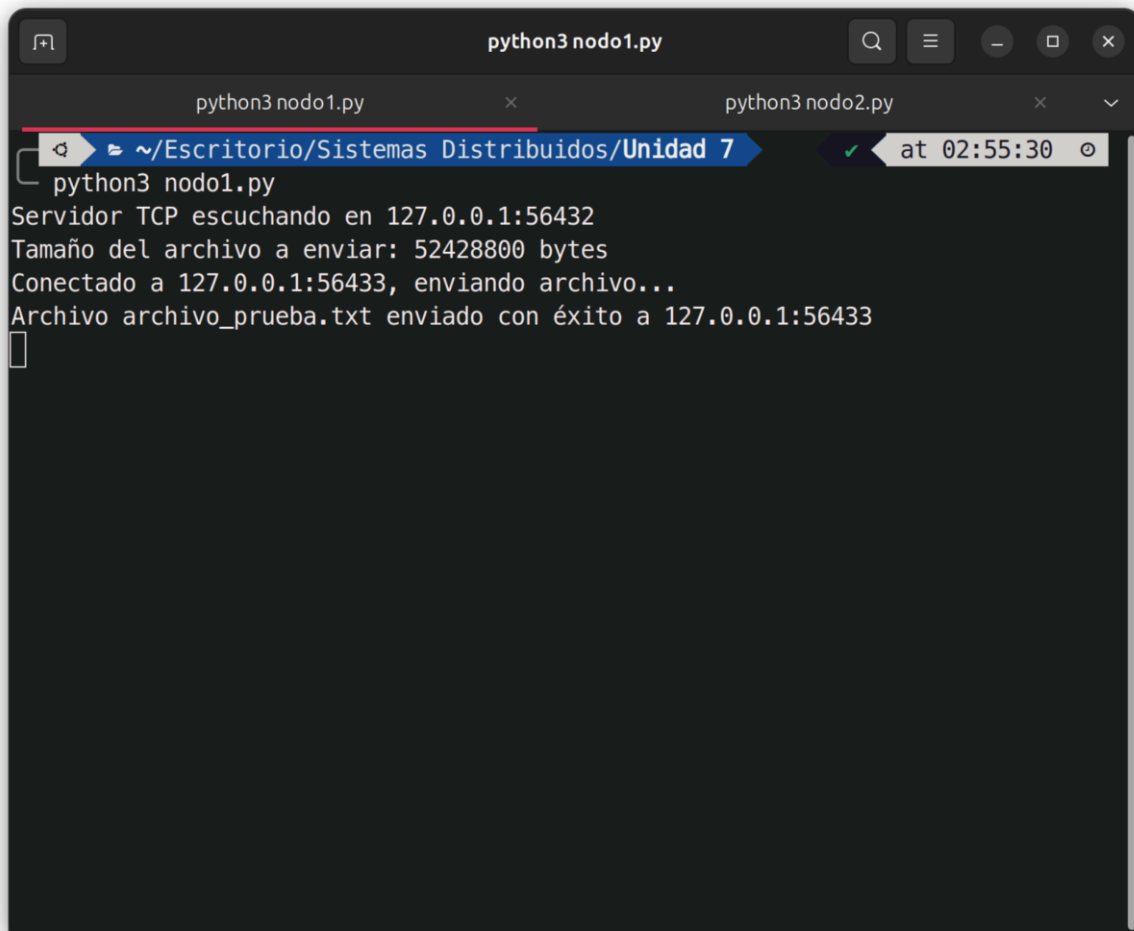
```
print(f"Archivo {ruta_archivo} generado con éxito. Tamaño: {tamano_actual} bytes")
```

En este código se emplea una semilla aleatoria que nos permitirá generar texto para crear archivos específicos, los cuales son los que serán enviados al receptor para que se envíen archivos de mínimo 50 MB.

Evidencia de funcionamiento.



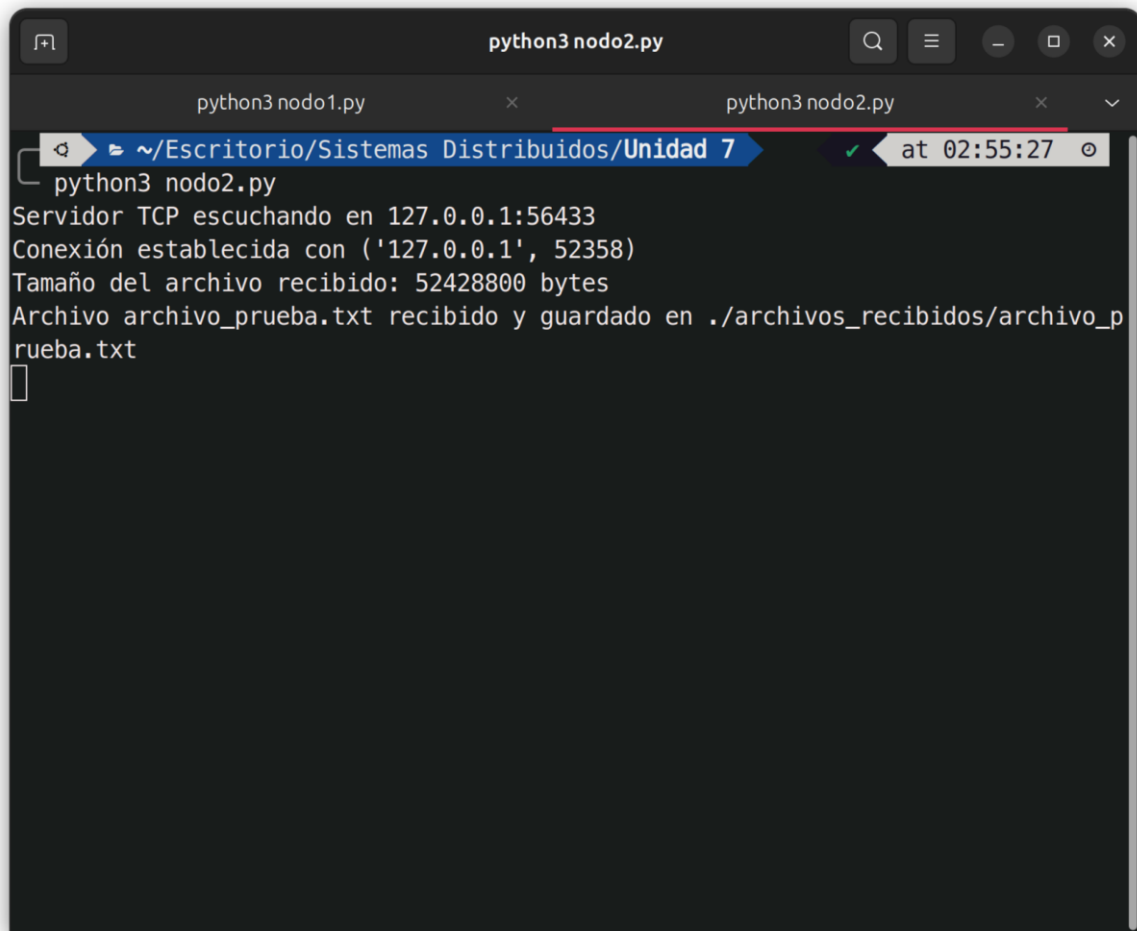
Aquí podemos ver que el archivo que será enviado se llama `archivo_prueba.txt`. Este archivo será enviado a una carpeta donde podremos distinguir el archivo enviado y el original el cual es el que se visualiza actualmente.



A terminal window titled "python3 nodo1.py" with a dark background. The window has a standard macOS title bar with a search icon, a menu icon, and window control buttons. Below the title bar, there are two tabs: "python3 nodo1.py" (active) and "python3 nodo2.py". The active tab shows the following output:

```
python3 nodo1.py
Servidor TCP escuchando en 127.0.0.1:56432
Tamaño del archivo a enviar: 52428800 bytes
Conectado a 127.0.0.1:56433, enviando archivo...
Archivo archivo_prueba.txt enviado con éxito a 127.0.0.1:56433
```

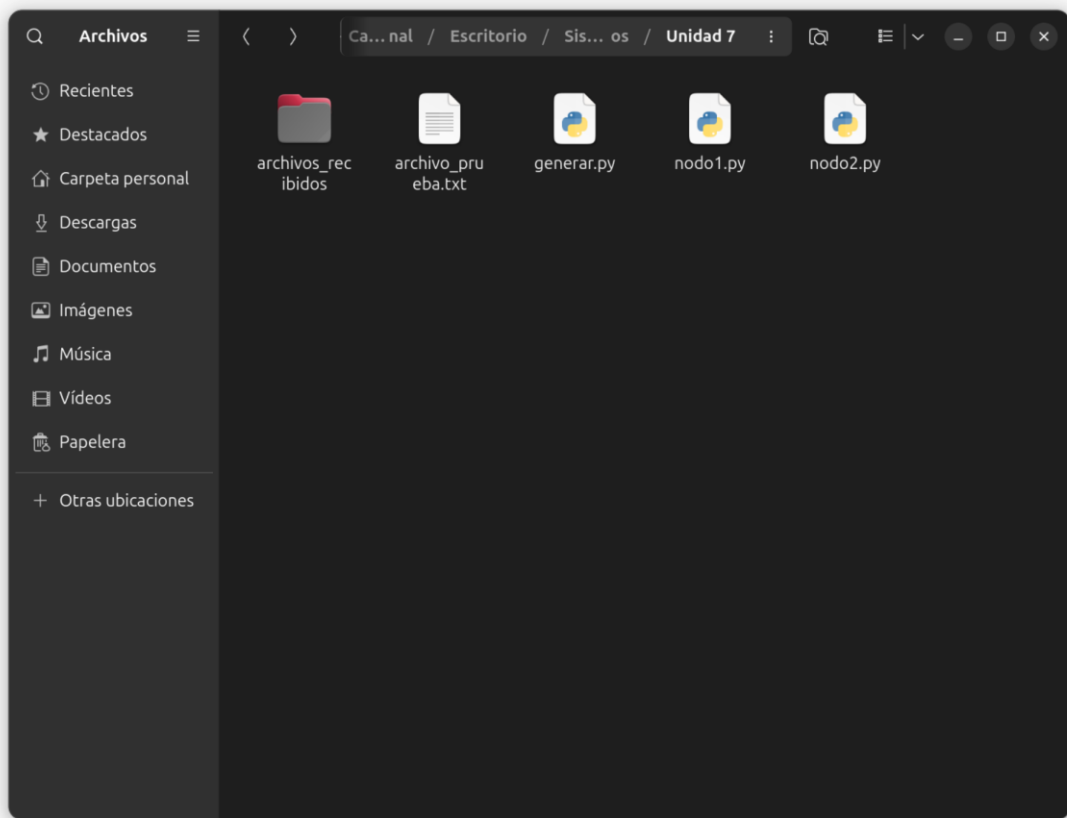
The path bar shows the current directory as `~/Escritorio/Sistemas Distribuidos/Unidad 7` with a green checkmark icon. The clock in the top right corner indicates the time is 02:55:30.

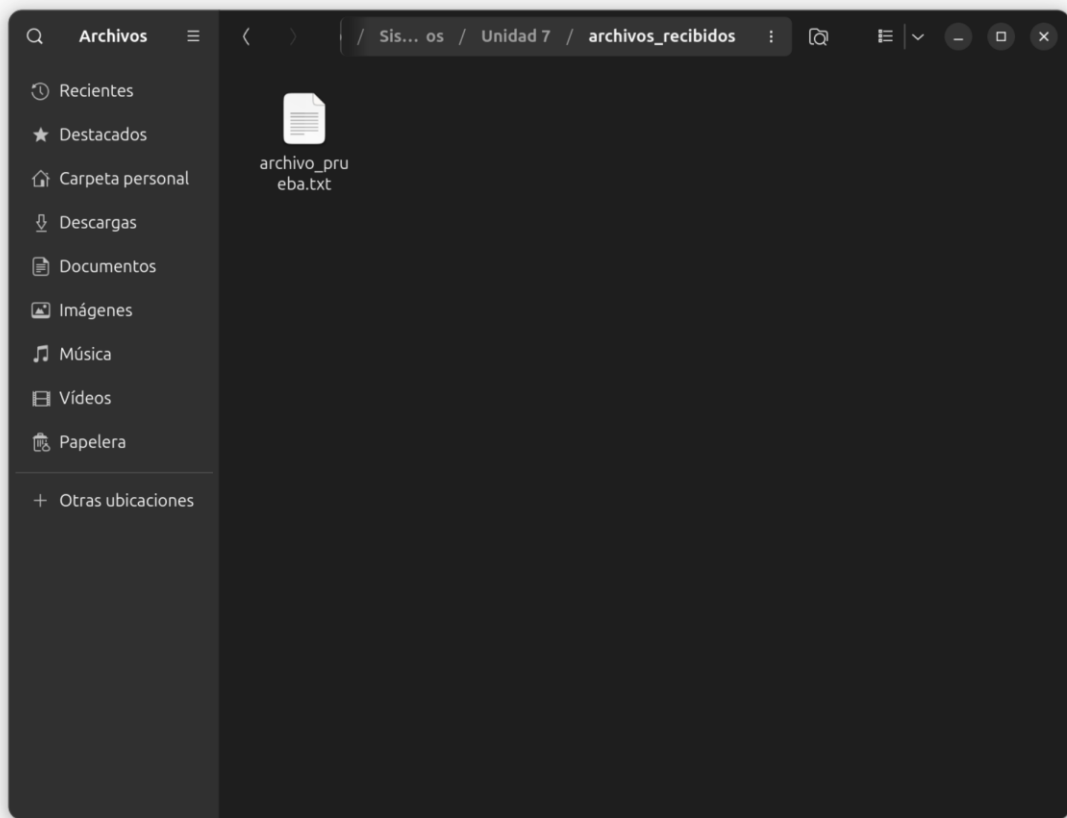


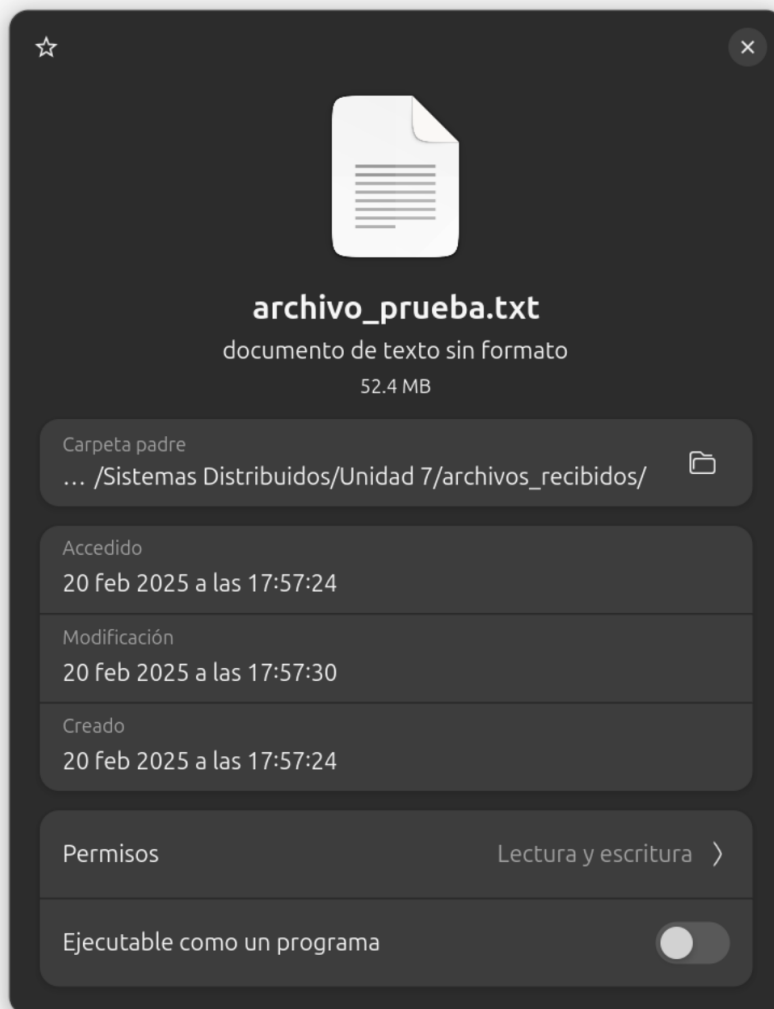
The image shows a terminal window titled "python3 nodo2.py". The window has two tabs: "python3 nodo1.py" and "python3 nodo2.py". The active tab is "python3 nodo2.py". The terminal output shows the following text:

```
python3 nodo2.py
Servidor TCP escuchando en 127.0.0.1:56433
Conexión establecida con ('127.0.0.1', 52358)
Tamaño del archivo recibido: 52428800 bytes
Archivo archivo_prueba.txt recibido y guardado en ./archivos_recibidos/archivo_p
rueba.txt
```

Aquí podemos ver que tanto para el nodo 1 y nodo 2, se nos presentan los datos del archivo, se presenta el tamaño, y la ubicación del mismo una vez fue gestionado por el nodo 2.







Aquí podemos ver que el archivo fue recibido con éxito y además podemos ver que efectivamente se creó una carpeta donde se almacenan los archivos enviados y que fueron recibidos por el nodo 2, adicional a esto podemos ver el archivo pesa un poco más de 52.4 MB cumpliendo así con la importante tarea de poder enviar archivos de tamaño considerable.

Conclusiones.

Podemos argumentar que se cumplió con éxito la práctica, además de que se utilizó un método adicional de generación de texto aleatorio el cual nos permite poder brindar texto suficiente para poder crear archivos lo suficientemente grandes, fue un reto total el poder implementar la función de enviar archivos fragmentados en porciones tan pequeñas, por lo que fue necesario leer documentación de Python y de foros de ayuda. Esto además del problema antes mencionado del uso del cifrado AES