Logotipo

Descripción generada automáticamenteDibujo en blanco y negro

Descripción generada automáticamente con confianza baja

Instituto Politécnico Nacional

Escuela Superior de Cómputo

Olay Silis Jose Eduardo

4CV12

Prof. Carlos Pineda Guerrero

Tarea 2: Transferencia de archivos utilizando datagramas confiables

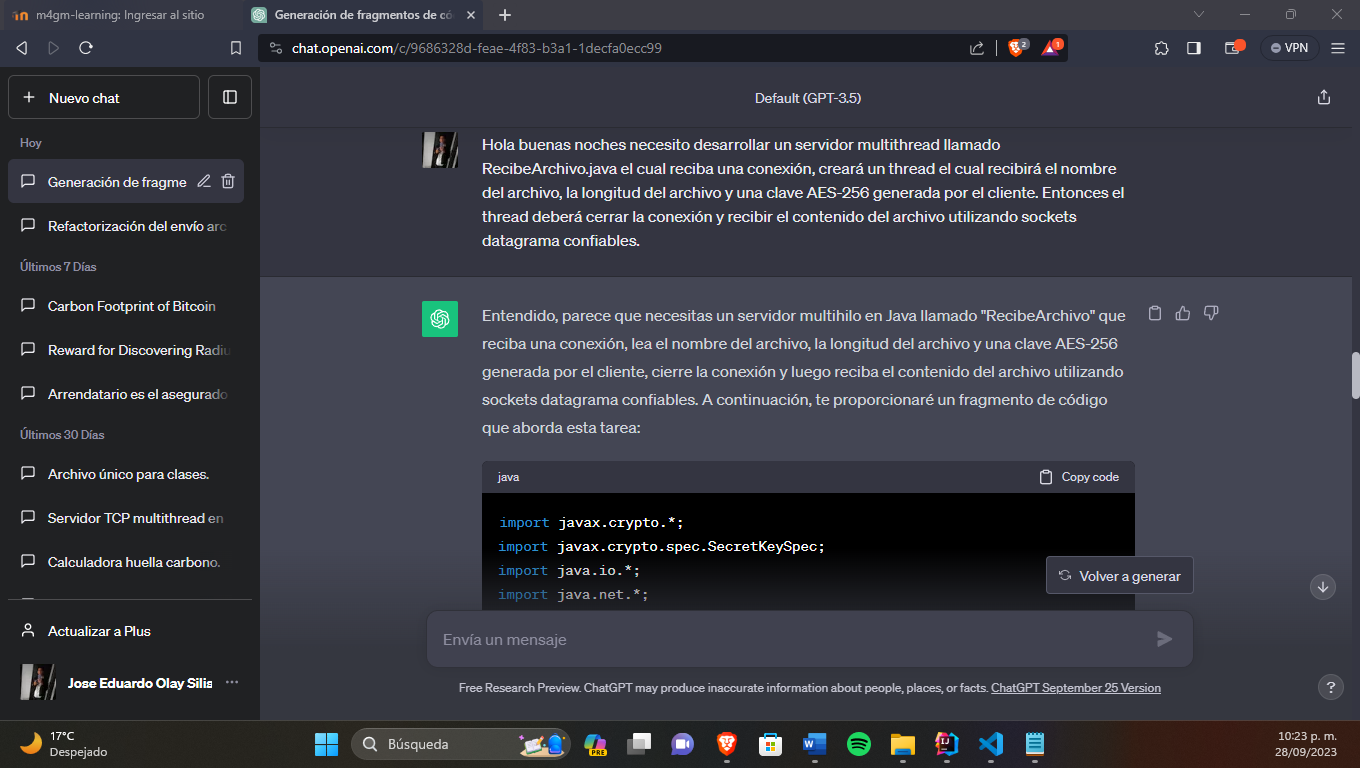
**Introducción**

En la presente práctica se implementarán un par de programas diseñados para la transferencia segura y confiable de archivos a través de una red. Estos programas utilizan cifrado AES y verificación CRC32 para garantizar la privacidad y la integridad de los datos transferidos. "EnviaArchivo" cifra y envía archivos desde el cliente, mientras que "RecibeArchivo" recibe, descifra y almacena los archivos en el servidor, asegurando una transmisión segura y fiable de datos.

**Desarrollo**

**Fase 1**

En primera instancia se realizó un servidor multithread el cual lo llamamos **RecibeArchivo.java** dicho programa se realizó con ayuda de la IA ChatGPT de la siguiente manera.

****

**Captura de pantalla de computadora

Descripción generada automáticamente**

**Captura de pantalla de computadora

Descripción generada automáticamente**

**Captura de pantalla de computadora

Descripción generada automáticamente**

**Captura de pantalla de computadora

Descripción generada automáticamente**

**Captura de pantalla de computadora

Descripción generada automáticamente**

**Captura de pantalla de computadora

Descripción generada automáticamente**

Tras la conversación realizada con el chat, el código en un principio no funcionó por lo que tuve que meterle mano por mi cuenta. A continuación, iré explicando poco a poco como es que en el desarrollo de este programa y lo que proporciono el chat pude ir implemento la integridad, llegada de paquetes, descarte de duplicados, orden y encriptado.

* **Confiabilidad (Verificación de CRC32):**

La verificación de CRC32 se realiza en el servidor después de recibir un paquete y antes de procesarlo. Si se detecta un error en la suma de comprobación, se solicita un reenvío.

|  |
| --- |
| // Dentro del método run() de la clase ClienteHandler  // ...  while (fileOutputStream.getChannel().size() < longitudArchivo) {      socket.receive(solicitud);      byte[] decryptedBytes = descifrador.doFinal(solicitud.getData(), 0, solicitud.getLength());      // Verificar la suma de comprobación CRC32      ByteBuffer packetBuffer = ByteBuffer.wrap(decryptedBytes);      int packetId = packetBuffer.getInt();      int bytesRead = packetBuffer.getInt();      byte[] content = new byte[bytesRead];      packetBuffer.get(content);      byte[] receivedCrcBytes = new byte[8];      packetBuffer.get(receivedCrcBytes);      long receivedCrcValue = ByteBuffer.wrap(receivedCrcBytes).getLong();      Checksum crc32 = new CRC32();      crc32.update(content);      long calculatedCrcValue = crc32.getValue();      if (calculatedCrcValue != receivedCrcValue) {          System.err.println("Error en la verificación CRC32 para el paquete " + packetId + ". Se solicita reenvío...");          requestPacketResend(socket, solicitud.getAddress(), solicitud.getPort(), packetId);          continue;      }      // Continuar con el procesamiento del paquete      // ...  }  // ... |

* I**ntegridad (Cálculo de CRC32 en el cliente):**

En el cliente, se calcula el valor CRC32 para cada paquete antes de enviarlo al servidor.

|  |
| --- |
| // En el cliente antes de enviar un paquete  Checksum crc32 = new CRC32();  crc32.update(content);  long crcValue = crc32.getValue();  // Agregar crcValue al paquete antes de enviarlo  // ... |

* **Descarte de duplicados:**

En el servidor, se utiliza un conjunto (receivedPacketIds) para realizar un seguimiento de los identificadores de paquetes recibidos y evitar el procesamiento de paquetes duplicados.

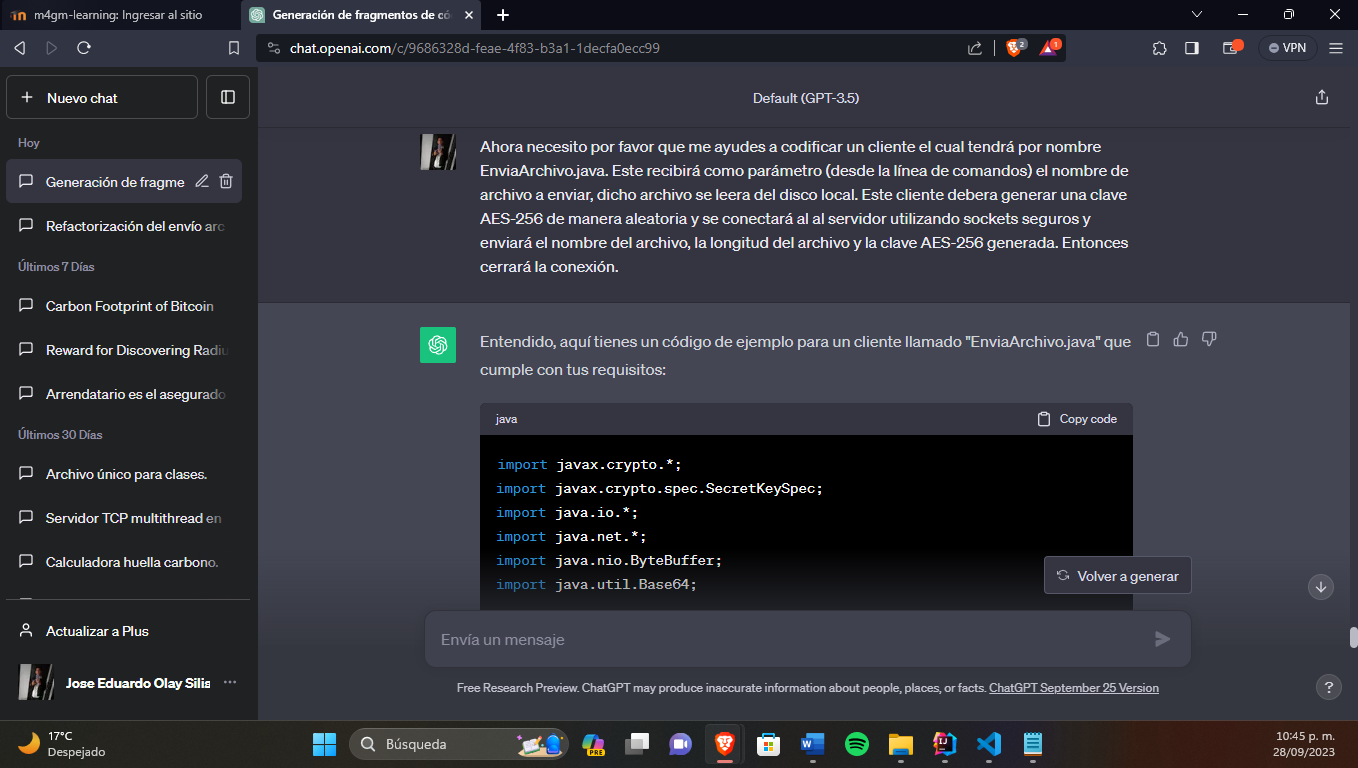
|  |
| --- |
| // Dentro del método run() de la clase ClienteHandler  // ...  Set<Integer> receivedPacketIds = new HashSet<>();  while (fileOutputStream.getChannel().size() < longitudArchivo) {      socket.receive(solicitud);      // Verificar si el paquete ya ha sido recibido      int packetId = /\* obtener el identificador del paquete \*/;      if (receivedPacketIds.contains(packetId)) {          continue; // Paquete duplicado, ignorar      }      // Marcar el paquete como recibido      receivedPacketIds.add(packetId);      // Continuar con el procesamiento del paquete      // ...  }  // ... |

* **Orden (Variable expectedPacketId):**

Se utiliza una variable (expectedPacketId) para rastrear el identificador de paquete esperado. Si se recibe un paquete con un identificador fuera de orden, se puede tomar una acción apropiada, como el descarte o el reenvío.

|  |
| --- |
| // Dentro del método run() de la clase ClienteHandler  // ...  int expectedPacketId = 1;  while (fileOutputStream.getChannel().size() < longitudArchivo) {      socket.receive(solicitud);      // Verificar si el identificador de paquete coincide con el esperado      int packetId = /\* obtener el identificador del paquete \*/;      if (packetId != expectedPacketId) {          // Paquete fuera de orden, tomar una acción apropiada          // ...          continue;      }      // Continuar con el procesamiento del paquete      // ...      expectedPacketId++; // Actualizar el identificador de paquete esperado  }  // ... |

Una vez realizado esto tocó realizar el cliente llamado **EnviaArchivo.java** el cual de igual manera se realizó con ayuda del Chat. A continuación, se muestras las instrucciones realizadas.



Captura de pantalla de computadora

Descripción generada automáticamente

Captura de pantalla de computadora

Descripción generada automáticamente

Captura de pantalla de computadora

Descripción generada automáticamente

Captura de pantalla de computadora

Descripción generada automáticamente

Dicho programa como tal al momento de compilarlo tal cual se genero con errores debido a que faltaban cosas por agregar. Dichas implementaciones se hicieron de mano propia.

**Fase 2**

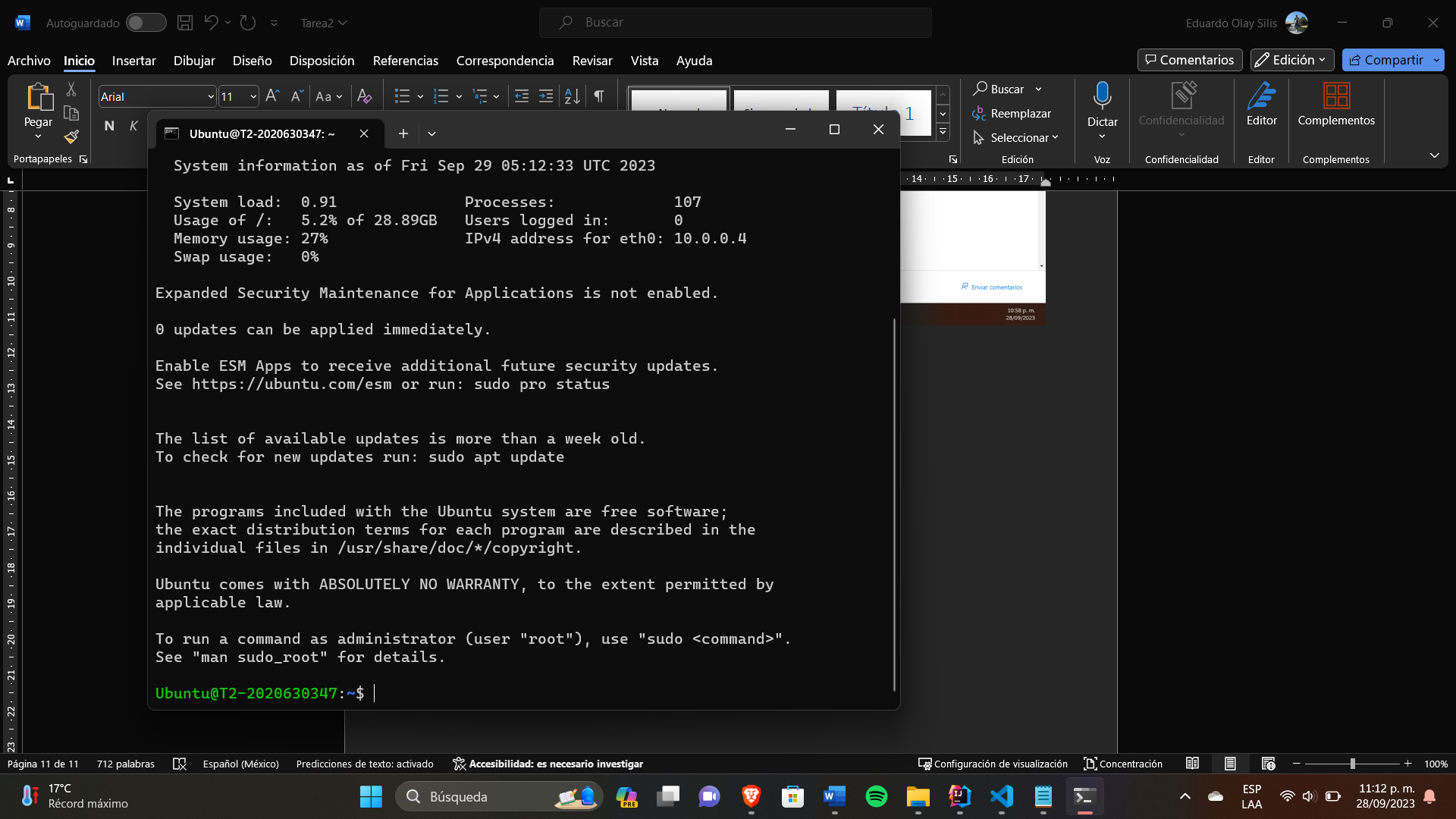
Una vez teniendo los archivos EnviaArchivo.java y RecibeArchivo.java se comenzó a realizar la máquina virtual la cual simbolizará el servidor.

Comenzamos creando la máquina virtual con Ubuntu 20 en Azure con los nombres y características específicas que se mencionaron en las instrucciones de la práctica.

Captura de pantalla de computadora

Descripción generada automáticamente

Una vez creada procedemos a ingresar a la máquina creada mediante el ssh

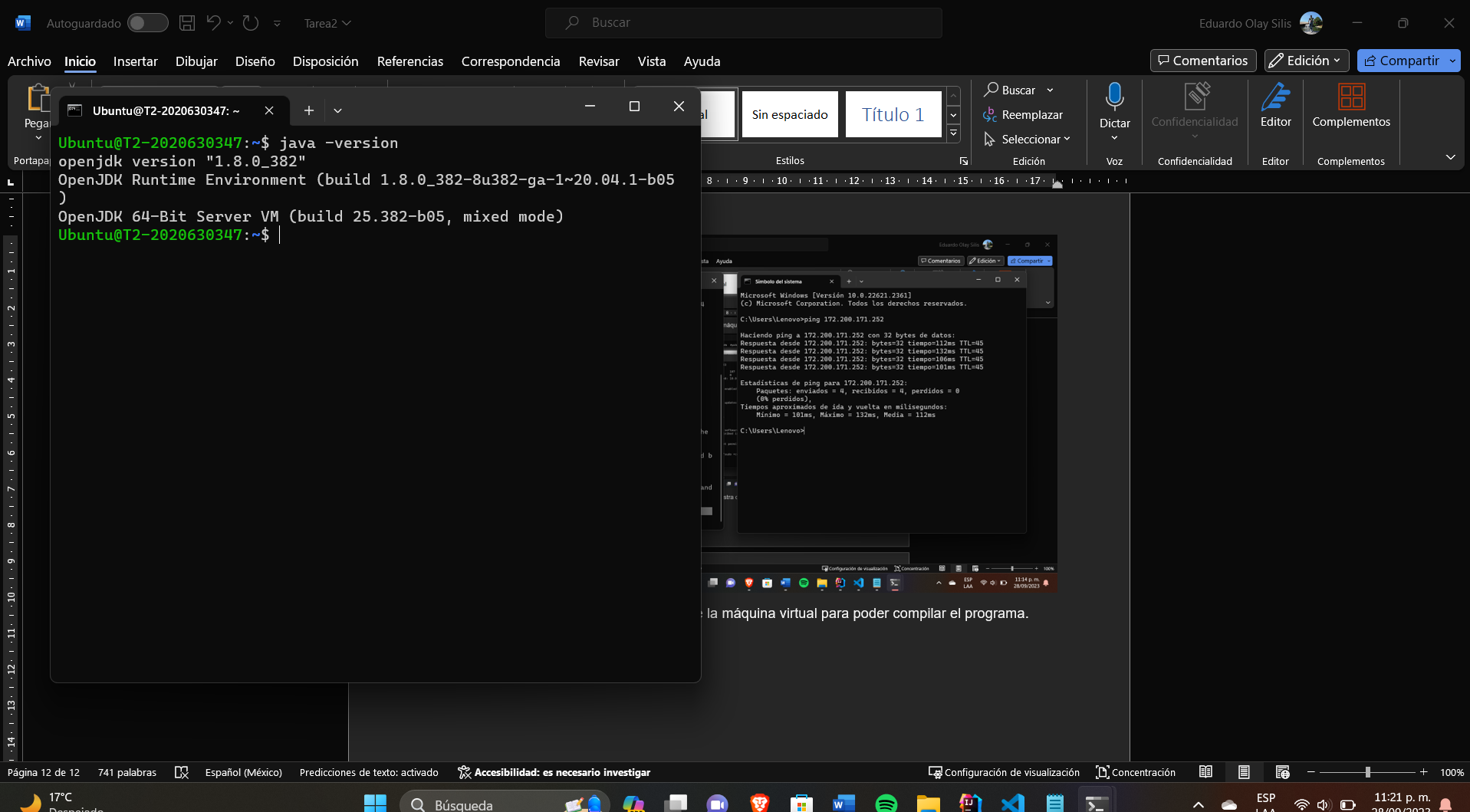


Ahora procedemos a hacerle ping desde nuestra computadora local para verificar la conexión.

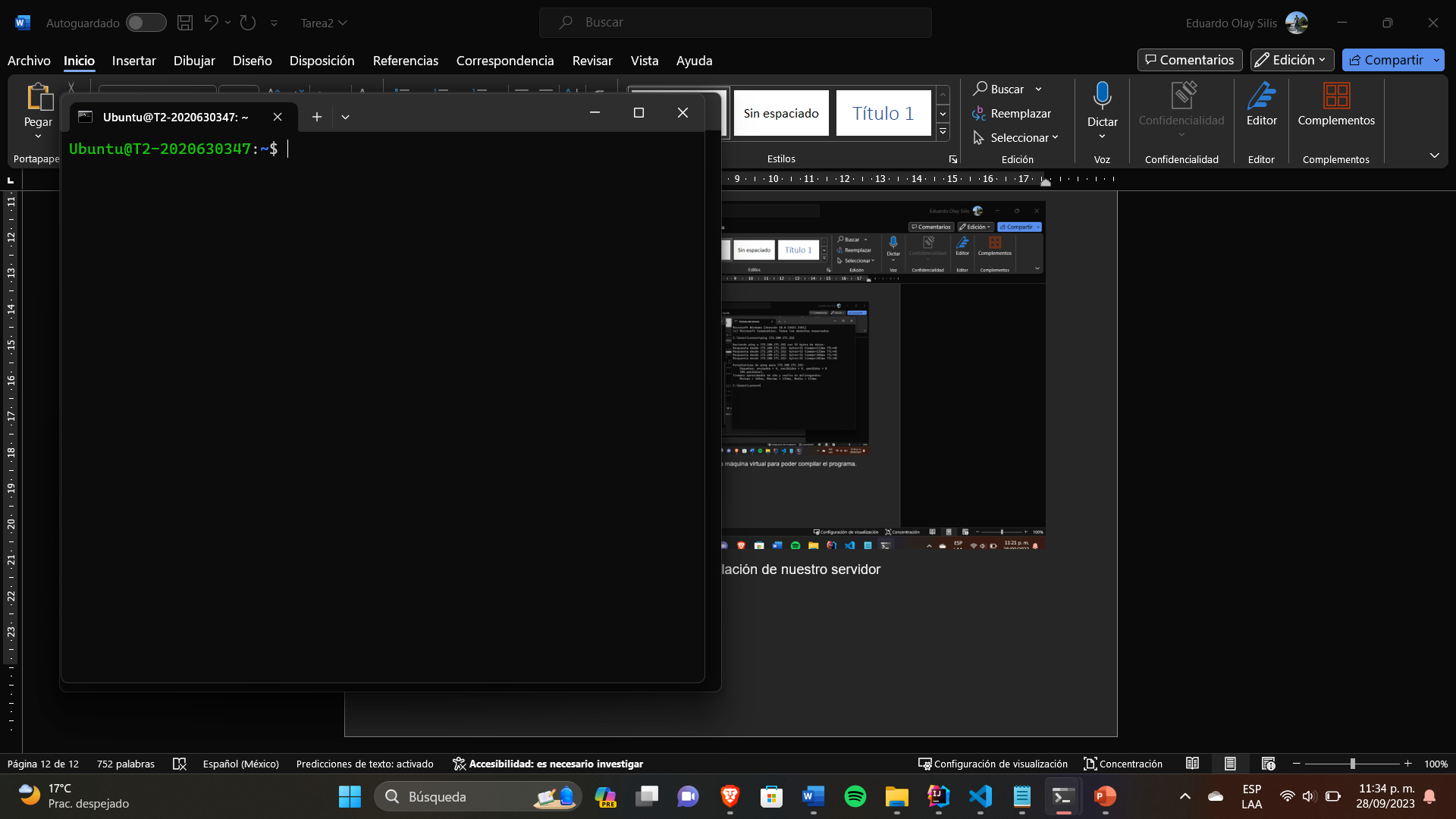
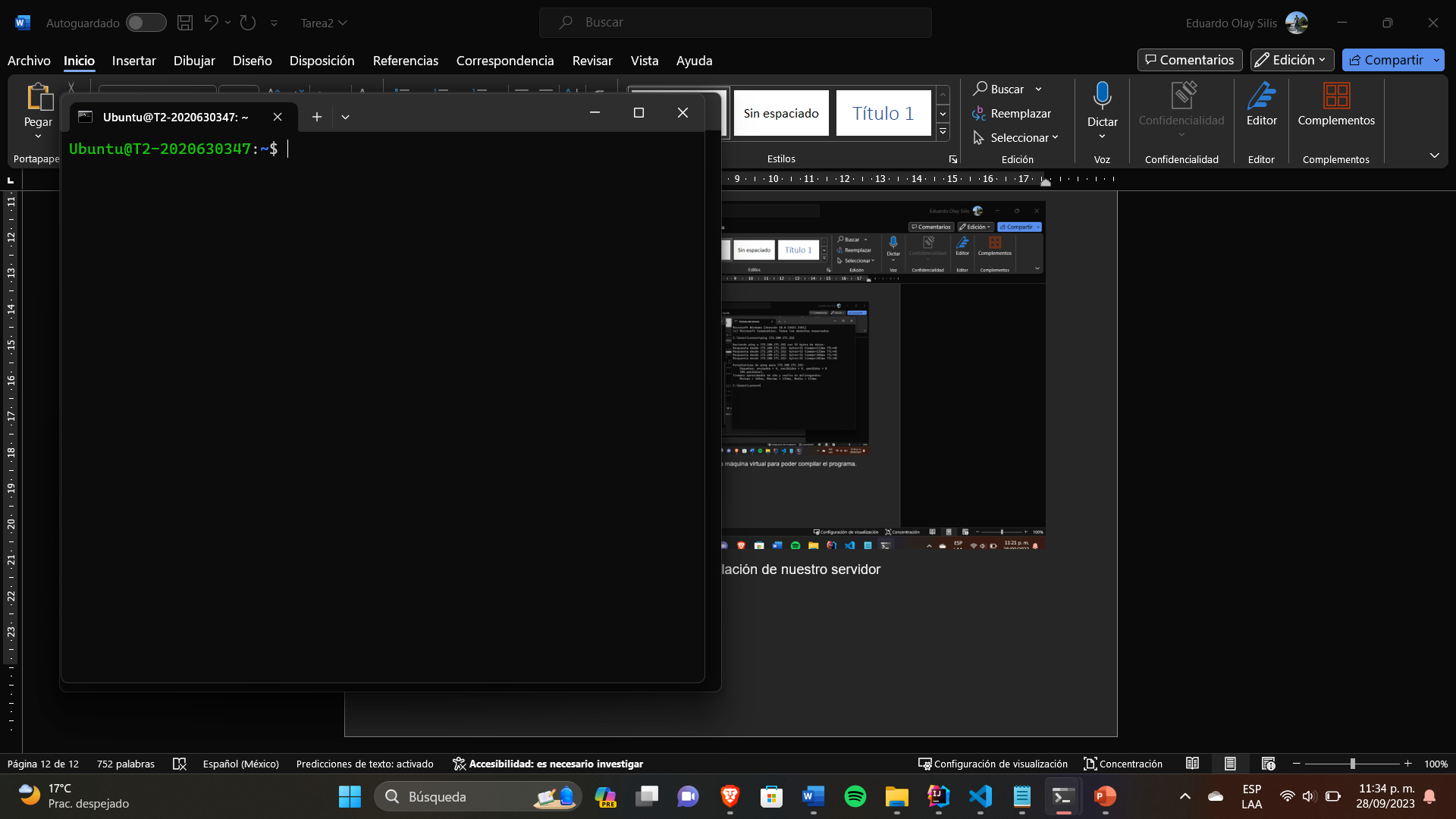
Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

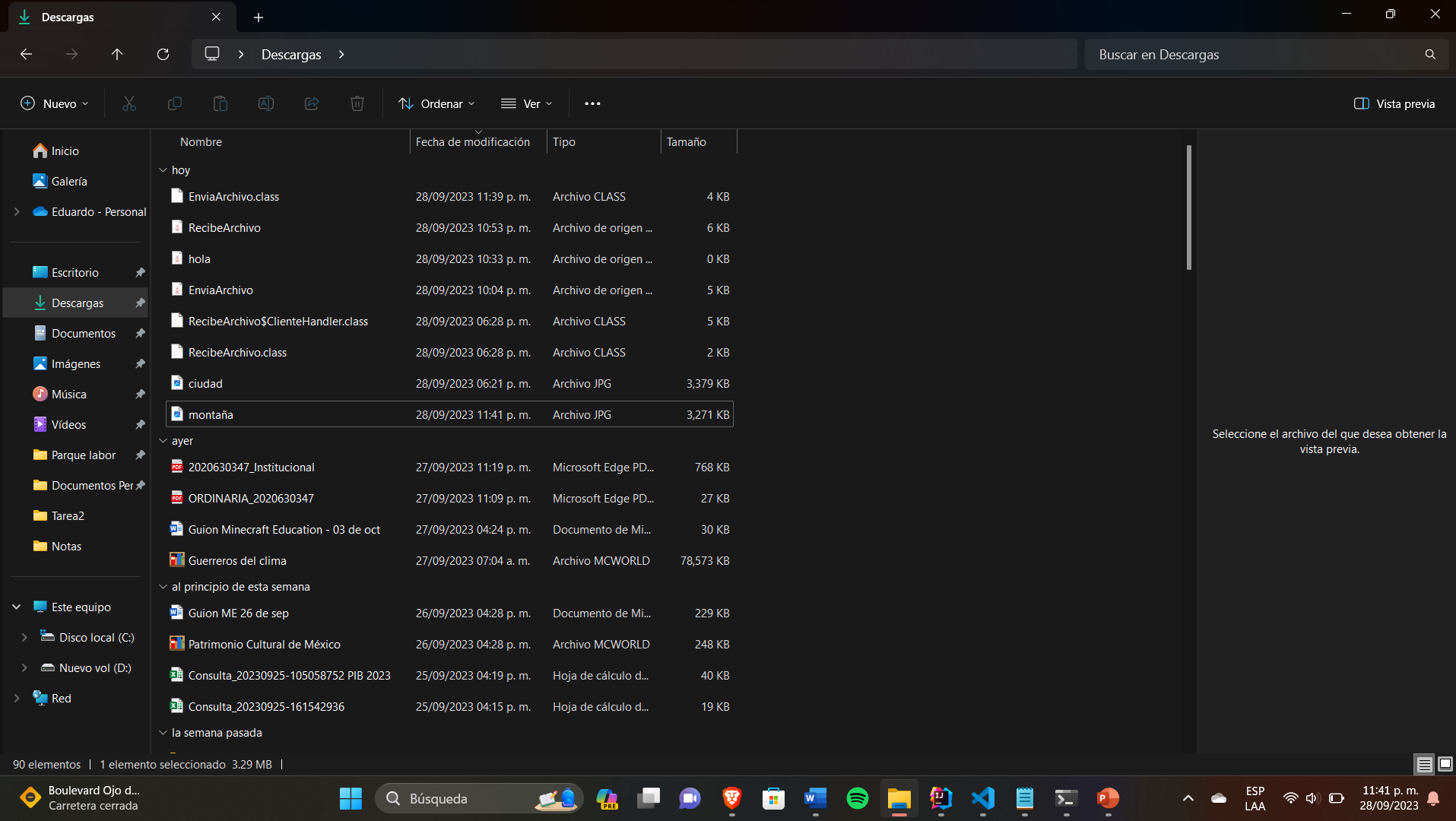
Ahora procedemos a instalar el jdk dentro de la máquina virtual para poder compilar el programa.



Una vez pasado el archivo comenzamos la compilación de nuestro servidor

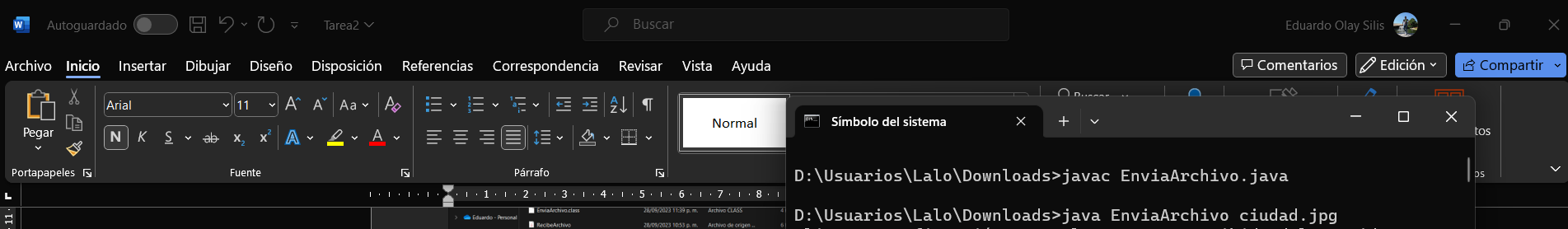


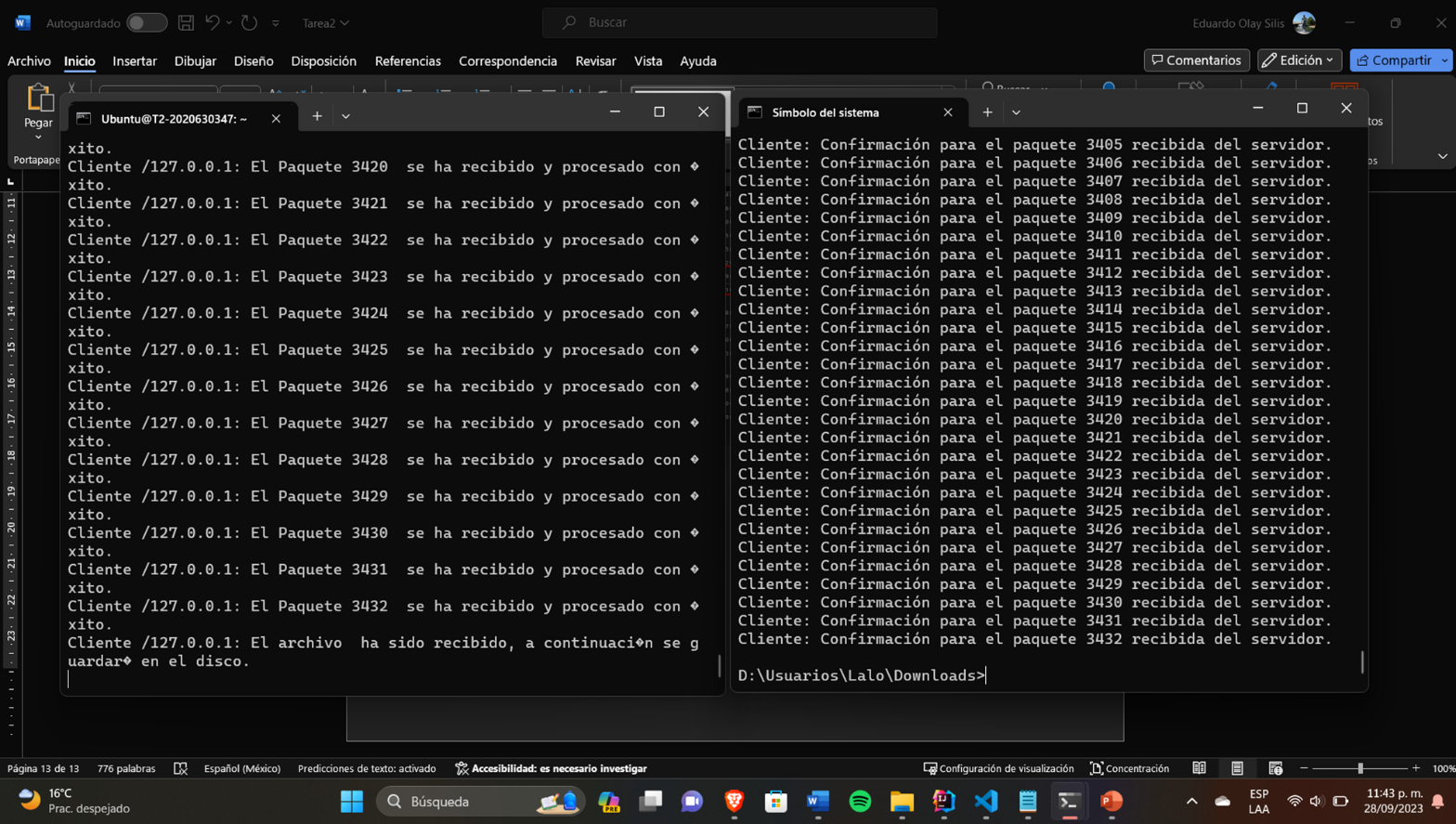
Para el envió de archivos utilizaremos 2 imágenes una de un bosque y la otra de una ciudad



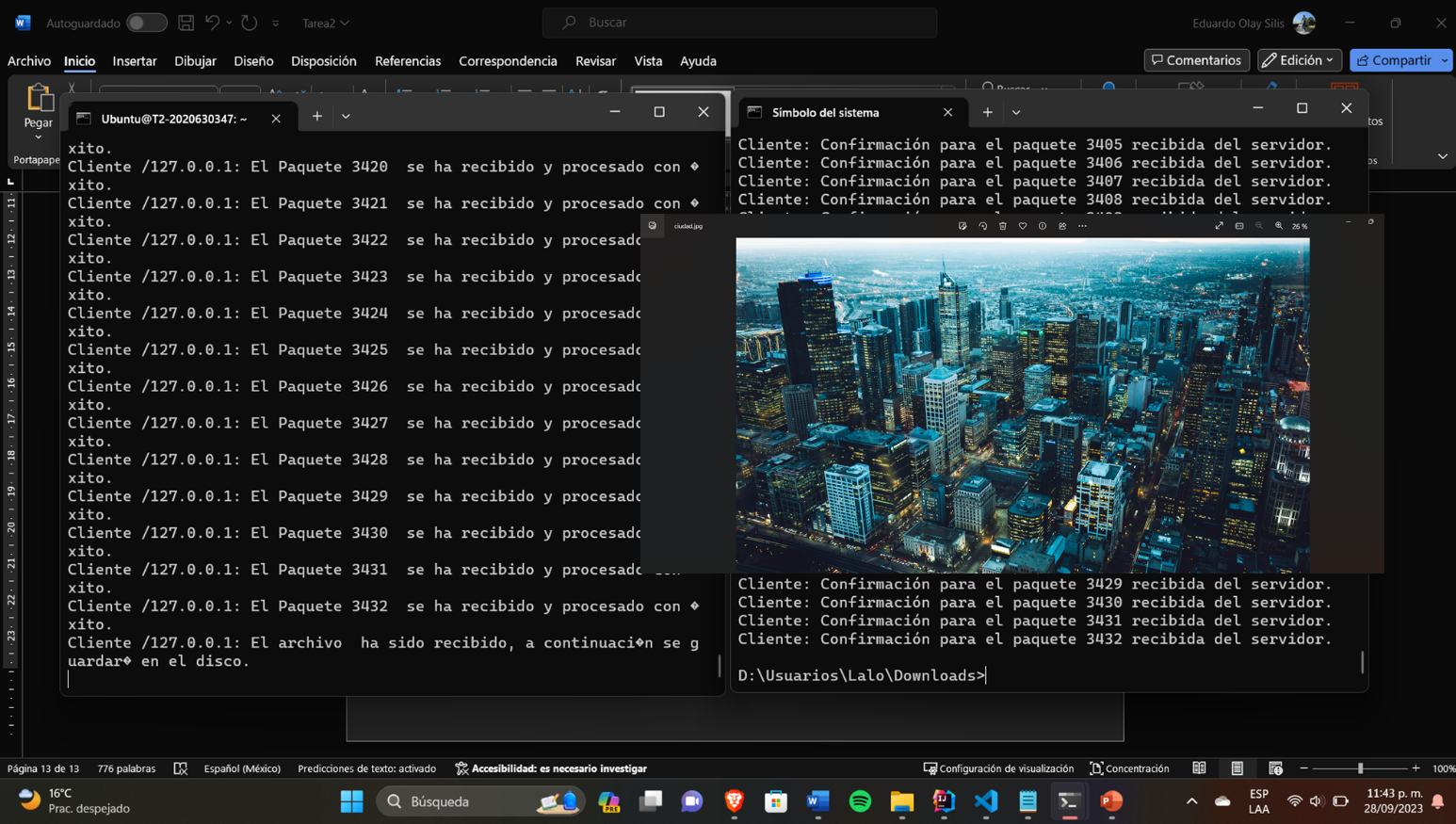
Dando los siguientes resultados

**Envío Ciudad.jpg**

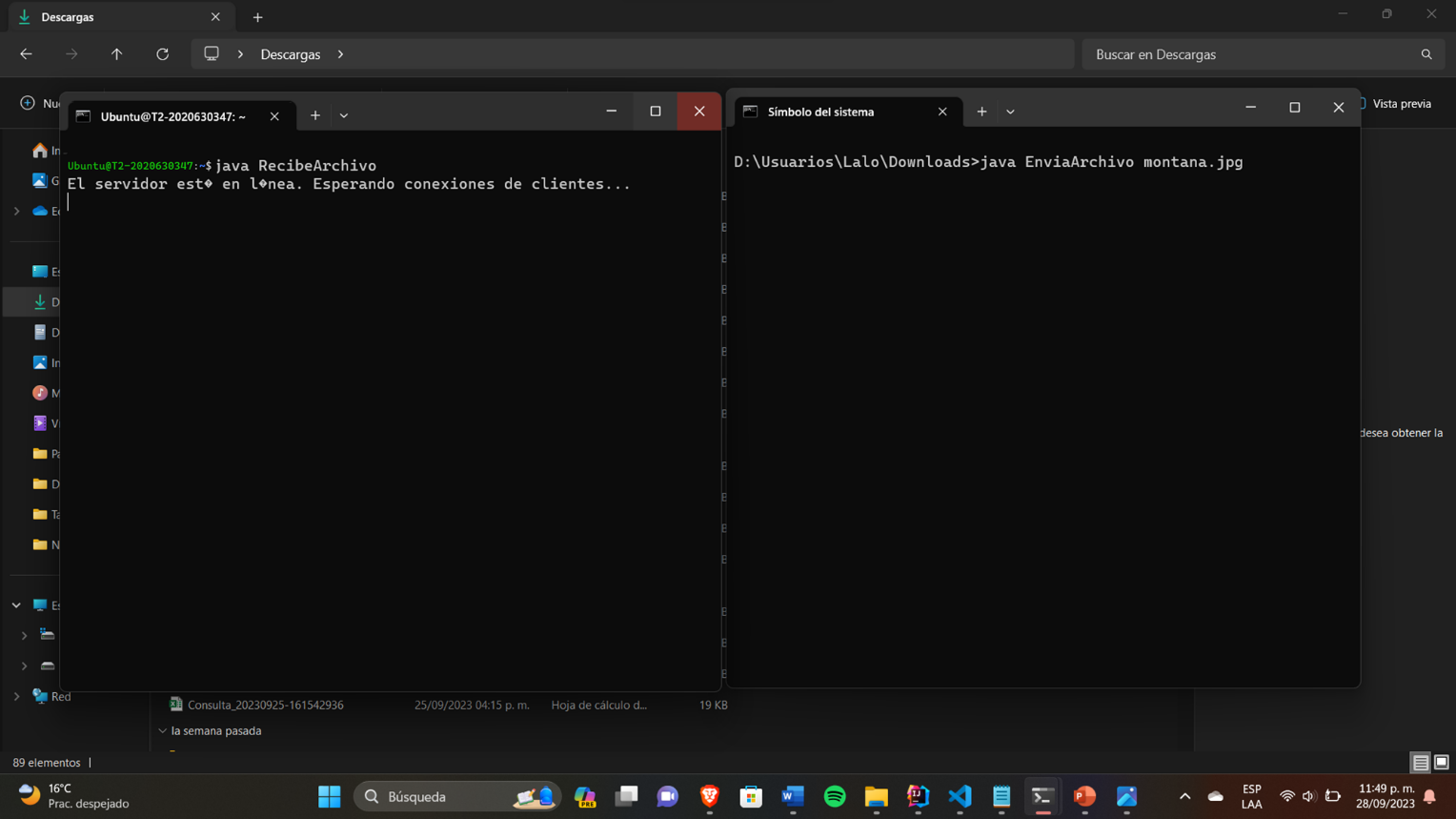
****

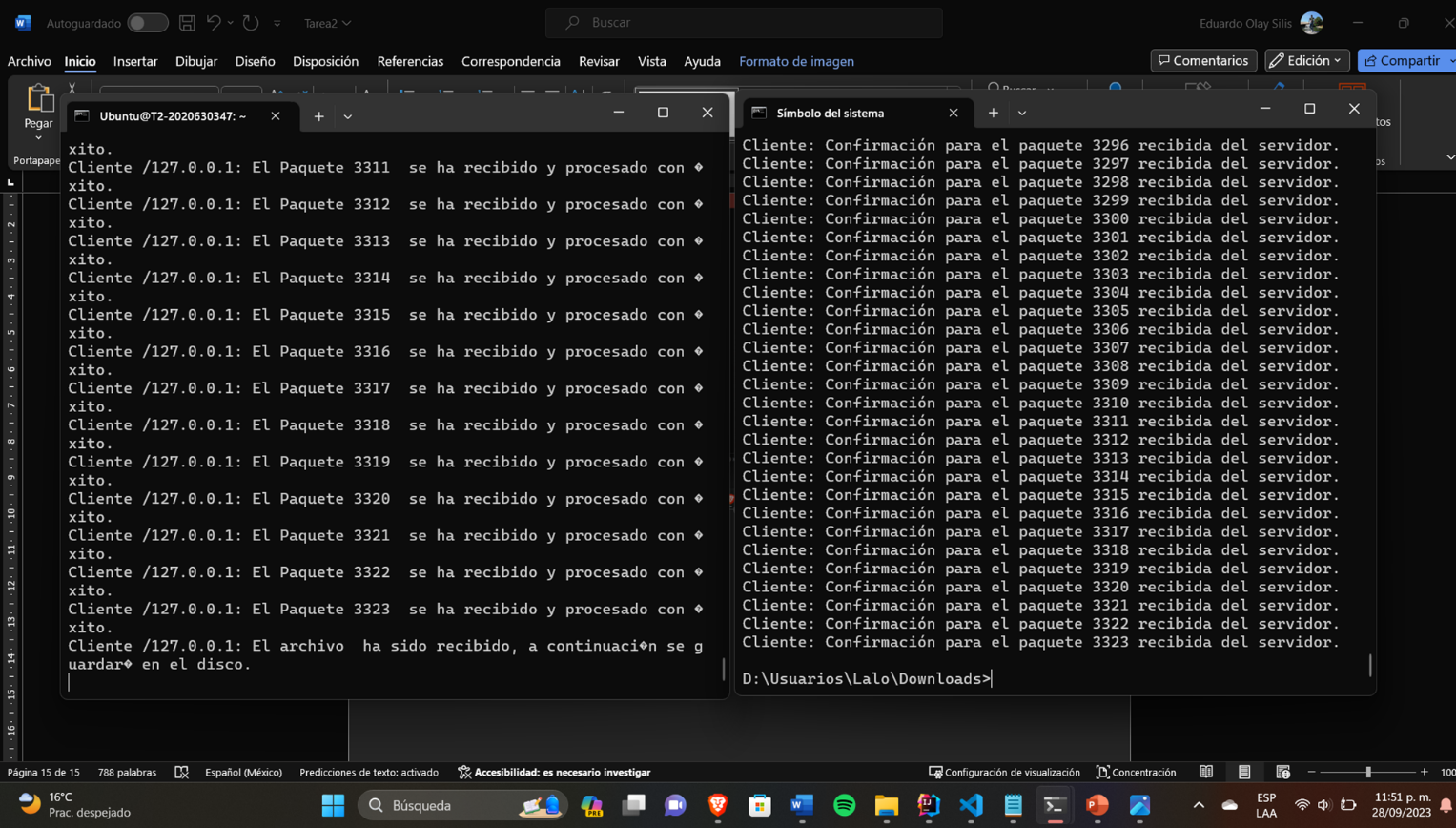
****

Y al abrir el archivo observamos que se envió correctamente



**Envío Ciudad.jpg**





Con esto verificando que ambos paquetes se recibieron sin ninguna pérdida.

**Conclusión:**

En este trabajo, hemos desarrollado dos aplicaciones, un servidor llamado "RecibeArchivo" y un cliente llamado "EnviaArchivo", que juntos permiten la transferencia segura y confiable de archivos entre un cliente y un servidor utilizando sockets Datagrama UDP y AES-256 para el cifrado de datos. A continuación, destacamos los puntos clave de nuestra implementación:

Servidor (RecibeArchivo): El servidor está diseñado como un servidor multihilo que maneja las solicitudes entrantes de clientes. Implementa una serie de características importantes:

Confiabilidad: Utiliza la verificación de suma de comprobación CRC32 para garantizar la integridad de los paquetes recibidos y solicitar reenvíos en caso de errores.

Descarte de duplicados: Evita el procesamiento de paquetes duplicados mediante el seguimiento de los identificadores de paquetes recibidos.

Orden: Mantiene el orden de los paquetes recibidos utilizando un identificador de paquete esperado.

Encriptado: Utiliza el algoritmo de encriptación AES-256 para cifrar los datos transmitidos de manera segura.

Cliente (EnviaArchivo): El cliente es responsable de generar una clave AES-256 de manera aleatoria, leer el archivo especificado y enviar el nombre del archivo, su longitud y la clave AES-256 al servidor.

Generación de clave: Genera de manera segura una clave AES-256 aleatoria para cada transferencia de archivo.

Encriptado: Utiliza el mismo algoritmo AES-256 para cifrar los datos antes de enviarlos al servidor.

Envío de metadatos: Envía metadatos importantes al servidor, incluido el nombre del archivo y su longitud, para facilitar la recepción y el procesamiento en el lado del servidor.

En conjunto, estos códigos permiten la transferencia confiable y segura de archivos entre el cliente y el servidor a través de conexiones UDP. La implementación de características como la integridad, el orden, la confiabilidad y el cifrado asegura que los datos se transmitan de manera efectiva y segura entre las partes involucradas. Sin embargo, es importante recordar que en una aplicación en producción, se deben considerar aspectos adicionales como la gestión de errores, la autenticación y la seguridad adicional según los requisitos específicos del sistema.