Logotipo

Descripción generada automáticamenteDibujo en blanco y negro

Descripción generada automáticamente con confianza baja

Tarea 4: Multiplicación de matrices utilizando objetos distribuidos

Instituto Politécnico Nacional

Escuela Superior de Cómputo

Olay Silis Jose Eduardo

4CV12

Prof. Carlos Pineda Guerrero

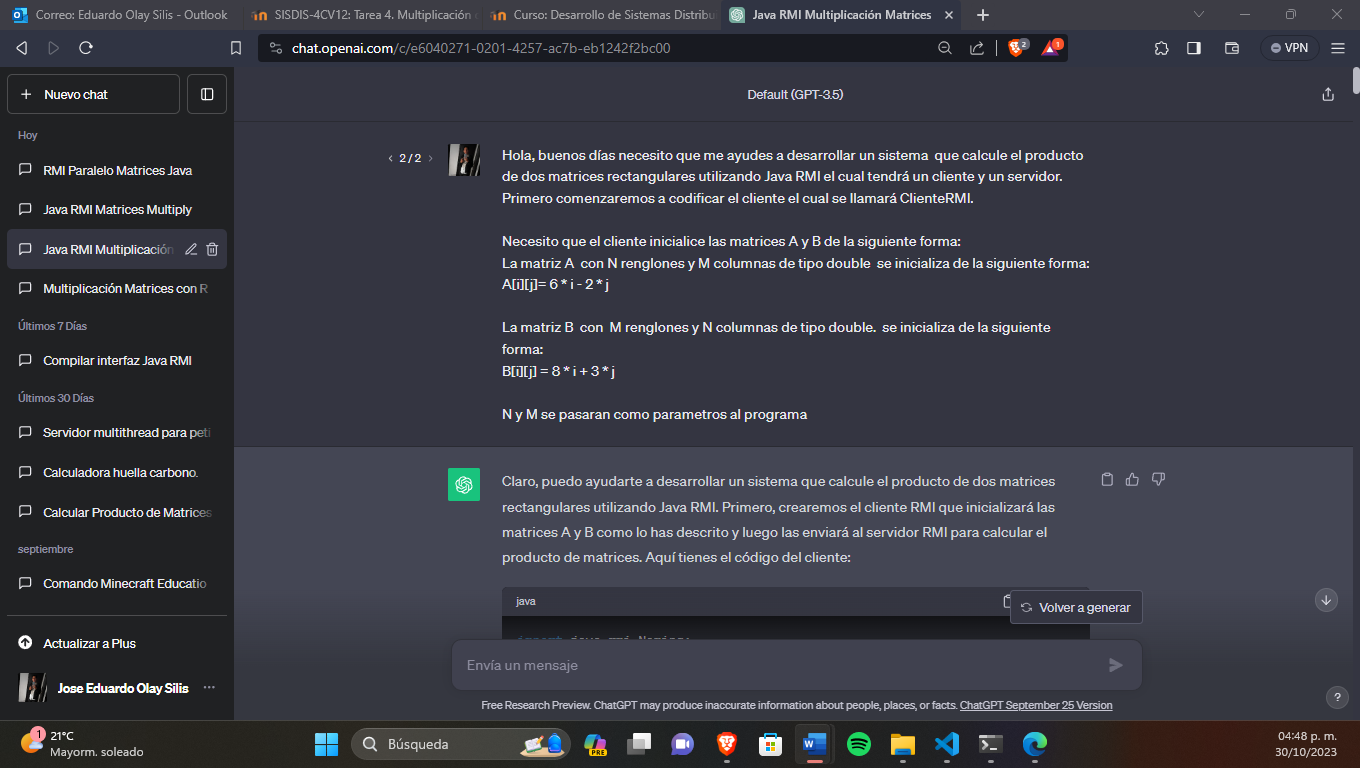
**Introducción**

Durante el desarrollo de esta práctica se implementó un programa en Java RMI en el cual el cliente inicializa y divide 2 matrices rectangulares para que mediante un método remoto pueda calcular la multiplicación de la matriz resultante a través de una interfaz RMI.

**Desarrollo**

**Fase 1**

El desarrollo del código de esta tarea lo realice utilizando ChatGPT y el programa de multiplicación de matrices proporcionado por el profesor y el cual se vio en clase. En primera instancia le fui preguntando al chat los pasos para ir creando las operaciones que realizaba el cliente las cuales eran inicializar las matrices, calcular la transpuesta y dividir dichas matrices en 4 partes iguales obteniendo lo que se muestra en las imágenes siguientes.



Captura de pantalla de computadora

Descripción generada automáticamente

Captura de pantalla de computadora

Descripción generada automáticamente

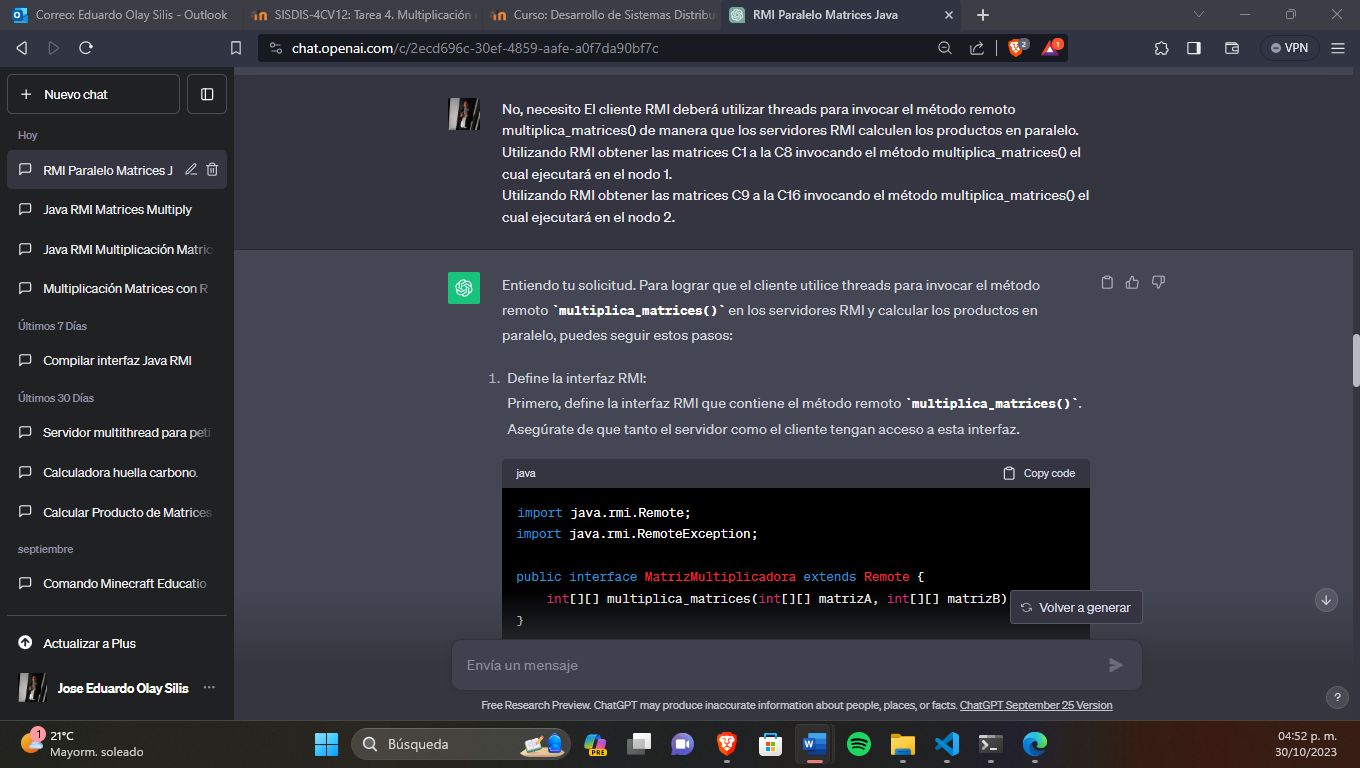
Captura de pantalla de computadora

Descripción generada automáticamente

Captura de pantalla de computadora

Descripción generada automáticamente

Una vez proporcionado esto le pedí que me diera una idea para implementar los threads dentro del cliente para que de la matriz que se dividieran se fueran multiplicando paralelamente en los nodos correspondientes.



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

Del código anterior lo que utilice para la codificación del programa fue la ultima parte del cliente el cual mandaba llamar a los nodos y el método remoto utilizando threads. Una vez teniendo esas ideas base comencé por mi propia cuenta a implementar el código de los programas basándome en lo proporcionado por el Chat y lo visto en clase.

Comencé codificando la **InterfazRMI.java**  la cual contiene el método multiplicaMatrices() la cual recibe como parámetros las 2 matrices y el número de líneas y columnas.

|  |
| --- |
| import java.rmi.Remote;  import java.rmi.RemoteException;  /\*      Declarar firma de métodos que serán sobrescritos  \*/  public interface InterfazRMI extends Remote {      public double[][] multiplicaMatrices(double[][] A, double[][] B,int N, int M) throws RemoteException;  } |

Posteriormente comencé con la codificación del ClienteRMI.java el cual implemente el Restrigy de manera diferente a la vista en clase debido a que mediante la URL no sabía que estaba realizando mal que al momento de ejecutar el programa no funcionaba, debido a esto la IP y el numero de puerto los declare en una variable privada y mediante la interfaz Registry y el nombre del registro referencie los 2 Nodos.

|  |
| --- |
| public class ClienteRMI {      private static final String IP = "localhost"; // Puedes cambiar a localhost      private static final String IP2 = "localhost"; // Puedes cambiar a localhost      private static final int PUERTO = 8080; //Si cambias aquí el puerto, recuerda cambiarlo en el servidor      public static void main(String[] args)              throws MalformedURLException, RemoteException, NotBoundException, InterruptedException {          if (args.length != 2) {              System.out.println("Uso: java ClienteRMI <N> <M>");              System.exit(1);          }          int N = Integer.parseInt(args[0]);          int M = Integer.parseInt(args[1]);          try {              Registry registry = LocateRegistry.getRegistry(IP, PUERTO);              Registry registry2 = LocateRegistry.getRegistry(IP2, PUERTO);              InterfazRMI Nodo1 = (InterfazRMI) registry.lookup("Nodo1"); //Buscar en el registro...              InterfazRMI Nodo2 = (InterfazRMI) registry2.lookup("Nodo2"); //Buscar en el registro...              // Listo para ocupar los metodos |

Por la parte del servidor (**ServidorRMI.java**) igual cambio un poquito la implementación ya que utilice la interface Remote el cual exporta un objeto remoto de la clase “ClaseRMI” registrándolo en el registro RMI en el puerto 8080, bajo los nombres "Nodo1" y "Nodo2". Haciendo que el cliente pueda conectarse a este servidor RMI y realizar llamadas remotas a los métodos del objeto remoto utilizando los nombres "Nodo1" y "Nodo2".

|  |
| --- |
| import java.rmi.Remote;  import java.rmi.registry.LocateRegistry;  import java.rmi.registry.Registry;  import java.rmi.server.UnicastRemoteObject;  public class ServidorRMI {      private static final int PUERTO = 8080; //Si cambias aquí el puerto, recuerda cambiarlo en el cliente      public static void main(String[] args) throws Exception {       Remote remote = UnicastRemoteObject.exportObject(new ClaseRMI(), 0);          Registry registry = LocateRegistry.createRegistry(PUERTO);          System.out.println("Servidor escuchando en el puerto " + String.valueOf(PUERTO));          registry.bind("Nodo1", remote); // Registrar Nodo           registry.bind("Nodo2", remote); // Registrar Nodo      }  } |

Posteriormente regresando al client Inicializamos las matrices A y B y calculamos su Transpuesta para posteriormente dividir tanto la Matriz A como la matriz BT en 4 partes iguales, quedando submatrices de N/4 renglones por M columnas.

|  |
| --- |
| // Inicializar las matrices A y B              double[][] A = new double[N][M];              double[][] B = new double[M][N];                System.out.println("Inicializando Matrices..");              for (int i = 0; i < N; i++) {                  for (int j = 0; j < M; j++) {                      A[i][j] = 6 \* i - 2 \* j;                  }              }              for (int i = 0; i < M; i++) {                  for (int j = 0; j < N; j++) {                      B[i][j] = 8 \* i + 3 \* j;                  }              }              // Obtener la matriz traspuesta BT de la matriz B              double[][] BT = new double[N][M];              for (int i = 0; i < M; i++) {                  for (int j = 0; j < N; j++) {                      BT[j][i] = B[i][j];                  }              }              // Dividimos las matrices A y BT en N/4 renglones por M columnas              System.out.println("Dividiendo Matrices....");              double[][] A1 = separa\_matriz(A, 0);              double[][] A2 = separa\_matriz(A, (N / 4));              double[][] A3 = separa\_matriz(A, (N / 4) \* 2);              double[][] A4 = separa\_matriz(A, (N / 4) \* 3);              double[][] BT1 = separa\_matriz(BT, 0);              double[][] BT2 = separa\_matriz(BT, (N / 4));              double[][] BT3 = separa\_matriz(BT, (N / 4) \* 2);              double[][] BT4 = separa\_matriz(BT, (N / 4) \* 3); |

Una vez hecho esto, en 2 threads comenzamos a multiplicar cada uno de los renglones de la matriz A y BT invocando la método remoto multiplicaMatrices() dando como resultado una sección de la matriz resultante C la cual mediante el método acomoda\_matriz() se va colocando en su renglón y columna correspondiente para que de esta manera se vaya creando la matriz resultante C.

|  |
| --- |
| // Utilizamos hilos para comenzar a calcular las multiplicaciones mediante              double[][] C = new double[N][N];                System.out.println("\nCalculando la multiplicacion de Matrices....");              // threads y el metodo remoto              Thread thread1 = new Thread(() -> {                  try {                      double[][] C1 = Nodo1.multiplicaMatrices(A1, BT1,N,M);                      double[][] C2 = Nodo1.multiplicaMatrices(A1, BT2,N,M);                      double[][] C3 = Nodo1.multiplicaMatrices(A1, BT3,N,M);                      double[][] C4 = Nodo1.multiplicaMatrices(A1, BT4,N,M);                      double[][] C5 = Nodo1.multiplicaMatrices(A2, BT1,N,M);                      double[][] C6 = Nodo1.multiplicaMatrices(A2, BT2,N,M);                      double[][] C7 = Nodo1.multiplicaMatrices(A2, BT3,N,M);                      double[][] C8 = Nodo1.multiplicaMatrices(A2, BT4,N,M);                      // Obtenemos la matriz C de las submatrices obteidas por los métodos remotos                      acomoda\_matriz(C, C1, 0, 0,N);                      acomoda\_matriz(C, C2, 0, (N/4),N);                      acomoda\_matriz(C, C3,0,(N/4)\*2,N);                      acomoda\_matriz(C, C4,0, (N/4)\*3,N);                      acomoda\_matriz(C, C5, (N/4),0,N );                      acomoda\_matriz(C, C6, (N/4), (N/4),N );                      acomoda\_matriz(C, C7, (N/4), (N/4)\*2,N);                      acomoda\_matriz(C, C8, (N/4), (N/4)\*3,N);                  } catch (RemoteException e) {                      // Manejo de excepciones específico para el thread 1.                      e.printStackTrace();                  }              });              Thread thread2 = new Thread(() -> {                  try {                      double[][] C9 = Nodo2.multiplicaMatrices(A3, BT1,N,M);                      double[][] C10 = Nodo2.multiplicaMatrices(A3, BT2,N,M);                      double[][] C11 = Nodo2.multiplicaMatrices(A3, BT3,N,M);                      double[][] C12 = Nodo2.multiplicaMatrices(A3, BT4,N,M);                      double[][] C13 = Nodo2.multiplicaMatrices(A4, BT1,N,M);                      double[][] C14 = Nodo2.multiplicaMatrices(A4, BT2,N,M);                      double[][] C15 = Nodo2.multiplicaMatrices(A4, BT3,N,M);                      double[][] C16 = Nodo2.multiplicaMatrices(A4, BT4,N,M);                     // Obtenemos la matriz C de las submatrices obteidas por los métodos remotos                      acomoda\_matriz(C, C9, (N/4)\*2, 0,N);                      acomoda\_matriz(C, C10, (N/4)\*2, (N/4),N);                      acomoda\_matriz(C, C11, (N/4)\*2, (N/4)\*2,N);                      acomoda\_matriz(C, C12, (N/4)\*2, (N/4)\*3,N);                      acomoda\_matriz(C, C13, (N/4)\*3 ,0,N );                      acomoda\_matriz(C, C14, (N/4)\*3, (N/4),N);                      acomoda\_matriz(C, C15, (N/4)\*3, (N/4)\*2,N);                      acomoda\_matriz(C, C16, (N/4)\*3, (N/4)\*3,N);;                  } catch (RemoteException e) {                      // Manejo de excepciones específico para el thread 2.                      e.printStackTrace();                  }              });              thread1.start();              thread2.start();                thread1.join();              thread2.join(); |

Al final se calcula el cheksum de la matriz C y para el primer caso que se menciona dentro de la práctica se imprimen las matrices resultantes.

|  |
| --- |
| System.out.println("\nObteniendo Cheksum....");              //Se calcula el cheksum de la matriz C              double cheksum =0;              for(int i=0;i<N;i++){                  for(int j=0;j<N;j++){                      cheksum += C[i][j];                  }              }              // Mostrar el resultado              if(N==8 && M==4){                  System.out.println("Matriz A:");                  imprimirMatriz(A);                  System.out.println("\nMatriz B:");                  imprimirMatriz(B);                  System.out.println("\nMatriz c:");                  imprimirMatriz(C);                  System.out.println("\nEl Cheksum del producto de matrices es: "+cheksum);              }else{                  System.out.println("\nEl Cheksum del producto de matrices es: "+cheksum);              }          } catch (Exception e) {              e.printStackTrace();          }      } |

Por último creamos la claseRMI la cual contiene el método remoto multiplicaMatrices() en este caso como la clase implementa directamente la Interfaz no es necesaria agregar la SuperClase debido a la implementación anterior realizada tanto en el cliente como en el Servidor por lo que simplemente se codifica de la siguiente manera.

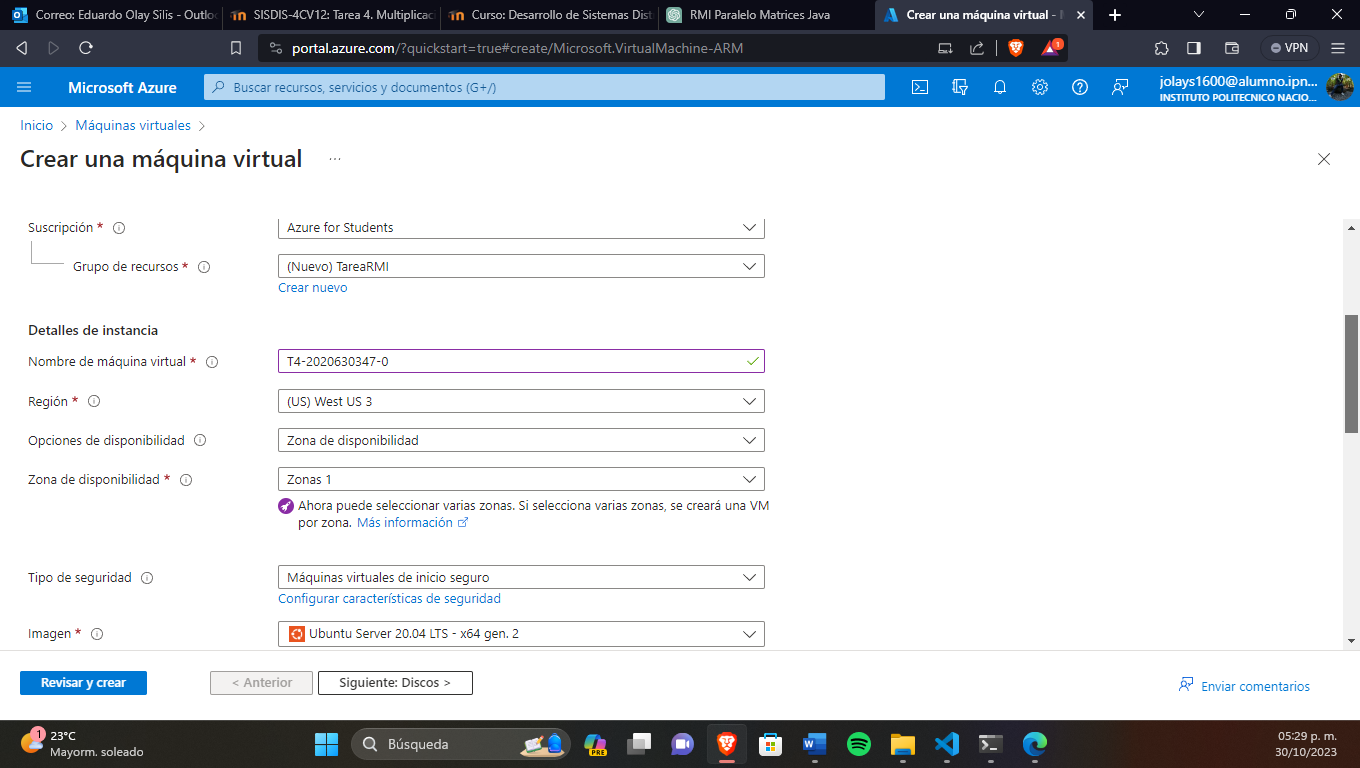
|  |
| --- |
| import java.rmi.RemoteException;  public class ClaseRMI  implements InterfazRMI {    public double[][] multiplicaMatrices(double[][] A, double[][] B, int N,int M) throws RemoteException{      double[][]C=new double[N/4][N/4];      for(int i=0;i<N/4;i++)        for(int j=0; j<N/4;j++)          for(int k=0;k<M;k++)              C[i][j] += A[i][k] \* B[j][k];        return C;    }    } |

Y ya con estos programas queda listo nuestro sistema.

**Fase 2**

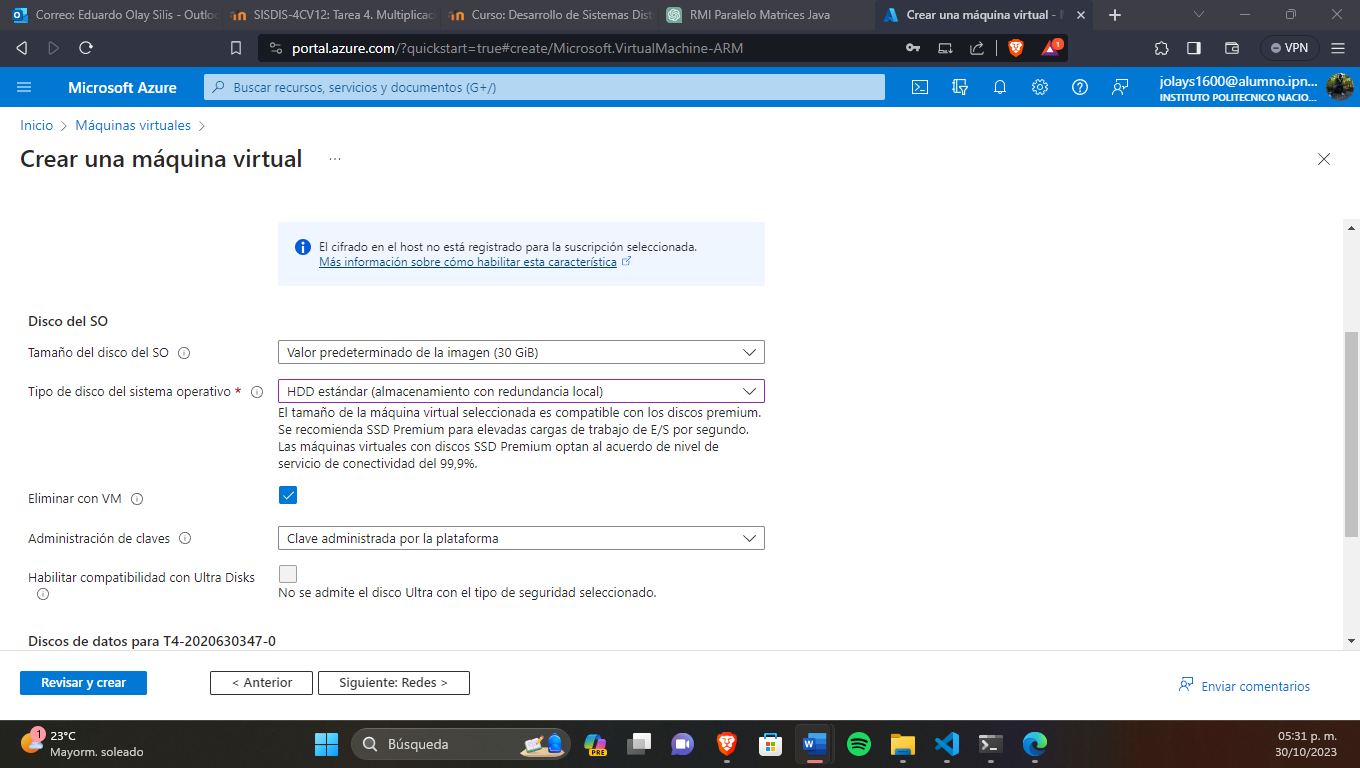
Una vez teniendo listo el programa comenzamos el proceso de creación de las máquinas virtuales los cuales simularan los nodos. A continuación, se pondrá el proceso de creación de la máquina virtual que simulará el Nodo 0.

Creamos un grupo de recursos para que los 3 nodos estén en el mismo, colocamos su nombre, la Imagen la cual será Ubutnu 20,el tipo de memoria el cual será de 4GB debido a la matriz grande del segundo caso que habrá que calcular, el usuario y contraseña incluyendo el puerto de entrada SSH.

 Una captura de pantalla de una computadora

Descripción generada automáticamente

Seleccionamos el HHD estándar



Comprobamos que se haya generado por default la Ip Pública

Una captura de pantalla de una computadora

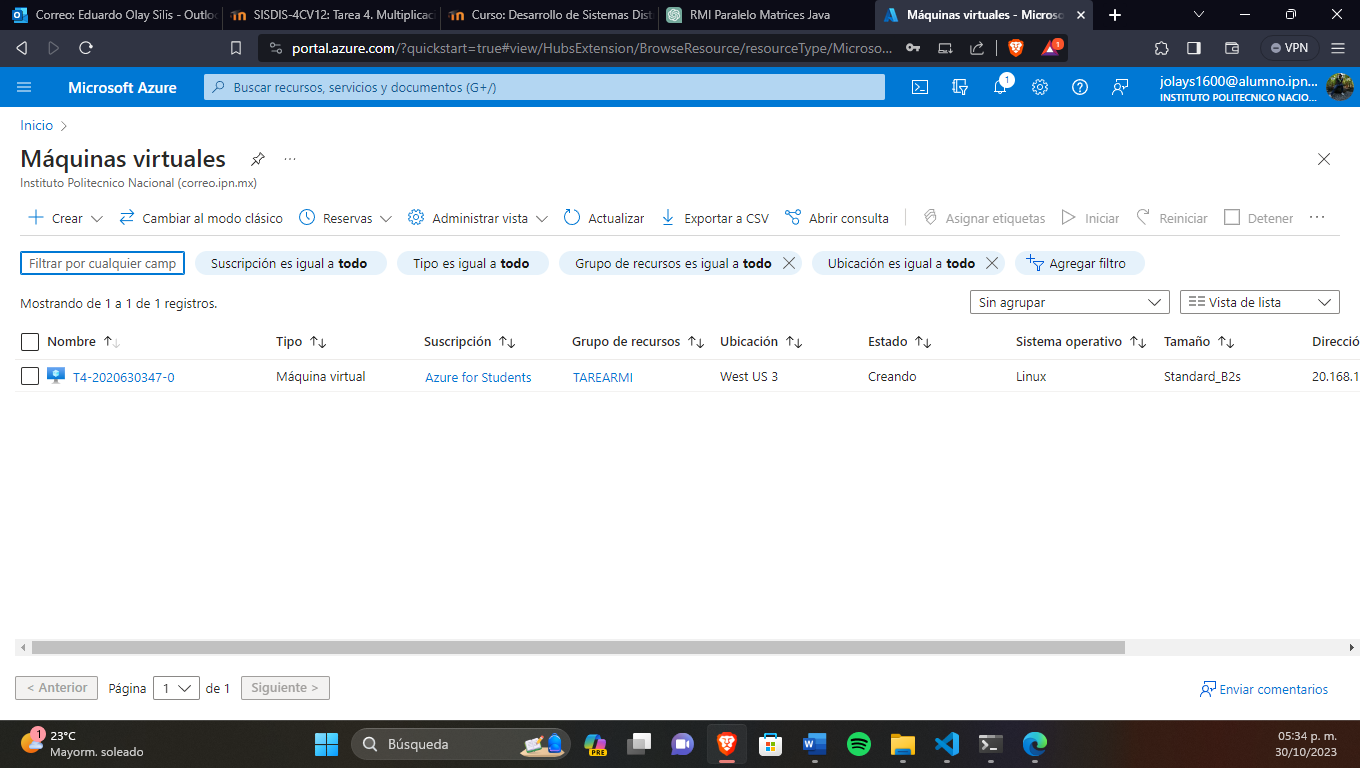
Descripción generada automáticamente

Desactivamos el diagnóstico de arranque

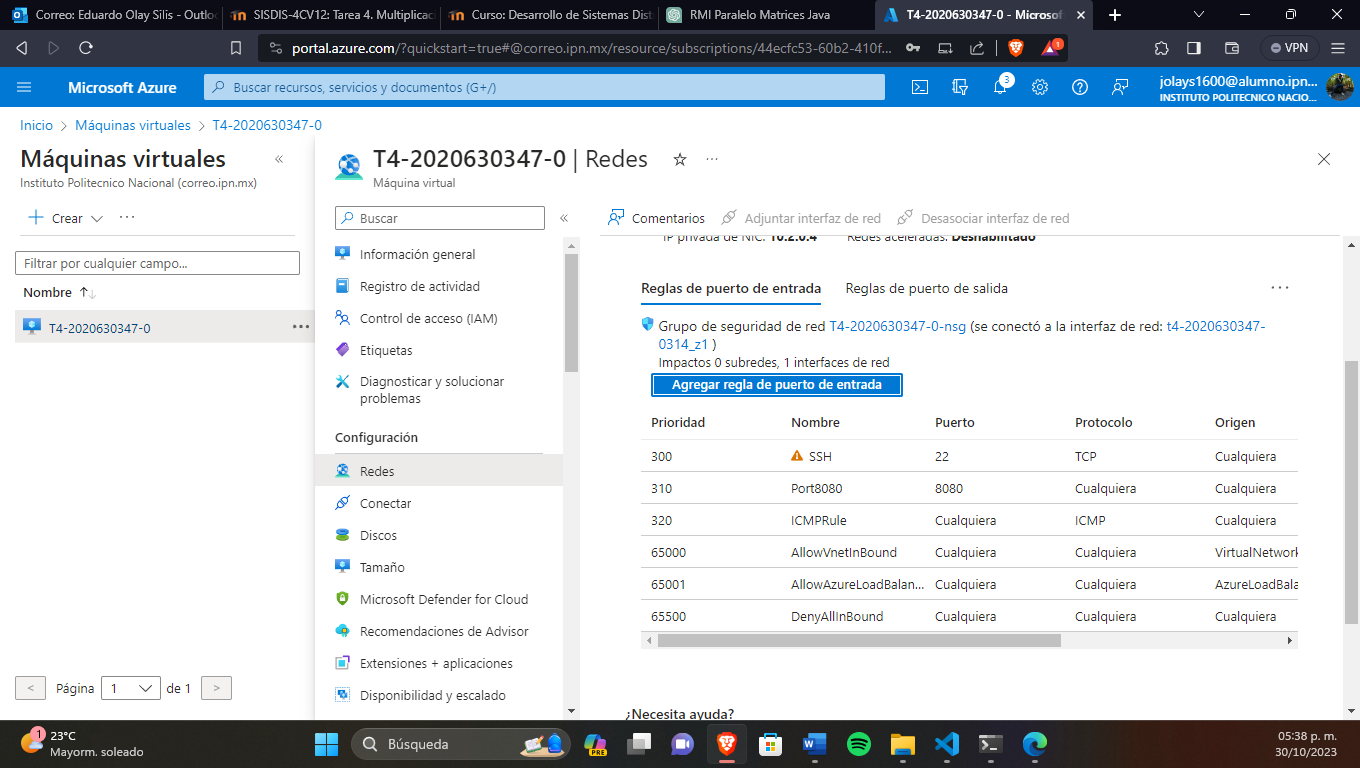
Una captura de pantalla de una computadora

Descripción generada automáticamente

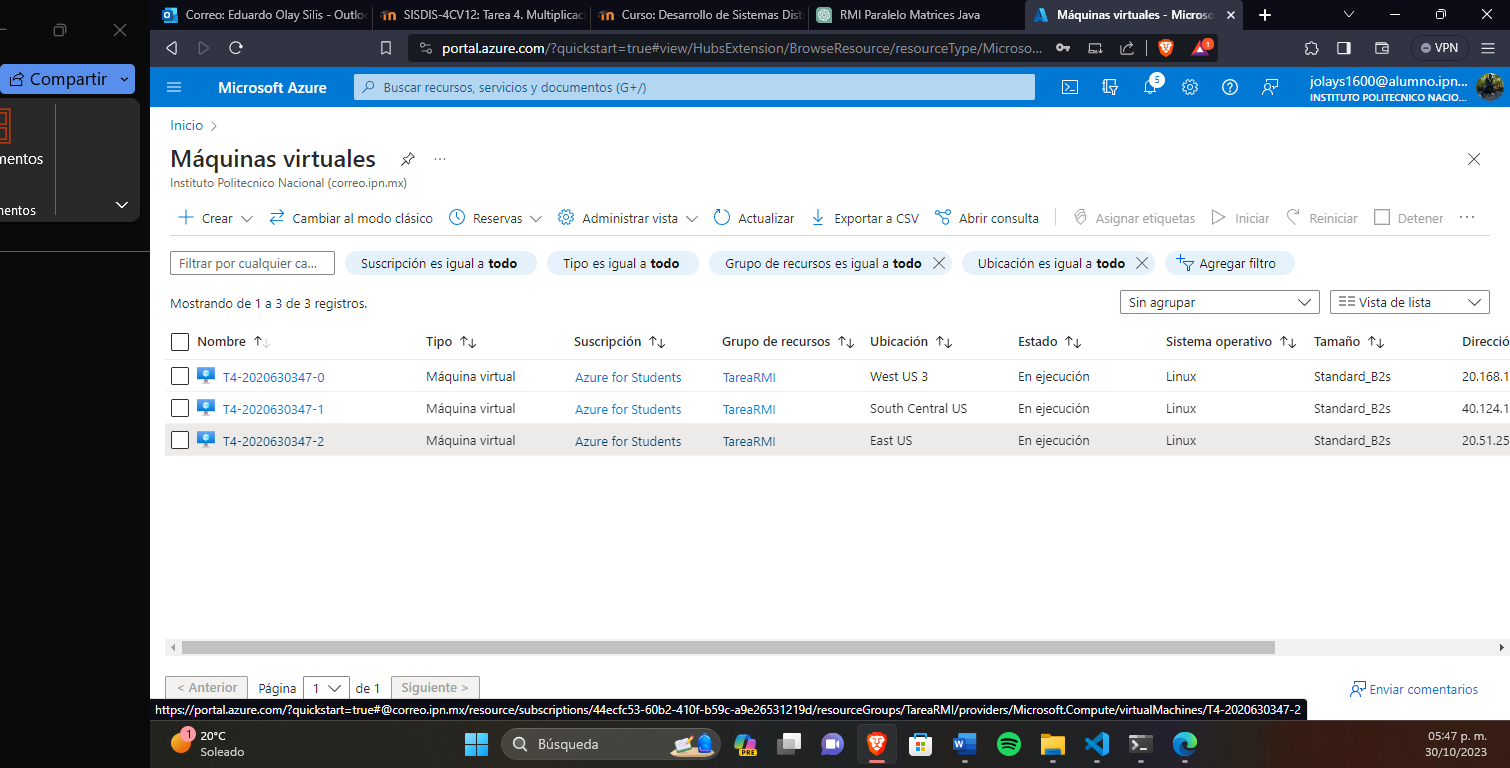
Y le damos en revisar y crear, para que una vez haya finalizado la revisor le demos en crear y esta se pueda visualizar dentro del grupo de recursos.



Una vez observada la Máquina virtual creada le damos en la configuración de red y creamos 2 nuevas reglas de entrada, la primera una del puerto 8080 ya que en el código de nuestra aplicación es el que colocamos y otra regla ICMP para que se pueda realizar una conexión sin problema entre las máquinas virtuales que creemos.



Dicho proceso se repetirá para el Nodo 1 y 2 el quedando al final el grupo de máquinas virtuales de la siguiente manera.



Ahora procedemos a instalar el jdk en cada una de las máquinas virtuales mediante los siguientes comandos:

*sudo add-apt-repository ppa:openjdk-r/ppa*

*sudo apt-get update*

*sudo apt install openjdk-8-jdk-headles*

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Por último, mediante el protocolo sftp le pasamos los archivos necesarios a compilar a la máquina virtual.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Captura de pantalla de computadora

Descripción generada automáticamente

**Fase 4**

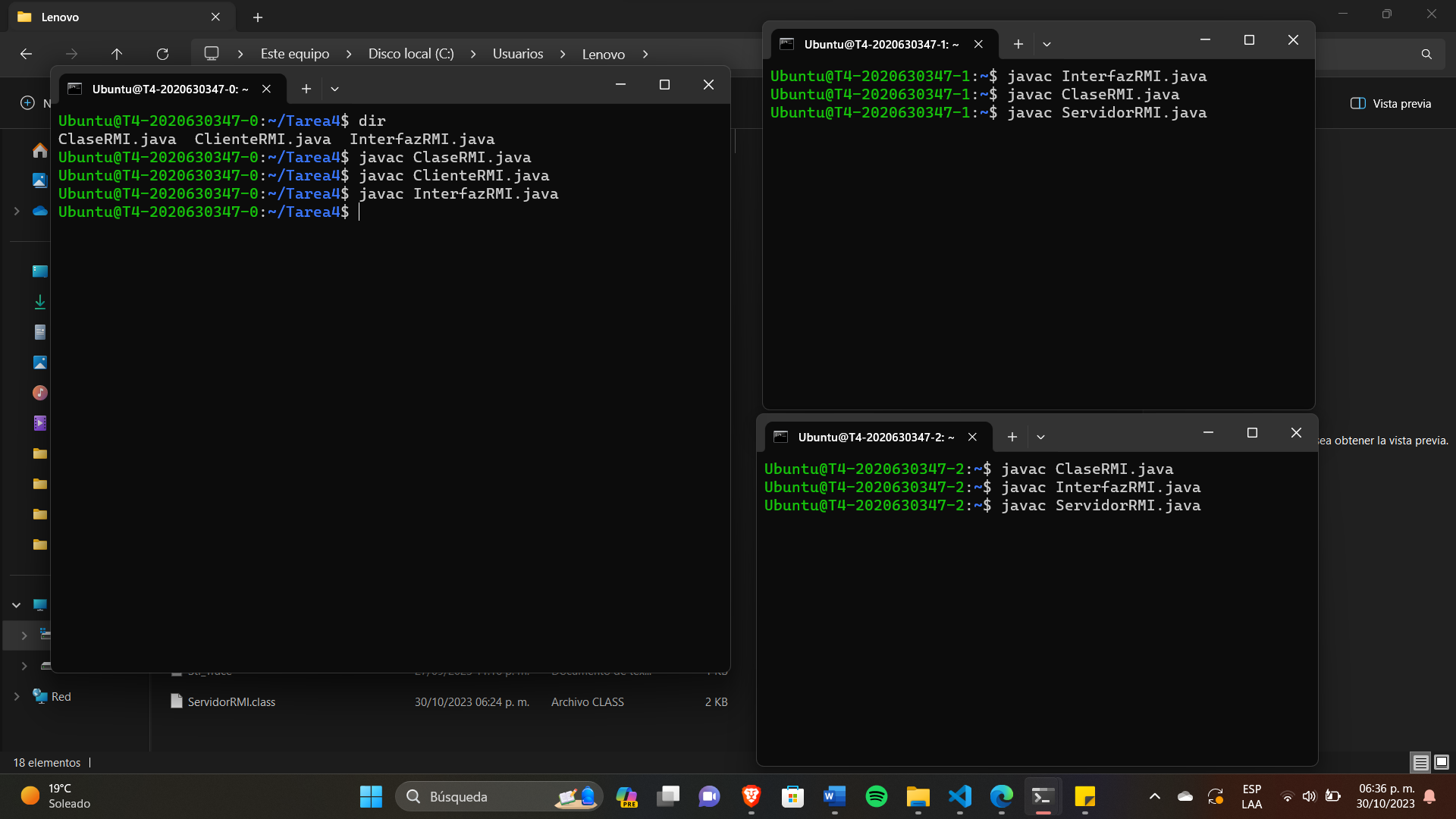
Ya que pasamos los archivos a cada una de las máquinas virtuales mediante el Protocolo sftp, procedemos a realizar el proceso de compilación y ejecución de los programas.

En primera instancia haremos un ping del nodo 0 a cada uno de los nodos 1 y 2 para poder ver que haya conexión entre ellos.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Ya verificando que hay conexión entre las 3 máquinas comenzamos a compilar cada uno de los programas correspondientes.



Ahora ejecutaremos el comando rmregistry para iniciarlo

Interfaz de usuario gráfica, Aplicación

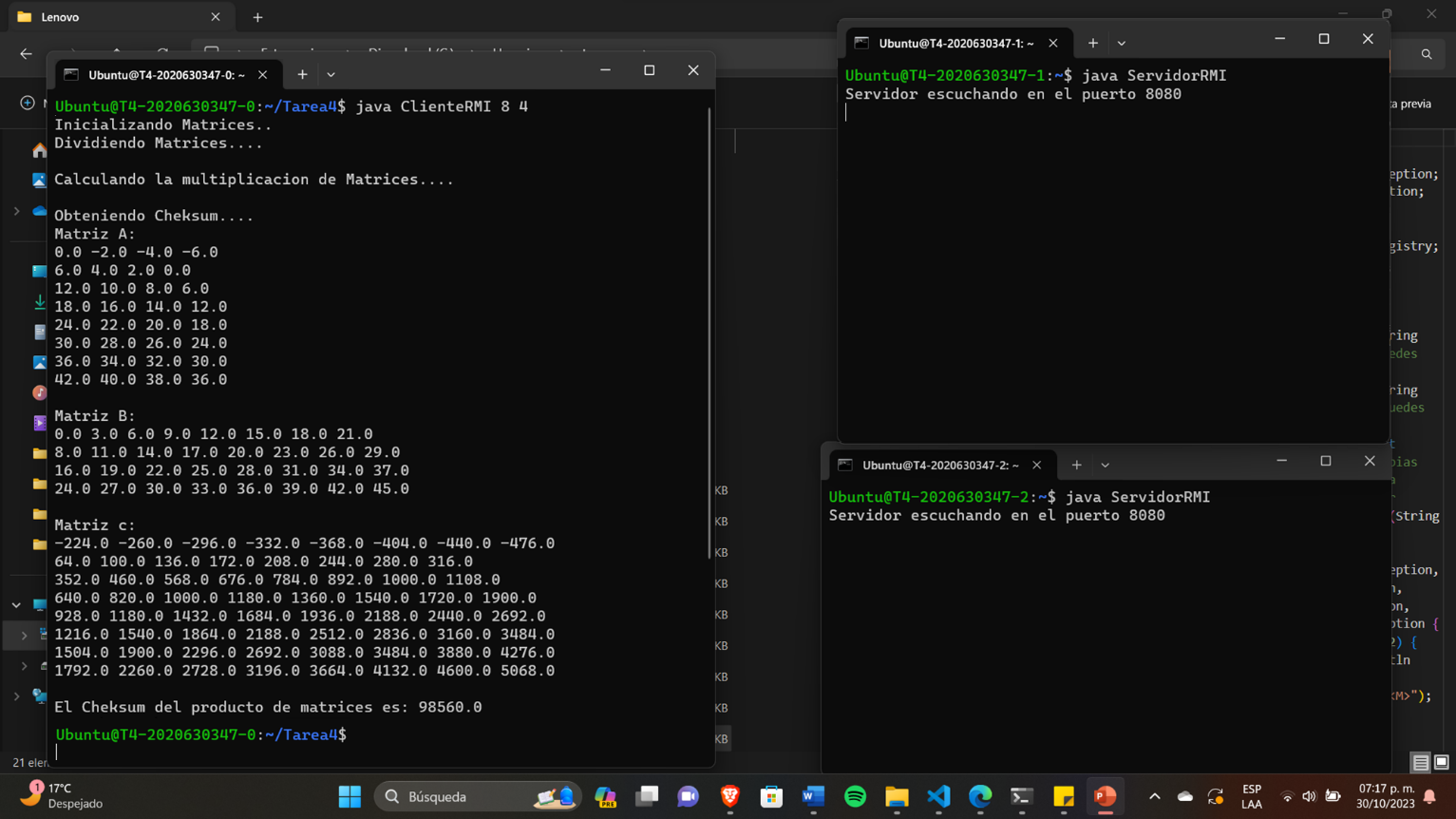
Descripción generada automáticamente

Iniciamos el servidor tanto en el nodo 1 como en el Nodo 2

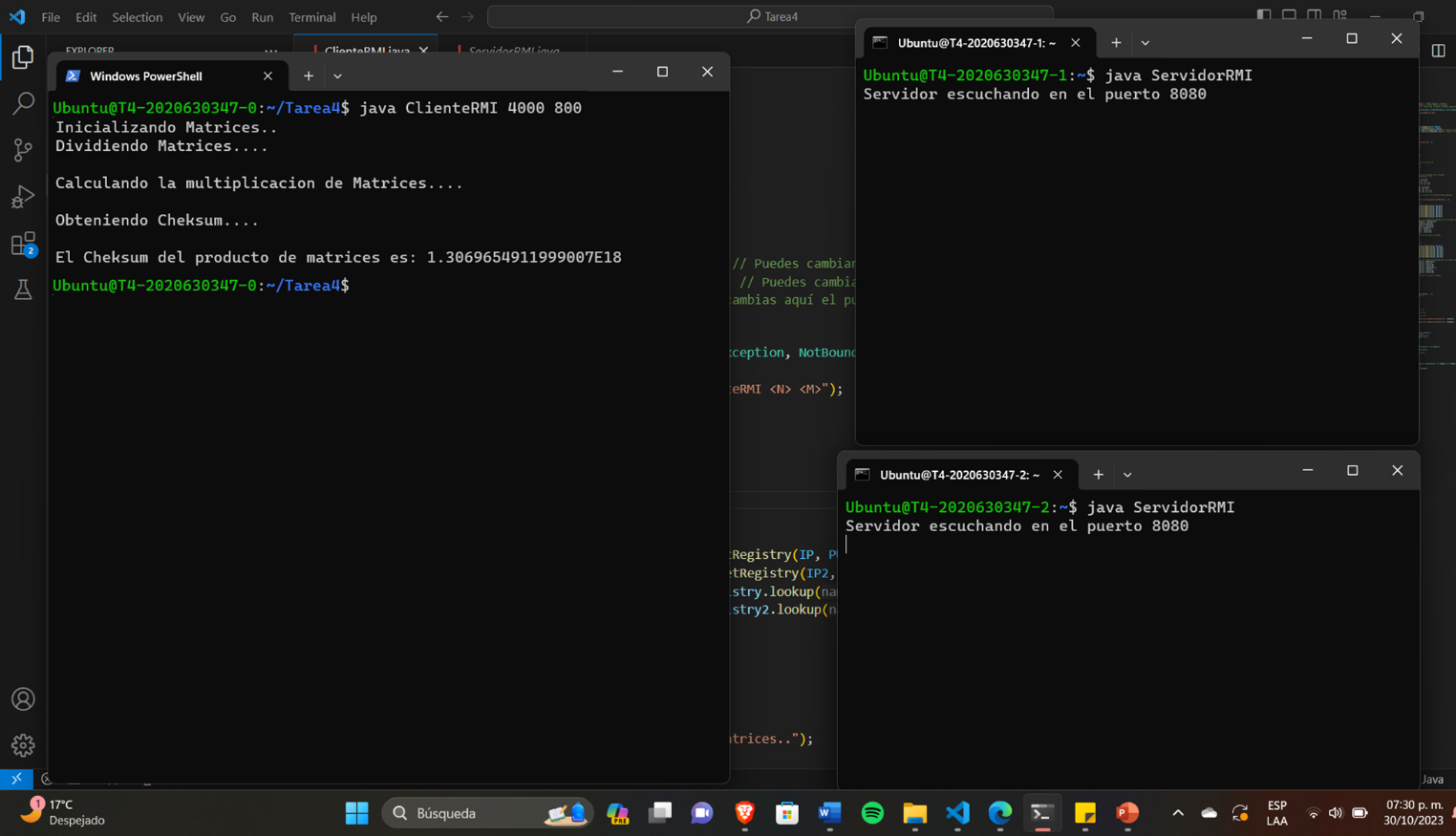
Captura de pantalla de computadora

Descripción generada automáticamente

Ahora inicializamos el cliente con los primeros casos que es con **N=8 y M=4** dando como resultado lo siguiente.



Ahora procedemos a ejecutar el segundo caso donde N=4000 y M=800 saliendo los resultados siguientes:



**Conclusión**

Durante el desarrollo de esta práctica pude comprender más el tema de la transferencia de objetos distribuidos mediante Java RMI el cual es un API el cual invoca métodos remotos. Dicho método que utilizamos fue el de multiplicaMatrices por lo que al momento de desarrollarlo e implementarlo dentro del código, pude comprender un poco más acerca de este y como es que se instancia dentro de un programa.

Algo importante en el desarrollo de esta práctica fue la división de matrices la cual se realizó renglón por renglón reforzando lo visto en la clase de “jerarquía de memoria” utilizando esta implementación pude comprender mejor como es que se iban realizando dichas multiplicaciones y los resultados de estas como irlos acomodando, dependiendo el renglón y la columna.

Considero que esta implementación es un poco más sencilla que la realizada en la tarea 3 el cual fue por paso de mensajes ya que simplemente se puede mandar llamar al método remoto y puedes dejar de preocuparte si se pasó o no bien el mensaje ya que simplemente se manda llamar al método.

Estas y otras herramientas sé que me serán de gran ayuda em un futuro y seguirán complementado mi conocimiento y mis habilidades como programador.