Práctica 4
Programación Distribuida y Tiempo Real

Lucas Di Cunzolo
Santiago Tettamanti

Abstract—JADE

Index Terms—Programación Distribuida y Tiempo Real, java, LaTeX, JADE

1 Punto 1

Programar un agente para que periódicamente recorra una secuencia de computadoras y reporte al lugar de origen

a - El tiempo total del recorrido para recolectar la información
b - La carga de procesamiento de cada una de ellas.
c - La cantidad de memoria total disponible.
d - Los nombres de las computadoras.

Para este punto, se creó un agente que crea 10 containers, y agrega sus ID a un `ArrayList`, agregando al final al container inicial.

Luego simplemente se itera sobre el `ArrayList`, juntando la información del container en el que se encuentra. Para almacenar la información, se creo una clase anidada (`ContainerInfo`). Al conseguir la información, se guarda una instancia de esa clase en otro `ArrayList`.

Al terminar la iteración, nos encontraremos en el container origen, con una lista de instancias de `ContainerInfo`.

Luego se recorre ese array, imprimiendo los valores encontrados.

Ver apéndice de código, Sección Punto 1
Ver apéndice de capturas, Sección Punto 1

Para ejecutar, se debe levantar el `jade.Boot`, esto se puede hacer mediante el comando `make`

Simplemente se puede ejecutar `make start-gui` Luego, para ejecutar el punto, se puede ejcutar `make run-punto1`

2 Punto 2

Programe un agente para que calcule la suma de todos los números almacenados en un archivo de una computadora que se le pasa como parámetro. Comente cómose harílo mismo con una aplicación cliente/servidor. Comente quepasaría si hubiera otros sitios con archivos que deben ser procesados de manera similar

Para este punto, se programó un agente que recibe el nombre de una computadora, y migra el agente. Una vez en la computadora destino, se realiza la lectura del archivo, junto con la suma. Al finalizar, se retora al container origen, y se muestra el resultado.

Ver apéndice de código, Sección Punto 2
Ver apéndice de capturas, Sección Punto 2

Para ejecutar, se debe levantar el `jade.Boot`, esto se puede hacer mediante el comando `make`

Simplemente se puede ejecutar `make start-gui` Luego, para ejecutar el punto, se puede ejcutar `make run-punto2`

3 Punto 3

Defina e implemente con agentes un sistema de archivos distribuido similar al de las prácticas an-

teriores.

Para este punto, se programó un agente que pueda recibir argumentos, y en base a lo recibido, realiza diferentes acciones.

Al ser métodos java, se pudo reutilizar gran parte del la lógica de la práctica anterior.

El agente comprende 4 acciones:

- list: La acción más simple. Recibe un parámetro extra a la acción, que marca cual es el directorio a listar. Una vez parseados los argumentos, se realiza la migración al servidor (recordemos que nos encontramos en el cliente). Al terminar la migración, se leen los archivos del directorio, y se almcenan en una variable de instancia del agente. Luego se vuelve al cliente y se informa lo leído. Ver apéndice de capturas, Sección Punto 3
- read: Recibe 2 argumentos extra a la acción, que marcan el nombre del nuevo archivo y el nombre del archivo remoto (en ese orden) Una vez parseados los argumentos, se realiza la migración al serviodr. Al termionar la migración, se leen los primeros 2000 bytes del archivo, y el tamaño total del archivo a leer y se almacenan en variables de instancia. En el cliente, se realiza la copia de los 2000 bytes, y se consulta si el tamaño total del archivo es menor o igual al tamaño copiado (en la primer vuelta va a ser igual a 2000), de no ser asi, se suman a una variable de `bytes copiados`, y se realiza la migración al servidor. En el servidor se copian los siguientes 2000 bytes `a partir` de los bytes copiados.(Esta acción se repite tantas veces como sea necesaria)
  Ver apéndice de capturas, Sección Punto 3
- write: Trabaja de igual manera que el read, pero a manera inversa, siendo el cliente del item anterior, el servidor, y el servidor del item anterior el cliente. Ver apéndice de capturas, Sección Punto 3
- readwrite: Trabaja de igual manera que ejecutar primer un read y lugo un write de ese archivo. Ver apéndice de capturas, Sección Punto 3

Ver apéndice de código, Sección Punto 3

Para ejecutar, se debe levantar el `jade.Boot`, esto se puede hacer mediante el comando `make`

Simplemente se puede ejecutar `make start-gui`

Luego, para interactuar con el servidor, se cuenta con el script `run`

Para ver la sección de ayuda del comando, se puede utilizar la flag `-h`

APPENDIX

Código Java

.1 Punto 1

```
import jade.core.*;
import jade.util.leap.Serializable;
import jade.wrapper.ContainerController;
import java.util.ArrayList;
import java.util.Iterator;

public class AgentePunto1 extends Agent
{

        private Location origen;
        private long startTime;
        private ArrayList<String> containers  = new
           ArrayList<String>();
        private int index;
        private ArrayList<ContainerInfo> info = new
           ArrayList<ContainerInfo>();
        // Ejecutado por unica vez en la creacion
        public void setup()
        {
                this.origen = here();
                System.out.println("\n\nHola, agente con nombre local
                   " + getLocalName());
                System.out.println("Y nombre completo... " +
                   getName());
                System.out.println("Y en location " +
                   this.origen.getID() + "\n\n");

                //Creacion de los containers
                for (int i=0; i<10;i++){
                        String containerName="Contenedor-"+i;
                        if (!containerName.equals(origen.getName())) {
                                createContainer(containerName);
                                containers.add(containerName);
                        }
                }
                //agrego el origen para que sea el final de la cadena
                containers.add(this.origen.getName());

                try {
                        index=0;
                        // Para migrar el agente
                        ContainerID destino = new
                           ContainerID(containers.get(index++), null);
```

```
                System.out.println("Migrando el agente a " +
                   destino.getID());
                startTime = System.currentTimeMillis();
                doMove(destino);
        } catch (Exception e) {
                System.out.println("\n\n\nNo fue posible
                   migrar el agente\n\n\n");}
}

// Ejecutado al llegar a un contenedor como resultado de una
   migracion
protected void afterMove()
{
        Location actual = here();
        if (actual.getName().equals(origen.getName())){
                this.originAction();
        }
        else{
                this.remoteAction(actual);
        }
}

private void remoteAction(Location actual) {
        long startContainerTime =System.currentTimeMillis();
        ContainerInfo currentContainerInfo= new
           ContainerInfo();
        System.out.println("\n\nHola, agente migrado con
           nombre local " + getLocalName());
        System.out.println("Y nombre completo... " +
           getName());
        currentContainerInfo.setFreeMemory(java.lang.Runtime.getRuntime
        currentContainerInfo.setName(actual.getName());
        try {
                ContainerID destino = new
                   ContainerID(containers.get(index++), null);
                System.out.println("Migrando el agente a " +
                   destino.getID());
                info.add(currentContainerInfo);
                long
                   finishContainerTime=System.currentTimeMillis()
                   - startContainerTime;
                currentContainerInfo.setProcessingTime(finishContainer
                doMove(destino);
        } catch (Exception e) {
                System.out.println("\n\n\nNo fue posible
                   migrar el agente\n\n\n");
        }
}
```

```
private void originAction() {
        long finishTime=System.currentTimeMillis() - startTime;
        System.out.println("\n\n----------------------------------------
        System.out.println("Termine la vuelta, tiempo total:
           "+ finishTime +" milisegundos");

        for (ContainerInfo containerInfo : info) {
                System.out.println("\n\nInformacion container
                   " + containerInfo.getName() +" :");
                System.out.println("    Memoria libre: "+
                   (containerInfo.getFreeMemory()/1024)/1024
                   +"Mb");
                System.out.println("    Tiempo de
                   procesamiento: "+
                   containerInfo.getProcessingTime() +"
                   milisegundos");
                float processingPercentage =
                   (containerInfo.getProcessingTime()*100)/finishTime;
                System.out.println("    Carga de
                   procesamiento: "+ processingPercentage
                   +"%");
        }
}

protected ContainerController createContainer(String name){
        //Get the JADE runtime interface (singleton)
        jade.core.Runtime runtime =
           jade.core.Runtime.instance();
        //Create a Profile, where the launch arguments are
           stored
        Profile profile = new ProfileImpl();
        profile.setParameter(Profile.CONTAINER_NAME, name);
        profile.setParameter(Profile.MAIN_HOST, "localhost");
        //create a non-main agent container
        return runtime.createAgentContainer(profile);
}

public class ContainerInfo implements Serializable {

        private long freeMemory;
        private String name;
        private long processingTime;

        public long getFreeMemory() {
                return freeMemory;
        }
        public void setFreeMemory(long freeMemory) {
                this.freeMemory = freeMemory;
        }
```

```
                public String getName() {
                        return name;
                }
                public void setName(String name) {
                        this.name = name;
                }
                public long getProcessingTime() {
                        return processingTime;
                }
                public void setProcessingTime(long processingTime) {
                        this.processingTime = processingTime;
                }
        }
}
```

.2 Punto 2

```
import java.nio.charset.Charset;
import java.nio.file.Files;

import jade.core.*;
import java.io.IOException;
import java.util.List;
import java.nio.file.Paths;

public class AgentePunto2 extends Agent
{
    private final String machine = "Main-Container";
    private Location origen = null;
    private int suma;
    public void setup()
    {
        this.origen = here();
        System.out.println("\n\nHola, agente con nombre local " +
           getLocalName());
        System.out.println("Y nombre completo... " + getName());
        System.out.println("Y en location " + origen.getID() + "\n\n");
    try {
        ContainerID destino = new ContainerID(this.machine, null);
        System.out.println("Migrando el agente a " + destino.getID());
        doMove(destino);
    } catch (Exception e) {
        System.out.println("\n\n\nNo fue posible migrar el
           agente\n\n\n");}
    }

    protected void afterMove()
    {
        Location actual = here();
```

```
        if (!actual.getName().equals(this.origen.getName())) {
            try {
                List<String> numbers =
                   Files.readAllLines(Paths.get("/tmp/file"),
                   Charset.forName("utf8"));
                int result = 0;
                for (String number: numbers) {
                    result += Integer.parseInt(number);
                }
                suma = result;
            } catch(NumberFormatException e) {
                System.out.println("Solo se admiten numeros");
            } catch(IOException e) {
                System.out.println("El archivo no existe");
            } catch(Exception e) {
                System.out.printf("Algo salio mal\n\n%s",
                   e.getMessage());
            }

            ContainerID destino = new
               ContainerID(this.origen.getName(), null);
            doMove(destino);
        } else {
            System.out.printf("La suma es: %d\n", this.suma);
        }
    }
}
```

.3 Punto 3

```
import jade.core.*;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.nio.file.DirectoryStream;
import java.util.Arrays;
import java.util.concurrent.TimeUnit;
import java.nio.file.StandardOpenOption;
import java.io.IOException;
import java.io.InputStream;
import java.io.FileInputStream;

public class AgentePunto3 extends Agent
{
    private String action;
    private String sourcePath;
    private String destinationPath;
    private byte[] file = null;
```

```
private String files;
private Location origen;
private long fileSize;
private int actualSize = 0;

public void setup()
{
    //Chequeo parametros y seteo variables
    Object[] args = getArguments();
    this.checkAndSetArguments(args);

    //informacion del agente
    this.origen = here();

try {
    //migracion a otro container
    ContainerID destino = new ContainerID("Main-Container", null);
    System.out.println("Migrando el agente a " + destino.getID());
    doMove(destino);
} catch (Exception e) {
    System.out.println("\n\n\nNo fue posible migrar el
       agente\n\n\n: " + e);}
}

protected void afterMove()
{
    //Muestro info del container actual
    Location actual = here();
    try{
        switch (this.action) {
            case "write":
                if(!actual.getName().equals(this.origen.getName())){
                    this.write(this.destinationPath,this.file);
                    doMove(new ContainerID(this.origen.getName(),
                       null));
                } else {
                    if (this.fileSize > this.actualSize) {
                        this.file = this.read(this.sourcePath,
                           this.actualSize);
                        System.out.printf("Se leyeron %d de %d
                           bytes\n\n", this.file.length,
                           this.fileSize);
                        doMove(new ContainerID("Main-Container",
                           null));
                    } else {
                        System.out.println("El archivo
                           "+this.sourcePath+" se escribio
                           correctamente en el directorio remoto
                           "+this.destinationPath);
```

```
                }
            }
            break;
        case "read":
            if(!actual.getName().equals(this.origen.getName())){
                this.file = this.read(this.destinationPath,
                   this.actualSize);
                System.out.printf("Leyendo %d bytes de %d\n",
                   this.file.length, this.fileSize);
                doMove(new ContainerID(this.origen.getName(),
                   null));
            }
            else{
                try{
                    Files.write(Paths.get(this.sourcePath),
                       this.file,StandardOpenOption.APPEND);
                    System.out.printf("Escribiendo %d bytes de
                       %d\n", this.file.length, this.fileSize);
                }
                catch (IOException e) {
                    Files.createFile(Paths.get(this.sourcePath));
                    Files.write(Paths.get(this.sourcePath),
                       this.file,StandardOpenOption.APPEND);

                } finally {
                    if (this.fileSize > this.actualSize) {
                        System.out.println("Faltaron bytes");
                        doMove(new
                           ContainerID("Main-Container",
                           null));
                    } else {
                        System.out.println("\n\n----------------------
                        System.out.println("El archivo
                           "+this.destinationPath+" se leyo
                           correctamente y se guardo en
                           "+this.sourcePath);
                    }
                }
            }
            break;
        case "rw":
        case "readwrite":
            if(!actual.getName().equals(this.origen.getName())){
                this.file = this.read(this.destinationPath,
                   this.actualSize);
                System.out.printf("Leyendo %d bytes de %d\n",
                   this.file.length, this.fileSize);
                doMove(new ContainerID(this.origen.getName(),
                   null));
```

```
                }
                else{
                    try{
                        Files.write(Paths.get(this.sourcePath),
                           this.file,StandardOpenOption.APPEND);
                        System.out.printf("Escribiendo %d bytes de
                           %d\n", this.file.length, this.fileSize);
                    }
                    catch (IOException e) {
                        Files.createFile(Paths.get(this.sourcePath));
                        Files.write(Paths.get(this.sourcePath),
                           this.file,StandardOpenOption.APPEND);

                    } finally {
                        if (this.fileSize > this.actualSize) {
                            System.out.println("Faltaron bytes");
                            doMove(new
                               ContainerID("Main-Container",
                               null));
                        } else {
                            System.out.println("\n\n----------------------
                            System.out.println("El archivo
                               "+this.destinationPath+" se leyo
                               correctamente y se guardo en
                               "+this.sourcePath);
                            this.actualSize = 0;
                            this.file = this.read(this.sourcePath,
                               this.actualSize);
                            this.action="write";
                            this.destinationPath+="-copia";
                            doMove(new
                               ContainerID("Main-Container",
                               null));
                        }
                    }
                }
                break;
            case "list":
            case "ls":
                if(!actual.getName().equals(this.origen.getName())){
                    this.files = this.list(this.destinationPath);
                    doMove(new ContainerID(this.origen.getName(),
                       null));
                } else {
                    System.out.println(this.files);
                }
                break;
        }
    }catch (IOException e) {System.out.println(e);}
```

```
}

private byte[] read(String path, int position)
{
    try {
        int chunck = 2000;
        int noBytes = ((int)this.fileSize - this.actualSize) <
           chunck ? (int)(this.fileSize - this.actualSize) :
           chunck;
        System.out.printf("Leyendo %d bytes a partir de %d\n",
           noBytes, this.actualSize);
        InputStream in = new FileInputStream(path);
        byte[] contents = new byte[noBytes];
        in.skip(this.actualSize);
        in.read(contents, 0, noBytes);
        this.actualSize += contents.length;
        return contents;
    } catch(IOException e) {
        System.out.println(e);
        return new byte[0];
    }
}

private int write(String path,byte[] data)
{
    try {
        try{
            Files.write(Paths.get(path),
               data,StandardOpenOption.APPEND);
        }
        catch (IOException e) {
            Files.createFile(Paths.get(path));
            Files.write(Paths.get(path),
               data,StandardOpenOption.APPEND);
        }
        System.out.printf("Se escribieron %d de %d bytes\n",
           this.actualSize, this.fileSize);
        return data.length;
    } catch (IOException e) {
        System.out.println(e.toString());
        return -1;
    }
}

private String list(String path)
{
    String directoryPaths = "";
    try (DirectoryStream<Path> paths =
```

```
        Files.newDirectoryStream(Paths.get(path))) {
          System.out.printf("Listing files in %s\n", path);
          for (Path p : paths) {
              directoryPaths += p.toString();
              directoryPaths += "\n";
          }
      } catch(Exception e) {
          System.out.println(e);
      }
      return directoryPaths;
  }

  private void checkAndSetArguments(Object[] args){
      try{
          switch ((String) args[0]) {
              case "write":
              case "read":
              case "rw":
              case "readwrite":
                      if (args.length != 3)
                      {
                          System.out.println("3 argument needed:
                             command , local directory and remote
                             directory");
                          System.exit(1);
                      }
                      else {
                          this.action          = (String) args[0];
                          this.sourcePath      = (String) args[1];
                          this.destinationPath = (String) args[2];
                          if (this.action.equals("write")) {
                              this.file =
                                 this.read(this.destinationPath,
                                 this.actualSize);
                              this.fileSize =
                                 Files.size(Paths.get(this.sourcePath));
                          } else {
                              this.fileSize =
                                 Files.size(Paths.get(this.destinationPath));
                          }
                      }
                      break;
              case "list":
              case "ls":
                      if (args.length != 2)
                      {
                          System.out.println("2 argument needed:
                             command and directory");
                          System.exit(1);
```

```
                        }
                        else {
                            this.action          = (String) args[0];
                            this.destinationPath = (String) args[1];
                        }
                        break;

                default: System.out.printf("Command %s unavailable\n",
                   (String) args[0]);
                        System.exit(1);
                        break;
            }
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

.4 Capturas

.4.1 Punto 1

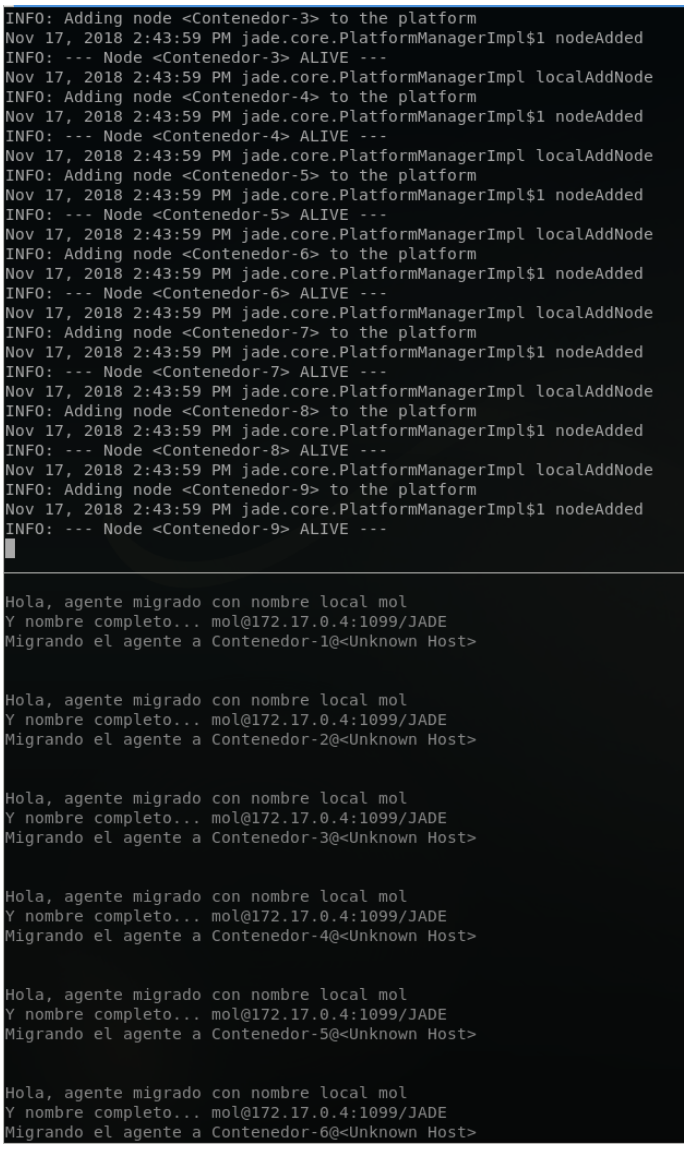

Fig. 1. Migraciones

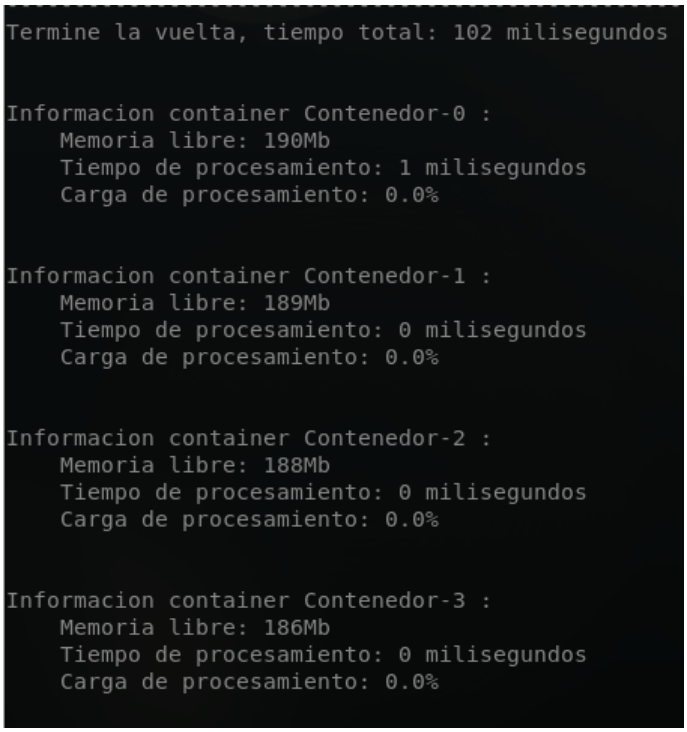

Fig. 2. Muestra de información

.4.2 Punto 2

Fig. 3. Archivo a sumar

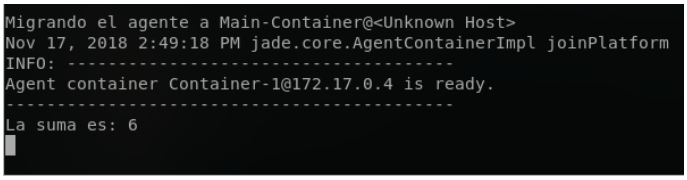

Fig. 4. Muestra de la suma

.4.3 Punto 3

```
Listing files in /tmp

INFO: Service jade.core.management.AgentManagement initialized
Nov 17, 2018 3:32:32 PM jade.core.BaseService init
INFO: Service jade.core.messaging.Messaging initialized
Nov 17, 2018 3:32:32 PM jade.core.BaseService init
INFO: Service jade.core.resource.ResourceManagement initialized
Nov 17, 2018 3:32:32 PM jade.core.BaseService init
INFO: Service jade.core.mobility.AgentMobility initialized
Nov 17, 2018 3:32:32 PM jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
Nov 17, 2018 3:32:32 PM jade.core.AgentContainerImpl startBootstrapAgents
SEVERE: Cannot create agent rma: Name-clash Agent rma@172.17.0.4:1099/JADE already present in the platform
Migrando el agente a Main-Container@<Unknown Host>
Nov 17, 2018 3:32:32 PM jade.core.AgentContainerImpl joinPlatform
INFO: ------------------------------------
Agent container Container-1@172.17.0.4 is ready.
------------------------------------------
/tmp/file.pdf
/tmp/passwd
/tmp/.X11-unix
/tmp/file
/tmp/.X1-lock
/tmp/hsperfdata_root
/tmp/.xfsm-ICE-HPTPSZ
/tmp/.xfsm-ICE-YOTPSZ
/tmp/file-output.pdf
/tmp/.ICE-unix
```

Fig. 5. Listado

```
root@c230d960a60c:/pdytr/Entrega/punto3# ls -l /tmp/file.pdf
-rw-r--r-- 1 root root 102046 Nov 17 13:29 /tmp/file.pdf
```

Fig. 6. Archivo para las pruebas

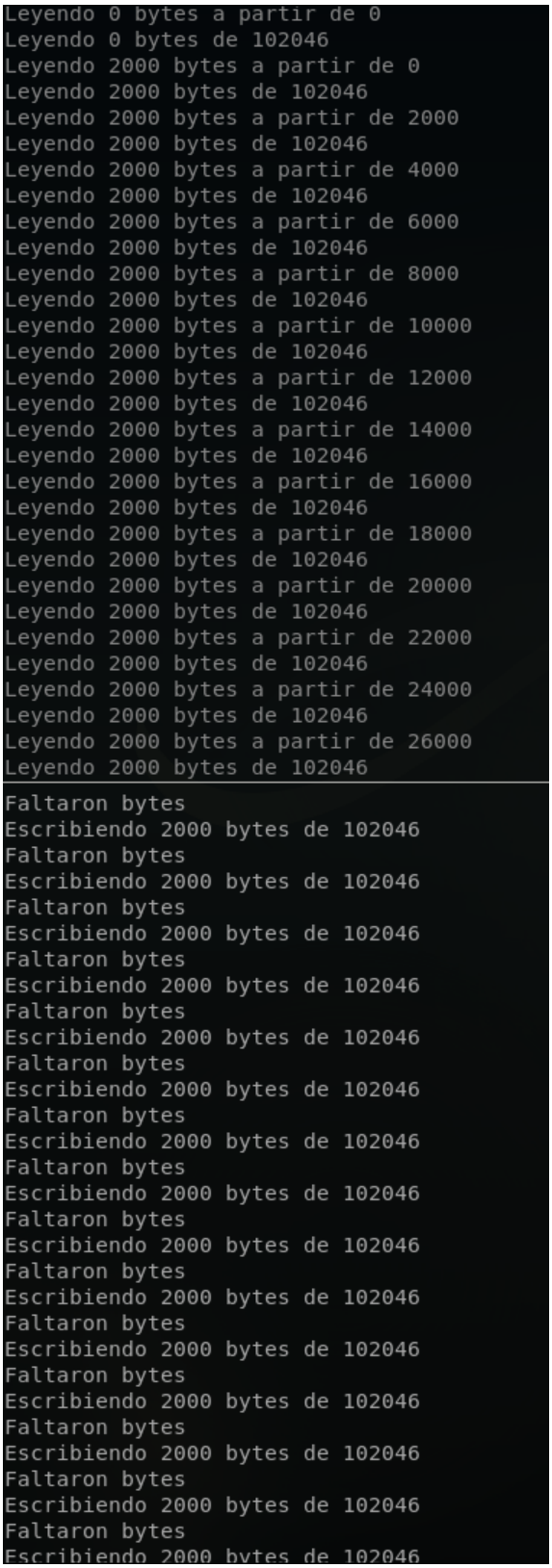

Fig. 7. Primera parte de la lectura

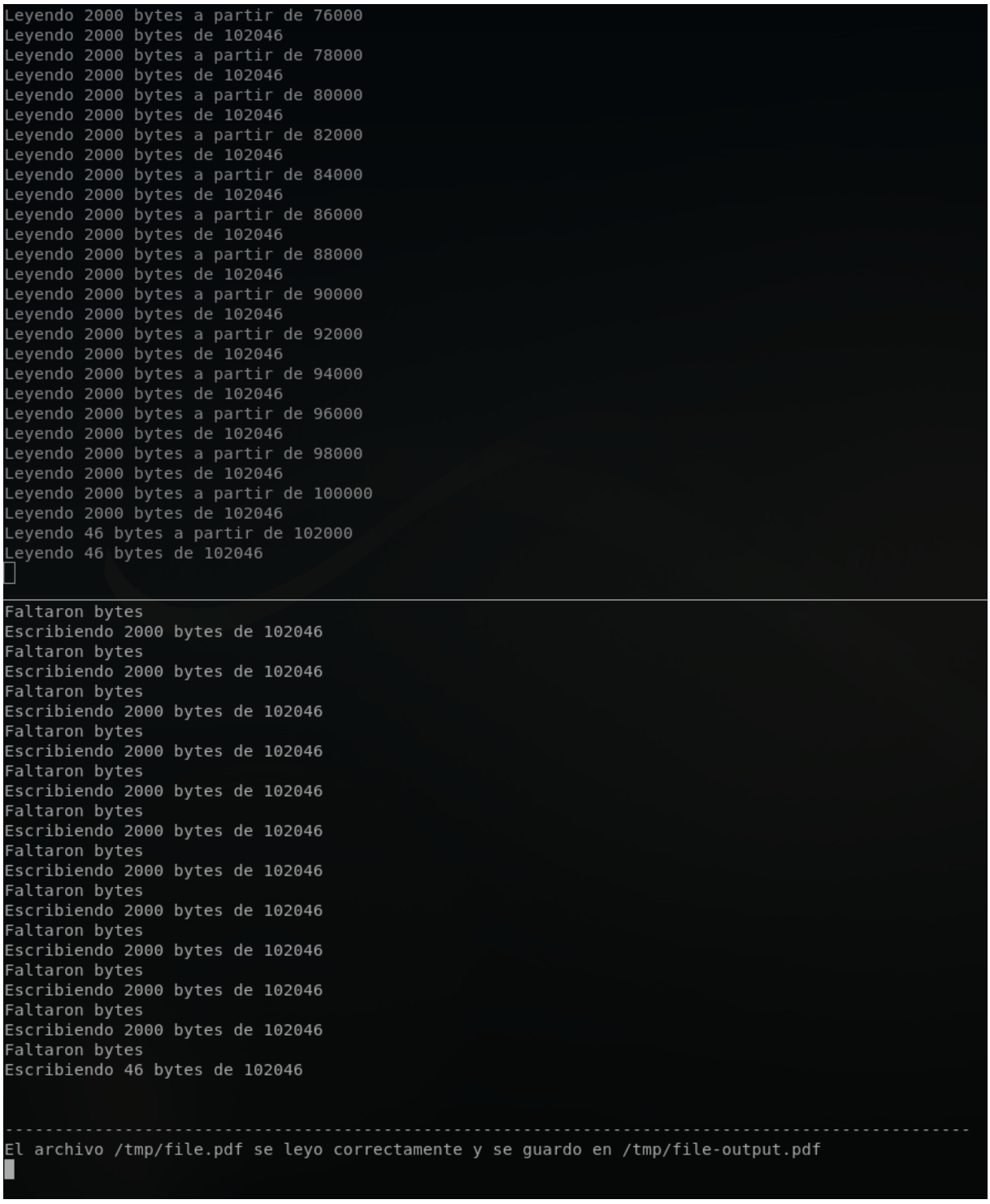

Fig. 8. Fin de la lectura

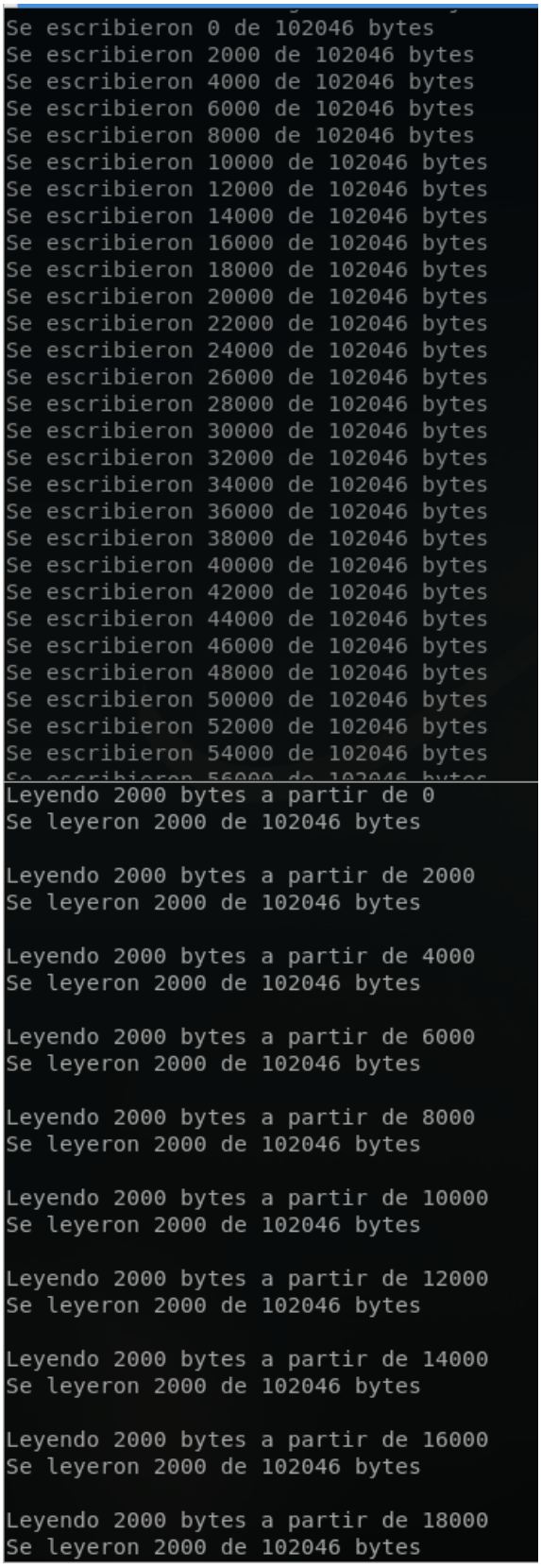

Fig. 9. Primera parte de la escritura

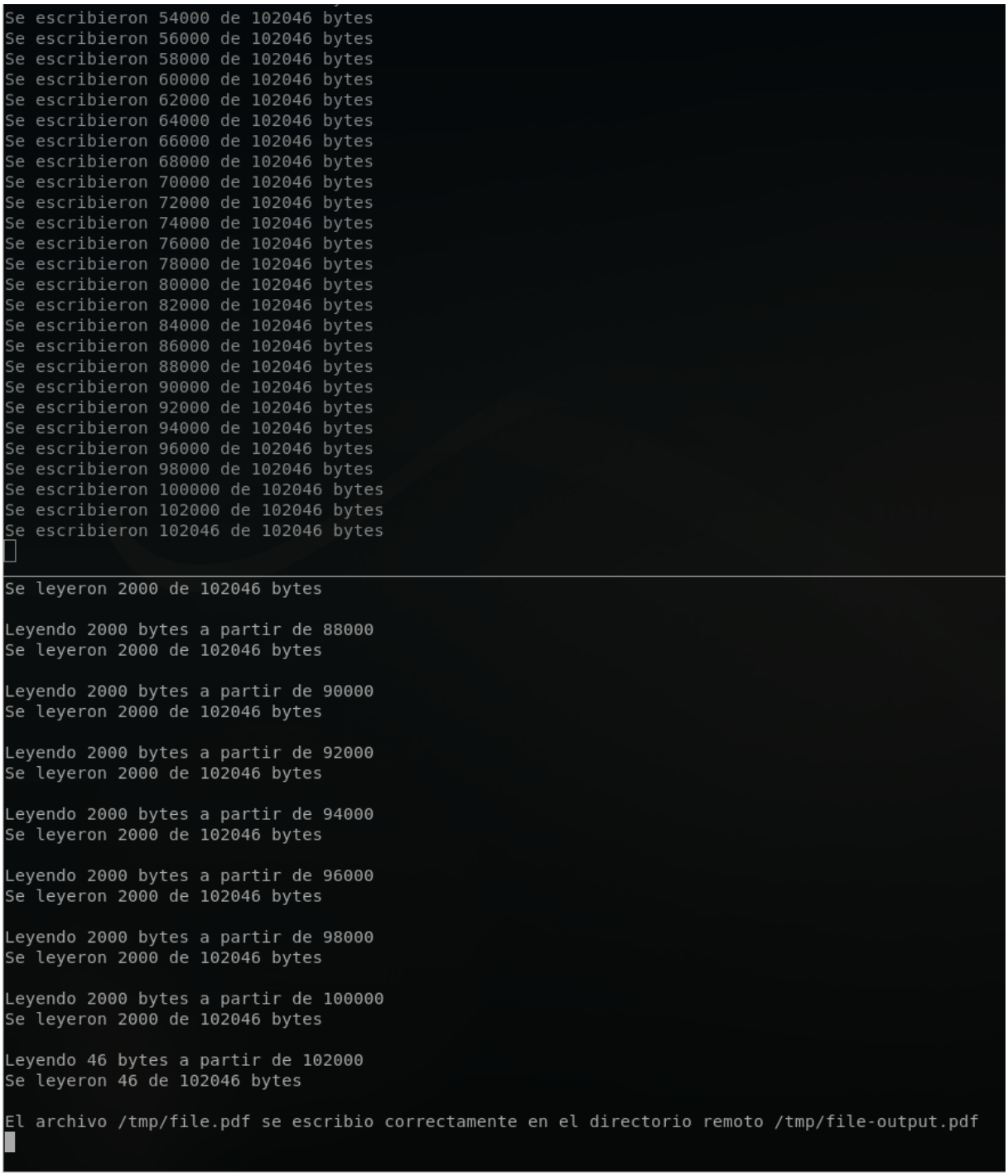

Fig. 10. Fin de la escritura

```
java -cp lib/jade.jar:classes jade.Boot -gui
Leyendo 2000 bytes a partir de 0
Leyendo 2000 bytes de 1023179
Leyendo 2000 bytes a partir de 2000
Leyendo 2000 bytes de 1023179
Leyendo 2000 bytes a partir de 4000
Leyendo 2000 bytes de 1023179
Leyendo 2000 bytes a partir de 6000
Leyendo 2000 bytes de 1023179
Leyendo 2000 bytes a partir de 8000
Leyendo 2000 bytes de 1023179
Leyendo 2000 bytes a partir de 10000
Leyendo 2000 bytes de 1023179
Leyendo 2000 bytes a partir de 12000
Leyendo 2000 bytes de 1023179
Leyendo 2000 bytes a partir de 14000
Leyendo 2000 bytes de 1023179
Leyendo 2000 bytes a partir de 16000
Leyendo 2000 bytes de 1023179
Leyendo 2000 bytes a partir de 18000
Leyendo 2000 bytes de 1023179
Leyendo 2000 bytes a partir de 20000
Leyendo 2000 bytes de 1023179
Leyendo 2000 bytes a partir de 22000
Leyendo 2000 bytes de 1023179
Leyendo 2000 bytes a partir de 24000
Leyendo 2000 bytes de 1023179
:
```

```
Faltaron bytes
Escribiendo 2000 bytes de 1023179
Faltaron bytes
Escribiendo 2000 bytes de 1023179
Faltaron bytes
Escribiendo 2000 bytes de 1023179
Faltaron bytes
Escribiendo 2000 bytes de 1023179
Faltaron bytes
```

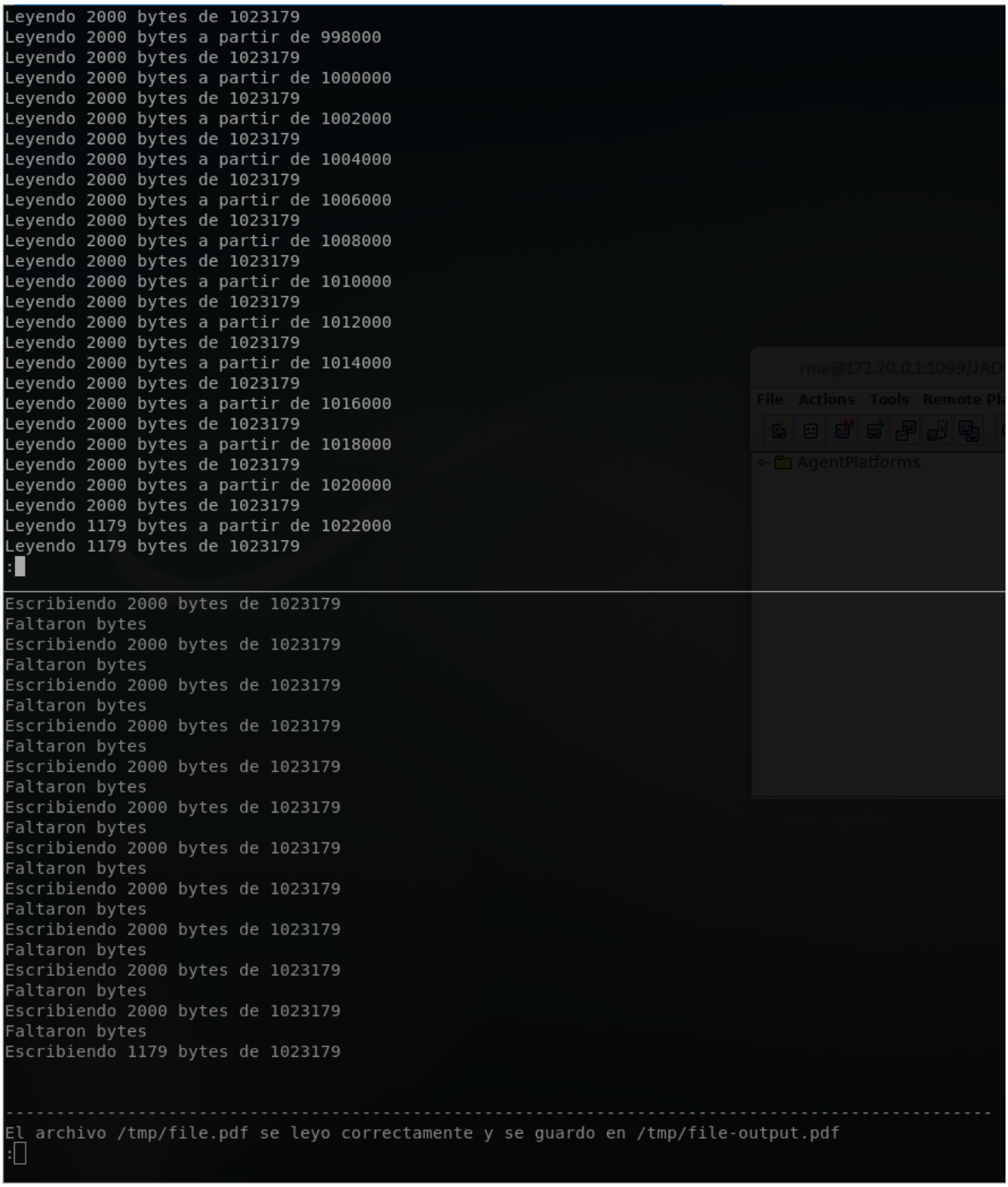

Fig. 12. Fin de la lectura

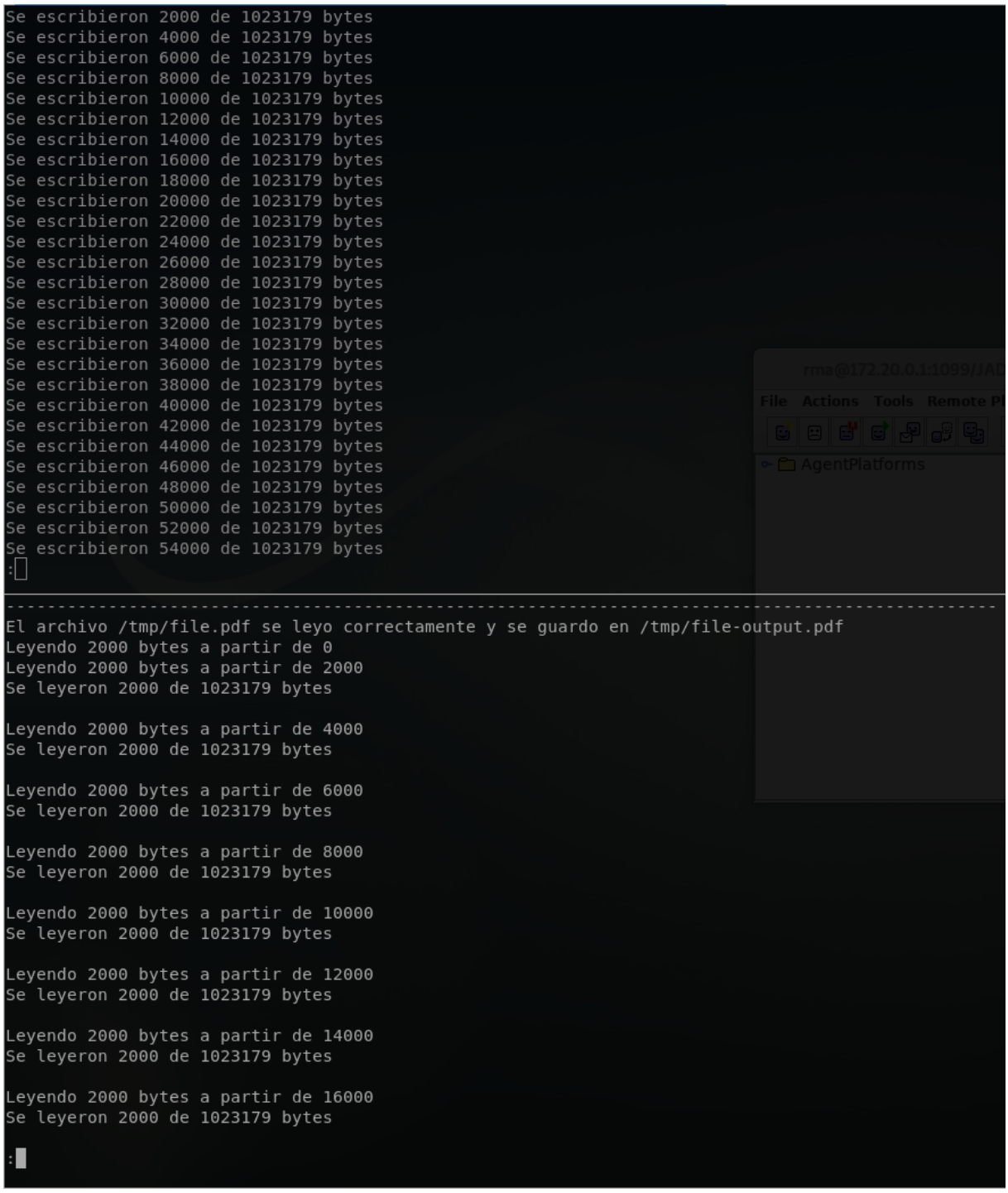

Fig. 13. Inicio de la escritura (backup)

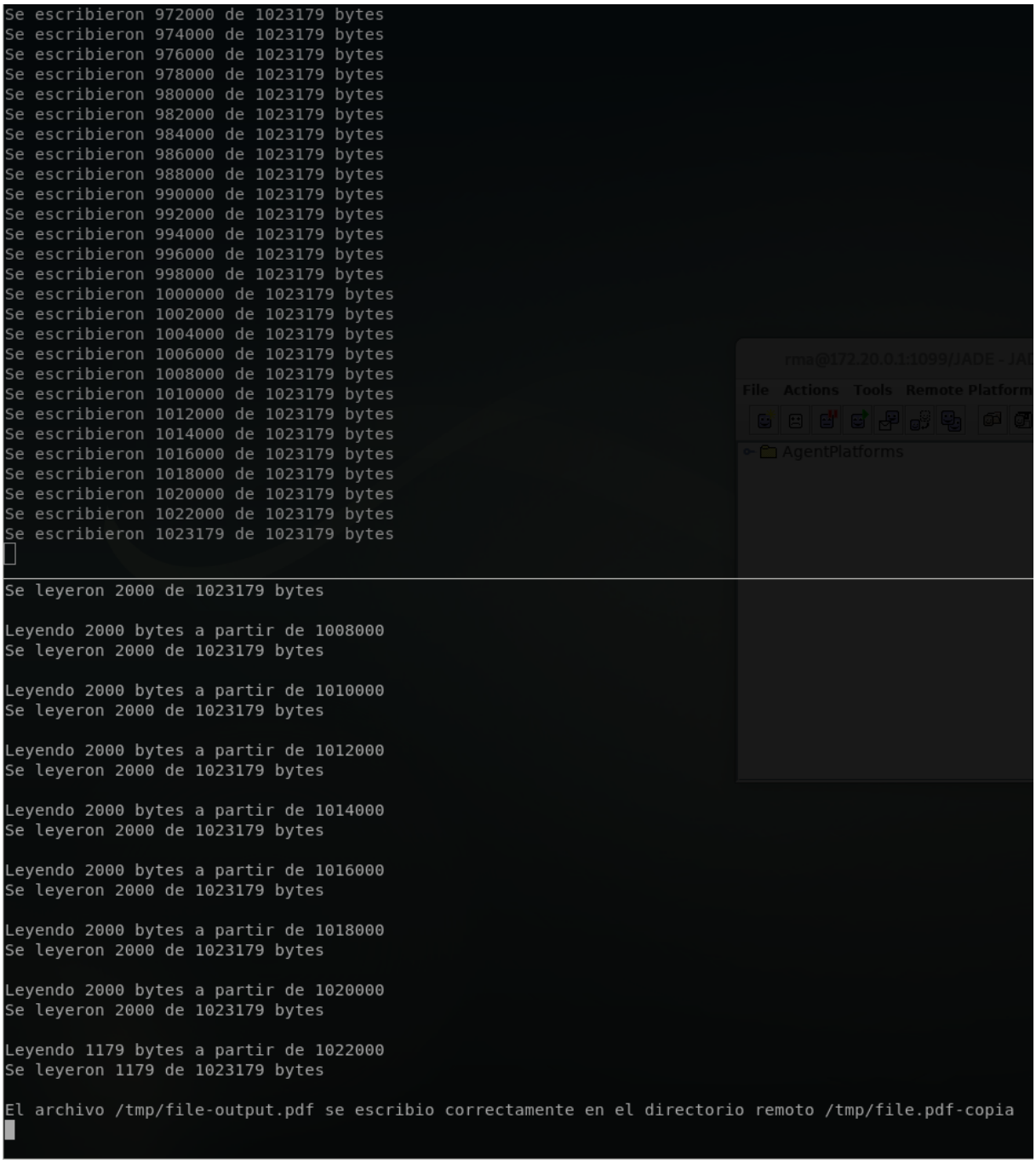

Fig. 14. Fin de la escritura (backup)

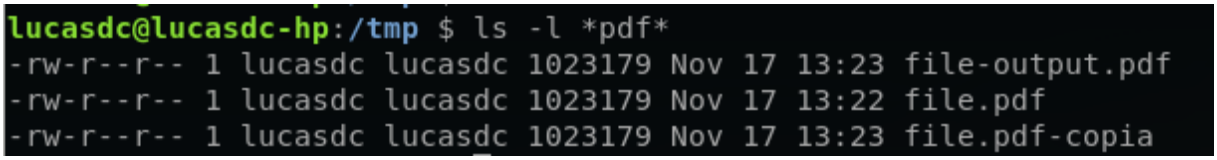

Fig. 15. Estado final de los archivos

Fig. 16. Diff final entre archivos

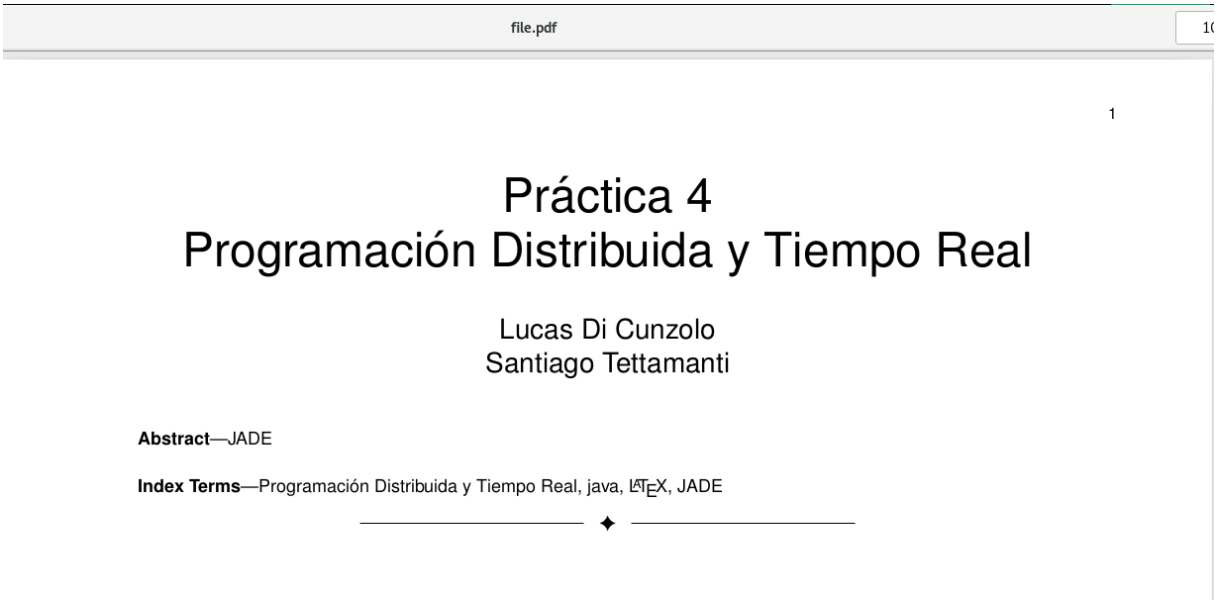

Fig. 17. Apertura archivo original

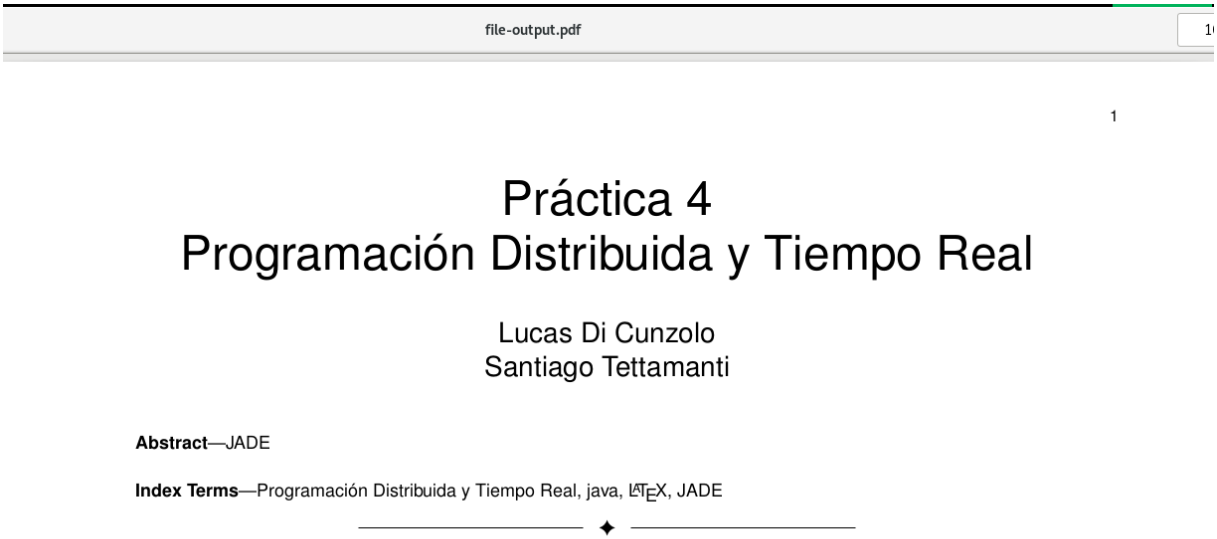

Fig. 18. Apertura archivo copia

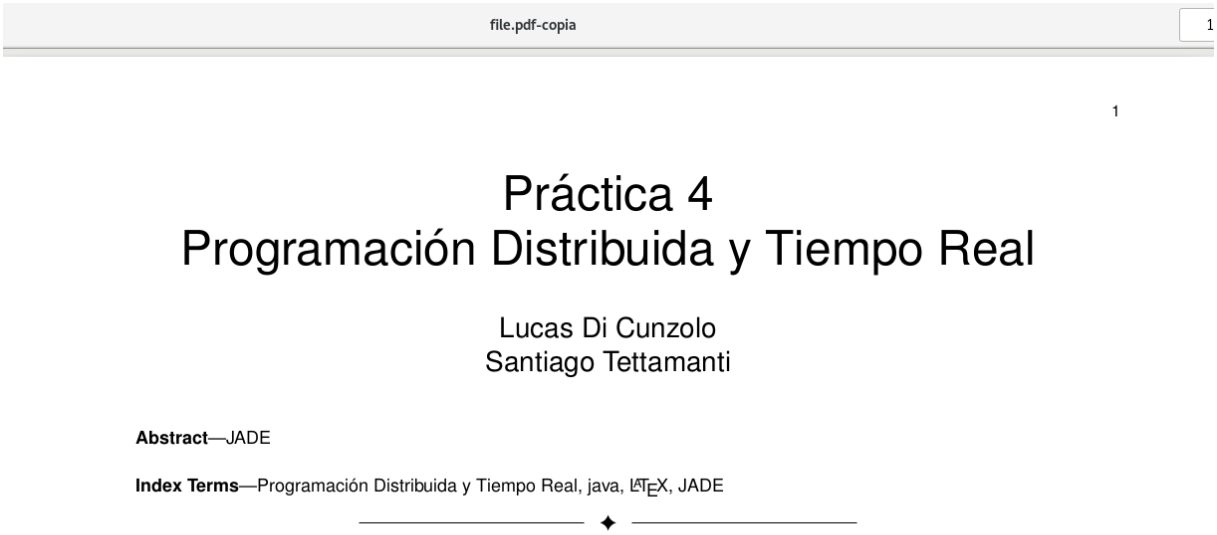

Fig. 19. Apertura archivo backup