Practica 4

Inti María Tidball 17612/3
Juan Pablo Sanchez Magariños 13238/3

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.

   Comente la relación entre este posible estado del sistema distribuido y el estado que se obtendría implementando el algoritmo de instantánea.

Se creó un agente **`MonitoringAgent`** que crea 10 containers y agrega sus IDs a una lista con el fin de recordarlos. Luego, itera sobre dicha lista recolectando la información que encuentra. Para almacenar la información crea la clase **`ContainerInfo`**, dentro de la cual se almacena la información en una nueva lista.
Por último, se recorre la última lista imprimiendo toda la información solicitada.

Nota: Para compilar y ejecutar todos los ejercicios de esta práctica es necesario obtener y colocar el `.jar` de JADE en el directorio `lib/`. No se incluye en la entrega como pedía la consigna. Para cada ejercicio, tenemos 3 scripts, para compilar, correr la gui y ejecutar los agentes, los cuales explicamos en cada punto.

En este caso el código se encuentra en el directorio **`1/`**

Instrucciones para compilar (**`./1.compilacion.sh`**):

```
javac -classpath ../lib/jade.jar -d ./classes ./myexamples/MonitoringAgent.java
```

Instrucciones para ejecutarlo:
Terminal 1 (**`./2.gui.sh`**)

```
java -cp ../lib/jade.jar:classes jade.Boot -gui
```

Terminal 2 (**`./3.agente.sh`**)

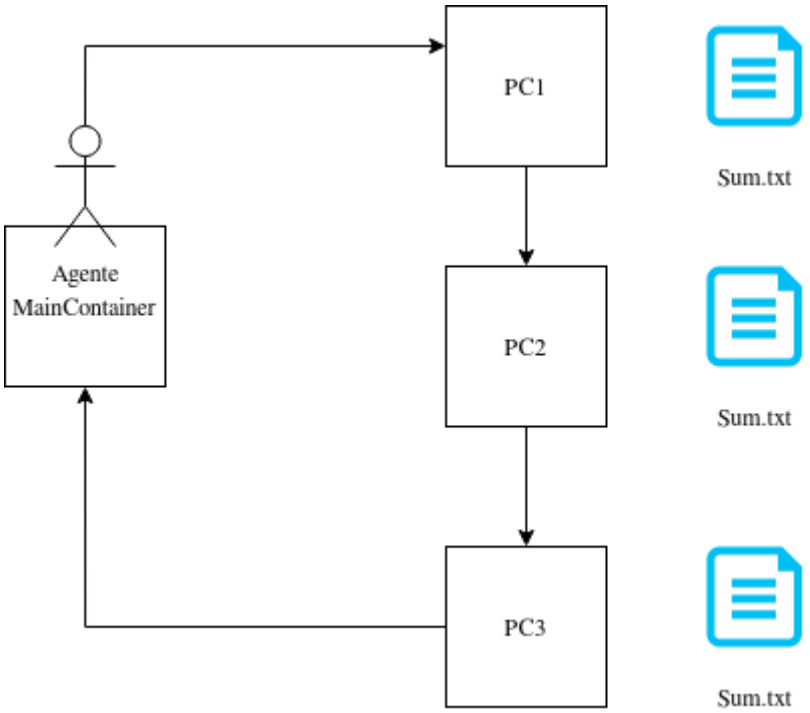
```
java -cp ../lib/jade.jar:classes jade.Boot -gui -container -host localhost
-agents mol:MonitoringAgent
```

Si se utilizara el algoritmo de instantánea, cada uno de los containers tendría un proceso y estos deberían sincronizarse entre ellos para ejecutarse al mismo tiempo.

Es por ello que la principal diferencia se daría en el tiempo de ejecución, el algoritmo de instantánea tendrá un tiempo menor al original.

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 cómo se haría lo mismo con una aplicación cliente/servidor. Comente qué pasaría si hubiera otros sitios con archivos que deben ser procesados de manera similar.

Decidimos arrancar resolviendo el problema con múltiples contenedores para poder también considerar lo que se plantea en la pregunta final. Se creó una clase **`MultipleContainers`** que crea varios contenedores, y por otro lado, un agente **`AgenteMovil`**, que recibe por parámetro el *path* del archivo, y luego de migrar a la primera computadora, lee el archivo, realiza la suma de los números almacenados en dicho archivo, se guarda el resultado, migra a la próxima computadora, vuelve a leer el archivo, realiza la suma de los números almacenados en ese archivo, guarda la suma, y así en cada computadora hasta retornar al *Main-Container*, adonde suma todas las sumas almacenadas y muestra el resultado.



Instrucciones para compilar (**`./1.compilacion.sh`**):

```
javac -classpath ../lib/jade.jar -d classes myexamples/AgenteMovil.java
myexamples/MultipleContainers.java -Xdiags:verbose
```

Instrucciones para ejecutarlo

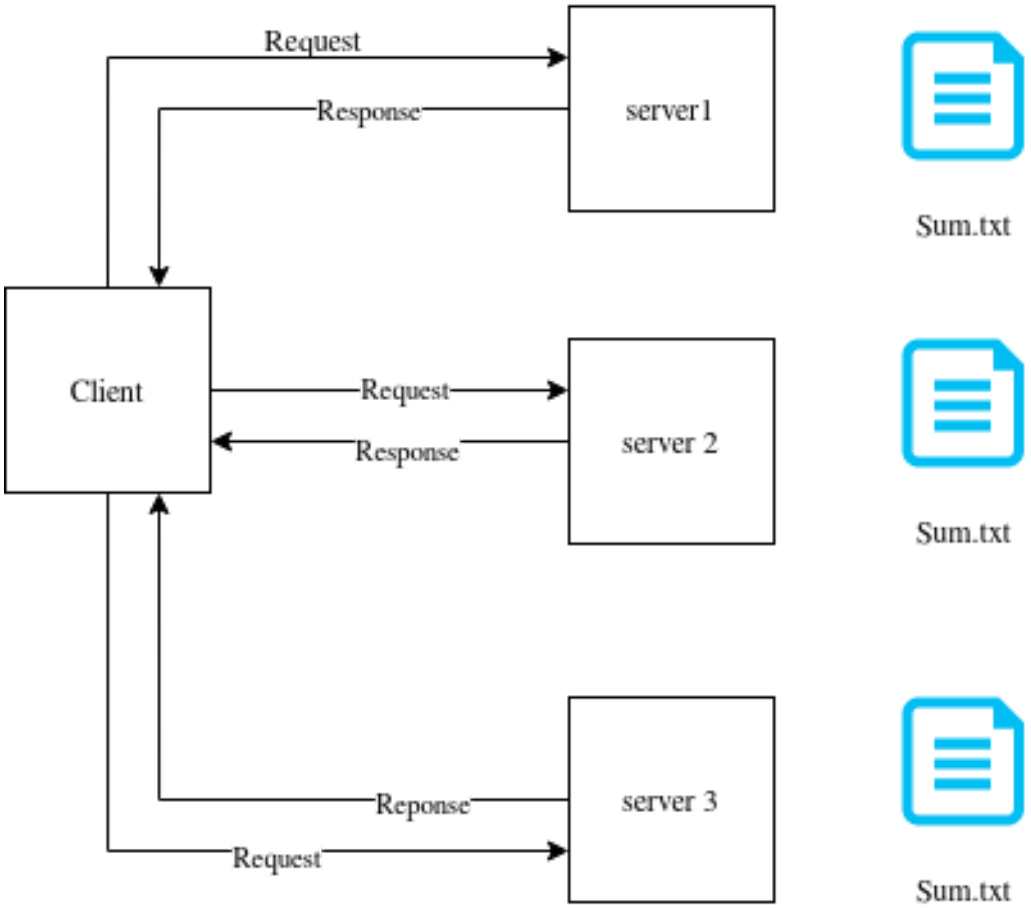
Terminal 1 (**`./2.gui.sh`**)

```
java -cp ../lib/jade.jar:classes myexamples.MultipleContainers
```

Terminal 2 (**`./3.agente.sh`**)

```
java -cp ../lib/jade.jar:classes jade.Boot -container -host localhost
-port 1099 -agents "AgenteMovil:myexamples.AgenteMovil($PWD/sum.txt)"
```

Si se quisiera implementar algo similar en una app cliente/servidor, el servidor debería tener acceso al filesystem en el cual se almacena el archivo, y se debería exponer un servicio que lee los contenidos el archivo, los procese y retorne la suma solicitada, o que al recibir los números, el cliente los pueda sumar. En el caso de haber que leer varios archivos distribuidos, debería haber varios servidores con servicios similares expuestos al cual el cliente pueda acceder.



3. Defina e implemente con agentes un sistema de archivos distribuido similar al de las prácticas anteriores.

   a. Debería tener como mínimo la misma funcionalidad, es decir las operaciones (definiciones copiadas aquí de la práctica anterior):

      - leer: dado un nombre de archivo, una posición y una cantidad de bytes a leer, retorna 1) la cantidad de bytes del archivo pedida a partir de la posición dada o en caso de haber menos bytes, se retornan los bytes que haya y 2) la cantidad de bytes que efectivamente se retornan leídos

      - Escribir: dado un nombre de archivo, una cantidad de bytes determinada, y un buffer a partir del cual están los datos, se escriben los datos en el archivo dado. Si el archivo existe, los datos se agregan al final, si el archivo no existe, se crea y se le escriben los datos. En todos los casos se retorna la cantidad de bytes escritos.

   b. Implemente un agente que copie un archivo de otro sitio del sistema distribuido en el sistema de archivos local y genere una copia del mismo archivo en el sitio donde está originalmente. Compare esta solución con la de los sistemas cliente/servidor de las prácticas anteriores.

En este proyecto, planteamos un agente llamado FPTAgent, con una clase auxiliar que procesa los comandos. La clase del FTPAgent ejecuta operaciones de lectura, escritura y lectura-escritura sobre archivos en un sistema de archivos entre contenedores. Este agente se comporta tanto como cliente como servidor.

Funcionalidad:

- Lectura (read): El agente lee una porción del archivo remoto, específicamente 200,000 bytes cada vez, y los almacena en una variable. Si el archivo es más grande, se repite este proceso hasta completar la lectura.
- Escritura (write): Similar a la lectura, pero en sentido inverso. Aquí, el agente escribe los datos en el servidor, leyendo del archivo local.
- Lectura y Escritura (readwrite): Esta función combina las dos anteriores, copiando un archivo del sistema distribuido al local y luego creando una copia en su ubicación original.

Usamos los scripts para su ejecución, reutilizando los scripts de ejemplo:

1.compilacion.sh

```
javac -classpath ../lib/jade.jar -d classes FTPAgent.java
FTPCommand.java
```

2.gui.sh

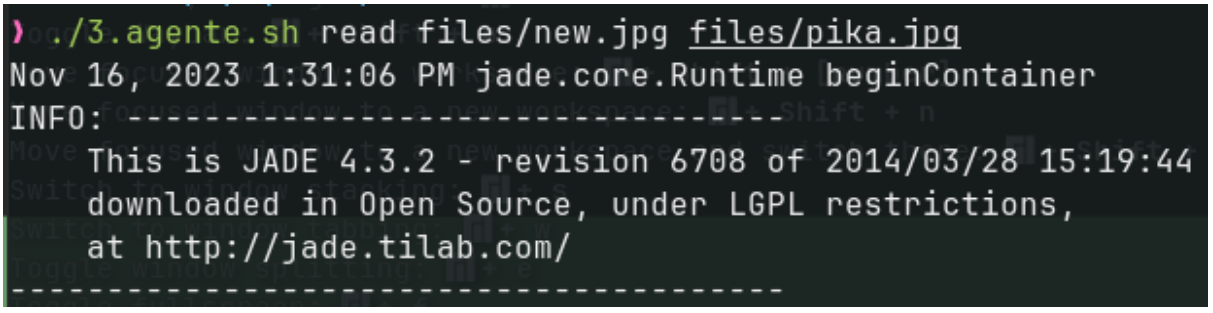
```
java -cp ../lib/jade.jar:classes jade.Boot -gui
```

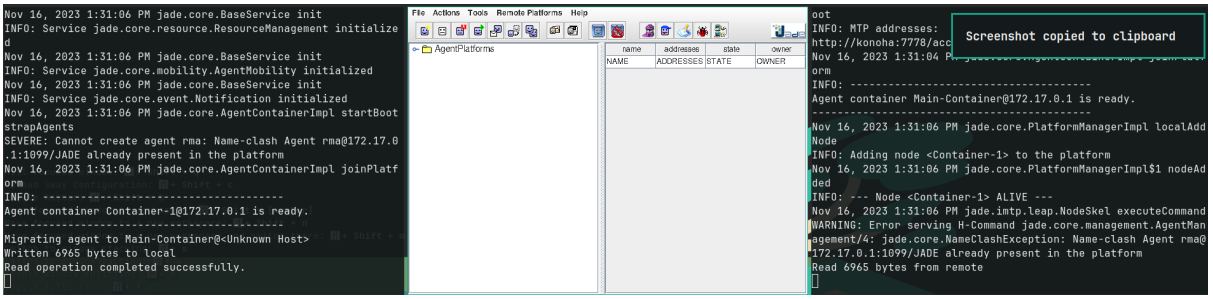
Y al **3.agente.sh** le agregamos logica para recibir paramentros:

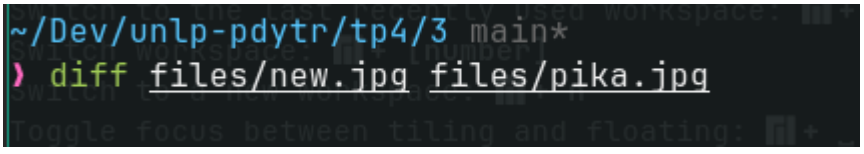
```
#!/bin/bash

if [ "$#" -ne 3 ]; then
      echo "Usage: $0 [read|write|readwrite] localPath remotePath"
      exit 1
fi

op=$1
local_path=$2
remote_path=$3

java -cp ../lib/jade.jar:classes jade.Boot -gui -container -host
localhost -agents "agent:FTPAgent($op,$local_path,$remote_path)"
```

Ejemplos con archivos binarios (una imagen):

READ:

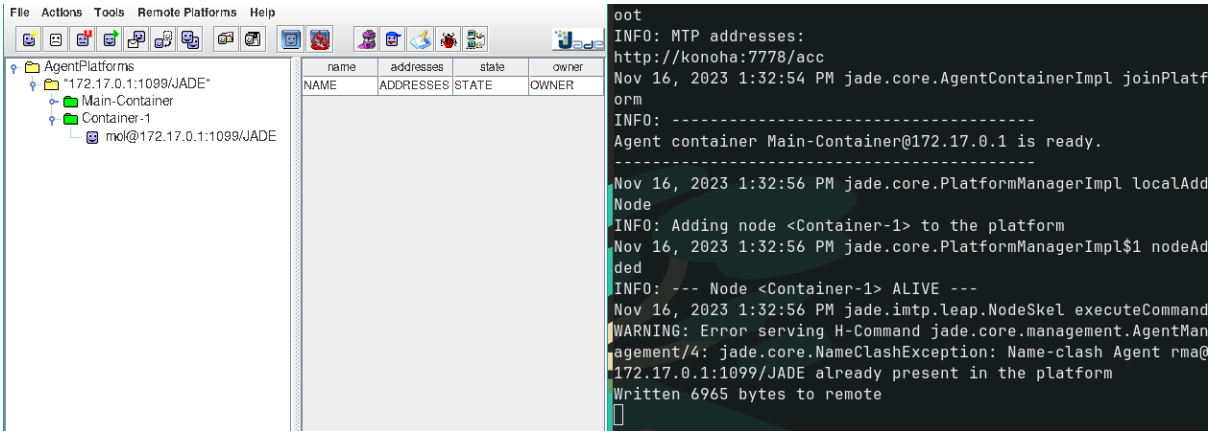






WRITE:

```
) ./3.agente.sh write files/new.jpg files/new-new.jpg
Nov 16, 2023 1:32:56 PM jade.core.Runtime beginContainer
INFO: -----------------------------------
    This is JADE 4.3.2 - revision 6708 of 2014/03/28 15:19:44
    downloaded in Open Source, under LGPL restrictions,
    at http://jade.tilab.com/
-----------------------------------------
```

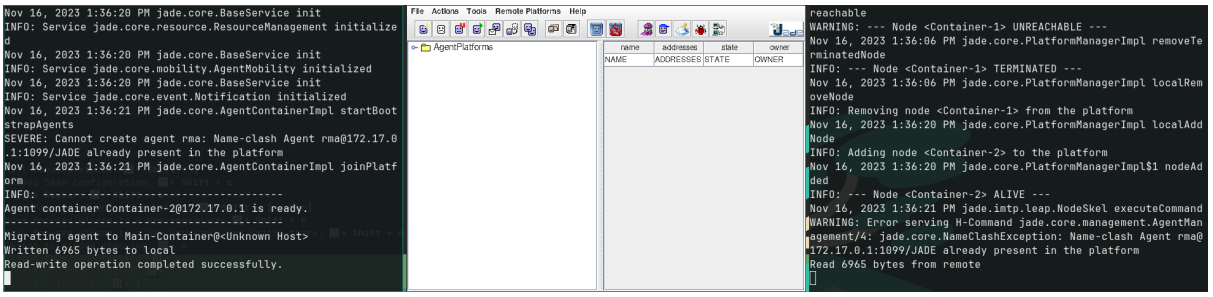


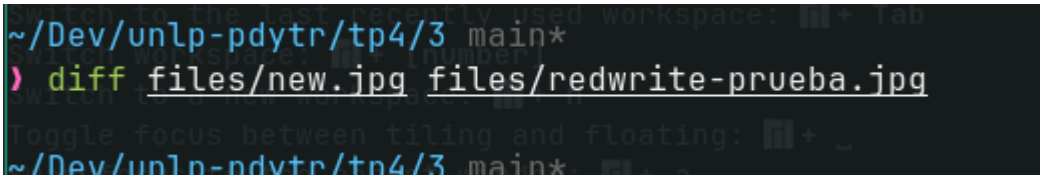

```
~/Dev/unlp-pdytr/tp4/3 main*
) diff files/new.jpg files/new-new.jpg
```

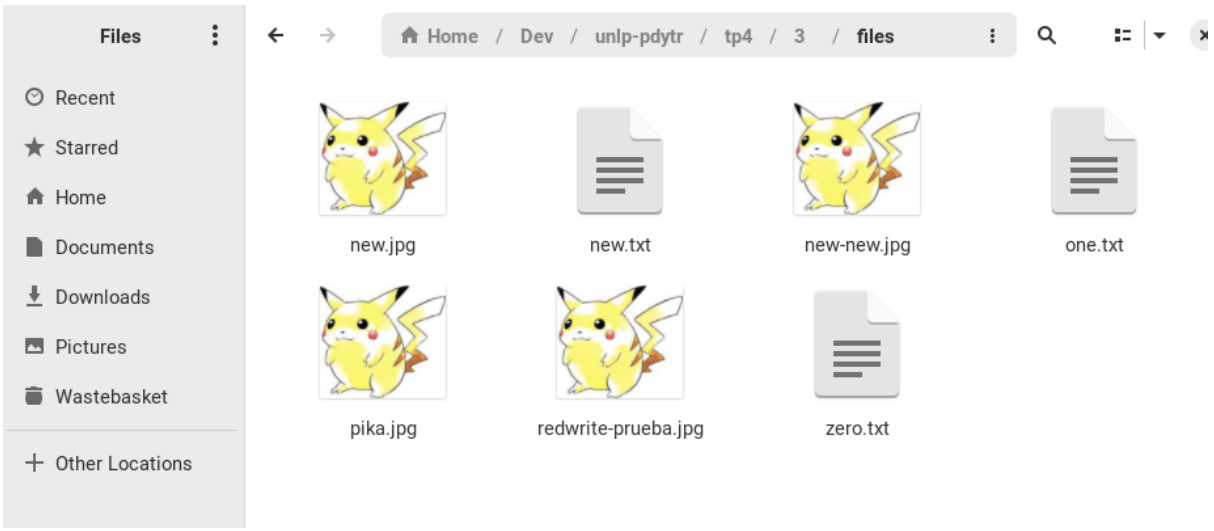
:

READWRITE:

```
) ./3.agente.sh readwrite files/redwrite-prueba.jpg files/new.
jpg
Nov 16, 2023 1:36:20 PM jade.core.Runtime beginContainer
INFO: -----------------------------------
    This is JADE 4.3.2 - revision 6708 of 2014/03/28 15:19:44
    downloaded in Open Source, under LGPL restrictions,
    at http://jade.tilab.com/
-----------------------------------------
```







Este enfoque, aunque unifica el cliente y el servidor en un solo agente, resulta en una mayor complejidad y potencialmente en una menor escalabilidad y mantenimiento. Además, el uso intensivo de migraciones puede incrementar el overhead en comparación con las implementaciones cliente/servidor tradicionales.