

# INDICACIONES PRÁCTICA

## 2

\* Nombre de Archivo Java:

***Mandar dentro del archivo comprimido la carpeta sistemaDistribuido***

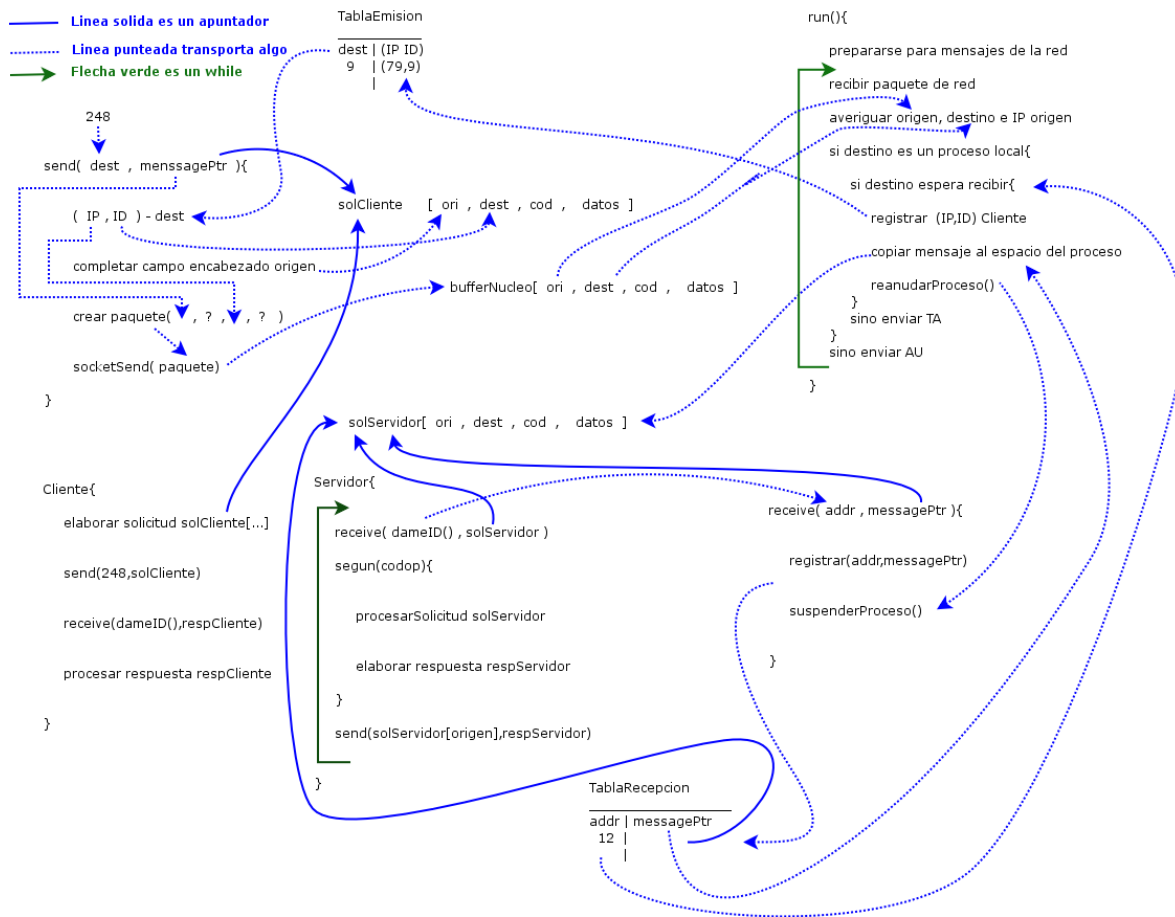
\* Ejemplo Nombre Archivo Comprimido:

GutierrezAvilesLuisP2TSOAD0413B.zip

Nota: Al igual que en la práctica 1, hay que seguir las indicaciones del archivo .pdf y se debe de poner en todos los archivos .java que sean modificados su nombre completo, código y Número de práctica para la cual se hicieron estas correcciones.

La práctica 2, debe de implementar lo que viene en esta imagen:

# Taller De Sistemas Operativos Avanzados – 2013B – D04



Los archivos a corregir se encuentran dentro de los paquetes  
 sistemaDistribuido.sistema.clienteServidor.modosMonitor y  
 sistemaDistribuido.sistema.clienteServidor.modosUsuario

## A) ¿Qué se debe corregir en ProcesoCliente.java?

Aquí debería estar implementado esta parte del dibujo:

## Taller De Sistemas Operativos Avanzados - 2013B - D04

---

```
Cliente{  
    elaborar solicitud solCliente[...]  
  
    send(248,solCliente)  
  
    receive(dameID(),respCliente)  
  
    procesar respuesta respCliente  
}
```

Esto supuestamente debería de estar hecho desde la práctica 1, pero sólo habría de corregir la forma de la solicitud cliente de la siguiente forma:

```
solCliente [ ori , dest , cod , datos ]
```

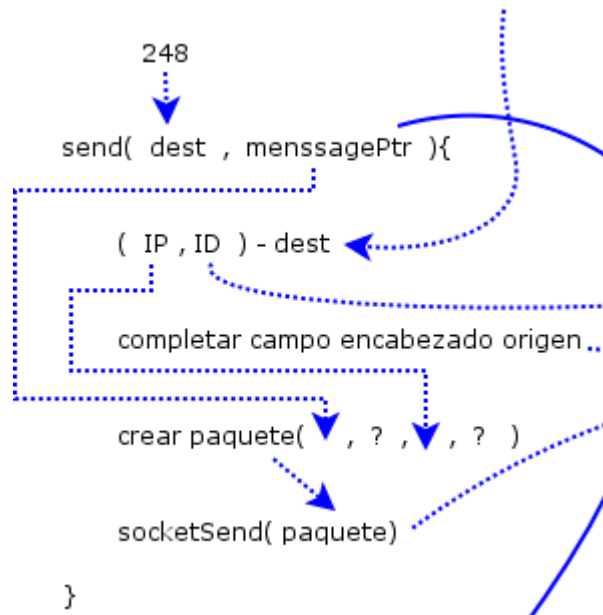
Es decir dejar 4 bytes libres al principio para que ahí sea colocado el **ID** del proceso **origen**, después otros 4 bytes para que se coloque el **ID** del proceso **destino**, otro byte para colocar el **código de operación** y los bytes restantes guardaran los **datos** necesarios para la operación. Como les dije esto ya debería de estar implementado desde la práctica 1 pero algunos no dejaron los espacios libres para colocar los datos del proceso destino y del proceso origen.

### B) ¿Qué se debe corregir en MicroNucleo.Java?

Aquí son varias cosas a corregir, empecemos con el Send, que sería esta parte de la imagen:

# Taller De Sistemas Operativos Avanzados - 2013B - D04

---



Originalmente así viene implementado el Send dentro del Micronucleo.java

```
protected void sendVerdadero(int dest,byte[] message){
    sendFalso(dest,message);
    imprimeln("El proceso invocante es el
"+super.dameIdProceso());
    /*
    //lo siguiente aplica para la práctica #2
    ParMaquinaProceso pmp=dameDestinatarioDesdeInterfaz();
    imprimeln("Enviando mensaje a IP="+pmp.dameIP()+"
ID="+pmp.dameID());
    suspenderProceso(); //esta invocacion depende de si este
hilo de control no realiza el envio del mensaje
    */
}
```

Este método debe de ser corregido de la siguiente manera:

Primeramente hay que checar si el ID del proceso que se recibe como el parámetro dest se encuentra dentro de una TablaEmision, pero ¿Qué esa tablaEmision?

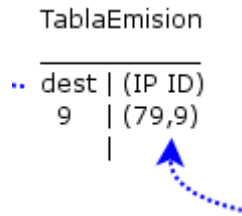
En la misma imagen bien esto:

# Taller De Sistemas Operativos Avanzados - 2013B - D04

---

TablaEmision

dest	(IP ID)
9	(79,9)



La tabla emisión es una HashTable que debería de almacenar como Key el ID del proceso destino y como dato una instancia de alguna clase que tenga como atributos la IP del proceso origen y el ID de ese número de proceso (Aquí van a servir lo aprendido en la tarea Java de la clase administradora), en esta tabla tendrían que registrarse todos los procesos que lograron transmitir de forma exitosa, más adelante veremos en que momento se hace el registro.

Como decíamos, dentro del Send se busca por medio del parámetro dest en la tablaEmision, en este caso en que el Cliente hizo la solicitud mandó un 248 como valor de dest, número que no ha sido registrado en la TablaEmision y que por tanto no nos da ningún resultado, en este caso el ID del proceso destino y su IP se toman de la interfaz del núcleo, vean que al correr el programa hay dos campos de texto donde se pueden escribir estos datos.

En el código que se encuentra comentado esta esto:

```
ParMaquinaProceso pmp=dameDestinatarioDesdeInterfaz();
```

Precisamente con esta sentencia obtienen estos datos,

```
pmp.dameIP() //Asi se obtiene la IP
```

```
pmp.dameID() //Asi se obtiene la ID
```

Una vez que tenemos estos datos se completa el encabezado del mensaje que se va a enviar por la red y que se recibió como el parámetro message, lo que hay que completar es el espacio con el ID del proceso Origen y el espacio con el ID del proceso destino.

Por último se prepara un paquete Datagrama como se hizo en la tarea de Java del chat y se envía a la IP que se obtuvo de la interfaz. Y terminamos el send.

Ahora bien, hay que corregir otro método que convocan los clientes y servidores para esperar respuesta y solicitudes respectivamente, hay que implementar en este método receive lo que viene en esta parte de la imagen:

```
receive( addr , messagePtr ){  
    registrar(addr,messagePtr)  
    suspenderProceso()  
}
```

Donde addr es la ID del proceso que convocó al método y messagePtr es el arreglo de bytes donde se esperan los datos, aquí entra en juego otra HashTable llamada TablaRecepcion que debe de tener como key el ID del proceso que convocó a receive y como datos el arreglo de bytes donde se espera los datos con que trabajar.

Este método se encuentra originalmente en Micronucleo.Java de la siguiente manera:

```
protected void receiveVerdadero(int addr,byte[] message){  
    receiveFalso(addr,message);  
    /*  
    //el siguiente aplica para la práctica #2  
    suspenderProceso();  
    */  
}
```

## Taller De Sistemas Operativos Avanzados - 2013B - D04

---

Hay que hacer las correcciones necesarias, que no son muchas, para que se registre en una tabla y se suspenda el proceso, más adelante veremos en que momento se borran de la TablaRecepcion.

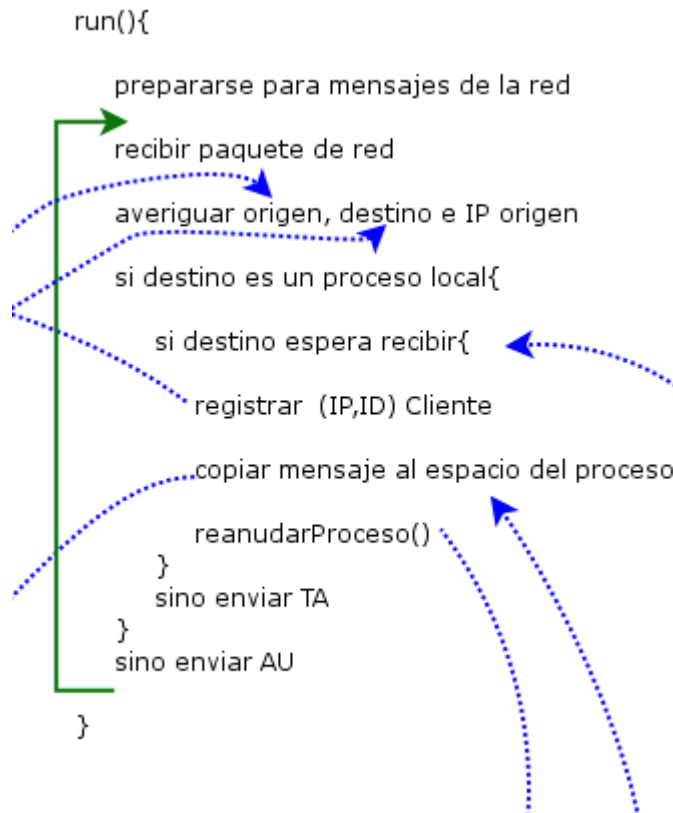
A continuación hay que corregir el método run del Micronucleo.Java que originalmente esta así:

```
public void run(){
    while(seguirEsperandoDatagramas()){
        /* Lo siguiente es reemplazable para la práctica #2,
        * si lo comentamos en práctica #1 incrementa el uso de
CPU según la JRE
        */
        try{
            sleep(60000);
        }catch(InterruptedException e){
            System.out.println("InterruptedException");
        }
    }
}
```

Aquí es necesario poner esta parte de la imagen:

## Taller De Sistemas Operativos Avanzados – 2013B – D04

---



El prepararse para recibir mensajes de la red y recibir paquetes de la red es tan sólo implementar lo visto en la tarea del Chat como el hilo receptor, recordando que todo este código debería de encontrarse dentro de un ciclo que estuviera recibiendo un mensaje tras otro, incluso la condición de paro ya esta en el código

```
while(seguirEsperandoDatagramas()){
```

Una vez que se recibe el mensaje se averigua el origen, el destino el IP Origen, los dos primero se encuentran en el mensaje que se recibió en la red en los espacios que se reservaron para este motivo, la IP origen se puede averiguar del Datagrama recibido por medio de `dp.getAddress().getHostAddress();`, tal como se uso en la tarea del Chat.



## Taller De Sistemas Operativos Avanzados - 2013B - D04

---

Una vez que se obtienen estos datos se averigua si el proceso destino es un proceso local por medio de un método que ya nos ofrece el micronúcleo:

```
dameProcesoLocal(destino);
```

Donde destino es el dato que sacamos del mensaje, este método nos regresa una instancia de la clase Proceso que debemos de asignarla a una variable dentro de este run().

En caso de que el proceso destino no sea un proceso local debemos de regresar el error AU (Address Unknown), para que el nucleo del cliente identifique que se trata de un mensaje AU podemos utilizar números negativos.

Ahora bien, suponemos que el proceso destino es un proceso local, ahora debemos validar si ese proceso está esperando recibir un mensaje, es decir si convocó a receive(), para esto buscamos en la tablaRecepcion y en caso de que si se encuentre nos regresa el arreglo de bytes donde se espera sean copiados los datos que vienen en el datagrama proveniente de la red.

Como este mensaje se logró recibir exitosamente, es el momento en que se registra en la TablaEmision el IP origen y el ID origen, datos que sacamos del mismo mensaje, a su vez tenemos que borrar de la tablaRecepcion al proceso destino porque ya no va a estar esperando mensajes.

Una vez que se copia el arreglo de bytes que venía como buffer en el Datagrama que se recibió de la red al arreglo de Bytes del proceso destino se procede a despertar al proceso destino por medio de la sentencia reanudarProceso(p); que es parte de los métodos del micronúcleo, y donde p es la instancia de la clase Proceso que hace referencia al proceso destino.

Puede haber el caso de que el proceso destino no haya convocado a `receive()` y que por tanto no esté esperando mensajes y tendría que regresarse un paquete TA (Try Again) pero no es necesario que se implemente en esta práctica.

### c) **¿Qué se debe corregir en `ProcesoServidor.java`?**

Aquí es muy parecido que el `ProcesoCliente`, sólo habría que corregirse sino se respetaron los primeros bytes de la Respuesta para guardar los datos del proceso origen y del proceso destino ya que la `solServidor` debe de tener estos datos

## Taller De Sistemas Operativos Avanzados – 2013B – D04

---

`solServidor[ ori , dest , cod , datos ]`

El código a implementar es lo que refiere esta parte de la imagen

The diagram illustrates the implementation of a server process. A blue curved arrow points from the `solServidor` array to the `receive` function. A green arrow points from the `send` function to the `solServidor` array. A blue dotted line connects the `ori` field of `solServidor` to the `dest` field of `solServidor`. A blue solid line connects the `dest` field of `solServidor` to the `ori` field of `solServidor`. A green arrow points from the `send` function to the `solServidor` array.

```
Servidor{  
    receive( dameID() , solServidor )  
    segun(codop){  
        procesarSolicitud solServidor  
        elaborar respuesta respServidor  
    }  
    send(solServidor[origen],respServidor)  
}
```

OJO, en la práctica 1, al utilizar send para enviar la respuesta al cliente en vez de un cero, hay que colocar el id del proceso origen que se toma de la misma solicitud que se le llegó al servidor, de esta manera al convocar a send, que ya debió ser modificado como se dijo en el inciso B, buscará en la TablaEmision el id del proceso que se recibe en el parámetro como dest, en este caso si se debería de encontrar ya que antes de despertar el proceso servidor el hilo núcleo hizo este registro, se obtienen de esta tabla los datos de ID e IP destinos, en este caso los correspondientes al cliente, se procede a eliminar a este proceso de la TablaEmision, ya que los datos son utilizados y si ya fue corregido el send según se detallo en el inciso B, el cliente deberá recibir su respuesta sin ningún problema y la práctica 2 estaría completa.

# Taller De Sistemas Operativos Avanzados - 2013B - D04

---

## Criterios de Evaluación

- Cumplir con las “Reglas de Operación y Evaluación del Taller de Sistemas Operativos Avanzados”.
- Fecha de asignación: 09 de Octubre del 2013
- Fecha planeación de entrega: 16 de Octubre del 2013
- Fecha límite de entrega (Sólo 55% del total): 06 de Noviembre del 2013
- Observación: Ninguna
- Calificación en base a cobertura de requerimientos y fecha de entrega en horas clase