



INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO



Unidad de Aprendizaje:

Aplicaciones para Comunicaciones de Red

Profesora:

Sandra Ivette Bautista Rosales

Alumnos:

Ramírez Álvarez Ángel Negib
Valentín Mondragón Alfredo Emmanuel

Grupo:

3CV6

Equipo:

1

Práctica Número:

02

Fecha de Entrega:

10 de abril de 2019

Objetivo

El estudiante implementará una aplicación de carrito de compra para la selección y adquisición de artículos, generación de recibo de compra y el envío de múltiples objetos serializados a través de la red haciendo uso de sockets de flujo.

Introducción

En la actualidad, muchos de los procesos que se ejecutan en una computadora requieren obtener o enviar información a otros procesos que se localizan en una computadora diferente. Para lograr esta comunicación se utilizan los protocolos de comunicación TCP y UDP.

El protocolo TCP (Transmission Control Protocol) establece un conducto de comunicación punto a punto entre dos computadoras, es decir, cuando se requiere la transmisión de un flujo de datos entre dos equipos, el protocolo TCP establece un conducto exclusivo entre dichos equipos por el cual los datos serán transmitidos y este perdurará hasta que la transmisión haya finalizado, gracias a esto TCP garantiza que los datos enviados de un extremo de la conexión lleguen al otro extremo y en el mismo orden en que fueron enviados. Las características que posee TCP hacen que el protocolo sea conocido como un protocolo orientado a conexión.

Mientras tanto también existe un protocolo no orientado a la conexión llamado UDP. El protocolo UDP no es orientado a la conexión debido a que la forma de transmitir los datos no garantiza en primera instancia su llegada al destino, e incluso si este llegara al destino final, tampoco garantiza la integridad de los datos. UDP hace la transmisión de los datos sin establecer un conducto de comunicación exclusiva como lo hace TCP, además utiliza datagramas, los cuales contienen una porción de la información y que son enviados a la red en espera de ser capturados por el equipo destino. Cuando el destino captura los datagramas debe reconstruir la información, para esto debe ordenar la información que recibe ya que la información transmitida no viene con un orden específico, además se debe tener conciencia de que no toda la información va a llegar. El funcionamiento del protocolo UDP se puede comparar con el servicio postal.

Principio de funcionamiento.

Un socket queda definido por un par de direcciones IP local y remota, un protocolo de transporte y un par de números de puerto local y remoto. Para que dos programas puedan comunicarse entre sí es necesario que se cumplan ciertos requisitos:

Que un programa sea capaz de localizar al otro.

Que ambos programas sean capaces de intercambiarse cualquier secuencia de octetos, es decir, datos relevantes a su finalidad.

Para ello son necesarios los tres recursos que originan el concepto de socket:

Un protocolo de comunicaciones, que permite el intercambio de octetos.

Un par de direcciones del Protocolo de Red (Dirección IP, si se utiliza el Protocolo TCP/IP), que identifica la computadora de origen y la remota.

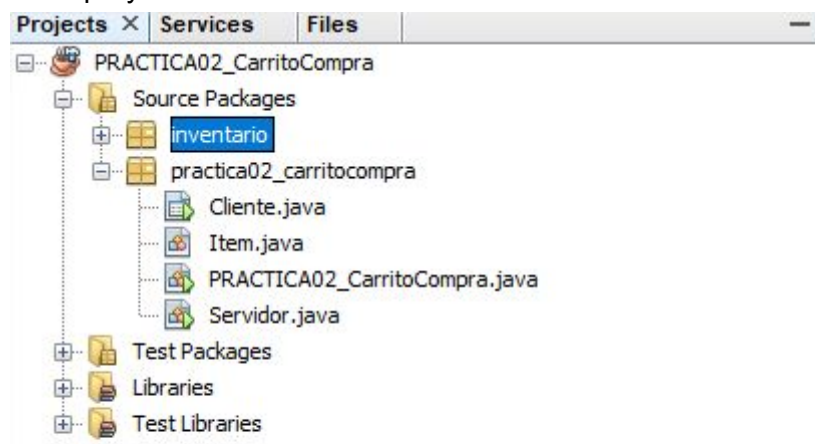
Un par de números de puerto, que identifica a un programa dentro de cada computadora.

Los sockets permiten implementar una arquitectura cliente-servidor o peer to peer. La comunicación debe ser iniciada por uno de los programas que se denomina programa cliente. El segundo programa espera a que otro inicie la comunicación, por este motivo se denomina programa servidor. Cabe mencionar que hay distintos tipos de sockets utilizados con TCP y UDP, como sockets `dg_ram`, sockets `_stream`, etc.

Desarrollo

Durante el análisis de la problemática concluimos en que la manera de envío de los datos sería serializado con formato JSON mediante una librería externa

Para la creación del proyecto utilizamos tres clases de las cuales se hablará posteriormente.



Cliente

Para el diseño de la interfaz del cliente utilizamos una IDE llamada (netBeans). Podemos ver la interfaz en la siguiente imagen. Como se nos solicitó se muestra el catalogo que está en el servidor y la parte de la lista de compras

PRACTICA 02 Carrito de Compra

Se hará la conexión y se mostrarán las existencias aquí abajo:

SKU	Nombre	Existencias
-----	--------	-------------

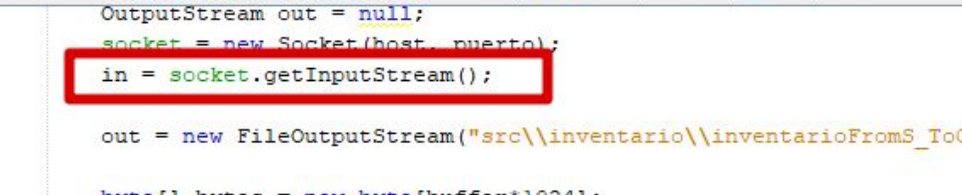
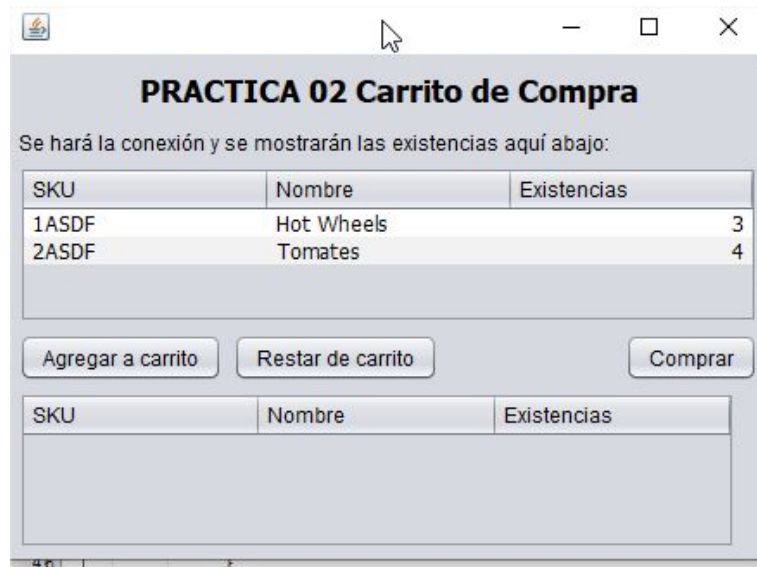
Agregar a carrito Restar de carrito Comprar

SKU	Nombre	Existencias
-----	--------	-------------

Adentrándonos en el código como ya vimos en prácticas anteriores la conexión hacia el servidor sigue siendo parecida. Se crea un socket se asigna puerto y al host al que va dirigido.

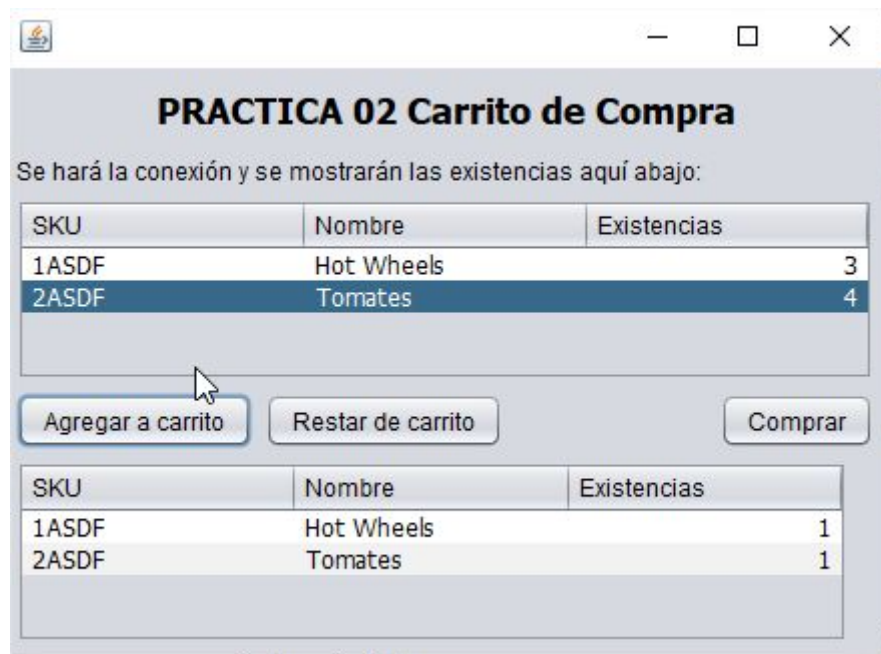
```
public void connectToServer(String host, int puerto, int buffer) throws IOException{  
  
    InputStream in = null;  
    OutputStream out = null;  
    socket = new Socket(host, puerto);  
    in = socket.getInputStream();  
}
```

Para la recepción del catálogo ahora del lado del cliente primero se recibe un flujo al hacer la conexión y posteriormente se muestra en la interfaz previamente mencionada.



```
Source  Design  History  [Icons]
34      OutputStream out = null;
35      socket = new Socket(host, puerto);
36      in = socket.getInputStream();
37
38      out = new FileOutputStream("src\\inventario\\inventarioFromS_ToC.json");
39
40      byte[] bytes = new byte[buffer*1024];
41      int count;
42      int i=0;
43      while ((count = in.read(bytes)) > 0) {
44          out.write(bytes, 0, count);
45          i++;
46          System.out.println("línea "+i+" . Cuenta de recepción: "+count);
47      }
48
49      System.out.println("Recibió el archivo.");
```

Para agregar en el carrito solo nos posicionamos en el producto con un click y damos en agregar y se agrega un producto a la lista.

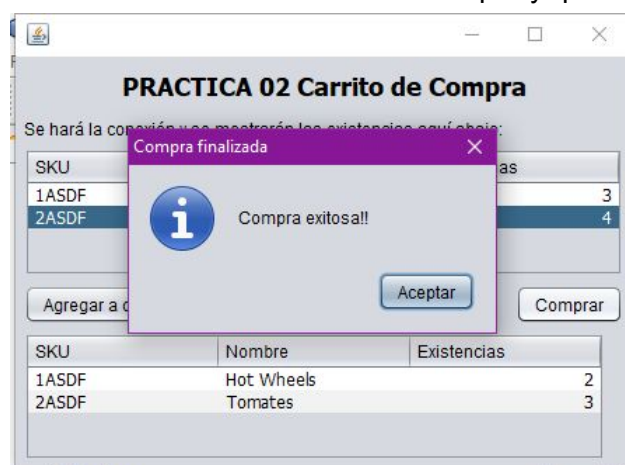


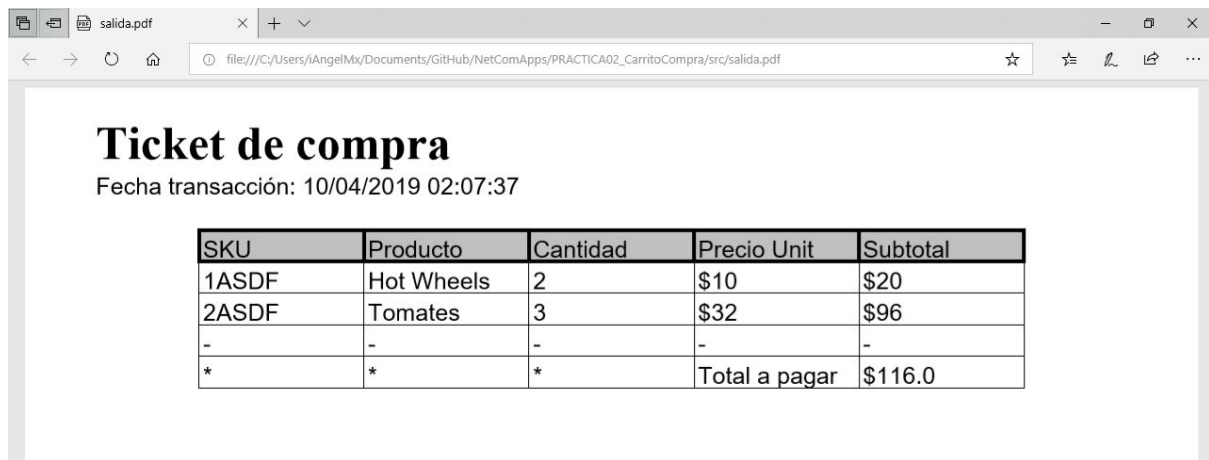
Cuando hacemos uso del boton comprar este nos genera un nuevo archivo el cual es la “lista de compras” que se enviará al servidor para poder hacer la actualización necesaria del catálogo y retirar del stock. El archivo se encuentra serializado en formato JSON.

```
[{"sku": "1ASDF", "nombre": "Hot Wheels", "exis": 2, "precio": 10}, {"sku": "2ASDF", "nombre": "Tomates", "exis": 3, "precio": 10}]
```

En este caso nos resto uno de cada uno en el stock.

Cuando se termina de realizar el procedimiento anterior en el servidor, la aplicación cliente notifica al usuario que ha finalizado correctamente su compra y que verifique su ticket.





El cual se genera mediante la librería iTextPdf

```
try{
    PdfWriter.getInstance(document, new FileOutputStream("src\\salida.pdf"));

    document.open();
    Font font = FontFactory.getFont(FontFactory.TIMES_BOLD, 24, BaseColor.BLACK);
    Chunk chunk = new Chunk("Ticket de compra", font);

    document.add(chunk);

    Font font2 = FontFactory.getFont(FontFactory.COURIER, 16, BaseColor.BLACK);
    String salida = "";

    DateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
    Date date = new Date();
    String fecha = dateFormat.format(date);

    PdfPTable table = new PdfPTable(5);
    addTableHeader(table);

    document.add(new Paragraph(""));
    document.add(new Paragraph("Fecha transacción: "+ fecha+"\n\n"));
    document.add(new Paragraph(""));

    double total = 0;
    for(int a=0; a<clonProd.length; a++){
        if( clonProd[a].exis > 0 ){
            addRows(table, new String[]{clonProd[a].sku, clonProd[a].nombre, String.val
            total += (clonProd[a].precio)*clonProd[a].exis;
```



```

Gson gson = new Gson();
String jsonShipping = gson.toJson(clonProd);

try {
    File archivo = new File("src\\inventario\\shippingList.json");
    long length = archivo.length();
    byte[] bytes = new byte[200 * 1024];
    out = new FileOutputStream(archivo);
    out.write(jsonShipping.getBytes());

    in = new FileInputStream(archivo);
    out = socket.getOutputStream();
    System.out.println("Escribiendo en socket -cliente");
    while ((count = in.read(bytes)) > 0) {
        out.write(bytes, 0, count);
    }
    socket.close();
    JOptionPane.showMessageDialog(null, "Compra exitosa!!", "Compra finalizada", JOptionPane.INFORMATION_MESSAGE);
    System.exit(0);
    //out.flush();
} catch (Exception ex) {
    System.out.println("Excepción encontrada!: " + ex);
}
}

```

Servidor

En el servidor para enviar el archivo de vuelta al se usó la clase `Items` que previamente definimos la cual contendrá la descripción de los productos existencia, nombre, etc. La clase instanciada se serializa a formato JSON y se escribió en un archivo que posteriormente se envió al cliente en su primer conexión.

```

37 serverSocket = new ServerSocket(puerto);
38 System.out.println("Se ha iniciado el servidor...");
39 while(true){
40     Socket socket = null;
41     InputStream inFile = null, inSocket=null;
42     OutputStream outSocket = null, outFile=null;
43     int count;
44     if(flag == false){
45         System.out.println("Llegó aquí");
46         socket = serverSocket.accept();
47         System.out.println("Aceptó una conexión");
48
49         byte[] bytes = new byte[buffer * 1024];
50         System.out.println(json.toJson(getInventario()));
51         inFile = new ByteArrayInputStream(json.toJson(getInventario()).getBytes());
52
53
54
55         outSocket = socket.getOutputStream();
56

```

después de esto se espera a recibir otra conexión en la cual el cliente nos enviará su lista de compras para poder hacer la actualización de nuestro stock de lado del servidor


```

try{
    socket = serverSocket.accept();
    inSocket = socket.getInputStream();
    System.out.println("Trató de obtener un dato de entrada");
    outFile = new FileOutputStream("src\\inventario\\recibidoDeCliente.json");
    byte[] bytes = new byte[buffer*1024];
    int i=0;
    String msg;
    System.out.println("Esperará archivo");
    while ((count = inSocket.read(bytes)) > 0) {
        outFile.write(bytes, 0, count);
        i++;
        System.out.println("línea "+i+" Cuenta servidor: "+count);
    }

    System.out.println("Salió del while");
    setInventario("src\\inventario\\recibidoDeCliente.json");
    flag=false;
}
catch(Exception ex){

```

Para poder seguir recibiendo más listas de compras fue necesario poner una bandera la cual indica que volvamos a enviar el stock actualizado al cliente otra vez.

Cuestionario

1. ¿Qué es serialización?

La serialización es el proceso de convertir un objeto en una secuencia de bytes para almacenarlo o transmitirlo a la memoria, a una base de datos o a un archivo.

Es un proceso de codificación de un objeto en un medio de almacenamiento (como puede ser un archivo, o un buffer de memoria) con el fin de transmitirlo a través de una conexión en red como una serie de bytes o en un formato humanamente más legible como XML o JSON, entre otros. La serie de bytes o el formato pueden ser usados para crear un nuevo objeto que es idéntico en todo al original, incluido su estado interno

2. ¿En qué difiere del marshalling?

Marshalling consiste en transformar parámetros de aquí a allá, mientras que la serialización se trata de copiar datos estructurados hacia o desde una forma primitiva, como lo son las cadenas de bytes. En ese sentido, la serialización es un medio para realizar el cálculo de referencias, generalmente implementando semántica de paso por valor.

También es posible que un objeto pueda ser marshaled por referencia, en ese caso la información concerniente a esto, es información simple ubicada para el objeto original. Como siempre, sólo los objetos aún pueden ser susceptibles a un valor de serialización

3. ¿Por qué es importante la serialización?

La importancia de la serialización no se limita al envío de bytes por la red. Son varias las razones por las que muchos lenguajes de programación como C, C++, C#, Python y Perl (entre otros) han incluido paquetes, módulos o API enfocadas en serializar datos.

ya que:

- Permite hacer llamados a procedimientos remotos (RPC).
- Sirve para identificar cambios de datos en ejecución.
- Posibilidad de almacenamiento de objetos en dispositivos de almacenamiento como discos duros.
- Si un programa en ejecución termina inesperadamente o detecta algún problema, puede cargar un respaldo completo o parcial.
- Permite intercambiar objetos entre programas independientes.

Conclusiones

Ramirez Alvarez Angel Negib:

En la presente se pudieron observar el comportamiento de la serialización de datos y de marshalling, siendo nuestro caso particular en donde usamos más el marshalling que la serialización directa, puesto que, al emplear la librería externa de Google Gson para transformar los datos a un archivo json, estamos volviendo a los datos a un estándar de intercambio de datos sin tener su forma “raw” en bytes de la información en memoria. Sin embargo, sí se sigue empleando para la transmisión de información sobre la escritura de sockets.

Valentin Mondragon Alfredo Emmanuel:

En esta práctica aprendimos el uso de la serialización y las ventajas que tiene sobre el envío de datos de forma plana, puesto que al leer y enviar información se hace de una manera más sencilla tanto para cliente como para el servidor. Podemos destacar la comunicación bidireccional de ambas partes de esta práctica.

Referencias

[1] Bonilla, I. (2012). *Sockets: Protocolos de comunicación TCP y UDP*. [online] Dsp.mx. Available at: <http://dsp.mx/blog/sistemas-de-informacion/49-sockets-tcp-udp> [Accessed 10 Apr. 2019].

[2] Es.wikipedia.org. (n.d.). *Socket de Internet*. [online] Available at: https://es.wikipedia.org/wiki/Socket_de_Internet [Accessed 10 Apr. 2019].

[3] Melado, D. (n.d.). *Puertos y sockets - Protocolos de la familia Internet*. [online] Personales.upv.es. Available at: <http://personales.upv.es/rmartin/tcpip/cap02s10.html> [Accessed 10 Apr. 2019].

[4] Ecured.cu. (n.d.). *Socket - EcuRed*. [online] Available at: <https://www.ecured.cu/Socket> [Accessed 10 Apr. 2019].