Instituto Politécnico Nacional

Escuela Superior de Cómputo

Departamento de Ciencias e Ingeniería de la Computación

Aplicaciones para Comunicaciones en Red

Practica 05 "Carrito de compras musical con socket no bloqueante"

Profesor: Axel Ernesto Moreno Cervantes

Grupo: 6CM1

Integrantes:

Ruvalcaba Flores Martha Catalina

Sandoval Hernández Eduardo

Fecha de entrega: 13/01/2022

Objetivo

El estudiante implementará una aplicación de carrito de compra para la selección y adquisición de canciones, generación de recibo de compra y el envío de múltiples objetos serializados a través de la red haciendo uso de sockets de datagrama usando el no bloqueante, con el fin de que el servidor no se quede esperando, sino que posea un periodo de activación.

Planteamiento del problema

En la actualidad, un sin número de empresas utilizan los medios electrónicos para ofertar sus productos a través de Internet, ya sea mediante portales web, aplicaciones móviles o aplicaciones de escritorio; para ofrecer tales productos las aplicaciones hacen uso de un catálogo, en el que se muestra información detallada sobre los productos que se ofertan, dentro de los datos más importantes que se muestran están los siguientes: distintas imágenes del producto, descripción, colores disponibles, tamaños, precio, tiempo de entrega, etc. También se cuenta con un carrito de compra para que los usuarios puedan realizar la compra de los artículos seleccionados. Dos características importantes de los carritos de compra son: La información de disponibilidad de los productos debe estar siempre actualizada y debe ser consistente. Además, la información del catálogo deberá generarse dinámicamente y ser enviada a la aplicación cliente para que pueda ser exhibida al usuario. Para esto se tendrá que enviar dicho catálogo como una colección de datos de distintos tipos (imágenes, números, cadenas, etc.)

Los carritos de compra son una herramienta importante para la gran mayoría de las empresas, porque les permite incrementar su alcance en el mercado global dadas sus características (difusión, alcance, disponibilidad, etc.). Las principales características que deben tener los carritos de compras de e-comerce son: exhibición del catálogo en línea y dinámico para mostrar en tiempo real la disponibilidad de productos, inclusión dinámica de la información del catálogo. Interfaz de usuario de uso intuitivo y con un diseño que permita mostrar desde 1 hasta cientos de productos de forma ordenada, así como también soporte para la realización de pago electrónico.

Introducción Teórica

Sockets datagrama

Los sockets datagram se basan en el protocolo UDP y ofrecen un servicio de transporte sin conexión. Es decir, podemos mandar información a un destino sin necesidad de realizar una conexión previa. El protocolo UDP es más eficiente que el TCP, pero tiene el inconveniente de que no se garantiza la fiabilidad. Además, los datos se envían y reciben en datagramas (paquetes de información) de tamaño limitado. La entrega de un datagrama no está garantizada: estos pueden duplicarse, perderse o llegar en un orden diferente del que se envió.

La gran ventaja de este tipo de sockets es que apenas introducen sobrecarga sobre la información transmitida. Además, los retrasos introducidos son mínimos, lo cual los hace especialmente interesantes para aplicaciones en tiempo real, como la transmisión de audio y vídeo sobre Internet.

UDP

El Protocolo de datagrama de usuario (UDP) es un protocolo ligero de transporte de datos que funciona sobre IP.

UDP proporciona un mecanismo para detectar datos corruptos en paquetes, pero no intenta resolver otros problemas que surgen con paquetes, como cuando se pierden o llegan fuera de orden. Por eso, a veces UDP es conocido como el protocolo de datos no confiable.

Al enviar paquetes con UDP sobre IP, la porción de datos de cada paquete IP está formateada como un segmento UDP. Cada segmento UDP contiene un encabezado de 8 bytes y datos de longitud variable. Los siguientes dos bytes

del encabezado de UDP almacenan la longitud (en bytes) del segmento (incluyendo el encabezado). Entonces la longitud máxima de un segmento UDP es 65,535 bytes, analizar en la figura 1 el funcionamiento del UDP.

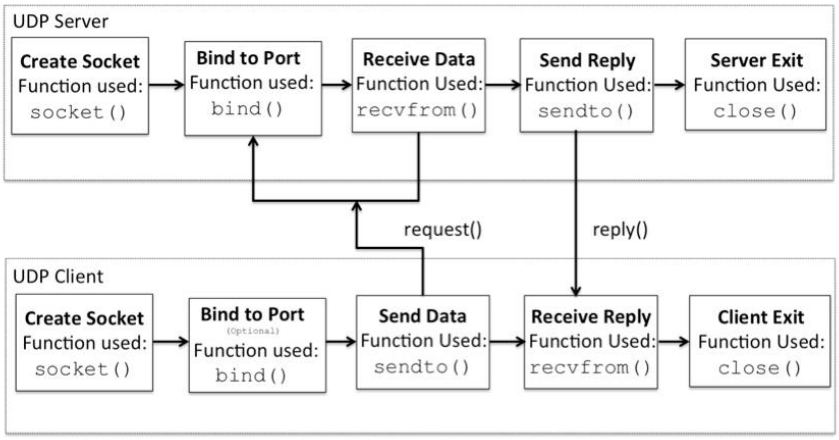


Figura 1. UDP Servidor y Cliente

Eyed3d

EyeD3 es una herramienta de Python para trabajar con archivos de audio, específicamente archivos MP3 que contiene metadatos ID3 (es decir, información de la canción). Proporciona una herramienta de línea de comandos () y una biblioteca de Python () que se pueden utilizar para escribir sus propias aplicaciones o Complementos a los que se puede llamar desde la herramienta de línea de comandos: eyeD3, import eyed3.

Socket no bloqueante

Por defecto al establecer la conexión de nuestros sockets en python el servidor queda total mente enchufado (conectado) con ese cliente en particular poniendo a todos los demás en espera, esto resulta ser un problema cuando queremos responderle a todos los clientes al tiempo, lograr ese *"Real Time"*.

Los sockets pueden configurarse como bloqueantes (por defecto) o hacerlos no bloqueantes mediante "fcntl" con el "flag" O_NONBLOCK. Y la diferencia es brutal.Un socket bloqueante detendrá la ejecución del programa cuando se haga una llamada a "recv" y no haya datos para recibir. Y cuando se reciban los datos la ejecución continuará.

Desarrollo de la práctica

En esta práctica se implementó una aplicación de cliente – servidor de un carrito de compra para la venta y descarga de canciones en formado mp3. El catálogo de productos está cargado por el servidor a partir de una carpeta. Las canciones poseen su multimedia y su respectiva imagen. El carrito de compras permite al usuario agregar, eliminar y comprar canciones de su respectivo carrito.

Tomando como punto de partida el desarrollo de la práctica 1, desarrolla lo que a continuación se te pide:

- El servidor deberá dar servicio en el puerto 9090 (de UDP)
- Una vez iniciado el servidor, deberá cargar en memoria el catálogo de productos que se ofertarán a los clientes (deberás crear un módulo que se encargue de cargar el catálogo de productos, ya sea desde un archivo, una base de datos o manualmente (utilizar serialización). Cada producto (canción) deberá contener metainformación, tal como nombre del álbum, año, artista, duración de la canción y precio de descarga).

- Implementa un método que permita enviar un archivo desde el servidor hacia el cliente. Este método se utilizará para descargar las imágenes de los productos que se desplegarán en el catálogo del cliente, así como las canciones a ser descargadas.
- La aplicación cliente además de mostrar los productos del catálogo, junto con su información, también deberá dar al usuario la posibilidad de agregar/eliminar/comprar, validando las existencias de cada producto.
- Deberá ser posible revisar el estado del carrito en cualquier momento, así como realizar la operación de compra.
- Una vez efectuada la compra, la aplicación cliente deberá permitir al usuario iniciar la descarga de las canciones adquiridas.
- En caso de que el cliente ya no desee realizar más compras, el usuario podrá cerrar la aplicación cliente y el servidor estará listo para aceptar al siguiente comprador.

Estructura del programa

El código consta de dos archivos, el primero es el "serverSocketUDP", el cual realizará las funciones de un servidor, es decir, enviará el catálogo de canciones, la metainformación de la canción escogida por el cliente puede agregar, eliminar canciones en el carrito, puede enviar el estado actual del carrito y finalmente descargar las canciones cuando el cliente desee realizar la compra de su carrito. Por otro lado, el segundo archivo es el "clientSocketUDP" el cual se le dará un menú de opciones para conocer la información de una canción, agregar y eliminar canciones de su carrito, ver el carrito y por último realizar la compra de su carrito. Estas acciones se ejecutarán de acuerdo con la opción escogida por el cliente. Además de estos archivos, se requiere de una carpeta que contenga las canciones considerado como el catalogo que posee el servidor, una carpeta vacía para el cliente, ya que será en este dónde reciba sus canciones descargadas.

Simulación

En las siguientes figuras se muestra la interfaz del cliente y del servidor junto a las funciones que puede realizar.

Servidor

Inicia el servidor en la figura 1, cargando con el catalogo obtenido de su carpeta. A continuación, se queda en espera hasta que el cliente sea activado.

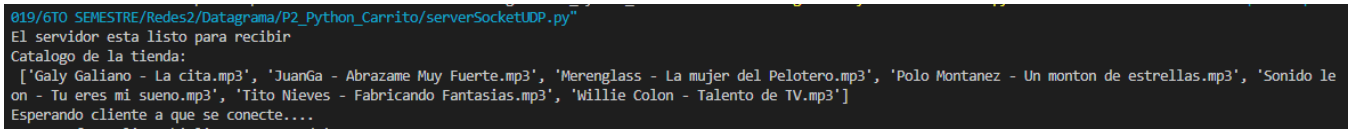
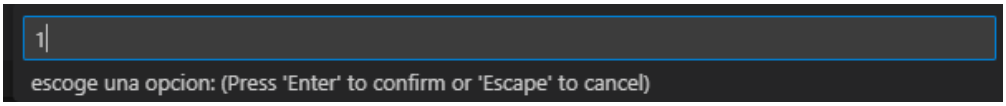


Figura 2. Servidor inicializado

Cliente

En la figura 2, se inicia el cliente y se muestra en la consola el catalogo enviado por el servidor. Además, se le da al usuario un menú de opciones para seleccionar y esta sea ejecutada. Por lo tanto, se empezará con la primera opción.

El usuario ingresa en numero 1, para la opción "Información de la cancion" y escribe el nombre de la canción "Polo Montanez - Un monton de estrellas.mp3".




```
Catalogo enviado por Servidor
 ['Galy Galiano - La cita.mp3', 'JuanGa - Abrazame Muy Fuerte.mp3', 'Merenglass - La mujer del Pelotero.mp3', 'Polo
Montanez - Un monton de estrellas.mp3', 'Sonido leon - Tu eres mi sueno.mp3', 'Tito Nieves - Fabricando Fantasias.mp3',
'Willie Colon - Talento de TV.mp3']
Bienvenido, tienda de musica
[1] Informacion de la cancion
[2] Agregar cancion
[3] Eliminar cancion
[4] Ver carrito
[5] Finalizar Compra
[6] Salir de la tienda
```

Figura 3. Cliente inicializado

Servidor

En la figura 3, el servidor recibe al cliente, junto con la opción seleccionada y el nombre de la canción para obtener su información. De tal forma que, el servidor comprobará que existe dicha canción en el catálogo, una vez aprobado, se mostrará en la terminal la metainformación de la respectiva canción la cual será enviada al cliente.

```
Message from Client:b'Cliente conectado'
Client IP Address:('127.0.0.1', 52472)
1
Informacion de una cancion
Message from Client 2:b'Polo Montanez - Un monton de estrellas.mp3'
Client IP Address 2:('127.0.0.1', 52473)
True
Envio de metadatos por eyed3 hacia el Cliente
Non standard genre name: Pop, Bachata, Regional Colombian, Música tropical
artista:  Polo Montañez
titulo:  Un Monton de Estrellas
album:  Guajiro natural
Fecha:  2004
Genero:  Pop, Bachata, Regional Colombian, Música tropical
```

Figura 4. Servidor envía metainformación de una canción

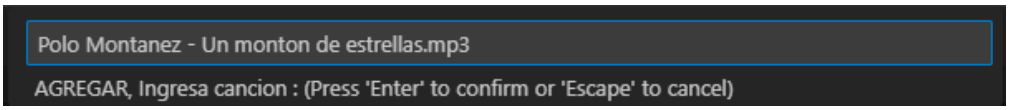
Cliente

En la figura 4, el cliente recibe la metainformación de la canción que deseó buscar.

```
Informacion de una cancion:  1
Informacion de la cancion Polo Montanez - Un monton de estrellas.mp3  es:
artista:  Polo Montañez
titulo:  Un Monton de Estrellas
album:  Guajiro natural
Fecha:  2004
Genero:  Pop, Bachata, Regional Colombian, Música tropical
```

Figura 5. Cliente recibe metainformación

Una vez finalizada la primera opción, en la figura 5 se le envía al usuario nuevamente el menú opciones para continuar dentro de la tienda, ahora se seleccionará la opción 2 “Agregar canción”, e ingresando la canción “Polo Montanez - Un monton de estrellas.mp3”, la opción y el nombre de la canción será enviado al servidor.




```
Catalogo enviado por Servidor
 ['Galy Galiano - La cita.mp3', 'JuanGa - Abrazame Muy Fuerte.mp3', 'Merenglass - La mujer del Pelotero.mp3', 'Polo
Montanez - Un monton de estrellas.mp3', 'Sonido leon - Tu eres mi sueno.mp3', 'Tito Nieves - Fabricando Fantasias.mp3',
'Willie Colon - Talento de TV.mp3']
Bienvenido, tienda de musica
[1] Informacion de la cancion
[2] Agregar cancion
[3] Eliminar cancion
[4] Ver carrito
[5] Finalizar Compra
[6] Salir de la tienda
Agregar cancion:  2
```

Figura 6. Cliente agrega la primera canción

Servidor

En la figura 6, el servidor recibe la opción 2 y el nombre de la canción por parte del usuario, por lo tanto, se evalúa que este pertenezca al catalogo y si es True entonces será agregado al carrito, finalmente se muestra en la consola un mensaje de aprobación y el carrito actual. Por lo tanto, se le enviará al usuario un mensaje de aprobación para informar que su canción ha sido agregada exitosamente.

```
2
Agregar cancion
Message from Client 3:b'Polo Montanez - Un monton de estrellas.mp3'
Client IP Address 3:('127.0.0.1', 52474)
True
La cancion  Polo Montanez - Un monton de estrellas.mp3  ha sido agregada al carrito
['Polo Montanez - Un monton de estrellas.mp3']
```

Figura 7. Servidor agrega canción al carrito

Cliente

El cliente recibe un mensaje sobre el estado de su carrito.

```
Agregar cancion:  2
(b'La cancion ', 'Polo Montanez - Un monton de estrellas.mp3', ' ha sido agregada al carrito')
```

Figura 8. Cliente recibe mensaje sobre el estado de su carrito

Esta opción se repetirá 2 veces más, con el fin de crecer el listado de canciones del carrito, a continuación, se mostrarán en las figuras 8 y 9 el estado final del carrito por parte del servidor, así como también del cliente.

```
Agregar cancion:  2
(b'La cancion ', 'Willie Colon - Talento de TV.mp3', ' ha sido agregada al carrito')
```

```
Agregar cancion:  2
(b'La cancion ', 'JuanGa - Abrazame Muy Fuerte.mp3', ' ha sido agregada al carrito')
```

Figura 9. Cliente agrega 2 canciones a su carrito.

```
2
Agregar cancion
Message from Client 3:b'Willie Colon - Talento de TV.mp3'
Client IP Address 3:('127.0.0.1', 52475)
True
La cancion  Willie Colon - Talento de TV.mp3  ha sido agregada al carrito
['Polo Montanez - Un monton de estrellas.mp3', 'Willie Colon - Talento de TV.mp3']
2
Agregar cancion
Message from Client 3:b'JuanGa - Abrazame Muy Fuerte.mp3'
Client IP Address 3:('127.0.0.1', 54757)
True
La cancion  JuanGa - Abrazame Muy Fuerte.mp3  ha sido agregada al carrito
['Polo Montanez - Un monton de estrellas.mp3', 'Willie Colon - Talento de TV.mp3', 'JuanGa - Abrazame Muy Fuerte.mp3']
```

Figura 10. Servidor agrega dos canciones al carrito del cliente

Una vez finalizada la segunda opción agregando 3 canciones, en la figura 10 se le envía al usuario nuevamente el menú opciones para continuar dentro de la tienda, ahora se seleccionará la opción 3 “Eliminar canción”. Por lo tanto, se le enviará al servidor un mensaje de sobre el deseo de eliminar una canción, así como la opción.

```
Catalogo enviado por Servidor
 ['Galy Galiano - La cita.mp3', 'JuanGa - Abrazame Muy Fuerte.mp3', 'Merenglass - La mujer del Pelotero.mp3', 'Polo
Montanez - Un monton de estrellas.mp3', 'Sonido leon - Tu eres mi sueno.mp3', 'Tito Nieves - Fabricando Fantasias.mp3',
'Willie Colon - Talento de TV.mp3']
Bienvenido, tienda de musica
[1] Informacion de la cancion
[2] Agregar cancion
[3] Eliminar cancion
[4] Ver carrito
[5] Finalizar Compra
[6] Salir de la tienda
Eliminar cancion del carrito:  3
```

Figura 11. Cliente selecciona eliminar canción del carrito

Servidor

En la figura 11, el servidor recibe la opción y el mensaje del usuario sobre la opción de eliminar una canción.

```
3
Eliminar cancion
b'Cliente solicita eliminar cancion del carrito por ID'
```

Figura 12. Servidor recibe mensaje del cliente sobre eliminar canción

Cliente

En la figura 12, el cliente recibe el carrito actual por el servidor y este selecciona por ID la canción por eliminar.

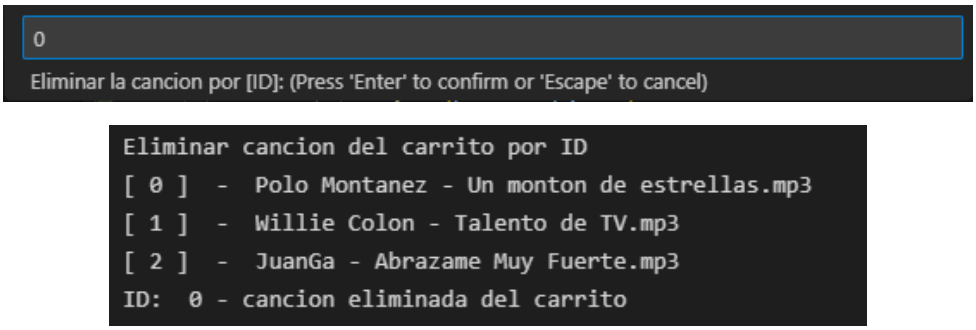


Figura 13. Cliente elimina canción del carrito por ID

Servidor

En la figura 13, el servidor recibe el ID de la canción por eliminar, por lo tanto, el servidor se encarga de extraerlo del carrito, mandando un mensaje de la canción eliminada del carrito.

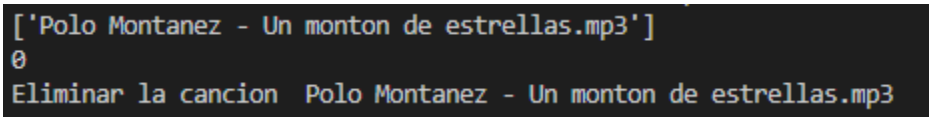


Figura 14. Servidor elimina canción del carrito

Cliente

Una vez finalizada la tercera opción, en la figura 14 se le envía al usuario nuevamente el menú opciones para continuar dentro de la tienda, ahora se seleccionará la opción 4 “Ver carrito”. Por lo tanto, se le enviará al usuario el carrito actual. En este ejemplo, se observará que la canción “Polo Montanez - Un monton de estrellas.mp3” ya no se encuentra en el carrito, debido a que anteriormente fue eliminado, analizar la figura 9 en la parte del carrito.

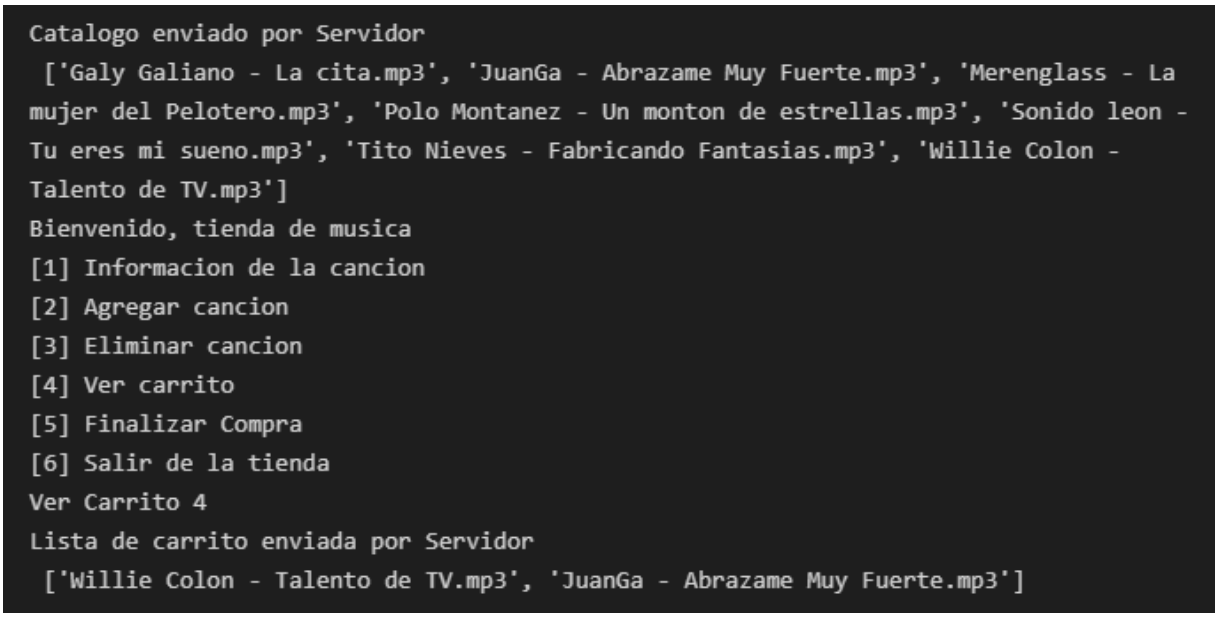


Figura 15. Cliente solicita ver carrito

Por último, en la figura 15 se le envía al usuario nuevamente el menú opciones para continuar dentro de la tienda, ahora se seleccionará la última opción 5 “Finalizar Compra”. Despues, le enviará al usuario la opción. Por lo tanto, el servidor enviará la lista actual del carrito, y cuando el servidor haya acabado de descargar todos los archivos, se notificará al cliente a través de un mensaje.

```
Catalogo enviado por Servidor
 ['Galy Galiano - La cita.mp3', 'JuanGa - Abrazame Muy Fuerte.mp3', 'Merenglass - La mujer
del Pelotero.mp3', 'Polo Montanez - Un monton de estrellas.mp3', 'Sonido leon - Tu eres mi
sueno.mp3', 'Tito Nieves - Fabricando Fantasias.mp3', 'Willie Colon - Talento de TV.mp3']
Bienvenido, tienda de musica
[1] Informacion de la cancion
[2] Agregar cancion
[3] Eliminar cancion
[4] Ver carrito
[5] Finalizar Compra
[6] Salir de la tienda
Finalizar compra 5
Lista de carrito para comprar por el Servidor
 ['Willie Colon - Talento de TV.mp3', 'JuanGa - Abrazame Muy Fuerte.mp3']
Willie Colon - Talento de TV.mp3
Sending Willie Colon - Talento de TV.mp3 ...
(b'La cancion ', 'Willie Colon - Talento de TV.mp3', ' ha sido descargada')
```

Figura 16. Cliente solicita finalizar compra

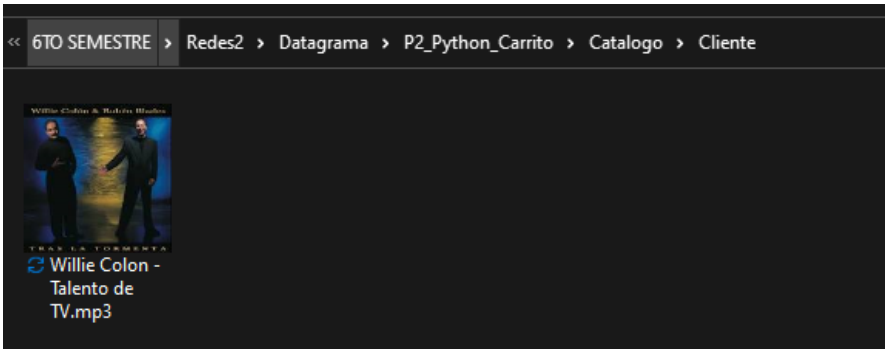
Servidor

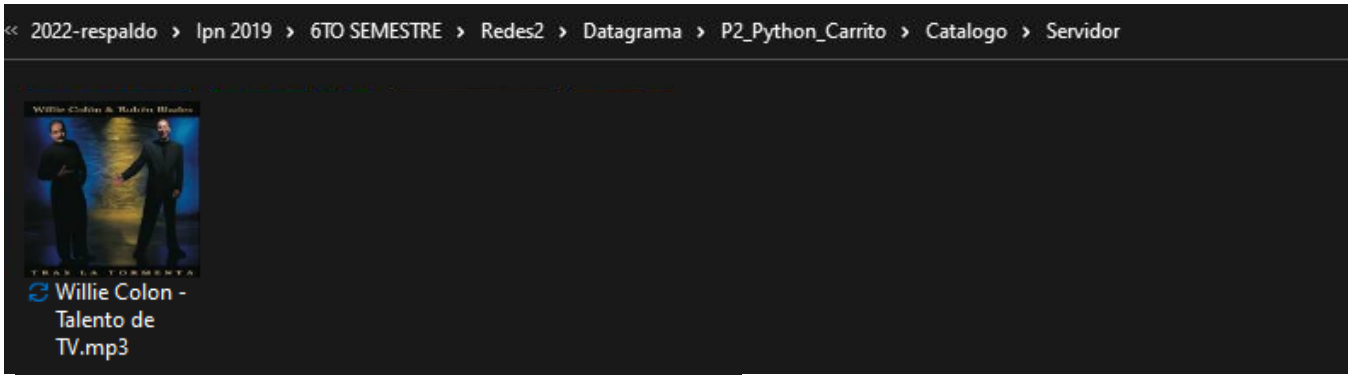
En la figura 16, el servidor a través de un método irá descargando la información y este pasará a la carpeta del cliente donde se descargarán todas las canciones que hay en el carrito.

```
File name b'Willie Colon - Talento de TV.mp3'
b'Willie Colon - Talento de TV.mp3' Finish!
La cancion  Willie Colon - Talento de TV.mp3  ha sido descargada
```

Figura 17. Servidor descarga canciones y las almacena en la carpeta del cliente

Resultado de la descarga de canciones





Preguntas sobre el resultado final de la práctica

1. ¿Qué es serialización?

Es un mecanismo que facilita el almacenamiento y transmisión del estado de un objeto. La serialización de un objeto consiste en generar una secuencia de bytes lista para su almacenamiento o transmisión. Después, mediante la deserialización, se puede reconstruir el estado original del objeto.

2. ¿En qué difiere del marshalling?

La serialización en realidad, cuando serializa un objeto, solo los datos del miembro dentro de ese objeto se escriben en el flujo de bytes, es decir, la serialización se trata de copiar datos estructurados hacia o desde una forma primitiva, como un flujo de bytes como lo muestra figura 2.

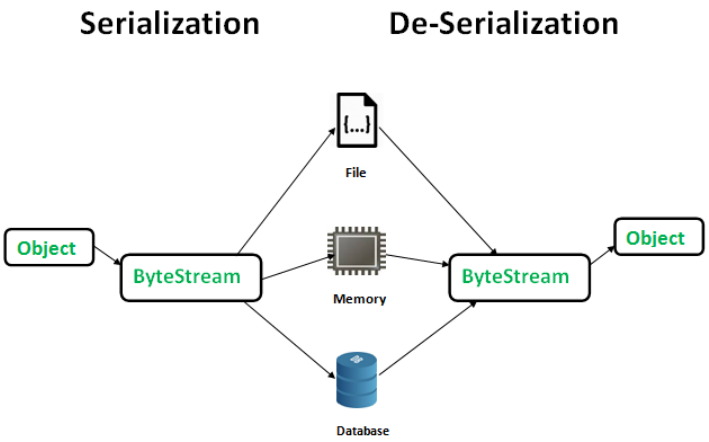


Figura 2. Serialización

Por otro lado, el Marshalling es cuando se desea pasar un objeto a objetos remotos, es decir, es el proceso de crear un puente entre código administrado y no administrado. Marshalling es útil cuando trabaja con código no administrado, ya sea que esté trabajando con API de Windows o componentes COM.

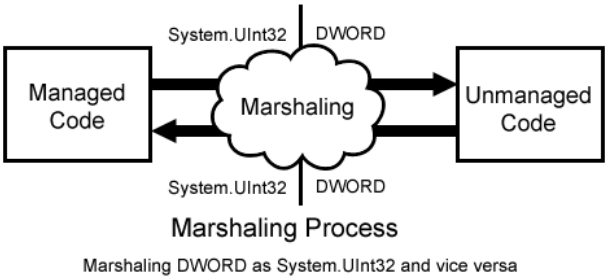


Figura 3. Marshalling

Por lo tanto, se difieren en cuando la semántica, pues el Marshalling implica mover los datos, pero no los transforma, entonces se refiere a la acción de almacenar el estado de un objeto junto con su código, mientras que serializar solamente crea copias de objetos como los flujos de bytes.

3. ¿Por qué es importante la serialización?

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

- Permite hacer llamados a procedimientos remotos (RPC).
- Sirve para identificar cambios de datos en ejecución.
- Podemos almacenar 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 Individuales

Ruvalcaba Flores Martha Catalina

Para la realización de la práctica “Carrito de compras musical”, el equipo se propuso como reto implementarlo en el lenguaje de programación Python. En base a una previa investigación el equipo observó que los sockets no bloqueantes pueden ser implementados a través de un diferente número de canales, tales como: sockets de dominio UNIX, TCP, UDP, etc. Así mismo, el socket no bloqueante permite que el servidor no espere la respuesta de un cliente, puesto que se implementa un setTimeout para limitar el tiempo de respuesta. Para cumplir el objetivo de la práctica y el tema a implementar, se utilizó el socket de dominio UDP, manejando así el uso de paquetes de datagramas. Para su funcionamiento, se descargaron varias canciones en formato mp3 junto con su metainformación ya que es uno de los requisitos de la práctica, además de que se empleó el modelo de cliente – servidor para construir la aplicación en red. Para el desarrollo de la practica fue importante designar los roles que podía realizar el servidor y el cliente, es decir, el servidor tiene las funciones de mostrar el catálogo, de enviar la metainformación de una canción, de agregar la canción al carrito, de eliminar la canción del carrito, de enviar el estado actual del carrito y finalmente de realizar la descarga de canciones contenidas en el carrito. Por otro lado, el cliente tiene las funciones de seleccionar alguna opción de la tienda, así como de escribir las canciones que desea ya sea para obtener su información, para agregar, eliminar y ver el carrito, finalmente realizar la compra de su carrito. Dicha comunicación entre cliente y servidor fue a través de la serialización, haciendo uso de banderas para indicar la opción seleccionada por el cliente.

Sandoval Hernández Eduardo

Esta práctica resultó ser un tanto complicada y sencilla a la vez, al principio intentamos modificar las prácticas 1 y 2 puesto que en teoría solo había que adaptarlas con sockets no bloqueantes, sin embargo, por la forma en la que implementamos dichas prácticas se complicó la codificación de las mismas con sockets lo no bloqueantes así que optamos por intentar con la práctica del servidor HTTP pues al estar en programada en Python esta sería más fácil de modificar para implementar sockets no bloqueantes, solo teniendo una complicación con la función setTimeout ya que esta funciona de manera distinta con sockets no bloqueantes a diferencia de los bloqueantes, aun así no representó mayor problema. El desempeño de la aplicación mejoró puesto que en esta ocasión el servidor no tiene que esperar la respuesta del cliente para seguir trabajando y aceptando más clientes. Seguro que el conocimiento del uso de los 2 tipos de sockets nos servirá para implementaciones futuras.

Bibliografía

Pawan, V. (2021, June 10). *Serialization vs marshalling*. Linkedin.com. https://www.linkedin.com/pulse/serialization-vs-marshalling-pawan-verma

(N.d.). Ugr.Es. Retrieved December 8, 2022, from https://elvex.ugr.es/decsai/java/pdf/C3-serializable.pdf

UDP - Client and Server example programs in Python. (n.d.). Pythontic.com. Retrieved December 8, 2022, from https://pythontic.com/modules/socket/udp-client-server-example

Ruelas, U. (n.d.). ¿Qué es la serialización o marshalling? Codingornot.com. Retrieved December 8, 2022, from https://codingornot.com/que-es-la-serializacion-o-marshalling

Fernandez, R. (2018, October 30). Programación de redes en Python: Sockets. ▷ Cursos de Programación de 0 a Experto © Garantizados. https://unipython.com/programacion-de-redes-en-python-sockets/

Protocolo de datagrama de usuario (UDP). (n.d.). Khan Academy. Retrieved December 8, 2022, from https://es.khanacademy.org/computing/ap-computer-science-principles/the-internet/x2d2f703b37b450a3:transporting-packets/a/user-datagram-protocol-udp

Welcome to eyeD3 — eyeD3 0.9.4 documentation. (n.d.). Readthedocs.Io. Retrieved December 8, 2022, from https://eyed3.readthedocs.io/en/latest/

Anexo 1: Código de serverSocketUDP.py

```
#--------------------------------------------#
# RUVALCABA FLORES MARTHA CATALINA
# SANDOVAL HERNANDEZ EDUARDO
# 6CM1
# PRACTICA: CARRITO DE COMPRAS
#--------------------------------------------#


#--------------------------------------------#
# Librerias
#--------------------------------------------#
```

```python
from socket import *
import os
import pickle
import eyed3
from pathlib import Path

#--------------------------------------------#
# Librerias para recibir archivo y copiar archivo
#--------------------------------------------#
import select
import shutil
timeout = 3

#--------------------------------------------#
# Servidor
#--------------------------------------------#
serverPort = 9090
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.setblocking(False)

serverSocket.bind(('',serverPort)) ##127.0.0.1 Standard loopback interface address (localhost)


#la función setblocking recibe como parámetro un valor booleano el cual indica si el
#socket podrá desenchufarse del cliente

#Se diferencia en que send(), recv(), connect() y accept()
# pueden retornar sin haber hecho nada.

#--------------------------------------------#
# FUNCION = CONECTAR CON EL CLIENTE
#--------------------------------------------#
def conectarCliente(catalogo):

  serverSocket.setblocking(False)
  serverSocket.settimeout(20)

  bytesAddressPair = serverSocket.recvfrom(65000)
  message = bytesAddressPair[0]
  address = bytesAddressPair[1]
  clientMsg = "Message from Client:{}".format(message)
  clientIP  = "Client IP Address:{}".format(address)
  print(clientMsg)
  print(clientIP)
  serverSocket.sendto(pickle.dumps(catalogo),address) #enviar bytes el catalogo
```

```
#-------------------------------------------#
# FUNCION = INFORMACION DE LA CANCION, METADATOS
#-------------------------------------------#
def infoCancion(catalogo):
  serverSocket.setblocking(False)
  serverSocket.settimeout(20)
  metainformacion = []
  bytesAddressPair = serverSocket.recvfrom(65000) #recibir cliente data
  cancion = bytesAddressPair[0]
  address = bytesAddressPair[1]
  clientMsg = "Message from Client 2:{}".format(cancion)
  clientIP  = "Client IP Address 2:{}".format(address)
  print(clientMsg)
  print(clientIP)
  cancion = str(cancion, 'UTF-8') #de bytes a string
  print(cancion in catalogo)
  if True :
    print("Envio de metadatos por eyed3 hacia el Cliente")
    ruta = Path("Catalogo/Servidor")
    ncancion = cancion
    ruta_con_variable = ruta.joinpath(ncancion)
    audiofile = eyed3.load(ruta_con_variable)
    metainformacion.append(audiofile.tag.artist)
    metainformacion.append(audiofile.tag.title)
    metainformacion.append(audiofile.tag.album)
    metainformacion.append(audiofile.tag.recording_date)
    metainformacion.append(audiofile.tag.genre)
    print ("artista: ",metainformacion[0])
    print ("titulo: ",metainformacion[1])
    print ("album: ",metainformacion[2])
    print ("Fecha: ",metainformacion[3])
    print ("Genero: ",metainformacion[4])
  serverSocket.sendto(pickle.dumps(metainformacion),address)


#-------------------------------------------#
# FUNCION = AGREGAR CANCION
#-------------------------------------------#
def agregarCancion(catalogo,carrito):
  serverSocket.setblocking(False)
  serverSocket.settimeout(20)
  bytesAddressPair = serverSocket.recvfrom(65000) #recibir cliente data
  cancion = bytesAddressPair[0]
  address = bytesAddressPair[1]
  clientMsg = "Message from Client 3:{}".format(cancion)
  clientIP  = "Client IP Address 3:{}".format(address)
  print(clientMsg)
```

```
  print(clientIP)
  cancion = str(cancion, 'UTF-8') #de bytes a string
  print(cancion in catalogo)
  if True :
    if cancion in carrito:
      print("La cancion ",cancion," ya esta en el carrito")
      mensaje=b"La cancion ",cancion," ya esta en el carrito"
      serverSocket.sendto(pickle.dumps(mensaje),address) #envia al Cl mensaje en bytes
    else:
      carrito.append(cancion)
      print("La cancion ",cancion," ha sido agregada al carrito")
      mensaje=b"La cancion ",cancion," ha sido agregada al carrito"
      serverSocket.sendto(pickle.dumps(mensaje),address) #envia al Cl mensaje en bytes

#-------------------------------------------#
# FUNCION = VER CARRITO
#-------------------------------------------#
def verCarrito(carrito):
  serverSocket.setblocking(False)
  serverSocket.settimeout(20)
  bytesAddressPair2 = serverSocket.recvfrom(65000) #recibir cliente data
  avisoCl = bytesAddressPair2[0]
  address = bytesAddressPair2[1]
  bytesToObject3= pickle.loads(avisoCl)
  print (bytesToObject3)
  print(carrito)
  serverSocket.sendto(pickle.dumps(carrito),address)

#-------------------------------------------#
# FUNCION = ELIMINAR CANCION DEL CARRITO
#-------------------------------------------#
def eliminarCancionC(catalogo,carrito):
  serverSocket.setblocking(False)
  serverSocket.settimeout(20)
  bytesAddressPair3 = serverSocket.recvfrom(65000) #recibir cliente data
  avisoCl = bytesAddressPair3[0]
  address = bytesAddressPair3[1]
  bytesToObject3= pickle.loads(avisoCl)
  print (bytesToObject3)

  print(carrito)
  serverSocket.sendto(pickle.dumps(carrito),address)

  eliminarIDS = serverSocket.recvfrom(65000) #recibir cliente data
  eliminarIDS = int.from_bytes(eliminarIDS[0], "big")
  print(eliminarIDS)
  for i in range(len(catalogo)):
```

```
    if i == eliminarIDS:
      print("Eliminar la cancion ",carrito[eliminarIDS])
      carrito = carrito.pop(eliminarIDS)

  print("carritoNuevo",carrito)

#-------------------------------------------#
# FUNCION = GENERAR TICKET Y DESCARGAR
#-------------------------------------------#
def generarTicket(carrito):
  serverSocket.setblocking(False)
  serverSocket.settimeout(20)
  bytesAddressPair3 = serverSocket.recvfrom(65000) #recibir cliente data
  avisoCl = bytesAddressPair3[0]
  address = bytesAddressPair3[1]
  bytesToObject3= pickle.loads(avisoCl)
  print (bytesToObject3)

  print(carrito)
  serverSocket.sendto(pickle.dumps(carrito),address)

  while True:
    data, address = serverSocket.recvfrom(1024)
    if data:
      print("File name", data)
      file_name = data.strip()
    f = open(file_name, 'wb')
    while True:
        ready = select.select([serverSocket], [], [], timeout)
        if ready[0]:
            data, address = serverSocket.recvfrom(1024)
            f.write(data)
        else:
            print("%s Finish!" % file_name)
            f.close()
            break
    rutaO = Path("Catalogo/Servidor")
    rutaD = Path("Catalogo/Cliente")
    fileBtoS = str(file_name, 'UTF-8') #de bytes a string
    descargarF = fileBtoS
    ruta_con_variable = rutaO.joinpath(descargarF)
    shutil.copy2(ruta_con_variable,rutaD)
    print("La cancion ",descargarF," ha sido descargada")
    mensaje=b"La cancion ",descargarF," ha sido descargada"
    serverSocket.sendto(pickle.dumps(mensaje),address) #envia al Cl mensaje en bytes
    print("")
    f.close()
```

```
#-------------------------------------------#
# MENU DE OPCIONES
#-------------------------------------------#
print("El servidor esta listo para recibir")
carrito = []
catalogo = os.listdir('Catalogo/Servidor')
print("Catalogo de la tienda: \n",catalogo)
print("Esperando cliente a que se conecte....")
conectarCliente(catalogo)

#-------------------------------------------#
# Banderas de acuerdo al menu del ServerSocketUDP
#-------------------------------------------#
bandera = serverSocket.recvfrom(65000) #recibir cliente data
banderaI = int.from_bytes(bandera[0], "big")
print(banderaI)

while banderaI !=0:
  if banderaI == 1:
    print("Informacion de una cancion")
    infoCancion(catalogo)

  elif banderaI == 2:
    print("Agregar cancion")
    agregarCancion(catalogo,carrito)
    carritoN = carrito
    print(carritoN)
  elif banderaI == 3:
    print("Eliminar cancion")
    eliminarCancionC(catalogo,carritoN)

  elif banderaI == 4:
    print("Ver carrito")
    verCarrito(carritoN)

  elif banderaI == 5:
    print("Finalizar compra") #Genera Ticket y descarga canciones en la carpeta
    generarTicket(carritoN)

  elif banderaI == 6:
    bytesAddressPair3 = serverSocket.recvfrom(65000) #recibir cliente data
    avisoCl = bytesAddressPair3[0]
    bytesToObject3= pickle.loads(avisoCl)
    print (bytesToObject3)
    print("Servidor finaliza")
    break
```

```
  else:
    print("Opcion invalida")

  bandera = serverSocket.recvfrom(65000) #recibir cliente data
  banderaI = int.from_bytes(bandera[0], "big")
  print(banderaI)
```

Anexo 2: Código de clientSocketUDP.py

```
#-------------------------------------------#
# RUVALCABA FLORES MARTHA CATALINA
# SANDOVAL HERNANDEZ EDUARDO
# 6CM1
# PRACTICA: CARRITO DE COMPRAS
#-------------------------------------------#

#-------------------------------------------#
# Librerias
#-------------------------------------------#
from socket import *
import pickle
from playsound import playsound

import time
import sys
import os
from pathlib import Path

#-------------------------------------------#
# FUNCION = MENU DE OPCIONES
#-------------------------------------------#
def menu():
  print("Bienvenido, tienda de musica")
  print("[1] Informacion de la cancion ")
  print("[2] Agregar cancion ")
  print("[3] Eliminar cancion ")
  print("[4] Ver carrito ")
  print("[5] Finalizar Compra ")
  print("[6] Salir de la tienda ")
#-------------------------------------------#
# Cliente socket UDP
#-------------------------------------------#
serverName = '127.0.0.1' #localhost
serverPort = 9090
clientSocket = socket(AF_INET,SOCK_DGRAM)
```

```
clientSocket.setblocking(False)
clientSocket.settimeout(10)
file_name = sys.argv[1]

clientSocket.sendto("Cliente conectado".encode(),(serverName,serverPort))
catalogo, serverAddress = clientSocket.recvfrom(65000) #recibir del servidor catalogo
bytesToObject= pickle.loads(catalogo) #bytes a lista
catalogo = bytesToObject
print ("Catalogo enviado por Servidor\n",catalogo)

#-------------------------------------------#
# Menu opciones
#-------------------------------------------#
opcion = 0
carrito = []
menu()
opcion = int(input("escoge una opcion: "))

while opcion !=0:
  clientSocket = socket(AF_INET,SOCK_DGRAM)
  clientSocket.setblocking(False)
  clientSocket.settimeout(10)
  if opcion == 1 : # Informacion de la cancion
    print("Informacion de una cancion: ",opcion)
    opcionB = opcion.to_bytes(2, byteorder="big")
    clientSocket.sendto(opcionB,(serverName,serverPort)) #envia la opcion para la bandera 1

    #------Codigo de la opcion 1-----------------#
    nombreC = input("INFO, Ingresa cancion : ")
    clientSocket.sendto(nombreC.encode(),(serverName,serverPort))
    metainformacion, serverAddress = clientSocket.recvfrom(65000)
    bytesToObject2= pickle.loads(metainformacion)
    print("Informacion de la cancion",nombreC," es: ")
    print ("artista: ",bytesToObject2[0])
    print ("titulo: ",bytesToObject2[1])
    print ("album: ",bytesToObject2[2])
    print ("Fecha: ",bytesToObject2[3])
    print ("Genero: ",bytesToObject2[4])


  elif opcion == 2 : # Agregar cancion
    print("Agregar cancion: ",opcion)
    opcionB = opcion.to_bytes(2, byteorder="big")
    clientSocket.sendto(opcionB,(serverName,serverPort)) #envia la opcion para la bandera 1

    #------Codigo de la opcion 2-----------------#
    AnombreC = input("AGREGAR, Ingresa cancion : ")
```

```
    clientSocket.sendto(AnombreC.encode(),(serverName,serverPort))
    mensaje, serverAddress = clientSocket.recvfrom(65000)
    bytesToObject3= pickle.loads(mensaje)
    print (bytesToObject3)

  elif opcion == 3 : # Eliminar cancion
    print("Eliminar cancion del carrito: ",opcion)
    opcionB = opcion.to_bytes(2, byteorder="big")
    clientSocket.sendto(opcionB,(serverName,serverPort)) #envia la opcion para la bandera 1

    #------Codigo de la opcion 3------------------#
    aviso=b"Cliente solicita eliminar cancion del carrito por ID"
    clientSocket.sendto(pickle.dumps(aviso),(serverName,serverPort))

    print("Eliminar cancion del carrito por ID")
    carrito, serverAddress = clientSocket.recvfrom(65000) #recibir del servidor catalogo
    bytesToObject5= pickle.loads(carrito) #bytes a lista
    for i in range(len(bytesToObject5)):
      print("[",i,"]"," - ",bytesToObject5[i])

    eliminarID = int(input("Eliminar la cancion por [ID]: "))
    print("ID: ",eliminarID,"- cancion eliminada del carrito")
    eliminarID = eliminarID.to_bytes(2, byteorder="big")
    clientSocket.sendto(eliminarID,(serverName,serverPort)) #envia la opcion para la bandera 1


  elif opcion == 4 : # Ver carrito
    print("Ver Carrito",opcion)
    opcionB = opcion.to_bytes(2, byteorder="big")
    clientSocket.sendto(opcionB,(serverName,serverPort)) #envia la opcion para la bandera 1

    #------Codigo de la opcion 4------------------#
    aviso=b"Cliente solicita ver carrito"
    clientSocket.sendto(pickle.dumps(aviso),(serverName,serverPort))

    carrito, serverAddress = clientSocket.recvfrom(65000) #recibir del servidor catalogo
    bytesToObject= pickle.loads(carrito) #bytes a lista
    print ("Lista de carrito enviada por Servidor\n",bytesToObject )

  elif opcion == 5 : # Finalizar compra
    print("Finalizar compra",opcion)
    opcionB = opcion.to_bytes(2, byteorder="big")
    clientSocket.sendto(opcionB,(serverName,serverPort)) #envia la opcion para la bandera 1

    #------Codigo de la opcion 5------------------#
    aviso=b"Cliente solicita generar ticket"
    clientSocket.sendto(pickle.dumps(aviso),(serverName,serverPort))
```

```
  carrito, serverAddress = clientSocket.recvfrom(65000) #recibir del servidor catalogo
  bytesToObject= pickle.loads(carrito) #bytes a lista
  print ("Lista de carrito para comprar por el Servidor\n",bytesToObject )

  for i in range(len(bytesToObject)):
    file_name = bytesToObject[i]
    file_name = str(file_name)
    print(file_name)
    clientSocket.sendto(file_name.encode(),(serverName,serverPort))
    print("Sending %s ..." % file_name)


    filePath = os.path.abspath(file_name)
    #print(filePath)
    f = open(filePath,'r', encoding="utf-8", errors='ignore')
    data = f.read(65000)
    while(data):
        if(clientSocket.sendto(data,(serverName,serverPort))):
            data = f.read(65000)
            time.sleep(0.02) # Give receiver a bit time to save

    mensajeDescarga, serverAddress = clientSocket.recvfrom(65000)
    bytesToObject6= pickle.loads(mensajeDescarga)
    print (bytesToObject6)
  clientSocket.close()
  f.close()

elif opcion == 6:
  print("Salir tienda",opcion)
  opcionB = opcion.to_bytes(2, byteorder="big")
  clientSocket.sendto(opcionB,(serverName,serverPort)) #envia la opcion para la bandera 1

  aviso=b"Cliente sale de la tienda"
  clientSocket.sendto(pickle.dumps(aviso),(serverName,serverPort))
  clientSocket.close()
  break

else:
  print("invalido")

print()
print ("Catalogo enviado por Servidor\n",catalogo)
menu()
opcion = int(input("escoge una opcion: "))
```