



**Instituto Politécnico
Nacional**

**Escuela Superior de
Cómputo**



Aplicaciones para Comunicaciones en Red

Docente: **M. en C. Rangel González Josué .**

Práctica 4. ***Protocolo TCP y UDP***

Presentan:

César Uriel Hernández Castellanos

Mauricio Joel Martínez Islas

Carlos Pimentel González

Fecha de entrega: **2 de marzo de 2020**

Grupo: **3CV5**

Índice

1. Introducción.	3
1.1. Modelo cliente-servidor	3
1.2. Características del modelo cliente-servidor	3
1.2.1. Características del cliente	3
1.2.2. Características del servidor	4
1.2.3. Características generales modelo cliente-servidor	4
1.3. Definición de Socket	5
1.4. Atributos del Socket	5
1.5. Tipos de Socket	5
1.5.1. Socket orientado a conexión	5
1.5.2. Socket no orientado a conexión	6
1.5.3. UDP(User Datagram Protocol)	6
1.5.4. TCP(Transmission Control Protocol)	6
2. Descripción del problema.	8
2.1. Implementando el modelo cliente servidor en TCP y UDP	8
3. Descripción de la solución.	9
4. Salida del programa.	16
5. Conclusiones.	17

Índice de figuras

1.	Diagrama cliente-servidor via internet	3
2.	Cliente	3
3.	Servidor	4
4.	Modelo cliente-servidor	4
5.	Sockets cliente y servidor	5
6.	Protocolo TCP y UDP	6
7.	Pasos a seguir para implementar UDP	7
8.	Pasos a seguir para implementar TCP	7
9.	Gráfico que nos muestra el funcionamiento de la aplicación	8
10.	Función que elige entre un protocolo UDP o TCP	9
11.	Funcionamiento del script que se encarga de recolectar los archivos del cliente	9
12.	Función que permite crear una lista de hilos	10
13.	Función que permite particionar cada archivo	11
14.	Función que permite aceptar conexiones	12
15.	Función que permite aceptar conexiones	13
16.	Función que envía las banderas al cliente	14
17.	Función que indica cuando enviar los archivos	15
18.	Salida de la terminal de Servidor y Cliente.	16
19.	Backup de archivos e el servidor.	16

1. Introducción.

1.1. Modelo cliente-servidor

La arquitectura cliente-servidor es un modelo de diseño de software en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servicios, y los demandantes, llamados cliente. Un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta.

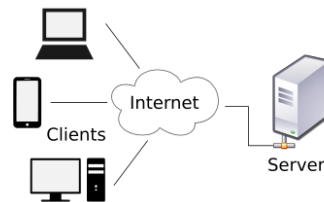


Figura 1: Diagrama cliente-servidor via internet

1.2. Características del modelo cliente-servidor

1.2.1. Características del cliente

En el modelo cliente-servidor el remitente de una solicitud es conocido como cliente. Sus características son:

- Es quien inicia solicitudes o peticiones, tienen por tanto un papel activo en la comunicación
- Espera y recibe las respuestas del servidor.
- Por lo general, puede conectarse a varios servidores a la vez.
- Normalmente interactúa directamente con los usuarios finales mediante una interfaz gráfica de usuario.



Figura 2: Cliente

1.2.2. Características del servidor

Al receptor de la solicitud enviada por el cliente se conoce como servidor. Sus características son:

- Al iniciarse esperan a que lleguen las solicitudes de los clientes, desempeñan entonces un papel pasivo en la comunicación
- Tras la recepción de una solicitud, la procesan y luego envían la respuesta al cliente.
- Por lo general, acepta las conexiones de un gran número de clientes



Figura 3: Servidor

1.2.3. Características generales modelo cliente-servidor

- El Cliente y el Servidor pueden actuar como una sola entidad y también pueden actuar como entidades separadas, realizando actividades o tareas independientes.
- Las funciones de Cliente y Servidor pueden estar en plataformas separadas, o en la misma plataforma.
- Cada plataforma puede ser escalable independientemente
- Su representación típica es un centro de trabajo

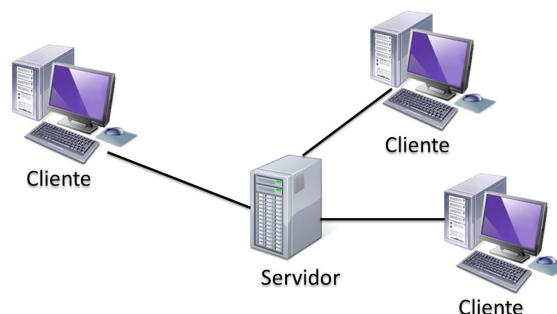


Figura 4: Modelo cliente-servidor

1.3. Definición de Socket

Un socket es la conexión que se establece entre dos aplicaciones en dos hosts diferentes, una aplicación cliente en un host y otra aplicación servidor en otro (siguiendo la arquitectura cliente-servidor) a través de una red (LAN, WAN, . . .)

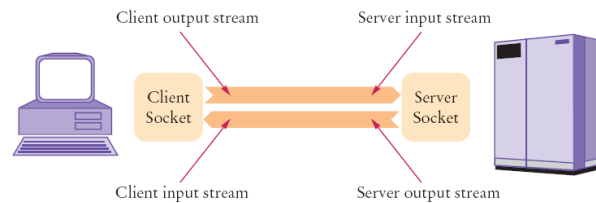


Figura 5: Sockets cliente y servidor

1.4. Atributos del Socket

- **Dominio:** Especifica el medio de comunicación de la red que el socket utilizará
- **Protocolo:** Especifica que protocolo se va usar
- **Tipo:** Los protocolos de internet proveen dos niveles distintos de servicio: flujo y datagramas

1.5. Tipos de Socket

Existen diferentes tipos de sockets. Los dos más empleados son:

- Socket orientado a conexión
- Socket no orientado a conexión

1.5.1. Socket orientado a conexión

Al conectar se realiza una búsqueda de un camino libre entre origen y destino. Se mantiene el camino en toda la conexión

- Primero se establece correctamente la conexión.
- Ninguno de los dos puede transmitir datos.
- Se usa el protocolo TCP del protocolo TCP/IP para gestionar la conexión.
- Se garantiza que todos los datos van a llegar de un programa al otro correctamente.
- Se utiliza cuando la información a transmitir es importante, no se puede perder ningún dato.
- No importa que los programas se queden bloqueados esperando o transmitiendo datos.

1.5.2. Socket no orientado a conexión

No se fija un camino. Cada paquete podrá ir por cualquier sitio.

- Es el llamado protocolo UDP.
- No es necesarios que los programas se conecten.
- Cualquiera de ellos puede transmitir datos en cualquier momento, independiente de que el otro programa esté escuchando o no.
- Se utilizan cuando es muy importtande que el programa no se quede bloqueado.
- No importa que se pierdan datos.

1.5.3. UDP(User Datagram Protocol)

- Protocolo de transporte sin conexión.
- No garantiza un servicio extremo a extremo fiable.
- No controla la pérdida de paquetes, los errores o la duplicidad
- Utilizado en aplicaciones en que la rapidez en el entrega es más importante que la seguridad.

1.5.4. TCP(Transmission Control Protocol)

- Permite colocar los datagramas nuevamente en orden cuando vienen del protocolo IP.
- Permite que el monitoreo del flujo de los datos y así evita la saturación de la red.
- permite que los datos se formen en segmentos de longitud variada para .^{en}tregarlos.^{al} protocolo IP.
- Permite multiplexar los datos, es decir, que la información que viene de diferentes fuentes. .

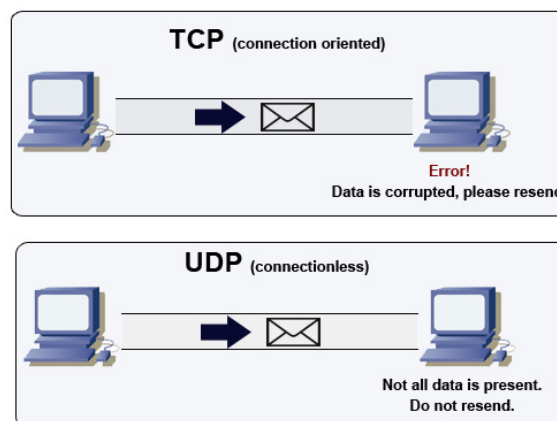


Figura 6: Protocolo TCP y UDP

UDP Client-Server

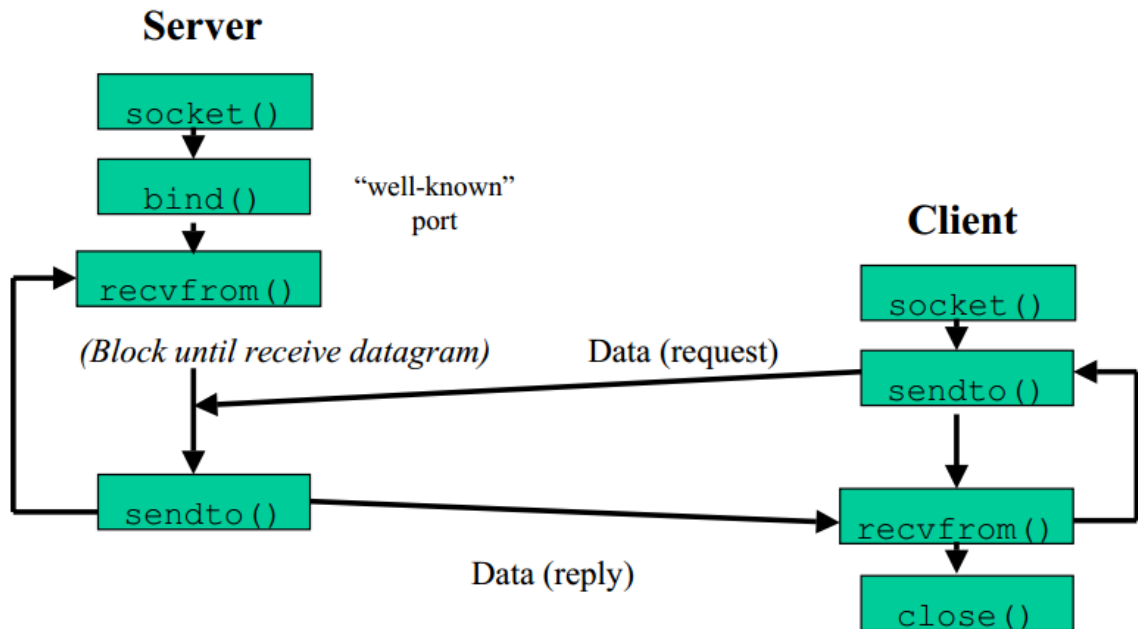


Figura 7: Pasos a seguir para implementar UDP

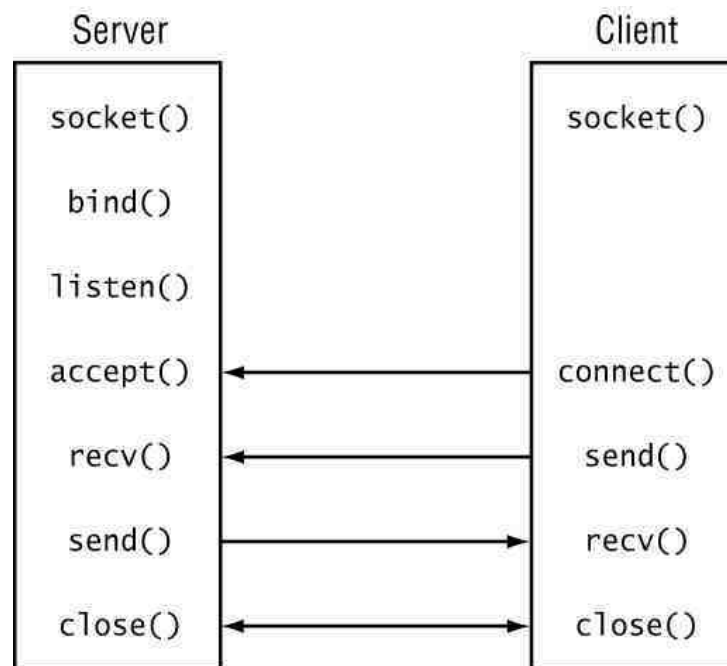


Figura 8: Pasos a seguir para implementar TCP

2. Descripción del problema.

2.1. Implementando el modelo cliente servidor en TCP y UDP

Implementar un menú el cual tenga la opción de transmitir archivos por medio del protocolo UDP y TCP.

Generar un servidor **no** bloqueante orientado a conexión, con el siguiente funcionamiento. Cada cliente se encargará de recolectar todas las imágenes, pdf, audios y videos que existan en la pc donde se estan ejecutando.

Enviando la recopilación de información al servidor, el cual detectará de que cliente viene la información, u la almacenará en una carpeta en una carpeta, que tendrá como nombre la dirección IP del cliente.

- **Restricción:** No se pueden enviar archivos completos en un solo envío. Cada archivo se enviará en 4 partes.

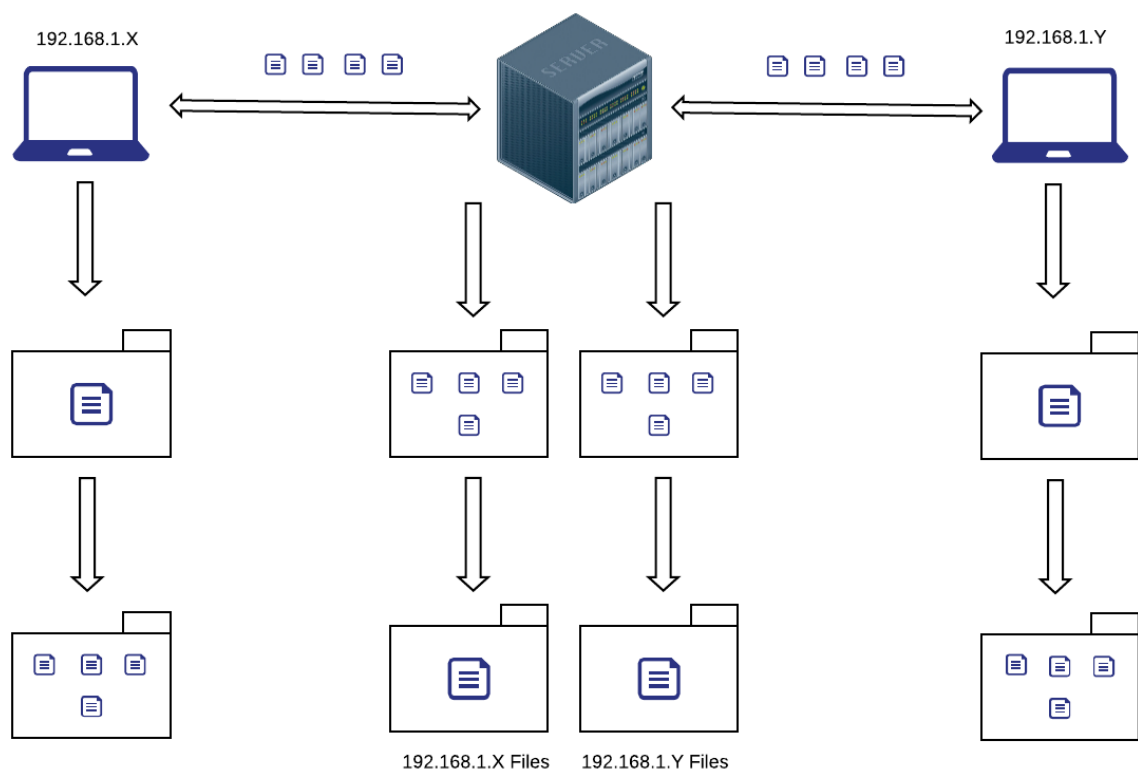


Figura 9: Gráfico que nos muestra el funcionamiento de la aplicación

3. Descripción de la solución.

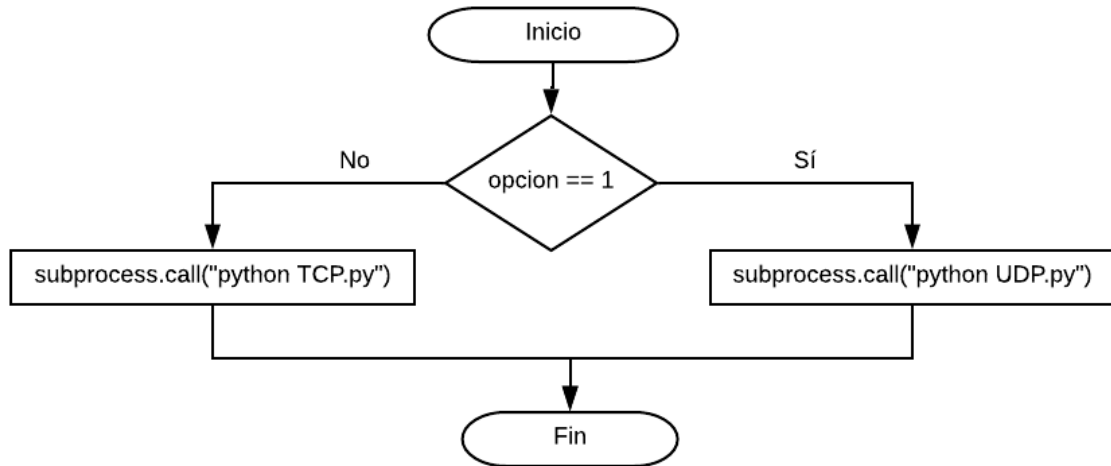


Figura 10: Función que elige entre un protocolo UDP o TCP

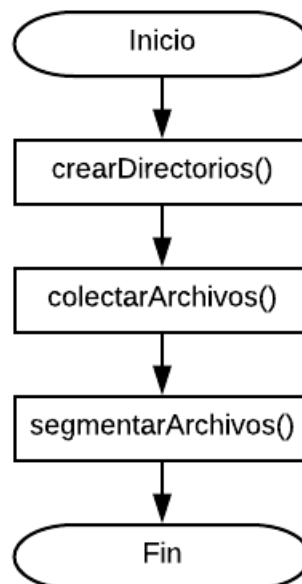


Figura 11: Funcionamiento del script que se encarga de recolectar los archivos del cliente

En la figura anterior es posible observar el funcionamiento de la función que se encarga de recolectar los archivos del cliente y guardarlo en una carpeta con el nombre de "secretFiles"

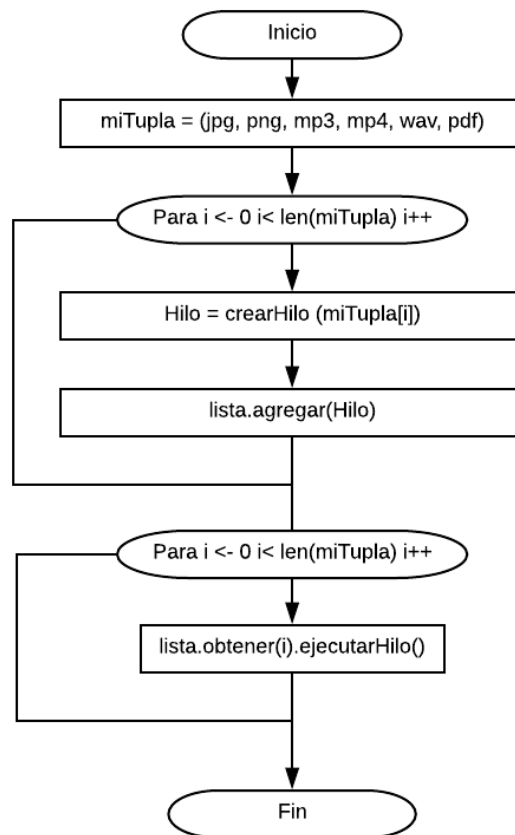


Figura 12: Función que permite crear una lista de hilos

En la figura 10 es posible observar una función que tiene como finalidad crear una lista de hilos y ejecutarlos, estos hilos se encargaran de recopilar cada uno de los tipos de archivos que se encuentren contenidos en la tupla.

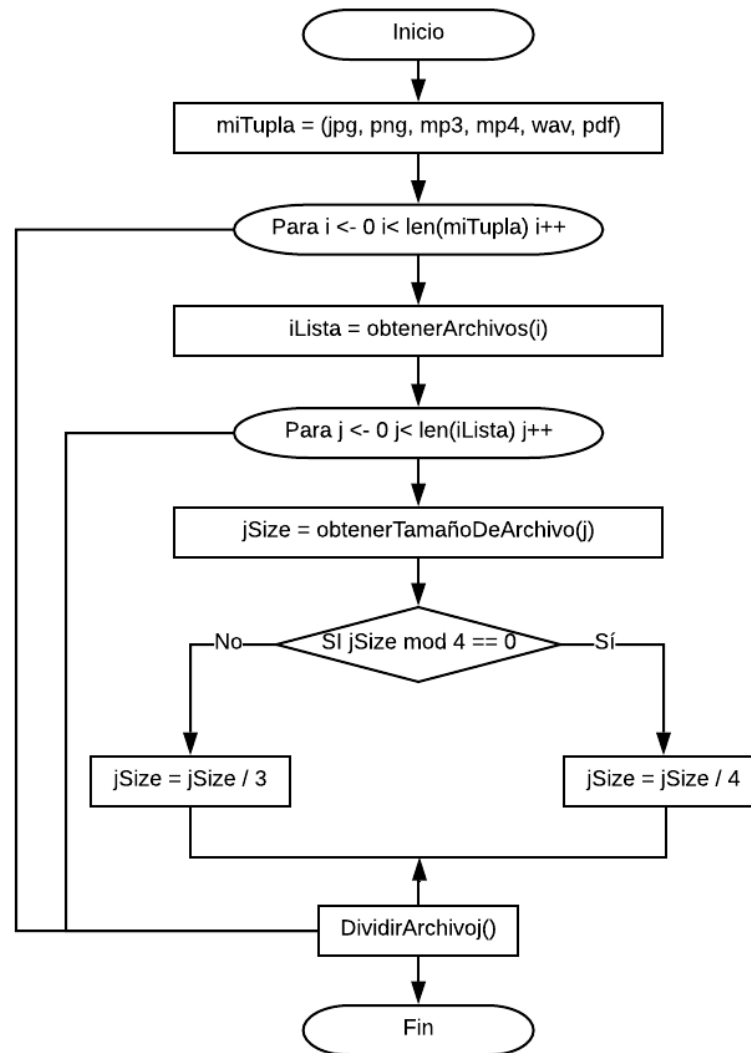


Figura 13: Función que permite particionar cada archivo

En la figura 11 se muestra la función que particiona cada uno de los archivos recolectados por la función anterior, para esto nos auxiliamos del comando `split` que nos proporciona linux.

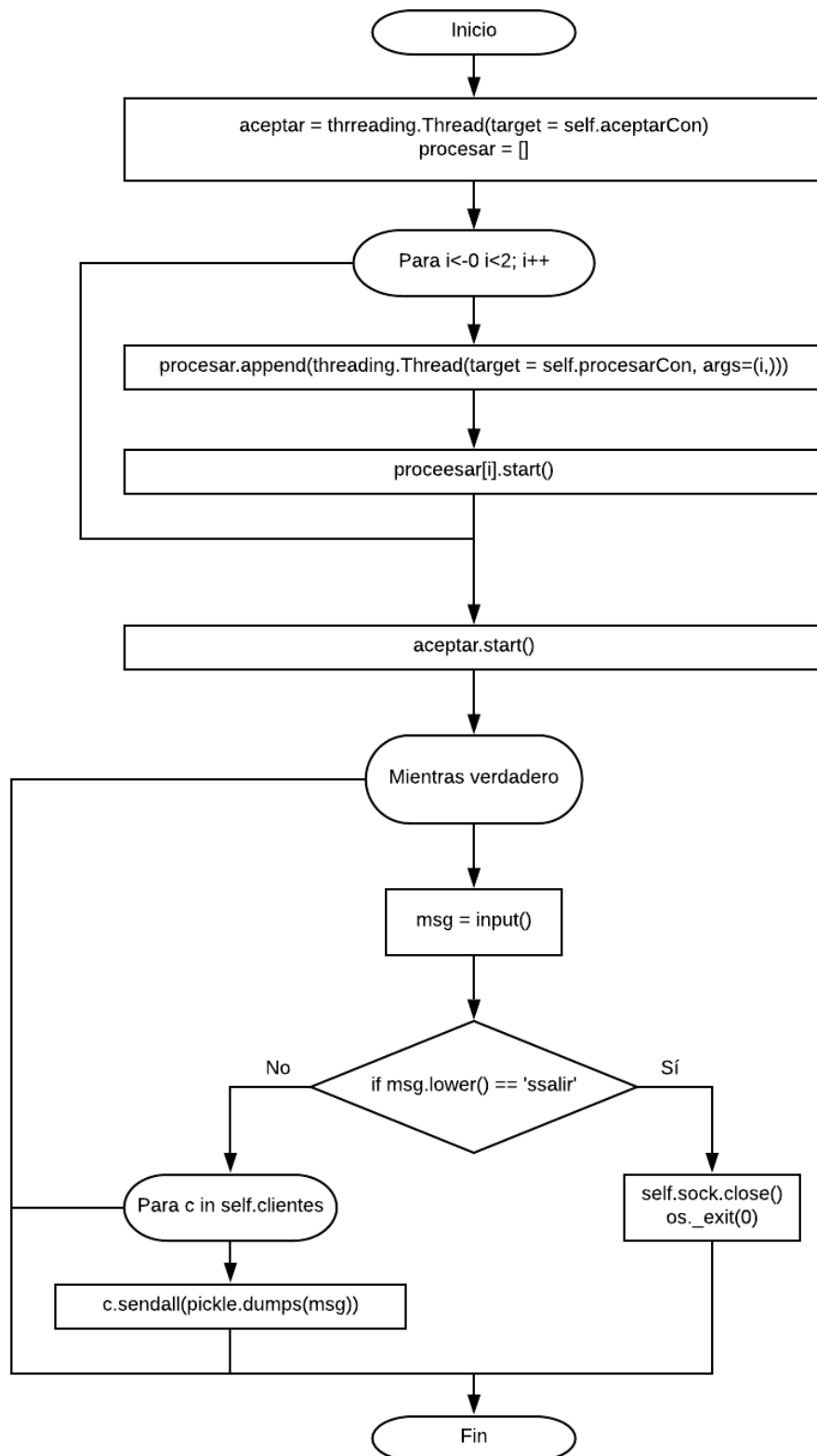


Figura 14: Función que permite aceptar conexiones

En la figura 12 es posible observar las función que se encarga de aceptar y procesar las conexiones, para esto se crea un ciclo infinito que mantendrá vivo nuestro hilo principal.

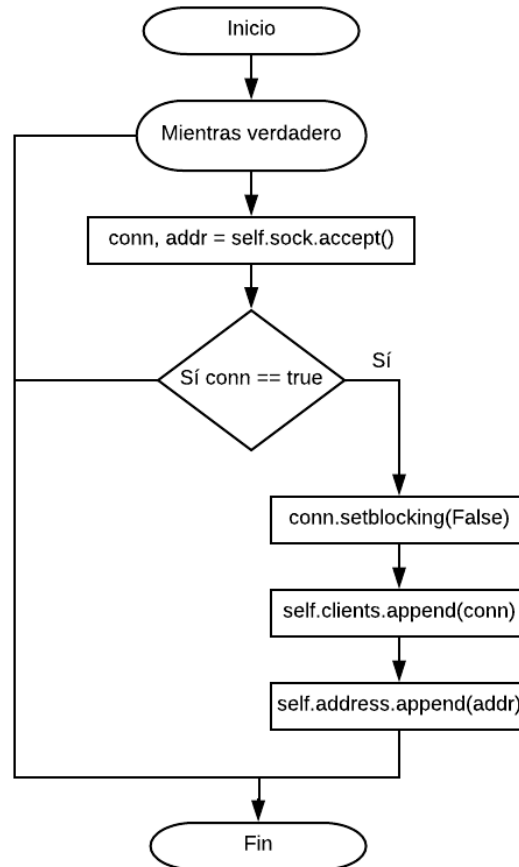


Figura 15: Función que permite aceptar conexiones

En la figura 13 se observa la función con la misión de aceptar las conexiones y almacenar los clientes en una lista.

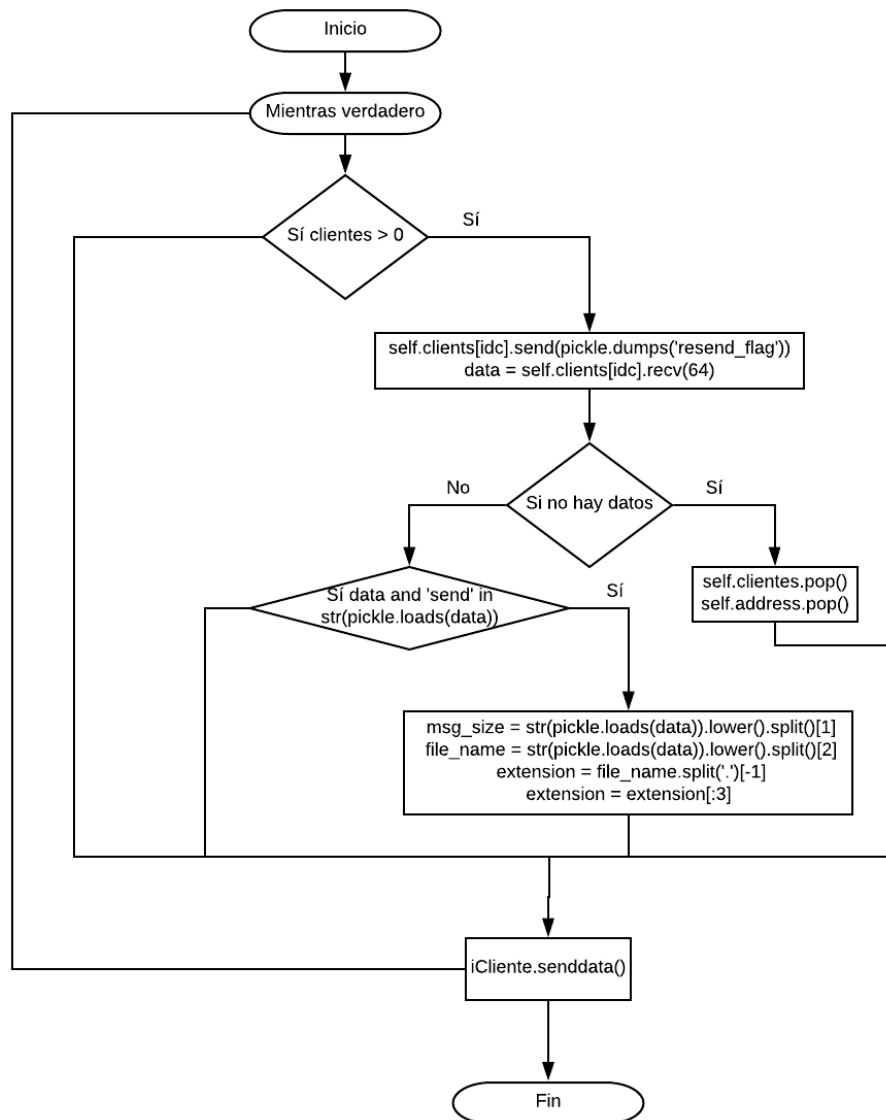


Figura 16: Función que envía las banderas al cliente

La figura anterior se muestra el funcionamiento de la función del servidor encargada de enviar las banderas al cliente que indican que se recibió o no recibió algún archivo

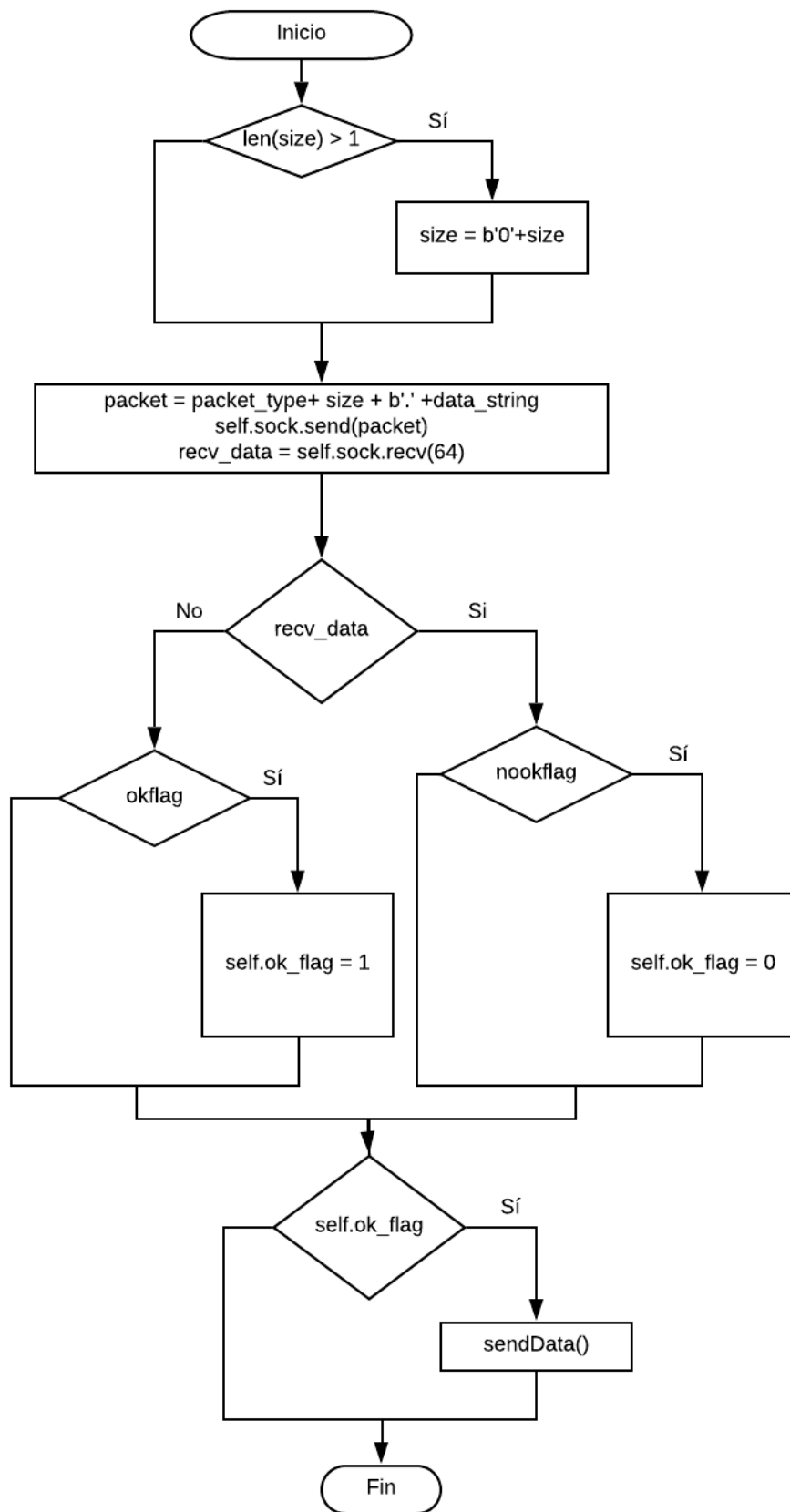
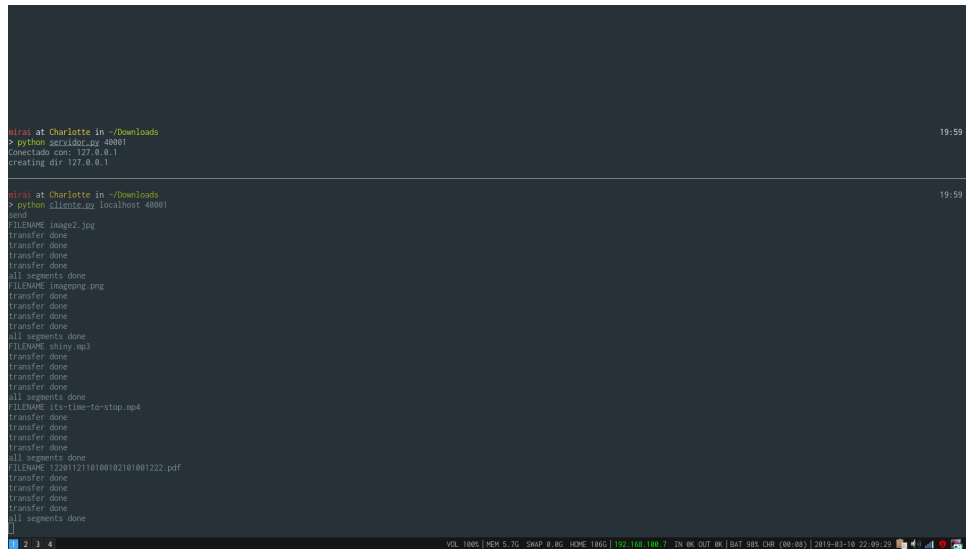


Figura 17: Función que indica cuando enviar los archivos

En la figura 15 es posible apreciar la función que tiene como objetivo principal de enviar cada

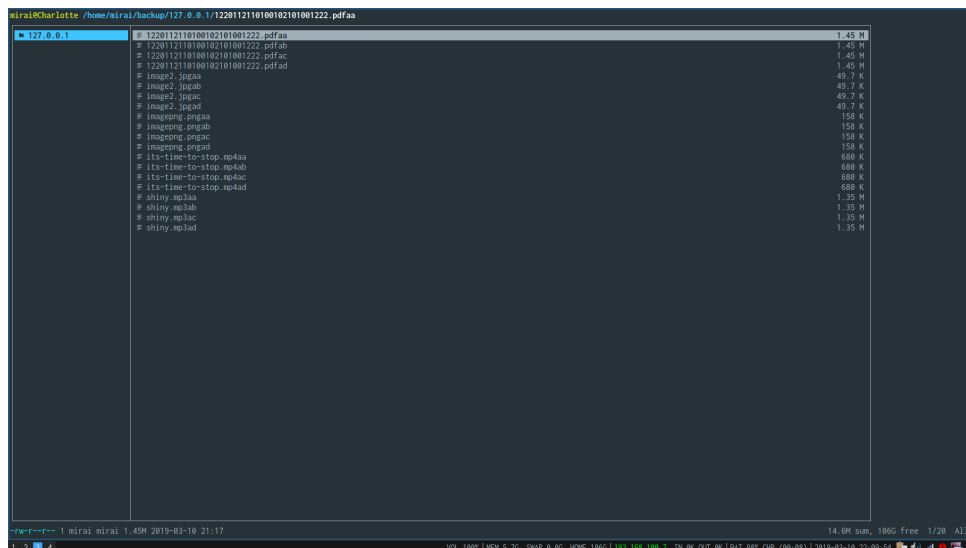
4. Salida del programa.



```
mirai at Charlotte in ~/Downloads
> python servidor.py 40001
Conectado con: 127.0.0.1
creating dir 127.0.0.1

mirai at Charlotte in ~/Downloads
> python cliente.py localhost 40001
send
FILENAME image2.jpg
transfer done
transfer done
transfer done
transfer done
all segments done
FILENAME imagepng.png
transfer done
transfer done
transfer done
transfer done
all segments done
FILENAME shiny.mp3
transfer done
transfer done
transfer done
transfer done
all segments done
FILENAME its-time-to-stop.mp4
transfer done
transfer done
transfer done
transfer done
all segments done
FILENAME 1220112110100102101001222.pdfaa
transfer done
transfer done
transfer done
transfer done
all segments done
[]
```

Figura 18: Salida de la terminal de Servidor y Cliente.



```
mirai@Charlotte: /home/mirai/backup/127.0.0.1/1220112110100102101001222.pdfaa
127.0.0.1
# 1220112110100102101001222.pdfaa 1.45 M
# 1220112110100102101001222.pdfab 1.45 M
# 1220112110100102101001222.pdfac 1.45 M
# 1220112110100102101001222.pdfad 1.45 M
# image2.jpgaa 49.7 K
# image2.jpgab 49.7 K
# image2.jpgac 49.7 K
# image2.jpgad 49.7 K
# imagepng.pngaa 158 K
# imagepng.pngab 158 K
# imagepng.pngac 158 K
# imagepng.pngad 158 K
# its-time-to-stop.mp4aa 680 K
# its-time-to-stop.mp4ab 680 K
# its-time-to-stop.mp4ac 680 K
# its-time-to-stop.mp4ad 680 K
# shiny.mp3aa 1.35 M
# shiny.mp3ab 1.35 M
# shiny.mp3ac 1.35 M
# shiny.mp3ad 1.35 M
```

Figura 19: Backup de archivos e el servidor.

5. Conclusiones.

Hernández Castellanos César Uriel

- UDP es útil para aplicaciones que necesitan transmisión rápida y efectiva
- UDP no tiene un orden inherente y los paquetes de data son independientes uno del otro. Si requieren un orden, esto se maneja a nivel de aplicación.
- TCP reordena paquetes de data en el orden especificado
- TCP es útil para aplicaciones que requieren confiabilidad alta y donde el tiempo de transmisión es menos crítico.
- TCP tiene verificación de errores
- UDP tiene verificación de errores, pero no tiene opciones para recuperar/corregir los mismos
- La capacidad de transferencia sin conexiones de UDP le hace útil para servidores que reciben una gran cantidad de peticiones pequeñas de un alto número de clientes.
- En la práctica se presentaron diferentes dificultades, como el que nuestra aplicación no funcionaba para n clientes si no únicamente para uno solo, además de que en algunas ocasiones no se recibían todos los segmentos del archivo, pero afortunadamente logramos sobrellevar estos problemas y finalizamos con un programa funcional.

Pimentel González Carlos

Con esta práctica concluimos que no es para nada fácil utilizar sockets y menos en UDP. Se tiene que combinar junto con hilos y quizá si el problema es demasiado complejo, también con semáforos. El envío de archivos a través de UDP no es lo ideal para nada ya que llegan con perdidas o en desorden, aparte de que no hay un canal específico de conexión para cada cliente, esto fue una de las cosas que complicó más la práctica. Se tiene que tener una lógica precisa ya que simplemente cambiar el orden de ciertas funciones puede arreglar o empeorar el programa. También el manejo de la información enviada y recibida, debe hacerse muy cuidadosamente.

Martínez Islas Mauricio Joel

UDP se puede considerar como el complemento necesario en un universo de orden. El protocolo no orientado a conexión nos permite ver el balance necesario

El modelo cliente servidor es el molde de galletas que ha logrado cortar a una industria entera. Ver el fundamento que ha formado a miles de aplicaciones de software es invaluable.

La creación ha sido una de las preguntas principales del ser humano. Siempre existirá esa duda. El pensar nos ha hecho mal muchas veces, pero la realidad es que hay preguntas que son demasiado grandes para nosotros. solamente como seres espirituales interconectados es que realmente podremos vivir libres de los contaminantes mentales de los cuales hemos sido testigos desde jóvenes.

Pero nosotros, al jugar a dios, nos comportamos como los creadores. En busca de explicar y replicar la naturaleza de la mejor manera posible, hemos avanzado tecnológicamente. Es ahí en donde nosotros algún día podremos responder a nuestras preguntas iniciales.