MANUAL DE USO Y ESPECIFICACIONES

¿QUÉ DEBE HACERSE PARA EJECUTAR CORRECTAMENTE EL CÓDIGO?

1. Descargar el repositorio de la siguiente ruta:
    a. https://github.com/bchaparro11/os_second_practice.git
2. Descargar el archivo CSV de la siguiente ruta:
    a. https://drive.google.com/file/d/1c5HLLx4Vyvb367isgOLEOmYICU2dHFtz/view
3. Colocar dentro de la carpeta toCreation el archivo CSV que se descargó en el punto anterior
4. Abra una terminal en la ruta de la carpeta toCreation
5. Ejecute los siguientes comandos
    1. *make script1*
    2. *make script2*
    3. *make script3*
6. Abra una terminal en la ruta de la carpeta toQuery para el servidor
7. Ejecute el siguiente comando
    1. *make sserver*
8. Abra las terminales que requiera para los clientes que se van a conectar al servidor en la ruta de la carpeta toQuery
9. Ejecute el siguiente comando
    1. *make sclient*
10. Ejecute los comandos según crea necesario del menú, cuando estén ingresados los datos con la opción 4 del menú puede consultar el tiempo medio del viaje.
11. Repita el punto 10 para cada uno de los clientes.

¿POR QUÉ DEBE HACERSE LO ANTERIOR?

Esta parte del documento no se enfocará en cómo se elaboró el código sino en qué pasos se deben seguir para que el código entre en funcionamiento. A grandes rasgos la carpeta donde se encuentra el código está dividida en dos partes una carpeta que se llama *toCreation* y una carpeta que se llama *toQuery*:

toCreation:

Esta carpeta tiene tres scripts ".c", cada script tiene un propósito específico, los scripts con sus correspondientes propósitos son:

- El código llamado **1.c,** lo que hace primero es obtener los primeros 4 datos de cada registro, los únicos que se van a utilizar en toda la práctica, del archivo .csv de UBER, para por último guardarlos de manera binaria en un archivo nuevo llamado *data.bin* que queda guardo en la carpeta *toQuery*.

- El código llamado **2.c** crea un archivo llamado *hashtable.bin,* que queda guardado en la carpeta *toQuery,* que será donde se almacenará toda la pseudo hash table. Esta pseudo hash table tiene 1300 posiciones que es un valor aproximado a la cantidad de órigenes o destinos que están en el archivo .csv.

- El código llamado **3.c** complementa los archivos creados anteriores, data.bin y hashtable.bin, de tal manera que quede implementada la pseudo hash table y las pseudo listas enlazadas comprendidas por todos los registros del archivo data.bin.

Entonces, para que el código funcione lo que hay que hacer es primero ejecutar el código **1.c,** luego el **2.c** y por último el **3.c.** mediante el archivo **Makefile** que se encuentra también en la carpeta *toCreation.*

toQuery:

Esta carpeta está compuesta por los dos archivos resultantes, luego de ejecutar los tres scripts situados en *toCreation,* dos scripts ".c" y un archivo Makefile. También existirá un nuevo archivo llamado *log.txt* donde quedarán registrados los datos de cada cliente conectado al servidor. Los dos códigos ".c" con sus correspondientes propósitos son:

- El código **p2-server.c** es el encargado de implementar el servidor y de gestionar los 32 clientes posibles, mediante hilos que también crea el servidor. Además, también crea un archivo llamado *log.txt* que guardará el momento en el que se guarda cada cliente, su dirección IP y los valores encontrados en el archivo data.bin.

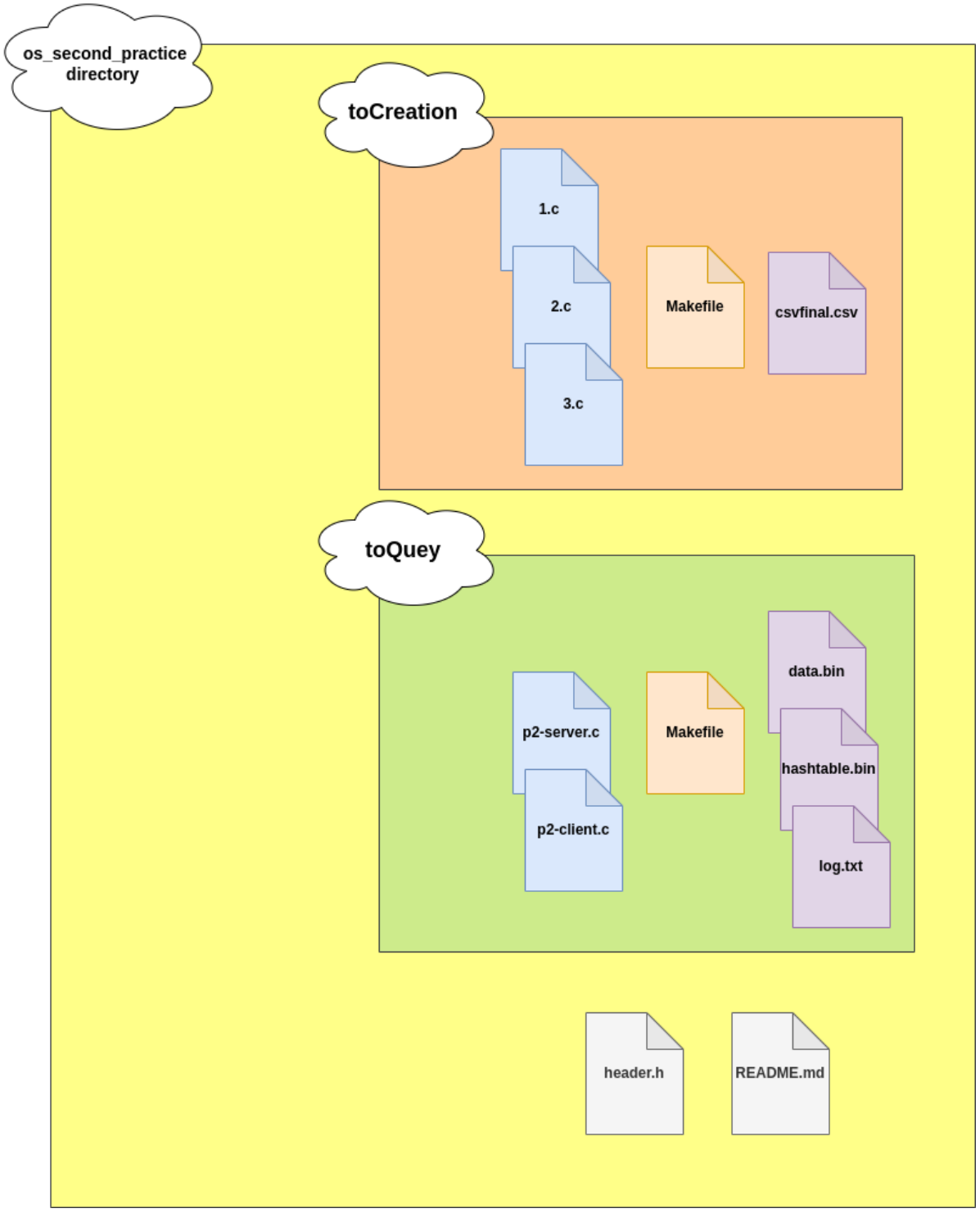
- El código **p2-client.c** es el encargado de implementar los clientes y el menú para interactuar con cada usuario. Luego que cada usuario interactúa con el menú, cada cliente, correspondiente a cada usuario, le envía la información obtenida al servidor, el servidor procesa la información (interactúa con los archivos data.bin y hashtable.bin para encontrar alguna coincidencia) y le envía la información procesada al cliente. Por último, esta última información obtenida se imprime en la consola de cada usuario.

Entonces, para poner en funcionamiento esta comunicación *cliente-servidor* primero se debe ejecutar el script p2-server.c y luego el p2-client.c, tantas veces hasta 32, mediante la ayuda del archivo Makefile. Seguir al pie de la letra estas indicaciones será el camino para ejecutar con éxito el código aquí almacenado y poder obtener información del archivo de casi seis millones de usuarios mediante una comunicación cliente-servidor.

INFORME DE ELABORACIÓN

DIAGRAMA DE CONTENENCIA:

A partir del *Manual de uso* y de la siguiente gráfica se explicará el informe de elaboración



1. **os_second_practice directory** está compuesto por:
    a. **toCreation:**
        i. **Scripts en C:**
            1. **1.c,** contiene:
                a. **main():** Lee desde csvfinal.c y escribe en data.bin, cada registro, a través de una estructura llamada *row* que se explica en *header.h.*
            2. **2.c,** contiene:

a. **main():** Crea hashtable.bin a partir de una estructura llamada *hashrow* que se explica en *header.h*.

3. **3.c,** contiene:
    a. **main():** Adiciona información en data.bin y hashtable.bin para completar una pseudo hashtable con sus correspondientes pseudo listas enlazadas con la ayuda de las estructuras *row* y *hasrow* explicadas en *header.h*
    b. **reversetravel():** Viaja desde el último registro de data.bin hasta el primero a través del manejo del puntero FILE correspondiente a data.bin.

ii. Archivo **Makefile**, con tres etiquetas para ejecutar los tres scripts en C que están en esta carpeta (toCreation)

iii. Archivo **csvfinal.csv**, que contiene los casi seis millones de registros crudos de información de viajes de UBER.

b. toQuery:

i. Scripts en C:

1. **p2-server.c**, contiene:
    a. **main():** Crea el servidor, lo configura y ejecuta la función threading para poder atender a los 32 clientes posibles.
    b. **threading():** Lanza 32 hilos que servirán para atender a los 32 clientes. Cada hilo ejecuta la función accepting.
    c. **accepting():** Espera cada cliente a partir de la función accept, recibe información del origen, destino y hora del viaje para agregarlos como argumento en la función searching(). Además, guarda en el archivo *log.txt* el momento y la dirección IP del cliente, esta escritura luego se complementa en la función *loggin().*
    d. **searching():** Interactúa con el archivo data.bin y hashtable.bin para obtener el valor, si existe, del tiempo medio en recorrer desde el origen al destino. Después envía el valor medio al cliente a partir de la estructura *row* que se explica en *header.h.* Por último, ejecuta la función *loggin()* que guarda la información del origen, destino y valor medio y lo guarda en el archivo *log.txt.*
    e. **logging():** Guarda la información del origen, destino y valor medio y lo guarda en el archivo *log.txt*
2. **p2-client.c,** cliente:
    a. **main():** Gestiona el menú a partir de la estructura *sharerow* que se explica en *header.h.* Luego envía esta estructura como argumento en la función *creating_client().*
    b. **creating_client():** Crea el cliente, lo configura y envía, a partir de la función connect(), la estructura *sharerow* recibida por la función *main().*

ii. Archivo **Makefile**, contiene dos etiquetas que sirven para ejecutar p2-server.c y p2-client.c

iii. **Archivos almacenadores:**

1. **data.bin,** almacena origen, destino, hora y valor viaje medio en formato binario de todos los registros que están almacenados en csvfinal.csv. Además, también guarda tres valores adicionales como mpos (MyPosition), para guardar la posición del registro, npos (NextPosition), para guardar la posición del siguiente registro con el mismo valor de origen e isfound, que es un atributo utilizado por la función searching() y creating_client() para saber si se encontró una coincidencia en data.bin.
2. **hashtable.bin,** almacena la primera posición, en data.bin, de cada valor de origen.
3. **log.txt,** almacena la información identificadora de todos los clientes conectados al servidor.

c. header.h, contiene:

i. **struct row,** es una estructura para guardar:

1. **mpos:** El nombre viene de MyPosition. Sirve para guardar la posición del registro en el archivo data.bin.
2. **npos:** El nombre viene de NextPosition. Sirve para guarda la posición del siguiente registro que tiene el mismo valor de origen al actual.
3. **sourceid:** Valor origen en data.bin
4. **dstid:** Valor destino en data.bin
5. **hod:** Valor hora en data.bin
6. **mean:** Valor viaje medio entre el origen y el destino, almacenado en data.bin
7. **isfound:** Atributo usado por *searching()* y *creating_client()* para verificar si se encontró una coincidencia en data.bin

ii. **struct hashrow,** es una estructura para guardar:

1. **img:** La posición en data.bin, del primer registro que tiene un valor de origen específico

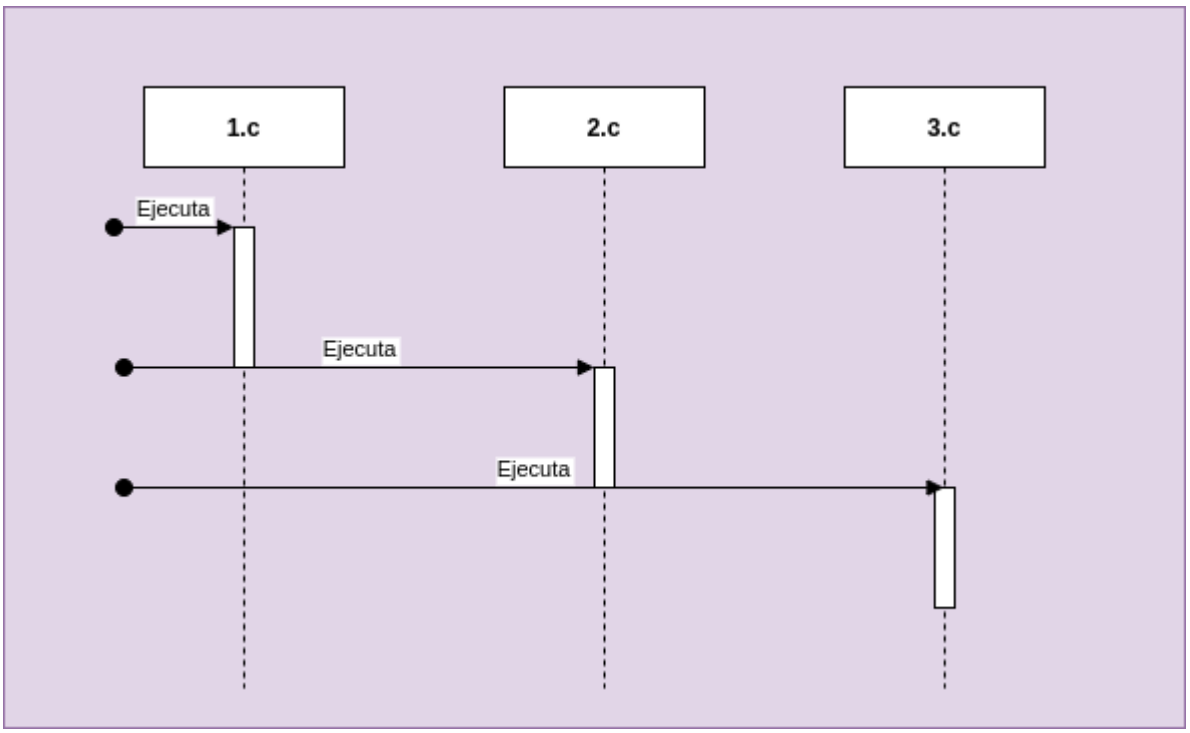
iii. **struct sharerow,** es una estructura para guardar:

1. **sourceid:** Ver struct row para entender
2. **dstid:** Ver struct row para entender
3. **hod:** Ver struct row para entender

d. **README.md**

DIAGRAMAS DE CONEXIONES

PARA CÓDIGOS EN *toCreation:*



PARA CÓDIGOS EN *toQuery:*

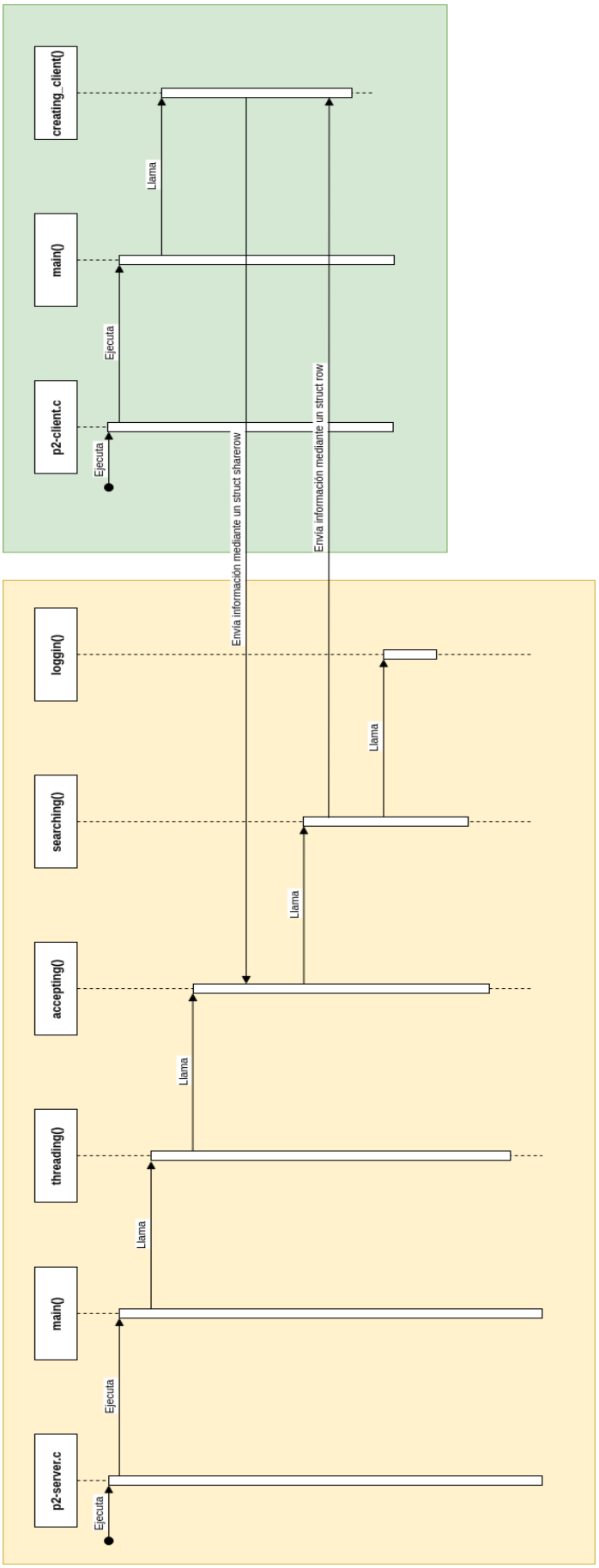
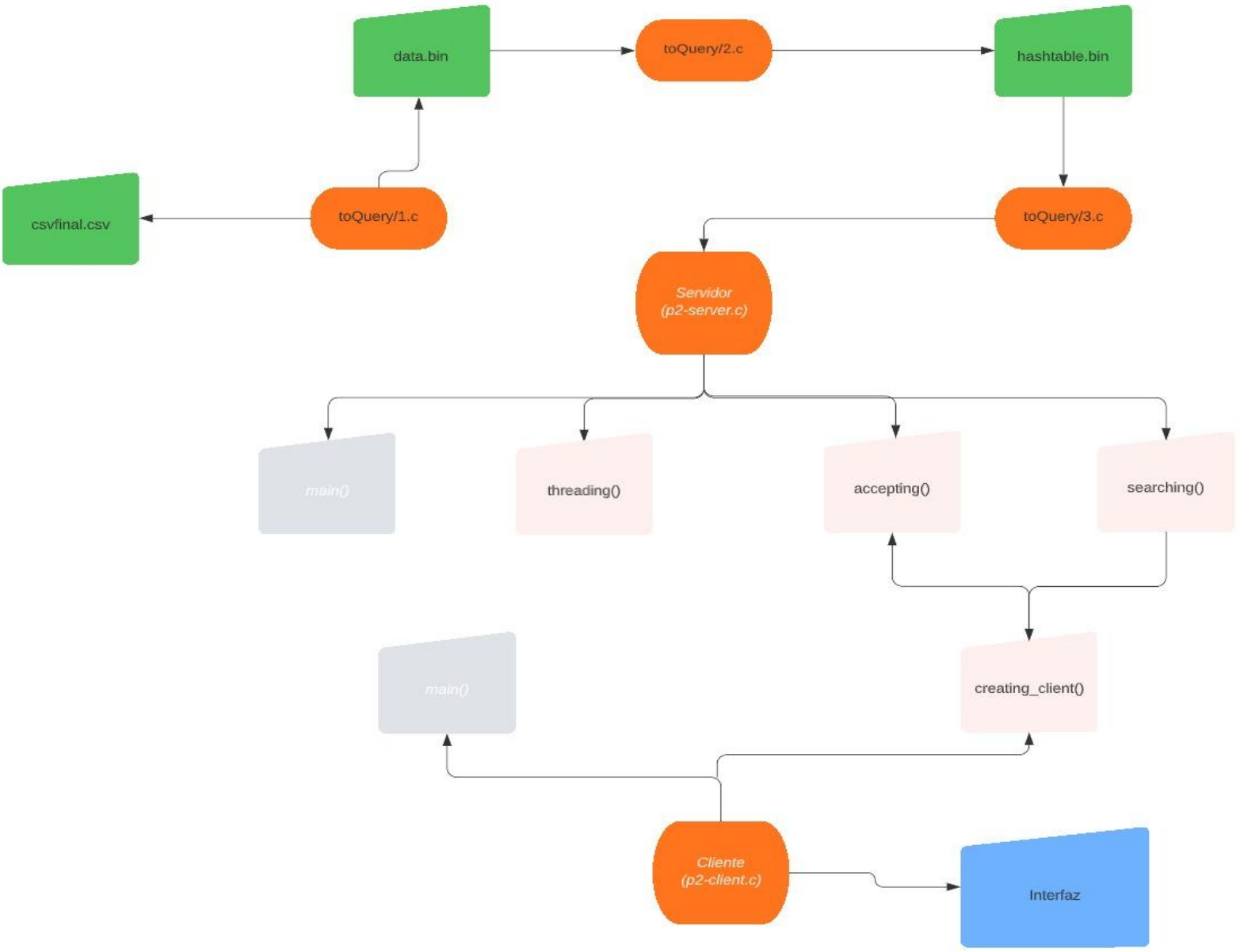


Diagrama de Bloques:



EXPERIENCIAS DE ELABORACIÓN:

Dentro de los principales desafíos para el desarrollo de esta práctica, se identificaron los siguientes:

Joe Zafir Mendez Leon Por mi parte un desafío que se me presentó inicialmente fue el manejo de múltiples conexiones con el servidor, pero gracias a el tema de hilos visto en clase se pudo superar satisfactoriamente. A nivel más personal, la falta de tiempo y mas en

tiempos de entregas finales ya que es difícil poder coincidir para poder trabajar todos juntos, lo bueno es que el grupo de trabajo fue muy responsable y se cumplió con todo lo esperado.

Federico Borja Sarmiento: Coordinación y definición de tiempos de trabajo, homologar términos y prácticas de programación.

Brian Chaparro Cetina: El desafío más grande fue la coordinación del tiempo para poder cumplir con todos los aspectos de la *Práctica 2* por parte de todo el equipo y que al mismo tiempo no perjudicara con las demás obligaciones de cada integrante.

Por otra parte, se identificaron los siguientes beneficios y ventajas en el desarrollo de esta:

Joe Zafir Méndez León El poder implementar diferentes cosas que se vieron en clase y poder ver de primera mano el funcionamiento del mismo para poder consolidar todos los conocimientos que se adquirieron durante el curso, además el mejorar el trabajo en equipo y poder confiar en que se están apoyando mutuamente y gracias a esto reforzar conocimientos. Finalmente creo que es muy importante el mencionar que entre más se avanza en la carrera todo lo que se ha visto es acumulable y va siendo de mucha ayuda y esencial para estar realmente preparado

Federico Borja: El trabajo en equipo permite expandir los conocimientos, establecer espacios de diálogo donde se pueden confrontar conocimientos, y establecer planes de trabajo acordes a las habilidades de los integrantes del equipo. La solución de dudas entre nosotros, aceleró el rendimiento de este.

De igual manera, el tema de la práctica permite tener un acercamiento a experiencias cercanas a las que podemos encontrarnos en entornos laborales. La implementación de un servidor, expande el panorama de posibilidades sobre las cuáles planear nuestro crecimiento profesional.

Brian Chaparro Cetina: Es enriquecedor poder notar cómo un programa complejo como el de la *Práctica 1* puede implementar la comunicación de procesos de varias maneras y ofrecer ventajas y consecuencias diferentes en cada una de las maneras de implementación. En el caso particular de la *Práctica 2* se implementó la comunicación de procesos usando la infraestructura de red interna de un computador, comunicación llamada *Paso de mensajes,* junto con hilos. La práctica 2 permitió tener un acercamiento de cómo funciona un servidor real atendiendo concurrentemente a muchos usuarios. En conclusión, me permitió juntar los conocimientos sobre *Redes de computadores* y *atención empresarial para múltiples usuarios* junto a los conceptos de *Sistemas Operativos,* mostrando que en el área del software hay muchas cosas conectadas y no sólo ejecutar un *HolaMundo.*