# Sistemas Operativos
# Práctica 2

Nicolás López Nieto
nilopezn@unal.edu.co

Fabián Alejandro Torres Ramos
fatorresra@unal.edu.co

28 de junio de 2022

## 1. Instrucciones de uso

En primer lugar, los datos de búsqueda deberán ir en un `archivo.csv` en la carpeta principal; este archivo se indexará ejecutando en una terminal la instrucción `make ejecutarIndexador`. Adicionalmente se pueden obtener los archivos de prueba ya indexados en este documento en Drive, el cual es descomprimido directamente en la carpeta principal.

Para compilar y ejecutar se tienen los siguientes comandos:

- `make` ejecutará el programa servidor en la terminal y creará una segunda terminal para ejecutar un programa cliente.
- `make ejecutarServidor` y `make ejecutarCliente` solo ejecutarán uno de los dos programas en la terminal.
- `clean` y `clean Indexador` limpiarán los archivos binarios y el archivo *log*.

El archivo fuente `p2-client.c` inicialmente tiene como dirección IP del servidor al localhost (127.0.0.1). Este valor se puede cambiar para comunicarse con un servidor diferente, siempre que se encuentre en la misma red de área local.

### 1.1. Ejecución

El programa cliente mostrará el siguiente menú después de confirmar la conexión con el servidor:

```
Bienvenido
1. Ingresar origen
2. Ingresar destino
3. Ingresar hora
4. Buscar tiempo de viaje medio
5. Salir
```

El origen y el destino están dados por IDs que tienen rango entre 1 y 1160 según los datos proporcionados por Uber. Una vez ingresados los tres datos se procede a realizar la búsqueda al servidor. Después de cada opción se requiere una confirmación del usuario, ingresando cualquier tecla para continuar.

En caso de no encontrar un registro con los parámetros indicados se mostrará el mensaje "`NA`". De lo contrario se describirá el registro encontrado con su tiempo de viajo medio.

# 2. Informe de elaboración

Se realizaron tres programas: uno para el cliente, uno para el servidor y uno para indexar los datos del archivo `.csv`, el cual solo se necesita ejecutar una vez. Se utilizó el paso de mensajes para la comunicación entre clientes y servidor.

## 2.1. Funciones

### 2.1.1. Paso de mensajes

Para tanto el cliente como el servidor se crean los sockets `clientefd` y `serverfd`, y se configuran con las estructuras `client` y `server` respectivamente (el servidor también requiere las funciones `bind()`, `listen()` y `setsocketopt()`). Se conectan con las funciones `connect()` y `accept()` y se envían mensajes con `send()` y `recv()`.

### 2.1.2. Cliente

Una vez conectado a un servidor se tiene un bucle `do-while` y la instrucción `switch` para manejar el menú. Los datos se guardarán en una estructura tipo `Datos` que contiene todos los campos proporcionados en el archivo `.csv`, y mediante la cual se mandarán los mensajes de búsqueda y respuesta.

Se tienen las funciones `idlugar()` y `formatoHora()` para validar los datos ingresados.

El cliente mandará gradualmente la estructura mediante `send()` hasta que se envíen todos los bytes. Como la función `recv()` es bloqueante, el programa esperará a que el servidor envíe el mensaje de respuesta con el tiempo de viaje medio; si este valor es -1 significará que no se encontraron registros con los parámetros indicados.

### 2.1.3. Indexador

Inicialmente se tiene una "función hash"que por el momento es casi innecesaria, puesto que los IDs proporcionados están bien ordenados entre 1 y 1160.

El programa trabaja con tres archivos: uno de entrada (`archivo.csv`) y dos de salida (`salidaIndex` con todos los datos indexados en binario y `salidaHash` con la *posición del primer registro* en `salidaIndex` con cierto ID de origen). Se hacen dos recorridos a los archivos:

1. Se recorre el `.csv` línea por línea con `fgets()` y se separan los campos con `strtok()` para guardar los datos en estructuras en binario. Se agrega un campo `sig` para denotar la *posición del siguiente registro* con el mismo ID de origen y se guarda la primera aparición de cada ID de origen en la `tablaHash`.

2. Se recorre el archivo con los registros en binario de abajo hacia arriba, usando un arreglo `enlistador` para llenar los campos `sig` con las posiciones necesarias.

Guardar los datos en binario normaliza el tamaño de todos los registros, lo que facilita recorrer el archivo al revés. La tabla hash se guarda en el archivo correspondiente para iniciar las búsquedas. Mediante las funciones `fseek()`, `fwrite()` y `fread()` se evita cargar el archivo entero a memoria.

### 2.1.4. Servidor

Por la complejidad se separaron las funciones `configuracionServidor()` y `aceptarCliente()`, y para atender múltiples clientes se utiliza la función `fork()`, creando un nuevo proceso hijo que recibirá los datos, realizará la búsqueda correspondiente y enviará la respuesta.

Para la búsqueda se toma el `idOrigen` a través de la función hash y se busca en `salidaHash` la posición del primer registro con dicho ID de origen (si no existe la tabla tendrá -1). Esta posición se busca en `salidaIndex` y se comparan los tres parámetros; en caso de que no coincidan se salta a la posición que indique `sig`. El proceso se repite hasta que se encuentre el

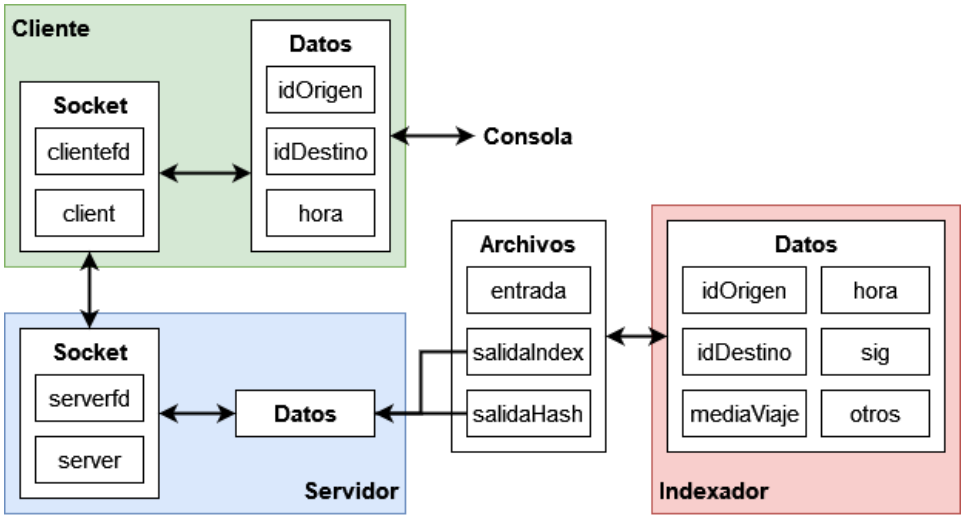
registro correcto o no hayan más registros siguientes, en cuyo caso el tiempo de viaje medio será -1. En todo caso se devuelve una estructura `Datos` con el último registro encontrado.

Por otro lado se escribe a un archivo `log.txt` el siguiente formato:

```
[YYYYMMDDHHMMSS] [IP] [mediaViaje - idOrigen - idDestino]
```

donde el IP corresponde al cliente que pidió la búsqueda. Se utiliza un mutex para evitar conflictos en caso de recibir múltiples clientes a la vez.

## 2.2. Diagrama de bloques




## 2.3. Diagrama de comunicaciones