# Trabajo Práctico final 2024/1C

Objetivo: diseñar un protocolo que sea una extensión al protocolo ECHO visto en clase, e implementar un servidor concurrente.
En el servidor deberá configurarse una carpeta donde se alojen archivos de texto, y el protocolo deberá tener los comandos necesarios para recuperar el texto de un archivo
Los protocolos a desarrollar deberán ser **de texto**, no binarios.

El protocolo debe cumplir como mínimo con los siguientes requisitos:

1) El servidor debe aceptar una conexión usando TCP en la cual
    a) Recibe líneas de texto US-ASCII finalizadas con \r\n
    b) Cada línea recibida debería tener una longitud máxima de 1000 caracteres (sin contar los \r\n al final de la línea)
    c) Si una línea supera la longitud, los caracteres excedentes se ignoran
    d) Si la línea contiene algún carácter no US-ASCII se ignora desde ese carácter hasta el final de la línea
    e) Cada línea será uno de los siguientes comandos
        i) **`LIST FILES\r\n:`** el servidor responderá con una lista de nombres de archivos habilitados para su recuperación. Cada grupo deberá definir el formato de la respuesta.
        ii) **`ECHO `*`texto`*`\r\n`**: el servidor le enviará como respuesta al cliente el texto recibido, finalizado con \r\n
        iii) **`GET `*`fileName`*`\r\n`**: si el archivo existe, el servidor responde primero con una línea indicando OK, un espacio y la cantidad de bytes del archivo (por ejemplo "OK 132456\r\n") y luego el contenido del archivo. Si el archivo no existe debe indicar un mensaje de error.

Por defecto el protocolo es case-insensitive para los comandos, los siguientes mensajes son equivalentes

- GET errores.txt
- get errores.txt
- Get errores.txt

El protocolo no contempla que se puedan subir archivos, los mismos deberán alojarse en la carpeta correspondiente utilizando algún otro servicio

2) A su vez el servidor puede recibir los siguientes datagramas (vía UDP)
    a) **`SET case ON`**: a partir de ese momento, el servidor será case-sensitive para los comandos.
    b) **`SET case OFF`**: a partir de ese momento, el servidor no será case-sensitive
    c) **`STATS`**: le devuelve al cliente un datagrama con las siguientes estadísticas en modo texto, cada una separada por \r\n,
        i) cantidad de conexiones realizadas desde que inició la ejecución

ii) cantidad de líneas incorrectas recibidas
iii) cantidad de líneas correctas recibidas
iv) cantidad de datagramas incorrectos recibidos (comandos inválidos o inexistentes)
v) cantidad de archivos descargados
vi) cantidad de archivos subidos (sólo para los que entreguen en segunda fecha)

3) En ambos casos el puerto por defecto a usar por el servidor será el 9999, pero podrá recibir por línea de comandos en qué puerto escuchar

Los grupos que decidan presentarse a la segunda fecha de final, deberán además cumplir con los siguientes requisitos:

4) El servidor deberá cerrar la conexión de los clientes que hayan estado ociosos por más de 5 segundos.
5) Agregar un comando para subir un archivo, que luego pueda recuperarse. Queda a criterio del grupo las reglas a seguir para poder subir archivos, como así también manejar posibles errores
6) Soportar el comando **`GET base64 fileName\r\n`**: si el archivo existe, el servidor responde primero con una línea indicando OK, un espacio y la cantidad de bytes **del archivo** (por ejemplo "OK 132456\r\n") y luego el contenido del archivo codificado en base 64. Si el archivo no existe debe indicar un mensaje de error.

## A tener en cuenta:

- Se desea que el servidor utilice la menor cantidad de memoria posible.
- Antes de desarrollar el TPE, aconsejamos ver los ejercicios de la guía "Programación con sockets", especialmente los que plantean algunas deficiencias o limitaciones de los códigos vistos en clase.

Texto sobre repositorio privado, que debe reflejar el trabajo de cada alumno y permita ver la evolución del mismo, no se aceptará un repositorio que tenga sólo un par de commits, etc.

## Material a entregar

Un archivo compactado conteniendo como mínimo los siguientes archivos:

- Archivos fuentes y de encabezado de la aplicación servidor.
- Archivo de texto README con la explicación de cómo generar los ejecutables y de cómo ejecutarlos.
- Una carpeta con archivos de ejemplo a ser recuperados (para probar tanto LIST FILES como GET FILE)
- Uno o más programas cliente de prueba. Dichos programas deben contemplar tanto los casos exitosos como casos de error (enviar comandos incorrectos, solicitar archivos no existentes, etc.)
- `makefile`. ( se deben usar gcc y los flags `-Wall` y `-fsanitize=address` ).

- Carpeta .git

**El servidor se probará en Pampero, los clientes tanto en Pampero como en computadoras en otras redes.**

La fecha límite para la entrega es el _____ a las 23:59 hs