2. Parte 1. Utilizando sockets, colas e hilos.

En este caso tendremos solo dos procesos: el servidor de ficheros y un cliente que emula ser múltiples clientes haciendo consultas a la vez. Ambos usarán programación multihilo.

2.1. Proceso servidor

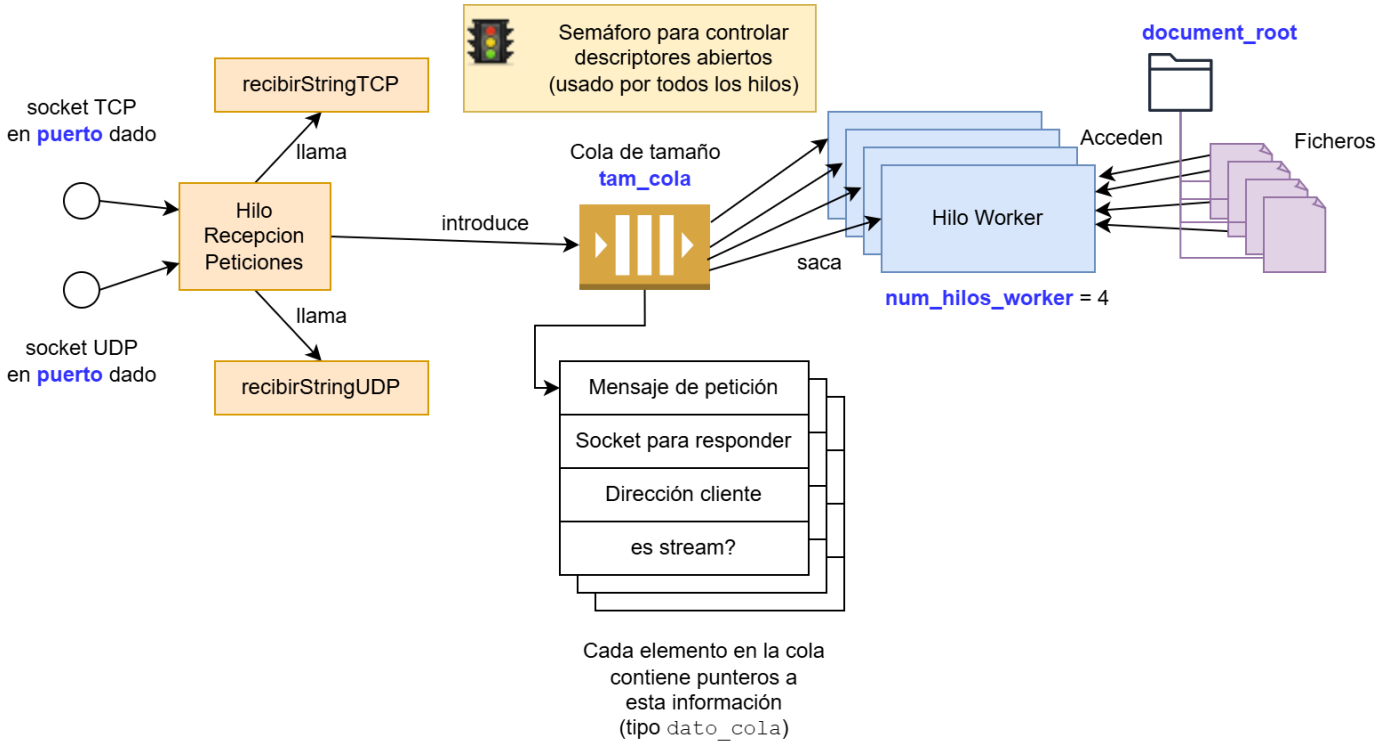
El proceso servidor se denominará `srvftp` (fichero fuente `srvftp.c`). Este proceso se lanzará mediante la siguiente línea de comandos:

Forma de uso de `servidor`

```
$ srvftp puerto tam_cola num_hilos_worker ruta_absoluta_document_root
Parámetros:
   - puerto: puerto en el que escuchará el servidor
   - tam_cola: tamaño de la cola interna con la que sincroniza con los hilos trabajadores
   - num_hilos_worker: número de hilos que procesan las peticiones de la cola interna
   - ruta_absoluta_document_root: ruta absoluta al directorio raíz de los ficheros a servir
```

2.1.1. Explicación de la arquitectura del servidor

La arquitectura interna del proceso se muestra en la figura siguiente:



El servidor es un servidor multihilo que implementa el paradigma jefe-trabajadores con cola de trabajos. El servidor espera peticiones en el mismo puerto para los protocolos TCP y UDP, es decir crea dos sockets, uno TCP y otro UDP vinculados al mismo puerto. La comunicación entre el jefe y los hilos trabajadores se realiza a través de una cola circular cuyo tamaño se recibe como argumento a través de la línea de comandos.

Las posiciones de la cola circular son del tipo `dato_cola`:

Estructura de cada dato en la cola circular

```
typedef struct {
    char msg[TAMMSG]; //peticion enviada por el cliente
    int s; //socket de comunicacion con el cliente
```

```
    struct sockaddr_in d_cliente; //dirección IP del cliente
    unsigned char es_stream; //es o no orientado a conexion (TCP)
} dato_cola;
```

El jefe ejecuta la función `RecepcionPeticiones()`. Esta función utiliza la llamada al sistema `select()` para conocer en cuál de los dos sockets hay alguna actividad. En caso de haber actividad en el socket TCP, se llama a la función `recibirStringTCP()` y, con los valores devueltos por la misma, se reserva espacio para un nuevo dato a insertar en la cola circular para el cual se rellenan los datos correspondientes al mensaje enviado por el cliente (campo `msg`), socket TCP de comunicación con el cliente (campo `s`) y se coloca el valor CIERTO en el campo `es_stream`. En caso de detectar actividad en el socket UDP, se llama a la función `recibirStringUDP()` y, con los valores devueltos por la misma se reserva espacio para un nuevo dato a insertar en la cola circular rellenando los campos correspondientes al mensaje enviado por el cliente (campo `msg`), se coloca una copia del socket UDP en el campo `s`, se pone a FALSO el valor del campo `es_stream` y por último se coloca en el campo `d_cliente` los datos del cliente (IP y puerto) a los que se le deberá remitir la respuesta.

Los hilos trabajadores ejecutan la función `Worker()`. El algoritmo que implementa esta función es un bucle infinito en el cual, lo primero que se hace en cada iteración es esperar a obtener el siguiente elemento de la cola de peticiones. Con el elemento extraído de la cola de peticiones se puede acceder al campo `msg` que contiene la cadena de texto que indica qué está solicitando el cliente (más adelante se dan detalles sobre qué puede haber en esta cadena). Se analiza (*tonekiza*) esta cadena y en función del tipo de mensaje recibido, se envía prepara la respuesta que se enviará al cliente, bien por el socket TCP o bien por el socket UDP en función del valor del campo `es_stream` del elemento extraído de la cola.

El semáforo que se menciona en la parte superior de la figura existe para evitar tener abiertos a la vez demasiados ficheros. Dado el elevado número de ficheros que puede llegar a manejar el servidor, y para no consumir todas las entradas disponibles en la tabla de descriptores de archivo, antes de realizar una operación que implique abrir un fichero o aceptar una petición de conexión en un socket servidor TCP (que también consumiría un descriptor), se realiza una operación `sem_wait()` sobre el semáforo de cuenta `sem_desc`. Cuando se cierra un fichero se realiza una operación `sem_post()` sobre el mismo semáforo. El valor del semáforo por tanto indica cuántos descriptores aún quedan libres en ese momento. Dicho semáforo se inicializa en la función `main()` del programa. El valor al que se inicializa `sem_desc` es `_SC_OPEN_MAX/2`, siendo (`_SC_OPEN_MAX`) una constante predefinida en la cabecera `<limits.h>` que indica cuál es el número máximo de ficheros que puede tener abiertos un proceso en un momento dado.

2.1.2. Protocolo de comunicación

En esta sección se detallan qué comandos puede enviar un cliente (a través de una cadena de texto) y qué debe hacer el servidor en cada caso.

- `"ISFILE nom_fichero"`: Ante este mensaje el hilo trabajador comprobará si en el repositorio de ficheros del servidor (document root) existe un fichero con ese nombre. En caso de existir envía un mensaje `"OK"` y en caso contrario `"NOK"`.
- `"ISREADABLE nom_fichero"`: Ante este mensaje el hilo trabajador comprobará si en el repositorio de ficheros del servidor (document root) el fichero referenciado tiene permiso de lectura. En caso de tener permiso de lectura envía un mensaje `"OK"` y en caso contrario `"NOK"`.
- `"GETNUMLINES nom_fichero"`: Ante este mensaje el hilo trabajador enviará como resultado un mensaje con la representación en formato cadena del número de líneas de texto que tiene el fichero `nom_fichero`. Por ejemplo la respuesta podría ser `"124"` si el fichero tiene 124 líneas.
- `"GETLINE num_line nom_fichero"`: Ante este mensaje el hilo trabajador enviará como resultado un mensaje con la línea que ocupa la posición `num_line` en el fichero `nom_fichero`. Las líneas se numeran desde cero.

2.2. Proceso cliente

Este proceso simula a varios clientes realizando múltiples peticiones de forma concurrente al servidor. Para ello lanza varios hilos (uno por cada cliente a simular) y cada uno de los hilos intenta descargar un fichero del servidor usando TCP o UDP, como se describirá más adelante. El nombre del fichero a descargar y el protocolo a usar se lee de un fichero (todos los hilos leen del mismo fichero, pero cada uno una línea diferente).

El proceso cliente se denominará `cliente` (fichero fuente `cliente.c`) y se ejecuta mediante la siguiente línea de comandos:

Forma de uso de `cliente`

```
$ cliente ip_srvftp puerto_srvftp fich_pet nhilos fich_log
Parámetros:
  - ip_srvftp: dirección IP del servidor
  - puerto_srvftp: puerto en el que escucha el servidor
  - fich_pet: fichero de peticiones (ver más adelante)
  - nhilos: número de hilos que se lanzarán para simular a varios clientes
  - fich_log: fichero de log donde se escribirá información sobre peticiones fallidas
```

El fichero de peticiones es un fichero de texto en el que cada línea tiene la siguiente sintaxis:

Sintaxis de una línea del fichero de peticiones

```
Nom_fichero [T|U]
```

El primer token de cada línea es el nombre del fichero que se quiere descargar del servidor y el segundo token puede ser o bien una `T` o una `U`. Si es una `T`, la comunicación con el servidor se realizará a través del protocolo TCP y si es una `U`, la comunicación se realizará a través del protocolo UDP.

El cliente en primer lugar comprueba los parámetros recibidos a través de la línea de comandos. Luego abre el fichero de peticiones y reserva espacio para los objetos de hilo y crea tantos hilos como indique el parámetro `nhilos` recibido por línea de comando. Cada uno de esos hilos ejecuta la función `hilo_descarga()`.

El algoritmo de `hilo_descarga()` consiste en un bucle en el que en cada iteración, se lee la siguiente línea no leída del fichero de peticiones. La forma de lograr que cada hilo lea una línea diferente consiste en que el fichero se accede a través de una variable global compartida por todos los hilos, y que ha sido inicializada antes de lanzar los hilos. Cada vez que un hilo haga una operación de lectura sobre este fichero usando `fgets()`, se obtendrá una línea, y el cursor de lectura del fichero avanzará automáticamente a la siguiente. No obstante, hay que impedir que dos hilos lean la misma línea a la vez, por lo que se utilizará un mútex (`mutex_fpet`) para proteger la lectura del fichero.

Por cada línea leída del fichero de peticiones, se hace una primera petición para comprobar si el primer token (`Nom_fichero`) es un fichero válido (mensaje `ISFILE` del protocolo). En caso de que la respuesta sea `"OK"`, se hace una segunda petición para comprobar si el fichero anterior es legible (mensaje `ISREADABLE` del protocolo). En caso de que la respuesta sea de nuevo `"OK"`, se envía un tercer mensaje para obtener el número de líneas del fichero (mensaje `GETNUMLINES` del protocolo). Si el número de líneas es mayor de 0, se abre en el disco del cliente el fichero `Nom_fichero` en modo escritura y se ejecuta un bucle para pedir una a una las líneas del fichero (mensaje `GETLINE` del protocolo).

En caso de que en alguna de las peticiones anteriores devuelva el valor `"NOK"` se genera una línea en el fichero de *log*. Este fichero de *log* es compartido entre todos los hilos de descarga, por tanto, para realizar la escritura en el mismo se utiliza un mútex (`mutex_fplog`) para realizar la exclusión mutua de la escritura del fichero.

El resultado de los ficheros descargados se almacena en el mismo directorio en el que se lanza el cliente.

2.3. Pruebas

Se compila y ejecuta en primer lugar el servidor. Seguidamente se compila y ejecuta el cliente (podría hacerse desde otra máquina, o podría incluso haber varios clientes ejecutándose a la vez desde diferentes máquinas).

En la máquina del servidor debe haber una carpeta conteniendo ficheros de prueba, y el nombre de esa carpeta es lo que le tienes que pasar al servidor como argumento `ruta_absoluta_document_root`. Los nombres de esos ficheros son los que tienes que poner dentro de `fichpet.txt` del cliente, aunque también deberías probar con otros no existentes.

En la corrección se usarán otros ficheros de peticiones.

Al finalizar cada ejecución del cliente, en la máquina del cliente se podrá comprobar que se ha creado un fichero por cada petición de descarga realizada de forma exitosa. Estos ficheros contendrán el contenido descargado del servidor y estarán ubicados en el mismo directorio desde el cual se ejecutó el cliente. Además, se generará un fichero de log que contendrá información sobre las peticiones fallidas, si las hubiera.