Introducción

En la primera práctica de la asignatura de Redes de comunicaciones 2, se ha pedido que desarrollemos un servidor http en lenguaje C. El servidor soporta los siguientes verbos:

- GET: Para pedir recursos .gif .jpg .jpeg .txt .html .doc .docx .pdf .mpe y adicionalmente, para ejecutar scripts .py o .php.
- POST: Para pedir recursos .gif .jpg .jpeg .txt .html .doc .docx .pdf .mpe y adicionalmente, para ejecutar scripts .py o .php.
- OPTIONS: Para solicitar información sobre verbos soportados por el servidor.

Para el desarrollo, se ha escogido un servidor concurrente que crea un proceso mediante un fork() por cada petición que recibe, en contreto, este servidor recibe un máximo de 10 peticiones a la vez, pero esto puede ser configurado en fichero server.conf.

Dentro de la carpeta de la práctica podemos encontrar los siguientes ficheros de relevancia:

- server.conf: Fichero de configuración.
- makefile: Makefile para comiplar todos los ficheros necesarios para el correcto funcionammiento del servidor.
- README: ficher "leeme" con instrucciones para probar el servidor.

Por otro lado, Tenemos las siguientes carpetas:

- includes: ficheros .h
- obj: .o creados por el makefile
- recursos: carpeta que contiene recursos (necesariamente se tiene que llamar "recursos", pues así está configurado en el server.conf).
- src: Fichero principal del servidor server.c
- srclib: otros ficheros .c

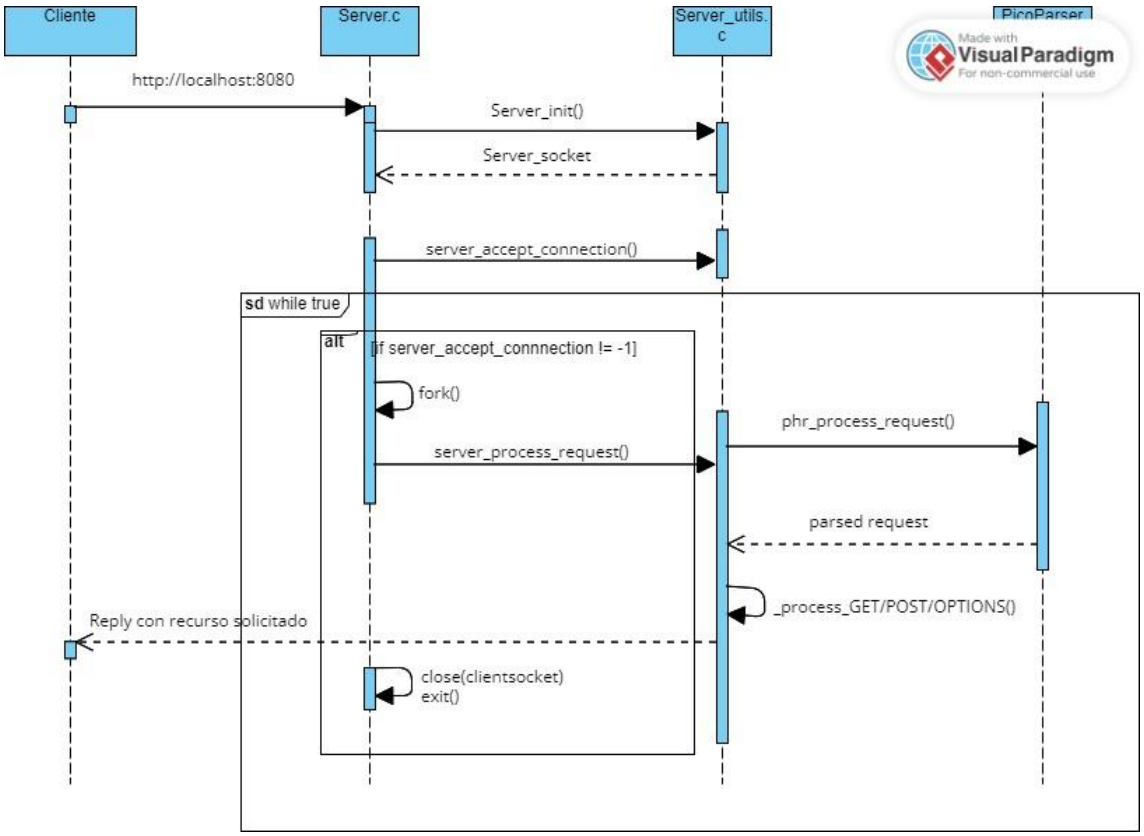
Diseño

Especificación de requisitos: Se identificaron los siguientes requisitos mencionados por el profesor y el enunciado:

- El servidor deberá enviar el mensaje *200 OK* en caso de que el procesamiento de la petición sea correcto.
- El servidor deberá enviar el mensaje *400 Bad Request* cuando no se pueda procesar una petición correctamente, debido a un error de sintaxis, por ejemplo.
- El servidor deberá enviar el mensaje _404 Not found _ en caso de que no se encuentre el recurso solicitado.
- El servidor deberá soportar el verbo GET para peticiones de recursos y ejecución de scripts. En el caso de los scripts, los argumentos se enviarán en la misma URL y deberán ser parseados.
- El servidor deberá soportar el verbo POST para petición de recursos y ejecución de scripts. En el caso de scripts, los argumentos se enviarán en el cuerpo de la petición y deberán ser parseados.

- El servidor deberá tener un fichero de configuraciñon para configurar: raíz del servidor, número máximo de clientes, puerto del servidor, nombre del servidor.
- El servidor deberá ser capaz de enviar los siguientes tipos de recursos: .txt, .html, .htm, .jpeg, .jpg, .mpeg, .mpg, .doc, .docx, .pdf.
- Las cabeceras de las respuestas por parte del servidor deberán tener como mínim: Date, Server, Last-Modified, Content-Lenght, Content-Type.

Documentos de diseño Hemos decidido incluir un diagrama de secuencia para el caso de uso *cliente hace una petición* http:



Desarrollo Técnico

Como mencionado previamente, la funcionalidad principal de esta práctica consiste en implementar el soporte para los verbos GET/POST/OPTIONS y los errores 404/400. A continuación, se explicará las decisiones tomadas para desarrollar cada uno de ellos a lo largo de la práctica.

Funcionalidad del servidor

El servidor, como mencionado anteriormente, crea un nuevo proceso mediante una llamada a la función fork(), por cada petición que llega, el proceso finalizará cuando la petición termine, se produzca un error o se genere una interrupción por el teclado. El proceso principal, el del servidor, siempre estará en ejecución, esperando peticiones de nuevas conexiones hasta que se genere una interrupción por el teclado.

GET

Hay dos funciones que procesan una petición con el verbo GET, dependiendo si la petición lleva argumentos en la URL o no.

- Sin argumentos: Se ejecuta la función `int _process_GET(int clientsocket, char *server_root, char *path, char *server_signature, HttpRequest *request)` .Lo primero que hace la función es comprobar que el recurso solicitado existe, si no existe se enviará el error 404 Not Found. Luego, se consigue el tipo de extensión, la fecha y se construyen las cabeceras de la petición. Para el envío, hemos decidido primero enviar la cabecera y posteriormente el recurso solicitado para no sobrecargar el servidor.

  Para enviar el recurso se utiliza la función read(), que lee el contenido en un char y lo envía.

- Con argumentos: Se ejecuta la función `int _process_GET_ARGS(int clientsocket, char *server_root, char *path, char *server_signature, HttpRequest *request)` .A diferencia de la anterior función, esta comprueba que la extensión del recurso sea tipo .py o .php. Si no es así envía el error 400 Bad Request. A continuación parsea los argumentos y si hay más de un argumento reserva memoria para cada argumento que haya.

Posteriormente, construye el comando de ejecución y procede a ejecutarlo con la función popen(). Decidimos usar esta última función dado que inicializa un nuevo proceso para la ejecución de un script sin necesidad de llamar a funciones adicionales como fork().

Finalmente, se lee el resultado en un string, se construye la cabecera http y se envía la respuesta utilizando una sola llamada a send().

POST

Para procesar una petición con el verbo POST, se ha desarrollado una sola función. `int _process_POST(int clientsocket, char *server_root,char *path,char *server_signature,HttpRequest *httprequest, char *args_post)`. Esta función primero comprueba si se debe ejecutar un script o de lo contrario, se trata de una simple petición de un recurso. Si es un script, a diferencia del GET, se cogerán los argumentos del cuerpo de la petición, serán parseados, se construirá el comando, se ejecutará el script con popen() y finalmente, se enviará la respuesta. Por otro lado, si no es un script, se hará el mismo procedimiento para un GET sin argumentos.

OPTIONS

La implementación del options es la más sencilla entre los verbos soportados. Simplemente, se construye la cabecera con la fecha, nombre del servidor y métodos permitidos, que en este caso son GET, POST y OPTIONS. Finalmente, se envía haciendo una sola llamada a send().

ERRORES

Los errores 404 y 400 se implementan de manera similar, simplemente se construye una cabecera con el mensaje de error que corresponda, se hace send() de esta, y posteriormente se hace un send() adicional para enviar un documento .html informando dicho error.

Fichero de configuración

Cabe mencionar, que se proporciona un fichero de configuración: server.conf que permite modificar parámetros del servidor como: número máximo de cliente, raíz del servidor, puerto usado y nombre del servidor.

Puesta en marcha del servidor

Para ejecutar y probar el servidor, basta con ejecutar *make all* desde el directorio donde se encuentra el makefile. En ese mismo, se creará un ejecutable *server* , habrá que ejecutar el siguiente comando: ./server

Conclusiones:

En conclusión, en esta práctica se entendió mejor la implementación del servidor http, pues se tuvo que desarrollar el mismo. Se aprendió a utilizar las funciones de los sockets y a la misma vez, se evaluaron las ventajas y desventajas de distintos tipos de servidores para tomar una decisión sobre el tipo de servidor a implementar. Por otro lado, se mejoraron las habilidades en el lenguaje de programación C.

La mayor parte de dificultades encontradas corresponden al lenguaje de programación usado, pues C es un lenguaje que exige prestar mucha atención a los tipos de datos que se usan y punteros.