75:42 - Taller de Programación I

Ejercicio Nº 1 Padrón: 100498

Alumno: Di Maria, Franco Martin

Nota:		Corrige:		Entrega #1
				Fecha de entrega
				2/4/2019
				Fecha de devolución

Nota:		Corrige:		Entrega #2
				Fecha de entrega
				Fecha de devolución

Problemas y Soluciones

El cliente debe tomar un archivo de texto que contiene la petición http y la debe enviar al servidor. Para lograr esto de manera eficiente, se opta por leer el archivo de texto de a paquetes de bytes, que por decisión arbitraria se toma 1kb (1024 bytes), pues de esta manera evitamos tener todo el archivo en memoria en caso de que la petición sea muy larga, y solo tenemos un parte de ella, al mismo tiempo. Para recibir la respuesta del servidor, tras enviar el petitorio, se opta por una política idéntica. Se va recibiendo la respuesta de a paquetes de 1 kb, y se lo va imprimiendo por pantalla a medida que llega cada uno.

En el caso del servidor, se realizó algo parecido a lo mencionado con el cliente, pero con algunas diferencias. Dado que solo se aceptan clientes de uno por vez, se decidió mantener el archivo binario abierto y extraer las temperaturas censadas de una a la vez hasta llegar al final del mismo. Por otro lado, para el archivo template, dado que por cada respuesta para un petitorio recibido “bien escrito”, se enviará todo el contenido del mismo, modificando cierta parte de él, se decidió que es mucho más óptimo cargarlo todo en memoria, y tenerlo separado en dos partes que vienen antes y después de la cadena “{{datos}}”, para luego volver a unirlo separados por la temperatura correspondiente y enviarlo junto a la cabecera de la respuesta.

Una parte dificultosa del trabajo fue recibir los petitorios http (del cliente) en el servidor.
Similar a casos anteriores, se cicla recibiendo paquetes de bytes de un máximo de 1 kb por ciclo (procesar_peticion es el nombre de la función que se encarga de esto). Por cada paquete recibido es necesario verificar si con él ha llegado algún ‘\n’ que marca el fin de una línea. Para ello se implementa el tda linea_t el cual tiene una primitiva que permite ir agregando cadenas de caracteres, que devuelve la cantidad de caracteres que no agrego si es que se encuentra con un \n en el camino. De esta manera, se delega en este tda la tarea de almacenar los caracteres que vamos recibiendo, con todo el manejo de memoria que implica, así como una forma de identificar cuando terminamos de leer una línea.

Tras confirmar el fin de una línea se la procesa según si es la primera o no.

En caso de ser la primera, se llama a una función responder_peticion que nos devuelve un puntero a cadena de caracteres con toda la respuesta, lo cual incluye cabecera, y si el petitorio es correcto, el cuerpo de la misma, es decir el template con su dato reemplazado. En este caso, se tiene toda la respuesta en memoria, por lo que al momento de enviarla al cliente, no nos restringimos a enviarla de a paquetes, sino que se envía todo junta.

Tras procesar la primer línea, hace falta procesar el resto de caracteres que sobraron según nos indica el valor de retorno de la primitiva de linea_t. Pero antes de continuar, hay que verificar si la petición recibida fue válida. Si la petición fue válida, significa que la respuesta incluye un cuerpo compuesto por el template modificado. Esto último es similar a decir, que el largo de la respuesta es mayor a cierta cantidad de caracteres que corresponde al largo máximo que puede tener la cabera de la respuesta. Aprovechando que se conoce este largo correspondiente al largo de la cabecera de la respuesta (más los caracteres de salto de línea que le siguen), solo se necesita corroborar que el largo de respuesta sea mayor a dicho valor.

Una vez corroborado la validez de la petición se procede a procesar el cuerpo de la misma. Para ello

Di Maria, Franco Martin
100498

se tiene la función procesar_cuerpo y procesar_linea_cuerpo quienes se encargan procesar los caracteres sobrantes del último paquete recibido hasta que no quede ningún ‘\n’ entre los mismo (pues en los caracteres sobrantes puede llegar a haber más \n que aun no se verificaron) y los siguientes paquetes que irán llegando.

Para procesar las líneas que no son la cabecera de la petición, se implementa una función split para separar cada línea del cuerpo de la petición en su campo y valor. De esta manera, se puede verificar que campos son “User-Agent”, y procesar su recurso asociado. Con respecto a este último, se implementan dos tda’s : recursoVisitado_t y recursosVector_t . El primero almacena el nombre del recurso, y lleva cuenta de las veces que es visitado, mientras que el segundo es un vector de los primeros, que se encarga de verificar si cada nuevo recurso que es procesado, está o no esta agregado, y visitarlo (llamar a primitiva de recursoVisitado_t que aumenta su contador de visitas) en caso de ser lo primero.

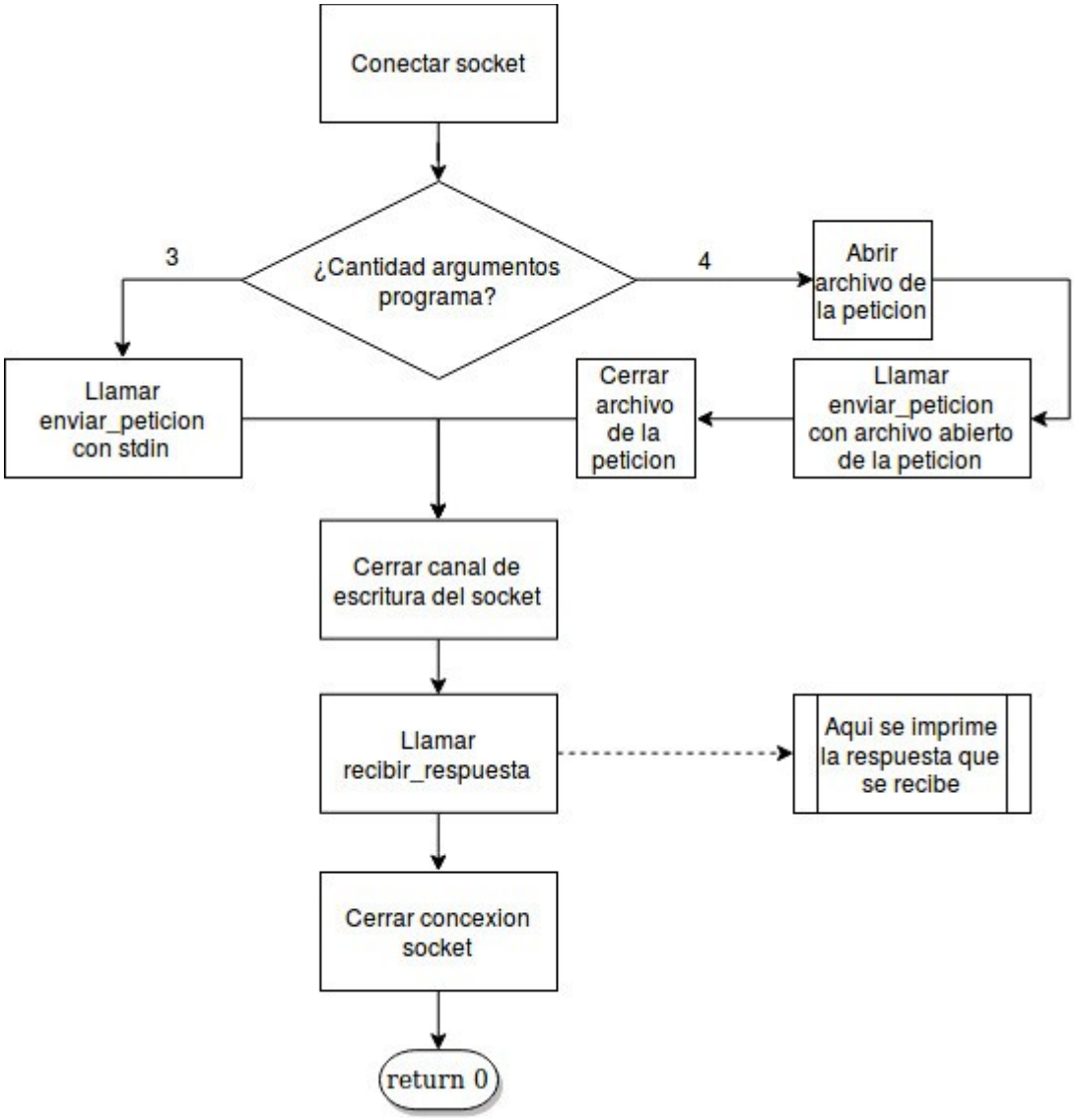
Con respecto al uso de sockets, todo lo correspondiente a la inicialización de los mismos, se extrae en funciones aparte (una para el cliente y otra para el servidor) que se encargan de setearlos y conectarlos al host y puerto correspondiente. Una vez inicializados se usan en las funciones principales de cada programa.

Esquemas del diseño

Diagramas de Flujo del programa:

Cliente envía petición y recibe respuesta.

El diagrama de flujo muestra a grandes rasgos un procedimiento normal (omitiendo posibles errores de ejecución) en el que un cliente envía una petición al servidor y recibe una respuesta del mismo.
Nota: no se muestra llamadas a sub-funciones.

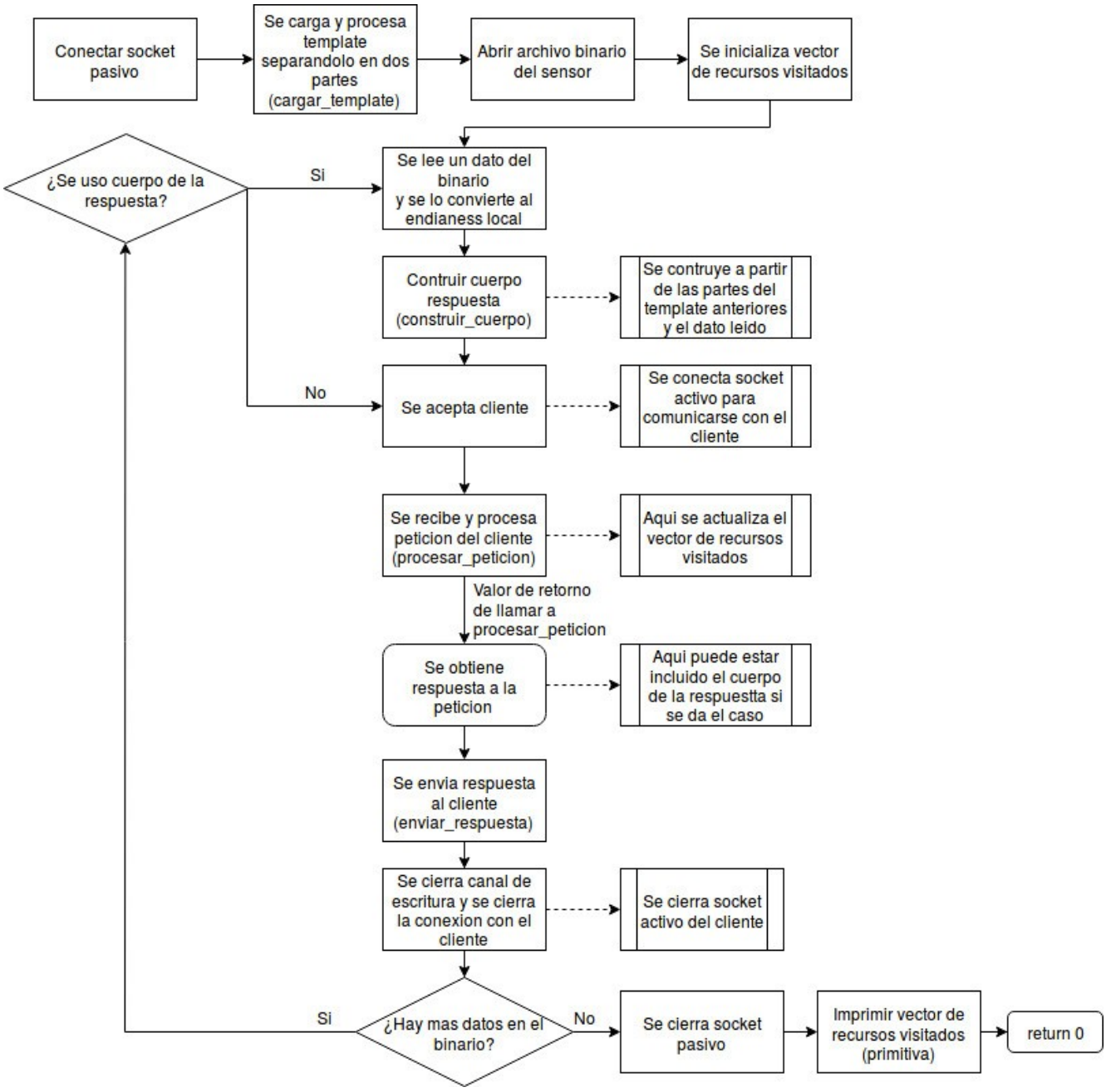


Di Maria, Franco Martin
100498

Servidor ciclo recibir petición y responder

El diagrama de flujo muestra a grandes rasgos el procedimiento normal (omitiendo errores de ejecución) de aceptación de clientes, recibir sus peticiones y responderlas.

En algunos recuadros puede verse, encerrado entre paréntesis, el nombre de la función que ejecuta los recuadros que le suceden. Estas funciones están separadas por un guión bajo.

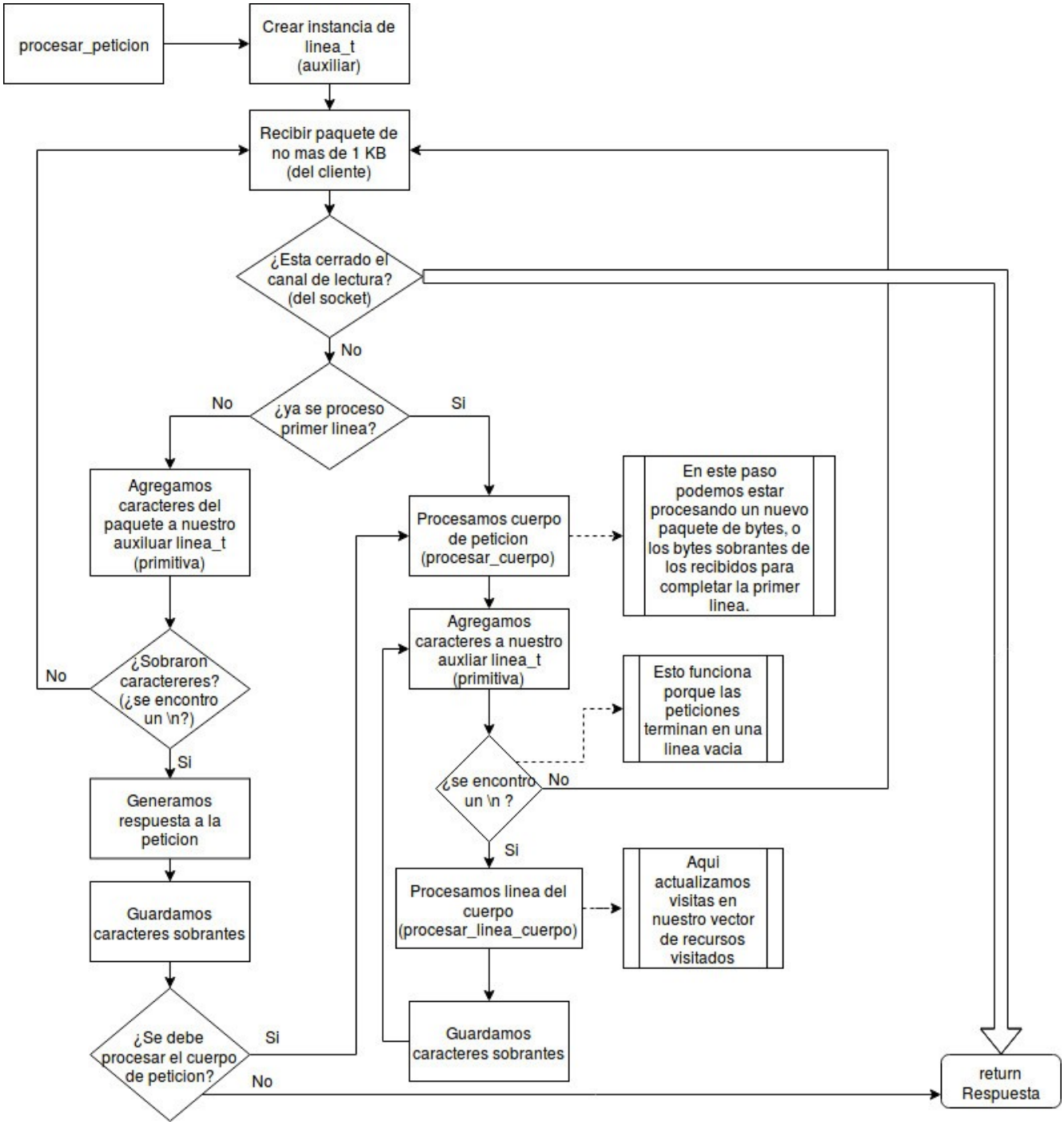


Di Maria, Franco Martin
100498

Servidor recibe petición, la procesa armando respuesta y actualizando visitas de los recursos (User-Agent).

El diagrama de flujo muestra el procedimiento normal (omitiendo errores de ejecución) de como se recibe y procesa la petición de un cliente, hasta obtener la respuesta a dicha petición, y actualizar datos de los recursos (User-Agent) que leen en el cuerpo de la petición.

En algunos recuadros puede verse, encerrado entre paréntesis, el nombre de la función que ejecuta los recuadros que le suceden. Estas funciones están separadas por un guión bajo.



Di Maria, Franco Martin
100498