Sensor Arduino sobre HTTP

Ejercicio N° 1

Objetivos	● Buenas prácticas en programación de Tipos de Datos Abstractos (TDAs) ● Modularización de sistemas ● Correcto uso de recursos (memoria dinámica y archivos) ● Encapsulación y manejo de Sockets
Instancias de Entrega	**Entrega 1:** clase 4 (02/04/2019). **Entrega 2:** clase 6 (16/04/2019).
Temas de Repaso	● Uso de structs y typedef ● Uso de macros y archivos de cabecera ● Funciones para el manejo de Strings en C ● Funciones para el manejo de Sockets
Criterios de Evaluación	● Criterios de ejercicios anteriores ● Cumplimiento de la totalidad del enunciado del ejercicio ● Ausencia de variables globales ● Ausencia de funciones globales salvo los puntos de entrada al sistema (*main*) ● Correcta encapsulación en TDAs y separación en archivos ● Uso de interfaces para acceder a datos contenidos en TDAs ● Empleo de memoria dinámica de forma ordenada y moderada ● Acceso a información de archivos de forma ordenada y moderada

Introducción

El presente trabajo está inspirado en un ejercicio de las olimpiadas de electrónica del INET 2018, donde alumnos de secundaria utilizaban Arduino para controlar un ascensor, y utilizaban un shield de red para sensar el estado del mismo.

Descripción

Arduino es una famosa familia de placas de desarrollo, que permite programar de forma sencilla microcontroladores Atmel. Estas placas, generalmente usadas con fines educativos, poseen variedad de módulos compatibles, como pueden ser sensores, dispositivos de entrada y salida, etc. Uno de estos módulos más utilizados es el shield Ethernet.

El shield Ethernet le permite conectarse al Arduino a una red cableada, utilizando una biblioteca sencilla que provee funciones como:

- Ethernet.begin(mac): Intenta conectarse a una red con servidor DHCP (para configurarse IP de forma automática).
- Ethernet.linkStatus(): permite saber si el dispositivo está conectado a la red.
- client.print(msg): Envía una cadena de texto msg por medio de una conexión TCP, utilizando el cliente client

En el presente trabajo utilizaremos sockets TCP para simular un shield Ethernet, y utilizaremos un archivo binario para simular lecturas de un sensor de temperaturaf del Arduino.
Se implementará una aplicación que poseerá dos modos:

Cliente: Se conecta a un servidor y envía un petitorio HTTP. Una vez que lo envía espera una respuesta del servidor y se cierra.

Servidor: Abre un archivo binario, y mientras tenga datos escucha conexiones entrantes. Una vez que se conecta un cliente, escucha su petitorio HTTP y extrae de él qué recurso necesita y qué navegador (User-Agent) es.
Luego envía un template con datos de la lectura al cliente, y cierra la conexión. El servidor escucha clientes hasta que no hay más datos. Luego imprime un resumen de los clientes conectados.

Cliente

La funcionalidad del cliente es muy básica, por lo que se recomienda comenzar por la misma. El cliente debe conectarse a un *host* y un *puerto*, utilizando el protocolo *TCP*. Si recibió el *path* de un archivo de entrada enviará su contenido al servidor y al finalizar cerrará el canal de

escritura del socket. Si no recibe el *path* se utiliza la entrada estándar como fuente de datos. Luego de enviar el petitorio, el cliente recibe la respuesta del servidor y la escribe en pantalla. Una vez que termina de recibir datos, se cierra el socket, se liberan todos los recursos y se termina la aplicación con 0 como código de salida.

Formato de un pedido HTTP

Un pedido HTTP del cliente (*request*) está formada por lineas de texto separadas por un caracter de retorno y otro de salto de línea (*CRLF*). Estas líneas forman 3 bloques

- Método del request y sus parámetros:

Se envía la acción deseada (GET, PUT, POST, etc), y sus parámetros, separados por espacios. Sus parámetros son: el recurso al que se quiere acceder, y la versión del protocolo HTTP. Para este proyecto se utilizará el protocolo HTTP/1.1
Ejemplo:

```
GET /template.html HTTP/1.1
```

- Cabecera (header):

Le provee información de la conexión al servidor. Está compuesto por lineas con el formato `clave: valor`. El único parámetro obligatorio para el protocolo HTTP/1.1 es el de Host, debido a que se utiliza para discernir hosts virtuales.
En este proyecto se utilizará el parámetro User-Agent para identificar la aplicación del cliente.
Ejemplo:

```
Host: localhost:8080
User-Agent: netcat
```

- Cuerpo (body):

Este bloque se utiliza para mandar datos en los pedidos principalmente de tipo *POST*. La longitud de este bloque se especifica en el header, con el parámetro *Content-Length*.
No se utilizará en este proyecto.

El cuerpo y cabecera finalizan con una línea en blanco. Una vez que el cliente envía el petitorio, cierra el canal de escritura y espera la respuesta del servidor.

Formato de la respuesta HTTP

El formato de la respuesta (*response*) posee dos partes separadas por una línea en blanco. La primer parte, la cabecera, está compuesta por la línea de estado (*status*), la sección de cabecera, similar a la de un *request*, y el cuerpo del mensaje.

La línea de estado tiene el siguiente formato:
El protocolo utilizado será "HTTP/1.1"
Si el petitorio fue válido (ver detalle más adelante en la sección del servidor), el status es "200" y la descripción "OK".
Si el petitorio no tiene un formato válido o es una acción no válida, el status es "400" y la descripción es "Bad request".
Si el petitorio es válido, pero el recurso buscado no es válido, el status es "404" y la descripción es "Not found".

Servidor

El servidor abre un archivo binario que simula ser un sensor. Luego escucha conexiones en un puerto, y, mientras haya datos para leer en el archivo binario, realiza lo siguiente:

- Se leen los datos del sensor.
- Acepta un cliente y recibe un *request* HTTP.
- Verifica que el método utilizado sea del tipo "GET" y el recurso sea "/sensor". Si el método no es "GET", la respuesta será un error de tipo "400 Bad request", y si el recurso no es "/sensor", la respuesta será un error de tipo "404 Not found". Si el método y recurso son válidos, la respuesta es de tipo "200 OK".
- Lee la cabecera del *request*, si encuentra la clave "User-Agent", lee su valor y cuenta una visita de este agente. La cabecera finaliza con una línea en blanco.
- Una vez que procesó la cabecera, le envía el cuerpo del mensaje:
    - Si el *request* es válido, el cuerpo del mensaje es el contenido del archivo template, reemplazando el marcador `{{datos}}` por los datos del sensor (ver más adelante).
    - Si el *request* es inválido, no habrá cuerpo del mensaje.

Si el request fue válido, se intenta leer otro dato del archivo del sensor, y se repite el ciclo. Si el request fue inválido, se utiliza la lectura anterior del sensor.

Sensor

Los datos del sensor de temperatura son simulados por un archivo binario. Este archivo está compuesto por números de 16 bits en formato *big-endian*. Los mismos se interpretan de la siguiente forma: Temperatura = (datos - 2000) / 100.
Ejemplo:
Dado el archivo cuyo contenido es: `10 00 03 E8`, (4 bytes),
extraigo 2 números de 16 bits cada uno: `0x1000` (4096) y `0x03E8` (1000),
que corresponden a las temperaturas "20.96" y "-10.00" respectivamente.

> Nótese que el archivo tiene 4 bytes: la notación `10 00 03 E8` es solo para representar el contenido del mismo (binario) en números hexadecimal.

Template

El archivo *template* es un archivo de texto con la plantilla de respuesta del servidor. El mismo posee código HTML, y en alguna parte del archivo el texto `{{datos}}`. Este texto será suplantado por la temperatura leída por el sensor.

Formato de línea de comandos

El cliente se ejecuta con la siguiente línea de comandos:

```
./client <host> <port> [<filename>]
```

Donde **<host>** y **<port>** son la dirección IPv4 o *hostname* y el puerto o servicio donde el servidor estará escuchando la conexión TCP.

<filename> es un argumento opcional que indica el ***archivo de texto*** con el request a enviar. Si el argumento no es pasado, el cliente leerá el request de la ***entrada estándar***.

El servidor se ejecuta con la siguiente línea de comandos:

```
./server <port> <sensor-filename> <template-filename>
```

Donde **<port>** es el puerto de escucha del servidor, **<sensor-filename>** es el nombre del archivo del sensor, y **<template-filename>** es el template HTML para responder los petitorios HTTP exitosos.

Entrada y salida estándar

Si el cliente tiene un número de parámetros incorrecto, se imprime por salida de error estándar lo siguiente:

```
Uso:
./client <direccion> <puerto> [<input>]
```

Si el servidor tiene un número de parámetros incorrecto, se imprime por salida de error estándar lo siguiente:

```
Uso:
./server <puerto> <input> [<template>]
```

El cliente, en caso de no recibir por parámetro el nombre del archivo de entrada, lee el contenido del request mediante la entrada estándar.
Luego de enviar el *request HTTP*, espera la respuesta del servidor y la imprime por salida estándar

El servidor no recibe información de la entrada estándar. Luego de finalizar la lectura de datos del sensor, escribe por salida estándar un resumen de los navegadores que visitaron al dispositivo.

El informe del servidor posee el siguiente formato:

```
# Estadisticas de visitantes

* <nombre-agente-1>: <cantidad-visitas>
* <nombre-agente-2>: <cantidad-visitas>
* <nombre-agente-n>: <cantidad-visitas>
```

Los agentes se numeran según el **orden de aparición**.

Códigos de Retorno

Tanto el cliente como el servidor deben retornar **`0`** si todo salió correctamente o **`1`** en caso contrario.

Consideraciones

Se puede suponer, para simplificar la implementación del servidor, que los *requests* entrantes siempre escriben texto y que el cliente cierra el canal de escritura al enviar el request.
El protocolo HTTP utiliza '`\r\n`' como caracteres de salto de linea. Para facilitar la escritura de pruebas, se utilizará '`\n`' como caracter de fin de linea. Esto debe ser contemplado si se utiliza un cliente HTTP real para probar el servidor.

Restricciones

La siguiente es una lista de restricciones técnicas exigidas por el cliente:

1. El sistema debe desarrollarse en ISO C (C99).
2. Está prohibido el uso de variables globales.

Bibliografía

- Sesión http típica: https://developer.mozilla.org/en-US/docs/Web/HTTP/Session
- RFCs 7230, 7231:
    - Content Length: https://tools.ietf.org/html/rfc7230#section-3.3.2
    - https://tools.ietf.org/html/rfc7231#section-3.1.1.5
    - Date: https://tools.ietf.org/html/rfc7231#section-7.1.1.2
- Respuesta HTTP: https://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html