INTEGRACIÓN DE APLICACIONES MEDIANTE LA GENERACIÓN DE ARCHIVOS XML

PRODUCTO 4

Daniel Carrasco Luque

FP.047 – (P) Programa comandos personalizados para el sistema operativo

ÍNDICE

Introducción

Para este producto el programa ha de ser capaz de generar un documento XML con los datos que se nos solicitan de un adaptador de red seleccionado de nuestra máquina. Se ha realizado un estudio previo de los productos creados anteriormente para identificar que código se podía aprovechar directamente, cual se debe adaptar y que funcionalidades nuevas se deben crear para obtener los datos necesarios para nutrir el fichero XML. Teniendo ya planificado que se debe desarrollar, era importante conocer que implica la creación del fichero XML y como debemos estructurarlo para obtener un resultado optimo, sabiendo que en el producto 5 el fichero resultante debe ser tratado mediante una hoja de estilo XSLT. Este enfoque me ha ayudado a realizar modificaciones sobre el planteamiento inicial de la estructura de etiquetas del fichero XML obteniendo un resultado que considero optimo para poder ser tratado en el producto 5.

Funcionalidades aprovechadas

Se ha aprovechado sin tener que realizar modificaciones seleccionarAdaptadorRed(), esta función evalúa los adaptadores de red del sistema y los muestra en un menú indexado para poder escoger el adaptador con el cual trabajar, esta función llama de forma interna a obtenerDNSAdaptador() la cual obtiene los DNS primario y secundario del adaptador. También se ha aprovechado tiempoMedioPing() que dada una ip retorna el tiempo medio de llamada a la ip pasada por parámetro usando ping para ello.

Nuevas funcionalidades y mejoras

Para poder aprovechar por completo las funciones anteriores y unificar la forma de almacenar los datos en las nuevas funcionalidades se ha ampliado la estructura de datos “AdaptadorRed” ya creada en el producto anterior. Esta estructura ha pasado de estar formada por 3 strings para contener los datos del nombre de adaptador y las dos DNS a estar formado por todos los parámetros que se nos requiere en el producto, inclusive un listado de las ip de los saltos hasta la DNS principal si es que la hubiera.

```
typedef struct {
    char adaptador[300];
    char dnsPrimario[20];
    char dnsSecundario[20];
    char ip[20];
    char puerta[20];
    char mascara[20];
    int tiempoMedio;
    int saltos;
    char listaSaltos[30][20];
} AdaptadorRed;
```

Se ha seguido manteniendo en la estructura la variable “dnsSecundario[20]” pese a no solicitarse trabajar con los datos resultantes de esta, por compatibilidad con las funciones desarrolladas, de esta forma no hay que adaptar el código.

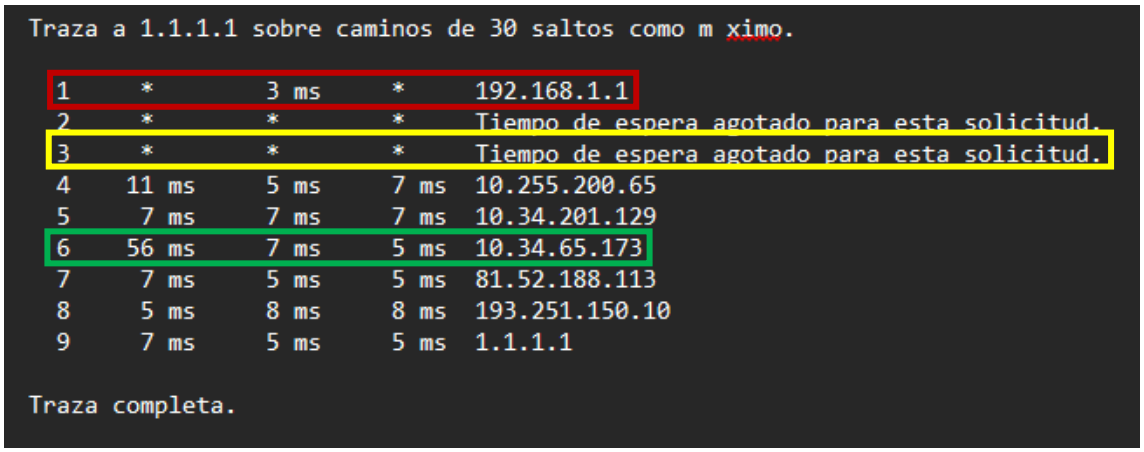
Se ha tenido que desarrollar obtenerIpPuertaMascar() ya que la obtención de los datos de IP, máscara de subred y puerta de enlace los desarrollé en el producto 2 a partir del comando “ipconfig”. Pese a que en el producto 2 pudo ser correcto, para este producto no me parecía correcto ya que en todo momento trabajamos con los datos ofrecidos por “Netsh”. De esta forma estandarizo la forma de trabajar y me ha servido para seguir desarrollando diferentes formas de recorrer los ficheros.

“obtenerIpPuertaMascara()” recibe por parámetro un puntero de tipo “AdaptadorRed”, aportando a esta estructura la IP, la máscara de subred y la puerta de enlace del adaptador de red contenido en la estructura de datos. Para ello se realiza una llamada con el comando “netsh interface ip show config name=”nombre_adaptador” > netsh_temp.txt”. Para filtrar línea a línea el fichero se utiliza una estructura condicional anidada en un bucle while que avanza línea a línea. Mediante strstr() se busca la línea que contenga “IP:” para extraer la IP mediante el uso de strtok() avanzamos por el string separándolo en subcadenas. Máscara de subred y puerta de enlace siempre están en las siguientes líneas así que se avanza de forma manual para analizar las 2 siguientes líneas mediante strtok() y obtener los resultados.

Debido a que en mi equipo tengo 3 adaptadores de red diferentes y he estado jugando con las configuraciones, he podido ver como este método puede arrojar diferentes resultados considerando el más critico que no tenga una IP, se han implementado casos por defecto para evitar errores por falta de datos, estableciendo la palabra “ninguno” en caso de que no exista valor para el parámetro buscado.

La siguiente funcionalidad que se ha tenido que implementar para obtener el resto de datos faltantes es “obtenerSaltos()” el cual recibe por parámetro un puntero de tipo “AdaptadorRed” y retornará mediante este el número de saltos realizados y las IPs incluida la DNS primaria. El método analiza el fichero resultante de usar “tracert” mediante la siguiente sentencia “tracert -d -w 1 “dns_primario” > tracert.txt ” donde -d implica que solamente retorne la IP y no el nombre asignado que pueda tener y “-w 1” establecemos el tiempo de respuesta al mínimo para acelerar lo máximo posible la ejecución de nuestro programa.

He de comentar que este método es el que más tiempo de desarrollo le he tenido que dedicar ya que en una de las diferentes pruebas que realicé con “tracert” pude observar cómo existe la posibilidad de que alguna de las 3 peticiones que realiza sobre la IP retorne “*”, se detalla con un ejemplo, marcado en rojo, en el siguiente fichero modificado para las pruebas. En verde un resultado con respuesta en las 3 peticiones. En amarillo los casos donde existe IP, pero excede el tiempo de respuesta, normalmente un firewall o una máquina que no admite respuestas.



A la hora de recorrer el fichero línea a línea lo primero que se analiza mediante strstr() es si existe el string "Tiempo" de esta forma damos un tratamiento a las posible líneas que contienen un salto a contabilizar pero no contienen IP, en el campo correspondiente de "listaSaltos" la IP se almacena como "Desconocida" y se incrementa el contador de saltos también contenido en la estructura "AdaptadorRed".

Teniendo en cuenta lo comentado anteriormente marcado con cuadro rojo y verde no podía analizar las líneas mediante strtok porque no siempre serán ideales (marca en verde) por lo cual, he buscado una solución que mediante un bucle combinado con strstr() busca la subcadena "ms" y guarda la posición en un puntero para posteriormente incrementarlo en 2 asignándolo a un segundo puntero de esta forma consumo los "ms". Una vez se han consumido los "ms" el string resultante puede contener solamente espacios o espacios y asteriscos, lo siguiente que he hecho ha sido recorre el string con un segundo while incrementando la posición del puntero cada vez que encuentra espacio ' ' o asterisco '*'.

Una vez tenia la solución a esto descubrí que los strings resultantes con la IP seguían teniendo un espacio antes de "\n", así que mediante strlen() y una variable de tipo "size_t" obtuve la longitud de la cadena para sustituir " \n" por "\0" en la posición longitud – 2 obteniendo un string listo para almacenar en "listaSaltos". La ejecución del bucle se finaliza antes de recorrer la totalidad del archivo en caso de que la ip encontrada sea la DNS primaria.

Desarrollo del fichero XML

Como comenté en la introducción, para crear el fichero XML no solo me he basado en lo que he ido aprendiendo de XML, si no que he decidido adelantarme al producto 5 para ver como se trata este documento, esto ha hecho cambiar el planteamiento inicial que tenia sobre mi estructura XML. Al analizar el código XML y XSLT sobre una colección de CD's proporcionado en los recursos del aula y otros códigos de ejemplo que he ido evaluando, he visto que XSLT hace uso de bucles "for-each", esto me ha hecho comprender como se recorren los ficheros XML y la importancia de definir correctamente las etiquetas para facilitar su procesado posterior.

Primera versión:

En esta primera versión pese a que todo esta definido aparentemente correcto y mi planteamiento era contemplar la posibilidad de dejar el formato abierto a un <DNSSecundario> que tenga la misma estructura de subetiquetas no sería fácil de recorrer al implementar el XSLT, lo mismo pasa con la correlación de <Salto1>...<Salto9>. Este planteamiento no es escalable y requeriría de mucho código o de varias plantillas XSLT. De momento desconozco todas las posibilidades de XSLT, pero se hace evidente.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<adaptador>
    <nombre>Wi-Fi</nombre>
    <ip>192.168.1.13</ip>
    <mascara>255.255.255.0</mascara>
    <puerta>192.168.1.1</puerta>
    <DNSPrimario>
        <ip>1.1.1.1</ip>
        <tiempo>8</tiempo>
        <saltos>9</saltos>
        <salto1>192.168.1.1</salto1>
        <salto2>Desconocida</salto2>
        <salto3>Desconocida</salto3>
        <salto4>10.255.200.65</salto4>
        <salto5>10.34.201.129</salto5>
        <salto6>10.34.65.173</salto6>
        <salto7>81.52.188.113</salto7>
        <salto8>193.251.150.10</salto8>
        <salto9>1.1.1.1</salto9>
    </DNSPrimario>
</adaptador>
```

Versión definitiva:

Aprendiendo de la anterior versión se ha desarrollado la versión definitiva de la estructura XML, donde <DNSPrimario> se sustituye por <DNS> y en sus subetiquetas se crea <tipo> para indicar el tipo de DNS primario o secundario, con este cambio conseguimos recorrer infinidad de bloques de datos con etiqueta DNS. El segundo cambio importante ha sido definir la etiqueta <saltar> y renombrar todas sus subetiquetas de la misma forma <ip> de esta manera definimos un bloque <saltar> que puede contener todas las <ip> y ser recorridas en bucle sin complicaciones.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<adaptador>
    <nombre>Wi-Fi</nombre>
    <ip>192.168.1.13</ip>
    <mascara>255.255.255.0</mascara>
    <puerta>192.168.1.1</puerta>
    <DNS>
        <tipo>primario</tipo>
        <ip>1.1.1.1</ip>
        <tiempo>8</tiempo>
        <saltos>9</saltos>
        <saltar>
            <ip>192.168.1.1</ip>
            <ip>Desconocida</ip>
            <ip>Desconocida</ip>
            <ip>10.255.200.65</ip>
            <ip>10.34.201.129</ip>
            <ip>10.34.65.173</ip>
            <ip>81.52.188.113</ip>
            <ip>193.251.150.10</ip>
            <ip>1.1.1.1</ip>
        </saltar>
    </DNS>
</adaptador>
```

Teniendo claro el fichero objetivo a generar y habiendo obtenido todos los datos necesarios del adaptador previamente, la codificación en C ha sido sencilla. Se ha creado un método llamado crearXML() el cual recibe por parámetro un puntero de tipo “AdaptadorRed”. Teniendo la posibilidad de elegir los adaptadores de red se ha creído oportuno crear un nombre de fichero diferenciado para cada uno de ellos por lo cual el nombre del fichero será “nombre_adaptador_red.xml”.

Se crea el fichero y mediante fprintf() se imprime la cabecera "<?xml version="1.0" encoding="UTF-8"?>" para indicar que se trata de un documento XML y posteriormente se van extrayendo los datos línea a línea y acomodando en sus etiquetas correspondientes.

Webgrafía

- XML Validator
- XML Tutorial
- XSLT Tryit Editor v1.2