TP6 — Sistema de Caché DNS con Tabla Hash

«Tu Nombre»
«Curso y Legajo»

6 de noviembre de 2025

Resumen

En este trabajo se desarrolla un sistema de caché DNS utilizando una **tabla hash con encadenamiento** para el manejo de colisiones. El objetivo es aplicar los conceptos de memoria dinámica, estructuras anidadas y listas enlazadas vistos en la **Guía 3** de la materia, junto con las **Consideraciones Finales** de modularización, manejo de errores y pruebas.

1. Enunciado resumido

El trabajo práctico requiere implementar un sistema que guarde registros DNS en una *tabla hash*, permitiendo:

- Insertar, buscar, actualizar y eliminar entradas DNS.
- Limpiar entradas expiradas según su TTL.
- Mostrar estadísticas del caché: cantidad de entradas, colisiones, factor de carga.
- Interactuar con el usuario mediante un menú de opciones.

2. Fundamentos de la Guía 3

- Uso de `malloc` y `free` para reservar y liberar memoria dinámica.
- Estructuras anidadas (`typedef struct`) para modelar entidades complejas.
- Listas simplemente enlazadas para recorrer, insertar y eliminar nodos dinámicos.

3. Descripción de las estructuras de datos utilizadas

El sistema se basa en las siguientes estructuras principales:

EntradaDNS

Agrupa toda la información de un registro DNS:

- **Registro**: dominio, tipo, IP (IPv4 e IPv6).
- **Metadatos**: TTL, timestamp de cacheo, cantidad de hits y servidor origen.
- **Estadísticas**: tiempo de resolución, prioridad (MX), alias (CNAME).

Nodo

Cada nodo de lista enlazada contiene una `EntradaDNS` y un puntero al siguiente nodo. Se utiliza para resolver colisiones dentro de cada bucket de la tabla hash.

TablaHash

Arreglo de punteros de tamaño fijo (50–100 buckets). Cada posición apunta al inicio de una lista de nodos. El índice se calcula mediante una función hash aplicada al nombre del dominio.

4. Explicación de los algoritmos implementados

Función hash

Convierte el dominio (string) en un índice de la tabla:

$$\text{hash}(\textit{dominio}) = (h \times 33 + c) \text{ mód TAM\_TABLA}$$

donde h se inicializa en 5381 (algoritmo DJB2). Esto permite una distribución uniforme de claves y minimiza colisiones.

Inserción

1. Calcular índice con la función hash. 2. Si el bucket está vacío, crear el primer nodo. 3. Si ya existe una lista, recorrerla:

- Si el dominio ya existe, actualizar su información.
- Si no, insertar el nuevo nodo al inicio (O(1)).

Búsqueda

1. Calcular índice hash del dominio. 2. Recorrer la lista correspondiente hasta encontrar coincidencia. 3. Si se encuentra, retornar la entrada e incrementar el contador de *hits*. Complejidad: $O(1)$ promedio, $O(n)$ en el peor caso.

Eliminación

1. Calcular índice hash. 2. Buscar nodo y ajustar punteros según sea cabeza, medio o final. 3. Liberar memoria con `free()`. Complejidad: $O(1)$ promedio.

Limpieza por TTL

Recorre toda la tabla, compara el tiempo actual con el almacenado y elimina entradas expiradas.

Estadísticas

Calcula:

- Número total de entradas.
- Buckets vacíos y con colisiones.
- Longitud máxima de lista.
- Factor de carga = $\frac{\text{entradas}}{\text{buckets}}$.

5. Ejemplos de uso del programa

A continuación se muestra una sesión típica:

Listing 1: Ejecución del programa

```
=== Sistema de Cache DNS ===
1. Cachear nueva entrada
2. Buscar dominio
3. Actualizar entrada
4. Eliminar entrada
5. Limpiar expirados
6. Mostrar todos los dominios
7. Mostrar estadisticas
8. Salir
> 1
Dominio: www.example.com
IP: 93.184.216.34
TTL: 3600
Servidor: 8.8.8.8
Entrada cacheada correctamente!

> 2
Dominio a buscar: www.example.com
IP encontrada: 93.184.216.34
Hits: 1

> 7
Entradas totales: 12
Buckets con colision: 3
Factor de carga: 0.24
```

6. Gestión de memoria y modularización

Cada nodo se reserva dinámicamente y se libera al eliminar una entrada o al finalizar el programa. El código se organiza en módulos:

- `hash.c/h` — gestión de la tabla hash y funciones principales.
- `dns.c/h` — estructuras, carga y visualización de datos DNS.
- `utiles.c/h` — manejo de tiempo y funciones auxiliares.
- `main.c` — menú principal e interacción con el usuario.

7. Código fuente (espacios para insertar)

main.c

Listing 2: main.c (placeholder)

```
// TODO: implementar men  principal
```

hash.c

Listing 3: hash.c (placeholder)

```
// TODO: implementar funciones hash(), insertar(), buscar(), eliminar()
```

dns.c

Listing 4: dns.c (placeholder)

```
// TODO: implementar funciones de creaci n, carga y visualizaci n de
    entradas
```

8. Conclusiones

- La tabla hash permite accesos promedio O(1), optimizando el rendimiento del sistema DNS.
- El manejo dinámico de memoria y la liberación correcta evitan fugas.
- Se cumplieron los requisitos de modularización, validación y documentación.