MIDDLE EAST TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

CENG 140
Spring 2022
THE 3

REGULATIONS

- **Deadline:** 18th June, 23:55 (Not subject to postponement).
- **Submission:** You will use OdtuClass for the homework just like Lab Exams. You can use the system as the editor or work locally and upload the source files. Late submission **is not** allowed.
- **Team:** There is **no** teaming up. The homework has to be done/turned in individually.
- **Cheating:** All parts involved (source(s) and receiver(s)) get zero and face disciplinary action. ChatGPT and solutions from the WEB (if any) will be considered cheating.

1 INTRODUCTION

For this homework, imagine that you are an entrepreneur and you your friends decided to found a company that serves media files to clients through the web via browser or standalone applications just like **Youtube**, **Spotify** or **Netflix**. You manage **huge** databases in servers all around the world which hold the data and metadata of **videos** or **music files**. When users enter the **URL** on the web browser or click on the media files in your application, these servers provide these media files to the user.

You've made great initial success and you have millions of clients now. However, this creates a problem. Servers now get overloaded with requests just like **METU registration system every single year**. One solution is to increase the number of database servers but, this costs a lot of money and it is not suitable.

Instead, you decided to open up **Proxy Servers** that hold a **Cache** with **LRU(Least Recently Used)** logic where only the most requested files are stored. The cache will have a lot less size than actual databases. So, this time, when a user asks for a file:

- The system first asks the **Proxy Server** if that file is already in the cache.
- If the file is in the cache, it is directly given to the user without bothering actual database servers.
- If the file is not in the cache, **Proxy Server** gets that file from the actual database server and put that media to the beginning of the cache.
- If the cache's size limit is exceeded with the added file, the last elements of the cache are popped until there is enough space for the new media.

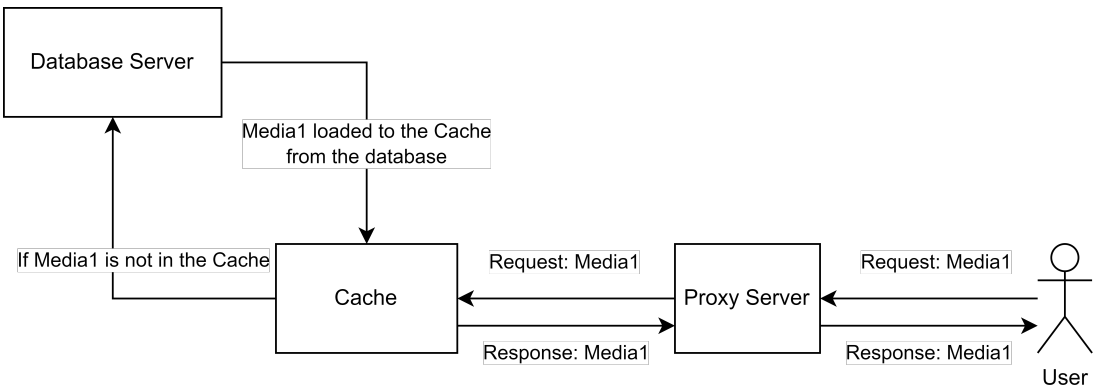

Figure 1: System logic

2 STRUCTURES

In this homework, you will implement the **Cache** with **LRU(Least Recently Used)** logic. You will use **doubly connected linked list** data structure for the cache. The structs you need to use are already given to you in **the3.h** file.

2.1 Cache

Members of the cache are:

- **CacheNode* head:** Pointer to the head of the cache.
- **CacheNode* tail:** Pointer to the tail of the cache.

- **int mediaCount:** This integer value keeps track of the number of media files in the cache.
- **int currentSize:** This is the size of all media that are currently registered in the cache.
- **int cacheLimit:** This is the maximum limit of the cache's size in kilobytes(KB).

2.2 CacheNode

The nodes of the cache are in this structure:

- **CacheNode* prev:** Pointer previous node in the cache.
- **CacheNode* next:** Pointer to the next node in the cache.
- **DomainFreqList* domainFreqList:** Every cache node holds another **doubly connected linked list** that keeps the list of domains and their frequencies. It will be explained in further sections.
- **Media media:** This is a struct for media information the node holds. It will be explained in the next subsection.

2.3 Media

This struct has the media name as a char pointer and its size in kilobytes(KB). The struct looks like the following:

- **char* name**
- **int size**

2.4 DomainFreqList

As explained above, every node in the cache holds another **doubly connected linked list** for keeping track of which domains users requested these media from and their frequencies. Its struct is:

- **DomainFreqNode* head:** Pointer to the head of the DomainFreqList
- **DomainFreqNode* tail:** Pointer to the tail of the DomainFreqList

In this list, the nodes will be held sorted according to these conditions:

- Nodes will be sorted according to their frequencies
- If their frequencies are equal, to break the tie, Nodes will be sorted according to their **Domain Names** alphabetically ascending order.

For example, the figure below shows the structure of a Domain Frequency List and you can see how these nodes are sorted;

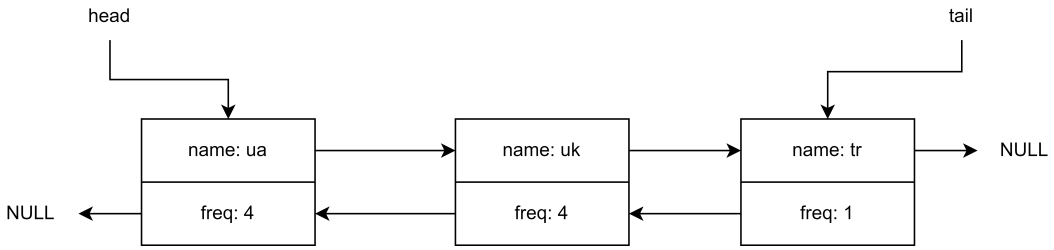

Figure 2: Domain Frequency List Example

2.5 DomainFreqNode

The structure of **DomainFreqNode** looks like the following:

- **char* name:** The name of the domain.
- **int freq:** Frequency of that domain.
- **DomainFreqNode* prev:** Pointer to the previous node in the DomainFreqList.
- **DomainFreqNode* next:** Pointer to the next node in the DomainFreqList.

3 TASKS

3.1 CreateCache (30 pts)

You first have to implement **createCache()** function that returns a pointer to the **Cache** struct. This function reads the following from an input file from the standard input. The input file format is like the following:

<CacheSize> <n>

n { <mediaName> <mediaSize> <domain1> <domain1_freq> <domain2> <domain2_freq> .....

<mediaName> <mediaSize> <domain1> <domain1_freq> .....

.....

Figure 3: Example of an Input

For example, let's think of this input:

```
1000 3
video1 200 tr 1 us 4 ua 4 uk 4
video2 300 tr 1 us 5
video3 200 en 1 sa 4
```

According to this input:

- Size of the cache is 1000 KB
- There are three media for the cache with sizes (200, 300 and 200 KB)
- Each node has a different number of domain and frequency pairs. Note that domain names are not sorted in the input. You have to sort them while reading from stdin.

The cache looks like the following visually:

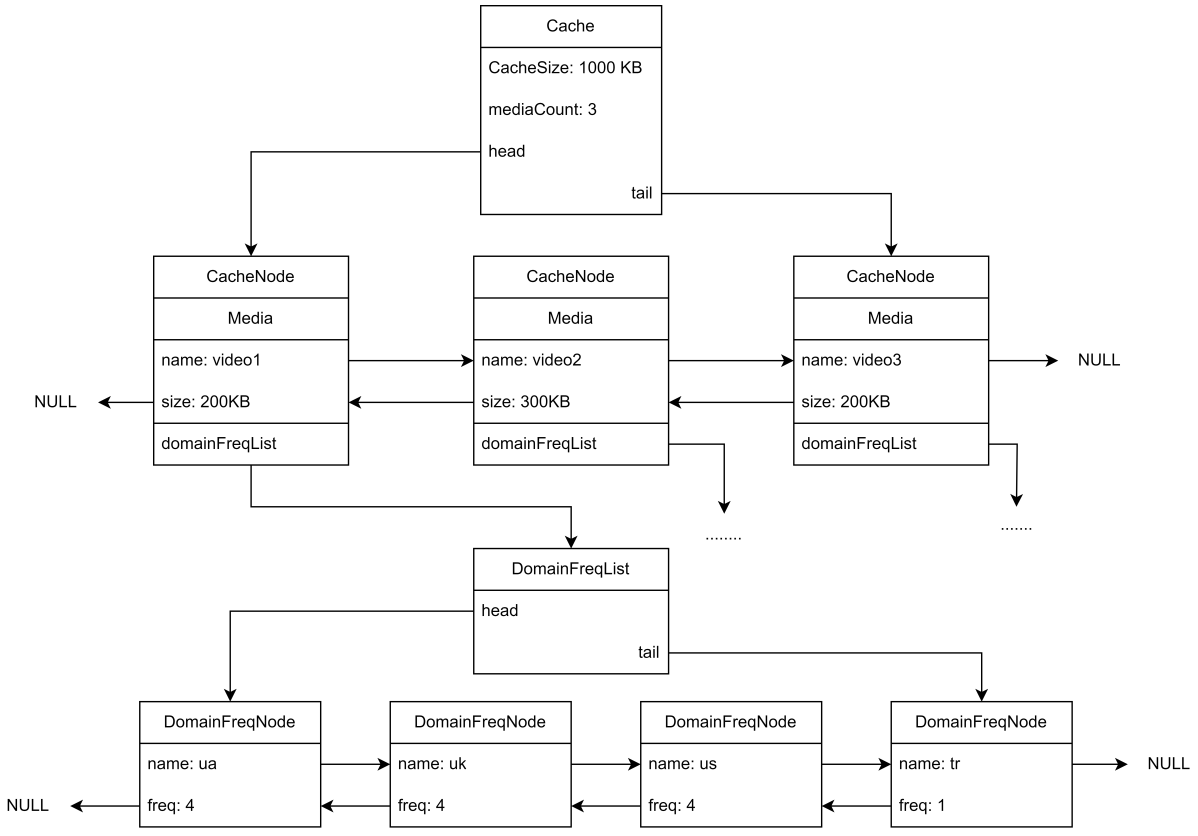

Figure 4: Cache Example

In the figure above, you can only see the DomainFreqList of the first node. Other DomainFreqLists also follow the same convention.

3.1.1 Specifications

- The total size of the media in the input file will always be less than the maximum cache size.
- The function returns a pointer to the cache.
- For reading input from the file using stdin, you can use **scanf** for the first line to acquire cache size and media count, and for the next lines, you can use **fgets** to read the whole line in a string and then, you can tokenize domain and frequency pairs with **strtok**
- Do not forget that you have to dynamically allocate memory for domain and media names, domainFrequency and cache nodes and also for the cache itself.

3.2 PrintCache (10 pts)

You will implement the function:

```
void printCache(Cache* cache);
```

This function prints the contents of the cache since we also want to see the contents for inspections and to be more verbose. For the example cache in 3.1 the function prints the following:

```
-------------- Cache Information --------------
Cache Limit: 1000 KB
Current Size: 700 KB
Media Count: 3
Cache Media Info:
        Name: video1    Size: 200 KB
        ------- Media Domain Frequencies -------
                Name: ua        Freq: 4
                Name: uk        Freq: 4
                Name: us        Freq: 4
                Name: tr        Freq: 1
        ----------------------------------------
        Name: video2    Size: 300 KB
        ------- Media Domain Frequencies -------
                Name: us        Freq: 5
                Name: tr        Freq: 1
        ----------------------------------------
        Name: video3    Size: 200 KB
        ------- Media Domain Frequencies -------
                Name: sa        Freq: 4
                Name: en        Freq: 1
        ----------------------------------------
-----------------------------------------------
```

If the cache does not have any nodes (this means the cache is empty) you must print:

```
The Cache is Empty
```

3.3 Specifications

In the printing format:

- Every indentation is a **tab**.
- There is a tab between each **Name** and **Size** pairs of cache node medias.
- There is a tab between each **Name** and **Freq** pairs of domain freq nodes.

3.4 AddMediaRequest (30 pts)

All the modifications under this section are separately applied to the initial cache state that is shown in **print cache**3.2 section. You will implement the function:

```
CacheNode* addMediaRequest(Cache* cache, Media media, char* domain);
```

This function should first check that given media exists in the cache:

- If it exists, it should put the corresponding node to the head of the list and also update the domain frequency list. So let's consider we have the cache from 3.1 part. We want to add the following media with domain :

  ```
  media1.name = "video2";
  media1.size = 300;
  addMediaRequest(cache, media1, "tr");
  ```

  The cache becomes:

```
-------------- Cache Information --------------
Cache Limit: 1000 KB
Current Size: 700 KB
Media Count: 3
Cache Media Info:
        Name: video2    Size: 300 KB
        ------- Media Domain Frequencies -------
                Name: us        Freq: 5
                Name: tr        Freq: 2
        ----------------------------------------
        Name: video1    Size: 200 KB
        ------- Media Domain Frequencies -------
                Name: ua        Freq: 4
                Name: uk        Freq: 4
                Name: us        Freq: 4
                Name: tr        Freq: 1
        ----------------------------------------
        Name: video3    Size: 200 KB
        ------- Media Domain Frequencies -------
                Name: sa        Freq: 4
                Name: en        Freq: 1
        ----------------------------------------
-----------------------------------------------
```

You see that **video2** comes to the head and in domain frequency, we see that frequency of **tr** is increased by 1. If we would add a different domain name that did not exist in the domain frequency list it would be added to the list. For example if we would add this media request:

```
media1.name = "video2";
media1.size = 300;
addMediaRequest(cache, media1, "sa");
```

The cache would become:

```
-------------- Cache Information --------------
Cache Limit: 1000 KB
Current Size: 700 KB
Media Count: 3
Cache Media Info:
        Name: video2    Size: 300 KB
        ------- Media Domain Frequencies -------
                Name: us        Freq: 5
                Name: sa        Freq: 1
                Name: tr        Freq: 1
        ----------------------------------------
        Name: video1    Size: 200 KB
        ------- Media Domain Frequencies -------
                Name: ua        Freq: 4
                Name: uk        Freq: 4
                Name: us        Freq: 4
                Name: tr        Freq: 1
        ----------------------------------------
        Name: video3    Size: 200 KB
        ------- Media Domain Frequencies -------
                Name: sa        Freq: 4
                Name: en        Freq: 1
        ----------------------------------------
-----------------------------------------------
```

Notice that **sa** domain becomes before than the **tr** becomes they have the same frequency and **sa** is alphabetically before than **tr**. So, you have to keep domain frequency lists sorted all the time.

- If the media **does not exist** in the cache, you have to add a new cache node (for this media) by allocating memory and checking the current total media size in the cache and new node. If it is not larger than the maximum cache size, you just add the node to the **head** of the cache. For example, let's add a new media like this:

```
media1.name = "newVideo";
media1.size = 300;
addMediaRequest(cache, media1, "tr");
```

We get this cache:

```
-------------- Cache Information --------------
Cache Limit: 1000 KB
Current Size: 1000 KB
Media Count: 4
Cache Media Info:
        Name: newVideo  Size: 300 KB
        ------- Media Domain Frequencies -------
                Name: tr        Freq: 1
        ----------------------------------------
        Name: video1    Size: 200 KB
        ------- Media Domain Frequencies -------
                Name: ua        Freq: 4
                Name: uk        Freq: 4
                Name: us        Freq: 4
                Name: tr        Freq: 1
        ----------------------------------------
        Name: video2    Size: 300 KB
        ------- Media Domain Frequencies -------
                Name: us        Freq: 5
                Name: tr        Freq: 1
        ----------------------------------------
        Name: video3    Size: 200 KB
        ------- Media Domain Frequencies -------
                Name: sa        Freq: 4
                Name: en        Freq: 1
        ----------------------------------------
-----------------------------------------------
```

Notice that **newVideo** has been added to the cache head because with its size added, we do not exceed the maximum cache limit. Notice how **Media Count** changes to 4 and **tr** domain is added to the new node with frequency 1. However, what happens if with the new node, we exceed the cache limit? Lets think about this example. We add the media:

```
media1.name = "newVideo";
media1.size = 500;
addMediaRequest(cache, media1, "tr");
```

We ge the cache:

```
-------------- Cache Information --------------
Cache Limit: 1000 KB
Current Size: 1000 KB
Media Count: 3
Cache Media Info:
        Name: newVideo  Size: 500 KB
        ------- Media Domain Frequencies -------
                Name: tr        Freq: 1
        ----------------------------------------
        Name: video1    Size: 200 KB
        ------- Media Domain Frequencies -------
                Name: ua        Freq: 4
                Name: uk        Freq: 4
                Name: us        Freq: 4
                Name: tr        Freq: 1
        ----------------------------------------
        Name: video2    Size: 300 KB
        ------- Media Domain Frequencies -------
                Name: us        Freq: 5
                Name: tr        Freq: 1
        ----------------------------------------
-----------------------------------------------
```

We see that **video3** is deleted. So we start deleting nodes from the tail until there is enough space for the **newVideo**, which is again added to the **head** of the cache. Now, let's see what happens if the **new video** has had an even larger size, say **900 KB**. In this case, again starting with the cache in 4:

```
media1.name = "newVideo";
media1.size = 900;
addMediaRequest(cache, media1, "tr");
```

The cache becomes:

```
-------------- Cache Information --------------
Cache Limit: 1000 KB
Current Size: 900 KB
Media Count: 1
Cache Media Info:
        Name: newVideo  Size: 900 KB
        ------- Media Domain Frequencies -------
                Name: tr        Freq: 1
        ----------------------------------------
-----------------------------------------------
```

We see that we have deleted all **video3**, **video2**, and **video1** until we had enough space. And then, we added **newVideo**.

3.5 Notes

- Do not forget to **free** what you have allocated while deleting the nodes.
- Do not forget to return the pointer to the node you have created or updated.

3.6 FindMedia (15 pts)

You will implement the function:

```
CacheNode* findMedia(Cache* cache, char* name);
```

This function searches the cache with the given media name and if it finds the node just returns it. If it does not find the node, it returns **NULL**

3.6.1 Specifications

- You will not modify anything in the cache, just return the node if it is found, if is does not exist, return NULL.

3.7 DeleteMedia (15pts)

You will implement the function:

```
void deleteMedia(Cache* cache, char* name);
```

In this function you have find the media with its name, if it exists in the cache, delete it. If it does not exist, do nothing. For example, lets consider the cache in 3.1. If we delete **video2**:

```
deleteMedia(cache, "video2")
```

The cache becomes:

```
-------------- Cache Information --------------
Cache Limit: 1000 KB
Current Size: 400 KB
Media Count: 2
Cache Media Info:
        Name: video1    Size: 200 KB
        ------- Media Domain Frequencies -------
                Name: ua        Freq: 4
                Name: uk        Freq: 4
                Name: us        Freq: 4
                Name: tr        Freq: 1
        ----------------------------------------
        Name: video3    Size: 200 KB
        ------- Media Domain Frequencies -------
                Name: sa        Freq: 4
                Name: en        Freq: 1
        ----------------------------------------
-----------------------------------------------
```

Notice that **video2** has been deleted from the cache, **Current Size** and **Media Count** is also decremented after the deletion.

3.7.1 Notes

- Do not forget to free the memory for deleted items if you have allocated memory for it in the first place.
- To test your code, we have provided you a **main.c** file so you can use the functions that you have implemented. **main.c** looks like the following:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "the3.h"


int main(int argc, char** argv)
{
    Cache* cache;
    Media media1;

    /* Create the cache */
    cache = createCache();

    /* Add new media */
    media1.name = "newVideo";
    media1.size = 300;

    addMediaRequest(cache, media1, "tr");

    deleteMedia(cache, "video2");

    printCache(cache);

    return 0;
}
```

- If you are working on your local machine, you **must** compile your code like the following:

```
gcc the3.c main.c -g -Wall -ansi -pedantic-errors -o main
```

- While testing your codes, you always have to use an input file and read it via stdin. You can assume that you always have to use **createCache()** function. As explained above in detail, input files will be like this:

```
1000 3
video1 200 tr 1 us 4 ua 4 uk 4
video2 300 tr 1 us 5
video3 200 en 1 sa 4
```

  And you can run your executables by:

```
./main < inp.txt
```

- You **must not** change the contents of **the3.h**. We will test your implementations of the functions declared there.
- In **the3.c**, you can use helper functions if you want.

4 Grading

- Make sure that your code compiles and works. We will not do partial grading for this homework for non-working code.

Good luck,