INFO198 - SISTEMAS OPERATIVOS
Prueba 2 - parte práctica

2023-02 (Dr. Luis Veas C.)

La prueba 2 consta de 2 partes, una teórica y una práctica.

- La parte teórica se realizará el día viernes 02/11/2023 en horario de clases
- La parte práctica se entrega a los alumnos el día lunes 30/10/2023, y se debe entregar al profesor a más tardar:
    - Jueves 09/11/2023 a las 23:59 horas, para optar al 7
    - Domingo 12/11/2023 a las 23:59 horas, para optar al 6
- Esta sección debe realizarse con dos personas.
- La forma de entrega es vía email, bajo el siguiente formato:

    asunto: INFO198 - SISTEMAS OPERATIVOS - Prueba 2 - práctica
    cuerpo: nombre de los integrantes
    adjunto: archivo zip comprimido con el siguiente formato
    ⇒ si es un integrante nombre-apellido1.zip
    ⇒ si son dos integrantes nombre-apellido1_nombre-apellido2.zip

La ponderación de las pruebas es la siguiente:

- parte teórica 58% del total de la calificación, **para revisar la parte práctica debe tener por lo menos una calificación 2.5 de 7 en la parte teórica**
- parte práctica 42% del total de la calificación

Se deberá agendar hora con el profesor fuera del horario de clases para revisar en conjunto la parte práctica, si son dos los integrantes deberán asistir los dos, ya que se les harán preguntas y retroalimentación basado en lo que hicieron

Problema

Crear 3 programas en c++ los cuales:

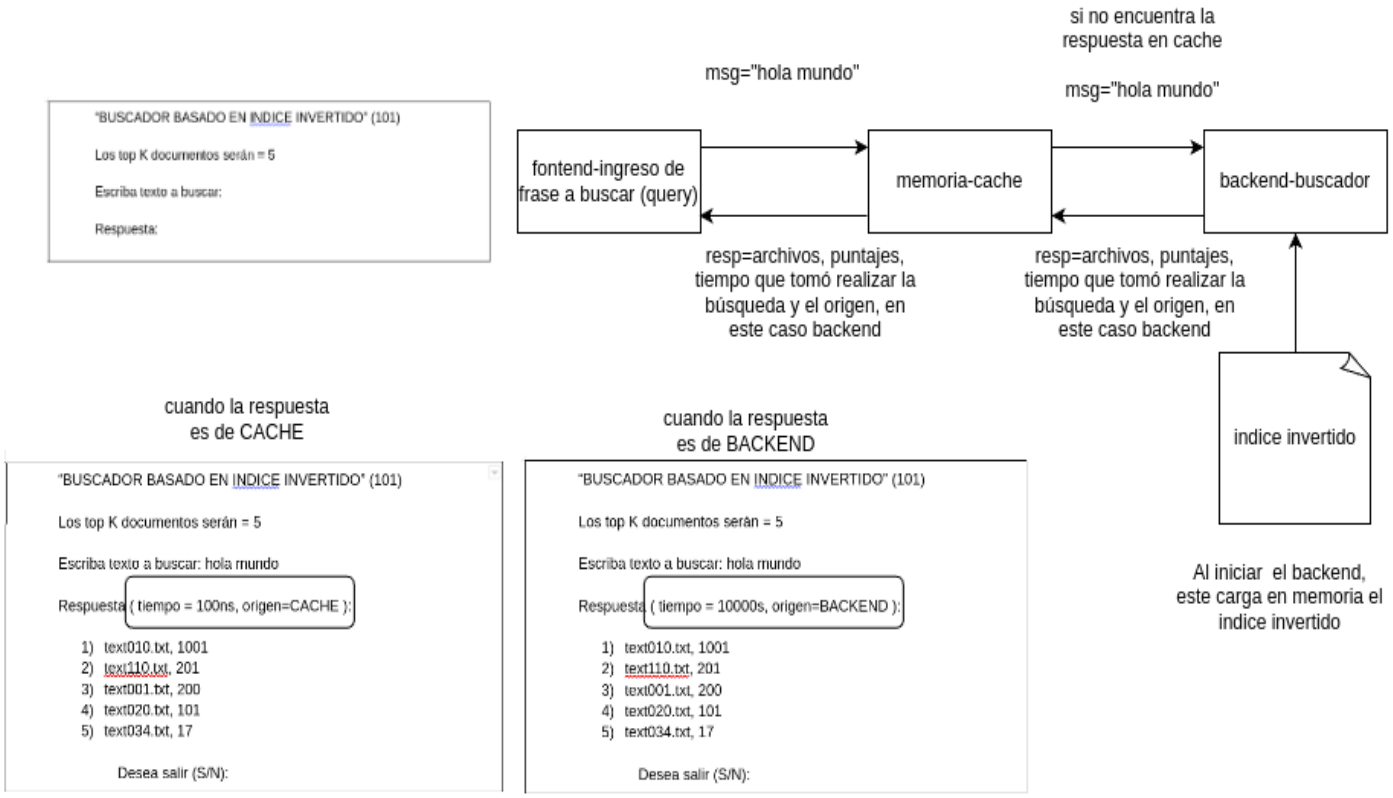
1) Para el correcto funcionamiento de estos programas debe tener un índice invertido creado con 20 archivos de mínimo 1MB cada 1.

El índice invertido es el archivo con el formato.

hola:(file001.txt;3);(file004.txt;13);(file006.txt;31);(file007.txt;30)
casa:(file001.txt;4);(file014.txt;133);(file006.txt;131);(file017.txt;80);(file030.txt;10)
perro:(file001.txt;35);(file002.txt;3)
pc:(file005.txt;3)
celular:(file0014.txt;300);(file0015.txt;3);(file0016.txt;1);
.
.

2) Los 3 programas deben hacer uso de variables de entorno. Los programas conformarán 3 capas, frontend, cache, backend y se deben comunicar mediante mensajes. Ustedes deben definir el canal de comunicación, el cual puede ser: Socket o FIFO

Diagrama de funcionamiento del sistema completo



3) Los nombres de los programas deben ser:
    a) frontend => searcher
    b) cache=>memcache
    c) backend=>invertedindex

4) El frontend es una aplicación de consola que permite ejecutar consultas sobre el índice invertido (interacción humano-computador), debe enviar el mensaje al programa cache. También, debe ser capaz de recibir la respuesta desde el mismo cache, el mensaje debe indicar:

mensaje={origen:"XXXX",destino:"XXXX",contexto:{topk:"XXXX", txtToSerarch:"XXXX"}}

Ejemplo:

mensaje={
origen:"./searcher",
destino:"./memcache",
contexto:{ txtToSerarch:"hola mundo"}
}

Nota: ustedes definen el protocolo de comunicación, es decir, el formato del mensaje, pero por lo menos el mensaje debe contener la información señalada anteriormente (origen, destino, contexto). En el caso de los ejemplos que doy en este documento, el protocolo de comunicación está definido con mensajes tipo json, pero ustedes pueden definir el formato que mejor les acomode, cuando me muestren los códigos, me explicarán el porqué de la decisión.

5) Las mínimas variables de entorno que debe presentar el frontend

    a) FROM=./searcher
    b) TO=./memcache

6) El Cache es la aplicación encargada de mantener en memoria las últimas X respuestas de búsqueda, donde X forma parte de las variables de entorno. Estas son las mínimas variables de entorno a manejar.

    a) HOST=./memcache
    b) FRONT=./searcher
    c) BACK=./invertedindex
    d) MEMORYSIZE=4 (esta variable representa la X)

7) El cache debe desempaquetar el mensaje

8) El cache debe realizar la búsqueda sobre su memoria

9) Si el texto buscado se encuentra en la cache, entonces el cache envía al frontend la respuesta con el resultado de la búsqueda

mensaje={origen:”XXXX”,destino:”XXXX”,contexto:{tiempo:”XXXX”, ori=”CACHE”, isFound=true, resultados:[{archivo:”XXXX”, puntaje:”XXXX”}] }}

Ejemplo:

```
mensaje={
        origen:”./memcache”,
        destino:”./searcher”,
        contexto: {
                tiempo:”100ns”, ori=”CACHE”, isFound=true,
                resultados:[
                        {archivo:”file-text1.txt”, puntaje:”122”}
                        {archivo:”file-text20.txt”, puntaje:”34”}
                        {archivo:”file-text3.txt”, puntaje:”10”}
                        {archivo:”file-text2.tx”, puntaje:”2”}
                        ]
                }
        }
```

10) Si cache no encuentra la respuesta en su memoria, debe enviar el mensaje con la consulta la backend

11) El backend es el programa encargado de leer y mantener en memoria el índice invertido

12) El backend realiza las búsquedas cuando no se encuentra el texto buscado en el cache. Debe mantener como mínimo las siguientes variables de entorno.

    a) FROM=./searcher
    b) TO=./memcache
    c) FILE=inverted_index_file.idx
    d) TOPK=4

13) El backend debe desempaquetar el mensaje de consulta

14) El backend debe realizar la búsqueda sobre el índice invertido cargado en memoria

15) El backend debe enviar la respuesta al cache, la respuesta debe contener los TOPK resultados de la búsqueda

Ejemplo: cuando encuentra

```
mensaje={
        origen:”./invertedindex”,
        destino:”./memcache”,
        contexto: {
                tiempo:”1000ns”, ori=”backend”, isFound=true,
                resultados:[
                        {archivo:”file-text1.txt”, puntaje:”122”}
                        {archivo:”file-text20.txt”, puntaje:”34”}
                        {archivo:”file-text3.txt”, puntaje:”10”}
                        {archivo:”file-text2.txt”, puntaje:”2”}
                        ]
                }
        }
```

Ejemplo: cuando no encuentra

```
mensaje={
        origen:”./invertedindex”,
        destino:”./memcache”,
        contexto: {
                tiempo:”1000ns”, ori=”backend”, isFound=false,
                resultados:[ ]
                }
        }
```

16) El cache debe interpretar el mensaje que proviene del backend

17) Si el mensaje indica que se encontró respuesta al texto buscado, debe agregar la respuesta a la memoria de cache, teniendo presente el CACHESIZE, es decir, si la memoria está completa, debe reemplazar alguno de los textos de la memoria cache (memory swapping)

18) El cache debe enviar la respuesta de la búsqueda al Frontend (haya encontrado o no la respuesta)

19) El frontend debe interpretar la respuesta

20) El frontend debe mostrar el resultado, según el ejemplo entregado en el diagrama de más arriba.