Operating Systems

Practical Assignment

Document Indexing and Searching Service

Distributed Systems Group
University of Minho

March 9, 2025

General information

- Each group must have 3 members;
- The work must be handed in by 23:59 on May 17th;
- The source code, scripts and a report of up to 10 pages of content must be submitted (A4, 11pt) in PDF format (covers and annexes are not counted towards the 10-page limit), justifying the solution, particularly with regard to process architecture, functionalities, and the specific choice and use of communication mechanisms;
- The work must be carried out using the Linux operating system as the development and execution environment;
- The work must be submitted in a Zip file with the name `grupo-xx.zip`, where `xx` should be replaced by the number of the working group (*e.g.*, `grupo-01.zip`);
- The presentation of the work will take place on a date to be announced, probably between June 2 and 5;
- The work represents 50% of the final classification final grade.
- It should be noted that the maximum mark for the different stages of the practical assignement depends on both the correct implementation of the stages and the quality of each group's report and discussion with the teaching team.
- The teaching team will be updating a TP FAQ with answers to questions raised by students on: `https://docs.google.com/document/d/1PGrnMWpFf2JQOeKSqVJBsoZ9VUFtKzFZJBzEwSOEXkA/edit?usp=sharing`.

Summary

The aim is to implement a service that allows indexing and searching text documents stored locally on a computer. The server program is responsible for recording meta-information about each document (e.g., unique identifier, title, year, author, path), as well as allowing a set of queries regarding this meta-information and the content of the documents.

Users must use a client program to interact with the service (*i.e.*, with the server program). This interaction will allow users to add or remove the indexing of a document in the service, and to perform searches (queries) on the indexed documents. Note that the client program only performs one operation per invocation, i.e. it is not an interactive program (*i.e.*, which reads several operations from *stdin*).

Server and Client (12 values)

A client should be developed (program *dclient*) to be used by the user via the command line. A server should also be developed (program *dserver*), with which the client program will interact. The server must keep relevant information in memory and on disk to support the functionalities described in this assignment.

The *standard output* should be used by the client program to display the necessary responses to the user, and by the server program only to display information for debugging (*debug*) as it deems necessary. You can use the *printf* function for this type of operation.

The client and server programs should be written in C and communicate via *FIFOs*. The system calls taught in the operating systems course should be used for process management, communication between processes, and interaction with files. You should not execute programs directly or indirectly through the command interpreter (*p.ex.*, *sh*, *bash* or `system()`). Likewise, you cannot use functions such as *fopen*, *fwrite*, *fread*, etc. The use of libraries (*e.g.*, Glib) to create and manage data structures is permitted.

The service should support the following basic functionalities:

Indexing, querying, and removing meta-information from documents. To index a new document, the user invokes the client program with the command:

```
dclient -a "title" "authors" "year" "path"
```

- `title`: title of the document.
- `authors`: author(s) of the document (*e.g.*, separated by a semicolon (';') when there are several authors).
- `year`: year of the document.
- `path`: relative path of the document, *i.e.,* from the base directory configured for the service.

Note that the document must have been previously created by the user; the server program will only index it. The client program must communicate the indexing request to the server program, whose response must contain a unique identifier for the document, which must be communicated to the user. The choice of identifier is up to each group. The server program must index the meta-information of each document, using the data structure(s) that each group deems most appropriate.

Notes: Assume that the total size of the arguments to the previous operation does not exceed 512 bytes (*e.g.*, the `title` and `authors` fields have a maximum of 200 bytes each, the `path` field has a maximum of 64 bytes and the `year` field occupies a maximum of 4 bytes).

In addition, the client program must be able to query (option *-c*) and remove (option *-d*) a document's meta-information via its identifier *key*.

```
dclient -c "key"
dclient -d "key"
```

The client program must notify the user of the success of the operations and, for the query operation, it must print the document's meta-information on the *stdout*. The remove operation should not delete the content of the document in question, it should only remove its meta-information indexed by the server program.

Document content search. In addition to managing and consulting meta-information, the client program must also allow users to search the content of indexed documents.

In detail, it should be possible (option *-l*) to return the number of lines of a given document (*i.e.*, identified by its *key*) that contain a given keyword (*keyword*).

```
dclient -l "key" "keyword"
```

It should also be possible (option *-s*) to return a list of document identifiers that contain a given keyword (*keyword*).

```
dclient -s "keyword"
```

Note: To implement these operations you can use the programs *grep* and *wc* that were discussed in the practical classes.

General notes: The computation to respond to all the above operations must be carried out by the server. The client simply sends the request and waits for the response, presenting it to the user. In addition, your implementation must prevent a client from being blocked due to **query** and **search** operations (options *-c, -l and -s*) being carried out by other clients.

Optimizations and Evaluation (8 values)

Starting from your basic implementation, you must optimize the server program to incorporate the following functionalities.

Concurrent search. The search operation for documents containing a given keyword (option *-s*) must be able to be carried out concurrently by several processes. By supporting this advanced operation, it now receives an extra argument, namely the maximum number of processes to run simultaneously.

```
dclient -s "keyword" "nr_processes"
```

Note: This functionality must be implemented by the group without using the capacity of external programs to parallelize execution. Note that the group can still use the `grep` program to search each file.

Persistence. Assume, for reasons of efficient memory management and information durability, that the meta-information of the documents managed by the server program must be persisted on disk. Therefore, ensure that your server program stores a persistent copy of the documents' meta-information (*i.e.*, which can be retrieved by stopping and restarting the program). The server is stopped via a special client command (option *-f*).

```
dclient -f
```

Caching. Change the server program so that you can control the number of meta-information entries on disk that are also stored in memory. To do this, implement an in-memory *cache* that stores up to *N* meta-information entries (argument set at server program startup). The policies for choosing which *items* are kept and served by the *cache* are up to each group.

Experimental evaluation. Develop and run tests to evaluate the performance gain of parallelizing document search (option *-s*) and the impact on performance of different configurations (*i.e.*, sizes) and *caching* policies developed by the group. Use scripts to automate test execution. You should also perform tests (*i.e.*, different configurations, number of documents, etc.) that allow understanding the impact on performance in different scenarios. The report should include a reflection on the results of the tests carried out.

Note: Along with this assignment, the teaching team has provided a set of documents for testing, and a *script* to automate the indexing of the meta-information in these documents that you can use with your program. Each group can test with other sets of documents that they find relevant.

Interface and Mode of Use

The service should be used as follows:

- Run the server:

  ```
  $ ./dserver document_folder cache_size
  ```

  Arguments:

  1. `document_folder`: folder where the documents to be indexed are located.
  2. `cache_size`: number of items (*i.e.*, meta-information from different documents) to be stored by the *cache* in memory.

- Submit a request to index a document, as in the example below.

  ```
  $ ./dclient -a "Romeo and Juliet" "William Shakespeare" "1997" "1112.txt"
  Document 1 indexed
  ```

- Submit a request to query the meta-information of a document.

  ```
  $ ./dclient -c 1
  Title: Romeo and Juliet
  Authors: William Shakespeare
  Year: 1997
  Path: 1112.txt
  ```

- Submit a request to remove an index.

  ```
  $ ./dclient -d 1
  Index entry 1 deleted
  ```

- Search number of lines containing a certain keyword.

  ```
  $ ./dclient -l 1 "Romeo"
  150
  ```

- Search for a list of document identifiers containing a certain keyword.

  ```
  $ ./dclient -s "praia"
  [2, 3, 1438]
  ```

- Search for a list of document identifiers containing a certain keyword using multiple processes (*e.g.*, 5).

  ```
  $ ./dclient -s "praia" 5
  [2, 3, 1438]
  ```

- Stop the server program via the client program.

  ```
  $ ./dclient -f
  Server is shuting down
  ```

Makefile

Please note that the Makefile presented here should be used as a starting point. You may need to adapt it in order to satisfy other dependencies in your source code. In any case, you should always keep the goals (*targets*) specified: `all`, `dserver`, `dclient`, and `clean`. Do not forget that, by convention, the indentation of a Makefile is specified with a tab at the beginning of the line (never with blanks).

```
CC = gcc
CFLAGS = -Wall -g -Iinclude
LDFLAGS =

all: folders dserver dclient

dserver: bin/dserver

dclient: bin/dclient

folders:
	@mkdir -p src include obj bin tmp

bin/dserver: obj/dserver.o
	$(CC) $(LDFLAGS) $^ -o $@

bin/dclient: obj/dclient.o
	$(CC) $(LDFLAGS) $^ -o $@

obj/%.o: src/%.c
	$(CC) $(CFLAGS) -c $< -o $@

clean:
	rm -f obj/* tmp/* bin/*
```