CS69201: Computing Lab - 1

Project: memFS - A Fast, In-Memory File System

Deadline: 11:59 PM, 17th November, 2024 **Maximum Marks : 100**

General Instructions:

1. You need to do this assignment in **C/C++** language
2. The **deadline** of the assignment is set as **11:59 PM, 17/11/2024**
3. Implementation **modularity**, and **code quality** will be considered in evaluation

---

memFS (pronounced as *memphis*) is an in-memory file system designed to provide high-speed data access and storage by leveraging volatile memory(RAM). Unlike traditional disk-based file systems, memFS enables rapid read and write operations, making it ideal for applications that require quick data processing, such as caching and testing environments. While it offers significant performance advantages, its data is temporary and disappears upon closing the application, limiting its use to scenarios where persistence is not critical. This simplicity and speed make memFS a valuable tool for developers looking to optimize file handling without the complexities of disk I/O.

Building on its core design, memFS operates directly from the command line. By executing `./memFS` in your shell, you can initiate the in-memory file system, allowing you to input commands and interact with memFS seamlessly within your terminal.

memFS is designed to handle any number of files, constrained only by the available RAM on the system. Each file within memFS is flexible in size, accommodating file sizes from as small as 32 bytes to as large as 2 KB.

In this assignment, you will be designing and implementing memFS in C/C++. Additionally, you would also be benchmarking your implementation against the specified workloads. The details about memFS are as follows:

Part 1 - Implementing memFS

Your implementation of memFS needs to support the following set of commands:

1. `create` command: The create command can create one or more files and store them in the memory. It has the following structure:

   ```
   create [-n (number of files)] [filenames (in order)]
   ```

   Note: In case only one file needs to be created the -n flag is optional. In case any filename conflicts with a previous file, the error should be reported and in such case the remaining files should be created.

   Examples:

- Creating a single file:

```
memfs> create todo.txt
file created successfully
```

- Creating more than one file:

```
memfs> create -n 3 todo1.txt todo2.txt todo3.txt
files created successfully
```

- Creating files with conflicting filenames:

```
memfs> create todo.txt
file created successfully

memfs> create todo.txt
error: another file with same name exists
```

2. `write` command: The write command is used to write data into the files that have been created using the create command

```
write [-n (number of files)] [<filename> "<text to write>"]
```

Note: In case writing to a file which does not exist, the program should report an error and writing to the remaining files should be done.

Examples:

- Writing to a single file:

```
memfs> write todo.txt "Wake up at 7 AM"
successfully written to todo.txt
```

- Writing to multiple files:

```
memfs> write -n 2 todo1.txt "Wake up at 7 AM" todo2.txt "Have Breakfast"
successfully written to the given files
```

- Writing to a non existent file:

```
memfs> write profile.txt "My name is John Smith"
Error: profile.txt does not exist
```

3. `delete` command: The delete command can delete one or more files. It has the following structure:

```
delete [-n (number of files)] [filenames (in any order)]
```

Note: In case any filename doesn't exist, the error should be reported and in such case, the remaining files should be deleted.

Examples:

- Deleting a single file:

```
memfs> delete todo.txt
file deleted successfully
```

- Deleting more than one file:

```
memfs> delete -n 3 todo1.txt todo2.txt todo3.txt
Files deleted successfully
```

- Deleting files that doesn't exist [assuming todo5.txt doesn't exist]

```
memfs> create -n 2 todo5.txt todo6.txt
File todo6.txt doesn't exist. remaining files deleted successfully
```

4. `read` command: The read command can be used to read the contents of a file.

```
read [filename]
```

Note: In case the file does not exists, report the error

Examples:

- Reading from a valid file:

```
memfs> read todo.txt
Wake up at 10 AM
```

- Reading from a non existent file:

```
memfs> read hello.txt
Error: hello.txt does not exist
```

5. `ls` command (file listing): This command will be used to list the files that are present in memFS with their size in bytes, date of creation and date of last modified (based on flags)
Examples:

```
memfs> ls
todo.txt
todo1.txt
profile.txt
test.cpp

memfs> ls -l
size    created     last modified    filename
10     27/10/2024   28/10/2024       todo.txt
15     26/09/2023   15/10/2024       todo1.txt
25     19/01/2022   13/03/2024       profile.txt
100    15/01/2017   15/10/2024       test.cpp
```

6. `exit` command: The exit command is used to quit from the memFS.

   Examples:

```
memfs> exit
exiting memFS
```

Note: Since there are commands which supports manipulation of multiple files, you are required to add support of multi-threading to such commands (`create`, `write` and `delete`). Also, try to design memFS in such a way that the average latency of each command is in order of few milliseconds(ms).

Part 2 - Benchmarking memFS

In this part you would be benchmarking your implementation against some specified workloads. The workloads are as follows:

1. 100 creates , writes, reads and deletes
2. 1000 creates, writes, reads and deletes
3. 10000 creates, writes, reads and deletes

You need to run each of the above workloads against different number of threads (1, 2, 4, 8, 16) and report the following metrics in each case:

1. Average Latency
2. CPU and Memory Utilization

Additionally, explain your observation of the impact of the number of threads on latency.

Tip: Since you need to implement both memFS and its benchmark, it might be beneficial for you to separate the implementation of memFS functions and its interpreter(the one which is parsing the commands entered by user) so that in case of benchmarking you can directly invoke the functions bypassing the interpreter.

Submission Guidelines

Deliverables: For part 1 you need to create a design doc, its main source code(.c/cpp file) along with its makefile. For part 2 submit a report explaining comprehensively the benchmark results and observation along with the simulation code to run the benchmark.

Submission Guideline: Zip all the files into a single zip file named CL_Project_<Roll_Number>.zip and upload it on moodle.

Evaluation Guidelines

The total marks for this assignment = 100.
The marks breakup is given below:

Parts	Items	Marks
Part 1: Implementing memFS	`create`, `write` and `delete`	$15 \times 3 = 45$
	`read` and `ls`	$10 \times 2 = 20$
	Design Document	15
Part 2: Benchmarking memFS	Benchmarking Code	10
	Report	10
Total		**100**