

Design Document: Mini Project Three



Courtney Gosselin - 60186160
Benjamin Tisserand - 37615168
Alex Qin - 35732156
COSC 315 - Operating System
Dr. Apurva Narayan
April 8, 2020

Table of Contents

1.0 Introduction:	3
1.1 Part I: Memory management - Paging:	3
1.2 Part II: File Systems:	3
2.0 Prioritization/Planning:	4
3.0 Contributions:	4
4.0 Code Outline:	5
4.1 Part I: Memory management - Paging:	5
4.2 Part II: File Systems	5
5.0 Code structure/Folder layout:	7
6.0 Build, Compilation, and Run:	7
6.1 Part I: Memory management - Paging:	7
6.2 Part II: File Systems	8
7.0 Code output examples:	10
7.1 Part II: Memory Management:	10
7.2 Part II: File Systems:	11

1.0 Introduction:

1.1 Part I: Memory management - Paging:

This part of the program is to implement a portion of virtual to physical address translation in a pure paging based scheme. The program takes a filename as input that contains the first value n (n lowest significant bits that represent the offset), the second value m (the next m bits represent the page number), and all the following values are a sequence of virtual addresses and then the page number and offset are extracted for each virtual address using bitwise operations. The n and m values should add up to 16 and the virtual addresses are unsigned integers.

The input should be a file name:

inputPart1.txt

The output should be as follows:

Virtual address v1 is in page number p and offset d

virtual address v2 is in page number p and offset d

.....

1.2 Part II: File Systems:

This part of the program is to write a simple UNIX-like file system. The program should have the file system reside on a disk that has a size of 128KB with a maximum of 16 files (the maximum size of a file is 8 blocks; each block is 1 KB in size), each file has a unique name, the file name can be no longer than 8 characters. The system should also support the commands create, delete, list directory, read block, and write block. There should be only one root directory; no subdirectories are allowed.

An sample input file looks like this:

```
disk0
C file1 3
W file1 0
W file1 1
C mini-project.java 7
L
C file2 4
R file1 1
D mini-project.java
L
```

2.0 Prioritization/Planning:

The following is a breakdown of the project into a list of measurable and timed checkpoints

Product functionality:

- Part 1 and Part 2 (March 25, 2020)
- Manage code structures and put comments on (April 1, 2020)
- Design document (April 5, 2020)
- README.md (April 5, 2020)

3.0 Contributions:

Benjamin: @UBCbent

- Part 2 code
- Planning
- Code advisor
- Documentation (Design document)
- Document Editing

Courtney: @CourtneyGosselin

- Part 1 code
- Scrum master
- Documentation (Design document, README)
- Planning
- GitHub ticket system

Alex: @QinAlex

- Planning
- Feedback and Testing
- Documentation (Design document)

4.0 Code Outline:

4.1 Part I: Memory management - Paging:

Code:

- **Main()**: runs when you start the program. It initially states “Input file name: “ then opens the inputted file in read mode, and returns “Error the file does not exist” if the files cannot be found, otherwise continuing through the paging algorithm. Declaration of variables count, n, m, and input are made. Count keeps track of what line of the file we are on to differentiate n and m from the virtual addresses. The program enters a while loop that runs while there is a new line within the file. Here it sets the n and m values for the calculations of page number and offset for the virtual addresses and prints the N and M value. Finally once n and m have been set and we enter the 3rd file line we set the virtual address on the third line to v and we can now calculate page number and offset. These are set by calling the method `pageNumberCalculation(v, n)` for page number and `offsetCalculation(v, n)` to set the offset. It then prints out “Virtual address v# is in page number p and offset d” where # is the virtual address number we are on which can be 3 to as many lines there are in the code. P and d will be the values calculated by the method. This continues on till there are no more lines in the input file. Finally the input file is closed and the program ends.
- **pageNumberCalculation(v, n)**: Calculates the page number using bitwise operations and the virtual address and n the lowest significant bits that represent the page number.
- **offsetCalculation(v, n)**: Calculates the offset using bitwise operation using the input of the virtual address and lowest significant bits n.

Testing:

- **testPageNumberCalculation(n, v)**: Calculates using $v \text{ DIV } 2^n$ instead of bitwise calculations to check answers
- **testOffsetCalculations(n, v)**: Calculates using $v \text{ MOD } 2^n$ instead of bitwise calculations
- **runTesting(v, n, count)**: Is the method that needs to be triggered to run testing with the normal output that is currently commented out in the code that we handed in.

4.2 Part II: File Systems

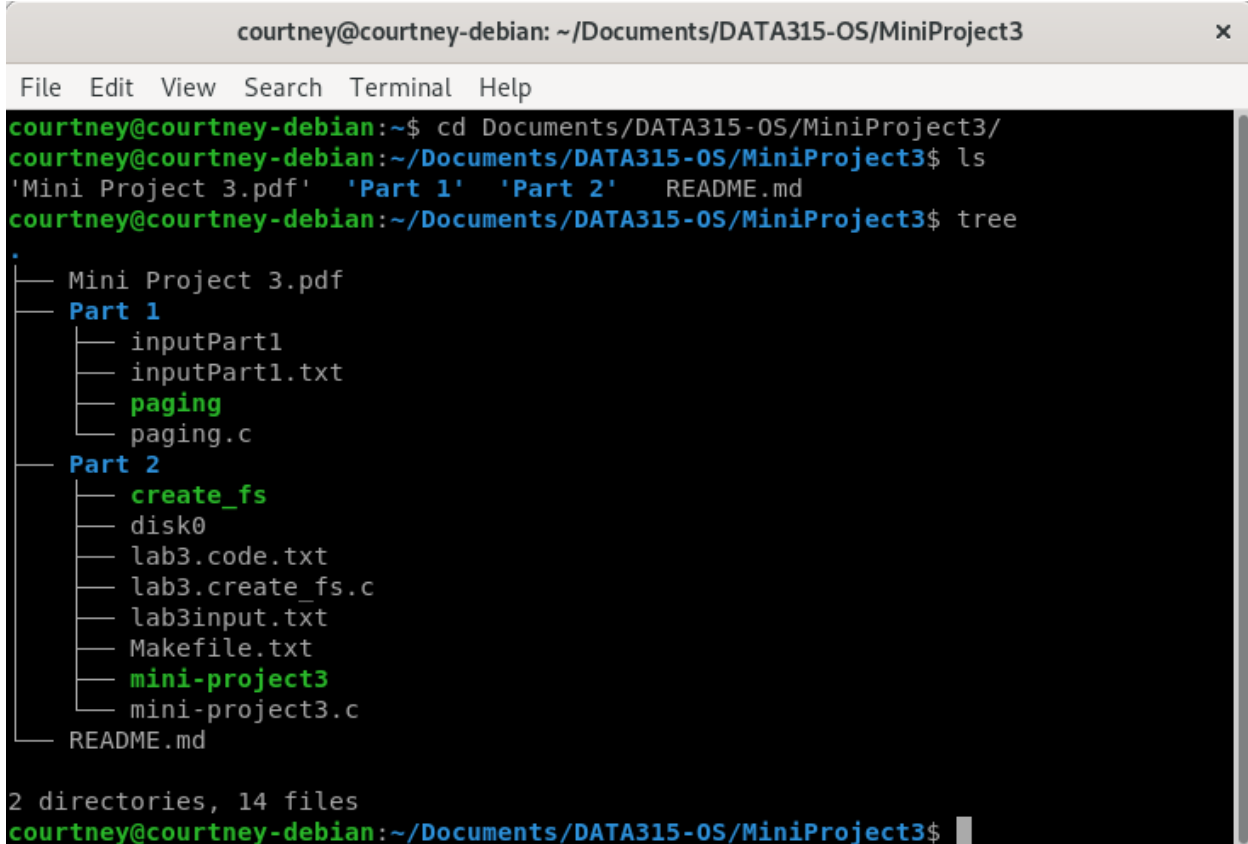
Code:

- **myFileSystem(const char* diskName)**: First, it opens the file with the name diskName, reads the first 1KB and parses it to structs representing the super block. It uses the sizeof operator to help cleanly determine the position of the next piece of the super struct. Last, cast the pointer to a pointer of the struct type.
- **create(char name[8], int size)**: This method creates a file with this name and this size. First: it looks for a free inode by searching the collection of objects representing inodes within the super block object. If none exist or another file is in use with the same name, then return an error. Second: look for a number of free blocks equal to the size variable passed to this method. If not

enough free blocks exist, then return an error. Third: now, we know we have the inode and free blocks necessary to create the file. Such that, mark the inode and blocks as used and update the rest of the information in the inode. Last, write the entire super block back to disk by seeking to the beginning of the disk and write the 1KB memory chunk.

- **delete(char name[8]):** This method deletes a file with this name. First, it looks for an inode that is in use with the given name by searching the collection of objects representing inodes within the super block object. If one does not exist, then return an error. Then, it frees the blocks of the file being deleted by updating the object representing the free block list in the super block object. After marking the inode and blocks as free, it writes the entire super block back to the disk. At the end, print out the delete information.
- **ls(void):** This method is to list names of all files on disk. We use a for loop to check all the inodes and print the name and size of the file if the inode is in use.
- **readBlock(char name[8], int blockNum, char buf[1024]):** This method reads this block from this file into the char buffer provided, returning an error if the file does not exist. First it locates the inode for this file and checks if blocknum is valid. Then, it uses fseek to seek the blockPointers[blockNum] and read the block from disk to buf. (We multiply by 1024 to convert block number to byte number, since each block is 1KB = 1024B.) In the end, we print out the read information.
- **writeBlock(char name[8], int blockNum, char buf[1024]):** This method writes into this block of this file and returns an error if blockNum exceeds the file size-1. First, it locates the inode for this file and checks if blocknum is valid. Then, it uses fseek to seek to blockPointers[blockNum] and write buf to the disk. (We multiply by 1024 to convert block number to byte number, since each block is 1KB = 1024B.) In the end, we print out the write information.
- **Main():** First, it opens the instruction file, reading the first line to open disk. Then, it iterates through instructions line by line and executes them by using a loop while there exists another line. Inside the loop, we first cut the line down to remove \n character, and split the string by its spaces using strtok(). After that, by using the switch case to find base instruction, acting accordingly. In the end, we close the file.

5.0 Code structure/Folder layout:



```
courtney@courtney-debian: ~/Documents/DATA315-OS/MiniProject3
File Edit View Search Terminal Help
courtney@courtney-debian:~$ cd Documents/DATA315-OS/MiniProject3/
courtney@courtney-debian:~/Documents/DATA315-OS/MiniProject3$ ls
'Mini Project 3.pdf'  'Part 1'  'Part 2'  README.md
courtney@courtney-debian:~/Documents/DATA315-OS/MiniProject3$ tree
.
├── Mini Project 3.pdf
├── Part 1
│   ├── inputPart1
│   ├── inputPart1.txt
│   ├── paging
│   └── paging.c
├── Part 2
│   ├── create_fs
│   ├── disk0
│   ├── lab3.code.txt
│   ├── lab3.create_fs.c
│   ├── lab3input.txt
│   ├── Makefile.txt
│   ├── mini-project3
│   └── mini-project3.c
└── README.md

2 directories, 14 files
courtney@courtney-debian:~/Documents/DATA315-OS/MiniProject3$
```

6.0 Build, Compilation, and Run:

The following contains the build instructions stating how to compile the code on a Linux OS for our programs. It also contains Run instructions stating how to execute the compiled code on Linux.

6.1 Part I: Memory management - Paging:

The code has already been compiled with gcc and can be found within the folder structure. The compiled program for Part 1 can be found in the Part 1 directory and is called paging. To run the program you need to navigate inside the folder structure.

1. Clone the repository to your local machine
2. Use the gcc to compile and use the `-lm` flag in Part 1 for compiling due to the Math.h library.
- `gcc paging.c -o paging -lm`
3. To run the program you need to type the following into the terminal (**start here if you are using our compilation**)

- `./paging`
- 4. You will be prompted to input a filename you can use “inputPart1.txt” as it was the test file we were using.
 - “Input a file name: “
 - “inputPart1.txt”
- 5. After taking these inputs the program will iterate through the input file taking the first two values as n and m and the following are the virtual addresses and will return the page number and offset.
 - “Virtual address v1 is in page number p and offset d”

6.2 Part II: File Systems

The code has already been compiled with gcc and can be found within the folder structure. The compiled program for Part 1 can be found in the Part 1 directory and is called paging. To run the program you need to navigate inside the folder structure.

1. Clone the repository to your local machine
2. Use the gcc to compile and use the `-lm` flag in Part 1 for compiling due to the Math.h library.
 - `gcc mini-project3.c -o mini-project3`
3. To run the program you need to type the following into the terminal (**start here if you are using our compilation**). Running the `./create_fs disk0` resets the disk so we are working from a fresh state.
 - `./create_fs disk0`
 - `./mini-project3`
4. You will be prompted to input a filename you can use “inputPart1.txt” as it was the test file we were using.
 - “Input a file name: “
 - “lab3input.txt”

- ```

Enter name of instruction file: lab3input.txt
Created file file1.c (size 3) allocated to blocks 1 2 3
Created file file2.c (size 8) allocated to blocks 4 5 6 7 8 9 10 11
Created file file3.c (size 4) allocated to blocks 12 13 14 15
Created file a.out (size 5) allocated to blocks 16 17 18 19 20
Created file lab1.jav (size 6) allocated to blocks 21 22 23 24 25 26
Name: file1.c Size: 3
Name: file2.c Size: 8
Name: file3.c Size: 4
Name: a.out Size: 5
Name: lab1.jav Size: 6
Wrote to file1.c block 0 (disk address 1)
Wrote to file1.c block 1 (disk address 2)
Wrote to file1.c block 2 (disk address 3)
Wrote to file2.c block 3 (disk address 7)
Wrote to file2.c block 7 (disk address 11)
Wrote to file2.c block 2 (disk address 6)
Wrote to file2.c block 4 (disk address 8)
Wrote to file2.c block 5 (disk address 9)
Wrote to a.out block 0 (disk address 16)
Wrote to a.out block 1 (disk address 17)
Wrote to a.out block 2 (disk address 18)
Wrote to a.out block 3 (disk address 19)
Wrote to a.out block 4 (disk address 20)
Deleted file file3.c (size 4) and freed blocks 12 13 14 15
Read file1.c block 2 (disk address 3) into buffer.
This file has been written to
Created file file4.c (size 7) allocated to blocks 12 13 14 15 27 28 29
Name: file1.c Size: 3
Name: file2.c Size: 8
Name: file4.c Size: 7
Name: a.out Size: 5
Name: lab1.jav Size: 6
Read file2.c block 4 (disk address 8) into buffer.
This file has been written to
Read file2.c block 5 (disk address 9) into buffer.
This file has been written to
Read file2.c block 6 (disk address 10) into buffer.

Deleted file lab1.jav (size 6) and freed blocks 21 22 23 24 25 26
Created file lab2.jav (size 7) allocated to blocks 21 22 23 24 25 26 30
Read a.out block 1 (disk address 17) into buffer.
This file has been written to
Read a.out block 3 (disk address 19) into buffer.
This file has been written to
Read a.out block 0 (disk address 16) into buffer.
This file has been written to
Name: file1.c Size: 3
Name: file2.c Size: 8
Name: file4.c Size: 7
Name: a.out Size: 5
Name: lab2.jav Size: 7

```

## 7.0 Code output examples:

### 7.1 Part II: Memory Management:

```
courtney@courtney-debian:~/Documents/DATA315-OS/MiniProject3/Part 1$./paging
Input file name: inputPart1.txt
inputPart1.txtError the file does not exist
courtney@courtney-debian:~/Documents/DATA315-OS/MiniProject3/Part 1$./paging
Input file name: inputPart1.txt
inputPart1.txtThe N value 7
The M value: 9
virtual address v1 is in page number 0 offset 4
virtual address v2 is in page number 23 offset 56
courtney@courtney-debian:~/Documents/DATA315-OS/MiniProject3/Part 1$
```

## 7.2 Part II: File Systems:

```
Enter name of instruction file: lab3input.txt
Created file file1.c (size 3) allocated to blocks 1 2 3
Created file file2.c (size 8) allocated to blocks 4 5 6 7 8 9 10 11
Created file file3.c (size 4) allocated to blocks 12 13 14 15
Created file a.out (size 5) allocated to blocks 16 17 18 19 20
Created file lab1.jav (size 6) allocated to blocks 21 22 23 24 25 26
Name: file1.c Size: 3
Name: file2.c Size: 8
Name: file3.c Size: 4
Name: a.out Size: 5
Name: lab1.jav Size: 6
Wrote to file1.c block 0 (disk address 1)
Wrote to file1.c block 1 (disk address 2)
Wrote to file1.c block 2 (disk address 3)
Wrote to file2.c block 3 (disk address 7)
Wrote to file2.c block 7 (disk address 11)
Wrote to file2.c block 2 (disk address 6)
Wrote to file2.c block 4 (disk address 8)
Wrote to file2.c block 5 (disk address 9)
Wrote to a.out block 0 (disk address 16)
Wrote to a.out block 1 (disk address 17)
Wrote to a.out block 2 (disk address 18)
Wrote to a.out block 3 (disk address 19)
Wrote to a.out block 4 (disk address 20)
Deleted file file3.c (size 4) and freed blocks 12 13 14 15
Read file1.c block 2 (disk address 3) into buffer.
This file has been written to
Created file file4.c (size 7) allocated to blocks 12 13 14 15 27 28 29
Name: file1.c Size: 3
Name: file2.c Size: 8
Name: file4.c Size: 7
Name: a.out Size: 5
Name: lab1.jav Size: 6
Read file2.c block 4 (disk address 8) into buffer.
This file has been written to
Read file2.c block 5 (disk address 9) into buffer.
This file has been written to
Read file2.c block 6 (disk address 10) into buffer.

Deleted file lab1.jav (size 6) and freed blocks 21 22 23 24 25 26
Created file lab2.jav (size 7) allocated to blocks 21 22 23 24 25 26 30
Read a.out block 1 (disk address 17) into buffer.
This file has been written to
Read a.out block 3 (disk address 19) into buffer.
This file has been written to
Read a.out block 0 (disk address 16) into buffer.
This file has been written to
Name: file1.c Size: 3
Name: file2.c Size: 8
Name: file4.c Size: 7
Name: a.out Size: 5
Name: lab2.jav Size: 7
```