Introduction

Project 1 will cover the following:

* Part 1 - Implementing the uniq command
* Part 2 - Implementing the head command

This project is a total of 600 points.

As we are creating simplified versions of actual Unix commands, please test out the actual Unix commands that you are unfamiliar with before you begin to develop your own.

Assignment Notes

General Notes

• Please read the assignment instructions carefully, including what files to include/submit.

• Don’t assume any limitations unless they are explicitly stated.

• Developing optimized solutions is good, but I won’t deduct any points if your code is not optimized

• Please ensure the functionalities requested in these assignments are included; you can add extra but cannot remove any.

• When in doubt regarding what needs to be done, ask. Another plausible option is to test it in the real UNIX or Ubuntu operating system. Unix is your friend; use it wisely. Does your code behave the same way? If not, debug. Remember to use man "command name" for reference.

• Test your solutions and make sure they work. Attach your screenshots showing your steps, your console, and other details.

**File Submission Instructions:**

The file should be submitted in zip format, with 2 folders (uniq and head)

Each folder should consist of the following documents:

• Screenshots showing successful execution of the command "make clean".

• Screenshot showing successful execution of the command "make qemu" or "make qemu-nox".

• Output of all conditions provided in your rubric.

• A detailed Readme file that explains your logic (code section), and steps followed to run your code

(Attach a screenshot and then explain the steps). If README files are missing, then we will deduct 25 points for each command (25 for uniq and 25 points for head).

• All c scripts and modified Makefile.

*NOTE: Make sure your code compiles using the Makefile! You will****automatically lose 50% of the points if your code does not compile or does not add the command to xv6****. The remaining points will be up to the discretion of the person grading the assignment based on whether what exists seems to properly implement the expected functionality.*

**Academic Dishonesty:** We will be checking your code against other submissions in the class for logical redundancy. If you copy someone else’s code and submit it with minor changes, we will know. These cheat detectors are quite hard to fool, so please don’t try. We trust you all to submit your own work only; *please* don’t let us down. If you do, we will pursue the strongest consequences available to us.

**Getting Help:** You are not alone! If you find yourself stuck on something, contact the course TAs/Instructor for help. Office hours are there for your support; please use them. If you can’t make our office hours, let us know and we can talk over email or schedule a one-on-one meeting. We want these projects to be rewarding and instructional, not frustrating and demoralizing. But, we don’t know when or how to help unless you ask.

**Course Effort:**Start your assignment early, this not only depicts effort put into this class but allows for ample time for TA access and help. If there are questions/useful points to make note of, please use the course discussion page to discuss and ask relevant questions.

**Part 1 - Implementing uniq on xv6 as user program and also as system call - 300 points**

The uniq command in Linux is a command-line utility that reports or filters out the repeated lines in a file. In simple words, uniq is the tool that helps to detect the adjacent duplicate lines and also deletes the duplicate lines. Uniq filters out the adjacent matching lines from the input file (that is required as an argument) and writes the filtered data to the output file. For instance, look at a simple example from a file called OS611example.txt. The cat command will print out all the contents from the file, and uniq removes duplicates from the adjacent line.

• I understand the Operating system.

• I understand the Operating system.

• I understand the Operating system.

• I love to work on OS.

• I love to work on OS.

• Thanks xv6.

Now, when I run the uniq command on the example.txt I get the following $uniq OS611 example.txt

• I understand the Operating system.

• I love to work on OS.

• Thanks xv6.

Part 1 Task-1 - User space -

create uniq.c that performs the above task by replicating the functionality of uniq. This program should display/print the following message ” Uniq command is getting executed in user mode, ” followed by content. In the above case, you will get the following output

$uniq OS611example.txt

"Uniq command is getting executed in user mode."

• I understand the Operating system.

• I love to work on OS.

• Thanks xv6.

Part 1 Task-2 - Kernel space -

Create a system call named uniq. This should display/print the following message "Uniq command is getting executed in kernel mode", then followed by content.

Rubric for Part-1

We assume all commands are working in user and kernel mode.

• Program is working in user mode but not kernel mode. (-100)

• If uniq does not compile in any mode (user or kernel mode). (-200)

• If nothing works, then based on logic, some points will be provided (range 0 -100).

• If any debug statements (printf) are getting printed alongside output. (-15)

• If the program does not terminate successfully (-100)

• cat "filename" | uniq does not work (-25)

• uniq -c "filename" does not work (-15)

• uniq -i "filename" does not work (-25)

• uniq -d "filename" does not work (-15)

**Part 2 - Implementing head on xv6 as user program and also as system call - 300 points**

It is complementary to the Tail command. .e head command, as the name implies, prints the top N number of data of the given input. By default, it prints the first 14 lines of the specified files. If more than one file name is provided, then data from each file is preceded by its file name. If no filename is provided, the head should read from standard input. You should also be able to invoke it without a file and have it read from standard input. (Important: If the first argument does not start with -, the number of lines to be printed should default to 14).

Like uniq, you will also implement head command in user and kernel mode.

Part 2 Task-1 - User space -

create head.c that performs the above task by replicating the functionality of the head. This program should display/print the following message: "Head command is getting executed in user mode, " followed by content.

Part 2 Task-2 - Kernel space -

create a system call named head. This should display/print the following message: "Head command is getting executed in kernel mode", then followed by content.

Rubric for Part-2

• The program is working in user mode but not kernel mode. (-100)

• If the head program does not compile in any mode (user or kernel mode). (-200)

• If nothing works, then based on logic, some points will be provided (range 0 -100).

• If any debug statements (printf) are ge.ing printed alongside output. (-15)

• If the program does not terminate successfully (-100)

• head "filename" does not work. Assuming it works but doesn’t respect the default settings = 14 lines. (-50)

• head -n "some number" "filename" does not work (-45)