Despina Patronas
CSE 130

Design Document Assignment 0: Shoulders

Introduction

I. **Goals and Objectives:**

This assignment is to implement *shoulders* as the unix command head (without support for any flags) in quiet mode with C programming language.

II. Usage and Scope of *Shoulders*

i. *Shoulders* must take an argument for line numbers (n).

ii. *Shoulders* must work for any given data input.

iii. *Shoulders* must compile without warnings for the following flags: -Wall -Wextra -Werror -Wpedantic -std=c99 -g

iv. *Shoulders* may not use C FILE* functions (fread() or printf()) *with the exception of fprintf(stderr).

v. FILES: If given file argument – (dash) as file, then *shoulders* echos stdin for (n).

1. If given no file argument, then *shoulders* will similarly echo stdin for (n).
2. If there is a file error, *shoulders* will print an error message for that file, then continue reading remaining file arguments, if any.
3. *Shoulder* should handle multiple dash input for file names.

vi. *shoulders* should be equivalent in performance to head.

vii. *shoulders* should be able to take arbitrarily large file sizes.

III. Design

Shoulders relies on open(), read(), write(), close() system calls for I/O.
Note: **these are expensive operations as the system has to fetch from disk!**

i. **Open()**

Usage: *int fd = open(const char *filename, int flags);*
If successful, returns a file descriptor for the opened file, otherwise, returns -1
flags: O_RDONLY to read the open file

ii. **Read()**

Usage: *int r_count = read(int fd, void *buff, int count);*
If successful, returns number of bytes read from file descriptor to buff (0 for End of File (EOF)), otherwise returns -1.

shoulders: Read will be done in a loop, with constant buffer size until EOF.
Note: the size which you read in is subjective and performance will vary based on your implementation.

iii. Write()

Usage: *int w_count = write(int fd, void * buff, int count);*

If successful, returns number of bytes written to file descriptor from buff, otherwise returns -1.

shoulders: Write will be done in a loop, with constant buffer size until EOF OR End of Line.

Example Outline:

(note: prog_name is argv0)
(line arguments is argv1)
(file arguments are argv2+)

main(argc, argv):

If: ***(Error0: no arguments passed into shoulders)***
argc == 1:
Show error msg for usage
shoulders: requires an argument
Exit program
endIf

Loop argv [0,argc):

If argv == 1: (***Check argv1 is line number digit)***

Store argv1 as n

If: (***Error 1 - bad usage***)
n == "-qn"
Print error message:
shoulders: option requires an argument -- 'n'
Try 'shoulders --help' for more information.
Exit Program with error
end if

ElseIf: (***Error 2 – bad line number input***)

n is not comprised of digits || n is negative:
Generate error msg similar to head.
use fprintf(stderr) :
prog_name*: invalid number of lines: 'argv2'.*
Exit Program with error

end Elseif

If: **(Edge Case Line number is 0 –> no processing needed)**

N == 0
Exit Program successfully

endif

If: ***(No files argument– take stdin for line number n)***

argc == 2 (only program name and line argument)
Process(STDIN , n)

end if
end if

if argv >= 2 ***(Check rest of argv (files))***

if: file name contains '-' (Dash)

Process(STDIN , n)
move onto next file (if any)

end if

if: trying to open file results in open returning -1

warn(file open error)
move onto next file (if any)

end if

otherwise process opened file

Process(file descriptor, n)

end if

end Loop

End main

Two Methods for processing:
The one that yielded the fastest results was Processing1 (Which is used in the final implementation of shoulders)

Const int SIZE = 2048

In parameters: infile = file descriptor (either stdin or file)
n = line number from CLI arg
Empties buffer to stdout

Processing1(infile, n):
Create an inBuffer for reading in data *(SIZE = large enough for bulk data transfer ex: 1KiB+)*
Set linesfound to 0 *(keeps track of newlines found in buffer(s))*

Loop:

readIn = Read (infile, inBuffer, SIZE)
Set a lastLine to 0 *(last linehas not been found, is negative to start)*

If readIn < SIZE
set lastLine to 1 (readIn reached EOB)

Iterate from i[0,SIZE] (check the buffer for newlines)

If (inBuffer[i]),
increment linesFound counter

end if

If linesFound == n (reached the expected line count)
OR lastLine == 1 AND I == readIn (buffer has last line AND i is at EOB)

Write out buffer to stdout
break loop

end if:

if: linesFound != n

write out entire buffer to stdout

End if:

End Loop:

Free Buffer

End Processing1

In parameters: infile = file descriptor (either stdin or file)
n = line number from CLI arg
Empties buffer to stdout

Processing2(infile, n):

Create an inBuffer for reading in data *(SIZE = large enough for bulk data transfer ex: 1KiB+)*

Set linecounter to 0 *(keeps track of lines written)*

Loop:

readIn = Read (infile, inBuffer, SIZE)
Set a tempArray as inBuffer
Set a charcounter to 0 *(keeps track of characters for iterating the inBuffer)*

Loop: iterating I [0, readIn]

Increment charcounter

If: inBuffer[charactercounter] == newline

Increment linecount

Write(stdout, readbuff, character counter)

Increment charcounter

Set tempArray to next character

If: iterator i == readIn *(aka end of buffer)*

Write(stdout, readbuff, charcounter)

Increment charcounter

Set tempArray to next character

If: linecounter >= n

Break loop

While: linecounter < n

End Loop

End Processing2

End Outline