FZA2100 Assignment Task 1 Building a tail utility with C Programming Semester 2 2020

Dr Olalekan Samuel
Ogunleye.
Email: ogunleyeos@iiemsa.co.za

September 11, 2020

Contents

1 Introduction

In this assignment, you will be required to build a simplified version of the Linux tail utility, which is used for viewing the last portion of a long text file. tail is generally useful for viewing the last few lines in long log files that the operating system produces (for example, the /var/log/kern.log file in your virtual machine environment). Linux server administrators use the tail utility for checking recent activity in network logs.
This document constitutes the requirement specification for this assignment. You will be assessed on your ability to both comprehend and comply with the requirements as specified herein.
Due: 23rd October 2020 (Friday) 6 pm.
Late submissions: Under no circumstance will late submission be allowed. It is essential to take note of this.

READ THE INSTRUCTIONS VERY CAREFULLY FAILURE TO DO SO WILL RESULT IN DEDUCTION OF MARKS

2 Caveats

To complete these tasks, you are allowed to use any of the standard C libraries found on your virtual machine environment[1], except one:
You are NOT allowed to use any functions available in <stdio.h>. This means you cannot use printf() to produce output. (For example: to print output in the terminal, you will need to write to standard output directly, using appropriate file system calls.) This makes the assignment more challenging, but also means you are interacting with services of the operating system directly.
Your main C source file should be named with your student ID as: ctail-123456789.c, where 123456789 is your student ID.[2]

[1] For example, you may use getopt according to your preference, but use of any particular library is neither expected nor required, except as required to make system calls.

[2] If your completed program contains multiple source iles, you may name other source and header iles as you wish.

3 If you require extra help

This assignment is an independent learning and assessment exercise. You may utilize the student forums on the Unit page to ask questions and obtain clarification. However, you may not share details or code in your implementation with other students, nor may you show your code to the lecturer and the tutor (either to seek clarification about whether the code is correct or to find assistance) before submission. This is an assessment. The tutor and lecturer are not permitted to help you with your assignment code directly (you are expected to debug and test your code). However, they can help with more general queries, such as queries related to C programming syntax, concepts, and debugging tips.
You may make use of online references with an appropriate citation following academic integrity policies; however, your work must be your own.

3.1 References

You are required to conduct some more research if need be to complete this assignment

Section A

4 Task 1: Character-based functionality with hard- coded options

Write a utility called ctail ('character tail') which does the following:

1. 0pens a file named logfile.txt in the current working directory.
2. 0utputs (to standard output-usually the terminal) only the last 200 characters from that file. If the file contains fewer than 200 characters, output the entire file contents. (Where the program usually completes, no other output should be produced.)

Your program should always exit 'cleanly,' i.e. close any open files (and free any resources you allocate) before termination.

If there is a problem accessing the file (e.g. file does not exist), your program should display an error message (you should output this to the program's standard error stream rather than standard output - why?) and exit cleanly with a return value of 1. 0n successful completion, your program should return an error code of 0.

Write up an instruction manual (user documentation) in a plain text file, explaining how to compile your program and how to use it.

You may test your program using either your user documentation itself, or by using system log data found in many files under the /var/log/ directory.

5 Task 2: Support command-line options

Now extend your program to allow the user to run your program using command-line arguments as follows:

1. Instead of the default 200 characters, allow the user to specify an -n argument at the command line in order to specify a different number of characters. Where the -n option is used, the argument immediately following it is treated as the number of characters to be used. If the next argument after an -n argument is not a non-negative integer, the program arguments are invalid (see below).
2. Allow the user to specify a different filename (instead of logfile.txt) by putting the filename in a command-line argument.

Example commands:

```
$ ./ctail /var/log/kern.log
[output will be last 200 chars from kern.log]
$ ./ctail readme.txt −n 80
[output will be last 80 chars from readme.txt]
$ ./ctail −n 5
[output will be last 5 chars from logfile.txt]
```

Both command-line options are optional: if an argument is not provided, your program should use the default settings from task 1.

If an argument is invalid (for example: ./ctail -n filename.txt), your program should display an error message and then exit cleanly with an error code of 2.

Do not collect these options from standard input, or prompt the user to enter them. They should be specified as program arguments.

Extend your user documentation to include the added usage functionality.

6 Task 3: Add support for line-based functionality

Add support for one additional (optional) argument:

1. The -L argument, if specified within the program arguments, should switch the program to line-based mode. In this mode, ctail outputs the last 10 lines[3] in the file by default. If the -n argument is specified, the value given by the user should be used as the number of lines rather than characters. If the file is shorter than 10 lines, the entire file contents should be output.

Extend your user documentation to include the new functionality.

6.1 Important: commenting is required

Commenting your code is essential as part of the assessment criteria (refer to Section 6.2). All program code should include three types of comments: (a) File header comments at the beginning of your program file, which specify your name, your Student ID, the start date and the last modified date of the program, as well as with a high-level description of the program.
(b) Function header comments at the beginning of each function should describe the function, arguments and interpretation of return value. (c) In-line comments within the program are also part of the required documentation.

6.2 Marking Criteria

Each task is worth an equal share of the total. The same marking criteria will be applied on all tasks:

[3] Each line in a text is separated by a newline '\n' character.

- 50% for working functionality according to specification.
- 20% for code architecture (algorithms, use of functions for clarity, appropriate use of libraries, correct use of pointers, etc. in your implementations of the three tasks.)
- 10% for general coding style (clarity in variable names, function names, blocks of code clearly indented, etc.)
- 20% for documentation (user documentation describes functionality for the relevant task, code is well-commented.)

6.3 Hints and tips

Do...	Don't... (!)
+ read up on low-level I/0 system calls	– use fancy C <stdio.h> library functions that don't demonstrate your understanding of mak- ing system calls to the 0S directly.
+ write normal program output to standard output	– write error messages to standard output (use standard error instead)
+ close open files before exiting and free any manually-allocated memory	– terminate without cleaning up first
+ check return values and handle error condi- tions	– act in undefined ways when a parameter is invalid or a file can't be opened
+ access and modify valid memory	– attempt to set values in undefined pointers
+ read up on argc and argv	– prompt the user to enter options after your program has started
+ write user documentation in a plain text file	– submit user documentation in a Word docu- ment or PDF
+ in user documentation: explain all function- ality to those who might use your program	– expect your users to understand the C code inside your program (users are not always pro- grammers!)
+ use descriptive, self-explanatory variable names	– use vague single-character variable names
+ break your program logic into multiple func- tions	– stuff everything into a single main function
+ use consistent code indentation for clarity	– use inconsistent indentation or fail to indent nested code blocks
+ comment your code with file header, function header and inline comments	– write uncommented code or add comments at the last minute
+ test your program in the specified VM envi- ronment	– forget to run your finished program through valgrind to test for memory access bugs
+ read the assignment specification care- fully and thoroughly	– treat this table as a substitute for reading the spec carefully.

7 Submission

There will be N0 hard copy submission required for this assignment. You are required to archive and compress all your deliverables into a single .tar.gz/zipped/rar file named with your Student ID, for submission. For example, if your Student ID is 12345678, you would submit a zipped file named 12345678_Ass.tar.gz./ 12345678_Ass.zip/12345678_Ass.rar

Your submission is via the assignment submission link on the FZA2100 Moodle site by the deadline specified in Section 1, i.e. 23rd Octoiber 2020 (Friday) by 6:00pm.

Note: You must ensure you complete the entire Moodle submission process (do not simply leave your assignment in draft status) to signify your acceptance of academic integrity requirements.

7.1 Deliverables

Your submission should be archived and compressed into a single .tar.gz file containing the following documents:

- Electronic copies of ALL your files (e.g. C source file(s)) that are needed to compile and run your program. (Note that your program must run in the Linux Virtual Machine environment which has been provided for this unit. Any implementation that does not run at all in this environment will receive no marks.)
- A user documentation file (not more than 3 pages) in plain .txt format with clear and complete instructions on how to compile and run your program.

Marks will deducted for any of these requirements that are not strictly complied with.

7.2 Academic Integrity: Plagiarism and Collusion

Plagiarism Plagiarism means to take and use another person's ideas and or manner of express- ing them and to pass them off as your own by failing to give appropriate acknowledgement. This includes materials sourced from the Internet, staff, other students, and from published and unpublished works.

Collusion Collusion means unauthorised collaboration on assessable work (written, oral, or practical) with other people. This occurs when you present group work as your own or as the work of another person. Collusion may be with another Monash student or with people or students external to the University. This applies to work assessed by Monash or another university.

It is your responsibility to make yourself familiar with the University's policies and procedures in the event of suspected breaches of academic integrity. (Note: Students will be asked to meet with the lecturer should such a situation is detected.)