Natalie Petrosian

CSE 130 Principles of Computer System Design

Assignment 0

11 April 2020

1.0 Introduction

This assignment implements the basic Unix command ‘cat’,*without support for any flag*s *and* ***with file arguments in reverse order***. For example, dog file1 file2 file3 will copy all of the data from file3, file2, and file1 to standard output, in that order. If - (dash) is given as a filename, dog uses standard input for that file until it encounters an EOF.

1.1 Goals and objectives

Per the rubric, there are three goals for Assignment 0. The first is to install an Ubuntu 18.04 virtual machine on your personal computer. The second is to write a simple program that acts similar to the UNIX cat command with a simple modification. This lab will require the submission of a design document, a README file, and a lab writeup along with code in the git repository. The third goal is to ensure readiness for UCSC’s CSE 130 Principles of Computer System Design course by administering a straightforward assignment that will act as a “self-test”.

1.2 Statement of scope

The software emulates the functionality of the Unix command ‘cat’ without support for any of the ‘cat’ flags. Its parameters can be files (text or binary) and a - (dash), which implies accepting input from the standard input. It writes the result to standard output in the reverse order of the parameters. Hence, the major inputs are either files or the standard input. The major outputs are standard out and standard error (for handling error messages). This is a simple assignment, and all of the requirements are deemed essential and should be implemented in the current version without leaving anything for the future.

1.3 Software context

The purpose of this software is to demonstrate an understanding of file descriptors and Unix I/O, while under the limitations described in 1.4.

1.4 Major constraints

This program must be written in C. Using FILE pointers and any related library functions are prohibited. Similarly, any iostream and fstream library functions shall not be used. File I/O buffers shall not exceed 32 KiB of memory. The program must be able to handle large and small text and binary files specified as either specified parameters or redirected from other processes with pipes or redirection.

2.0 Data design

Fixed size arrays will be used in this program. For I/O operations, the write() and read() library calls will be employed. A fixed array character buffer of 32,000 bytes will be used in the major I/O operations and a 500 character fixed buffer size will be used to construct error messages passed to perror(), which will be used for error handling.

2.1 Internal software data structure

char buff [MAX_SIZE]; This buffer will be used for the main read/write operations.

char error_message [ERR_MSG_SIZE]; This buffer will be used for handling error messages.

MAX_SIZE shall be defined as 32,000 bytes and ERR_MSG_SIZE shall be defined as 500 bytes.

2.2 Global data structure

No global data structures are needed. Data structures, as described in 2.1, should be local to functions.

2.3 Temporary data structure

The temporary data structures in this program are quite trivial. An integer called chars_read shall be used to capture the return value of the read() library call and the file descriptor returned by the open() call shall be captured in an integer called input_fds.

2.4 Database description

No databases are used in this program.

3.0 Architectural and component-level design

This is a trivial program, and shall contain only two functions: main() and readFromStandardInput().

main() shall be designed as follows: When argc is 1 (no input parameters are specified), input shall be read from standard-in by calling the function readFromStandardInput(). Otherwise, the input parameters will be processed in reverse order by a for loop. If a - (dash) is encountered, input shall be read from standard-input by calling the function readFromStandardInput(). Otherwise, the open() system call will be used to retrieve a file descriptor, and that file descriptor will be used by the read() library call to fill up a 32,000 character buffer named buff and write it to standard-output. Note that the input file might be larger than 32,000 bytes, therefore, this reading and writing must continue until the read() library call returns 0, which means an EOF has been detected. Any errors in I/O operations will be reported by perror().

readFromStandardInput() shall be designed as follows: This is a short function that employs a while loop to call the read() library function (with a 32 KiB limitation) and writes the characters that have been read until an EOF has been encountered indicated by a return value of 0 from read().

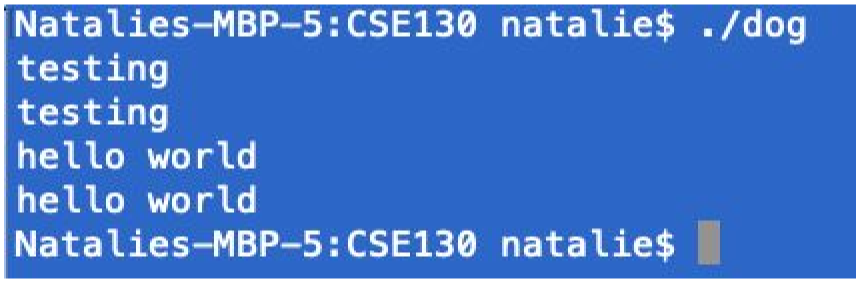
4.0 User interface design

A description of the user interface design of the software is presented.

This is a simple command-line program with no graphical user interface. The program will be called ‘dog’, and it should be able to process file names passed to it as well as a - (dash), which implies accepting input from standard-in.

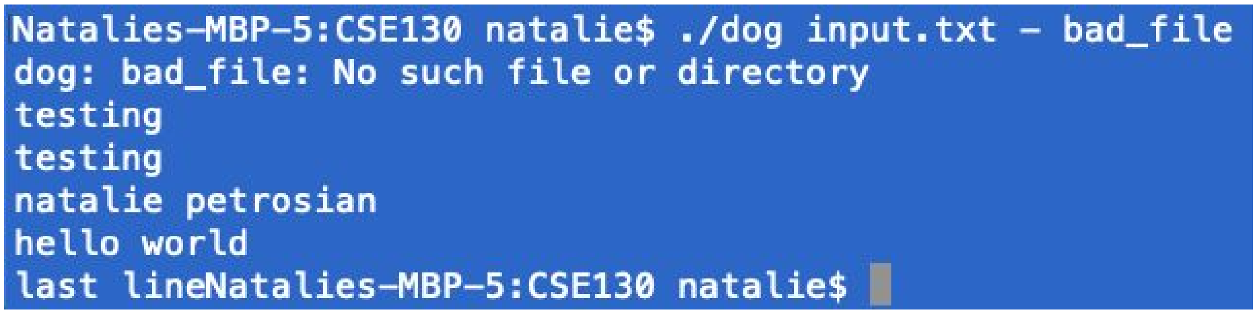
4.1.1 Screen images

Example of providing no parameters to ‘dog’:



```
Natalies-MBP-5:CSE130 natalie$ ./dog
testing
testing
hello world
hello world
Natalies-MBP-5:CSE130 natalie$ 
```

Example of providing a file with content, the - (dash), followed by a file that does not exist.



```
Natalies-MBP-5:CSE130 natalie$ ./dog input.txt - bad_file
dog: bad_file: No such file or directory
testing
testing
natalie petrosian
hello world
last lineNatalies-MBP-5:CSE130 natalie$ 
```

6.0 Testing Issues

Ensure that parameters passed to ‘dog’ are processed in the reverse order.

Ensure that the 32 KiB limitation has been implemented in the code.

Ensure that no FILE pointers and related library calls have been used.

Ensure that large files (text and binary) can be handled by the code.

Ensure that error messages are going to standard-error.

6.1 Classes of tests

Tests for this program fall into two categories:

(1) Limitations by the design document are followed by the code
(2) Functionality specified by the design document has been implemented

6.2 Expected software response

The software should mimic the functionality of the Unix command ‘cat’ (excluding flags) but in reverse order of the specified parameters. The - (dash) parameter should indicate that the input is standard-in.