Problem 1 -- System calls, error checking and reporting

The objective of this assignment is to write a simple C program which is invoked from the command line in a UNIX environment, to utilize the **UNIX system calls** for file I/O, and to properly handle and report error conditions.

The program is described below as a "man page", similar to that which describes standard UNIX system commands. The square brackets [ ] are not to be typed literally, but indicate optional arguments to the command.

```
minicat - concatenate and copy files

USAGE:

      minicat [-b ###] [-o outfile] infile1 [...infile2....]
      minicat [-b ###] [-o outfile]

DESCRIPTION:

This program opens each of the named input files in order, and concatenates
the entire contents of each file, in order, to the output.  If an
outfile is specified, minicat opens that file (once) for writing, creating it
if it did not already exist, and overwriting the contents if it did.
If no outfile is specified, the output is written to standard output, which
is assumed to already be open.

Any of the infiles can be the special name - (a single hyphen).  minicat
will then concatenate standard input to the output, reading until end-of-file,
but will not attempt to re-open or to close standard input.  The hyphen can be
specified multiple times in the argument list, each of which will cause
standard input to be read again at that point.

If no infiles are specified, minicat reads from standard input until eof.

The optional -b### argument can be used to specify the buffer size in bytes.

EXIT STATUS:

program returns 0 if no errors (opening, reading, writing or closing)
were encountered.
Otherwise, it terminates immediately upon the error, giving a proper
error report, and returns -1.
```

• Use UNIX system calls directly for opening, closing, reading and writing files. Do not use the stdio library calls such as

fopen for this purpose. [You may use stdio functions for error reporting, arguments, etc.]

- As part of your assignment submission, show sample runs which prove that your program properly detects the failure of system calls, and makes appropriate error reports to the end user. For example, you can test the `open` system call error handling by specifying an input file that does not exist. Read, write and close errors are harder to generate at this stage of the course -- you could optionally try using a USB memory device that you yank out while the program is running -- but regardless you must still properly check for and report errors on these system calls.
- Experiment with different read/write buffer sizes. As part of your submission, measure the throughput of your program in MB/sec for these various buffer sizes, from 1 byte to 256K, in powers of 2. This can be accomplished by timing each program run using the UNIX command time(1). If you are ambitious, you can write a script in shell, Perl, python, etc. to automate the tests and data collection, but this is optional.
- Present and discuss your experimental results. E.g. show a graph relating buffer size to performance. What effect, if any, does buffer size have? Why do you think this is the case? Be sure to describe the type of hardware and operating system on which your test cases were run. *Note: in performing your tests, avoid using device special files such as /dev/null or /dev/zero. Read/write to these special files has certain semantics, which we have not yet covered, which will greatly skew your results. If you are running from a live-cd environment, make sure to note that in your write-up, and whether your output file was on an actual hard disk.*
- As a matter of programming elegance and style, avoid cut-and-paste coding! *(E.g. the case of reading from standard input vs reading from a file)* The program should be around 100 lines of C code. Programs which are say 200 lines long are inelegant and will be graded accordingly.
- Make sure to consider unusual conditions, such as partial writes, even though you will not necessarily be able to generate these conditions during testing. Will your program handle them correctly?
- Your program must have proper error reporting (what/how/why) as discussed in class and/or lecture notes.
- *Question to ponder: How can you specify an input file which is literally a single hyphen, and not have it be confused with a command-line flag or the special symbol for standard input?*

Submission

Submit (at least) the following, in hard-copy:

- Program source code listing, in black & white, monospaced font.
- Legible screenshot or cut-and-paste of text terminal window commands and responses demonstrating successful run of your program and error condition properly detected and reported.
- Experimental raw data (run times vs buffer sizes)
- Analysis of results. [A paragraph or two will suffice]