Problem 1 -- System calls and error 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.

```
copycat - concatenate and copy files

USAGE:

      copycat [-b ###] [-o outfile] infile1 [...infile2....]
      copycat [-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, copycat 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.

Any of the infiles can be the special name - (a single hyphen).  copycat
will 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.

If no infiles are specified, copycat reads from standard input.

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

EXIT STATUS:

program returns 0 if no errors (opening, reading or writing) 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]
- 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.

- 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 perhaps 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*
- As a matter of programming elegance and style, avoid cut-and-paste coding!
- Make sure to consider unusual conditions, such as partial writes. Will your program handle them correctly?