`copycat`: A Basic Implementation of the UNIX Utility `cat`

by Arthur Christopher Watkins

Department of Electrical Engineering

The Cooper Union for the Advancement of Science & Art
ECE-357: Computer Operating Systems
Prof. Jeff Hakner, Instructor

September 22, 2016

Contents

1 Introduction

The objective of this assignment was to create a "clone" of the UNIX utility **`cat`** called **`copycat`** that provided complete error handling and reporting. The given constraints were that only standard UNIX system calls (`open()`, `read()`, `write()`, and `close()`) could be utilized in the manipulation of the user-specified output and input files, along with any data transfer associated with that file manipulation. More information about specifications, requirements, and usage of the **`copycat`** utility can be found in the **`copycat` Manual Page** section.

Over the course of working on the assignment, knowledge about standard library functionality, system call use, and good programming techniques and conventions was obtained.

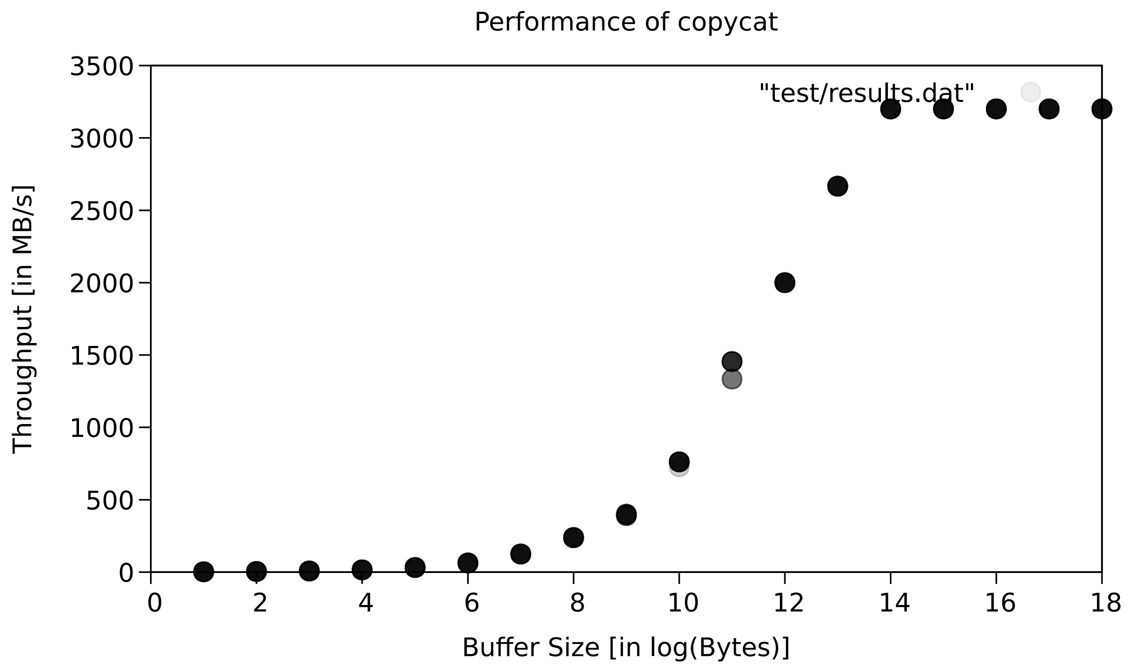
2 **copycat** Manual Page

Listed is the man page for **copycat**, which provides the specifications and usage information for the completed program:

```
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, which is assumed to
already be open.

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

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.
```

3 Performance Analysis

As stated in the **copycat** man page, there exists an optional 'b' argument that can be used to specify the buffer size in bytes. One may notice that, for large data transactions among the input and output files, smaller buffer sizes require more and more `read()` and `write()` system calls to carry out the data transaction in its entirety. Because system calls are so expensive with respect to clock cycles, it suits the miserly user to employ a comparatively larger buffer size so that fewer system calls are made, thus improving performance. Over 36 iterations, **copycat** average throughput in MB/s versus specified buffer size for an input file of size 16MB was tested:

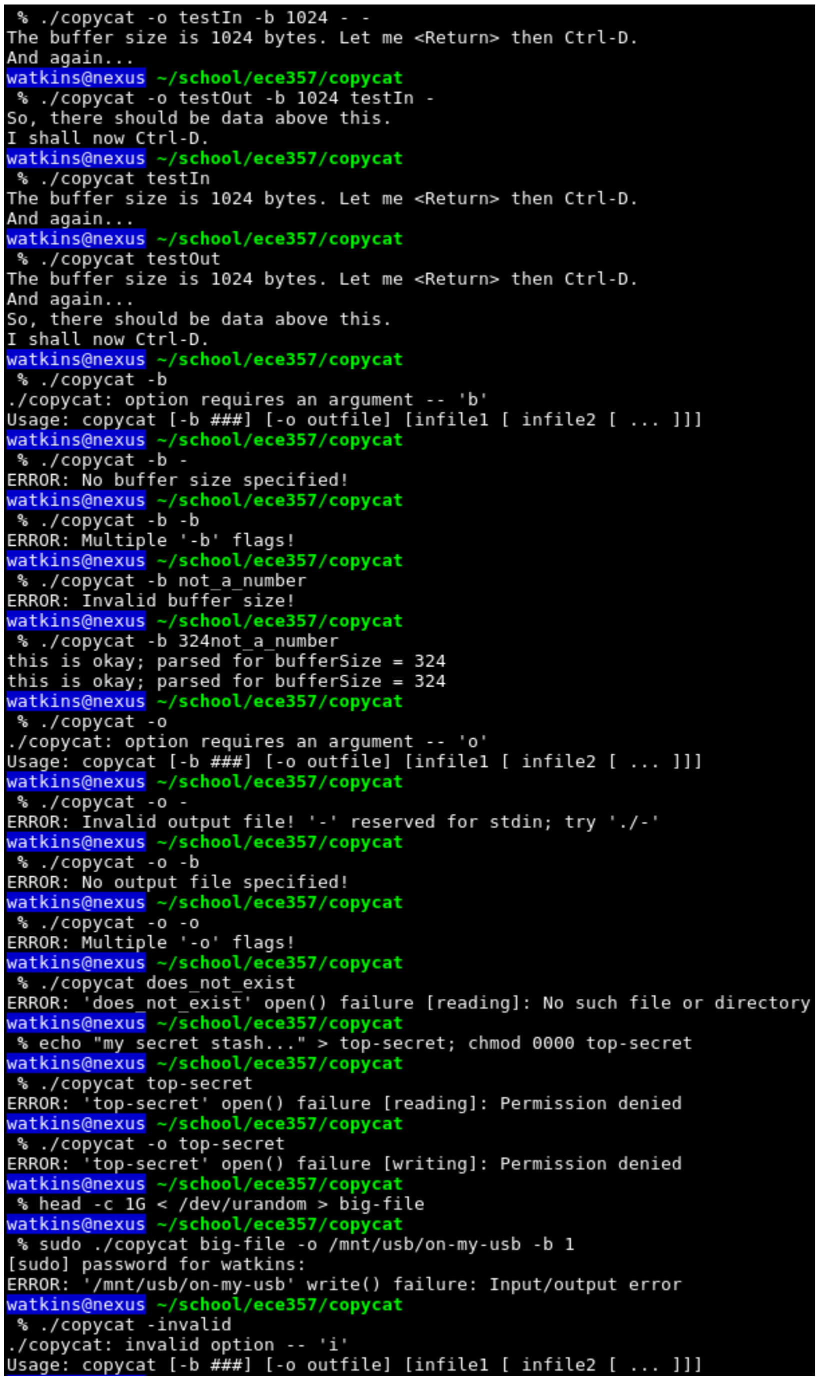


Notice how the plot trends upward as buffer size increases, implying relatively greater throughput for relatively greater buffer sizes. As mentioned earlier, this is due to fewer system calls for larger buffer sizes (which can be verified by examining the **copycat** source code). Also, notice how the plot appears to level after a buffer size of 2^{12} bytes. This result is indicative of the bandwidth/throughput limits of the testing hardware[1].

[1]Tested on ASUS K501UW-NB72 with Intel Core i7 6500U running @ 2.50 GHz, NVIDIA GeForce GTX 960M, 8GB DDR4 RAM, and Samsung EVO 850 500GB SSD. No overclocking or any other modifications besides the 500GB SSD addition.

4 Error Handling

copycat will handle all possible `open()`, `read()`, `write()`, and `close()` errors that arise in file manipulation and data transfer (including partial writes). Below is a screenshot of a terminal[2] documenting the error handling capabilities of **copycat**. Unfortunately, due to the restricted nature of the testing environment, some `read()`, `write()`, and `close()` errors could not be caused using basic commands.



```
 % ./copycat -o testIn -b 1024 - -
The buffer size is 1024 bytes. Let me <Return> then Ctrl-D.
And again...
watkins@nexus ~/school/ece357/copycat
 % ./copycat -o testOut -b 1024 testIn -
So, there should be data above this.
I shall now Ctrl-D.
watkins@nexus ~/school/ece357/copycat
 % ./copycat testIn
The buffer size is 1024 bytes. Let me <Return> then Ctrl-D.
And again...
watkins@nexus ~/school/ece357/copycat
 % ./copycat testOut
The buffer size is 1024 bytes. Let me <Return> then Ctrl-D.
And again...
So, there should be data above this.
I shall now Ctrl-D.
watkins@nexus ~/school/ece357/copycat
 % ./copycat -b
./copycat: option requires an argument -- 'b'
Usage: copycat [-b ###] [-o outfile] [infile1 [ infile2 [ ... ]]]
watkins@nexus ~/school/ece357/copycat
 % ./copycat -b -
ERROR: No buffer size specified!
watkins@nexus ~/school/ece357/copycat
 % ./copycat -b -b
ERROR: Multiple '-b' flags!
watkins@nexus ~/school/ece357/copycat
 % ./copycat -b not_a_number
ERROR: Invalid buffer size!
watkins@nexus ~/school/ece357/copycat
 % ./copycat -b 324not_a_number
this is okay; parsed for bufferSize = 324
this is okay; parsed for bufferSize = 324
watkins@nexus ~/school/ece357/copycat
 % ./copycat -o
./copycat: option requires an argument -- 'o'
Usage: copycat [-b ###] [-o outfile] [infile1 [ infile2 [ ... ]]]
watkins@nexus ~/school/ece357/copycat
 % ./copycat -o -
ERROR: Invalid output file! '-' reserved for stdin; try './-'
watkins@nexus ~/school/ece357/copycat
 % ./copycat -o -b
ERROR: No output file specified!
watkins@nexus ~/school/ece357/copycat
 % ./copycat -o -o
ERROR: Multiple '-o' flags!
watkins@nexus ~/school/ece357/copycat
 % ./copycat does_not_exist
ERROR: 'does_not_exist' open() failure [reading]: No such file or directory
watkins@nexus ~/school/ece357/copycat
 % echo "my secret stash..." > top-secret; chmod 0000 top-secret
watkins@nexus ~/school/ece357/copycat
 % ./copycat top-secret
ERROR: 'top-secret' open() failure [reading]: Permission denied
watkins@nexus ~/school/ece357/copycat
 % ./copycat -o top-secret
ERROR: 'top-secret' open() failure [writing]: Permission denied
watkins@nexus ~/school/ece357/copycat
 % head -c 1G < /dev/urandom > big-file
watkins@nexus ~/school/ece357/copycat
 % sudo ./copycat big-file -o /mnt/usb/on-my-usb -b 1
[sudo] password for watkins:
ERROR: '/mnt/usb/on-my-usb' write() failure: Input/output error
watkins@nexus ~/school/ece357/copycat
 % ./copycat -invalid
./copycat: invalid option -- 'i'
Usage: copycat [-b ###] [-o outfile] [infile1 [ infile2 [ ... ]]]
```

[2]Using Z shell on an Arch Linux distribution.

5 Code Listing

Code Listing 1: copycat.c

```c
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

// abortWith prints error messages and returns -1 status to the parent process of main ()
void abortWith (char * message, char * extended, char * extVar) {
  if (!extended) fprintf (stderr, "%s\n", message);
  else fprintf (stderr, "ERROR:␣’%s’␣%s:␣%s\n", extVar, message, extended);
  exit (-1);
}

// dataTransaction attempts to read from and write to specified opened files
void dataTransaction (int fdOut, int fdIn, char * fOut, char * fIn, char * buf, long * bufSiz) {
  int rdBytes, wrBytes;
  while ((rdBytes = read (fdIn, buf, *bufSiz)) > 0)
    while (1)
      if ((wrBytes = write (fdOut, buf, rdBytes)) <= 0)
        abortWith ("write()␣failure", strerror (errno), fOut);
      else if (wrBytes != rdBytes) {
        buf += wrBytes;
        rdBytes -= wrBytes;
      } else break;
  if (rdBytes < 0) abortWith ("read()␣failure", strerror (errno), fIn);
}

int main (int argc, char ** argv) {
  int fdOut = STDOUT_FILENO, fdIn = -1, bCount = 0, oCount = 0, rdBytes = 0, wrBytes = 0;
  long bufferSize = -1;
  char option = 0, * outputFile = 0;

  // Here, we collect option and option argument information from the argument vector
  while ((option = getopt (argc, argv, "b:o:")) != -1)
    switch (option) {
      case ’b’:
        if (++bCount > 1 || !strcmp ("-b", optarg))
          abortWith ("ERROR:␣Multiple␣’-b’␣flags!", 0, 0);
        else if (!strcmp ("-", optarg) || !strcmp ("-o", optarg))
          abortWith ("ERROR:␣No␣buffer␣size␣specified!", 0, 0);
        else if ((bufferSize = strtol(optarg, NULL, 0)) <= 0)
          abortWith ("ERROR:␣Invalid␣buffer␣size!", 0, 0);
        break;
      case ’o’:
        if (++oCount > 1 || !strcmp ("-o", optarg))
          abortWith ("ERROR:␣Multiple␣’-o’␣flags!", 0, 0);
        else if (!strcmp ("-", optarg))
          abortWith ("ERROR:␣Invalid␣output␣file!␣’-’␣reserved␣for␣stdin;␣try␣’./-’", 0, 0);
        else if (!strcmp ("-b", optarg))
          abortWith ("ERROR:␣No␣output␣file␣specified!", 0, 0);
        outputFile = optarg;
        break;
      default:
        abortWith ("Usage:␣copycat␣[-b␣###]␣[-o␣outfile]␣[infile1␣[␣infile2␣[␣...␣]]]", 0, 0);
    }

  // Here, if an output file was specified, we try to open the file; else, we use stdin as output
  if (!outputFile) outputFile = "stdout";
  else if ((fdOut = open (outputFile, O_WRONLY | O_CREAT, 0666)) < 0)
    abortWith ("open()␣failure␣[writing]", strerror (errno), outputFile);

  // Here, we declare the dataBuffer with given or default buffer size
  if (bufferSize < 0) bufferSize = 4096;
  char dataBuffer[bufferSize];

  // Here, we take care of opening, reading, & closing input files; and writing output files
  for (; optind < argc; ++optind) {
    if (!strcmp ("-", argv[optind])) {
      fdIn = STDIN_FILENO;
      argv[optind] = "stdin";
    }
    else if ((fdIn = open (argv[optind], O_RDONLY)) < 0)
      abortWith ("open()␣failure␣[reading]", strerror (errno), argv[optind]);
    dataTransaction (fdOut, fdIn, outputFile, argv[optind], dataBuffer, &bufferSize);
    if (fdIn != STDIN_FILENO && close (fdIn) < 0)
      abortWith ("close()␣failure", strerror (errno), argv[optind]);
  }

  // Here, if no input file was specified, we use stdin as the input file
  if (fdIn < 0)
    dataTransaction (fdOut, STDIN_FILENO, outputFile, "stdin", dataBuffer, &bufferSize);

  // Here, we try to close the output file if it isn't stdout
  if (fdOut != STDOUT_FILENO && close (fdOut) < 0)
    abortWith ("close()␣failure", strerror (errno), outputFile);

  return 0;
}
```