Donghyun Park

Prof. Hakner

ECE 357

9/22/2017

Problem 1 – System Calls, Error Checking and Reporting

Source Code (121 Lines)

#include <fcntl.h>

#include <stdio.h>

#include <errno.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

//manages error display

int printError(char \*msg, char \*file, char \*detail)

{

    if(!detail)

    {

        fprintf(stderr,"Error: %s\n", msg);

    }

    fprintf(stderr,"Error: %s [%s] - %s \n", msg, file, detail);

    return -1;

}

//manages read/write operations

void doRDWR(int inFD, int outFD, char \*input, char \*output, char\* buffer, int buffersize)

{

    int wrSize, wrBytes;

    while((wrSize = read(inFD, buffer, buffersize)) > 0)

    {

        if(wrSize == -1)

        {

            printError("Failed to read following file to buffer", input, strerror(errno));

        }else{

            while(1)

            {

                if((wrBytes = write(outFD, buffer, wrSize)) <=0)

                {

                    printError("Failed to write following file to output", output, strerror(errno));

                }else if(wrBytes != wrSize){

                    buffer += wrBytes;

                    wrSize -= wrBytes;

                }else{

                    break;

                }

            }

        }

    }

}

int main(int argc, char \*\*argv)

{

    int outFD, option;

    int inFD = -1, bflag = 0, oflag = 0, bufsize = 0;

    char \*outputName = 0;

//gather option arguments from stdin

    while( (option=getopt(argc, argv, "b:o:")) != -1)

    {

        switch (option)

        {

            case 'b':

                if(++bflag > 1 || !strcmp("-b",optarg))

                {

                    printError("Multiple '-b' flags!", 0, 0);

                }else if((bufsize = strtol(optarg, NULL, 0)) <= 0){

                    printError("Not a valid buffer size", 0, 0);

                }else if(!strcmp("-o",optarg) || !strcmp("-", optarg)){

                    printError("No size of buffer mentioned!", 0, 0);

                }

                break;

            case 'o':

                if(++oflag > 1 || !strcmp("-o", optarg))

                {

                    printError("Multiple '-o' flags!", 0, 0);

                }else if(!strcmp("-b",optarg)){

                    printError("No output file specified!", 0 ,0);

                }else if(!strcmp("-",optarg)){

                    printError("- reserved for stdin", 0, 0);

                }

                outputName = optarg;

                break;

            default:

                exit(EXIT\_FAILURE);

        }

    }

    //set default size of buffer and initialize a buffer as bufsize

    if(bufsize <= 0)

    {

        bufsize = 8192;

    }

    char buf[bufsize];

    //if there is outputName = 0, set it to stdout. otherwise, open file with the given name

    if(!outputName)

    {

        outputName = "stdout";

    }else if((outFD = open(outputName, O\_WRONLY | O\_CREAT | O\_TRUNC, 0666)) == -1){

        printError("Failed to open following file for writing", outputName, strerror(errno));

    }

    //if input files were specified go thorugh loop and RD/WR/Close

    for(; optind < argc; ++optind)

    {

        if(!strcmp(argv[optind],"-"))

        {

            argv[optind] = "stdin";

            inFD = STDIN\_FILENO;

        }else if((inFD = open(argv[optind], O\_RDONLY)) == -1){

            printError("Failed to open follwing file for reading", argv[optind], strerror(errno));

        }

        doRDWR(inFD, outFD, argv[optind], outputName, buf, bufsize);

        if(close(inFD) == -1 && inFD != STDIN\_FILENO)

        {

            printError("Failed to close following input file", argv[optind], strerror(errno));

        }

    }

    //if no input files were specified then optind = argc so it skips the for loop and reads from stdin

    if(inFD == -1)

    {

        doRDWR(STDIN\_FILENO, outFD, "stdin", outputName, buf, bufsize);

    }

    //close output file and error check

    if(close(outFD) == -1 && outFD != STDOUT\_FILENO)

    {

        printError("Failed to close following output file", outputName, strerror(errno));

    }

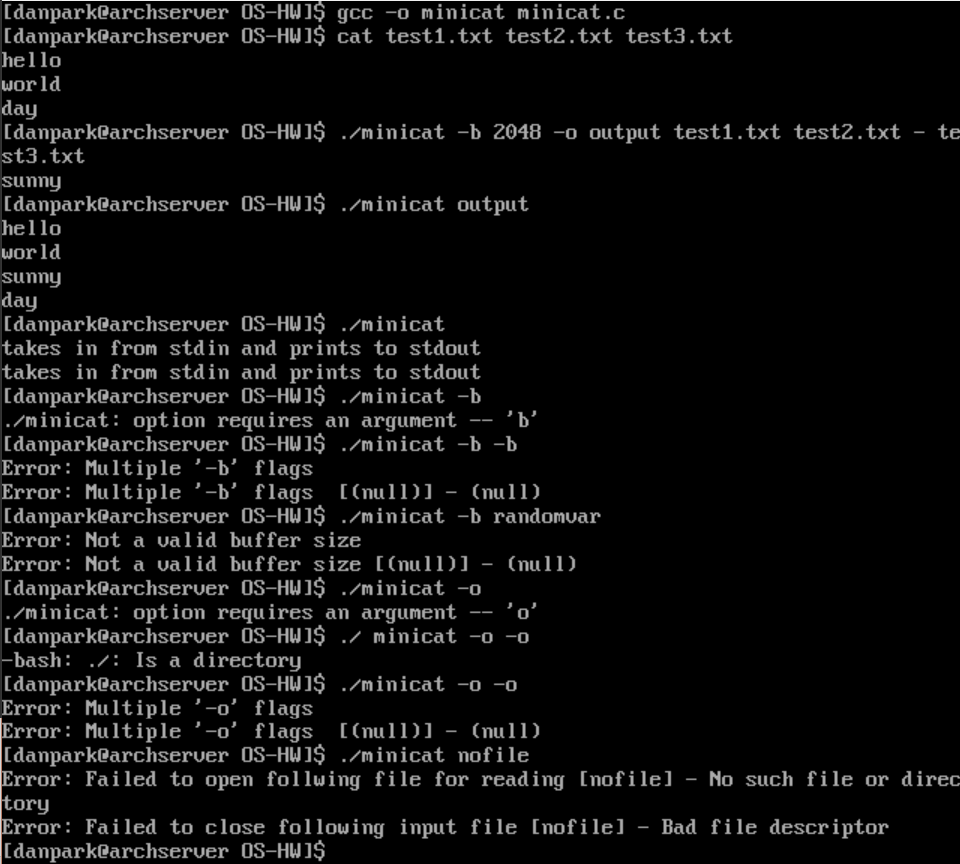
    //there were no errors and program ran successfully

    return 0;

}

Sample Run and Error Handling

Run on ArchLinux VirtualBox, includes initial run and then error handling including all system calls. Not all error handling situations such as partial write errors could be tested due to their difficulty in artificially generating them.



Experimental Raw Data (Run Times vs Buffer Sizes)

For large files and smaller-than-file buffer sizes, it is logical to see that the program would need to cycle through multiple read and write system call loops to be able write the same amount of data that a larger buffer size would be able to in a single loop. By using a sufficiently large buffer, time performance would increase but after a certain point, having a greatly larger buffer not give improved performance due to the hardware limitations. The program was run on a MSI GS60 2PC Ghost, which has Intel i7-4710HQ @ 2.50 GHz, NVIDIA GeForce GTX 860M, 12GB DDR3, Kingston 128 GB SSD. A 12 MB file was generated using:

cat -> file.txt

for i in {1..20}; do cat file.txt file.txt > file2.txt && mv file2.txt file.txt; done

This generated a file with lines in it, duplicating the original two lines of ‘hello’ and world’. The buffer sizes were sampled between 1 byte up to 256K ( = 262,144 bytes) in powers of 2, which means 18 measurements. Time was measured using the following command, where test.txt was the 12 MB file:

time ./minicat -b [buffer size] -o output.txt test.txt

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Buffer Size [bytes]** | **Real Time Elapsed [s]** | **Throughput [MB/s]** |
| 1 | 2 | 6.765 | 1.774 |
| 2 | 4 | 3.398 | 3.531 |
| 3 | 8 | 1.756 | 6.834 |
| 4 | 16 | 0.922 | 13.015 |
| 5 | 32 | 0.445 | 26.966 |
| 6 | 64 | 0.235 | 51.064 |
| 7 | 128 | 0.128 | 93.750 |
| 8 | 256 | 0.075 | 160.000 |
| 9 | 512 | 0.046 | 260.870 |
| 10 | 1024 | 0.038 | 315.789 |
| 11 | 2048 | 0.033 | 363.636 |
| 12 | 4096 | 0.028 | 413.793 |
| 13 | 8192 | 0.027 | 444.444 |
| 14 | 16384 | 0.026 | 461.538 |
| 15 | 32768 | 0.026 | 461.538 |
| 16 | 65536 | 0.027 | 444.444 |
| 17 | 131072 | 0.026 | 461.538 |
| 18 | 262144 | 0.027 | 444.444 |

From the table and chart above, one can tell that after the buffer size of bytes, the throughput plateaus off, showing that the limit of the hardware has been reached and any further increase in the buffer will not give better results.