Nithilam Subbaian
Computer Operating Systems
ECE-357-1
Prof. Hakner
9/20/18

Problem 3 -- Use of system calls in a simple concatenation program

Source Code:

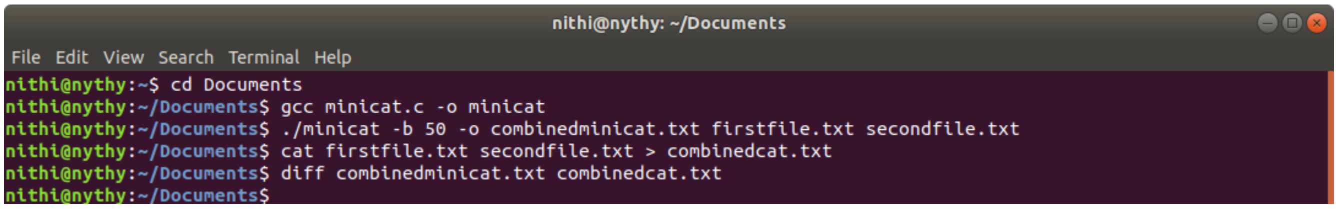
```
#include <stdio.h>
#include <fcntl.h>
#include <ctype.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <getopt.h>
#include <unistd.h>

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

	int buffer_size = 1024;
	int gresult, fdin;
	int fdout = 1;
	int Num_inputs = 0;
	int no_input = 0;
	int outflag = 1;
	char* outfile = "stdout";
	char* infile;
	int m = -1;
	int n= -1;
	int Oseen = 0;
	int Bseen = 0;

	opterr = 0;

	while((gresult = getopt(argc, argv, "b:o:")) != -1) {
		switch(gresult) {
		case 'b':
			buffer_size = atoi(optarg);

			if (Bseen++>0) {
				printf("WARNING: Multiple buffer sizes were entered\n");
			}

			if (buffer_size <1) {
				fprintf(stderr, "ERROR: Invalid buffer size in bytes was entered after the '-
b'\n");
				exit(EXIT_FAILURE);
			}
```

```
                        break;

                case 'o':

                        outfile = optarg;
                        outflag = 0;
                        fdout = 0;

                        if (Oseen++>0) {
                                printf("WARNING: Multiple output files were specified\n");
                        }

                        fdout=open(outfile, O_WRONLY|O_CREAT|O_TRUNC, 0666);
                        if (fdout < 0) {
                                fprintf(stderr, "ERROR: Could not open output file %s for writing due
to %s\n", outfile, strerror(errno));
                                exit(EXIT_FAILURE);
                        }
                        break;

                case '?':
                        fprintf(stderr,"ERROR: Invalid option of '%s' was entered\n", argv[optind-1]);
                        exit(EXIT_FAILURE);
                default:
                        printf("%s\n","ERROR: Invalid arguments entered for getopt function\n" );
                        exit(EXIT_FAILURE);
                }
        }

        char* buffer = malloc(sizeof(char)* buffer_size);

        Num_inputs= argc-optind;

        if (Num_inputs <= 0) {
                Num_inputs=1;
                no_input = 1;
        }


        for (int i = 0; i < Num_inputs; i++) {
                if (no_input == 1) {
                        infile = "-";
                } else{
                        infile = argv[optind + i];
                }

                if ((strcmp(infile,"-")) == 0) {
                        fdin = 0;
                        infile = "stdin";
```

```
		}  else{
			fdin = open(infile, O_RDONLY);
			if(fdin <0) {
				fprintf(stderr, "ERROR: File'%s' could not be open for reading due to '%s' 
\n", infile, strerror(errno));
				exit(EXIT_FAILURE);
			}
		}

		while ((n = read(fdin, buffer, sizeof(char)*buffer_size)) > 0) {
			if (n <0) {
				fprintf(stderr, "ERROR: Could not read from input file '%s' due to %s\n", 
infile, strerror(errno));
				exit(EXIT_FAILURE);
			} else {
				int written = 0;
				while(written < n) {
					if ((m = write(fdout, buffer+written, n))<0) {
						fprintf(stderr, "ERROR: Could not write to output file %s due 
to %s\n", outfile, strerror(errno));
						exit(EXIT_FAILURE);
					}
					written += m;
				}
			}
		}

		if ((fdin != 0) && (close(fdin) < 0)) {
			fprintf(stderr, "ERROR: Could not close input file %s due to %s\n", infile, 
strerror(errno));
			exit(EXIT_FAILURE);

		}
	}

	if ((fdout != 1) && (close(fdout) < 0)) {
		fprintf(stderr, "ERROR: Could not close output file %s due to %s\n", outfile, 
strerror(errno));
		exit(EXIT_FAILURE);
	}

	return(0);

}
```

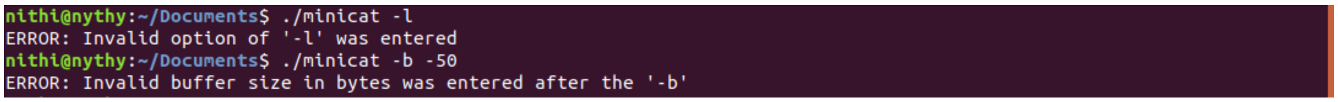
Screenshots of Successful run of Program and Error Reports:

I wrote this program using Atom, and then ran it using the terminal on an Ubuntu operating system. The following screenshots show examples of successful runs of various arguments entered for minicat as well as the error messages for unsuccessful runs. Certain errors were not able to be tested, but those errors, such as partial writes were accounted for in the code.



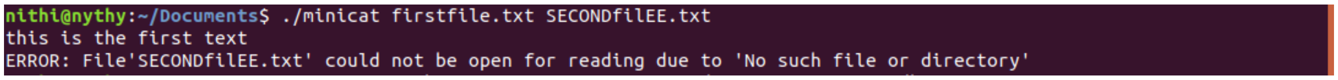
```
nithi@nythy:~$ cd Documents
nithi@nythy:~/Documents$ gcc minicat.c -o minicat
nithi@nythy:~/Documents$ ./minicat -b 50 -o combinedminicat.txt firstfile.txt secondfile.txt
nithi@nythy:~/Documents$ cat firstfile.txt secondfile.txt > combinedcat.txt
nithi@nythy:~/Documents$ diff combinedminicat.txt combinedcat.txt
nithi@nythy:~/Documents$
```

The above screenshot shows a successful run of the program. The minicat file was compiled and run. The minicat program was instructed to run with a buffer size of 50 and combine the files firstfile.txt and secondfile.txt, and output the results into a file called combinedminicat.txt. The same procedure was then implemented with the existing cat command but the results were saved in combinedcat.txt. The diff command was then used to show that there were no differences in output.



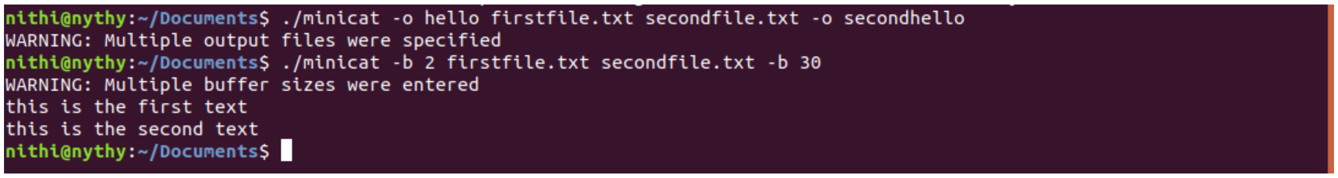
```
nithi@nythy:~/Documents$ ./minicat -l
ERROR: Invalid option of '-l' was entered
nithi@nythy:~/Documents$ ./minicat -b -50
ERROR: Invalid buffer size in bytes was entered after the '-b'
```

In the above screenshot, an invalid option was entered as only the options '-b' and '-o' can be entered. As '-l' is not a valid option, the corresponding error was returned. Also in the screenshot you can see an invalid buffer size entered, as there can be no negative buffer size, and so the corresponding error was reported.



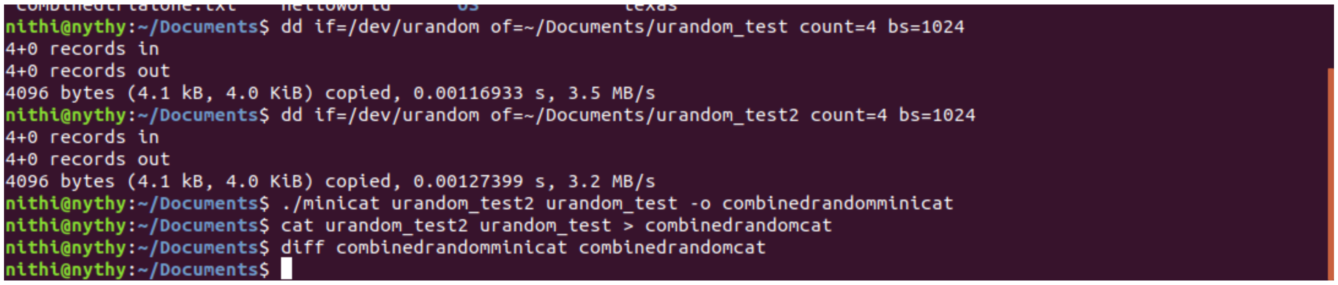
```
nithi@nythy:~/Documents$ ./minicat firstfile.txt SECONDfilEE.txt
this is the first text
ERROR: File'SECONDfilEE.txt' could not be open for reading due to 'No such file or directory'
```

In the above screenshot the error was generated as the file SECONfilEE.txt does not exist, and therefore only the contents of firstfile.txt were shown on standard output, while the ERROR was brought up for the other file.



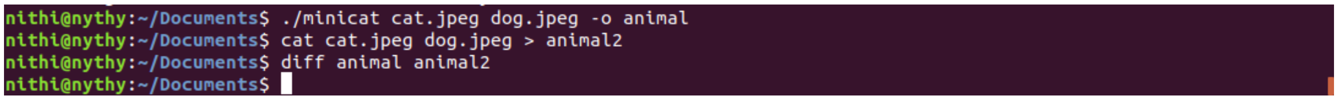
```
nithi@nythy:~/Documents$ ./minicat -o hello firstfile.txt secondfile.txt -o secondhello
WARNING: Multiple output files were specified
nithi@nythy:~/Documents$ ./minicat -b 2 firstfile.txt secondfile.txt -b 30
WARNING: Multiple buffer sizes were entered
this is the first text
this is the second text
nithi@nythy:~/Documents$ █
```

In the above screenshot there are two examples of warning messages. These are warning messages because the program is still able to combine the specified files, but it warns the user, if they are unaware that multiple entries were entered for the same option. In the first line, there were two specified output files, and in the third line, there were two specified buffer sizes. As you can see, when the output file was not specified, the program still completes its function by combining the files.



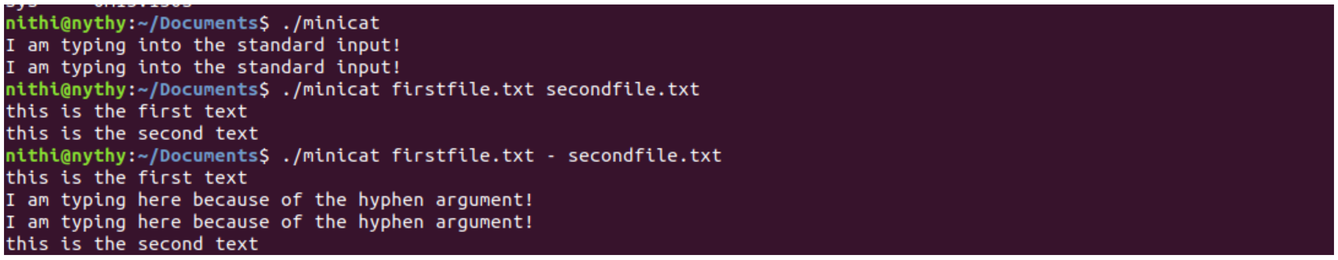
```
nithi@nythy:~/Documents$ dd if=/dev/urandom of=~/Documents/urandom_test count=4 bs=1024
4+0 records in
4+0 records out
4096 bytes (4.1 kB, 4.0 KiB) copied, 0.00116933 s, 3.5 MB/s
nithi@nythy:~/Documents$ dd if=/dev/urandom of=~/Documents/urandom_test2 count=4 bs=1024
4+0 records in
4+0 records out
4096 bytes (4.1 kB, 4.0 KiB) copied, 0.00127399 s, 3.2 MB/s
nithi@nythy:~/Documents$ ./minicat urandom_test2 urandom_test -o combinedrandomminicat
nithi@nythy:~/Documents$ cat urandom_test2 urandom_test > combinedrandomcat
nithi@nythy:~/Documents$ diff combinedrandomminicat combinedrandomcat
nithi@nythy:~/Documents$ 
```

In the above screenshot, there are two files created that are each 4.1K big, and contain random bytes. The minicat program is shown in the screenshot to combine the two files the same way that the cat program combines the two files.



```
nithi@nythy:~/Documents$ ./minicat cat.jpeg dog.jpeg -o animal
nithi@nythy:~/Documents$ cat cat.jpeg dog.jpeg > animal2
nithi@nythy:~/Documents$ diff animal animal2
nithi@nythy:~/Documents$ 
```

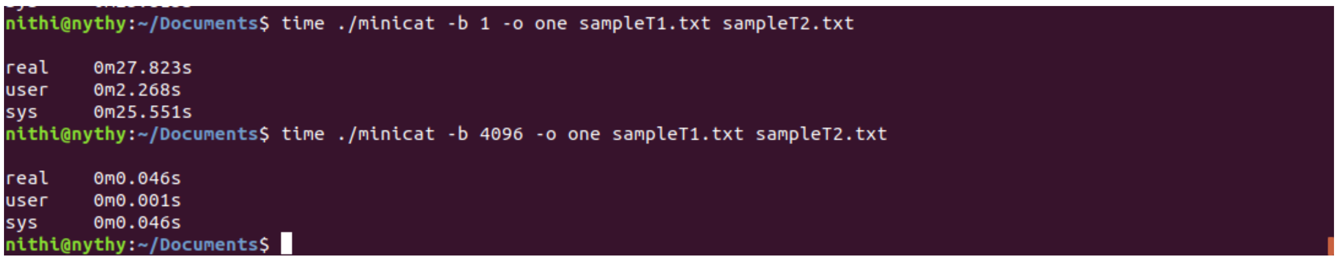
In the above screenshot, two jpeg images are combined to produce the same result through the minicat program and the cat command.



```
nithi@nythy:~/Documents$ ./minicat
I am typing into the standard input!
I am typing into the standard input!
nithi@nythy:~/Documents$ ./minicat firstfile.txt secondfile.txt
this is the first text
this is the second text
nithi@nythy:~/Documents$ ./minicat firstfile.txt - secondfile.txt
this is the first text
I am typing here because of the hyphen argument!
I am typing here because of the hyphen argument!
this is the second text
```

In the above screenshot, the first run shows how no input arguments for getopt() goes to the standard input, where I then typed in the phrase “I am typing into the standard input!”, and that phrase was sent to standard output. In the next run, when no output file was specified, the contents were shown on the standard output. In the last run in the screenshot, it is an example of the minicat program being run and the hyphen being using as a means to enter standard input, and then revert back to the next file in the argument list.

Experimental Raw Data:



```
nithi@nythy:~/Documents$ time ./minicat -b 1 -o one sampleT1.txt sampleT2.txt

real	0m27.823s
user	0m2.268s
sys	0m25.551s
nithi@nythy:~/Documents$ time ./minicat -b 4096 -o one sampleT1.txt sampleT2.txt

real	0m0.046s
user	0m0.001s
sys	0m0.046s
nithi@nythy:~/Documents$ 
```

The time shell command was used to measure the run time of the program. In the data table, the real time is used. In reality, all of these times vary from one execution to another as the time is affected by other factors such as how many other processes are running in the CPU.

Buffer Size	**Run Time**
1	27.823s
2	4.271s
4	7.474s
8	3.850s
16	1.902s
32	1.010s
64	0.532s
128	0.248s
256	0.137s
512	0.120s
1024	0.069s
2048	0.058s
4096	0.046s

The two files used in testing the run times are sampleT1.txt and sampleT2.txt, which are both 4.2MB. It was important to use a file this big, otherwise the run times of the program were too similar to compare for small and large buffer sizes.

In the table, it is clear that as the buffer size is increased, the run time decreases. This makes sense as when the buffer size is small, processes such as the read/write system calls have to be executed more often. When the buffer size is made larger, more characters can be loaded in one step, and therefore fewer system calls are needed, decreasing the time, although more memory is used.

Question to Ponder:
You can specify an input file which is literally a single hyphen and not have it confused with a command-line flag or the special symbol for standard input by also inputting the file path in front of it. If the file is in the current directory, then ./- will be enough to indicate that it is not a command-line flag or the special symbol for standard input, and rather, that it is a file.