**ECE 357: Problem 1 -- System Calls, Error Checking and Reporting**

1. **Source Code**

#include <stdio.h>

#include <fcntl.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

int main(int argc, char \*argv[]){

//file descriptors for output, read, write, and open operations

int fd, wr, rd, op;

//store optcode result

int opt;

//default buffer size

int bytes = 0x100;

//default output: stdout if output\_file\_name = NULL

char \*output\_file\_name = NULL;

//gets optcode and sets them for analysis

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

switch( opt ){

case 'b':

bytes = strtol(optarg, (char \*\*) NULL, 10);

if(bytes <= 0) {

fprintf(stderr, "Buffer size has to be a positive integer: %d\n", bytes);

return -1;

}

//should add error handing for case where wrong parameter (non-integer) is passed

break;

case 'o':

output\_file\_name = optarg;

break;

case '?':

if(optopt == 'b' || optopt == 'o') fprintf(stderr, "Missing argument from -%c\n", optopt);

default:

return 1;

}

}

//if no ouput file is specified, output result to terminal

fd = (output\_file\_name == NULL ? STDOUT\_FILENO:open(output\_file\_name, O\_WRONLY | O\_CREAT | O\_TRUNC);

//initialize buffer to specified size or otherwise to default size

char buf[bytes];

//read in rest of the arguments to output

for(int i = optind; i < argc; i++){

op = open(argv[i], O\_RDONLY);

if(op < 0){

fprintf(stderr, "Failed to open %s: %s\n", argv[i], strerror(errno));

return -1;

}

while((rd = read(op, buf, bytes)) != 0){

if(rd < 0){

fprintf(stderr, "Failed to read %s: %s\n", argv[i], strerror(errno));

return -1;

}

if(wr = write(fd,buf,rd) < 0){

fprintf(stderr, "Failed to write to %s: %s\n", output\_file\_name, strerror(errno));

return -1;

} else {

//partial rewrite

if(wr > 0 && wr < rd) write(fd, &buf[wr], rd-wr);

}

}

if(close(op) < 0){

fprintf(stderr, "Failed to close %s: %s\n", argv[i], strerror(errno));

return -1;

}

}

//no infile, read from stdin

if(argv[optind] == NULL){

while(rd = r­ead(STDIN\_FILENO, buf, bytes) > 0) wr = write(fd, buf, strlen(buf));

}

if(close(fd) < 0){

fprintf(stderr, "Failed to close %s: %s\n", output\_file\_name, strerror(errno));

return -1;

}

//no errors found until the end of file

return 0;

}

1. **Experimental Raw Data**

File Size: 280608 Bytes

|  |  |  |
| --- | --- | --- |
| **Trial** | **Buffer Size (Bytes)** | **Time (s)** |
| 1 | 1 | 4.316 |
| 2 | 2 | 2.143 |
| 3 | 4 | 1.143 |
| 4 | 8 | 0.579 |
| 5 | 16 | 0.284 |
| 6 | 32 | 0.166 |
| 7 | 64 | 0.121 |
| 8 | 128 | 0.073 |
| 9 | 256 | 0.056 |
| 10 | 512 | 0.050 |
| 11 | 1024 | 0.046 |
| 12 | 2048 | 0.042 |
|  | … | … |
| 19 | 262144 (=256kB) | 0.041 |

*Table 1. Program Run-time for different Buffer Sizes.*

1. **Analysis of results**

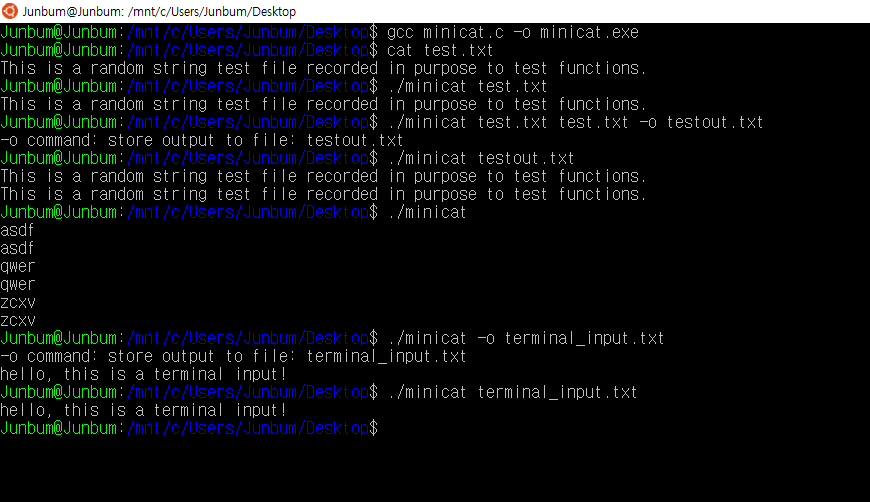
From the times collected in Table 1, one can infer that the time decays exponentially as we increase buffer size. The program run-time asymptotes to 0.041 seconds. For buffer size 1 -16, one can clearly see that the time halves when buffer size is doubled. This is because the number of syscall to read, open, and write operation halves as buffer size doubles. The value asymptotes to 0.041 second as other runtime factor such as read and write operation time becomes dominant.

On the following page, there is a screenshot of a sample run and a screenshot of error reports. All modes of operation work perfectly as intended. For demonstrative purposes, the sample run shows how minicat does the same operation as cat does. It also shows how minicat takes input from the terminal and either writes it to a file or the terminal line-by-line. The program also does an adequate job for the error report. One interesting note: when

./minicat -o -b test.txt

ran, I thought it would give an error because neither -o nor -b was given a parameter, but the program took -b as a parameter for -o and wrote the content of test.txt into a new file named -b.

1. **Screenshot of a Sample Run**



1. **Screenshot of Error Reports**

