Yacine Manseur

09/15/2015

Operating Systems

Problem Set 1

**System Information:**

Processor: Intel® Core™ i7-2670QM CPU @ 2.20GHz

OS: 64-bit Windows 7 Home Premium

RAM: 6.00GB

Storage: 279 GB HDD

**Examples of Failure Detection:**

To compile the program, the following was typed into the console:

Yacine@Yacine-Laptop ~/OS/OS/PS1

$ gcc –o copycat copycat.c

Example of incorrect argument for buffer size:

Yacine@Yacine-Laptop ~/OS/OS/PS1

$ ./copycat.exe -b v

Usage: incorrect argument. Must be an integer greater than 0.

Example of missing argument for buffer size:

Yacine@Yacine-Laptop ~/OS/OS/PS1

$ ./copycat.exe -b

copycat: option requires an argument -- b

Usage: -b missing argument.

Example of missing argument for output file:

Yacine@Yacine-Laptop ~/OS/OS/PS1

$ ./copycat.exe -o

copycat: option requires an argument -- o

Usage: -o missing argument.

Example of inputting a file that does not exist:

Yacine@Yacine-Laptop ~/OS/OS/PS1

$ ./copycat.exe -o out fileThatDoesNotExist

-1

Cannot open input file: No such file or directory

**Experimental Results:**

The throughput of this program was measured at changing buffer sizes. The time was found by adding the User Time with the System Time to get a measurement of the actual time spend doing useful work. A file of size 1,025,570 Bytes was created for the purpose of testing how long the actual time of the program would be at different buffer sizes. Below is the data recorded for this experiment:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Buffer Size | Total Bytes Processed | User Time | Sys Time | Actual Time (User + Sys) | Bytes/Sec | MB/Sec |
| 1 | 1,025,570 | 0.639 | 8.158 | 8.797 | 116581.8 | 0.116582 |
| 2 | 1,025,570 | 0.436 | 3.946 | 4.382 | 234041.5 | 0.234042 |
| 4 | 1,025,570 | 0.202 | 2.059 | 2.261 | 453591.3 | 0.453591 |
| 8 | 1,025,570 | 0.093 | 1.076 | 1.169 | 877305.4 | 0.877305 |
| 16 | 1,025,570 | 0.061 | 0.577 | 0.638 | 1607476 | 1.607476 |
| 32 | 1,025,570 | 0.046 | 0.342 | 0.388 | 2643222 | 2.643222 |
| 64 | 1,025,570 | 0.015 | 0.187 | 0.202 | 5077079 | 5.077079 |
| 128 | 1,025,570 | 0.015 | 0.14 | 0.155 | 6616581 | 6.616581 |
| 256 | 1,025,570 | 0.03 | 0.093 | 0.123 | 8337967 | 8.337967 |
| 512 | 1,025,570 | 0.015 | 0.077 | 0.092 | 11147500 | 11.1475 |
| 1024 | 1,025,570 | 0.015 | 0.061 | 0.076 | 13494342 | 13.49434 |
| 2048 | 1,025,570 | 0.015 | 0.062 | 0.077 | 13319091 | 13.31909 |
| 4096 | 1,025,570 | 0.03 | 0.046 | 0.076 | 13494342 | 13.49434 |
| 8192 | 1,025,570 | 0.015 | 0.062 | 0.077 | 13319091 | 13.31909 |
| 16384 | 1,025,570 | 0.015 | 0.061 | 0.076 | 13494342 | 13.49434 |
| 32768 | 1,025,570 | 0.015 | 0.061 | 0.076 | 13494342 | 13.49434 |
| 65536 | 1,025,570 | 0 | 0.077 | 0.077 | 13319091 | 13.31909 |
| 131072 | 1,025,570 | 0.015 | 0.061 | 0.076 | 13494342 | 13.49434 |
| 262144 | 1,025,570 | 0.015 | 0.061 | 0.076 | 13494342 | 13.49434 |

Below is a graph to help visualize the data collected:

As you can see in the graph above, the size of the buffer has a direct effect on the compile time at lower sizes when multiple read/writes have to occur to finish the operation. This makes sense because multiple read/writes should impact total runtime. Notice that after 210 bytes, the throughput levels off. This is because the buffer size is larger than the file itself. This means that no matter how much larger the buffer size gets, only one read/write occurs.

**Appendix:**

// Yacine Manseur

// Cooper Union Fall 2015

// ECE 357: Operating Systems

// Problem Set 1

// copycat.c

#include <fcntl.h>

#include <errno.h>

#include <fcntl.h>

#include <unistd.h>

#include <stdio.h>

#include <sys/stat.h>

#include <stdlib.h>

#include <string.h>

int processFiles(int iFD, int oFD, int bufferSize)

{

int bytesRead = 0, bytesWrite = 0, bytesMissed = 0;

char \*buffer = malloc (bufferSize);

// Check to ensure success with allocating memory

if (buffer == NULL)

{

fprintf(stderr, "Cannot allocate buffer with size: %d\n", bufferSize);

return -1;

}

// Loop until end of file

while ((bytesRead = read(iFD, buffer, bufferSize)) > 0)

{

bytesWrite = write(oFD, buffer, bytesRead);

while (bytesWrite != bytesRead) //Partial write

{

bytesMissed = bytesRead - bytesWrite;

// write remaining bytes

bytesWrite += write(oFD, buffer+bytesWrite, bytesMissed);

}

}

// Error when trying to read the input file

if (bytesRead < 0)

{

fprintf(stderr, "Cannot read file: %s\n", strerror(errno));

return -1;

}

return 0;

}

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

{

int opt = -1;

int bufferSize = 4096;

char\* outfile = NULL;

int numFiles = 0;

// Default input file descriptor is stdin

// Default output file descriptor is stdout

int iFD = 0, oFD = 1;

int ii;

// Read Command Line Arguments

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

{

switch (opt)

{

// Manually assign buffer size

case 'b':

if(atoi(optarg) == 0)

{

fprintf(stderr, "Usage: incorrect argument. Must be an integer greater than 0.\n");

return -1;

}

else

bufferSize = atoi(optarg);

break;

// Initialize output filename

case 'o':

outfile = optarg;

break;

// if getopt() does not recognize an option character

case '?':

if (optopt == 'b' || optopt == 'o')

fprintf(stderr, "Usage: -%c missing argument.\n", optopt);

else

fprintf(stderr, "Usage: %s [-b ###] [-o outfile] infile1 [...infile2....]\n", argv[0]);

return -1;

break;

// Just in case

default:

fprintf(stderr, "Usage: %s [-b ###] [-o outfile] infile1 [...infile2....]\n", argv[0]);

return-1;

}

}

// Initialize array of input files

char \*infile[argc-optind];

while(optind < argc)

{

infile[numFiles] = argv[optind];

numFiles++;

optind++;

}

if (outfile != NULL)

{

oFD = open(outfile, O\_WRONLY | O\_CREAT | O\_TRUNC, 0777);

if (oFD == -1)

{

fprintf(stderr, "Cannot open output file: %s\n", strerror(errno));

return -1;

}

}

else

oFD = 1;

if (numFiles > 0)

{

for (ii = 0; ii < numFiles; ii++)

{

if(\*infile[ii] != '-')

{

iFD = open(infile[ii], O\_RDONLY);

if (iFD == -1)

{

fprintf(stderr, "Cannot open input file: %s\n", strerror(errno));

return -1;

}

}

else

{

//set iFD to stdin

iFD = 0;

}

if (processFiles(iFD, oFD, bufferSize) == -1)

{

// Error was already reported. Terminate the program.

return -1;

}

// close input file descriptor unless it is stdin

if(iFD != 0)

{

if (close(iFD) == -1)

{

fprintf(stderr, "Cannot close input file: %s\n", strerror(errno));

return -1;

}

}

}

}

else

{

if (processFiles(iFD, oFD, bufferSize) == -1)

{

// Error was already reported. Terminate the program.

return -1;

}

}

// Close the output file and end the program

if (close(oFD) == -1)

{

fprintf(stderr, "Cannot close output file: %s\n", strerror(errno));

return -1;

}

return 0;

}