**CIS 452 Program 2**

Streamed Vector Processing

Gaelen McIntee

**Program Design**

vector takes arguments in the form of “vector <number\_size> <vector\_size> <vector\_A> <vector\_B> <out\_file>”. When my program launches, its first task is to set up the process tree and pipes and open up the input and output files. Error checking is performed for all of these operations, and the program quits if any of them fail, producing an error message. A signal handler is also installed for the SIGINT signal before forking so that each process can unpause after a “^C” is entered. After each process is forked, they rename their assigned ends of the pipe and close all unneeded references to the pipe. After everything has been set up, each process pauses and waits for a “^C”. From here, each process is executing separate code.

The complementer starts by reading each character one by one from the file *vector\_B* and flips the bit that it reads. If it encounters a newline character, it skips that character. After reading in *number\_size* bits, it will write the complemented version of the entire number to the pipe connected to the Incrementer.

Before doing anything else, the Incrementer makes a bit string with the value of 1, and adds zeros in front of it within the bounds of *number\_size*. It then reads the complemented bit number from the Complementer and calls add() with the complemented number and the number of value 1 as its arguments, which increments the value. It then sends its incremented value through the pipe connected to the Adder, producing the 2’s complement of the original number.

The add() function checks through all possible combinations of the bits at position *i* of the bit strings *a* and *b*, while also considering a carry flag. The output bit string at position *i* is set according to which case was triggered. The entire bit string is saved in the value passed into *y*. The function returns 0 when successful, and -1 on error.

The Adder then reads the pipe for the 2’s complemented number and saves it. It then reads the file *vector\_A* one character at a time and saves it. A call to add() is made with the number from *vector\_A* and the 2’s complemented number, effectively performing subtraction. The resulting number is then saved to the file *out\_file*.

See source code on next page.

**Code**

/\*

CIS 452 Program 2 - Streamed Vector Processing

Gaelen McIntee

2/15/2017

Usage: vector <number\_size> <vector\_size> <vector\_A> <vector\_B> <out\_file>

This program takes two files with identical number\_size and vector\_size and

subtracts the numbers in vector\_B from vector\_A as a vector processor would,

using pipes. The result is stored in out\_file.

\*/

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

#include <string.h>

#include <sys/types.h>

/\* Preset read/write values for readable code \*/

#define READ 0

#define WRITE 1

#define VECTOR\_SIZE\_MAX 500

/\* Catches the ^C signal to unpause the program. \*/

void unpause(int signum);

/\* Adds two bit strings a and b and stroes the result in y. \*/

int add(char const \*a, char const \*b, char \*y);

/\* We know how much input we're working with \*/

int numberSize, vectorSize;

/\* PID's for the incrementer and adder \*/

pid\_t increm = -1, adder = -1;

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

{

/\* initial file descriptors for the pipes \*/

int fd\_compl\_increm[2], fd\_increm\_adder[2];

FILE \*a, \*b, \*out;

/\* All processes get this signal installed to unpause later \*/

signal(SIGINT, unpause);

/\* Capture command line arguments w/ error checking ––––––––––––––––– \*/

/\* must have 6 arguments \*/

if (argc < 6) {

fprintf(stderr, "Usage: vector <number\_size> <vector\_size> <vector\_A> <vector\_B> <out\_file>\n");

exit(1);

}

/\* retrieve our number\_size and vector\_size \*/

numberSize = atoi(argv[1]);

vectorSize = atoi(argv[2]);

if (vectorSize > VECTOR\_SIZE\_MAX) {

fprintf(stderr, "Vector size too big, must be <=500.\n");

exit(1);

}

fprintf(stderr, "Initializing vector processor with Number size %i and Vector size %i\n", numberSize, vectorSize);

/\* All files must open successfully to continue \*/

a = fopen(argv[3], "r");

b = fopen(argv[4], "r");

out = fopen(argv[5], "w");

if (a == NULL || b == NULL || out == NULL) {

perror("File open error");

exit(1);

}

fprintf(stderr, "Opened input files %s and %s.\n", argv[3], argv[4]);

/\* Pipe and Fork setup ––––––––––––––––––––––––––––––––––––––––––––– \*/

if ( pipe(fd\_compl\_increm) < 0 || pipe(fd\_increm\_adder) < 0 ) {

perror("Piping error");

exit(1);

}

fprintf(stderr, "Pipes created.\n");

/\* The parent (Complementer) creates the Incrementer \*/

increm = fork();

if (increm < 0) {

perror("Fork error");

exit(1);

}

if (increm > 0)

fprintf(stderr, "Incrementer child process created with PID %i.\n", increm);

/\* The Incrementer creates the Adder (Complementer is blocked from executing this) \*/

if (!increm)

{

adder = fork();

if (adder < 0) {

perror("Fork error");

exit(1);

}

if (adder > 0)

fprintf(stderr, "Adder child process created with PID %i.\n", adder);

}

/\* Processes now execute their independent code \*/

/\* Complementer (if our child process is the Incrementer) \*/

if (increm > 0)

{

/\* Redirect our stdout to the write end of the pipe connected to

the Incrementer, and close all other references. \*/

dup2(fd\_compl\_increm[WRITE], STDOUT\_FILENO);

close(fd\_compl\_increm[READ]);

close(fd\_compl\_increm[WRITE]);

close(fd\_increm\_adder[READ]);

close(fd\_increm\_adder[WRITE]);

fprintf(stderr, "Complementer: pipes redirected.\n");

pause();

char temp, complement[numberSize];

/\* Loop through each bit string in the file, and get each bit

one by one. Flip each bit that comes in. If there's a newline,

ignore it. \*/

for (int j = 0; j < vectorSize; j++)

{

int i;

for (i = 0; i < numberSize; i++)

{

temp = fgetc(b);

switch (temp) {

case '0':

temp = '1'; break;

case '1':

temp = '0'; break;

case '\n':

i--; break; // Prevent loop from moving ahead if a newline is encountered.

}

complement[i] = temp;

}

complement[i] = '\0'; // Null terminate

fprintf(stderr, "Complementer: complement is %s\n", complement);

/\* Send the complemented bit string through the pipe connected

to the Incrementer \*/

write(STDOUT\_FILENO, complement, strlen(complement)+1);

}

}

/\* Incrementer (if our child process is the Adder) \*/

else if (adder > 0)

{

/\* Redirect our stdin to the read end of the pipe connected to

the Complementer, redirect stdout to the write end of the pipe

connected to the Adder. Close all other references. \*/

dup2(fd\_compl\_increm[READ], STDIN\_FILENO);

dup2(fd\_increm\_adder[WRITE], STDOUT\_FILENO);

close(fd\_compl\_increm[READ]);

close(fd\_compl\_increm[WRITE]);

close(fd\_increm\_adder[READ]);

close(fd\_increm\_adder[WRITE]);

fprintf(stderr, "Incrementer: redirected pipes.\n");

pause();

/\* Before doing anything, make a bit string of the value 1 so

we can use it to increment the bit string. \*/

char compl[numberSize], one[numberSize], inc[numberSize];

for (int i = 0; i < numberSize; i++) {

if (i == numberSize-1)

one[i] = '1';

else

one[i] = '0';

}

one[numberSize] = '\0';

fprintf(stderr, "Incrementer: Bit string %s created.\n", one);

/\* Now we loop through the entire vector, reading in the value

sent to us from the Complementer and calling add() with on it

with our bit string of value 1, effectively incrementing it.

to get the 2's complement. \*/

for (int j = 0; j < vectorSize; j++)

{

read(STDIN\_FILENO, compl, numberSize+1); // read pipe

fprintf(stderr, "Incrementer: read in %s from Complementer.\n", compl);

add(compl, one, inc); // Do "compl + 1" and save it in inc

fprintf(stderr, "Incrementer: %s incremented by 1 is %s\n", compl, inc);

/\* Send the incremented value (2's comp) through the pipe connected

to the Adder. \*/

write(STDOUT\_FILENO, inc, strlen(inc)+1);

}

}

/\* Adder (if we have no children) \*/

else

{

/\* Redirect our stdin to the read end of the pipe connected to

the Incrementer, and close all other references. \*/

dup2(fd\_increm\_adder[READ], STDIN\_FILENO);

close(fd\_compl\_increm[READ]);

close(fd\_compl\_increm[WRITE]);

close(fd\_increm\_adder[READ]);

close(fd\_increm\_adder[WRITE]);

fprintf(stderr, "Adder: redirected pipes.\n");

pause();

char temp, num\_A[numberSize], num\_B[numberSize], output[numberSize];

int i;

/\* Loop through the entire vector, first reading in the 2's comp

given from the Incrementer, then reading in the corresponding value

from the vector in vector\_A one character at a time. Newlines are

ignored in the read. After both bit strings have been read, call add()

on both of them to perform subtraction and then write the result to out\_file. \*/

for (int j = 0; j < vectorSize; j++)

{

read(STDIN\_FILENO, num\_B, numberSize+1);

fprintf(stderr, "Adder: read in %s from Incrementer.\n", num\_B);

for (i = 0; i < numberSize; i++)

{

temp = fgetc(a);

if (temp == '\n')

i--; // ignore newline

num\_A[i] = temp;

}

num\_A[i] = '\0'; // null terminate

add(num\_A, num\_B, output); // Do "num\_A + num\_B" and save it in output

/\* Report our answer and send it to the file with a newline. \*/

fprintf(stderr, "Adder: %s added to %s is %s\n", num\_A, num\_B, output);

fputs(output, out);

fputs("\n", out);

}

fprintf(stderr, "Vector subtraction complete.\n"); // we're done

}

return 0;

}

/\* The add() function checks through all possible combinations of

the bits at position i of the bit strings a and b, while also

considering a carry flag. The output bit string at position i

is set according to which case was triggered. \*/

int add(char const \*a, char const \*b, char \*y)

{

int carry = 0;

for (int i = (numberSize-1); i >= 0; i--)

{

if (a[i]=='0' && b[i]=='0' && carry==0)

{ y[i] = '0'; carry = 0; }

else if (a[i]=='1' && b[i]=='0' && carry==0)

{ y[i] = '1'; carry = 0; }

else if (a[i]=='0' && b[i]=='1' && carry==0)

{ y[i] = '1'; carry = 0; }

else if (a[i]=='0' && b[i]=='0' && carry==1)

{ y[i] = '1'; carry = 0; }

else if (a[i]=='1' && b[i]=='0' && carry==1)

{ y[i] = '0'; carry = 1; }

else if (a[i]=='0' && b[i]=='1' && carry==1)

{ y[i] = '0'; carry = 1; }

else if (a[i]=='1' && b[i]=='1' && carry==1)

{ y[i] = '1'; carry = 1; }

else if (a[i]=='1' && b[i]=='1' && carry==0)

{ y[i] = '0'; carry = 1; }

else {

fprintf(stderr, "Unrecognized pattern in add(). a=%s, b=%s, y=%s. PID: %i\n", a, b, y, getpid());

return -1;

}

}

y[numberSize] = '\0'; // Null terminate

return 0;

}

/\* Report that we're unpaused and return. \*/

void unpause(int signum)

{

if (increm > 0)

fprintf(stderr, " Program unpaused.\n\n\n");

return;

}

**Test Run**

[~/cis452/prg2-vectors]

@dc20 -> ./vector 8 8 inputA inputB out

Initializing vector processor with Number size 8 and Vector size 8

Opened input files inputA and inputB.

Pipes created.

Incrementer child process created with PID 3111.

Complementer: pipes redirected.

Adder child process created with PID 3112.

Adder: redirected pipes.

Incrementer: redirected pipes.

^CIncrementer: Bit string 00000001 created.

Program unpaused.

Complementer: complement is 11011011

Complementer: complement is 11010011

Complementer: complement is 00010010

Complementer: complement is 00001001

Complementer: complement is 11110010

Complementer: complement is 11011111

Complementer: complement is 00100111

Complementer: complement is 00010101

Incrementer: read in 11011011 from Complementer.

Incrementer: 11011011 incremented by 1 is 11011100

Incrementer: read in 11010011 from Complementer.

Incrementer: 11010011 incremented by 1 is 11010100

Incrementer: read in 00010010 from Complementer.

Incrementer: 00010010 incremented by 1 is 00010011

Incrementer: read in 00001001 from Complementer.

Incrementer: 00001001 incremented by 1 is 00001010

Incrementer: read in 11110010 from Complementer.

Incrementer: 11110010 incremented by 1 is 11110011

Incrementer: read in 11011111 from Complementer.

Incrementer: 11011111 incremented by 1 is 11100000

Incrementer: read in 00100111 from Complementer.

Incrementer: 00100111 incremented by 1 is 00101000

Incrementer: read in 00010101 from Complementer.

Incrementer: 00010101 incremented by 1 is 00010110

[~/cis452/prg2-vectors]

@dc20 -> Adder: read in 11011100 from Incrementer.

Adder: 01100100 added to 11011100 is 01000000

Adder: read in 11010100 from Incrementer.

Adder: 01000010 added to 11010100 is 00010110

Adder: read in 00010011 from Incrementer.

Adder: 00101101 added to 00010011 is 01000000

Adder: read in 00001010 from Incrementer.

Adder: 00001010 added to 00001010 is 00010100

Adder: read in 11110011 from Incrementer.

Adder: 11001110 added to 11110011 is 11000001

Adder: read in 11100000 from Incrementer.

Adder: 11100000 added to 11100000 is 11000000

Adder: read in 00101000 from Incrementer.

Adder: 11000000 added to 00101000 is 11101000

Adder: read in 00010110 from Incrementer.

Adder: 11101110 added to 00010110 is 00000100

Vector subtraction complete.

**Test files**

**vector\_A:**

01100100

01000010

00101101

00001010

11001110

11100000

11000000

11101110

**vector\_B:**

00100100

00101100

11101101

11110110

00001101

00100000

11011000

11101010

**out\_file:**

01000000

00010110

01000000

00010100

11000001

11000000

11101000

00000100