# Program 2 Design Document

Keith Schmitt

CIS 452-10

Program 2

Professor Dulimarta

Data Communicated between parent and child via pipes:

The data that I am using pipes to transfer is to communicate the word that each child process is trying to find and an upstream pipe to communicate the number of occurrences each child found. The downstream pipe (myPipe[2]) communicates a string, while the upstream pipe (my\_out\_pipe[2]) communicates the occurrences.

The Word Search Implementation:

So before I fork, I create a vector of files and pair them up with a vector of counts. Once I have those created, I prompt the users for the files and then I will fork afterwards. I then enter a loop for the parent and prompt the user for words to find or input a -2 to quit. The parent will send downstream the word to find and then the children can begin parsing the files. Then the children will send back the number of occurrences to the parent with the vector of ints correlating to the file.

How the parent process manages all detailed information pertaining to children:

The parent has a vector of filenames and has a pid. I utilize fork to create a process for each file and send the word to find using a pipe to the children. Using upstream and downstream pipes I communicate between processes on the word to find in a file to the and the number of occurrences to the parent. Utilizes a shared Boolean to end processes. The parent also waits for all children to complete the searches before printing the statistics and then asking for another user input.

How graceful termination is implemented:

I use a signal handler and a Boolean called shutdown in order to handle termination. The sigHandler will set the Boolean and the parent will wait patiently for all searches to be done before it closes just to be safe.

Other Notes:

* Not sure if you wanted to spawn the children at the time of the file acquisition, so I decided to fork afterwards and figured it would be pretty readable if I put the filenames in the vector.
* Decided that -2 would be a decent word to quit the process
* Decided that case sensitive search was alright.

#include <signal.h>

#include <stdlib.h>

#include <unistd.h>

#include <stdio.h>

#include <vector>

#include <sys/wait.h>

#include <iostream>

#include <sstream>

#include <fstream>

#define READ 0

#define WRITE 1

void sigHandler(int sigNum);

bool shutdown = false;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Here is Program 2

It is a program that will first take a list of txt files from the command line.

It then will create processes and appropriate data structures in order to manage

these files. The next thing it will do once it has all the file information

organized is to prompt the user for a word to find in all of these files.

We then create a process for each file and will appropriately get information

via upstream and downstream pipes that connect the two processes.

Once all the searchng is done, the parent gets information from upstream pipes

to print out a summary of the searches.

@author: Keith Schmitt

@version 1.0 2/10/2018

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(){

//my upstream pipe

int myPipe[2];

//my downstream pipe

int my\_out\_pipe[2];

int status;

pid\_t pid;

//vector of my files

std::vector<std::string> my\_files;

//vector to be in relation to my\_files keeping track of the word occurances

std::vector<int> word\_occurs;

//string for input stream to vector conversion

std::string files\_to\_find;

//string used for parsing files

std::string single\_file;

//word that we will try to find in the files

std::string word\_to\_find;

//file stream

std::fstream fs;

//boolean used for shutdown

bool shutdown = false;

signal (SIGUSR2, sigHandler);

//my upstream pipe being initialized

if ((pipe(myPipe))<0){

perror("Pipe error");

exit(1);

}

//my downstream pipe being initialized

if ((pipe(my\_out\_pipe))<0){

perror("Pipe error");

exit(1);

}

// getting the files the children will work on and put them into a vector

std::cout<<"Please list the text files you would like to scan for: ";

std::getline(std::cin, files\_to\_find);

std::stringstream input\_stream(files\_to\_find);

//parsing the line into their files by

while (input\_stream >> single\_file){

my\_files.push\_back(single\_file);

word\_occurs.push\_back(0);

}

//printing out which files they are searching just for clarity sake

std::cout<<"Files you are searching are: \n";

for (std::vector<std::string>::iterator printout = my\_files.begin() ; printout != my\_files.end(); ++printout){

std::cout << \*printout<<std::endl;

}

//will exit on non-alphabetic input of -2

std::cout<<"Please Enter -2 if you would like to quit"<<std::endl;

while ( !shutdown){

//parent gets output

std::cout<<"\nPlease list the word(s) you would like to search for: ";

std::getline(std::cin, word\_to\_find);

//if input is -2 send a signal

if (word\_to\_find == "-2"){

sigHandler(SIGUSR2);

wait(&status);

break;

}

for (unsigned int i = 0; i < my\_files.size(); ++i){

if ((pid = fork()) < 0){

perror("fork failure");

exit(1);

}

if (pid == 0){

//count to keep track of how many times the word comes up in a child file

int count = 0;

//open specified file from vector

fs.open(my\_files[i],std::fstream::in);

//print out child id

std::cout<<"Child "<<getpid()<< "\tHas opened the file: "<<my\_files[i]<<" with a parent of " <<getppid()<<std::endl;

//string to load file into

std::string occurs;

//string variable for the word to find

std::string correct\_word;

//getting the word to find from downstream pipe child will be blocked

//until the write from the parent succeeds due to the read() call

read(my\_out\_pipe[READ], &correct\_word, sizeof(correct\_word));

//going to count from filestream

while (fs >> occurs ){

if (occurs == correct\_word){

++count;

}

}

//close the file

fs.close();

//write to upstream pipe

write(myPipe[WRITE], &count, sizeof(count));

return 0;

}

else{

//sending downstream pipe for the word\_to\_find

write(my\_out\_pipe[WRITE], &word\_to\_find, sizeof(word\_to\_find));

//responsible parent will wait for children

wait(&status);

//read from upstream pipes

int parent\_count = 0;

//read blocks until child is done writing

read(myPipe[READ], &parent\_count, sizeof(parent\_count));

//for organization purpose, we put it in an array

word\_occurs[i] = parent\_count;

}

}

//last thing for the parent to do is do some statistics

if (pid != 0){

//parent prints out statistics

std::cout<<"\nStatistics: "<<std::endl;

int total\_occurrences = 0;

for (unsigned int i =0; i < my\_files.size(); ++i){

//print results

std::cout<<my\_files[i]<<" has found "<< word\_to\_find<<" "<<word\_occurs[i]<< " times"<<std::endl;

//sum a total

total\_occurrences += word\_occurs[i];

}

//print out total

std::cout<<"Total finds: "<<total\_occurrences<<std::endl;

//averages are nice :)

std::cout<<"Average find per file: "<< double(total\_occurrences)/my\_files.size()<<std::endl;

}

}

return 0;

}

void sigHandler(int sigNum){

//sending a signal to back to the parent program to tell it to wait

if (sigNum == SIGUSR2){

shutdown = true;

return;

}

//default behavior

else if (sigNum == SIGINT){

exit(0);

}

}