Ryan Hennes

November 28th, 2018

CECS 326, 9:00 AM, MW

[Ryan.c.hennes@gmail.com](mailto:Ryan.c.hennes@gmail.com)

Program Description:

This program shows my knowledge of using the fork system call. The program will first prompt the user for a target string that will be the string that is somewhere in the text document. Then the user is prompted for a replacement string. Both of these inputs can also accept the string :wq which will quit out the program. Once both the target and replacement string have been entered, the program will call the fork system call which will spawn a child process to replace all of the target strings in the document with the replacement string. While the child process does this, the parent process will wait for it to finish. Once the child process finished replacing the strings, it exits itself and resumes the parent process. There is a built in bug for the child process where it will become stuck in an infinite loop, printing a (.) every iteration of its attempt at replacing the string. The only way to stop the process is by opening a separate terminal and killing the child process.

Program Text Document:

In recent years, JavaScript has become the de-facto language of choice when building websites, and one of the most popular ones to build web applications. There are many reasons behind this, including it’s treatment of objects and the ability to build applications quickly. There are some people who feel that it’s unjustly popular, due to a lot of JavaScript’s perceived shortcomings when compared with more traditional programming languages. JavaScript has some strange quirks and behavior that bothers some engineers. One thing everyone can agree on is that JavaScript is one of the most used languages today, and for all intents and purposes it seems to be staying that way. Knowing JavaScript is essential for any web developer. To remedy JavaScript’s shortcomings, developers have created a vast amount of libraries, each trying to solve their own problem, or plenty trying to solve the same problem in marginally different ways. Being familiar these libraries has become essential, even if it is merely to be aware that you can use them to ease your development process. Some of the most widely used libraries are as follows: Lodash, Ramda, and TypeScript. The maintainers of JavaScript language are of course aware of these needs and are adding these sought after features into the core JavaScript standard as time passes. There is one distinction that needs to be addressed, however, the differences between ECMAScript and JavaScript.

Test cases:

Test One: This test shows the normal case where an input was made and the correct replacement was made.

Target String: There

Replacement String: \\\\

Expected Output: 3 replacements made

Console output:

The amount of strings replaced by the process was: 3

The new document looks like: In recent years, JavaScript has become the de-facto language of choice when building websites, and one of the most popular ones to build web applications. \\\\ are many reasons behind this, including it’s treatment of objects and the ability to build applications quickly. \\\\ are some people who feel that it’s unjustly popular, due to a lot of JavaScript’s perceived shortcomings when compared with more traditional programming languages. JavaScript has some strange quirks and behavior that bothers some engineers. One thing everyone can agree on is that JavaScript is one of the most used languages today, and for all intents and purposes it seems to be staying that way. Knowing JavaScript is essential for any web developer. To remedy JavaScript’s shortcomings, developers have created a vast amount of libraries, each trying to solve their own problem, or plenty trying to solve the same problem in marginally different ways. Being familiar these libraries has become essential, even if it is merely to be aware that you can use them to ease your development process. Some of the most widely used libraries are as follows: Lodash, Ramda, and TypeScript. The maintainers of JavaScript language are of course aware of these needs and are adding these sought after features into the core JavaScript standard as time passes. \\\\ is one distinction that needs to be addressed, however, the differences between ECMAScript and JavaScript.

Test Two: This test demonstrates the programmed in bug.

Target String: goooop

Replacement String: t

Expected Output: infinite while loop printing out dots

Console output:

.

.

.

.

.

.

.

.

ect.

#include <unistd.h>

#include <iostream>

#include <cstdlib>

#include <sys/types.h>

#include <sys/wait.h>

#include <string>

using namespace std;

/\*

This program will prompt the user to enter in a target string from the below document, then ask the user to enter a replacement string. The program then spawns a child process to handle the replacement of the string. If the string does not exist within the document a programmed in bug will begin where a (.) is printed out everytime the child process attempts to replace a string within the document. The only way for the child process to end is by killing it through another terminal. On completion of the child process it exits itself.

\*/

int main() {

//counter for the child process to determine the amount of strings that were replaced.

int stringsReplaced;

//position of the most recently replaced string to be used in the string manipulation section

int stringPosition;

//status id used by the wait() sys call

int status;

//value of what should be the child PID

long childPID;

//bool value to create infinite while loops

bool forever = true;

//input strings for both the target and replacement strings that the user picks

string inputTarget;

string inputReplacement;

//hard coded text document

string textDocument = "In recent years, JavaScript has become the de-facto language of choice when building websites, and one of the most popular ones to build web applications. There are many reasons behind this, including it’s treatment of objects and the ability to build applications quickly. There are some people who feel that it’s unjustly popular, due to a lot of JavaScript’s perceived shortcomings when compared with more traditional programming languages. JavaScript has some strange quirks and behavior that bothers some engineers. One thing everyone can agree on is that JavaScript is one of the most used languages today, and for all intents and purposes it seems to be staying that way. Knowing JavaScript is essential for any web developer. To remedy JavaScript’s shortcomings, developers have created a vast amount of libraries, each trying to solve their own problem, or plenty trying to solve the same problem in marginally different ways. Being familiar these libraries has become essential, even if it is merely to be aware that you can use them to ease your development process. Some of the most widely used libraries are as follows: Lodash, Ramda, and TypeScript. The maintainers of JavaScript language are of course aware of these needs and are adding these sought after features into the core JavaScript standard as time passes. There is one distinction that needs to be addressed, however, the differences between ECMAScript and JavaScript.";

//main menu loop the program runs through

while (forever)

{

//collecting user input for the target string, as well as giving them the option of exiting the program

cout << "Enter a target string that will be replaced within the text document or :wq to exit the program" << endl;

cin >> inputTarget;

if(inputTarget == ":wq")

{

exit(0);

}

//collecting user input for the replacement string as well as giving the option of exiting the program

cout << "Enter a replacement string that will replace the target string in the text document or :wq to exit the program" << endl;

cin >> inputReplacement;

if(inputReplacement != ":wq")

{

//here fork() is called creating a child process where the PID of the new process is stored.

childPID = fork();

//in this if statement we would know the child process is executing thus it does the replacement calculations.

if (childPID == 0)

{

//initiallizing both position and count to 0 for each process

stringPosition = 0;

stringsReplaced = 0;

//first run condition allows the bug to override my normal exit condition to continue running

bool firstRun = true;

//exit condition allows the loop to break when the string position hits the end of the text document

while ((stringPosition = textDocument.find(inputTarget, stringPosition)) != std::string::npos || firstRun == true)

{

//setting the bug flag to false to ensure that the loop does not get stuck if it is working as planned

firstRun = false;

//creating bug where string position is set to -1 due to the searched string not existing

if (stringPosition == -1)

{

cout << "." << endl;

//using small sleep to slow down print statments otherwise my laptop slowed down so much I could not open another terminal to kill the process

sleep(.1);

//creating the bug by resetting the exit condition

firstRun = true;

}

//if the bug does not exist the proper increments are made to find and replace the next string

else

{

textDocument.replace(stringPosition, inputTarget.length(), inputReplacement);

stringPosition += inputReplacement.length();

stringsReplaced++;

}

}

//having the child process output the amount of strings replaced in the program

cout << "The amount of strings replaced by the process was: " << (stringsReplaced) << endl;

//debug or possible output statement to show the text document after string replacement

cout << "The new document looks like: " << textDocument << endl;

exit(0);

}

//this would be the parent process that will wait until its child process terminates to continue

else if (childPID > 0)

{

wait(&status);

}

//this would be the scenario where the child process was not created properly

else if (childPID < 0)

{

cout << "Error creating child process, returning to initial prompt" << endl;

}

}

//this exit statement triggers when the user enters :wq

else

{

exit(0);

}

}

}