/\*test machine: csel-kh1250-04.cselabs.umn.edu

\* date: 10/07/21

\* name: Michael Allrich, Lee Knutson, Ke Wang

\* x500: allri004, knut0577, wan00802

\*/

-purpose of the program

The purpose of this program is to gain an understanding of how to run multiple processes sequentially and in parallel using the fork system call in c. Given an input file representing a tree of commands, a graph is created that represents the links between and order of the commands. Print the commands in their proper order to a new output file, then execute the given commands in the proper order. Some of these will run better or properly depending on their execution mode: sequential or parallel.

-how to compile

Navigate to a working directory within a terminal, copy file contents into the directory. Then execute the makefile with `make`. Then execute the program with `./depGraph <input file>`. Where <input file> is one of the included input files titled “input1.txt”, “input2.txt” etc.. For sequential execution just type the file name after `./depGraph`. For parallel execution type `-p` before the input file name. Then the output of the program should be displayed onto the terminal. And additional output will be created into a “results.txt file” in the same directory. If this file is deleted it will return after the next time the program is run.

-what the program does

Produces an output file with the given commands in the proper order, then executes the given commands.

struct AdjListNode\* newAdjListNode(int dest)

Allocate the memory for and initialize a new adjacency list node with dest = dest and next = NULL

struct DepGraph\* createDepGraph(FILE \*input, char cmds[][550])

Which is creating the graph using the input command as the content, and by the order of the DFS order creating the Graph.

void addEdge(struct DepGraph\* graph, int src, int dest)

Each time using the input graph(Also the source and the destination of it) to add the edge on the graph

void DFSVisit(struct DepGraph\* graph, int node, char cmds[][550], int mode)

The DFSVisit will take the command of the input and also adjust the mode according to the number of input commands.

void processGraph(struct DepGraph\* graph, char cmds[][550], int mode)

Using the fork function to create the new process and make the parent process wait for all the child processed

void freegraph(struct DepGraph\* graph)

Calls on freearray to free the graph’s array, then frees the memory allocated for the graph itself.

void freearray(struct DepGraph\* graph)

Helper function for freegraph. Loops through each index of the array and calls freenode on the head node. Then frees the memory allocated for the array.

void freenode(struct AdjListNode\* mynode)

Helper function for freearray. Recursively loops through the linked list, calling itself on the next node until it reaches the end of the list before freeing the current node. When called on the head node, it calls itself on every node following. This means it frees all nodes after the head node before freeing itself.

-assumptions

No more than 32 nodes are present in the graph.

No more than 10 arguments are present in a command.

No more than 50 characters are present in a command line.

-contributions

Micheal Allrich: Completed the DFS algorithm on the created graphs, and function adding an edge to the graph.

Ke Wang: Assist the function of command line input, and parallel execution of nodes

Lee Knutson: Wrote a draft for addedge. Wrote functions to free allocated memory and edited the code to prevent leaks.

All: Lots of debugging, contributed to DFSvisit