Name, Surname: Beren Aydoğan
ID: 31938

Program Overview

The program, treePipe.c, implements the command "treePipe" which constructs a binary tree structure whose nodes are processes that execute two different programs based on their position, whether they are the left or right child of their parent. The program achieves this by utilizing several system calls including:

- fork() to spawn child processes from already existing parent processes,
- execvp() to transform a process to the given program,
- pipe() to create an anonymous file and establish communication between processes, and
- dup2() to copy a file descriptor and redirect the standard input and output to the pipe.

Helper Functions

Function Name: void printDashes(int times){}

- **What it does:** Prints 3*"times" dashes onto the console.
- **Purpose:** Through the program, during the outputs, arrows of length equal to (current depth multiplied by three) are printed before the information. To match this form of the output given in sample runs, this function prints dashes according to the integer parameter "times".
- **Method:** Uses a for loop to print three times the integer parameter "times" of dashes consecutively to the console using stderr.

Function Name: void transformNode(int curDepth, int maxDepth, int lr) {}

- **What it does:** Transforms the current process into a new instance of the treePipe program using its parameters curDepth, maxDepth, and lr to construct the new instance's command line arguments to move down the tree.
- **Purpose:** Creating new instances of the treePipe program is essential for adding new nodes and creating the tree structure. The new instances of treePipe programs act as

orchestrators that fork to create left and right children processes if needed and also create a worker process to run its designated program.

- **Method:** Declares character arrays curDepthStr, maxDepthStr, and lrStr to store curDepth, maxDepth, and lr integer values as strings. It converts these values to strings by utilizing the sprintf() method. Before turning curDepth into a string, it increments it by to move down the tree. Then, it constructs an argument array for execvp() and replaces the current process with the new instance of the treePipe program by executing the specified command.

Function Name: void createWorker(int curDepth, int lr) {}

- **What it does:** Forks the process running this function to create a worker process to execute the left or right program depending on the parameter lr. Also, prints the result of the worker process according to the provided format using printDashes() function and parameter curDepth.
- **Purpose:** Each node is an instance of the treePipe program that executes another program, precisely the left or right program, depending on its location with regard to its parent. To do this, each node must fork a child process to use as a worker process and use execvp() to execute the designated program. This function exactly does that.
- **Method:** Declare resStr char array to hold the result of the program. Forks to create a worker-child process. The child process constructs the argument array depending on the parameter "lr" and utilizes execvp() to transform itself into an instance of the executed program. The parent process waits for the child to finish executing using the wait() system call, then reads the result printed by the worker process from the pipe and stores it in resStr. Prints result using stderr to the console using the printDashes() function and the parameter curDepth to stick to the format of the provided output files. Writes the result to the pipe to be read by its parent.

<u>Main Function</u>

Function Name: int main(int argc, char *argv[]) {}

main() serves as the entry point of the program. It initializes the tree based on the given command line arguments. It controls the flow of the program by implementing the correct sequence of operations and providing checks for possible errors.

The main function starts by parsing command-line arguments using the array argv. The program requires three arguments: the current depth of the node, the maximum depth of the tree, and whether the node is a left or right child. main() retrieves these values from the array argv by using their indexes, converts them from string to integer using atoi(), and assigns them to integer variables (curDepth, maxDepth and lr, respectively) for ease of use. If these arguments are not provided, it prints the usage of the command to give the user an idea of how to use the command and returns 1 (the error code).

Then, some variables are declared and initialized for later use. These variables are:

- Flags to keep the return of system calls
  - int left_rc = 0;
  - int right_rc = 0;
  - int pipeReturn = 0;
  - int dup2Return = 0;
- Char arrays to read the values from pipe to
  - char num1Str[11] = {'\0'};
  - char num2Str[11] = {'\0'};
  - char resStr[11] = {'\0'};
- An array of two integers to hold file descriptors returned by the pipe() system call
  - int fd[2];

If the current process is the root of the process tree, in other words, if (curDepth==0), some operations are done as initial setup tasks. These include:

- printing the depth and it is a left/right node (root is always left) using stderr to the console,
- prompting the user for the num1 value, scanning the num1 value the user provides from the console into the num1Str char array, and printing it,

- creating a pipe to be used for communication between the process nodes; checking if it was successful, and exiting if not,
- copying pipe to stdin and stdout of the process; checking if it was successful, and exiting if not,
- closing the original file descriptors as they will no longer be used,
- writing num1 value to the pipe to be read by the child process.

Subsequently, the program moves into recursive process creation.

The program checks if the current depth is not equal to the maximum depth (curDepth != maxDepth), that is, if the process is of a leaf node. If this condition is true, it forks a left child process. If the fork() system call was successful, fork() returns 0 which is stored in its left_rc variable. The left child process (left_rc == 0) checks whether its current depth satisfies the condition (curDepth < maxDepth) and if it does, it transforms itself into another instance of the treePipe program with curDepth incremented by 1 executing transformNode(curDepth, maxDepth, 0) function. The execvp() system call replaces the child process with a new instance of treePipe which is a recursive call as it is executing the same program with different parameters.

The left child continues its execution by printing num1 and num2 values to the pipe to be read by the worker process and forking a worker child process using the createWorker() function with the appropriate parameters. createWorker() function writes the result of the left program to the pipe for the parent process to read and use.

Upon the execution of the left child, the parent reads the result printed by the left child's worker process to the pipe as its num2. Initially, it prints its current depth, num1, and num2 using stderr to the console. Then, it writes the current value of num1 and num2 to the pipe to be read by the worker process, which is forked using createWorker() function with parameters depending on the type of program (left/right) it's supposed to run. createWorker() function writes the result of the program run by the worker process to the pipe to be read by the right child.

Then, the process forks the right child. If the fork() system call was successful, fork() returns 0 which is stored in its right_rc variable. The right child process (right_rc == 0) directly executes transformNode(curDepth, maxDepth, 1) and it transforms itself into another instance of the treePipe program with curDepth incremented by 1. If needed, this right process creates its own children. If it is a leaf node, this node prints num1 and num2 values to the pipe to be read by the worker process and forks a worker child process by utilizing createWorker() function. If it is not a leaf node, upon completion of its left child, it does the same and then it writes the result of the program run by the worker process to the pipe to be read by the right child. Upon the right child's completion, it directly gives the output of the right child to the pipe without any additional computation. This is the case for a left type node too, when it receives from its right child.

After this recursion, which is essentially an inorder traversal of a binary tree, the result obtained from the right-bottommost node of the tree is transferred to the root through the pipe and it is printed to the console.

Role of Pipes

Processes (considered as nodes of the tree structure) communicate with each other through pipes. In this program, pipes are used to carry parameters num1, num2, and the results of the operations done by the worker processes. This flow allows the data to propagate up and down the tree structure.

The program sets up a pipe and uses dup2() system call to copy the file descriptors of the pipe to the standard input and standard output which allows the program to use scanf(), and printf() to read from and write to the pipe instead of read() and write() system calls. It sets the root's file descriptors which are transferred to child processes by fork(). Because their standard input and standard output are connected to the pipe, the processes use the standard error to write something to the console.