|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Error** |
| fork() | Clones a new child process from a parent | “could not fork process” |
| dup2(oldfd, newfd) | Closes newfd if needed, then duplicates oldfd into newfd | “could not dup fd” |
| excve(exe\_path, cmd, envp) | Receives full path of executable parameters and environment. Replaces current process with that of the command | “PATH variable is not set” |
| pipe(fd[2]) | Receives fd[2] and opens fd[0] for reading and fd[1] for writing | “could not create pipe” |
| access(path, F\_OK | X\_OK) | Receives program name(cmd) and checks the path to see if the binary file exists and if we can execute it | “no such file or directory” |
| waitpid(pid, status, 0) | Waits for a specific child process to end (matching pid) | No error |

**Pipex Functions**

Understanding the pipe operator

cat text.txt | wc -l – this command will display the number of lines of text in the text.txt file. The pipe operator redirects cat’s standard output into wc’s standard input. The cat command writes its result into pipe and then wc looks for the data in the pipe.

Pipex must be executed by ./pipex file1 cmd1 cmd2 file2 which is equivalent to

< file1 cmd1 | cmd2 > file2

< : used to denote that we will be passing the next argument as the stdin

file1 : path to the file we want to open as the stdin. It must exist and should be opened as read-only

cmd1 : first command. It will recive the stdin and run a command with it

| : transforms the stdout of the first command into the stdin for the next command

cmd2 : receives the stdout of the first command as stdin and runs a command with it

> : redirects whatever is on the stdout into a file, creating it if it doesn’t exist

file2 : path to an output file which may or may not exist. If it exists it will be truncated and should be opened read-only.

fd[0] = Read end

fd[1] = Write end

envp = environment pointer. Contains system information including the location of PATH, which we need to find the binary files for the shell program (or cmd) we want to execute (eg, ls or wc)

[0] = stdin

[1] = stdout

Getting the PATH line – places all possible listed paths in PATH into a string array and adds a “/” to make them valid paths.

* use ft\_strncmp to check for “PATH=” in the envp
* use ft\_split to split the paths using delimiter ‘:’
* use ft\_strjoin to join the split path to a “/”

Check all the possible paths in PATH to see if the command (cmd) can be accessed

* use ft\_strjoin to join the path found in get\_path to cmd
* use access to check the path to see if the binary file (cmd) exists and if we can execute it

In the parent process

* dup2(infile, STDIN\_FILENO)
* create a pipe and error check
* fork a child pid
* if !pid then we are in the child process
* close the write end fd[1]
* dup2(fd[0]), STDIN\_FILENO)
* wait for the child using waitpid
* close the read end fd[0]

In the child process

* close the read end (fd[0])
* dup2(fd[1], STDOUT\_FILENO) – error check
* if dup2(outfile, STDOUT\_FILENO) – error check
* close the write end (fd[1])
* close the infile
* close the outfile
* execve(exe\_path, cmd, envp)
* return error if cmd does not exist

The main function

* if argc == 5 – return error
* if not open infile
* open outfile
* dup2(infile, STDIN\_FILENO)
* dup2(outfile, STDOUT\_FILENO)
* parent
* child