Tuan Nguyen

Shell program

-will provided users a list of 8 built in commands.

-running process in background (by appending & after the command)

-I/O redirection and piping between commands

Two .c file

-main.c and command.c

Shell support 8 built in commands:

-cdschange the current default directory to <directory>.

-clr : clear the screen

-dir<directory>: list all contents under <directory>

-environ : list all environment variables

-echo<comment> : display <comment> in the screen

-help : display the user manual

-pause : pause the shell until Enter was hit

-quit : quit the shell

**Pseudocode for main.c**

- how to run the shell, create processes, I/O redirection and piping

//build\_in commands strings

**char \*internal\_cmds[] = {**

"cd",

"clr",

"dir",

"environ",

"echo",

"help",

"pause",

"quit"

};

//function pointers to build\_in functions

**int (\*internal\_funcs[]) (char \*\*) =** {

&cds,

&clr,

&dlr,

&environt,

&echos,

&helps,

&paused,

&quits,

};

//the shell runs in a loop

**function loop(){**

while(true):

prompt = getcwd(prompt, size) // current directory

print prompt

char \*input = read\_input() // read user input

char args = parse\_arg(input) // parse to get the command & argument

execute(args) //execute and check state of the shell

}

//read user input and store in a buffer

**function read\_input(){**

read the input and

return pointer to the string

}

//parse user input into args which separated by t\r\n\a

**function parse\_arg(char \*input){**

allocate an array strings: (strings[])

split input string into token characters

while(token != whitespace):

strings[i++] = token;

strings[i] = NULL

return strings

}

//execute the cmds

**function execute(char \*\*arg){**

if arg[0] == NULL: // empty command

return

symbol = get\_symbol(arg) // will be used to check redirection or pipe

if symbol < 0: // that means no redirection or pipe, just built-in command

call builtin\_comm(arg) with the command string matches the parsed command

else: // I/O redirection or piping

if symbol == 0:

invoke\_pipe()

else if symbol == 1:

redirect(arg1, arg2, 0) // I/O redirection for '<'

else if symbol == 2:

redirect(arg1, arg2, 1) // I/O redirection for '>'}

//create process to run (with background execution)

**function launch\_process(char \*\*arg){**

bool background = false // boolean to indicate background process

char \*name // process's name

if name contains "&" at the end:

background = true;

pid = Fork();

if(pid == 0): // Child's process

if(execvp(name , arg) < 0) //error executing

exit(-1)

else: // Parent's process

if background = true: don't wait

else wait()

}

**//I/O redirection**

function redirect(arg1 , arg2, int type){

char \*filename = arg2[0]

if(type == 0): // the file will be input source

dup2(STDIN)

else: // the file will be output source

dup2(STDOUT)

}

//piping two commnads

**function invoke\_pipe(char \*arg1 , char \*arg2){**

pid1 , pid2 // process's id

int fd[2] // file descriptor

if arg1 is built\_in commands:

pid = Fork()

if(pid == 0): // Child's process will read input

close(fd[Write])

dup2(fd[Read] , STDIN)

if(execvp() < 0) // error executing

exit(-1)

else: // Parent's process will send the input

close(fd[Read])

dup2(fd[Write] , STDOUT)

call builtin\_comm(arg) with the command string matches the parsed command

wait()

else if arg1 is not built\_in\_commands:

basically similar to the previous case with

the exception of two new child processes

}

**Pseudocode for command.c**

- all the internal commands and helper functions

//get the len of args

**function arglen(char \*\*arg){**

return len of args

}

// function to change directory

**function cd(char \*\*arg){**

if no argument:

change directory to #home

else:

change directory to arg[1]

}

//function to clear command

**function clr(char \*\*arg){**

invoke system("clear")

}

//function to quit the command

**function quit(char \*\*arg){**

return

}

//function to echo command

**function echo(char \*\*arg){**

if no argument:

print nothing

else:

for i: 1 to arg[i]!=NULL

print arg[i]}

//function to get environment variable

**function env(char \*\*arg){**

for i: 1 to env[i]!=NULL

print env[i]

}

//function to open directory

**function dlr(char \*\*args)**

struct dir

char \*cur = size of cur

// if loop to check for current directory

if( args[1] == empty

print cannot find current directory

else // find current directory

open directory

//function for help command

**function help(char \*\*arg){**

char \*res = {"more",...}

print res

}

//function to pause command

**function pause(char \*\*arg){**

print: "Press enter to continue"

while getchar!= \n:

//process is blocking

}