**File: main.c**

int main(int argc, char \*\*argv) {

// execution loop.

shell\_loop();

return 0;

}

**File: function.c**

**void shell\_loop(void)** {

char \* cmd\_string;

char \*\*args;

int status;

while (true){

// print prompt --- depends on current directory

cmd\_string = **shell\_read\_line**();

args = **shell\_parse\_line**(cmd\_string);

status = **shell\_execute**(args);

};

}

#define LINE\_LENGTH 100

char \* **shell\_read\_line**(void){

// initialize & allocate memory for a reading buffer

while (true) {

// Read a character

// If we hit EOF, replace it with a null character and return.

// else, feed “character” into buffer & increment buffer array position

}

return buffer;

}

#define DELIMITERS " \t\r\n\a"

char \*\* **shell\_parse\_line**(char \* cmd\_string){

// initialize & allocate memory for a reading buffer (token\_list)

// use strok to tokenize cmd\_string

token = strtok(cmd\_string, DELIMITERS);

// set last position of token list to NULL

// return token\_list;

}

#define NUM\_BUILT\_INS 8 // 8 built-in functions

**shell\_execute**(char \*\* args){

int i;

// check if file exists

// Identify built-ins

char \* builtin\_cmds[] = { "cd", “clr”, “ls”, “environ”, “echo”, "help", “pause”, "exit"};

int (\*builtin\_func[]) (char \*\*) = {&shell\_cd, &shell\_clr, &shell\_ls, &shell\_environ, &shell\_echo, &shell\_help, &shell\_pause, &shell\_exit};

// detect which builtin to use

// find files in environment $PATH

for (i = 0; i < NUM\_BUILT\_INS; i++) {

if (strcmp(args[0], builtin\_cmds[i]) == 0) {

return (\*builtin\_func[i])(args);

}

}

// if doesnot match the builtins, start normal execution  
 shell\_normal\_execution(args);

}

**shell\_normal\_execution(){**

**// create child process**

**// if (PID == 0) check pID if child was created (pID < 0 – error)**

handle any input/output redirection by changing the file

descriptors of the child process using the **dup2** system call.

handle possible piping between programs by using the **pipe()** system call

**// execvp (name of program to be exec.ed, input\_arguments) – execute the external file** char \* input[3] = {ls, -la, NULL}  
 exec(input[0], input)

// check if “.txt” file for output

**// else wait()** // use & **// waitpid(pid, NULL< 0)**

**I/O redirection(&file){**

**}**

**Pipe()  
 2 channels** READ  
 WRITE  
 process 1 (close read)  
 check PipeAndExec.c on Canvas

**09/24 ppt slides**

While (1) { Char \*cmd = getcmd(); Int retval = fork(); If (retval == 0) { // This is the child process // Setup the child’s process environment here // E.g., where is standard I/O, how to handle signals? exec(cmd); // exec does not return if it succeeds printf(“ERROR: Could not execute %s\n”, cmd); exit(1); } else { // This is the parent process; Wait for child to finish int pid = retval; wait(pid); } }

}

**shell\_cd()**{

// if arg[1] == NULL --> return error

// else

if (chdir(args[1]) != 0) { // change directory of prompt also

// print error message

}

}

return 1;

}

**shell\_clr()**{

//…

}

**shell\_ls()**{

//…

// check if valid or not

// take flags

}

**shell\_environ**{

Environ should emulate what the

env

command in bash does:

Prints out the environment variables.

To get an environment variable in C, use the system call

‘getenv().

Example: to get “USER” environment variables, use

getenv(“USER”)

You do not need to implement

ALL

the environment variables

that env

does; Only print out the ones you think are the most

beneficial for the user to know.

}

**shell\_echo**(){

}

**shell\_help**(){

//…

}

**shell\_paus**e(){

//…

}

**shell\_exit()**{

return 0;

}

Makefile?

Done builtin cmds,

External programs:

Fork() – test

Exec () & wait()

Redirection

Pipe

Background

myshell input.txt ----- not myshell < input.txt

Check if user pass in argument

 









