Group member：戴陽(Function Create)、易頡(History Feature)

Environment：Linux

Instructions：

1. gcc z\_shell.c -o z-shell.o

2. run ./z-shell.o

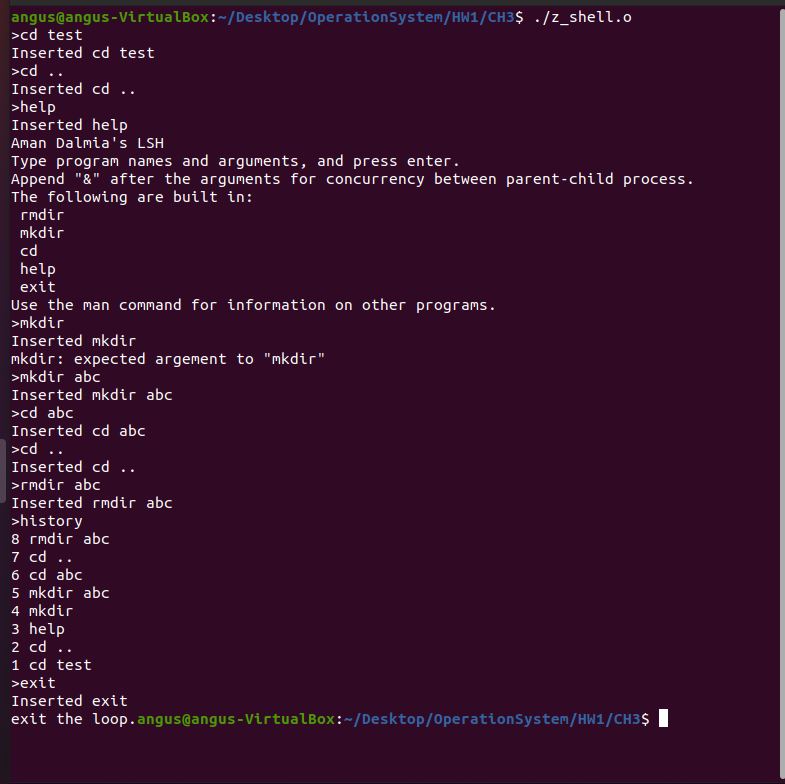
3. enter help or other commands

4. enter history to print all the commands entered before including invalid commands

5. exit

6. done

Functions & history feature



Source code：

#include <sys/types.h>

#include <stdio.h>

#include <unistd.h>

#include <string.h>

#include <stdlib.h>

typedef int bool;

#define true 1

#define false 0

#define LSH\_RL\_BUFSIZE 1024 /\* buffer size for reading user input \*/

#define LSH\_TOK\_BUFSIZE 64 /\* buffer size for splitting the arguments \*/

#define LSH\_HIST\_SIZE 10 /\* buffer size for storing history of commands \*/

#define LSH\_TOK\_DELIM " \t\r\n\a" /\* delimiters for parsing the arguments \*/

/\* global variable to check parent and child process concurrency \*/

bool conc = false;

/\* global variable to point to the last command executed \*/

int cur\_pos = -1;

/\* global variable storing the history of commands executed \*/

char \*history[LSH\_HIST\_SIZE];

int cur\_bufsize = LSH\_TOK\_BUFSIZE;

/\* Function declarations for built-in shell commands \*/

int lsh\_rmdir(char \*\*args);

int lsh\_mkdir(char \*\*args);

int lsh\_cd(char \*\*args);

int lsh\_help(char \*\*args);

int lsh\_exit(char \*\*args);

/\* List of built-in commands, followed by their corresponding functions \*/

char \*builtin\_str[] = {

"rmdir",

"mkdir",

"cd",

"help",

"exit"};

int (\*builtin\_func[])(char \*\*) = {

&lsh\_rmdir,

&lsh\_mkdir,

&lsh\_cd,

&lsh\_help,

&lsh\_exit};

int lsh\_num\_builtins()

{

return sizeof(builtin\_str) / sizeof(char \*);

}

/\* Built-in function implementations \*/

int lsh\_mkdir(char \*\*args)

{

if (args[1] == NULL)

{

fprintf(stderr, "mkdir: expected argement to \"mkdir\"\n");

}

else

{

if (mkdir(args[1], 0700) != 0)

{

perror("lsh");

}

}

return 1;

}

int lsh\_rmdir(char \*\*args)

{

if (args[1] == NULL)

{

fprintf(stderr, "rmdir: expected argement to \"rmdir\"\n");

}

else

{

if (rmdir(args[1]) != 0)

{

perror("lsh");

}

}

return 1;

}

int lsh\_cd(char \*\*args)

{

if (args[1] == NULL)

{

fprintf(stderr, "lsh: expected argument to \"cd\"\n");

}

else

{

if (chdir(args[1]) != 0)

{

perror("lsh");

}

}

return 1;

}

int lsh\_help(char \*\*args)

{

int i;

printf("Aman Dalmia's LSH\n");

printf("Type program names and arguments, and press enter.\n");

printf("Append \"&\" after the arguments for concurrency between parent-child process.\n");

printf("The following are built in:\n");

for (i = 0; i < lsh\_num\_builtins(); i++)

{

printf(" %s\n", builtin\_str[i]);

}

printf("Use the man command for information on other programs.\n");

return 1;

}

int lsh\_exit(char \*\*args)

{

return 0;

}

/\* Launch a program \*/

int lsh\_launch(char \*\*args)

{

pid\_t pid, wpid;

int status;

pid = fork();

if (pid == 0)

{ /\* child process \*/

if (execvp(args[0], args) == -1)

perror("lsh");

exit(EXIT\_FAILURE);

}

else if (pid > 0)

{ /\* parent process \*/

if (!conc)

{

do

{

wpid = waitpid(pid, &status, WUNTRACED);

} while (!WIFEXITED(status) && !WIFSIGNALED(status));

}

}

else

{ /\* error forking \*/

perror("lsh");

}

conc = false;

return 1;

}

/\* Parse input to get the arguments \*/

char \*\*lsh\_split\_line(char \*line)

{

cur\_bufsize = LSH\_TOK\_BUFSIZE;

int position = 0;

char \*\*tokens = malloc(cur\_bufsize \* sizeof(char \*));

char \*token;

if (!tokens)

{

fprintf(stderr, "lsh: allocation error\n");

exit(EXIT\_FAILURE);

}

token = strtok(line, LSH\_TOK\_DELIM);

while (token != NULL)

{

tokens[position] = token;

position++;

if (position >= cur\_bufsize)

{

cur\_bufsize += LSH\_TOK\_BUFSIZE;

tokens = realloc(tokens, cur\_bufsize \* sizeof(char \*));

if (!tokens)

{

fprintf(stderr, "lsh: allocation error\n");

exit(EXIT\_FAILURE);

}

}

token = strtok(NULL, LSH\_TOK\_DELIM);

}

if (position > 0 && strcmp(tokens[position - 1], "&") == 0)

{

conc = true;

tokens[position - 1] = NULL;

}

tokens[position] = NULL;

return tokens;

}

/\* History of commands \*/

int lsh\_history(char \*\*args)

{

if (cur\_pos == -1 || history[cur\_pos] == NULL)

{

fprintf(stderr, "No commands in history\n");

exit(EXIT\_FAILURE);

}

if (strcmp(args[0], "history") == 0)

{

int last\_pos = 0, position = cur\_pos, count = 0;

if (cur\_pos != LSH\_HIST\_SIZE && history[cur\_pos + 1] != NULL)

{

last\_pos = cur\_pos + 1;

}

count = (cur\_pos - last\_pos + LSH\_HIST\_SIZE) % LSH\_HIST\_SIZE + 1;

while (count > 0)

{

char \*command = history[position];

printf("%d %s\n", count, command);

position = position - 1;

position = (position + LSH\_HIST\_SIZE) % LSH\_HIST\_SIZE;

count--;

}

}

else

{

char \*\*cmd\_args;

char \*command;

if (strcmp(args[0], "!!") == 0)

{

command = malloc(sizeof(history[cur\_pos]));

strcat(command, history[cur\_pos]);

printf("%s\n", command);

cmd\_args = lsh\_split\_line(command);

int i;

for (i = 0; i < lsh\_num\_builtins(); i++)

{

if (strcmp(cmd\_args[0], builtin\_str[i]) == 0)

{

return (\*builtin\_func[i])(cmd\_args);

}

}

return lsh\_launch(cmd\_args);

}

else if (args[0][0] == '!')

{

if (args[0][1] == '\0')

{

fprintf(stderr, "Expected arguments for \"!\"\n");

exit(EXIT\_FAILURE);

}

/\* position of the command to execute \*/

int offset = args[0][1] - '0';

int next\_pos = (cur\_pos + 1) % LSH\_HIST\_SIZE;

if (next\_pos != 0 && history[cur\_pos + 1] != NULL)

{

offset = (cur\_pos + offset) % LSH\_HIST\_SIZE;

}

else

{

offset--;

}

if (history[offset] == NULL)

{

fprintf(stderr, "No such command in history\n");

exit(EXIT\_FAILURE);

}

command = malloc(sizeof(history[cur\_pos]));

strcat(command, history[offset]);

cmd\_args = lsh\_split\_line(command);

int i;

for (i = 0; i < lsh\_num\_builtins(); i++)

{

if (strcmp(cmd\_args[0], builtin\_str[i]) == 0)

{

return (\*builtin\_func[i])(cmd\_args);

}

}

return lsh\_launch(cmd\_args);

}

else

{

perror("lsh");

}

}

}

/\* Execute the parsed arguments \*/

int lsh\_execute(char \*line)

{

int i;

// printf("%s\n", line);

char \*\*args = lsh\_split\_line(line);

if (args[0] == NULL)

{ /\* empty command was entered \*/

return 1;

}

else if (strcmp(args[0], "history") == 0 ||

strcmp(args[0], "!!") == 0 || args[0][0] == '!')

{

return lsh\_history(args);

}

cur\_pos = (cur\_pos + 1) % LSH\_HIST\_SIZE;

history[cur\_pos] = malloc(cur\_bufsize \* sizeof(char));

char \*\*temp\_args = args;

int count = 0;

while (\*temp\_args != NULL)

{

strcat(history[cur\_pos], \*temp\_args);

strcat(history[cur\_pos], " ");

temp\_args++;

}

// printf("%s\n", history[cur\_pos]);

// history[count] = '\0';

if (cur\_pos > -1) // changed to >-1

printf("Inserted %s\n", history[cur\_pos]); // changed cur\_pos - 1 to cur\_pos

for (i = 0; i < lsh\_num\_builtins(); i++)

{

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

{

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

}

}

return lsh\_launch(args);

}

/\* Read input from stdin \*/

char \*lsh\_read\_line(void)

{

cur\_bufsize = LSH\_RL\_BUFSIZE;

int position = 0;

char \*buffer = malloc(sizeof(char) \* cur\_bufsize);

int c;

if (!buffer)

{

fprintf(stderr, "lsh: allocation error\n");

exit(EXIT\_FAILURE);

}

while (1)

{

/\* Read a character \*/

c = getchar();

if (c == EOF || c == '\n')

{

buffer[position] = '\0';

return buffer;

}

else

{

buffer[position] = c;

}

position++;

/\* If buffer exceeded, reallocate buffer \*/

if (position >= cur\_bufsize)

{

cur\_bufsize += LSH\_RL\_BUFSIZE;

buffer = realloc(buffer, cur\_bufsize);

if (!buffer)

{

fprintf(stderr, "lsh: allocation error\n");

exit(EXIT\_FAILURE);

}

}

}

}

/\* Loop for getting input and executing it \*/

void lsh\_loop(void)

{

char \*line;

int status;

do

{

printf(">");

line = lsh\_read\_line();

status = lsh\_execute(line);

free(line);

} while (status);

printf("exit the loop.");

}

int main(void)

{

lsh\_loop();

return EXIT\_SUCCESS;

}