**OS & SP - MINI PROJECT**

**Name : Mini Linux Shell Interpreter**

**Problem Solution**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/wait.h>

#include <ctype.h>

#include <fcntl.h>

#define LSIZ 128

#define RSIZ 10

int process\_str(char \*str, char arglist[100][100], int \*nargs)

{

int i = 0, j = 0, k = 0;

while (str[i] != '\0')

{

if (str[i] == '\n')

{

i++;

continue;

}

else if (str[i] != ' ')

{

arglist[j][k++] = str[i++];

}

else

{

(\*nargs)++;

arglist[j][k] = '\0';

j++;

k = 0;

i++;

}

}

return 0;

}

int main(int argc, char \*argv[])

{

int n, argno = 0, i = 0, fd1, fd2, count, pid, l;

char str[100], arglist[100][100];

char path[100], buff[100], path1[100], path2[100];

char line1[RSIZ][LSIZ];

int cnt=0;

char history[50][100];

while (1)

{

printf("Welcome to Mini-Shell\n");

printf("Please Enter your command below\n");

strcpy(path, "./");

argno = 0;

fflush(stdin);

n = read(0, str, 100);

str[n] = '\0';

process\_str(str, arglist, &argno);

cnt=cnt%50;

strcpy(history[cnt++],str);

for (i = 0; i < argno; i++)

{

printf("arguments passed = %s\n", arglist[i]);

}

if (strcmp(arglist[0], "mycp") == 0) //copies one file into another

{

pid = fork();

if (pid == 0)// child process

{

strcat(path, arglist[1]);

fd1 = open(path, O\_RDONLY);

if (fd1 < 0)

{

printf("Open file error\n");

exit(1);

}

strcpy(path, "./");

strcat(path, arglist[2]);

printf("Destination path: %s \n", path);

fd2 = open(arglist[2], O\_RDWR | O\_CREAT, 0666);

if (fd2 < 0)

{

printf("Open file error\n");

exit(1);

}

while ((n = read(fd1, buff, 100)) > 0)

{

write(fd2, buff, n);

}

close(fd1);

close(fd2);

}

else if (pid == -1) // parent process

{

printf("Command execution failed\n");

}

else

{

wait(NULL);// waits for mycp child implementation

printf("shell> ");

}

}

else if (strcmp(arglist[0], "mycat") == 0)//reads a file and prints the content of the file

{

int pipefds[2];

if (pipe(pipefds) == -1)

{

printf("Pipe creation error\n");

exit(1);

}

pid\_t pid1 = fork();

if (pid1 == 0) // child process

{

FILE \*fptr;

if ((fptr = fopen(arglist[1], "r")) == NULL)

{

printf("File pointer error\n");

exit(1);

}

char singleLine[150];

close(pipefds[0]);

while (fgets(singleLine, sizeof(singleLine), fptr))

{

write(pipefds[1], singleLine, strlen(singleLine));

}

int error = ferror(fptr);

fclose(fptr);

if (error)

{

printf("error reading file\n");

}

}

else if (pid1 > 0) // parent process

{

close(pipefds[1]);

ssize\_t n;

while ((n = read(pipefds[0], buff, sizeof(buff) - 1)) > 0)

{

buff[n] = '\0';

printf("%s", buff);

}

close(pipefds[0]);

}

}

else if (strcmp(arglist[0], "mymv") == 0) // renames a file

{

pid = fork();

if (pid < 0)// child process

{

printf("Command execution failed\n");

}

if (pid == 0)

{

strcpy(path1, "./");

strcpy(path2, "./");

strcat(path1, arglist[1]);

strcat(path2, arglist[2]);

rename(path1, path2);

}

else

{

wait(NULL); // waits for child process to execute

printf("Command execution completed!\n");

}

}

else if (strcmp(arglist[0], "myrm") == 0)// removes a file

{

pid = fork();

if (pid < 0)

{

printf("Command execution failed\n");

}

if (pid == 0) //child process

{

strcpy(path1, "./");

strcat(path1, arglist[1]);

remove(path1);

}

else

{

wait(NULL); //waits for the child process to execute

}

}

else if (strcmp(arglist[0], "myhead") == 0) // reads first 10 lines of the file

{

FILE \*fptr = NULL;

int i = 0;

int tot = 0;

fptr = fopen(arglist[1], "r");

while (fgets(line1[i], LSIZ, fptr))

{

line1[i][strlen(line1[i]) - 1] = '\0';

i++;

}

tot = i;

for (i = 0; i < 10; ++i)

{

printf(" %s\n", line1[i]);

}

fclose(fptr);

}

else if (strcmp(arglist[0], "mytail") == 0) // reads last 10 lines of the file

{

FILE \*fptr = NULL;

int i = 0;

int tot = 0;

fptr = fopen(arglist[1], "r");

while (fgets(line1[i], LSIZ, fptr))

{

line1[i][strlen(line1[i]) - 1] = '\0';

i++;

}

tot = i;

for (i = tot - 10; i < tot; ++i)

{

printf("%s\n", line1[i]);

}

printf("\n");

fclose(fptr);

}

else if (strcmp(arglist[0], "exit") == 0) // exits from the program

{

printf("exited!!!!!!!!!!!!!!!!");

// break;

// exit(0);

return 0;

}

else if (strcmp(arglist[0], "history") == 0) // prints the commands used in the program

{

printf("My version of History commands\n");

int a=0;

for (int i=0;i < cnt; i++)

{

a=i+1;

printf("<%d> %s \n",a, history[i]);

}

}

}

}