1. 개요

OS 커널과 상호작용하여 새로운 프로세스를 실행할 수 있는 쉘을 구현하였습니다. 환경변수 PATH에 등록된 디렉토리를 참고하여 흔히 사용하는 cat 명령어나 ls 명령어 등 사용할 수 있도록 하였고, PATH에 등록된 디렉토리에 프로그램이 존재하지 않는 경우 쉘과 같은 디렉토리에 존재하는 mybin 디렉토리에서 프로그램을 찾아 실행하도록 하였습니다.

1. 프로그램 구조 설명
   1. 함수에 대한 설명
2. myshell.c

int main();

* 시작 프로세스
* gettoken 함수를 실행하여 사용자에게 명령어를 입력받아 execute 함수에 전달
* 사용자가 quit 입력 시 프로세스 종료

char\*\* gettoken(char\*);

* 사용자가 입력한 문자열을 strtok 함수를 이용해 분리하여 문자열 배열에 저장
* exec 함수의 인자인 문자열 배열 argv[]는 명령행 인자의 끝을 알려주기 위해 마지막 원소로 NULL이 저장되어야 하므로 문자열 배열의 끝에 NULL 저장, 문자열 배열 반환

int execute(char\*\*);

* fork 함수를 이용하여 새로운 프로세스 생성, exec 함수를 이용하여 원하는 프로그램 실행
* execvp 함수를 이용하여 환경변수 PATH에 등록된 디렉토리에 실행하려는 프로그램이 존재하는 경우 실행
* PATH에 실행하려는 프로그램이 존재하지 않는 경우, 쉘과 같은 디렉토리에 존재하는 mybin 디렉토리에서 프로그램을 찾아 실행

1. add.c

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

* 인자로 입력받은 정수들의 합을 1 + 2 + 3 = 6 형태로 출력
* 인자의 개수 제한 없음

1. sub.c

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

* 인자로 입력받은 정수들의 차를 1 – 2 = -1 형태로 출력
* 인자의 개수 2개로 제한
* 2개 이상의 인자 입력 시 “Too many arguments” 출력

1. mul.c

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

* 인자로 입력받은 정수들의 곱을 1 \* 2 \* 3 = 6 형태로 출력
* 인자의 개수 제한 없음

1. div.c

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

* 인자로 입력받은 정수들의 몫을 2 / 1 = 2 형태로 출력
* 인자의 개수 2개로 제한
* 2개 이상의 인자 입력 시 “Too many arguments” 출력
  1. 다이어그램

텍스트, 전자기기이(가) 표시된 사진

자동 생성된 설명

1. 실행 결과

실패 1)

텍스트이(가) 표시된 사진

자동 생성된 설명

quit 명령어를 입력하였음에도 바로 쉘이 종료되지 않는 문제가 발생하였습니다. “MyShell>”까지 출력된 상태에서 사용자의 입력을 기다리고 있으므로 메인 함수는 정상적으로 동작하고 있다고 가정한 후 quit 명령어를 여러 번 입력하였습니다. 몇 번의 테스트 후에 명령어를 정상적으로 실행시키지 못한 횟수만큼 quit 명령어를 입력해야 쉘이 종료되는 것을 확인하였습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

명령어가 정상적으로 실행되지 않은 경우 fork 함수를 통해 생성한 새로운 shell 프로세스를 종료하지 않아 발생하는 문제였습니다. 명령어가 정상적으로 실행되지 않은 경우 exec 함수가 실행되지 않기 때문에 동적할당한 메모리가 여전히 남아있으므로 free(args) 함수를 실행하여 gettoken 함수에서 할당한 문자열 배열 args를 해제하고 exit(EXIT\_FAILURE) 함수를 실행하여 정상적으로 실행되지 않았음을 알리면서 프로세스를 종료하도록 수정하였습니다.

실패 2)

텍스트이(가) 표시된 사진

자동 생성된 설명

cd 명령어가 정상적으로 동작하지 않는 문제가 발생하였습니다. 검색을 통해 cd 명령어는 현재 작업중인 디렉토리를 바꾸는 것이지 프로그램을 실행하는 것이 아니기 때문에 exec 함수를 이용해 프로그램을 실행하는 형식이 아니라 쉘 내부에서 현재 작업중인 디렉토리를 바꾸는 코드를 작성하는 형식으로 구현해야 함을 알게 되었습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

fork 함수를 실행하기 전에 명령어가 cd인지 검사하여 만약 cd 명령어가 입력되었다면 chdir 함수를 이용해 현재 작업중인 디렉토리를 변경하고 디렉토리가 변경된 쉘 프로세스에서 계속 작업해야 하기 때문에 exit 함수로 프로세스를 종료하지 않고 return 함수를 이용해 chdir 함수가 잘 실행되었는지 여부만 반환하도록 하였습니다.

실패 3)

텍스트이(가) 표시된 사진

자동 생성된 설명

cd 명령어를 이용하여 현재 디렉토리를 변경하자 mybin 디렉토리를 찾지 못하는 문제가 발생하였습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

“./mybin” 경로에서 사용자 지정 쉘 명령어를 찾아 실행하는데 현재 디렉토리를 변경하자 그 디렉토리에서 mybin 디렉토리를 찾기 때문에 문제가 발생하였습니다. 쉘 프로세스를 실행할 때 argv[0]을 통해 시작 경로를 받아와 이용해보려 했으나 윈도우와 다르게 유닉스에서는 argv[0]이 절대 경로가 아니라 “./” 형태로 상대 경로로 작성되어 있음을 확인하였습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

쉘 실행 직후 현재 작업 디렉토리의 절대 경로를 반환하는 getcwd 함수를 이용하여 전역변수에 최초 실행 경로를 저장하고 execute 함수에서 활용하는 형식으로 문제를 해결하였습니다.

성공)

텍스트이(가) 표시된 사진

자동 생성된 설명

ls 명령어, cat 명령어 실행

텍스트이(가) 표시된 사진

자동 생성된 설명

vim 실행

텍스트이(가) 표시된 사진

자동 생성된 설명

vim 종료, cd 명령어 실행, 디렉토리 변경 후 add, sub 명령어 실행

텍스트이(가) 표시된 사진

자동 생성된 설명

최초 실행한 디렉토리에서 mul, div 명령어 실행, quit 명령어로 쉘 종료

1. 고찰

프로그램을 실행하는 함수인 exec 함수는 전달받는 인자에 따라 여러 종류가 존재함을 알게 되었습니다. 나열형으로 전달받을지, 배열형으로 전달받을지에 따라 l, v로 나누어지고 PATH를 참조하는지 여부에 따라 p, 환경변수를 넘겨받는지 여부에 따라 e가 exec 뒤에 붙어 필요한 경우에 따라 사용되는 것을 알게 되었습니다. 쉘 명령어에 사용되는 인자들 중에 기본적으로 적용되는 것들이 있는데 그런 인자들이 배열형으로 미리 정의되어 전달되는 것이 아닌가 짐작해보았습니다.

쉘에서 사용되는 명령어들은 모두 프로그램 형태로 구현되어 있는 것이 아니라 cd, quit 등 프로세스를 프로그램 형태로 구현할 수 없는 명령어들은 쉘에 내장된 형태로 구현되어 있음을 알게 되었습니다.

이전에는 비주얼 스튜디오 등 IDE를 사용하여 편하게 디버깅 작업을 할 수 있었지만 프로세스가 몇 개 실행되었는지 등 시스템과 관련된 작업은 IDE를 이용해 확인하는데 어려움이 있었습니다. 이런 시스템을 다루는 코드가 잘 동작하는지 확인하기 위해 ps 등의 쉘 명령어를 더 능숙하게 사용할 수 있도록 연습해야 되겠다 생각했습니다. 그리고 편리한 GUI 환경이 아닌 우분투 쉘에서도 디버깅 작업을 할 수 있도록 방법을 알아봐야 되겠다 느꼈습니다.

1. 프로그램 소스 코드

myshell.c

#include <sys/wait.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#define MAXARGC 20

#define MAXLINE 80

int execute(char\*\*);

char\*\* gettoken(char\*);

char\* cwd; // initial working directory

int main()

{

char line[MAXLINE], \* token;

char\*\* args;

int status = 0;

printf("My Shell Started.... \n");

/\*if argument buf == NULL,

getcwd() fuction automatically malloc memory\*/

cwd = getcwd(NULL, 0);

while (1) {

printf("MyShell>");

fgets(line, MAXLINE, stdin);

if (strncmp(line, "quit", 4) != 0) {

args = gettoken(line);

status = execute(args);

printf("\n");

}

else {

free(cwd); // free memory malloc'd by getcwd()

exit(1);

}

} /\* end of while\*/

free(args);

return(1);

}

char\*\* gettoken(char\* line)

{

int idx = 0;

char delim[] = " \t\r\n";

char\*\* tokenList = malloc(MAXLINE);

char\* token;

if (!tokenList) {

printf("malloc error \n");

exit(EXIT\_FAILURE);

}

token = strtok(line, delim);

while (token != NULL) {

tokenList[idx] = token;

idx++;

if (idx > MAXARGC - 1) {

printf("Too many args \n");

break;

}

token = strtok(NULL, delim);

} // end of while

tokenList[idx] = NULL; // insert end of string mark

return tokenList;

} // end of gettoken

int execute(char\*\* args)

{

pid\_t pid;

int status;

int i = 1;

char\* path; // directory for custom commands

if (args[0] == NULL) { // empty command

return 1;

}

printf(" %s ", args[0]);

while (args[i] != NULL) {

printf(" %s ", args[i]);

i++;

}

printf("\n");

// if command is "cd", no need to create a new process

if (strcmp(args[0], "cd") == 0) {

status = chdir(args[1]); // use chdir() fuction to move directory

free(args); // prepare to receive new commands

if (status == 0) {

return 0; // continue working in a shell process

}

else {

perror("");

return 1; // continue working in a shell process

}

}

// source : https://nomad-programmer.tistory.com/106

pid = fork(); // create new process

if (pid < 0) { // failed to create process

perror("");

}

else if (pid == 0) { // child process

status = execvp(args[0], args); // searching PATH directory

if (status < 0) {

// searching mybin directory

path = malloc(strlen(cwd) + strlen("/mybin/") + strlen(args[0]) + 1);

strcpy(path, cwd);

strcat(path, "/mybin/");

strcat(path, args[0]);

status = execv(path, args);

/\*If the program doesn't exist,

clean up resources of child shell processes\*/

if (status < 0) {

perror("");

free(path);

free(args);

free(cwd);

exit(EXIT\_FAILURE);

}

}

}

else { // parent process

// parent will wait for the child to complete

wait(NULL);

}

free(args);

return (1);

}// end of execute

add.c

#include <stdio.h>

#include <stdlib.h>

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

{

int num = 0;

int result = 0;

num = atoi(argv[1]);

result += num;

printf("%d ", num);

for (int i = 2; i < argc; i++) {

num = atoi(argv[i]);

result += num;

printf("+ %d ", num);

}

printf("= %d\n", result);

return 0;

}

sub.c

#include <stdio.h>

#include <stdlib.h>

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

{

int num = 0;

int result = 0;

if (argc > 3) {

printf("Too many arguments\n");

return 1;

}

num = atoi(argv[1]);

result = num;

printf("%d ", num);

num = atoi(argv[2]);

result -= num;

printf("- %d ", num);

printf("= %d\n", result);

return 0;

}

mul.c

#include <stdio.h>

#include <stdlib.h>

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

{

int num = 0;

int result = 0;

num = atoi(argv[1]);

result = num;

printf("%d ", num);

for (int i = 2; i < argc; i++) {

num = atoi(argv[i]);

result \*= num;

printf("\* %d ", num);

}

printf("= %d\n", result);

return 0;

}

div.c

#include <stdio.h>

#include <stdlib.h>

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

{

int num = 0;

int result = 0;

if (argc > 3) {

printf("Too many arguments\n");

return 1;

}

num = atoi(argv[1]);

result = num;

printf("%d ", num);

num = atoi(argv[2]);

result /= num;

printf("/ %d ", num);

printf("= %d\n", result);

return 0;

}

1. 자료 출처

“[Programming/C] exec 함수 사용”, nomad-programmer, 2020.6.19, https://nomad-programmer.tistory.com/106