1. 개요

리눅스 환경에서 one line으로 입력을 받아 fork와 exec 계열 함수를 사용해서

ls, mkdir 등의 명령어를 수행하는 쉘 프로그램을 만들어 보도록 한다.

2. 프로그램 구조 설명

2.1 함수에 대한 설명

-command.h

// ls와 mkdir 함수를 정의한 헤더이다.

int ls(char\* factor)

// main.c에서 one line으로 읽어들인 입력을 토큰으로 나눴을 때 첫번째 요소가 “ls”인 경우 실

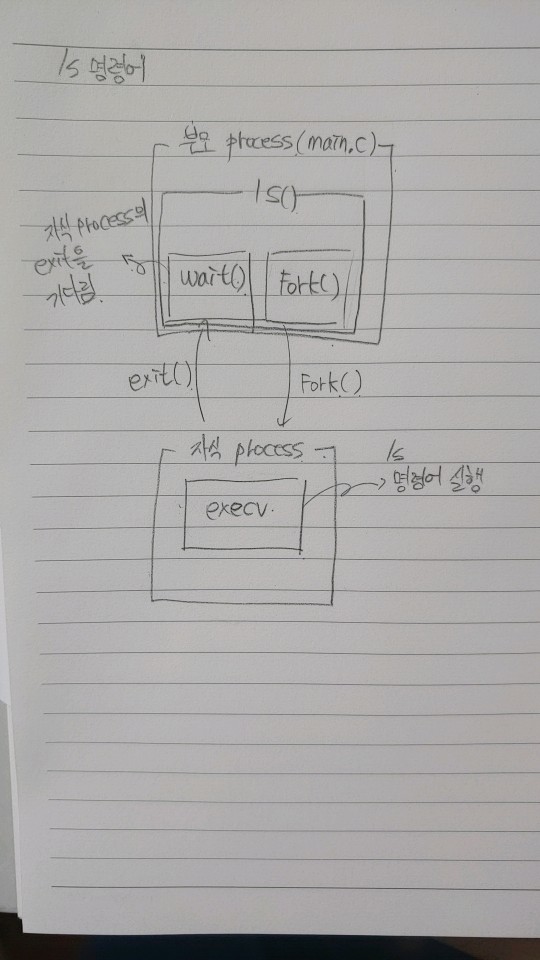
행 되는 함수. 이 함수 내부에서도 입력을 토큰으로 좀더 나눠서 ls 이후에 다른 옵션이 있을

경우 argv 인자에 추가하여 쉘프로그램에서 인식할 수 있게 해준다.

- int mkdir(char\* factor)

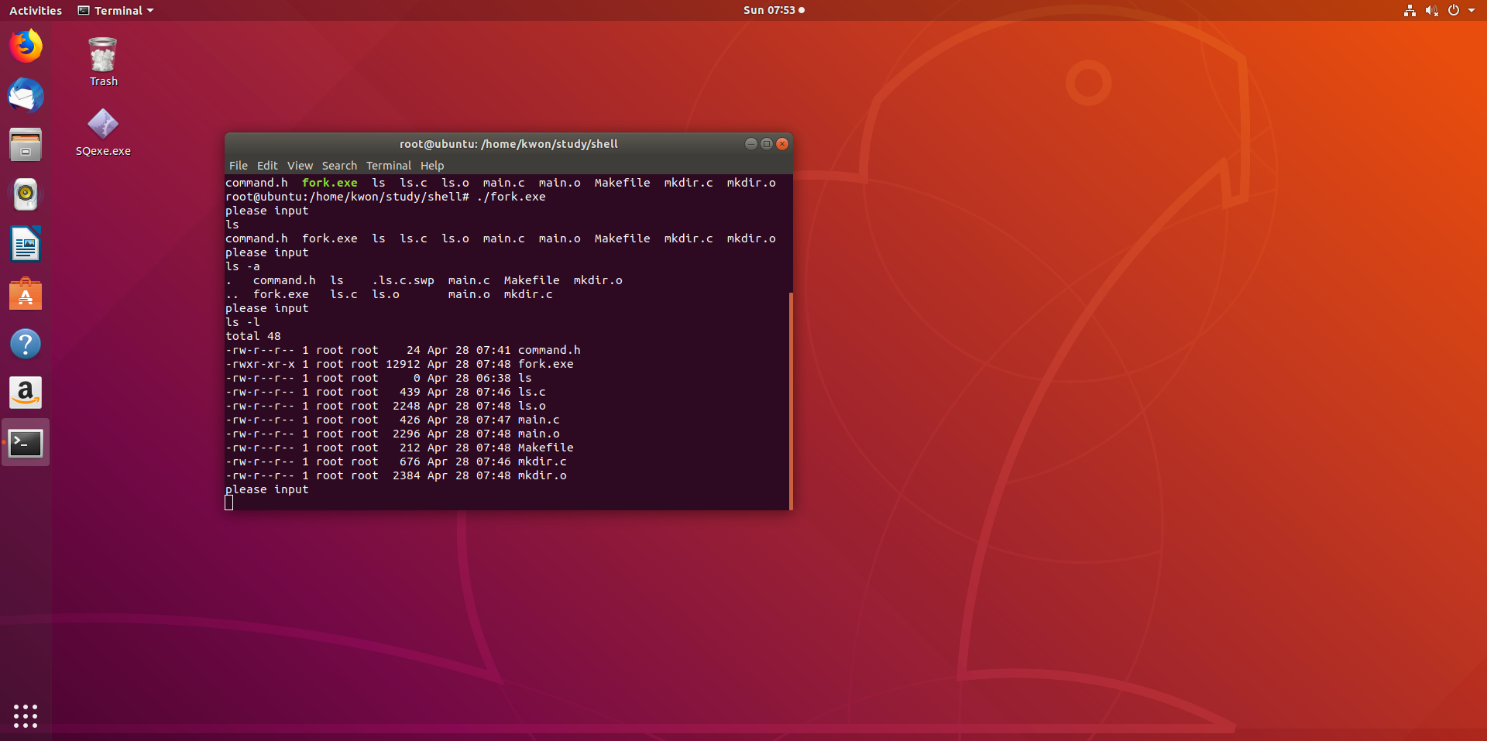
// 구조적으로 ls와 거의 유사합니다.

2.다이어그램

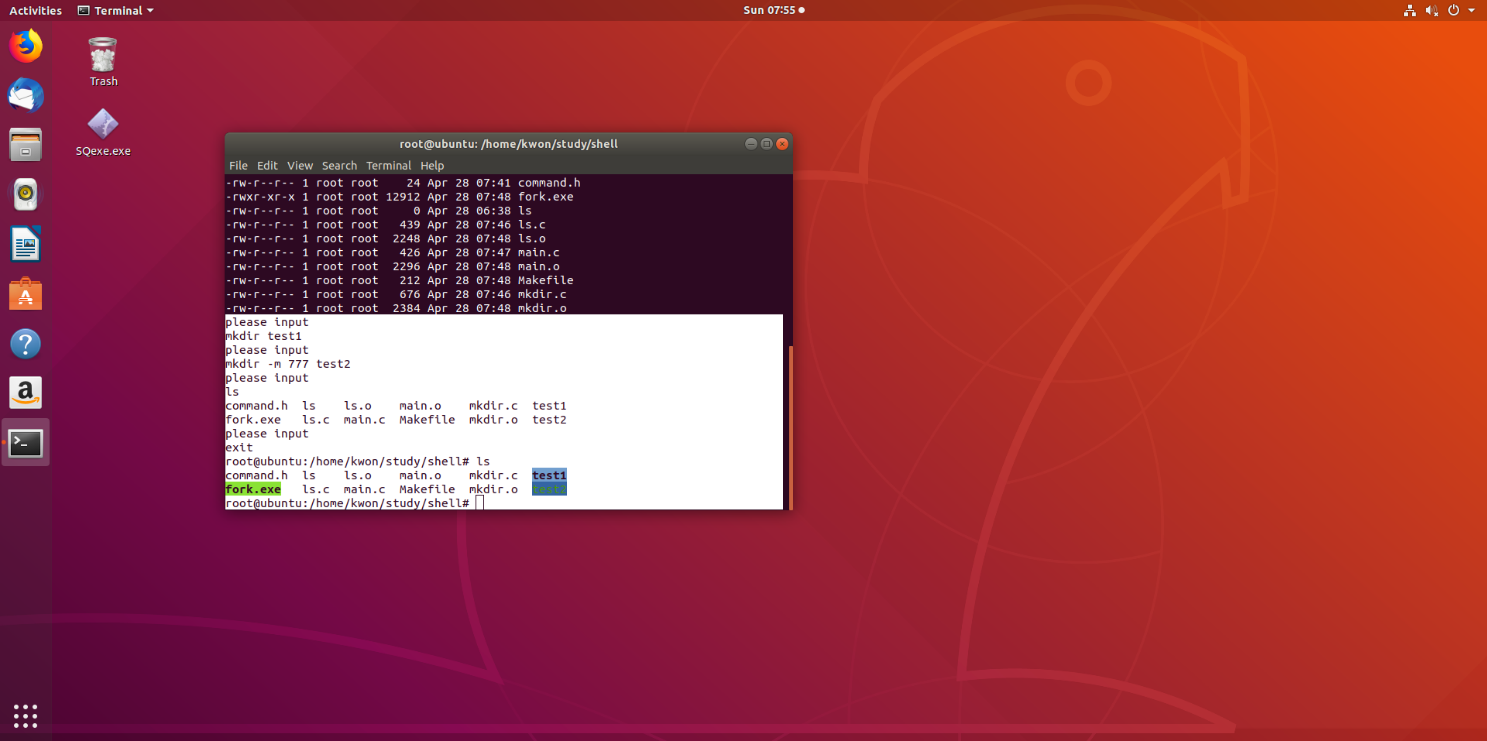


3. 실행 결과

- ls 결과



- mkdir 결과



**(실행결과는 자신의 컴퓨터환경에서 실행한 결과 캡쳐(그림파일)해서 문서에 첨부하여 제출합니다. 실행결과가 성공한 경우뿐만 아니라 실패한 경우도 포함되어야 함.**

**만약 과제로 올려서 실행한 결과와 실행 결과로 캡쳐한 결과가 다를 경우 감점합니다.)**

4. 고찰 (제일 중요함 - **점수배점이 가장 높음!** 자세히 쓸 것)

이번 프로그래밍은 저번 과제였던 queue와 stack 프로그래밍할 때보다 훨씬 리눅스 명령어를 사

용하는 실력이 늘어서 인지 리눅스에서 코딩하는 것이 그렇게 불편하지 않았습니다. 하지만 생소

한 fork()와 exec관련 함수를 쓰는 것이 힘들었습니다. 분명 교수님이 가르쳐 주실 때만해도 할 수

있을 것 같았지만, 직접 부딪혀보니 왜 이렇게 코딩하는 것인지, 리턴 값과 인자들은 무슨 의미를

가진 것인지를 파악하는데 시간이 꽤 걸렸습니다. 이번 기회로 교수님의 자료들과 인터넷에서

다양한 자료를 찾아보면서 어느정도 프로세스에 대한 이해도가 높아졌고

그 이후에는 복잡한 코딩을 요하는 프로그램이 아니라 코딩에는 큰 어려움이 없었습니다.

5. 프로그램 소스 파일 (주석을 꼭 달 것)

-command.h

int ls(char\* factor);

int mkdir(char\* factor);

-main.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

#include "command.h"

#define SIZE 20

int main(){

char input[SIZE];

char token[SIZE];

while(true){

printf("please input\n");

gets(input); // 한줄 읽기

strcpy(token,input);// input을 토큰으로 끊으면 2번째 인자를 활용할 수 없으므로

// input과 내용이 똑같은 token 문자열을 만들어 활용한다.

strtok(token," "); // token을 “ “로 토큰 실행

if(!strcmp("ls",token)){ //토큰으로 잘린 첫번째 내용이 ls면

ls(input);

}

if(!strcmp("mkdir",token)){// 토큰으로 잘린 첫번째 내용이 mkdir이면

mkdir(input);

}

if(!strcmp("exit",token)){ // exit는 종료

break;

}

}

return 0;

}

-ls.c

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include "command.h"

int ls(char\* factor){

int pid, stat;

char\* argv[3];

argv[0] = strtok(factor," "); // strtok으로 인자들을 넣어줌

argv[1] = strtok(NULL," ");

argv[2] = 0;

pid = fork();

switch(pid) {

case -1: //fork가 에러가 나면

printf("fork error\n");

exit(1);

case 0: //현재 프로세스가 child이면

execv("/bin/ls",argv); // 프로세스에 기존 코드는 없애고 /bin/ls의 기능을 실음

printf("exec error \n");

exit(1);

default://현재 프로세스가 parent이면

wait(&stat);

}

return 0;

}

-mkdir.c

// ls와 거의 동일

#include <stdio.h>

#include <unistd.h>

#include <string.h>

#include "command.h"

int mkdir(char\* factor){

int stat,pid;

char\* argv[4];

argv[0] = strtok(factor," ");

argv[1] = strtok(NULL," ");

argv[2] = strtok(NULL," ");

argv[3] = strtok(NULL," ");

pid = fork();

switch(pid) {

case -1: // error

printf("fork error\n");

exit(1);

case 0: // child

execv("/bin/mkdir",argv);

printf("exec error \n");

exit(1);

default:// parents

wait(&stat);

}

return 0;

}

6. 자료 출처 (어떤 책을 참고했으며 어떤 웹사이트를 참고했는지 출처를 적습니다)

<https://blog.naver.com/on21life/221446796605>

<https://www.joinc.co.kr/w/man/2/fork>

<https://bbolmin.tistory.com/35>

등등을 참고했습니다.