System Software Experiment 2
2014313299 Park Sejin

designing swsh shell

swsh shell 제작

내가 제작한 swsh 프로그램은 최대한 bash shell과 비슷한 동작을 할 수 있도록 만들었다. 과제의 4가지 종류의 command 외에 일반적으로 사용할 수 있는 모든 command가 올바른 입력으로 주어진다면 실행가능하다.

Make

기본적으로 shell에 들어가는 모든 입력이 execv함수를 이용하여 동작할 수 있게 만들었다. 그러나 command 2 같은 경우에는 내가 작성한 함수로 동작을 시키지만 기존의 shell과 같은 동작을 할 수 있도록 만들기 위해 새로운 c 파일을 만들고 컴파일을 통해 실행가능한 파일로 만들었다. 그리고 기존 execv에서 bin/... 의 경로에 있는 shell 파일을 실행 시키지 않고 현재 swsh가 존재하는 폴더 경로에 있는 실행 파일을 실행하게 했다.

그래서 모든 실행파일을 컴파일을 하기 위해 make 명령을 작성하고 ./swsh으로 실행을 시킨다.

SWSH Shell Directory structure

2014313299(folder) - swsh.c(file), swsh(executable file)

cat.c(file), cat(executable file)

cp.c(file), cp(executable file)

head.c(file), head(executable file)

mv.c(file), mv(executable file)

pwd.c(file), pwd(executable file)

rm.c(file), rm(executable file)

tail.c(file), tail(executable file)

makefile

command file

1. cat.c

   cat filename

   cat은 두 가지 경우가 있다. 먼저 filename이 argument로 들어 올 경우, pipeline으로 input으로 들어올 경우.

   두 가지 모두 파일을 읽어서 표준 출력으로 출력하는 것은 동일하지만, file read를 파일 이름으로 할 지, 표준 입력으로 할지를 정해서 입력받게 된다.

2. cp.c

   cp는 두 개의 파일이름을 받아서 앞의 argument를 뒤의 파일로 복사하는 명령어이다.

   복사할 파일을 MAXLINE 씩 읽어서 쓸 파일에 읽어 들인 bytes만큼 작성한다.

3. head.c

   head는 세 가지 경우가 있다. 옵션의 유무와 file name이 한 가지 또는 여러개의 파일이 들어올 경우, pipeline으로 표준 입력으로 들어왔을 때 출력해주는 경우.

   먼저 head는 옵션의 유무를 확인하고 출력할 line을 확인한다. 그리고 파일의 개수에 따라 출력의 form을 다르게 해주었다. 마지막으로 file name이 들어오지 않아서 표준 입력으로 들어온 내용을 출력하는 경우에는 \n 의 개수를 내가 정한 line 수 만큼 들어왔을 때까지 출력한다.

4. mv.c

   mv의 경우 두 개의 파일 이름이 arguments로 들어왔을 때, 앞의 파일이름을 뒤의 파일 이름으로 변경해주는 rename함수를 이용해서 실행했다. 단 현재 파일 directory의 경로를 파일 이름에 붙여서 사용했다.

5. pwd.c

   pwd는 getcwd함수를 이용하여 현재 working directory의 경로를 표준 출력으로 출력한다.

6. rm.c

   rm은 unlink 함수를 이용해 working directory 경로에 내가 지울 파일을 붙여서 파일의 절대 경로를 인자로 넘겨서 파일을 지운다. 만약 파일 지우는 것이 오류나면 errno에 따라 error의 문구를 출력한다.

7. tail.c

   tail도 head와 매우 유사하다. 세가지 경우: 옵션 유무, 파일 개수, 표준 입력 이 세가지 경우를 처리하고 표준 출력으로 나타내준다.

swsh file

1. header file

```
#define MAXPATH 100
#define MAXARGS 128
#define MAXLINE 256
#define MAXPIPE 30
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <fcntl.h>
#include <errno.h>
#include <pwd.h>
#include <sys/types.h>
#include <sys/wait.h>
```

2. 추가된 함수

```
void eval(char *cmdline);
char *which_command(char **argv);
void changedir(char *argv);
void pipeline(char **argv, int bg);
int parseline(char *buf, char **argv);
int builtin_command(char **argv);
void binding(char **argv);
void phandler_int(int sig);
void chandler_int(int sig);
void chandler_stp(int sig);
void ignore(int sig);
char condir[MAXPATH];
pid_t masterpid;
pid_t chdpid, mstpid;
```

3. main 함수

메인 함수에서 추가된 부분은 3가지가 있다.

먼저, 이 실행파일의 절대 경로를 realpath 함수를 이용해 condir이라는 전역 변수에 담았다.

그리고 이 swsh의 가장 상위 pid를 전역 변수인 masterpid에 저장해 놓았다.

마지막으로 SIGINT와 SIGTSTP를 handler로 처리하게 하여 ctrl+c와 ctrl+z를 주어진 역할을 할 수 있게 만들었다.

4. eval 함수

eval 함수는 더 간략하게 command line을 parse하여 내가 만든 pipeline함수로 인자들을 넘기기만 할 수 있도록 만들었다.

5. which_command 함수

execl 함수를 사용하여 내가 사용하려는 command가 기본 bash에서 지원하는 명령어라면 which 명령어를 이용해 위치를 찾고 그 위치를 반환해 pipeline함수에서 execv 함수의 인자로 이 path에 있는 실행파일을 실행하게 하도록 했다. 다만 command_2나 내가 구현해야하는 명령어의 경우 실제 bash에서 지원한다고 하더라도 내가 만든 함수들을 이용하기 위해 main 함수에서 이 실행파일들이 존재하는 경로를 나타내는 condir 변수로 전체 경로를 반환하게 했다.

which command 는 fork한 후에 execl 함수로 실행시켰다.

7. changedir 함수

changedir 함수는 cd 명령어가 들어왔을 때 현재 working directory를 변경해주는 기능을 한다. 함수 인자로 argv를 받는데 이 인자가 없는 경우에는 home directory로 이동하고 인자가 존재하는 경우에는 ~/...(home directory에 대한 상대경로)로 되어 있는 경우 /...(root directory에 대한 상대경로)로 되어있는 경우, 현재 working directory에 대한 상대경로로 인자가 들어오는 경우로 나누어 strcpy, strcat 등의 함수를 이용해 내가

이동하려고 하는 폴더의 전체 경로를 chdir 함수로 변경한다. 만약 에러 시 과제에서 명시되어 있는 5가지 경우는 error 메시지를 출력하고 그 외에는 에러 번호를 출력한다.

8. binding 함수

이 binding 함수의 경우에는 작은 따옴표나 큰 따옴표로 argument가 묶여있는 경우 하나로 만들어주고 이 따옴표들을 없애는 기능을 하는 함수이다.

모든 인자들을 확인하면서 작은 따옴표로 시작하는 게 있다면, 작은 따옴표가 다시 나올 때까지 모든 인자들을 하나로 묶는다. 큰 따옴표도 마찬가지이다. grep이나 awk 명령어의 경우 따옴표를 제거하지 않고 실행하면 제대로 실행되지 않는 것을 고려하기 위해 만든 함수이다.

9. handler

내가 만든 swsh는 크게 부모 프로세스가 하나의 자식 프로세스를 부르고 이 자식 프로세스는 자신의 pid와 group pid가 같도록 만든 후 이 자식 프로세스에서 command를 실행하기 위해 자식프로세스를 fork하여 실행하고 종료하는 것을 반복한다.

phandler_int 함수는 가장 처음 프로세스인 부모 프로세스에서 SIGINT 나 SIGTSTP 신호를 받으면 부모 프로세스에서는 이 신호를 무시하고 자식 프로세스들에게만 이 신호를 그대로 전달하는 handler이다.

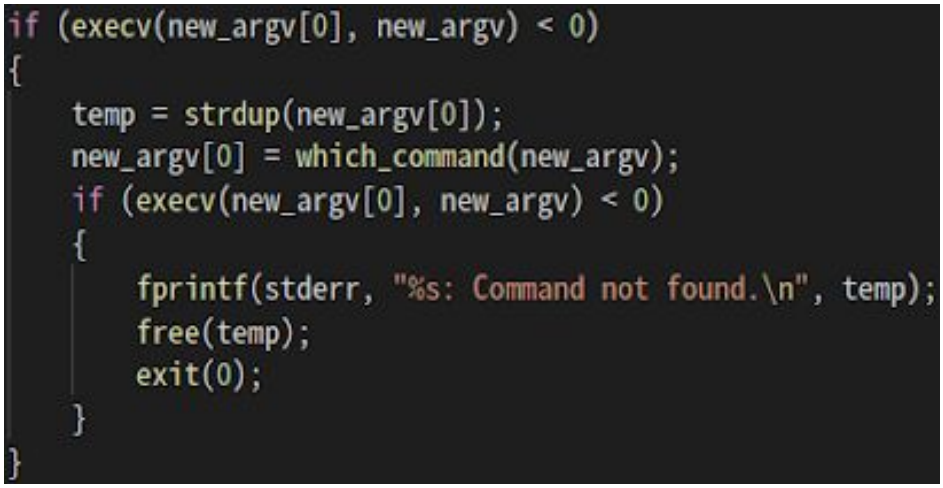
chandler_int 함수는 자식 프로세스에서 SIGINT 신호를 받으면 가장 상위의 자식 프로세스에서 이 handler 함수를 호출하게 되는데, 같은 group pid를 가지고 있는 자식 프로세스들을 모두 수거한 후 exit 함수로 종료하는 역할을 한다.

chandler_stp 함수는 자식 프로세스에서 SIGTSTP 신호를 받았을 때, 같은 group pid를 가지고 있는 자식 프로세스 중 현재 정지되어 있는 프로세스의 pid를 받아 SIGINT 신호를 전해주고 모든 자식 프로세스를 수거한 후 exit 함수로 종료한다.

10. pipeline

이 pipeline 함수는 가장 먼저 parse되어 있는 모든 command들이 |라는 기호로 나뉘어 new_argv에 새로 저장된다. cd와 exit 경우 |과 같이 실행되면 아무런 실행을 안하기 때문에 우선 pip 변수를 통해 pipeline 실행을 하는 지 파악하고 cd나 exit command를 실행한다.

그리고 크게 fork를 한 후 setpgrp 함수로 맨 처음 생성된 자식 프로세스는 자신의 pid와 group pid가 같게 만들어 준 후, while으로 새로 parse 된 new_argv로 which_command와 execv함수를 이용해 command를 실행시킨다.



pipeline이 존재하는 경우, 자식 프로세스의 매 fork마다 pipe를 열어 int** fd 를 설정해주고

서로 통신할 수 있도록 했다.

```
if (conn < pip)
{
    close(fd[conn][0]);
    dup2(fd[conn][1], 1);
}
else if (pipe_out != NULL)
{
    if (app)
    {
        if ((fout = open(pipe_out, O_CREAT | O_WRONLY | O_APPEND, 0755)) < 0)
            exit(0);
    }
    else
    {
        if ((fout = open(pipe_out, O_CREAT | O_WRONLY | O_TRUNC, 0755)) < 0)
            exit(0);
    }
    dup2(fout, 1);
}
if (conn > 0)
{
    close(fd[conn - 1][1]);
    dup2(fd[conn - 1][0], 0);
}
else if (pipe_in != NULL)
{
    if ((fin = open(pipe_in, O_RDONLY)) < 0)
        exit(0);
    dup2(fin, 0);
}
```

그리고 redirection은 new_argv로 parsing할 때, 이 신호가 존재하면 pipe_in, pipe_out 변수에 파일 이름을 받아 인자의 command의 실행 입/출력을 표준 입/출력 대신 바꿔주었다.

마지막으로 부모 프로세스에서는 이 맨 처음 생긴 자식 프로세스가 끝나길 기다린 후, 이 함수를 종료한다.

+ swsh를 실행시킬 때, 맨 처음 생성된 자식 프로세스를 자신의 pid와 gpid를 같게 만들었는데 이것을 모든 command에 적용하면 bc나 man과 같은 다른 프로그램을 실행시키는 command의 경우에는 제대로 실행이 되지 않는다는 문제점이 있다. 따라서 이것의 경우에만 setpgrp을 하지 않고 실행시켰고, 나머지 부분에서는 제대로 group pid가 묶인다.
+ pipe, signal, file I/O, process fork and handling 등을 제대로 사용하여 process 간 통신을 제대로 이해할 수 있었다.