쉘 구현 프로젝트 과제 #1

12171702 조대환

제출일 : 2021-11-07

1.요구사항 정의

1-1 cd 명령이 제대로 먹히지 않는 버그를 수정

텍스트이(가) 표시된 사진

자동 생성된 설명

1-2 exit 명령을 구현



1-3 백그라운드 실행을 구현

Foreground



Background

텍스트이(가) 표시된 사진

자동 생성된 설명

Background는 프롬프트가 바로 출력된다.

테스트사항

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

백그라운드 기다리는 테스트 문제

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

2.구현 방법 서술(구현 스크린샷 첨부)

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

1. Exit 구현
   * 1. Exit 명령이 들어올경우 fork 진행없이 현재 프로세스에서 바로 종료하도록 구현하였습니다.
2. cd 구현
   * 1. Cd가 명령으로 들어오게되면 fork를 진행하지않고 현재프로세스에서 작동을 하도록 구현했습니다. 현재 프로세스에서 chdir()을 통해 프로세스의 workingdirectory를 변경해주었습니다
3. Background 실행
   * 1. 명령어 뒤에 &가 붙을 경우 background 실행으로 인지하여 다른명령어들과는 다르게 fork를 두번 해준다. 손자 프로세스를 생성하여 명령을 수행하고 자식프로세스는 손자프로세스를 생성후 exit 하게된다.
     2. 이후 고아프로세스가 된 손자프로세스가 명령을 실행한다.

2-2 고찰

* 1. 좀비 프로세스가 생기는 이유
     1. 프로세스가 종료되어도 메모리상에서 지워지지 않는 경우 좀비 프로세스라고 한다. 이러한 좀비 프로세스가 생기는 이유는 프로세스가 종료된 후 부모프로세스에 종료했다는 것 을 반환 해야하는데 부모프로세스가 먼저 종료되어 메모리상에서 사라졌을 때 자식프로세스는 종료를 반환하지 못하여 메모리에 계속 남게 된다. 이러한 경우 좀비프로세스가 된다.

1. 백그라운드 기다리는 테스트 문제에서의 문제점과 문제점을 해결하기 위한 방법
   * 1. 고아 프로세스 생성원리를 통해 background에서 프로세스를 실행시켰기 때문에 좀비프로세스가 생기는 문제점을 가지고 있다. 이러한 좀비프로세스 를 해결하기 위해서는 해당 프로세스의 부모프로세스가 종료되는 시점에서 부모프로세스를 init프로세스로 연결하여 정상 종료하도록 해야한다.

3.구현 시 문제점

A. Ppt 예시와 다르게 구현한 코드의 백그라운드 실행에서는(ps -u 1000 &) 출력결과를 표준출력으로 출력하는 경우가 발생했다. 이문제를 해결하기 위해 dup2를 통해 dev/null로 stdout을 redirect하여 표준출력을 없애주었다.

B. wait(NULL)을 할 경우 여러 자식 프로세스중 임의의 자식이 종료되어 반환되면 wait이 끝나게 된다. 이러한 문제를 해결하기 위해 waitpid(pid)를 주어 해당 명령의 fork시 생성된 자식프로세스가 반환되어야 waitpid가 반환되도록 한다.

4.Psuedo-code

main(){

while(1){

output "myshell>";

input cmdline;

numtokens = makelist(cmdline, " \t", cmdvector, MAX\_CMD\_ARG);

if(strcmp(cmdvector[0], "exit") == 0)

exit;

else if(strcmp(cmdvector[0], "cd") == 0)

{

cd\_exe(numtokens, cmdvector);

continue;

}

switch(pid = fork()){

case -1:

fatal("main()");

case 0:

exe\_cmd(numtokens);

default:

waitpid(pid, NULL, 0);

}

}

}

int cd\_exe(int cmd\_num, char \*\* cmdvec){

if(cmd\_num == 1)

chdir(getenv("Home"));

else if(cmd)num == 2)

{

if(chdir(cmdvect[1]))

print "No directory\n";

}

else

print "Error\n";

}

void exe\_cmd(int cnt)

{

if(check\_background(cnt))

{

switch(fork())

{

case -1:

fatal("Error calling fork()");

case 0:

null = open("/dev/null")

dup2(null, 1);

exec(cmdline);

default:

exit(1);

}

}

exec(cmdline);

}

int check\_bacground(int cnt)

{

for i=0, i< cnt, i++;

{

if(!strcmp(cmdvector[i], "&"))

{

cmdvector[i] = NULL;

return 1;

}

}

return 0;

}

}

쉘 구현 프로젝트 과제 #2

12171702 조대환

제출일 : 2021-11-07

테스트 사항1 Background & Foreground

텍스트이(가) 표시된 사진

자동 생성된 설명

좀비 프로세스가 생성되지 않습니다.



텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

SIGCHLD 신호를 catch하고 수행할 함수인 zombie\_controller를 sigaction 구조체의 sa\_handler에 등록해 줍니다. 해당 신호는 자식프로세스가 종료되었을 때 발생하며, 해당 작업을 통해 자식 프로세스가 종료될 경우 해당 프로세스를 zombie\_controller의 waitpid를 통해 프로세스를 처리해주어 좀비프로세스가 생기는 것을 방지해줍니다.

테스트 사항2 – 쉘 프롬포트 상에서의 신호 무시



쉘은 종료되지 않습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

shell에서는 해당 signal들을 모두 무시하여 프로세스가 종료되는 것을 방지했습니다.

테스트사항3 – Foreground 실행시 Interrupt

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

포어그라운드에서 실행시 무시처리했던 신호들을 다시 받아들이도록 설정하고 터미널 제어권을 포어그라운드에 넘겨줬습니다.

이후 포어그라운드 프로세스인 자식프로스세스는 ^C나 ^\같은 신호를 받아들이고 해당 포어그라운드 프로세스는 종료되도록 설정했습니다.

자식프로세스가 종료되면 부모인 shell이 tcsetpgrp를 통해 터미널 제어권을 회수합니다.

쉘 구현 프로젝트 과제 #3

12171702 조대환

제출일 : 2021-12-11

테스트 사항 1 : Redirection

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

명령어 실행하는 함수에서 redirection함수를 실행시킨다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Redirection 함수에서는 해당 명령어를 인자로 받아

<와 >에 따라 dup을 통해 입출력을 바꾸어준다.

테스트사항 2 : 리다이렉션과 파이프 테스트

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

파이프로 명령어들을 구분해주고 해당 명령어 실행마다 자식 프로세스를 생성하여 부모 프로세스와 파이프로 입출력을 연결하여 파이프 명령어를 실행하도록 구현했다.

테스트 사항 3 : 리다이렉션과 파이프 테스트 (백그라운드)

텍스트이(가) 표시된 사진

자동 생성된 설명

이전과제에서 수행한 백그라운드 실행도 정상적으로 잘 작동하는 것을 확인 했습니다.

파이프와 리다이렉션 모두 이론수업에서 배운대로 dup2와 pipe를 사용하여 해결했으며 해당 기능들 사용에는 별로 어려움을 못느꼇습니다. 그러나 명령어 해석관점에서 명령어 구조를 분해하는 과정을 고민하고 구현 하는 것에 있어 어려움을 느꼈습니다.