**System Programming**

**Assignment #3**

2020320054

컴퓨터학과 박민욱

1. **전체 구조 설명**

대부분의 메인 로직은 모두 lib.rs에 구현하였고, main.rs는 loop를 통해 이를 호출하는 역할만 수행한다. lib.rs의 모듈은 Interact, Subprocess 총 두 가지이며, Interact는 사용자에게 stdin 문자열 입력을 받아서 약간의 전처리를 거친 뒤, 이 입력을(벡터) Subprocess 모듈로 넘기는 모듈이고, Subprocess 모듈은 건네받은 입력 벡터를 기반으로 파이프라인 여부를 판단하여 child process를 형성하여 실질적인 명령을 실행토록 하는 구조이다.

1. **Interact 모듈**

해당 모듈에서는 크게 1가지의 public 함수, 4가지의 private 함수가 있다. 4가지의 private 함수들은 interact 함수를 지원하는 함수이다. is\_path() 함수는 주어진 문자열이 존재하는 경로인지를 boolean으로 반환하고, supported() 함수의 경우는 이 과제에서 구현한 프로그램은 완전하지 않을 수 있어서 리눅스의 대표적인 기본 명령어들인지 판단하기 위해 만들었다. preprocess\_vector()는 약간의 전처리를 수행한다. 기본적으로 interact함수에서는 입력받은 문자열을 공백으로 구분하여 argument를 처리하지만, 만약 echo “hello world”의 경우 이를 공백으로 나누면 “hello, 와 world”가 되어 “로 감싸진 문자열이라고 생각되는 것들은 하나의 argument로 묶도록 하였다. Terminating()의 경우는, exit가 입력받았을 때, 현재 프로그램, 즉, oh my shell을 종료하는 함수이다.  
 그리고 마지막으로, interact()는 상기 설명한 함수들을 사용하여 실질적으로 입력을 받고, 전처리를 한 후에 입력 받은 문자열을 벡터로 파싱하여 Subprocess 모듈의 subprocessing 모듈을 호출한다.

정리하자면, 유효한 명령어를 subprocess 모듈로 넘겨주는 역할을 한다.

1. **Subprocess 모듈**

Subprocess 모듈에는 interact 모듈에도 포함되어 있는 함수들이 있어, 이것들은 제외하고 설명한다. 기본적인 아이디어는 다음과 같다.   
subprocessing() 함수는 해당 모듈의 진입점으로, Interact::interact() 함수가 호출하는 함수이며, 이 파일의 핵심 로직을 담당하는 함수이다. 가장 먼저, 입력 받은 명령어 벡터를 다시 한 번, 파이프라인 명령어를 기준으로 다시 commands 벡터에 나눠 넣는다. 파이프라인을 기준으로 각기 다른 child 프로세스가 해당 명령어들을 수행하고, 결과를 파이프라이닝 하도록 하기 때문에, 각 child가 사용할 명령어들로 세분화하여 나누는 것이다.

명령어가 1개라면, 그냥 바로 fork/exec을 통해 명령어를 실행한다. 다만, child의 명령어 부분에 handle\_redirection() 이라는 부분이 있는데, 이는 명령어에 redirection >, <가 포함 되었을 경우도 컨트롤 할 수 있게끔 명령을 수행한다. handle\_redirection()은 리다이렉션 기호가 있는 경우, 없는 경우로 나누고 있는 경우는 dup2를 활용하여 표준 출력이 아닌 target file로 출력 되도록 file descriptor를 덮어씌운다. 그리고 이 상태에서 exec를 통해 명령어를 넘겨서 실질적인 실행이 가능하도록 한다. 그리고 exec\_command()는 command를 C 스타일의 문자열로 바꾼 후, exec를 실행하는 간단한 함수이다.

주석 //Leveraging named pipe 밑의 줄은 주석의 설명처럼 pipe를 설명하는 부분이다. 명령어 개수를 나타내는 subprocess\_trial이 N개라고 할 때, 총 N-1개의 자식 프로세스가 사용되므로 N-1개의 named pipe를 생성하여 fifo\_paths에 저장한다. 이때, temp\_dir를 통해서 임시 디렉토리를 얻고, ohmyshell\_fifo\_pid\_i 형태의 파일명을 생성한다. 이후에 파일 경로를 통해 이 파이프를 삭제할 때 사용한다. mkfifo 시스템 콜을 사용해서 fifo\_cstr 경로에 해당하는 named pipe를 권한 설정(읽기, 쓰기를 소유자에게만 부여)하고, 실패시 이를 모두 지운다. 각 파이프는 고유한 파일 이름을 가진다.

그 다음 N개의 명령어 하나씩에 대응되는 자식 프로세스를 fork한다. 그리고 이 때 각 child는 이전 단계에서 생성한 FIFO(mkfifo 이용)를 통해 자신의 input, output을 다른 프로세스와 연결한다. 첫 번째 명령어 프로세스는 파이프라인의 첫부분이므로, 출력만 설정하고, 마지막 명령어 프로세스는 파이프라인의 마지막 부분이므로, 입력만 설정한다. 그리고 나서, 앞서 설명한 handle redirection을 수행하면, 각 프로세스는 리다이렉션이 포함되거나 포함되지 않은 명령어들을 올바르게 처리하고, 파이프라이닝이 진행된다.

그리고 이후의 줄은 pid를 출력하고, 설정했던 파이프를 제거하는 것이다.

몇가지 명령어를 실행했을 때, 다음과 같은 결과를 볼 수 있다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 블랙이(가) 표시된 사진

자동 생성된 설명