**운영체제**

**텀 프로젝트 Report**

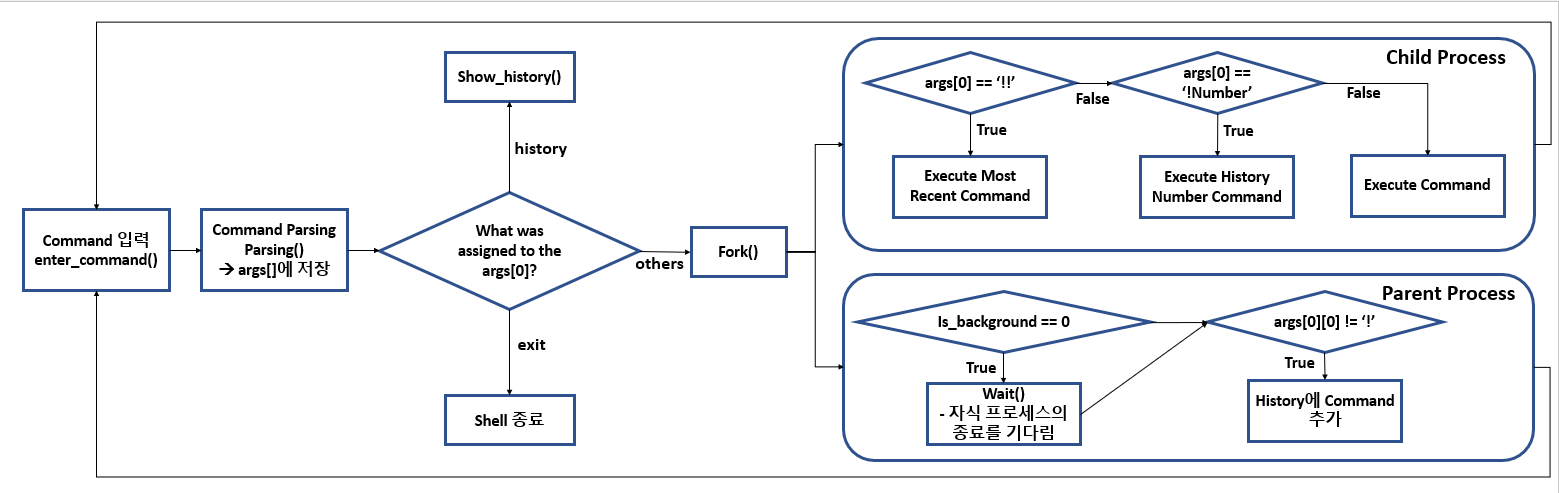
**Mini Shell구현**

**학과 : 컴퓨터공학과**

**학번 : 2017112068**

**이름 : 김관중**

**A. Flow Chart**



**B. 주요 코드 설명**

**1. main()**

- main 함수는 다음과 같은 실행 흐름을 반복하며, shell을 실행하게 된다

- user의 command 입력 🡪 command parsing 🡪 execute command

* **Command 입력 & Parsing**

텍스트이(가) 표시된 사진

자동 생성된 설명

- enter\_command()를 호출하여 input에 command를 저장하고, command의 길이를 return받는다

- 이후, parsing()을 호출하여, 입력받은 command를 parsing하여 args에 나눠서 저장한다

- 이때, is\_background의 주소를 파라미터로 전송하여, 만약 input에 &가 포함되어 있다면, is\_background를 1로 할당하여, 추후에 command의 background 실행을 돕는다

* **Execute Command**

- Command의 실행은 exit, history, !!, !number, others로 구분한다

**(1) history**

텍스트이(가) 표시된 사진

자동 생성된 설명

- exit의 경우, shell을 바로 종료하며, history의 경우, show\_history()를 통해, 지금까지의 command 실행 list를 출력한다

**(2) !! command**

- exit, history를 제외한 나머지 command들은 fork()를 통해 자식 process를 생성하고, execvp()를 통해, command를 실행시키는 공통된 과정을 포함한다

- 다만, !!의 경우, history에 기록된 가장 최근의 command를 실행해야 하기 때문에, history에 저장된 command를 가져오는 과정이 필요하다

텍스트, 스크린샷, 실내, 화면이(가) 표시된 사진

자동 생성된 설명

- history\_cnt가 존재하는지 예외처리를 진행한 후에, strcpy를 통해, 가장 최근의 history를 new에 저장하고 새롭게 parsing을 진행한다

- 이렇게, input을 새롭게 담고 parsing을 다시 진행하는 이유는 history에 parsing한 결과가 아닌 입력받은 input 전체를 기록했기 때문이다

- 최종적으로, parsing 결과를 execvp를 통해 실행시키며 마무리된다

**(2) !number command**

텍스트, 스크린샷, 화면, 닫기이(가) 표시된 사진

자동 생성된 설명

- !number command는 number에 기록된 숫자를 참조하여, 해당 숫자와 맞는 history number의 command를 실행시키게 된다

- 우선, 입력받은 number를 파악해서 cnt에 저장하게 된다

- 다만, history\_cnt의 범위를 벗어나는 숫자를 입력받으면, 예외처리를 통해 error를 산출하게 된다

- 이후, cnt에 맞는 history command를 new에 저장하고 parsing한 command를 실행하게 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명**(3) 나머지 모든 command**

- 나머지 command들은 parsing 결과인 args를 execvp로 바로 실행시키게 된다

**(4) Parent Process**

- 자식 프로세스에서 command를 실행하는 동안, Parent는 다음과 같은 작업을 수행한다

텍스트이(가) 표시된 사진

자동 생성된 설명

- 우선, Command의 &유무에 따라, is\_background 변수에 0이 저장되었다면, &가 없다는 소리이기에, 자식 프로세스가 종료되는 것을 wait를 통해 기다린다

- 만약, 1이 저장되었다면, 자식 프로세스의 종료를 기다리지 않게 된다

- 이후, history command에 input을 저장하게 되는데, !!, !number는 따로 history에 기록하지 않는 것을 원칙으로 하였다

**2. enter\_command()**

텍스트, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명- User로부터 commnad를 받아 input에 저장하는 함수이다

- 이때, read의 반환 값인 input의 길이를 따로 저장하여, 반환하게 된다

- 해당 길이 정보는 추후, parsing과정에서 이용된다

텍스트, 모니터, 화면, 실내이(가) 표시된 사진

자동 생성된 설명**3. parsing()**

- 저장된 input을 parsing하여 args[]에 저장하는 함수이다

- 알고리즘은 다음과 같다

(1) input의 첫 글자부터 반복문으로 순회

(2) 공백, tab, 띄어쓰기가 아니라면, token에 해당 글자를 이어붙인다

(3) 만약, &가 존재하면, is\_background를 1로 갱신

(3) 공백, tab, 띄어쓰기를 만나면, token에 null character를 추가하고 args에 해당 token을 저장한다

**4. show\_history()**

텍스트이(가) 표시된 사진

자동 생성된 설명- history에 담긴 command list를 출력해주는 함수이다

- 기본적으로 최대 10개의 command를 산출하게 된다

- 하지만, history에 command가 10개를 넘어가게 될 경우, 추가로 command를 보겠냐는 물음을 출력하게 된다

- 만약, Y를 답하면, 추가로 Command를 출력하게 된다

- N을 답하면, Command 출력을 멈추게 된다

**C. 실행 결과 분석**

**1. 일반 Command 실행**

텍스트이(가) 표시된 사진

자동 생성된 설명

- ls, ls -l, cat, Who를 순차적으로 실행시킨 결과, 정상적으로 실행된 것이 확인 가능하다

- 이를 history로 확인한 결과는 다음과 같다



**2. !! Command 실행**

- 새롭게 minishell을 실행하고, 바로 history를 실행하면 cmd list가 없기 때문에 error를 산출한다

텍스트이(가) 표시된 사진

자동 생성된 설명

- 이후, who, ls, !! cmd를 순차적으로 입력합니다

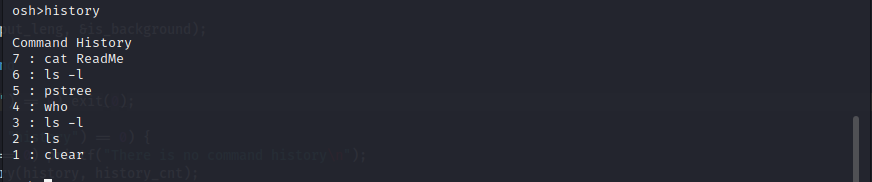
텍스트이(가) 표시된 사진

자동 생성된 설명

- !!의 결과로, ls가 정상적으로 실행된 것이 확인 가능하다

**3. !Number Command 실행**

- !Number를 실행하기 위해, 여러 개의 Command를 실행하고 history로 확인합니다



- !4를 입력하여, who가 정상적으로 실행되는지 확입합니다



- who가 정상적으로 실행되는 것을 확인할 수 있습니다

- 예외처리를 확인하기 위해, 이번에는 범위를 벗어나는 !8을 입력합니다



- 정상적으로, 예외처리가 일어난 것이 확인 가능합니다

**4. &을 통한, background 실행 확인**

- background에서 program이 동작하면서, shell이 사용가능한지 확인하기 위해서, wireshark라는 프로그램을 background에서 동작시켜봤습니다

텍스트이(가) 표시된 사진

자동 생성된 설명

- 실행된 wireshark를 유지한 채로, who command를 입력합니다

텍스트이(가) 표시된 사진

자동 생성된 설명

- 그 결과, wireshark를 유지하면서도 who가 실행된 것이 확인 가능합니다

**5. History 기능 확인**

- 10개 이상의 command를 입력하고, history를 실행시킵니다

텍스트이(가) 표시된 사진

자동 생성된 설명

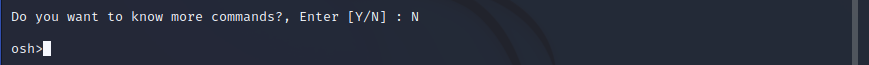
- 위와 같이, 추가적인 Command를 출력할 것인지, User에게 물어봅니다

- 이때, Y를 입력하면, 다음과 같이 추가적인 Command를 출력해줍니다

텍스트, 장치, 게이지, 조종판이(가) 표시된 사진

자동 생성된 설명

- 반면, N을 입력하면, 다음과 같이, history를 종료하게 됩니다



- 예외처리로, 만약 Y도 N도 아닌 다른 문자를 입력하면, 다음과 같이, Y 또는 N을 입력할 때 까지 반복적으로 input을 다시 요구하게 됩니다

