**System Programming Project 4**

담당 교수 : 김영재 교수님

이름 : 주어진사랑

학번 : 20191654

1. **개발 목표**

해당 프로젝트에서 fork를 이용하여 쉘의 기본 명령어들을 수행하고 piping을 이용한 명령어를 수행하는 기능을 구현한다.

Phase 1의 경우, 입력이 들어오면 builtin 명령어인지 확인하고 맞을 경우 해당하는 기능을 수행한다. 아닌 경우 fork를 이용하여 cd, ls, mkdir, rmdir, touch, cat, echo, exit을 포함한 linux 쉘의 주요 명령어들의 기능을 수행하도록 한다. echo의 경우 `를 사용할 경우 경로를 출력한다. 에러가 발생할 경우 해당 에러를 출력한다.

Phase 2의 경우, 입력이 들어오면 | (pipe)에 따라 명령어를 나누어 fork 및 pipe를 실행한다. 마찬가지로 builtin 명령어인지 확인 후 아닌 경우 fork한다. pipe와 임의의 배열, dup2 함수를 사용하여 fork된 명령어의 입력과 출력을 지정한다. 이를 통해 순차적으로 파이프를 통해 이전의 출력이 다음의 입력이 되도록 하고, 이를 전부 수행한 결과를 출력한다. 에러가 발생할 경우 해당 에러를 출력한다.

1. **개발 범위 및 내용**
   1. **개발 범위**
2. Phase 1

Phase 1을 구현하였을 때, “CSE4100-SP-P4>”가 계속 출력되어 입력을 받고, 그에 맞는 기능을 수행 및 출력하고 반복한다. 입력 받은 문자열을 단어 별로 나눈 뒤, fork를 통해 child process에서 커맨드를 수행하고 그 결과를 parent process로 돌려준다. cd, ls, mkdir, rmdir, touch, cat, echo, exit을 포함한 linux 쉘의 주요 명령어들의 기능을 수행한다.

1. Phase 2

Phase 2를 구현하였을 때, 마찬가지로 “CSE4100-SP-P4>”가 계속 출력되어 입력을 받고, 그에 맞는 기능을 수행 및 출력하고 반복한다. 이 때 파이프 ‘|’를 사용할 경우, 두 개 이상의 입력 커맨드로 한 개의 결과를 갖는다. 이전 커맨드의 결과인 출력 값을 다음 커맨드가 입력으로 받는다. 이와 같이 모든 기능을 순차적으로 동작하고 그 결과를 수행 및 출력한다.

1. Phase 3
   1. **개발 내용**

* **Phase1 (fork & signal)**
  + fork를 통해서 child process를 생성하는 부분에 대해서 설명

먼저 들어온 command가 builtin command인지 그 여부를 확인하고, 맞을 경우 해당하는 기능을 수행하고 그렇지 않은 경우 fork를 통해 기능을 수행하였다. fork의 결과가 0보다 작을 경우 에러 메시지를 출력하고, 0일 경우 child process로 분류하여 command 기능을 수행하고, 0보다 클 경우 parent process로 분류하여 waitpid 함수를 통해 child process의 종료를 기다린다.

* **Phase2 (pipelining)**
  + Pipeline( ‘|’ )을 구현한 부분에 대해서 간략히 설명 (design & implementation)

pipeline 커맨드가 들어올 경우, fork를 통해 child process와 parent process를 구분한 뒤 pipe를 통해 입력과 출력을 임의로 지정한다. parent process의 경우 파이프의 출력은 닫고 그 입력을 임의의 배열에 저장해준다. 이는 다음 pipeline 커맨드의 child process로 그 값을 넘기기 위함이다. child process의 경우 파이프의 입력은 닫고 출력을 파이프에 쓴다. 이전 parent process에서 임의의 배열에 저장해준 값을 입력으로 이어받아, 이전 pipeline 커맨드의 출력을 입력으로 지정한다.

* + Pipeline 개수에 따라 어떻게 handling했는지에 대한 설명

while문을 통해 커맨드를 임의의 문자열에 저장한다. 커맨드 뒤에 파이프가 있을 경우 원래와 같이 pipe 및 fork를 진행하고 반복한다. 반대로 파이프가 없을 경우, pipe 및 fork를 진행하고 마지막 커맨드로 판단해 while문을 break한다. 이 때 마지막 커맨드일 경우 pipe를 통해 다음으로 출력을 넘기지 않고 원래대로 STDOUT\_FILENO로 출력을 넘긴다.

* **Phase3 (background process)**
  + Background (’&’) process를 구현한 부분에 대해서 간략히 설명
  + connection을 종료할 때 parent process에게 signal을 보내는 signal handling하는 방법 & flow
  1. **개발 방법**
* **Phase 1**

builtin 커맨드인지 판단하는 함수를 통해 cd, quit, exit, &의 명령어를 수행한다. 기존에 배포된 unix\_error, app\_error, Fgets 함수를 사용하고, 큰 따옴표(“)와 작은 따옴표(‘)를 문자열에서 삭제하는 Remove 함수를 추가하여 사용한다.

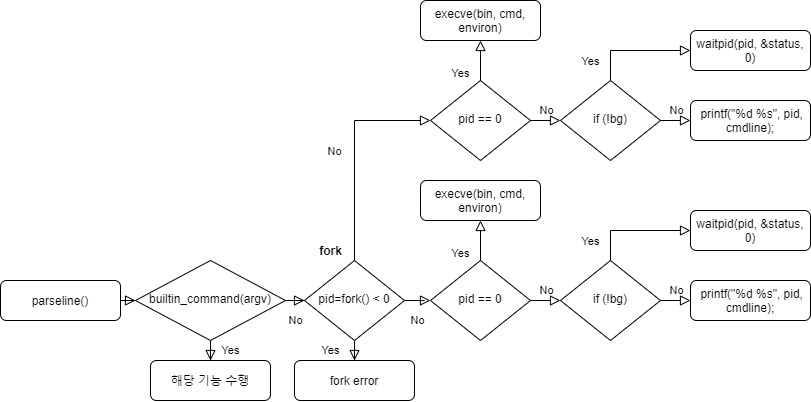
eval 함수에서는 parseline 함수를 통해 입력을 단어 단위로 나누어 argv[]에 저장한다. fork를 통해 child process를 생성하고 각각 child process와 parent process에 맞는 기능을 수행한다. child는 입력 커맨드를 수행하고 parent는 child의 종료를 기다리는 wait 기능을 수행한다. child process에서 입력 커맨드를 알맞은 형태로 변환하여 execve 함수를 호출하기 위해 임의의 문자열을 선언하여 형태를 맞추어 준다.

* **Phase 2**

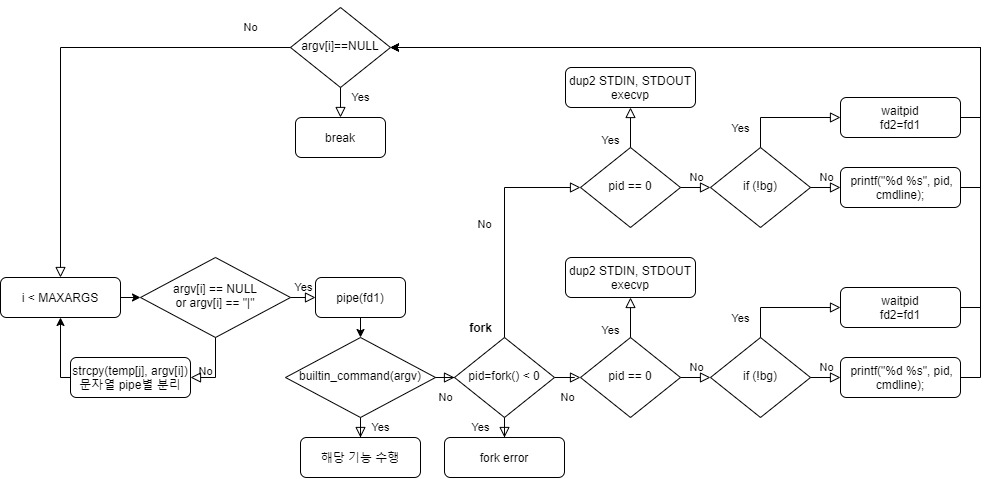
Phase 1과 마찬가지로 쉘의 주요 명령어들을 수행한다. 파이프라인 명령어를 처리하기 위해 evalpipe 함수를 추가하여 사용한다.

evalpipe 함수에서는 마찬가지로 pareline 함수를 통해 입력을 argv[]에 저장한다. 이후 pipe(|)에 따라 커맨드 단위로 나누어 temp 문자열에 저장하고 while문을 빠져나온다. pipe를 생성하고 fork를 통해 child process를 생성한다. 각각 child process와 parent process에 맞는 기능을 수행한다. 이 때 pipe와 임의의 배열을 통해 입력과 출력을 지정해주어 이전 파이프의 출력이 다음 파이프의 입력이 되도록 구현한다. 임의의 배열에 이전 결과를 연결하여 다음 fork에서 dup2 함수를 통해 그 값을 pipe()를 통해 생성한 입력 파이프에 쓰도록 한다. child process에서 입력 커맨드를 알맞은 형태로 변환하여 execvp 함수를 호출하기 위해 임의의 문자열을 선언하여 형태를 맞추어 준다.

1. **구현 결과**
   1. **Flow Chart**
2. **Phase 1 (fork)**



1. **Phase 2 (pipeline)**

****

1. **Phase 3 (background)**