**System Programming Project 1**

담당 교수 : 김영재

이름 : 김도현

학번 : 20181600

1. **개발 목표**

* 해당 프로젝트에서는 Linux의 쉘 기능 일부를 이용하는 프로그램을 만든다.
* /bin/ 디렉토리 내부에 내장된 명령어와 그 옵션을 실행할 수 있으며
* 리다이렉션과 Pipe를 통한 프로세스간 통신을 구현하고
* 프로세스에 signal이 전달될 때 handler를 통해 이를 통제할 수 있다.

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

명령어를 입력받고, 해당 명령어에 해당하는 프로세스를 fork()하여 실행하게 한다. /bin/ 디렉토리에 내장되어 있지 않은 명령어들에 대해서는 직접 구현한다.

1. Phase 2

리다이렉션, 파이프를 이용하여 프로세스간 통신을 구현한다. 이 때 파이프의 개수는 제한이 없으며 한 프로세스에서 다른 프로세스로 출력값을 리다이렉션하는 기능을 구현한다.

1. Phase 3

Background 프로세스를 실행하였을 때 이를 jobs 명령어를 통해 확인하고, kill 명령어를 통해 이를 강제 종료시키거나 fg 명령어를 통해 foreground와의 전환을 할 수 있도록 한다.

또한 Signal Handler를 만들어 Ctrl+C, Ctrl+Z 등의 입력에 대해 적절히 대처할 수 있또록 한다.

* 1. **개발 내용**
* **Phase1 (fork & signal)**
  + 명령어를 입력 받아 이를 parsing한 후, parsing한 명령어가 올바른 명령어라면 해당 프로세스를 fork()한다. Fork()된 child process에서는 해당 명령어를 처리한 후 프로세스를 종료한다.
  + Fork()를 수행한 parent process는 child process의 signal을 waitpid() 함수를 통해 wait하여 종료를 확인한다.
* **Phase2 (pipelining)**
  + ‘|’ 기호가 나오는 부분은 parseline() 내부에서 처리한다. 받은 명령어 전체를 검색해서 ‘|’ 기호가 나오는 부분을 idx[] 배열에 위치로 저장하고, 해당 argument를 NULL 문자로 치환한다. 치환된 NULL 문자를 전후하여 pipe에 출력할 프로세스와 pipe로부터 입력을 받을 프로세스를 구분한다.
  + Pipeline이 2개 이상일 경우 iteration으로는 file descriptor 등의 변수가 계속 필요해질 것으로 판단되어 pipe를 통해 전달할 프로세스의 출력 file descriptor를 다음 프로세스의 입력 file descriptor로 재귀적으로 리다이렉션하도록 구현하였다.
* **Phase3 (background process)**
  + Parseline() 함수에서 명령어 라인의 마지막 문자가 ‘&’라면 background 프로세스로 판단하여 이를 eval() 함수에 인자로 넘겨준다. Background 프로세스로 판단되면 해당 프로세스를 jobs 배열에 넘겨주어 그와 별개의 Foreground 프로세스를 또 실행할 수 있게 해준다.
  1. **개발 방법**

Phase1 구현을 위해 크게 함수를 수정해야 하는 부분은 거의 없었으며, /bin/ 파일 내의 명령어들에 접근할 수 있는 함수를 사용하고 해당 디렉토리 내에 없는 명령어는 직접 구현하였다.

Phase2 구현을 위해 parseline() 함수의 인자에 pipe의 위치를 저장하는 idx 배열을 추가하였고, pipe가 하나 이상이라면 eval() 함수와 유사하지만 재귀적으로 동작하는 pipe\_command() 함수에서 명령어를 수행한다. 이 때 재귀적으로 전달하는 인자는 각각 명령어, 재귀 횟수(파이프의 개수), 파일 디스크립터이다.

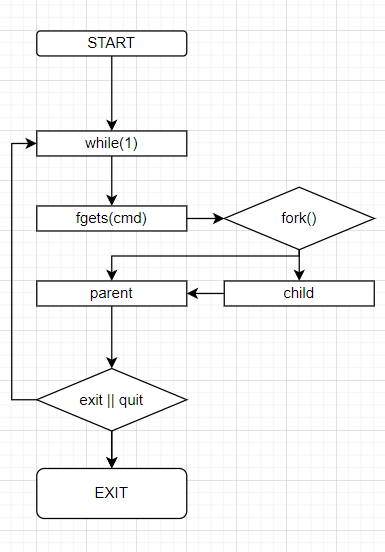
Phase3 구현을 위해서 job 구조체를 만들었다. 구조체는 프로세스 ID, job id, 명령어, 그리고 상태를 enum으로 정의한 네 가지 변수들로 구성되어 있으며 이 구조체로 이루어진 배열 jobs를 전역변수로 선언하여 모든 함수에서 접근할 수 있도록 하였다.

추가적으로 Builtin\_command() 함수에서 jobs, fg, bg, 등의 사용자 정의 명령어를 정의하였다.

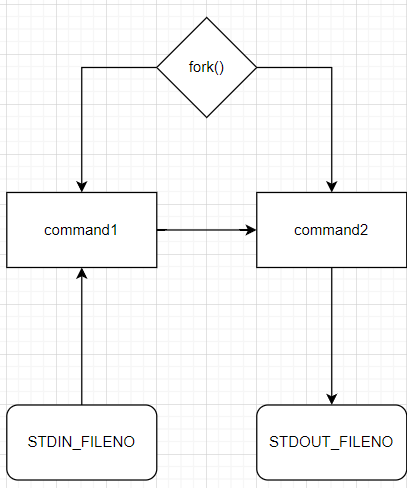
또한 Ctrl+C를 제어하기 위한 Signal Handler와 프로세스를 정지하기 위한 Ctrl+Z Signal Handler를 만들었다.

다만 Ctrl+Z의 경우 이 프로그램을 탈출하고 좀비 프로세스를 생성하기도 하여 완벽하게 구현했다고 하기는 어렵다.

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

****

1. **Phase 2 (pipeline)**

****

1. **Phase 3 (background)**

