**System Programming Project 2**

담당 교수 : 김영재

이름 : 김효림

학번 : 20221549

1. **개발 목표**

Fork를 이용하여 parents process와 child process의 concurrent한 system을 이용하여 unix shell을 구현한다. 프로세스 생성, 파이프를 통해 명령어 입력, background와 foreground 사이의 process와 시그널 처리를 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Phase 1 : 리눅스 쉘에서 동작하는 command를 foreground에서 수행한다.
2. Phase 2 : 파이프를 포함하여 여러 개의 command를 foreground에서 수행한다.
3. Phase 3 : background process를 구현하고 signal handler를 구현하여 처리한다.
   1. **개발 내용**

* **아래 항목의 내용만 서술**
* **(기타 내용은 서술하지 않아도 됨. 코드 복사 붙여 넣기 금지)**
* **Phase1 (fork & signal)**
  + fork를 통해서 child process를 생성하는 부분에 대해서 설명

Fork() 명령어를 통해 child process를 생성하고, pid에 Fork()의 return 값을 저장한다.

* + connection을 종료할 때 parent process에게 signal을 보내는 signal handling하는 방법 & flow

waitpid 함수를 사용해서 child process가 끝나기를 기다린 후 reaping하여 zombie process를 없앤다.

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

Command line에 존재하는 |를 감지하여, 파이프 수 + 1 만큼 command의 개수를 count해 준다. 그 후 run\_pipeline 함수에서 child process를 생성하고, dup2 함수를 사용하여 parent – child 간 입출력이 가능하도록 설정한다.

* + Pipeline 개수에 따라 어떻게 handling했는지에 대한 설명

Command의 개수는 pipeline의 개수 + 1이기 때문에, 해당 크기만큼의 index를 가지는 cmd 배열을 재설정하여 명령에 따라 command를 재정의했다. 파싱 이후에는 파일 디스크립터에 따라 읽기 쓰기를 진행한다. fd[i][0] 에서는 i 번째 명령에서 읽기를 fd[i][1]에서는 i 번째 명령에서 쓰기를 진행한다.

* **Phase3 (background process)**
  + Background (’&’) process를 구현한 부분에 대해서 간략히 설명

명령어 끝에 ‘&’ 기호를 사용하여 background process를 구현한다. Passing 함수에서 ‘&’이 감지되면, bg로 들어가게 된다. 부모 프로세스는 reaping하기 위해 child process를 wait하지 않고, bg와 독립적으로 동작하게 되며 eixt을 통해 종료될 때 SIGCHLD를 통해 자식 프로세스에도 signal을 보낸다. Builtin\_command 함수에 bg, fg, kill, jobs 명령어가 들어왔을 때 동작하는 내용을 추가하여 구현하였다.

* 1. **개발 방법**
* **B.의 개발 내용을 구현하기 위해 어느 소스코드에 어떤 요소를 추가 또는 수정할 것인지 설명. (함수, 구조체 등의 구현이나 수정을 서술)**

Pipe line command를 처리하기 위해 void type의 run\_pipeline(char\*\* argv) 함수를 구현하였다. 전역변수로 설정된 pipe\_line의 개수를 이용하여 command의 수를 계산하여, 그에 따라 command line을 cmd로 재설정하여 각 명령에 인덱스를 부여하여 처리할 수 있도록 한다. Command 수 만큼 for 문을 반복하여 fork를 생성하고, dup2를 이용해 읽기와 쓰기를 변경한다. fd[i][0]은 i번째 명령에서 읽기, fd[i][1]은 i번째 명령에서 쓰기에 해당한다.

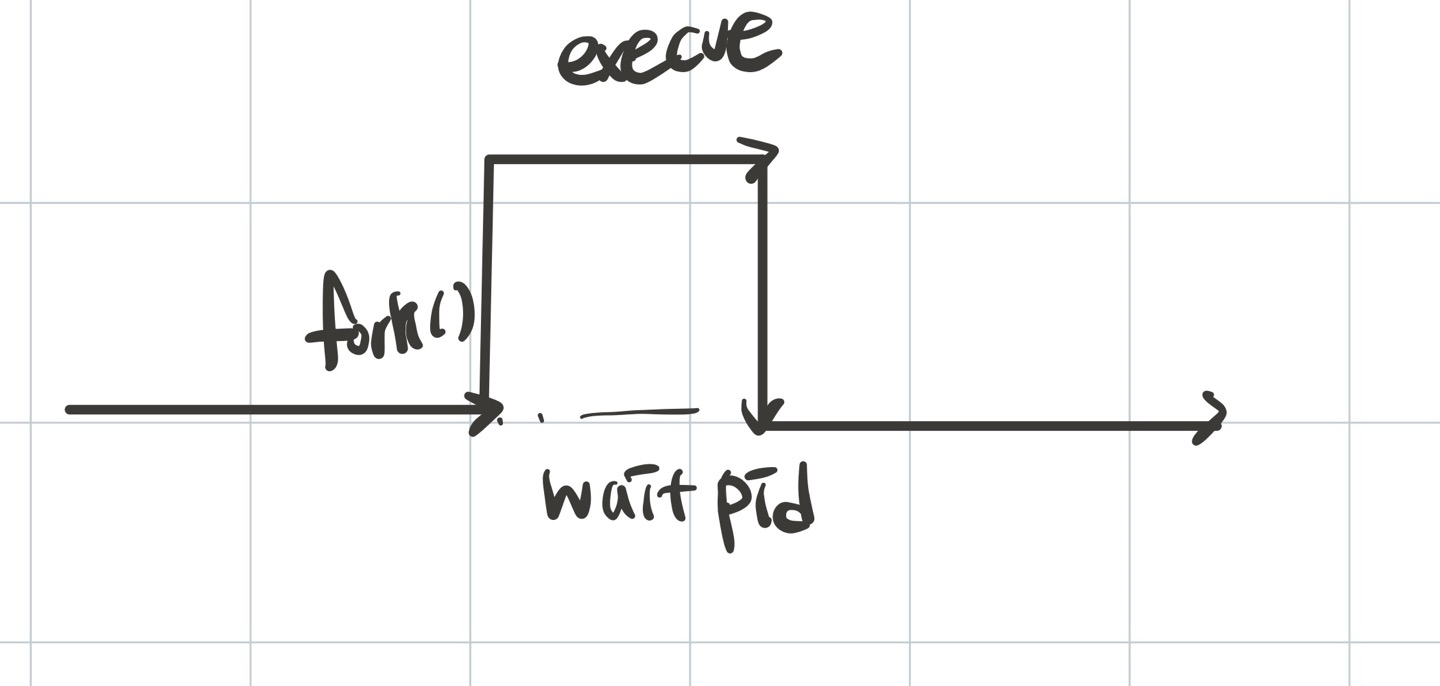
Background process를 관리하기 위해서는 bg가 1인 경우 부모가 기다리지 않고 프로세스를 실행하도록 한다.(waitpid 생략) jobs 명령어를 수행하기 위해 jobs를 구조체로 만들어 PID, status, cmd를 저장할 수 있도록 만들었다. Status는 running, suspend, stop 세 가지 상태를 담게 된다. 프로그램이 실행되며 addjob 함수를 통해 추가를, delete\_job 함수를 통해 작업 목록을 관리할 수 있다.

Background process와 foreground process 사이에 발생하는 signal은, Ctrl-Z가 눌린 경우, Ctrl-C가 눌린 경우, 마지막으로 child process가 종료되어 SIGCHLD를 받은 세 가지 경우에 대하여 핸들러를 작성하였다.

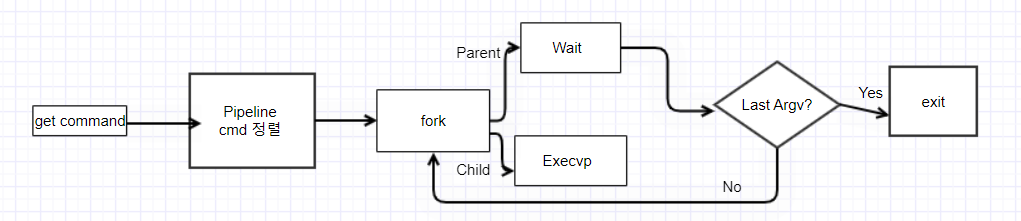
1. **구현 결과**
   1. **Flow Chart**

* **2.B.개발 내용에 대한 Flow Chart를 작성.**
* **(각각의 방법들에서 추가된 내용(fork, pipeline, background)만 특성이 잘 드러나게 그리면 됨.)**

1. **Phase 1 (fork)**



1. **Phase 2 (pipeline)**

****

1. **Phase 3 (background)**

**도표, 스케치, 기술 도면, 평면도이(가) 표시된 사진

자동 생성된 설명**