**System Programming Project 4**

담당 교수 : 김영재 교수님

이름 : 김소흥

학번 : 20171618

1. **개발 목표**

* **해당 프로젝트에서 구현할 내용을 간략히 서술.**
* **(MyShell을 만드는 전체적인 개요에 대해서 작성하면 됨.)**

linux 서버 상에서 동작하는 간단한 shell을 직접 만들어 사용자의 명령어 입력에 따라 알맞은 내용을 수행하고 결과를 출력하도록 한다. 단계별로 프로젝트를 진행하며 이전 단계의 내용을 포함해 점점 더 복잡한 명령어 기능을 수행할 수 있게끔 한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Phase 1

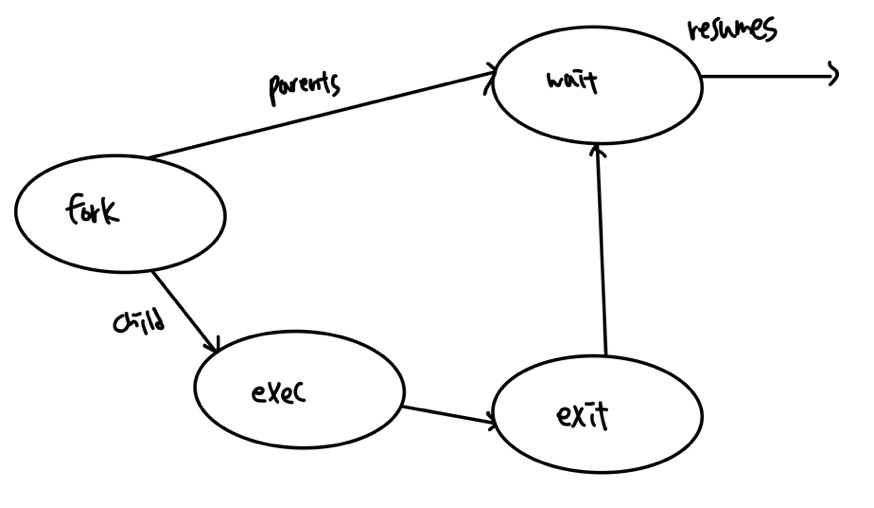
사용자의 입력에 따라 파이프가 포함되지 않은 간단한 단일 명령어를 수행할 수 있다.

1. Phase 2

여러 개의 파이프 (“|”)가 포함된 좀 더 복잡한 명령어를 수행할 수 있다.

1. Phase 3
   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
    - fork를 하게 되면 생성된 child process는 parents process의 모든 code를 똑같이 복제받는다. 이때 두 번의 pid를 return 하는데 parents process 일 경우 해당 pid를, child process일 경우 0을 return 한다. 따라서 fork를 통해 process를 생성한 후 pid를 확인하여 parents process가 아닌 child process에서 execve를 통해 입력받은 명령어를 실행한다.
  + connection을 종료할 때 parent process에게 signal을 보내는 signal handling하는 방법 & flow
    - 명령어는 모두 child process에서 실행하므로, child process가 종료될 때 까지 parents process는 기다려야 한다. child process는 execve() 후에 exit을 통해 종료된다. 이를 waitpid() 함수를 통해 기다리고 있는 parent process가 감지하여 zombie process의 생성을 막는다.



* **Phase2 (pipelining)**
  + Pipeline( ‘|’ )을 구현한 부분에 대해서 간략히 설명 (design & implementation)
    - 명령어 줄에서 pipeline(“|”)이 하나라도 감지되면 예외 처리를 해 pipeline 함수를 통해 단일 명령어와는 다르게 처리하도록 했다. pipe 함수를 통해 pipe를 생성하고 두 번의 fork를 통해 생성된 read 할 process와 write 할 process를 구분해 연결시켜 준다. write process의 output이 read process에 반영된다.
  + Pipeline 개수에 따라 어떻게 handling했는지에 대한 설명
    - 단일 파이프와 다중 파이프를 따로 구분짓지 않고 하나의 모듈로 구현하였다. 이때 다중 파이프를 고려하여 recursion을 사용했다.
* **Phase3 (background process)**
  + Background (’&’) process를 구현한 부분에 대해서 간략히 설명
  1. **개발 방법**
* **B.의 개발 내용을 구현하기 위해 어느 소스코드에 어떤 요소를 추가 또는 수정할 것인지 설명. (함수, 구조체 등의 구현이나 수정을 서술)**

phase 1

void exec\_cd() : fork를 사용하지 않는 cd 명령어를 구현하기 위해 추가한 함수이다.

void eval() : cd 명령어를 따로 처리하기 위한 코드를 추가 했다. fork와 execve, waitpid를 사용하기 위한 코드를 추가했다. fork를 통해 read할 process, write할 process를 생성하고 piperec를 호출한다.

phase 2

void fatal() : 에러 발생한 상황을 control 하기 위해 추가한 함수이다.

void pipeline( ) : 파이프라인 명령어를 처리하기 위해 추가한 함수이다. fork를 통해 read할 process, write할 process를 생성하고 piperec를 호출한다.

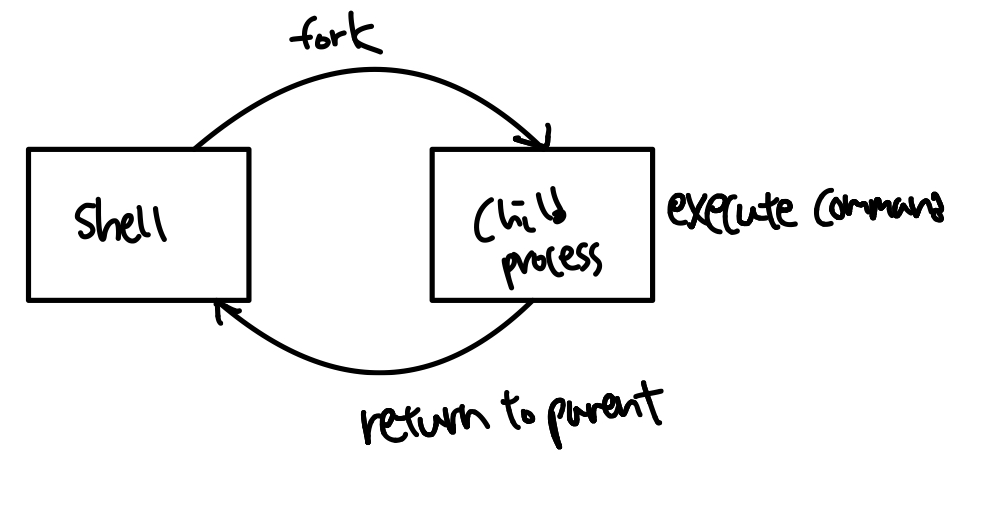
void piperec() : 실질적으로 recursion을 통해 다중 pipeline 명령어를 수행한다. recursion 횟수는 pipeline의 수에 따라 결정된다. 첫번쨰 pipeline을 기준으로 앞의 명령어를 우선 수행 후 recursion을 통해 뒷부분 명령어에 대한 piperec을 다시 호출한다.

void eval() : pipeline 수가 한 개 이상일 경우, parsing한 argv[]의 결과를 활용하여 “|”을 기준으로 command 배열을 처리하고 pipeline()을 호출하는 코드를 추가하였다.

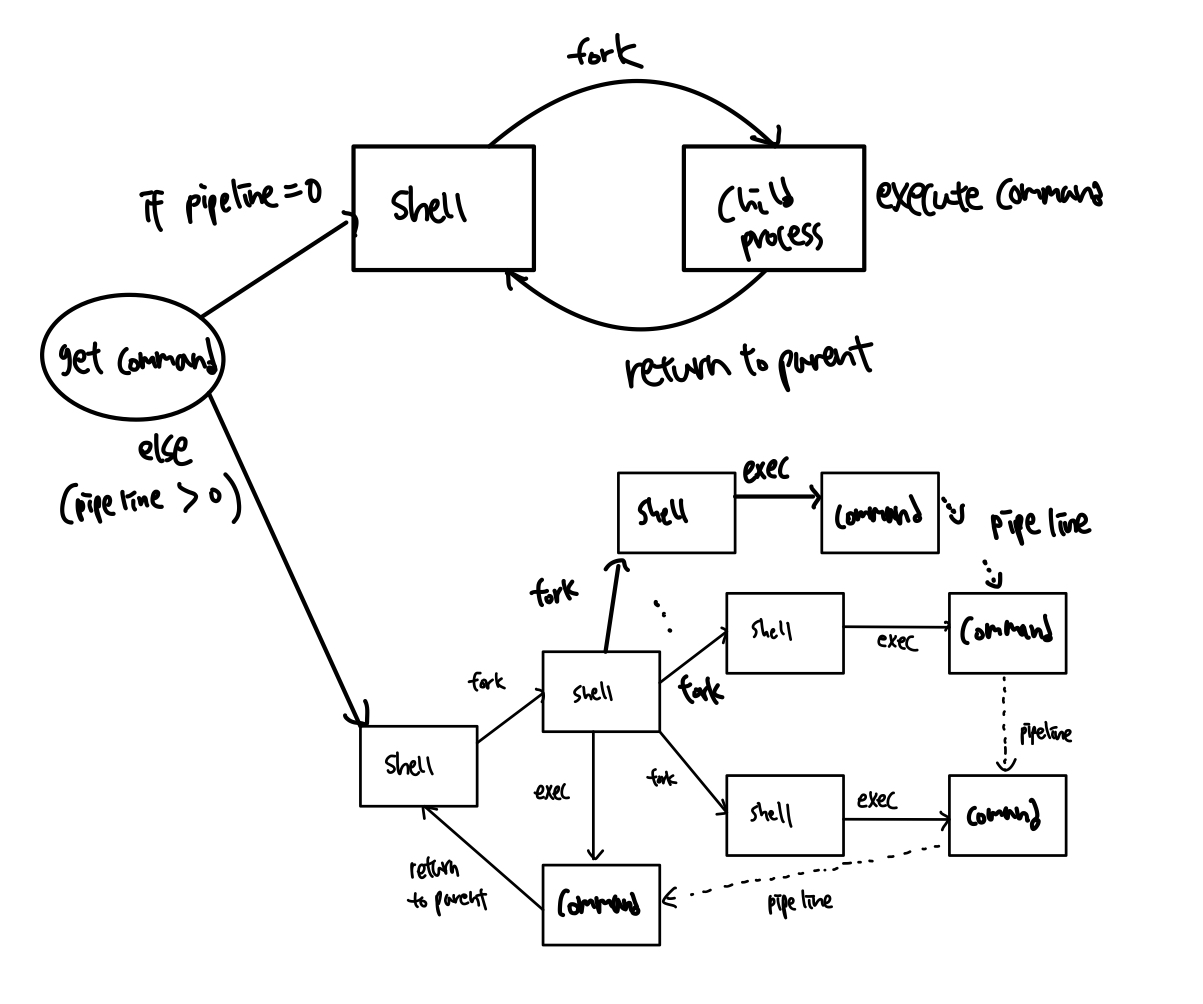
1. **구현 결과**
   1. **Flow Chart**

* **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)**