**System Programming Project 4**

담당 교수 : 김영재

이름 : 권기윤

학번 : 20161565

1. **개발 목표**

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

- 이 프로젝트는 system-level process control, process signalling, interprocess communication과 프로세스 작동과 Linux shell의 background의 jobs 작동을 구현하여 구현한 내용들의 이론과 작동방식에 대해 이해할 수 있다.

- phase1 에서는 간단한 명령어를 처리하는 shell 구현, phase2 에서는 phase1을 바탕으로 명령어 간의 pipeline이 발생했을 때의 구현, phase3에서는 background process의 처리를 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Phase 1

간단한 linux shell 명령어(ex. cd, ls… )이 작동되는 simple shell을 구현한다.

1. Phase 2

Phase 1에 추가적으로 입력으로 multiple chain of pipe이 주어졌을 때 이를 순차적으로 해결하고, 구현한다.

1. Phase 3

미구현

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

eval 함수에서 main에서 받은 사용자의 입력이 valid input(cd command 제외)이라면 Fork()를 이용해 child process를 생성한다. Child process에서는 execve 함수를 사용하여 명령어에 대한 처리를 담당한다. Execute 함수는 정상작동이 된다면 해당 process가 종료되는 특징이 있다. 따라서 해당 child process가 정상적으로 execve함수를 실행했다면 자동으로 process가 종료가 된다. Parent process는 background 명령어가 입력되지 않은 이상 child process가 해당 명령어의 처리를 다 할 때까지 waitpid함수를 사용하여 기다린다.

* **Phase2 (pipelining)**
  + Pipeline( ‘|’ )을 구현한 부분에 대해서 간략히 설명 (design & implementation)
  + Pipeline 개수에 따라 어떻게 handling했는지에 대한 설명

Phase1에서 구현한 내용을 기반으로 하여 코드를 작성하였다. checkPipe 함수를 통해 pipeline (‘|’)이 포인터 배열 argv 에 있는지 확인하고, 있으면 2차원 포인터 배열 pipearr에 pipeline 기준으로 argv를 잘라 저장해주었다. Pipeline이 없으면 Phase1과 동일한 방식으로 명령어를 실행시켜 주었다. Pipeline이 있다면, doPipeline() 함수에서 처리해 주었다. 최대 pipe의 개수는 #define MAXPIPE의 개수로 결정이 되며 초기값은 9개까지 pipelining이 가능하도록 구현하였다. 구현 방법은 아래와 같다. Pipearr[idx]의 첫번째 index가 null 일 때, 즉 명령어가 없을 때까지 while문을 돌며 각 명령어마다 pipe(fd)로 pipe를 만들어주고, Fork() 로 process를 복제한다. Child process는 출력을 fd[1]로 넘겨주고, fd를 닫고 해당 명령어를 실행하며 종료한다. Parent process 입력을 fd[0]로 넘겨주고 child가 끝날끝날 때 기다린다. 그 후에 마지막 명령어를 execvp명령어로 실행하여준다.

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

- 미구현

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

Phase1

- skeleton code의 eval() 함수에서 shell 명령어 실행을 위해 execve(argv[0], argv, environ); 함수에서 argv[0]를 앞에 해당 명령어가 저장되어있는 “/bin/” 으로 path를 지정해주기 위해 strcpy, strcat과 같은 함수를 사용하여 “/bin/” + argv[0] 의 형태로 저장하여 다시 위 execve함수의 parameter로 넘겨주었다.

- builtin command에 포함되어있지 않은 ,즉, “/bin/” directory에 구현되어있지 않은 “cd” command에 대해서는 builtin\_command() 함수에서 따로 경우를 나누어 구현해 주었다.

1. “cd”, “cd ~” 과 같은 현재 directory에서 home directory로 이동하는 명령어의 경우, chdir(getenv(“HOME”))을 사용하여 환경변수 “HOME”이 저장하고 있는 home경로의 directory로 이동하게 구현하였다.

2. “cd filename” 와 같은 일반적인 명령어는 chdir(argv[1]), 즉, filename에 해당하는 path로 directory이동이 가능하도록 해주었다.

- grep “abc def” filename과 같은 명령어의 경우, parseline에서 “abc def” 를 하나의 argument로 구현하기 위해 ‘ ’로 초기화한 parse변수를 사용해 delim = strchr(buf, parse) 로 구현하여 “\”” 문자가 나오면 parse = “\””로 바꿔 처리해 주었다.

Phase2

- Phase1의 코드를 기반으로 작성하였다. Pipeline을 기점으로 명령어를 저장하기 위해 2차원 pointer 배열 pipearr를 선언하였다.

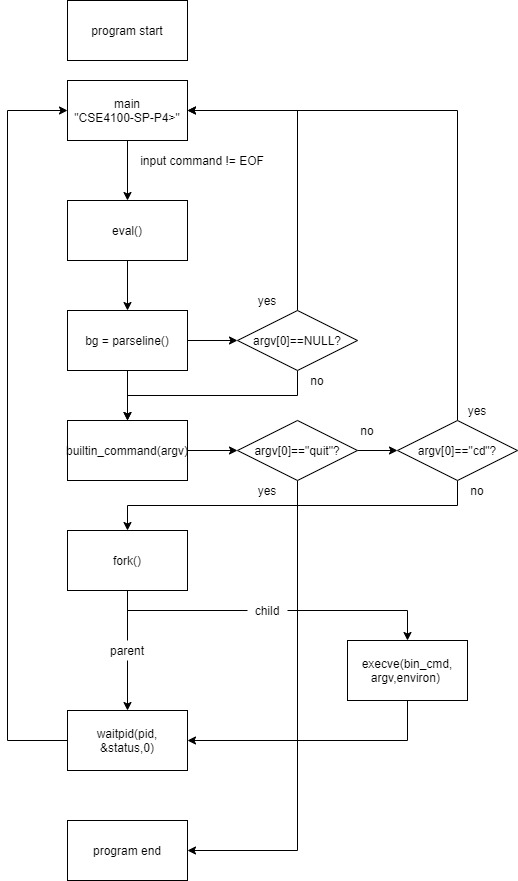
- checkpipe() 함수에서 builtin\_command를 호출하기 이전에 argv를 검사하여 pipeline이 있는지를 check한다. 없다면 phase1과 동일하게 출력되도록 만들었고, 있다면, doPipeline()함수를 호출하였다.

- doPipeline()함수에서는 checkpipe에서 완성된 pipearr를 가지고 수행할 명령어가 없을 때까지 반복하여 pipelining과 fork()를 해주었다. Child process에서 출력이 화면으로 출력되지 않고, fd[0]를 통해 출력되고, fd[0]에서 출력값을 받아올수 있도록했다.

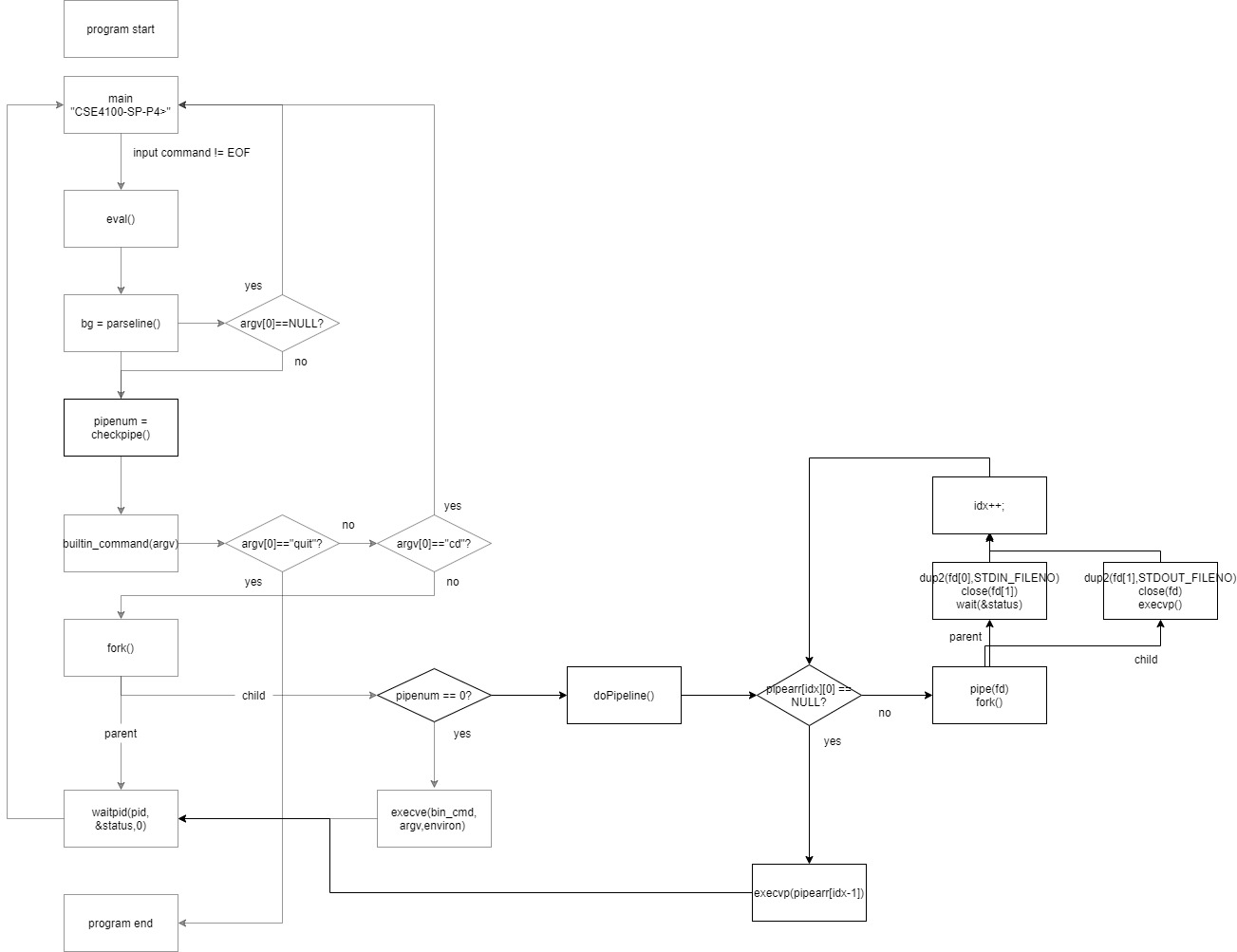
1. **구현 결과**
   1. **Flow Chart**

* **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)**

**미구현**