**System Programming Project 1**

담당 교수 : 김영재 교수님

이름 : 조다민

학번 : 20181297

1. **개발 목표**

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

Linux shell에서의 동작을 이해하고 총 세 개의 단계로 이를 나누어 구현한다. 첫 번째 단계에서 fork를 이용해 child process를 생성해 ‘cd’, ‘ls’, ‘echo’ 등 shell에서의 기본적인 명령어들이 실행가능 하도록 구현, 두 번째 단계에서는 첫 번째 단계에 더해서 pipe(“|”)line 기능 또한 작동할 수 있도록 구현, 세 번째 단계에서는 background, foreground 프로세스로 나누어 작동할 수 있도록 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Phase 1

Command line에서 입력 받은 내용을 공백(‘ ‘)을 기준으로 분리 후, 명령어를 확인해 ‘cd’와 ‘exit’을 제외한 ‘ls’, ‘echo’등의 기본 명령어는 fork()로 child process를 생성하고, execvp 함수를 이용해 명령어를 실행할 수 있다.

‘exit’의 경우 shell을 정지할 수 있도록 exit 함수를 이용해 프로그램을 종료하도록 프로그램을 작성하였다.

‘cd’의 경우 changeDir 함수를 작성해 이동하고자 하는 위치를 파악 후 chdir로 이동하도록 프로그램을 작성하였다.

프로그램 실행 결과는 다음과 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그림 : phase1 실행 결과. 따옴표의 처리 또한 구현하였다.

1. Phase 2

Phase 1에서와 같이 Command line을 이용해 받은 명령어를 처리하는데, pipe가 있는 경우와 없는 경우로 나누어 명령어를 처리함에 있어 차이를 두었다.

Pipe가 있는 경우는, pipeCmd 함수를 이용해 recursive하게 명령어를 처리할 수 있도록 구현하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그림 : phase2 실행 결과. pipe 가 잘 작동함을 확인할 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

: eval 함수 내에서 exit, cd를 제외한 명령어는 fork를 이용해서 child processor를 생성한다. 이 때, pid\_t pid 변수를 이용해 프로세스 id를 저장하고, 그 결과를 이용해 pid 가 제대로 생성되었는지 아닌지 확인하고, pid가 0인 경우, execvp를 사용해 명령어가 실행될 수 있도록 한다.

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

: parent process는 fork를 이용해 child process를 생성하고, waitpid 함수를 이용해 생성된 child process의 종료 시그널을 기다린다. 이 때 child process가 종료되면 signal을 감지하고 종료된 child process를 reaping한다.

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

: pipeline을 구현하기 위해 먼저 pipe가 있는 경우 phase1에서 띄어쓰기를 이용해 command를 argv에 저장한 것처럼, cmd에 pipe를 기준으로 parsing한 결과를 저장하고, pipeCmd 함수에서 재귀적으로 명령어를 처리한다. 명령어 실행은 phase1과 같이 execvp를 이용해 실행한다.

* + Pipeline 개수에 따라 어떻게 handling했는지에 대한 설명

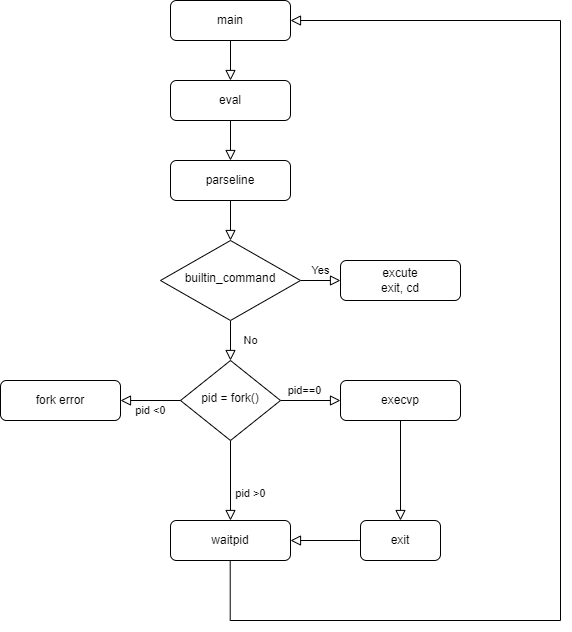
: Pipeline 처리를 위해 명령어를 저장하는 배열을 MAXARGS 개의 pipeline을 담을 수 있도록 변경하였다. 하지만 한번에 입력할 수 있는 최대의 cmdline 길이를 MAXLINE 으로 한정해두었기 때문에 이를 넘어가지 않도록 유의해야 한다.

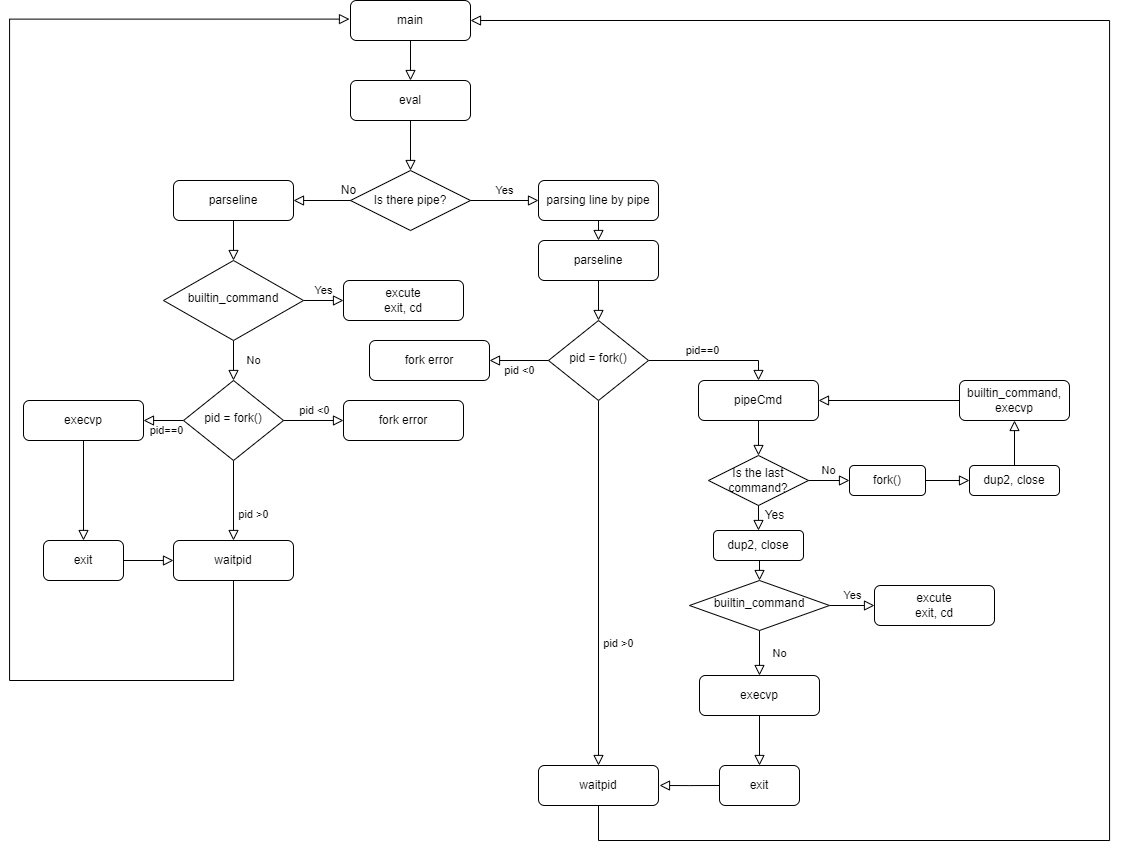
* **Phase3 (background process)**
  + Background (’&’) process를 구현한 부분에 대해서 간략히 설명
  1. **개발 방법**
* **B.의 개발 내용을 구현하기 위해 어느 소스코드에 어떤 요소를 추가 또는 수정할 것인지 설명. (함수, 구조체 등의 구현이나 수정을 서술)**
* Phase 1
* int main(): signal 함수를 이용해 <ctrl>+<C>, <ctrrl>+<Z>를 눌렀을 때 shell 프로그램이 완전히 종료되지 않고, shell에 남아있거나, myshell에서 실행되던 프로그램만 stop될 수 있도록 하였다. 또한 받은 cmdline을 eval 함수를 불러 해결하는 역할을 하는 함수이다. 종료 message가 입력될 때까지 입력 받는 것을 반복한다.
* void eval(char\* cmdline): 받아온 command line을 paseline함수를 이용해 char \* agrv[MAXARGS]에 저장, builtin\_command로 명령어의 종류를 확인 한 후 일반적인 명령어의 경우만 fork()를 사용해 child process를 만들고 execvp 함수를 통해 실행한다. 이 때 execvp함수의 전달 인자로는 argv[0](명령어의 종류)와 명령어 전체를 전달한다. 그리고 fork 하면서 parent process는 waitpid를 통해 child process에서 전달해주는 종료 signal을 대기하고 있다가 signal이 도착하면 child process를 reaping한다.
* int builtin\_command(char\*\* argv) & void changeDir(char\*\* argv): 이 두 함수는 fork 후 일반적으로 실행하는 명령어들과 달리 exit, cd의 명령어를 처리하기 위해 작성된 함수이고, 여기서 background process를 위한 ‘&’ 또한 무시하기로 한다. “exit”이 입력되면 myshell을 종료할 수 있도록 exit(1)을, “cd”가 입력되면 changeDir함수를 이용해 원하는 폴더로 이동할 수 있도록 함수를 작성하였다. changeDir 함수 내부에서는 chdir함수를 이용해 폴더 이동을 실시한다.
* int parseline(char\* buf, char\*\* argv): cmdline을 띄어쓰기 단위로 분리하는 함수. cmdline(buf) 내부에 따옴표(‘\”’, ‘\’’)의 유무에 따라 저장하는 방식이 조금 달라진다. buf를 기본적으로는 공백(“ “)을 기준으로 나눠 argv[i]에 저장한다. 만약 따옴표가 내부에 있다면 공백을 기준으로가 아닌 따옴표 쌍을 기준으로 argv에 buf를 나눠 저장할 수 있도록 한다.
* Phase 2
  + - void eval(char \* cmdline): phase1에서와 다르게 argv를 char\* argv[MAXARGS][MAXARGS]로 선언하였고, 추가로 pipe를 기준으로 나눈 command를 저장하기 위해 char cmd[MAXARGS][MAXLINE]을 추가로 선언하였다. 처리는 pipe가 있는 경우와 없는 경우로 나누어 실행될 수 있도록 하였고, pipe가 있는 경우는 Phase1에서의 따옴표 처리와 비슷한 방식을 이용해 pipe 기준으로 command line에 입력된 내용을 분리해 cmd에 저장, 분리가 완료되면 phaseline 함수를 호출해 cmd에 저장된 내용을 argv에 저장한다. 이 후 fork가 성공적으로 되면, pipeCmd를 호출해 처리가 된 명령어들을 실행할 수 있도록 한다. Pipe가 없는 경우는 phase1에서의 eval 함수의 내용을 수정된 변수에 맞춰 작성하였다.
    - void pipeCmd(char\* argv[][MAXAGRS], int pos, int in\_fd): 함수 내부에서 last command인지 아닌지를 판단해 last command가 아닌경우, command 별로 pipe함수를 실행하여 discripter를 fd[0], fd[1]에 저장, dup2와 close 함수를 이용해 intpu과 output을 STDIN\_FILENO, STDOUT\_FILENO에 전달하고 execvp를 이용해 command를 실행 후 또 다음 pipe에서 사용할 수 있도록 함수를 작성하였다. 다음 command는 pipeCmd를 재귀적으로 호출하여 실행할 수 있도록 하고, parent process는 waitpid를 이용해 child process의 signal을 기다려 처리한다.

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)**
2. **Phase 3 (background)**