System Programming Project 1

담당 교수 : 김영재 교수님

이름 : 박성환

학번 : 20160169

1. 개발 목표

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

나만의 myshell을 구현한다. 기본적인 bash shell과 같이 cd, touch, mkdir, rm, echo등의 명령어를 수행하도록 하고 pipe를 구현하여 한 번에 여러 개의 명령을 처리할 수 있도록 한다. 마지막으로 foreground와 background를 구현하여 명령어가 background에서 수행될 수 있도록 한다.

2. 개발 범위 및 내용

A. 개발 범위

- 아래 항목을 구현했을 때의 결과를 간략히 서술

1. Phase 1
    - 기본적인 shell 명령어들이 작동하도록 한다. myshell에서 cd, touch, mk, mkdir, rm, rmdir, echo, sort file, grep 등의 명령어들을 수행한다. grep이나 echo의 경우에는 따옴표 안에 있는 문자열로 명령이 수행되도록 하였다.
2. Phase 2
    - Pipeline을 구현하였다. cat myshell.h | grep “char” | sort 와 같이 한 번에 여러 개의 명령을 처리할 수 있도록 하였다.
3. Phase 3
    - Background와 Foreground를 구분하여 명령을 수행할 수 있도록 하였다. 명령을 background로 수행하면 해당 명령이 수행되는 동안 foreground에서 다른 명령을 수행할 수 있다.

B. 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

  ◆ built_in_command가 아닐 경우에 fork를 통해 child process를 생성한다. 이후 명령어에 해당하는 file의 위치와 parsing된 command를 execve 함수에 넘겨 명령을 수행한다.

  ✓ connection을 종료할 때 parent process에게 signal을 보내는 signal handling 하는 방법 & flow

  ◆ waitpid 함수를 통해 child process가 종료될 때까지 parent process가 기다릴 수 있도록 한다. waitpid 함수가 child process의 pid를 반환하면 chlid를 회수하고 waitpid 함수가 음수를 리턴하여 오류가 났음을 알리면 shell을 종료한다.

- **Phase2 (pipelining)**

  ✓ Pipeline('|')을 구현한 부분에 대해서 간략히 설명 (design & implementation)

  ◆ eval 함수를 재귀호출하여 구현하였다. Command에 pipe가 포함되어 있으면 첫 번째 pipe까지만 parsing하여 parsing된 명령을 수행하고 그 결과와 남아 있는 명령어를 다시 eval 함수에 전달하는 식으로 재귀호출하였다.

  ✓ Pipeline 개수에 따라 어떻게 handling했는지에 대한 설명

  ◆ Recursive하게 구현하여 개수 제한은 없다. 다만 pipe의 read, write에 사용할 fd 배열의 개수를 1024개로 설정하였다.

- **Phase3 (background process)**

  ✓ Background ('&') process를 구현한 부분에 대해서 간략히 설명

  ◆ 우선 입력되는 모든 명령들을 jobList 배열에 추가한다. Foreground 명령인 경우에는 waitpid 함수가 child를 회수할 때 바로 jobList 배열에서 삭제를 한다. Background command가 입력되면 일단 waitpid 함수로 parent process가 기다리지 않도록 하였다. 그리고 시그널로 background command를 제어하는 경우에 사용될 modifiedWait 함수를 따로 구현하

였다.

C. 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

◆ Builtin_command 함수에서 cd 명령어만 처리하도록 해주었다. Cd 명령어는 chdir 함수를 이용하여 구현하였다. Builtin_command 함수에서 명령어가 grep 또는 exit일 경우에는 그 다음 argument를 적절히 parsing 하기 위한 코드를 추가하였다. Exit 명령어의 경우에는 fork된 child process에서 SIGUSR2 시그널을 보내도록 하였고 parent process에 SIGUSR2를 받는 signal handler를 설치하여 SIGUSR2 시그널을 받으면 main 함수의 while문이 종료되도록 하였다. Child process를 회수하기 위해 waitpid 함수를 사용하였다.

✓ Phase 2

◆ eval 함수에 pipe의 존재 유무를 확인하고 만약 존재한다면 첫 번째 pipe 이전의 명령어만 추출하여 parseline 함수에 넘기도록 하였다. 이를 통해 첫 번째 명령어를 수행할 수 있도록 한다. 그리고 main 함수에서 받은 pipe를 위한 fd 배열을 이용하여 pipe를 생성하였다. 현재 command에서 파이프의 존재가 확인된 경우에는 다시 eval 함수를 호출하는데 첫 번째 pipe까지 제거된 명령어와 fd 포인터를 2만큼 증가시켜 호출하였다. 이를 통해 parent와 child process가 pipe를 통해 통신할 수 있도록 하였고 현재 명령어의 결과를 다음 명령어가 처리할 수 있도록 하였다.

✓ Phase 3

◆ 우선 foreground, background를 구분하기 위한 enum type ground를 선언하고 내부값으로 NONE, FG, BG, SUSPEND를 선언하였다. 그리고 process의 상태를 나타내기 위해 enum type state를 선언하고 RUN, STOP, DONE, DONENOSIG를 값으로 주었다. DONENOSIG는 백그라운드 프로세스가 특정한 시그널을 받지 않고 종료된 경우를 의미한다. 그리고 Job

구조체를 선언하였다. 구조체 멤버로 job에 해당하는 process의 pid, jobList 배열에서의 인덱스, command를 위한 문자 배열, command의 길이를 저장하는 cmdLen을 선언하였다. 또한 앞에 언급한 두 열거형 변수도 선언하였다. 마지막으로 job 자료형의 jobList 배열을 선언하였다.

- ◆ jobList 배열을 관리하기 위해 job을 삽입하는 addJob, 삭제하는 delJob, job을 출력하는 printJob, 현재 job의 command를 foreground, background에 따라 변경하는 updateJobCMD 함수 등을 선언하였다.
- ◆ Eval 함수에서는 일단 fg, bg, kill을 제외한 모든 command를 jobList에 삽입한다. Foreground 커맨드의 경우 waitpid로 child process가 회수된 경우 jobList에서 바로 삭제한다. Background command를 받은 경우에는 jobList에 삽입한 후 명령을 실행하지만 wait를 하지 않는다.
- ◆ Fg, bg, kill 명령어는 builtin_command 함수에서 처리하였으며 jobList 배열에서 현재 job에 해당하는 job을 검색하여 job의 ground 변수와 state 변수를 통해 process의 상태를 조종하도록 하였다. 그런데 background process는 process가 종료되어도 waitpid로 회수가 안되기 때문에 modifiedWait 함수를 구현하여 background process를 회수할 수 있도록 해주었다. modifiedWait 함수는 특정 프로세스의 상태에 대하여 무한루프를 돌다가 프로세스의 상태가 STOP, DONE 등으로 바뀌면 무한루프를 탈출한다.
- ◆ 또한 SIGINT, SIGTSTP, SIGCHLD 시그널들을 관리하기 위한 함수를 만들었다. SIGINT 핸들러는 종료된 job들을 job에서 제거한다. SIGTSTP 핸들러는 jobList에서 foreground이면서 RUN 중인 job을 찾아 상태를 STOP으로 바꾼다. SIGCHLD 핸들러는 waitpid 함수를 통해 child process를 회수하고 waitpid가 제대로 수행된 경우 jobList에서 해당 job을 검색하여 상태를 DONE으로 바꾼다. DONE으로 바뀐 job은 main 함수에서 DONE을 출력한 후 jobList에서 삭제된다.

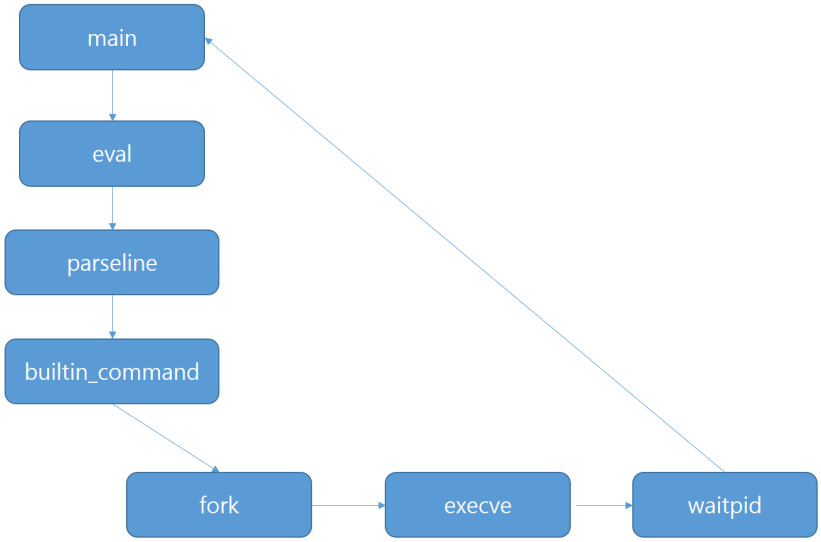
3. 구현 결과

A. Flow Chart

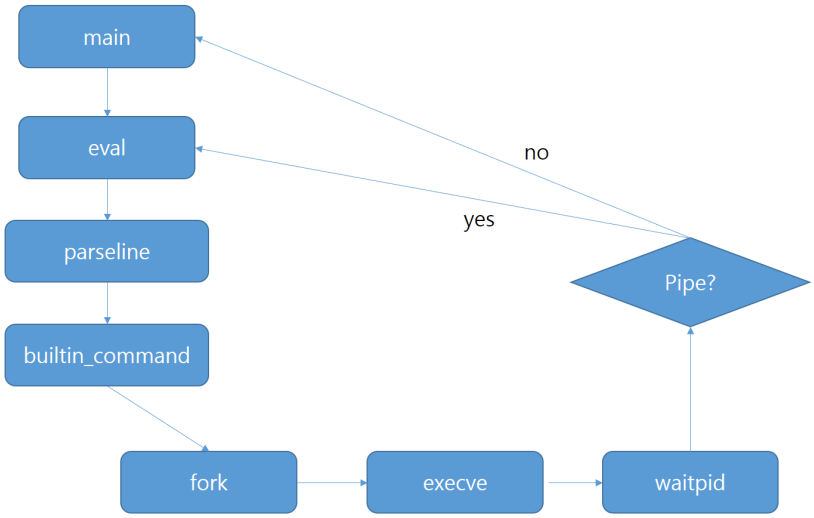
- **2.B.개발 내용에 대한 Flow Chart를 작성.**

- (각각의 방법들에서 추가된 내용(fork, pipeline, background)만 특성이 잘 드러나게 그리면 됨.)

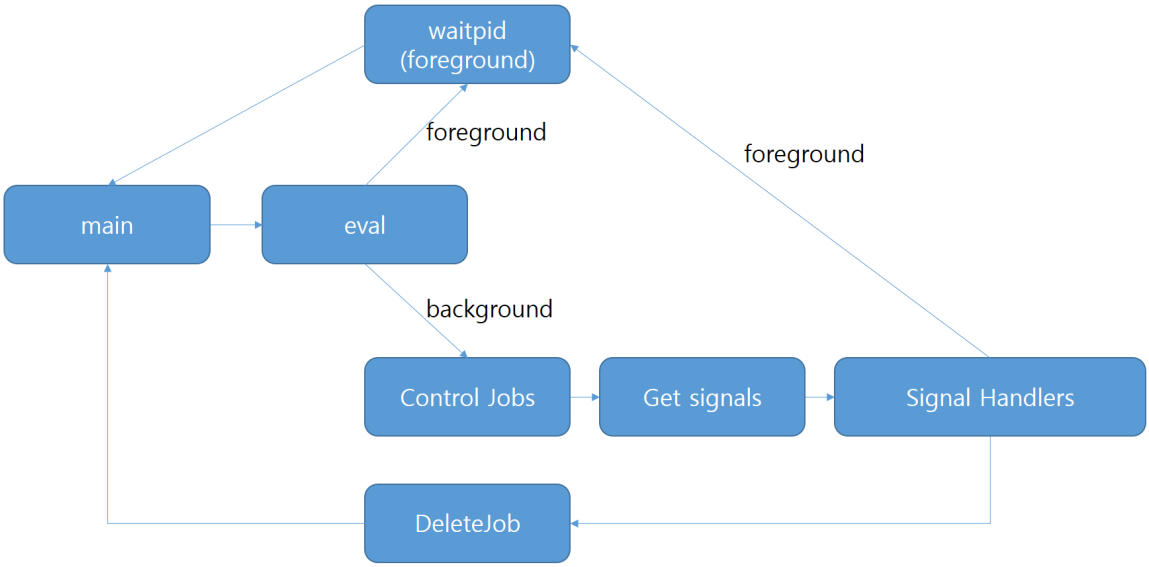
1. **Phase 1 (fork)**



2. **Phase 2 (pipeline)**



3. **Phase 3 (background)**


waitpid
(foreground)
foreground
foreground
main
eval
background
Control Jobs
Get signals
Signal Handlers
DeleteJob