**System Programming Project 1**

담당 교수 : 김영재 교수님

이름 : 전용본

학번 : 20181683

1. **개발 목표**

Myshell 프로젝트는 bash shell을 c언어를 통해 3개의 phase를 수행하며 구현하는 것이 목표이다.

Phase1에서는 fork()를 이용해 child process는 입력된 command를 execute하고 parent process 는 wait()를 이용해 child process가 끝나기를 기다리며 가장 기본적인 형태의 shell을 구현한다.

Phase2에서는 phase1에 pipe ‘|’ 기능을 추가한다. Pipe를 기준으로 command를 parsing하고 한 파이프의 output이 다음 파이프의 input이 되고 마지막 파이프의 output이 출력되도록 설계한다. Pipe는 recursive하게 구현한다.

Phase3에서는 phase2에 background ‘&’ 기능을 추가한다. Command의 마지막에 ‘&’이 있으면 background에서 process를 수행한다. Background process이 끝날 때까지 wait을 통해 기다려주지 않는다. Background job들을 control하기 위한 built-in command(jobs, bg, fg, kill) 또한 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**
2. Phase 1

Myshell을 실행하고 command를 입력하면 결과가 출력된다. Built-in command인 경우에는 개발자가 정의한 함수가 실행되고 built-in command가 아닌 경우에는 child process에서 exec()를 이용해 실행되고 parent process에서는 wait()을 이용해 끝날 때까지 기다린다. Command ‘exit’ 을 입력하기 전까지는 계속 myshell을 사용할 수 있다.

1. Phase 2

입력한 Command에 pipe ‘|’가 존재하면 각 pipe의 output은 다음 파이프의 input이 되며 마지막 pipe의 output이 결과로 출력된다. 각 파이프는 child process에서 실행되고 parent process는 파이프의 실행이 끝나면 recursive하게 다음 파이프를 실행하게 한다. 입력한 command에 pipe ‘|’가 존재하지 않을 경우 phase1 과 똑같이 실행된다.

1. Phase 3

입력한 command에 background ‘&’가 존재하면 command는 background에서 수행된다. Background에서 수행될 동안 쉘은 기다리지 않으며 다음 command를 수행할 수 있다. Background에서 수행되고 있는 job들을 보거나 control하기 위해 jobs, kill, bg, fg command를 사용할 수 있다. 또한 ctrl+c, ctrl+z 를 이용해 foreground에서 실행중인 process를 suspend하거나 종료할 수 있다.

* 1. **개발 내용**
* **Phase1 (fork & signal)**
  + fork를 통해서 child process를 생성하는 부분에 대해서 설명
  + connection을 종료할 때 parent process에게 signal을 보내는 signal handling하는 방법 & flow

command를 입력한 후에 eval함수가 호출된다. Eval함수에서 command를 space ' ‘를 기준으로 command를 parsing하고 command가 built-in함수가 아니라면 pid=fork()를 통해 child process를 생성하고 command를 수행한다.

Fork wrapper를 이용해 pid < 0인 경우 오류 메세지를 출력하도록 했다.

Pid = 0 인 경우 child process 이므로 execvp함수를 이용해 parsing된 command를 수행하고 끝나면 parent process에게 signal을 보낸다.

Pid >0 인 경우 parent process 이므로 waitpid함수를 이용해 child process에서 signal이 올 때까지 기다리고 signal이 온 경우 다시 작업을 수행한다.

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

Command에서 Pipe ‘|’를 기준으로 공백유무에 상관없이 실행되어야하고 parseline함수에서는 공백을 기준으로 parsing하기 때문에 의도적으로 pipe앞뒤로 공백을 추가해주고 parsing했다. Parsing의 결과물에 argument들과 pipe ‘|’ 이 존재하는데 이를 통해 pipe ‘|’를 NULL값으로 바꿔주어 argument들이 pipe ‘|’를 기준으로 나뉠 수 있도록 했고 바꿔줄 때마다 위치와 개수를 기록해 다음 수행할 argument의 위치를 알 수 있도록 했다.

go\_pipe함수에 parameter로 지금 수행할 argument, 다음 argument index, 남은 pipe ‘|’ 개수를 넘긴다. pipe함수를 이용해 파이프를 생성하고

Pid < 0 인 경우 에러 메세지를 출력한 후 종료한다.

Pid = 0 인 경우 child process이므로 parameter로 받은 argument를 수행하고 결과값을 dup2()를 이용해 output을 출력하지 않고 전송한다.

Pid > 0 인 경우 parent process이므로 child process에서 수행이 끝날 때까지 기다린 후에 go\_pipe함수를 호출하며 다음 수행할 argument, 그 다음 argument index, 남은 pipe ‘|’ 개수를 넘긴다.

남은 pipe ‘|’의 개수가 0일 경우 마지막 argument이므로 수행한 argument의 output을 전송하지 않고 출력한다.

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

입력된 command에 background ‘&’가 존재할 경우 phase1,2과는 다르게 parent process에서 process의 종료를 기다려주지 않는다. 또 해당 process의 정보(pid, running, bg, cmdline)를 job배열에 추가한다.

Foreground에서 수행되던 process에 ctrl+z를 입력한 경우 해당 process의 정보(pid, stopped, bg, cmdline)를 job배열에 추가한다.

Foreground에서 수행되던 process에 ctrl+c를 입력한 경우 해당 process를 종료한다.

Command ‘bg’를 이용해서 background stopped process를 background running process로 control할 수 있다.

Command ‘fg’를 이용해서 background stopped/running process를 foreground running process로 control할 수 있다.

Command ‘kill’를 이용해서 process를 kill할 수 있다.

Command ‘jobs’를 이용해 현재 background process들의 상황을 볼 수 있다.

.

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

Eval함수는 입력받은 command를 수행해주는 함수이다. Change\_cmdline함수를 호출해 pipe ‘|’, background ‘&’ 앞뒤로 공백을 붙여 command를 parsing하게 쉽게 하고 parseline함수를 호출해 space ‘ ‘를 기준으로 parse하고 각 문자열들을 argv에 저장한다. Built-in함수를 호출해 built-in command이면 따로 구현한 작업을 수행하고 eval함수를 종료한다.

Built-in command가 아니라면 fork를 이용해 child process를 생성하고 작업을 수행한다. Command에 pipe ‘|’가 있다면 go\_pipe함수를 이용해 재귀적으로 pipe command를 수행한다. Pipe ‘|’가 없다면 execvp함수를 이용해 command를 수행한다. Parent process에서는 background일 경우 child process의 종료를 기다리지 않고 process의 정보(pid, status, bg, cmdline)을 job자료구조에 저장하고 종료한다. Foreground일 경우 child process가 종료될 때까지 waitpid()함수를 이용해 기다리고 종료한다.

* + 1. Int builtin\_command(char \*\*argv)

Builtin\_command함수는 parsing된 command, argv를 parameter로 받아서 command가 builtin\_command일 경우 따로 구현한 작업(Cd, exit, jobs, fg, bg, kill)을 수행하고 1을 return하고 builtin\_command가 아닐 경우 0을 return한다.

* + 1. Int parseline(char \*buf, char \*\*argv)

Parseline함수는 공백을 기준으로 command를 parsing해 argv에 저장하는 함수이다. Parsing외에도 따옴표가 있는 argument도 수행할 수 있도록 기능을 추가했다. Background command일 경우 1을 return하고 아닐 경우 0을 반환한다.

* + 1. Void handler(int signum)

Handler함수는 signal을 처리한다. SIGINT는 foreground에 수행중인 process를 종료하고 job자료구조에서 삭제하게 하고 SIGTSTP는 foreground에 수행중인 process를 suspend하고 job자료구조에 추가하도록 하고 SIGCHLD의 경우 zombie 상태의 process들을 reaping하게 한다. Signal handling 속도가 job자료구조를 처리하는 속도보다 느린 듯해서 sleep함수를 이용해 아주 작은 딜레이를 만들었다.

* + 1. Void updateJob()

이 함수는 foreground 작업이 종료될 때마다 background process 중에 종료된 것이 있나 확인하고 있다면 종료문구를 출력하고 Job 자료구조를 최신화한다.

* + 1. Void delJob(int pid)

이 함수는 job자료구조에서 parameter로 받은 pid를 가진 구조체를 삭제하는 함수이다.

* + 1. Void addJob(pid\_t pid, char \*cmdline)

이 함수는 parameter로 받은 pid와 command를 가진 구조체를 job자료구조에 추가하는 함수이다.

* + 1. Int getIdx(pid\_t pid)

이 함수는 parameter로 받은 pid를 job자료구조에서 찾아 그 index를 return하는 함수이다. 없을 경우 -1을 반환한다.

* + 1. Int getStatus(int pid)

이 함수는 parameter로 받은 pid를 가진 process의 status를 반환하는 함수이다. Suspend : 0, sleep : 1, running: 2, zombie: 3을 각각 반환한다.

* + 1. Int do\_kill(char \*\*argv)

이 함수는 builtin\_command 중 “kill”을 수행하는 함수이다. Command로 받은 jobspec에 해당하는 process를 종료시키고 job자료구조에서 삭제하기 위한 delJob함수를 호출한다. 해당하는 process가 job자료구조에 없을 경우 -1을 반환한다.

* + 1. Int do\_fg(char \*\*argv)

이 함수는 builtin\_command 중 “fg”를 수행하는 함수이다. Command로 받은 jobspec에 해당하는 process가 background stopped/running 상태라면 process에 SIGCONT를 보내 foreground running 상태로 바꿔준다. Background와 foreground process의 차이는 wait() 유무이기에 waitpid()함수로 종료를 기다린 후에 0을 반환한다. Jobspec에 해당하는 process가 job자료구조에 없을 경우 -1을 반환한다.

* + 1. Int do\_bg(char \*\*argv)

이 함수는 builtin\_command 중 “bg”를 수행하는 함수이다. Command로 받은 jobspec에 해당하는 process가 background stopped 상태라면 process에 SIGCONT를 보내 background running 상태로 바꿔준다. Background 이기에 wait은 하지 않고 0을 반환한다. Jobspec에 해당하는 job자료구조가 없으면 -1을 반환한다.

* + 1. Void Printcommand(char \*cmdline)

이 함수는 프로젝트 명세서의 양식을 맞추기 위한 함수이다. printJobs함수에서 job들을 출력할 때 사용한다. 긴 공백과 ‘&’를 지우고 command를 출력한다.

* + 1. Void PrintJobs()

이 함수는 builtin\_command 중 “jobs”를 수행하는 함수이다. Job자료구조를 순회하면서 유효한 background job들을 출력한다.

* + 1. Void change\_cmdline(char\* cmdline)

이 함수는 command에 의도적으로 공백을 만들어주는 함수이다. Pipe ‘|’ 가 존재하는 command를 수행할 때 pipe ‘|’를 기준으로 앞뒤에 공백이 없어도 수행되어야 하지만 parseline함수에서는 space ‘ ‘를 기준으로 parsing하기에 공백을 만들어주었다.

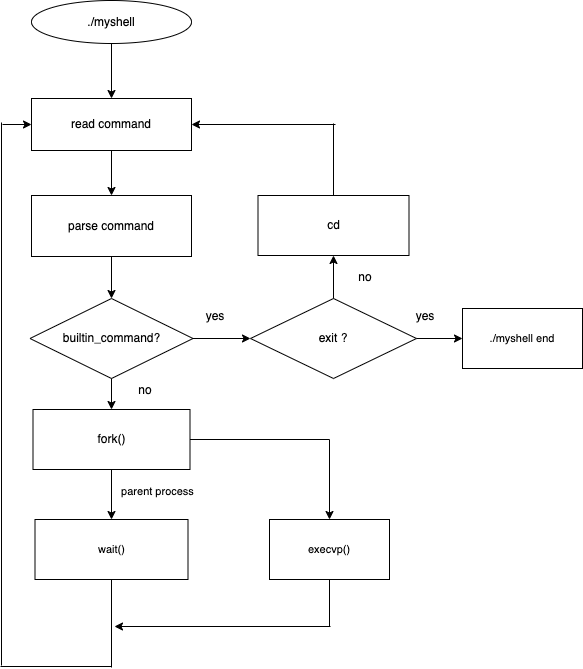
* + 1. Void mydup(int oldfd, int newfd)

이 함수는 go\_pipe함수를 도와주는 macro함수이다. Dup2()와 close()함수의 반복적인 사용을 도와준다.

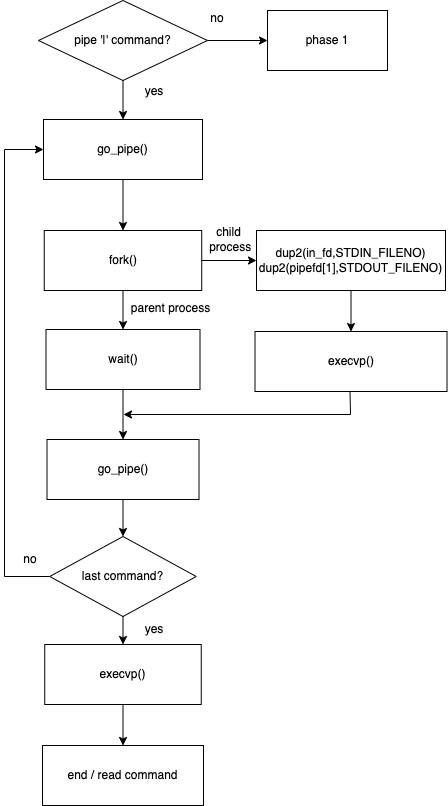
* + 1. void go\_pipe(char \*\*argv, int \*pipe\_idx, int pipe\_cnt, int in\_fd)

이 함수는 pipe ‘|’가 존재하는 command를 수행하는 함수이다. Parameter로 현재 수행할 command, 다음 수행할 command의 index, 남은 pipe ‘|’의 개수를 인자로 받는다. Fork를 통해 생성된 child process에서는 execvp의 output을 출력하지 않고 pipe로 전송하도록 하고 parent process에서는 child process가 종료될 때까지 기다린 후 go\_pipe함수에 다음 수행할 command, 그다음 수행할 command의 index, 남은 pipe ‘|’의 개수를 인자로 보내며 호출한다. 만약 남은 pipe ‘|’의 개수가 0이면 마지막 command이므로 execvp의 결과를 전송하지 않고 화면에 출력한다.

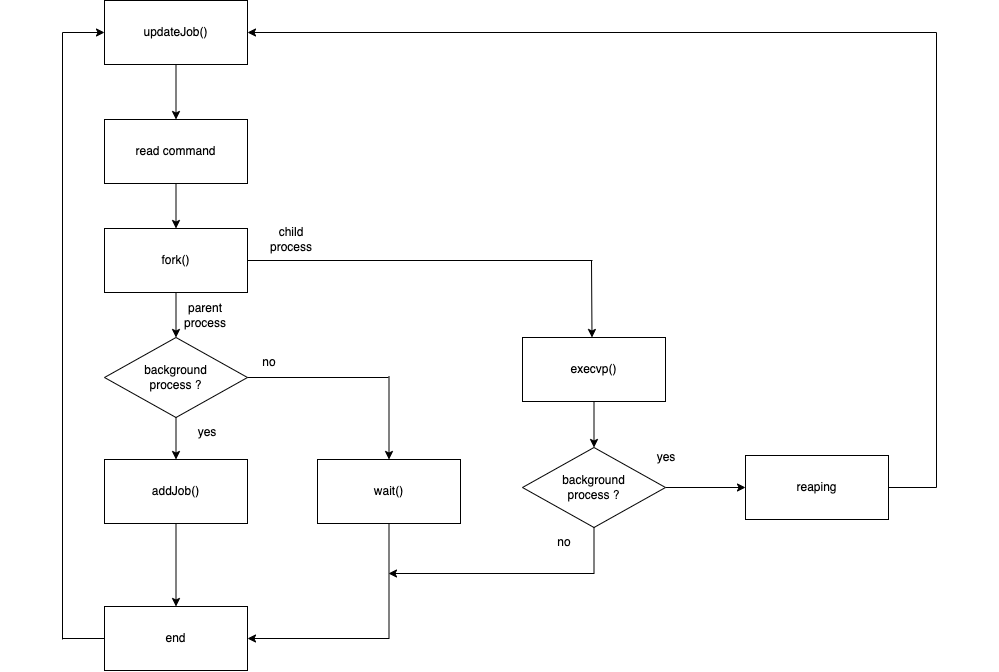
1. **구현 결과**
   1. **Flow Chart**
2. **Phase 1 (fork)**

****

1. **Phase 2 (pipeline)**

****

1. **Phase 3 (background)**

****