**System Programming Project 1**

담당 교수 : 김영재

이름 : 최원빈

학번 : 20181693

1. **개발 목표**

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

이번 프로젝트는 linux 환경에서의 shell을 직접 자신이 구현해본다. Phase1에서는 리눅스 명령어를 받으면 child process를 생성해서 실행하고, builtin\_command인 경우는 parent에서 직접 처리하는 프로그램을 만든다. Phase2에서는 파이프라이닝 기능을 file descriptor와 dup함수들로 구현한다. Phase 3에서는 백그라운드 기능과 간단한 job command를 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

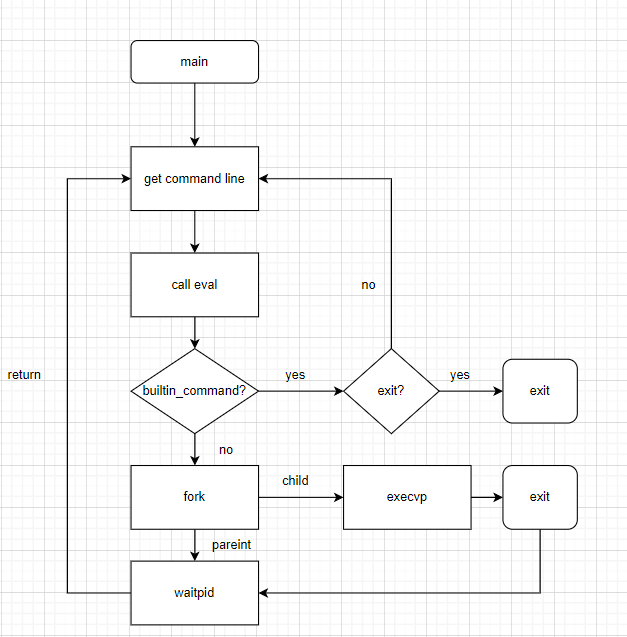
1. Phase 1 : myshell 프로그램의 뼈대를 잡는 단계다. 기본적인 linux 명령어를 fork한 child process에서 execp 함수로 불러와 수행한다. 그외 exit나 cd 등의 command는 builtint\_command에 작성해둬 parent에서 직접 처리하는데 exit이면 종료, cd면 디렉토리를 바꾼다.
2. Phase 2 : 위의 작성한 myshell 프로그램에서 pipeline 기능을 추가한다. |와 함께 여러 명령어를 동시에 commnd로 받으면 알맞게 명령어를 분리해 앞서 명령어의 output이 다음 명령어의 input이 되고 마지막의 경우는 터미널에 출력한다
3. Phase 3 : 작업을 foreground와 background로 나눈다. Command의 마지막에 &를 붙일 경우 작업을 background로 실행하여 바로 다음 명령어를 받을 수 있게 한다. 또 background 작업과 Suspended된 작업들을 여러 jobs command로 관리할 수 있게 한다. Jobs시 현재 멈추거나 백그라운드로 실행 중인 작업들을 출력하고, bg 명령은 멈춘 작업을 백그라운드로 실행시키고, fg명령은 멈추거나 백그라운드로 실행중인 작업을 포그라운드로 실행시킨다. Kill 명령시 멈추거나 백그라운드로 실행중인 프로세스를 종료시킨다.
   1. **개발 내용**

* **아래 항목의 내용만 서술**
* **(기타 내용은 서술하지 않아도 됨. 코드 복사 붙여 넣기 금지)**
* **Phase1 (fork & signal)**
  + Parent의 eval 함수에서 command가 builtin\_command가 아닌 경우 Fork()를 통해서 child process를 생성한다. 이후 child에서 execvp를 통해 받은 명령어를 parseline을 통해 분석한 결과로 프로그램을 찾아 실행한다. Parent에서는 waitpid를 통해 child가 종료되기를 기다린다. 이후 child process가 수행 후 잘 종료되면 signal을 parent에 보내고 parent는 child를 청소한다. 이후 return되고 다음 입력을 받는다.
* **Phase2 (pipelining)**
  + 맨 처음에 command에 |가 있는 경우를 strchr로 감지하면 파이프 전용함수 pipe\_eval을 호출한다. 이후 strtok 함수로 |단위로 나누어 eval을 수행한다. 이때 미리 pipe함수로 열어둔 fd들에 dup2로 STDIN\_FILENO, STDOUT\_FIlENO와 연결해둬 앞 명령어의 출력 결과를 다음 명령어의 입력으로 받을 수 있게 해준다. 기본적으로 pipe\_eval을 recursion 형태로 구현해 개수에 상관없이 잘 작동하게 했고 마지막의 경우 터미널에 결과를 출력하고 종료한다.
* **Phase3 (background process)**
  + 명령어에 &가 포함될 경우 parent에서는 waitpid를 하지 않고 바로 프롬프트를 출력하고 입력을 기다린다. 그리고 phase3에서는 fork를 2번 해주는데 처음 생성한 것을 p2, 그 다음 것을 p3라 했을 때 p2에서 p3를 waitpid로 기다려준다. P3에서 execvp로 프로그램을 실행하고 종료한다.
  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 (fork & signal)**
  + 기본적으로 주어진 skeleton code를 책을 보면서 완성했다. 이후 builtin\_command에 exit일 때 종료, cd일 때 chdir함수를 사용하게 했다. 그외 명령어들은 eval에서 fork로 child process를 생성하고 거기서 execvp를 통해 프로그램을 실행 후 종료했다. Parent process에서는 waitpid로 child process를 기다려주고 청소까지 한다. 주어진 함수들 외 다른 함수나 구조체는 없다.
* **Phase2 (pipelining)**
  + main에서 command에 |가 있는 경우를 strchr로 감지하면 파이프 전용함수 pipe\_eval을 호출한다. 이후 pipe\_eval에서는 cmdline을 strtok 함수로 |단위로 나누어 part\_cmd에 저장하고 이를 인자로 eval을 수행한다. 이때 이전 명령의 출력을 다음 명령의 입력으로 해주기 위해 pipe함수를 이용했는데 인자로는 전역변수로 선언한 크기 2짜리 정수 배열 fd의 주소를 넘겨주었다. Child process에서는 전역변수로 선언한 pipe\_num, pipe\_last, fd2을 통해 마지막이 아닌 경우 fd[1]과 STDOUT\_FILENO를 dup2로 연결해 이번 명령어의 결과를 parent로 보내고, 처음 명령어가 아니면 이전 명령의 결과를 parent에서 저장한 fd2와 STDIN\_FILENO를 연결해준다. 이후 execvp로 실행 후 종료한다. Parent process에서는 waintpid로 기다려주고 청소한다. 만약 pipe\_last가 처음 선언하대로 0이면 fd[1]을 fd2에 연결해주고 닫는다. 마지막일 모든 파이프를 닫아준다. 이후 pipe\_eval에서는 마지막이 아니라면 실행한 명령어를 제외한 cmdline을 인자로 다시 pipe\_eval을 호출한다(재귀의 형태). 마지막임을 아는 방법은 조건에 마지막에 |가 오는 경우는 없다 했으므로 strchr을 했는데 더 이상 안 나오는 경우다.
* **Phase3 (background process)**
  + Phase3에서는 많은 것을 수정하였다. 위에서부터 차례대로 얘기해보면 SIGINT, SIGTSTP, SIGCHLD에 대해 HANDLER를 작성해준다. 이는 CTRL+C, CTRL+Z에 의해 myshell이 종료되거나 정지하지 않기 위해, 그리고 종료한 백그라운드 프로세스를 job\_list에서 삭제하기 위함이다. 또 job이라는 구조체를 선언했는데 이것으로 suspended 혹은 background running 프로세스들에 대한 정보를 job\_list에 저장할 수 있다. 또 fg\_pgid는 우리가 foreground로 정한 job의 그룹아이디를 저장하고 delete·add job 또한 구현했다. main에서는 이전의 eval의 내용을 그대로 실행한다. Command에 &가 존재한다면 Is\_bg를 1로 지정한다. 그뒤 fork를 해주고 child에서는 |의 존재유무, 백그라운드인지 아닌지에 따라 pipe\_eval, eval을 실행한다. 이때 setpgid로 자신의 pid를 그룹아이디로 지정한다. 이는 나중에 signal을 그룹단위로 보내기 위함이다. 그뒤 parent에서는 foreground면 waitpid를 걸어주고, background면 바로 다음 입력을 받는다. Fork된 child에서는 다시 명령어에 따라 fork를 해주고 실행후 종료한다. 즉 parent입장에서는 손주인 프로세스에서 실제로 execvp를 호출하는 것이다. Ctrl+c나 ctrl+z 입력시 핸들러에 의해 우리가 실행한 myshell은 종료 혹은 정지하지 않고 앞서 지정한 fg\_pgid로 kill함수로 sigkill· sigstop을 보내준다. Jobs, bg, fg, kill 명령은 builtin\_command에 추가해 각각 job\_list로 출력, 혹은 해당 pid(그룹아이디)를 찾아서 kill함수로 SIGCONT, SIGKILL을 보내주고 job\_list를 수정한다.

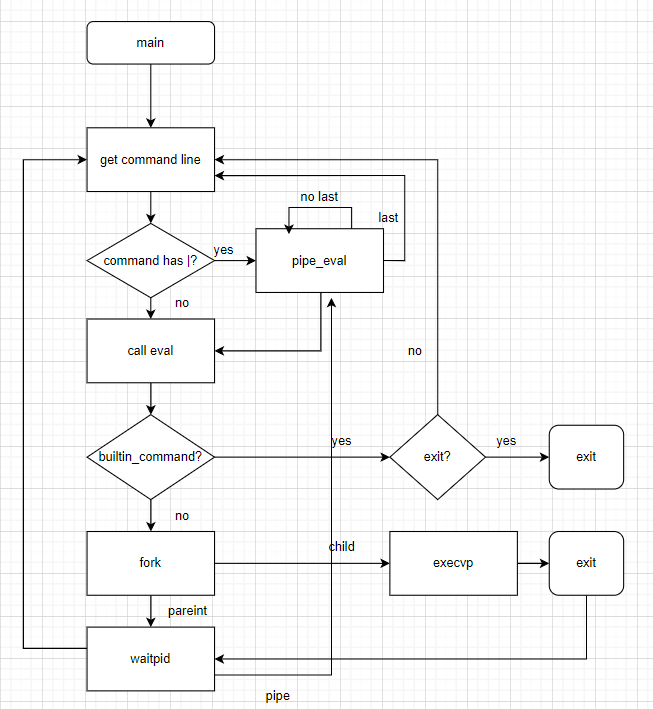
1. **구현 결과**
   1. **Flow Chart**

* **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)**

