**System Programming Project 1**

담당 교수 : 김영재

이름 : 이상윤

학번 : 20171664

1. **개발 목표**

쉘의 대표적인 기능들을 하는 shellex를 Fork()를 통해 구현한다. phase1 에서는 기본적인 명령어, phase2 에서는 파이프라인 명령어, phase3 에서는 백그라운드 실행 명령어를 구현한다.

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

cd, cd..

ls

mkdir, rmdir

touch, cat, echo

exit

등과 같은 기본적인 쉘 명령어들을 실행한다.

cd 의 명령어는 직접 구현하였고, quotation이 있을때의 예외를 최대한 처리하려 하였다.

1. Phase 2

| (파이프라인) 명령어를 실행한다.

사이의 공백과 quotation이 있을때의 예외를 처리하였다.

1. Phase 3

back-ground job, fore-ground job을 수행할 수 있게 하였다.

쉘의 jobs, bg, fg, kill commands 를 기능 추가.

* 1. **개발 내용**
* **Phase1 (fork & signal)**
  + fork를 통해서 child process를 생성하는 부분에 대해서 설명

eval() 함수에서 명령어를 우선 파싱(parsing) 한 후 argv에 저장한다. 그리고 child process에서 수행할 수 있는 유효한 명령어인 경우 Fork()를 통해 child process를 생성한다. 자식 프로세스는 Fork() 에 대한 반환값으로 0을 받게 되고, 부모 프로세스와 구별이 가능하다. Execvp() 함수가 실행이 된 후, child process는 종료된다. parent process는 child process가 종료될 때 까지 기다릴 수 있도록 한다.

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

child process가 종료될 때, parent process에게 SIGCHLD를 보내게 된다. sig\_chld\_handler를 등록하여 만약 context switching 하고 실행되기 딱 바로전에 SIGCHLD을 확인했을때, 실행해야 한다면 parent process에서 sig\_chld\_handler가 호출된다. 그리고 wait\_child\_flag를 통해 만약 sig\_chld\_handler가 실행 되었다면 Sigsuspend로 기다리는 것을 그만하도록 한다.

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

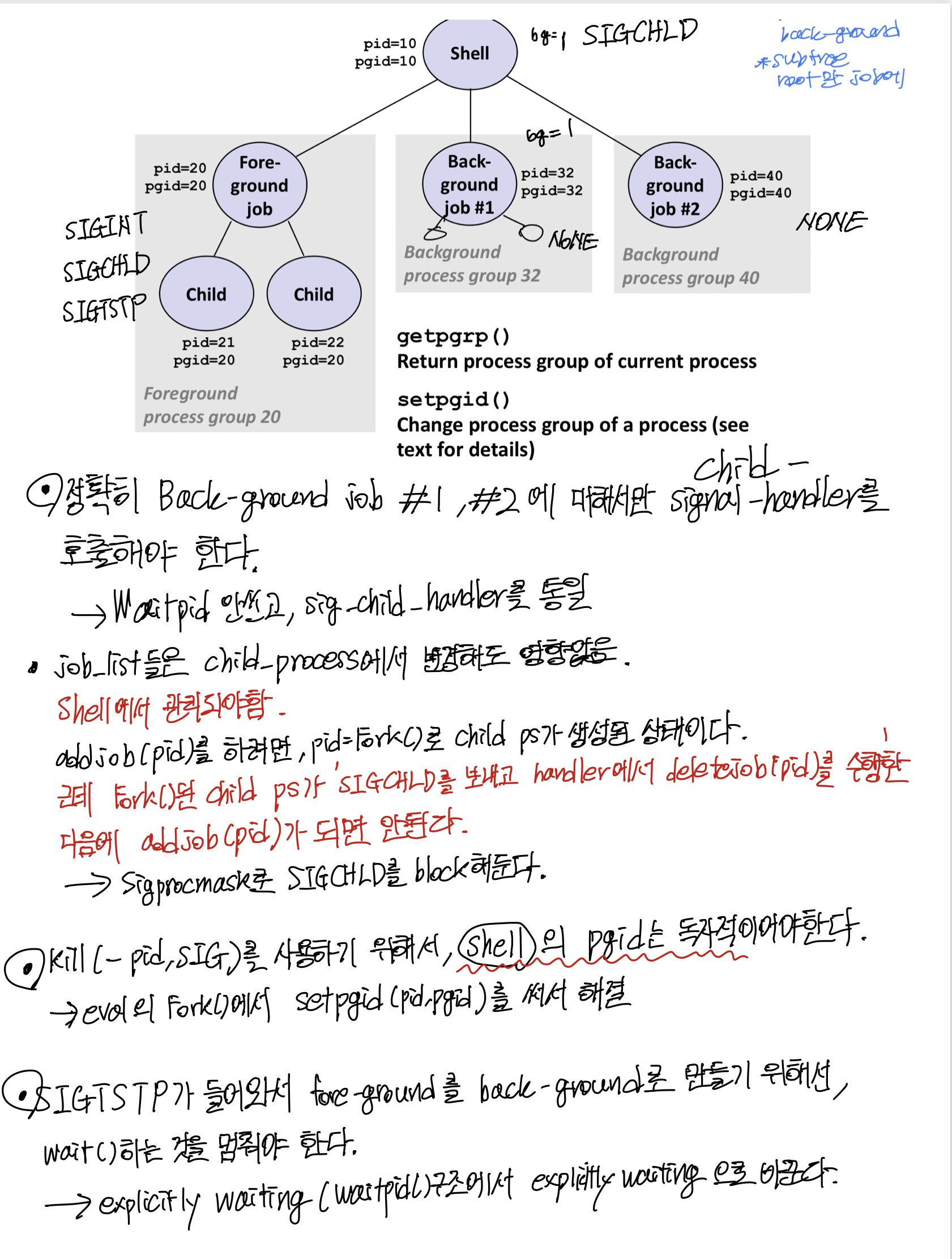
명령어를 파싱(parsing)해서 파이프라인이 존재한다면, pipe\_execution을 실행한다. 설명의 편의를 위해 parent\_process를 pipe\_execution을 실행하는 child\_process로 생각하기로 한다. pipe\_execution(first\_cmd,after\_cmd)는 우선 child\_process에서 first\_cmd를 실행하게 한 다음에, 출력값을 fd[1]에 넣는다. 그리고 parent\_process는 입력값은 fd[0]로 하고 waitpid를 통해 child\_process가 종료될 때 까지 기다린다. 그 후 after\_cmd를 파싱한 다음, 다시 pipe\_execution을 실행한다. 즉, 재귀적으로 해결 가능하다. base case는 더이상 파이프 라인이 존재하지 않을 때이다.

* + Pipeline 개수에 따라 어떻게 handling했는지에 대한 설명

개수가 몇개든지 간에, 위에서 기술한 바와 같이 recursive하게 문제를 해결하였기 때문에 문제가 없다.

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

background process는 foreground process에서 Sigsuspend를 이용해 명시적으로 기다렸던 것과 다르게, 기다리지 않고 바로 다음 명령이 실행되도록 한다. 간단히 구현을 위해 추가한 부분을 기술하면 아래와 같다.



wait\_child\_flag를 통해 명시적으로 기다리게 하였다.

* 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. 전역변수

wait\_child\_flag: (explicitly waiting)parent가 현재 fore-ground job child를 wait하는지의 여부. 0인 경우는 child가 back-ground job으로 바뀌었거나, terminated 되었을때이다.

job\_t job\_lists[MAXJOBS]: parent process와 직접적으로 자식인 child process들을 저장한다. (job을 저장한다) 최대한 링크드 리스트 처럼 동작한다.

int job\_lists\_size=0 : job\_lists.size()

2. 함수

int deletejob(pid\_t pid) : pid와 일치하는 job을 삭제하고, 해당 피드가 없으면 0, 있으면 1을 반환한다.

void addjob(pid\_t pid,char status,char\* cmdline) : 인수로 받은 내용으로 새로운 job을 추가한다.

void sig\_chld\_handler(int sig) : SIGCHLD가 들어왔을때 호출된다.

void sig\_tstp\_handler(int sig) : SIGTSTP가 들어왔을때 호출된다.

void sig\_int\_handler(int sig) : SIGINT가 들어왔을때 호출된다.

void exe\_jobs\_command() : foreground job이 없을때, jobs 명령어가 들어왔을때 실행한다.

void exe\_bg\_command(char\*\* argv) : foreground job이 없을때, bg 명령어가 들어왔을때 실행한다.

void exe\_fg\_command(char\*\* argv) : Change a stopped/running background job to a running forefround job.

void exe\_kill\_command(char\*\* argv) : kill 명령어를 실행한다.

void exe\_cd\_command : /bin/cd는 존재하지 않으므로 직접 구현한다. cd 명령을 실행한다.

void eval(char \*cmdline) 수정내용 : 파이프라인이 존재하는 경우, 백그라운드 명령어가 존재하는 경우도 수행될 수 있게 수정.(파이프라인이 존재하는 경우 pipe\_execution 호출, setpgid로 현재 process의 pid로 pgid를 설정, addjob 함수 실행, wait함수 대신 명시적으로 child process를 기다리게 함 등등)

int builtin\_command(char \*\*argv) 수정내용 : cd, jobs, bg, fg, kill 명령어가 들어왔을때 exe\_jobs\_command()와 같은 함수를 호출하게 수정.

void pipe\_execution(char\*\* first\_pipe\_argv,char\* after\_pipe\_commands) : 파이프라인이 있을때 실행된다. 첫번째 인수는 child process에서 Execvp를 통해 실행된다. 두번째 인수는 파싱된 후, parent process에서 재귀적으로 처리된다.

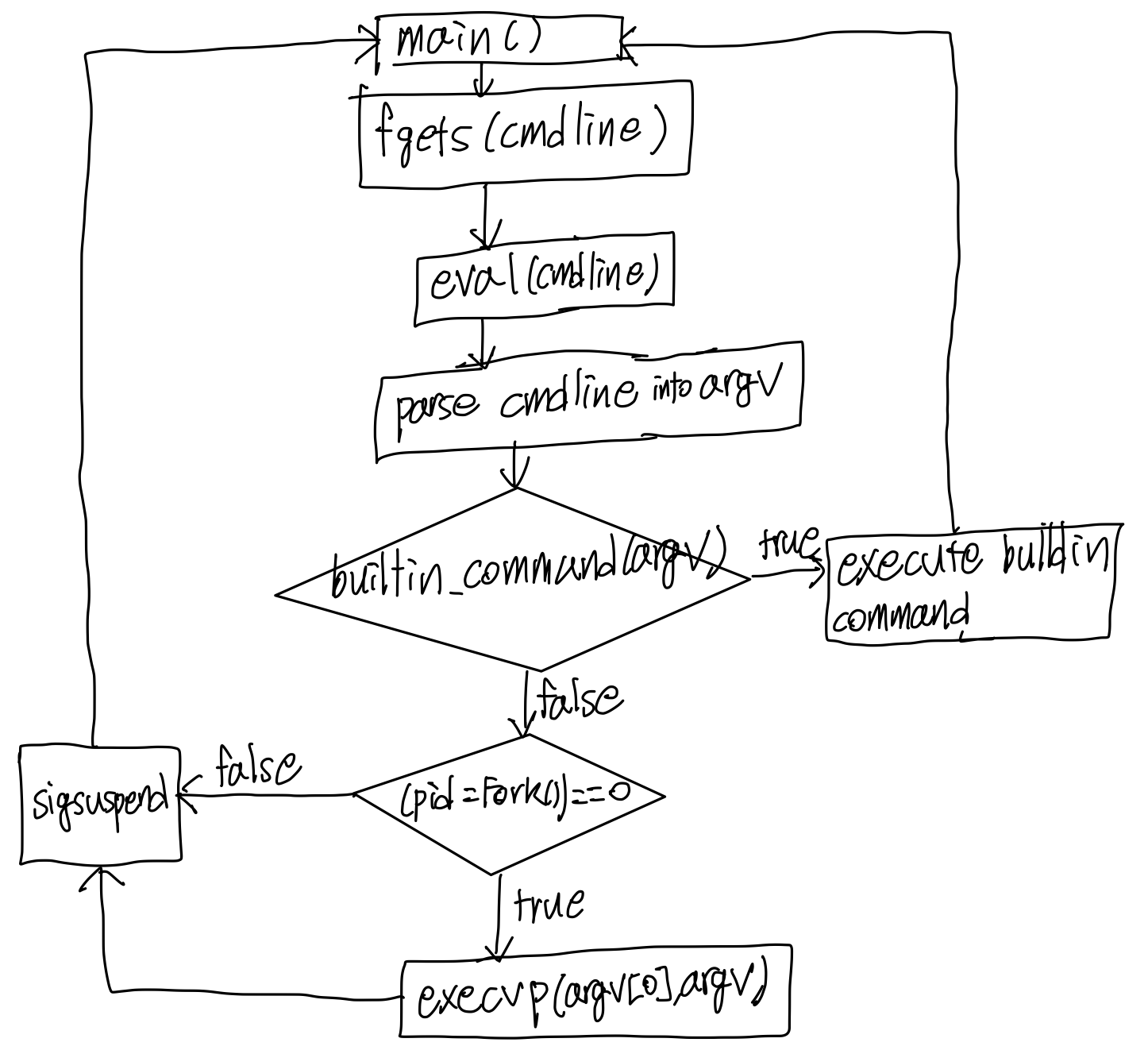
void parsepipe(char\*\* argv,char\* first\_pipe\_command,char\* after\_pipe\_commands) : 파이프라인이 존재하는 경우, argv를 통해 명령어를 파싱한다.

int parseline(char \*buf, char \*\*argv) 수정내용 : quotation이 존재할때 경우를 stack 자료구조를 통해 해결할 수 있도록 수정.

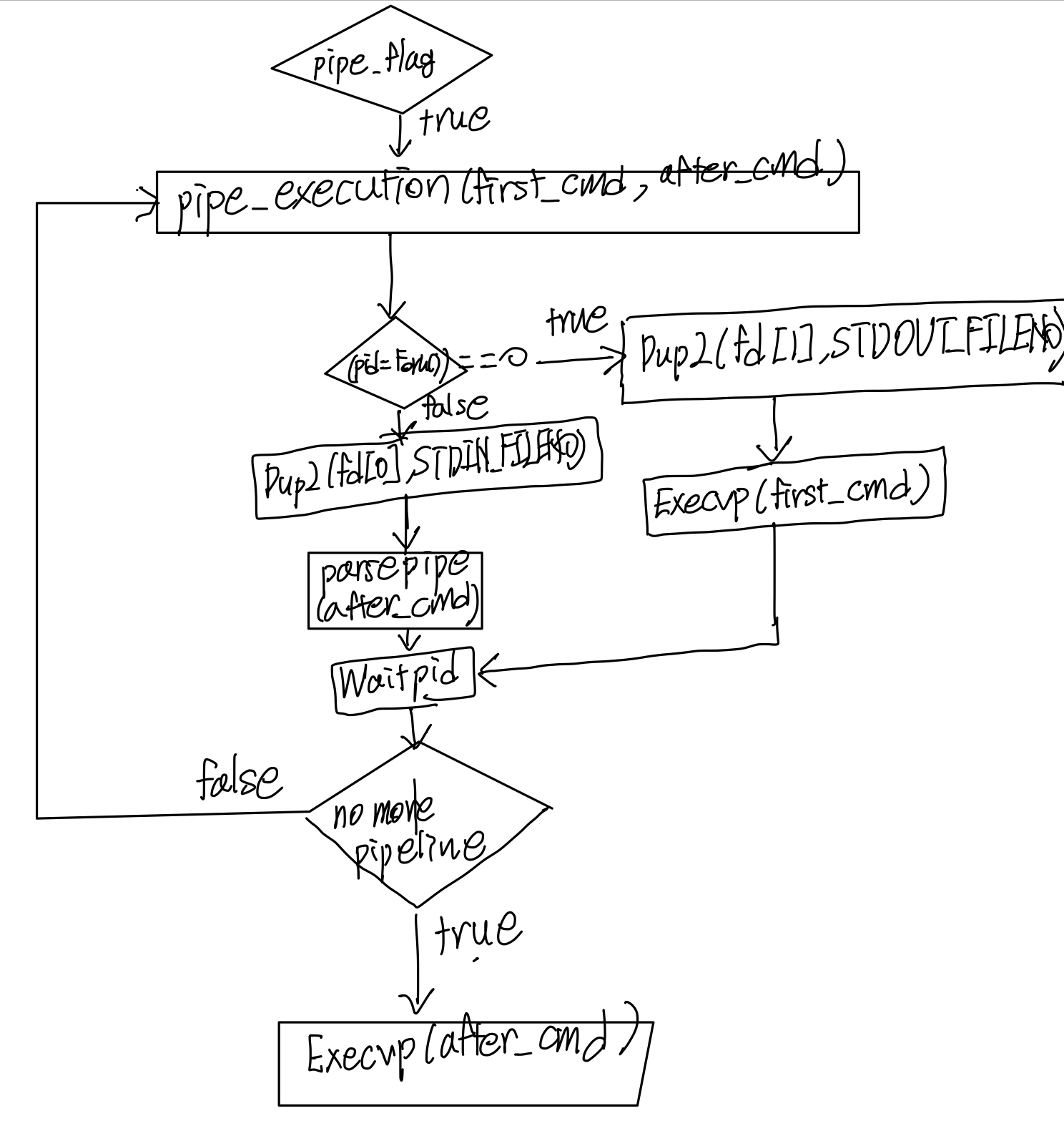
1. **구현 결과**
   1. **Flow Chart**

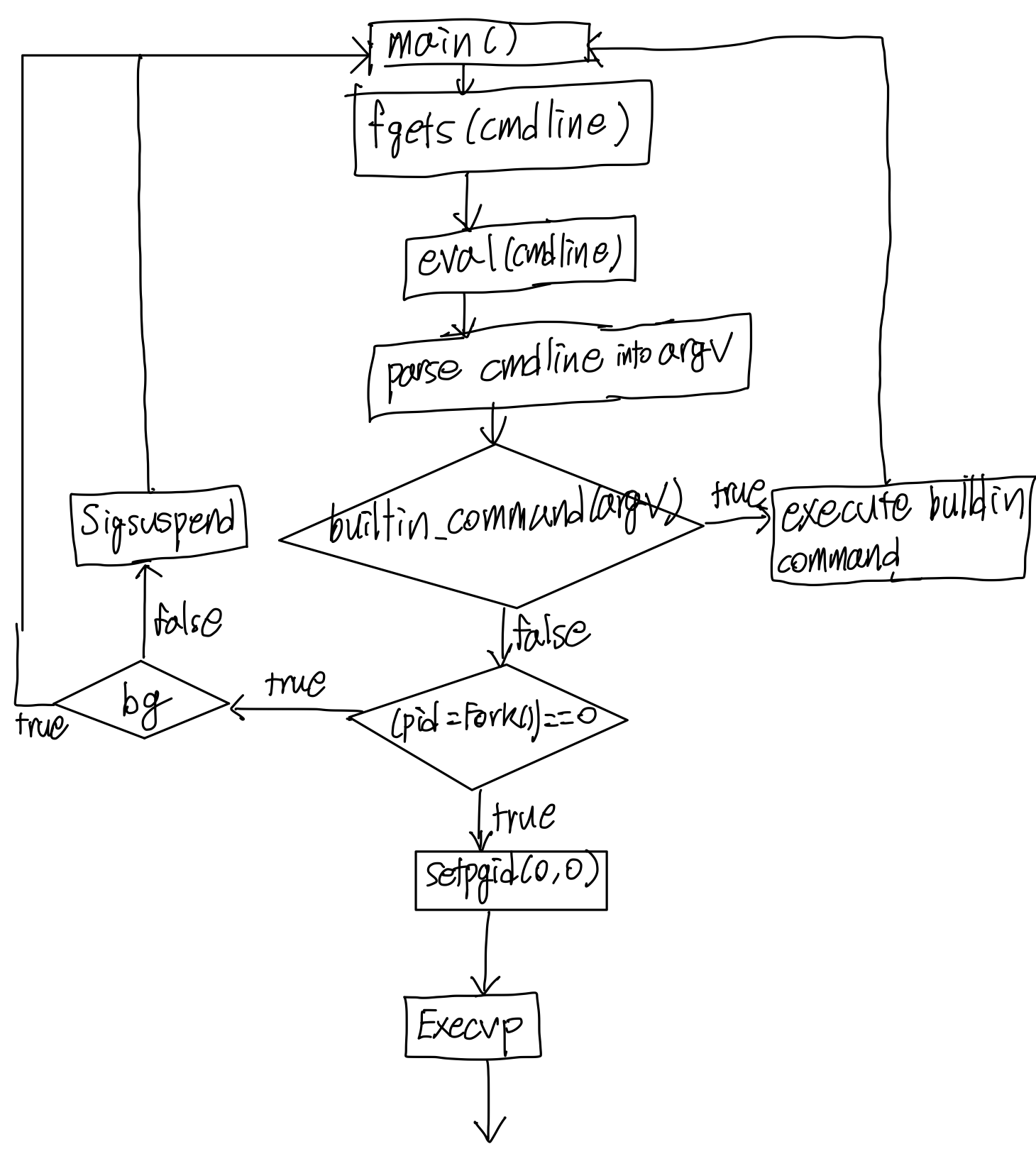
* **2.B.개발 내용에 대한 Flow Chart를 작성.**
* **(각각의 방법들에서 추가된 내용(fork, pipeline, background)만 특성이 잘 드러나게 그리면 됨.)**

1. **Phase 1 (fork)**

****

1. **Phase 2 (pipeline)**

****

1. **Phase 3 (background)**