**System Programming Project 1**

담당 교수 : 김영재

이름 : 송재현

학번 : 20190554

1. **개발 목표**

본 프로젝트에서는 Linux shell을 구현하게 된다. Shell에서의 command들을 fork를 통해 생성한 child process로 실행시키고 pipe를 이용하여 다른 process의 output을 자신의 input으로 사용하는 command까지 실행시킬 수 있다. 마지막으로 signal을 이용하여 background에서 command를 실행시킬 수 있게 된다.

1. **개발 범위 및 내용**

**A. 개발 범위**

1. Phase 1

cd, ls, mkdir 등과 같은 기본적인 shell command를 실행시킬 수 있다. 또한 ctrl + c 를 통해 현재 foreground에서 실행되고 있는 process를 종료시킬 수 있다.

1. Phase 2

phase1에서의 기본적인 shell command와 더불어 pipeline을 이용해 process간 communication이 가능하다. ls -al | grep abc에서와 같이 ls -al의 결과를 바탕으로 grep을 실행시킬 수 있게된다.

1. Phase 3

phase2까지의 command들을 background로 실행시킬 수 있게된다. 또한 그와 더불어 fg, bg, jobs, kill이라는 builtin command를 추가로 구현하여 현재 running, stopped 상태인 background job을 확인하는 등 job과 관련된 command를 실행시킬 수 있다.

**B. 개발 내용**

* **Phase1 (fork & signal)**

fork를 통해서 child process를 생성하면 child process는 parent process를 duplicate하지만 separate한 address space를 갖게된다. 이후 execve를 통해 child process는 parseline함수에서 나눠진 cmdline을 바탕으로 명령어를 실행시켜 현재 process(child)에서 load한 후 run하게 된다. 이후 child가 프로그램을 run한 이후에 parent process는 wait을 통해 child가 terminate될 때까지 suspend하게 한다. 이후 child가 terminate되면 parent도 signal을 받게되어 child가 zombie가 되는것을 막는다.

* **Phase2 (pipelining)**

cmdline을 pipeline을 기준으로 나눠서 n개의 command가 입력되었다고 가정할 때 각각의 command를 phase1에서와 같이 실행한다. 이 때 pipe를 통해서 input과 output이 달라진다는 점만 다르다. pipeline의 개수에 따른 구현 방법은 우선 i번째 command를 다루고 있다고 가정하고 설명하도록 한다. Phase2에선 pipeline이 한 개만 있는 경우를 다루는 것이 아니기 때문에 fd를 command가 들어올 수 있을만큼 공간을 할당해준다. 우선 첫 번째 명령어의 경우 pipeline이 하나 있을때의 parent와 마찬가지로 pipe[i][0]을 닫는다. 이후 pipe[i][1]을 stdout의 출력 pipe를 지정해주고 pipe[i][1]을 닫은 후 명령어를 실행시킨다. 마지막 명령어의 경우 pipeline이 하나 있을때의 child와 마찬가지로 pipefd[i-1][1]을 닫고 output에 따로 pipe를 연결해줄 필요가 없기 때문에 pipe[i-1][0]을 stdin과 할당해주고 닫은 후명령어를 실행시킨다. 첫 번째 명령어나 마지막 명령어가 아닐 경우 pipefd[i-1][0]을 stdin에 pipefd[i][1]을 stdout에 할당하여 입력과 출력 pipe를 모두 지정해준다.

* **Phase3 (background process)**

phase 2에 job에 관한 처리와 백그라운드 실행이 추가되었다. ‘&’ 기호를 command 뒤에 붙이면 command를 백그라운드에서 실행한다. 또한 이와 같은 명령어들을 수행할 수 있다.

- jobs : running, stopped background job을 출력한다.

- bg job : 중지되어 있는 background job을 running 상태로 바꾼다. job은 job id로 들어올 수도 있고 process id로 들어올 수도 있다.

- fg job : 중지되어 있거나 실행중인 background job을 running foreground job으로 바꾼다.id는 위와 마찬가지로 job은 job id로 들어올 수도 있고 process id로 들어올 수도 있다.

- kill job : job을 종료시킨다.

키보드로 ctrl-c를 누르면 SIGITN 시그널이 Foreground job에 있는 각 프로세스에 전달이 되고 SIGINT를 받은 프로세스는 종료된다. ctrl-z를 누르면 SIGTSTP 시그널이 foreground job에 있는 각 프로세스에 전달이 되고 SIGTSTP을 받은 프로세스는 중지 상태가 된다. 중지된 프로세스는 SIGCONT 시그널을 받으면 다시 실행이 된다.

**C. 개발 방법**

phase1

eval함수에서 child일 경우 Execve를 통해 프로그램을 run시킨다. cd의 경우 builtin command이기 때문에 builtin command 함수에 chdir을 이용하여 따로 작성한다. parent일 경우 background가 아닐 경우 wait을 해서 child가 zombie가 되는 것을 막는다. 또한 따옴표가 들어오는 것을 생각하여 parseline에서 따옴표안에서의 공백을 따로 처리해주는 코드를 작성하고 Ctrl + C를 통해 현재 실행중인 프로세스를 종료시키기 위해 sigint\_handler를 사용하여 이를 main함수에 install한다.

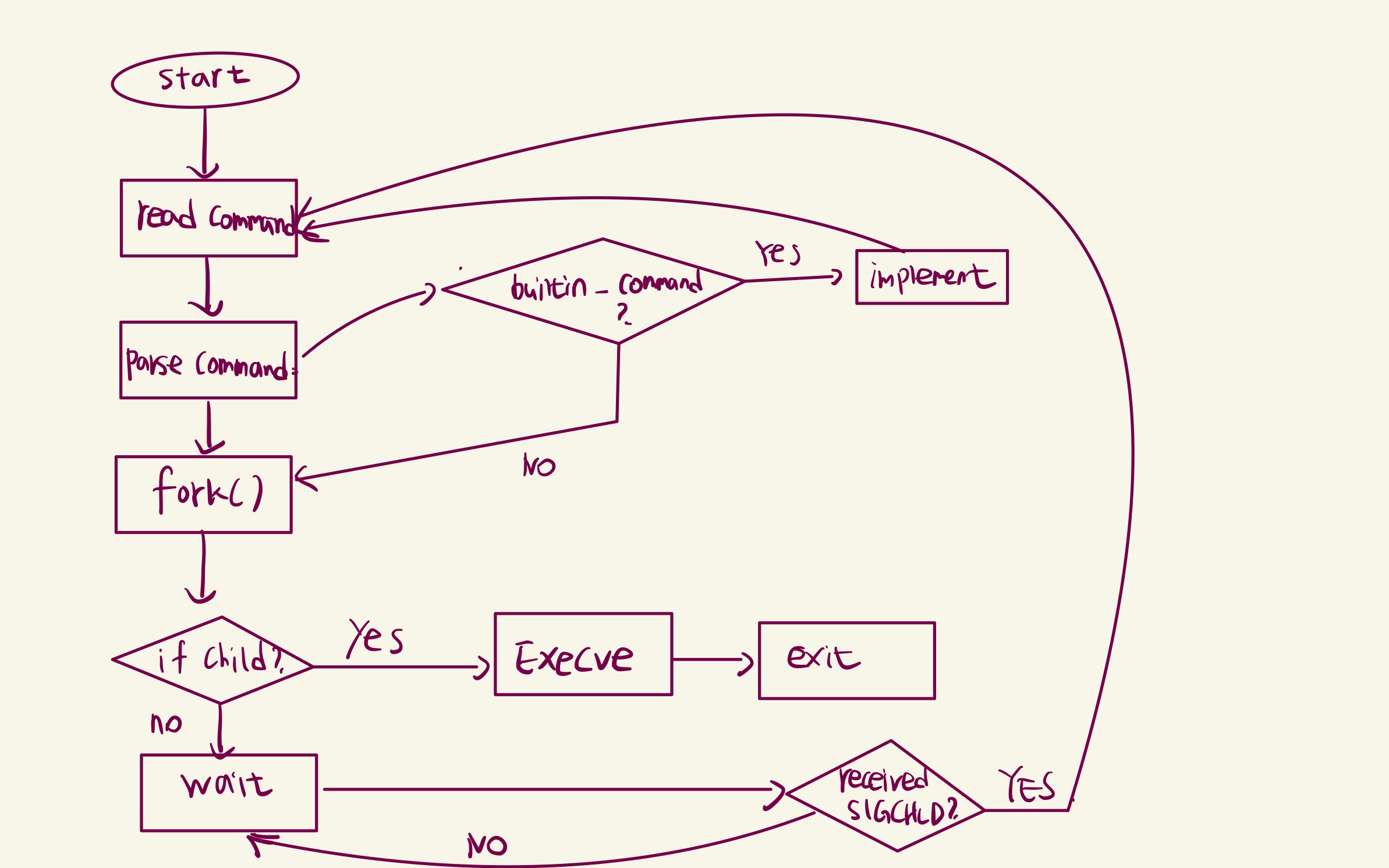
phase2

eval함수에서 pipeline이 있을 경우 pipeline을 토대로 command를 구분하고 child에서 Execve할 때 pipe가 없을 경우 그대로 Execve를 하고 있을 경우 pipeline을 토대로 command를 구분한 pipe\_cmd\_list를 인자로 갖는 pipe\_execve 함수를 호출한다. 이후 pipe\_execve의 구현은 B에서 작성한대로 구현하게 된다.

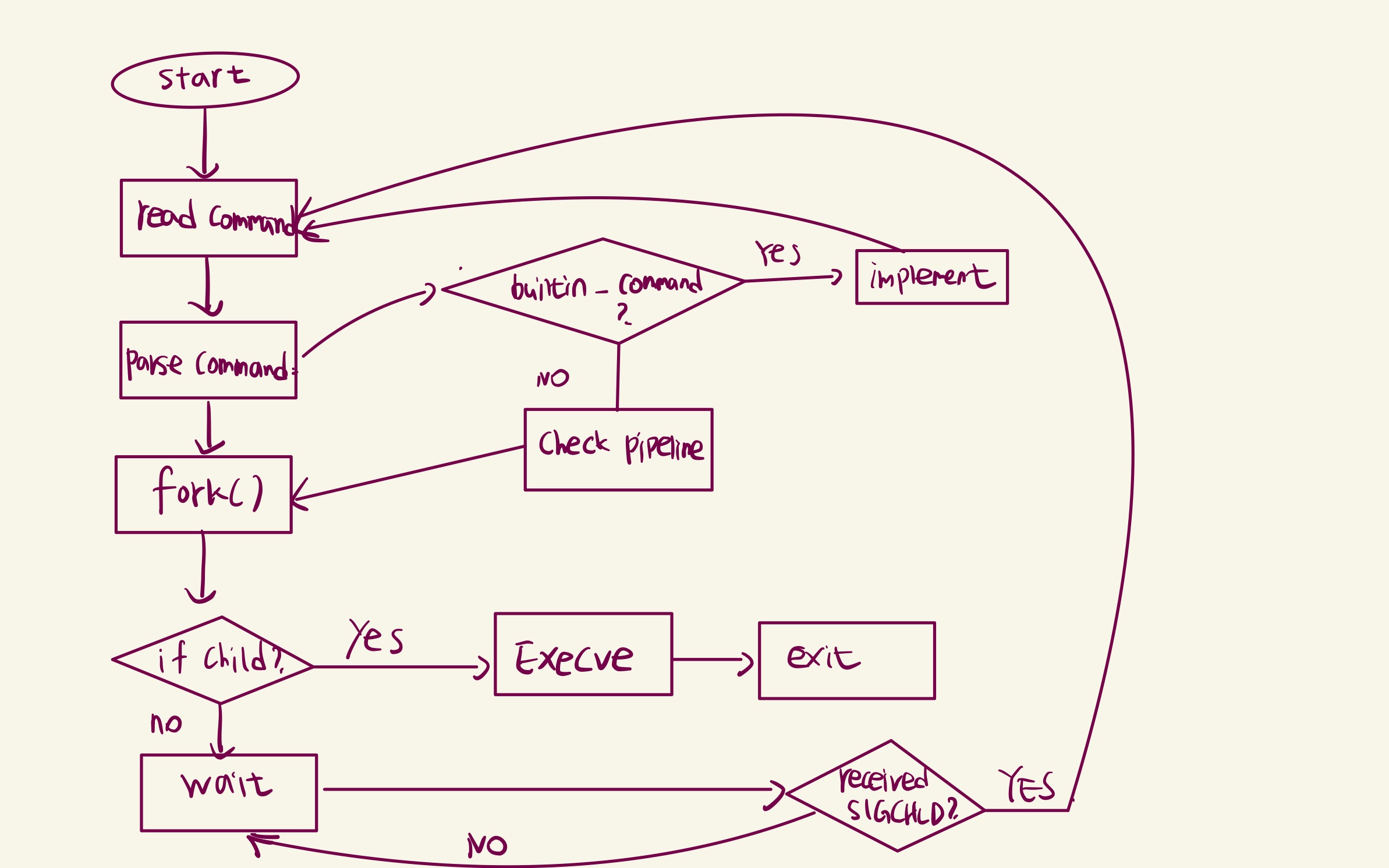
phase3

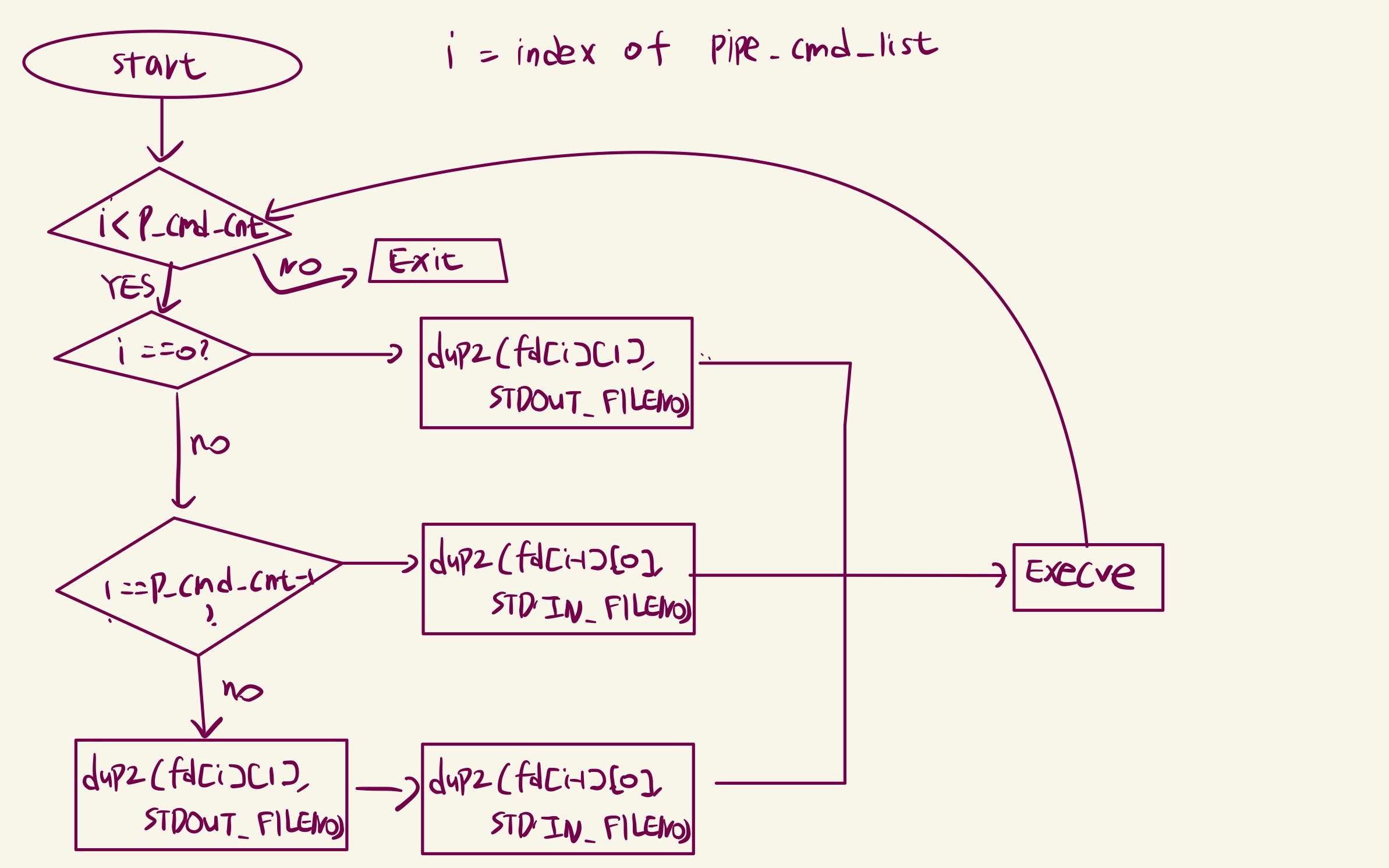
프로세스가 실행되는 동안 background 기능을 통해서 다른 프로세스가 실행 가능하도록 한다. background 방식으로 명령어를 실행하면 작업이 끝나기 전에 다음 명령어를 실행할 수 있다. 자식 프로세스에서는 background에서도 명령에 알맞은 작업을 실행한다. 부모 프로세스에서는 foreground에서는 자식 프로세스가 끝날 때까지 부모 프로세스가 기다리고 background 프로세스의 경우에는 자식 프로세스를 기다리지 않는다. 자식 프로세스의 process id, job id와 command에 대한 정보를 출력한 다음 바로 다음 명령을 받는다. Signal() 함수를 사용하여 background 프로세스의 종료시 시그널을 받아 종료된 프로세스를 회수할 수 있도록 구현한다. 시그널을 보낼 때 부모의 메인 루틴과 시그널 핸들링 흐름들의 일부 중첩, race를 해결하기 위해 sigprocmask로 작업을 더하기 전에 지우는 경우가 발생하지 않도록 했다. 자식 프로세스가 종료되었을 때 SIGCHLD 이벤트가 발생하고 sigchld\_handler가 호출된다. 핸들러 함수에는 waitpid 함수를 사용해여 종료된 프로세스를 회수하고 해당 job을 삭제한다.

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

****

1. **Phase 2 (pipeline)**

****

**pipe\_execve flow chart**