**System Programming Project 1**

담당 교수 : 김영재 교수님

이름 : 윤영인

학번 : 20192135

1. **개발 목표**

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

Foreground와 Background에서 쉘의 기본적인 command와 파이프 라인을 통한 command를 수행하도록 process를 개발한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Phase 1

쉘의 기본적인 command인 cd, ls, mkdir, touch, cat, exit등을 수행하도록 구현하였다.

1. Phase 2

‘|’를 사이에 두고 입력 받은 2개 이상의 command를 pipeline을 통해 한 프로세스에서 다른 프로세스로 데이터를 전달하도록 하였다. 전송된 데이터를 통해 수행한 최종 명령어의 결과를 출력한다.

1. Phase 3

Phase1과 phase2에서 구현한 명령어들을 foreground와 background에서 수행하도록 구현하였다. Background에서 수행한 명령어들은 jobs 명령어를 통해 확인할 수 있도록 구현하였다. 하지만 bg, fg, kill의 명령어 구현은 실패하였다.

* 1. **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

쉘에서 fork()함수를 수행하여 자식프로세스를 생성한다. (pid=Fork())==0인 자식 프로세스에서 execvp() 함수를 이용하여 명령어를 수행한다.

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

자식 프로세스가 명령어를 처리하고 종료하면서 부모 프로세스로 SIGCHLD 시그널을 보낸다. 이때 부모 프로세스는 waitpid 또는 wait 함수를 통해 자식 프로세스의 상태를 확인하면서 자식 프로세스가 종료할 때까지 기다린다.

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

tokenpipe 함수를 통해 입력 받은 명령어를 공백, 엔터, NULL문자를 기준으로 잘라 파이프라인 문자까지 모두 배열 안에 각각 넣어주었다. runpipe 함수에선 위에서 받아온 배열을 돌면서 파이프라인을 발견하면 NULL문자로 변경하였다. 데이터를 보내 명령어들을 순차적으로 수행하기 위해 파이프라인 이후의 명령어들의 시작 지점을 찾아서 cmdstart 배열에 넣어주었다.

파이프 라인이 하나일 때는 fork를 2번 처리하여 자식 프로세스 두개를 생성하고 첫 번째 자식 프로세스에서 첫번째 명령어를 수행하여 이 때의 데이터 값을 연결해준 파이프를 통해 다음 명령어의 파이프에서 read하여 최종 명령어에서 받아온 데이터 값을 이용하여 명령어를 수행한다.

파이프라인이 2개 이상일 경우, fork로 명령어의 개수만큼 자식 프로세스를 생성한다. 파이프라인이 1개일 때와 마찬가지로, 명령어를 수행하여 얻은 데이터 값을 연결해준 파이프로 이동하여 최종 명령어까지 데이터가 전달되면서 수행하도록 하였다.

* + Pipeline 개수에 따라 어떻게 handling했는지에 대한 설명

첫번째 명령어에서 마지막 직전 명령어까지 for문을 돌면서 pipe 함수를 통해 PIPES 배열을 연결한다. Fork()함수로 생성한 자식 프로세스에서 STDOUT\_FILENO를 닫아주고 dup2함수를 통해 파이프에서 쓰는 역할을 하는 PIPES[i][1]과 STDOUT\_FILENO를 연결하고 이 파이프를 닫아준다. 만약 첫번째 명령어일 경우, STDIN\_FILENO를 닫아서 dup2함수를 통해 읽는 역할을 하는 파이프인 PIPES[i][0]과 STDIN\_FILENO를 이어준다. 연결 후 이 파이프를 닫아주고, execvp에서 명령어가 시작하는 지점을 넣어주어 명령어를 수행한다. 부모 프로세스에서 PIPES[i][1]을 닫아준 후 자식 프로세스가 종료하여 시그널을 보낼 때까지 기다린다. 이 과정으로 첫 번째부터 마지막 직전 명령어까지 수행한다.

마지막 명령어일 경우, 다시 fork로 자식 프로세스를 생성하고 이 자식 프로세스에서 STDIN\_FILENO를 닫아준다. dup2 함수를 통해 위에서 마지막 명령어의 직전 명령어를 처리하기 위해 pipe연결을 한 PIPE[pipe\_cnt-1][0]와 STDIN\_FILENO를 이어준 후 이 파이프를 닫고 execvp 함수를 통해 마지막 명령어를 수행한다.

이 때 부모프로세스는 자식 프로세스가 종료하여 시그널을 보낼 때까지 wait 한다.

**Phase3 (background process)**

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

eval함수에서 명령어 안에 strchr 함수를 통해 ‘&’가 있으면 state를 BG로, 없으면 FG로 설정한다. 자식 프로세스에서 명령어를 수행 후 이 명령어들을 모두 job 연결리스트 안에 추가하고, status가 FG인 노드를 찾아 지워준다. 만약 status가 BG일 경우, 연결리스트를 순회하면서 각 노드의 id, pid, 명령어를 출력한다.

이때 백그라운드에선 SIGINT, SIGTSTP, SIGCHLD와 같은 시그널을 무시해줘야 하므로 핸들러 함수를 생성하여 이 시그널들을 kill하고, SIGCHLD 핸들러에선, waitpid함수를 통해 자식 프로세스가 정상 종료했거나 시그널에 의해 종료했을 경우 상태를 DONE으로 바꿔준다. jobs 명령어를 입력 받았을 경우, printjob 함수를 통해 job 연결리스트를 순회하면서 노드 안의 정보들을 출력한다.

* 1. **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**

1. myHandler 함수 추가

SIGINT와 SIGTSTP 시그널을 무시하기 위한 핸들러 함수를 추가하였다.

2. eval 함수에 execvp 함수 추가

주어졌던 execve 함수 대신 execvp 함수로 변경하여 명령들의 경로를 따로 추가하지 않아도 자동적으로 명령어를 수행하도록 처리하였다.

3. if(!strncmp(cmdline, “exit”, 4)) break;

입력 받은 명령어가 exit 일 경우, 함수를 종료하도록 코드를 추가하였다.

4. void make\_cd(char\* cmdline)

Execvp 함수로 처리할 수 없는 명령어인 cd를 수행하기 위해 현재 디렉토리 변경함수인 chdir를 이용하여 cd 명령어를 수행한다.

5. parseline 함수에 “ ”, ‘ ’가 붙여진 명령어를 수행하기 위해 strchr 함수를 이용하여 따옴표를 찾아 NULL문자로 변경한다.

**- Phase 2**

1. char\*\* tokenpipe(char\* cmdline)

파이프라인과 공백, 엔터 키 등을 기준으로 명령어를 잘라서 2차원 포인터 배열에 넣어준다.

2. void runpipe(char\* cmdline)

Tokenpipe 함수를 통하여 명령어를 잘라 2차원 포인터 배열에 넣어주고, 명령어의 시작 지점을 넣어줄 배열과 파이프 배열을 선언한다. ‘|’ 문자를 발견하면 이를 NULL 문자로 바꿔주고 파이프 다음 명령어의 시작 지점 인덱스를 배열에 넣어준다. 첫 번째부터 마지막 직전 명령어는 반복분을 돌면서 fork로 자식 프로세스를 생성하여 이 자식프로세스 안에서 파이프를 연결하고 명령어를 수행하면서 얻은 데이터를 전달한다. 마지막 명령어는 직전 명령어를 수행하면서 얻은 데이터 값을 파이프를 통해 전달 받아서 명령어를 수행한다. 각 명령어에 대한 부모 프로세스는 자식 프로세스가 종료하여 시그널을 보낼 때까지 기다린다.

**- Phase 3**

1. job 구조체, job 연결리스트 추가 및 jobs 명령어 처리를 위한 함수

Jobs 명령어를 실행하려면 background와 foreground의 명령어들을 저장하는 list가 필요하다. Job 구조체를 선언하여 default, bg, fg, done, stop의 상태를 넣어주기 위한 state, job의 id인 id, process id인 pid, 명령어를 넣어줄 cmd배열, 그리고 다음 노드를 가리킬 포인트를 포함하였다. 다음 노드를 가리킬 포인터를 이용하여 명령어 list를 단일 연결 리스트로 생성하였다. 이 리스트를 초기화하기 위한 initJobs(), 해당하는 노드를 지워줄 clear\_Job, 노드를 추가할 add\_job, 노드를 done의 상태로 바꿔줄 done\_job, 그리고 이 리스트의 명령어들을 모두 출력할 print\_job 함수를 추가하였다.

clear\_Job함수는 while문으로 리스트를 모두 돌면서 지우고 싶은 노드의 이전 노드를 찾아 이 노드의 next 포인터가 지우려는 노드의 next를 가리키도록 하였다.

add\_job함수는 새로운 노드를 하나 만들어서 넣고 싶은 pid, state, cmd를 모두 넣어주고 현재 리스트의 마지막 노드의 next 포인터가 가리키도록 연결하였다.

done\_job함수는 입력 받아온 pid를 가지는 노드를 찾아 이 노드의 state를 DONE으로 바꿔준다.

print\_job함수는 리스트의 모든 노드들을 방문하면서 명령어와 상태, pid를 출력한다. ‘jobs’를 입력하였을 때, print\_job 함수가 수행되도록 하기 위해 builtin\_command함수에 입력 받은 명령어가 “jobs”이면, print\_job() 함수가 수행되도록 코드를 추가했다.

2. foreground 처리 수정, background 처리 추가 및 SIGINT, SIGTSTP, SIGCHLD handler

eval 함수에서 strchr 함수를 이용하여 입력 받은 cmdline에 ‘&’문자가 있으면 state를 BG로, 없으면 FG로 설정하였다. Cmdline에 파이프 라인이 있으면 fork 함수로 자식 프로세스를 생성하고, state가 FG일 땐, SIGINT와 SIGTSTP 시그널들을 SIG\_DFL 처리하고, BG 일 땐, SIGINT와 SIGTSTP를 SIG\_IGN하여 무시하였다. 그 후, runpipe 함수에 cmdline을 넣어 명령어를 실행하도록 하였다. job 리스트에 입력 받은 명령어를 넣은 노드를 추가하고, 부모 프로세스에선 state == FG일 경우 waitpid 함수로 자식 프로세스가 종료할 때까지 기다리고 자식 프로세스가 종료하면 연결 리스트에서 state가 FG인 노드를 찾아서 이 노드를 지워준다. 또한 state == BG일 경우, 현재 부모 프로세스와 같은 pid를 갖는 노드를 찾아서 이 노드의 pid, state, cmdline을 출력한다.

파이프 라인이 없는 명령어는 자식 프로세스를 생성하여 state가 FG일 땐, SIGINT와 SIGTSTP 시그널들을 SIG\_DFL 처리하고, BG 일 땐, SIGINT와 SIGTSTP를 SIG\_IGN하여 무시하였다. 그리고 execvp 함수를 이용하여 명령어 수행을 처리하였다. 입력 받은 명령어를 리스트에 추가하고, foreground 명령어일 경우, waitpid 함수로 자식 프로세스가 종료될 때까지 기다리며 종료되면 자식이 정상적으로 종료되었을 경우 리스트에서 state가 FG인 노드를 찾아 지워준다. Background 명령어일 경우, 현재 부모 프로세스와 같은 pid를 갖는 노드를 찾아 프린트 해준다.

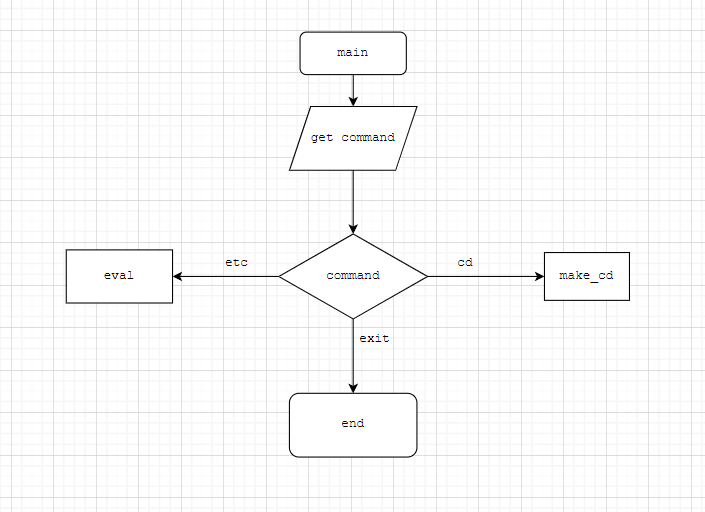
SIGINT 핸들러는 연결리스트를 돌면서 state가 FG인 노드를 찾아 그 노드의 pid를 kill 함수에 넣어 KILL 처리를 하였고, SIGTSTP 핸들러도 마찬가지로 노드를 찾아 SIGTSTP 시그널을 Kill 처리하고 그 노드의 state를 STOP으로 변경한 후, STOPPED 되었다는 문장을 출력하였다. SIGCHLD 핸들러는 waitpid 함수로 자식 프로세스가 정상적으로 종료했거나 시그널을 받아 종료했을 경우 done\_job함수로 해당하는 노드의 상태를 DONE으로 변경하였다.

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)**

main



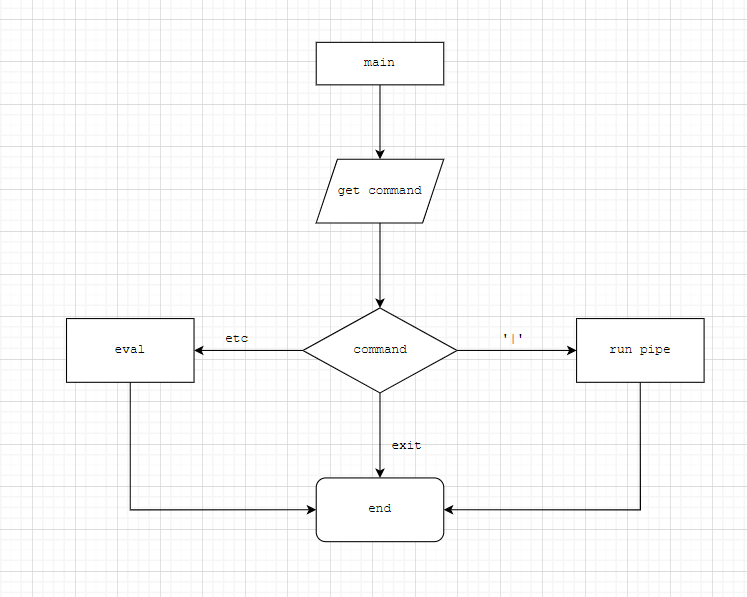
eval

텍스트, 실내이(가) 표시된 사진

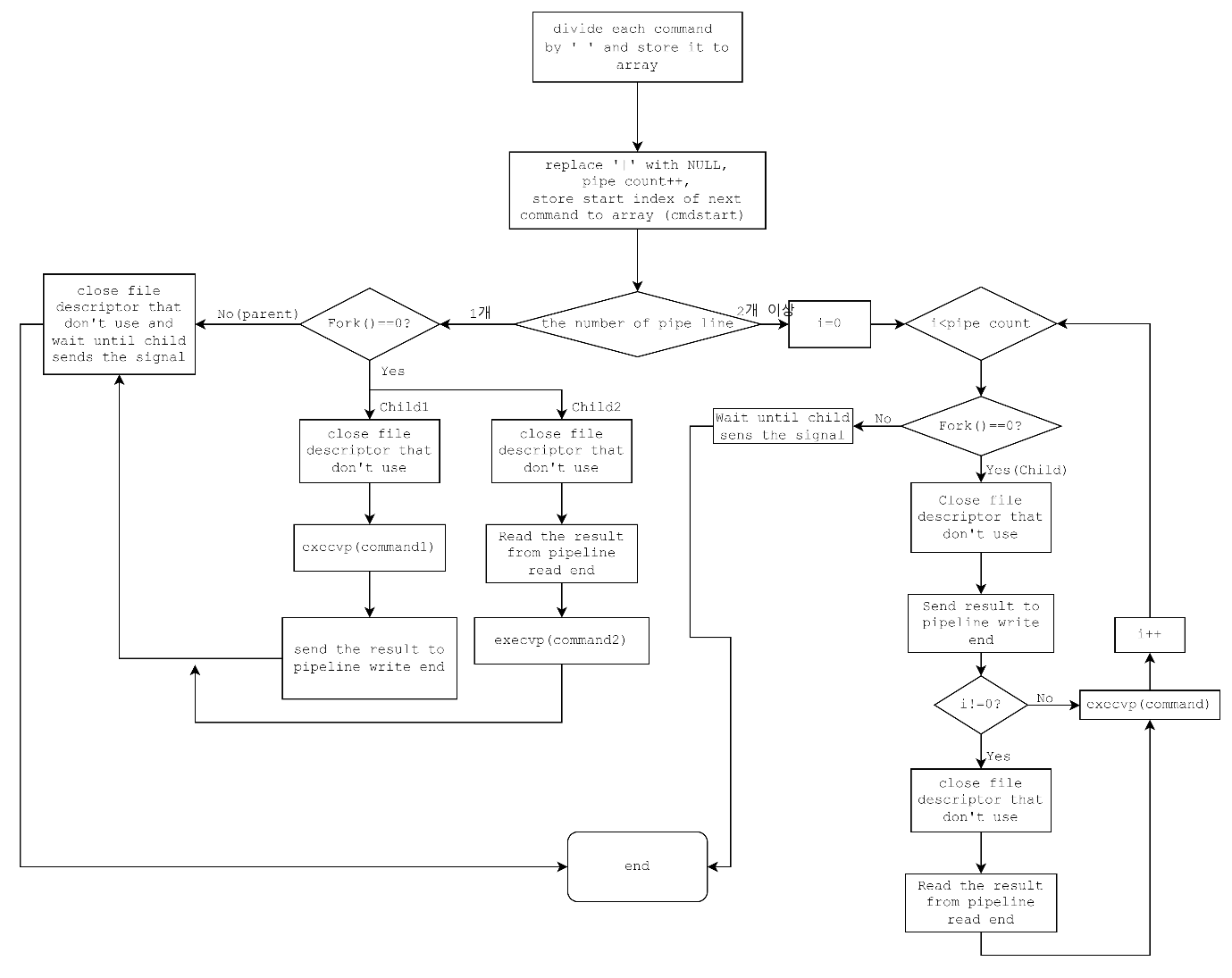
자동 생성된 설명

1. **Phase 2 (pipeline)**

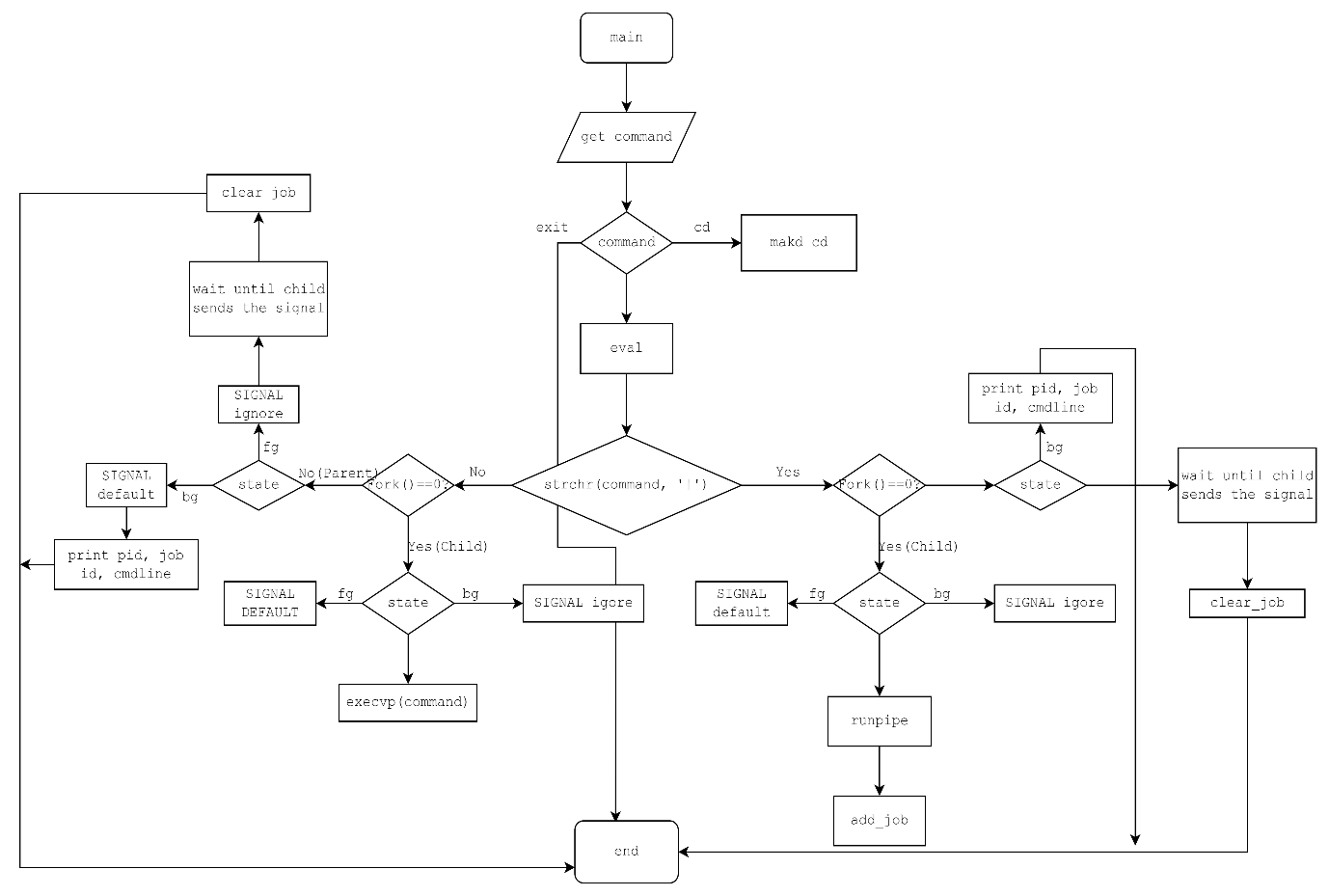
main



runpipe



1. **Phase 3 (background)**

****