**System Programming Project 4**

담당 교수 :김영재 교수님

이름 :김세영

학번 :20191571

1. **개발 목표**

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

fork를 이용하여 Myshell을 만든다. 입력을 받아 built in command가 아닌 경우에는 자식 프로세스를 생성하고, 해당 입력을 실행한 다음 에러가 발생한 경우는 에러메시지를 출력한다. 입력으로 exit을 받으면 프로그램을 종료한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Phase 1

입력을 받아 built in command가 아닌 경우에는 fork 하여 자식프로세스에서 해당 입력을 실행한다. built in command는 fork하지 않고 부모프로세스에서 해당 명령을 실행한다.

1. Phase 2

phase1에서 기능을 추가하여 pipe를 이용하여 pipeline이 사용된 command에서도 작동하도록 한다. fork하여 각각의 command를 자식 프로세스에서 실행하고, 이때 앞쪽 command의 실행결과가 뒤쪽 command의 input이 될 수 있도록 pipe로 연결한다.

1. Phase 3

Phase2에서 기능을 추가하여 background(&) command에서도 동작할 수 있도록 한다. jobs,bg,fg,kill 명령어를 수행할 수 있도록 하고, ctrl c를 누르면 foreground job이 종료되고, ctrl z를 누르면 foreground job이 정지되도록 한다.

* 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
* 입력을 받으면 built in command 인지 확인한다. built in command 가 아니면 fork를 하고, 자식프로세스라면 pid가 0이고, 부모 프로세스라면 0이 아니기 때문에 if문을 이용한다. 자식프로세스 일 때 argv 중에 ‘’나 “”를 사용한 것이 있다면 제거하고, 실행한다. 부모 프로세스라면 자식프로세스가 종료될 때 까지 기다린 후, reap 한다.
* **Phase2 (pipelining)**
  + Pipeline( ‘|’ )을 구현한 부분에 대해서 간략히 설명 (design & implementation)
* command에 pipeline이 있는지 확인한다. pipeline이 있다면 해당 | 전까지의 command를 child\_argv에 저장한 후, 파이프를 생성한다. built in command가 아니라면 fork를 하고 자식프로세스라면 input과 output을 연결한 후 command를 실행한다.
  + Pipeline 개수에 따라 어떻게 handling했는지에 대한 설명

pipeline 개수와 상관없이 실행이 이루어질 수 있도록 recursion을 이용하였다. 기존 command의 끝이 child\_argv의 끝이라면( command를 다 수행한 경우거나 pipeline이 없는 경우) 다음 함수 호출에서 return하여 recursion을 종료하고, 끝이 아닌 경우는 다음 함수 호출에서 다음 pipeline전까지의 command를 수행한다.

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

pipeline 전에 command에 &가 있는지 확인한다. &가 있는 경우에는 부모 프로세스에서 pid와 cmdline(command)를 출력한 후 background process를 저장하는 구조체에 pid와 cmdline, flag(running)을 저장한다. 실행이 끝나면 sigchld 시그널을 보내고, 핸들러에서 reaping한다.

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

fork전에 sigchld를 blocking 하고, 자식프로세스의 경우 command를 확인한 후 sigchld blocking을 해제한다. 부모프로세스는 waitpid()실행 후, bg면 (background &가 있는 경우) command출력전에 blocking을 해제한다.

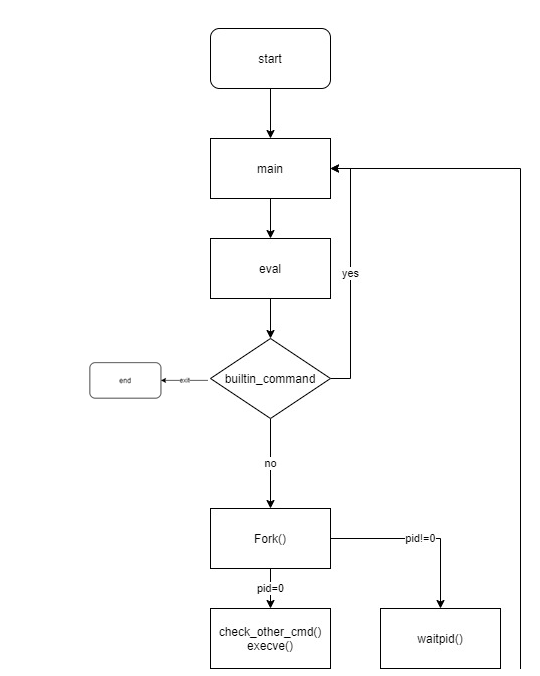
sigchld handler에서는 현재 background에서 flag가 running 인 프로세스가 종료했는지 확인하고, 종료했으면 reaping 한다. 이외에도 reaping해야 할 프로세스가 있는지 확인한 후 reaping한다.

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

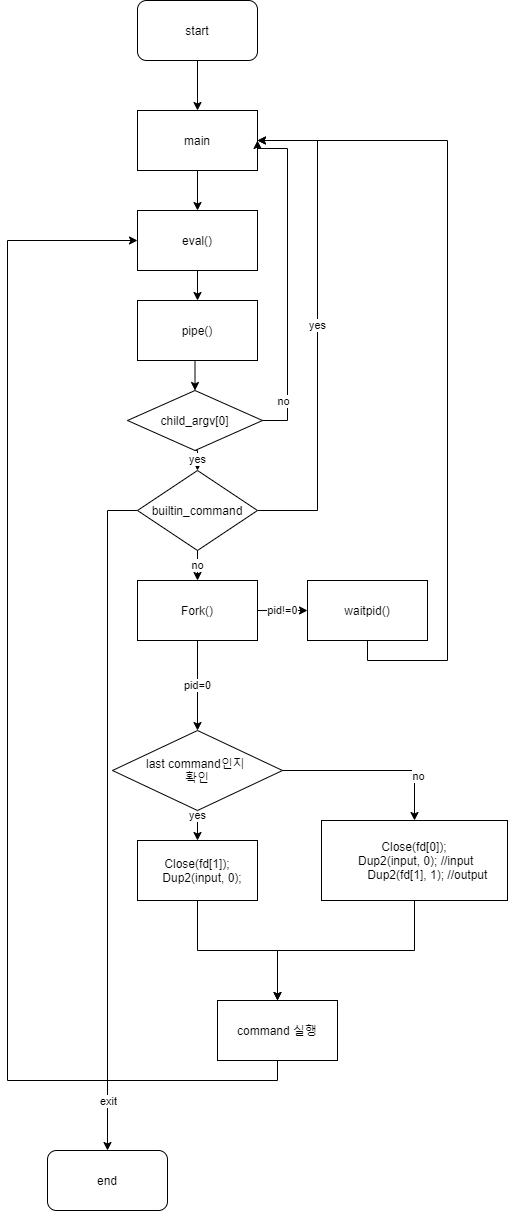
1. cd를 builtin command에 추가한다. built in 이 아닌경우 check\_other\_cmd()에서 exit는 sigterm을 보내 부모프로세스를 종료하고, /bin으로 시작하지 않으면 argv[0]의 앞쪽에 /bin/을 추가한다. 기존의 sigterm은 메시지를 출력후 종료하기 때문에 별도의 sigterm handler를 만들어 메시지를 출력하지 않고 종료하도록 한다.
2. pipeline으로 나눠진 argv를 구하기 위해 get\_child\_argv()라는 함수를 만들어 child\_argv에 각각의 argv를 저장한다. eval()에서 fork하는 부분을 분리하여 pipeline() 함수를 만든다. fd[]를 사용하여 파이프를 구현한다. pipeline은 recursion을 사용한다. ‘sort’는 /bin에 없기 때문에 별도로 sort()를 구현한다.
3. background process를 저장할 구조체 bg\_p를 추가한다. 새로운 background process를 추가하는 insert\_bg(), 삭제하는 delete\_bg, flag를 바꾸는 change\_status(), 출력하는 print\_bg()를 구현한다. sigchld handler에서는 waitpid에서의 status를 WHOHANG으로 바꾸고 종료된 process에서만 delete\_bg()를 실행한다. 함수 Job()을 추가하여 jobs, bg, fg, kill의 명령어를 수행한다. 해당 명령어들은 built in command이므로 부모 프로세스에서 수행한다.
4. **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.)**

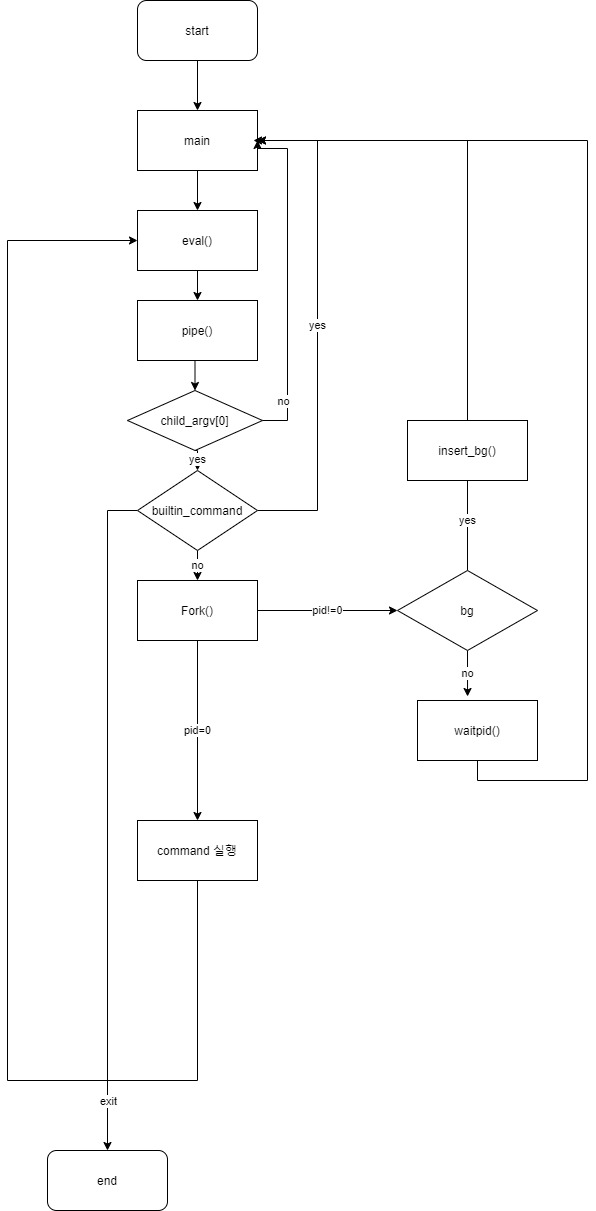
1. **Phase 1 (fork)**

****

1. **Phase 2 (pipeline)**

****

1. **Phase 3 (background)**

****