**System Programming Project 4**

담당 교수 : 박운상 교수님

이름 : 배성현

학번 : 20161595

1. **개발 목표**

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

해당 프로젝트를 통해 사용자로부터 명령을 받고 이를 해석하여 실행하여 주는 간단한 shell프로그램인 MyShell프로그램을 작성한다. 이를 위하여 Fork를 통한 Process의 실행, Pipelining, Signal Handling, Background Process의 실행 등을 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Phase 1

Shell Program에서는 부모 프로세스가 사용자의 명령어를 해석하고 그 결과에 따라 자식 프로세스를 만들도록 하고 또 자식 프로세스에서 해당 프로그램이 실행되게 하여 cd, cd.., ls, mkdir, rmdir, touch, cat, echo등의 간단한 Shell Command가 수행될 수 있게 된다.

1. Phase 2

Shell Program에서는 부모 프로세스가 파이프 라인의 각 명령에 대하여 pipe와 새로운 자식 프로세스를 만들고 pipe를 통하여 파이프 앞쪽의 명령을 담당하는 자식 프로세스의 출력을 파이프 뒤쪽의 명령을 담당하는 자식프로세스의 입력으로 들어가도록 연결시켜 줌으로써 리눅스의 파이프라인 구조를 수행할 수 있게 된다.

1. Phase 3

Shell Program에서 background에서 수행되고 종료된 프로세스를 끝났을 때 따로 Reaping할 수 있도록 하여 메모리 누수를 방지하고 이를 통하여 Foreground뿐만 아니라 백그라운드에서도 프로세스를 수행할 수 있게 된다. 또한 jobs, bg, fg, kill명령어를 사용하여 process의 background와 foreground 작업 사이에서의 이동을 제어할 수 있으며 RUNNING, STOPPED, TERMINATED등의 상태로 프로세스의 상태를 변경할 수 있게 된다.

* 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

사용자로부터 받은 명령(cmd line)을 argument단위로 parsing하게 되면 argv[0]에 실행되어야 할 프로그램의 이름이 들어가게 된다. 이후 fork 명령을 통해서 Child Process를 생성하게 되는데 부모 프로세스의 경우 fork의 반환 값으로 자식 프로세스의 pid값을 반환 받고 생성된 자식 프로세스는 fork의 반환 값으로 0을 반환 받게 된다. 따라서 반환 값을 이용하여 부모 프로세스와 자식 프로세스를 구분하고, 자식 Process에서는 복사된 address space를 exec를 이용하여 프로그램이 대치하여 실행될 수 있도록 하고 부모 프로세스는 자식 프로세스가 끝날 때까지 기다리도록 하여 reaping할 수 있도록 해준다.

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

connection 종료 시에 child process는 parent process에게 SIGCHLD라는 신호를 보내게 된다. 따라서 parent process에서 처음에 SIGCHLD에 관련된 handler를 등록해 놓으면 parent process로의 context switching이 일어났을 때 SIGCHLD 신호를 체크하여 해당 handler가 호출되게 된다. 그리고 이 handler에서 waitpid를 통해 종료된 child process를 reaping 해주고 현재 parent process는 suspend 되어있는 상태이기 때문에 이러한 suspend를 풀어줄 수 있도록 하는 변수를 handler에서 set해주게 된다. 이후 flow가 signal handler에서 parent의 user code로 들어오게 되면 parent process는 suspend 상태를 멈추고 다음 명령을 받을 수 있도록 한다.

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

pipe단위로 구별되는 각 명령에 대해서 순차적으로 fork를 통한 자식 프로세스(2)의 생성 및 실행과 해당 자식 프로세스(2)에 대한 wait을 반복하도록 하여 pipe 단위 명령이 순차적으로 진행 될 수 있도록 하였다. 이 때 각 fork 직전마다 pipe를 만들어 주도록 하고 생성된 자식 프로세스(2)에서는 dup2를 통해 출력을 만들어진 pipe로 돌리도록 함으로써 이전 자식 프로세스(2)가 pipe에 쓴 결과를 다음 자식 프로세스(2)가 해당 파이프로의 dup2를 통해 입력으로 받을 수 있도록 하였다.

* + Pipeline 개수에 따라 어떻게 handling했는지에 대한 설명

Pipe의 개수에 따라 handling방법을 달리하지 않고 pipe의 개수에 따른 반복문을 통해 일반화 시킬 수 있도록 하였다. 먼저 부모프로세스에서 전체적으로 pipeline을 통제할 수 있도록 하는 프로세스(1) 하나를 fork를 통해 만들었다. 그리고 해당 프로세스(1)에서 각 pipe단위로 쪼개지는 명령의 개수(반복)만큼 순차적으로 fork를 통해 새로운 프로세스(2)들을 생성하여 각각의 명령을 수행할 수 있도록 하였다. 이 때 생성에 있어 프로세스(2) 하나가 끝날 때까지 부모 프로세스는 다음 새로운 프로세스(2)를 생성하지 않고 wait하도록 하여 이전 프로세스(2)에서 결과를 파이프에 쓰고 난 이후에 다음 프로세스(2)에서 해당 파이프를 통하여 입력으로 받아 사용할 수 있도록 하는 것을 반복하도록 하였다.

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

foreground process에서는 해당 프로세스가 끝날 때까지 부모 프로세스에서 진행하지 않고 wait하도록 하여 reaping하게 해주었다면 background process역시 해당 process가 끝났을 때에 reaping을 해주어야 한다. 따라서 부모 프로세스의 Shell의 명령어를 받기 직전마다 zombie process를 처리해주는 함수를 실행하고 그 안에서 while(waitpid())를 호출하여 주도록 하여 끝난 background 자식 프로세스를 reaping 할 수 있도록 하였다. 또한 background process에는 job id를 부여하고 이를 list(job[pid, job\_id, state, cmdline])로 관리함으로써 foreground와 background로의 전환 및 프로세스의 상태변화를 가능하도록 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)**
* **Phase1 (fork & signal)**

fork를 통해서 child process를 생성하기 위하여 void eval(char \*cmdline)함수에 fork()함수를 추가하여 줄 것이다. 그리고 해당 fork()함수의 return 값을 pid 변수에 저장하도록 하여 해당 pid 변수를 통해 부모 프로세스와 자식 프로세스를 구분하여 줄 것이다. pid가 0인 경우는 자식 프로세스를 의미하기 때문에 Exec 함수를 호출하여 부모로부터 복사한 address space를 대치하여 실행 할 수 있도록 할 것이고 pid>0인 경우는 부모 프로세스를 의미하기 때문에 이 경우에는 waitpid를 호출하도록 하여 자식 프로세스가 끝날 때까지 기다리고 또 자식 프로세스에 대한 메모리 공간을 reaping할 수 있도록 할 것이다. 또한 위에서 언급한 Exec 함수를 구현하여 줄 것인데 주요 shell command 는 ‘/bin’ 또는 ‘/user/bin’에 존재하기 때문에 먼저 ‘/bin/${argv[0]}’을 execve를 통해 실행하도록 하고, 만약 없다면 ‘/user/bin/${argv[0]}’을 execve를 통해 실행하는 방식으로 구현할 것이다.

* **Phase2 (pipelining)**

pipelining을 구현하기 위하여 먼저 eval 함수 안에 pipe가 있는지의 여부를 체크하는 소스코드를 추가할 것이다. 이후 pipe가 있는 경우에는 전체 명령을 pipe를 기준으로 여러 명령들로 쪼개어 저장할 것이고 이를 기반으로 Pipe에 대한 Execution을 담당하는 함수를 추가할 것이다. 이후 해당 함수에서 각 명령들에 대해서 pipe(fd)명령을 통해 pipe를 생성해주고 또 순차적으로 fork를 통한 자식 프로세스(2)의 생성 및 실행과 해당 자식 프로세스(2)에 대한 wait을 반복하도록 하는 소스 코드를 추가하여 줄 것이다. 그리고 각 자식 프로세스(2)에서는 STDOUT과 STDIN을 특정 파이프로 바꾸어주는 dup2 소스 코드를 추가하여 주고 또 이후에 명령에 대한 EXECUTION 소스코드를 추가하여 줄 것이다.

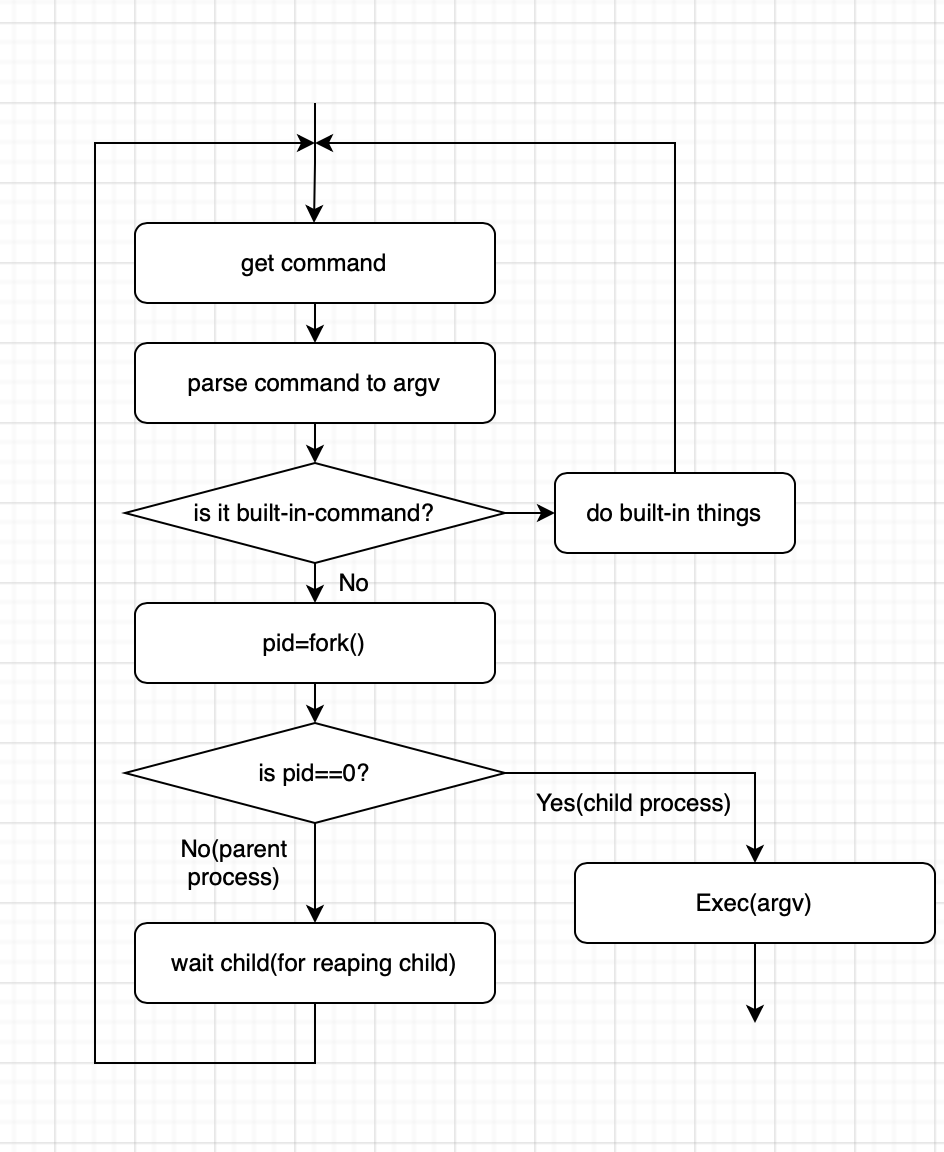
* **Phase3 (background process)**

background process를 관리하기 위해 먼저 process의 정보(pid, job\_id, state, cmdline)를 담고 있는 job\_entry라는 구조체를 만들 것이고 여러 job\_entry들을 list로써 관리할 것이다. 그리고 해당 list를 통해 각 process들의 상태(running, stopped, terminated)의 변경 및 관리를 할 수 있도록 할 것이다. 또한 부모 프로세스의 Shell의 명령어를 받기 직전마다 zombie process를 처리해주는 함수를 실행하고 그 안에서 while(waitpid())를 호출하여 주도록 하여 끝난 process들에 대한 reaping을 수행하여 줄 수 있도록 할 것이고 또 해당 부분에 대한 메시지를 출력할 수 있도록 할 것이다. 마지막으로 signal stop에 대한 handler함수와 bg, kill, fg command에 대한 함수를 추가하여 줄 것이고 해당 함수에서 특정 job에 kill(SIGTSTP, SIGINT, SIGCONT) system call을 보내도록 하여 각각에 해당되는 작업을 수행할 수 있도록 할 것이다.

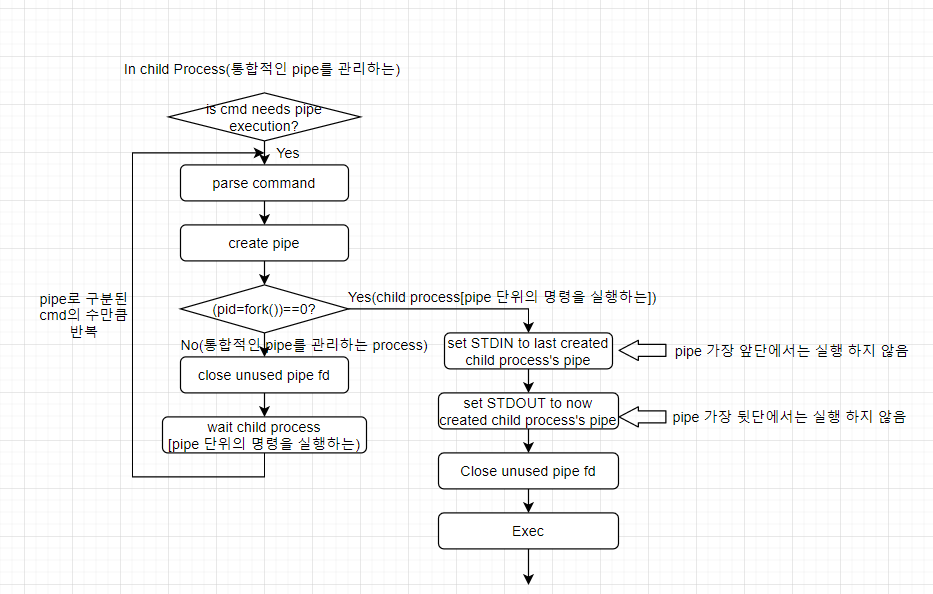
1. **구현 결과**
   1. **Flow Chart**

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

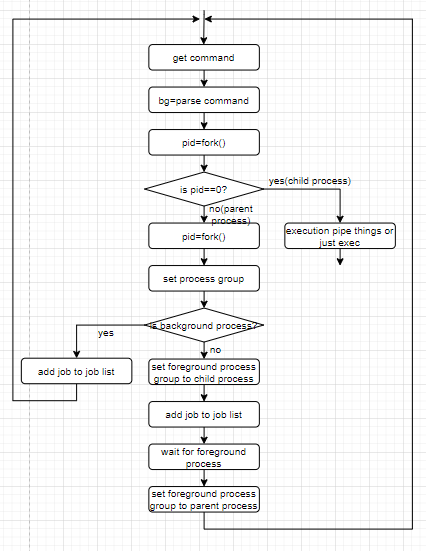
1. **Phase 1 (fork)**

****

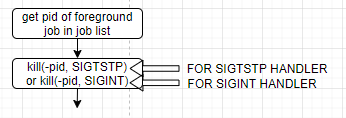
1. **Phase 2 (pipeline)**



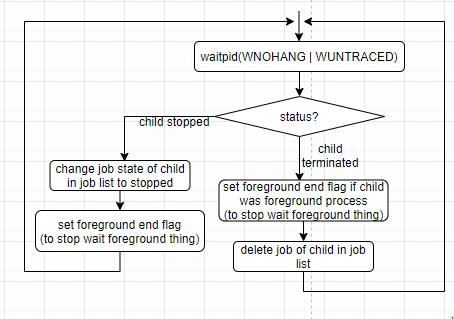
1. **Phase 3 (background)**



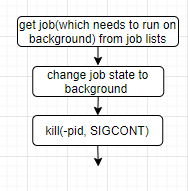
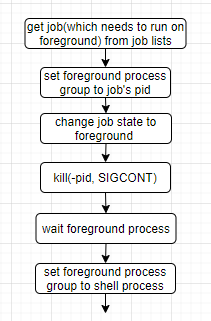
**<전체적인 흐름>**



**<SIGINT HANDLER and SIGTSTP HANDLER in shell>**



**<Reaping Zombie in shell>**



**<fg command and bg command>**