**System Programming Project 1**

담당 교수 : 김영재

이름 : 임종환

학번 : 20171683

1. **개발 목표**

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

시스템 프로그래밍 프로젝트1의 목표는 MyShell을 구현하는 것이다.

**Phase1**: 단일 명령어를 입력 받으면 fork()를 통해 자식 process를 생성하여 입력 받은 명령어를 foreground로 실행한다. 자식 process가 종료되면 부모 process가 자식 process를 reaping한다.

**Phase2**: Phase1에 추가로 다중 명령어가 pipe를 통해 입력되었을 때 정상적으로 실행되게 구현한다.

**Phase3**:

- 명령어를 background process로 입력 받았을 때 정상적으로 실행, 종료되게 구현한다.

- Job list를 구현하여 fg, bg, jobs, kill 명령어를 통한 job control을 가능하게 한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Phase 1

명령어를 foreground로 입력 받으면 자식 process를 생성하여 명령어를 실행한다. 부모 process는 자식 process가 종료될 때까지 대기한다. 자식 process가 종료되면 좀비 process로 남지 않도록 부모 process가 자식 process를 reaping한다.

1. Phase 2

Phase 1에 추가로 pipe 입력을 처리하도록 구현한다. Pipe(|)를 통해 다중 명령어를 입력 받으면, 명령어의 수 만큼 자식 process를 생성하여 명령어를 실행한다. 각 자식 process의 결과는 다음 명령어를 실행하는 자식 process의 입력으로 전달되고, 이러한 과정을 거쳐 최종 명령어에 대한 실행 결과를 shell에 출력한다.

1. Phase 3

Process job list를 구현하여 실행중인 process의 정보들을 저장한다. Process의 pid, 실행 상태(running, done, terminated, suspend), job\_id, 명령어들이 저장된다.

SIGCHID signal handler를 통해 background로 실행한 자식 process가 종료되었을 때 좀비 process로 남지 않도록 정상적으로 reaping한다. 추가로 SIGTSTP, SIGINT signal handler를 통해 foreground로 명령어를 실행 중인 자식 process를 중단, 종료하고 Process job 상태를 변경한다.

fg, bg, kill 명령어를 통해 특정 job\_id값을 갖는 process를 foreground 또는 background로 실행하거나, 종료한다. jobs 명령어를 통해 실행 중이거나 중단된 자식 process들을 출력한다.

* 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

명령어를 입력 받으면 fork함수를 통해 자식 process를 생성한다. fork함수가 자식 process를 정상적으로 생성하면 자식 process에게 0을 return 하고, 부모 process에게 자식 process의 pid값을 return한다. 생성된 자식 process는 execve함수를 통해 입력 받은 명령어를 실행 후 종료된다.

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

부모 process는 waitpid함수를 통해 자식 process가 종료될 때 까지 기다린다. 자식 process가 종료되어 부모 process에게 SIGCHLD signal이 전달되면 부모 process는 자식 process를 reaping 하고 다시 정상적으로 실행된다.

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

Pipe를 통해 다중 명령어가 입력되면, 각 명령어를 실행하는 자식 process를 생성한다. pipelining()함수를 통해 자식 process간의 pipe를 연결한다. 연결된 pipe를 통해 이전 명령어를 실행한 process의 출력이 다음 명령어를 실행할 process에게 입력된다.

* + Pipeline 개수에 따라 어떻게 handling했는지에 대한 설명

N개의 명령어가 입력된다면, (N-1)개의 pipeline을 생성하여 process들을 연결한다. 1번 pipeline을 통해 1번 process의 출력을 2번 process의 입력으로 연결하고, 2번 pipeline을 통해 2번 process의 출력을 3번 process의 입력으로 연결한다. 일련의 과정을 통해 (N-1)번 pipeline을 통해 (N-1)번 process의 출력을 N번 process의 입력으로 연결한다. 마지막 process의 실행 결과는 shell에 출력된다.

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

Foreground process의 경우 부모 process가 자식 process가 종료될 때까지 대기하고, 자식 process가 종료될 경우 해당 process를 reaping 해준다. 하지만 background process의 경우 부모 process가 자식 process가 종료될 때까지 기다리지 않기 때문에 자식 process가 종료되고 나서 reap 되지 않아 좀비 process로 남게 된다.

Background 자식 process를 reaping 해주려면 SIGCHLD signal handler를 작성해야 한다. 부모 process 실행 도중에 background process가 종료되면 SIGCHLD signal이 전송되고, signal handler가 이를 catch하여 종료된 process를 reaping 해준다.

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

**공통**

* myshell\_parseinput(): 명령어 parsing 함수
* myshell\_execute(): 명령어 실행 함수
* builtin\_command(): built-in 명령어 실행 함수
* fork(), execve(), waitpid() 등의 wrapper 함수
* sigint\_handler(): SIGINT signal을 catch하여 처리하는 함수

**Phase2, 3**

* pipelining(), dup2(): pipe를 연결하는 함수

**Phase3**

* JOBS{

pid: process의 pid

state: process의 실행상태 ( RUNNIG, SUSPENDED, DONE, TERMINATED )

job\_id: Job number

command: process 실행 명령어

JOBS\* link: linked list의 링크

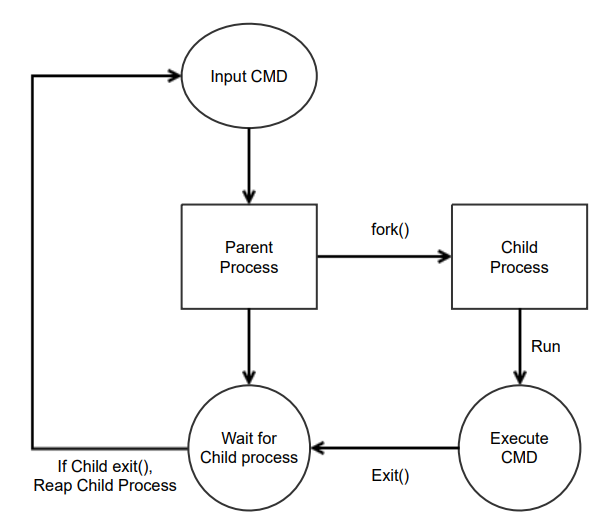
} : process JOB을 저장해 놓은 linked list

* JOBS \*head, \*tail: JOBS list의 처음과 마지막
* sigchld\_handler(): SIGCHLD signal을 catch하여 처리하는 함수
* sigtstp\_hander(): SIGTSTP signal을 catch하여 처리하는 함수
* cmd\_jobs(): JOBS list에 저장되어 있는 process를 실행시키는 함수
* cmd\_fg(%num): JOBS list에서 job\_id가 num인 process를 foreground로 실행
* cmd\_bg(%num): JOBS list에서 job\_id가 num인 process를 background로 실행
* cmd\_kill(%num): JOBS list에서 job\_id가 num인 process를 찾아 종료
* add\_job(): JOBS list에 새로운 process를 추가하는 함수
* delete\_job(process\_id): JOBS list에서 pid가 process\_id인 process를 삭제하는 함수
* print\_end\_bg(): 종료된 background process에 대한 정보를 출력해주는 함수

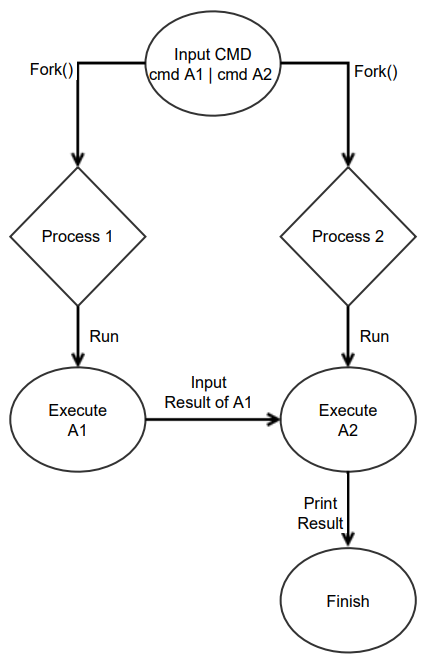
1. **구현 결과**
   1. **Flow Chart**

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

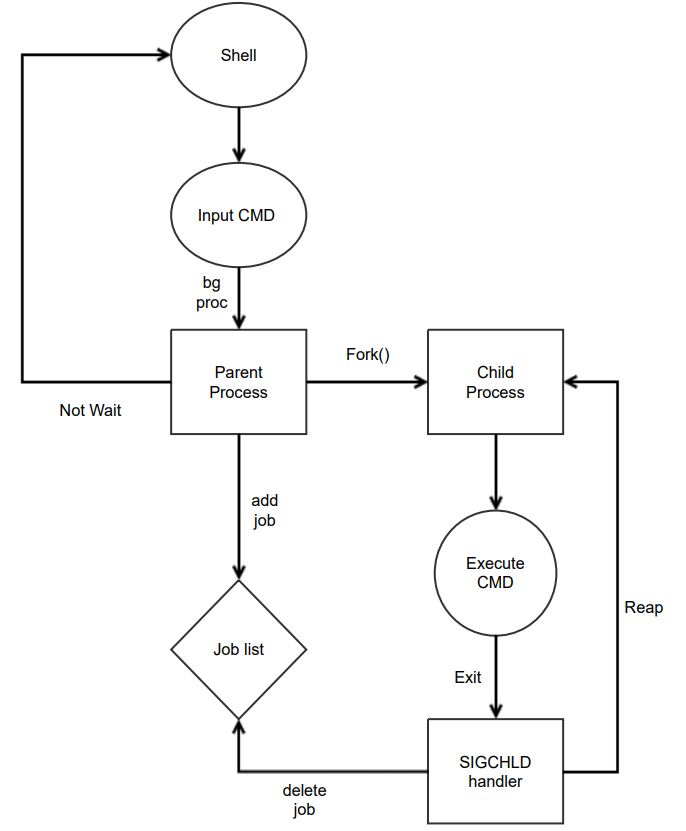
1. **Phase 1 (fork)**

****

1. **Phase 2 (pipeline)**

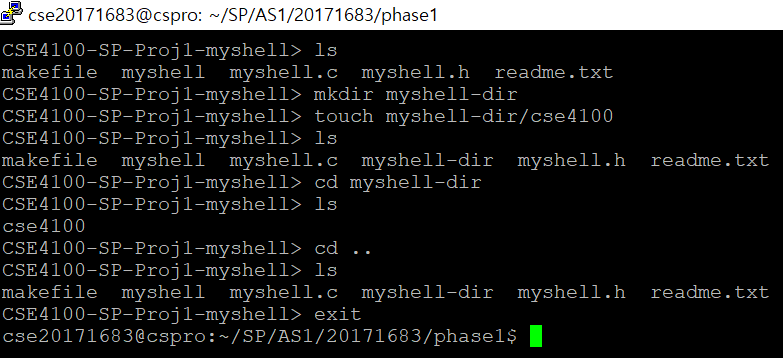
****

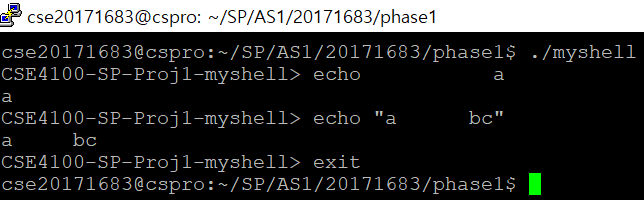
1. **Phase 3 (background)**

****

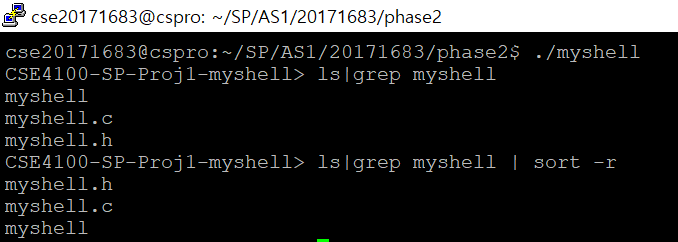
* 1. **실행 화면**

1. **Phase 1**



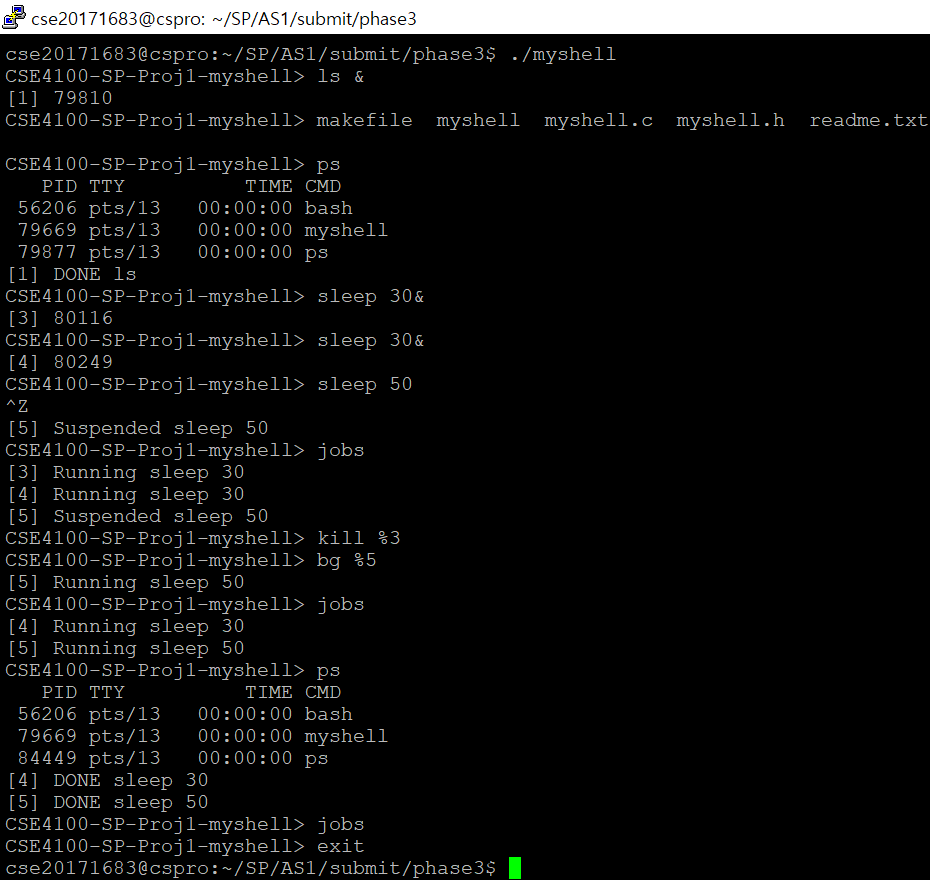


1. **Phase 2**

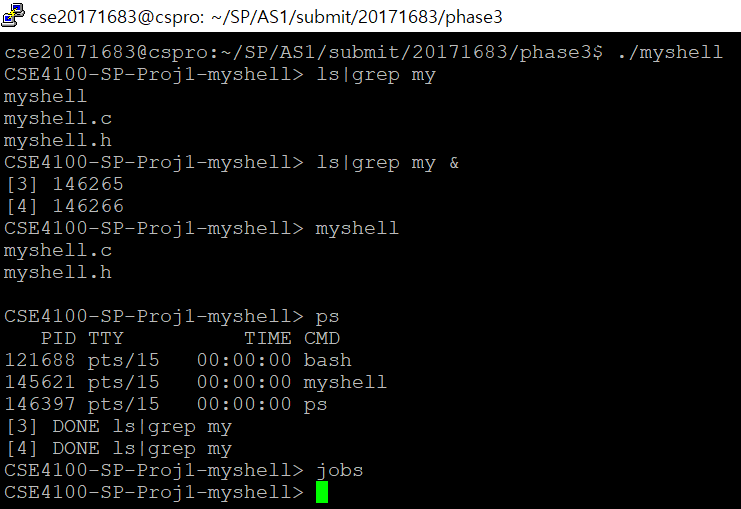


1. **Phase 3**

**Single command (Implementation Success)**



**Multiple command (Implementation Fail)**



pipe를 통해 여러 명령어를 입력 받을 때, JOB list에 1개의 job만 추가하는 경우 signal handling 과정에서 오류가 발생하여 process가 종료되지 않는다. 실행을 위하여 project는 JOB list에 모든 process를 추가하는 방식으로 구현하여 signal handling은 정상적으로 처리했지만 이러한 방식은 process 개수만큼 JOB list에 중복 저장되는 문제가 있다.