**System Programming Project 4**

담당 교수 : 김영재

이름 : 원성현

학번 : 20161614

1. **개발 목표**

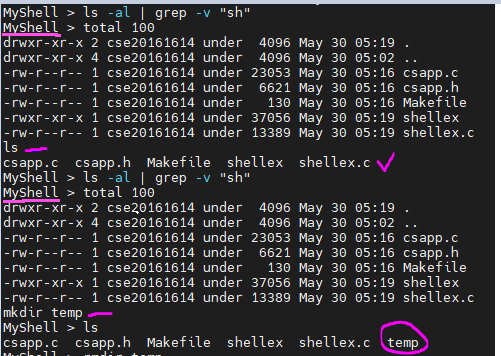
* **해당 프로젝트에서 구현할 내용을 간략히 서술.**
* **(MyShell을 만드는 전체적인 개요에 대해서 작성하면 됨.)**

이번 프로젝트의 목표는 custom shell을 만드는 것이다. 이 쉘은 명세서에 정의된 몇 가지 명령어를 수행할 수 있어야 하는데, 이 명령어는 몇가지 예외를 제외하고 parent process에서 child를 fork하는 방식으로 이루어진다.  
다음으로, pipeline을 구현해본다. 이론적으로 n중 파이프까지 구현하는 것이 목표. 이것은 file descriptor을 이용해 구현된다.  
마지막으로, background process를 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

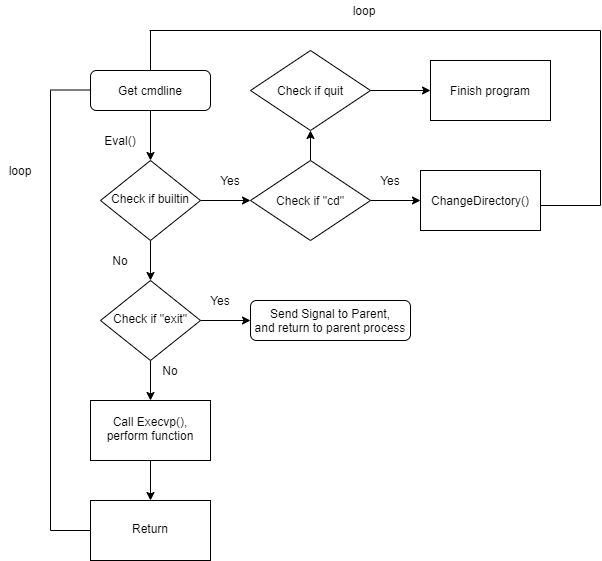
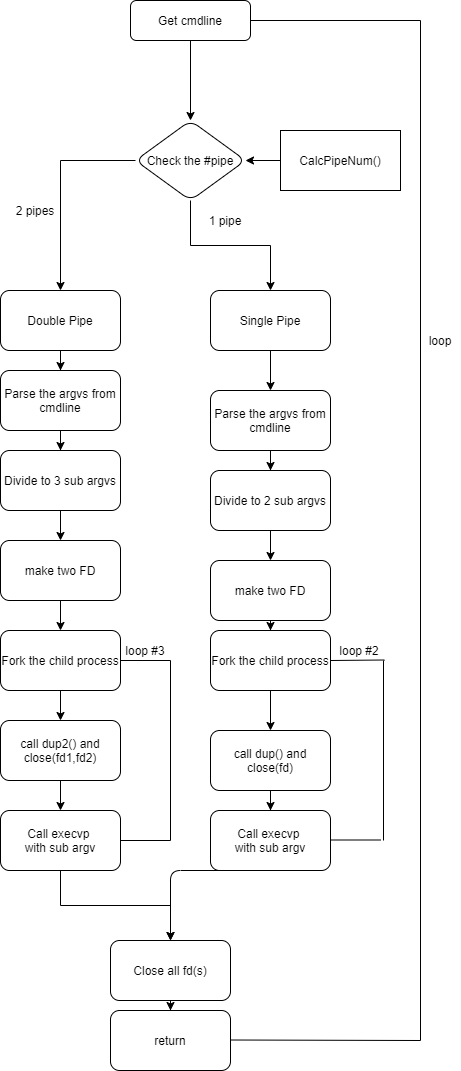
* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Phase 1  
   Phase1을 구현한 결과는 다음과 같다.  
   명세서에서 요구한 몇 가지 명령어들 (ex. ls, mkdir, rmdir, cd, exit …)을 실행할 수 있는 코드를 만든다. Cd를 제외하고, 다른 명령어들은 명세서에서 요구한 대로 parent에서 child를 fork하는 방식을 통해 만들어진다.
2. Phase 2  
   Phase 2를 구현한 결과는 다음과 같다.  
   pipeline을 file descriptor를 이용해 (pipe) 이전 단계의 data를 pipe를 통해 전달해서 다음 명령어를 실행할 수 있도록 한다.  
   이론적으로 n중 파이프를 거쳐 마지막 단계까지 전부 데이터가 pipe를 타고 이동해야 한다.
3. Phase 3  
   Phase 3를 구현한 결과는 다음과 같다.  
   우선, foregound가 아니라, background에서 어떤 명령을(job) 돌릴수 있게 만드는 것이다. 또한, foreground에서 실행되는 명령(job)과 background에서 실행되는 명령을 managing하는 함수인 jobs, 정지한 Background job을 실행시키는 bg, 정지한 foreground job을 실행시키는 fg, 어떤 job을 종료시키는 kill 등의 명령어를 추가로 구현한다.
   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  
    project\_baseline 과 교과서에 나오는 코드를 참고로 했다.  
    Parent는 Fork()함수를 호출함으로서 parent의 내용을 복제한 child process를 만드는데, 호출한 Fork함수의 리턴값이 0이면 성공적으로 child process를 생성한 것이다. 이것을 이용해서 error checking도 가능하다.
  + connection을 종료할 때 parent process에게 signal을 보내는 signal handling하는 방법 & flow  
    Exit의 구현방법에 대해 설명하자면, parent process에게 signal을 보낸다.  
    즉, 현재 process를 kill하고 그 signal을 parent process에게 전달하는 방식의 signal handling으로 exit을 구현하게 된다.   
    본인이 구현한 코드에서는, 현재 process의 pid를 kill함수를 통해 SIGINT의 signal을 parent process에게 전달하는 방식으로 exit을 구현했다.
* **Phase2 (pipelining)  
    
  \*\* Phase 2의 경우, 본인의 코드에 robust함의 문제가 있는데 2~번째 명령어를 실행했을 때, Myshell > 입력창이 잘못된 곳에 뜨지만, 명령어를 입력하면 정상실행하는 경우가 존재한다 \*\* Like :**
  + Pipeline( ‘|’ )을 구현한 부분에 대해서 간략히 설명 (design & implementation)  
    Pipelining은 기본적으로 file descriptor을 통해 구현된다.  
    Pipe가 1개인 경우, 단 1개의 file descriptor을 통해 구현할 수 있다.  
    본인이 구현한 코드에서는, 강의자료에 나온 것 처럼 int fd[2]를 선언하고, fork와 dup함수를 이용해서 구현했다. 1번째 process에서는 STD\_OUT을 close하고, dup를 이용해 복사한 후, 두 fd를 닫고, 프로그램을 실행한다.  
    2번쨰 Process에서는 그와 대칭되는 과정을 실행하고, 프로그램을 실행한다.  
    Pipe가 2개인 경우, 2개의 file descriptor을 이용해서 구현했다.  
    Pipe가 1개인 경우와 거의 동일하지만, 2번쨰 process에서 fd1과 fd2를 전부 dup2()를 통해 복사하고, 두 descriptor를 전부 close하는 과정이 추가로 들어간다.
  + Pipeline 개수에 따라 어떻게 handling했는지에 대한 설명  
    pipeline이 1개일 때에는 file descriptor array 1개를 사용하고, 2개일 때에는 2개를 사용했다. 3개 이상부터는 구현하지 못했다.
* **Phase3 (background process)**
  + Background (’&’) process를 구현한 부분에 대해서 간략히 설명  
    구현하지 못했다.
  1. **개발 방법**
* **B.의 개발 내용을 구현하기 위해 어느 소스코드에 어떤 요소를 추가 또는 수정할 것인지 설명. (함수, 구조체 등의 구현이나 수정을 서술)**앞서 설명한 함수들이 실제로 어떻게 소스코드에 구현되었는지, 함수와 구조체 등의 구현을 설명하겠다.   
    
  첫째로, Phase1에 대해 설명하겠다.  
  우선 exit함수임을 가장 먼저 확인한다. 이것은 argv[0]와 “exit”을 strcmp함수를 통해 비교하고, 리턴값을 확인해서 kill 함수를 호출, sigint signal을 parent에게 전달한다.   
  Cd의 경우, 이것을 구현하기 위해 changeDirectory라는 함수를 작성했다.   
  cd는 현재 argv의 정보를 받는다. Cd xx라고 input을 넣으면, xx 즉 우리가 이동하고 싶은 곳은 argv[1]에 저장되어 있다. 이 argv[1]을 target으로 삼아 이동을 하는데, 여기서 핵심은 chdir함수를 사용한다. 만약 argv[1]이 null이라면, 가장 최상위 디렉터리인 HOME으로 이동하고, 아니라면 chidr로 argv[1]로 이동한다. 만약 존재하지 않는 디렉터리로 이동한다면, 즉 리턴값이 -1이라면 에러메세지를 출력한다.  
  그 외에 다른 함수들은 execvp함수들을 통해 실행한다.  
  초기에 Base code가 execve였던 것으로 기억하는데, 이 함수는 실제 구현때 큰 문제가 있었다. 이 함수는 인자로 받는 실행(filename)이 “/bin/”이 포함된 값이 와야했다는 점이다. 즉, 명세서에 나온 것 처럼 ‘cd’를 입력하면 execve함수는 에러를 뱉는다. 이것을 해결하기 위해 argv[]를 수정해보려 했지만 argv[0]을 수정하는 순간 ARGV[1]도 연쇄적으로 바뀌어서 총체적 난장판이 되는 문제가 있다.  
  결과적으로 구글링을 하다가, 절대경로를 사용하는 것이 아니라, 상대경로를 사용하는 execvp함수를 사용하면 /bin/를 입력해도 명령어가 실행된다는 것을 알게되어 이것을 사용했다.  
    
  둘째로 Phase2에 대해 설명하겠다.  
  Phase2에서는 앞서 설명한 것처럼 파이프의 구현은 dup2함수와 int array인 file descriptor을 이용해서 구현했다. 우선 명령어에 파이프의 갯수를 파악한다.  
  그리고 명령어를 Pipe전후로 parse를 한다. 이렇게 얻은 명령어와 옵션들을 가지고, 파이프의 갯수만큼 fork를 호출해서 child process를 만든다.  
  그리고 파이프가 1개인 경우에는 1개의 fd[2], 2개인 경우에은 FD1[2], fd2[2]를 만든다. 그리고 총 2번/3번 명령을 해야하므로 2개/3개의 명령어/인자 array를 만든다. 이것은 execvp를 2번/3번 호출할때 필요한 array를 분리해야 하기 떄문이다.  
  이렇게 execvp에 쓸 array들이 준비되었다면, 본격적인 PIPELINE을 구성한다. 파이프를 열고 닫기전에, pipe가 1개일떄는 DUP함수를, 2개일떄는 DUP2함수를 이용해서 parent의 file descriptor을 복사했다.  
  pipeline이 1개면 1번 Fork, pipeline이 2개면 2번 fork를 해서 child를 만들고, fd[2]/fd1[2],fd2[2]를 이용해서 파이프를 열고 닫는다.   
  그리고 각 명령어를 execvp를 이용해서 실행하면 phase2가 마무리된다.

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)  
   **
2. **Phase 2 (pipeline)  
   **
3. **Phase 3 (background)**