**Multicore Programming Project 1**

담당 교수 : 최재승

이름 : 남현준

학번 : 20181625

1. **개발 목표**

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

해당 프로젝트에서는 쉘에서 사용되는 fork(), pipeline, jobs in background, signal 등의 개념들에 대해 배운 뒤, 해당 내용을 fork(), dup(), dup2(), user-defined signal handler 및 함수 등을 이용해서 직접 구현해보는 것이 목표인 프로젝트이다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Phase 1

Phase1 에서는 fork() 함수를 통해 입력으로 주어진 command를 실행하게 된다. 입력으로 주어진 command가 builtin command, 즉 linux에서 정의되어 있는 동작이라면 미리 정의되어 있는대로 작동을 하게 될 것이고, 그게 아니라 다른 곳에 위치해있는 excutable file형식이라면 fork()를 통해 parent와 동일한 정보를 가진 child를 생성하여 해당 파일의 실행을 진행하게 될 것이다.

1. Phase 2

Phase2 에서는 pipeline(‘|’)입력이 포함되어 있는 command를 입력으로 받을 경우의 동작에 대해 구현하는 것이다. 입력으로 명령어 | 명령어 형태의 user input이 들어오게 되고 첫번째 명령어를 실행한 뒤, 이어서 ‘|’ 다음에 들어오게 되는 다음 명령어에 대한 수행결과를 화면 상에 출력하게 된다.

1. Phase 3

Phase3 에서는 background job을 구현하고 signal handler들의 개념을 이해하고 해당 부분을 구현하는 것이 주된 목적이었다. 추가적인 구현을 통해, 현재 실행되고 있거나 중지 상태의 process들을 “jobs” 명령어로 확인할 수 있고 fg/bg 을 통해 foreground/background process로 전환시킬 수 있다. Kill을 통해 특정 process를 종료시키는게 가능해진다.

* 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

Fork를 실행하며 해당 부분까지의 동일한 정보를 가지게 되는 child process를 생성하게 된다. Child process인지 parent process인지는 pid를 통해서 구분하게 되는데 일반적으로 pid = 0인 process를 편의상 child process라고 칭하고, pid > 0인 process를 parent process라고 부르게 된다. 입력으로 들어오게 되는 command는 parent와 동일한 정보를 가지는 child process 에서 실행되게 된다.

Fork()를 기준으로 분기가 되며 pid가 다르다는 특징을 이용해 child process에서의 logic과 parent process에서의 logic을 각기 선언해 줄수 있다.

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

fork()를 통해 생성된 child process는 실행이 종료됨이 따라 SIGCHLD signal을 parent process에 전달하게 된다. parent process는 SIGCHLD signal을 받음에 따라 child process가 종료되었다는 것을 알고 이를 reaping 해주어야 할 필요가 있다. 이를 위해 parent process 쪽에서 waitpid(pid, &signal, NULL) 함수를 호출하여 종료된 child process의 reaping을 진행한다.

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

Parseline() 함수 부분을 수정하여 ‘|’ 문자를 기준으로 들어오게 되는 명령어들을 나눠서 저장해줘야 할 필요가 있다. 이 때, 들어오게 되는 명령어 문자열을 탐색하며 따옴표 표시가 있을 경우, 해당 부분을 무시하고 다음 문자부터 저장하도록 parseline() 함수를 구성했다. 해당 명령어들을 ‘|’ 문자를 기준으로 명령어를 저장하는 배열의 index번째에 나눠서 저장했다.

이 작동 이후에, eval() 함수에서 user input에 ‘|’가 존재했을 경우, 명령들에 대해 pipe를 선언하고 i번째 명령어의 출력을 i+1번째의 입력으로 넘겨주었다.

* + Pipeline 개수에 따라 어떻게 handling했는지에 대한 설명

입력으로 들어오는 user input에 ‘|’가 존재하지 않았다면 phase1에서 선언되어 있는 parseline() 로직에 따라 작동하도록 않았다.

만약 ‘|’가 존재했다면, pipe를 열어 첫 번째 명령어의 STDOUT을 다음 명령어의 STDIN으로 pipe를 통해 넘겨주고 이걸 명령어를 저장한 배열에 저장되어 있는 값이 NULL이 아닐 때까지 반복하게 된다. 이후에, 이렇게 넘겨진 명령어들을 execve를 통해 정의된 로직에 따라 실행이 된다.

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

Parseline() 함수에서 ‘&’ 문자의 입력 여부를 확인하게 된다. 만약 ‘&’가 입력에 있었다면 해당 문자를 삭제한 다음 문자열을 argv에 저장하게 된다. 이 때, foreground로 실행되는 process는 background process와 달리 사용자와의 상호작용이 있을 수 있고, 해당 process의 child process가 끝날 때까지 대기를 해야한다.

* 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**

Phase1 에서는 eval() 함수 내부에서 fork() 동작을 통해 process를 복제할 수 있도록 해당 부분을 함수 내부에 추가해줄 것이다. 구체적으로는, 만약 builtin command가 아니라면 return 값이 존재하지 않아 즉시 종료되어 main 함수 내부의 while문을 탈출하게 될 것이다. 따라서, 해당 부분을 방지하기 위해 builtin command가 아니라면 외부 실행파일 일 가능성이 있으므로 fork()를 통해 child process를 생성 후, child process에서 해당 실행을 진행할 수 있도록 할 것이다.

Phase1 에서의 구현에서 history의 내역을 저장할 수 있는 배열 char history[MAXARGS][MAXLINE]와 저장된 history의 숫자를 알려주는 int histiry\_num을 선언하여 user input으로 들어오게 되는 command를 저장할 수 있도록 했다.

History 명령어 부분에서 중복된 command가 연속으로 들어오게 될 경우, 첫 번째 명령만 저장되야 한다는 조건이 있었다. 해당 부분을 위해 char last\_command(MAXLINE)을 선언하여 user input으로 들어온 마지막 command가 뭐였는지 기억할 수 있게 했다.

* **Phase2**

Parseline() 함수 부분에 변화를 주었다. 기존의 parseline() 함수는 입력으로 주어진 문자열을 그대로 읽어서 저장하게 되는데 이럴 경우, pipeline을 관리할 수 없는 문제가 발생한다. 따라서 parseline() 부분에 우선 입력된 command를 읽을 수 있는 부분을 새롭게 정의했다. 이 결과에 따라서, 기존에 있던 parseline()의 부분이 실행될지 말지 결정하게 했다. 이를 위해, number\_of\_pipes와 pipe\_flag 변수를 선언해서 입력된 command에 ‘|’의 개수가 몇 개였는지 센 다음, pipe\_flag가 1이라면 새롭게 정의한 내용에 맞게 입력된 command를 ‘|’을 기준으로 쪼개어서 저장할 수 있게 하였다. Pipe\_flag가 0이라면 기존에 있던 내용을 그대로 따라가서 실행된 다음 함수가 종료된다.

‘|’을 기준으로 새로운 명령이 들어오게 된다. 따라서, command에 있는 명령을 모두 저장하기 위해서는 n개의 명령을 ‘|’을 기준으로 쪼개서 저장할 수 있어야 한다. 해당 부분을 위해서 char \*pipes[MAXLINE]을 선언했고, ‘|’가 나올 때마다 number\_of\_pipes를 증감시킨다. 이 때, pipes에는 pipes[number\_of\_pipes++]의 형태로 command가 저장되게 된다. 이게 의미하는 것은 n번째 command command를 pipes의 number\_of\_pipes-1번째 index에 저장하라는 의미이다.

한편, 이렇게 pipes에 저장된 명령어들의 집합을 그대로 pipe()를 이용해 사용하는 것에는 조금 부족한 부분들이 있다. 따라서, pipe에 저장된 명령어들을 기본으로 한 pipe 실행을 위해 추가적인 요소들을 포함한 CMD 구조체를 선언했다. (myshell.h 파일 참조). 해당 구조체의 argv에는 pipes에 저장되어 있는 명령어들이 저장될 것이고, argc는 명령어의 개수를 저장하게 될 것이다. 해당 구조체에 선언되어 있는 fd[2]가 pipe 실행을 통해 i번째 명령어의 STDOUT과 i+1번째 명령어의 STDIN을 연결해줄 때 사용될 것이다.

Eval() 함수에서, pipe가 user input에서 있었을 때와, 없었을 때를 기준으로 작동을 나누었다. Pipe가 있었다면, pipe의 정의에 맞게 pipe를 열고 닫고 하여, 각 command들에 대해 input과 output을 정의된 로직에 따라 연결해줄 수 있도록 하였다.

* **Phase3**

phase3에서는 signal의 구현 및 생성된 process의 정보를 저장할 수 있는 수단이 필요했다. 이를 위해 signalint\_handler(), signaltstp\_handler(), signalchld\_handler(), 등을 통해 해당 signal이 들어왔을 때 어떤 방식으로 처리하게 될지 handler를 구현했다.

signalint\_handler(), signaltstp\_handler()에서는 SIGINT, SIGTSTP signal이 들어왔을 때, foreground 상태의 process를 찾아서 terminate 시키거나 중지 상태로 변경시킬 수 있는 logic을 정의하였다.

signalchld\_handler() 함수는 child process에서의 signal에 따라 다르게 작동할 수 있도록 정의해주었다.

command 가 입력되었을 때, child process가 생성되고 해당 process들을 관리해줄 필요성이 있다. 이를 위해 해당 정보를 저장하게 될 JOBS 구조체를 선언했다. 해당 구조체에는 process의 pid, state, 실행된 command 등에 대한 정보를 저장할 수 있다.

JOBS 구조체에 원소를 추가 및 삭제를 할 수 있게 add\_job(), remove\_job\_by\_pid(), remove\_job() 함수를 추가했다.

JOBS에 추가되는 process가 foreground process일 경우, 해당 process에서의 작업이 끝날 때 까지 기다려야 한다. Fg\_wait() 함수를 선언해서 해당 process ID에서의 작업이 완료될 때까지 대기하는 logic을 선언했다. Background process일 경우, 대기할 필요 없이 다른 command도 같이 수행할 수 있다.

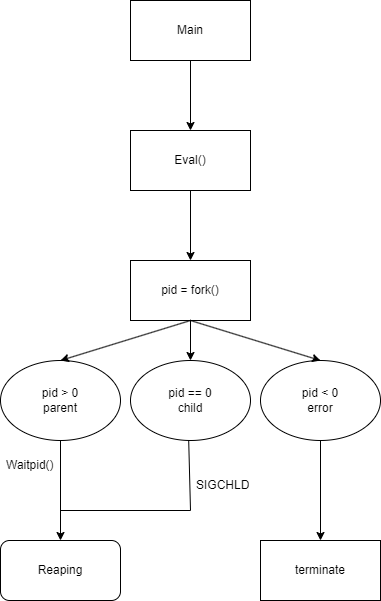
Builtin\_command() 함수 부분에 bg, fg, kill, jobs 명령어가 input으로 주어졌을 때 실행될 수 있도록 해당 부분들에 대한 처리를 함수 내부에 진행했다.

Eval()함수에서 signal bit를 관리할 수 있게 sigset\_t mask를 선언하고 fork()를 실행하기 전에 race에 의해 발생하는 원하지 않는 결과를 배제하기 위해 sigprocmask()를 통해 SIGCHLD signal을 set한다. 이렇게 함으로 parent process에서 SIGCHLD가 부모에서 handle되기 전에 receive되는 걸 막을 수 있다.

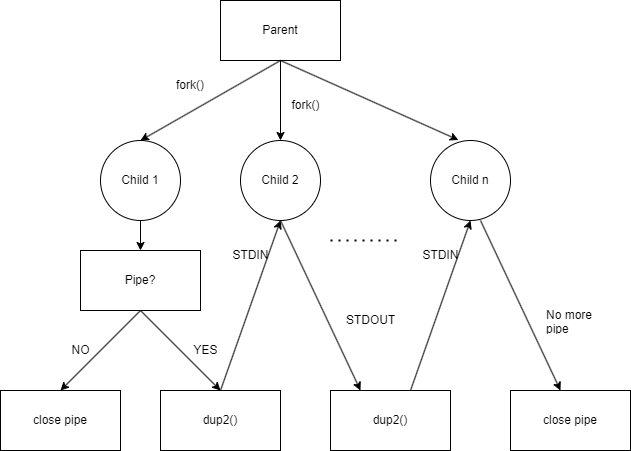
1. **구현 결과**
   1. **Flow Chart**

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

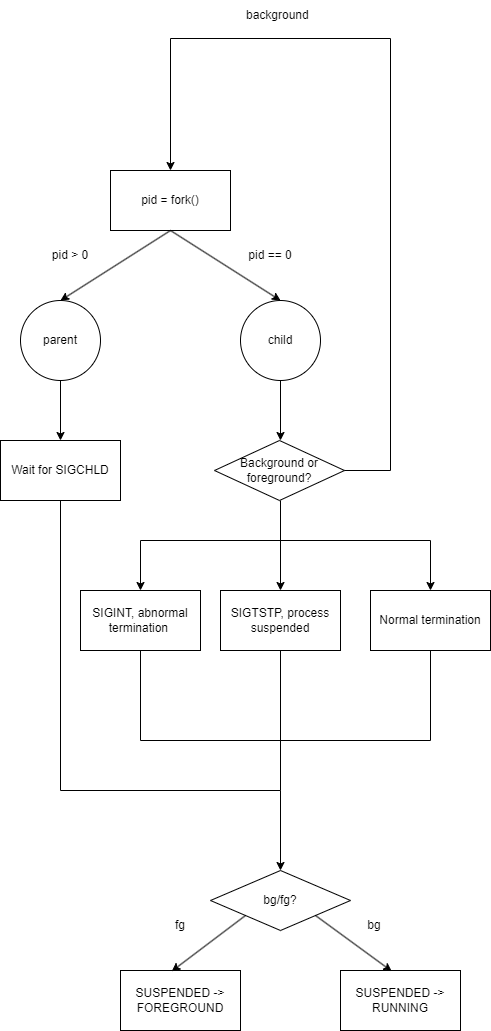
1. **Phase 1 (fork)**

****

1. **Phase 2 (pipeline)**

****

1. **Phase 3 (background)**

****