**Multicore Programming Project 1**

담당 교수 : 최재승

이름 : 남기동

학번 : 20180032

1. **개발 목표**

* 이번 프로젝트의 목표는 쉘의 기본적인 기능을 탑재한 나만의 쉘을 설계하고 구현하는 것이다.
* Phase1에서는 fork함수와 waitpid함수를 활용하여 child process에서 command를 실행하고 난 후 child process의 signal을 받고 reaping하는 과정을 통해 기본적인 명령어들을 구현한다.
* Phase2에서는 phase1에서 구현한 쉘을 기반으로, pipeline(‘|’)이 포함된 명령어를 처리하여 한 명령어의 출력이 다음 명령어의 입력으로 이어질 수 있는 기능을 구현한다.
* Phase3에서는 phase2에서 구현한 쉘을 기반으로, 백그라운드에서 프로세스가 돌아갈 수 있도록 하는 기능을 추가하고 jobs, bg, fg, kill과 같은 프로세스 제어 명령어들이 정상적으로 작동할 수 있도록 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **Phase1 (fork & signal)**

1) main함수에서 cmdline을 입력받아서 eval함수로 넘긴다. //1. READING

2) eval함수에서 parseline함수를 통해 argv 단위로 cmdline을 분리한다 //2. PARSING

3) test\_history함수로 argv[0]에 !!나 !#이 있는지 검사하고 있다면, history파일에서 찾아서 해당 라인으로 대체

4) echo함수에서 따옴표를 제거하기 위해 따로 command\_echo를 통해 처리

5) argv[0]에 "history"가 들어오면, command\_history함수를 통해 history를 출력

6) 그 이후의 명령어들은 builtin\_command 함수를 통해 내장 명령어인지 검사

7) 내장 명령어가 아니라면 Fork를 이용하여 child process에서 command를 실행한다 //3. EXECUTING

8) 전반적으로 eval 함수가 끝날 때, save\_command\_history함수로 cmdline을 .history 파일에 저장한다

* **Phase2 (pipelining)**

파이프라인를 포함한 명령어를 수행할 수 있게 된다. 파이프라인을 2, 3개와 같이 적은 숫자로 한정되는 것이 아니라 파이프의 개수가 늘어나도 문제 없이 파이프라인 명령어를 수행할 수 있다. 파이프라인 명령어는 pipe함수와 dup2함수를 사용하여 한 프로세스의 입력을 이전 프로세스에서 가져오고 출력은 다음 프로세스로 보내는 기능을 말한다.

* **Phase3 (background process)**

백그라운드 프로세스 (‘&’)를 실행시킬 수 있으며 jobs 명령어를 통해 현재 작동하거나 멈춰져 있는 작업들의 목록을 확인하고 kill 명령어를 통해 특정 작업을 강제로 종료시키고 fg 명령어를 통해 foreground로 job를 전환하고 bg 명령어로는 foreground job을 background로 전환할 수 있다.

또한 시그널 핸들러를 만들어서 Ctrl + C(SIGINT), Ctrl + Z(SIGTSTP) 등의 시그널을 적절히 처리하게 된다.

* 1. **개발 내용**
* **Phase1 (fork & signal)**

명령어(cmdline)을 입력 받아서 parseline함수를 통해 명령어 단위를 구분하고 argv[0]에 위치한 명령어가 올바른 명령어라면 fork함수를 통해 child process를 생성하고 child process에서 해당 명령어를 execve함수를 통해 실행한다.

명령어를 실행하기 이전에 Fork()를 수행하고 ‘pid == 0’은 child process를 뜻하며 child process가 명령어를 실행을 하고 종료하게 된다. 이 때 if(pid == 0)의 아래 부분에는 Waitpid()함수를 사용하여 parent process가 child process의 signal을 확인하고 reaping하도록 한다.

* **Phase2 (pipelining)**

우선 명령어에 파이프라인(‘|’)이 포함되어 있는지를 파악하고 파이프라인의 위치, 개수 등을 파악한 다음 이를 토대로 파이프라인에서 사용하기 편하도록 명령어 배열을 만든 후 파이프라인이 있는 경우에만 파이프라인 명령어를 수행할 수 있는 함수를 이용하여 적절한 수행을 취한다.

파이프라인 명령어를 수행하기 위한 함수는 pipe함수와 dup2함수를 사용한다. pipe함수를 통해 파일 디스크립터를 만들고 이를 dup2함수를 통해 입력 파이프와 출력 파이프를 통해 명령어를 연달아 계산해 나가면 된다.

* **Phase3 (background process)**

Parseline 함수에서 명령어 라인의 마지막 문자가 &인 경우 백그라운드 프로세스로 판단하고 eval함수에 넘긴다. 백그라운드 프로세스와 foreground 프로세스가 job에 추가되면 jobs, kill, fg, bg 명령어를 적절히 수행할 수 있다.

* 1. **개발 방법**

**Phase1 (fork & signal)**

Phase1의 fork와 signal을 구현하기 위해서는 수업 시간에 사용했던 수업 자료와 교재에 나와 있는 코드를 참고하였다. 앞서 이야기했던 것처럼 if((pid = Fork()) == 0)을 사용해서 fork를 이용하여 자식 프로세스를 만들고 자식 프로세스의 프로세스 id를 pid에 저장한다. 자식 프로세스의 pid는 0이기 때문에 해당 if문은 자식 프로세스만 접근하게 되며 부모 프로세스는 if절 하단으로 이동한다. If문 안에서는 execve함수를 사용하여 입력된 명령어를 수행한다. execve함수는 에러가 발생하지 않는 이상 반환값이 없으며 따라서 정상적으로 수행했을 때 자연스럽게 자식 프로세스가 종료된다. 에러가 발생하면 execve함수 하단에 에러 출력 코드를 통해 적절한 에러를 출력하고 exit함수를 통해 자식 프로세스를 종료시킨다. 부모 프로세스는 Waitpid함수를 통해 자식 프로세스가 terminate될 때까지 대기하다가 status에 종료 상태를 받아오게 된다.

* **Phase2 (pipelining)**
* parseline함수를 통해 cmdline을 명령어 단위로 구분짓는 과정에서 파이프라인(‘|’)의 위치와 개수를 알아내고 이를 토대로 eval함수에서 파이프라인이 있는지에 따라 서로 다른 명령어를 실행시킨다. 파이프라인이 없는 경우에는 이전 phase1에서 구현한 것과 동일한 흐름에 따라 명령어를 실행시키는 반면 파이프라인이 있는 경우에는 파이프라인을 실행시킬 수 있도록 짠 pipe\_command 함수를 호출하여 파이프라인 명령어를 적절하게 수행시킨다.
* Pipe\_command함수에서는 마지막 명령어가 아닌 경우마다 pipe함수를 통해 pipefd에 파일 디스크립터를 만들어낸다. 이후 fork함수를 통해 자식 프로세스를 만들고 나서 이전 프로세스의 입력을 dup2 함수를 통해 표준 입력으로 연결하고 현재 프로세스의 쓰기를 표준 출력으로 연결한다. 마지막 명령어의 경우에는 입력만 연결하고 명령어를 실행하는 과정을 거친다.
* **Phase3 (background process)**

시그널 핸들러로 SIGINT, SIGSTP, SIGCHILD를 만들었고 job을 structure로 만들어서 job의 id(순서), pid, status, command 등을 저장할 수 있도록 했다. Built-in command로 인식되는 kill, fg, bg, jobs를 각각의 기능을 수행할 수 있는 함수들을 만들어 연결시켰고 인자로 넘긴 id와 동일한 id를 갖고 있는 job이 있는지 탐색한 후 해당 id와 동일한 id를 갖고 있는 경우에 적절한 조치를 취한다.

1. **구현 결과**
   1. **Flow Chart**
2. **Phase 1 (fork)**

**도표이(가) 표시된 사진

자동 생성된 설명**

1. **Phase 2 (pipeline)**

**도표이(가) 표시된 사진

자동 생성된 설명**

1. **Phase 3 (background)**

**도표이(가) 표시된 사진

자동 생성된 설명**