**Multicore Programming Project 1**

담당 교수 : 박성용

이름 : 안희원

학번 : 20211550

1. **개발 목표**

리눅스 MyShell을 3단계로 나눠 직접 각각의 단계에 맞는 기능을 구현해본다. 첫째로, 직접 함수 코드를 짜서 cd, history 등의 built-in 함수를 구현하고, built-in이 아닌 함수들은 fork()와 execve()함수를 이용해 구현한다. 둘째로는 두 개 이상의 명령어를 연결해주는 pipe 명령어를 파일 디스크립터 개념과 리눅스에서 제공하는 pipe()를 이용해 구현한다. 셋째로는 지금까지는 fore-ground에서만 프로세스가 돌아갔다면, child process와 parent process의 개념뿐만 아니라 signal의 개념을 응용하여 back-ground 기능을 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Phase 1

Built-in command가 아닌 ls, mkdir, rmdir, touch, cat 등의 명령어를 shell의 command line에 입력하였을 때, 각각의 명령어가 잘 작동하는 것을 확인할 수 있었다. 또한 specification에 나와있는 예제 또한 실행해 보았을 때 잘 되는 것을 확인할 수 있었다. Built-in command인 cd, history, exit의 경우도 잘 되는 것을 확인할 수 있었다. 또한 history와 관련된 명령어인 !!, !# 역시 하나의 command line에서 나온 횟수와 상관없이 쉘과 동일하게 잘 실행되는 것을 볼 수 있다.

1. Phase 2
2. Phase 3
   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

eval() 함수 내에 있는 fork() 함수를 실행시키게 되면 현재 실행 중인 process에 대한 child process가 생성된다. 이후, child process의 역할은 parseline() 함수를 거쳐 ‘ ‘ 단위로 쪼개진 argv[]에 저장된 명령어를 execvpe() 함수를 통해 실행한다. 이때, child process보다 parent process가 먼저 종료되어 zombie process가 되는 것을 막기 위해 parent process는 child process가 종료될 때까지 기다려야 한다.

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

fork()함수를 호출하게 되면 pid값을 return하게 되는데, 이 return값을 가지고 child process와 parent process를 다르게 처리할 수 있다. 먼저, return값이 0이면 현재 process가 child process임을 알 수 있다. child process에서 command line으로 들어온 명령어를 수행된다. 명령어에 맞는 처리가 끝나고 나면 child process는 종료되게 된다. 그러나 parent process는 child process가 종료되기 전까지 종료되면 안 되므로 waitpid() 함수를 이용해 child process가 종료될 때까지 기다린 후, 종료될 경우에 보내지는 SIGCHLD 시그널을 받아 처리하도록 하였다.

* **Phase2 (pipelining)**
  + Pipeline( ‘|’ )을 구현한 부분에 대해서 간략히 설명 (design & implementation)
  + Pipeline 개수에 따라 어떻게 handling했는지에 대한 설명
* **Phase3 (background process)**
  + Background (’&’) process를 구현한 부분에 대해서 간략히 설명
  1. **개발 방법**
* **B.의 개발 내용을 구현하기 위해 어느 소스코드에 어떤 요소를 추가 또는 수정할 것인지 설명. (함수, 구조체 등의 구현이나 수정을 서술)**

Child process를 생성하는 이유는 command line으로 들어온 명령어가 built-in command가 아닌 경우, child process를 만들고 system call을 이용해 명령어를 처리하기 위함이다. 따라서 실제로 명령어가 실행되는 eval()에서 child process를 생성하였다. 먼저 if문을 사용해 builtin\_command()를 이용해 주어진 명령어가 built-in command가 아니라면 fork()를 이용해 child process를 생성한다. 이후, execvpe()를 이용해 명령어를 실행시키도록 한다. 만약, 들어온 명령어가 built-in command인 경우에는 child process를 생성할 필요가 없다. 이때, fork()의 return값을 이용해 child process와 parent process를 구별하여 parent process인 경우에는 waitpid()를 호출하여 child process가 끝날 때까지 기다린 뒤, child process가 종료되면 signal을 전달받아 처리하도록 하였다.

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)**

**도표이(가) 표시된 사진

자동 생성된 설명**

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