**Multicore Programming Project 1**

담당 교수 : 최재승

이름 : 강상원

학번 : 20191559

1. **개발 목표**

* (MyShell을 만드는 전체적인 개요에 대해서 작성하면 됨.)
* Linux Shell 기능을 모방하여 커스텀 Shell을 작성하는 프로젝트이다.
* 먼저 cd, ls, mkdir, cat과 같은 linux 내장 기능들을 사용케 하고 history 기능 중 (!#와 !!)을 직접 구현하여 본다.
* 다음으로 Pipe를 통해 여러 커맨드의 입출력을 조합할 수 있게끔 한다.
* Process Signal Handler를 통해 백그라운드 프로세스와 각종 프로세스 상태를 제어하게 된다. (fg, bg, kill)

1. **개발 범위 및 내용**
   1. **개발 범위**
2. **Phase 1**

cd, ls, mkdir, cat과 같은 명령어를 받아 해당 결과를 출력해준다. fork()를 통해 자식 프로세스를 만들어 그곳에서 커맨드를 실행하며 각 명령어는 exec()를 통해 리눅스 내장 명령어를 실행한다. history, cd와 같은 명령어는 직접 구현하여 작동한다.

1. **Phase 2**

Phase 1의 구현사항에 더해 pipeline이 추가된 경우에도 알맞게 동작할 수 있게 한다. File descriptor를 조작하여 먼저 번의 명령어의 결과가 pipeline 다음의 입력으로 들어가 실행되도록 구현하였다.

1. **Phase 3**

Commandline 끝에 ‘&’가 추가된 경우에 해당 명령어가 background로 실행되게 된다. 또한 다음과 같은 명령어들이 추가 구현되어 작동한다.

* + - jobs

실행 중이거나 stop된 background 프로세스 정보들을 나열한다.

* + - bg <job>

해당 job number에 해당하는 명령어를 background로 실행되게 한다.

* + - fg <job>

해당 job number에 해당하는 명령어를 foreground로 실행되게 한다.

* + - kill <job>

해당 job number에 해당하는 명령어를 끝낸다.

* 추가로 Ctrl+Z 입력으로 foreground 프로세스를 background stop된 프로세스로 변경 가능하다.
  1. **개발 내용**
* **Phase1 (fork & signal)**
  + fork를 통해서 child process를 생성하는 부분에 대해서 설명

parseline() 함수로 정리된 commandline을 builtin\_command() 함수로 builtin command 여부를 판단하고, 이후 fork()를 통해 자식 프로세스를 생성하여 입력한 commandline이 실행되게 한다. 생성된 자식 프로세스에서 실행되었으므로 해당 동작이 끝나면 프로세스를 종료한다. (waitpid() 사용, 종료 확인)

* + connection을 종료할 때 parent process에게 signal을 보내는 signal handling하는 방법 & flow
* **Phase2 (pipelining)**
  + Pipeline( ‘|’ )을 구현한 부분에 대해서 간략히 설명 (design & implementation)

Commandline에 pipe 기호가 있는지 체크하고, 그 개수와 위치를 체크한다.

Close(), Dup2() 함수를 통해 파일 디스크립터를 파이프 다음의 프로세스로 넘겨 주는 역할을 한다.

* + Pipeline 개수에 따라 어떻게 handling했는지에 대한 설명

파이프가 여러 개일 수 있으므로 파이프 다루는 함수를 재귀 호출하여 처리한다. 이 때 아까 저장한 파이프 개수를 체크하여 모든 파이프라인을 처리하였는지 확인한다. 파이프 처리 함수는 파이프 기호 사이의 단일 커맨드를 받아 execve() 함수로 실행한다.

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

parseline()의 반환값으로 마지막 ‘&’ 기호 유무를 받은 eval() 함수는 Background 프로세스가 아니라면 기존과 같이 실행하고, background process라면 새로이 설계한 jobs 구조체 링크드 리스트에 해당 정보를 추가하고 적절한 signal handling이 뒤따른다.

* 1. **개발 방법**
* **Phase 1**

history 명령어: builtin\_command() 내에서 입력된 commandline을 확인하여 알맞은 history command가 실행되도록 한다. History를 저장하기 위하여 fopen()으로 .myshell\_h\_20191559 파일을 생성하고 이곳에 한 줄씩 기록한다.

cd 명령어: chdir() 함수로 인자값에 해당하는 디렉토리로 이동한다.

그 외의 명령어는 execve()로 리눅스에 내장된 기능을 호출하였다.

* **Phase 2**

기존 eval() 함수 내에 strstr() 호출로 파이프 기호가 존재하는지 확인하고 존재한다면 새로이 만든 파이프 핸들링 함수를 호출한다. 이후 작동 방식은 2.B의 Phase2에 서술한 내용과 동일하다.

파이프 개수를 저장하여 파이프가 여러 개일 경우 모든 파이프 라인을 다루었는지를 확인하고, 파일디스크립터 배열을 생성하여 각각의 파이프라인 프로세스마다의 파일 디스크립터를 저장하고 Dup2(), Close()를 통해 다룰 수 있게끔 한다.

* **Phase 3**

Ctrl+C, Ctrl+Z를 다루는 signal handler 함수를 생성하였으며, 프로세스의 상태값을 다루는 전역 enum 값을 생성하였다.

Job들의 상태를 다루는 구조체 링크드 리스트를 선언하여 fg, bg, kill에 의한 프로세스 상태 변화를 dlt\_job(), add\_job() 등 함수로 다룬다.

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**