**Multicore Programming Project 1**

담당 교수 : 최재승 교수님

이름 : 김태곤

학번 : 20191583

1. **개발 목표**

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

이 프로젝트의 목표는 linux 환경의 shell을 구현하는 것이다. 시스템 수준의 프로세스 제어, 프로세스 신호, 프로세스 간 통신 및 Linux 쉘에서 백그라운드에서 실행되는 프로세스 및 작업을 익히고 이해하는 것이 주된 목적이다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Phase 1

Linux shell의 기본적인 명령어를 수행할 수 있다. 쉘이 반복적으로 프롬프트를 출력하고 stdin을 통해 터미널에서 명령 줄을 대기한 다음 명령 줄의 내용에 따라 작업을 수행한다. fork와 exec system call을 사용해 다양한 명령어와 builtin command를 수행할 수 있다. (external command : ls, mkdir, rmdir, touch, cat, echo) (built in command : cd, history, exit)

1. Phase 2

Phase1에서 구현한 간단한 shell의 기능을 확장한다. 각 파이프라인 명령에 대해 새로운 프로세스를 생성하고 부모 프로세스가 마지막 명령을 기다리도록 한다. 단방향 데이터 흐름으로 프로세스들의 명령들이 서로 연결되고, 명령어의 결과 데이터가 다른 명령어의 input이 되게한다.

1. Phase 3

백그라운드 작업이 추가된다. Jobs를 통해 백그라운드에서 실행중인 작업을 확인하고, kill을 이용해 작업을 종료시킨다. 중지된 백그라운드 작업츨 실행중인 백그라운드 작업으로 변경이 가능하고, 중지된 또는 실행중인 백그라운드 작업을 포그라운드에서 실행중인 작업으로 변경할 수 있다. &가 주어지면 백그라운드에서 명령을 시작해야 한다.

* 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 통해 현재 실행중인 프로세스의 복사본을 만들고, 완전히 독립적인 프로세스인 child process를 생성한다. 이후 child process에서 execvp를 호출해 다양한 명령어들이 실행된다.

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

parent process는 waitpid를 통해 child process가 종료될 때까지 기다린다. SIGCHLD signal을 받아 child process의 종료 상태를 확인한다. Signal handler에서는 이 signal을 받으면 child process의 종료 상태를 확인하고, waitpid를 통한 대기나, 종료된 child process의 정보를 처리하기도 한다. Child process가 종료되면서 signal을 보내면 parent는 waitpid를 끝내고 다음 입력 작업을 수행하게 된다.

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

Pipe함수를 통해 읽기용과 쓰기용인 두개의 File Descriptor(fd)를 사용한다. 파이프를 통해 연결된 두 프로세스는에 각각의 파일 디스크립터를 할당한다. 한쪽은 파일을 읽고, 다른 한쪽은 쓰기를 하는 것이다. 파이프를 구현하는 과정에서는 fork() 함수를 이용하여 자식 프로세스를 생성하고, exec() 함수를 이용하여 새로운 프로세스를 실행한다. 이때, 자식 프로세스는 파일 디스크립터를 상속받으므로, 부모 프로세스에서 열었던 파일 디스크립터를 그대로 사용할 수 있다. dup2 함수를 이용해 자신의 stdin을 파이프(pipe)의 읽기 엔드(read end)와 연결시킨다. dup2 함수가 동작할 때, 두 번째 파라미터의 파일 디스크립터는 먼저 닫힌다.

* + Pipeline 개수에 따라 어떻게 handling했는지에 대한 설명

입력 받은 문장에서 pipeline을 기준으로 문장을 나누었다. 일단 파이프가 없다면 execvp함수를 호출해 명령어를 바로 실행시킨다. 파이프가 한 개 이상이라면 fork를 통해 프로세스를 만든다. Child process에서는 리다이렉트 시켜주고 execvp를 통해 해당 문장을 실행시킨다. Parent에서는 두번째 파라미터의 파일 디스크립터를 닫아주고 해당 함수를 재귀적으로 호출해 다음 문장을 실행시킨다.

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

아쉽게도 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)**
* **Phase1** 
  + 먼저 기본 명령어들을 수행하기 위해 eval 함수 내 fork하여 생성된 child process에서 execvp를 통해 구현하였다. Execvp를 통해 실행된 echo는 따옴표까지 함께 출력되므로 따옴표와 큰따옴표를 제거하는 코드를 추가하였다. builtin 명령어를 구현하기 위해 builtin\_command를 수정하였다. Quit과 동일한 기능의 exit을 추가해 주고, chdir을 통해 cd를 구현하였다.
  + History는 총 3개의 함수로 나누어 구현하였다. 먼저 write\_history함수를 통해 입력받은 명령어를 파일에 저장하는 역할을 수행하였다. 입력했던 내용이 없으면 새로운 파일을 만들어 추가하고, 입력했던 내용이 있으면 현재 입력된 명령어 직전 명령어와 비교하여 중복 여부를 판단하였다. 이 과정은 fseek, ftell 등을 사용하여 파일에서 가장 최근 명령어를 가져오고, 파싱해서 비교하였다. check\_history 함수는 !!와 !#을 history에서 체크하여 해당 내용으로 교체하는 역할을 한다. 교체 과정은 memmove와 memcpy를 사용해 !!, !#의 길이만큼 이동하고 command line을 넣어 주었다. show\_history 함수는 history 내역을 보여주는 역할을 한다. Cntflag로 -2가 입력되면 history 명령어로 모든 history를 보여준다. -1이면 !!로 가장 최근의 history 명령어를 보여주며 그 외는 !#의 내역을 보여준다. 만약 #이 history파일 내 줄 수보다 크다면 오류를 출력한다.
* **Phase2**
  + 파이프는 총 4개의 함수를 통해 구현하였다. Check\_pipe 함수는 파이프라인의 수를 return한다. Strchr을 통해 개수를 구하였고, 없을시 -1을 리턴한다. Show\_pipe 함수는 입력된 명령어를 pipe\_line 수만큼 나누고, 분리된 문자열을 call\_pipe함수로 넘기는 역할을 한다. 문자열을 파이프 기준으로 분리할 때는 strtok함수를 사용해 분리하였다. Redir 함수는 이전 파일 디스크립터와 새로운 파일 디스크립터를 리다이렉션해준다. Call\_pipe함수는 리다이렉션된 것을 바탕으로 execvp함수가 실행된다. 개발 방법은 개발 내용에 적었던 내용과 유사하다. 이렇게 show\_pipe함수는 eval에서 실행된다. 파이프가 들어왔는지 check\_pipe를 통해 확인한 후 eval에서 fork를 통해 자식 프로세스를 생성시키고 실행된다.

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**

Figure Phase1 (fork)

1. **Phase 2 (pipeline)**

**도표이(가) 표시된 사진

자동 생성된 설명**

Figure Phase2 (pipeline)

1. **Phase 3 (background)**