**System Programming Project 1**

담당 교수 : 김영재 교수님

이름 : 이건영

학번 : 20181662

1. **개발 목표**

수업 시간에 배운 process와 signal 등의 개념을 활용하여 리눅스 Shell의 기능을 수행하는 Myshell 프로그램을 구현한다.

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

ls, cd, echo 등의 shell 내부 명령어를 수행하는 기능을 구현한다.

Bash 내에서의 명령어 실행 결과와 동일한 결과를 출력한다.

1. Phase 2

‘|’ 기호를 사용하여 shell의 pipeline 기능을 구현한다.

‘|’ 기호를 기준으로 앞에 있는 process의 결과가 뒤의 process의 입력으로 전달되어 뒤의 process의 인자로 활용된다.

1. Phase 3

명령어를 background process로 실행하는 기능과 background / foreground process를 관리하는 기능을 구현한다.

Jobs 명령어를 통해 현재 background process의 종류와 상태를 볼 수 있으며, kill 명령어를 통해 process들을 종료시킬 수 있고, bg와 fg 명령어를 통해 해당 프로세스를 background / foreground에서 진행되도록 변경할 수 있다. 또한 Ctrl+Z 커맨드를 통해 foreground process를 멈추어 background로 보낼 수 있다.

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

Shell이 실행되면 종료되기 전까지 사용자로부터 명령어를 입력 받는다. 입력 받은 명령어가 built-in command에 해당된다면 builtin\_command 함수 내에서 처리한다. 만약 그 외의 명령에 해당된다면 child process를 Fork 함수를 통해서 생성하고, child process 내에서 execve 함수를 통해 사용자가 입력한 명령어를 실행한다. 이 때 parent process는 waitpid 함수를 통해 child process의 pid를 이용해 해당 process가 종료되면 reaping 하도록 구현하였다. 이후 다시 사용자의 입력을 받는 단계로 돌아간다. 만약 사용자가 ‘quit’ 또는 ‘exit’을 입력했다면 shell을 종료한다.

* **Phase2 (pipelining)**

사용자로부터 받은 입력에 ‘|’ 문자가 존재한다면 기존 명령어 처리 부분에서 pipe 부분으로 넘어간다. 이후 ‘|’ 문자를 기준으로 명령어를 parsing 해주는데, ‘|’ 문자를 발견할 때마다 개수를 기록한다. 이후 pipe\_command 함수를 실행한다.

pipe\_command 함수는 재귀적으로 구현했으며 앞의 명령어를 실행하여 출력한 fd를 다음 pipe\_command 호출 시에 넘겨주었다. 이후 마지막 명령어까지 실행된 이후에 waitpid로 reaping 후 사용자 입력 대기 상태로 돌아간다.

* **Phase3 (background process)**

사용자의 입력에 ‘&’이 존재하는지 확인한다. 만약 존재한다면 parseline 함수의 return 값이 1이 되어 background 처리 부분으로 넘어간다. 해당 command line, process id를 jobList에 추가하고 사용자의 입력을 받는 상태로 돌아가도록 구현했다. 또 foreground에서 작업이 진행중일 때 Ctrl+Z를 입력한다면 해당 process를 중지시키고 background로 보낸다.

bg 명령어를 사용하여 background에서 중지된 process를 다시 run 시킬 수 있으며, fg 명령어를 사용해 background의 process를 foreground에서 run 시킬 수 있다. 또한 kill 명령어를 사용해 background의 process를 종료시킬 수 있다.

* 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 (fork & signal)**

우선 명령어는 built-in command인 것과 아닌 것으로 나뉜다. 우선 bulit-in command에 해당하는 명령어(exit, quit, cd)는 builtin\_command 함수 내에서 실행된다.

exit / quit : exit(0)으로 넘어가게 해 프로그램을 종료시킨다.

cd : chdir 함수를 이용하여 디렉토리를 변경한다.

이외의 명령어는 execve 함수를 이용해서 /bin 또는 /usr/bin 에 있는 프로그램을 실행하도록 구현하였다. 이를 위해 Fork 함수를 이용해서 자식 프로세스를 생성했다. 이 때 자식 프로세스에서는 interrupt와 stop signal을 받아야 하므로 signal 함수로 handler를 재설정해줬다. 부모 프로세스에서는 자식 프로세스의 process id를 받아서 waitpid 함수로 reaping 해주고 eval 함수를 리턴하여 다시 main의 루프 속으로 돌아갈 수 있도록 처리했다.

/bin/과 /usr/bin/ 경로는 PATH, PATH2 배열에 저장하여 입력받은 argv[0]에 더해 execve 함수를 실행할 때 사용되도록 구현하였다.

* **Phase2 (pipelining)**

사용자로부터 받은 입력에 ‘|’ 문자가 존재한다면 기존 명령어 처리 부분에서 pipe 부분으로 넘어간다. 이후 ‘|’ 문자를 기준으로 명령어를 parsing 해주는데, ‘|’ 문자를 발견할 때마다 개수를 기록한다. 이후 pipe\_command 함수를 실행한다.

pipe\_command 함수는 재귀적으로 구현했으며 앞의 명령어를 실행하여 출력한 fd를 다음 pipe\_command 호출 시에 넘겨주었다. 이후 마지막 명령어까지 실행된 이후에 waitpid로 reaping 후 사용자 입력 대기 상태로 돌아간다.

* **Phase3 (background process)**

사용자의 입력에 ‘&’이 존재하는지 확인한다. 만약 존재한다면 parseline 함수의 return 값이 1이 되어 background 처리 부분으로 넘어간다. 해당 command line, process id를 jobList에 추가하고 사용자의 입력을 받는 상태로 돌아가도록 구현했다. 또 foreground에서 작업이 진행중일 때 Ctrl+Z를 입력한다면 해당 process를 중지시키고 background로 보낸다.

bg 명령어를 사용하여 background에서 중지된 process를 다시 run 시킬 수 있으며, fg 명령어를 사용해 background의 process를 foreground에서 run 시킬 수 있다. 또한 kill 명령어를 사용해 background의 process를 종료시킬 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)**
2. **Phase 2 (pipeline)**
3. **Phase 3 (background)**