**Multicore Programming Project 1**

담당 교수 : 박성용

이름 : 엄석훈

학번 : 20181536

1. **개발 목표**

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

쉘의 기본적인 동작을 수행할 수 있도록 코드를 작성한다. 사용자로부터 명령어를 받아들여 명령어의 종류를 판단하여 해당 명령어가 기본적인 쉘의 명령어라면 자식 프로세스를 생성하여 명령을 수행한다. 만약 builtin 명령어라면 쉘이 직접 해당 명령어를 수행한다. 그리고 파이프된 명령어라면 파이프 단위로 명령어를 분해해서 앞선 명령어의 출력과 뒤의 명령어의 입력을 연결해주어 명령이 수행될 수 있도록 해준다. 마지막으로 백그라운드 명령어라면 백그라운드로 해당 명령어를 수행할 수 있도록 만들어준다. 그리고 추가로 모든 명령어에 대해 항상 history에 추가하여 관리할 수 있도록 코드를 작성한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Phase 1

쉘의 기본적인 명령어를 수행할 수 있게 된다. 예를 들어 ls, mkdir, rmdir, touch, cat, echo 등의 기본적인 명령어를 수행할 수 있게 한다. 사용자가 해당 명령어를 입력하면 자식 프로세스 생성하여 명령어를 수행시킨다. 그리고 cd, history, !!, !#, exit 의 명령어가 들어오면 자식 프로세스를 만들지 않고 쉘이 직접 명령어를 수행시킨다. history 명령어의 경우 지금까지 사용된 명령어를 모두 보여주며 !!명령어는 가장 최근 명령어를 다시 수행하며 !# 명령어는 #번째로 사용된 명령어를 다시 수행한다. 즉 Phase 1을 전부 구현한 결과 쉘의 기본적인 명령어들을 사용할 수 있다.

1. Phase 2

쉘의 기본적인 명령어들을 사용할 때 파이프를 이용해서 연결할 수 있게 해준다. 첫번째 명령어의 stdout 결과를 화면에 출력하지 않고 파이프로 연결된 다음 명령어의 stdin 입력으로 해주어 명령어를 수행시킨다. 그리고 만약 파이프가 여러 단계로 구성되어 있으면 각 앞뒤 명령어의 출력과 입력을 연결해서 명령어를 수행시켜준다. 또한 기본적인 명령어는 모두 그대로 사용할 수 있게 하고 history또한 그대로 동작하도록 해준다.

1. Phase 3

쉘의 명령어를 foreground뿐만 아니라 background로도 돌릴 수 있도록 해준다. 명령어 뒤이 &를 붙이면 자동으로 명령어를 background로 돌려준다. 그리고 foreground의 명령어를 ctrl+z로 forground의 명령어를 background 스탑 상태로 바꿀 수 있고 ctrl+c로는 종료할 수 있다. 그리고 background 스탑 상태의 프로세스를 다시 bg명령어로 background running상태로 바꾸고 다시 fg명령어로 foreground running상태로 바꿀 수 있다. 그리고 언제든지 jobs명령어를 통해 수행 생성된 모든 잡과 상태를 확인할 수 있고 kill 명령어를 통해서는 원하는 특정 job을 종료시킬 수 있다.

* 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

명령어가 들어와 builtin 명령어가 아니라면 자식 프로세스를 fork를 통해 생성한다. Fork한 뒤 return 값을 보고 0이면 자식 프로세스 임으로 명령어를 execve를 통해 수행시키고 부모 프로세스라면 자식 프로세스가 끝나길 기다린다.

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

자식 프로세스를 fork하고 나서 자식프로세스가 먼저 수행될 수 있도록 부모 프로세스는 wait를 하도록 한다. 그리고 자식 프로세스가 죽으면 wait하고 있던 부로 프로세스가 reaping을 한다.

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

가장 먼저 쉘에 사용자 입력이 들어오면 파이프의 개수를 확인한다. 그리고 파이프 명령어라면 가장 앞에 있는 파이프라인을 기준으로 명령어를 나누는 함수를 이용해서 파이프 명령어를 2개로 나눠준다. 그리고 파이프를 열고 해당 파이프를 각 명령어의 stdout과 stdin으로 연결한 뒤 각 명령어를 수행시켜주었다.

* + Pipeline 개수에 따라 어떻게 handling했는지에 대한 설명

파이프라인이 1개라면 파이프를 기준으로 명령어를 나누어 각각 명령어를 수행하였다. 그리고 그 이상이라면 파이프를 한 개씩 나누어서 파이프를 수행하는 함수로 보낸 뒤 만약 뒤의 명령어도 파이프로 연결되어 있다면 재귀함수 꼴로 다시 파이프를 나누고 나뉜 함수를 수행시키도록 해주었다.

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

명령어 뒤에 &가 붙어있는지 여부를 parseline 함수에서 판단해준 뒤 background로 돌려야 하면 bg=1로 return 해준다. 그리고 부모 프로세서에서 fork를 통해 자식 프로세스를 생성해 명령어를 수행시킨다. 그리고 부모 프로세서에서는 자식이 죽을 때 까지 wait하는 것이 아니라 바로 수행시킨 프로세스의 jobid와 pid를 출력하고 다시 shell의 명령어 프롬프트를 출력하도록 해준다.

* 1. **개발 방법**
* **B.의 개발 내용을 구현하기 위해 어느 소스코드에 어떤 요소를 추가 또는 수정할 것인지 설명. (함수, 구조체 등의 구현이나 수정을 서술)**

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

eval함수 내부에서 builtin 명령어가 아니라면 자식프로세스를 생성하여 명령어를 수행해야 한다. 따라서 pid=Fork()를 통해서 자식 프로세스를 생성하고 만약 pid에 0이 저장되면 자식 프로세스 임으로 execve함수를 통해서 자식 프로세스의 명령어를 수행해준다. 그리고 0이 아니라면 phase1에서는 무조건 foreground이기 때문에 Wait함수를 통해서 자식 프로세스가 죽을 때 까지 기다리게 된다.

-------------------------------------------------------------------------------------------

스크린샷, 텍스트, 폰트이(가) 표시된 사진

자동 생성된 설명

먼저 eval함수에서 명령어가 들어오면 파이프로 나뉜 명령어의 갯수를 세어서 pipe\_num에 적어준다. 즉 pipe\_num이 1이면 파이프된 명령어가 아니고 2이상이라면 그만큼 명령어가 있는 것이다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

그리고 다음으로 Fork를 통해 자식 프로세스가 생성되고 자식 프로세스에서는 split\_pie와 exec\_pipe 명령어를 통해서 파이프 명령어를 수행해준다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

split\_pipe함수에서는 argv에 저장되어 있는 명령어를 첫번째 | 문자 전까지 전부 first에 저장해준다. 그리고 | 명령어를 지우고 나머지 뒷부분을 second에 저장해주는 간단한 명령어이다.

그리고 아래 사진의 코드는 exec\_pipe함수이다. 우선 파이프 명령어끼리 입출력을 공유할 수 있도록 pipe명령어를 통해서 pipe명령어를 통해서 pipe를 만들어 준다. 그리고 Fork를 통해서 자식 프로세스를 생성한다. 자식 프로세스에서는 위에서 자른 first의 명령어의 stdout을 pipe에 연결하고 수행해준다. 그리고 second가 만약 하나의 명령어라면 first가 종료될 때 까지 기다리고 stdin을 위의 pipe와 연결한 뒤 명령어를 수행해준다. 하지만 만약 second가 다시 파이프된 명령어라면 똑같이 split\_pipe와 exec\_pipe명령어를 호출해서 다시 한번 파이프 명령어를 를 처리하도록 해준다. 즉 재귀적으로 파이프 명령어를 하나씩 처리하는 과정을 거쳐 어려 파이프로 연결된 명령어를 수행해준다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

-------------------------------------------------------------------------------------------

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

이는 phase3의 parseline함수에서 &문자를 처리하는 방법이다. &문자를 발견하면 bg를 1로 하고 &를 NULL로 바꿔서 evel함수에게 백그라운드 명령어임을 알려준다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

그리고 만약 bg=1이어서 background명령어라면 해당 프로세스의 jobid와 pid를 출력하고 eval함수를 종료한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

그리고 background에서 프로세스가 수행 가능하면 여러 프로세스가 동시에 수행될 수 있음으로 job을 만들어서 관리해주어야 하는데 따라서 job\_t라고 새로운 job 구조체를 만들어서 해당 구조체에서 job의 pid, jid, state, 명령어까지 모두 관리해준다.

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)**

**텍스트, 스크린샷, 도표, 평면도이(가) 표시된 사진

자동 생성된 설명**

1. **Phase 2 (pipeline)**

**텍스트, 도표, 스크린샷, 평면도이(가) 표시된 사진

자동 생성된 설명**

1. **Phase 3 (background)**

**텍스트, 스크린샷, 도표, 평면도이(가) 표시된 사진

자동 생성된 설명**