**System Programming Project 2**

담당 교수 : 이영민

이름 : 조준희

학번 : 20190328

1. **개발 목표**

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

이번 프로젝트의 목표는 기본적인 기능을 갖춘 쉘을 구현하는 것이다. phase1에서는 fork()와 exec() 함수를 통한 기본적인 명령어와 쉘 기능의 구현을 목표로 하며, phase2에서는 pipelining과 redirection을 구현하는 것을 목표로 한다. 마지막으로 phase3에서는 백그라운드 프로세스와 job control을 구현하는 것을 목표로 한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

1. Phase 1
   * Cd, ls, mkdir, rmdir, touch, cat, echo, exit 등 기본적인 쉘 명령어를 수행할 수 있는 쉘 프로그램을 구현한다.
2. Phase 2
   * Pipe와 redirection을 구현한 쉘 프로그램을 구현한다. 이 단계에서는 여러 명령어를 연결해 작업을 수행하고 터미널을 통한 입출력 외에도 파일 입출력 등 다양한 입출력을 지원하게 된다.
3. Phase 3
   * 백그라운드 프로세스의 실행 및 job control을 구현하여 jobs, bg, fg, kill 등의 명령어를 지원하고 여러 작업을 동시에 다루는 쉘 프로그램을 구현하게 된다.
   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
  + connection을 종료할 때 parent process에게 signal을 보내는 signal handling하는 방법 & flow

fork() 함수를 통해 child process를 만들게 되면, fork가 성공적으로 이루어졌다는 가정 하에 그 반환값은 자식 프로세스에서는 0이며 부모 프로세스에서는 자식 프로세스의 프로세스 ID이다. 자식 프로세스가 종료될 때 자동으로 SIGCHLD 신호가 호출되며 부모 프로세스에서는 waitpid()와 WIFSIGNALED() 함수를 통해 그 신호를 잡아낸다. Signal() 함수를 통해 SIGCHLD가 호출될 때 원하는 작업을 수행할 수 있도록 myshell\_SIGCHLD() 함수를 SIGCHLD 신호의 핸들러로 등록했다.

* **Phase2 (pipelining)**
  + Pipeline( ‘|’ )을 구현한 부분에 대해서 간략히 설명 (design & implementation)
  + Pipeline 개수에 따라 어떻게 handling했는지에 대한 설명

Pipeline을 1차 구분자로 삼아 전체 명령어를 pipeline을 기준으로 쪼개 각 부분을 문자열에 저장한 다음, 반복문을 통하여 Pipeline을 기준으로 분리된 각 명령어를 수행한다. 공백을 2차 구분자로 삼아 명령어를 파싱한 다음 execvp 함수를 통해 해당 명령어를 수행한다. 이때 두 개의 pipe인 prev\_pipe와 curr\_pipe를 통해 직전에 수행했던 명령어의 결과를 prev\_pipe[1]에 저장한 다음, 파일 디스크립터의 리다이렉션을 통해 그 다음 명령어가 curr\_pipe[0]을 통해 그 내용을 읽을 수 있도록 구현했다. 이후 prev\_pipe를 curr\_pipe로 덮어씌워 pipeline 개수와 무관하게 Pipelining을 수행할 수 있도록 구현했다.

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

파이프라인을 기준으로 하는 1차 파싱 이후 공백을 기준으로 각 명령어를 다시 파싱하는 과정에서 마지막에 ‘&’가 붙어있는 경우 해당 작업을 백그라운드 프로세스로 구분하고 명령어, 프로세스 ID, 프로세스 상태 등 이 작업에 대한 여러 정보를 작업 리스트에 추가한다. Myshell이 여러 작업을 다루게 하기 위해 프로세스를 잠시 멈추는 SIGSTOP/SIGTSTP, 프로세스를 강제로 종료시키는 SIGINT, 그리고 중지된 프로세스를 재개하는 SIGCONT 시그널에 대한 핸들러를 구현했다.

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

Phase 1은 명세서에 있는 pseudocode를 골격으로 삼아 구현하였다. 명령어 입력과 명령어 파싱은 각각 myshell\_readInput() 함수와 myshell\_parseInput() 함수로 수행하였으며, 명령어를 수행하는 부분은 myshell\_execCommand() 함수로 구현하였다. 명령어를 수행하기 위해 기본적으로 execvp() 함수를 사용하였으나, exit이나 cd처럼 자체적으로 구현해야 하는 명령어의 경우 명령어를 수행하는 myshell\_execCommand() 함수에서 fork()로 자식 프로세스를 생성하기 이전에 미리 처리했다. cd 명령어는 chdir() 함수를 통해 구현했으며, exit의 경우 \_exit()으로 구현했다.

Phase 2에서는 파이프를 지원하기 위해 pipe() 함수를 통한 파이프의 생성을 myshell\_execCommand() 함수에 추가했으며, myshell\_handleRedirection() 함수에서 리다이렉션 작업을 구현해 execvp() 실행 직전에 리다이렉션을 하도록 했다.

Phase 3에서는 백그라운드 프로세스와 작업 컨트롤을 지원하기 위해 작업에 대한 정보를 담고 있는 job\_t라는 구조체를 추가했으며, 이를 통해 작업 테이블 Jobs\_list 하에서 모든 작업을 관리할 수 있도록 여러 함수들을 구현했다. 또한 SIGINT나 SIGTSTP와 같은 신호들이 myshell 자체를 종료하지 않도록 Myshell\_SIGINT나 myshell\_SIGTSTP 같은 핸들러를 만들고, sigsetjmp/siglongjmp 함수를 통해 이 신호들이 myshell의 입력 루프로 돌아갈 수 있도록 했다.

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)**

**스크린샷, 텍스트, 도표, 디자인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.**

1. **Phase 2 (pipeline)**

**텍스트, 스크린샷, 도표, 천문학이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.**

1. **Phase 3 (background)**

스크린샷, 텍스트, 도표, 디자인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.