Operating Systems
Project #1

컴퓨터학부

2024년 3월 18일

작은 유니스 셸 tsh (tiny shell)

이 과제는 기능이 제한적인 작은 유닉스 셸을 구현하는 것이다. 과제의 목적은 프로세스 생성과 프로세스간 통신을 익히는 데 있다. 셸 구현을 통해 프로세스 생성, 명령어 실행, 표준 입출력 변경, 파이프를 통한 프로세스간 통신을 배운다.

셸의 구성

일반적으로 셸은 사용자로부터 명령어를 입력 받아 실행한다. 실행이 끝나면 다음 명령어를 입력 받아 실행한다. 셸은 이 과정을 사용자가 `exit`이라는 명령어를 입력할 때까지 반복한다. 셸은 명령어를 읽어오기는 하지만 직접 실행하지 않고 자식 프로세스를 생성하여 대신 실행하게 한다. 아래 명령어는 파일 `tsh.c`를 화면에 출력하는 것으로 자식 프로세스가 작업을 마칠 때까지 부모 프로세스는 기다린다.

```
tsh> cat tsh.c
```

반면에 부모 프로세스는 자식이 종료될 때까지 기다리지 않고 다음을 진행할 수 있는데, 우리는 이런 경우를 백그라운드 실행이라고 한다. 명령어를 백그라운드로 실행하려면 끝에 '&' 기호를 붙여준다. 아래 명령어는 앞 선 명령어를 백그라운드로 실행한다.

```
tsh> cat tsh.c &
```

명령어를 백그라운드로 실행하면 부모와 자식 프로세스는 병행으로 진행된다. 부모 프로세스는 자식 프로세스의 종료 유무와 관계없이 다음 명령어를 입력 받아서 실행할 수 있다. 백그라운드로 실행된 자식 프로세스는 부모가 기다리지 않기 때문에 끝나면 좀비로 남는다. 좀비 프로세스는 실행을 마쳤기 때문에 모든 자원을 반납한다. 하지만 프로세스 테이블 엔트리는 유지한 채 부모를 만나거나 커널로 입양될 때까지 계속 그 상태로 남아 있게 된다. 좀비 프로세스는 아무 것도 하지 않으면서 프로세스 테이블 공간을 차지하고 있기 때문에 제거하는 것이 좋다. 일반적으로 셸은 다음 명령어를 실행하기 전에 좀비 프로세스가 있는지 검사하여 제거한다.

자식 프로세스와 명령어 실행

자식 프로세스는 `fork()` 함수를 사용하여 생성한다. 명령어 실행은 부모가 하지 않고 자식이 한다. 자식이 명령어를 실행하는 동안 부모는 다음 명령어를 입력 받는다. 명령어 실행은 `exec` 계열의 함수인 `execl()`, `execlp()`, `execle()`, `execv()`, `execvp()`, `execvpe()`를 사용한다. 예를 들어 명령어 `ps -a -el`을 실행하려면 다음과 같이 인자 배열을 설정하고 `execvp()` 함수를 호출할 수 있다.

```
argv[0] = "ps";
argv[1] = "-a";
argv[2] = "-el";
argv[3] = NULL;
execvp(argv[0], argv);
```

execvp()를 포함한 exec 계열 함수의 자세한 용법은 매뉴얼 페이지를 참조한다. exec 계열의 함수는 프로세스의 메모리 공간을 새로운 프로그램을 덮어 씌우기 때문에 호출한 곳으로 다시 돌아오지 않는다.

표준 입출력 리다이렉션

입출력을 키보드나 화면이 아닌 파일에서 받거나 보내는 것을 표준 입출력 리다이렉션이라고 한다. 표준 입출력을 파일로 바꾸려면 명령어에서 기호 '>'와 '<'를 사용한다. 셸에서 사용되는 리다이렉션의 기호와 종류는 이것 말고도 여러가지가 있다. 이 과제에서는 다음과 같은 세 가지만 고려한다.

- command < file: 입력을 표준 장치인 키보드에서 읽어 오지 않고 파일에서 읽어 온다. 이를 위해 먼저 open() 함수를 사용하여 읽어 올 파일을 연다. 이 때 열린 파일의 서술자가 fd라면 아래와 같이 dup2() 함수를 사용하여 바꾼다.

  ```
  dup2(fd, STDIN_FILENO);
  ```

  표준 입력을 fd와 공유한다는 뜻으로 서술자 STDIN_FILENO는 더 이상 키보드가 아님을 의미한다.

- command > file: 출력을 표준 장치인 화면으로 보내지 않고 파일에 저장한다. 이를 위해 먼저 open() 함수를 사용하여 저장할 파일을 생성한다. 이 때 열린 파일의 서술자가 fd라면 아래와 같이 dup2() 함수를 사용하여 바꾼다.

  ```
  dup2(fd, STDOUT_FILENO);
  ```

  표준 출력을 fd와 공유한다는 뜻으로 서술자 STDOUT_FILENO는 더 이상 화면이 아님을 의미한다.

- command < file1 > file2: 입출력을 표준 장치가 아닌 파일에서 읽고 파일에 저장한다. 앞에서 설명한 두 가지 방식을 다 사용한다.

파이프

파이프는 프로세스간 통신을 하기 위한 수단으로 셸에서는 앞 명령의 출력을 다음 명령의 입력으로 만들기 위해 사용한다. 예를 들어 다음 명령어는 ls -l의 결과를 화면에 출력하지 않고 less 명령의 입력으로 사용함을 뜻한다.

```
tsh> ls -l | less
```

파이프의 기본 형태는 아래와 같다.

```
command_1 | command_2
```

이 명령을 실행하기 위해 셸은 자식 프로세스를 생성한다. 자식 프로세스가 파이프 명령을 실행할 동안 셸 자신은 기다리거나 다음을 진행한다. 자식 프로세스는 파이프를 생성한 다음 손자 프로세스를 생성한다. 손자 프로세스는 command1을 실행하고, 자식 프로세스는 command2를 실행한다. 파이프는 자식과 손자가 통신하기 위한 것으로 command1의 출력을 command2의 입력으로 보낸다.

파이프는 아래와 같이 여러 단계로 구성될 수 있다.

```
command_1 | command_2 | ... | command_n
```

이런 경우 command_2 | ... | command_n을 command_x로 간주하고 두 명령으로 구성된 파이프를 수행한다.

```
command_1 | command_x
```

즉, 파이프 하나를 생성하여 command_1의 출력을 command_x의 입력으로 사용한다. 만일 command_x에 파이프가 있으면 같은 방법으로 분할해서 처리한다. command_x에 파이프 명령이 없을 때까지 이 과정을 반복한다.

골격파일

골격파일 tsh.skeleton.c에는 기본적인 명령을 실행할 수 있는 코드가 완성되어 있다. 아래 단락에 있는 검증 1단계가 완성되어 있다는 뜻이다. 학생들은 나머지 기능인 표준 입출력 리다이렉션과 파이프 기능을 추가한다. 파일을 변경하게 되면 머리 부분에 변경한 사람의 정보와 날짜, 변경 내용 등을 기록한다. 골격파일과 함께 컴파일 과정을 쉽게 해주는 Makefile도 제공한다.

검증

프로그램이 완성되면 다음 단계별로 검증을 수행한다. 단계별로 예시된 명령을 순서대로 실행하고 그 결과를 캡쳐하여 제출한다. 예시에는 없지만 추가로 실행한 명령과 결과를 캡쳐해서 제출할 수 있다. 다만 검증할 때 실행화일과 소스파일을 같은 폴더에 두고 시작한다.

1. 기본 명령

```
tsh> grep int tsh.c
tsh> grep "if.*NULL" tsh.c &
tsh> ps
```

2. 표준 입출력 리다이렉션

```
tsh> grep "int " < tsh.c
tsh> ls -l
tsh> ls -l > delme
tsh> cat delme
tsh> sort < delme > delme2
tsh> cat delme2
```

3. 파이프

```
tsh> ps -A | grep -i system
tsh> ps -A | grep -i system | awk '{print $1,$4}'
tsh> cat tsh.c | head -6 | tail -5 | head -1
```

4. 조합

```
tsh> sort < tsh.c | grep "int " | awk '{print $1,$2}' > delme3
```

명령어 파싱과 오류

이 과제는 프로세스 생성과 프로세스간 통신을 이해하는 것이 주요 목적이다. 입력된 명령어를 파싱해서 형식에 어긋나는 명령어 오류를 유연하게 처리하는 것은 필요하지만 복잡한 오류는 고려하지 않아도 무방하다. 그러나 연속된 공백문자를 하나의 공백문자로 인식하는 것은 필요하다. 골격파일 `tsh.skeleton.c`에는 명령어에 있는 공백문자와 큰 따옴표와 작은 따옴표로 이루어진 문자열을 처리하는 코드가 구현되어 있다.

제출물

`tsh`이 잘 설계되고 구현되었다는 것을 보여주는 자료를 보고서 형식으로 작성한 후 PDF로 변환하여 `이름_학번_PROJ1.pdf`로 제출한다. 여기에는 다음과 같은 것이 반드시 포함되어야 한다.

- 본인이 작성한 함수에 대한 설명
- 컴파일 과정을 보여주는 화면 캡처
- 실행 결과물에 대한 상세한 설명
- 과제를 수행하면서 경험한 문제점과 느낀점
- 프로그램 소스파일 (`tsh.c`) 별도 제출
- 프로그램 실행 결과 (`tsh.txt`) 별도 제출

평가

- Correctness 50%: 프로그램이 올바르게 동작하는 지를 보는 것입니다. 여기에는 컴파일 과정은 물론, 과제가 요구하는 기능이 문제없이 잘 작동한다는 것을 보여주어야 합니다.
- Presentation 50%: 자신의 생각과 작성한 프로그램을 다른 사람이 쉽게 이해할 수 있도록 프로그램 내에 적절한 주석을 다는 행위와 같이 자신의 결과를 잘 표현하는 것입니다. 뿐만 아니라, 프로그램의 가독성, 효율성, 확장성, 일관성, 모듈화 등도 여기에 해당합니다. 이 부분은 상당히 주관적이지만 그러면서도 중요한 부분입니다. 컴퓨터과학에서 중요하게 생각하는 best coding practices를 참조하기 바랍니다.

HK