[SWE2024: System Programming Experiment – Spring 2023]

Programming Assignment #2

Due: 17th May. (Wed), 23:59:59

1. Introduction

리눅스 환경에서 동작하는 `Mini shell` 을 제작한다. 이 과제의 주요 목표는 수업 전반기에 배운 주제인 파일, 프로세스, 시그널, IPC 를 다루는 프로그램을 제작하는 것이다.

2. Problem specification

Shell 은 사용자의 1) 입력을 받아, 2) 해석하고, 3) 명령을 실행하는 프로그램이다. `Mini shell` 은 아주 작은 기능만 구현한 shell 로 정해진 특정 명령어를 실행시킬 수 있으며, input/output redirection, pipeline 기능을 지원한다.

다음 표는 사용자의 입력이 될 수 있는 문자열을 BNF(Backus-Naur Form)와 유사한 형태로 기술한 것이다. (이하, <표>라고 표현)

```
commandline   ::=   pipeline
pipeline      ::=   commands                                  or
                    pipeline | commands
commands      ::=   command                                   or
                    command < filename                        or
                    command > filename                        or
                    command >> filename                       or
                    command < filename > filename             or
                    command < filename >> filename
command       ::=   executable [option(s)] [argument(s)]      or
                    builtin_cmd [argument]
executable    ::=   { ls, man, grep, sort, awk, bc, head,
                    tail, cat, cp, mv, rm, pwd }              or
                    path
builtin_cmd   ::=   { cd, exit }
path          ::=   { pathname with leading "./" }
```

<표>에서 사용한 `[filename]`은 임의의 파일명을 뜻하고, `[option(s)]`는 해당 프로그램의 옵션(들), 그리고 `[argument(s)]`는 해당 프로그램의 인자(들)을 의미한다.

|	Pipeline
<	Input redirection
>	Output redirection
>>	Output redirection (appending)

각 기호의 기능은 Pipe 수업 시간에서 배운 것과 같고, 간략하게 다시 설명하면 다음과 같다.

- Pipeline: 앞 프로세스의 표준 출력을 뒤 프로세스의 표준 입력으로 **파이프**를 통해 연결
- Input redirection: 해당 프로세스의 표준 입력(fd 0)을 [filename] 파일로 대체
- Output redirection: 해당 프로세스의 표준 출력(fd 1)을 [filename] 파일로 대체
  - > 으로 redirection 되었을 경우, 해당 파일을 초기화하고 새로 쓴다.
  - >> 으로 redirection 되었을 경우, 해당 파일의 뒤에서부터 덧붙인다.

또한, 구현의 편의를 위해 **<표>의 각 키워드 사이는 무조건 한 칸의 space로 구분된다**고 가정한다. 만약 입력이 command < filename의 경우, [command] 와 '<' 문자 사이엔 한 칸의 space, '<' 와 filename 사이에도 한 칸의 space로 구성된다.

2.1. Interpreting commandline

사용자는 200바이트 이내의 문자열을 입력한다. 만약 사용자의 입력이 `commandline`으로부터 도출(derive)할 수 있는 문자열이면 유효하다. 예를 들면, 다음 명령의 경우,

```
cd dir3
```

- `commandline` ➔ `pipeline` ➔ `commands` ➔ `command` ➔ `builtin_cmd [argument]`
  - `builtin_cmd` ➔ `cd`
  - `[argument]` ➔ `dir3`

순으로 해석될 수 있어 유효하다.

다음 명령 또한 commandline으로부터 유도될 수 있어 유효하다.

```
ls -al /etc | sort -r > ls_sorted.txt
```

하지만, 다음 명령은, `less`라는 프로그램을 `commandline`에서 유도할 수 없어 (어느 command에도 포함되어 있지 않음) 유효하지 않다.

```
ls -al /etc | less
```

<표>의 `path`의 경우, shell에서 현재 디렉토리 내의 프로그램을 실행하는 것을 의미한다. 현재 디렉토리의 `a.out` 파일을 실행시킨 땐 통상적으로 다음 명령을 사용한다.

```
./a.out
```

즉, `command`가 "./" 로 시작하는 문자열인 경우, `executable`에서 유도된다.
(과제의 단순화를 위해, 절대경로 등의 방식은 고려하지 않는다. 또한, 홈 디렉토리를 의미하는 '~' 또한 사용하지 않는다.)

입력이 유효할 경우 **입력을 평가(2.2. Evaluating commandline)**하고, **유효하지 않은 경우** 다음과 같은 에러 메시지를 **표준 에러**(fd 2)에 출력한다.

```
mini: command not found
```

2.2. Evaluating commandline

입력 값을 평가하는 과정을 구현하는 방식은 일반적으로 다음과 같다.

1) 명령어의 개수 파악 (e.g., pipeline 등으로 명령어들이 연결된 경우)
2) Input/output redirection이 사용되었는지 파악
3) 명령어 종류를 분석하여,
   A. 직접 구현한 경우, (필요하다면) 자식 프로세스를 생성하여 해당 루틴을 호출
   B. 구현하지 않은 경우, 자식 프로세스를 생성하여 해당 프로그램을 실행
4) 자식 프로세스를 생성했을 경우, 종료될 때까지 기다림

2.2.1. Input/output redirection

Shell의 명령으로 redirection이 사용될 경우, 사용자가 명시한 파일이 존재하는지, 혹은 존재하더라도 해당 파일이 정상적인 파일인지 (디렉토리가 아닌지), 해당 파일을 사용할 권한이 있는지 등을 확인할 필요가 있다. 하지만, `Mini shell`에선 아래 다음과 같은 경우만 파악한다.

만약 input redirection으로 명시된 파일이 존재한다면, 무조건 읽기 권한이 있는 정상적인 파일이라고 가정한다. 존재하지 않으면 다음과 같은 문자열을 출력한다.

```
mini: No such file or directory
```

만약 아래와 같이 4개의 명령이 파이프로 연결되었다고 생각하자.

```
A | B | C | D
```

Input redirection은 제일 앞의 A 프로세스에서만 존재할 수 있고, output redirection는 제일 뒤의 D 프로세스에서만 존재할 수 있다. (B와 C에서 redirection이 존재하는지 검사할 필요가 없음)

```
A < file1 | B | C | D > file2
```

2.2.2. Command

Mini shell에서 처리할 명령은 크게 2가지 종류로 구분된다.

- `executable`: 해당 프로그램을 로드(fork - exec*)해야 하는 프로그램
- `builtin_cmd`: 직접 구현하는 `Mini shell` built-in command

`executable`에 해당하는 명령은, 옵션이나 인자를 추출해 자식 프로세스를 생성하여 exec계열 함수를 통해 해당 바이너리를 실행한다. (바이너리 실행 경로는 which 명령어를 이용하여 찾는다)

builtin_cmd는 Mini shell 내부에서 동작하도록 구현해야 한다.
executable 중 구현해야 하는 command에 대해서는 아래 3장에서 각 command의 기능을 정의하고 필요한 시스템 콜/라이브러리 함수를 언급할 것이다.

2.3. Process group

Mini shell이 새로운 자식 프로세스를 생성할 경우, **해당 프로세스를 새로운 process group에 속하도록 한다**. (해당 자식 프로세스의 process group ID는 자신의 process ID와 일치하도록 만든다)
만약, 파이프의 사용으로 인해 두 개 이상의 프로세스를 생성할 경우, 모든 프로세스는 제일 먼저 생성한 프로세스의 그룹에 속하도록 한다. (프로세스의 process group ID는 제일 먼저 생성한 자식 프로세스의 process ID와 일치해야 함)

2.4. Reaping child processes

Mini shell이 자식 프로세스를 생성했다면, wait계열 시스템 콜을 이용해 자식 프로세스를 확인하고 제거하여 좀비 프로세스가 생기지 않도록 한다.

프로세스가 실행 도중 Ctrl+Z 키를 통해 생성하는 SIGTSTP 시그널을 수신하면, 일반적인 경우 프로세스 실행을 멈추고 중단된다. 동시에 부모 프로세스는 SIGCHLD 시그널을 수신하고, 이 때 부모가 다음과 같이 waitpid를 호출해서 얻는 status 값에 WIFSTOPPED 매크로 함수를 실행하면 true가 반환된다. (waitpid 시스템 콜의 옵션 WUNTRACED)

```
waitpid(-1, &status, WNOHANG | WUNTRACED)
```

(참고: $ man 2 waitpid)

만약 자식 프로세스가 중지되었을 경우 (WIFSTOPPED가 true일 때), 해당 자식이 속한 **프로세스 그룹**에 SIGKILL 시그널을 보내 모든 연관 프로세스를 강제로 종료시킨다.

2.5. Signals

Mini shell은 SIGINT, SIGTSTP 시그널을 수신해도 종료되지 않는다.
(프로그램의 종료는 3.9 exit 을 참고한다)

3. Programs

`executable` 및 `builtin_cmd` 중 구현해야 하는 command는 다음과 같다. 구현한 `executable`은 Makefile을 통해 make를 수행하면 모두 컴파일 되어 실행 바이너리가 생성되어야 한다.

3.1. head

SYNOPSYS

`head [OPTION] [file]`

DESCRIPTION

`file` 파일의 위에서부터 10 라인을 표준 출력으로 출력한다.

-n K

10 라인 대신 K 라인을 출력한다.

`file` 은 항상 존재한다고 가정한다.

3.2. tail

SYNOPSYS

`tail [OPTION] [file]`

DESCRIPTION

`file` 파일의 아래에서부터 10 라인을 표준 출력으로 출력한다.

-n K

10 라인 대신 K 라인을 출력한다.

`file` 은 항상 존재한다고 가정한다.

3.3. cat

SYNOPSYS

`cat [file]`

DESCRIPTION

`file` 파일을 표준 출력으로 출력한다.

`file` 은 항상 존재한다고 가정한다.

3.4. cp

SYNOPSYS

`cp file1 file2`

DESCRIPTION

`file1` 파일의 복사본 `file2` 파일을 만든다.

인자가 입력되지 않는 경우, cp: missing file operand라는 메시지를 출력한다.

인자가 1개만 입력되는 경우, cp: missing destination file operand after 'FILE1'이라는 메시지를 출력한다.

`file1` 은 항상 존재한다고 가정한다.

3.5. mv

SYNOPSYS

`mv source destination`

DESCRIPTION

`source` 파일의 이름을 `destination`으로 바꾸거나, `source` 파일의 경로를 `destination`으로 옮긴다.

인자가 입력되지 않는 경우, mv: missing file operand라는 메시지를 출력한다.

인자가 1개만 입력되는 경우, mv: missing destination file operand after 'SOURCE'라는 메시지를 출력한다.

SEE ALSO

`rename(2)`

3.6. rm

SYNOPSYS

`rm file`

DESCRIPTION

`file` 파일을 삭제한다.

SEE ALSO

`unlink(2)`

3.7. cd

SYNOPSYS

`cd dir`

DESCRIPTION

현재 working directory를 `dir`로 변경한다.

SEE ALSO

`chdir(2)`

3.8. pwd

SYNOPSYS

`pwd`

DESCRIPTION

현재 working directory를 표준 출력으로 출력한다.

SEE ALSO

`getcwd(3)`

3.9. exit

SYNOPSYS

`exit [NUM]`

DESCRIPTION

표준 에러로 `exit`이라는 문자열을 출력한 뒤, `Mini shell`을 종료한다.

`NUM`이 명시되었을 경우 프로그램의 종료 값으로 `NUM`을, 아닐 경우 `0`을 반환한다.

SEE ALSO

`exit(3)`

3.10. Errors

`head, tail, cat, cp, mv, rm, cd` 명령 수행 중 에러가 발생한 경우, 에러의 종류에 따라 아래 표에 명시된 메시지를 다음과 같이 조합하여 표준 에러로 출력한다.

`EACCES`	Permission denied
`EISDIR`	Is a directory
`ENOENT`	No such file or directory
`ENOTDIR`	Not a directory
`EPERM`	Operation not permitted
`Other errors`	Error occurred: <ERRNO> (에러 번호 출력)

```
command: ERROR_MESSAGE
e.g.)
mv: Permission denied
cd: Not a directory
```

4. Grading policy

- 각 구현에 따른 배점은 다음과 같다. (총 100 점)
  - `executable` 동작 여부 44 점
    - `[ls, man, grep, sort, awk, bc]` 동작 여부 16 점
    - `[head, tail, cat, cp, mv, rm, pwd]` 구현 및 동작 여부 26 점
    - path 상의 임의의 바이너리 동작 여부 2 점
  - `builtin_cmd` 구현 및 동작 여부 6 점
  - Pipe and redirection 기능 구현 및 동작 여부 40 점
  - `SIGINT, SIGTSTP` 으로 `Mini shell` 이 종료되거나 멈추지 않는 지 여부 5 점
  - 보고서 5 점
- 지각 제출은 매 24시간마다 **10점**씩 감점된다.
- **만약 Mini shell에서 생성한 자식 프로세스의 process group ID가 Mini shell의 process group ID와 동일할 경우, 해당 제출은 채점되지 않는다.**
  - 이는 하나의 터미널에서 Mini shell을 실행시키고 `commandline`을 입력한 후에, 다른 터미널에서 다음의 명령어 입력을 통해 확인할 수 있다.

```
$ ps fo user,pid,pgid,cmd | grep pa2
```

- Makefile을 작성하여 같이 제출하여야 하며, make 수행 시, **"pa2"**라는 이름의 바이너리가 생성되도록 하고, 구현한 `executable`에 대한 바이너리도 같이 생성한다.
- Makefile과 *.c파일들을 pa2폴더에 넣고, submit 커맨드를 활용하여 제출한다.

```
$ ~swe2024-41_23s/bin/submit pa2 pa2
```

5. Restrictions

- 본 과제는 개인 과제이다.
- 과제 구현을 위해 지금까지 배운 리눅스 시스템 콜/라이브러리 함수를 이용한다.
- 과제 수행 시 system() 함수를 사용할 수 없다.
- 어떤 자원을 동적으로 할당 받았다면, 프로그램 종료 전에 반드시 해제해야 한다.
  - 여기서 자원이란 파일, 메모리, 자식 프로세스를 뜻한다.
- **제출한 코드가 컴파일 되지 않거나 위의 예시와 같이 동작하지 않을 경우, 해당 제출은 채점되지 않는다**
- 프로그램 코드와 별도로, 디자인 및 구현에 대한 내용을 담은 보고서를 iCampus 에 제출한다. 보고서의 파일 포맷은 pdf 로써, 이름은 [studentID_report_pa2.pdf]로 지정한다.