System Programming

Exercise

Term project – shell 기본 동작과 main loop

요구사항

- 설계 및 구현해야 할 smsh는 다음과 같은 명령어를 처리할 수 있어야 한다.
  - foreground and background execution (&)
  - multiple commands separated by semicolons
  - history command
  - shell redirection (>, >>, >|, <)
  - shell pipe (ls –la | more)
  - Multiple pipe (ls | grep “^d” | more)
  - cd command
- Deadline
  - 학기말까지

전면 처리와 후면 처리

- 전면 처리
  - 입력된 명령어를 전면에서 실행하고
    쉘은 명령어 실행이 끝날 때까지 기다림

  $ 명령어

- 후면 처리
  - 명령어를 후면에서 실행하고
    전면에서는 다른 작업을 실행하여
    동시에 여러 작업을 수행할 수 있음

  $ 명령어 &

```
[[Eunui-Macmini:~] hasoo% (sleep 100; echo done)&
 [1] 42082
[[Eunui-Macmini:~] hasoo% find . -name test.c -print &
 [2] 42085
```

- 요구사항
- 쉘 기능 소개
- smsh
  main loop

전면 처리와 후면 처리

- 후면 작업 확인
  - 후면에서 실행되고 있는 작업들을 나열함
  - 작업 번호를 명시하면 해당 작업만 표시함

  $ jobs [%작업번호]

```
ubuntu@server:~$ (sleep 100; echo done)&
[1] 6151
ubuntu@server:~$ sudo find ./ -name test.cpp -print &
[2] 6168
ubuntu@server:~$ jobs
[1]-  Running                 ( sleep 100; echo done ) &
[2]+  Stopped                 sudo find ./ -name test.cpp -print
ubuntu@server:~$ jobs %1
[1]-  Running                 ( sleep 100; echo done ) &
```

- 후면 작업을 전면 작업으로 전환하기
  - 작업번호에 해당하는 후면 작업을 전면 작업으로 전환시킴

  $ fg [%작업번호]

```
ubuntu@server:~$ (sleep 100; echo done)&
[4] 6769
ubuntu@server:~$ fg %4
( sleep 100; echo done )
```

여러 개의 명령어 사용하기

- 명령어 열 (command sequence)
  - 나열된 명령어들을 순차적으로 실행한다.

  $ 명령어1; …; 명령어n

```
ubuntu@server:~$ date; pwd; ls
Fri May  1 18:02:44 KST 2020
/home/ubuntu
a.c  Desktop    err.txt           list1.txt  list.txt   Pictures
b.c  Documents  examples.desktop  list2.txt  Music      Public
c.c  Downloads  fontconfig        list3.txt  names.txt  snap
```

여러 개의 명령어 사용하기

- 명령어 그룹 (command group)
  - 나열된 명령어들을 하나의 그룹으로 묶어 순차적으로 실행한다.

$ (명령어1; …; 명령어n)

```
ubuntu@server:~$ date; pwd; ls > out1.txt
Fri May  1 18:08:35 KST 2020
/home/ubuntu
ubuntu@server:~$ (date; pwd; ls) > out2.txt
ubuntu@server:~$ cat out2.txt
Fri May  1 18:08:48 KST 2020
/home/ubuntu
a.c
b.c
c.c
Desktop
```

여러 개의 명령어 사용하기

- 이전에 사용했던 명령을 모두 보려면 history 명령을 입력
- 다시 사용하려면 !와 앞의 번호를 입력
- 방향키 ↑와 방향키 ↓를 사용하면 명령을 입력했던 순서대로 보여주며 선택하려면 Enter를 누르면 됨 (추가 점수)

- 예를 들어 17번 ls명령을 다시 수행하려면 # !17 입력

```
root@server:/etc/systemd# !17
ls
journald.conf   network         system
logind.conf     resolved.conf   system.conf
root@server:/etc/systemd#
```

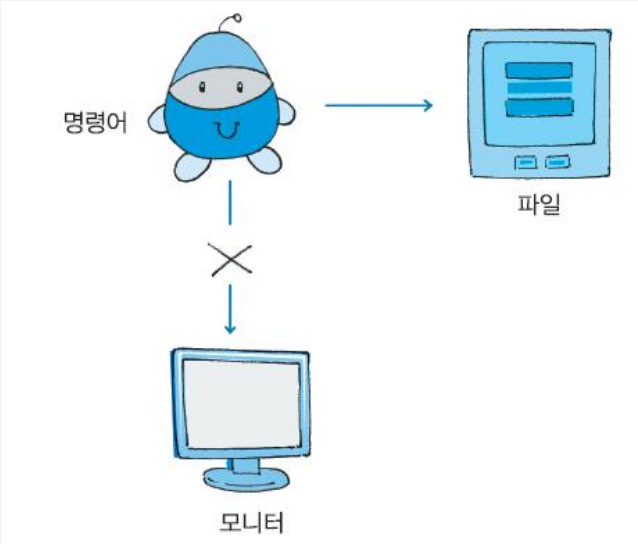
```
root@server:/etc/systemd# history
    1  passwd
    2  whoami
    3  exit
    4  gedit /etc/gdm3/custom.conf
    5  gedit /etc/pam.d/gdm-password
    6  gedit /etc/pam.d/gdm-autologin
    7  gedit /root/.profile
    8  reboot
    9  exit
   10  cd /lib/systemd/system/
   11  pwd
   12  ls
   13  clear
   14  cd
   15  clear
   16  cd /lib/systemd/
   17  ls
   18  clear
   19  cd /etc/systemd/
   20  clear
   21  history
root@server:/etc/systemd#
```

입출력 재지정

- 출력 재지정 (output redirection)
  - 명령어의 표준 출력을 모니터 대신에 파일에 저장

  $ 명령어 > 파일

```
ubuntu@server:~$ who > names.txt
ubuntu@server:~$ cat names.txt
ubuntu   :0           2020-05-01 17:08 (:0)
ubuntu@server:~$ ls / > list.txt
ubuntu@server:~$ cat list.txt
bin
boot
cdrom
dev
etc
home
initrd.img
initrd.img.old
lib
```



입출력 재지정

- 출력 재지정 활용법: 간단한 파일 만들기
  - 표준 입력 내용을 모두 파일에 저장한다.
  - 만일 파일이 없으면 새로운 파일을 생성한다.

  $ cat > 파일

  - 내용 작성 후 ctrl + d를 눌러서 입력을 마친다.

```
ubuntu@server:~$ cat > list1.txt
Hi!
This is the first file.
ubuntu@server:~$ cat > list2.txt
Hello!
This is the second file.
```

```
ubuntu@server:~$ cat list1.txt
Hi!
This is the first file.
ubuntu@server:~$ cat list2.txt
Hello!
This is the second file.
```

입출력 재지정

- 출력 재지정 활용법: 두 파일을 붙여서 새로운 파일 만들기
  - 파일1과 파일2의 내용을 붙여서 새로운 파일3을 만든다.

  $ cat 파일1 파일2 > 파일3

```
ubuntu@server:~$ cat list1.txt list2.txt > list3.txt
ubuntu@server:~$ cat list3.txt
Hi!
This is the first file.
Hello!
This is the second file.
```

- 출력내용 추가하기
  - 명령어의 표준 출력을 모니터 대신 파일에 추가한다. (이어 쓰기)

  $ 명령어 >> 파일

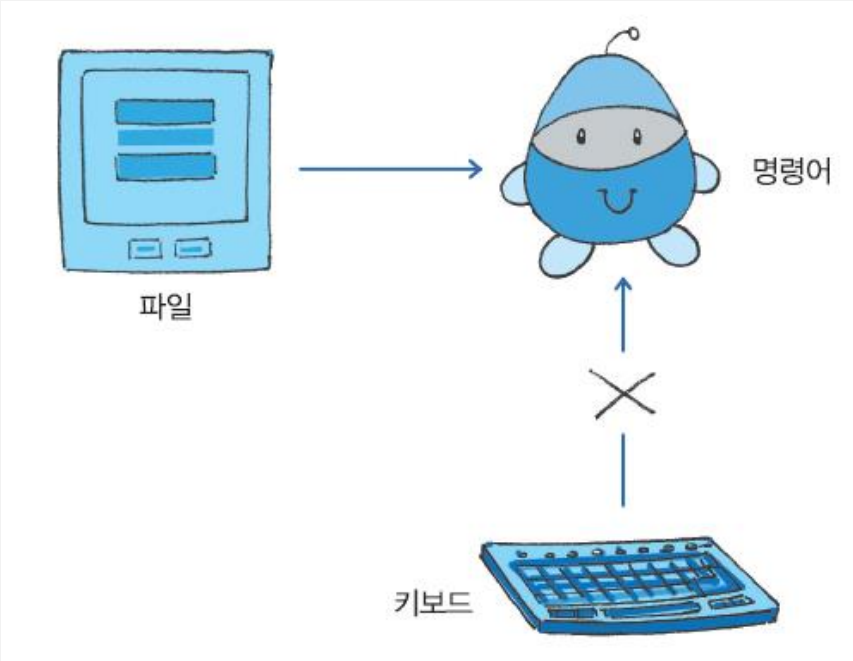
```
ubuntu@server:~$ date >> list1.txt
ubuntu@server:~$ cat list1.txt
Hi!
This is the first file.
Fri May  1 19:30:34 KST 2020
```

입출력 재지정

- 입력 재지정 (input redirection)
  - 명령어의 표준 입력을 키보드 대신 파일에서 받는다.

$ 명령어 < 파일

```
ubuntu@server:~$ wc < list1.txt
 3 12 57
```



입출력 재지정

- 문서 내 입력 (here document)
  - 해당 단어가 나타날 때까지의 내용을 입력으로 받는다.

$ 명령어 << 단어

…

단어

```
ubuntu@server:~$ wc << END
> hello!
> Can ya here me up there?
> END
 2  7 32
```

입출력 재지정

- 오류 재지정
  - 명령어의 표준오류를 모니터 대신 파일에 저장한다.

  $ 명령어 2> 파일

- 명령어의 실행 결과
  - 표준출력(standard output): 정상적인 실행에 대한 출력
  - 표준오류(standard error): 비정상적인 실행에 대한 출력

```
ubuntu@server:~$ ls -l /bon/usr 2> err.txt
ubuntu@server:~$ cat err.txt
ls: cannot access '/bon/usr': No such file or directory
```

입출력 재지정

- >와 >|의 차이점
  - >는 쉘의 noclobber 옵션 설정 시 덮어쓰기 불가
  - >|는 noclobber 옵션이 설정되어 있더라도 덮어쓰기 가능
  - set [+|-]o noclobber 또는 set [+|-]C 로 설정 또는 설정 취소 가능

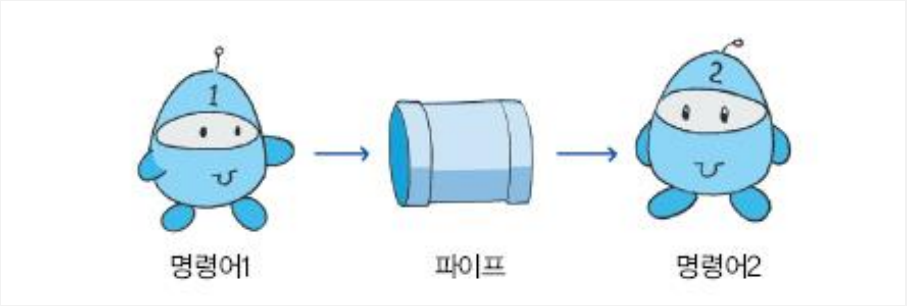
```
runner@repl.it:~$ echo hasoo1 > test
runner@repl.it:~$ cat test
hasoo1
runner@repl.it:~$ echo hasoo2 >| test
runner@repl.it:~$ cat test
hasoo2
runner@repl.it:~$ set -o noclobber
runner@repl.it:~$ echo hasoo3 > test
bash: test: cannot overwrite existing file
runner@repl.it:~$ cat test
hasoo2
runner@repl.it:~$ echo hasoo4 >| test
runner@repl.it:~$ cat test
hasoo4
runner@repl.it:~$ set +o noclobber
runner@repl.it:~$ echo hasoo5 > test
runner@repl.it:~$ cat test
hasoo5
```

입출력 재지정

- 파이프
  - 명령어1의 표준출력이 파이프를 통해 명령어2의 표준입력으로 사용된다.

$ 명령어1 | 명령어2

```
ubuntu@server:~$ who > names.txt
ubuntu@server:~$ sort < names.txt
ubuntu   :0           2020-05-01 17:08 (:0)
ubuntu@server:~$ who | sort
ubuntu   :0           2020-05-01 17:08 (:0)
```



- 요구사항
- 쉘 기능 소개
- smsh
  main loop

smsh main loop (1/3)

```
/* read command line until "end of file" */
while (read(stdin, buffers, numberchars)) {
  /* parse command line */
  if (/* command line contains & */)
      amp = 1;
  else
      amp = 0;

  if (fork() == 0) {
    if (/* redirect output */) {
       fd = creat(newfile, fmask);
       close(stdout);
       dup(fd);
       close(fd);
       /* stdout is now redirected */
    }
```

- 요구사항
- 쉘 기능 소개
- smsh
  main loop

smsh main loop (2/3)

```
if (/* piping */) {
   pipe(files);
   if ( fork() == 0 ) {
       /* first component of command line */
       close(stdout);
       dup(files[1]);
       close(files[1]);
       close(files[0]);
       /* stdout now goes to pipe */
       /* child process does command1 */
       execlp(command1, command1, 0);
    }
```

- 요구사항
- 쉘 기능 소개
- smsh
  main loop

smsh main loop (3/3)

```
        /* 2nd command of command line */
        close(stdin);
        dup(files[0]);
        close(files[0]);
        close(files[1]);
        /* stdin now comes from pipe */
      }
      execve(command2, command2, 0);
    }

    /* parent continues over here,
     * wait for child to exit if required
     */

    if (amp == 0)
        reid = wait(&status);
} // end of while loop
```

THANK YOU

Presented by Hasoo Eun