과목:	AI기반시스템프로그래밍
교수님:	최창희 교수님
학과:	정보보호학과
학번:	20011705
이름:	이지섭

제목 : Assignment_#3_Xsh

목차

I. REPL 루프 / EOF(Ctrl+D) 처리

```
s20011705@ubuntu-desktop:~/aisys/dev/assignment_3$ ./xsh
xsh[29384]:assignment_3> ls
README.md  sleep_print.sh  tests  xsh  xsh.c
xsh[29384]:assignment_3>
xsh[29384]:assignment_3>
xsh[29384]:assignment_3>
s20011705@ubuntu-desktop:~/aisys/dev/assignment_3$
```

위와 같이 REPL 루프가 동작하며, EOF(Ctrl + D)를 입력받으면 종료되는 것을 확인할 수 있습니다.

```
if (fgets(line, sizeof(line), stdin) == NULL) {
            printf("\n");
            break;  // EOF (Ctrl+D)
        }
if (line[0] == '\n') continue;
```

위 코드와 같이 실습 시간에 구현했던 코드와 동일하게 EOF 처리를 구현했습니다.

그리고 터미널 컬러 제어에 의해 xsh 프롬프트의 색이 초록색과 파란색의 조합으로 가시성이 좋아진 것을 확인할 수 있습니다.

II. 프롬프트 (PID + 디렉토리)

```
s20011705@ubuntu-desktop:~/aisys/dev/assignment_3$ ./xsh
xsh[29598]:assignment_3> cd ../
xsh[29598]:dev> ls
assignment_1  assignment_2  assignment_3  ch05  ch06  ch07  ch08  ch09  ch10
xsh[29598]:dev> cd ch10
xsh[29598]:ch10> ls
mini_shell_ai_sk.c  mini_shell_toggle_sk  mini_shell_toggle_sk.c
xsh[29598]:ch10>
```

위와 같이 xsh 프롬프트에 PID와 현재 디렉토리의 마지막 부분이 정상적으로 표시되는 것을 확인할 수 있었습니다. `cd` 명령어를 통한 경로 이동도 정상적으로 반영됩니다.

III. 기본 명령 실행 (fork/exec/wait + 에러처리)

우선 다음 코드는 제 코드의 일부분입니다.

```
/* fork a child process */
    child_pid = fork();
    if (child_pid < 0) {
        perror("fork");
        return;
    } else if (child_pid == 0) {
        setpgid(0, 0);
```

```
        if (redir_count > 0) {
            if (redirect_fds(redir_items, redir_count) < 0) {
                fprintf(stderr, "xsh: failed to redirect file
descriptors\n");
                exit(1);
            }
        }
        execvp(argv[0], argv);
        fprintf(stderr, "xsh: command not found: %s\n", argv[0]);
        exit(127);
    } else {
        setpgid(child_pid, child_pid);
        if (is_background) {
            /* 백그라운드 작업시 부모 프로세스는 대기하지 않음 */
            printf(ANSI_YELLOW "[bg] started pid=%d" ANSI_RESET "\n",
                   (int)child_pid);
            fflush(stdout);
            return;
        } else {
            tcsetpgrp(STDIN_FILENO, child_pid);
            if (waitpid(child_pid, &status, WUNTRACED) < 0) {
                perror("waitpid");
            }
            tcsetpgrp(STDIN_FILENO, getpgrp());
            return;
        }
    }
```

코드에서 확인할 수 있듯이, 기본 명령 실행은 fork -> exec -> wait 절차를 따릅니다.
그리고 자식 프로세스에서 exec 후 **명령어가 존재하지 않아 에러**가 발생할 경우, stderr를 **통해** 에러 메시지를 출력하는 로직을 볼 수 있습니다.

실제 작동 여부를 확인해 보겠습니다.

```
xsh[31659]:assignment_3> df -h
Filesystem                         Size  Used Avail Use% Mounted on
tmpfs                              387M  1.5M  386M   1% /run
/dev/mapper/ubuntu--vg-ubuntu--lv   18G   15G  2.9G  84% /
tmpfs                              1.9G     0  1.9G   0% /dev/shm
tmpfs                              5.0M     0  5.0M   0% /run/lock
/dev/sda2                          1.8G  192M  1.5G  12% /boot
tmpfs                              387M   12K  387M   1% /run/user/1000
xsh[31659]:assignment_3> ls
home.txt  out.txt  README.md  sleep_print.sh  sorted_out.txt  tests  xsh  xsh.c
xsh[31659]:assignment_3> cp home.txt home2.txt
xsh[31659]:assignment_3> ls
home2.txt  home.txt  out.txt  README.md  sleep_print.sh  sorted_out.txt  tests  xsh  xsh.c
xsh[31659]:assignment_3> asdf
xsh: command not found: asdf
xsh[31659]:assignment_3> non_function
xsh: command not found: non_function
xsh[31659]:assignment_3> fooba
xsh: command not found: fooba
xsh[31659]:assignment_3>
```

`df -h`와 같은 일반 명령어들도 정상적으로 작동됩니다. `-h` 옵션도 정상적으로 인식되어서 실행되는 것을 확인할 수 있습니다.
`ls`와 `cp` 명령어 또한 정상적으로 작동됩니다. `ls`를 통해 현재 디렉터리 내의 파일 목록을 출력하고, `cp` 명령어를 통해 파일이 복사되는 것을 확인했습니다.
그 외에, `asdf`, `non_function`, `fooba` 등 존재하지 않는 명령어를 실행하려 할 때 에러 메시지가 정상적으로 출력됩니다.

IV. 내장 명령어 (exit, cd, pwd)

A. exit

```
xsh[31173]:assignment_3> exit
s20011705@ubuntu-desktop:~/aisys/dev/assignment_3$
```

`exit` 명령어를 실행하면 프로그램 내부 반복문을 탈출하여 **프로그램이 종료**되고, 원래의 bash 셸로 돌아갑니다.

B. cd

```
xsh[31659]:assignment_3> cd .
xsh[31659]:assignment_3> cd ..
xsh[31659]:dev> ls
assignment_1  assignment_2  assignment_3  ch05  ch06  ch07  ch08  ch09  ch10
xsh[31659]:dev> cd ch10
xsh[31659]:ch10> ls
mini_shell_ai_sk.c  mini_shell_toggle_sk  mini_shell_toggle_sk.c
xsh[31659]:ch10> cd ..
xsh[31659]:dev> cd assignment_3
xsh[31659]:assignment_3>
```

`cd` 명령어를 통해 **디렉터리를 이동**할 수 있습니다.

C. pwd

```
xsh[31659]:dev> pwd
/home/s20011705/aisys/dev
xsh[31659]:dev> ls
assignment_1  assignment_2  assignment_3  ch05  ch06  ch07  ch08  ch09  ch10
xsh[31659]:dev> cd assignment_3
xsh[31659]:assignment_3> pwd
/home/s20011705/aisys/dev/assignment_3
xsh[31659]:assignment_3>
```

`pwd` 명령어를 통해 **현재 작업 디렉터리의 절대 경로**를 확인할 수 있습니다.

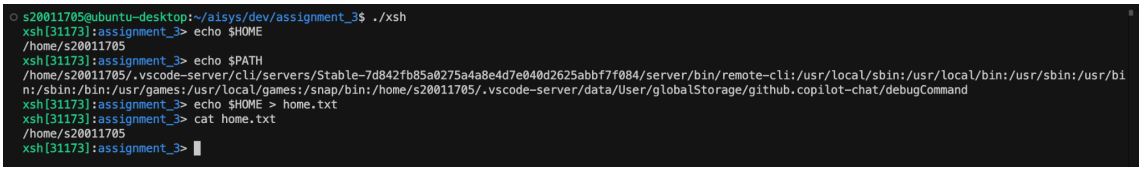
V. 리다이렉션 (<, >)

```
xsh[31659]:assignment_3> ls > out.txt
xsh[31659]:assignment_3> cat out.txt
home.txt
out.txt
README.md
sleep_print.sh
tests
xsh
xsh.c
xsh[31659]:assignment_3> sort < out.txt > sorted_out.txt
xsh[31659]:assignment_3> cat sorted_out.txt
home.txt
out.txt
README.md
sleep_print.sh
tests
xsh
xsh.c
xsh[31659]:assignment_3>
```

```
xsh[31659]:assignment_3> echo "Hello world" > hello.txt
xsh[31659]:assignment_3> cat hello.txt
"Hello world"
xsh[31659]:assignment_3> cat < hello.txt
"Hello world"
xsh[31659]:assignment_3>
```

`ls > out.txt` 를 실행하여 ls의 출력 결과를 터미널이 아니라 `out.txt`로 리다이렉트했습니다.
이후 `cat out.txt`명령어를 통해 ls의 출력 결과를 확인할 수 있었습니다.
그리고 `sort < out.txt > sorted_out.txt` 를 실행해서 sort에 `out.txt`를 입력으로 사용하고, 그 결과를 `sorted_out.txt`에 출력했습니다. `cat sorted_out.txt`로 결과를 확인할 수 있었습니다.
`echo` 명령어를 통한 문자열 출력도 `hello.txt`로 리다이렉트되어 출력된 것을 확인할 수 있습니다.
또한, `cat` 명령어에 `hello.txt` 파일을 리다이렉션으로 입력하여 정상적으로 출력되는 것을 확인할 수 있었습니다.

VI. 환경 변수 출력 ($VAR)



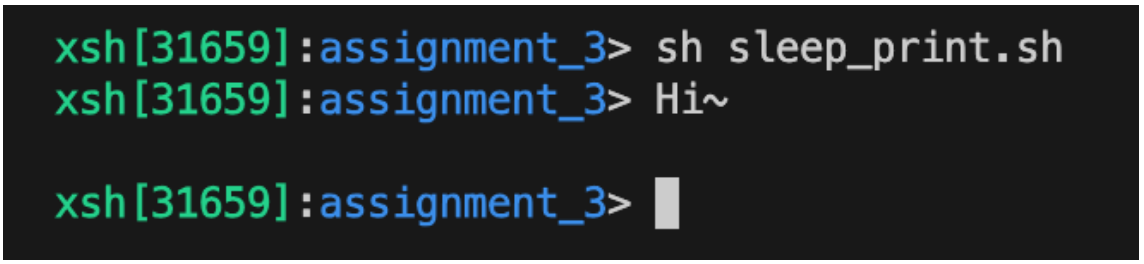
환경 변수 출력입니다. 기본적으로 `echo $HOME` 이나 `echo $PATH` 같이 $로 시작하는 문자열을 처리해서 환경 변수를 정상적으로 출력합니다.
추가적으로 환경 변수를 리다이렉션을 통해 `home.txt` 파일에 출력하는 것도 정상적으로 작동됩니다.

VII. 백그라운드 실행(&)

백그라운드 실행에 앞서 `sleep_print.sh` 파일을 만들었습니다. 파일 내용은 다음과 같습니다.

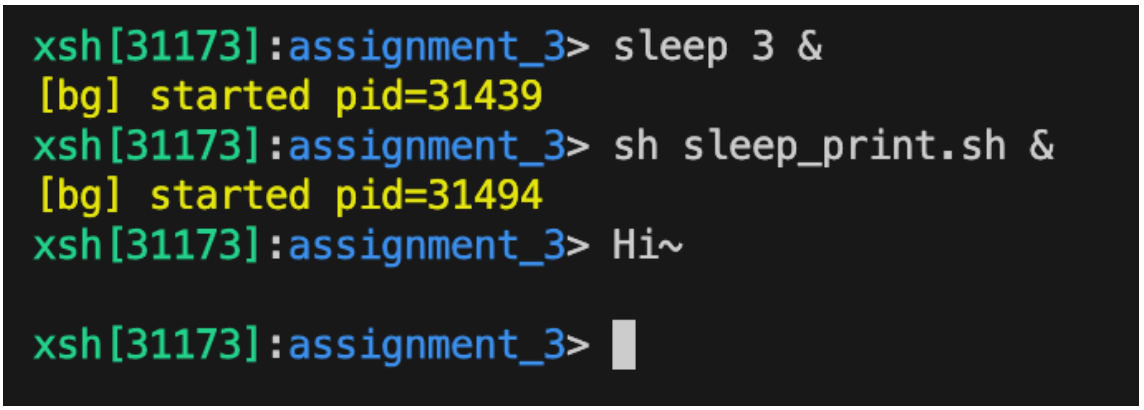
```
sleep 3; echo Hi~ &
```

해당 파일을 xsh에서 **백그라운드로 실행했을 때**와 **그렇지 않을 때**의 차이를 살펴보겠습니다.



먼저 `sleep_print.sh` 스크립트를 백그라운드 없이 실행했습니다. 이 경우 제어권이 자식 프로세스로 넘어가 실행 완료까지 대기해야했습니다.

자식 프로세스에 새 그룹을 생성하고 해당 그룹을 foreground로 설정했기 때문에, `SIGINT` 시그널을 보내도 xsh 셸은 종료되지 않고 자식 프로세스만 종료됩니다.



& 문자를 사용하여 백그라운드로 실행했을 때, 백그라운드 실행 메시지가 출력되고 백그라운드로 실행했으므로 제어권이 자식 프로세스에게 넘어가지 않은 걸 확인할 수 있었습니다.

VIII. Reference

- https://chatgpt.com/
- https://copilot.microsoft.com/
- https://gemini.google.com/