HW1: Simple Shell(SiSH)

32214362 모바일시스템공학과 조윤서

Free days remaining : 5 / Used : 0

Features

1) 내부 명령어

a) cd

```
choyoonseo@SiSH:~/Library/CloudStorage/OneDrive-Personal/모시공/3-2/운영체제/OS_Lab/2024-os-hw1$ cd shell-test
choyoonseo@SiSH:~/Library/CloudStorage/OneDrive-Personal/모시공/3-2/운영체제/OS_Lab/2024-os-hw1/shell-test$ cd ..
choyoonseo@SiSH:~/Library/CloudStorage/OneDrive-Personal/모시공/3-2/운영체제/OS_Lab/2024-os-hw1$ cd ..
choyoonseo@SiSH:~/Library/CloudStorage/OneDrive-Personal/모시공/3-2/운영체제/OS_Lab$ cd ..
choyoonseo@SiSH:~/Library/CloudStorage/OneDrive-Personal/모시공/3-2/운영체제$ cd ~
choyoonseo@SiSH:~$
```

b) echo

```
choyoonseo@SiSH:~/Library/CloudStorage/OneDrive-Personal/모시공/3-2/운영체제/OS_Lab/2024-os-hw1$ echo $PATH
/opt/homebrew/bin:/opt/homebrew/sbin:/Library/Frameworks/Python.framework/Versions/3.11/bin:/usr/local/bin:/System/Cryptexes/App/usr/bin:/usr/bin:/bin:/usr/sbin:/sbin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/local/bin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/bin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/appleinternal/bin:/Library/Apple/usr/bin:/Applications/VMware Fusion.app/Contents/Public:/opt/homebrew/bin:/opt/homebrew/sbin:/Library/Frameworks/Python.framework/Versions/3.11/bin:/Users/choyoonseo/flutter/bin:/Users/choyoonseo/flutter/bin
choyoonseo@SiSH:~/Library/CloudStorage/OneDrive-Personal/모시공/3-2/운영체제/OS_Lab/2024-os-hw1$ echo $USER
choyoonseo
choyoonseo@SiSH:~/Library/CloudStorage/OneDrive-Personal/모시공/3-2/운영체제/OS_Lab/2024-os-hw1$ echo $SHELL
/bin/zsh
choyoonseo@SiSH:~/Library/CloudStorage/OneDrive-Personal/모시공/3-2/운영체제/OS_Lab/2024-os-hw1$ echo "Hello World!"
"Hello World!"
choyoonseo@SiSH:~/Library/CloudStorage/OneDrive-Personal/모시공/3-2/운영체제/OS_Lab/2024-os-hw1$ echo myshell
myshell
```

c) ctrl+c 강제종료**(signal handler)**

```
choyoonseo@SiSH:~/Library/CloudStorage/OneDrive-Personal/모시공/3-2/운영체제/OS_Lab/2024-os-hw1/sish-demo$ make
gcc -g -Iheader -c src/alu.c -o obj/src/alu.o
gcc -g -Iheader -c src/control_signal.c -o obj/src/control_signal.o
gcc -g -Iheader -c src/cpu.c -o obj/src/cpu.o
gcc -g -Iheader -c src/executionStats.c -o obj/src/executionStats.o
gcc -g -Iheader -c src/memory.c -o obj/src/memory.o
^C
make: *** [obj/src/memory.o] Interrupt: 2
choyoonseo@SiSH:~/Library/CloudStorage/OneDrive-Personal/모시공/3-2/운영체제/OS_Lab/2024-os-hw1/sish-demo$ sleep 5
^C
choyoonseo@SiSH:~/Library/CloudStorage/OneDrive-Personal/모시공/3-2/운영체제/OS_Lab/2024-os-hw1/sish-demo$
```

d) exit

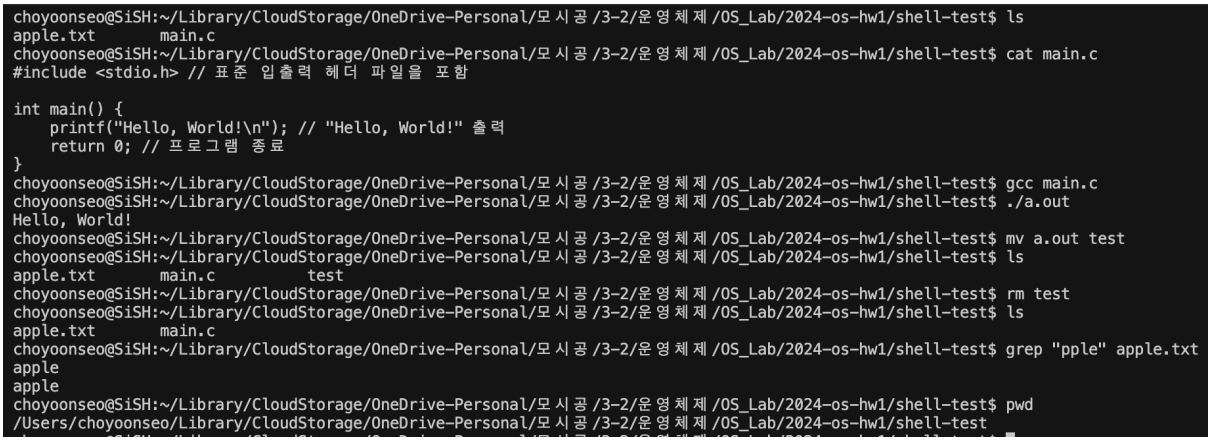
```
choyoonseo@SiSH:~/Library/CloudStorage/OneDrive-Personal/모시공/3-2/운영체제/OS_Lab/2024-os-hw1$ exit
Exiting SiSH...
```

e) pipe

```
choyoonseo@SiSH:~/Library/CloudStorage/OneDrive-Personal/모시공/3-2/운영체제/OS_Lab/2024-os-hw1/shell-test$ cat apple.txt | grep egg
egg
choyoonseo@SiSH:~/Library/CloudStorage/OneDrive-Personal/모시공/3-2/운영체제/OS_Lab/2024-os-hw1/shell-test$ ls | grep "apple"
apple.txt
choyoonseo@SiSH:~/Library/CloudStorage/OneDrive-Personal/모시공/3-2/운영체제/OS_Lab/2024-os-hw1/shell-test$ cat apple.txt | sort | uniq
apple
banana
cat
dog
egg
frog
giraffe
```

2) 외부 명령어

a) ls, gcc, ./a.out, pwd, cat, rm, mv, clear, grep



3) 기타기능

a) quit

b) tab키 자동완성

c) 방향키 **history**

Implementation

1. 내부 명령어 처리 (cd, exit, echo)

기능 개요

내부 명령어는 셸 자체에 내장되어 있으며, 외부 프로그램을 실행하지 않고 셸의 상태를 직접 변경하거나 제어하는 명령어이다. cd, exit, echo 등이 이에 해당한다.

구현 함수

- handle_internal_command(char **args)

동작 방식

1. **cd** 명령어 처리:
    - 목적: 현재 작업 디렉토리를 변경한다.
    - 사용된 함수:
        - getenv("HOME"): 환경 변수 HOME의 값을 가져와 홈 디렉토리를 기본 경로로 설정
        - chdir(path): 지정된 경로로 작업 디렉토리를 변경
    - 동작 과정:
        - cd 다음에 오는 인자가 없거나 ~일 경우, 홈 디렉토리로 이동
        - 인자가 있을 경우 해당 경로로 이동 시도. 실패 시 에러 메시지

출력

2. **exit** 명령어 처리:
    - 목적: 셸을 종료한다.
    - 사용된 함수:
        - exit(0): 프로그램 종료
3. **echo** 명령어 처리:
    - 목적: 인자로 받은 문자열이나 환경 변수의 값을 출력한다.
    - 사용된 함수:
        - getenv(args[i] + 1): $로 시작하는 인자의 경우, 해당 환경 변수의 값을 가져옴
    - 동작 과정:
        - echo 다음에 오는 각 인자를 순회하면서 $로 시작하는 경우 환경 변수 값을, 그렇지 않은 경우 그대로 출력

---

2. 외부 명령어 처리 **(ls, gcc, ./a.out** 등**)**

기능 개요

외부 명령어는 셸 외부에 존재하는 실행 파일을 실행하는 명령어이다. /bin, /usr/bin 등 경로에 위치한 실행 파일을 호출하여 다양한 기능을 수행한다.

구현 함수

- handle_external_command(char **args)
- find_command(const char* command)

동작 방식

1. 명령어 경로 찾기 **(find_command)**:
    - 목적: 입력된 명령어의 전체 경로를 $PATH 환경 변수에서 검색하여 찾는다.
    - 사용된 함수:
        - getenv("PATH"): PATH 환경 변수 값을 가져옴
        - strdup(), strtok(): PATH를 콜론(:)으로 분리하여 개별 디렉토리 경로를 추출
        - snprintf(): 디렉토리 경로와 명령어를 결합하여 전체 경로 생성
        - access(full_path, X_OK): 해당 경로에 명령어가 실행 가능한지 확인
    - 동작 과정:
        - PATH에 지정된 각 디렉토리를 순회하면서 명령어가 존재하고 실행 가능한지 확인
        - 발견 시 전체 경로를 반환. 발견하지 못하면 NULL 반환
2. 명령어 실행 **(handle_external_command)**:
    - 목적: 외부 명령어를 실행함

- 사용된 함수:
  - fork(): 자식 프로세스 생성
  - execve(): 외부 명령어 실행
  - waitpid(): 자식 프로세스 종료 대기
  - access(): 현재 디렉토리에 있는 실행 파일(a.out) 확인
- 동작 과정:
  - find_command로 명령어의 전체 경로를 찾음
  - 찾지 못한 경우, 현재 디렉토리에 실행 파일(a.out)이 있는지 access로 확인
  - 자식 프로세스에서 execve로 명령어 실행
  - 부모 프로세스는 자식 프로세스가 종료될 때까지 대기

---

3. 파이프 처리 **(|)**

기능 개요

파이프는 한 명령어의 출력을 다른 명령어의 입력으로 연결하여 여러 명령어를 조합하는 기능이다. 예를 들어, cat apple.txt | sort | uniq 와 같은 형태로 사용된다.

구현 함수

- handle_piped_commands(char *commands[MAX_ARG_SIZE][MAX_ARG_SIZE], int num_commands)
- execute_command(char *command)

동작 방식

1. 명령어 분리 및 파이프 확인 **(execute_command)**:
   - 목적: 전체 명령어 문자열을 분리하여 각 명령어와 파이프의 유무를 확인한다.
   - 사용된 함수**:**
     - strtok(): 명령어를 공백 단위로 분리
2. 파이프 설정 및 프로세스 연결 **(handle_piped_commands)**:
   - 목적: 여러 명령어 간의 파이프 연결을 설정하고 실행한다.
   - 사용된 함수:
     - pipe(): 파이프 생성
     - fork(): 각 명령어를 실행할 자식 프로세스 생성
     - dup2(): 파이프의 파일 디스크립터를 표준 입력/출력으로 복제
     - execve(): 명령어 실행
     - close(): 불필요한 파이프 닫기
     - waitpid(): 모든 자식 프로세스가 종료될 때까지 대기
   - 동작 과정:
     - 명령어 수에 맞춰 파이프 배열(pipefd) 생성
     - 각 명령어마다 자식 프로세스를 생성

        - 첫 번째 명령어의 출력은 첫 번째 파이프의 쓰기 단자로 연결
        - 두 번째 명령어의 입력은 첫 번째 파이프의 읽기 단자로 연결
        - 마지막 명령어는 표준 출력으로 연결
        - 모든 파이프는 부모 프로세스에서 닫고, 자식 프로세스는 자신의 입출력에 맞게 파이프 연결
        - 부모는 모든 자식 프로세스가 종료될 때까지 대기

---

4. 시그널 핸들링 **(SIGINT - Ctrl+C)**

기능 개요

시그널 핸들링은 특정 시그널이 발생했을 때, 셸이 이를 적절히 처리하여 예상치 못한 종료나 오류를 방지하는 기능이다. 특히 SIGINT(Ctrl+C)를 눌렀을 때 현재 실행 중인 프로세스를 종료시키거나, 셸 자체를 종료하지 않고 새로운 프롬프트를 출력하도록 한다.

구현 함수

- sigint_handler(int sig)

동작 방식

1. 시그널 핸들러 설정 **(main)**:
    - 목적: SIGINT 시그널이 발생했을 때 sigint_handler 함수가 호출되도록 설정한다.
    - 사용된 함수:
        - signal(SIGINT, sigint_handler): SIGINT 시그널을 sigint_handler에 연결
2. 시그널 핸들러 구현 **(sigint_handler)**:
    - 목적: SIGINT 시그널 발생 시 현재 실행 중인 자식 프로세스를 종료시키거나, 셸에 새로운 프롬프트를 출력한다.
    - 사용된 함수:
        - kill(child_pid, SIGTERM): 자식 프로세스에 종료 시그널 전송
        - fflush(stdout): 새로운 프롬프트 출력
    - 동작 과정:
        - child_pid가 -1이 아닌 경우(즉, 자식 프로세스가 실행 중인 경우) 해당 자식 프로세스를 종료
        - 자식 프로세스가 없는 경우, 현재 프롬프트를 새로운 줄에 출력하여 사용자에게 셸이 계속 실행 중임을 알림

---

5. 자동완성 및 히스토리 관리 **(Tab** 키**,** 방향키**)**

기능 개요

자동완성은 사용자가 명령어나 파일명을 입력할 때 Tab 키를 눌러 입력을 보조하는 기능이다. 히스토리 관리는 방향키를 사용하여 이전에 입력한 명령어를 불러오는 기능이다.

구현 함수

- tab_complete(const char* text, int start, int end)
- completion_generator(const char* text, int state)

사용된 전역 변수

- char* completion_array[1000];: 자동완성 후보를 저장하는 배열
- int completion_count = 0;: 자동완성 후보의 개수를 추적

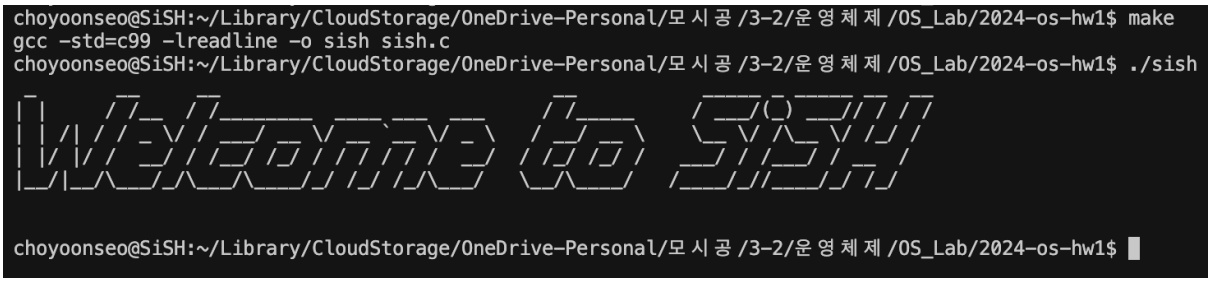
동작 방식

1. 자동완성 설정 **(main)**:
    - 목적: readline 라이브러리를 사용하여 자동완성 기능을 활성화한다.
    - 사용된 함수:
        - rl_attempted_completion_function = tab_complete: 자동완성 콜백 함수 설정
2. 자동완성 구현 **(tab_complete, completion_generator)**:
    - 목적: 현재 디렉토리의 파일명을 기반으로 자동완성 후보를 제공한다.
    - 사용된 함수:
        - opendir(), readdir(): 현재 디렉토리의 파일 목록 읽기
        - strncmp(): 입력한 텍스트와 파일명이 일치하는지 확인
        - strdup(): 일치하는 파일명 복제
        - rl_completion_matches(): 자동완성 후보 반환
    - 동작 과정:
        - tab_complete 함수는 현재 디렉토리의 모든 파일명을 읽어와 completion_array에 저장
        - completion_generator 함수는 text와 일치하는 파일명을 하나씩 반환하여 자동완성 후보로 제시
3. 히스토리 관리 **(readline.h, history.h)**:
    - 목적: 사용자가 이전에 입력한 명령어를 저장하고, 방향키를 통해 불러올 수 있도록 한다.
    - 사용된 함수:
        - readline(prompt): 사용자 입력을 받으면서 자동완성 기능과 히스토리 관리를 통합
        - add_history(input): 입력된 명령어를 히스토리에 추가
        - 방향키는 readline 라이브러리가 기본적으로 지원하여 이전 명령어를 탐색할 수 있게 함

Build Configuration / Environment

- Development Environment: **Visual Studio Code**
- Programming Language: **C**

- Build Configuration:
  - compilation: '**make**'
  - execution: '**./sish**'



Lesson

나만의 Simple Shell을 만들면서, cd, SIGINT, pipe와 같은 내부 명령어는 말 그대로 셸 자체에 내장된 기능이기 때문에 각각을 직접 구현해야 했습니다. 반면, 외부 명령어는 /bin 디렉토리 아래 설치된 실행 파일이므로 경로만 올바르게 찾아주면 자동으로 실행된다는 점이 흥미로웠습니다. 평소에 이러한 기능들을 단순히 사용하기만 했는데, 직접 구현할 수 있다는 것이 매우 재미있었습니다.

특히, 외부 명령어를 처리할 때 execvp와 execve의 차이점이 가장 인상 깊었습니다. execvp는 환경 변수 PATH에 지정된 디렉토리를 자동으로 검색하여 실행 파일을 찾기 때문에 ls, gcc, pwd와 같은 외부 명령어를 전체 경로 없이도 실행할 수 있습니다. 반면, execve는 PATH 환경 변수를 자동으로 검색하지 않기 때문에 이러한 외부 명령어를 실행하려면 /bin/ls, /bin/gcc, /bin/pwd처럼 전체 경로를 명시해야 합니다. 이번 과제에서는 execve 사용을 권장했는데, 처음에는 execvp로 구현했던 것보다 복잡했습니다. 명령어들의 경로를 찾아주는 find_command 함수를 직접 구현해야 했기 때문입니다. 하지만 이러한 과정을 통해 각 외부 명령어의 경로를 파악하는 방법을 배울 수 있어 매우 흥미로웠습니다.