2025_2주차_과제

스마트보안학부 2024350034 한주영

코드 작성 내용 (+ 과정)

총 네 개의 C파일을 기능을 기준으로 분리했다. 분리한 네 개의 파일은 main.c, builtin.c, executor.c, support.c로 구성했다. 쉘 내부에서 직접 처리해야 하는 cd, pwd, exit과 같은 명령어들은 builtin.c에 작성하여 내장 명령 처리를 하도록 했다. 반대로, fork와 같은 외부 명령을 실행하는 경우 executor.c를 통해 외부 명령과 파이프를 실행할 수 있도록 했다. 공통적으로 진행되는 사용자 입력 내용의 공백 및 특수 문자 기준을 나누거나 문자열 끝의 개행 제거, 그리고 동적 메모리 해제 등은 support.c에서 실행될 수 있게 했다. 사용자 입력을 받아 내장 명령인지 외부 명령인지 판단하고 적절한 함수로 실행될 수 있도록 분류하도록 하는 건 main.c를 통해 할 수 있게 했다. 이렇게 네 가지로 큰 틀을 나누어 좀 더 가독성을 높일 수 있도록 코드를 작성했다.

Makefile

Makefile이 작성법이 익숙하지 않아 강의자료에서 ChatGPT의 도움을 받아도 된다고 되어 있어 이 부분의 경우 GPT의 도움을 어느 정도 받았다. 하지만 이번 기회에 각 변수값들이나 명령어들이 무엇을 의미하는지 꼼꼼히 공부하였다.

기본 변수 설정

우선 사용할 컴파일러를 gcc로 사용하기 위해 `CC = gcc`를 작성했다.

컴파일 옵션을 설정하기 위해 일반적인 경고 메시지를 모두 출력하기 위한 `-Wall`, 추가적인 경고도 출력을 위한 `-Wextra`, 실행 성능을 위한 최적화를 위해 `-02`를 설정했다. 이 부분의 경우 변수가 익숙하지 않아 GPT의 도움을 받았다.

컴파일 후 만들어질 중간 파일들은 `OBJS = main.o builtin.o executor.o support.o`로 설정했다.

최종 실행 파일은 `TARGET = myshell`로 하여 myshell로 설정했다.

빌드 규칙 설정

C파일을 컴퓨터가 실행할 수 있는 프로그램으로 만들려면 .c파일을 .o파일로 바꾸는 컴파일 과정과 모든 .o 파일을 하나로 합쳐 실행 파일을 만드는 링크 과정을 거쳐야 한다. 이를 실행하기 위해 빌드 규칙을 아래와 같이 작성했다. 참고로 여기서 컴퓨터가 실행할 수 있는 프로그램은 앞서 `TARGET = myshell`을 통해 설정한 것처럼 myshell이다.

make 명령을 입력했을 때 기본으로 실행되는 작업은 `all: $(TARGET)`으로 설정했다. `$(TARGET)`은 우리가 앞서 지정한 myshell이다.

강의 자료에서 배웠던 Target, Dependencies, Recipe를 기준으로 빌드가 어떻게 이뤄지는지 작성했다. Target은 이전에 계속 myshell이라고 작성했기 때문에 설명을 생략하겠다. Dependencies의 경우 강의 자료대로 빌드 대상이 의존하는 Target 이나 파일 목록이다. 이걸 다 그대로 쓰기에는 너무 길어서 `$(OBJS)`로 썼고 실제로는 `main.o builtin.o executor.o support.o`이다.

빌드 대상을 생성하는 명령인 Recipe는 `$(CC) $(CFLAGS) -o $@ $(OBJS)`로 하여금 실제로 명령어를 실행해서 myshell을 만들도록 했다. $@는 현재 타겟인 myshell이고 `$(OBJS)`는 필요한 오브젝트 파일 목록이다. 따라서 이 명령어는 실제로는 `gcc -Wall -Wextra -02 -o myshell main.o builtin.o executor.o support.o` 이렇게 번역된다.

각 파일 컴파일 규칙 설정

-c는 컴파일만 하고 링크는 하지 마라는 뜻이다. 즉, .o 파일만 만들겠다는 뜻이다. 이를 그대로 main.c, builtin.c, executor.c, support.c에 적용하여 컴파일해서 main.o를 만들도록 설정했다.

정리 규칙 설정

`make clean`이라고 치면 .o 파일들과 myshell을 지워줄 수 있도록 했다.

main.c

`echo a && false`처럼 앞 명령이 실패했을 때 뒤 명령이 멈추지 않고 계속 실행됐다. 앞에서 실행된 명령이 잘 됐는지 아닌지 확인해서 다음 명령을 건너뛰는 부분이 제대로 작동하지 않았다. 따라서 `if (앞에꺼 && 뒤에꺼 실패)` 또는 `if (앞에꺼 || 결과가 성공)`일 때 뒤에 나오는 모든 단어를 `;`까지 건너뛰도록 while로 묶어주었다. 건너뛴 뒤에는 다시 if문 없이 새 명령부터 실행하도록 고쳤다.

파이프 처리 문제의 경우 `ls | grep .c` 같은 파이프를 쓰면 아예 프로그램이 멈추거나 출력이 안 나왔다. 파이프 사이에 들어갈 각 명령을 배열에 담아주지 않아서 실행 함수로 빈 목록이 넘어갔다. 파이프 처리 문제점을 해결하기 위해 조사를 한 결과 나와 동일하게 문제가 발생한 블로그가 있었다. 이를 참고하여 `|`를 만날 때마다 바로 건너뛰지 말고 그 전까지 모은 단어들로 하나의 명령 목록을 만들어 배열에 넣도록 바꾸어 문제를 해결할 수 있었다. 그리고 특히 블로그를 참고하면서 쉘 명령어를 토큰으로 나누는 아이디어를 참고하여 main.c를 작성할 때 명령어를 토큰화하는 방법을 구현할 수 있었다.

(참고한 블로그 1: https://jisu-log.tistory.com/6?utm)

(참고한 블로그 2: https://brennan.io/2015/01/16/write-a-shell-in-c/)

builtin.c

cd의 경우 `cd ../`무언가 처럼 상대 경로를 주면 파일이 없다고 하는 에러가 떴다. 경로 인자가 없을 때만 `HOME`으로 가게 해놓고 상대 경로가 있을 때도 놓치는 부분이 있다는 걸 발견했다. 그래서 argv[1]이 있으면 그대로 쓰고 없으면 `gentenv("HOME")`이나 `"."`으로 바뀌게 간단히 고쳤다. gentenv를 사용하게 된 건 블로그에 검색을 하다가 우연히 아래에 있는 gentenv 블로그를 읽어보다가 구현했는데 성공하게 되어서 그대로 두었다.

(참고 블로그: https://blog.naver.com/hoon2huny/50005636846)

pwd같은 경우는 작동되지 않았던 이유를 알 수 없어 지피티한테 물어본 결과 그냥 권한 문제로 `getcwd()`가 실패해도 왜 실패했는지 알 수 없는 거라고 해줬다. 인터넷에 찾아본 결과 `getcwd()`에 대해 친절하게 설명해주는 블로그와 자료가 있어 이를 참고하여 getcwd()가 `NULL`을 반환하면 `perror("pwd")`로 구체적인 에러 메시지를 띄우게 하면 됐었다. 그렇게 하니 바로 해결됐다.

(참고 블로그: https://ethical-hack.tistory.com/78)

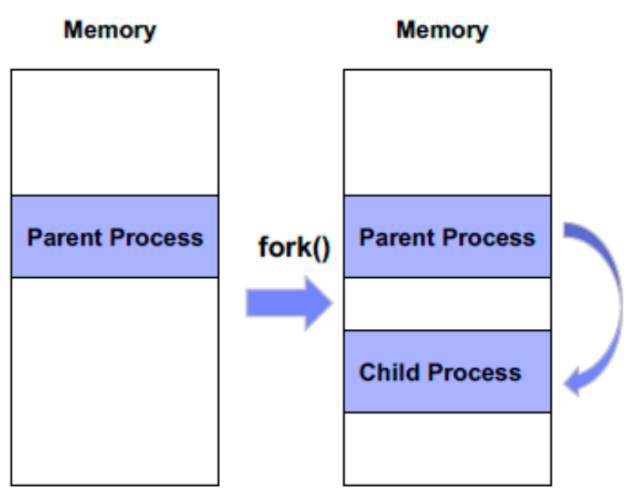
(참고 자료: https://www.gnu.org/software/libc/manual/html_node/Working-Directory.html) (← 이 자료 같은 경우 바로 위에 블로그에서 추천해주어 이번 과제에 같이 참고하게 되었다.)

executor.c

파이프 연결 문제가 있어 프로세스 연결 과정을 수정할 필요가 있었다. `ls | grep .c | wc -l`와 같은 명령을 실행해 보니 중간 단계인 `grep`나 마지막 wc에서 결과가 아예 안 나오거나 아예 멈춰버리기도 했다. 이건 구글링을 해도 케이스가 너무 다양하고 내 코드에서 직접적으로 문제점을 딱 찾기가 어려워 지피티에게 도움을 받을 수 밖에 없었다. 물어본 결과, 프로세스들이 연결될 때 `dup2()`로 표준 입출력을 바꿔 준 뒤에도 원래 파이프의 읽기나 쓰기 끝을 닫아주지 않아서 데이터가 한쪽으로만 흐르거나 전혀 흐르지 않아서 문제가 발생했다고 분석해주었다. 즉, 쉽게 말해 닫히지 않아 있었기 때문에 파이프가 제대로 다리 역할을 못 한 셈이었다.

리눅스 쉘의 동작원리 - 프로세스

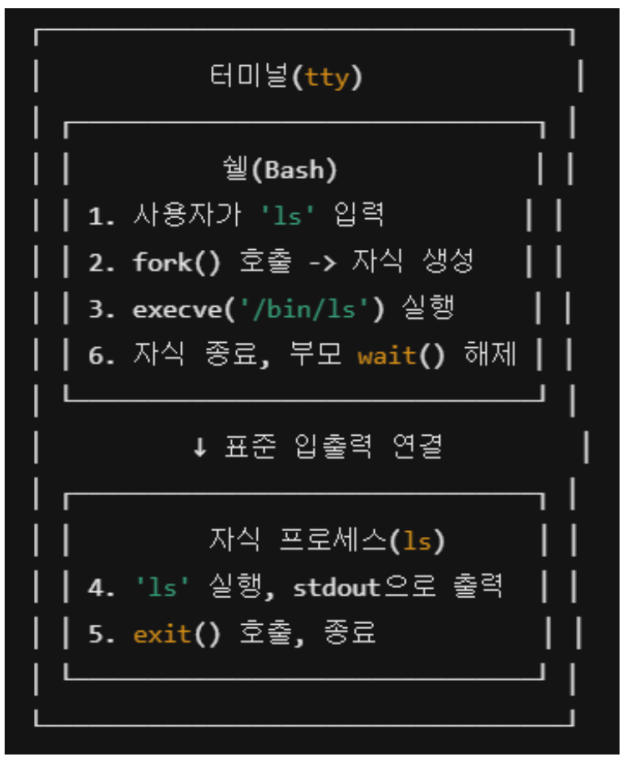
- 리눅스에서 프로세스를 생성하는 방법?
  - fork: 부모 프로세스에서 호출되면 새롭게 자식 프로세스를 생성한다.
  - fork에 의해 생성된 자식 프로세스는 부모 프로세스의 메모리를 그대로 복사하여 가짐 (이전에 배웠던 프로세스의 메모리 구조를 생각하면 됨)



정확히는 효율성을 위해 복사가 아니라
"Copy-On-Write"라는 개념을 사용함.
더 자세하게 알고 싶다면 아래 링크 참고 및 구글링
https://code-lab1.com/copy-on-write/

리눅스 쉘의 동작원리 - 프로세스

- 생성한 자식 프로세스에서 명령어를 실행하는 방법
  - exec 계열 함수: 현재 실행되고 있는 프로세스를 다른 프로세스로 대신하여 새로운 프로세스를 실행
  - 프로세스를 대체하는 것이지 새로 생성하는 것이 아니므로 fork와는 다른 개념
  - 보통 fork로 자식프로세스를 생성하고 exec으로 특정 실행파일을 실행하는 것이 일반적



처음에는 자식 프로세스 쪽은 모두 파이프 양쪽을 제대로 닫아주고 부모 프로세스는 복사한 내용을 닫는 함수를 불러 호출해 닫고 다시 다음 단계에 전달할 읽기 끝만 남기면 된다고 생각했다. 하지만 코드 작성에 있어서 실수가 많았고 오히려 혼란스러워 다시 처음 작성했던 코드로 돌아와야 했다. 따라서 이렇게 수정하고 싶다고 지피티에게 도움을 요청한 결과 자식 프로세스 쪽은 `dup2(pipe_fd[1], STDOUT_FILENO);` 이렇게 `STDOUT`으로 연결한 뒤 더 이상 필요 없는 파이프 양쪽 끝을 닫아주기 위해 `close(pipe_fd[0]);`와 `close(pipe_fd[1]);` 이렇게 작성했고 부모 프로세스의 경우는 `in_fd`를 `dup2()`로 복사한 뒤에 `close(in_fd)`를 호출해 닫은 다음 `pipe_fd[0]`만 남기도록 코드를 수정하는 데 도와주었다.

이렇게 수정하니 `ls | grep .c | wc -l`을 실행했을 때 `grep`와 `wc` 단계에서 모두 정상적으로 출력됐다.

support.c

토큰화하는 과정에서 `||`와 `&&`의 경우 각각 하나씩으로 인식이 되는게 아니라 `||`의 경우 `|`와 `|`로 두 개로 잘려서 제대로 비교가 되지 않았다. 따라서 먼저 두 글자짜리 `||`와 `&&`를 확인하도록 순서를 바꾸고 맞으면 한 번에 `||`나 `&&`를 strdup하게 했다.

(토큰화 참고 블로그: https://wooni1849.tistory.com/21)

(strdup 참고 블로그: https://salguworld.tistory.com/entry/CC-strdup-문자열-복사)

출력 결과

내장 명령

```
● (base) hanjooyoung@hanjuyeongdeMacBook-Pro shell % ls
  builtin.c       executor.c      main.c          Makefile        support.c
● (base) hanjooyoung@hanjuyeongdeMacBook-Pro shell % make
  gcc -Wall -Wextra -O2 -c main.c
  gcc -Wall -Wextra -O2 -c builtin.c
  gcc -Wall -Wextra -O2 -c executor.c
  gcc -Wall -Wextra -O2 -c support.c
  gcc -Wall -Wextra -O2 -o myshell main.o builtin.o executor.o support.o
● (base) hanjooyoung@hanjuyeongdeMacBook-Pro shell % ./myshell
  /Users/hanjooyoung/Desktop/shell$ pwd
  /Users/hanjooyoung/Desktop/shell
  /Users/hanjooyoung/Desktop/shell$ cd ..
  /Users/hanjooyoung/Desktop$ pwd
  /Users/hanjooyoung/Desktop
  /Users/hanjooyoung/Desktop$ exit
○ (base) hanjooyoung@hanjuyeongdeMacBook-Pro shell %
```

파이프라인

```
○ (base) hanjooyoung@hanjuyeongdeMacBook-Pro shell % ./myshell
  /Users/hanjooyoung/Desktop/shell$ ls -l | grep .c
  -rw-r--r--@ 1 hanjooyoung  staff    858 May  8 02:55 builtin.c
  -rw-r--r--@ 1 hanjooyoung  staff   1634 May  8 02:58 executor.c
  -rw-r--r--  1 hanjooyoung  staff   1688 May  8 03:17 executor.o
  -rw-r--r--@ 1 hanjooyoung  staff   4452 May  8 02:52 main.c
  -rw-r--r--@ 1 hanjooyoung  staff   2187 May  8 02:59 support.c
```

백그라운드 실행

```
○ (base) hanjooyoung@hanjuyeongdeMacBook-Pro shell % ./myshell
  /Users/hanjooyoung/Desktop/shell$ sleep 2 &
  /Users/hanjooyoung/Desktop/shell$ echo "백그라운드 실행 중"
  "백그라운드 실행 중"
```

복합 명령

```
  /Users/hanjooyoung/Desktop/shell$ echo one; echo two
  one
  two
  /Users/hanjooyoung/Desktop/shell$ true && echo OK
  OK
  /Users/hanjooyoung/Desktop/shell$ false || echo FAIL
  FAIL
```