프로젝트 설명

CPS LAB

요구 사항

팀별 : 주어진 구현 조건에서 shell 기능 구현 + 확장된 기능 추가

제약조건:

1) 사용자에게 문자열(shell 명령어)를 입력 받는다. (fgets 사용)

2) 입력 받은 문자열을 "₩n" 단위로 쪼갠다. (strtok 사용)
   - 만약, 입력 받은 문자열이 비어 있다면,
   - goto 문을 써서 while 문 내에서 마지막으로 jump 시킨다.

3) 쪼개진 문자열을 비교한다. (strcmp 사용)
   1) 쪼개진 문자열이 "exit"와 같다면, 프로그램을 종료 시킨다.
   2) 쪼개진 문자열이 "&"로 마무리 된다면, Background 프로세스로 실행하기 위해 flag 를 활성화한다.

요구 사항

팀별 : 주어진 구현 조건에서 shell 기능 구현 + 확장된 기능 추가

제약조건:

4) 현재 프로세스의 자식 프로세스를 생성한다. (fork 사용)

a. 만약 pid 가 음수이면, fork 과정에 문제가 발생한 것임으로 error 를 출력하고 빠져나온다.

b. 만약 pid 가 0 이면, 자식 프로세스를 실행시킨다.
   - 자식 프로세스가 쪼개진 문자열(명령 한 줄)을 수행한다. (execvp 사용)

c. 만약 pid 가 0 보다 크면, 부모 프로세스를 실행시킨다.
   - 만약 background flag 가 활성화되어 있지 않다면, 자식 프로세스의 종료를 기다린다. (waitpid 사용)
   - 그렇지 않다면, 부모프로세스(프로그램)를 바로 종료 시킨다.

요구 사항

팀별 : **주어진 구현 조건에서 shell 기능 구현 + 확장된 기능 추가**

확장된 기능 추가:

- **제약조건에 맞춰서 전부다 구현되었으면,**
- **본인이 생각에서는 이 shell에게 뭐가 더 필요했을까?**

요구 사항

팀별 : 주어진 구현 조건에서 shell 기능 구현 + 확장된 기능 추가

실행 예)

```
(base) kyhooon@kyh:~$ cd sysprogram_practice/project/
(base) kyhooon@kyh:~/sysprogram_practice/project$ ls
example.c  example.out  my_shell.c  my_shell.out
(base) kyhooon@kyh:~/sysprogram_practice/project$
(base) kyhooon@kyh:~/sysprogram_practice/project$ ./my_shell.out
my_shell>
my_shell>
my_shell> ls -l
total 48
-rw-rw-r-- 1 kyhooon kyhooon   255 11월  20 21:28 example.c
-rwxrwxr-x 1 kyhooon kyhooon 16808 11월  20 21:28 example.out
-rwxr-xr-x 1 kyhooon kyhooon  2074 11월  21 00:24 my_shell.c
-rwxrwxr-x 1 kyhooon kyhooon 17360 11월  21 00:24 my_shell.out
my_shell>
my_shell>
my_shell> ls -l &
It's background process
my_shell> total 48
-rw-rw-r-- 1 kyhooon kyhooon   255 11월  20 21:28 example.c
-rwxrwxr-x 1 kyhooon kyhooon 16808 11월  20 21:28 example.out
-rwxr-xr-x 1 kyhooon kyhooon  2074 11월  21 00:24 my_shell.c
-rwxrwxr-x 1 kyhooon kyhooon 17360 11월  21 00:24 my_shell.out

my_shell>
my_shell> pwd
/home/kyhooon/sysprogram_practice/project
my_shell>
my_shell> exit
(base) kyhooon@kyh:~/sysprogram_practice/project$
```

만든 shell이 명령어+인자로 실행 가능해야 함

만든 shell이 background 프로그램 지원하도록 함

shell이 탈퇴했을 때, "exit"로

```
(base) kyhooon@kyh:~/sysprogram_practice/project$ ./my_shell.out
my_shell>
my_shell> nop
command not found | invalid option
my_shell>
```

만든 shell에서 없는 명령어를 쓸 때, 아래 내용출력

요구 사항

팀별 : 주어진 구현 조건에서 shell 기능 구현 + 확장된 기능 추가

구현한 shell에 불편점 대해 기능추가 OK

그러나 색깔 표시, zshell/csh 바꿈 등, 인증하지 않겠다

예) 1. 리눅스에 있는 bash셸이 tab 기능을 제공하여, 명령어/파일이름 검색할 수 있음

```
(base) kyhooon@kyh:~$ gr    명령어 입력하다가, 이름을 몰라서 일단 tab를 사용하여 명령어/ 파일 검색
grep                    groupadd                grpc_ruby_plugin        grub-macbless           grub-mkstandalone
grep-aptavail           groupdel                grpck                   grub-menulst2cfg        grub-mount
grep-available          groupmems               grpconv                 grub-mkconfig           grub-ntldr-img
grep-dctrl              groupmod                grpunconv               grub-mkdevicemap        grub-probe
grep-debtags            groups                  grub-bios-setup         grub-mkfont             grub-reboot
grep-status             grpc_cpp_plugin         grub-editenv            grub-mkimage            grub-render-label
gresource               grpc_csharp_plugin      grub-file               grub-mklayout           grub-script-check
groff                   grpc_node_plugin        grub-fstest             grub-mknetdir           grub-set-default
grog                    grpc_objective_c_plugin grub-glue-efi           grub-mkpasswd-pbkdf2    grub-syslinux2cfg
grops                   grpc_php_plugin         grub-install            grub-mkrelpath
grotty                  grpc_python_plugin      grub-kbdcomp            grub-mkrescue
(base) kyhooon@kyh:~$ gr
```

예) 2. 리눅스에서 Ctrl+u를 이용하여 현재 작성하고 있는 내용을 지우는 방법이다

```
(base) kyhooon@kyh:~/sysprogram_practice/project$ trencate
                                                  Ctrl + u
```

감사합니다.

CPS LAB

#include <string.h>

int strtok (char *str, const char *delim);

Return Value:

The strtok() function return a pointer to the next token, or NULL if there are no more tokens.

```c
#include <stdio.h>
#include <string.h>

int main() {
    char str[] = "Hello, World! Welcome to programming.";
    const char s[5] = " .,!"; // Delimiter is a space character
    char *token;

    // Getting the first token
    token = strtok(str, s);

    // Walking through other tokens
    while(token != NULL) {
        printf("%s\n", token);
        token = strtok(NULL, s);
    }

    return 0;
}

```

/* 구분자(delim)를 통해 주어진 string(str) 잘라내기 */

goto 문

현재 이러한 문법으로 되어 있는 코드가
사용자 애플리케이션에서 보기 어렵다

단, 커널에서 이러한 goto문을 써서,에러 구분.
코드관리 등 사용하고 있음

예)
Linux Kernel 5.4.0
Device Driver 중, pblk 모듈의 일부 코드:
"모듈에서 정의된 pblk_rec 스레드 함수를 통해 메모리 할당 요청을 한다."

- goto 정의된 여러 문단(*_destroy_ws, *_destroy_rec, *_fail_destroy_g_rq)을 통해 쉽게 에러관리 할 수 있음

```
static int pblk_create_global_caches(void)
{

        pblk_caches.ws = kmem_cache_create("pblk_blk_ws",
                                sizeof(struct pblk_line_ws), 0, 0, NULL);
        if (!pblk_caches.ws)
                return -ENOMEM;

        pblk_caches.rec = kmem_cache_create("pblk_rec",
                                sizeof(struct pblk_rec_ctx), 0, 0, NULL);
        if (!pblk_caches.rec)
                goto fail_destroy_ws;

        pblk_caches.g_rq = kmem_cache_create("pblk_g_rq", pblk_g_rq_size,
                                0, 0, NULL);
        if (!pblk_caches.g_rq)
                goto fail_destroy_rec;

        pblk_caches.w_rq = kmem_cache_create("pblk_w_rq", pblk_w_rq_size,
                                0, 0, NULL);
        if (!pblk_caches.w_rq)
                goto fail_destroy_g_rq;

        return 0;

fail_destroy_g_rq:
        kmem_cache_destroy(pblk_caches.g_rq);
fail_destroy_rec:
        kmem_cache_destroy(pblk_caches.rec);
fail_destroy_ws:
        kmem_cache_destroy(pblk_caches.ws);

        return -ENOMEM;
}
```