시스템프로그래밍 3분반 4번째 과제 32203660 이현정

mysh 프로그램 보고서

1. 소개

mysh은 리눅스의 shell을 구현해보는 프로그램입니다.

기본적으로 계속 $가 떠서 명령어를 받을 수 있어야 하므로 while문을 통해서 프로그램 종료 조건 외에는 while문이 계속 돌아 가야합니다. 그리고 shell에서는 명령어와 인자를 받아서 해당하는 것에 따라서 처리를 하기 때문에 각 명령어와 인자를 잘라서 구분해야 하는 기능이 필요합니다. 추가적으로 타 명령어와 달리 동작을 해야 하므로 redirection와 background는 다른 방향으로 구현이 되어야 합니다.

2. 본문

1) snapshot

```
sys32203660@embedded:~$ ls
alpha_attr.txt  examples.desktop  my_favorite_poem.txt  test.c       test.s
alpha_new.txt   hello.c           mysh.c                test_copy.c
alpha.txt       mycp              new.c                 test.o
a.txt           mycp.c            sh                    test.out
sys32203660@embedded:~$
sys32203660@embedded:~$ gcc -o mysh mysh.c
sys32203660@embedded:~$
sys32203660@embedded:~$ ./mysh
HyeonJeong/home/3-class/sys32203660 $
HyeonJeong/home/3-class/sys32203660 $ ls
alpha_attr.txt  examples.desktop  my_favorite_poem.txt  sh           test.out
alpha_new.txt   hello.c           mysh                  test.c       test.s
alpha.txt       mycp              mysh.c                test_copy.c
a.txt           mycp.c            new.c                 test.o
HyeonJeong/home/3-class/sys32203660 $
HyeonJeong/home/3-class/sys32203660 $ gcc -o hello hello.c
HyeonJeong/home/3-class/sys32203660 $
HyeonJeong/home/3-class/sys32203660 $ ./hello
Hello World
HyeonJeong/home/3-class/sys32203660 $
HyeonJeong/home/3-class/sys32203660 $ help
/***************Simple Shell***************/
You can use it just as the conventional shell


Some examples of the bult-in commands
cd      :change directory
exit    : exit this shell
quit    : quit this shell
help    : show this help
? : show this help
/******************************************/
HyeonJeong/home/3-class/sys32203660 $
HyeonJeong/home/3-class/sys32203660 $ ps
  PID TTY          TIME CMD
 9173 pts/30   00:00:00 ps
10861 pts/30   00:00:00 mysh
12530 pts/30   00:00:00 bash
HyeonJeong/home/3-class/sys32203660 $
HyeonJeong/home/3-class/sys32203660 $ exit
sys32203660@embedded:~$
sys32203660@embedded:~$ ps
  PID TTY          TIME CMD
12530 pts/30   00:00:00 bash
15992 pts/30   00:00:00 ps
sys32203660@embedded:~$
sys32203660@embedded:~$ whoami
sys32203660
sys32203660@embedded:~$ date
2021. 11. 05. (금 ) 15:58:19 KST
```

2) mysh.c

sys32203660@embedded: ~

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/wait.h>
#include <stdbool.h>
#include <fcntl.h>


int tokenize(char *line, char *tokens[], int back) {
  int index = 0;
  int cnt = 0;
   while (*line != '\0') {
      if (*line == ' ' || *line == '&') {
        if (*line == '&') {
          back = 1;
        }
        *line = '\0';
         line = line + 1;
      }
      else {
         tokens[index] = line;
         index = index + 1;
         while ((*line != '\0') && (*line != ' ')) {
           cnt = cnt + 1;
            line = line + 1;
         }
         cnt = 0;
      }
   }
   tokens[index] = '\0';
   return index;
}

bool run(char* line) {
  int token_count, back;
    char* tokens[100];
    token_count = tokenize(line, tokens, back);

  if(token_count == 0){
    return true;
  }
  if(strcmp(tokens[0], "exit") == 0){
    exit(0);
  }
  if(strcmp(tokens[0], "quit") == 0){
    return false;
  }
  if(strcmp(tokens[0], "cd") == 0){
    chdir(tokens[1]);
    return true;
  }
  if(strcmp(tokens[0], "help") == 0 || strcmp(tokens[0], "?") == 0 ){
    printf("/***************Simple Shell***************/\n");
    printf("You can use it just as the conventional shell\n");
    printf("\n");
    printf("\n");
    printf("Some examples of the bult-in commands\n");
    printf("cd\t:change directory\n");
    printf("exit\t: exit this shell\n");
    printf("quit\t: quit this shell\n");
```

sys32203660@embedded: ~

```c
    printf("help\t: show this help\n");

    printf("? : show this help\n");
    printf("/****************************************/\n");
    return true;
  }

pid_t pid;


    // background
    if (back > 0) {
      tokens[token_count-1] = NULL;
      if((pid = fork()) == 0) {
        execvp(tokens[0], tokens);
        exit(0);
      }
     // not wait
      return true;
    }

    // redirection
    int redir = 0, i=0;
    char* pre[100];
    for (; i < token_count; i++) {
      if (strcmp(tokens[i], ">") == 0) {
        redir = i;
        break;
      }
      else {
        pre[i] = tokens[i];
      }
    }

    if (redir > 0) {
      if ((pid = fork()) == 0) {
        int fd = open(tokens[redir+1], O_WRONLY | O_CREAT | O_TRUNC, 0644);
        dup2(fd, STDOUT_FILENO);
        execvp(pre[0], pre);
        close(fd);
      }
      wait(0);
      return true;
    }

    //try execve
    /*
    char *path[] = {"ls", "-a", NULL};
    tokens[0] = "ls";
    if ((pid = fork()) == 0) {
      if(execve("user/bin/ls", tokens, path) == -1) {
        fprintf(stderr, "프 로 그 램  실 행 ");
      }
    }
    */

    if ((pid = fork()) == 0) {
      execvp(tokens[0], tokens);
    }
    wait(0);
    return true;
}
```

main 함수에서 while문으로 종료 상태 지정 외에는 계속 mysh을 유지하도록 하였습니다. run()함수가 while문 내에서 동작하게 됩니다. line으로 받은 명령어 문장을 토큰으로 자릅니다. 추가로 제 코드는 추가로 라인에 '&'가 존재하는지 확인하고 있다면, 1을 back 변수에 넣습니다.

exit, quit, cd, help, ?가 들어온 경우, 해당 명령어는 토큰의 첫번째에 존재하는지 확인 후 존재한다면 해당하는 명령어의 역할을 실행시킵니다.

background는 앞선 토큰에서 준 back이 0보다 크면 존재하는 것이므로 해당 '&' 위치에 '₩0' 넣고, fork() 후 execvp()하고, 주어진 명령어가 background로 실행될 수 있게 부모는 wait()하지 않고 바로 다시 return true로 while문을 돌게 됩니다.

for문으로 토큰을 돌리면서 redirection이 들어왔는지 확인 후 있다면 해당 인덱스 수를 redir 변수에 넣습니다.위의 변수가 0보다 크다면, '>'가 있으므로 fork() 후 출력 STDOUT_FILENO 대신 주어진 파일에 작성될 수 있게 dup2()함수를 이용하고, 열린 동안 execvp()를 통해서 주어진 '>'전 명령어가 실행되고 출력 내용이 주어진 파일에 저장됩니다.

그리고 앞서 언급하지 않은 명령어는 fork()후 execvp()하고, 부모는 먼저 도는 경우 wait()을 하고 자식이 끝나면 그 다음에 true가 반환되어 while문을 돌게 됩니다.

3. 시행 착오 및 결론

1) 1번에 토큰으로 분리와 background 확인

초기에는 당연히 토큰을 위한 함수이므로 토큰의 역할만 하는 것이 맞다고 생각하기도 했지만, redirection과 background에서 다시 해당 토큰에 해당하는 것이 주어졌는지 확인을 해야 하기에 효율적인 코딩이 될 수 있게 구성하고 하는 생각이 있었습니다. background를 확인할 때, for문을 다시 돌리지 않고 한번 돌릴 때 바로 background까지 확인하면 좋을 것이라는 생각에서 시도해 보았습니다.

다만, redirection 또한 같이 최대한 정보를 가져올 수 있게 하여 for문의 이용을 줄여보려고 했지만 '>'전까지 주어진 명령어의 분리와 '>' 다음 오는 파일명을 알기 위해서 처리를 해야 하므로 모두 토큰 생성 단계에서 하는 것은 독자가 보기 어려울 것이라는 판단 하에 따로 for문을 돌려서 먼저 redirection인지 확인과 위에서 언급한 2가지 필요한 정보들을 가져갈 수 있게 하였습니다.

2) 프로세스가 죽지 않아서 exit를 여러 번 해야 하는 문제 발생

해당 내용을 보면서 exec()함수들이 원활하게 동작하지 못했을 때 발생하는 문제라는 것을 알 수 있었습니다. 또한, 해당 내용을 확인하기 위해서 중간중간 토큰을 다시 출력해보는 과정을 해 본 결과, 특정부분에서 토큰에 '₩0'값이 제대로 넣어지지 않을 것을 우려해서 작성해 놓은 부분이 사실은 그보다 1개 앞 인덱스를 가리키고 있어서 토큰의 마지막 값들이 사라져서 exec()함수 실행에 문제가 발생했다는 것을 인지할 수 있었습니다.

3) '₩0'과 '₩n'

하다 보니 생각보다 두 값에 의해서 발생하는 문제가 많았습니다. 제거하지 못해서 토큰에 포함이 되어버리는 경우도 있었고, 또 명령어의 인자로 인식해서 제대로 프로그램이 동작하지 못하게 막는 경우도 있었습니다. 그렇기에 각 함수가 정확히 어디까지 반환하고, 마지막에 '₩0'값을 넣는지 등의 유무를 확실히 인지하고 있는 것이 필요하다는 생각을 하였습니다.

4) 결론

그리하여 mysh은 while문을 통해서 run()이 계속 돌아가면서 주어진 명령문을 토큰으로 분리해 정해진 명령어의 존재시에는 주어진 내용에 따라서 작동하고, 그 외는 받은 명령어가 그대로 자식 프로세스에서 원활히 실행될 수 있게 하였습니다. redirection과 background는 위에 설명한 방법으로 따로 구현을 하였으며, 프로그램을 종료하는 명령, cd, help 등도 추가적으로 확인 즉시 실행될 수 있게 구현하였습니다.