리눅스시스템 Lab13

분반: 001 학과: 컴퓨터과학전공 학번: 2312282 이름: 임다희

1. 쉘 인터프리터 작성

p3의 코드를 작성하고 실행했을 때의 결과를 바탕으로 p3의 물음에 대해 **설명**한다. (단답X)

Q1. 생성된 프로세스는 몇 개인가?

```
u2312282@dahee-VirtualBox:~/linux/chap13$ ./test
[3836] 프로세스 시작
[3836] 프로세스 : 반환값 3837
[3836] 프로세스 :: 반환값 3838
[3837] 프로세스 : 반환값 0
[3837] 프로세스 :: 반환값 3839
[3839] 프로세스 :: 반환값 0
[3838] 프로세스 :: 반환값 0
```

4개이다. 프로그램을 실행하면 3836번 프로세스가 생성되고(1개), main()의 첫번째 fork()에 의해 프로세스가 자가복제되어 자식 프로세스 3837번이 생성된다(2개). 이 직후 3836번 프로세스 main의 두 번째 fork에 의해 자식 프로세스 3838번이 생성된다(3개).

3836번의 자식 프로세스 3837번은 main()의 첫 번째 fork 이후의 printf문부터 실행된다. 이후 두 번째 fork를 만나면 프로세스가 자가복제되어 자식 프로세스 3839번을 생성한다. (4개)

3836번의 자식 프로세스 3838번은 main()의 두 번째 fork 이후의 printf문부터 실행되며, 이후로 만나는 fork()문이 없으므로 자식 프로세스를 생성하지 않는다.

따라서 생성되는 프로세스는 총 4개이다.

Q2. 그들 사이의 관계는 무엇인가?

3836번 프로세스에서 fork() 두 번이 실행되어 생성된 프로세스가 3837번, 3838번이며 이들이 실행될 때 자신의 반환값을 0으로 출력하는 것을 볼 수 있다. 따라서 3836번 프로세스는 3837번, 3838번 프로세스의 부모이다.

3837번 프로세스에서 fork() 한 번이 실행되어 생성된 프로세스가 3839번이며 해당 프로세스가 실행될 때 자신의 반환값을 0으로 출력하는 것을 볼 수 있다. 따라서 3837번 프로세스는 3839번의 부모이다.

Q3. fork()를 n번 하면 몇 개의 프로세스가 생성되는가?

fork()문이 n개 존재할 때 1번째 fork()문에서는 1개의 자식 프로세스가, 2번째 fork() 문에서는 2개의 자식 프로세스가, 3번째 fork()문에서는 4개의 자식 프로세스가 생성되고, n번째 프로세스에서는 2의 n-1 제곱 개의 자식 프로세스가 생성된다. 이를 모두 더하면 생성되는 자식 프로세스는 1+2+4+....2^(n-1)=(2^n)- 1, 즉 2의 n제곱 -1 개의 자식 프로세스가 생성된다. 프로그램 생성 시 맨 처음으로 생성되는 프로세스 1개를 합하면 총 프로세스 수는 2의 n제곱 개이다.

2. 쉘 인터프리터 작성

P7의 (1)~(6)를 중심으로 쉘 인터프리터 코드에 대해 설명한다. (기능 구현이 100% 안 되었더라도 구현한 범위까지 설명을 작성, 코드첨부는 자유)

```
//001 컴퓨터과학과 2312282 임다희
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <string.h>

int main(){
        int MAXARG=100;
        int pid,i;
        int j=1;
        int enable=1;
        int is_background=0;
        char str[MAXARG];
        char *args[MAXARG];
        char *command;
        char *saveptr;

        while(enable==1){

                printf("[shell] ");
                fgets(str,sizeof(str),stdin);
                str[strcspn(str,"\n")]='\0';

                if(strcmp("quit",str)==0){
                        enable=0;
                        break;}

                i=0;
                command=strtok_r(str," ",&saveptr);
                while(command!=NULL&&i<MAXARG){
                        args[i]=command;
                        i++;
                        command=strtok_r(NULL," ",&saveptr);}

                if(i>0 && strcmp(args[i-1], "&")==0){
                        is_background=1;
                        args[i-1]=NULL;}
                else{is_background=0;}

                if(i<MAXARG){
                        args[i]=NULL;}

                pid=fork();

                if(pid==0){
                        execvp(args[0],args);}
                else{
                        if(is_background==0){
                                wait(NULL);}
                        else{
                                printf("[%d] %d\n",j,pid);
                                j++;}
                }
        }
        return 0;
}
```

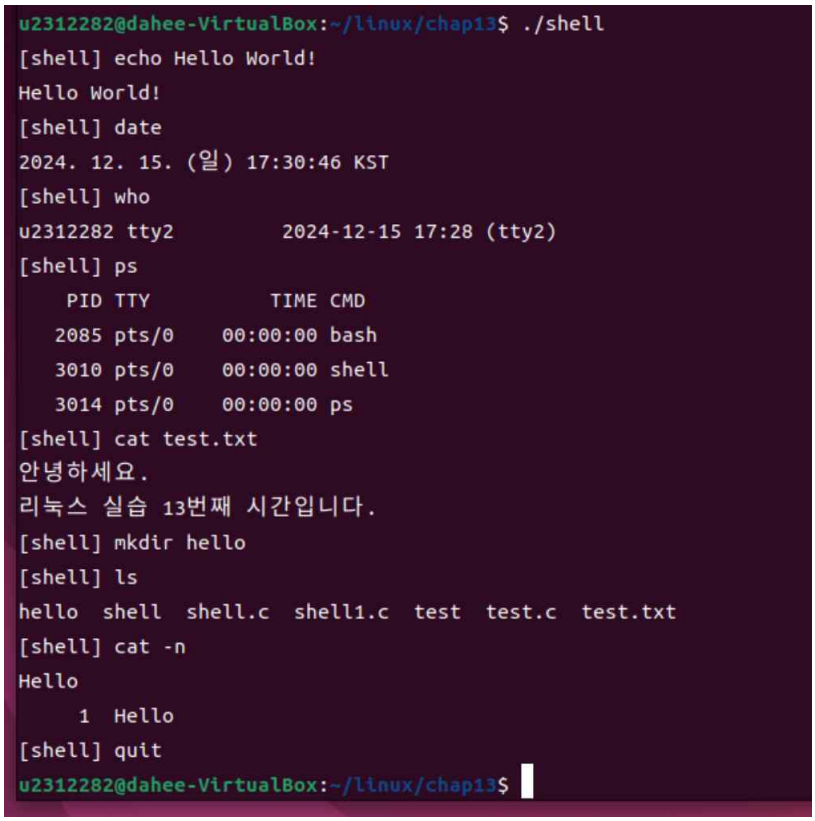
main 함수가 시작되면 [shell] 프롬프트를 출력하고 fgets(str,sizeof(str),stdin) 을 통해 사용자에게 명령어를 입력받아 char 타입의 배열 str에 저장한다. str[strcspn(str,"\n")]="\0" 을 통해 명령어 끝의 줄바꿈 기호를 제거하고, strcmp("quit", str) 를 통해 입력받은 문자열이 quit일 경우 반복 실행이 종료되도록 한다.

strtok_r(str," ", &saveptr)를 통해 입력받은 문자열을 공백(띄어쓰기) 단위의 단어로 분리하고, while 반복문을 통해 문자열이 끝날 때까지 계속 분리를 진행하며 각 단어를 args 배열에 저장한다. 명령어 문자열의 끝에 후면 처리를 나타내는 기호가 붙어있을 경우 후면 처리 여부를 나타내는 is_background 변수의 값을 1로 변경한다.

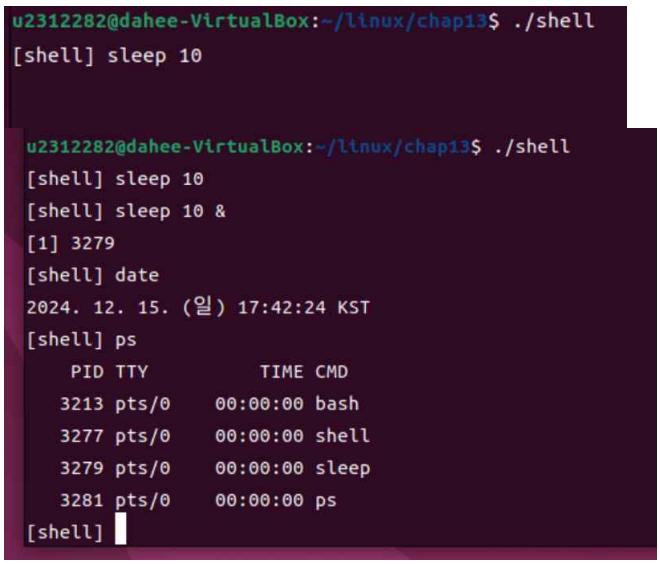
pid=fork() 를 통해 자식 프로세스를 생성한다. pid가 0일 경우, 즉 생성된 자식 프로세스에서는 execvp(args[0], args) 의 시스템 호출을 통해 입력받은 명령어를 실행한다. pid가 0이 아닌 경우, 즉 전면 작업일 때와 후면 작업일 때를 구분한다. is_background가 0일 경우(전면 작업)에는 wait(NULL) 을 통해 자식 프로세스가 끝나기를 기다린 후 처음의 프롬프트 출력으로 돌아가 같은 작업을 반복한다. 후면 작업의 경우에는 프로세스 번호와 작업 번호를 출력하고, 그 직후 자식 프로세스를 기다리지 않고 처음으로 돌아가 같은 작업을 반복한다.

3. 쉘 인터프리터 실행 결과

(1) p8을 참고하여 서로 다른 5개 이상의 명령을 실행하고, “quit” 입력으로 종료하기까지의 터미널 창을 캡처한다.



(2) p9를 참고하여 ‘sleep 10’ 과 ‘sleep 10 &’ 명령의 실행 결과에 대해 설명한다.



sleep 10 을 전면 실행하면 해당 명령어의 실행이 종료될 때까지 다른 명령어를 실행할 수 없다. sleep 10을 후면 실행하면 해당 명령어가 실행되는 동안에도 다른 명령어를 전면에서 실행할 수 있고, 실행 종료 시 다시 명령어 반복 실행문의 처음으로 돌아가 새로운 명령어를 입력받을 수 있다.