Shell

Homework 1
모바일시스템공학과
32143153 이대훈

1. Project Introduction

Shell이라는 Background에서 command를 입력 받아 Shell이라는 프로세스 내에서 돌아가도록 합니다. 이때 fork 함수를 사용하여 Shell을 parent 프로세스로 두고 child 프로세스를 생성하여 child 프로세스 대신 command를 실행하도록 합니다.

2. Motivation

컴퓨터를 사용할 때 OS는 기본적인 Background 요소입니다. OS는 시스템 서비스를 제공합니다. 입출력과 메모리 할당 같은 하드웨어 기능의 경우 OS는 프로세스와 하드웨어 사이의 중재 역할을 합니다. 유저는 OS에서 system call를 통해 커널을 불러옵니다. 이처럼 가장 기본적인 기능들을 실행시켜줍니다.

3. Instruction

3-1. Main

Main에서는 처음 입력하는 command를 받습니다. 이때 white space 또한 받을 수 있도록 하기 위해 fgets라는 함수를 사용하여 입력을 받았습니다. 이때, fgets로 받은 command를 배열로 저장하고 이 때 생기는 newline('\n')을 지우기 위해 배열의 크기를 strlen으로 구한다음 마지막에 null('\0')을 입력합니다. 이렇게 만든 command를 토큰화 시키는 작업을 수행합니다. 이때 white space로 구분하고 값을 string 배열에 저장합니다. 다음으로 입력받은 command가 quit일 경우 command와 quit을 비교하여 같을 경우 프로세스가 끝나도록 exit을 사용합니다. command에 들어온 값이 cd일 경우 함수를 사용하여 다르게 처리를 해주고 이 외의 경우에서는 getenv 함수를 통해 값을 확인합니다. getenv 함수를 사용하여 만든 path를 stat 함수로 확인하고 있다면 fork를 실행하도록 합니다. fork를 실행하면 parent 프로세스와 child 프로세스가 생성되며 parent 프로세스는 wait으로 멈춰두고 child 프로세스에서 execve을 실행하여 프로세스를 대체하게 만듭니다. execve 함수는 기존 프로세스를 대체하고 새로운 프로세스를 실행하게 하는 함수로써 만일 fork를 사용하지 않고 background에서 execve을 수행하면 background 프로세스 대신 ececve을 실행하여 Shell이 끝납니다. 또한 wait을 사용하는 것은 프로세스를 parent와 child로 나누어 졌을 때 종료된 child 프로세스로부터 return 값을 받기 위해서입니다. parent에

서 child 프로세스의 return 값을 커널을 통해 받지 못하는 경우 child 프로세스는 좀비 프로세스로 남게 됩니다. 이러한 점을 wait 함수를 통해 해결할 수 있습니다.

3-2. CD

CD 함수는 Shell 내에서 cd가 기능하도록 하기 위해 만든 함수입니다. cd는 다른 command와는 다르게 stat 함수를 사용해보면 환경변수에 저장되어 있는 command 가 아닙니다. 따라서 chdir 함수를 사용하여 cd와 같은 기능을 할 수 있도록 만들어 주었습니다. chdir 함수의 변수로 가고자 하는 directory를 입력하면 됩니다.

3-3. Stat

Stat 함수는 command와 path를 사용하여 만든 파일이름을 PATH 내에 있는지를 확인해 주는 함수입니다. 이를 통해 PATH에 있다면 이 기능을 실행할 수 있습니다.

3-4. Getenv

Getenv 함수는 main에서 입력 받은 command로 토큰을 만든 값과 PATH에서 가지고 오는 값을 strcat 기능을 통해 이어 붙입니다. getenv 함수를 사용하여 환경변수를 입력하면 그 환경변수 안에 어떠한 directory가 있는지 꺼낼 수 있습니다.

3-5. execve()

execve는 main 함수 안에서 동작하는 함수입니다. execve에는 3개의 parameter를 입력해야하는데 처음 들어가는 parameter는 파일이름입니다. 이 때 정확한 파일이름을 입력 해주어야합니다. 두 번째로 들어가는 parameter는 입력한 command를 string 배열을 사용하여 입력해줍니다. 첫 string에는 command가 들어가고 뒤에 들어가는 string은 앞의 command가 실행할 때 사용할 수 있는 인자들입니다. 이 때 주의해야하는 점이 string 배열에서 마지막에는 꼭 null('\0')을 입력해주어야 합니다. 마지막 parameter에는 환경변수로 null을 입력해주면 됩니다.

4. Program Structure

Loop

Get command

1, Command 토큰화 및 저장

2. Command 토큰 값이 quit일 경우 종료

3. Command 토큰 값이 cd일 경우 CD 함수 실행

4. 그 외의 경우 Getenv와 Stat 함수를 사용하여 값을 반환

    4-1. Getenv에서 path를 받아오고 main에서의 토큰 값과 붙이기

4-2. 완성된 파일이름을 Stat함수로 확인

5. main의 조건 확인

5-1. 반환한 값은 파일이름으로 execve 함수에서 사용

5-2. 반환된 값이 있다면 fork를 실행

5-3. child 프로세스에서 execve를 사용

5-4. parent 프로세스는 child 프로세스가 종료될 때까지 wait

5. Loop 종료

---

5. Problems and solutions

5-1. stat 함수 사용할 때 뒤에 newline이 있으면 안 된다는 것에 대해 확인하는 것이 어려웠습니다. 토큰화하여 만들어진 값을 보면 입력하고자 하는 값이 맞지만 실제 stat 함수에서 ret가 -1로 반환하는 것을 보면 찾지 못하였다는 것을 알았고 이 문제에 대해 해결법을 찾기 위해 어떠한 문제가 있는지 보았습니다. 문제를 해결하기 위해서 newline을 지워주는 작업을 하였고 이를 통해 stat 함수에서 ret가 0을 반환하도록 하였습니다.

5-2. 환경변수에 대해 이해하는 과정에서 많은 어려움이 있었습니다. 추가구현 부분에서 PWD, USER 등의 환경변수는 getenv를 사용하면 값이 출력되었지만 TIME은 값이 null이었습니다. 이러한 문제를 해결하기 위해 setenv라는 함수를 사용하였습니다. setenv는 기존에 존재하는 환경변수 명일 경우 값을 바꿀 수도 있고 없는 경우에는 직접 생성살 수 있는 함수였습니다. 이를 통해 TIME을 직접 설정하였습니다. 이때 실시간으로 시간을 불러오기 위해 loop안에서 같이 돌게 하였습니다. 시간을 설정하기 위해 시간을 불러오는 time 함수를 사용하였고 헤더 파일로 time.h를 사용하였습니다. 하지만 특이한 점은 setenv로 변환한 환경변수는 값이 종속되지 않는 다는 점입니다. 수정된 변수 값이나 새로 추가된 환경변수 값은 실행 중인 프로그램에서만 유효하며 외부적으로 변경되지 않습니다. 즉, 프로그램의 실행 당시에만 값이 저장됩니다.

5-3. 시간 함수 time()은 크게 두 가지로 나눌 수 있었습니다. time_t로 반환하는 time()함수와 structure로 반환해주는 localtime()함수가 있었고 이 중에서 time()함수를 사용하였습니다. time()함수를 출력하기 위해 ctime이라는 시간을 문자 string으로 변환해주는 함수를 통해 출력하였습니다.

5-4. Makefile을 만들 때 사용하는 문법을 정리하였습니다. 특히 이때 tab을 사용하여 만들어 주는 것이 중요했습니다. 따라서 vi 설정에서 expandtab으로 설정되어 있다면 :set noexpandtab으로 설정해줘야 합니다. vim을 이용하여 Makefile을 작성합니다. 크게 5가지로 나눌 수 있는데 Target, Dependency, Recipe, Macro, Dummy Target이 있습니다. 이렇게 Makefile을 만들고 make문으로 실행파일을 생성합니다. 이 때 전역변수에 대해서 에러가 발생했습니다. Main에서 선언한 문자가

다른 함수에서 다시 선언이 되어서 생기는 문제였습니다. 이러한 문제점을 해결하기 위해서 코드에 작성하였던 전역변수를 지우고 parameter로 받아주는 작업을 하여 해결하였습니다.
5-5. echo 기능을 수행하기 위해 살펴보았습니다. Shell을 실행시키고 내부에서 echo를 보면 stat함수에서 ret = 0을 반환합니다. 하지만 echo $PATH를 실행시키면 안의 내용들이 나와야하지만 직접 만든 Shell에서는 나오지 않고 $PATH 자체를 문자열로 출력하였습니다. 이러한 부분이 execve를 사용할 때 $PATH가 string 배열에 문자열로 들어가 출력이 되어서 그렇다는 것을 알게 되었습니다. getenv(PATH)를 string 배열에 집어넣어 보니 getenv함수가 실행하는 것이 아닌 getenv(PATH) 자체 문자열이 출력되었습니다. 이 부분에 대해서는 문제점을 파악했지만 해결해 주지는 못했습니다.

6. Build enviroment

Compliation : Linux Assam, with GCC

7. Additional

7-1. Makefile을 통해서 Shell이 돌아가도록 하였습니다.
7-2. Command에 PATH, HOME, USER, TIME, PWD 등 환경변수가 입력될 경우 처음 문자가 대문자인 경우 getenv를 통해 환경변수를 출력하도록 하였습니다.
7-3. Command를 받을 때 추가적인 인자들을 더 집어넣어 추가적인 기능들이 가능하도록 하였습니다.
ex) ls -a, ls -alh, cp test.c test1.c, mkdir test, rmdir test ...
7-4. getenv에서 TIME 환경변수 값이 출력되도록 하였습니다.

8. Personal feelings

Shell을 만들어보면서 생각보다 흥미로운 생각을 많이 했습니다. 과거 모바일프로세스 시간과 비교해보면 CPU에 대한 이해보다는 우리가 자주 사용하고 알지만 실제적으로는 어떠한 방법으로 돌아가는지 생각해보지 못한 것에 대해서 기능적으로 알아보고 배우는 시간이어서 더 좋았습니다. Shell을 통해 command를 받아들여서 받아들인 command에 따라 어떻게 작동하는지 살펴보는 것이 꽤 흥미 있는 작업이었습니다. 특히 이번 과제를 진행하면서 Shell 내에서 돌아가는 부분들에 대해 일일이 검색하는 과정을 통해 특히 fork라는 부분에 대해 많이 알아보았습니다. fork는 특히 중요하게 다룬 이유가 forkbomb이라는 문제를 일으킬 수도 있기 때문입니다. child 프로세스에서 변경한 내용이 parent 프로세스에서 작동하지 않는다는 것이 신기했습니다. 종속된 프로세스에서는 상속된 프로세스에 영향을 줄 수 없다는 점을 보면서 프로세스를 진행할 때 정해야 하는 순서에 대해서도 생각해보는 시간이었습니다.

9. Code

Shell.h

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <errno.h>
#include <ctype.h>
#include <time.h>
#define MAX_LINE 256

int Stat(int argc, char* argv);
char* Getenv(int argc, char* argv[]);
void CD(int argc, char* argv);
```

Shell.c

```
#include "Shell.h"
int main(int argc, char* argv[])
{
        char buf[MAX_LINE];
        char buf1[255];
        char *item;
        char *token[100], *saveptr;
        int i, j;
        int res;
        char *arr = (char*)malloc(sizeof(char)*100);
        pid_t pid;
        getcwd(buf1, 255);
        while(1)
        {
                printf("SiSH:%s$ ", buf1);
                memset(buf, 0x00, MAX_LINE);
                fgets(buf, MAX_LINE, stdin);
                item = buf;
                item[strlen(item)-1]='\0';
                for(j = 0; ;item = NULL, j++)
                {
```

```
                token[j] = strtok_r(item, " ", &saveptr);
                if(token[j] == NULL)
                {
                        break;
                }
        }
        if(strncmp(buf,"quit", 4) == 0)
        {
                exit(0);
        }
        if(!strcmp("cd", token[0]))
        {
                CD(2, token[1]);
        }
        else
        {
                arr = (Getenv(2, &token[0]));
                if(!(arr == 0))
                {
                        pid = fork();
                        if(pid == 0)
                        {
                                int res = execve(arr, token, NULL);
                                if(res == -1)
                                {
                                        printf("Execve failure\n");
                                }
                                exit(0);
                        }
                        else if(pid < 0)
                        {
                                perror("fork error");
                                return 0;
                        }
                        else
                        {
                                wait(0);
                        }
                }
        }
}
```

```
	return 0;
}
```

cd.c

```
#include "Shell.h"
void CD(int argc, char* argv)
{
	char *path;
	if(argc > 1)
	{
		path = argv;
	}
	//else if((path = (char*)getenv("HOME")) == NULL)
	//{
	//	path = ".";
	//}
	if(chdir(path) < 0)
	{
		printf("error : No directory\n");
	}
}
```

Getenv.c

```
#include "Shell.h"
char* Getenv(int argc, char* argv[])
{
	int i, j=0;
	int k = 0;
	char *env, *str;
	char *tok[100], *saveptr;
	char copy[MAX_LINE];
	char *p;
	time_t timer;
	p = argv[0];
	char *arr = (char*)malloc(sizeof(char)*100);

	if(argc == 1)
	{
		printf("usage: getenv env_vars ... \n");
		return 0;
	}
```

```
else
{
        for (i = 0 ; i < argc-1 ; i++)
        {
                env = getenv("PATH");
                strcpy(copy, env);
                for (j=0, str = copy; ;str = NULL, j++)
                {
                        tok[j] = strtok_r(str, ":", &saveptr);
                        if (tok[j] == NULL)
                        {
                                break;
                        }
                }
        }
}
if(isupper(*p))
{
        if((strncmp(*argv, "TIME", 4))==0)
        {
                time(&timer);
                setenv("TIME", ctime(&timer), 1);
        }
        env = getenv(*argv);
        printf("%s=%s\n", *argv, env);
}
else
{
        for(k; k < j-1; k++)
        {
                if((strncmp(tok[k], *argv, 4))==0)
                {
                        if(Stat(argc, *argv))
                        {
                                strcpy(arr, *argv);
                                return arr;
                        }
                }
                strcpy(arr, tok[k]);
                strcat(arr, "/");
                strcat(arr, *argv);
```

```
                        if(Stat(argc, arr))
                        {
                                return arr;
                        }
                }
                printf("%s: command not found\n", *argv);
        }
        return 0;
}
```

Stat.c

```
#include "Shell.h"
int Stat(int argc, char* argv)
{
        struct stat fstat_buf;
        int i;
        int ret;
        if (argc == 1)
        {
                printf("usage: stat file_path ... \n");
                return 0;
        }
        else
        {
                for (i = 0 ; i < argc-1 ; i++)
                {
                        ret = stat(argv, &fstat_buf);
                        if (ret == -1)
                        {
                                return 0;
                        }
                        return 1;
                }
        }
}
```

Makefile을 위해 각각의 함수들을 각각 하나의 c파일로 만들었습니다.