운영체제

[Project1 Simple Shell]

전공 : 모바일시스템공학과
학번 / 이름 : 32143165 이동구
담당교수 : 유시환

1. 소개

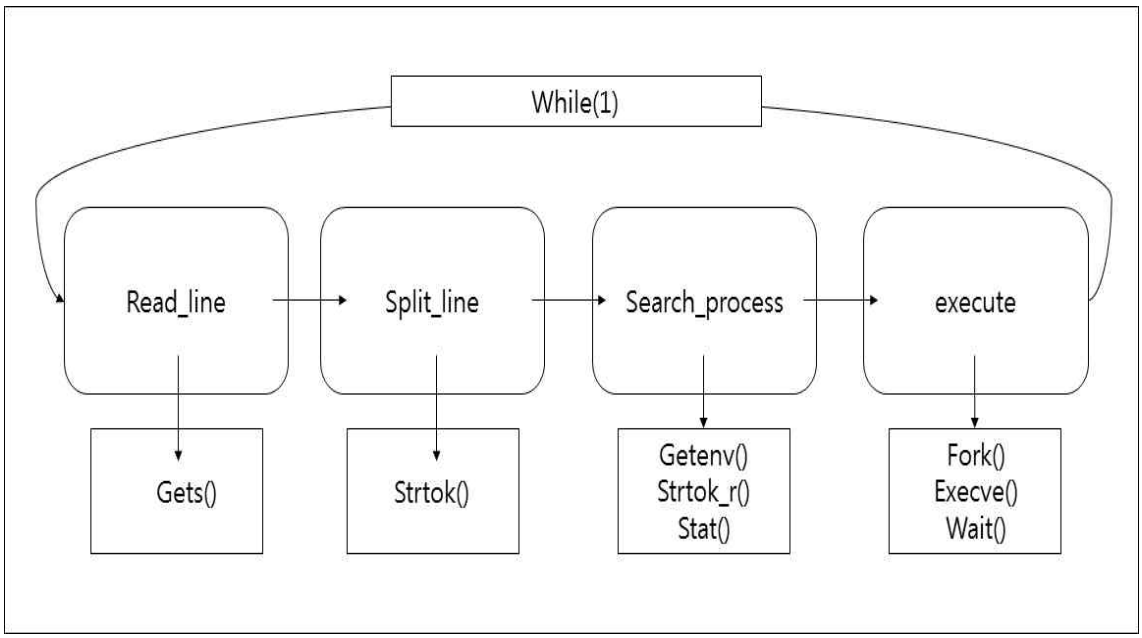
Shell은 사용자가 운영 체제와 직접 상호 작용 할 수 있는 하나의 작은 프로그램이다. 사용자는 쉘을 통해 프로그램을 실행하도록 명령 할 수 있다. 이를 위해 쉘은 사용자로부터 입력 문자열을 받아서 실행하게 된다. 지정된 프로그램이 실행을 완료하면 쉘은 다른 프로그램을 실행하기 위해 다른 입력을 받는다. 이번 과제는 그러한 기능을 하는 Simple shell을 제작해 보았다.

2. 동기

Shell은 일반적으로 시스템 호출과 같은 Low-level의 OS 기능과 함께 작동한다. 이를 구성하고 이해함으로써 좀 더 운영체제의 전반적인 기능에 대해 이해 할 수 있게 되었다.

3. 구성

Simple Shell은 크게 4개의 함수로 구현된다.
프로그램의 시작인 Read_line()함수는 Input을 받고 저장하는 역할을 하게 된다. 이후, Split_line() 함수는 저장한 Input을 tokenize를 통해 pointer array에 저장한다. 그리고 그 값을 이용하여 Search_process() 함수를 통해 tokenized된 값이 현재 PATH에 있는지 확인한다. 마지막으로 execute() 함수는 fork(), execve(), parent의 wait()를 하기 위한 함수라고 볼 수 있다.



<그림 1 : Simple Shell의 모형도 >

4. 코드 특징

1. main은 Loop안에 4개의 function이 들어가 있는 구조이다.

```
void main ()
{
        while(1)
        {
                read_line();
                split_line();
                search_process();
                execute();
        }
}
```

<그림 2. void main() >

2. NO_EXECUTE의 경우 잘못된 command나 환경변수를 이용할 경우 사용하는 변수이다.
   gets()로 Command를 받고 이를 저장하는 부분이다.
   프로그램을 종료하기 위한 quit 명령도 이곳에서 처리를 하게 된다.

```
void read_line(void)
{
        printf("INPUT THING : ");
        NO_EXECUTE = 0;
        gets(line);
        if (strcmp(line, "quit") == 0)
        {
                exit(0);
        }
}
```

<그림 3. Read_line() >

3. 받은 값을 띄어쓰기를 기준으로 tokenize하는 부분이다.

```
void split_line(void)
{
        char *token = NULL;
        int position = 0;

        token = strtok(line, " ");
        tokens[position] = token;
        position++;

        while(token != NULL)
        {
                token = strtok(NULL, " ");
                tokens[position] = token;
                position++;
        }
}
```

<그림 4. split_line() >

4. 가장 많은 부분을 차지하는 함수인 search_process() 함수 이다. 우선 가장크게 “/” 의 유무, 즉 처음 input에 경로의 유무를 판단해 바로 execute()로 넘어가는 것을 판단한다.

```
if(*(tokens[0]) != '/') // Find the path using 'getenv'
{
        if(isupper(*(tokens[0])) != 0)  // Just Using 'getenv'
        {
                savearray = getenv(tokens[0]);
                printf("%s : %s\n", tokens[0], savearray);
                NO_EXECUTE = 1;
                return;
        }

        else if(strcmp(tokens[0], "cd") == 0)
        {
                chdir(tokens[1]);
                NO_EXECUTE = 1;
                return;

        }

        savearray = getenv("PATH");
        strcpy(savepath, savearray);  /// initialize the path

        for (position = 0, token = savepath; ; token= NULL, position++) {
                savearray2[position] = strtok_r(token, ":", &saveptr);
                if (savearray2[position] == NULL)
                        break;
        }

        position++;
        for(int i = 0; i< position; i++)
        {
                temp = (char*)malloc(strlen(savearray2[i])-1);

                strcpy(temp, savearray2[i]);
                strcat(temp, "/");
                strcat(temp, tokens[0]);
                value = stat(temp, &stat_buf);
                PATH_VALUE++;

                if(value == 0){
                        tokens[0] = temp;
                        return;
                }
                else if(PATH_VALUE == 11){
                        NO_EXECUTE = 1;
                        printf("Wrong Command\n");
                        return;
                }
        }
}
else    // case of using "/"
{
        value = stat(tokens[0], &stat_buf);
        if(value == 0)
                return;
        else
                perror("stat error\n");
}
```

<그림 5. search process 전체 및 “/” 조건문 >

5. 대문자를 판단하여 input이 환경변수를 요하는 작업인지 판단한다. 또한 cd 명령어를 사용 할 경우 chdir() 함수를 사용하게 만든다.

```
if(isupper(*(tokens[0])) != 0)  // Just Using 'getenv'
{
        savearray = getenv(tokens[0]);
        printf("%s : %s\n", tokens[0], savearray);
        NO_EXECUTE = 1;
        return;
}

else if(strcmp(tokens[0], "cd") == 0)
{
        chdir(tokens[1]);
        NO_EXECUTE = 1;
        return;

}
```

<그림 6. isupper()를 사용하는 if문 >

5. 5의 두 가지 상황이 아닐 경우 getenv("PATH")를 통해 받은 경로를 tokenize한다.

```
savearray = getenv("PATH");
strcpy(savepath, savearray);  /// initialize the path

for (position = 0, token = savepath; ; token= NULL, position++) {
        savearray2[position] = strtok_r(token, ":", &saveptr);
        if (savearray2[position] == NULL)
                break;
}
```

<그림 5. Path Tokenize>

6. 이후 받은 경로에 "/"와 input을 붙혀 존재하는 실제경로인지 판단하게 만들었다.

```
position++;
for(int i = 0; i< position; i++)
{
        temp = (char*)malloc(strlen(savearray2[i])-1);

        strcpy(temp, savearray2[i]);
        strcat(temp, "/");
        strcat(temp, tokens[0]);
        value = stat(temp, &stat_buf);
        PATH_VALUE++;

        if(value == 0){
                tokens[0] = temp;
                return;
        }
        else if(PATH_VALUE == 11){
                NO_EXECUTE = 1;
                printf("Wrong Command\n");
                return;
        }
}
```

<그림 8. 경로 확인 코드 >

7. 마지막 excute는 처음 NO EXECUTE 값을 받을 경우 실행이 되지 않게 해놨으며, 그 외의 경우는 child에는 execve(), parent는 wait(), 다른경우는 error를 출력하게 만들었다.

```
void execute(void)
{
        if(NO_EXECUTE == 1)
                return;
        pid_t pid;
        pid = fork();
        if (pid == 0) {
                execve(tokens[0] , tokens , NULL );
        }
        else if (pid > 0){
                wait(0);
        }
        else if (pid < 0){
                perror("fork error");
        }
}
```

<그림 9. execute() >

5. Trial and Error

1. User가 입력한 program이 path내에 있는 지 확인하는 부분에서 동적 할당을 지정해 주지 않아 data가 손실이 되어 잘못 값이 출력되는 문제가 발생하였다. 이를 동적 할당을 해줌으로써 해결을 하였다.

2. 첫 번째 실행과 달리 두 번째 실행에서 getenv("PATH")를 받아올 때 첫 번째 path만 받아지는 오류가 발생하여, 이를 strcpy함수를 활용하여 다른 pointer array에 복사하고 복사한 값을 이용하여 path를 수정하게 만들었다.

3. 처음 잘못된 커맨드 혹은 getenv 환경변수를 사용 할 경우 계속 fork()를 하여 잘못된 child process를 만드는 오류가 발생하였다. 이를 NO_EXECUTE 변수를 생성하여 getenv 환경변수를 사용할 경우와 잘못된 command를 입력 할 경우, cd명령어를 사용할 경우 NO_EXECUTE를 1을 주어, execute() 함수 시작에서 해당변수(NO_EXECUTE)를 확인하게 하여, 1일 경우 필요 없는 실행을 하지 않게 하였다.

4. Input에 해당하는 path를 찾을 때 그 경로가 없는 경우, 프로그램을 실행시키지 않고 error를 출력해야 하는데 이때 error에 대한 적절한 조건이 없어 임의의 value를 생성하고 존재하는 path를 찾을 때마다 1씩 증가하게 하여 모든 path를 찾고 나서도 없을 경우 NO_EXECUTE = 1을 하고 return을 하게 만들었다.

6. 구성 환경

Linux Assam, with GCC
Upload to Github.

7. 느낀 점

운영체제 첫 과제를 진행하면서, 지금껏 사용해보지 못한 다양한 함수, 예를 들어, fork(), execve(), wait()등의 함수를 사용하고 shell이라는 것을 이해하는 과정이 흥미롭고 재미있게 느껴져서 이번 과제를 진행하면서 크게 답답했던 부분은 없었던 것 같다. 사실 1학기에 이러한 프로젝트를 하면서 수도코드의 중요성을 많이 느꼈던 터라, 이번 프로젝트 역시 코드를 짜고 진행할 때보다 수도코드를 짜고 프로젝트를 이해하는데 더 오랜 시간이 걸린 것 같다. 다만 아직 이번과제의 경우 난이도 자체는 그렇게 어려운 과제가 아니라 오랜 시간이 걸린 것 같지는 않다. 추가적으로 이번부터 운영체제 과제부터 github를 이용하여 작업을 하였는데, 예전부터 이용은 해보고 싶었지만 자세하게 사용방법을 몰라 아쉬운 부분이 많았는데, 수업을 통해 조금씩 사용하고 알게 되어 여러모로 배워가는 것이 많았던 것 같다. 앞으로의 과제가 더욱 기대가 된다.