운영체제 2 조 보고서

팀장 : 2017112168_이주호

팀원 : 2018111688_전나연

팀원 : 2018111794_신태섭

팀원 : 2020112166_윤영서

2022 Operating System Project

1. 주제

- AWS 환경에서의 Mini Operating System 구현

2. 목적

이번 프로젝트의 목적은 리눅스 시스템의 이해와 응용에 있다. 본 프로젝트는 C 언어를 통해서 리눅스 시스템을 구현함에 있다. 리눅스의 기본 동작 구조와 체계를 이해하고 동일하게 작동 가능한 시스템을 구현한다.
또한 AWS 환경을 사용하면서 클라우드 기본 동작 구조와 체계를 이해하고 동일하게 작동 가능한 시스템을 구현한다.

3. 내용

본 프로젝트에서 구현하는 내용은 다음과 같다.

- AWS EC2(t2.micro)에 리눅스 시스템과 동일한 파일 탐색기 구조를 구현한다. (자료구조 알고리즘 사용)
- 각 팀의 모든 인원은 본인의 AWS 계정이 있어야 하며 EC2 를 하나씩 생성해야 함
- **팀장의 EC2 에는 파일 탐색기 구조 코드가 있어야 하며 팀원은 본인의 로컬환경에서 SSH 프로그램을 통해 본인의 EC2 접속 후 EC2 에 저장한 key 를 사용하여 팀장의 EC2 에 접속가능해야 한다.** (그 외 IP 에서는 팀장의 EC2 에 접속 불가능하게 할 것)
- 서울 리전 내 구성 (VPC)
- Subnet 구성
- SSH 로 접속할 수 있는 환경
- **필수 구현 명령어** : cd, mkdir, pwd, ls, rm, cat, cp, find

- 명령어 별 수업 시간에 설명한 옵션들 구현
- cat 명령어를 통해서 파일 생성 및 읽기 구현
- 파일 입출력을 통해서 폴더 및 파일 생성된 현황을 저장하고 읽어야 함
- mkdir 명령어를 통해서 다수의 폴더를 동시에 생성할 수 있어야 함
- 위 명령어 이외의 명령어 구현 시 추가 점수 (최대 3 개)
- mkdir 명령어에서 다수의 폴더 생성 시 멀티스레딩을 이용하여 동시에 생성할 것
- 이 외의 명령어에서도 동시 작업 발생 시 멀티스레딩 적용
  (어느 작업에서 동시 작업이 발생하는지 잘 생각해 볼 것)

4. 제출 및 발표 기간

- 최종 보고서 및 총 프로젝트 제출 : **(6 월 7 일 화요일 23:59 분까지 제출)**

- 발표 자료 제출: **(6 월 8 일 수요일 23:59 분까지 제출)**

- 발표 날짜 : **(6 월 9 일 목요일 실습시간에 진행)**

5. 참고 사항

- 최종보고서는 워드를 사용하여 작성
- 목차는 하단 양식을 참고 자유롭게 작성
- 팀 회의 일지 작성
- 제출 예제 : 오전_X 조_OS 프로젝트
- 완성된 프로젝트의 소스 파일, 실행 파일은 모두 한 디렉토리에 넣고 tar.gz 로 압축제출
- 발표는 10 분 이내로 자기 조만의 장점을 설명, 시연 시간 6 분 이내
- 제출본과 시연파일은 동일해야 합니다.
  (제출 기한 이후 메인 코드 수정날짜, 내용 변경 시 0 점 처리)

- 조교가 테스트 할 때 실행되지 않으면 0 점 처리 될 수 있습니다.
- 프리티어 내에서 AWS Terms 에 따라 주어진 크레딧과 리소스 내에서 사용할 것
- 위 부분들은 팀의 모든 팀원이 실시간으로 확인해야할 의무가 있으며 확인하지 않아서 생기는 문제는 개인의 책임
- AWS 의 모든 자원을 Cryptomining 용도로 사용시 향후 문제 소지가 있을 수 있으며 이로써 발생하는 문제는 개인의 책임이므로 주의할 것.
- 본인의 로컬환경에서 SSH 프로그램을 통해 본인의 EC2 접속 후 해당 EC2 에 저장한 key 를 사용하여 팀프로젝트가 진행중인 메인 EC2 에 접속한 후 파일 탐색기 코드 시연

목차

1. 프로젝트 목표 및 방향

1.1. 추진목표

AWS 환경에서의 Mini Operation System 구현

1.2. 연구의 목적

리눅스 시스템의 이해하고 C 언어를 통해서 리눅스 시스템을 설계, 구현한다. 명령어와 명령어 옵션 동작 시 Process 와 Thread 를 사용하여 명령을 처리하게 함으로써 Linux 파일, 디렉토리 관리, Process 와 Thread 에 대한 이해를 향상시킨다. 이 내용을 기반으로 Linux 조작방법을 익히고 Process 와 Thread 사용을 통해 파일탐색기를 설계하기 위한 방법을 배운다. Linux 의 기본 동작 구조를 이해하고 작동 가능한 시스템을 구현한다.

또한 AWS 환경을 사용하면서 클라우드 기본 동작 구조와 체계를 이해하고 동일하게 작동 가능한 시스템을 구현한다. 팀원들은 SSH 프로그램을 통해 파일 탐색기 구조 코드가 있는 팀장의 EC2 인스턴스에 접속 가능하다.

1.3. 연구과제의 필요성

우선 c 언어라는 우리에게 익숙한 프로그래밍 언어를 이용해 Linux 에서 기본적인 작업 환경을 구축하고 Linux 와 Linux 명령어를 직접 설계함으로써 Linux 라는 운영체제의 이해도를 높일 수 있다.

AWS EC2 의 사용법을 익히고 활용할 수 있다. 또한 Putty 에서 SSH 프로그램을 활용하여 원격으로 다른 EC2 인스턴스에 접속 할 수 있다.

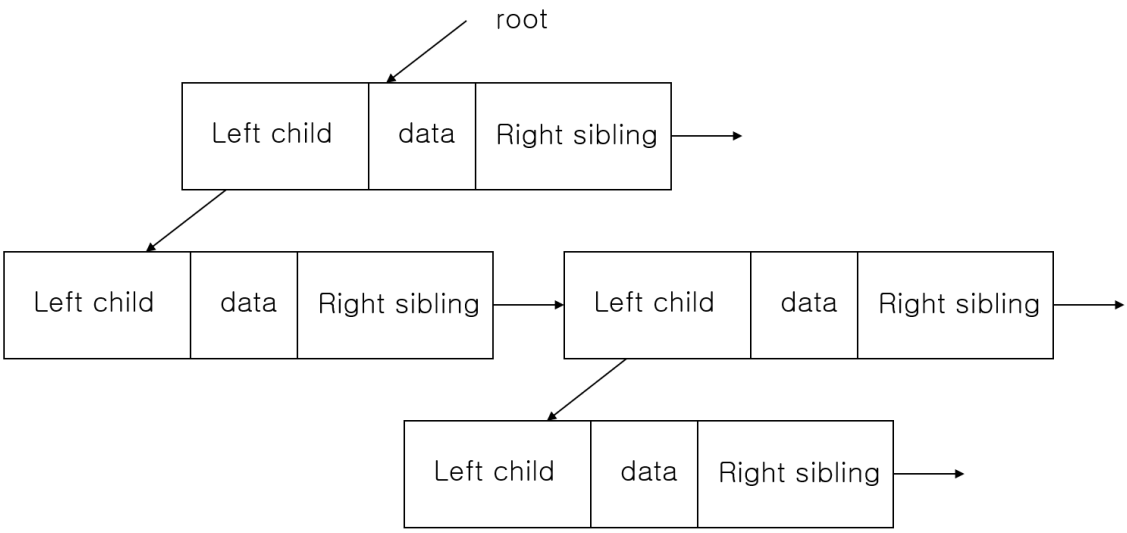
더불어 배열과 포인터, 구조체에 대한 이해를 높일 수 있고 Linux 환경의 파일, 디렉토리 관리 함수를 익힐 수 있다.

1.4. 관련연구

유닉스로부터 발전한 리눅스는 윈도우보다 짧은 역사를 가지며, 1990 년대 초 리누즈 토발즈에 의해 개발된 리눅스 커널에 기반을 둔다. 리눅스는 오픈 소스로 커널 개발 후 여러 개발자가 함께 만들어 지금의 리눅스에 이르렀다. 이는 사용자도 개발자가 될 수 있는 독특한 운영체제이다. 운영체제의 가장 중요한 역할은 파일시스템 관리이다. 리눅스 시스템 역시 파일을 생성하고 기록을 저장하고, 읽어 들이고, 지우는 등의 작업을 하는 독자적 파일시스템을 운영한다. 리눅스 커널에서 지원하는 파일 핸들링 함수는 어떤 것이 있으며 c 프로그래밍을 할 때 사용하는 표준 입출력 라이브러리들과는 어떤 관계가 있는지 알아보고 프로그램을 구현한다.

2. LCRS

2.1. LCRS(Left-Childe-Right-Sibling)의 구조



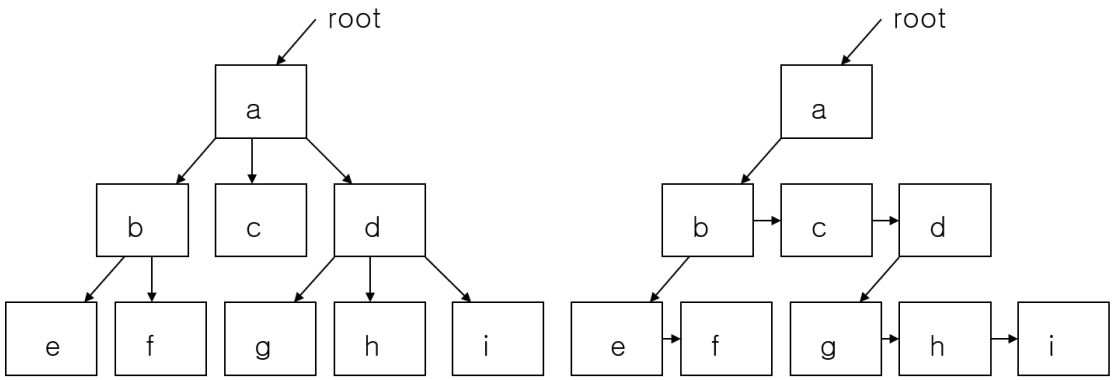
일반적인 Tree 구조로는 파일 탐색기의 중요한 특성인 폴더, 파일을 구분 짓기에 쉽지 않다. 그러므로 파일탐색기의 구조를 생성할 때 Left-Child-Right-Sibling(LCRS)의 구조를 사용하였다.

LCRS 구조는 시작점인 root 에서 Left child 로 포인터가 연결되고 한 LCRS 구조 안에 각각의 Left child, data, Right Sibling 이 포함되어 있다. 다른 LCRS 트리로 이동할 땐 다른 LCRS 트리의 Left child 에 포인터를 연결시켜 포인터를 따라서 각각 이동할 수 있다.

여기서 Child 란 자식 노드를 의미하고 Sibling 이란 같은 level 에 있는 노드를 의미한다. LCRS 이진 트리의 특징은 루트 노드를 제외한 모든 노드가 부모 노드의 왼쪽 자식 노드이거나 혹은 오른쪽 자식 노드인 것을 말한다. 즉 노드의 방향이 한쪽 방향으로만 치우쳐 있는 특성을 갖는다. 이러한 LCRS 구조를 이용하여 파일탐색기 구조를 생성하기로 결정했다.

2.2. 일반적인 트리와 LCRS 트리의 구조 비교

- regular tree LCRS



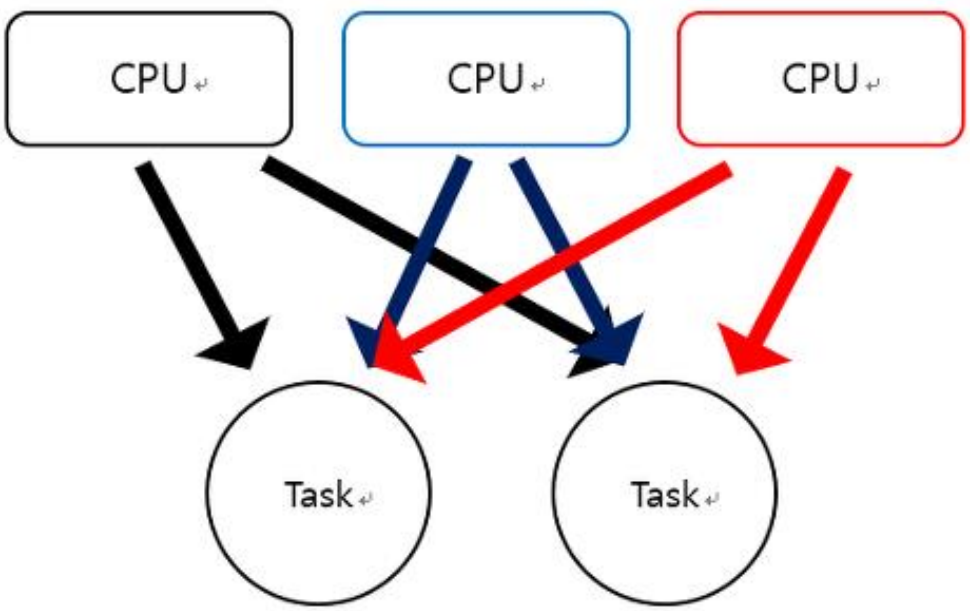
일반적인 트리는 계층적 자료구조 그래프의 일종인데, 그래프(Graph)는 vertex(정점)과 edge(간선) 으로 이루어졌다. 한 부모 노드로부터 자식 노드로 뻗어져 나가는 구조를 갖고 있으며 각각의 노드들이 어떤 한 노드의 부모 노드가 될 수도 있고 자식 노드가 될 수도 있다.

LCRS 구조에서는 같은 level 에서는 한 노드가 대표적으로 부모 노드가 되는 특성을 갖는다. 이러한 특성 덕분에 폴더(Directory)와 파일(File)의 특성을 표현하기에 적합하다. 예를 들어 위 사진에서 봤을 때 a 라는 한 폴더(directory)안에 b, d 라는 폴더(directory)와 c 라는 파일(file)을 갖고 있는 구조이고 b 폴더(directory)안에는 e 와 f 라는 파일을 갖고 있는 것이다. 또한 d 폴더 안에는 g, h, I 라는 파일을 갖고 있는 구조의 양상을 띤다. 이를 통해 파일과 폴더의 특성을 표현하여 파일탐색기를 구현했다.

3.멀티 프로세싱

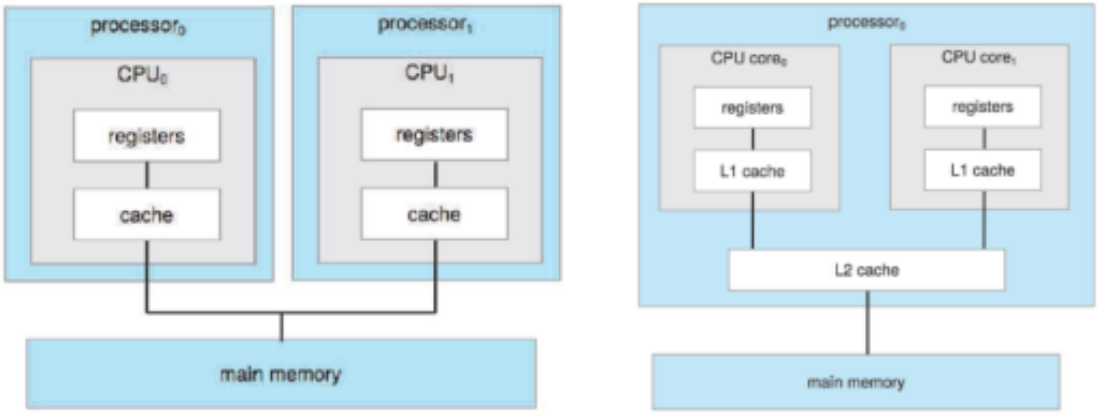
3.1. 멀티 프로세싱의 개념

멀티 프로세싱이란 하나의 응용 프로그램을 여러 개의 프로세스로 구성해서 각각의 프로세스가 하나의 작업을 처리하는 것이다. 다수의 프로세서가 서로 협력해서 일을 처리하는 구조이며, 컴퓨터는 1 대이고 프로세서는 2 개 이상이다. 멀티 프로세싱은 여러 프로세서와 메모리, 클럭, 시스템 버스, 그리고 주 메모리 장치 등을 공유한다. 각 프로세스는 자원이 다르게 할당되며 독립적으로 동작한다. 하나의 작업은 하나의 프로세서에서만 처리되는 것이 아닌, 다수의 프로세서에 의해 처리된다.



멀티 프로세싱은 여러 자식 프로세스들 중 하나의 프로세스에 문제가 발생해도 다른 프로세스들에 영향을 주지 않고 문제가 생긴 프로세스만 죽일 수 있어 신뢰성이 높다는 장점이 있다. 따라서 프로그램을 안정적으로 운용할 수 있다. 그리고 여러 개의 프로세스를 처리해야 할 때 동일한 데이터를 사용한다고 하면, 이러한 데이터를 한 개의 디스크에 두고 모든 프로세서가

이를 공유하면 비용적으로도 저렴하다. 또한, 프로세서가 2 개 이상 있으므로 빠르고 구현이 비교적 간단하다.



단점은 scheduling 에 따른 context switching 이 잦아서 성능이 저하될 우려가 있다는 것이다. Context switching 은 cache memory 초기화 등의 무거운 작업이 실행되므로 시간을 많이 낭비한다. 멀티 프로세싱은 메모리 사용량이 많으며, 프로세스 간 공유를 위해 통신을 하므로 까다롭다.

3.2. 파일 탐색기에 구현한 멀티 프로세싱 방법

```
// Multi Processing
int state;
int fd[2];
int k = -1;
char size_buffer[200];
pid_t process__id[3];
state = pipe(fd);

state = sem_init(&semp, 1, 1);
while (command != NULL) {
    if (k < 3) {
        process__id[++k] = fork(); // make child process
    }
    write(fd[1], size_buffer, 200);

    if (process__id[k] == 0) {
        sem_wait(&semp);
        read(fd[0], size_buffer, 200);
        Function__mkdir(structure, command, option);
        command = strtok(NULL, " ");
        write(fd[1], size_buffer, 200);
        sem_post(&semp);
    }
    else if (process__id[k] > 0)
        wait(&state);
    else
        printf("Error Exist!₩n");
}
sem_destroy(&semp);
```

멀티 프로세싱을 구현하기 위해서 fork()함수를 사용하였는데 fork() 함수를 통해서 한 프로세스의 자식프로세스를 만들 수 있다. 이 때 원래 동작하던 프로세스는 새로 만들어진 자식 프로세스의 부모 프로세스가 된다.

Mkdir 명령어가 동작할 때 만약 폴더를 여러 개를 한 번에 제작하고 싶다면 fork() 함수가 실행됨에 따라 pid_t process_id 배열에 새로운 자식 프로세스의 고유한 process_id 값이 저장되게 된다.

처음 생성된 프로세스는 process_id[k] == 0 이 되고 그 외의 자식들은 0 보다 큰 process id 를 갖게 되므로 그에 따라 구분 지어서 부모와 자식프로세스를 관리하여 mkdir 명령어에서 한 번에 여러 개의 폴더를 멀티

프로세싱을 통해 구현했다. 만약 존재하지 않는 process_id 가 처리되는 경우에는 에러 출력문을 띄우고 종료했다.

4. 코드 설명

- **C 언어 라이브러리 함수**

```c
#define TRUE 1
#define FALSE 0

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include <unistd.h>
#include <semaphore.h>
#include <wait.h>
```

우선 프로젝트를 위한 필요한 C 언어의 라이브러리 함수들을 정의해준다. 리눅스 환경에서 필요한 #include <unistd.h>, #include <semaphore.h>, #include <wait.h> 등을 사용했다.
코드의 가독성과 편리를 위해서 TRUE 를 상수 1 로 지정해 두고 FALSE 를 0 으로 지정했다.

- **Struct<File>**

```c
/// File, Directory, LCRS, Stack's struct
// 1) In linux, file's struct
typedef struct File {
    char name[50]; // file's name
    char content[1024]; // file's contents
    char viewtype; // file's viewtype
    int chmod__; // file's authority

    struct tm* t;
    struct File* Rsibling;
}file;
```

본 조에서 사용할 구조체들을 정의해준다. 우선 File 이라는 이름을 가진 파일을 나타내는 구조체를 정의한다. char name[50];으로 파일의 이름은 50 자로 제한하고 char content[1024];로 내용은 1024 자로 제한한다. char

viewtyp;은 문자형으로 하여 어떤 타입인지 정의해 놓은 것이며 int chmod__;로 허가와 관련된 내용을 정의한다. struct tm* t;로 파일이 어느 시간에 생성되었는지 알 수 있으며 File* Rsibling;의 R 은 right 를 의미하며 다음(next) 파일에 관한 내용을 포함한다.

- **Struct<Lcrsnode>**

```
29  // 2) In linux, Directory's struct
30  typedef struct Lcrsnode {
31      char name[50]; // Directory's name
32      char viewtype; // Directory's viewtype
33      int chmod__; // Directory's authority
34
35      struct tm* t; // Directory's Creation time
36      struct Lcrsnode* Pptr; // Upper Directory
37      struct Lcrsnode* Lchild; // Lower Directory
38      struct Lcrsnode* Rsibling; // Next DIrectory
39
40      file* infile; // In Directory's file ( LCRS's next nodes)
41  }directory;
```

리눅스 내에서 디렉토리를 나타내는 구조체이다. 이 구조체 안에서 위의 File 구조체 안의 요소들과 같은 이름으로 정의된 것들은 내용이 같으므로 설명을 생략하도록 한다. struct Lcrsnode* Pptr;에 P 는 parent 를 의미하며 상위 디렉토리를 뜻한다. struct Lchild;의 L 은 Left, Lower 를 의미하며 하위 디렉토리를 뜻한다. file* infile;은 디렉토리의 파일이며 LCRS 의 다음 node 를 뜻한다.

- **Struct<Lcrsnodestructure>, Struct<stack>**

```
43  // 3) Implementation LCRS__Structure
44  typedef struct Lcrsnodestructure {
45      directory* Hnode; // Upper node
46      directory* Cnode; // Current node
47  }tree;
48
49  // 4) Impementation Stack__Structure
50  typedef struct stack {
51      char* strname[2000]; // string's name
52      int top; // Initial top = -1
53  }stack;
```

Lcrsnodestructure 는 디렉토리를 나타내는 구조체이다. direcotory* Hnode;의 H 는 header 를 의미하며 상위 디렉토리를 뜻한다. direcotory* Cnode;의 C 는 current 를 의미하며 현재 디렉토리를 뜻한다.

stack 은 디렉토리 출력시에 사용되는 스택을 나타내는 구조체이다. char* strname[2000];으로 디렉토리의 이름을 2000 자로 제한하고 int top;으로 카운트용 변수를 정의해준다. top 의 초기값은 -1 이다.

- **Initial__File__Creation**

```
57  //// file, directory, tree's initial setup
58  // 1) Initial setup with file
59  file* Initial__File__Creation(char* __fileName) {
60      time_t timer; // Set timer
61      timer = time(NULL); // representation date
62
63      struct tm* temptime = (struct tm*)malloc(sizeof(struct tm));
64      file* temp = (file*)malloc(sizeof(file));
65
66      if (temp != NULL) { // when temp is not NULL
67          temp->chmod__ = 755; // initialized chmod's mode 755
68          strncpy(temp->name, __fileName, 20); // Copying file name
69
70          temp->Rsibling = NULL; // File's Right sibling is NULL;
71
72          if (__fileName[0] == '.')  // make file to hide in screen
73              temp->viewtype = 'u'; // view mode = 'u' mode
74          else
75              temp->viewtype = 's'; // view mode = 's' mode
76          memcpy(temptime, localtime(&timer), sizeof(struct tm));
77          temp->t = temptime;
78          return temp;
79      }
80  }
```

파일, 디렉토리, tree 의 초기값을 설정해준다. 초기 file 을 생성하는 함수이며 __fileName 을 입력 값으로 받는다. timer 를 설정해주고 timer = time(NULL);이면 날짜를 표현한다. malloc 함수를 이용해 struct tm* temptime 을 위한 공간을 할당해주며, 이에 필요한 크기는 sizeof(struct tm)이다. 마찬가지로 이 함수로 file* temp 를 위한 공간 또한 할당해주며, 이에 필요한 크기는 sizeof(file)이다. temp 가 not NULL 이면 chomd 의 node 는 초기값으로 755 를 가지며 파일 이름을 복사한다. 이 때 파일의 right sibling 은 비어 있다. 즉 파일은 생성시간, 권한, 이름에 관한 정보를 가진다.

만약 파일 이름 앞에 ' . '이 붙는다면 파일이 숨김 상태가 되며 viewtype 이 'u' mode 가 된다. 그렇지 않다면 viewtype 은 's' mode 가 된다. memcpy 함수는 간단하게 memcpy(복사 받을 메모리, 복사할 메모리, 길이)이다. 따라서 localtime(&timer)에 있는 원본을 sizeof(struct tm)의 길이 만큼 복사해서 temptime 에 넣는 것이다. 그 후 temp 값을 반환한다.

- **Initial__Tree__Creation**

```
82  // 2) Initial setup with tree
83  tree* Initial__Tree__Creation(directory* root) {
84      tree* temp = (tree*)malloc(sizeof(tree)); // Allocation in temp
85      if (temp != NULL) {
86          temp->Hnode = root; // Initialized header node NULL
87          temp->Cnode = root; // Initialized child  node NULL
88          return temp; // Return Directory
89      }
90  }
91  tree* structure;
```

처음으로 트리구조 생성할 때 쓰이는 함수이다. 이 함수는 루트 포인터를 함수 인자로 받아서 사용하는 함수이며 트리는 동적할당 받는 함수 tmep 가 NULL 이 아니면 템프의 포인트로 통해 상위 디렉토리, 자식 디렉토리로 가고 temp 값을 반환한다.

- **Initial__Directory__Creation**

```
93  // 3) Initial setup with directory
94  directory* Initial__Directory__Creation() {
95      time_t timer; // Set timer
96      struct tm* temptime = (struct tm*)malloc(sizeof(struct tm));
97      timer = time(NULL); // representation date
98
99      directory* temp = (directory*)malloc(sizeof(directory));
100     if (temp != NULL) {
101         temp->Lchild = NULL; // leftchild's NULL
102         temp->Rsibling = NULL; // rightchild's NULL
103         temp->Pptr = NULL; // Parent's NULL
104
105         strncpy(temp->name, "/", 20);
106
107         temp->chmod__ = 755; // initialized chmod's mode 755
108         temp->infile = NULL; // initialized infile NULL
109         temp->viewtype = 's'; // view mode = 's' mode
110
111         // copying memory
112         memcpy(temptime, localtime(&timer), sizeof(struct tm));
113         temp->t = temptime;
114         return temp;
115     }
116 }
117 directory* Leave__Directory(tree* dtree, char* dirname);
```

디렉토리 중 가장 상위 디렉토리인 root 를 생성하는 함수이다. timer 설정으을 통해 이 디렉토리가 언제 생성 되었는지 확인할 수 있다.

현재 어떤 디렉토리가 생성될지 알 수 없기 때문에 Lchild, Rsibling, Pptr 을 모두 NULL 값으로 설정해준다. 권한은 초기값을 755 로 설정해주고 infile 의 초기값은 NULL 로 설정해준다. viewtype 은 's' mode 로 설정해주고 **Initial__File__Creation** 에서와 같이 localtime(&timer)에 있는 원본을 sizeof(struct tm)의 길이 만큼 복사해서 temptime 에 넣고 temp 값을 반환한다.

- **Making command**

```
//// Making command (Init__Stack, Is__empty, Push, Pop)
// 1) Init__Stack
void Init__Stack(stack* pstack) {
    pstack->top = -1;
}
stack* Stack_Create()
{
    stack* pstack = (stack*)malloc(sizeof(stack));
    Init__Stack(pstack);
    return pstack;
}
stack* qstack;

// 2) Is__Empty
int Is__Empty(stack* pstack) {
    if (pstack->top == -1)
        return TRUE;
    else
        return FALSE;
}
```

명령들을 만드는 함수이다. 우선 Init__Stack 은 스택을 초기화하는 함수이다. top 의 초기값은 -1 로 설정해준다. stack* Stack_Create()는 stack 길이 만큼 복사해서 생성한다. Is__Empty 는 스택에 데이터가 비어있는지 확인하는 함수이다. 만약 top 의 값이 -1 이면 empty 값이 TRUE, -1 이 아니면 FALSE 를 반환한다.

```
140     // 3) Push
141     void Push(stack* pstack, char* str) {
142         pstack->top += 1;
143         pstack->strname[pstack->top] = str;
144     }
145
146     // 4) Pop
147     void Pop(stack* pstack) {
148         if (Is_Empty(pstack) == TRUE)
149             return;
150         else
151             pstack->top -= 1;
152     }
153
154     // set global pointer
155     sem_t semp;
```

Push 는 스택에 데이터를 삽입하는 함수이다. 데이터가 추가하면 top 의 값을 1 증가 시켜준다. strname 이 top 의 값에 접근해 값을 할당한다. Pop 은 스택에서 데이터를 추출하는 함수이다. 만약 스택이 비어 있다면 실행을 종료하고 아니라면 top 의 값을 -1 시켜준다.

전역 포인트 sem_t semp;를 선언한다.

- **Main 문**

```
157  // ------------------------------ main ---------------------------------
158  int main()
159  {
160     // Make Root with Directory, Tree, Stack
161     directory* root = Initial__Directory__Creation();
162     structure = Initial__Tree__Creation(root);
163     qstack = Stack_Create();
164
165     char* look__file; // look file
166     int len;
167     char command[100]; // command
168     char Buffer[100]; // buffer
169
170     // "recording.txt" is able to record past command's
171     // get a role with saving previous file, directory
172     // Invoke already created file "recording.txt"
173     FILE* fp = fopen("saving.txt", "a+");
174     if (fp != NULL) {
175        while (1) {
176           look__file = fgets(Buffer, sizeof(Buffer), fp);
177           if (feof(fp))
178              break;
179
180           // calculate file's length
181           len = strlen(look__file);
182
183           // Allocation command's str
184           char* command__str = (char*)malloc(len + 1);
185           strncpy(command__str, look__file, len - 1);
186
187           *(command__str + len - 1) = '\0';
188           User__Command(command__str, 0);
189
190        }
191     }
192     // close file
193     fclose(fp);
194
195
196     while (1)
197     {
198        printf("OS__team2 : "); // First defaulted name
199        Function__pwd(structure, qstack, 1); // First position
200
201        // get command
202        gets(command);
203
204        // carry out command with command number
205        User__Command(command, 1);
206     }
207  }
```

디렉토리, 트리, 스택으로 루트를 먼저 만들어준다. char* look__file 는 포인터를 사용해 파일을 볼 수 있다. command 와 Buffer 는 100 으로 제한한다. saving.txt 는 과거 명령을 기록할 수 있다. 이전 파일, 디렉토리를 저장하는 역할을 얻을 수 있으며 이미 만들어진 파일인 saving.txt 를 호출한다. 파일을 닫고, 메인 함수로 이전에 실행 되었던 명령어들을 다시 실행 시켜 이전 상황으로 되돌린다. 이후에 다시 명령을 시작한다.

- **Initial__Directory**

```
210  // Making new Directory ( node )
211  directory* Initial__Directory(char* str)
212  {
213     time_t timer;
214     timer = time(NULL); // representation of date
215
216     struct tm* time__tmp = (struct tm*)malloc(sizeof(struct tm));
217     directory* New__Directory = (directory*)malloc(sizeof(directory));
218
219     New__Directory->Lchild = NULL; // new Directory's lchild = NULL
220     New__Directory->Rsibling = NULL; // new Directory's rsibling = NULL
221     New__Directory->Pptr = NULL; // new Directory's Parent = NULL
222
223     strncpy(New__Directory->name, str, 20);
224
225     New__Directory->chmod__ = 755; // initialized chmod's mode 755
226     New__Directory->infile = NULL; // Directory's inside is NULL
227
228     if (str[0] == '.')
229        New__Directory->viewtype = 'u';
230     else
231        New__Directory->viewtype = 's';
232     // copying memory
233     memcpy(time__tmp, localtime(&timer), sizeof(struct tm));
234
235     New__Directory->t = time__tmp;
236
237     return New__Directory;
238  }
```

새로 만들어지는 디렉토리를 생성하는 함수이다. 새로 만들어진 디렉토리의 Lchild, Rsibling, Pptr 는 NULL 이다. 새로 만들어지는 디렉토리의 이름은 20 자로 제한한다. 권한의 초기값은 755, infile 은 NULL 이다. 즉, 이 함수는 디렉토리의 위치를 설정해 주지 않고 이름, 권한, 시간, 파일 유무, 파일 숨김 여부 등의 정보를 포함한다.

- **Connection__Directory**

```
241  // when creating lower directory
242  // connect upper directory's leftchild with new directory
243  void Connection__Directory(directory* parent, directory* child) {
244      directory* temp = NULL;
245
246      // if lefchild is NULL, connect this position
247      if (parent->Lchild == NULL) {
248          parent->Lchild = child;
249          child->Pptr = parent;
250      }
251      // if leftchild is full, connect directory's rsibling
252      else {
253          temp = parent->Lchild;
254          while (temp->Rsibling != NULL)
255          {
256              temp = temp->Rsibling;
257          }
258
259          temp->Rsibling = child;
260          child->Pptr = parent;
261      }
262  }
```

하위 디렉토리를 생성할 때 상위 디렉토리의 Lchild 와 새롭게 생성한 디렉토리의 노드를 이어주는 함수이다. 부모 노드의 Lchild 가 비어 있으면 그 자리에 이어주고 만약 다른 노드가 있다면 그 노드의 Rsibling 에 이어준다.

- **Function**

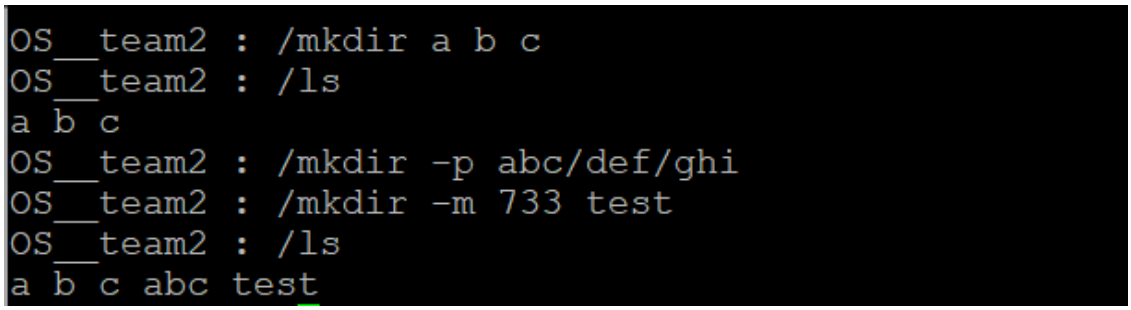
1) Function_mkdir

```
265  //// Command's Muster
266  // Command 1) mkdir
267  directory* Function__mkdir(tree* dtree, char* str, int option) {
268      if (strchr(str, " ") == NULL) {
269          if (Leave__Directory(dtree, str) != NULL) {
270              printf("Same Directory name is already existed!!\n");
271              return;
272          }
273      }
274      if (option == 0) {
275          directory* NewDir = Initial__Directory(str);
276          directory* Parent = dtree->Cnode;
277          Connection__Directory(Parent, NewDir);
278      }
279      else if (option == 1) { // Command = "mkdir -m" (give privilege)
280          directory* NewDir = Initial__Directory(str);
281          directory* parent = dtree->Cnode;
282          Connection__Directory(parent, NewDir);
283          return NewDir;
284      }
285      else {
286          // Command = "mkdir -p"
287          // ex) mkdir -p abc/def/ghi
288          directory* NewDir = Initial__Directory(str);
289          directory* parent = dtree->Cnode;
290          Connection__Directory(parent, NewDir);
291      }
292  }
```

mkdir 명령어를 구현하는 함수로 디렉토리를 생성하는 함수이다. 옵션이 0 일 때는 mkdir 명령어를 통해 디렉토리를 생성할 수 있다.

```
OS__team2 : /ls

OS__team2 : /mkdir a b c
OS__team2 : /ls
a b c
```

옵션이 1 일 때는 mkdir 뒤에 -m 이 붙었을 때이다. 이는 권한을 부여해 줄 수 있다. 옵션이 2 일때는 mkdir 뒤에 -p 가 붙었을 때이다. 이는 존재하지 않은 디렉토리까지 한번에 만들 수 있다.



2) Function__pwd

```
294  // Command 2) pwd
295  void Function__pwd(tree* dtree, stack* tempstack, int type) {
296      directory* present = dtree->Cnode;
297      if (present->Pptr == NULL) {
298          printf("/");
299          if (type == 0)
300              printf("\n"); // In case, root
301          return;
302      }
303      // Put every Directory in Stack
304      while (present->Pptr != NULL) {
305          Push(tempstack, present->name);
306          present = present->Pptr;
307      }
308      printf("/");
309
310      // Until Stack is empty, print every Directorys
311      while (1) {
312          if (Is__Empty(tempstack) == TRUE)
313              break;
314          else {
315              printf("%s/", tempstack->strname[tempstack->top]);
316              Pop(tempstack);
317          }
318      }
319      if (type == 0)
320          printf("\n");
321  }
```

pwd 명령어를 구현하는 함수로 현재 디렉토리의 경로를 출력할 수 있다. 만약 present->pPtr 이 NULL 이면 /를 출력하고 타입이 0 이라면 공백이 출력된다. 하지만 NULL 이 아니라면 스택에 있는 모든 디렉토리 노드들의 이름을 스택에 넣어 놓고 하나씩 pop 하며 출력한다.

```
OS__team2 : /pwd
/
OS__team2 : /cd abc
OS__team2 : /abc/cd def
OS__team2 : /abc/def/cd ghi
OS__team2 : /abc/def/ghi/pwd
/abc/def/ghi/
```

- **Leave__Directory, Leave__File**

```
323  // Out Directory
324  directory* Leave__Directory(tree* dtree, char* dirname) {
325      directory* tmp = NULL;
326      tmp = dtree->Cnode->Lchild;
327      while (tmp != NULL) {
328          if (strcmp(tmp->name, dirname) == 0)
329              break;
330          tmp = tmp->Rsibling;
331      }
332      return tmp;
333  }
334
335  // Out File
336  file* Leave__File(tree* dtree, char* filename) {
337      file* tempfile = dtree->Cnode->infile;
338      while (tempfile != NULL) {
339          if (strcmp(tempfile->name, filename) == 0)
340              break;
341          tempfile = tempfile->Rsibling;
342      }
343      return tempfile;
344  }
```

위의 두 함수는 LCRS 트리를 탐색하여 인자로 받은 디텍토리의 이름 혹은 파일의 이름이 탐색하는 노드 혹은 파일과 같다면 해당 노드 또는 파일을 반환하는 함수이다.

첫째로 Directory 를 leave 하는 함수이고 둘째로 File 을 leave 하는 함수이다.

3) cd__utill

```
347  // Command 3) cd
348  char* cd__utill(char* path) {
349     int len = strlen(path);
350     char* temp = (char*)malloc(len);
351     strncpy(temp, path + 1, len);
352     *(temp + len - 2) = '\0';
353     return temp;
354  }
355
356  int Function__cd(tree* dtree, char* path) {
357     if (strcmp(path, ".") == 0) // '.' : Current Directory
358        return TRUE;
359     else if (strcmp(path, "..") == 0) {
360        if (dtree->Cnode == dtree->Hnode)
361           return TRUE;
362        else {
363           dtree->Cnode = dtree->Cnode->Pptr;
364           return TRUE;
365        }
366     }
367     else {
368        if (Leave__Directory(dtree, path) == NULL)
369           return FALSE;
370        else {
371           dtree->Cnode = Leave__Directory(dtree, path);
372           return TRUE;
373        }
374     }
375  }
```

cd 는 사용자가 사용하고자 하는 디렉토리 경로로 이동해주는 명령이다. int len 의 값을 사용하여 temp 의 성공 유무를 파악한다. 입력 받은 디렉토리 경로를 사용하기 전에 양쪽 끝의 /를 제거하고 새로운 문자열로 받는 함수이다. 이는 절대경로를 통해 이동할 때 사용한다.

cd.은 현재 위치한 폴더로 이동한다. 사실상 기능은 새로고침 기능과 동일하다고 볼 수 있다. cd..는 상위 디렉토리로 이동하게 된다. 예를들어 현재 위치가 a/b 라면 a 로 이동하게 된다.

```
OS__team2 : /ls
a b c abc test
OS__team2 : /cd abc
OS__team2 : /abc/cd def
OS__team2 : /abc/def/cd ghi
OS__team2 : /abc/def/ghi/cd .
OS__team2 : /abc/def/ghi/cd ..
OS__team2 : /abc/def/cd ..
OS__team2 : /abc/cd ..
OS__team2 : /ls
a b c abc test
```

- **Change__Directory__position**

-

```
377  void Change__Directory__position(tree* dtree, char* path) {
378      char* str = NULL;
379      char temp[200];
380      directory* nodetemp;
381
382      strncpy(temp, path, 100);
383
384      nodetemp = dtree->Cnode;
385      if (strcmp(path, "/") == 0)
386          dtree->Cnode = dtree->Hnode;
387
388      else if (path[0] == '/') {
389          path = cd__utill(path);
390          str = strtok(path, "/");
391          dtree->Cnode = dtree->Hnode;
392          while (str != NULL) {
393              if (Function__cd(dtree, str) == FALSE)
394                  break;
395              str = strtok(NULL, "/");
396          }
397      }
398      else
399          Function__cd(dtree, path);
400  }
```

cd/는 ROOT 디렉토리로 이동한다.

또한 입력 받은 디렉토리명과 같은 노드가 있을 시에 그 노드를 반환하게 된다. 없다면 /만을 출력하게 된다.

- **Initial__cat**

```
402  // Command 4) cat
403  // Initialize cat
404  void Initial__cat(tree* dtree, char* filename, char* cat_content) {
405     int len;
406     file* temp__file = Initial__File__Creation(filename);
407     file* current__file = dtree->Cnode->infile;
408
409     if (current__file == NULL) {
410        dtree->Cnode->infile = temp__file;
411        strncpy(dtree->Cnode->infile->content, cat_content, 1024);
412     }
413     else {
414        if (current__file->Rsibling == NULL) {
415           current__file->Rsibling = temp__file;
416           strncpy(dtree->Cnode->infile->Rsibling->content, cat_content, 1024);
417        }
418        else {
419           current__file = current__file->Rsibling;
420
421           while (current__file->Rsibling != NULL)
422              current__file = current__file->Rsibling;
423
424           current__file->Rsibling = temp__file;
425           strncpy(temp__file->content, cat_content, 1024);
426        }
427     }
428  }
```

위 사진상에 있는 코드의 마지막 부분의 cmd 함수에서의 cmdoption 인자가 0 일 때 cat 을 초기화 해주기 위한 함수이다.

4) Function__cat

```
429  // Implementation cat
430  void Function__cat(tree* dtree, char* filename, int option) {
431     directory* tempnode = dtree->Cnode;
432     file* temp__file = dtree->Cnode->infile;
433     file* current__file = NULL;
434
435     if (option == 0) {
436        if (temp__file == NULL)
437           printf("File or Directory No exist\n");
438        else {
439           if (strcmp(temp__file->name, filename) == 0)
440              printf("%s", temp__file->content);
441           else {
442              while (temp__file->Rsibling != NULL) {
443                 temp__file = temp__file->Rsibling;
444                 if (strcmp(temp__file->name, filename) == 0) {
445                    printf("%s", temp__file->content);
446                    return;
447                 }
448              }
449              printf("File or Directory No exist\n");
450           }
451        }
452     }
```

```
454     // option 1 ==> cat '>'
455     else if (option == 1) {
456        if (temp__file == NULL) {
457           char temp[1024];
458           temp__file = Initial__File__Creation(filename);
459           fgets(temp, 1024, stdin);
460
461           strncpy(temp__file->content, temp, 1024);
462
463           tempnode->infile = temp__file;
464        }
465        else {
466           if (temp__file->Rsibling == NULL) {
467              char temp[1024];
468              temp__file->Rsibling = Initial__File__Creation(filename);
469              fgets(temp, 1024, stdin);
470              strncpy(temp__file->Rsibling->content, temp, 1024);
471           }
472           else {
473              current__file = temp__file->Rsibling;
474              while (current__file->Rsibling != NULL)
475              {
476                 current__file = current__file->Rsibling;
477              }
478              char temp[1024];
479              temp__file = Initial__File__Creation(filename);
480              fgets(temp, 1024, stdin);
481              strncpy(temp__file->content, temp, 1024);
482              current__file->Rsibling = temp__file;
483           }
484        }
485
486     }
487  }
```

cat 명령어는 주로 파일 내용을 보기 위해 사용되는 명령어이다. 즉 파일을 순서대로 읽고 그 내용을 읽은 순서대로 표준 출력할 수 있게 하는 명령이다. 본 조는 적은 내용의 파일의 생성하고, 이를 터미널 창으로 보여주는 식으로 구현했다. 인자로 받는 option 의 값이 1 일 경우에는 fgets 를 통해 문자열을 받을 수 있는데 cat 내용 > file 을 입력하면 내용을 file 에 입력해준다. 이는 strncpy 함수를 이용해 파일의 content 부분에 저장한다. option 이 0 인 경우에는 저장 되어있는 파일을 터미널 창으로 보여준다.

```
OS__team2 : /cat > abc
fadjoiiodajogoida
OS__team2 : /ls
abc
OS__team2 : /cat abc
fadjoiiodajogoida
```

5) Function__ls

```
490  // Command 5) ls
491  void Function__ls(tree* dtree, int option) {
492     int i = 0;
493     directory* current = dtree->Cnode;
494     directory* tmp;
495     directory* Dir__Disk[40];
496     if (current->Lchild == NULL) {
497
498     }
499     else {
500        tmp = current->Lchild->Rsibling;
501        if (tmp == NULL)
502           Dir__Disk[i++] = current->Lchild;
503        else {
504           Dir__Disk[i++] = current->Lchild;
505           while (tmp != NULL) {
506              Dir__Disk[i++] = tmp;
507              tmp = tmp->Rsibling;
508           }
509        }
510     }
```

```
512     if (option == 0) {
513        int num = 0;
514        while (num < i) {
515           if (Dir__Disk[num]->viewtype == 's')
516              printf("%s ", Dir__Disk[num]->name);
517           num++;
518        }
519        if (current->infile != NULL)
520           PrintFile(structure, option);
521        printf("\n");
522     }
523     else if (option == 1) { // ls-a
524        int num = 0;
525        while (num < i) {
526           struct tm* dirtime = Dir__Disk[num]->t;
527           if (Dir__Disk[num]->viewtype == 's') {
528              printf("d");
529              Authority__Directory(Dir__Disk[num]);
530              Number__LCRS__Inside(Dir__Disk[num]);
531              printf("4096  ");
532              printf("%03d-%03d %03d:%03d:%03d    ",
533              dirtime->tm_mon + 1, dirtime->tm_mday, dirtime->tm_hour, dirtime->tm_min, dirtime->tm_sec);
534              printf("%s\n", Dir__Disk[num]->name);
535           }
536           num++;
537        }
538        if (current->infile != NULL) {
539           PrintFile(structure, option);
540        }
541        printf("\n");
542     }
```

```
543     else if (option == 2) { // ls-l
544         int num = 0;
545         while (num < i) {
546             printf("%s ", Dir__Disk[num]->name);
547             num++;
548         }
549         if (current->infile != NULL)
550             PrintFile(structure, option);
551         printf("\n");
552     }
553     else {
554         int num = 0;
555         while (num < i) {
556             struct tm* dirtime = Dir__Disk[num]->t;
557             printf("d");
558             Authority__Directory(Dir__Disk[num]);
559             Number__LCRS__Inside(Dir__Disk[num]);
560             printf("4096  ");
561             printf("%02d-%02d %02d:%02d:%02d    ", dirtime->tm_mon + 1, dirtime->tm_mday,
562             dirtime->tm_hour, dirtime->tm_min, dirtime->tm_sec);
563             printf("%s\n", Dir__Disk[num]->name);
564             num++;
565         }
566         if (current->infile != NULL)
567             PrintFile(structure, option);
568         printf("\n");
569     }
570 }
```

ls 는 디렉토리 내부의 파일과 하위 디렉토리 목록을 출력하는 명령어이다. option 이 없는 경우 즉 특정 디렉토리를 입력하지 않는 경우에는 현재 폴더에 관해서 명령이 실행된다. 만약 option 이 1 이라면 ls-a 가 실행된다. a 은 all 이며 숨겨진 파일이나 디렉토리까지포함해서 목록을 자세히 출력한다. option 이 2 일 때는 ls-l 이 실행되는데 이는 현재 디렉토리의 정보 혹은 사용자가 원하는 다른 디렉토리의 정보를 읽어오고 해당 결과를 출력해준다.

Authority__Directory 를 이용해서 함수에 대한 출력값과 모든 디렉토리와 파일, 그리고 현 시각이 동시에 촐력된다.

```
OS__team2 : /ls -l
drwxr-xr-x 1    4096  06-02 03:43:10    b
drwxr-xr-x 1    4096  06-02 03:43:10    c
drwx------ 1    4096  06-02 03:46:19    test
-rwxr-xr-x 1    1024  06-02 04:22:01    hello.txt

OS__team2 : /chmod 770 hello.txt
OS__team2 : /ls -l
drwxr-xr-x 1    4096  06-02 03:43:10    b
drwxr-xr-x 1    4096  06-02 03:43:10    c
drwx------ 1    4096  06-02 03:46:19    test
-rwxrwx--- 1    1024  06-02 04:22:01    hello.txt
```

- **Preorder__LCRS**

```
572  // representation of Tree (Preorder Tree)
573  int preorder__LCRS(directory* dir, int num) {
574      if (dir != NULL) {
575          num++;
576          if (dir->infile == NULL) {
577              num = preorder__LCRS(dir->Lchild, num);
578              num = preorder__LCRS(dir->Rsibling, num);
579          }
580          else {
581              file* temp = dir->infile;
582              while (temp != NULL) {
583                  num++;
584                  temp = temp->Rsibling;
585              }
586              num = preorder__LCRS(dir->Lchild, num);
587              num = preorder__LCRS(dir->Rsibling, num);
588          }
589      }
590      return num;
591  }
```

dir 이 비어있지 않다면 num 을 증가시켜준다. 만약 dir->infile 이 NULL 이라면 num 값은 dir 값이 각각 Lchilde, Rsibling 에 저장된다. 또는 temp 값이 NULL 이 아닐 동안에 num 은 증가되고 temp->Rsibling 이 된다.
전위출력으로 나타내주기 위함이다. (LCRS)

- **Number__LCRS__Inside**

```
593  // return preorder__LCRS's value
594  // minus number of searching temp node's level node
595  int Number__LCRS__Inside(directory* dir) {
596      int num = 0;
597      num = preorder__LCRS(dir, 0);
598      if (dir->Rsibling != NULL) {
599          directory* temp = dir->Rsibling;
600          while (temp != NULL) {
601              num--;
602              if (temp->Lchild != NULL) {
603                  directory* temp1 = temp->Lchild;
604                  while (temp1 != NULL) {
605                      num--;
606                      temp1 = temp1->Lchild;
607                  }
608              }
609              temp = temp->Rsibling;
610          }
611      }
612      printf("%d", num);
613      printf("   ");
614  }
```

초기값은 0 으로 설정해주고 만약 dir->Rsibling 이 NULL 이 아니면 directory* temp = dir->Rsibling; 가 된다.

- **Authority__Directory, Authority__File**

```
616  // Authority__Directory
617  // get Directory's authority from number and show each position
618  int Authority__Directory(directory* dir) {
619      int chm = dir->chmod__;
620      int i = 0;
621      int k = 100;
622      int num;
623
624      while (i < 3) {
625          num = chm / k;
626          chm = chm - num * k;
627          k = k / 10;
628          switch (num) {
629          case 0:
630              printf("---");
631              break;
632          case 1:
633              printf("--x");
634              break;
635          case 2:
636              printf("-w-");
637              break;
638          case 3:
639              printf("-wx");
640              break;
641          case 4:
642              printf("r--");
643              break;
644          case 5:
645              printf("r-x");
646              break;
647          case 6:
648              printf("rw-");
649              break;
650          case 7:
651              printf("rwx");
652              break;
653          }
654          i++;
655      }
656
657
658      printf(" ");
659  }
```

```
662  // Authority__File
663  // get file's authority from number and show each position
664  void Authority__File(file* fi) {
665     int chm = fi->chmod__;
666     int k = 100;
667     int i = 0;
668     int num;
669     while (i < 3)
670     {
671        num = chm / k;
672        chm = chm - num * k;
673        k = k / 10;
674        switch (num)
675        {
676        case 0:
677           printf("---");
678           break;
679        case 1:
680           printf("--x");
681           break;
682        case 2:
683           printf("-w-");
684           break;
685        case 3:
686           printf("-wx");
687           break;
688        case 4:
689           printf("r--");
690           break;
691        case 5:
692           printf("r-x");
693           break;
694        case 6:
695           printf("rw-");
696           break;
697        case 7:
698           printf("rwx");
699           break;
700        }
701        i++;
702     }
703     printf(" ");
704  }
```

Authority__Directory 는 디렉토리의 권한을 받고, Authority__File 은 인자로 파일의 권한을 받아 각 세자리 숫자에 따라 문자열로 각 자리는 나타내준다. 두 함수의 차이는 디렉토리와 파일 중 어떤것의 권한을 받느냐 이다. 허가권이란 필드 확인 명령어라고 하며 허가권 필드는 사용자(User), 그룹(Group), 기타 사용자(Other)의 의미이다. r 은 읽기로 디렉토리가 가지고 있는 파일 목록을 읽은 수 있는 권한을 뜻하며 w 는 쓰기 즉 파일을 삭제 하거나 생성한다. x 는 실행을 뜻하며 – (bar)은 실행권한이 없다는 것이다.

```
OS__team2 : /chmod 770 hello.txt
OS__team2 : /ls -l
drwxr-xr-x 1    4096  06-02 03:43:10    b
drwxr-xr-x 1    4096  06-02 03:43:10    c
drwx------ 1    4096  06-02 03:46:19    test
-rwxrwx--- 1    1024  06-02 04:22:01    hello.txt
```

- **PrintFile**

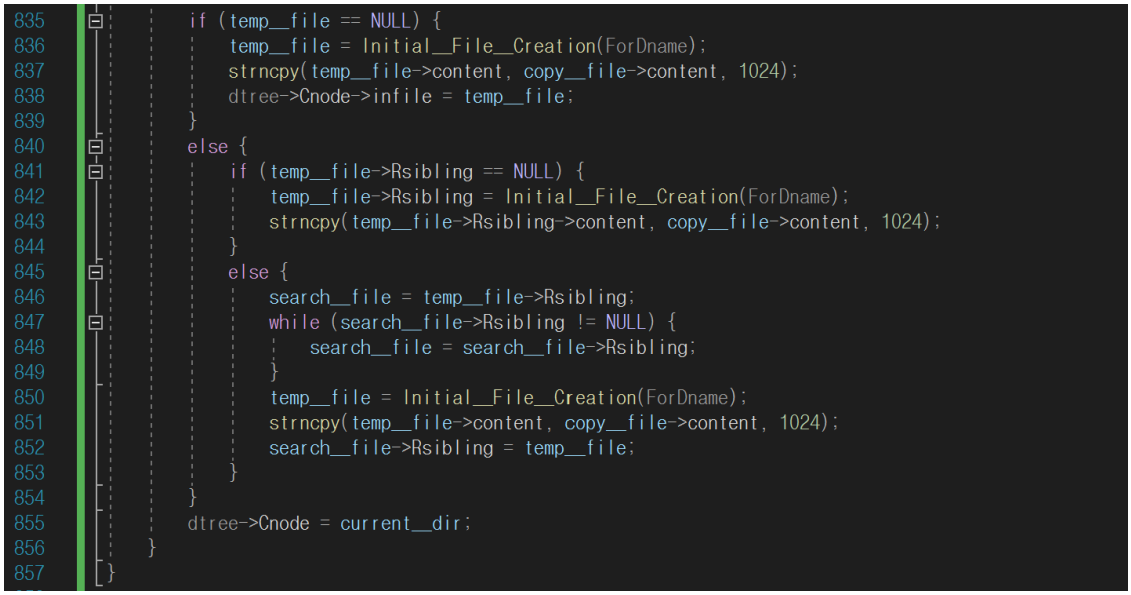
```
// Print__File
int PrintFile(tree* dtree, int option) {
    file* temp__file = dtree->Cnode->infile;
    if (option == 0) {
        if (temp__file->viewtype == 's')
            printf("%s  ", temp__file->name);
        while (temp__file->Rsibling != NULL) {
            temp__file = temp__file->Rsibling;
            if (temp__file->viewtype == 's')
                printf("%s ", temp__file->name);
        }
    }
```

```
    else if (option == 1) {
        struct tm* filetime = temp__file->t;
        printf("-");
        Authority__File(temp__file);

        if (temp__file->viewtype == 's') {
            printf("1   "); // file's inside
            printf("%d  ", sizeof(temp__file->content));
            printf("%03d-%03d %03d:%03d:%03d    ", filetime->tm_mon + 1, filetime->tm_mday,
            filetime->tm_hour, filetime->tm_min, filetime->tm_sec);
            printf("%s\n", temp__file->name);
        }

        while (temp__file->Rsibling != NULL) {
            temp__file = temp__file->Rsibling;
            filetime = temp__file->t;
            if (temp__file->viewtype == 's') {
                printf("-");
                Authority__File(temp__file);
                printf("1   ");
                printf("%d  ", sizeof(temp__file->content));
                printf("%03d-%03d %03d:%03d:%03d    ", filetime->tm_mon + 1, filetime->tm_mday,
                filetime->tm_hour, filetime->tm_min, filetime->tm_sec);
                printf("%s\n", temp__file->name);
            }
        }
    }
    else if (option == 2) {
        printf("%s ", temp__file->name);
        while (temp__file->Rsibling != NULL)
        {
            temp__file = temp__file->Rsibling;
            printf("%s ", temp__file->name);
        }
    }

    else { // option == 3
        struct tm* filetime = temp__file->t;
        printf("-");
        Authority__File(temp__file);
        printf("1   ");  //file inside
        printf("%d  ", sizeof(temp__file->content));
        printf("%03d-%03d %03d:%03d:%03d    ", filetime->tm_mon + 1, filetime->tm_mday, filetime->tm_hour, filetime->tm_min, filetime->tm_sec);
        printf("%s\n", temp__file->name);
        while (temp__file->Rsibling != NULL) {
            temp__file = temp__file->Rsibling;
            filetime = temp__file->t;
            printf("-");
            Authority__File(temp__file);
            printf("1   ");
            printf("%d  ", sizeof(temp__file->content));
            printf("%03d-%03d %03d:%03d:%03d    ", filetime->tm_mon + 1, filetime->tm_mday, filetime->tm_hour, filetime->tm_min, filetime->tm_sec);
            printf("%s\n", temp__file->name);
        }
    }
}
```

option 을 인자로 받아 그에 따른 조건문을 따라 실행이 된다. 이 탐색기를 실행하는 날짜와 시간이 뜨며, 트리를 탐색해가면서 파일들의 이름이 출력이 되는 함수이다. option 이 0 일 경우 viewtype 이 s 이면 파일의 이름을 출력한다.

6) Function_cp

```
772  // Command 6) cp
773 void Function_cp(tree* dtree, char* filename, char* ForDname) {
774     directory* current_dir = dtree->Cnode;
775     file* copy_file;
776     file* temp_file;
777     file* search_file;
778     if (Leave_File(dtree, filename) == NULL) {
779         printf("error exists!! : That file doesn't exist₩n");
780         return 0;
781     }
782     else
783         copy_file = Leave_File(dtree, filename);
784
785     if (strchr(ForDname, '/') == NULL) {
786         // file ->(copy) -> file
787         if (Leave_Directory(dtree, ForDname) == NULL) {
788             temp_file = Initial_File_Creation(ForDname);
789
790             strncpy(temp_file->content, copy_file->content, 1024);
791             search_file = current_dir->infile->Rsibling;
792
793             if (search_file == NULL)
794                 current_dir->infile->Rsibling = temp_file;
795             else {
796                 while (search_file->Rsibling != NULL) {
797                     search_file = search_file->Rsibling;
798                 }
799                 search_file->Rsibling = temp_file;
800             }
801         }
802
803         else {
804             Change_Directory_position(dtree, ForDname);
805             temp_file = dtree->Cnode->infile;
806
807             if (temp_file == NULL) {
808                 temp_file = Initial_File_Creation(filename);
809                 strncpy(temp_file->content, copy_file->content, 1024);
810                 dtree->Cnode->infile = temp_file;
811             }
812             else {
813                 if (temp_file->Rsibling == NULL) {
814                     temp_file->Rsibling = Initial_File_Creation(filename);
815                     strncpy(temp_file->Rsibling->content, copy_file->content, 1024);
816                 }
817                 else {
818                     search_file = temp_file->Rsibling;
819                     while (search_file->Rsibling != NULL)
820                         search_file = search_file->Rsibling;
821                     temp_file = Initial_File_Creation(filename);
822                     strncpy(temp_file->content, copy_file->content, 1024);
823                     search_file->Rsibling = temp_file;
824                 }
825             }
826             dtree->Cnode = current_dir;
827         }
828     }
829     else {
830         ForDname = strtok(ForDname, "/");
831         Change_Directory_position(dtree, ForDname);
832         temp_file = dtree->Cnode->infile;
833         ForDname = strtok(NULL, "/");
```



```
835         if (temp__file == NULL) {
836             temp__file = Initial__File__Creation(ForDname);
837             strncpy(temp__file->content, copy__file->content, 1024);
838             dtree->Cnode->infile = temp__file;
839         }
840         else {
841             if (temp__file->Rsibling == NULL) {
842                 temp__file->Rsibling = Initial__File__Creation(ForDname);
843                 strncpy(temp__file->Rsibling->content, copy__file->content, 1024);
844             }
845             else {
846                 search__file = temp__file->Rsibling;
847                 while (search__file->Rsibling != NULL) {
848                     search__file = search__file->Rsibling;
849                 }
850                 temp__file = Initial__File__Creation(ForDname);
851                 strncpy(temp__file->content, copy__file->content, 1024);
852                 search__file->Rsibling = temp__file;
853             }
854         }
855         dtree->Cnode = current__dir;
856     }
857 }
```

파일 또는 디렉토리를 복사하는 함수이다. cp를 구현하기 위해서는 strncpy 함수를 이용해서 copy__file 구조체의 content 배열을 temp__file 구조체의 content에 복사한다. 만약 존재하지 않는 파일이나 디렉토리를 입력하면 error exists!! : That file doesn't exist라는 메시지가 출력된다. Cp는 세 가지 옵션으로 구성되어 있는데, 첫 번째 옵션은 기존 파일 이름과 새로운 파일 이름을 옆에 같이 쓰게 됐을 때이다. 이는 해당 파일을 입력한 새로운 이름으로 복사한다(ex: cp abc.txt def.txt이면 abc.txt 파일을 def.txt 파일로 바꾸어 복사하게 된다). 두 번째 옵션은 기존 파일 이름과 디렉토리를 같이 썼을 때로, 기존의 파일이 그 디렉토리로 복사된다(ex: cp abc.txt xyz이면 xyz라는 디렉토리가 없을 때 abc.txt파일을 xyz파일로 복사, xyz라는 디렉토리가 있을 때는 xyz디렉토리 안에 abc.txt파일을 복사한다). 마지막 옵션은 기존 파일 이름과 디렉토리/새로운 파일 이름이 있을 때로, 그 디렉토리에 입력한 새로운 이름으로 복사된다(ex: cp abc.txt xyz/def.txt이면 abc.txt파일을 xyz라는 디렉토리 안에 def.txt으로 바꿔서 복사한다).

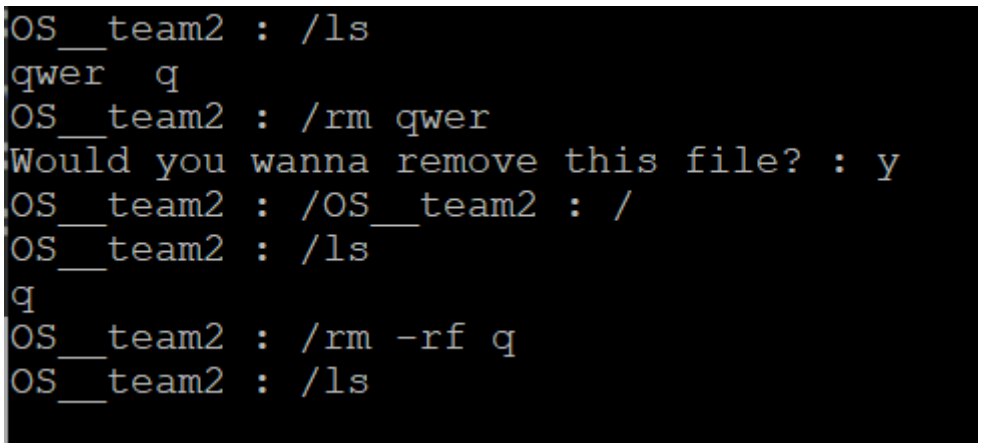
```
OS__team2 : /touch qwer
OS__team2 : /ls -al
-rwxr-xr-x 1    1024   06-06 01:48:34     qwer

OS__team2 : /cp qwer abcd
OS__team2 : /ls -al
-rwxr-xr-x 1    1024   06-06 01:48:34     qwer
-rwxr-xr-x 1    1024   06-06 01:48:41     abcd
```

7) Rm

```
859   // Command 7) rm
860   void Explore__Directory(tree* dtree, char* findname, int option) {
861       char YESORNO;
862       directory* rmdir;
863       if (option == 0) {
864           if (Function__File(dtree, findname, option) == FALSE) {
865               rmdir = Leave__Directory(dtree, findname);
866               if (rmdir != NULL)
867                   printf("removing directory is not possible₩n");
868           }
869       }
870
871       else if (option == 1) {
872           if (Function__File(dtree, findname, option) == FALSE) {
873               rmdir = Leave__Directory(dtree, findname);
874               if (rmdir != NULL)
875                   printf("removing directory is not possible₩n");
876           }
877       }
878
879       else if (option == 2) {
880           if (Function__File(dtree, findname, option) == FALSE) {
881               rmdir = Leave__Directory(dtree, findname);
882               if (rmdir != NULL) {
883                   Function__rmdir(structure, findname, option);
884               }
885           }
886       }
887       else if (option == 3) {
888           if (Function__File(dtree, findname, option) == FALSE) {
889               rmdir = Leave__Directory(dtree, findname);
890               if (rmdir != NULL) {
891                   Function__rmdir(structure, findname, option);
892               }
893           }
894       }
895   }
```

User__Command 함수에서 인자로 받은 option 값에 따라 조건이 나뉘어진다. Option이 0일 때는 rm, option이 1일 때는 rm -f, option이 2일 때는 rm -r, option이 3일 때는 rm -rf를 구현한다. Rm test는 test 파일을 삭제하고,

rm -f test는 test 파일을 삭제할 때 확인 과정을 거치지 않으며, rm -r /home은 /home 디렉토리를 삭제하며, rm -rf는 디렉토리 및 하위 모든 내용을 강제로 삭제한다. 그리고 Function_file 함수에서 반환된 값이 사용되며, Leave_directory 함수를 통해 탐색을 하여 Function_rmdir 함수를 이용해 디렉토리 삭제를 한다.



- **Function_File**

```
897  int Function_File(tree* dtree, char* findname, int option) {
898      file* rmfile = NULL;
899      file* find_file = NULL;
900      file* prev_file = NULL;
901
902      find_file = dtree->Cnode->infile;
903      prev_file = dtree->Cnode->infile;
904
905      char YESORNO;
906
907      int rm_case;
908
909      if (prev_file == NULL)
910          return FALSE;
911
912      else if (strcmp(find_file->name, findname) == 0) {
913          dtree->Cnode->infile = find_file->Rsibling;
914          rmfile = find_file;
915          rm_case = 0;
916      }
917      else {
918          rm_case = 1;
919          find_file = find_file->Rsibling;
920
921          while (find_file != NULL) {
922              if (strcmp(find_file->name, findname) == 0) {
923                  rmfile = find_file;
924                  break;
925              }
926              else {
927                  prev_file = find_file;
928                  find_file = find_file->Rsibling;
929              }
930          }
931      }
```

```
933     if (rmfile != NULL) {
934         if ((option == 0) || (option == 2)) {
935             printf("Would you wanna remove this file? : ");
936             scanf("%c", &YESORNO);
937 
938             if (YESORNO == 'y') {
939                 prev_file->Rsibling = rmfile->Rsibling;
940                 free(rmfile);
941                 return TRUE;
942             }
943 
944             else {
945                 if (rm_case == 0)
946                     dtree->Cnode->infile = rmfile;
947                 return FALSE;
948             }
949         }
950 
951         else {
952             prev_file->Rsibling = rmfile->Rsibling;
953             free(rmfile);
954             return TRUE;
955         }
956     }
957     else {
958         if ((option == 2) && (option == 3)) {
959             printf("That file don't exist in this directoryWn");
960             return FALSE;
961         }
962         return FALSE;
963     }
964 }
```

Function_File() 함수는 rm을 파일 쪽으로 실행시키는 함수이다. option 값을 인자로 받아서 실행되며, rm_case 변수의 값에 따라서 실행시키는 방향이 달라진다. free() 함수를 이용해 파일 노드를 해제하게 됨으로써 역할을 수행한다.

- **Rm_subdir**

```
966 // clear director's node (Recursive)
967 void rm_subdir(directory* dir) {
968     if (dir != NULL) {
969         rm_subdir(dir->Lchild);
970         rm_subdir(dir->Rsibling);
971         free(dir);
972     }
973 }
```

Rm 함수의 하위 파일까지 삭제해주기 위함이다.

- 추가 명령어

1) Rmdir: 디렉토리 삭제

```
975  //// Additional Commands
976  // 1) rmdir (delete directory)
977  int Function__rmdir(tree* dtree, char* finddirname, int option) {
978      directory* rmdir = NULL;
979      directory* find__dir = NULL;
980      directory* prev__dir = NULL;
981
982      find__dir = dtree->Cnode->Lchild;
983      prev__dir = dtree->Cnode->Lchild;
984
985      char YESORNO;
986
987      int rm__case;
988
989      if (prev__dir == NULL) {
990          return FALSE;
991      }
992
993      else if (strcmp(find__dir->name, finddirname) == 0) {
994          dtree->Cnode->Lchild = find__dir->Rsibling;
995          rmdir = find__dir;
996          rm__case = 0;
997      }
998      else {
999          rm__case = 1;
1000         find__dir = find__dir->Rsibling;
1001         while (find__dir != NULL)
1002         {
1003             if (strcmp(find__dir->name, finddirname) == 0)
1004             {
1005                 rmdir = find__dir;
1006                 break;
1007             }
1008             else
1009             {
1010                 prev__dir = find__dir;
1011                 find__dir = find__dir->Rsibling;
1012             }
1013         }
1014     }
```

```
1016    if (rmdir != NULL) {
1017        if (option == 2) {
1018            printf("Would you wanna remove this Directory? : ");
1019            scanf("%c", &YESORNO);
1020            if (YESORNO == 'y') {
1021                if (rm_case == 1) {
1022                    prev_dir->Rsibling = rmdir->Rsibling;
1023                    rm_subdir(rmdir->Lchild);
1024                    free(rmdir);
1025                }
1026                else {
1027                    rm_subdir(rmdir->Lchild);
1028                    free(rmdir);
1029                }
1030            }
1031            else {
1032                if (rm_case == 0) {
1033                    dtree->Cnode->Lchild = rmdir;
1034                }
1035            }
1036        }
1037        else {
1038            if (rm_case == 1) {
1039                prev_dir->Rsibling = rmdir->Rsibling;
1040                rm_subdir(rmdir->Lchild);
1041                free(rmdir);
1042            }
1043            else {
1044                rm_subdir(rmdir->Lchild);
1045                free(rmdir);
1046            }
1047        }
1048    }
1049    else {
1050        printf("That directory don't exist!!₩n");
1051    }
1052 }
```

Function_rmdir은 리눅스의 명령어 중 디렉토리를 삭제하는 역할을 하는 rmdir을 실행시키는 함수이다. 이 함수는 User_Command 함수에서 option 값을 인자로 받아 실행되며, rm_case의 값에 따라 실행시키는 루트가 달라진다. rm_subdir 함수를 이용해 rmdir의 Lchild 부분의 노드를 해제시키며, rmdir 노드도 같이 해제시킨다.

```
OS_team2 : /mkdir a b c
OS_team2 : /ls
a b c
OS_team2 : /rmdir a
Would you wanna remove this Directory? : y
OS_team2 : /OS_team2 : /ls
b c
```

2) Chmod: 파일 혹은 디렉토리의 접근 권한을 변경

```
1054  // 2) chmod
1055  // change file or directory's access authority
1056  void chmod__(tree* dtree, int permission, char* name) {
1057      directory* temp__node = Leave__Directory(structure, name);
1058      file* temp__file = Leave__File(structure, name);
1059      if (temp__node != NULL)
1060          temp__node->chmod__ = permission;
1061      else if (temp__file != NULL)
1062          temp__file->chmod__ = permission;
1063      else
1064          printf("error exists : Directory or File doesn't exit!₩n");
1065  }
```

Chmod__ 함수는 리눅스 명령어 중 파일 또는 디렉토리의 접근 권한을 변경시키는 chmod를 실행하는 함수다. Leave__Directory 함수와 Leave__File 함수를 통해 노드와 그 안의 파일에 대한 변수를 선언 받고 노드와 파일의 권한을 나타내는 구조체 멤버인 chmod__을 권한 정수 값인 permission으로 변경시킨다.

```
OS__team2 : /chmod 770 hello.txt
OS__team2 : /ls -l
drwxr-xr-x 1    4096  06-02 03:43:10    b
drwxr-xr-x 1    4096  06-02 03:43:10    c
drwx------ 1    4096  06-02 03:46:19    test
-rwxrwx--- 1    1024  06-02 04:22:01    hello.txt
```

3) Function_touch: 빈 파일 새로 생성, 파일의 날짜 혹은 시간 정보 변경

```
1068  // 3) Function_touch
1069  // create a blank file and update file's date, time
1070  void Function_touch(tree* dtree, char* filename, int option) {
1071      file* temp_file;
1072      file* current_file = dtree->Cnode->infile;
1073      directory* temp_node = dtree->Cnode;
1074
1075
1076      // case1) create a blank file
1077      if (option == 0) {
1078          temp_file = Leave_File(structure, filename);
1079          if (temp_file == NULL) {
1080              if (temp_node->infile == NULL) {
1081                  temp_file = Initial_File_Creation(filename);
1082                  temp_node->infile = temp_file;
1083              }
1084              else {
1085                  if (temp_node->infile->Rsibling == NULL) {
1086                      temp_file = Initial_File_Creation(filename);
1087                      temp_node->infile->Rsibling = temp_file;
1088                  }
1089                  else {
1090                      current_file = current_file->Rsibling;
1091
1092                      while (current_file->Rsibling != NULL)
1093                          current_file = current_file->Rsibling;
1094                      temp_file = Initial_File_Creation(filename);
1095                      current_file->Rsibling = temp_file;
1096                  }
1097              }
1098          }
1099          else {
1100              time_t timer;
1101              timer = time(NULL);
1102              temp_file->t = localtime(&timer);
1103          }
1104      }
```

```
1099          else {
1100              time_t timer;
1101              timer = time(NULL);
1102              temp_file->t = localtime(&timer);
1103          }
1104      }
1105      // set file's date and time
1106      else {
1107          temp_file = Leave_File(structure, filename);
1108          if (temp_file != NULL) {
1109              time_t timer;
1110              timer = time(NULL);
1111              temp_file->t = localtime(&timer);
1112          }
1113      }
1114  }
```

Function_touch는 새로운 빈 파일을 생성하거나, 혹은 파일의 시간과 날짜 정보를 변경하는 리눅스 명령어인 touch를 구현한 함수다. 파일의 시간, 그리고 날짜 정보를 나타내기 위해서 time_t 자료형과 time() 함수를 사용하였다.

```
OS__team2 : /touch abcd
OS__team2 : /ls -al
-rwxr-xr-x 1    1024   06-06 01:50:58     abcd

OS__team2 : /touch abcd
OS__team2 : /ls -al
-rwxr-xr-x 1    1024   06-06 01:52:35     abcd
```

abcd 의 빈 파일을 생성 후 touch abcd 명령어를 또 다시 하달 했을 때 빈 파일인 abcd 의 생성 시간이 초기화 된 것을 확인할 수 있다.

- **Touch_timer**

```
1116    // update file's time
1117    void touch__timer(tree* dtree, char* filetime, char* filename) {
1118        char str__time[2];
1119        file* temp__file;
1120        temp__file = Leave__File(structure, filename);
1121        if (temp__file != NULL) {
1122            int minute;
1123            int second;
1124            int month;
1125            int day;
1126            int hour;
1127
1128            strncpy(str__time, filetime + 4, 2);
1129            month = atoi(str__time);
1130            if (month > 12) {
1131                printf("Time is not correct!!₩n");
1132                return;
1133            }
1134            strncpy(str__time, filetime + 6, 2);
1135            day = atoi(str__time);
1136            if (day > 31) {
1137                printf("Time is not correct!!₩n");
1138                return;
1139            }
1140            strncpy(str__time, filetime + 8, 2);
1141            hour = atoi(str__time);
1142            if (hour > 23) {
1143                printf("Time is not correct!!₩n");
1144                return;
1145            }
1146            strncpy(str__time, filetime + 10, 2);
1147            minute = atoi(str__time);
1148            if (minute > 60) {
1149                printf("Time is not correct!!₩n");
1150                return;
1151            }
```

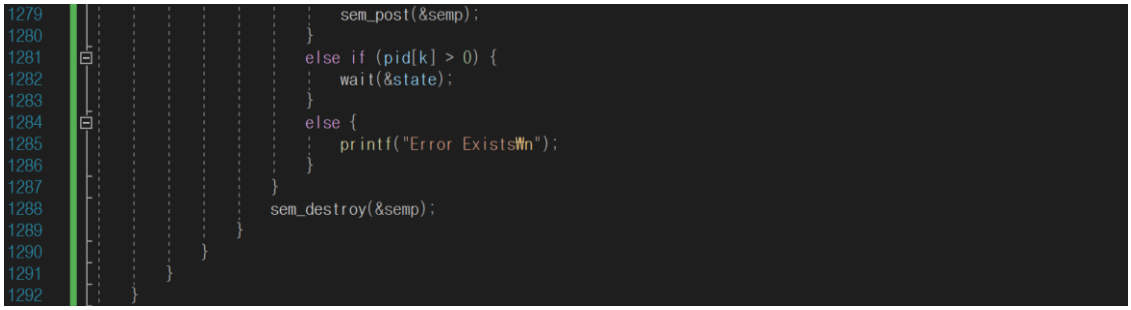
```
1152            strncpy(str__time, filetime + 12, 2);
1153
1154            second = atoi(str__time);
1155            if (second > 60) {
1156                printf("Time is not correct!!\n");
1157                return;
1158            }
1159            temp__file->t->tm_mon = month;
1160            temp__file->t->tm_mday = day;
1161            temp__file->t->tm_hour = hour;
1162            temp__file->t->tm_min = minute;
1163            temp__file->t->tm_sec = second;
1164        }
1165        return 0;
1166    }
```

- **User_command**

```
1169    // Assemble of every command's for user
1170    // Option 0 : Basic
1171    // Option 1~ : Specific
1172
1173    // Command
1174    void User__Command(char* str, int cmdoption) {
1175        char temp[50];
1176        char* command = NULL;
1177
1178        directory* temp__dir = NULL;
1179        file* temp__file = NULL;
1180
1181        char* Dir__Path;
1182        char* temp__str;
1183
1184
1185        int i;
1186        int h = 0;
1187        int chtemp;
1188
1189        int option;
1190        int Save__Command = 0;
1191        strncpy(temp, str, 30);
1192
1193        // Enter the File Explorer
1194        if (strcmp(str, "") == 0)
1195            return;
1196        command = strtok(str, " ");
1197
1198        // Command 1) mkdir
1199        if (strcmp(command, "mkdir") == 0) {
1200            Save__Command = 1;
1201            char* tempstr;
1202            command = strtok(NULL, " ");
1203
1204            if (command == NULL)
1205                printf("Nothing was entered in User's Command\n");
```

```
1206        else {
1207            // when creating a Directory, User authorithy is allocated
1208            if (strcmp(command, "-m") == 0) {
1209                option = 1;
1210                command = strtok(NULL, " ");
1211                chtemp = atoi(command);
1212                command = strtok(NULL, " ");
1213                temp_dir = Function_mkdir(structure, command, option);
1214                temp_dir->chmod_ = chtemp;
1215            }
1216
1217            // when creating a Directory, Upper directory is created with lower directory
1218            // ex) mkdir -p abc/def/ghi
1219            else if (strcmp(command, "-p") == 0) {
1220                option = 2;
1221                command = strtok(NULL, " ");
1222                tempstr = command;
1223                // present current node saving situation
1224                temp_dir = structure->Cnode;
1225
1226                Dir_Path = strtok(command, "/");
1227
1228                while (Dir_Path != NULL) {
1229                    if (Function_cd(structure, Dir_Path) == FALSE) {
1230                        Function_mkdir(structure, Dir_Path, 2);
1231                        Function_cd(structure, Dir_Path);
1232                    }
1233                    Dir_Path = strtok(NULL, "/");
1234                }
1235                structure->Cnode = temp_dir;
1236            }
1237            else {
1238                option = 0;
1239                // when creating Directory x, y, z, present x/y/z
1240                if (strchr(command, '/') != NULL) {
1241                    temp_dir = structure->Cnode;
1242                    Dir_Path = strrchr(command, '/');
```

```
1243                    Dir_Path = Dir_Path + 1;
1244                    command = strtok(command, "/");
1245                    while (command != Dir_Path) {
1246                        if (Function_cd(structure, command) == FALSE) {
1247                            printf("Can't exit this Directory₩n");
1248                            break;
1249                        }
1250                        command = strtok(NULL, "/");
1251                    }
1252                    if (Dir_Path == command)
1253                        Function_mkdir(structure, command, 0);
1254                    structure->Cnode = temp_dir;
1255                }
1256                // simple abc
1257                else {
1258                    int state;
1259                    int fd[2];
1260                    int k = -1;
1261                    char buffer[100];
1262                    pid_t pid[3];
1263                    state = pipe(fd);
1264
1265                    state = sem_init(&semp, 1, 1);
1266                    while (command != NULL) {
1267                        if (k < 3) {
1268                            pid[++k] = fork();
1269                        }
1270                        write(fd[1], buffer, 100);
1271
1272                        if (pid[k] == 0) {
1273                            sem_wait(&semp);
1274                            read(fd[0], buffer, 100);
1275                            //printf("child:%ld₩n", (long)getpid());
1276                            Function_mkdir(structure, command, option);
1277                            command = strtok(NULL, " ");
1278                            write(fd[1], buffer, 100);
```



User_command 함수는 명령어를 받아서 각 함수에 넣어주는 역할을 한다. Mkdir 이 입력되면 문자열을 자르는 strtok() 함수를 통해 mkdir 뒤에 오는 문자열을 확인한다. Mkdir 뒤에 아무것도 입력되지 않았으면 Nothing was entered in User's command 라는 문구가 출력된다. Mkdir 뒤에 -m 이 붙으면 디렉토리를 생성할 때 사용자 권한을 같이 지정하게 되는데, 뒤에 숫자가 붙으므로 문자 스트링을 정수로 변환해주는 atoi() 함수를 사용한다. 또한, option 값을 1 로 반환한다. Strtok() 함수로 확인한 문자열이 -p 이면 option 값을 2 로 반환하고, Function_mkdir() 함수와 Function_cd() 함수를 통해 만들 디렉토리의 상위 디렉토리까지 함께 생성한다. 만약 -p 를 사용하지 않고 상위 디렉토리까지 생성하려고 하면, can't exit this Directory 라는 문구가 출력된다. Mkdir 뒤에 바로 디렉토리를 하나 썼을 때는 option 값이 0 으로 저장되며 Function_mkdir()함수를 통해 새로운 디렉토리가 생성된다.

```
1294        // Command 2) pwd
1295        else if (strcmp(command, "pwd") == 0) {
1296            Function__pwd(structure, qstack, 0);
1297            Save__Command = 0;
1298        }
1299
1300        // Command 3) cd
1301        else if (strcmp(command, "cd") == 0) {
1302            Save__Command = 1;
1303            command = strtok(NULL, " ");
1304            Change__Directory__position(structure, command);
1305        }
1306        else if ((strcmp(command, "cat") == 0) && (cmdoption == 1)) {
1307            command = strtok(NULL, " ");
1308            if (strcmp(command, ">") == 0) {
1309                Save__Command = 1;
1310                option = 1;
1311                command = strtok(NULL, " ");
1312                Function__cat(structure, command, option);
1313                temp__file = Leave__File(structure, command);
1314            }
1315            else {
1316                option = 0;
1317                Function__cat(structure, command, option);
1318            }
1319        }
```

Pwd 가 입력되었을 때는 else if (strcmp(command, "pwd") == 0)을 통해 Function__pwd() 함수로 이동한다. Cd 가 입력되었을 때는 else if (strcmp(command, "cd") == 0)를 통해 Change_Directory_position() 함수로 이동한다. Cat 이 입력되었을 때, else if ((strcmp(command, "cat") == 0) && (cmdoption == 1))을 통해 확인 후 strtok() 함수를 사용해 cat 뒤에 >가 붙으면 option 값을 1 로, 붙지 않으면 option 값을 0 으로 저장한다.

```
1321        // Command 4) cat
1322        else if ((strcmp(command, "cat") == 0) && (cmdoption == 0)) {
1323            char* cat_content;
1324            char* f_name;
1325            int len;
1326            command = strtok(NULL, " ");
1327            command = strtok(NULL, " ");
1328            cat_content = command;
1329            command = strtok(NULL, " ");
1330            len = command - cat_content;
1331            char* cat_filename = (char*)malloc(len);
1332            strncpy(cat_filename, cat_content, len);
1333            *(cat_filename + len - 1) = '₩0';
1334            Initial__cat(structure, cat_filename, command);
1335        }
1336
1337        // Command 5) ls
1338        else if (strcmp(command, "ls") == 0) {
1339            if (strchr(temp, '-') == NULL) {
1340                option = 0;
1341                Function__ls(structure, option);
1342            }
1343            else {
1344                command = strtok(NULL, " ");
1345                // ls : "ls-l"
1346                if (strcmp(command, "-l") == 0) {
1347                    option = 1;
1348                    Function__ls(structure, option);
1349                }
1350                // ls : "ls-a"
1351                else if (strcmp(command, "-a") == 0) {
1352                    option = 2;
1353                    Function__ls(structure, option);
1354                }
1355                // ls : "ls-al"
1356                else {
1357                    option = 3;
1358                    Function__ls(structure, option);
1359                }
1360            }
1361        }
1362
1363        // Command 6) cp
1364        else if (strcmp(command, "cp") == 0) {
1365            command = strtok(NULL, " ");
1366            temp__str = command;
1367            command = strtok(NULL, " ");
1368            Function__cp(structure, temp__str, command);
1369        }
```

ls 가 입력되었을 때는 else if (strcmp(command, “ls”) == 0)을 통해 확인한 후, strchr() 함수를 통해 -라는 문자가 있는지 검사한다. -가 없으면 option 값을 0 으로 저장하고 Function__ls() 함수로 이동한다. Strtok() 함수를 사용해 ls 뒤에 있는 문자를 command 에 저장하고, strcmp() 함수를 사용해 command 값이 -l 과 같으면 option 값을 1 로 저장 후 Function__ls() 함수로 이동, command 값이 -a 와 같으면 option 값을 2 로 저장 후 Function__ls() 함수로 이동한다. Command 값이 -al 과 같으면 option 값을 3 으로 저장 후 Function__ls() 함수로 이동한다. Cp 가 입력되었을 때는 else if (strcmp(command, “cp”) == 0)을 통해

확인한 후 복사할 파일 또는 디렉토리를 temp_str 에 저장하고, 복사할 다른 이름을 command 에 넣은 후 Function_cp() 함수로 보낸다.

```
1371        // Command 7) rm
1372        else if (strcmp(command, "rm") == 0) {
1373            Save_Command = 1;
1374            command = strtok(NULL, " ");
1375
1376            // rm : "rm-f"
1377            if (strcmp(command, "-f") == 0) {
1378                option = 1;
1379                command = strtok(NULL, " ");
1380                Explore_Directory(structure, command, option);
1381            }
1382
1383            // rm : "rm-r"
1384            else if (strcmp(command, "-r") == 0) {
1385                option = 2;
1386                command = strtok(NULL, " ");
1387                Explore_Directory(structure, command, option);
1388            }
1389
1390            // rm : "rm-rf"
1391            else if (strcmp(command, "-rf") == 0) {
1392                option = 3;
1393                command = strtok(NULL, " ");
1394                Explore_Directory(structure, command, option);
1395            }
1396
1397            // rm : "rm"
1398            else {
1399                option = 0;
1400                Explore_Directory(structure, command, option);
1401            }
1402
1403        }
```

Rm 이 입력되면, else if (strcmp(command, "rm") == 0)을 통해 확인한 후, strtok() 함수를 통해 command 에 rm 뒤의 문자열을 넣고, strcmp() 함수를 이용해 command 값이 -f 과 같으면 option 값을 1 로 저장한 후 Explore_Directory() 함수로 이동, -r 과 같으면 option 값을 2 로 저장한 후 Explore_Directory() 함수로 이동, -rf 와 같으면 option 값을 3 으로 저장한 후 Explore_Directory() 함수로 이동, rm 뒤에 아무것도 붙지 않으면 option 값을 0 으로 저장한 후 Explore_Directory() 함수로 이동한다.

```
1400        // Additional Command 1) rmdir
1401        else if (strcmp(command, "rmdir") == 0) {
1402            Save__Command = 1;
1403            option = 2;
1404            command = strtok(NULL, " ");
1405            Explore__Directory(structure, command, option);
1406        }
1407
1408        // Additional Command 2) chmod
1409        else if (strcmp(command, "chmod") == 0) {
1410            Save__Command = 1;
1411            command = strtok(NULL, " ");
1412            chtemp = atoi(command);
1413            command = strtok(NULL, " ");
1414            chmod__(structure, chtemp, command);
1415        }
1416
1417        // Additional Command 3) touch
1418        else if (strcmp(command, "touch") == 0) {
1419            command = strtok(NULL, " ");
1420            if (strcmp(command, "-t") == 0) {
1421                command = strtok(NULL, " ");
1422                temp__str = command;
1423                command = strtok(NULL, " ");
1424                touch__timer(structure, temp__str, command);
1425            }
1426            else if (strcmp(command, "-c") == 0) {
1427                option = 2;
1428                command = strtok(NULL, " ");
1429                Function__touch(structure, command, option);
1430            }
1431            else {
1432                option = 0;
1433                Function__touch(structure, command, option);
1434            }
1435        }
```

Rmdir 이 입력되면, else if (strcmp(command, "rmdir") == 0)을 통해 option 에 2 를 넣고 Explore__Directory() 함수로 이동한다. Chmod 가 입력되면, else if (strcmp(command, "chmod") == 0)를 거친 후 chmod 뒤에 숫자가 오므로 문자 스트링을 정수로 변환해주는 atoi() 함수를 사용해 chtemp 에 저장 후 chmod__() 함수로 이동한다. Touch 가 입력되면, else if (strcmp(command, "touch") == 0)을 거친 후, strtok() 함수를 통해 command 에 touch 뒤의 문자열을 넣는다. 이후 strcmp() 함수를 이용해 command 값이 -c 과 같으면 option 값을 2 로 저장한 후 Function__touch() 함수로 이동, -t 와 같으면 touch__timer() 함수로 이동한다. Touch 뒤에 바로 파일명이 오면 option 에 0 을 넣고 Function__touch() 함수로 이동한다.

```
1437        // When wrong command's were entered
1438        else
1439            printf("Wrong command is enteredWn");
1440
1441        // update's file "saving.txt"
1442        if ((Save_Command == 1) && (cmdoption == 1)) {
1443            FILE* fp = fopen("saving.txt", "a");
1444            if (strstr(temp, "cat >") == NULL) {
1445                fputs(temp, fp);
1446                fputs("Wn", fp);
1447            }
1448            else {
1449                fputs(temp, fp);
1450                fputs(" ", fp);
1451                fputs(temp_file->content, fp);
1452            }
1453            fclose(fp);
1454        }
1455    }
```

명령어가 잘못 입력되었을 때는 Wrong command is entered 라는 문구가 출력된다.

5. 보안할 점

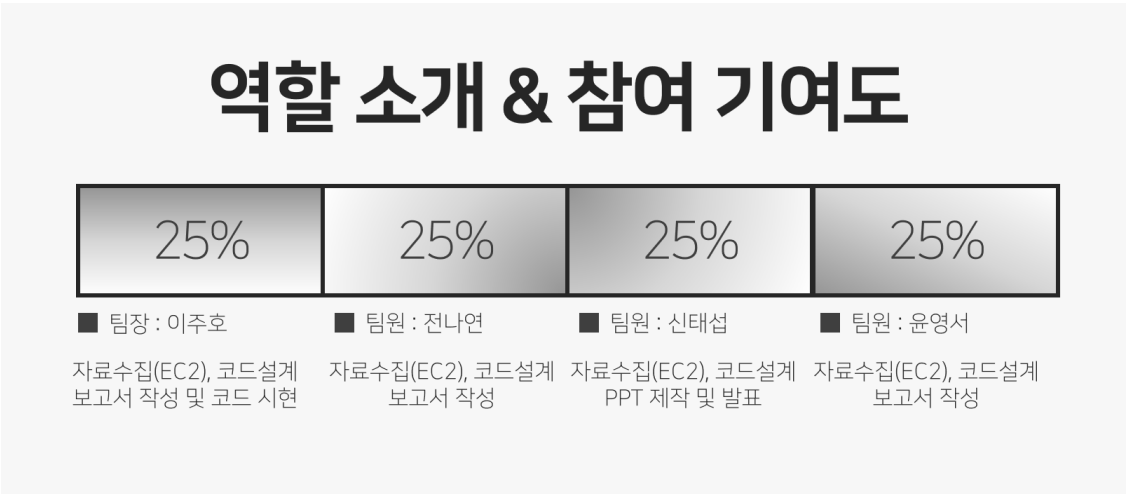
```
OS__team2 : /ls -al

OS__team2 : /cat < newfile
File or Directory No exist
OS__team2 : /ls -al

OS__team2 : /cat > newfile
qwertadfgzxccvff
OS__team2 : /ls -al
-rwxr-xr-x 1    1024   06-07 03:42:04     newfile
```

cat 명령어를 실행해 내용을 입력할 때 enter 키를 눌렀을 때 줄이 바뀌고, ctrl+d 를 누르면 파일이 저장되면서 종료되는데 본 조는 이 기능 구현이 안 되었다. Enter 키를 누르면 바로 저장되면서 끝이 난다.

그리고 find 명령어를 구현하지 못했다. 이는 본 조가 LCRS 를 사용했기 때문에 일반적인 Tree 와 달라서 구현하기가 힘들었기 때문이다.

6.자원 소요 계획

6.1. 프로젝트 팀원 역할 소개 및 참여기여도



6.2. 프로젝트 추진 단계

1 자료조사

2 역할 분담 및 프로젝트 일정 계획

3 EC2 계정 생성 및 SSH 원격 접속

4 명령어 설계 및 구현

5 오류 수정 및 최종점검

6 PPT, 보고서 작성

6.3. 전체 추진 일정

5/9	프로젝트 첫 회의 및 사전조사, 역할 분담
5/14	팀원들 각자 EC2 인스턴스 생성
5/16	인스턴스 생성 확인, putty 연결 및 명령어 분담
5/20	조장 EC2 에 SSH 명령어 사용하여 연결
5/24	명령어 구현
6/1	구현한 명령어 통합
6/4	ppt, 보고서 작성 회의
6/7	발표 예행 연습

6.4. 회의 일지

회의날짜	장 소	내 용
5 / 9	실준실	**전화번호 교환, 각자 인스턴스 생성, 프로젝트의 방향성에 대한 토론, 역할 분담, 다음 회의까지 기초 개념 공부해오기로 함**
5 / 16	실준실	**인스턴스 생성 확인, putty ssh 연결, 역할 분배: 신태섭: rm, mkdir 윤영서: cd, ls 이주호: cat, rmdir, touch 전나연: chmod, cp**
5 / 24	아리수	**구현한 명령어 코드의 오류 수정, 추가 명령어 구현**
6 / 1	아리수	**구현한 명령어 코드에 대한 토론 및 문제점 수정 보완 추가적인 명령어 구현**
6 / 4	실준실	**보고서 작성 및 ppt 작성**

7. 참고문헌

AWS 참고자료

https://www.youtube.com/playlist?list=PLfth0bK2MgIa6w63IglYQD_qljDntSh-H

1. EC2 기초
2. EC2 기초 2
3. Security group
6. vpc(1)

https://www.youtube.com/playlist?list=PLfth0bK2MgIan-SzGpHIbfnCnjj583K2m

1. 클라우드 컴퓨팅이란?
2. 클라우드 컴퓨팅의 종류
3. AWS 의 구조-리전,가용영역
4. AWS 계정만들기 및 첫 설정
5. IAM 기초
7. EC2 소개 및 맛보기