운영체제 프로젝트 최종보고서

File System Implementation With MiniOS

프로젝트 기간	2024.04.18 – 2024.06.12
담당 교수	박재근 교수
조명	5조
조원	박현빈, 안지훈, 이시호, 장진영

1 pages

INDEX

i. 프로젝트 소개

1. 프로젝트 개요

   본 프로젝트는 실습 환경 부족으로 인해 하드웨어를 직접 제어하는 커널 코드나 드라이버를 작성하는 낮은 수준의 프로그래밍은 진행하지 않고 소프트웨어 API를 활용하여 간단한 운영 체제를 구현한다.

   운영체제 기능을 설계하고 직접 구현하여 운영 체제의 기본적인 기능을 이해한다.

2. 프로젝트 목표

   본 프로젝트는 운영 체제의 여러가지 기능 중 System Call, Process Management, Memory Management, Scheduling 기능을 각자 구현하여 miniOS를 제작하려 했으나, API를 사용하여 간단한 코드를 작성하는 기초적인 작업보다 하나의 기능을 구체적으로 구현하여 이해도를 높이는 것이 중요하다고 판단했다.

   운영 체제의 필수적인 요소 중 하나인 파일 시스템을 구현하여 사용자와 시스템이 데이터를 효율적으로 저장하고 관리하는 방법을 보여준다.

   복잡한 파일 시스템의 일부를 단순화하여 구현함으로써 시스템 프로그래밍에 필요한 기술을 익히며. 이를 통해 파일 시스템의 설계와 관리에 대한 실질적인 이해를 높인다.

   Linux 운영체제에서의 파일 관련 기본 쉘 명령어의 작동 원리를 파악하여 User level 수준으로 새롭게 구현해 본다.

3. 프로젝트 수행 계획

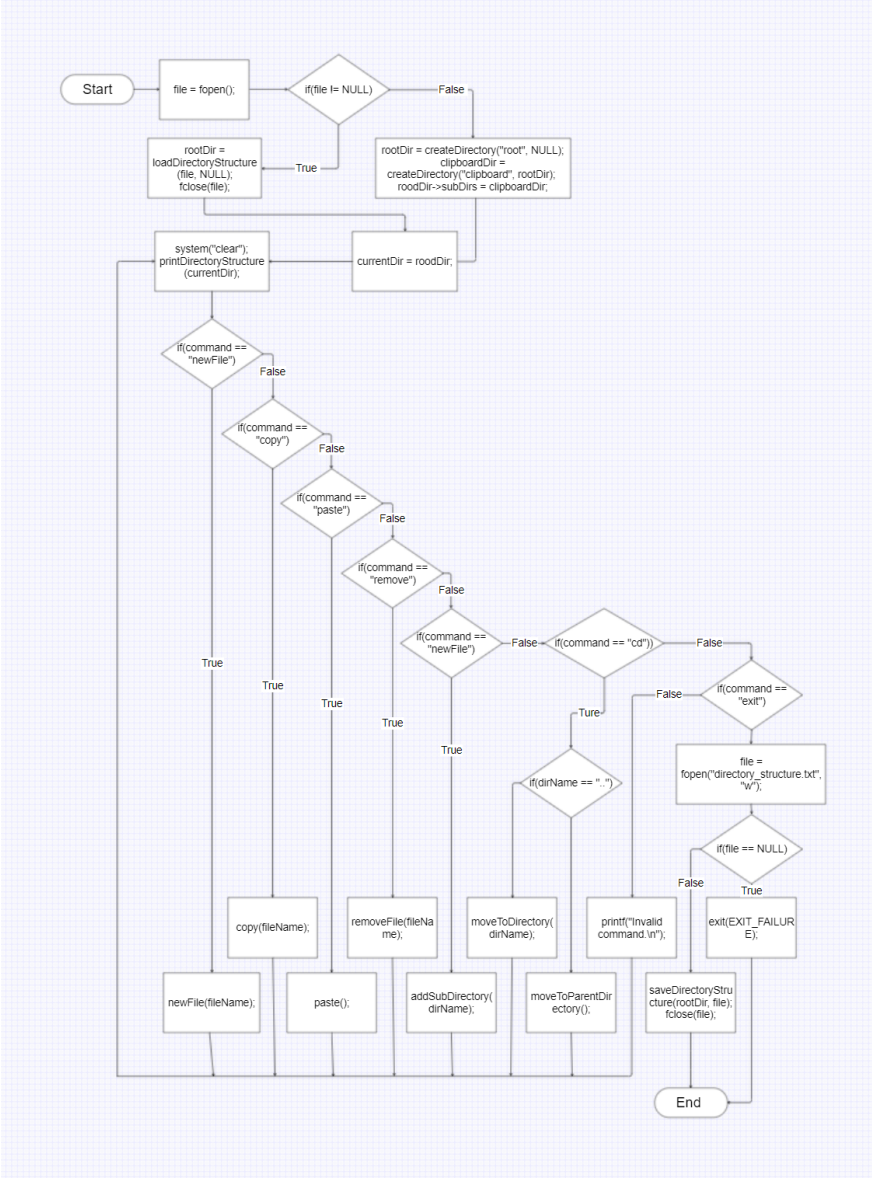
04.18	프로젝트 개요 파악, 팀원 네트워킹
04.25	구현할 OS 기능 논의
05.02	구현할 OS 기능 결정, 역할 분담

05.09	OS 기능 구현 및 Git upload
05.16	개인 중간결과보고서 작성 및 코드 설명
05.22	중간결과보고서 최종 통합 및 발표 준비
05.23	프로젝트 중간결과 발표
05.30	파일시스템 구현으로 프로젝트 방향 수정
06.06	파일 시스템 코드 작성, Minios 제작 및 통합
06.11	최종결과보고서 작성, 발표자료제작
06.12	최종 프로젝트 발표 준비
06.13	프로젝트 최종결과 발표

ii. 프로젝트 설명

1. 순서도

사용자가 사용할 수 있는 기능은 파일생성, 파일 복사, 파일 붙여넣기, 파일 삭제, 디렉토리 생성, 하위 디렉토리로 이동, 상위 디렉로리로 이동까지 총 7 가지 이며 각각 newfile, copy, paste, remove, mkdir, cd 명령어를 사용하여 실행할 수 있다. 최종 프로젝트에서 구현한 miniOS 구조를 다음과 같이 순서도로 작성하였다.



2. 코드 설명

FileNode 구조체

```
typedef struct FileNode {
 char fileName[256];
 struct FileNode* next;
} FileNode;
```

- 파일 시스템의 파일 노드
- FileName : 파일의 이름
- next : 디렉토리의 다음 파일 노드에 대한 포인터

Directory 구조

```
typedef struct Directory {
 char dirName[256];
 FileNode* files;
 struct Directory* parent;
 struct Directory* next;
 struct Directory* subDirs;
} Directory;
```

- 디렉토리를 나타냄
- dirName : 디렉터리 이름.
- files : 디렉토리의 첫번째 파일 노드에 대한 포인터
- parent : 부모 디렉토리에 대한 포인터
- next : 다음 디렉토리에 대한 포인터
- subDirs : 첫 번째 하위 디렉토리에 대한 포인터

전역변수

```
Directory* rootDir = NULL;
Directory* currentDir = NULL;
Directory* clipboardDir = NULL;
```

- rootDir : 루트 디렉터리에 대한 포인터
- currentDir : 작업이 수행되는 현재 디렉토리에 대한 포인터
- clipadDir : 복사 붙여넣기 작업에 사용되는 클립보드 디렉토리에 대한 포인터

CreateFileNode

```
FileNode* createFileNode(const char* fileName) {
 FileNode* newNode = (FileNode*)malloc(sizeof(FileNode));
 if (newNode == NULL) {
 perror("Failed to create file node");
 exit(EXIT_FAILURE);
 }
 strncpy(newNode->fileName, fileName, 255);
 newNode->fileName[255] = '\0';
 newNode->next = NULL;
 return newNode;
}
```

- 새 파일 노드를 만드는 수
- 새 FileNode에 대해 메모리를 할당
- 파일 이름을 fileName에 복사

- 다음 디렉토리에 대한 포인터를 NULL로 초기화

CreateDirectory

```
Directory* createDirectory(const char* dirName, Directory* parent) {
 Directory* newDir = (Directory*)malloc(sizeof(Directory));
 if (newDir == NULL) {
 perror("Failed to create directory");
 exit(EXIT_FAILURE);
 }
 strncpy(newDir->dirName, dirName, 255);
 newDir->dirName[255] = '\0';
 newDir->files = NULL;
 newDir->parent = parent;
 newDir->next = NULL;
```

- 새 디렉토리를 만드는 함수
- 새 디렉토리에 대해 메모리를 할당
- 디렉토리 이름을 dirName에 복사
- 포인터 파일, parent, next 및 subDir를 초기화

AddSubDirectory

```
void addSubDirectory(const char* dirName) {
 Directory* newDir = createDirectory(dirName, currentDir);
 newDir->next = currentDir->subDirs;
 currentDir->subDirs = newDir;
 currentDir = newDir;
 printf("New directory '%s' created and moved into it.\n", dirName);
}
```

- 현재 디렉터리에 새 하위 디렉터리를 추가하고 해당 디렉터리로 이동하는 함수
- createDirectory를 호출하여 새 디렉토리를 생성
- currentDir의 하위 디렉토리 목록 맨 앞에 생성한 디렉토리를 삽입
- currentDir를 생성한 디렉토리로 업데이트합니다.

MoveToParentDirectory

```
void moveToParentDirectory() {
 if (currentDir->parent != NULL) {
 currentDir = currentDir->parent;
 printf("Moved to parent directory '%s'.\n", currentDir->dirName);
 } else {
 printf("Already in the root directory.\n");
 }
```

- 현재 디렉토리 포인터를 상위 디렉토리로 이동하는 함수
- currentDir에 부모가 있는지 확인
- 부모가 있으면 currentDir를 부모로 업데이트, 없으면 root directory

moveToDirectory

```
void moveToDirectory(const char* dirName) {
 Directory* subDir = currentDir->subDirs;
 while (subDir != NULL) {
 if (strcmp(subDir->dirName, dirName) == 0) {
 currentDir = subDir;
 printf("Moved to directory '%s'.\n", dirName);
 return;
 }
 subDir = subDir->next;
 }
 printf("Directory '%s' not found in current directory.\n", dirName);
```

- 현재 디렉토리 포인터를 지정된 하위 디렉토리로 이동하는 함수
- currentDir의 하위 디렉토리를 통해 dirName과 일치하는 항목을 검색
- 일치하는 항목이 발견되면 currentDir를 업데이트, 못찾으면 오류 메시지를 출력

clearClipboard

```
void clearClipboard() {
 FileNode* current = clipboardDir->files;
 while (current != NULL) {
 FileNode* temp = current;
 current = current->next;
 free(temp);
```

- 클립보드 디렉토리를 지우는 함수
- 클립보드Dir의 파일 목록을 next를 통해 탐색하고 각 파일 노드 메모리 할당 해제
- 클립보드Dir->files을 NULL로 설정

newfile

```
void newfile(const char* fileName) {
    // 같은 이름의 파일이 이미 존재하는지 확인합니다.
    FileNode* current = currentDir->files;
    while (current != NULL) {
        if (strcmp(current->fileName, fileName) == 0) {
            printf("File '%s' already exists.\n", fileName);
            return;
        }
        current = current->next;
    }

    FileNode* newFile = createFileNode(fileName);
    newFile->next = currentDir->files;
    currentDir->files = newFile;
    printf("File '%s' created successfully.\n", fileName);
}
```

- 현재 디렉토리에 새 파일을 만드는 함수
- createFileNode를 호출하여 새 파일 노드를 만듬
- currentDir의 파일 목록 맨 앞에 새 파일을 삽입

copy

```
void copy(const char* fileName) {
 if (currentDir == clipboardDir) {
 printf("Cannot copy files from the clipboard directory.\n");
 return;
 }
 FileNode* current = currentDir->files;
 while (current != NULL && strcmp(current->fileName, fileName) != 0) {
 current = current->next;
 }
 if (current == NULL) {
 printf("File '%s' not found in current directory.\n", fileName);
 return;
 }
 clearClipboard();
 FileNode* copiedFile = createFileNode(fileName);
 copiedFile->next = clipboardDir->files;
 clipboardDir->files = copiedFile;
 printf("File '%s' copied to clipboard.\n", fileName);
}
```

- 파일을 현재 디렉터리에서 클립보드 디렉토리로 복사하는 함수
- 현재 디렉토리가 클립보드 디렉토리가 아님을 확인
- currentDir에서 매개변수로 들어온 파일명과 같은 파일을 검색
- 존재하지 않으면 not found 출력, 있다면 clearClipboard실행

- 파일 노드를 클립보드Dir에 복사

removeFile

```
void removeFile(const char* fileName) {
 FileNode* prev = NULL;
 FileNode* current = currentDir->files;
 while (current != NULL && strcmp(current->fileName, fileName) != 0) {
 prev = current;
 current = current->next;
 }
 if (current == NULL) {
 printf("File '%s' not found in current directory.\n", fileName);
 return;
 }
 if (prev == NULL) {
 currentDir->files = current->next;
 } else {
 prev->next = current->next;
 }
 free(current);
 printf("File '%s' removed from current directory.\n", fileName);
}
```

- 현재 디렉터리에서 파일을 제거하는 함수
- currentDir에서 매개변수로 들어온 파일명과 같은 파일을 검색
- 존재하지 않으면 not found 출력,
- 제거할 파일이 첫 번째 노드일 경우 현재 디렉토리의 파일을 다음 파일로 업데이트
- 제거할 파일이 첫 번째 노드가 아닐 경우 이전 노드의 next 포인터를 현재 노드 다음의 노드를 가르키도록 업데이트
- 제거할 노드 메모리 해제

paste

```
void paste() {
 if (clipboardDir->files == NULL) {
 printf("Clipboard is empty.\n");
 return;
 }
 FileNode* fileToPaste = clipboardDir->files;
 // 파일을 현재 디렉토리로 이동합니다.
 FileNode* pastedFile = createFileNode(fileToPaste->fileName);
 pastedFile->next = currentDir->files;
 currentDir->files = pastedFile;
 // 클립보드를 비웁니다.
 clearClipboard();
 printf("File '%s' pasted to current directory.\n", pastedFile->fileName);
}
```

- 클립보드 디렉토리에서 현재 디렉토리로 파일을 붙여넣는 함수
- 클립보드가 비어 있는지 확인
- 클립보드Dir의 파일로 currentDir에 새 파일 노드를 만듬
- 클립보드를 비움

printDirectoryStructure

```
void printDirectoryStructure(Directory* dir) {
 printf("now - %s\n", dir->dirName);
 FileNode* currentFile = dir->files;
 while (currentFile != NULL) {
  printf("|--- %s\n", currentFile->fileName);
  currentFile = currentFile->next;
 }
 Directory* currentSubDir = dir->subDirs;
 while (currentSubDir != NULL) {
  printf("|--- <DIR> %s\n", currentSubDir->dirName);
  currentSubDir = currentSubDir->next;
 }
}
```

- 디렉토리 구조 출력하는 함수
- 디렉토리 이름 출력, 디렉토리에 있는 모든 파일 및 하위 디렉토리 나열

3. Demo

처음 파일을 실행한 결과는 Figure 1에서 확인할 수 있다. 기본 디렉토리의 위치는 Root이며 사용자가 사용할 수 있는 명령어의 목록을 나열함으로써 UI를 구현하였다. newfile 명령어를 사용하여 파일명 minios파일을 생성하면 파일목록에 minios가 추가된 것을 Figure 3을 통해 확인 할 수 있다. Mkdir 명령어를 사용하여 Project 디렉토리를 생성하면 디렉토리가 생성하고 현재 디렉토리가 생성한 디렉토리로 변경한 것을 Figure 5를 통해 확인할 수 있다. Copy 명령어의 경우 파일명을 입력하면 clipboard에 복사되고 paste명령어를 통해 복사할 수 있다. 그에 따른 결과는 Figure 8에서 확인할 수 있다. Remove 명령어는 입력한 파일명을 디렉토리 내에서 찾아 삭제하는 기능이며 Figure 10을 통해 확인할 수 있다. 마지막으로 cd 명령어이다. cd명령어는 cd .. 와 cd (디렉토리명) 을 통해 각각 상위 디렉토리와 하위 디렉토리로 이동할 수 있으며 Figure 12를 통해 하위 디렉토리로 이동하는 결과를 보여준다.

Figure 1 : Base

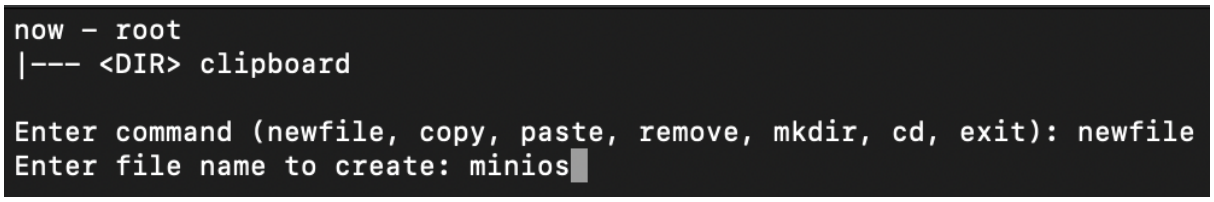


Figure 1 : newfile

Figure 2 : newfile 결과

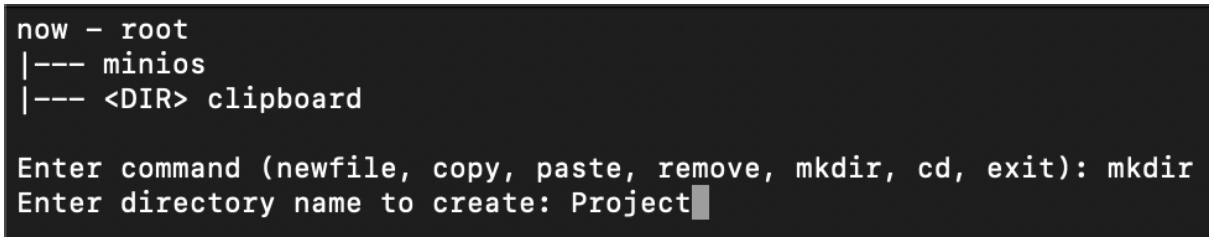


Figure 3 : mkdir

Figure 4 : mkdir 결과

Figure 5 : copy

Figure 6 : paste

Figure 7 **: copy & paste 결과**

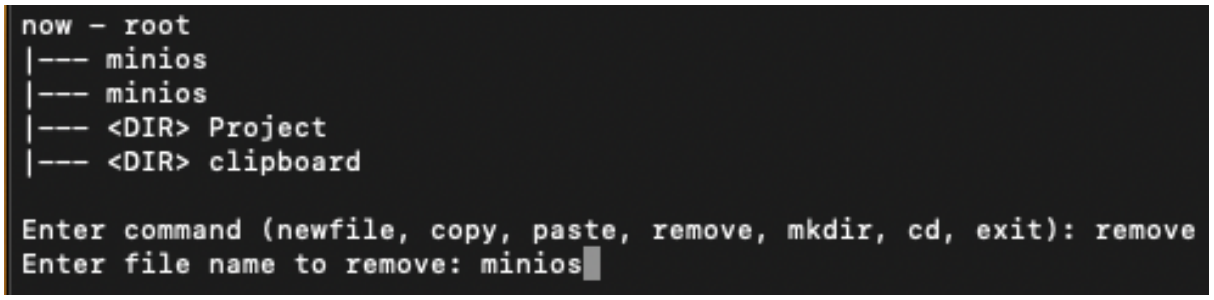


Figure 8 : remove

```
now - root
|--- minios
|--- <DIR> Project
|--- <DIR> clipboard

Enter command (newfile, copy, paste, remove, mkdir, cd, exit):
```

Figure 9 : remove 결과

```
now - Project

Enter command (newfile, copy, paste, remove, mkdir, cd, exit):
```

Figure 10 : cd

```
now - Project

Enter command (newfile, copy, paste, remove, mkdir, cd, exit): exit
[MiniOS SSU] MiniOS Shutdown.....%
(base) jangjin-yeong@jangjin-yeong-ui-MacBookAir minios %
```

Figure 11 **: cd 결과**

```
now - root
|--- minios
|--- <DIR> Project
|--- <DIR> clipboard

Enter command (newfile, copy, paste, remove, mkdir, cd, exit): cd Project
```

Figure 12 : exit

iii. Conclusion

1. Trouble Shooting

이번 프로젝트에서는 linux 환경에서 파일과 디렉토리를 생성하는 기능을 구현하고자 하였으나 세그멘테이션 오류가 발생하였다. 이는 linux 의 환경변수를 설정하고 포인터를 이용해 올바르게 접근해야 했으며, 본 프로젝트에서는 File System의 기본적인 동작 구조와 원리를 간단하게 살펴보고자 가상의 root 디렉토리 아래에서 동작하도록 구현하였다.

프로젝트 진행 시 파일생성, 파일 복사 명령어가 동일한 파일이 있어도 실행되는 문제를 보였다. 실제 파일 시스템에서 각 디렉토리의 파일 이름은 고유해야 하기 때문에 실제 파일시스템과 유사한 기능을 수행하기 위해 중복된 파일이 있을 시 명령어를 실행하지 않는 작업이 필요하다. 따라서 디렉토리에 있는 모든 파일명을 비교하여 같은 파일명이 존재하지

않을 때 파일생성과 복사 명령어가 실행되도록 구현하였다. 변경한 코드는 Figure 13과 14에서 확인할 수 있고 실행결과는 Figure 15에서 확인할 수 있다.

```
void newfile(const char* fileName) {
    // 같은 이름의 파일이 이미 존재하는지 확인합니다.
    FileNode* current = currentDir->files;
    while (current != NULL) {
        if (strcmp(current->fileName, fileName) == 0) {
            printf("File '%s' already exists.\n", fileName);
            return;
        }
        current = current->next;
    }

    FileNode* newFile = createFileNode(fileName);
    newFile->next = currentDir->files;
    currentDir->files = newFile;
    printf("File '%s' created successfully.\n", fileName);
}
```

Figure 13 : 수정된 newfile함수

```
void paste() {
 if (clipboardDir->files == NULL) {
 printf("Clipboard is empty.\n");
 return;
 }
 FileNode* fileToPaste = clipboardDir->files;
// 같은 이름의 파일이 이미 존재하는지 확인.
    FileNode* current = currentDir->files;
    while (current != NULL) {
        if (strcmp(current->fileName, fileToPaste->fileName) == 0) {
            printf("File '%s' already exists.\n", fileToPaste->fileName);
            return;
        }
        current = current->next;
    }
 FileNode* pastedFile = createFileNode(fileToPaste->fileName);
 pastedFile->next = currentDir->files;
 currentDir->files = pastedFile;
 clearClipboard();
 printf("File '%s' pasted to current directory.\n", pastedFile->fileName);
}
```

Figure 14 : 수정된 paste함수

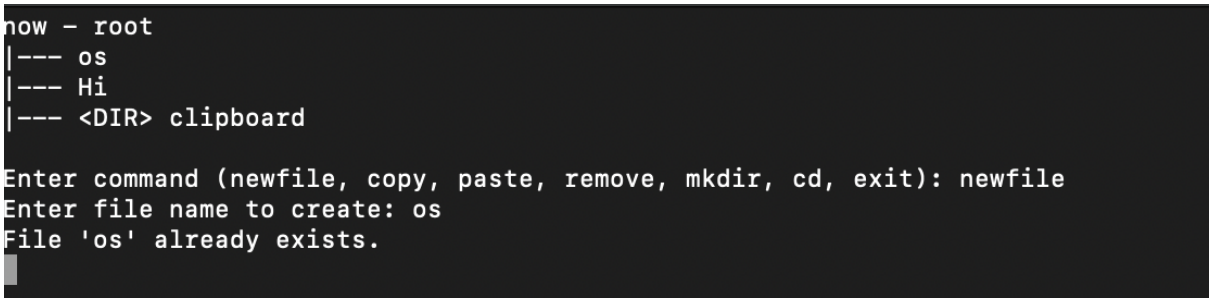


Figure 15 : 실행결과

2. Lesson Learning about OS

   컴퓨터의 시스템을 하드웨어와 소프트웨어 간의 상호작용과 프로세스 관리 등을 어떻게 구현할 지 고민하면서 이해할 수 있었다.

   시스템을 안정적으로 유지하고 관리하기 위해 어떤 방법을 사용하는지 알아보았고 실습을 통해 배운 이론을 실제로 적용해 봄으로써 시스템 관리의 중요성을 더욱 깊이 이해하게 되었다. 또한 OS가 단순히 하드웨어와 사용자 간의 중개 역할을 하는 것에 그치지 않고 시스템의 안정성과 효율성을 보장하는 중요한 소프트웨어임을 느꼈다.

   파일 시스템을 기반으로 운영체제를 구현하고 시각적으로 확인할 수 있어서 학습에 대한 깊은 이해도를 가질 수 있었다.

3. Lesson Learning about Team Project

   다수의 팀원들과 분업시 코드를 합칠 때 많은 문제점이 발생했다 따라서 프로젝트의 일관성을 유지하기 위해 조원들과 많은 소통과 협업이 필요함을 배울 수 있었으며 공동작업을 위한 Github 사용법을 익힐 수 있었다.