DKU 단국대학교 DANKOOK UNIVERSITY

REPORT

파일 시스템 시뮬레이터

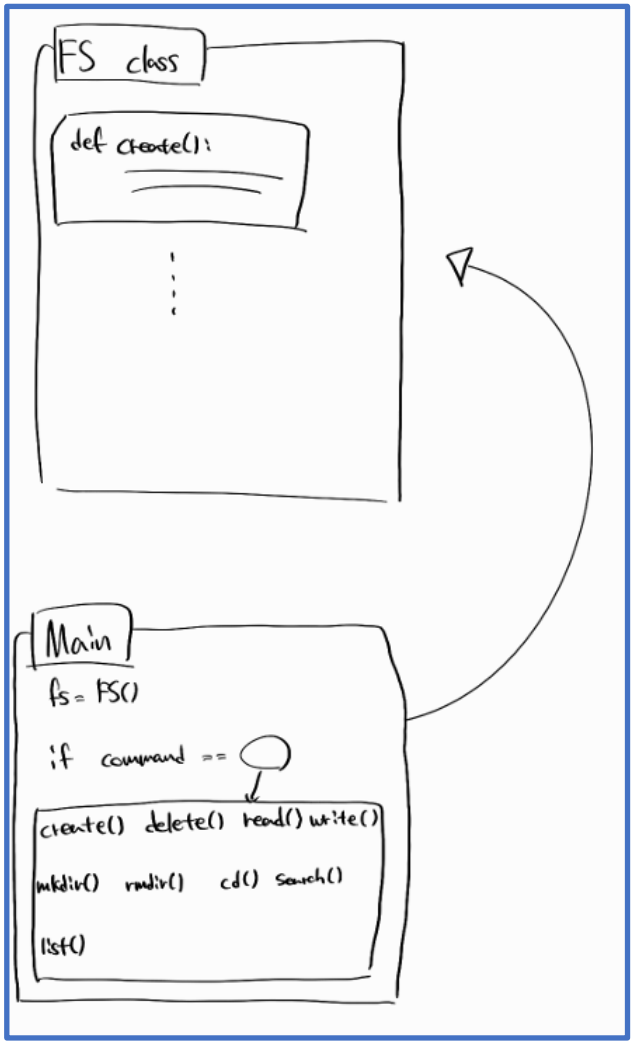
주 제 : 파일 시스템 시뮬레이터
교 수 : 김하연
소속학과: 소프트웨어
학 번 : 32207825
이 름 : 정재호
마 감 : 2024.06.11

단국대학교 DANKOOK UNIVERSITY

- 목차

■ 프로젝트 설계

➢ 기본 요구 사항 Ver.1

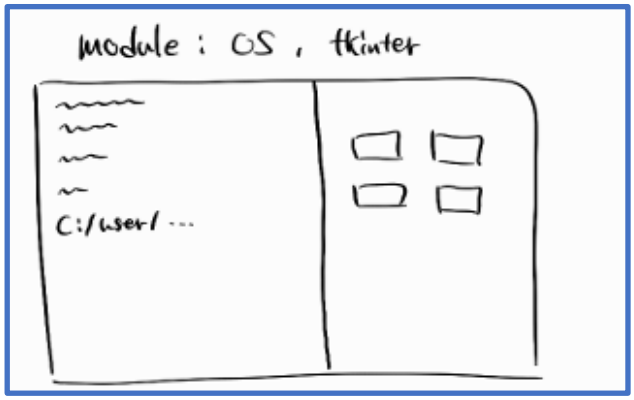


파일시스템의 기본적인 구조는 main에서 input값을 받아 명령어를 분리해내고, 명령어에 맞는 메소드를 실행시키면 될 것 같습니다. While문 안에 명령어를 분류하는 조건문들을 넣어서 별도의 종료 명령어를 입력하지 않으면 계속 실행되게 하면 좋을 것 같습니다. 그리고 메소드들은 FileSystem 클래스안에 넣어줬습니다.

입력 받은 값은 명령어 + 파일이름 + 파일 내용과 같은 형식인데 strip()이나 split()을 사용해서 쪼개어 관리해주면 될 것 같습니다.

하지만 이런 식으로 하면 긴 입력 값을 타이핑 해야 한다는 점이나 파일 내용을 쓰는 경우에는 입력 값을 쪼개는데 사용하는 구분점을 문자로 입력하기 복잡 해진다는 단점이 있었습니다. 예를 들어 구분점을 스페이스로 한다면 파일 내용을 입력할 때 스페이스 없는 문장을 입력해야 하기 때문에 파일 시스템의 퀄리티가 많이 떨어지는 모습을 보입니다. 따라서 GUI를 추가하며 이런 단점들을 제거한 버전으로 만들어볼 생각입니다.

➢ 기본 요구 사항 Ver.2



예상하는 GUI의 모습은 이런 형태이 될 것 같습니다. 왼쪽에는 list명령어를 입력해야 나오는 현재 디렉토리의 요소들을 계속 출력해주고, 오른쪽은 작업할 명령어들에 관련한 내용으로 채우려고 합니다.

os 모듈의 현재 패스를 불러오는 기능을 사용해서 현재 패스를 init안에 초기화해놓아서 클래스 객체를 만들면 모든 메소드들이 공유 할 수 있게 해주면 좋을 것 같습니다. 그리고 create와 같은 메소드를 실행하면 마지막에 display 메소드를 호출해서 변경된 부분을 바로 볼 수 있게 해주려 합니다.

display() 메소드 같은 경우에는 현 디렉토리의 요소들을 리스트에 담아 화면에 하나씩 표현해주는 방식을 사용하면 좋을 것 같습니다.

Main은 tkinter 모듈을 사용해서 필요한 GUI를 구성해보려 하는데 입력받을 공간과 성공여부 등을 표현할 출력 공간, 메소드를 호출할 버튼 등이 필요할 것으로 보입니다.

■ 주요 구현 사항

➢ 라이브러리

```
1  import tkinter
2  from tkinter import messagebox
3  import os
```

os모듈로 파일, 디렉토리 작업을 하려고 합니다. 이번 과제에서는 GUI를 추가해보고 싶어서 tkinter라는 모듈을 사용하게 되었습니다.

➢ class FileSystem

```
5  # [메소드 구간]
6
7  # 파일 시스템 클래스
8  class FileSystem:
9      # 초기화
10     def __init__(self):
11         # 현재 경로(os 모듈로 알아낸 경로는 str값입니다.)
12         self.curPath = os.getcwd()
```

파일 시스템을 만들 때 클래스로 만들어서 현재의 path 를 초기화 해줍니다.
여기서 os.getcwd()로 가져온 값은 str 값입니다.
pathlib 을 사용하는 경우에는 경로 객체로 가져오기 때문에 따로 str 으로 만들어주는 작업을 해줄 필요가 있는데, os 모듈은 그럴 필요가 없다는 특징이 있습니다.
이외엔 클래스 객체로 작업하기 때문에 매번 현재 경로를 불러오는 코드를 줄이고, 후에 display 메소드를 작동시키는 과정에서 복잡함을 줄여줄 수 있습니다.

- def createFile

```
14      # 파일 생성 메소드
15      def createFile(self):
16          # 입력값을 받아옵니다.
17          fileName = entryFilename.get()
18          content = entryContent.get()
19
20          # 현재 경로 + 파일 이름을 해준 값을 filePath에 저장합니다.
21          filePath = os.path.join(self.curPath, fileName)
22
23          if fileName:
24              try:
25                  with open(filePath, 'w') as file:
26                      if content:
27                          file.write(content)
28                      else:
29                          file.write("") # content를 입력하지 않으면 빈 파일을 생성합니다.
30                  statusLabel.config(text=f"'{fileName}' 파일을 생성했습니다.")
31              except Exception as e:
32                  messagebox.showerror("Error", f"파일을 생성하지 못했습니다 : {e}")
33          else:
34              messagebox.showwarning("Input Error", "파일의 이름을 입력하세요.")
35          # 메소드 종료 전에 display를 호출해서 화면 갱신을 해줍니다.
36          self.display()
```

파일을 생성하는 메소드입니다. fileName과 content변수는 후에 서술할 tkinter의 입력칸을 통해 입력 받게 된 파일이름, 파일내용 값을 저장합니다. filePath는 현재의 경로 변수인데 os.path.join()을 사용해서 self.curPath와 fileName변수를 합쳐줍니다. 후에는 조건문으로 에러처리를 해주면서 파일 생성에 대한 코드를 작성했습니다. fileName이 있다면, open()내장함수로 파일 작업을 해줍니다.

```
open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)
```

open() 함수의 구조는 위 그림과 같고 mode는 r, w, x, a, b, t, +가 있으며 여기서는 w로 쓰기작업을 하게 됩니다. 마지막으로 입력 받은 content값이 있으면 파일에 내용을 작성해주고 없으면 빈 파일을 생성해줍니다.

성공적으로 파일을 생성 했다는 걸 사용자에게 보여주기 위해서 tkinter의 Label에다 글씨를 출력해줍니다. 여기서 statusLabel이 성공여부를 보여주는 Label이고 config(text=보여줄 텍스트)와 같은 방법으로 출력이 가능합니다.

파일 생성이 끝났다면 self.display()를 해줘서 현재 디렉토리의 list를 표현해주는 메소드를 불러옵니다.

- def deleteFile

```
38      # 파일 삭제 메소드
39      def deleteFile(self):
40          fileName = entryFilename.get()
41          filePath = os.path.join(self.curPath, fileName)
42
43          if fileName:
44              # 파일이 있는지 검사해주고, 있으면 삭제해줍니다.
45              if os.path.exists(filePath):
46                  try:
47                      os.remove(filePath)
48                      statusLabel.config(text=f"'{fileName}' 파일을 삭제했습니다.")
49                  except Exception as e:
50                      messagebox.showerror("Error", f"'{e}' 파일을 삭제하지 못했습니다.")
51              else:
52                  messagebox.showwarning("File Not Found", f"'{fileName}' 파일을 찾을 수 없습니다.")
53          else:
54              messagebox.showwarning("Input Error", "파일의 이름을 입력하세요.")
55          self.display()
```

파일 삭제 메소드는 파일이름과 현재 경로를 나타내는 변수를 저장해줍니다. os.path.exists()로 입력 받은 파일이 존재하는지 확인하고, 존재한다면 os.remove()로 현재파일의 경로를 입력해서 파일을 삭제해줍니다.

파일 삭제 작업이 끝났다면 self.display()를 해서 현재 디렉토리 list를 GUI에 갱신해줍니다.

- def readFile

```
57      # 파일 읽기 메소드
58      def readFile(self):
59          fileName = entryFilename.get()
60          filePath = os.path.join(self.curPath, fileName)
61
62          if fileName:
63              # 파일이 있으면 읽고 없으면 에러처리 해줍니다.
64              if os.path.isfile(filePath):
65                  try:
66                      with open(filePath, 'r') as file:
67                          statusLabel.config(text=f"'{fileName}' 파일을 읽어옵니다.")
68                          printLabel.config(text=file.read())
69                  except Exception as e:
70                      messagebox.showerror("Error", f"'{e}' 파일을 읽지 못했습니다.")
71              else:
72                  messagebox.showwarning("File Not Found", f"'{fileName}' 파일을 찾을 수 없습니다.")
73          else:
74              messagebox.showwarning("Input Error", "파일의 이름을 입력하세요.")
```

파일 읽기 메소드는 os.path.isfile()을 통해 파일이 존재하는지 확인하고 open()으로 파일 읽기 작업을 해줍니다. printLabel은 tkinter로 만든 파일 내용 출력을 위한 Label인데 여기에 불러온 파일 내용을 넣어주면 GUI로 확인할 수 있습니다.

- def writeFile

```
75      # 파일 쓰기 메소드
76      def writeFile(self):
77          fileName = entryFilename.get()
78          content = entryContent.get()
79          filePath = os.path.join(self.curPath, fileName)
80
81          if fileName:
82              # 파일이 있으면 쓰고 없으면 에러처리 해줍니다.
83              if os.path.isfile(filePath):
84                  try:
85                      with open(filePath, 'a') as file:
86                          file.write(content)
87                          statusLabel.config(text=f"'{fileName}' 파일의 내용을 썼습니다.")
88                  except Exception as e:
89                      messagebox.showerror("Error", f"'{e}' 파일을 쓰지 못했습니다.")
90              else:
91                  messagebox.showwarning("File Not Found", f"'{fileName}' 파일을 찾을 수 없습니다.")
92          else:
93              messagebox.showwarning("Input Error", "파일의 이름을 입력하세요.")
```

파일 쓰기 작업은 파일의 내용을 추가하는 작업을 해줘야 하는데, 먼저 파일이 있는지 확인해주고 open()을 통해 파일 내용을 쓰는 작업을 해줍니다.

- def searchFile

```
96      # 파일 검색 메소드
97      # 파일 검색 시 현재 디렉토리와 하부 디렉토리를 검사하게 해줍니다.
98      # TODO : 과도한 양의 파일을 검사하는 경우 실패할 수 있음
99      def searchFile(self):
100         fileName = entryFilename.get()
101
102         if fileName:
103             # os.walk로 self.curPath부터 하위 디렉토리를 재귀 탐색합니다
104             # 각 반복에서 세개의 값을 반환합니다.
105             # - root는 현재 탐색중인 디렉토리 경로
106             # - dirs는 root 디렉토리의 하위 디렉토리 목록
107             # - files는 root 디렉토리에 있는 파일 목록
108             for root, dirs, files in os.walk(self.curPath):
109                 if fileName in files:
110                     statusLabel.config(text=f"'{fileName}' 파일을 찾았습니다.")
111                     printLabel.config(text=os.path.join(root, fileName))
112                     return
113             statusLabel.config(text=f"'{fileName}' 파일을 찾지 못했습니다.")
114         else:
115             messagebox.showwarning("Input Error", "디렉토리 명을 입력하세요.")
116         self.display()
```

파일 찾기는 os.walk(현재 경로)를 통해 하위 디렉토리를 재귀탐색해서 구현해보려 합니다. 한번의 실행마다 세개의 값을 반환하게 되고, 일치하는 파일이름이 files에 존재한다면 파일을 찾게 됩니다. 파일을 찾은 시점의 디렉토리 경로와 파일명을 합친 값을 출력Label에 입력해줘서, 파일이 어디에 위치하는지 알게 해줍니다.

- def mkDir

```
117     # 디렉토리 생성 메소드
118     def mkDir(self):
119         # 입력받은 디렉토리명
120         dirName = entryDirname.get()
121         # 현재경로 + 입력받은 디렉토리
122         dirPath = os.path.join(self.curPath, dirName)
123 
124         # 디렉토리 생성
125         if dirName:
126             try:
127                 os.makedirs(dirPath, exist_ok = True)
128                 statusLabel.config(text=f"'{dirPath}' 디렉토리를 생성했습니다.")
129             except Exception as e:
130                     messagebox.showerror("Error", f"'{e}' 디렉토리를 생성하지 못했습니다.")
131         else:
132             messagebox.showwarning("Input Error", "디렉토리 명을 입력하세요.")
133         self.display()
```

디렉토리 생성을 위해서 먼저 entryDirname을 통해 입력받은 값을 dirName에 저장해주고, dirName과 현재경로를 join을 통해 합쳐서 dirPath 변수에 저장해줍니다.

os.makedirs()은 새로 만들 디렉토리를 '현재경로' + '만들 디렉토리 명'로 받기 때문에 만들어준 dirPath를 사용해 디렉토리 생성 작업을 수행해줍니다.

디렉토리를 생성했다면 display()를 호출해줍니다.

- def rmDir

```
135     # 디렉토리 제거 메소드
136     def rmDir(self):
137         dirName = entryDirname.get()
138         dirPath = os.path.join(self.curPath, dirName)
139 
140         if dirName:
141             # 현재 디렉토리 + 입력받은 디렉토리인 디렉토리가 존재하는지 검사
142             if os.path.isdir(dirPath):
143                 try:
144                     os.rmdir(dirPath)
145                     statusLabel.config(text=f"'{dirPath}' 디렉토리를 삭제했습니다.")
146                 except Exception as e:
147                     messagebox.showerror("Error", f"'{e}' 디렉토리를 삭제하지 못했습니다.")
148             else:
149                 messagebox.showwarning("Dir Not Found", f"'{dirPath}' 디렉토리가 없습니다.")
150         else:
151             messagebox.showwarning("Input Error", "디렉토리 명을 입력하세요.")
152         self.display()
```

디렉토리 제거는 os.path.isdir()로 디렉토리가 존재하는지 검사해주고 os.rmdir()에 dirPath값을 넣어 디렉토리를 삭제해줍니다.

디렉토리를 제거한 이후 display()를 호출해줍니다.

- def changeDir

```
154     # 체인지 디렉토리 메소드
155     def changeDir(self):
156         beforPath = self.curPath
157         dirName = entryDirname.get()
158         dirPath = os.path.join(self.curPath, dirName)
159 
160         if dirName:
161             if os.path.isdir(dirPath):
162                 try:
163                     self.curPath = dirPath
164                     statusLabel.config(text=f"'{beforPath}' 에서 '{dirPath}' 디렉토리로 이동했습니다.")
165                 except Exception as e:
166                     messagebox.showerror("Error", f"'{e}' 디렉토리를 이동하지 못했습니다.")
167             else:
168                 messagebox.showwarning("Dir Not Found", f"'{dirPath}' 디렉토리가 없습니다.")
169         else:
170             messagebox.showwarning("Input Error", "디렉토리 명을 입력하세요.")
171         self.display()
```

디렉토리 이동 메소드는 먼저 os.path.isdir()로 이동하고 싶은 디렉토리가 존재하는지 확인하고 self.curPath값을 '현재 경로' + '입력받은 디렉토리 명' 값을 가지는 dirPath로 바꿔줍니다.

이전과 같이 display를 호출해서 화면을 갱신해줍니다.

- def display

```
173     # 화면 왼쪽에 파일 리스트와 디렉토리를 표현해주는 메소드
174     def display(self):
175         # 리스트를 보여 주기 전에 기존에 있는 값을 정리해줍니다.
176         # 리스트박스 내용물의 삭제는 0~(size-1)
177         listboxSize = listbox.size()
178         listbox.delete(0, int(listboxSize) - 1)
179 
180         elements = os.listdir(self.curPath)
181 
182         # listbox에 현재 파일의 구성요소들과 디렉토리를 넣어줍니다.
183         rowCount=0
184         for path in elements:
185             listbox.insert(rowCount, path)
186             rowCount += 1
187         listbox.pack()
188         scrollbar["command"]=listbox.yview
```

현재 디렉토리에 있는 파일과 폴더들을 표현해주기 위해서 요소들을 tkinter의 listbox에 넣어 출력해주게 되었습니다. 우선 리스트박스의 내용물을 제거해주고 elements 리스트에 os.listdir한 값을 지정해줍니다. 그리고 반복문을 통해 한 개씩 리스트박스에 넣어줍니다. 그리고 tkinter의 pack()을 써서 화면에 표현하게 됩니다. 현재 디렉토리에 많은 요소들이 있을 수 있기에 스크롤바 기능도 넣었습니다.

➢ GUI, main 구간

```
191 # [GUI, main 구간]
192
193 # 파일 시스템
194 fs = FileSystem()
195
196 # 기본 윈도우 생성
197 win = tkinter.Tk()
198 win.title("OS_FS_Jeong_Jae_Ho")
199 win.geometry("1280x600+300+300")
200 win.resizable(False, False)
201 win.iconbitmap('C:/Users/sean1/icon.ico')
202
203 # 프레임을 두개로 좌우 화면을 나눠줬습니다.
204 frame1 = tkinter.Frame(win)
205 frame1.pack(side="left", fill="both", expand=True)
206 frame2 = tkinter.Frame(win)
207 frame2.pack(side="right", fill="both", expand=False)
208
209 # 왼쪽화면은 리스트박스가 들어간 스크롤이 가능한 프레임
210 scrollbar=tkinter.Scrollbar(frame1)
211 scrollbar.pack(side="right", fill="y")
212 listbox=tkinter.Listbox(frame1, width=100, height=90, yscrollcommand = scrollbar.set)
213 # display메소드로 초기 파일들의 구성을 보여줍니다.
214 fs.display()
```

파일시스템 객체를 만들어주고 tkinter.TK()로 화면을 생성해줍니다. 화면의 크기, 크기변경 여부, 아이콘 등을 설정해줬습니다.

화면의 왼쪽은 list를 보여주고 싶고 오른쪽은 명령어들과 입력 받을 값들을 정리해주면 깔끔할 것 같아서 Frame을 통해 화면을 둘로 나누게 되었습니다.

왼쪽 화면의 list는 listbox기능을 통해 현재 디렉토리의 요소들을 출력한다 설명 했었는데, Scrollbar로 스크롤이 가능한 화면을 만들기 위해서는 scrollbar에 listbox를 넣어줘야 합니다. 따라서 먼저 scrollbar를 frame1에 할당해줍니다.

그리고 fs.display()를 호출해줘서 윈도우를 키자마자 리스트가 출력되게 해주었습니다.

```
220 # 오른쪽 프레임들은 각종 명령어와 입력창을 넣어줬습니다.
221 # create, delete, read, write, search 명령을 위한 설명, 입력 부분입니다.
222 # grid 기능을 써서 행렬로 정리해줬습니다.
223 labelFileName = tkinter.Label(frame2, text="파일 이름")
224 labelFileName.grid(column=0, row=1)
225 labelContent = tkinter.Label(frame2, text="파일 내용")
226 labelContent.grid(column=0, row=2)
227 # 입력 창
228 entryFilename = tkinter.Entry(frame2, width=18)
229 entryFilename.grid(column=1, row=1)
230 entryContent = tkinter.Entry(frame2, width=18)
231 entryContent.grid(column=1, row=2)
```

Label은 글자를 출력할 수 있는 기능이고, Entry는 입력 받을 수 있는 공간을 만들

어주는 기능입니다. 파일 이름과 파일 내용을 입력 받을 entry를 생성해줍니다.

grid는 윈도우의 구성요소들을 행렬표로 배치해주는 기능인데, 이 기능을 사용해서 frame2의 구성요소들을 grid로 배치해주게 되었습니다.

```
233  # 각 버튼은 createFile, deleteFile, readFile, writeFile, searchFile 메소드를 실행시킵니다.
234  buttonCreate = tkinter.Button(frame2, text="create", command=fs.createFile, overrelief="solid", width=13)
235  buttonCreate.grid(column=2, row=1)
236  buttonDelete = tkinter.Button(frame2, text="delete", command=fs.deleteFile, overrelief="solid", width=13)
237  buttonDelete.grid(column=3, row=1)
238  buttonRead = tkinter.Button(frame2, text="read", command=fs.readFile, overrelief="solid", width=13)
239  buttonRead.grid(column=2, row=2)
240  buttonWrite = tkinter.Button(frame2, text="write", command=fs.writeFile, overrelief="solid", width=13)
241  buttonWrite.grid(column=3, row=2)
242  buttonSearch = tkinter.Button(frame2, text="search", command=fs.searchFile, overrelief="solid", width=13)
243  buttonSearch.grid(column=2, row=3)
```

파일 생성, 삭제, 읽기, 쓰기, 탐색 작업에 필요한 버튼들을 생성해주었습니다. 각 버튼은 command='수행할 메소드' 를 통해 버튼 클릭이 메소드의 실행을 시작하게 해줍니다.

```
249  # mkdir, rmdir, cd를 위한 설명, 입력 부분입니다.
250  labelDirName = tkinter.Label(frame2, text="디렉토리 명")
251  labelDirName.grid(column=0, row=5)
252  # 입력 창
253  entryDirname = tkinter.Entry(frame2, width=50)
254  entryDirname.grid(column=1, row=5, columnspan=3)
255
256  # 각 버튼은 mkDir, rmDir, cd 메소드를 실행시킵니다.
257  buttonMakeDir = tkinter.Button(frame2, text="mkdir", command=fs.mkDir, overrelief="solid", width=13)
258  buttonMakeDir.grid(column=1, row=6, padx=10, pady=10)
259  buttonRemoveDir = tkinter.Button(frame2, text="rmdir", command=fs.rmDir, overrelief="solid", width=13)
260  buttonRemoveDir.grid(column=2, row=6, padx=10, pady=10)
261  buttonCD = tkinter.Button(frame2, text="cd", command=fs.changeDir, overrelief="solid", width=13)
262  buttonCD.grid(column=3, row=6, padx=10, pady=10)
```

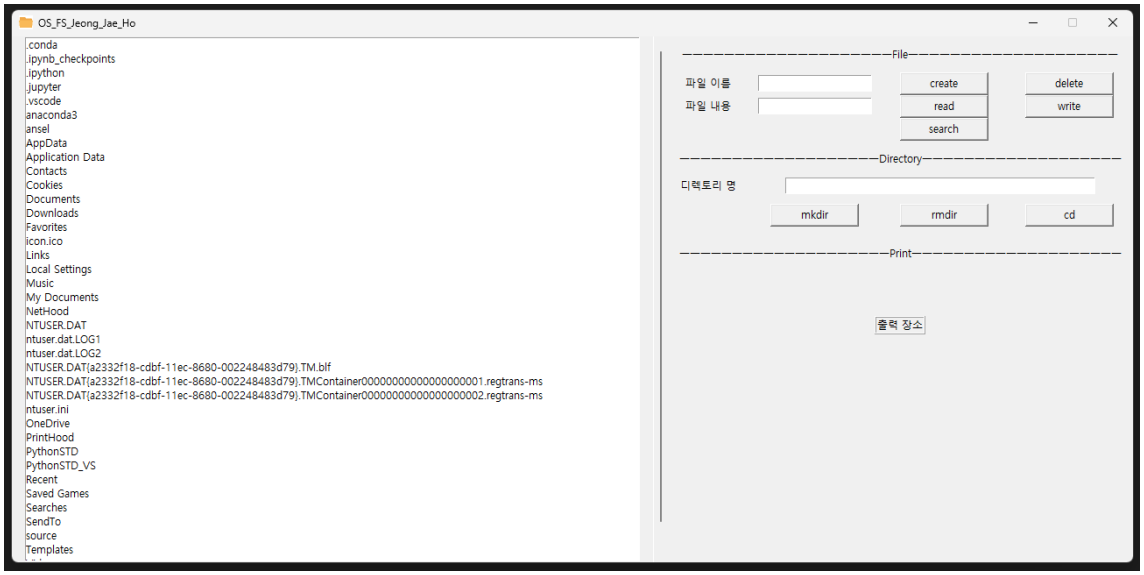
디렉토리 생성, 삭제, 이동을 위한 Label, Entry를 생성해주고 버튼을 생성해서 각 메소드를 실행할 수 있게 해줍니다.

```
268  # 하단에 작업 성공 결과를 알려줍니다.
269  statusLabel = tkinter.Label(frame2, text="", fg="red")
270  statusLabel.grid(column=0, row=8, columnspan=4, padx=10, pady=10)
271
272  # 불러온 결과를 보여줍니다.
273  printLabel = tkinter.Label(frame2, text="출력 장소", relief="ridge")
274  printLabel.grid(column=0, row=9, columnspan=4, padx=10, pady=10)
275
276  win.mainloop()
```

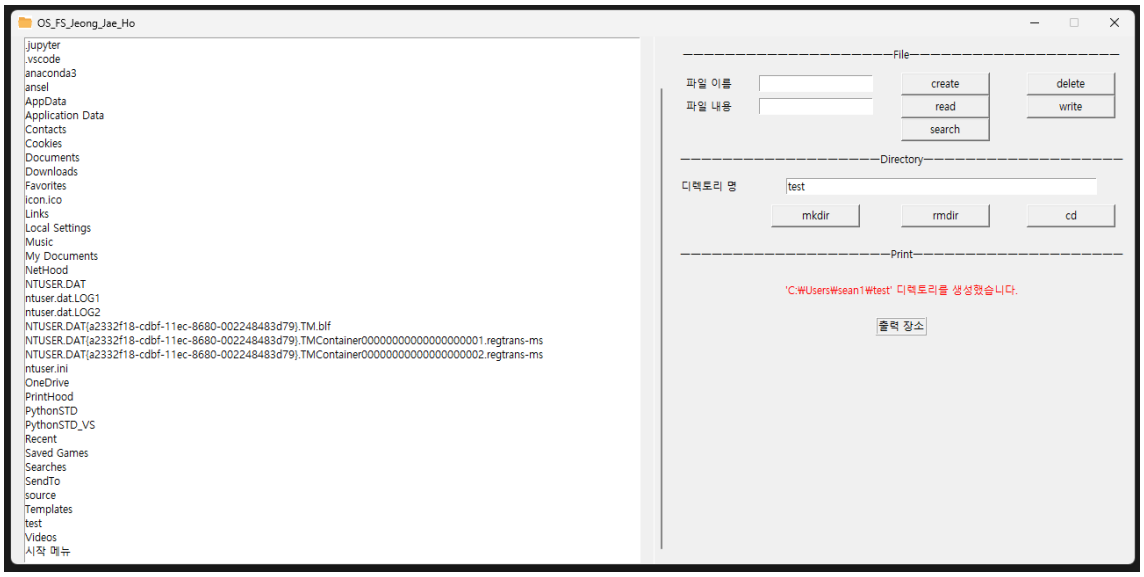
statusLabel은 작업 성공, 실패와 같은 알림을 출력해주는 Label입니다. printLabel은 파일 내용을 출력해주거나, 디렉토리 이동시 변경한 경로를 출력해주는 용도로 사용하게 되었습니다.

■ 실행 결과

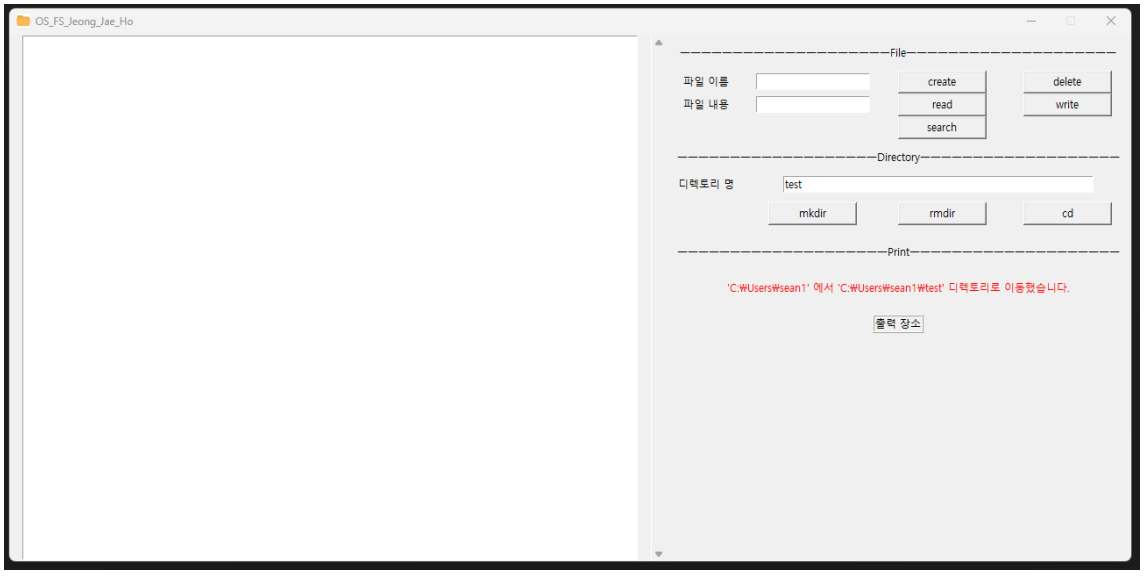
➢ 디렉토리 생성, 이동, 삭제와 파일 생성, 삭제, 읽기, 쓰기



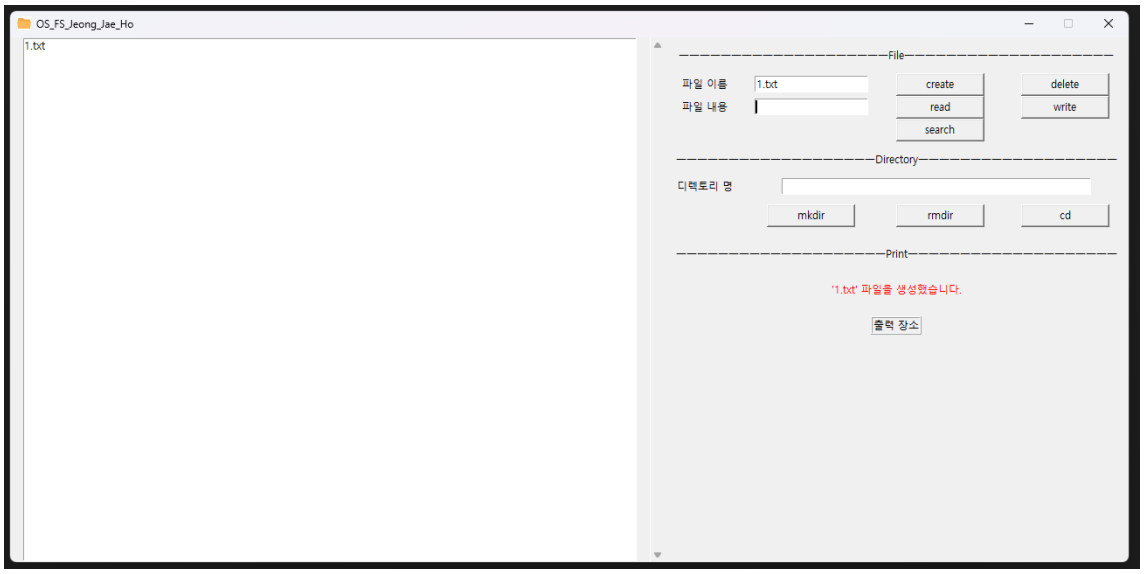
실행하면 현재 디렉토리의 구성요소들을 왼쪽(frame1)에 출력해줍니다. 오른쪽(frame2)에는 각 명령어 실행 버튼과 입력 창, 출력 장소가 보입니다.



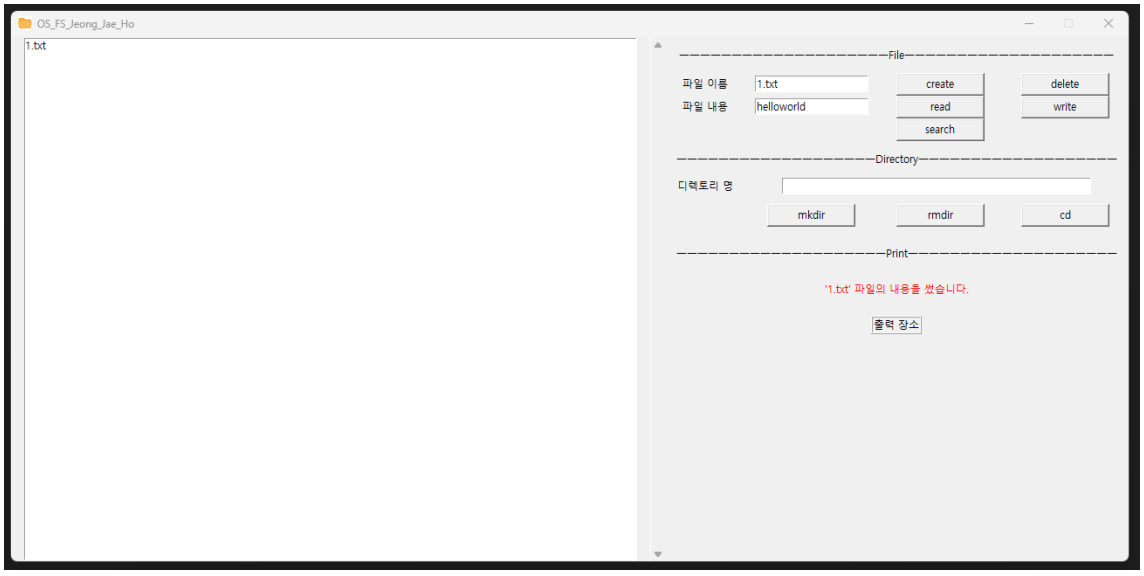
먼저 test디렉토리를 생성해보면 frame1에 test가 생성된 것을 확인할 수 있고 frame2에는 생성한 디렉토리의 절대 경로가 표시되는 것이 보입니다.



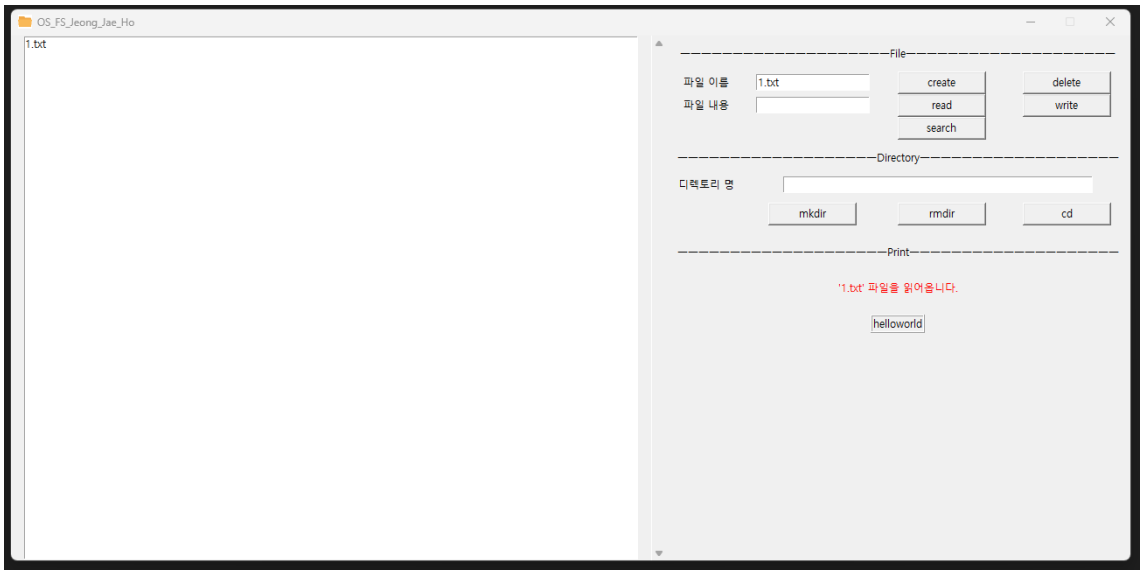
Cd로 test 디렉토리로 이동해줍니다.



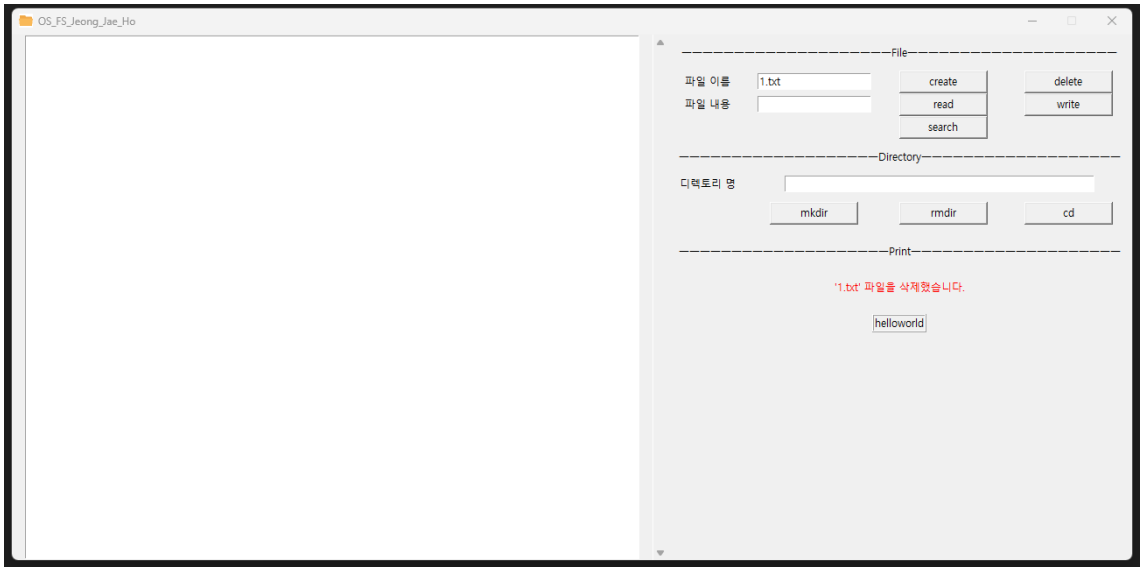
1.txt 파일을 생성했습니다.



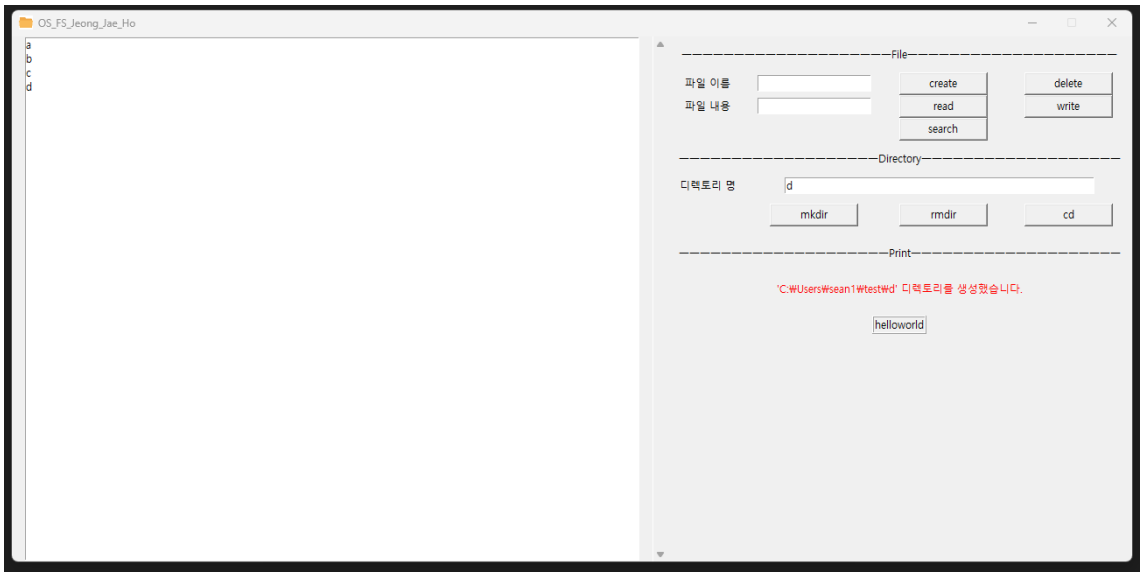
1. txt 파일에 write를 통해 helloworld를 추가해줬습니다.



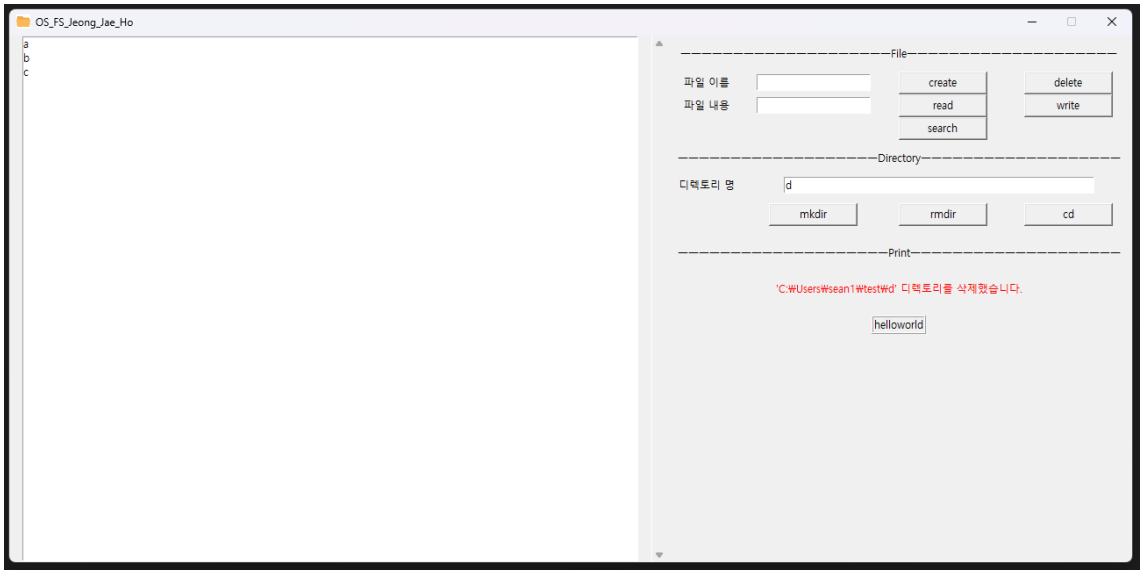
1.txt 파일을 읽어오면 helloworld 가 출력되는 것을 확인할 수 있습니다.



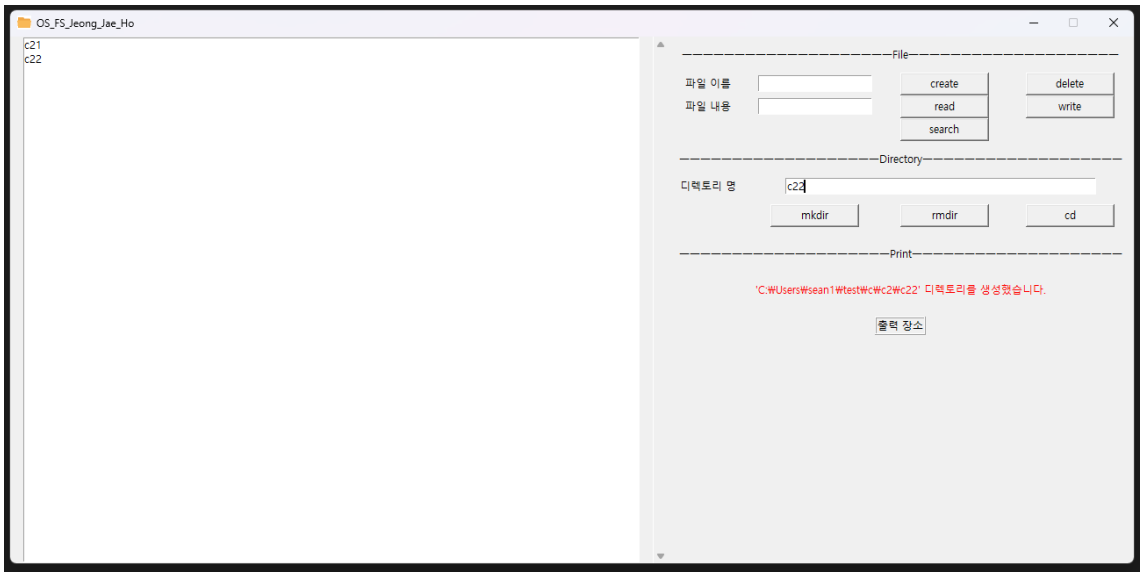
파일 삭제 시 정상적으로 삭제되어 frame1 에서 사라지는 것을 확인할 수 있습니다.



a, b, c, d 디렉토리를 추가해줍니다.



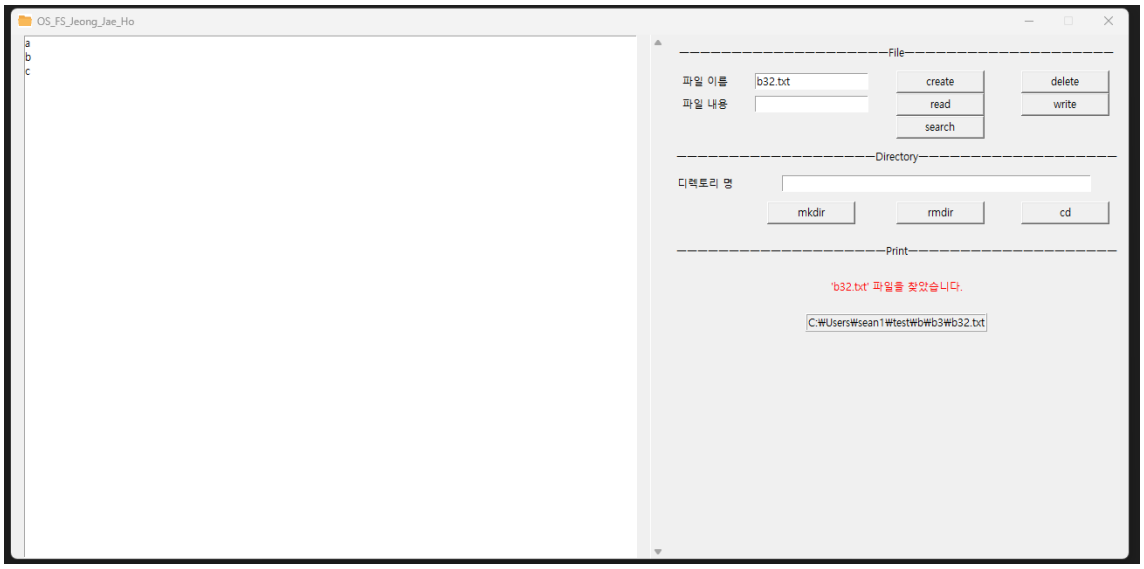
디렉토리 d 는 삭제하고 frame1 과 frame2 에 정상적으로 변경이 되었는지 확인했습니다.



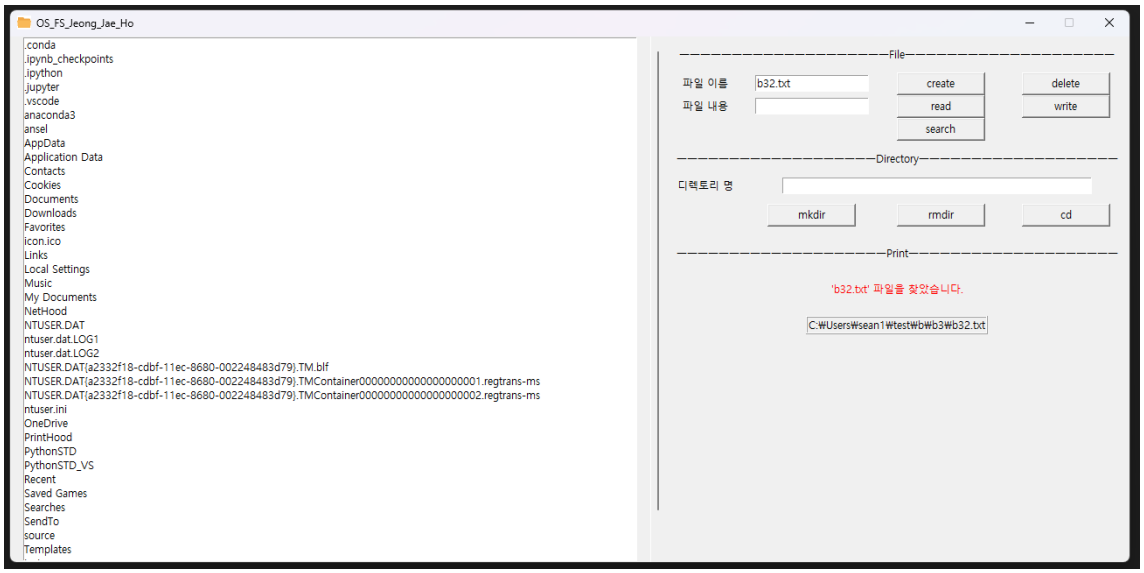
다음으로는 파일 탐색을 하기 위해서 a 디렉토리에 (a1, a2, a3, a4.txt) 를 추가해주고 b 디렉토리에는 (b1.txt, b2.txt, b3) 를 추가. b3 디렉토리에는 (b31.txt, b32.txt)를 추가. c 디렉토리에 (c1, c2, c3) 를 추가. c2 디렉토리에 (c21, c22)를 추가 하겠습니다.

➢ **파일 탐색**

test 디렉토리에서 b32.txt 파일을 찾으려 합니다.



성공적으로 파일의 경로를 추적해냈습니다.



그러면 test 디렉토리보다 상위 디렉토리인 sean1 디렉토리에서 검색을 해보았습니다. 사용자 정보가 담긴 sean1 디렉토리에는 수많은 파일이 있어서 검색에 시간이 좀 걸리는 점을 제외하면 성공적으로 찾아내었습니다.

- **느낀점**

초기에 os나 pathlib을 사용해서 파일 작업을 해주는 코드를 작성했는데, 생각보다 빨리 끝나서 GUI까지 해보면 어떨까 생각하게 되었습니다. 따라서 os모듈과 가장 간단해 보이는 tkinter를 사용해서 구현하게 되었습니다. 파이썬으로 GUI를 만드는 학습은 처음 해보는데 생각도 못한 로직으로 인해 오류가 많이 나게 되어 최대한 만들 수 있는 기능만 넣어 프로그램을 제작하게 되었습니다. 다행히 tkinter가 웹프로그래밍 할 때 쓰는 방식과 비슷한 점이 많아서 주어진 시간안에 완성할 수 있었습니다.

읽어주셔서 감사합니다.

- **참고 문헌**

https://076923.github.io/posts/Python-tkinter-1/

https://docs.python.org/ko/3/library/os.html#

https://docs.python.org/ko/3/library/functions.html#open