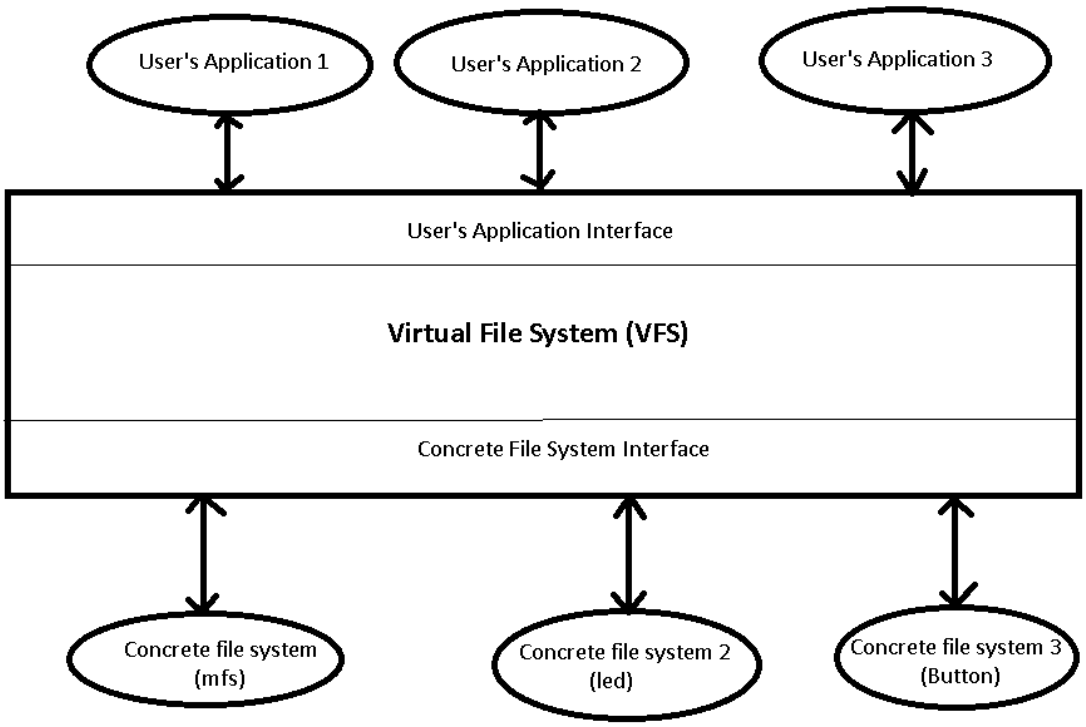

The diagram above represents my design for the file system problem. The acronym “mfs” stands for memory file system. The user’s applications call user’s application interface to avail the services of file system. The VFS (virtual file system) abstracts out the internal details of different devices and gives a common interface to user’s application and thus makes the interface device independent.

The details of application’s interface are as follows:

```
int vfs_open(const char *name,char mode)
Opens a file referred by name. The allowed modes are read, write and
append.
It returns a non-negative number which represents file descriptor of
the file opened by a user’s application.

void vfs_close(int fd)
Closes a open file referred by file descriptor fd.

int vfs_create(const char *name,char mode)
Creates a file with a given name if it does not exist and opens it in
a given mode.

void vfs_delete(const char *name);
Deletes a file referred by name.
```

```
int vfs_read(int fd,char *buff);
```
Reads one character from a file referred by file descriptor fd into buff at a position in file one byte next from previous read. When file is opened read starts from beginning of a file. It returns number of bytes read, i.e. 1 if read was successful, 0 otherwise (e.g. end of file). Multiple processes can open a file for read but if it is open for write or append, then the file cannot be opened.

```
int vfs_write(int fd,char ch);
```
Writes one character in file fd at a position in file one byte next to previous write. It returns number of bytes written, i.e. 1 if write is successful, 0 otherwise. Only one open for a file is allowed in write or append mode and further attempt to open file in any mode is unsuccessful.

```
void vfs_list()
```
Lists all the existing files in the file system on standard output.

The VFS has an interface which concrete file system is required to implement in order to hook themselves to VFS. As long as the requirements are fulfilled any concrete file systems can attach themselves to VFS and can be accessed through application's interface. Each interface that is implemented by concrete file systems has certain requirements and they have to fulfill certain promises. These requirements and promises are almost similar to user's application interfaces described earlier. Below is the additional information if any for each interface

The memory file system (mfs) implements the interface in the following functions.

```
void* mfs_open(const char *name,char mode);
```
Returns pointer to the stream object. This stream abstracts out memory file system related internal details.

```
void mfs_close(void *fp);
```
Expects corresponding stream object returned by open.

```
void mfs_create(const char *name);
```
No additional information.

```
void mfs_delete(const char *name);
```
No additional information.

```
int mfs_read(void *fp,char* buff);
```
Expects corresponding stream object returned by open.

```
int mfs_write(void *fp,char c);
Expects corresponding stream object returned by open.
```

```
char** mfs_list();
Returns array of file names in mfs and null indicates end of list.
```

The led and button file system have similar details.

Rules:

- File names are case sensitive
- Led file names are (case sensitive, e.g. /led/Orange will not work)
  /led/orange
  /led/yellow
  /led/blue
  /led/green
- Button file names are (case sensitive, e.g. /btn/Sw1 will not work)
  /btn/sw1
  /btn/sw2
- Memory file system allows only alpha numeric file names. First character of name should be an alphabet.
- When a file is open in write or append mode, it can be read on the same file descriptor. But is opened in read mode it cannot be written on same file descriptor.

Commands:

- **Open a file**:
  Usage: fopen <filename> <mode>
  mode can be 'r', 'w' or 'a'
  Eaxmples
  fopen foo a
  fopen /led/orange w
  fopen /btn/sw1 r
- **Close a file**:
  Usage: fclose <file descriptior>
  Examples
  fclose 1
  fclose 5
- **Read a file**:
  Usage: fgetc <file descriptor>
  Examples

fgetc 1
fgetc 3
reads 1 if led is on or 0 if off.
reads 1 if button is depressed , otherwise 0.

- **Write a file**:
  Usage: fputc <file descriptor> <char to write>
  Examples
  fputc 3 h
  fputc 5 1
  write 1 to turn led and 0 to turn off.
- **Create a file**:
  Usage: create <file name>
  Examples
  create foo
  create bar
- **Delete a file**:
  Usage: delete <file name>
  Examples
  delete foo
  delete bar
- **List all files**:
  Usage: list

Error Handling:

- If an invalid file name is entered. Only alpha-numeric cahracters are allowed and first character should be an alphabet.
- If led does not exist in the system. Also led names are case sensitive, e.g. Green or GREEN will not work. Only lower case letters are allowed.
- If button does not exist in the system. Also button names are case sensitive, e.g. Sw1 or SW1 will not work. Only lower case letters are allowed.
- Read when end of file is reached.
- Attempt to open in read when already open for write.
- Attempt to open in write if already open in read.
- If an invalid mode is entered. E.g. only ‘r’,’w’ or ‘a’ are allowed. ‘R’,’W’ and ‘A’ is also allowed.
- File size cannot be extended more than 512 bytes. Further write will fail.
- If a read or write is made on a file which is not opened in VFS.