The operating system is an important part of any modern computer. A few of the important tasks an operating system has are:

1. **CPU virtualization** - On modern computers, there are often many more running processes than there are computing cores available. It is the operating system's job to decide how to share the computing cores of a computer between all running processes.
2. **Memory virtualization and isolation** - Much like computing cores, memory is a limited physical resource in a computer as well. As all running processes require memory, it is the operating system's job to ensure each running process has the memory it needs. Also, as all running processes share the same physical memory, it is possible one process could affect the way another process executes by modifying memory belonging to the other process. This would create a huge security issue. Because of this, the operating system must also enforce memory isolation, or in other words must ensure a process can never access memory belonging to a different process.
3. **Provide interfaces for interacting with hardware components and virtualizing them** - CPU and memory virtualization often happens behind the scenes. Something that is beneficial to us as programmers is the fact that operating systems often handle the low-level details of interacting directly with hardware so that we don't have to. This includes components like the network interface card. The operating system provides an interface for sending and receiving data over the network that we can use when developing applications, however the low-level details of how this is actually accomplished are handled by the OS itself.
4. **Providing persistent and long-term data storage (the filesystem) -** The operating system is responsible for ensuring data written to files is stored safely. This may include features like data recovery after a crash or power outage, storing data redundantly to be able to withstand a broken storage device, encrypting data when stored for security, as well as enforcing permissions and access control on individual files. The OS also provides an appropriate interface to applications to easily create, delete, retrieve, and modify files.

In the remainder of the semester, you will be building a simulated filesystem. You will also write a few "commands" or programs that use the interface provided by the file system to create, remove, read, and write files. You will be able to interact with the filesystem by running these commands through a command prompt. As you work, I hope you gain some experience with file system interfaces, but the main goal is to study object-oriented design through implementing commonly used design patterns. Below is an outline of what you will implement in each of the coming studios and lab 5:

- **Studio 16:** A filesystem may hold many different file types (text file, executable file, image files, etc.). In studio 16, you will create an abstract base class that declares the interface shared by all file types and then implement this interface to create a class representing a text file. Topics covered: interface inheritance
- **Studio 17:** In this studio you will implement a second concrete file type, an image file. You will also create an abstract base class declaring the interface of a filesystem and implement this interface in a class representing a simple filesystem implementation. Topics covered: Programming to an interface

- **Studio 18:** In this studio, you will create a new family of classes responsible for creating new files based on the file's extension (".txt" for example). You will implement the abstract factory pattern to accomplish this while allowing your code to be easily extended to support new file types in the future. Topics covered: single responsibility principle, abstract factory pattern
- **Studio 19:** In this studio, you will use the visitor pattern to display the contents of files in unique ways. The "visitors" you create can be used to display different information about files or the files contents in different formats. A new "visitor" can be implemented for each new format a file should be displayed in. Topics covered: visitor pattern, delegation
- **Studio 20:** In this studio, you will add functionality for password protected files. You will use the proxy pattern to accomplish this. Topics covered: Proxy pattern, request forwarding, object composition
- **Studio 21:** In this studio, you will implement the command pattern to create a command prompt that listens for user input and executes a "command" based on the command issued by the user. You will implement the "touch" command, which creates a new file and adds it to the file system. Topics covered: Command pattern
- **Lab 5:** In lab 5, you will implement several more "commands" using a couple of different design patterns. At the end of lab 5, you will be able to interact with your filesystem implementation by interactively executing commands to do things like create files and password protected files, write to files, read from files, copy files, delete files, etc.

Although you will implement a very simple filesystem and the files will be stored in memory only, take a look at the C++ filesystem library to get an idea of how you can interact with the real filesystem in your C++ programs.