Table of Contents

demoOS Developers Guide

Introducing demoOS

Welcome to the developer's guide for demoOS! This comprehensive resource is designed to empower fellow developers seeking to enhance and customize the demoOS operating system. Whether you're looking to extend existing functionalities, modify components, or contribute new features, this guide serves as your roadmap through the inner workings of demoOS.

demoOS is an open-source operating system project developed by Group 5. It provides basic demonstration for understanding fundamental operating system concepts and serves as a learning platform for developers interested in OS development. This guide focuses on assisting developers to actively participate in the evolution of demoOS, shaping it to meet their diverse needs and preferences.

To embark on your journey of extending and modifying demoOS, proceed to the 'Getting Started' section. It outlines the prerequisites, setup instructions, and provides guidance on obtaining the demoOS source code. Feel free to navigate through the guide based on your specific interests and goals.

Getting Started

Prerequisites

Before you start, make sure you have the following prerequisites installed on your machine:

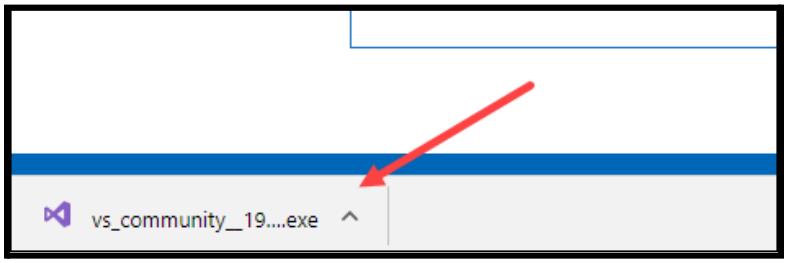
- Visual Studio: Download and install Visual Studio from the official website.
- Visual Studio 2019 Workload: .NET Core Tools: During Visual Studio installation, ensure to include the workload for .NET Core cross-platform development.
- .NET Framework 4.6.2 Developer Pack: Download and install the .NET Framework Developer Pack.
- VMware Player OR Workstation: Download and install VMware Player or Workstation. VMware Player is recommended for its free availability.

Download and Install

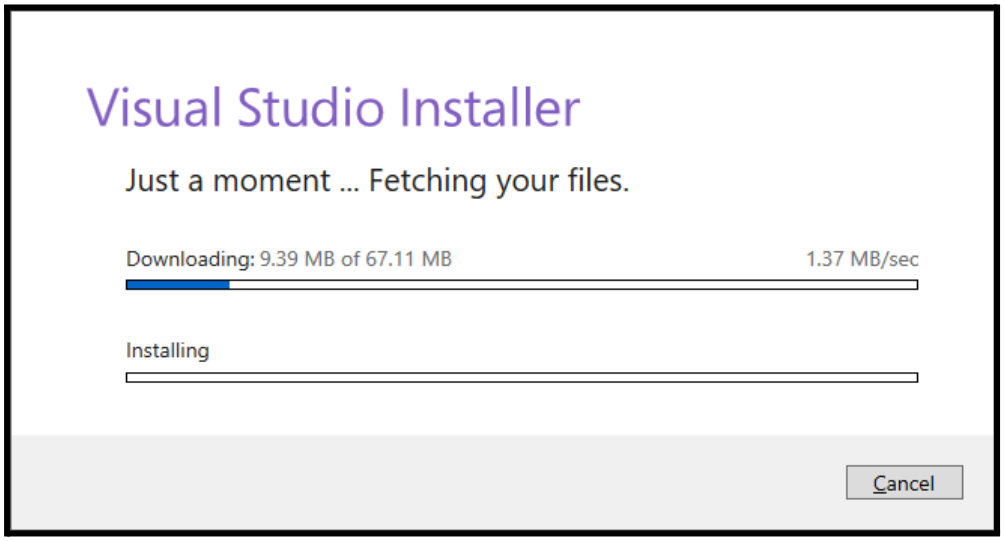
Visual Studio

Visual Studio is the integrated development environment (IDE) where you will write, debug, and compile your code.

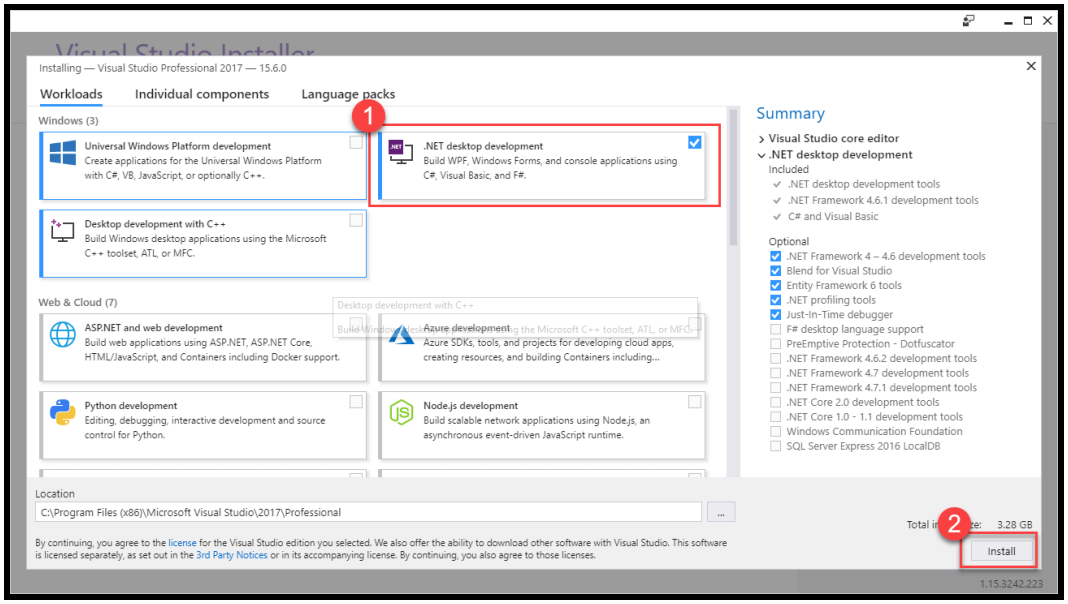
First, visit the following Visual Studio free download link. Among the presented editions, we recommend the Visual Studio Community Edition for free access.



Click on the downloaded exe file. In the next screen, click continue to start Visual Studio installation.



Visual Studio will start downloading the initial files. Download speed will vary as per your internet connection.



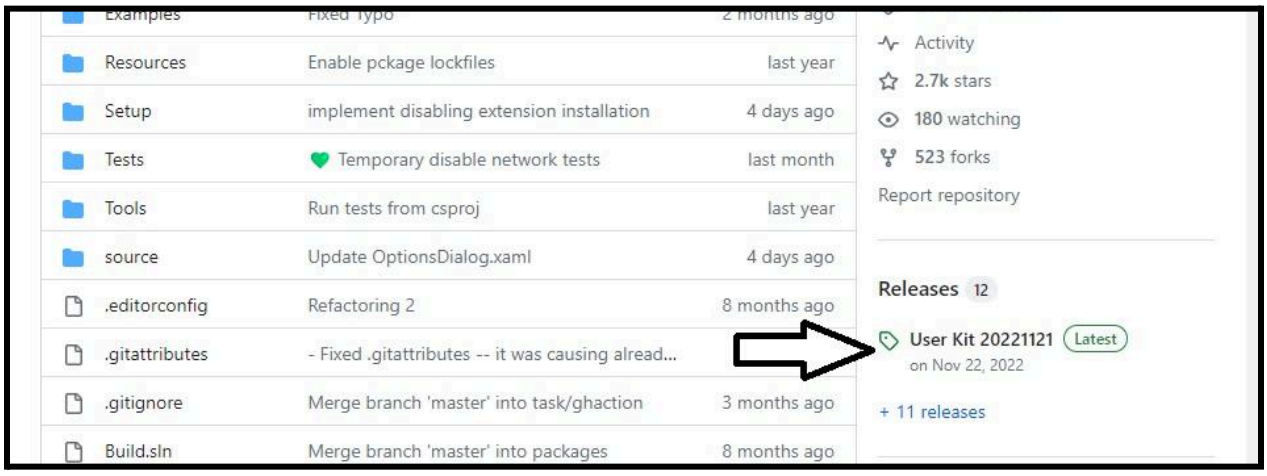
In the next screen, click install. Among the workloads, make sure to check.NET Core cross-platform development.

Visual Studio will download the relevant files for .NET Core cross-platform development. Once the download is done, you will be asked to reboot the PC to complete Visual Studio setup.

This workload includes essential tools for .NET Core development, ensuring compatibility with the cross-platform capabilities required for demoOS.

Latest Release of Cosmos

Download the latest release of Cosmos by downloading the executable (exe) file from Cosmos OS github repository.



Click Releases and scroll down to Assets. From there, download Cosmos User Kit by clicking it.

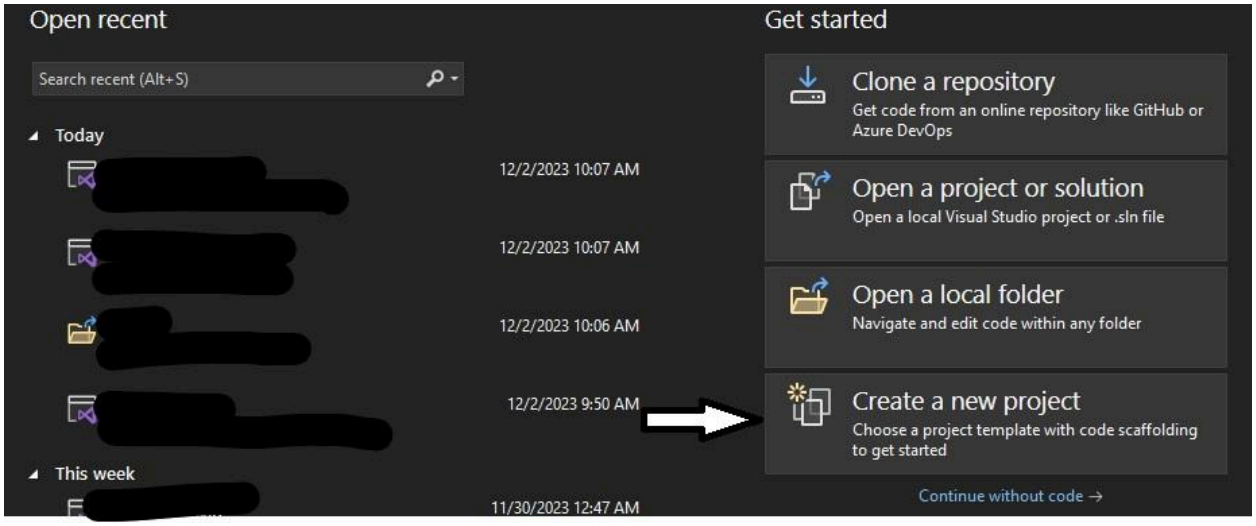
Upon downloading the installer, execute it with administrator privileges. It is crucial to ensure that Visual Studio is not running during the installation process.

Proceed by clicking "Next" and then "Install."

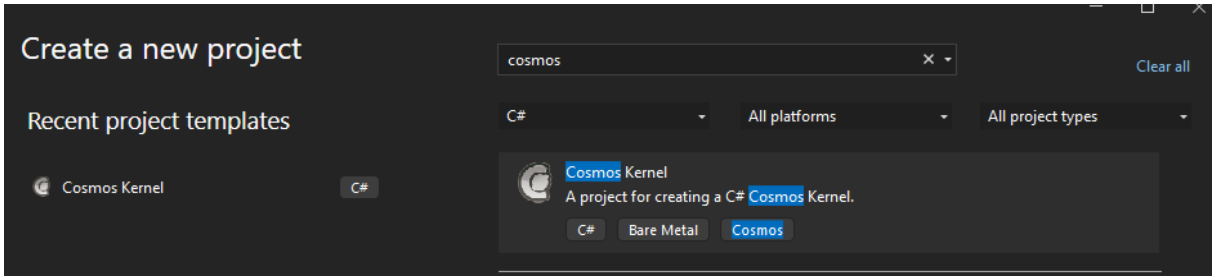
Wait for the "Finish" button to become available, indicating the completion of the installation process.

Verify Installation:

Cosmos should now be installed on your system.



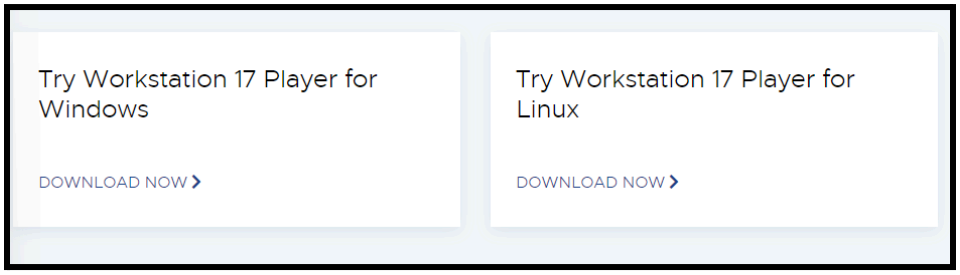
To verify, proceed to Visual Studio and go to “Create a new project.” Search for cosmos and Cosmos Kernel should be available as a project template.



Set up the Virtualization Software.

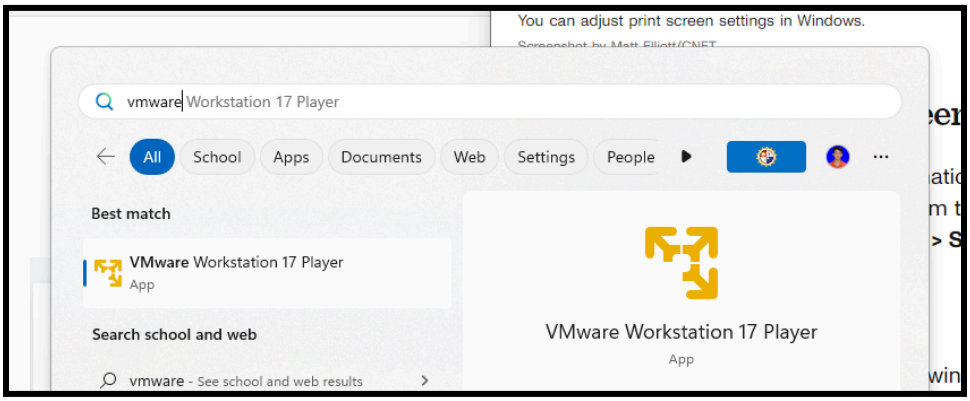
VMware Player or Workstation is the chosen virtualization software for testing demoOS. This allows you to run and evaluate your operating system in a virtual environment. VMware Player is recommended for its free availability, making it accessible to a broader audience.

Visit the official VMware website at VMware Website to download the software. Choose the version of VMware Player or Workstation that is compatible with your operating system (Windows, Linux, or macOS).



Click on the download link and follow the on-screen instructions to download the setup file.

Once the download is complete, run the installer and follow the installation prompts.

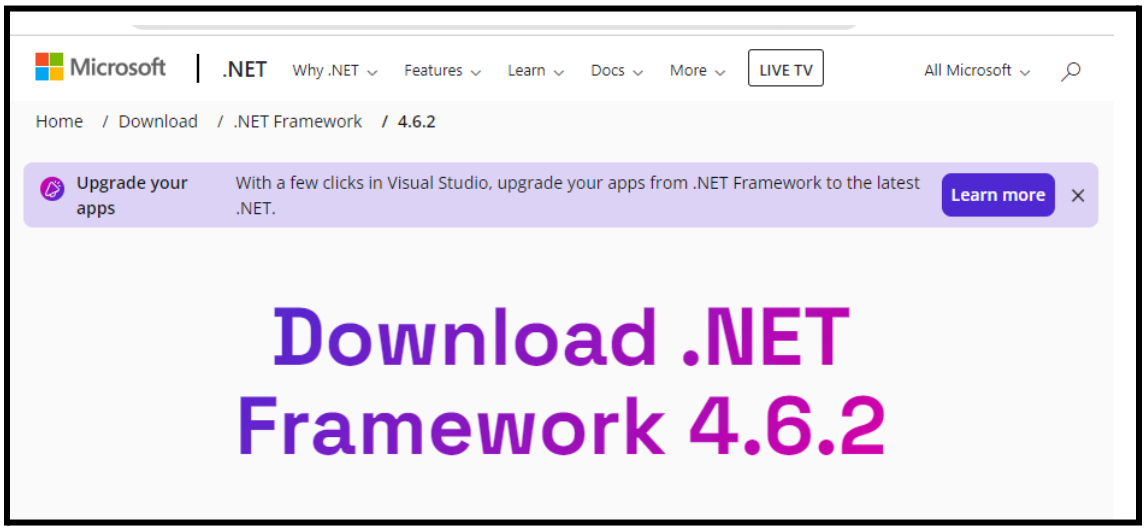


To verify the installation, you can search and check VMWare from your taskbar.

Download .NET Framework 4.6.2

The .NET Framework Developer Pack provides necessary libraries and tools for developing applications, including those for demoOS.

Use the search or navigation bar of your search engine to find the .NET Framework 4.6.2 download page on the Microsoft website.



Look for the download link or button for .NET Framework 4.6.2 on the official page.

Initiate the download of the installer file by clicking on the provided download link. After the download is complete, locate the saved installer file. Double-click on the installer file to initiate the installation process.

Follow each step of the installer, accepting any terms or agreements presented. Allow the installation to complete, ensuring a successful setup on your system.

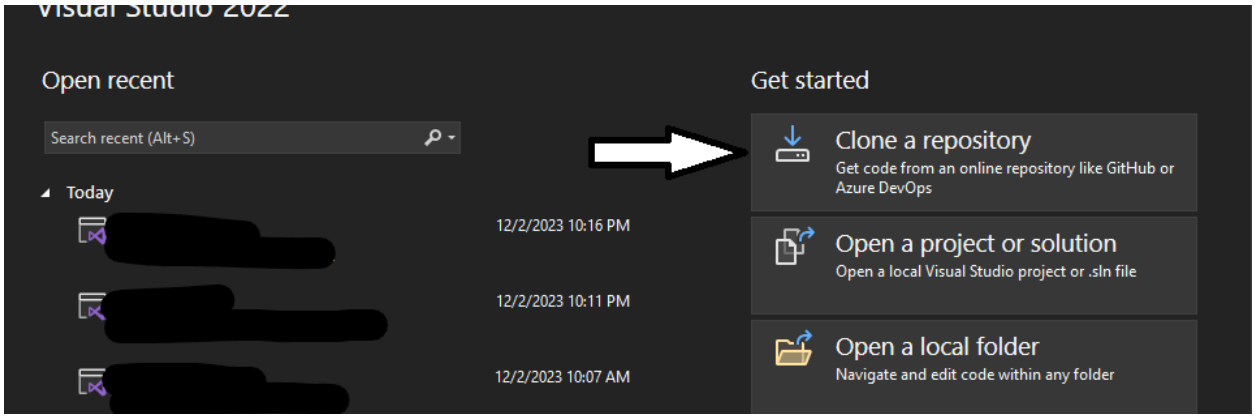
Setting Up the Development Environment

Now that you have installed and set up all the necessary software to begin working on demoOS, let's discuss how you can start getting your hands on our demoOS into your IDE. You can modify demoOS resources using two methods: GitHub cloning and direct file download. The first involves cloning our GitHub repository, while the second entails downloading our resources directly and running them in your Integrated Development Environment (IDE), such as Visual Studio.

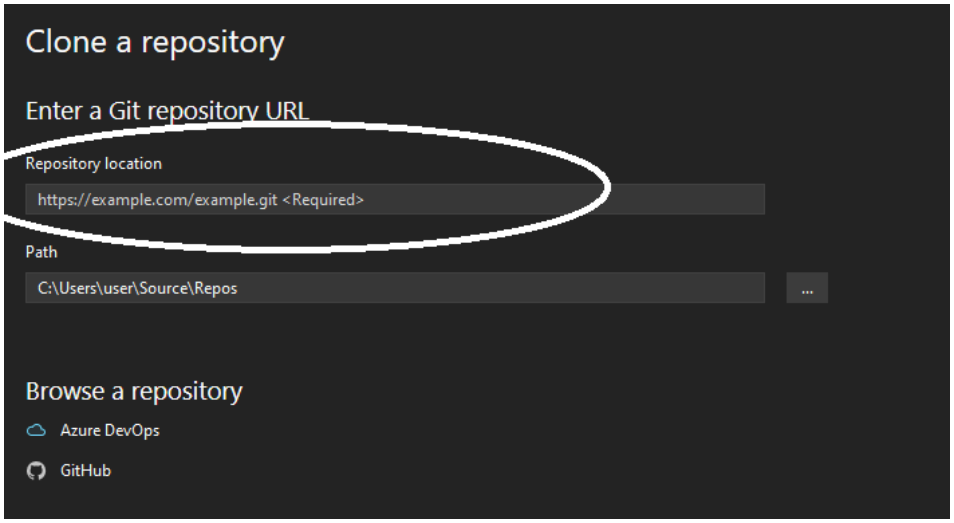
Method #1 GitHub Cloning

Open Visual Studio, and among the options to Get Started. Simply click "Clone a repository."

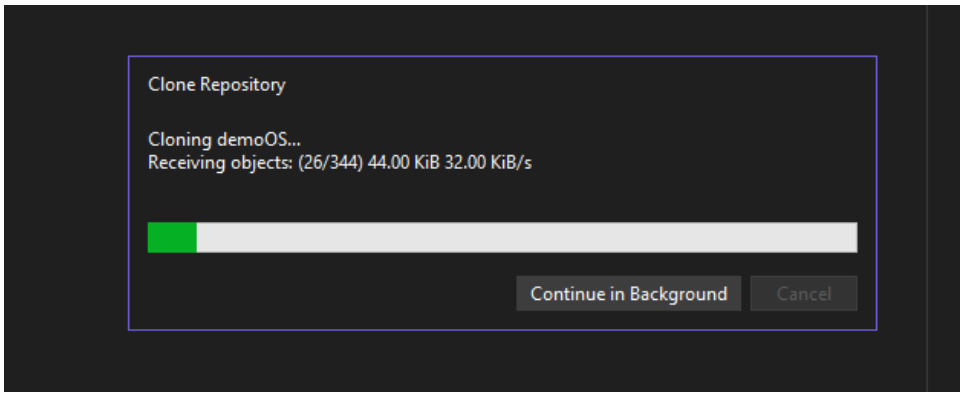
Clone the demoOS repository from https://github.com/honestovicente/demoOS.git



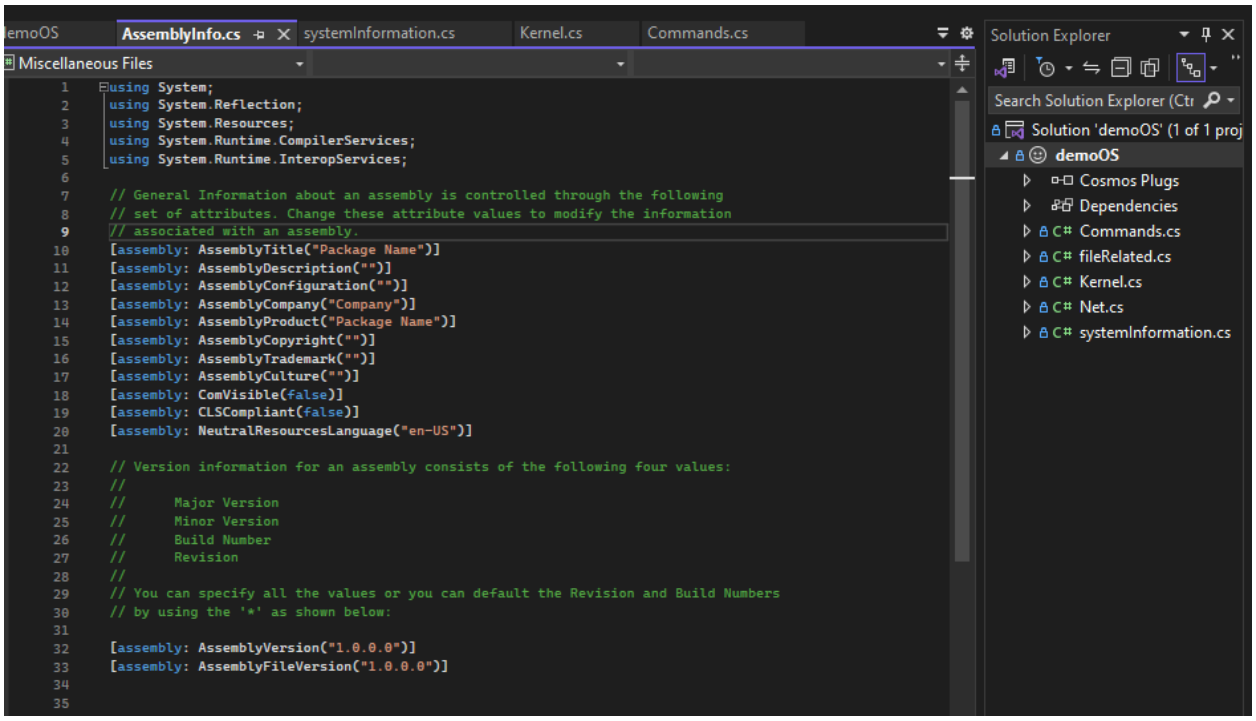
Insert demoOS repository link to the repository location.



After inputting the link and a valid local path, proceed to the Clone button located on the lower right of the window.



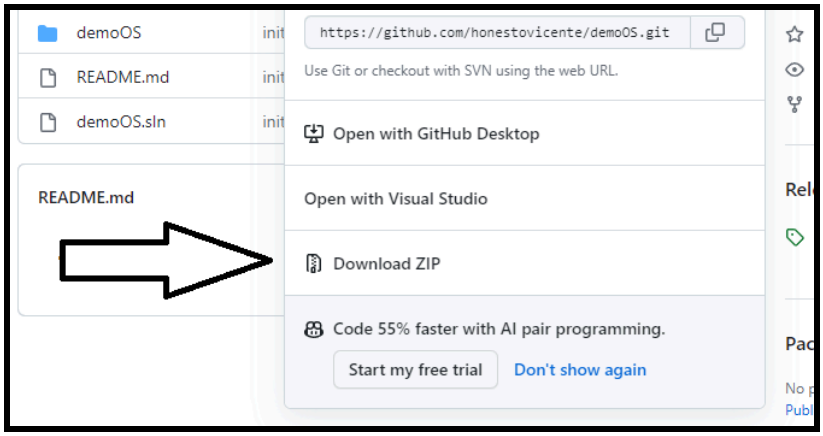
Wait for the cloning process to be complete. This may take some time according to your internet speed.



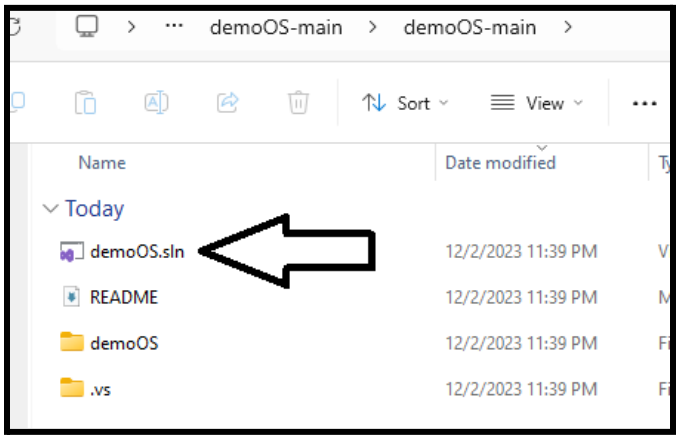
After the process is done, you have successfully set up all the resources of demoOS.

Method #2 Open .sln through Visual Studio

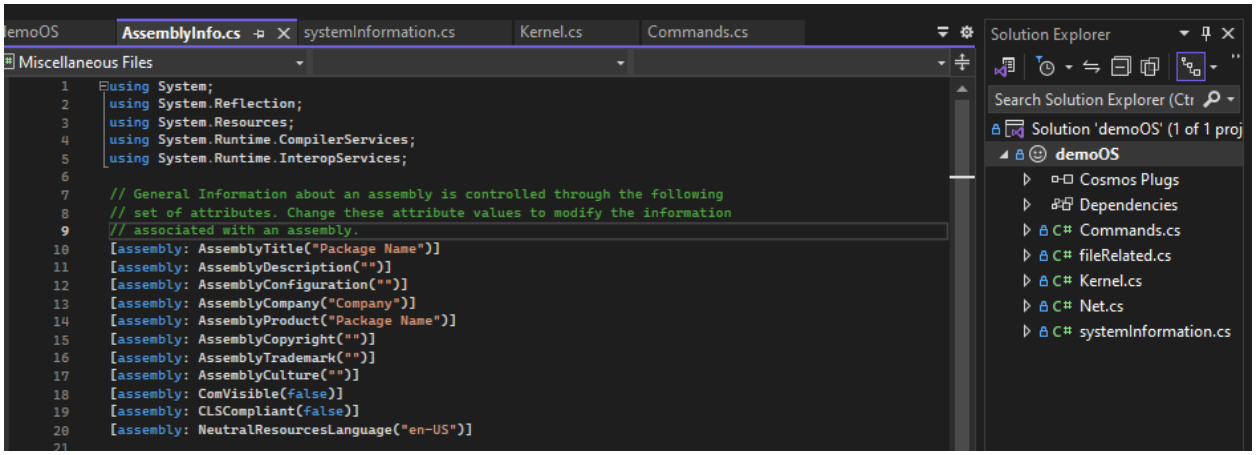
Open the GitHub link of the repository and download the source code as a ZIP.



Extract the contents of ZIP file and this should show up



You can simply click demoOS.sln and it will automatically run Visual Studio, opening all the demoOS resources.



You have now successfully opened the source folder containing all resources.

If Git is not set up on your device, I recommend opting for method #2 to avoid additional steps. However, if you already have Git set up, GitHub cloning is a preferable choice. This method not

only allows you to track modifications but also provides opportunities for easy collaboration with others.

Architecture Overview

Kernel Main Class

The Kernel.cs class serves as the heart of the operating system, containing core functionalities such as system initialization, file system management, and user interface elements.

Key Functions

```
// Method called before the operating system starts running
0 references
protected override void BeforeRun()
{
    initializeFileSys();
    Console.Clear();
    intro();
}
```

BeforeRun Method

- Initialization Before OS Execution:
    - Invoked before the operating system starts running.
    - Initializes the file system using the initializeFileSys() method.
    - Clears the console and displays the introduction screen using the intro() method.

```
// Method called during the operating system runtime
0 references
protected override void Run()
{
    // Display the menu and handle user commands
    Commands.upperTab("Menu Display");
    getSetSys();
    Commands.displayMenu(menu_language_options[selectedMenuLanguage]);
    Commands.Acceptcmd();
}
```

Run Method

- Operating System Runtime:
    - Continuously executed during the operating system runtime in a loop.
    - Calls the getSetSys() method to fetch and set system configurations.
    - Displays the menu and handles user commands using the Commands class.
    - Invokes the Acceptcmd() method to accept and process user input.

Supporting Methods

- `initializeFileSys()` Method:

```
1 reference
public static void initializeFileSys()
{
    // Create a new instance of the Cosmos Virtual File System
    var fs = new Sys.FileSystem.CosmosVFS();

    // Register the file system with the Virtual File System Manager
    Sys.FileSystem.VFS.VFSManager.RegisterVFS(fs);
}
```

- This method initializes the file system for the operating system. It creates a new instance of the Cosmos Virtual File System (CosmosVFS) and registers it with the Virtual File System Manager (VFSManager). This step is crucial for setting up the file system and enabling file-related operations in the operating system.

- `intro()` Method:

```
// Method to display the introduction screen
1 reference
public static void intro()
{
    Console.WriteLine("\n\n");
    Console.WriteLine("                                  #######  #####  ");
    Console.WriteLine("   #####  ###### #    #  ####  #     # #     # ");
    Console.WriteLine("   #    # #      ##  ## #    # #     # #       ");
    Console.WriteLine("   #    # #####  # ## # #    # #     #  #####  ");
    Console.WriteLine("   #    # #      #    # #    # #     #       # ");
    Console.WriteLine("   #    # #      #    # #    # #     # #     # ");
    Console.WriteLine("   #####  ###### #    #  ####  #######  #####  ");
    Console.WriteLine("\n   Developed by Group 5");
    Console.Write("\n   Welcome to demoOS");
    threedotsEffect();
    Thread.Sleep(4000);
}
```

- Displays an introduction screen, providing a visual welcome to users.

- getSetSys() Method:

```
public static void getSetSys()
{
    string[] filePaths = Directory.GetFiles(@"0:\");
    string filePath = $"0:\\set_config.txt";
    string[] system_reference = new string[3];

    // Default values
    if (File.Exists(filePath))
    {
        system_reference = File.ReadAllLines(filePath);
        if (system_reference.Length == 3)
        {
            selectedTimeFormat = Convert.ToInt32(system_reference[0]);
            selectedDateFormat = Convert.ToInt32(system_reference[1]);
            selectedMenuLanguage = Convert.ToInt32(system_reference[2]);
        } else
        {
            Console.WriteLine("Invalid system configuration. Boot up: Default");
            //Default
            selectedTimeFormat = 0;
            selectedDateFormat = 0;
            selectedMenuLanguage = 0;
        }
    }
    else
    {
        //Default
        selectedTimeFormat = 0;
        selectedDateFormat = 0;
        selectedMenuLanguage = 0;
    }
}
```

- Retrieves and sets system configurations, including time format, date format, and menu language from set_config.txt.

Supporting Classes

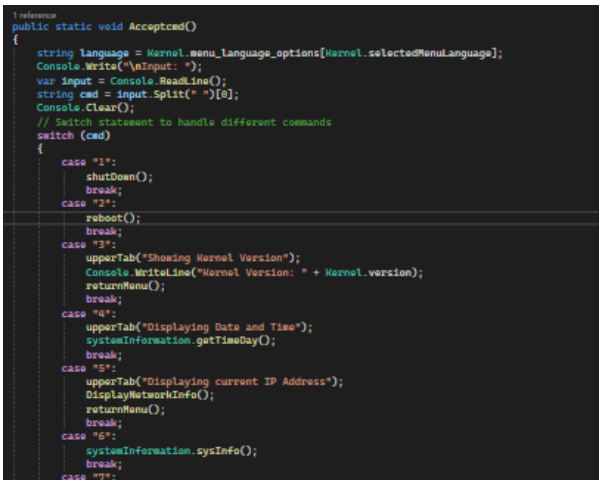
In the repository, alongside the previously discussed Kernel.cs, four additional classes are present, each making distinct and specialized contributions to enhance the functionality of the operating system."

System Commands Class

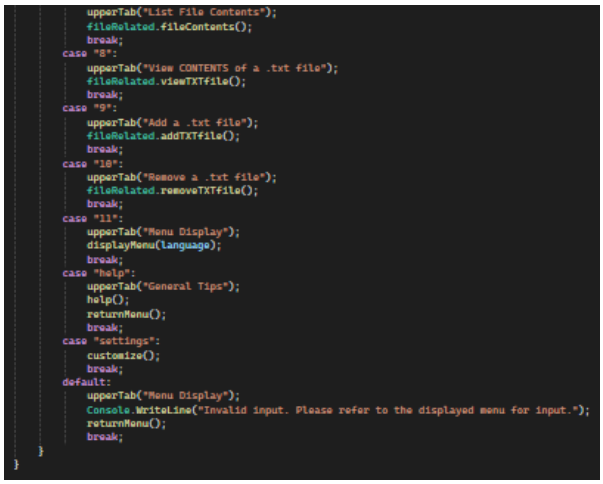
The Commands.cs class serves as the command handler for the operating system. It processes user input, executes corresponding actions, and manages the overall user interface.

Key Functions:

1. **Acceptcmd() Method:**



```
1 reference
public static void Acceptcmd()
{
    string language = Kernel.menu_language_options[Kernel.selectedMenuLanguage];
    Console.Write("\nInput: ");
    var input = Console.ReadLine();
    string cmd = input.Split(" ")[0];
    Console.Clear();
    // Switch statement to handle different commands
    switch (cmd)
    {
        case "1":
            shutDown();
            break;
        case "2":
            reboot();
            break;
        case "3":
            upperTab("Showing Kernel Version");
            Console.WriteLine("Kernel Version: " + Kernel.version);
            returnMenu();
            break;
        case "4":
            upperTab("Displaying Date and Time");
            systemInformation.getTimeDay();
            break;
        case "5":
            upperTab("Displaying current IP Address");
            DisplayNetworkInfo();
            returnMenu();
            break;
        case "6":
            systemInformation.sysInfo();
            break;
        case "7":
```



```
            upperTab("List File Contents");
            fileRelated.fileContents();
            break;
        case "8":
            upperTab("View CONTENTS of a .txt file");
            fileRelated.viewTXTfile();
            break;
        case "9":
            upperTab("Add a .txt file");
            fileRelated.addTXTfile();
            break;
        case "10":
            upperTab("Remove a .txt file");
            fileRelated.removeTXTfile();
            break;
        case "11":
            upperTab("Menu Display");
            displayMenu(language);
            break;
        case "help":
            upperTab("General Tips");
            help();
            returnMenu();
            break;
        case "settings":
            customize();
            break;
        default:
            upperTab("Menu Display");
            Console.WriteLine("Invalid input. Please refer to the displayed menu for input.");
            returnMenu();
            break;
    }
}
```

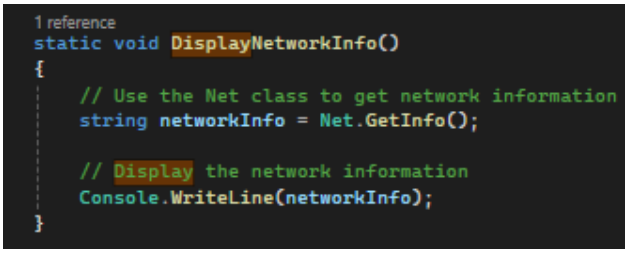
- Processes user input commands
- Directs the system to execute corresponding actions or display relevant information
- Serves as the central command interpreter
- Facilitates user interaction with the operating system
- Interprets and responds to user inputs effectively.

2. **customize() Method:**

```
public static void customize()
{
    string temp;
    string[] choices = new string[3];
    for (int j = 0; j < 3; j++)
    {
        Console.Clear();
        upperTab("Customize OS Experience");
        Console.WriteLine("Type 'cancel' to cancel and return to menu\n");
        switch (j)
        {
            case 0:
                Console.WriteLine("Change TIME Format\n\n OPTIONS: ");
                for (int i = 0; i < Kernel.time_format_options.Length; i++)
                {
                    Console.WriteLine("   > Enter " + i + " for " + Kernel.time_format_options[i] + " Ex: " + systemInformation.currentTime.ToString(Kernel.time_format_options[i]));
                }
                break;
            case 1:
                Console.WriteLine("Change DATE Format\n\n OPTIONS: ");
                for (int i = 0; i < Kernel.date_format_options.Length; i++)
                {
                    Console.WriteLine("   > Enter " + i + " for " + Kernel.date_format_options[i] + " Ex: " + systemInformation.currentDate.ToString(Kernel.date_format_options[i]));
                }
                break;
            case 2:
                Console.WriteLine("Change MENU Language\n\n OPTIONS: ");
                for (int i = 0; i < Kernel.menu_language_options.Length; i++)
                {
                    Console.WriteLine("   > Enter " + i + " for " + Kernel.menu_language_options[i]);
                }
                break;
        }
        Console.Write("\nChoice: ");
        temp = Console.ReadLine();
        if ((temp != "0" && temp != "1") || (temp.ToLower() == "cancel"))
        {
            Console.WriteLine("\nWill return to menu.");
            returnMenu();
            return;
        }
        choices[j] = temp;
    }
```

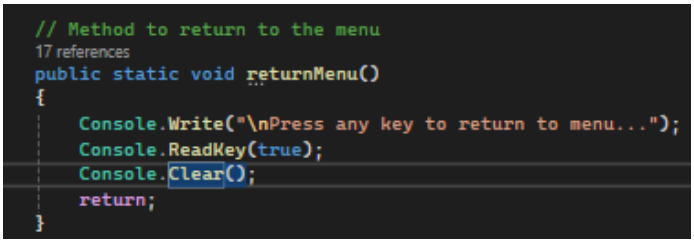
- Guides the user through customization options for time format, date format, and menu language.
- Accepts user choices and saves them to a configuration file (set_config.txt).
- Ensures valid user input and provides options for cancellation.

3. **DisplayNetworkInfo() Method:**



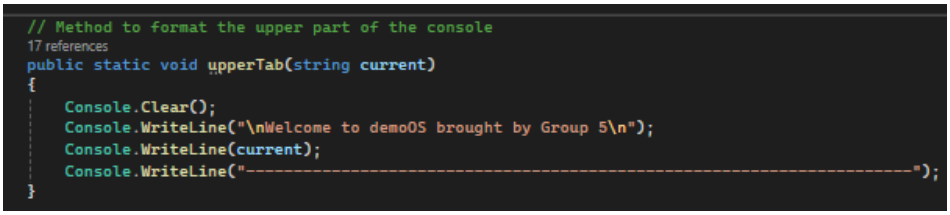
- Retrieves network information using the Net class.
- Displays the current IP address to the user.

4. **returnMenu() Method:**



- Prompts the user to return to the main menu.
- Waits for a key press before clearing the console and returning to the main menu.

5. **upperTab(string current) Method:**



- Formats and displays the upper part of the console with a header.
- Takes the current section as a parameter to customize the header based on the context.

6. **displayMenu(string language) Method:**

```
// Method to display the main menu
2 references
public static void displayMenu(string language)
{
    switch (language) {
        case "english":
            Console.WriteLine("Type 1 (+ enter) to SHUTDOWN");
            Console.WriteLine("Type 2 (+ enter) to REBOOT");
            Console.WriteLine("Type 3 (+ enter) to SHOW kernel version");
            Console.WriteLine("Type 4 (+ enter) to SHOW Current Date and Time");
            Console.WriteLine("Type 5 (+ enter) to DISPLAY IP Address");
            Console.WriteLine("Type 6 (+ enter) to DISPLAY System Info");
            Console.WriteLine("Type 7 (+ enter) to LIST File System Contents");
            Console.WriteLine("Type 8 (+ enter) to VIEW contents of .txt file");
            Console.WriteLine("Type 9 (+ enter) to ADD .txt file");
            Console.WriteLine("Type 10 (+ enter) to REMOVE .txt file");
            Console.WriteLine("Type 11 (+ enter) to REFRESH");
            break;
        case "tagalog":
            Console.WriteLine("Type 1 (+ enter) para ISARA ANG OS");
            Console.WriteLine("Type 2 (+ enter) para I-REBOOT ANG OS");
            Console.WriteLine("Type 3 (+ enter) para IPAKITA ang Bersyon ng Kernel");
            Console.WriteLine("Type 4 (+ enter) para IPAKITA ang Kasalukuyang Petsa at Oras");
            Console.WriteLine("Type 5 (+ enter) para IPAKITA ang IP Address");
            Console.WriteLine("Type 6 (+ enter) para IPAKITA ang Impormasyon ng System");
            Console.WriteLine("Type 7 (+ enter) para IPAKITA ang Nilalaman ng File System");
            Console.WriteLine("Type 8 (+ enter) para TINGNAN ang nilalaman ng .txt file");
            Console.WriteLine("Type 9 (+ enter) para MAGDAGDAG ng .txt file");
            Console.WriteLine("Type 10 (+ enter) para MAGTANGGAL ng .txt file");
            Console.WriteLine("Type 11 (+ enter) para I-REFRESH");
            break;
        default:
            Console.WriteLine("set_config.txt corrupted. Contact dev for assistance.");
            break;
    }
}
```

- Displays the main menu with different options based on the selected language.
- Uses a switch statement to present menu options in English or Tagalog.

7. **shutDown() and reboot() Methods:**

```
// Method to shut down the system
1 reference
public static void shutDown()
{
    upperTab("Confirm shut down");
    Console.Write("\nConfirm to shut down? (y/n): ");
    var choice = Console.ReadLine();
    if (choice == "y")
    {
        Console.Write("\nPreparing to shut down");
        Kernel.threedotsEffect();
        Cosmos.System.Power.Shutdown();
    }
}
```

```
// Method to reboot the system
1 reference
public static void reboot()
{
    upperTab("Confirm reboot");
    Console.Write("\nConfirm to reboot? (y/n): ");
    var choice1 = Console.ReadLine();
    if (choice1 == "y")
    {
        Console.Write("\nPreparing to Reboot");
        Kernel.threedotsEffect();
        Cosmos.System.Power.Reboot();
    }
}
```

- Confirm the user's intention to shut down or reboot the system.
- Initiate the corresponding system operation if confirmed.

8. **help() Method:**

```
1 reference
public static void help()
{
    Console.WriteLine("Welcome to demoOS Help Center!\nHere are some tips to enhance your experience:\n");
    Console.WriteLine(" > Use the provided menu to navigate through various commands.");
    Console.WriteLine(" > Follow the on-screen instructions to provide necessary inputs.");
    Console.WriteLine(" > Take advantage of the 'refresh' command to refresh the display.");
    Console.WriteLine(" > For file naming, use alphanumeric, avoid special characters or spaces.");
    Console.WriteLine(" > For a quick system overview, use '6' command.");
    Console.WriteLine(" > Type 'settings' for customization options to demoOS.");
    Console.WriteLine("\nFor an in-depth reference, please refer to our Manual.");
}
```

- Provides helpful tips and information to enhance the user experience.
- Offers guidance on using different commands and features.
- Prompts the user to return to the main menu after displaying help information.

File Related Operations Class

The fileRelated.cs class handles file-related functionalities in the operating system. It manages the organization, creation, deletion, and viewing of text files within the Cosmos Virtual File System (VFS).

Key Functions:

1. **fileContents() Method:**

```
// Display the contents of the files in the 0:\ directory
1 reference
public static void fileContents()
{
    string[] filePaths = Directory.GetFiles(@"0:\");
    var drive = new DriveInfo("0");
    Console.WriteLine("Directory of " + @"0:\");

    // Display the list of file system contents and their total size
    listFile();
    Console.WriteLine($"\n  Total Size: {drive.TotalSize / (1024 * 1024)} MB");
    Console.WriteLine($"  Free Space: {drive.AvailableFreeSpace / (1024 * 1024)} MB free");
    Commands.returnMenu();
}
```

- Retrieves and displays the contents of the files in the 0:\ directory.
- Lists file system contents and their total size.
- Utilizes the listFile() method to present the file list.

2. **addTXTfile() Method:**

```
string filePath = $"0:\\{name}.txt";
// Check if the file already exists
if (File.Exists(filePath))
{
    Console.WriteLine($"\nText file '{name}.txt' already exists at '0:\\'.");
    Commands.returnMenu();
}
else
{
    // Write the contents to the file
    using (StreamWriter writer = new StreamWriter(filePath))
    {

        writer.WriteLine("DateTimeCreated: " + currentDate.ToString("MM-dd-yyyy") + " @" + currentTime.ToString("HH:mm:ss"));
        writer.WriteLine("-------------------------------------");

        for (int j = 0; j < i - 1; j++)
        {
            writer.WriteLine(linesContents[j]);
        }
    }
    Console.WriteLine($"\nText file '{name}.txt' created successfully at '0:\\'.");
    Commands.returnMenu();
}
```

- Guides the user through creating a new .txt file.
- Accepts user input for the file name and contents.
- Automatically includes the Date and Time it was created.

```
//Filename validation
char[] invalidChars = { '\\', '/', ':', '*', '?', '"', '<', '>', '|', '.' };

foreach (char invalidChar in invalidChars)
{
    if (temp_name.Contains(invalidChar) || temp_name == "" || temp_name == null)
    {
        Console.WriteLine("\nInvalid Filename. Avoid using chararacters (\\ / : * ? \" < > | .)");
        Commands.returnMenu();
        return;
    }
}
// If no invalid characters were found, the filename is considered valid
var name = temp_name;
```

- Validates the filename and ensures valid content input.
- Writes the contents to the file and displays success messages.

3. **removeTXTfile() Method:**

```
// Remove a .txt file from 0:\
1 reference
public static void removeTXTfile()
{
    string[] filePaths = Directory.GetFiles(@"0:\");

    Console.WriteLine("NOTE: Do not include .txt when typing the name of file.");
    Console.WriteLine("      Type 'cancel' to cancel and return to menu.");
    listFile();
    Console.Write("\nName of File to Delete: ");
    var name = Console.ReadLine();
```

```
    var name = Console.ReadLine();

    if (name == "cancel")
    {
        return;
    }
    string filePath = $"0:\\{name}.txt";
    // Check if the file exists
    if (File.Exists(filePath))
    {
        // Delete the file
        File.Delete(filePath);
        Console.WriteLine($"\nText file '{name}.txt' deleted successfully.");
        Commands.returnMenu();
    }
    else
    {
        Console.WriteLine($"\nFile '{name}.txt' not found.");
        Commands.returnMenu();
        return;
    }
}
```

- Guides the user through removing an existing .txt file.
- Accepts user input for the file name to be deleted.
- Validates the filename and deletes the file if it exists.
- Displays success or error messages based on the operation.

4. **viewTXTfile() Method:**

```
// View the contents of a .txt file from 0:\
1 reference
public static void viewTXTfile()
{
    string[] filePaths = Directory.GetFiles(@"0:\");
    Console.WriteLine("NOTE: Refer to list provided below.");
    Console.WriteLine("      Type 'cancel' to return to menu");
    Console.WriteLine("      PS. Do not include .txt in filename");
    listFile();
    Console.Write("\nName of File to Display: ");
    var name = Console.ReadLine();
    if (name == "cancel")
    {
        return;
    }
    string filePath = $"0:\\{name}.txt";
    // Check if the file exists
    if (File.Exists(filePath))
    {
```

```
            // Read and display the contents of the file
            Console.Clear();
            Commands.upperTab("View CONTENTS of a .txt file");

            string[] lines = File.ReadAllLines(filePath);
            Console.WriteLine($"Contents of '{name}.txt':\n");

            foreach (string line in lines)
            {
                Console.WriteLine("  " + line);
            }
            Commands.returnMenu();
        }
        else
        {
            Console.WriteLine($"\nFile '{name}.txt' not found.");
            Commands.returnMenu();
            return;
        }
    }
```

- Guides the user through viewing the contents of an existing .txt file.
- Accepts user input for the file name to be viewed.
- Validates the filename and displays the contents if the file exists.
- Utilizes listFile() to provide a reference list of available files.

5. **listFile() Method:**

```
3 references
public static void listFile()
{
    string[] filePaths = Directory.GetFiles(@"0:\");

    Console.Write("\nList of File System Contents: \n");
    for (int i = 0; i < filePaths.Length; ++i)
    {
        string path = filePaths[i];

        if (System.IO.Path.GetFileName(path) != "set_config.txt")
        {
            Console.WriteLine(" > " + System.IO.Path.GetFileName(path));
        }
    }
}
```

- Lists the contents of the 0:\ directory, excluding the configuration file (set_config.txt).
- Provides a reference list for the user when dealing with file-related operations.

In the overall flow of file-related functionalities, users engage with the methods encapsulated in fileRelated.cs to interact with files. The system seamlessly guides users through the processes of creating, deleting, or viewing text files based on their issued commands. Robust validation mechanisms are integrated to scrutinize filenames and content inputs, ensuring adherence to specified criteria. Users are promptly informed about the outcome of their file-related operations,

receiving feedback on success or failure. Furthermore, the listFile() method serves as a valuable reference, furnishing users with a comprehensive list of available files within the system.

Network Information Class

The Net.cs class manages networking functionalities within the operating system. It handles the initialization of the network device, enables network configurations, and provides methods to retrieve network information.

Key Functions:

1. **Start() Method:**

```
public static void Start()
{
    Kernel.useNetwork = true;
    networkDevice = NetworkDevice.GetDeviceByName("eth0");
    IPConfig.Enable(networkDevice, new Address(192, 168, 0, 1), new Address(255, 255, 255, 0), new Address(192, 168, 1, 254));
    if (NetworkConfiguration.CurrentAddress != null)
    {
        Kernel.useNetwork = true;
    }
}
```

- Initiates network-related functionalities.
- Set up the network device, in this case, named "eth0."
- Enables network configurations using the IPConfig class.
- Checks if a current network address exists and updates the Kernel.useNetwork variable accordingly.

2. **GetInfo() Method:**

```
1 reference
public static string GetInfo()
{
    if (!Kernel.useNetwork) { Start(); }
    return "IP Address: " + NetworkConfiguration.CurrentAddress?.ToString() ?? "Not available";
}
```

- Checks if the network is already initialized; if not, calls the Start() method.
- Retrieves the current IP address using the NetworkConfiguration class.
- Returns the IP address as a string, or indicates that it is not available.

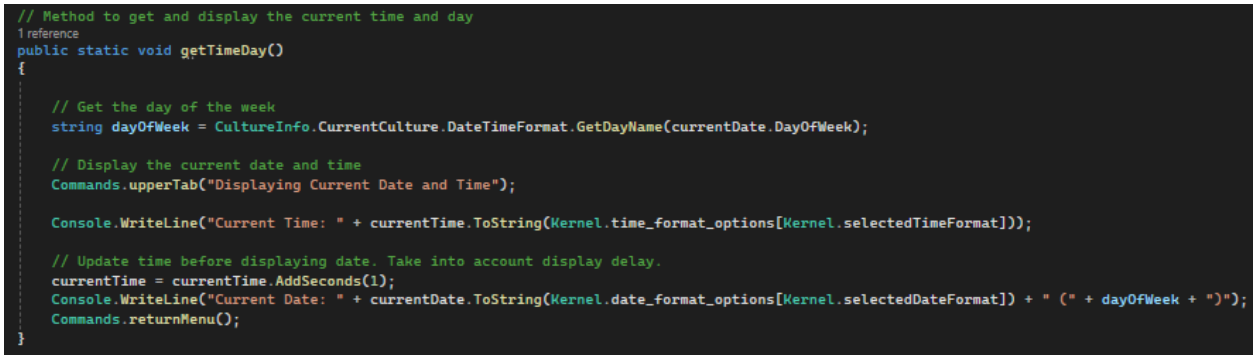
Network Initialization involves the initiation of network-related components through the Start() method, encompassing the initialization of the network device and relevant configurations. The subsequent step involves IP Address Retrieval, accomplished by the GetInfo() method, which retrieves the current IP address. The utility of the Kernel.useNetwork variable plays a pivotal role, dynamically updating based on the success of the network initialization process, reflecting the system's awareness of the network's operational status.

System Information Class

The systemInformation.cs class is responsible for gathering and displaying various system-related information. It covers aspects such as current time, date, RAM usage, CPU information, and general system details.

Key Functions:

1. **getTimeDay() Method:**



```
// Method to get and display the current time and day
1 reference
public static void getTimeDay()
{

    // Get the day of the week
    string dayOfWeek = CultureInfo.CurrentCulture.DateTimeFormat.GetDayName(currentDate.DayOfWeek);

    // Display the current date and time
    Commands.upperTab("Displaying Current Date and Time");

    Console.WriteLine("Current Time: " + currentTime.ToString(Kernel.time_format_options[Kernel.selectedTimeFormat]));

    // Update time before displaying date. Take into account display delay.
    currentTime = currentTime.AddSeconds(1);
    Console.WriteLine("Current Date: " + currentDate.ToString(Kernel.date_format_options[Kernel.selectedDateFormat]) + " (" + dayOfWeek + ")");
    Commands.returnMenu();
}
```

- Retrieves the current time and date.
- Formats and displays the time in the selected format.
- Retrieves the day of the week and displays the current date alongside it.

2. **sysInfo() Method:**

```
// Method to display system information
1 reference
public static void sysInfo()
{
    Kernel kernel = new Kernel();

    // Get information about the file system
    string[] filePaths = Directory.GetFiles(@"0:\");
    var drive = new DriveInfo("0");

    Commands.upperTab("Displaying System Information");

    // Get information about the CPU and RAM
    uint installedRAM = CPU.GetAmountOfRAM();
    uint reservedRAM = installedRAM - (uint)GCImplementation.GetAvailableRAM();
    double usedRAM = (GCImplementation.GetUsedRAM() / (1024.0 * 1024.0)) + reservedRAM;

    string processorBrand = $"Processor Brand: {CPU.GetCPUVendorName()}";
    string processorModel = $"Processor Model: {CPU.GetCPUBrandString()}";
```

- Retrieves information about the file system and drive.
- Retrieves information about the CPU, including brand, model, and uptime.

```
    // Display system information
    Console.WriteLine("SYSTEM INFORMATION\n");
    Console.WriteLine("Operating System: demoOS");
    Console.WriteLine("Developer: Group 5");
    Console.WriteLine("Kernel Version: " + Kernel.version);
    Console.WriteLine("\nRAM: " + usedRAM.ToString("0.00") + "/" + installedRAM.ToString("0.00") + " MB (" + (int)((usedRAM / installedRAM) * 100) + "%)");
    Console.WriteLine("Display Resolution: " + System.Console.WindowWidth + "x" + System.Console.WindowHeight);
    Console.WriteLine(processorBrand);
    Console.WriteLine(processorModel);
    Console.WriteLine(cpuUpTime);
    Console.WriteLine($"\nTotal Size: {drive.TotalSize / (1024 * 1024)} MB");
    Console.WriteLine($"Free Space: {drive.AvailableFreeSpace / (1024 * 1024)} MB free");

    Commands.returnMenu();
}
```

- Presents an overview of the operating system, including the version and developer details.
- Displays RAM usage, display resolution, and drive space details.

The getTimeDay() method facilitates the accurate display of the current time and date, ensuring precision in conveying temporal information. Simultaneously, the sysInfo() method operates as the engine for compiling and presenting comprehensive details about the operating system and its resources. In the operational context of the operating system, these methods are strategically employed during runtime to deliver real-time system information, contributing to an enhanced user experience through timely and pertinent data dissemination.

Extending demoOS

Adding New Commands

Developers can seamlessly extend the functionality of demoOS by strategically organizing the addition of new commands. This process involves constructing the core functionality within its respective class, whether related to files, system information, network, or other categories. Subsequently, developers integrate these features into the user interface by adding them to the displayMenu() method within the Commands class. To ensure a smooth transition back to the main menu after executing the new command, developers conclude the method with a call to returnMenu().

Step-by-Step Guide:

Implement Core Functionality

Choose the appropriate class based on the category of your feature:

- File-related Functionality: Add or modify methods within the fileRelated class in fileRelated.cs.
- System Information Display: Customize the sysInfo() method in the systemInformation class within systemInformation.cs.
- Network-related Functionality: Integrate new functionalities within the Net class in Net.cs.

- Other Categories: Depending on the nature of the feature, select the relevant class and implement the core functionality.
- Feel free to add another class if your specific objective requires you to do so.

Integrate into the User Interface

```
// Existing menu options...

// Add the new command (Example: Type 12 (+ enter) for a new feature)
Console.WriteLine("Type N (+ enter) for a new command");

// Existing menu options...

// Conclude with returning to the main menu
returnMenu();
```

Once the core functionality is defined, integrate it into the user interface by adding the command to the displayMenu() method in the Commands class (Commands.cs).

Ensure Smooth Navigation

```
// Inside the added command method
0 references
public static void NewFeatureCommand()
{
    // Core functionality...

    // Conclude with returning to the main menu
    returnMenu();
}
```

To maintain a user-friendly experience, always conclude the added command with a call to returnMenu(). This ensures a seamless return to the main menu, allowing users to explore other functionalities effortlessly.

This structured approach streamlines the process of adding new commands to demoOS, making it easier for developers to enhance the operating system's capabilities while maintaining a consistent and user-friendly interface.

Customizing the User Interface

Customize the UI in demoOS by updating the displayMenu() and upperTab() methods in the Commands class.

Step-by-Step Guide:

1.) Locate the displayMenu() and upperTab() methods:
Open the Commands.cs file and find the displayMenu() and upperTab() methods within the Commands class.

2.) Update Menu Options:

Modify the content within the displayMenu() method to add, remove, or rearrange menu options. Ensure that any changes align with the overall structure and purpose of demoOS.

Additionally, you can enable user input codes like sysinfo, net, add, listfile, etc instead of the set number assignment for each feature.

3.) Enhance Upper Tab Display:
Adjust the content within the upperTab() method to customize the upper part of the console. This may include adding additional information or refining the existing display.

4.) Testing:
After customization, thoroughly test the user interface to ensure that the modifications do not compromise usability and that the interface remains user-friendly.

Customization Options

Adding Custom Settings

Developers can add new settings to demoOS by modifying the getSetSys() and customize() methods in the Kernel and Commands classes, respectively.

Step-by-Step Guide:

1.) Locate the getSetSys() and customize() methods:
Open the Kernel.cs and Commands.cs files and find the getSetSys() method in the Kernel class and the customize() method in the Commands class.

2.) Update Settings:

```
public static string[] feature_options = new string[] { "Option 1", "Option 2" };
```

```
    // New customizable feature
case 3:
    Console.WriteLine("Customize (feature)\n\n OPTIONS: ");
    for (int i = 0; i < Kernel.date_format_options.Length; i++)
    {
        Console.WriteLine("   > Enter " + i + " for " + Kernel.feature_options[i]);
    }
    break;
```

Modify the content within these methods to add new settings or customize existing ones. Consider the type of settings users may want to adjust.

3.) Ensure Valid Input:
Implement checks within the methods to ensure that the input for new settings is valid. Handle edge cases and provide appropriate feedback.

Debugging and Testing

Debugging and testing are crucial phases in the development process of demoOS to ensure a robust and error-free operating system.

System Testing:

- Conduct comprehensive system tests to evaluate the overall behavior of demoOS. Test the complete system under various conditions, mimicking real-world usage scenarios. System testing helps identify potential issues arising from the interaction of multiple features.

User Experience Testing:

- Enlist testers who assume the role of end-users. Encourage them to use demoOS in various ways, including unconventional and careless interactions. This approach helps identify usability issues, user interface glitches, and unexpected behaviors that might not be apparent during regular testing.

Error Handling Assessment:

- Explicitly test error handling mechanisms. Intentionally trigger errors and observe how demoOS responds. Ensure that error messages are clear, informative, and guide users in resolving issues. Effective error handling is crucial for enhancing user experience.

Frequently Asked Questions (FAQs)

1.) **What is demoOS?**

demoOS is an operating system developed by Group 5 as a part of a project. It provides basic functionalities, including file management, system information display, and network-related features.

2.) **Which is a better option between GitHub Cloning or using Visual Studio to run the .sln file of the OS?**

GitHub Cloning is suitable if you prefer managing the source code through version control and want to contribute to the project. On the other hand, using Visual Studio to run the .sln file provides a more straightforward approach for those who prefer an integrated development environment (IDE). Choose the method that aligns with your development style and objectives.

3.) **Do you have an in-depth tutorial on how to download the required software?**

If you are still having a hard time setting up all the necessary software, we suggest you refer to YouTube tutorials and other helpful resources. For our recommendations, refer to the resources part of this guide.

Resources

https://github.com/honestovicente/demoOS

https://github.com/CosmosOS/Cosmos

Printed Source Code:
https://drive.google.com/file/d/1r9XIRikz8-sejzesg7T_810gyQ8NY77e/view?usp=sharing

References

https://www.youtube.com/watch?v=gEUTe38EP1g

https://www.youtube.com/playlist?list=PLKhwPCsHPWdFFdF2-ZHwX_IVuqU8h6tHf