A Course Based Project Report

On

Web-based OS Simulator & Command Shell

Submitted in partial fulfillment of requirement

for the completion of the

Operating Systems Laboratory

II B.Tech Computer Science and Engineering

of

VNR VJIET

By

B. ESWARI – 23071A05E7
D. NAGALAXMI – 23071A05F2
G. GNANA SRIHASA – 23071A05G0
K. HARSHITHA – 23071A05G3
K. KRISHNA PRAMODITHA – 23071A05G7

2024-2025

VALLURIPALLI NAGESWARA RAO
VIGNANA JYOTHI INSTITUTE OF ENGINEERING &TECHNOLOGY
(AUTONOMOUS INSTITUTE)
NAAC ACCREDITED WITH 'A++' GRADE
NBA Accreditation for B. Tech Programs
Vignana Jyothi Nagar, Bachupally, Nizampet (S.O), Hyderabad 500090
Phone no: 040-23042758/59/60, Fax: 040-23042761
Email:postbox@vnrvjiet.ac.in Website: www.vnrvjiet.ac.in

A Project Report On

Web-based OS Simulator & Command Shell

Submitted in partial fulfillment of requirement
for the completion of the
Operating Systems Laboratory

II B.Tech Computer Science and Engineering

of

VNRVJIET

2024-2025

Under the Guidance

of

DR. D N VASUNDHARA
Assistant Professor
Department of CSE

VNR VIGNANA JYOTHI INSTITUTE OF ENGINEERING & TECHNOLOGY

(AUTONOMUS INSTITUTE)

NAAC ACCREDITED WITH 'A++' GRADE

CERTIFICATE

This is to certify that the project entitled "WEB-BASED OS SIMULATOR & COMMAND SHELL" submitted in partial fulfillment for the course of Operating Systems laboratory being offered for the award of Batch (CSE-C) by VNRVJIET is a result of the bonafide work carried out by **23071A05E7**, **23071A05F2, 23071A05G0, 23071A05G3** and **23071A05G7** during the year **2024-2025**.

This has not been submitted for any other certificate or course.

Signature of Faculty **Signature of Head of the Department**

ACKNOWLEDGEMENT

An endeavor over a long period can be successful only with the advice and support of many well-wishers. We take this opportunity to express our gratitude and appreciation to all of them.

We wish to express our profound gratitude to our honorable **Principal Dr. C.D.Naidu** and **HOD Dr.V.Baby**, **CSE Department**, **VNR Vignana Jyothi Institute of Engineering and Technology** for their constant and dedicated support towards our career moulding and development.

With great pleasure we express our gratitude to the internal guide **DR. DN VASUNDHARA, Assistant Professor, CSE department** for his timely help, constant guidance, cooperation, support and encouragement throughout this project as it has urged us to explore many new things.

Finally, we wish to express my deep sense of gratitude and sincere thanks to our parents, friends and all our well-wishers who have technically and non-technically contributed to the successful completion of this course-based project.

DECLARATION

We hereby declare that this Project Report titled "WEB-BASED OS SIMULATOR & COMMAND SHELL" submitted by us of Computer Science & Engineering in **VNR Vignana Jyothi Institute of Engineering and Technology**, is a bonafide work undertaken by us and it is not submitted for any other certificate /Course or published any time before.

Name & Signature of the Students:

B. ESWARI – 23071A05E7
D. NAGALAXMI – 23071A05F2
G. GNANA SRIHASA – 23071A05G0
K. HARSHITHA – 23071A05G3
K. KRISHNA PRAMODITHA – 23071A05G7

Date:

TABLE OF CONTENTS

ABSTRACT

The Web-based OS Simulator & Command Shell is a browser-based simulation platform that replicates the core functionality of a traditional operating system command-line interface. It provides a secure, user-friendly environment where users can enter and execute predefined system commands, observe their behavior, and understand the underlying concepts of shell-based interactions. Developed with an emphasis on educational value and accessibility, the application is particularly useful for operating systems lab sessions, cybersecurity training, and systems programming tutorials.

This project focuses on emulating essential Windows shell commands such as `dir`, `echo`, `whoami`, `ping`, and `tasklist`. Each command is processed on the server side, where its logic is executed in a controlled context and the output is rendered back to the user in real time. The frontend interface, built using HTML, CSS, and JavaScript (or React for scalable versions), offers a clean input field, output display, and a run button—streamlining the learning experience for beginners who may not be familiar with command-line syntax.

From a systems design perspective, the simulator encapsulates key principles such as client-server communication, sandboxed command execution, and dynamic output rendering. The backend is responsible for parsing input, validating commands, managing access permissions, and ensuring safe execution through subprocess handling (in Python or Node.js), thereby mimicking a real operating system's task execution model without exposing system-level vulnerabilities.

To further enhance functionality, the simulator can be extended to support user sessions, command history tracking, error reporting, and simulated file system navigation. It can also integrate gamified learning modules or quizzes that test a user's understanding of commands and shell navigation. These enhancements position the tool as not only a teaching aid but also a practical entry point for learners progressing into scripting, automation, or DevOps workflows.

The project additionally introduces fundamental operating system concepts such as process management, I/O redirection, and permissions handling—all within a web-accessible interface. Error handling is also incorporated to manage invalid commands, security threats (such as command injection attempts), and backend execution failures. This improves the robustness of the tool and ensures its reliability in classroom or training environments.

By offering a cross-platform, installation-free method of engaging with shell commands, the Web-based OS Simulator & Command Shell demonstrates how modern web technologies can be combined with systems programming to create powerful and safe learning platforms. It bridges the gap between low-level OS operations and high-level web application development, making it ideal for both computer science students and software engineers seeking to understand command-line environments in a sandboxed, controlled format.

INTRODUCTION

The **Web-Based OS Simulator & Command Shell** is an interactive web-based platform that replicates the core functionalities of a traditional command-line interface (CLI) found in operating systems. Its main purpose is to provide students, educators, and developers with a hands-on environment to understand how commands are executed at the system level without accessing or altering the actual operating system. By simulating common shell commands such as `dir`, `echo`, `whoami`, `ping`, and `tasklist`, the tool enables users to explore file systems, display messages, identify users, and list active processes through a familiar interface—all from within a standard web browser. This simulator effectively removes the risks associated with running real system commands by sandboxing interactions and mimicking system responses, making it ideal for educational and training use.

The backend of the simulator is built using **Python** and the **Flask** framework, utilizing powerful modules such as `subprocess`, `os`, and optionally `platform` or `shlex` to safely handle system-level tasks. These modules allow the simulator to parse and execute commands, capture outputs, and handle exceptions or errors in real time. The frontend, designed using HTML, CSS, and JavaScript, provides a smooth and responsive interface where users can input commands and immediately view the output. This seamless client-server architecture enables real-time interaction and response generation, which mirrors the behavior of a local terminal. The use of Python not only ensures safety and modularity but also makes the system extensible, enabling future enhancements with ease.

From an educational standpoint, this project is invaluable for teaching **core operating system concepts** such as process management, command parsing, standard I/O handling, and system call invocation. It introduces users to the mechanics behind command execution, offering insights into how operating systems interpret user input, manage resources, and return results. The simulator also promotes understanding of important topics like filesystem navigation, user session management, and basic network diagnostics, all through practical engagement. Its controlled and user-friendly environment helps learners grasp complex topics like shell environments, scripting, and even early cybersecurity practices, without requiring admin rights or risking system integrity.

Furthermore, the project's **modular architecture** makes it highly extensible and adaptable to different learning or diagnostic scenarios. It opens possibilities for future development, such as adding file manipulation commands, user authentication, script execution support, or even integration with monitoring and logging tools. Features like real-time session tracking, reporting tools, and collaborative access can turn this into a powerful classroom or lab resource. Ultimately, the Web-Based OS Simulator & Command Shell demonstrates how high-level programming can be used to emulate low-level system behavior, bridging the gap between abstract theory and practical experience. It serves not only as a tool for learning but also as a scalable foundation for building more advanced system simulation and diagnostic tools.

METHODOLOGY

The development of the Web-Based OS Shell Simulator followed a modular, full-stack approach. It focused on creating a user-friendly web interface that could execute OS-level commands on a server backend using Python. The following stages describe the detailed methodology used for the successful development of this project:

1. Requirement Gathering and Objective Definition

- Defined core functionality: run shell commands from a web interface.
- Identified the need for cross-platform compatibility (Windows/Linux).
- Determined use cases: educational tool, remote command execution, OS simulations.

2. Technology Stack Selection

- **Frontend**: HTML, CSS, and JavaScript for interactive UI.
- **Backend**: Python with Flask for routing and handling command execution.
- **Command Execution**: Python's `subprocess` module for running system-level commands safely.

3. Frontend Design and Development

- Created an intuitive and responsive HTML interface with:
  - A tab-based system to select between Windows and Linux commands.
  - A command input field and "Run" button for execution.
  - A dynamically updating list of sample commands per OS.
  - Output display area styled with CSS for readability.
- Implemented JavaScript functions to:
  - Fetch and display available commands.
  - Toggle between OS tabs and show/hide command lists.
  - Send command input to the server asynchronously via `fetch()`.

4. Backend Development (Flask API)

- Developed Flask routes to:
  - Serve the HTML page (`/`).
  - Send available commands as JSON (`/commands`).
  - Accept and run shell commands (`/run`).
- Used `subprocess.run()` to execute the commands safely.
  - Ensured proper shell usage based on OS: `cmd.exe /c` for Windows, `bash -c` for Linux.
  - Captured `stdout` and `stderr` for display on the frontend.
  - Implemented timeout to prevent long-running or hanging processes.
  -

5. OS Simulation Module Integration

- Added backend Python modules simulating core OS concepts:
  - **CPU Scheduling (FCFS)**: Simulated process execution order.
  - **Banker's Algorithm**: Demonstrated deadlock avoidance.
  - **Paging (FIFO Replacement)**: Visualized page faults and memory usage.
  - **Disk Scheduling (FCFS)**: Simulated disk head movement and seek operations.
- Created a `main_menu()` driven CLI for terminal-based interaction with these simulations.
- Provided user inputs through prompts and printed outputs for educational use.

6. Security and Error Handling

- Validated command input to prevent injection attacks and ensure safe execution.
- Handled exceptions in backend logic with informative error messages.
- Used input restrictions on the frontend to limit potentially dangerous commands (e.g., `rm`, `shutdown`).

7. Testing and Debugging

- Verified functionality across Windows, Linux, and macOS environments.
- Tested command execution, output correctness, and OS simulations.
- Debugged asynchronous frontend-backend communication.
- Ensured UI responsiveness and error handling robustness.

8. Optimization and Enhancements

- Cached command lists to reduce redundant server calls.
- Ensured subprocess calls were resource-efficient with timeouts and output truncation.
- Designed frontend to be extensible — more OS tabs or features (e.g., command history) can be added.

9. Deployment and Use

- Hosted the Flask app on a local server using `app.run(debug=True)` for development.
- Prepared for deployment on platforms like Heroku or PythonAnywhere for public access.
- Provided instructions for users to interact with the simulator and contribute.

This methodology ensured a robust, educational, and practical tool that bridges web development and operating system simulation. The modular design allows easy expansion with new OS algorithms, command filters, or remote execution features, making it ideal for students, developers, and educators.

OBJECTIVES

The primary goal of the Web-based OS Simulator & Command Shell project is to create an interactive, user-friendly web platform that simulates core operating system functionalities, particularly focusing on command shell interactions. This system aims to provide a practical learning and testing environment for users to understand OS concepts through direct experience. The specific objectives are:

Accurate OS Command Simulation:
To implement a command shell environment within a web interface that accurately simulates essential OS commands and their behavior. This includes commands for file management, process control, directory navigation, and system information retrieval.

Interactive User Interface:
To design an intuitive and responsive web-based interface that mimics the look and feel of a real command-line shell. This allows users to enter commands, view outputs, and receive immediate feedback, thereby enhancing learning and experimentation.

Comprehensive OS Functionality Coverage:
To cover a wide range of OS functionalities such as file and directory operations (create, delete, move, copy), process management commands (list, kill, background tasks), and environment management (variables, paths) within the simulator.

Modular and Extensible Architecture:
To develop the system using modular design principles, separating core components such as command parsing, execution engine, file system simulation, and UI rendering. This modularity supports easier maintenance, debugging, and future feature additions.

Real-Time Command Processing:
To ensure that commands entered by users are processed and executed in real-time, providing a smooth and realistic simulation experience that mirrors actual OS shell performance.

Educational Reporting and Feedback:
To provide users with informative feedback on command outcomes, including error messages and system status updates. This helps users learn correct command usage and understand system responses.

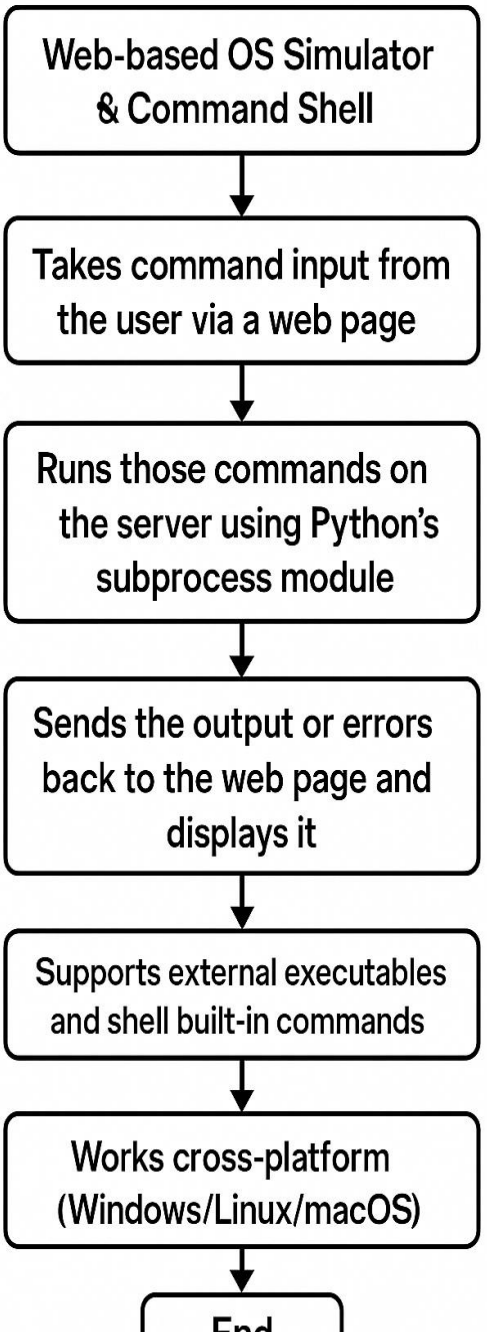
Cross-Platform Accessibility:
To implement the simulator as a web application, making it accessible from any modern web browser without installation, thus enabling widespread usage for students, educators, and developers interested in operating systems.

Future Integration and Feature Expansion:

To design the simulator with scalability in mind, allowing future integration with additional OS modules such as network commands, scripting support, or multi-user simulation, as well as the possibility of connecting to backend services for enhanced functionalities.

Performance and Security Optimization:
To ensure that the web application runs efficiently with minimal latency and is designed with security best practices to prevent misuse or harmful command execution in the simulated environment.

FLOW OF EXECUTION



IMPLEMENTATION OF PROGRAM

Code:

os.py:

```
def fcfs_cpu_scheduling(processes, burst_times):
    n = len(processes)
    waiting_time = [0]*n
    turnaround_time = [0]*n

    for i in range(1, n):
        waiting_time[i] = burst_times[i-1] + waiting_time[i-1]
    for i in range(n):
        turnaround_time[i] = burst_times[i] + waiting_time[i]

    print("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time")
    for i in range(n):
        print(f"{processes[i]}\t{burst_times[i]}\t\t{waiting_time[i]}\t\t{turnaround_time[i]}")

def bankers_algorithm(available, max_demand, allocation):
    n = len(allocation)    # number of processes
    m = len(available)     # number of resources

    need = [[max_demand[i][j] - allocation[i][j] for j in range(m)] for i in range(n)]

    finish = [False]*n
    safe_sequence = []

    work = available[:]

    while len(safe_sequence) < n:
        allocated_in_this_loop = False
        for i in range(n):
            if not finish[i] and all(need[i][j] <= work[j] for j in range(m)):
                for j in range(m):
                    work[j] += allocation[i][j]
                safe_sequence.append(i)
                finish[i] = True
                allocated_in_this_loop = True
        if not allocated_in_this_loop:
            break

    if len(safe_sequence) == n:
        print("System is in a safe state.")
        print("Safe sequence:", ' -> '.join([f"P{p}" for p in safe_sequence]))
    else:
        print("System is NOT in a safe state.")
```

```
def fifo_page_replacement(pages, frame_size):
    frames = []
    page_faults = 0
    for page in pages:
        if page not in frames:
            if len(frames) < frame_size:
                frames.append(page)
            else:
                frames.pop(0)
                frames.append(page)
            page_faults += 1
        print(f"Frames: {frames}")
    print(f"Total page faults: {page_faults}")

def fcfs_disk_scheduling(requests, head):
    seek_sequence = []
    seek_count = 0
    distance = 0
    current_track = head

    for track in requests:
        seek_sequence.append(track)
        distance = abs(track - current_track)
        seek_count += distance
        current_track = track

    print("Seek Sequence:", seek_sequence)
    print("Total seek operations:", seek_count)

def cpu_scheduling_menu():
    print("\nCPU Scheduling - FCFS")
    n = int(input("Enter number of processes: "))
    processes = [f"P{i+1}" for i in range(n)]
    burst_times = list(map(int, input(f"Enter burst times for {n} processes (space separated):
").split()))
    fcfs_cpu_scheduling(processes, burst_times)

def bankers_menu():
    print("\nBanker's Algorithm - Deadlock Avoidance")
    n = int(input("Enter number of processes: "))
    m = int(input("Enter number of resource types: "))
    print("Enter Available Resources (space separated): ")
    available = list(map(int, input().split()))
    print("Enter Max demand matrix:")
    max_demand = [list(map(int, input().split())) for _ in range(n)]
    print("Enter Allocation matrix:")
    allocation = [list(map(int, input().split())) for _ in range(n)]
    bankers_algorithm(available, max_demand, allocation)

def paging_menu():
    print("\nPaging - FIFO Page Replacement")
    pages = list(map(int, input("Enter page reference string (space separated): ").split()))
```

```
    frame_size = int(input("Enter number of frames: "))
    fifo_page_replacement(pages, frame_size)

def disk_scheduling_menu():
    print("\nDisk Scheduling - FCFS")
    head = int(input("Enter initial head position: "))
    requests = list(map(int, input("Enter disk queue requests (space separated): ").split()))
    fcfs_disk_scheduling(requests, head)

def main_menu():
    while True:
        print("\nOS Simulation Menu")
        print("1. CPU Scheduling (FCFS)")
        print("2. Banker's Algorithm (Deadlock Avoidance)")
        print("3. Paging (FIFO Page Replacement)")
        print("4. Disk Scheduling (FCFS)")
        print("0. Exit")
        choice = input("Enter choice: ")

        if choice == '1':
            cpu_scheduling_menu()
        elif choice == '2':
            bankers_menu()
        elif choice == '3':
            paging_menu()
        elif choice == '4':
            disk_scheduling_menu()
        elif choice == '0':
            print("Exiting...")
            break
        else:
            print("Invalid choice. Try again.")

if __name__ == "__main__":
   main_menu()
```

web_shell.py:

```
from flask import Flask, request, jsonify, Response, render_template
import subprocess

app = Flask(__name__)

os_commands = {
    "Windows": [
        "dir", "echo Hello World", "whoami", "ping 127.0.0.1 -n 2", "tasklist", "type README.txt",
"cls"
    ],
    "Linux": [
        "ls", "echo Hello World", "whoami", "ping -c 2 127.0.0.1", "ps aux", "cat README.txt",
"clear"
    ]
}

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/commands')
def get_commands():
    return jsonify(os_commands)

@app.route('/run', methods=['POST'])
def run():
    data = request.json
    cmd = data.get('command', '')
    os_type = data.get('os', 'Windows')

    if not cmd:
        return jsonify({'stderr': 'No command provided.'})

    try:
        if os_type == "Windows":
            shell_cmd = ['cmd.exe', '/c', cmd]
        else:
            shell_cmd = ['bash', '-c', cmd]

        result = subprocess.run(shell_cmd, capture_output=True, text=True, timeout=10)
        return jsonify({
            'stdout': result.stdout.strip(),
            'stderr': result.stderr.strip()
        })
    except Exception as e:
        return jsonify({'stderr': str(e)})

if __name__ == '__main__':
    app.run(debug=Tru
```

index.html:

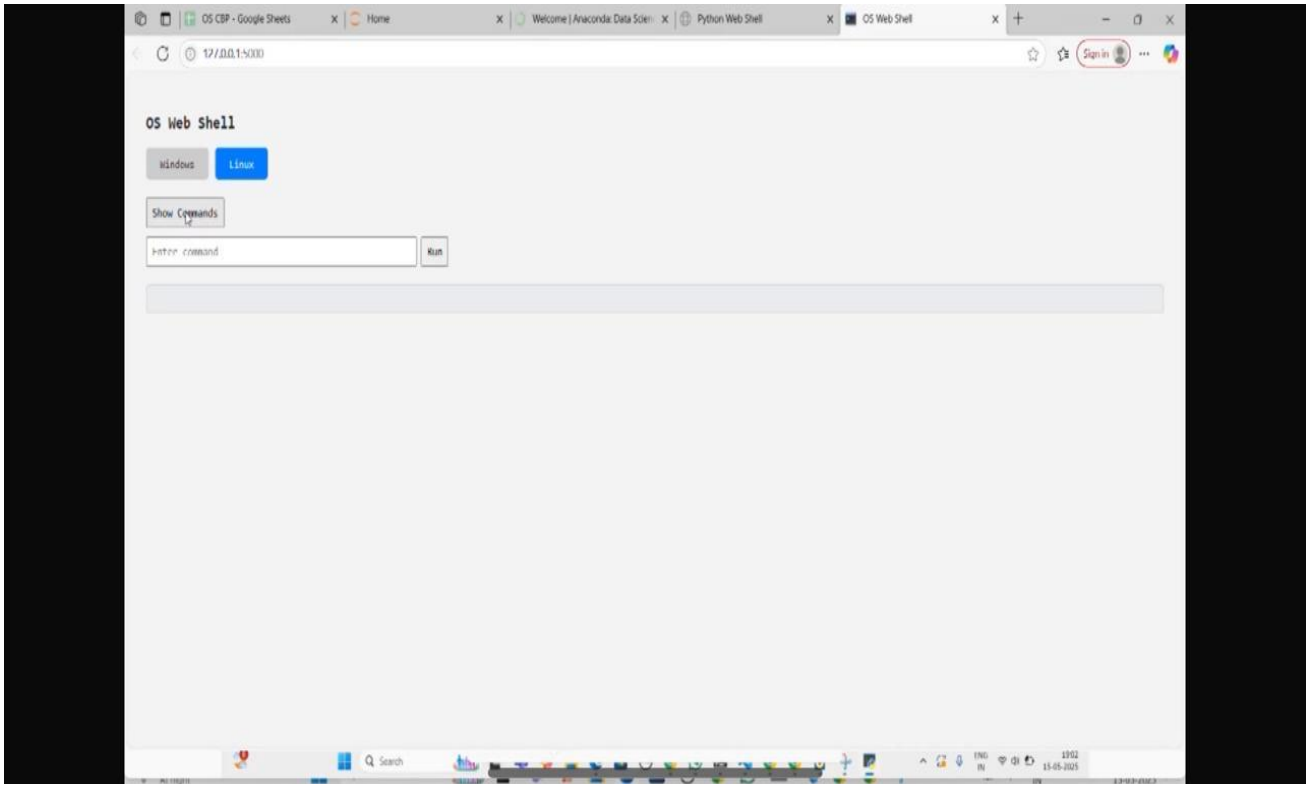
```
<!DOCTYPE html>

<html>
<head>
  <title>OS Web Shell</title>
  <style>
    body {
      font-family: monospace;
      background: #f4f4f4;
      color: #222;
      padding: 20px;
    }
    select, input, button {
      font-family: monospace;
      margin: 5px 0;
      padding: 7px;
    }
    .os-tabs {
      display: flex;
      gap: 10px;
      margin-bottom: 15px;
    }
    .os-tab {
      padding: 10px 20px;
      background: #ccc;
      cursor: pointer;
      border-radius: 5px;
    }
    .os-tab.active {
      background: #007BFF;
      color: white;
    }
    #commands {
      display: none;
      margin-top: 10px;
    }
    #commands div {
      background: #eee;
```

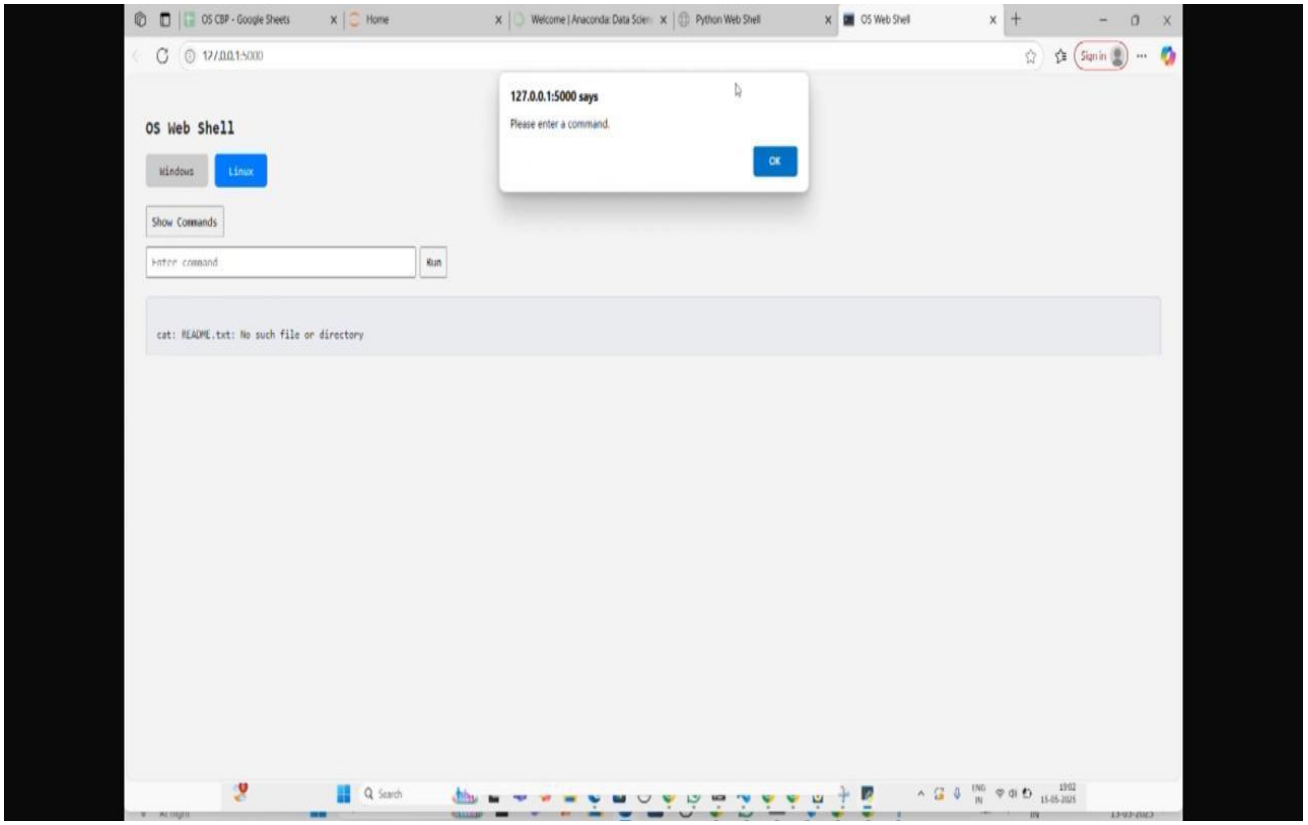
```
    margin: 3px 0;
    padding: 5px 8px;
    cursor: pointer;
    border-radius: 4px;
   }
   #commands div:hover {
    background-color: #cbd3da;
   }
   #output {
    background: #e9ecef;
    padding: 15px;
    white-space: pre-wrap;
    max-height: 300px;
    overflow-y: auto;
    border: 1px solid #ccc;
    border-radius: 4px;
    margin-top: 15px;
   }
  </style>
</head>
<body>
  <h2>OS Web Shell</h2>

  <div class="os-tabs">
   <div class="os-tab active" onclick="selectOS('Windows', event)">Windows</div>
   <div class="os-tab" onclick="selectOS('Linux', event)">Linux</div>
  </div>

  <button onclick="toggleCommands()" id="toggleButton">Show Commands</button>

  <div id="commands-container">
   <strong id="cmdHeader" style="display:none;">Available commands:</strong>
   <div id="commands"></div>
  </div>

  <input type="text" id="command" placeholder="Enter command" size="50">
  <button onclick="sendCommand()">Run</button>

  <pre id="output"></pre>
```

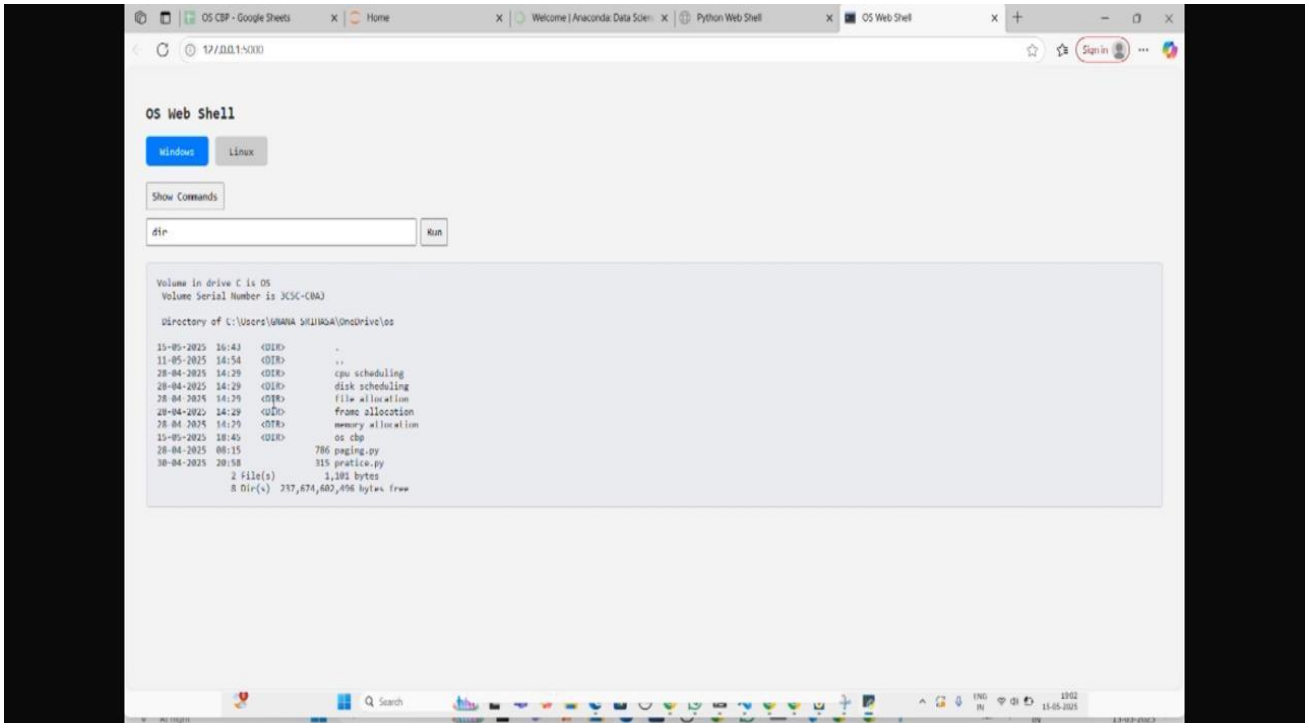
```
<script>
  let currentOS = 'Windows';
  let osCommandMap = {};
  let isCommandsVisible = false;

  async function fetchCommands() {
    const res = await fetch('/commands');
    osCommandMap = await res.json();
    updateCommandList();
  }

  function selectOS(osName, event) {
    currentOS = osName;
    document.querySelectorAll('.os-tab').forEach(tab => tab.classList.remove('active'));
    event.target.classList.add('active');
    updateCommandList();
  }

  function toggleCommands() {
    isCommandsVisible = !isCommandsVisible;
    document.getElementById('commands').style.display = isCommandsVisible ? 'block' : 'none';
    document.getElementById('cmdHeader').style.display = isCommandsVisible ? 'block' : 'none';
    document.getElementById('toggleButton').textContent = isCommandsVisible ? 'Hide
Commands' : 'Show Commands';
  }

  function updateCommandList() {
    const cmdDiv = document.getElementById('commands');
    cmdDiv.innerHTML = '';
    if (osCommandMap[currentOS]) {
      osCommandMap[currentOS].forEach(cmd => {
        const cmdEl = document.createElement('div');
        cmdEl.textContent = cmd;
        cmdEl.onclick = () => document.getElementById('command').value = cmd;
        cmdDiv.appendChild(cmdEl);
      });
    }
  }

  async function sendCommand() {
```

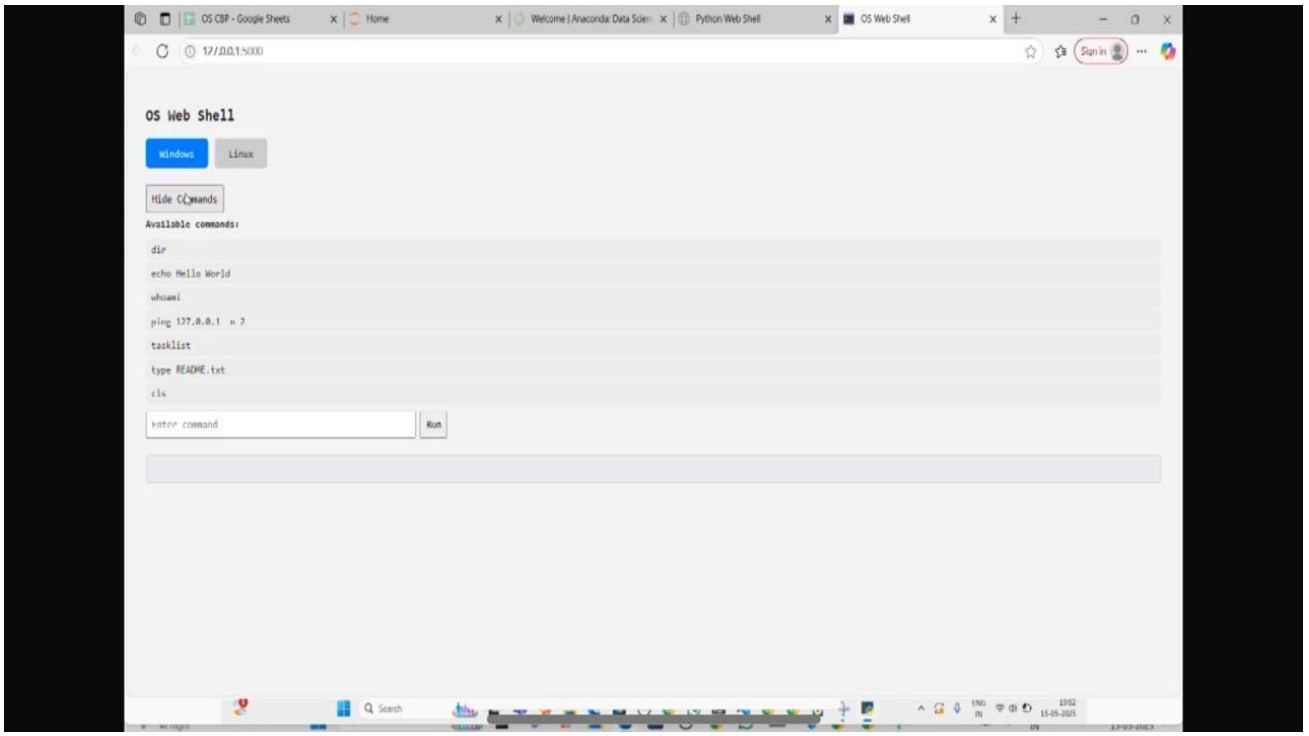
```
    const cmd = document.getElementById('command').value.trim();
    if (!cmd) return alert('Please enter a command.');
    const res = await fetch('/run', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ command: cmd, os: currentOS })
    });
    const data = await res.json();
    let output = '';
    if (data.stdout) output += data.stdout;
    if (data.stderr) output += '\n' + data.stderr;
    document.getElementById('output').textContent = output;
  }

  fetchCommands();

  document.getElementById('command').addEventListener('keydown', e => {
    if (e.key === 'Enter') sendCommand();
  });
 </script>
</body>
</html>
```

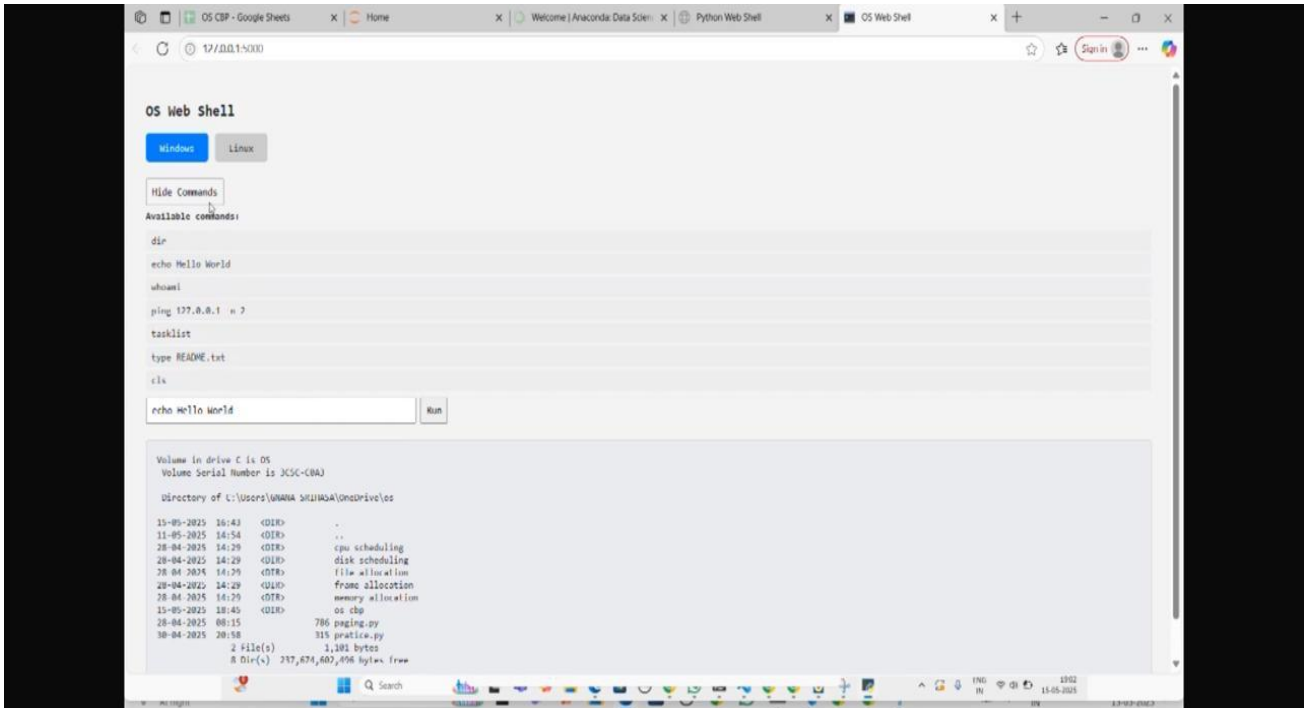
Output:

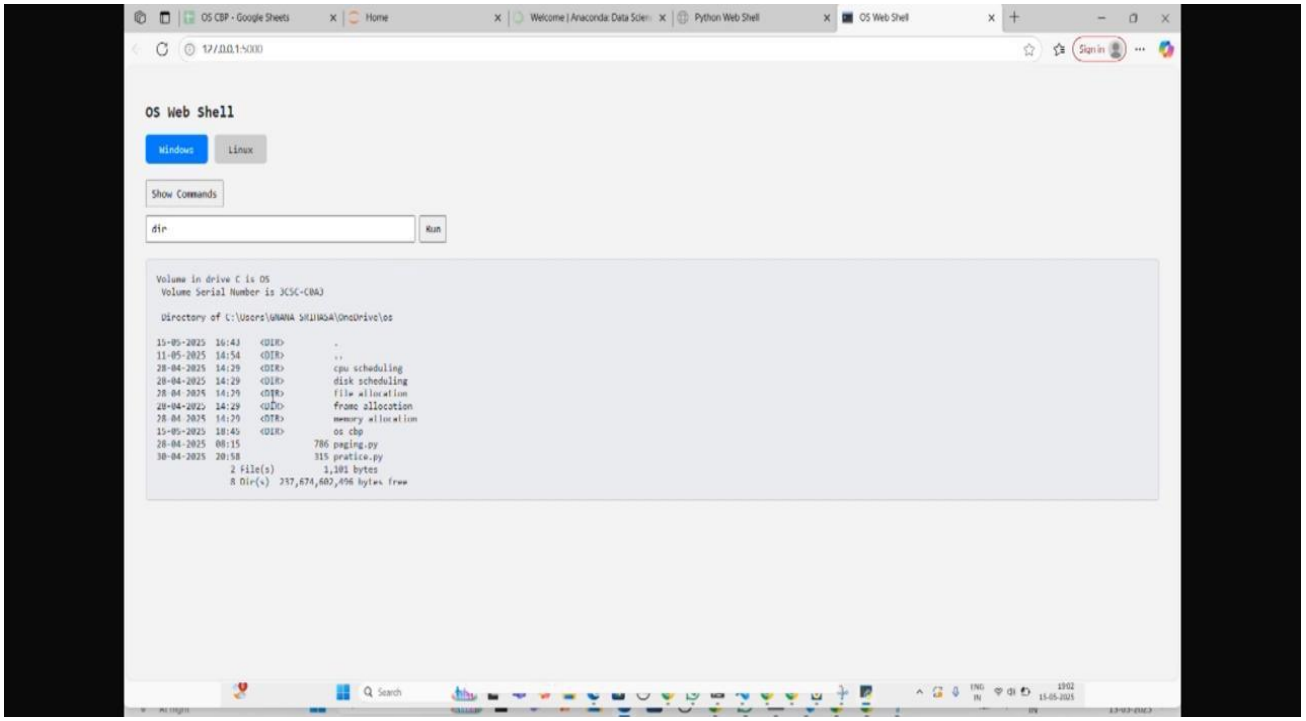

OS Web Shell
Windows
Linux
Show Commands
Enter command
Run

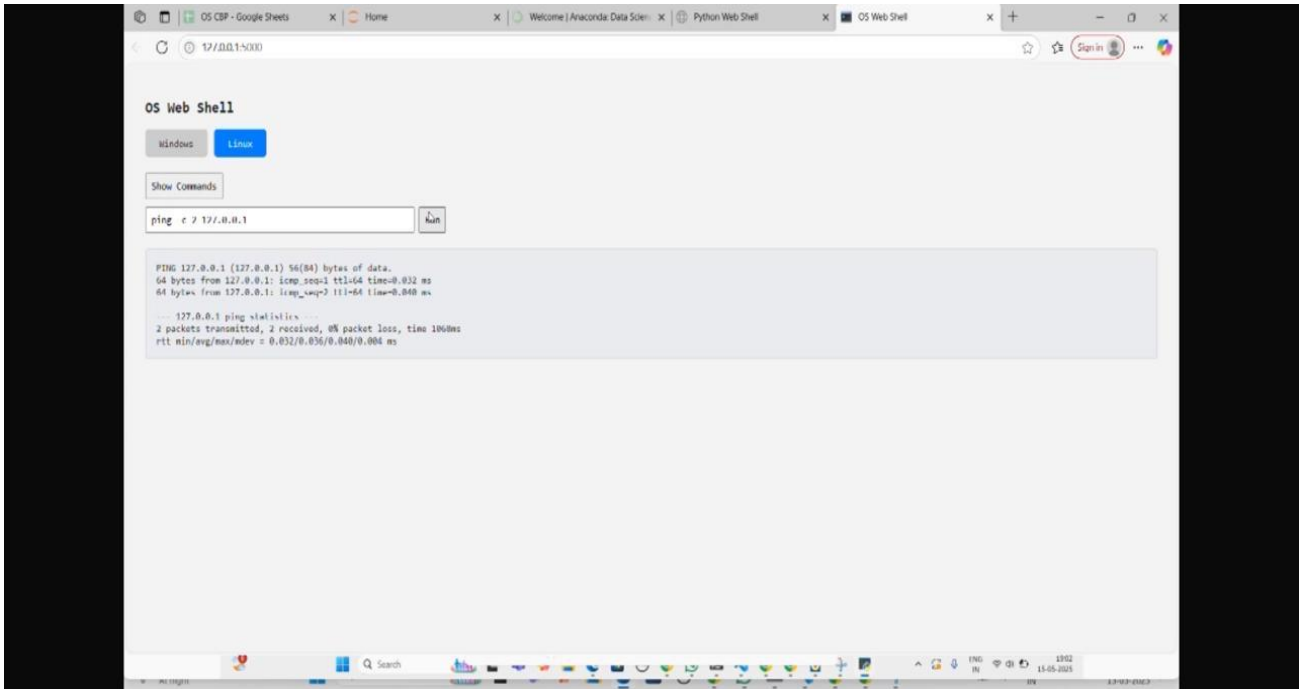

127.0.0.1:5000 says
Please enter a command.
OK
OS Web Shell
Windows
Linux
Show Commands
Enter command
Run
cat: README.txt: No such file or directory

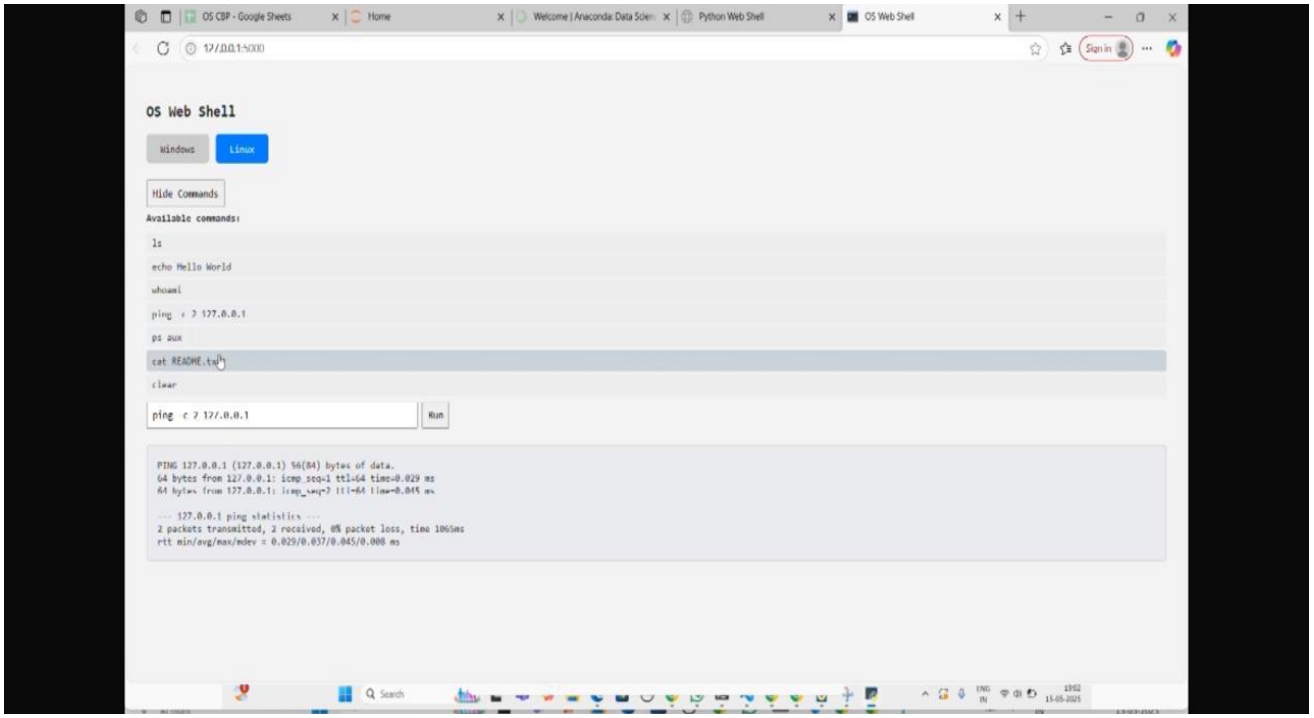

OS CBP - Google Sheets
Home
Welcome | Anaconda: Data Scien
Python Web Shell
OS Web Shell
127.0.0.1:5000
Sign in
OS Web Shell
Windows
Linux
Show Commands
dir
Run
Volume in drive C is OS
Volume Serial Number is 3C5C-C0AJ
Directory of C:\Users\GNANA SHIVADA\OneDrive\os
15-05-2025 16:43 <DIR> .
11-05-2025 14:54 <DIR> ..
28-04-2025 14:29 <DIR> cpu scheduling
28-04-2025 14:29 <DIR> disk scheduling
28-04-2025 14:29 <DIR> file allocation
28-04-2025 14:29 <DIR> frame allocation
28-04-2025 14:29 <DIR> memory allocation
15-05-2025 18:45 <DIR> os cbp
28-04-2025 08:15 786 paging.py
30-04-2025 20:58 315 pratice.py
2 File(s) 1,101 bytes
8 Dir(s) 257,674,682,496 bytes free
Search
ENG
IN
13:02
15-05-2025


OS CBP - Google Sheets
Home
Welcome | Anaconda: Data Scien
Python Web Shell
OS Web Shell
127.0.0.1:5000
Sign in
OS Web Shell
Windows
Linux
Hide Commands
Available commands:
dir
echo Hello World
whoami
ping 127.0.0.1 -n 2
tasklist
type README.txt
cls
Enter command
Run
Search
ENG
IN
13:02
15-05-2025


OS Web Shell
Windows
Linux
Hide Commands
Available commands:
dir
echo Hello World
whoami
ping 127.0.0.1 -n 3
tasklist
type README.txt
cls
echo Hello World
Run
Volume in drive C is OS
Volume Serial Number is 3C5C-C0AJ
Directory of C:\Users\GWANA SHIIWASA\OneDrive\os
15-05-2025 16:43 <DIR> .
11-05-2025 14:54 <DIR> ..
28-04-2025 14:29 <DIR> cpu scheduling
28-04-2025 14:29 <DIR> disk scheduling
28-04-2025 14:29 <DIR> file allocation
28-04-2025 14:29 <DIR> frame allocation
28-04-2025 14:29 <DIR> memory allocation
15-05-2025 18:45 <DIR> os cbp
28-04-2025 08:15 786 paging.py
30-04-2025 20:58 315 pratice.py
2 File(s) 1,101 bytes
8 Dir(s) 257,674,682,496 bytes free


OS Web Shell
Windows
Linux
Show Commands
dir
Run
Volume in drive C is OS
Volume Serial Number is 3C5C-C0AJ
Directory of C:\Users\GWANA SHIIWASA\OneDrive\os
15-05-2025 16:43 <DIR> .
11-05-2025 14:54 <DIR> ..
28-04-2025 14:29 <DIR> cpu scheduling
28-04-2025 14:29 <DIR> disk scheduling
28-04-2025 14:29 <DIR> file allocation
28-04-2025 14:29 <DIR> frame allocation
28-04-2025 14:29 <DIR> memory allocation
15-05-2025 18:45 <DIR> os cbp
28-04-2025 08:15 786 paging.py
30-04-2025 20:58 315 pratice.py
2 File(s) 1,101 bytes
8 Dir(s) 257,674,682,496 bytes free


OS CBP - Google Sheets
Home
Welcome | Anaconda: Data Scien
Python Web Shell
OS Web Shell
127.0.0.1:5000
Sign in
OS Web Shell
Windows
Linux
Show Commands
ping -c 2 127.0.0.1
Run
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.032 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.048 ms
--- 127.0.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1060ms
rtt min/avg/max/mdev = 0.032/0.056/0.040/0.004 ms
Search


OS CBP - Google Sheets
Home
Welcome | Anaconda: Data Scien
Python Web Shell
OS Web Shell
127.0.0.1:5000
Sign in
OS Web Shell
Windows
Linux
Hide Commands
Available commands:
ls
echo Hello World
whoami
ping -c 2 127.0.0.1
ps aux
cat README.txt
clear
ping -c 2 127.0.0.1
Run
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.029 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.045 ms
--- 127.0.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1065ms
rtt min/avg/max/mdev = 0.029/0.037/0.045/0.008 ms
Search

CONCLUSION

The Web-based OS Simulator & Command Shell project serves as a comprehensive bridge between theoretical operating system concepts and practical, hands-on user interaction within a fully web-accessible environment. By simulating core OS functionalities and command-line operations, the project transforms abstract notions of file management, process control, and system navigation into tangible learning experiences. This approach not only deepens understanding of operating system internals but also demonstrates how these concepts can be encapsulated in interactive software tools designed for education and experimentation.

Throughout development, the project emphasized modularity and scalability, structuring components such as command parsing, execution, and UI rendering as discrete, manageable units. This architectural choice enhances maintainability and paves the way for future expansions—whether by incorporating advanced shell features, integrating network commands, or enabling multi-user simulations. The web-based interface further broadens accessibility, allowing users from diverse backgrounds and platforms to engage with complex OS principles without the overhead of specialized software installations.

Functionally, the simulator processes user commands in real-time, mimicking the behavior of actual operating system shells, and provides informative feedback that helps users learn correct command usage and system responses. This real-time interaction, combined with a clear, user-friendly design, makes the simulator an effective educational tool that supports both self-guided learning and classroom instruction.

One of the key strengths of this project is its potential for future growth. Beyond basic shell command execution, the system's foundation is well-suited for enhancements such as scripting support, integration with backend services for persistent state management, or real-time collaboration features. Security and performance considerations were also integral to the design, ensuring that the application runs efficiently and safely within the browser context.

In summary, the Web-based OS Simulator & Command Shell project is both an instructive platform for mastering operating system concepts and a versatile foundation for building increasingly sophisticated OS simulation environments. It effectively marries theory with practice, equipping users—students, educators, and developers alike—with an engaging, accessible tool to explore and experiment with the fundamental operations that underpin modern computing systems

FUTURISTIC SCOPE

The Web-based OS Simulator & Command Shell project offers vast potential for growth and can evolve into a powerful platform for remote command execution, education, and system administration. As web technologies and cloud computing continue to advance, several exciting directions can expand the project's capabilities and impact.

Enhanced User Interface and Experience:
Introducing a richer, more interactive UI with features such as command history, autocomplete, syntax highlighting, and multi-tabbed sessions would greatly improve usability. Integrating drag-and-drop file management and real-time output streaming would simulate a full-fledged terminal experience, making it more appealing for both beginners and power users.

Multi-User and Role-Based Access:
Scaling the project to support multiple concurrent users with role-based permissions can transform it into a secure remote shell management platform. Admins could assign different access levels, monitor user activities, and audit commands, enabling collaborative and controlled server management through the web.

Cloud and Container Integration:
By linking the shell interface with cloud environments (AWS, Azure, GCP) and container orchestration systems like Kubernetes or Docker, users could seamlessly manage cloud instances or containers from their browsers. This would facilitate lightweight cloud administration and rapid testing of cloud-deployed applications.

Expanded Command Support and Customization:
Support for scripting languages, environment customization, and persistent user sessions can allow users to save scripts, automate workflows, and personalize their command environments. Adding plugins or API hooks would enable developers to extend functionality with custom commands or integrations.

Security Enhancements:
Future versions could incorporate advanced security features such as encrypted command channels (TLS), two-factor authentication, IP whitelisting, and sandboxed execution environments to prevent malicious commands from compromising the server. Logging, alerting, and anomaly detection would ensure safe and compliant use in enterprise settings.

Educational and Collaborative Features:
The tool can evolve into an interactive learning platform, allowing instructors to create guided tutorials with embedded exercises, real-time code sharing, and collaborative shells for team-based learning. Integration with LMS (Learning Management Systems) would bring command-line education to broader audiences.

Artificial Intelligence and Automation:
Incorporating AI-driven assistants to help users write commands, debug errors, or optimize shell scripts could dramatically improve productivity. Automation of routine administrative tasks via scheduled jobs, webhooks, or chatbot interfaces could reduce manual overhead.

Cross-Platform and Device Support:
Enhancing compatibility with mobile devices and integrating with voice assistants or augmented reality interfaces could provide novel ways to interact with shell environments remotely, opening doors to hands-free and ubiquitous command access.

In summary, the Web-based OS Simulator & Command Shell has the potential to grow from a simple remote terminal emulator into a comprehensive, secure, and user-friendly platform for remote system management, education, and cloud interaction. Its adaptability to evolving web and cloud ecosystems will make it an invaluable tool for developers, educators, sysadmins, and tech enthusiasts alike.

REFERENCES

1. **Silberschatz, A., Galvin, P. B., & Gagne, G. (2018).** *Operating System Concepts* (10th ed.). Wiley.
   – Provides core OS concepts including command shells, process management, memory handling, and system calls relevant to shell simulation and OS algorithm visualization.
2. **Stephenson, B. D. (2007).** *Teaching Operating Systems Using a Web-Based Simulation Tool*. *Journal of Computing Sciences in Colleges, 22*(3), 98–105.
   – Discusses the development and educational benefits of using web-based simulators to teach OS concepts like scheduling and memory management.
3. **Hsiao, H.-C., et al. (2011).** *A Web-Based Operating System Laboratory for Teaching and Learning*. *2011 IEEE International Conference on Advanced Learning Technologies*.
   – Presents a practical implementation of a web-based OS lab tool designed for interactive learning and experimentation. https://doi.org/10.1109/ICALT.2011.142
4. **Arpaci-Dusseau, R. H., & Arpaci-Dusseau, A. C. (2018).** *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books.
   – Offers practical exercises and tools (like `crash`) that simulate OS-level tasks, including simplified command shell operations and scheduling.
5. **Nand2Tetris. (n.d.).** *Project 12: Operating System and Shell*. Retrieved from https://www.nand2tetris.org/
   – A hands-on computer architecture and OS course that includes building a basic shell interface and simplified OS functionalities.
6. **GeeksforGeeks. (n.d.).** *Operating System Simulators and Shell Programming*. Retrieved from https://www.geeksforgeeks.org/
   – Offers conceptual explanations and implementations for OS shell operations and scheduling algorithms through web-based examples and code walkthroughs.
7. **Julia Evans. (2017).** *What happens when you type "ls -l"*. Retrieved from https://jvns.ca/blog/2017/03/26/what-happens-when-you-type-ls---l/
   – An educational blog post with interactive explanations on shell command execution and system calls.
8. **Red Hat. (n.d.).** *Using Web-Based Management Interfaces (Cockpit)*. Retrieved from https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/
   – Covers Cockpit, a web-based management tool for Linux systems, which allows real-time OS interaction and process control via the browser.
9. **GitHub – OS-Sim. (n.d.).** *Operating System Simulator*. Retrieved from https://os-sim.github.io
   – A browser-based simulator for visualizing OS algorithms like paging, scheduling, and memory allocation interactively.
10. **LittleOS Book. (n.d.).** *Writing a Simple Operating System from Scratch*. Retrieved from https://github.com/littleosbook/littleosbook
    – Includes command-line shell design and implementation at a low level, useful for understanding how shells interact with kernel components.