Virtual Memory Optimization Challenge

A PROJECT REPORT

Submitted by:

ABHINANDAN CHOUDHARY (12323328) (17)
UJJWAL PANDAY (12315072) (2)
JASHANPREET SINGH (12326597) (16)

KE016

in partial fulfilment for the award of the degree
of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

Lovely Professional University, Punjab

20 MARCH 2025

Lovely Professional University, Punjab

BONAFIDE CERTIFICATE

Certified that this project report " **Real-Time Process Monitoring Dashboard** " is the bonafide work of " *ABHINANDAN CHOUDHARY (12323328)*, UJJWAL PANDAY (12315072) (2), and JASHANPREET SINGH (12326597) (16)" who carried out the project work under the supervision of *HARDEEP KAUR*, Operating System Faculty (KE016).

SIGNATURE:
ABHINANDAN CHOUHDARY
UJJWAL PANDAY
JASHANPREET SINGH

SIGNATURE:
HEAD OF THE DEPARTMENT
Department of Operating System,

CSE, Lovely Professional University, Punjab

SIGNATURE
SUPERVISOR
Department of Operating System, CSE, Lovely Professional University, Punja

1. Project Overview

The Virtual Memory Optimization Challenge project aims to develop a system that effectively manages and optimizes virtual memory usage in a computing system. The goal is to improve the overall system performance by monitoring memory utilization patterns, identifying bottlenecks, and offering optimizations for better resource allocation.

This project is useful for:

- System administrators who want to manage server performance and memory usage.
- Developers working on memory-intensive applications.
- End-users who need better system responsiveness.
- IT support personnel optimizing resource allocation for applications.

Key features:

- Real-time virtual memory usage monitoring
- Memory allocation and optimization techniques.
- Visualizations of memory utilization trends.
- Suggestions for memory optimization strategies.
- Automatic adjustment of memory settings based on load.

2. Module-Wise Breakdown

2.1 System Overview Module

This module provides an overall view of the system's memory configuration and current usage. Key aspects include:

- Memory specifications (total memory, virtual memory, swap memory, etc.)
- Current memory usage (used, free, cached)
- Virtual memory status and paging activity
- Overall system performance (CPU, disk, etc.)

2.2 Memory Monitoring Module

The Memory Monitoring Module will track the system's memory usage in real time. This includes:

- Real-time memory usage (used, free, and cached memory)
- Memory fragmentation and paging statistics
- Monitoring of virtual memory (swap space) utilization
- Alerts for when memory usage exceeds a certain threshold

2.3 Memory Optimization Module

This module will provide memory optimization suggestions based on real-time data. It includes:

- Recommendations for reducing memory fragmentation
- Dynamic allocation adjustments based on current memory needs
- Swap space management (e.g., freeing up swap memory when necessary)
- Automatic adjustment of memory settings to optimize performance

2.4 Performance Visualization Module

This module will allow users to visualize memory usage trends. Key visualizations include:

- Memory usage over time (real-time graph)
- Swap memory usage vs. physical memory usage
- Memory fragmentation trends
- Heatmaps or pie charts for memory allocation
- Alerts for abnormal memory usage

2.5 Virtual Memory Management Module

This will be the core of the optimization strategy. This module will dynamically manage virtual memory to ensure efficient usage, including:

- Virtual memory page replacement strategies
- Fine-tuning memory paging to prevent excessive disk swapping
- Memory reallocation suggestions based on system load
- Recommendations for optimizing virtual memory settings (e.g., increasing or decreasing swap space size)

3. Functionalities

3.1 System Memory Overview

- Display total physical memory, virtual memory, and swap memory usage.
- Show real-time memory utilization, including free, used, and cached memory.
- Display memory fragmentation and paging statistics.

3.2 Real-time Memory Monitoring

- Monitor and display memory utilization over time.
- Track swap memory usage and paging activity.
- Provide alerts if memory usage exceeds predefined thresholds (e.g., 90% usage).

3.3 Memory Optimization Suggestions

- Provide actionable suggestions to reduce memory fragmentation.
- Suggest increasing or decreasing virtual memory settings.

- Recommend swapping strategies to improve memory access speed.

3.4 Memory Usage Visualization

- Visualize memory usage in interactive graphs.
- Provide a historical view of memory usage trends.
- Visualize memory fragmentation with heatmaps or charts.

3.5 Virtual Memory Page Replacement

- Implement memory page replacement strategies to optimize virtual memory performance.
- Minimize disk swapping by optimizing memory pages in use.

4. Technology Used

4.1 Programming Languages

- Python: The entire application is written in Python, leveraging its rich ecosystem of libraries and tools for system interaction and data visualization.

4.2 Libraries and Tools

1. Streamlit:
    - Used for creating the web application interface
    - Provides interactive widgets and layout options
    - Enables real-time updates of data and graphs
2. psutil (Python System and Process Utilities):
    - Used for retrieving information on system utilization and resources
    - Provides cross-platform compatibility for system monitoring
3. Plotly:
    - Creates interactive and dynamic graphs
    - Used for visualizing performance metrics over time

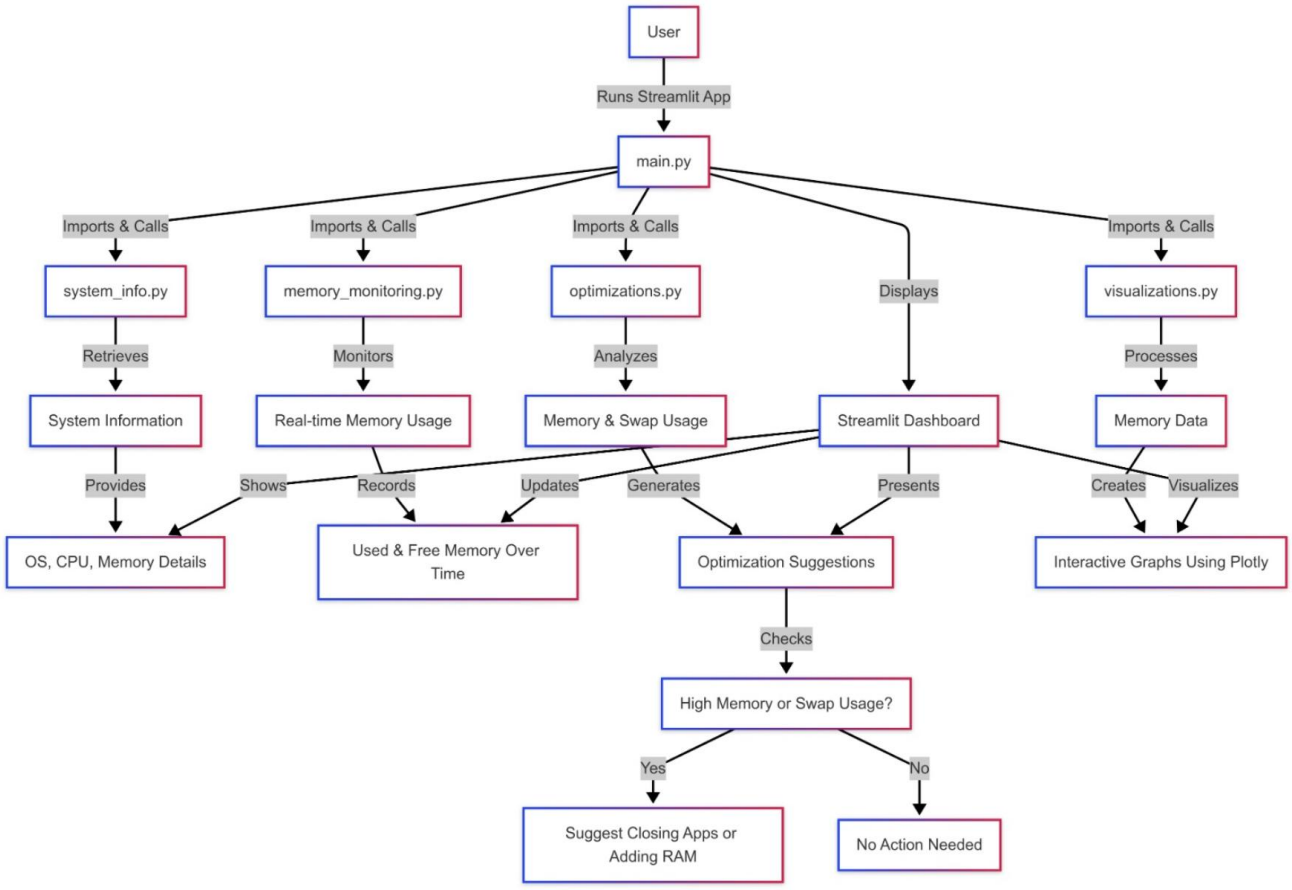
4. time and datetime:
    - Handle time-related operations and formatting
5. Numpy/pandas:
6. For efficient data storage, manipulation, and analysis.

4.3 Other Tools

- GitHub: Used for version control and collaborative development
- Virtual Environment: Recommended for managing project dependencies

Requirements.txt

5. Flow Diagram



6. Conclusion and Future Scope

The Task Manager application provides a comprehensive solution for system monitoring and management. It offers users valuable insights into their system's performance and resource utilization in a user-friendly interface.

Key achievements of the project:

- Real-time monitoring of system memory and swap space usage.
- Visualization of memory usage trends and fragmentation.
- Memory optimization strategies for improved system performance.
- Detailed suggestions for virtual memory management based on system behaviours.

Future enhancements could include:

1. **Advanced Memory Management Strategies:**
    - Implement more advanced algorithms for memory allocation and deallocation.
    - Explore machine learning techniques to predict memory needs based on usage patterns.

2. **Cross-Platform Support:**
    - Ensure compatibility with Windows, Linux, and macOS for broader usage.

3. **Historical Memory Data:**
    - Store long-term memory usage data for trend analysis and predictions.

4. **Remote Memory Monitoring:**
    - Implement remote monitoring of virtual memory on multiple systems from a central dashboard.

5. **Automated Memory Cleanup:**
    - Automatically adjust memory settings or clean up memory during idle periods for better resource usage.

Appendix

A. Installation Instructions

1. Clone the repository:

   https://github.com/abhi2004c/CSE316_Project.git

2. Create and activate a virtual environment:

   python -m venv venv

   source venv/bin/activate

3. Install required packages:

   pip install -r requirements.txt

4. Run the application:

   streamlit run main.py

B. Detailed Usage Guide

1. **System Overview:**
    - View memory utilization, swap space status, and real-time metrics.

2. **Memory Monitoring:**
    - Monitor real-time memory data.
    - Set thresholds for alerts when memory usage is too high.

3. **Optimization Suggestions:**
    - Receive recommendations for optimizing memory settings and preventing excessive paging.

4. **Visualization:**
    - View memory usage trends, including swap space usage, fragmentation, and historical data.

C. Troubleshooting Common Issues

1. **Application fails to start:**
    - Ensure all dependencies are correctly installed
    - Check for any error messages in the console
2. **Incorrect memory data:**
    - Ensure your system has sufficient privileges to access memory stats.
    - Check if psutil is correctly installed.

A. AI-Generated Project Elaboration/Breakdown Report

1. Project Overview

Purpose:
The Virtual Memory Optimization Challenge project is designed to optimize memory usage on systems through real-time monitoring, analysis, and management of virtual memory. This system

helps improve the overall performance of computing systems, especially those dealing with memory-heavy applications.

Key Features:

- **Real-time Virtual Memory Monitoring**: Keeps track of system memory usage.
- **Memory Allocation & Optimization**: Provides suggestions and techniques to optimize system memory.
- **Visualization**: Displays trends in memory usage, fragmentation, and swap space.
- **Memory Optimization Strategies**: Proposes dynamic changes to memory settings for better resource allocation.
- **Automatic Memory Adjustment**: Adjusts memory settings automatically based on system load.

---

2. Module-Wise Breakdown

2.1 System Overview Module
Provides the overall memory configuration and usage of the system, displaying details such as total memory, virtual memory, swap memory, and system performance metrics (CPU, disk, etc.).

2.2 Memory Monitoring Module
Tracks real-time memory usage, including free, used, cached memory, and swap space. Alerts are generated when memory usage exceeds a certain threshold, ensuring the system does not become resource-starved.

2.3 Memory Optimization Module
Suggests optimizations for memory allocation based on real-time data. This includes reducing fragmentation, managing swap space, and adjusting memory settings dynamically based on current needs.

2.4 Performance Visualization Module
Provides visual representations (graphs, charts) of memory usage trends, including fragmentation and swap space usage over time.

2.5 Virtual Memory Management Module
Manages virtual memory by implementing strategies for efficient memory allocation, such as page replacement and preventing excessive disk swapping.

---

3. Functionalities

3.1 System Memory Overview
Displays the total physical memory, virtual memory, and swap memory usage. Also shows the current state of memory utilization (free, used, cached).

3.2 Real-time Memory Monitoring
Monitors memory utilization in real time and displays alerts when the memory usage exceeds predefined thresholds.

3.3 Memory Optimization Suggestions
Offers suggestions on reducing memory fragmentation, adjusting virtual memory settings, and using better swap strategies.

3.4 Memory Usage Visualization
Visualizes the system's memory usage over time through interactive graphs and charts, providing users with a historical view of trends.

3.5 Virtual Memory Page Replacement
Implements strategies for managing memory pages and optimizing virtual memory performance to reduce disk swapping and ensure the system runs efficiently.

---

4. Technology Used

4.1 Programming Languages

- **Python**: The application is written in Python, utilizing its broad ecosystem of libraries for system interactions and data visualization.

4.2 Libraries and Tools

1. **Streamlit**: For building the interactive web interface with real-time updates of memory data.
2. **psutil**: To retrieve system memory and resource utilization data.
3. **Plotly**: To create interactive graphs and charts for visualizing memory performance metrics.
4. **time & datetime**: For managing time-related operations, ensuring real-time updates of memory data.
5. **Numpy/Pandas**: Used for data storage and manipulation.

4.3 Other Tools

- **GitHub**: For version control and collaborative development.
- **Virtual Environment**: Recommended for managing project dependencies and ensuring compatibility across systems.

---

5. Flow Diagram

A flow diagram would show how each module interacts:

- **Memory Data Collection**: Using psutil to gather system memory data.
- **Real-time Memory Status**: Presenting the data in the system overview module.
- **Analysis & Optimization**: Identifying potential issues like fragmentation and providing optimization suggestions.
- **Visualization**: Plotting memory trends and usage statistics.
- **Dynamic Memory Adjustment**: Automatically adjusting memory settings based on system load.

---

6. Conclusion and Future Scope

Key Achievements:

- Real-time monitoring of system memory and swap space.
- Visualization of memory usage trends and fragmentation.
- Suggestions for memory optimizations based on real-time data.

Future Enhancements:

1. **Advanced Memory Management**: Incorporate machine learning algorithms to predict memory requirements based on historical patterns.
2. **Cross-Platform Support**: Expand compatibility to support Windows, Linux, and macOS.
3. **Historical Data Storage**: Store long-term memory usage data for analysis and trend forecasting.
4. **Remote Monitoring**: Enable monitoring of multiple systems from a centralized dashboard.
5. **Automated Memory Cleanup**: Automatically manage system resources, clearing memory during idle periods for better efficiency.

---

7. Appendix

A. Installation Instructions

1. Clone the repository:

   https://github.com/abhi2004c/CSE316_Project.git

2. Create and activate a virtual environment:

python -m venv venv

source venv/bin/activate

3. Install required packages:

pip install -r requirements.txt

4. Run the application:

streamlit run main.py

B. Detailed Usage Guide

- **System Overview**: View memory status and real-time metrics.
- **Memory Monitoring**: Monitor memory usage and set alerts.
- **Optimization Suggestions**: Receive recommendations for memory settings.
- **Visualization**: View interactive graphs and charts of memory trends.

C. Troubleshooting Common Issues

1. **Application fails to start**:
    - Ensure all dependencies are installed.
    - Check the console for error messages.
2. **Incorrect memory data**:
    - Ensure proper privileges are granted to access memory stats.
    - Verify that psutil is correctly installed.

B. Problem Statement: [Paste Problem Statement]

7**. Virtual Memory Optimization Challenge Description:**

Create a virtual memory management tool that visualizes the behaviour of paging and segmentation, including page faults and demand paging. Allow users to experiment with custom inputs for memory allocation, simulate memory fragmentation, and evaluate page replacement algorithms like LRU and Optimal.

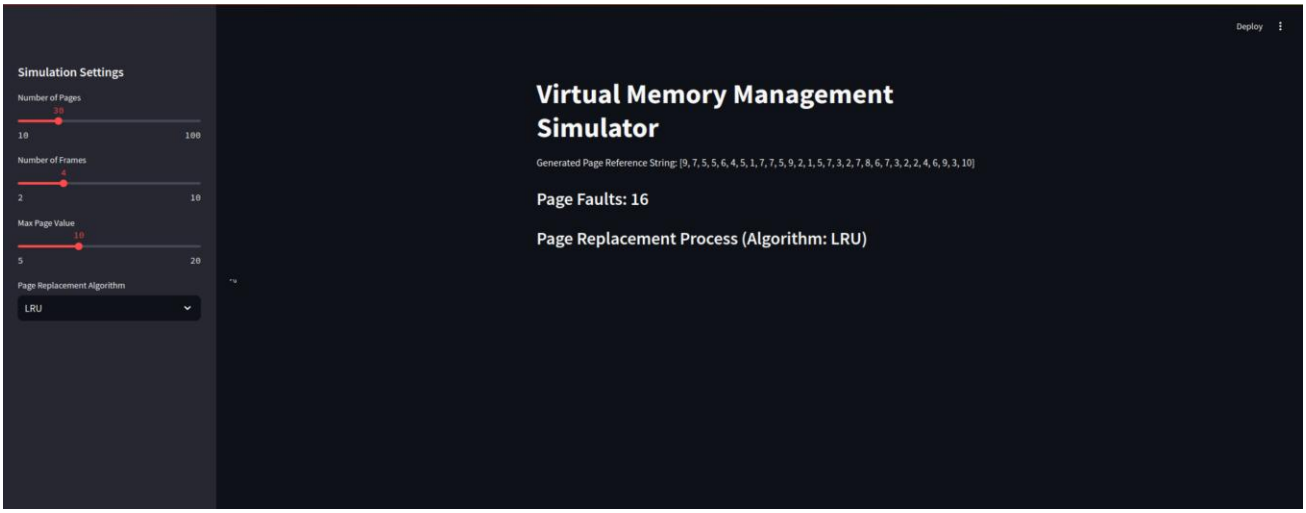
C. Solution/Code:

Paging_simulator.py

```
import numpy as np
import random
from collections import deque


def simulate_page_faults(pages, frames, algorithm='LRU'):
    """
    Simulates page faults using the selected page replacement algorithm.
    """
    page_faults = 0
    memory = []
    page_order = []  # For visualization purposes
    if algorithm == 'LRU':
        page_faults, memory, page_order = lru_page_replacement(pages, frames)
    elif algorithm == 'Optimal':
        page_faults, memory, page_order = optimal_page_replacement(pages, frames)


    return page_faults, memory, page_order


def lru_page_replacement(pages, frames):
    """
    Simulates Least Recently Used (LRU) page replacement algorithm.
    """
    memory = []
    page_order = []
    page_faults = 0
    recent_pages = deque()
```

```
    for page in pages:
        if page not in memory:
            page_faults += 1
            if len(memory) < frames:
                memory.append(page)
            else:
                # Remove least recently used page only if it exists in memory
                least_recently_used = recent_pages.popleft()
                if least_recently_used in memory:
                    memory.remove(least_recently_used)
                memory.append(page)
        recent_pages.append(page)
        page_order.append(list(memory))

    return page_faults, memory, page_order



def optimal_page_replacement(pages, frames):
    """
    Simulates Optimal page replacement algorithm.
    """
    memory = []
    page_order = []
    page_faults = 0

    for i, page in enumerate(pages):
        if page not in memory:
            page_faults += 1
            if len(memory) < frames:
                memory.append(page)
```

```
            else:
                # Replace page that won't be used for the longest time
                farthest = -1
                index_to_replace = -1
                for j, old_page in enumerate(memory):
                    try:
                        next_use = pages[i+1:].index(old_page)
                    except ValueError:
                        next_use = float('inf')
                    if next_use > farthest:
                        farthest = next_use
                        index_to_replace = j
                memory[index_to_replace] = page
        page_order.append(list(memory))

    return page_faults, memory, page_order


def generate_page_reference_string(num_pages, max_page_value=10):
    """
    Generates a random page reference string of length `num_pages`.
    """
    return [random.randint(1, max_page_value) for _ in range(num_pages)]
```



Visualization.py

```
import plotly.graph_objects as go


def visualize_page_replacement(page_order):
    """
    Visualize the page replacement process using Plotly.
    """
    fig = go.Figure()
    for i, memory_state in enumerate(page_order):
        fig.add_trace(go.Scatter(x=[i] * len(memory_state), y=memory_state, mode='markers+lines', name=f'Time {i}'))


    fig.update_layout(
        title="Page Replacement Visualization",
        xaxis_title="Time",
        yaxis_title="Page",
        template="plotly_dark",
        showlegend=False
```

```
    )

    fig.show()

def visualize_page_faults(page_faults):
    """
    Display the number of page faults.
    """
    fig = go.Figure(data=[go.Bar(x=['Page Faults'], y=[page_faults], name='Page Faults')])
    fig.update_layout(title="Number of Page Faults", xaxis_title="Page Faults", yaxis_title="Count")
    fig.show()
```

Main.py

```
import streamlit as st
import random
from paging_simulator import simulate_page_faults, generate_page_reference_string
from visualizations import visualize_page_replacement, visualize_page_faults
def app():
    st.title("Virtual Memory Management Simulator")

    st.sidebar.header("Simulation Settings")

    # User input for page reference string
    num_pages = st.sidebar.slider("Number of Pages", min_value=10, max_value=100, value=30)
    frames = st.sidebar.slider("Number of Frames", min_value=2, max_value=10, value=4)
    max_page_value = st.sidebar.slider("Max Page Value", min_value=5, max_value=20, value=10)

    # Choose algorithm
    algorithm = st.sidebar.selectbox("Page Replacement Algorithm", ['LRU', 'Optimal'])

    # Generate a random page reference string
    pages = generate_page_reference_string(num_pages, max_page_value)
    st.write(f"Generated Page Reference String: {pages}")

    # Run the page replacement simulation
    page_faults, memory, page_order = simulate_page_faults(pages, frames, algorithm)

    # Display page fault count
    st.subheader(f"Page Faults: {page_faults}")
    visualize_page_faults(page_faults)

    # Visualize page replacement behavior
```

```
    st.subheader(f"Page Replacement Process (Algorithm: {algorithm})")
    visualize_page_replacement(page_order)

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