

VIRTUAL MEMORY MANAGER

A MINI-PROJECT REPORT

Submitted by

JOSE MUGILAN D 231901016

KARTHIK R 231901017

NAVEEN C H 231901033

in partial fulfillment of the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

(CYBER SECURITY)



RAJALAKSHMI ENGINEERING COLLEGE

ANNA UNIVERSITY

CHENNAI – 600 025

MAY 2025

BONAFIDE CERTIFICATE

Certified that this project “**VIRTUAL MEMORY MANAGER**” is the bonafide work of “**JOSE MUGILAN D ,KARTHIK R &NAVEEN C H**” who carried out the project work under my supervision.

SIGNATURE

Mrs.V.JANANEE

ASSISTANT PROFESSOR(SG)

Department of Computer Science
and Engineering,
Rajalakshmi Engineering College,
Chennai

This mini project report is submitted for the viva voce examination to be held on _____

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We express our sincere thanks to our Head of the Department **Mr.S.MEGANATHAN** and the Chairperson **Dr.M.THANGAM MEGANATHAN** for Their timely support and encouragement.We are greatly indebted to our respected and honourable principal **Dr. S.N.MURUGESAN**, For his able support and guidance.We express our sincere thanks to our Head of the Department **Mr. Benedict JN**, for encouragement and being ever supporting force during our project work.We also extend our sincere and hearty thanks to our internal guide **Mrs.V.JANANEE**, for her valuable guidance and motivation during the completion of this project.Our sincere thanks to our family members, friends and other staff members of computer science engineering.

- 1. JOSE MUGILAN D (231901016)**
- 2. KARTHIK R (231901017)**
- 3. NAVEEN C H (231901033)**

ABSTRACT

Virtual Memory Manager Simulator that demonstrates how different page replacement algorithms such as FIFO, LRU, and Optimal Replacement handle memory management tasks. Users can input custom page reference strings and memory frame counts to observe detailed, real-time simulation outputs including page hits, page faults, and memory states. The simulator's user-friendly web and desktop interfaces, combined with visual outputs and statistical metrics like hit rate and page fault count, simplify complex memory management concepts. Designed as a practical learning tool, it bridges the gap between theoretical understanding and real-world memory management techniques.

TABLE OF CONTENTS

SNO	TOPIC	PAGE NO
1.	INTRODUCTION	6
2.	SCOPE OF THE PROJECT	8
3.	ARCHITECTURE	12
4.	FLOWCHART	14
5.	PROGRAM CODE	16
6.	RESULTS	28
7.	CONCLUSION	33
8.	FUTUREWORKS	34
9.	REFERENCES	36

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

This project focuses on simulating virtual memory management in operating systems, aiming to enhance understanding of how different page replacement algorithms impact system performance. The simulator allows users to visualize and compare algorithms like FIFO, LRU, and Optimal Replacement. By providing a graphical and interactive environment through both web and desktop applications, it helps learners grasp the logic behind memory management and page handling processes.

1.2 SCOPE OF THE WORK

The Virtual Memory Manager Simulator serves as an educational tool to demonstrate the behavior of page replacement algorithms under various memory access patterns. It enables users to input custom page reference sequences, select memory frame sizes, and analyze page faults and hit rates. The project can be used by students, educators, and developers to better understand memory management strategies and their performance impacts.

1.3 PROBLEM STATEMENT

Understanding virtual memory management and page replacement strategies is critical but often difficult due to their abstract nature and dynamic behaviors. Many learners struggle to connect theoretical

concepts with real-world memory access scenarios. This project addresses that gap by offering an intuitive, visual way to simulate and evaluate different page replacement algorithms, making complex topics easier to comprehend through hands-on interaction and statistical analysis

1.4 AIM AND OBJECTIVES OF THE PROJECT

The aim of this project is to develop a user-friendly simulator that effectively illustrates the functioning of different page replacement algorithms in virtual memory management. The project seeks to implement algorithms such as First-In-First-Out (FIFO), Least Recently Used (LRU), and Optimal Page Replacement within an interactive web and desktop interface. It allows users to input page reference sequences, set the number of memory frames, observe how pages are replaced in memory, and evaluate the performance based on metrics like page faults and hit rates. The ultimate goal is to simplify the learning process and bridge the gap between theoretical knowledge and practical application of memory management concep

CHAPTER 2

SCOPE OF THE PROJECT

The Virtual Memory Manager Simulator offers an interactive environment to visualize and analyze core page replacement algorithms including FIFO, LRU, and Optimal Replacement. Designed with an educational focus, the tool helps users understand how different memory management strategies affect system performance, page fault rates, and memory efficiency.

Page Replacement Strategy Implementation

- **Simulates essential page replacement algorithms:**
 - First-In-First-Out (FIFO): Replaces the oldest loaded page in memory first.
 - Least Recently Used (LRU): Replaces the page that has not been used for the longest period.
 - Optimal Page Replacement: Replaces the page that will not be used for the longest duration in the future.

- **Each algorithm's impact on performance is visualized through:**
 - **Step-by-step memory state displays** showing real-time page loading and replacement.
 - **Comparative performance graphs for:**
 - **Page Faults** (total number of faults for each algorithm)
 - **Hit Rates** (percentage of successful page hit).

Interactive GUI and Visualization

- **Tkinter-based GUI allows:**
 - Easy input of page reference sequences and memory frame sizes.
 - Selection of page replacement algorithms (FIFO or LRU).
 - Real-time table visualization showing memory states, page hits, and misses after each page reference.
- **Web-based GUI (Flask) provides:**
 - Easy-to-use form for entering page references and selecting frames.
 - Dynamic display of memory state changes and page fault statistics.
 - Interactive charts showing page fault trends over time.
- **Visualization Features:**
 - Step-by-step simulation logs displayed in tabular format.
 - Hit/Miss status indicators for each memory access.
 - Comparative performance analysis among FIFO, LRU, and Optimal algorithms.

User Customization and Flexibility

- **Users can configure:**
 - Number of memory frames available.
 - Page reference string (sequence of page numbers).
 - Selection of page replacement algorithms (FIFO, LRU, Optimal).
- **Supports varied simulation setups for in-depth comparative analysis:**
 - Simulate different memory access patterns and observe algorithm performance.
 - Analyze page fault rates and hit rates under various memory frame settings.
 - Compare efficiency and behavior of different algorithms interactively.

Educational and Research Relevance

- **Designed for:**
 - Students learning operating system concepts like memory management and page replacement strategies.
 - Educators as a visual aid in classroom demonstrations to explain virtual memory behavior.
 - Researchers for experimentation with different memory access patterns and replacement algorithms.

- **Provides foundational understanding of:**
 - Virtual memory management and page replacement techniques.
 - Effects of page faults and hit rates on system performance.
 - Comparative study of memory management strategies under varying conditions.

Performance Analysis and Reporting

- **Live metrics allow users to:**
 - Compare page replacement algorithms (FIFO, LRU, Optimal) based on page faults and hit rates.
 - Analyze efficiency and memory utilization under different page sequences and frame counts.
- **Offers concrete data for:**
 - Decision-making in choosing the most effective memory management strategy.
 - Deeper exploration and analysis of operating system memory management theories

CHAPTER 3

ARCHITECTURE

- **Page Table:** Stores the list of all page references input by the user for simulation.
- **Memory Frames Setup:** Stores the list of all page references input by the user for simulation.
- **Scheduler (Page Replacement Controller):** Manages and selects pages for replacement based on the selected algorithm.
- **Page Replacement Algorithms:**
 - **First-In-First-Out (FIFO):** Replaces the oldest loaded page first.
 - **Least Recently Used (LRU):** Replaces the page that was not used for the longest time.
 - **Optimal Replacement:** Replaces the page that will not be needed for the longest time in the future.
- **Memory State Visualization:** Real-time table showing the changes in memory after each page reference (HIT/MISS status).
- **Page Fault Count Calculation:** Calculates the total number of page faults that occurred during simulation.
- **Hit Rate Calculation:** Measures the percentage of successful page hits.

- **User Interface (Web/Desktop):**
 - Allows users to input page sequences, number of frames, and select algorithms.
 - Displays step-by-step simulation results and performance metrics.

CHAPTER 4

FLOWCHART

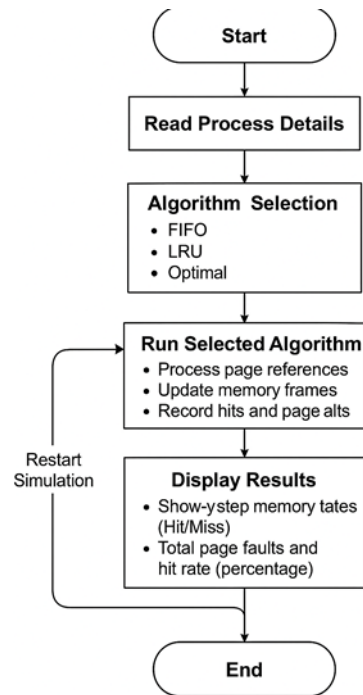


Figure 2: Flowchart of CPU Scheduling

1. Start

- This is the entry point of the application.
- It signifies the beginning of the page replacement simulation when the user runs the program or clicks "Run Simulation".

2. Read process details

- **The user inputs:**
 - Page reference string (sequence of page numbers).
 - Number of memory frames available.
 - Selected page replacement algorithm (FIFO, LRU, or Optimal).

3. Algorithm Selection

- **Based on user input, the program selects:**
 - **FIFO** (First-In-First-Out)
 - **LRU** (Least Recently Used)
 - **Optimal Page Replacement**

4. Run Selected Algorithm

- block The chosen algorithm processes the page references.
- Memory frames are updated step-by-step.
- Hits and page faults are recorded during simulation.

5. Display Results

- **Final output is displayed:**
 - Step-by-step memory states (Hit/Miss).
 - Total page faults and hit rate percentage.
 - Graphs comparing performance (in web application).

6. End

- The simulation ends.
- The user can restart by entering new inputs for a fresh simulate

CHAPTER 5

CODE IMPLEMENTATION

5.1. PROGRAM

5.1.1. APP.PY

```
from flask import Flask, render_template, request, jsonify
from collections import deque
from datetime import datetime
import statistics

app = Flask(__name__)

def fifo_replacement(frames, pages):
    memory = deque(maxlen=frames)
    page_faults = 0
    steps = []

    for i, page in enumerate(pages):
        is_hit = page in memory
        old_state = list(memory)

        if not is_hit:
            page_faults += 1
            if len(memory) == frames:
                memory.popleft() # Remove the first-in page
                memory.append(page)

        steps.append({
            'step': i + 1,
            'page': page,
            'memory_state': list(memory),
            'is_hit': is_hit,
            'page_faults': page_faults
        })
```


return steps

def lru_replacement(frames, pages):

memory = []

page_faults = 0

steps = []

page_last_used = {}

for i, page in enumerate(pages):

is_hit = page in memory

old_state = list(memory)

if not is_hit:

page_faults += 1

if len(memory) == frames:

Find least recently used page

lru_page = min(memory, key=lambda x: page_last_used[x])

memory.remove(lru_page)

memory.append(page)

Update last used time for current page

page_last_used[page] = i

steps.append({

'step': i + 1,

'page': page,

'memory_state': list(memory),

'is_hit': is_hit,

'page_faults': page_faults

})

return steps

def optimal_replacement(frames, pages):

```

memory = []
page_faults = 0
steps = []

for i, page in enumerate(pages):
    is_hit = page in memory
    old_state = list(memory)

    if not is_hit:
        page_faults += 1
        if len(memory) == frames:
            # Find the page that won't be used for the longest time
            future_use = {}
            for p in memory:
                try:
                    next_use = next(j for j in range(i + 1, len(pages)) if pages[j] == p)
                    future_use[p] = next_use
                except StopIteration:
                    future_use[p] = float('inf') # Page won't be used again

            # Remove the page that won't be used for the longest time
            page_to_remove = max(memory, key=lambda x: future_use[x])
            memory.remove(page_to_remove)
            memory.append(page)

        steps.append({
            'step': i + 1,
            'page': page,
            'memory_state': list(memory),
            'is_hit': is_hit,
            'page_faults': page_faults
        })

return steps

```

```

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

@app.route('/simulate', methods=['POST'])
def simulate():
    data = request.get_json()
    pages = [int(x.strip()) for x in data['pages'].split(',')]
    frames = int(data['frames'])

    # Run all three algorithms
    fifo_steps = fifo_replacement(frames, pages)
    lru_steps = lru_replacement(frames, pages)
    optimal_steps = optimal_replacement(frames, pages)

    # Get final page faults for each algorithm
    fifo_faults = fifo_steps[-1]['page_faults']
    lru_faults = lru_steps[-1]['page_faults']
    optimal_faults = optimal_steps[-1]['page_faults']

    # Calculate statistics
    all_faults = [fifo_faults, lru_faults, optimal_faults]
    avg_faults = statistics.mean(all_faults)
    best_algorithm = min(
        [("FIFO", fifo_faults), ("LRU", lru_faults), ("Optimal", optimal_faults)],
        key=lambda x: x[1]
    )

    worst_algorithm = max(
        [("FIFO", fifo_faults), ("LRU", lru_faults), ("Optimal", optimal_faults)],
        key=lambda x: x[1]
    )

```

```
# Calculate hit rates
total_pages = len(pages)
fifo_hit_rate = ((total_pages - fifo_faults) / total_pages) * 100
lru_hit_rate = ((total_pages - lru_faults) / total_pages) * 100
optimal_hit_rate = ((total_pages - optimal_faults) / total_pages) * 100

return jsonify({
    'algorithms': {
        'fifo': {
            'steps': fifo_steps,
            'page_faults': fifo_faults,
            'hit_rate': round(fifo_hit_rate, 2)
        },
        'lru': {
            'steps': lru_steps,
            'page_faults': lru_faults,
            'hit_rate': round(lru_hit_rate, 2)
        },
        'optimal': {
            'steps': optimal_steps,
            'page_faults': optimal_faults,
            'hit_rate': round(optimal_hit_rate, 2)
        }
    }
})
```

```
'statistics': {  
    'average_faults': round(avg_faults, 2),  
    'best_algorithm': {  
        'name': best_algorithm[0],  
        'faults': best_algorithm[1]  
    },  
    'worst_algorithm': {  
        'name': worst_algorithm[0],  
        'faults': worst_algorithm[1]  
    }  
}  
})  
  
if __name__ == '__main__':  
    app.run(debug=True)
```

5.1.2 DESKTOP_APP.PY

```

import tkinter as tk

from tkinter import ttk, messagebox

from collections import deque


class MemoryMatrixApp:

    def __init__(self, root):

        self.root = root

        self.root.title("Memory Matrix")

        self.root.geometry("1000x800")

        self.root.configure(bg='#0a0a0a')


        style = ttk.Style()

        style.configure("Cyberpunk.TFrame", background='#0a0a0a')

        style.configure("Cyberpunk.TLabel",

            background='#0a0a0a',

            foreground='#00ff41',

            font=('Courier', 10))

        style.configure("Cyberpunk.TButton",

            background='#00ff41',

            foreground='#0a0a0a',

            font=('Courier', 10))


        # Main frame

        self.main_frame = ttk.Frame(root, style="Cyberpunk.TFrame")

        self.main_frame.pack(padx=20, pady=20, fill=tk.BOTH, expand=True)


        # Title

        title = ttk.Label(self.main_frame,

            text="MEMORY MATRIX",

            style="Cyberpunk.TLabel",

```

```

font=('Courier', 24, 'bold'))
title.pack(pady=(0, 20))

# Status Guide
guide_frame = ttk.Frame(self.main_frame, style="Cyberpunk.TFrame")
guide_frame.pack(fill=tk.X, pady=(0, 20))
ttk.Label(guide_frame,
          text="HIT: Page found in memory (Fast)",
          style="Cyberpunk.TLabel",
          foreground='#00ff41').pack(side=tk.LEFT, padx=10)
ttk.Label(guide_frame,
          text="MISS: Page not in memory (Slow)",
          style="Cyberpunk.TLabel",
          foreground='#ff4141').pack(side=tk.RIGHT, padx=10)

# Input Frame
input_frame = ttk.Frame(self.main_frame, style="Cyberpunk.TFrame")
input_frame.pack(fill=tk.X, pady=(0, 20))

# Pages Input
ttk.Label(input_frame,
          text="INPUT SEQUENCE:",
          style="Cyberpunk.TLabel").pack(anchor=tk.W)
self.pages_entry = ttk.Entry(input_frame)
self.pages_entry.pack(fill=tk.X, pady=(5, 10))
self.pages_entry.insert(0, "1,2,3,4,1,2,5,1,2,3,4,5")

# Frames Input
ttk.Label(input_frame,
          text="MEMORY FRAMES:",
          style="Cyberpunk.TLabel").pack(anchor=tk.W)
self.frames_entry = ttk.Entry(input_frame)
self.frames_entry.pack(fill=tk.X, pady=(5, 10))
self.frames_entry.insert(0, "3")

```

```
# Algorithm Selection
```

```
ttk.Label(input_frame,
          text="ALGORITHM:",
          style="Cyberpunk.TLabel").pack(anchor=tk.W)
self.algorithm = tk.StringVar(value="fifo")
ttk.Radiobutton(input_frame,
                 text="FIFO (First In, First Out)",
                 variable=self.algorithm,
                 value="fifo",
                 style="Cyberpunk.TLabel").pack(anchor=tk.W)
ttk.Radiobutton(input_frame,
                 text="LRU (Least Recently Used)",
                 variable=self.algorithm,
                 value="lru",
                 style="Cyberpunk.TLabel").pack(anchor=tk.W)
```

```
# Run Button
```

```
ttk.Button(input_frame,
            text="RUN",
            command=self.run_simulation,
            style="Cyberpunk.TButton").pack(fill=tk.X, pady=(20, 0))
```

```
# Results Frame
```

```
self.results_frame = ttk.Frame(self.main_frame, style="Cyberpunk.TFrame")
self.results_frame.pack(fill=tk.BOTH, expand=True)
```

```
# Table
```

```
self.table = ttk.Treeview(self.results_frame,
                           columns=("step", "page", "memory", "status"),
                           show="headings",
                           style="Cyberpunk.Treeview")
self.table.heading("step", text="STEP")
self.table.heading("page", text="PAGE")
self.table.heading("memory", text="MEMORY STATE")
self.table.heading("status", text="STATUS")
```



```

self.table.pack(fill=tk.BOTH, expand=True)

# Stats Label
self.stats_label = ttk.Label(self.results_frame,
                             text="",
                             style="Cyberpunk.TLabel")
self.stats_label.pack(pady=10)

def fifo_replacement(self, frames, pages):
    memory = deque(maxlen=frames)
    page_faults = 0
    steps = []

    for i, page in enumerate(pages, 1):
        if page not in memory:
            page_faults += 1
            if len(memory) == frames:
                memory.popleft()
            memory.append(page)
            status = "MISS"
        else:
            status = "HIT"

        steps.append({
            'step': i,
            'page': page,
            'memory': list(memory),
            'status': status
        })

    return steps, page_faults

def lru_replacement(self, frames, pages):
    memory = []
    page_faults = 0

```

```
steps = []
```

```
for i, page in enumerate(pages, 1):
```

```
    if page not in memory:
```

```
        page_faults += 1
```

```
        if len(memory) == frames:
```

```
            memory.pop(0)
```

```
        memory.append(page)
```

```
        status = "MISS"
```

```
    else:
```

```
        memory.remove(page)
```

```
        memory.append(page)
```

```
        status = "HIT"
```

```
    steps.append({
```

```
        'step': i,
```

```
        'page': page,
```

```
        'memory': list(memory),
```

```
        'status': status
```

```
    })
```

```
return steps, page_faults
```

```
def run_simulation(self):
```

```
    try:
```

```
        # Clear previous results
```

```
        for item in self.table.get_children():
```

```
            self.table.delete(item)
```

```
    # Get input values
```

```
    pages = [int(x.strip()) for x in self.pages_entry.get().split(',')]

```

```
    frames = int(self.frames_entry.get())
```

```
    algorithm = self.algorithm.get()
```

```
    # Run simulation
```

```

if algorithm == "fifo":
    steps, faults = self.fifo_replacement(frames, pages)
else:
    steps, faults, = self.lru_replacement(frames, pages)

```

```

# Display results

```

```

for step in steps:
    memory_str = ','.join(map(str, step['memory']))
    self.table.insert("", 'end', values=(
        step['step'],
        step['page'],
        memory_str,
        step['status']
    ))

```

```

# Update stats

```

```

total_pages = len(pages)
hit_rate = ((total_pages - faults) / total_pages) * 100
stats_text = f"Total Pages: {total_pages}\n"
stats_text += f"Page Faults: {faults}\n"
stats_text += f"Hit Rate: {hit_rate:.1f}%"
self.stats_label.config(text=stats_text)

```

```

except ValueError as e:

```

```

    messagebox.showerror("Error", "Please enter valid numbers!")

```

```

except Exception as e:

```

```

    messagebox.showerror("Error", str(e))

```

```

def main():

```

```

    root = tk.Tk()
    app = MemoryMatrixApp(root)
    root.mainloop()

```

```

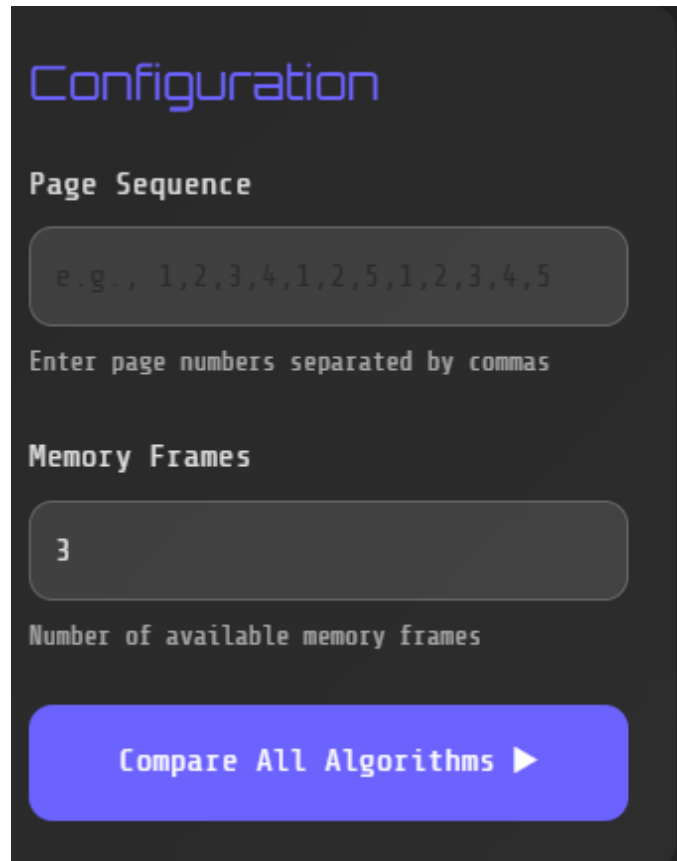
if __name__

```

CHAPTER 6

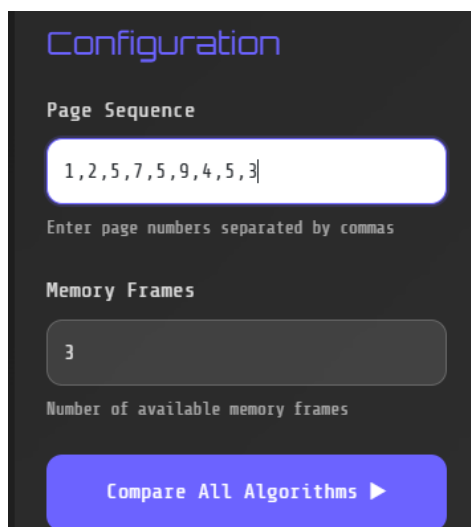
RESULTS

6.1 SCREENSHOTS



The screenshot shows a dark-themed configuration window titled "Configuration" in a light blue font. It contains two input sections. The first section, "Page Sequence", has a text input field with the example text "e.g., 1,2,3,4,1,2,5,1,2,3,4,5" and a label below it that says "Enter page numbers separated by commas". The second section, "Memory Frames", has a text input field containing the number "3" and a label below it that says "Number of available memory frames". At the bottom of the window is a large blue button with the text "Compare All Algorithms" and a right-pointing triangle icon.

Fig 1. Input Dialog box



The image shows a dark-themed configuration interface. At the top, the word 'Configuration' is written in a light blue font. Below it, the section 'Page Sequence' is labeled. A white input field contains the text '1,2,5,7,5,9,4,5,3'. Below the input field, a small grey text label reads 'Enter page numbers separated by commas'. The next section is 'Memory Frames', with a grey input field containing the number '3'. Below this field, another small grey text label reads 'Number of available memory frames'. At the bottom of the form is a large blue button with the text 'Compare All Algorithms' and a right-pointing triangle icon.

Configuration

Page Sequence

1,2,5,7,5,9,4,5,3

Enter page numbers separated by commas

Memory Frames

3

Number of available memory frames

Compare All Algorithms ►

Fig 2. Giving user input

FIFO				LRU				Optimal			
Step	Page	Memory	Status	Step	Page	Memory	Status	Step	Page	Memory	Status
1	1	1	MISS	1	1	1	MISS	1	1	1	MISS
2	2	1 2	MISS	2	2	1 2	MISS	2	2	1 2	MISS
3	5	1 2 5	MISS	3	5	1 2 5	MISS	3	5	1 2 5	MISS
4	7	2 5 7	MISS	4	7	2 5 7	MISS	4	7	2 5 7	MISS
5	5	2 5 7	HIT	5	5	2 5 7	HIT	5	5	2 5 7	HIT
6	9	5 7 9	MISS	6	9	5 7 9	MISS	6	9	5 7 9	MISS
7	4	7 9 4	MISS	7	4	5 9 4	MISS	7	4	5 9 4	MISS

Fig 3. Algorithm Comparison

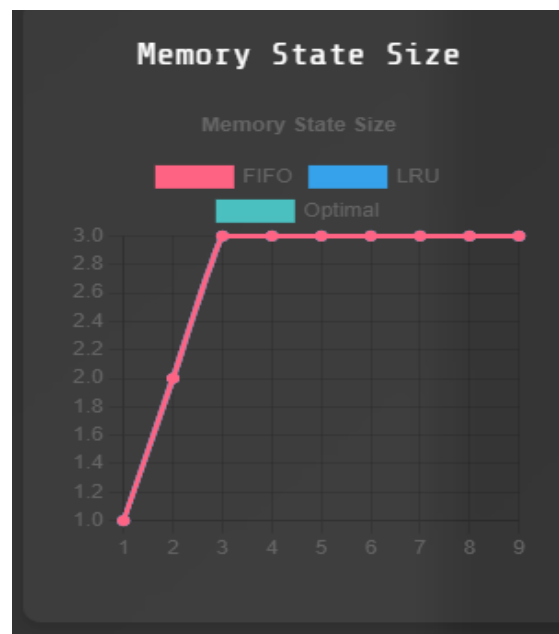


Fig 4. Memory State Size

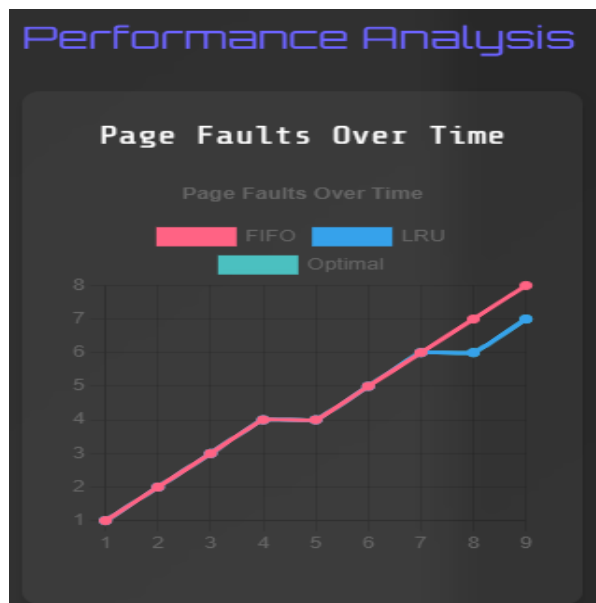


Fig 5. Page Faults

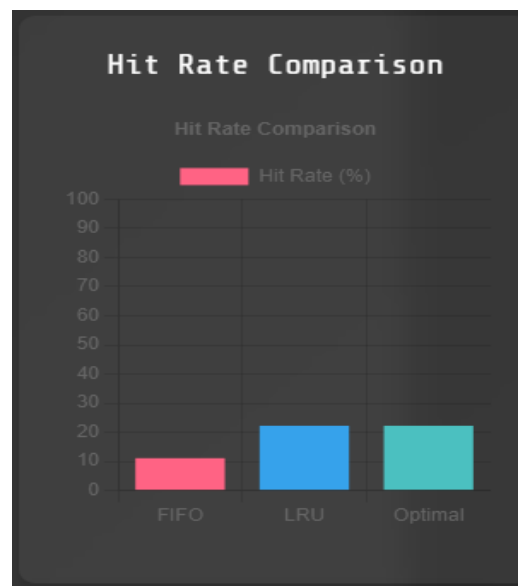


Fig 6. Hit Rate Comparison

CHAPTER 7

CONCLUSION

The Virtual Memory Manager Simulation project provides an insightful and interactive platform for exploring fundamental concepts of operating system memory management, focusing on page replacement algorithms like FIFO, LRU, and Optimal Replacement.

By combining theoretical principles with real-time graphical and tabular interfaces using Python's Flask and Tkinter frameworks, the project bridges the gap between textbook knowledge and practical application.

The simulation demonstrates the internal working of memory management, showcasing how different algorithms affect page fault rates and system performance. Through step-by-step memory state visualization and comparative performance metrics like hit rate and page faults, learners gain a deeper understanding of the trade-offs involved in memory management strategies.

Incorporating customizable page sequences, dynamic frame allocation, and algorithm selection, the project not only enhances educational experience but also serves as a powerful tool for instructors, students, and enthusiasts to experiment, visualize, and analyze virtual memory behavior.

Ultimately, this project fosters active learning and strengthens foundational concepts of virtual memory, page replacement strategies, and performance optimization in operating systems.

CHAPTER 8

FUTURE WORK

- **AI-Based Page Replacement Enhancements**

Implement machine learning models to dynamically predict and apply the most efficient page replacement strategy based on memory access patterns.

Real-Time System Integration

Connect the simulator with live system data (e.g., running processes memory access) for real-time virtual memory visualization.

- **Dynamic Page Replacement Algorithms**

Incorporate more advanced strategies such as Adaptive Replacement Cache (ARC) or machine learning-based page replacement policies.

- **Web-Based Deployment**

Extend the project with full deployment over cloud platforms using frameworks like Flask, Django, or FastAPI for broader accessibility.

- **Mobile Application Support**

Develop a mobile version to allow users to simulate virtual memory management on smartphones and tablets.

- **Cloud Hosting and Collaboration**

Host the simulator online, allowing multiple users to simulate and collaborate remotely.

- **Simulation of Multiprocessor Memory Management:**

Expand the simulator to support and visualize memory management in multi-core systems.

Gamification for Learning

Add challenge modes, quizzes, and achievements to make learning about memory management more engaging and interactive.

- **User Role Management**

Introduce role-based access (students, teachers, researchers) to manage simulation settings and track learning progress.

- **Data Export and Report Generation**

Enable exporting of simulation results, charts, and statistics into PDF or Excel formats for analysis and record-keeping.

- **Accessibility Features**

Add screen reader support, high-contrast themes, and keyboard navigation to make the application accessible to all users.

REFERENCES

1. William Stallings, (2018) “Operating Systems – Internals and Design Principles”, 9th Edition, Pearson.
2. Pavel Y., Alex I., Mark E., David A., (2017) “Windows Internal Part I - System Architecture, Processes, Memory Management and More”, 7th Edition, Microsoft Press.
3. Andrew S. Tanenbaum and Herbert Bos, (2016) “Modern Operating Systems”, 4th Edition, Pearson.