OPERATING SYSTEMS

PROJECT

Section: K23SA

TEACHER: ***Dr. Gurbinder Singh Brar***

Name: ***Shivam***

Reg_no.: ***12309409***

TOPIC: **Real-Time Process Monitoring Dashboard**

Repository name: **Resource_Monitoring_System**

Repository link: **https://github.com/007Edge/Resource_Monitoring_System.git**

LOVELY PROFESSIONAL UNIVERSITY

Real-Time Process Monitoring Dashboard

REAL-TIME PROCESS MONITORING DASHBOARD

The **Real-Time Process Monitoring Dashboard** is an advanced graphical interface designed to provide administrators with real-time insights into the performance and status of system processes. By continuously monitoring key metrics such as **process states, CPU usage, and memory consumption**, this dashboard enables proactive system management, helping to swiftly detect and resolve potential performance issues.

Key Features

1. **Real-Time System Insights**
    - Displays active processes and their states, ensuring administrators have an up-to-date view of system activity.
    - Continuously tracks CPU and memory utilization, allowing for efficient resource management.
2. **Intuitive Graphical Interface**
    - Provides interactive charts and visual indicators that simplify performance analysis.
    - Offers a customizable layout for administrators to focus on specific metrics based on their needs.
3. **Performance Analysis & Alerts**
    - Detects unusual spikes or anomalies in resource consumption, helping to identify potential system slowdowns or failures.
    - Sends real-time alerts when predefined thresholds are breached, ensuring timely intervention.
4. **Historical Data & Trend Analysis**
    - Maintains logs of past performance metrics to support detailed trend analysis.

    - Helps in forecasting potential system bottlenecks based on historical patterns.

5. **Process Control & Management**
    - Allows administrators to pause, resume, or terminate processes directly from the dashboard.
    - Provides insights into resource-hungry applications, helping optimize system efficiency.

Benefits

- **Enhanced System Performance**: Helps administrators maintain an optimized and well-managed system by tracking resource usage.
- **Proactive Issue Detection**: Enables rapid troubleshooting and problem resolution through real-time monitoring and alerts.
- **Improved Decision-Making**: Provides actionable insights based on historical data and trends, assisting in resource planning and system improvements.
- **Efficient Resource Allocation**: Ensures balanced CPU and memory usage to prevent system overload or inefficiency.

Conclusion

The **Real-Time Process Monitoring Dashboard** is an essential tool for administrators aiming to maintain a high-performance computing environment. By offering real-time insights, alerting mechanisms, and historical trend analysis, it significantly enhances system management and optimizes overall efficiency.

Would you like me to refine specific sections or add additional details?

Table of Contents

8. Future Enhancements

Real-Time Process Monitoring Dashboard: Detailed Explanation

1. Introduction

Efficient system monitoring plays a vital role in ensuring the **optimal performance and stability** of computing environments. The **Real-Time Process Monitoring Dashboard** is a **web-based application** designed to provide administrators with **real-time insights** into system resource usage, including **CPU, memory, disk, and network activity**.

This dashboard offers **a visual representation** of system performance, enabling administrators to analyze trends, detect anomalies, and ensure smooth operation. Additionally, it provides the ability to **manage running processes dynamically**, allowing administrators to **start, pause, and terminate processes** in real time.

2. Objectives

The primary goals of the Real-Time Process Monitoring Dashboard are:

- **Real-Time System Monitoring:** Continuously tracks and displays **system resource usage** to provide a clear overview of system health.
- **Graphical Representation:** Uses interactive **charts and visual indicators** to present **CPU, memory, disk, and network usage** in an easily interpretable format.
- **Process Management:** Allows administrators to **view and control active processes**, including the ability to terminate unresponsive or resource-heavy processes.
- **User-Friendly Interface:** Designed with a **web-based UI** to ensure ease of access and seamless navigation for system administrators.
- **Live Alerts for System Issues:** Implements a **real-time alert mechanism** to notify administrators about **critical issues**, such as excessive CPU or memory consumption, allowing for quick resolution.

3. Technologies Used

The Real-Time Process Monitoring Dashboard is built using **a combination of backend and frontend technologies** to ensure efficient data processing and visualization.

Backend Technologies:

- **Python (Flask, psutil, Flask-SocketIO):**
  - **Flask:** A lightweight web framework that powers the dashboard's backend.

    - **psutil:** A Python library that provides system monitoring capabilities, such as fetching CPU and memory usage statistics.
    - **Flask-SocketIO:** Enables real-time communication between the server and the client using WebSockets, ensuring seamless live updates.

Frontend Technologies:

- **HTML, CSS, JavaScript:** Used to design the dashboard's interface for displaying monitoring data effectively.
- **Chart.js:** A JavaScript library that provides dynamic and interactive charts for representing resource usage visually.
- **Socket.io:** Facilitates real-time updates between the client and server to keep monitoring data fresh without delays.

Essential Libraries & Dependencies:

The following Python libraries are used for backend development:

- Flask==2.2.2: Handles web application functionality and request routing.
- psutil==5.9.0: Enables real-time system monitoring by retrieving CPU, memory, and disk usage statistics.
- python-socketio==5.7.2: Supports WebSocket-based communication for instant data updates.
- eventlet==0.33.0: Provides asynchronous networking support, ensuring smooth real-time interactions.

4. System Design

The Real-Time Process Monitoring Dashboard is built using a modular approach, consisting of two main components:

1. **Backend Server (Python - Flask & SocketIO)**
    - The backend is responsible for gathering system resource data using psutil, a Python library that provides real-time metrics related to CPU, memory, disk, and network usage.
    - This data is **streamed to the frontend** using WebSockets (Flask-SocketIO), ensuring administrators receive updates in real time without needing manual page refreshes.
2. **Frontend (HTML, CSS, JavaScript)**

   - The frontend serves as a **visual interface**, presenting real-time system metrics using interactive graphs and dynamic tables.
   - Administrators can **monitor active processes** and **initiate management actions**, such as starting and terminating processes.

Data Flow:

- The **backend continuously collects** real-time system metrics and active process details.
- This data is **transmitted via WebSockets** to the frontend.
- The **frontend dynamically updates** its graphs and tables to reflect real-time changes without requiring a page reload.
- Users can **control system processes**, such as terminating or starting new processes, using API requests handled by Flask.

5. Implementation

Backend (server.py)

- Initializes a **Flask server** to handle requests and process system monitoring.
- Uses psutil to collect real-time **CPU, memory, disk, and network usage** data.
- Implements Flask-SocketIO to **stream live system metrics** to the frontend in real time.
- Provides **API endpoints** for process management, allowing users to **start or terminate** system processes.

Frontend (index.html, JavaScript)

- **Visualizes system metrics** using Chart.js, a JavaScript library for creating interactive graphs.
- Implements a **dynamic table** to display active running processes.
- Provides **search and sorting functionalities**, enabling administrators to filter and locate specific processes efficiently.
- Sends **requests to the backend** via API calls to manage system processes.

6. Features

- **Real-Time Performance Monitoring:**
   - Displays live graphs for **CPU, memory, disk, and network usage** to help administrators track system health dynamically.

- **Process Management:**
  - Allows users to **start or terminate processes** directly from the dashboard.
  - Provides visibility into **process states and resource consumption**.
- **Alerts & Notifications:**
  - Detects **high CPU or memory usage** and sends notifications, helping administrators react quickly to potential issues.
- **Interactive UI:**
  - Supports **search, sorting, and filtering** for active processes, improving system navigation and usability.
- **Web-Based Dashboard:**
  - Designed to be **accessible from any device** with a web browser, ensuring flexibility in system monitoring.

7. Conclusion

The **Real-Time Process Monitoring Dashboard** is a robust and efficient tool designed to **enhance system monitoring and management**. By providing **real-time visual insights** into **CPU, memory, disk, and network usage**, it empowers administrators to **maintain system performance** and **quickly detect potential bottlenecks or issues**.

With **interactive graphs, dynamic process management, live alerts, and a web-based interface**, the dashboard simplifies **system resource tracking** and **optimizes workflow efficiency**. Its **backend, built with Flask and SocketIO**, ensures **seamless data transmission**, while the **frontend, powered by JavaScript and Chart.js**, delivers an intuitive user experience.

Looking ahead, potential **future enhancements**, such as **user authentication, historical data storage, process prioritization, and multi-system monitoring**, can further improve its capabilities and scalability.

Ultimately, this dashboard serves as a **powerful asset** for system administrators, providing the necessary tools to ensure **real-time monitoring, proactive issue detection, and efficient system management**, all within an easily accessible web-based interface.

References

- Flask Documentation: https://flask.palletsprojects.com/
- psutil Documentation: https://psutil.readthedocs.io/
- Chart.js Documentation: https://www.chartjs.org/
- Flask-SocketIO Documentation: https://flask-socketio.readthedocs.io/

HERE ARE SOME SNAPSHOT'S OF SOURCE CODE

```python
from flask import Flask, render_template, request, jsonify
from flask_socketio import SocketIO
import psutil
import time
import os
import threading

app = Flask(__name__)
socketio = SocketIO(app)

# Store historical data (simple in-memory for now)
history = {'cpu': [], 'memory': [], 'disk': [], 'network': []}

# Gather detailed system and process data
def get_system_data():
    processes = []
    for proc in psutil.process_iter(['pid', 'name', 'cpu_percent', 'memory_percent', 'num_threads', 'io_counters']):
        try:
            io = proc.io_counters() if hasattr(proc, 'io_counters') else None
            processes.append({
                'pid': proc.info['pid'],
                'name': proc.info['name'],
                'cpu': proc.info['cpu_percent'],
                'memory': proc.info['memory_percent'],
                'threads': proc.info['num_threads'],
                'disk_read': io.read_bytes / 1024 / 1024 if io else 0,  # MB
                'disk_write': io.write_bytes / 1024 / 1024 if io else 0  # MB
            })
        except (psutil.NoSuchProcess, psutil.AccessDenied):
            continue

    net = psutil.net_io_counters()
    disk = psutil.disk_io_counters()

```

```python
    data = {
        'cpu_usage': psutil.cpu_percent(interval=1),
        'memory_usage': psutil.virtual_memory().percent,
        'disk_usage': psutil.disk_usage('/').percent,
        'net_sent': net.bytes_sent / 1024 / 1024,  # MB
        'net_recv': net.bytes_recv / 1024 / 1024,  # MB
        'disk_read': disk.read_bytes / 1024 / 1024 if disk else 0,  # MB
        'disk_write': disk.write_bytes / 1024 / 1024 if disk else 0,  # MB
        'uptime': time.time() - psutil.boot_time(),
        'processes': sorted(processes, key=lambda x: x['cpu'], reverse=True)[:15]  # Top 15 by CPU
    }

    # Store historical data (limit to 60 entries ~ 1 minute)
    for key in ['cpu', 'memory', 'disk', 'network']:
        history[key].append(data[f'{key}_usage'] if key != 'network' else data['net_sent'] + data['net_recv'])
        if len(history[key]) > 60:
            history[key].pop(0)

    return data

# Route for the dashboard
@app.route('/')
def index():
    return render_template('index.html')
```

```python
# API to kill a process
@app.route('/kill/<int:pid>', methods=['POST'])
def kill_process(pid):
    try:
        proc = psutil.Process(pid)
        proc.terminate()
        return jsonify({'status': 'success', 'message': f'Process {pid} terminated'})
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)})

# API to start a process (example: notepad on Windows)
@app.route('/start', methods=['POST'])
def start_process():
    try:
        process_name = request.json.get('name', 'notepad.exe')  # Default to notepad
        os.startfile(process_name) if os.name == 'nt' else os.system(f'{process_name} &')
        return jsonify({'status': 'success', 'message': f'Started {process_name}'})
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)})

# Background task for real-time updates
def background_task():
    while True:
        data = get_system_data()
        socketio.emit('update', data)
        # Check for alerts
        if data['cpu_usage'] > 80 or data['memory_usage'] > 80:
            socketio.emit('alert', {'message': f"High usage detected! CPU: {data['cpu_usage']}%, Memory: {data['memory_usage']}%"})
        time.sleep(1)

@socketio.on('connect')
def handle_connect():
    socketio.start_background_task(background_task)

if __name__ == '__main__':
    socketio.run(app, debug=True, host='0.0.0.0', port=5000)
```

```
            // Table filtering
            function filterTable() {
                const input = document.getElementById('search').value.toLowerCase();
                const table = document.getElementById('processTable');
                const tr = table.getElementsByTagName('tr');
                for (let i = 1; i < tr.length; i++) {
                    const td = tr[i].getElementsByTagName('td');
                    let match = false;
                    for (let j = 0; j < td.length; j++) {
                        if (td[j] && td[j].textContent.toLowerCase().indexOf(input) > -1) {
                            match = true;
                            break;
                        }
                    }
                    tr[i].style.display = match ? '' : 'none';
                }
            }
        </script>
    </body>
    </html>
```

```

    <div id="performance" class="tab-content active">
        <div class="container">
            <div><h2>CPU Usage</h2><canvas id="cpuChart"></canvas></div>
            <div><h2>Memory Usage</h2><canvas id="memoryChart"></canvas></div>
            <div><h2>Disk Usage</h2><canvas id="diskChart"></canvas></div>
            <div><h2>Network Usage</h2><canvas id="networkChart"></canvas></div>
        </div>
        <p><strong>Uptime:</strong> <span id="uptime">0</span> seconds</p>
    </div>

    <div id="processes" class="tab-content">
        <input type="text" id="search" placeholder="Search processes..." onkeyup="filterTable()">
        <button onclick="startProcess()">Start New Process</button>
        <table id="processTable">
            <thead>
                <tr>
                    <th onclick="sortTable(0)">PID</th>
                    <th onclick="sortTable(1)">Name</th>
                    <th onclick="sortTable(2)">CPU (%)</th>
                    <th onclick="sortTable(3)">Memory (%)</th>
                    <th onclick="sortTable(4)">Threads</th>
                    <th onclick="sortTable(5)">Disk Read (MB)</th>
                    <th onclick="sortTable(6)">Disk Write (MB)</th>
                    <th>Actions</th>
                </tr>
            </thead>
            <tbody></tbody>
        </table>
    </div>
```

```
        socket.on('update', (data) => {
            const tbody = document.querySelector('#processTable tbody');
            tbody.innerHTML = '';
            data.processes.forEach(proc => {
                tbody.innerHTML += `<tr>
                    <td>${proc.pid}</td>
                    <td>${proc.name}</td>
                    <td>${proc.cpu.toFixed(2)}</td>
                    <td>${proc.memory.toFixed(2)}</td>
                    <td>${proc.threads}</td>
                    <td>${proc.disk_read.toFixed(2)}</td>
                    <td>${proc.disk_write.toFixed(2)}</td>
                    <td><button onclick="killProcess(${proc.pid})">Kill</button></td>
                </tr>`;
            });
        });

        socket.on('alert', (data) => {
            document.getElementById('alerts').textContent = data.message;
            setTimeout(() => document.getElementById('alerts').textContent = '', 5000);
        });

        // Process control functions
        function killProcess(pid) {
            fetch(`/kill/${pid}`, { method: 'POST' })
                .then(res => res.json())
                .then(data => alert(data.message));
        }
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Advanced Process Monitoring Dashboard</title>
    <script src="https://cdn.socket.io/4.5.0/socket.io.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; background: #f5f5f5; }
        .tabs { overflow: hidden; border-bottom: 1px solid #ccc; }
        .tab-button { float: left; padding: 10px 20px; cursor: pointer; background: #ddd; }
        .tab-button.active { background: #fff; font-weight: bold; }
        .tab-content { display: none; padding: 20px; background: #fff; border: 1px solid #ccc; }
        .tab-content.active { display: block; }
        .container { display: flex; flex-wrap: wrap; gap: 20px; }
        canvas { max-width: 400px; }
        table { border-collapse: collapse; width: 100%; max-width: 800px; }
        th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }
        th { background-color: #f2f2f2; cursor: pointer; }
        button { padding: 5px 10px; cursor: pointer; }
        #alerts { color: red; font-weight: bold; }
    </style>
</head>
<body>
    <h1>Advanced Process Monitoring Dashboard</h1>
    <div class="tabs">
        <div class="tab-button active" onclick="openTab('performance')">Performance</div>
        <div class="tab-button" onclick="openTab('processes')">Processes</div>
    </div>
```

```
    <div id="alerts"></div>

    <script>
        const socket = io();
        const charts = {};
        const chartConfigs = {
            'cpu': { id: 'cpuChart', label: 'CPU Usage (%)', color: 'blue' },
            'memory': { id: 'memoryChart', label: 'Memory Usage (%)', color: 'green' },
            'disk': { id: 'diskChart', label: 'Disk Usage (%)', color: 'purple' },
            'network': { id: 'networkChart', label: 'Network Usage (MB)', color: 'orange' }
        };

        // Initialize charts
        Object.keys(chartConfigs).forEach(key => {
            const cfg = chartConfigs[key];
            charts[key] = new Chart(document.getElementById(cfg.id).getContext('2d'), {
                type: 'line',
                data: { labels: [], datasets: [{ label: cfg.label, data: [], borderColor: cfg.color, fill: false }] },
                options: { scales: { y: { min: 0, max: key === 'network' ? null : 100 } } }
            });
        });
```

```
        function startProcess() {
            const name = prompt('Enter process name (e.g., notepad.exe):');
            if (name) {
                fetch('/start', {
                    method: 'POST',
                    headers: { 'Content-Type': 'application/json' },
                    body: JSON.stringify({ name })
                })
                .then(res => res.json())
                .then(data => alert(data.message));
            }
        }

        // Tab functionality
        function openTab(tabName) {
            document.querySelectorAll('.tab-content').forEach(tab => tab.classList.remove('active'));
            document.querySelectorAll('.tab-button').forEach(btn => btn.classList.remove('active'));
            document.getElementById(tabName).classList.add('active');
            document.querySelector(`[onclick="openTab('${tabName}')"]`).classList.add('active');
        }
```

```
function sortTable(n) {
    const table = document.getElementById('processTable');
    let rows, switching = true, i, shouldSwitch, dir = 'asc', switchCount = 0;
    while (switching) {
        switching = false;
        rows = table.rows;
        for (i = 1; i < rows.length - 1; i++) {
            shouldSwitch = false;
            const x = rows[i].getElementsByTagName('TD')[n];
            const y = rows[i + 1].getElementsByTagName('TD')[n];
            const xVal = isNaN(x.innerHTML) ? x.innerHTML.toLowerCase() : parseFloat(x.innerHTML);
            const yVal = isNaN(y.innerHTML) ? y.innerHTML.toLowerCase() : parseFloat(y.innerHTML);
            if (dir === 'asc' ? xVal > yVal : xVal < yVal) {
                shouldSwitch = true;
                break;
            }
        }
        if (shouldSwitch) {
            rows[i].parentNode.insertBefore(rows[i + 1], rows[i]);
            switching = true;
            switchCount++;
        } else if (switchCount === 0 && dir === 'asc') {
            dir = 'desc';
            switching = true;
        }
    }
}
```

```
// Real-time updates
socket.on('update', (data) => {
    const time = new Date().toLocaleTimeString();
    ['cpu', 'memory', 'disk'].forEach(key => {
        const chart = charts[key];
        chart.data.labels.push(time);
        chart.data.datasets[0].data.push(data[`${key}_usage`]);
        if (chart.data.labels.length > 60) {
            chart.data.labels.shift();
            chart.data.datasets[0].data.shift();
        }
        chart.update();
    });

    charts['network'].data.labels.push(time);
    charts['network'].data.datasets[0].data.push(data.net_sent + data.net_recv);
    if (charts['network'].data.labels.length > 60) {
        charts['network'].data.labels.shift();
        charts['network'].data.datasets[0].data.shift();
    }
    charts['network'].update();

    document.getElementById('uptime').textContent = Math.floor(data.uptime);
```