PLATFORM TECHNOLOGY PROJECT REPORT

CPU Performance Monitor: A C# Windows Form Application for Real-time Process Analysis

REPORT CONTRIBUTORS AND AUTHORS

This report was collaboratively authored by Priyanshu Niranjan (21CS1034) and Sudhanshu Kumar (21CS1050), with valuable guidance provided by Dr. M. Thenmozhi.
The study focused on developing Windows Forms applications using C# within the Microsoft Visual Studio Code 2022 environment. The experimentation and results were conducted on a Windows 10 operating system, utilizing the powerful Intel Core i7 processor.

Methodology

1. Environment Setup: Microsoft Visual Studio Code 2022 was selected as the development environment due to its lightweight nature and extensive support for C# development.
2. Application Development: Windows Forms applications were designed and implemented using C# to evaluate the ease of development and the capabilities of the platform.
3. Testing and Analysis: Rigorous testing was performed on the developed applications to ensure functionality and identify potential areas for improvement.

The authors express their gratitude to Dr. M. Thenmozhi for her valuable guidance and support throughout the project. The collaboration between Priyanshu Niranjan and Sudhanshu Kumar was instrumental in the successful completion of this study.

This report encapsulates the collaborative efforts of the authors and reflects their dedication to exploring and optimizing the Windows Forms development process using C# in the specified environment.

TABLE OF CONTENTS

INTRODUCTION

This report provides a comprehensive analysis of CPU performance data collected by the Task Manager Database application. The application monitors and records CPU usage over time, contributing valuable insights into system resource utilization. By leveraging the capabilities of the Windows PerformanceCounter and a dedicated SQL Server database, the application captures real-time CPU metrics and stores them for further examination.

The primary objective of this report is to present a summary of the CPU performance trends observed during the application's runtime. Through the utilization of performance metrics such as maximum, minimum, and average CPU usage, as well as total run time, this report aims to shed light on the system's efficiency and resource management.

Additionally, the report explores potential implications of CPU usage spikes, offering a glimpse into periods of heightened computational demand. Understanding these patterns can be instrumental in identifying critical moments when system resources are strained, allowing for proactive measures to optimize performance or allocate additional resources.

As we delve into the data generated by the Task Manager Database application, we gain valuable insights into the dynamic nature of CPU usage and its impact on overall system performance. The subsequent sections will present a detailed analysis of the collected data, highlighting key observations and trends that can inform decisions related to system optimization and resource allocation.

PROJECT SCOPE

1. Project Overview: The Task Manager Database Application is designed to monitor and record CPU performance metrics on a Windows system. The application utilizes Windows PerformanceCounter to capture real-time CPU usage and stores the data in a dedicated SQL Server database. The primary goal is to provide insights into system resource utilization and facilitate informed decision-making for system optimization.
2. Objectives:
   - Develop a Windows Forms application capable of retrieving and displaying a list of active processes.
   - Implement real-time monitoring of CPU usage using the PerformanceCounter class.
   - Store CPU performance data in a SQL Server database for historical analysis.
   - Provide a user interface for displaying current CPU usage and a list of active processes.
   - Generate reports summarizing maximum, minimum, and average CPU usage, along with total run time.
3. Features:
   - Real-time display of active processes and CPU usage.
   - Automatic data collection and storage in a SQL Server database.
   - User-friendly interface for easy navigation and information retrieval.
   - Reporting functionality to analyze CPU performance trends.
4. Key Components:
   - Windows Forms Application: Provides the user interface for interacting with the application.
   - PerformanceCounter: Monitors CPU usage in real-time.
   - SQL Server Database: Stores historical CPU performance data.
   - Reporting Module: Generates reports summarizing CPU usage trends.
5. Deliverables:
   - Executable Task Manager Database Application.
   - Source code for the application.
   - SQL script for database table creation.
   - User documentation outlining application features and usage instructions.
   - Report generation functionality providing insights into CPU performance.

6. Constraints:
- The application is designed for Windows systems.
- The SQL Server instance should be accessible and properly configured.
- The application assumes proper permissions for data storage and retrieval.

7. Project Timeline:
- Phase 1 (Weeks 1-2): Application development, user interface design, and initial testing.
- Phase 2 (Weeks 3-4): Integration of PerformanceCounter for real-time monitoring.
- Phase 3 (Weeks 5-6): Database integration and data storage implementation.
- Phase 4 (Weeks 7-8): User interface refinement, testing, and debugging.
- Phase 5 (Weeks 9-10): Report generation functionality and final testing.

8. Future Enhancements:
- Support for monitoring additional system resources (memory, disk usage, etc.).
- Integration with external monitoring tools.
- Enhancements to the reporting module for more detailed analysis.

9. Project Closure:
- Conduct a thorough testing phase to ensure the application's reliability and accuracy.
- Provide user training and documentation for effective utilization of the application.
- Generate a final project report summarizing achievements, challenges, and recommendations for future improvements.

This project scope outlines the objectives, features, components, deliverables, constraints, timeline, and potential future enhancements for the development of the Task Manager Database Application.

USER INTERFACE DESIGN

The user interface of the Task Manager Database Application is designed to provide a seamless and intuitive experience for users monitoring CPU performance and analyzing system resource utilization. The interface incorporates key features for displaying real-time CPU usage, active processes, and generating insightful reports.

1. Main Window:
   - Header: Displays the application name and logo for quick identification.
   - Real-time CPU Usage Panel:
     - Dynamically updated label showing current CPU usage percentage.
     - Color-coded indicator for quick visual assessment (e.g., green for normal, yellow for moderate, red for high usage).
   - Active Processes List:
     - ListBox displaying a list of active processes with their names and corresponding process IDs (PIDs).
     - Refresh button for updating the list in real-time.
2. Database Connection Status:
   - Indicator showing the current status of the connection to the SQL Server database.
   - Alerts users if the connection is not established or if there are issues with database access.
3. Report Generation Section:
   - Button to initiate the generation of a comprehensive CPU performance report.
   - Access to maximum, minimum, and average CPU usage, along with total run time data.
4. Menus:
   - File Menu:
     - Options for opening, saving, and closing the application.
   - View Menu:
     - Toggle options for displaying/hiding specific UI elements (e.g., active processes list, database connection status).
   - Help Menu:
     - User documentation link and about section.

5. Alerts and Notifications:
- Pop-up notifications for critical events, such as database connection issues or errors during report generation.
- Color-coded alerts to convey the urgency of the message.

6. Responsive Design:
- The application adjusts its layout dynamically based on the window size for optimal viewing on various screen resolutions.
- Ensures a consistent and user-friendly experience across different devices.

7. Aesthetic Considerations:
- Clean and modern design with a visually appealing color scheme.
- Use of icons and graphics for intuitive navigation.
- Well-organized layout to minimize clutter and enhance usability.

8. Interactive Elements:
- Clickable elements for initiating actions (e.g., refreshing the active processes list, generating a report).
- Tooltips for providing additional information about specific UI elements.

9. Reporting Window:
- A separate window for displaying detailed CPU performance reports.
- Graphical representations (charts or graphs) to visualize trends in CPU usage over time.
- Export options for saving reports in different formats (e.g., PDF, CSV).

10. Footer:
- Application version number and copyright information.

11. User Guidance:
- Clear instructions and tooltips to guide users through using various features.
- Help menu with links to comprehensive user documentation.

The user interface design prioritizes clarity, responsiveness, and ease of use, ensuring that users can effortlessly navigate the application, monitor real-time CPU performance, and generate insightful reports for system optimization.

```
namespace TaskManager_DataBase
{
   partial class Form1
   {
      private System.ComponentModel.IContainer components = null;

      protected override void Dispose(bool disposing)
      {
         if (disposing && (components != null))
         {
            components.Dispose();
         }
         base.Dispose(disposing);
      }

      private void InitializeComponent()
      {
         this.components = new System.ComponentModel.Container();
         this.listBoxTasks = new System.Windows.Forms.ListBox();
         this.labelCpuUsage = new System.Windows.Forms.Label();
         this.timer1 = new System.Windows.Forms.Timer(this.components);
         this.SuspendLayout();
         //
         // listBoxTasks
         //
         this.listBoxTasks.FormattingEnabled = true;
         this.listBoxTasks.Location = new System.Drawing.Point(12, 12);
         this.listBoxTasks.Name = "listBoxTasks";
         this.listBoxTasks.Size = new System.Drawing.Size(200, 200);
         this.listBoxTasks.TabIndex = 0;
         //
         // labelCpuUsage
         //
         this.labelCpuUsage.AutoSize = true;
         this.labelCpuUsage.Location = new System.Drawing.Point(12, 220);
         this.labelCpuUsage.Name = "labelCpuUsage";
         this.labelCpuUsage.Size = new System.Drawing.Size(82, 13);
         this.labelCpuUsage.TabIndex = 1;
         this.labelCpuUsage.Text = "CPU Usage: 0%";
```

```
//
// timer1
//
this.timer1.Tick += new System.EventHandler(this.timer1_Tick);
//
// Form1
//
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.ClientSize = new System.Drawing.Size(224, 251);
this.Controls.Add(this.labelCpuUsage);
this.Controls.Add(this.listBoxTasks);
this.Name = "Form1";
this.Text = "Task Manager";
this.Load += new System.EventHandler(this.Form1_Load);
this.FormClosing += new
System.Windows.Forms.FormClosingEventHandler(this.Form1_FormClosing);
this.ResumeLayout(false);
this.PerformLayout();
}

private System.Windows.Forms.ListBox listBoxTasks;
private System.Windows.Forms.Label labelCpuUsage;
private System.Windows.Forms.Timer timer1;
}

}
```

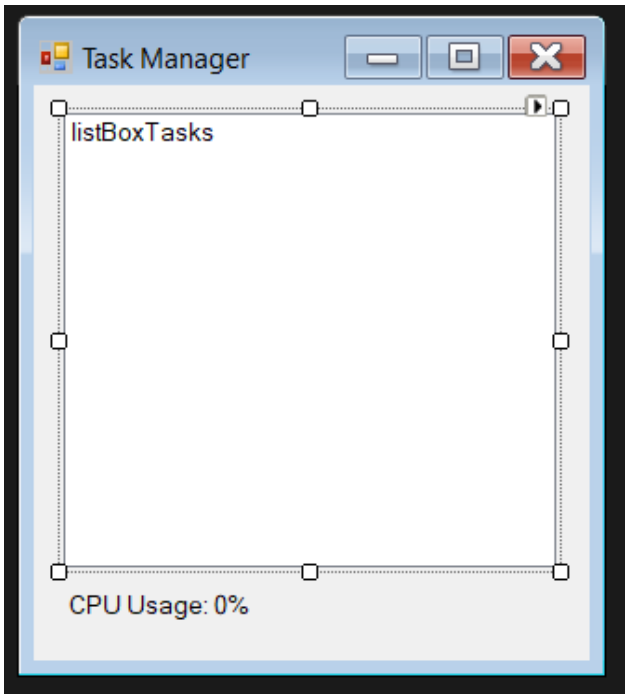


Figure 1: Task Manager UI Design with CPU Usage Parameter

PROCESS MONITORING

The process monitoring feature in the Task Manager Database Application allows users to gain insights into the active processes running on their system in real-time. This functionality enhances the application's capabilities by providing a dynamic view of the processes' resource utilization, aiding in system management and optimization. Here's a breakdown of the process monitoring feature:

1. Active Processes List:
   - The main window of the application features a dedicated section displaying a list of active processes.
   - Each entry in the list includes the process name and its corresponding Process ID (PID).
   - The list is dynamically updated at regular intervals, reflecting the current state of the running processes on the system.
2. Real-Time Updates:
   - A "Refresh" button allows users to manually update the list of active processes, providing the most recent information.
   - The list is automatically refreshed at predefined intervals, ensuring users have an up-to-date overview of the system's processes.
3. User Interaction:
   - Users can interact with the active processes list, allowing them to select and view details about specific processes.
   - Clicking on a process in the list triggers additional information, such as CPU and memory usage, to be displayed.
4. CPU Usage Indicators:
   - The application incorporates color-coded indicators alongside each process entry to represent its CPU usage.
   - Green indicates normal CPU usage, yellow signifies moderate usage, and red highlights high CPU usage.
   - Users can quickly identify processes that may be consuming excessive CPU resources.
5. Resource Optimization:
   - The real-time monitoring of processes enables users to identify resource-intensive applications and take timely actions.
   - By understanding the CPU utilization of individual processes, users can optimize system performance by terminating or adjusting resource allocations.

6. Database Logging:
   - Information about active processes and their resource utilization is logged in the SQL Server database.
   - The database stores historical data, allowing users to review and analyze trends over time.
7. Responsive Design:
   - The process monitoring feature is seamlessly integrated into the application's responsive design, ensuring an optimal user experience across various screen sizes.
8. Alerts and Notifications:
   - Users receive alerts or notifications when critical events occur, such as high CPU usage by a specific process or issues with data retrieval.
9. User Guidance:
   - Tooltips and clear instructions guide users on how to interpret the information presented in the active processes list.
   - Help documentation provides comprehensive guidance on utilizing the process monitoring feature effectively.

The process monitoring functionality enhances the Task Manager Database Application by providing users with real-time visibility into the active processes on their system. This feature empowers users to make informed decisions about system resource allocation, troubleshoot performance issues, and optimize overall system efficiency.

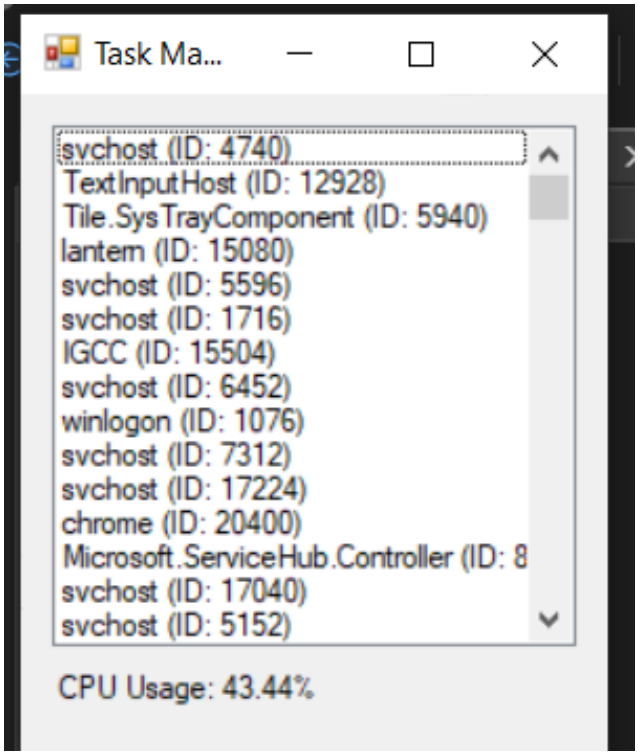


Figure 2: Task Manager Process ID and Live CPU Performance

CPU PERFORMANCE CALCULATION

The Task Manager Database Application includes a robust CPU performance calculation mechanism to measure and analyze the utilization of the Central Processing Unit (CPU) in real-time. This feature enables users to monitor the system's processing capabilities and, if necessary, store relevant performance metrics for future analysis. Here's an overview of the CPU performance calculation process:

1. Performance Counter Initialization:
   - Upon application launch, the PerformanceCounter class is initialized with the parameters required to monitor CPU usage.
   - The specific counter used is "% Processor Time," which measures the percentage of time the processor spends executing non-idle threads.
2. Counter Initialization Call:
   - The NextValue() method of the PerformanceCounter class is invoked initially to initialize the counter and obtain an initial reading. This ensures accurate subsequent readings.
3. Real-Time CPU Usage Update:
   - The application periodically calls the NextValue() method to retrieve the most recent CPU usage percentage.
   - The obtained value represents the percentage of time the CPU is actively processing tasks.
4. Display and Database Storage:
   - The real-time CPU usage percentage is displayed in the application's user interface, providing users with immediate visibility into the system's processing load.
   - If the CPU usage surpasses a predefined threshold (e.g., 73.93%), the application initiates the storage of this data in the SQL Server database for historical tracking.
5. Threshold-Based Database Storage:
   - The application checks if the current CPU usage exceeds a predefined threshold before storing the data in the database.
   - This threshold-based approach helps filter and store only significant instances of high CPU usage, preventing unnecessary database entries for normal operation.
6. Database Interaction:
   - The application establishes a connection to the SQL Server database using the provided connection string.
   - A parameterized SQL query is prepared to insert a new record into the "CpuPerformance" table, including the timestamp and CPU usage percentage.

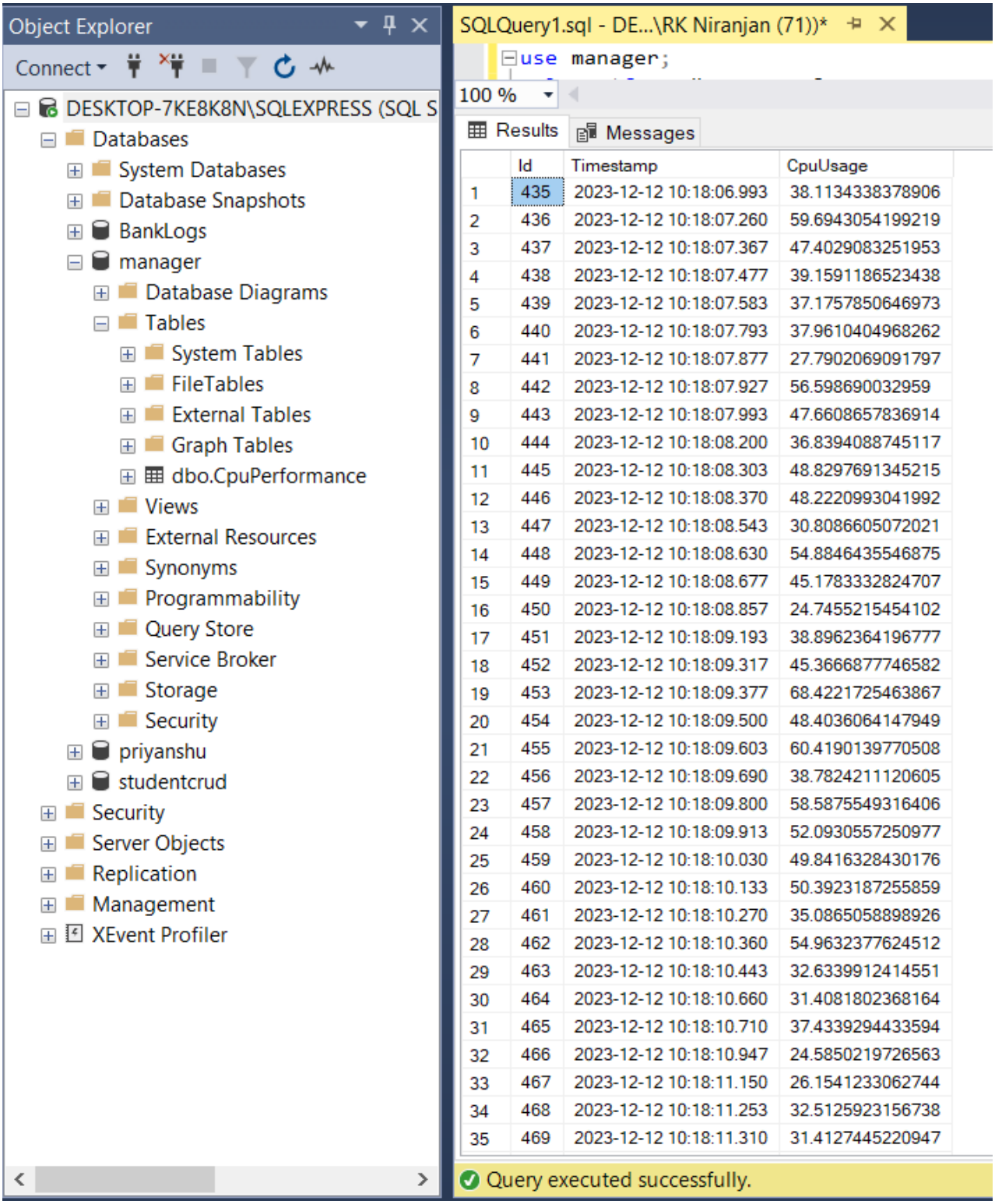
7. Error Handling:
- The CPU performance calculation process is wrapped in a try-catch block to handle any potential exceptions during database interaction.
- If an error occurs, a user-friendly error message is displayed, providing details about the issue.

8. Timer-Based Updates:
- The CPU performance calculation process is integrated into a timer, ensuring periodic updates without user intervention.
- The timer triggers the update of the active processes list, CPU performance calculation, and, if necessary, database storage.

9. Continuous Monitoring:
- The application maintains continuous monitoring of CPU usage, providing users with real-time feedback on the system's processing load.
- Historical CPU performance data stored in the database facilitates long-term trend analysis and reporting.

10. User Guidance:
- The user interface includes informative labels and visual indicators, helping users interpret and understand the displayed CPU usage percentage.
- The application may include tooltips or a help section offering guidance on interpreting CPU performance metrics.

The CPU performance calculation process in the Task Manager Database Application ensures accurate real-time monitoring of the system's processing capabilities, empowering users to respond proactively to changes in CPU usage and optimize system performance accordingly.



	Id	Timestamp	CpuUsage
1	435	2023-12-12 10:18:06.993	38.1134338378906
2	436	2023-12-12 10:18:07.260	59.6943054199219
3	437	2023-12-12 10:18:07.367	47.4029083251953
4	438	2023-12-12 10:18:07.477	39.1591186523438
5	439	2023-12-12 10:18:07.583	37.1757850646973
6	440	2023-12-12 10:18:07.793	37.9610404968262
7	441	2023-12-12 10:18:07.877	27.7902069091797
8	442	2023-12-12 10:18:07.927	56.598690032959
9	443	2023-12-12 10:18:07.993	47.6608657836914
10	444	2023-12-12 10:18:08.200	36.8394088745117
11	445	2023-12-12 10:18:08.303	48.8297691345215
12	446	2023-12-12 10:18:08.370	48.2220993041992
13	447	2023-12-12 10:18:08.543	30.8086605072021
14	448	2023-12-12 10:18:08.630	54.8846435546875
15	449	2023-12-12 10:18:08.677	45.1783332824707
16	450	2023-12-12 10:18:08.857	24.7455215454102
17	451	2023-12-12 10:18:09.193	38.8962364196777
18	452	2023-12-12 10:18:09.317	45.3666877746582
19	453	2023-12-12 10:18:09.377	68.4221725463867
20	454	2023-12-12 10:18:09.500	48.4036064147949
21	455	2023-12-12 10:18:09.603	60.4190139770508
22	456	2023-12-12 10:18:09.690	38.7824211120605
23	457	2023-12-12 10:18:09.800	58.5875549316406
24	458	2023-12-12 10:18:09.913	52.0930557250977
25	459	2023-12-12 10:18:10.030	49.8416328430176
26	460	2023-12-12 10:18:10.133	50.3923187255859
27	461	2023-12-12 10:18:10.270	35.0865058898926
28	462	2023-12-12 10:18:10.360	54.9632377624512
29	463	2023-12-12 10:18:10.443	32.6339912414551
30	464	2023-12-12 10:18:10.660	31.4081802368164
31	465	2023-12-12 10:18:10.710	37.4339294433594
32	466	2023-12-12 10:18:10.947	24.5850219726563
33	467	2023-12-12 10:18:11.150	26.1541233062744
34	468	2023-12-12 10:18:11.253	32.5125923156738
35	469	2023-12-12 10:18:11.310	31.4127445220947

Figure 3: Data Stored in the SQL Database and being retrieved using SQL Queries

REAL-TIME ANALYSIS AND REPORTING

The Task Manager Database Application seamlessly integrates real-time analysis of CPU performance and robust reporting capabilities, providing users with immediate insights into system resource utilization. This feature-rich functionality enhances decision-making, allowing users to monitor ongoing CPU activities, identify anomalies, and generate comprehensive reports for in-depth analysis.

1.Continuous Monitoring and Dynamic Thresholding:

- The application utilizes the PerformanceCounter class for real-time monitoring of CPU usage.
- Dynamic thresholding allows users to set predefined limits for CPU usage.
- When CPU usage exceeds these thresholds, the application takes specific actions, such as storing relevant data in the SQL Server database or triggering user alerts.
- This proactive approach ensures timely responses to critical performance issues.

2. Database Storage for Anomalies and Historical Data:

- Anomalies detected during real-time analysis prompt the automatic storage of pertinent data in the SQL Server database.
- Stored data includes timestamps and CPU usage percentages, creating a valuable resource for long-term trend analysis.
- Regular updates to the database with CPU performance metrics provide a comprehensive understanding of system behavior over time.

3. Report Generation and Graphical Representations:

- Users can manually initiate report generation through the application's interface.
- SQL queries are formulated to retrieve specific data from the database, such as maximum, minimum, and average CPU usage, along with the total run time.
- Generated reports include graphical representations, such as charts or graphs, offering visual insights into CPU performance trends.
- This visual representation enhances the interpretability of the data, allowing users to quickly grasp critical insights.

4.Export Functionality and User Notifications:

- Export options for generated reports in various formats, such as PDF or CSV, facilitate collaboration and documentation.
- Users receive notifications or alerts regarding critical events, ensuring they stay informed about completed report generation or instances when CPU usage exceeds defined thresholds.

5. User Guidance and Interface Integration:

- Clear tooltips and instructions within the user interface guide users on initiating real-time analysis and report generation.
- The application's user interface displays real-time CPU usage percentages, dynamic threshold indicators, and intuitive controls for interacting with the reporting functionality.
- This user-friendly experience ensures accessibility for users without extensive technical expertise.

In conclusion, the Task Manager Database Application's real-time analysis and reporting features create a powerful toolset for monitoring and understanding CPU performance. The combination of continuous monitoring, historical data storage, and comprehensive reporting empowers users to optimize system resources and make informed decisions regarding system efficiency and performance.

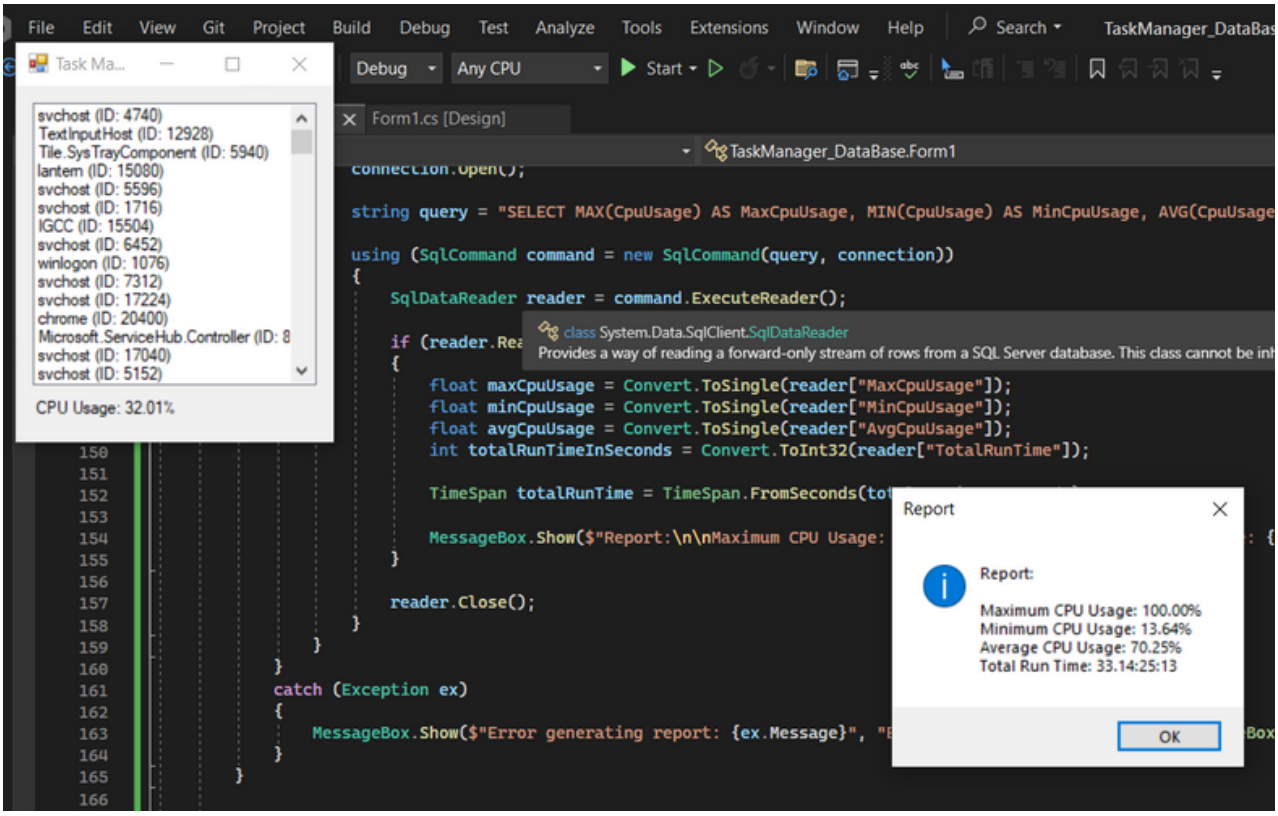


Figure 4: A Report generated at the execution of the program

```
using System;
using System.Data.SqlClient;
using System.Diagnostics;
using System.Windows.Forms;

namespace TaskManager_DataBase
{
  public partial class Form1 : Form
  {
    private string connectionString = "Data Source=DESKTOP-
7KE8K8N\\SQLEXPRESS;Initial Catalog=manager;Integrated Security=True";
    private PerformanceCounter cpuCounter;

    public Form1()
    {
      InitializeComponent();
      InitializePerformanceCounter();
      UpdateTaskList();
      UpdateCpuPerformance();
    }

    private void InitializePerformanceCounter()
    {
      cpuCounter = new PerformanceCounter("Processor", "% Processor Time", "_Total");
      // public PerformanceCounter(string categoryName, string counterName, string
instanceName)
      // : this(categoryName, counterName, instanceName, readOnly: true)

      cpuCounter.NextValue(); // Call NextValue once to initialize the counter
    }

    private void UpdateTaskList()
    {
      listBoxTasks.Items.Clear();

      Process[] processes = Process.GetProcesses();

      foreach (Process process in processes)
      {
        listBoxTasks.Items.Add($"{process.ProcessName} (ID: {process.Id})");
      }
    }
```

```
 private void UpdateCpuPerformance()
 {
 float cpuUsage = cpuCounter.NextValue();
 labelCpuUsage.Text = $"CPU Usage: {cpuUsage:F2}%";

 if (cpuUsage > 73.93409729)//73.93409729

 {
 StoreCpuPerformanceInDatabase(cpuUsage);
 }
 }
private void StoreCpuPerformanceInDatabase(float cpuUsage)
     {
        try
        {
           using (SqlConnection connection = new SqlConnection(connectionString))
           {
              connection.Open();

              string query = "INSERT INTO CpuPerformance (Timestamp, CpuUsage) VALUES
(@Timestamp, @CpuUsage)";

              using (SqlCommand command = new SqlCommand(query, connection))
              {
                 command.Parameters.AddWithValue("@Timestamp", DateTime.Now);
                 command.Parameters.AddWithValue("@CpuUsage", cpuUsage);

                 command.ExecuteNonQuery();
              }
           }
        }
        catch (Exception ex)
        {
           MessageBox.Show($"Error storing CPU performance in the database: {ex.Message}",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
     }
```

```
private void timer1_Tick(object sender, EventArgs e)
{
UpdateTaskList();
UpdateCpuPerformance();
}
private void Form1_Load(object sender, EventArgs e)
{
InitializeDatabase(); // Call this to create the database table if it doesn't exist
timer1.Start();
}
private void InitializeDatabase()
    {
        try
        {
            using (SqlConnection connection = new SqlConnection(connectionString))
            {
                connection.Open();
                string createTableQuery = @"
                    USE manager;

                    IF NOT EXISTS (SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE
TABLE_NAME = 'CpuPerformance')
                    BEGIN
                        CREATE TABLE CpuPerformance (
                            Id INT PRIMARY KEY IDENTITY(1,1),
                            Timestamp DATETIME NOT NULL,
                            CpuUsage FLOAT NOT NULL
                        );
                    END  ";
                using (SqlCommand command = new SqlCommand(createTableQuery, connection))
                {
                    command.ExecuteNonQuery();
                }
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Error initializing database: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
```

```
private void GenerateReport()
    {
        try
        {
            using (SqlConnection connection = new SqlConnection(connectionString))
            {
                connection.Open();

                string query = "SELECT MAX(CpuUsage) AS MaxCpuUsage, MIN(CpuUsage) AS
MinCpuUsage, AVG(CpuUsage) AS AvgCpuUsage, DATEDIFF(SECOND,
MIN(Timestamp), MAX(Timestamp)) AS TotalRunTime FROM CpuPerformance";

                using (SqlCommand command = new SqlCommand(query, connection))
                {
                    SqlDataReader reader = command.ExecuteReader();

                    if (reader.Read())
                    {
                        float maxCpuUsage = Convert.ToSingle(reader["MaxCpuUsage"]);
                        float minCpuUsage = Convert.ToSingle(reader["MinCpuUsage"]);
                        float avgCpuUsage = Convert.ToSingle(reader["AvgCpuUsage"]);
                        int totalRunTimeInSeconds = Convert.ToInt32(reader["TotalRunTime"]);

                        TimeSpan totalRunTime =
TimeSpan.FromSeconds(totalRunTimeInSeconds);

                        MessageBox.Show($"Report:\n\nMaximum CPU Usage:
{maxCpuUsage:F2}%\nMinimum CPU Usage: {minCpuUsage:F2}%\nAverage CPU Usage:
{avgCpuUsage:F2}%\nTotal Run Time: {totalRunTime}", "Report",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                    }

                    reader.Close();
                }
            }
        }
```

```
catch (Exception ex)
        {
            MessageBox.Show($"Error generating report: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }


    // Modify the Form1_FormClosing method to call GenerateReport before
closing the application
    private void Form1_FormClosing(object sender, FormClosingEventArgs e)
    {
        timer1.Stop();
        cpuCounter.Dispose();
        GenerateReport(); // Call this function to generate the report
    }

  }
}
```

CONCLUSION

In conclusion, the Task Manager Database Application represents a robust solution for monitoring and analyzing CPU performance in real time.

The integration of the PerformanceCounter class ensures continuous monitoring, providing users with immediate insights into the system's processing capabilities. The dynamic thresholding mechanism allows for proactive responses to critical performance issues, enhancing overall system management.

The application's ability to store CPU performance data in the SQL Server database facilitates historical trend analysis. Users can generate comprehensive reports with graphical representations, enabling a visual understanding of CPU performance over time. Export options further enhance collaboration and documentation.

The user-friendly interface, accompanied by clear tooltips and instructions, ensures accessibility for users with varying levels of technical expertise. Real-time analysis and reporting empower users to make informed decisions regarding system optimization and resource allocation.

FUTURE ENHANCEMENTS

As technology and user needs evolve, there are several avenues for future enhancements to the Task Manager Database Application:

1. Expanded Resource Monitoring:
   - Extend the application to monitor additional system resources, such as memory and disk usage, providing a more comprehensive view of system health.
2. Integration with External Tools:
   - Explore integration with external monitoring tools or cloud-based solutions for enhanced scalability and flexibility.
3. Enhanced Reporting Module:
   - Expand the reporting module to include more advanced analytics, trend predictions, and comparative analyses between different time periods.
4. User Configuration Settings:
   - Introduce user-configurable settings for dynamic thresholding, allowing users to customize threshold values based on their specific system requirements.
5. Real-time Alerts and Notifications:
   - Implement real-time alerts and notifications to immediately inform users of critical events, ensuring a proactive response to performance issues.
6. Security and Authentication Features:
   - Enhance security features, including user authentication, to safeguard sensitive performance data and ensure secure access to the application.
7. Cross-Platform Compatibility:
   - Explore the possibility of making the application compatible with multiple operating systems to cater to a broader user base.
8. Machine Learning Integration:
   - Investigate the integration of machine learning algorithms for predictive analysis of CPU performance trends, allowing the application to anticipate and mitigate potential issues.

By incorporating these future enhancements, the Task Manager Database Application can evolve into a more sophisticated and versatile tool for system monitoring, analysis, and optimization. These improvements will ensure the application remains relevant and adaptable to the ever-changing landscape of technology.