**Project SysMonitor –**

**Created for North Stars**

**Copyright 2020 - 2021 INCEDO INC**. All Rights Reserved. No part of this document may be reproduced, stored in a retrieval system, distributed or transmitted, in any form, or by any means, electronic or otherwise, including photocopying, reprinting, or recording, for any purpose, without the express prior written permission of Incedo Inc.

**REQUESTS** For information requests or for obtaining permission for the use of this work, please submit a written request to: Marketing and Communications, Incedo Inc, Plot No. 248 Udyog Vihar, Phase IV, Gurgaon - 122015. Email: [communications@incedoinc.com](mailto:communications@incedoinc.com)

|  |  |  |
| --- | --- | --- |
| Version | Owner | Date |
| Draft version – v1.0 | Joy Asthana [joy.asthana@incedoinc.com](mailto:joy.asthana@incedoinc.com) | January 13th, 2022 |

**Project Description: *SysMonitor***

|  |  |  |  |
| --- | --- | --- | --- |
| A central monitoring system built using basic client-server model to collect system information from multiple nodes over a network. This system information is supposed to be sent periodically to a server to update about its state with following parameters:   * System-name * RAM usage * CPU utilization * System Idle window * HDD utilization * Network stats, etc   Goal of this project is to give a significant flavor of interaction with file-system, operating-system, logging, database and socket-programming using C++. This project is specifically designed to be SILENT in nature without a graphical user interface that runs behind the scenes so that programmer can focus entirely on setting up the backbone/channel for a client-server architecture over a custom application-layer protocol but using conventional TCP/IP for network communication.  For a C++ programmer, its expected to know basics of system programming, in general and good sense of object-oriented principles.  Tools/SDKs: Aspirant who wish to attempt this project can use Visual Studio Code IDE, C/C++ programming constructs along with STL, Windows-API, boost, or any other C++ SDK to accomplish the task. | | | |
| Project Code: | SYSMON | Project Name: | Incedo SysMonitor |
| Duration : | 4 Weeks | Release Date: | TBD |
|  | | | |
| Project Content: SILENT CLIENT Client is supposed to periodically fetch relevant data via interacting with Windows system and send it over TCP/IP network to a configured server. Following could be a program-flow to accomplish its purpose:   * Read the configuration from a file about parameters related to establishing connection to a server, timer-period upon which client is supposed to capture and send the data. * Try connecting with the configured server. * Post successful connection, initiate a local timer with a handler. * Implement that handler to capture relevant system-information. * Structure this information and send it over to the server. * Log the SUCCESS/FAILURE messages locally to keep a record of success rate of publishing the system-info to the server.   Note 1: Programmer should be able to run several instances of such clients in same or different PCs to demonstrate/imitate distributed network of nodes.  Note 2: A console window can be used to display progress of the workflow with bare minimum messages/logs. CENTRAL SERVER A central server would be listening on a dedicated socket to accept the connections from various clients, receive and parse raw bytes received over the network, logs this information in a file or a database. A programmer can implement following workflow to achieve this purpose:   * Read the configuration from a file having parameters related to database connection string or path to a log-file and a listening socket that will accept connections from client. For example, a port number. * Initiate a connection to a database or set-up the file-pointers in write-mode. * Launch a separate thread to display the structured data read from the database or a file. * Start listening for new connections over the network on a dedicated port. * Accept the connection and establish a channel for further communication with that node. * Upon receiving the bytes over an established connection, parse them and send relevant SUCCESS/FAILURE response to that node. * Log the information parsed in last step into the database or a file. * Upon a local trigger, like a command on console, fetch desired information from database or a file and pretty-print the same.   Note 1: A console window can be used to display progress of the workflow with bare minimum messages/logs.  Note 2: Either on the same console or a separate console window can be used to interact with the user using basic custom commands. | | | |
| Possible timeline to work on this project:   * Week 1: Establish the basic client-server socket-connection and send Hello-messages with acknowledgements. Create and read the configuration file programmatically using CSV, XML or JSON text-format. Read the relevant system-information and store it in a structure to send it over the network. * Week 2: Set up a timer and invoke the function written previously. Actually, send the information to the server, parse it at server’s end using same interface/structure used at client-side. * Week 3: Set up a database or file-structure, use it to store information received at server’s end. Also write a separate thread to display appropriate information when asked using a command on console. * Week 4: Buffer to tackle anything pending from last week, Integration / Validation of end-to-end functionalities, documentation / presentation. | | | |

# Block-diagram



Server

10.20.21.5: 3000



Client 3



Client 2



Client 1

**System-info** sent from clients to central server:

- System-name

- RAM usage

- CPU utilization

- System Idle window

- HDD utilization

Network stats, etc

**Server database with consolidated information:**

- Client 1, Client 2, Client 3, … …

**TCP/IP Socket comm.**

**TCP/IP Socket comm.**

**TCP/IP Socket comm.**

|  |
| --- |
| Scope:  * Client-side: Structure of system information collected and sent over the network can be further expanded to include comprehensive analytics data to track work-time, work-output, in-focus apps, real-time file-changes stats, screenshots, idle-windows, etc. * Protocol: Custom payload for application-layer protocol to categorize and encapsulate system-information with timestamps. * Server: Consolidated analysis and meaningful interpretation on system-information gathered for a team, project or for entire organization. Publish the analysis over WebAPIs, that can be fed to web-clients to show graphical view of the same. |