**Universidade Federal de Campina Grande**

**Centro de Engenharia Elétrica e Informática**

**Departamento de Engenharia Elétrica**

**Sistemas Em Tempo Real**

**Sistema de Coleta e Processamento de Dados para uma Estufa Inteligente em C++**

3º Estágio – Processos, Threads e *WebSockets*

JOÃO VICTOR RODRIGUES RAMALHO – Matrícula: 117110955

Campina Grande, PB

2022

Sumário

[1 Descrição do Sistema 3](#_Toc100011572)

[2 Implementação na Linguagem de Programação C++ 4](#_Toc100011573)

[2.1 Threads 4](#_Toc100011574)

[2.1.1 Threads – Cliente 4](#_Toc100011575)

[2.1.2 Threads - Servidor 5](#_Toc100011576)

[2.2 Comunicação Cliente-Servidor 5](#_Toc100011577)

[2.2.1 *WebSockets* 5](#_Toc100011578)

[2.2.2 Implementação *Websockets* – Cliente 6](#_Toc100011579)

[2.2.3 Implementação *Websockets* – Servidor 7](#_Toc100011580)

[2.2.4 Diagrama de Sequência UML 8](#_Toc100011581)

[3 Discussões 8](#_Toc100011582)

# Descrição do Sistema

Tendo em vista o estudo de sistemas operacionais, de como os processos, threads e *websockets* funcionam, foi projetado Sistema de Coleta e Processamento de Dados para uma Estufa Inteligente na linguagem de programação C++. Desse modo, de acordo com arquitetura da figura a seguir, retirada dos slides da disciplina de Sistemas em Tempo Real, foram desenvolvidos os softwares e o sistema.

Nesse sentido, o sistema se baseia na leitura de dois sensores, um de temperatura e outro de luminosidade, ao qual possuem 2 softwares (em execução, processos) desenvolvidos, um software cliente e um software servidor, em que compartilham “variáveis”. No software do cliente ocorre: o processo por completo, definição das variáveis de buffer, leituras dos sensores e o envio de parte dessas variáveis para o servidor por meio das biblioteca de comunicação *Websockets* disponibilizadas pela linguagem abordada. Sendo assim, com o uso da biblioteca, o software servidor recebe as variáveis, executa a verificação da temperatura e luminosidade, alertando o sistema e envia a variável de saída do servidor para o cliente. Ademais, ambos são aplicações do prompt de comando, em que no software do cliente, o usuário pode interagir com o sistema por meio de comandos.

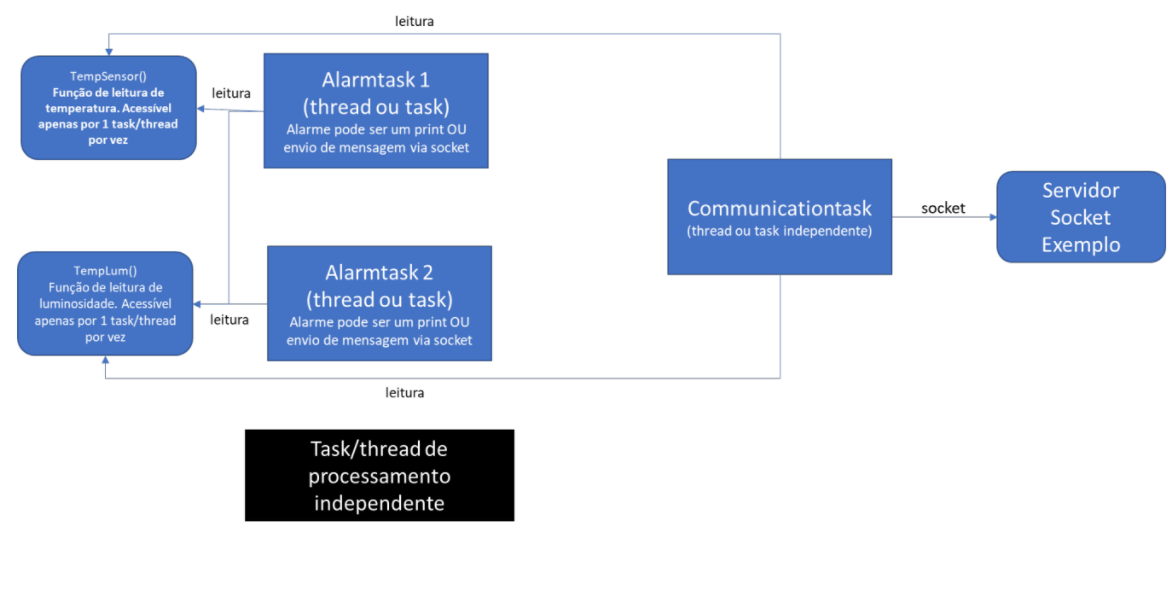


Figura 1: Arquitetura dos Softwares - Cliente e Servidor

# Implementação na Linguagem de Programação C++

A implementação na linguagem de programação C++ foi feita com um software servidor e outro software cliente, sendo desenvoldiso noVisual Studio 2022, no qual o foi realizado a compilação do Servidor por meio do Visual Studio, e realizado a execução via prompt de comando, já o Cliente do projeto foi compilado e depurando usando o Visual Studio, em cima sistema operacional Windows 11. Assim, de acordo com a Figura 1, foi realizado o desenvolvimento do código.

## Threads

Para a implementação de threads, foi utilizada a biblioteca disponibilizada pelo namespace std: <threads>.

Implementando, a técnica Thread Safe utilizada no projeto foi feita com o uso da biblioteca <atomic>, em que por meio de variáveis do tipo **atomic**, evitando condições de corrida para uma variável compartilhada entre threads. Dessa forma, o é criada uma variável do tipo atomic que é implementada utilizando *mutex locks*, ou seja, se uma thread requisita um *mutex lock*, nenhuma outra thread pode obter a variável até que seja liberada por essa mesma thread. Realizando um comparativo com o modelo de semáforos, não é criada uma seção crítica, nem é librerado o acesso ao recurso para mais de uma thread, a operação é somente com a variável a ser acessada.

Assim, uma vez que definidas as variáveis globais do tipo atomic, e nela um tipo genérico atrelado, como int, double, etc. Sendo considerado o tipo dessa variável. Dessa forma, para que se atribua os valores dentro das threads às referências atomic, de forma Thread Safe, foram utilizadas duas funções dessa struct: *.load()* e *.store()*.

Portanto, o cenário explorado necessário para Thread Safe utilizada as variáveis globais que precisam ser manipuladas por várias threads diferentes, em que foi utilizado o tipo *atomic* como o tipo dessas variáveis para que se possibilite os acessos por meio das threads e não ocorram condições de corrida.

### Threads – Cliente

Para o cliente foram definidas 5 threads:

* thread readCommandThread(readCommand, command);
* thread execCommandThread(executeCommand, command, COMMAND\_BUFF\_SIZE);
* thread modTthread(MOD\_T);
* thread modLthread(MOD\_L);
* thread tCommunicationThread(CommunicationThread, ConnectSocket);
* thread AlarmTask1Thread(AlarmTask1);
* thread AlarmTask2Thread(AlarmTask2);

. No qual, as duas primeiras realizam comunicação do usuário com o cliente, de forma que o usuário possa fazer alterações nas variáveis de buffer de Temperatura e Luminosidade e monitorar variáveis do sistema sem bloquear as outras threads. Nesse sentido, as outras threads são parte do funcionamento do sistema, como mostrado no esquema da Figura 1.

### Threads - Servidor

Para o servidor foram definidas 2 threads: handleListens e handleClients. Desse modo, a primeira teve o objetivo de tratar os pedidos de conexão por parte dos clientes, em que são aceitas nessa thread. Assim, essas conexões são repassadas para a thread handleClients, em que nela ocorrem as operações de enviar e receber dados para os clientes.

## Comunicação Cliente-Servidor

Sabendo que existia a necessidade de fazer a comunicação processos diferentes, com o intuito de trocar informações e variáveis, nesse caso, os valores dos buffers, foi feita a comunicação entre o servidor o cliente a fim de realizar o controle do sistema, por meio no protocolo de comunicação TCP/IP e dos *WebSockets*.

### *WebSockets*

Assim, para não se preocupar com os detalhes de comunicação da pilha TCP/IP, os WebSockets/Sockets abstraem toda ela, na forma de uma biblioteca. Desse modo, nessa pilha do protocolo de comunicação TCP/IP, os Websockets estão entre a camada de transporte e a camada de aplicação.Sendo asim, as aplicações implementadas fazem a chamada às funções do WebSocket, que se preocupam com toda a comunicação entre os softwares. A ideia que se têm é a de que as aplicações estão comunicando diretamente entre si, porém, ainda estão passando por toda a rede e pilha TCP/IP.

O protocolo TCP/IP é considerado seguro, eficiente, e que garante a entrega dos pacotes sem perdas, por meio da ordenação dos envios de pacotes, do “*handshakes*” de três vias e da forma de como funciona sua pilha de comunicação. Dessa forma, foi o protocolo escolhido para o projeto, por meio dos WebSockets.

Assim, para o uso dos WebSockets, as bibliotecas de WebSockets disponíveis na linguagem C++ fazem o uso da biblioteca compilada disponível no sistema operacional. Sendo necessário adicionar a biblioteca Ws2\_32.lib do windows.

Para o cliente realizar uma comunicação com servidor, bem como o contrário, é necessário que um conheça as informações de conexão. Essas informações estão contidas no que é chamado de “Socket”, em que o cliente e o servidor possuem sockets própios, e eles conhecem as informações do outro por meio das funções da biblitoeca <WinSock2.h>.

De posse das informações mostradas acima, sobre como o protocolo de comunicação funciona entre o cliente e o servidor, ou seja, entre os processos, existe uma sequência de chamadas da API WebSockets que têm que ser respeitadas para que ocorra a comunicação. Assim, para ambos, tanto o servidor como o cliente, têm que se inicializar o uso das DLLs do Windows Sockets, e isso é feito por meio da função WSAStartup(). Em seguida, ambos seguem uma própia sequência de implementação. Essas sequências são mostradas na figura a seguir:

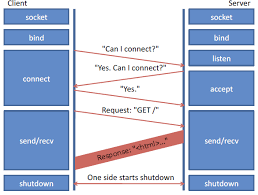
Fonte: Modelo Cliente/Servidor e Introdução a Sockets - UNICAMP

Figura 5: Exemplicação da processo

### Implementação *Websockets* – Cliente

Para o cliente, após a inicialização, é necessário fazer a configuração do socket desejado, em que a interpretação das configurações de endereços sockets é feita plea função getaddrinfo(). Logo após, é realizado a criação do socket, por meio da função socket() e, então, a conexão ao servidor utilizando a função connect(), em que é adquirido o socket do servidor. Essas funções foram utilizadas na main do programa.

Por fim, o socket com as informações do servidor é passado para uma thread, que irá receber pacotes/variáveis do processo, por meio das funções send() e recv(),

De forma a compactar a mensagem em uma string, utilizando o separador “,” utilizando formas de concatenação e sepração da linguagem C++. Quando a comunicação terminar, o programa termina essa comunicação do socket, por meio da função closesocket(), e encerra a utilização da DLL Windows Socket por meio da função WSACleanup().

### Implementação *Websockets* – Servidor

Já no servidor, sendo realizada a inicialização, é preciso fazer a configuração do socket desejado, em que a tradução dessas configurações para endereços sockets é feita por meio da função getaddrinfo(). Em seguida, é feita a criação do socket, por meio da função socket() e, então, o servidor precisa fazer um bind(), que associa o socket a uma porta do sistema operacional (27069, nesse caso).

Em seguida, esse socket do servidor é passado para uma thread que irá fazer o tratamento de novas conexões advindas dos cliente. Sendo assim, foi utilizada a função listen() para escutar novas conexões de clientes na porta do sistema operacional. Assim, quando o cliente utiliza da função connect, após ser escutado pela função listen(), o servidor começa a se comunicar com o cliente por meio da função accept().

Inicialmente no projeto, após ser aceita e inicalizada a conexão entre cliente e servidor, foi implementada a ideia de criação de uma nova thread, passando como parâmetro esse socket do cliente, e , nessa thread, o servidor encontra-se preparado a receber metodos/variáveis do processo, por meio das funções send() e recv()., por meio da concatenação/separação de strings, por meio do vetor iterado pelo comprimento do buffer.

Por fim, ao finalizar o servidor, são chamadas as funções de closesocket() e WASCleanup(), encerrando a utilização da DLL Windows Socket.