Projeto: *C-Chat*

Computação em Nuvem, 2025

Equipe de Desenvolvimento:

- Rafael Renó Corrêa;
- Lucas Ferreira Alves; e
- Gabriel da Silva Henrique.

Ementa

O projeto de desenvolvimento *C-Chat* constitui da elaboração de um sistema de *chatting* cliente-servidor na linguagem C, com comunicação simultânea de muitos usuários através de multiprocessamento e *multithreading* em uma máquina servidora principal, que pode ser hospedada em serviços de nuvem. Nesse contexto, essa ferramenta pode ser utilizada de duas formas: através de uma *interface* robusta pelo terminal do Linux ou por meio de uma *interface* em C++ para *desktop* ou *mobile*.

Sumário

Acordo de Nível de Serviço

O processo de implementação do projeto constitui de dois ramos codependentes bem definidos: *core* (ou núcleo) e *interface*. Isto é:

- *Core*: Aplicação em C responsável pela lógica de comunicação, implementação das regras de negócio e *interface* rudimentar para administração do servidor do sistema de *chatting*, com suporte a múltiplos clientes simultâneos por meio de multiprocessamento e *multithreading* na máquina servidora; e
- *Interface*: Aplicação Thin *Client* (ou Cliente Leve) em C/C++ responsável pela *interface* de comunicação do cliente com o servidor. Nesse contexto, serão desenvolvidas duas aplicações:
  - Em terminal do Linux (C): Implementação simples e robusta na máquina cliente para demonstração da comunicação cliente-servidor pelo terminal do Linux; e posteriormente,
  - Em alto nível (C++): *Software* para a *interface* de cliente em uma aplicação intuitiva e ergonômica desenvolvida a partir da biblioteca ImGui.

Nesse cenário, estão listadas nas páginas a seguir as expectativas para cada etapa do desenvolvimento do projeto.

Considerações Legais

Quaisquer aspectos do planejamento de implementação, atribuição de responsabilidades, prazos estipulados e resultados esperados, disponíveis neste documento, estão sujeitos à alteração sem aviso prévio.

Compatibilidade

A aplicação *core* será implementada em C, utilizando a biblioteca `<arpa/inet.h>` para a lógica de comunicação em rede, e bibliotecas como `<stdlib.h>`, `<unistd.h>`, `<sys/wait.h>`, `<pthread.h>`, `<omp.h>` (OpenMP) e `<signal.h>` para lidar com muitos clientes simultâneos por meio de multiprocessamento, *multithreading* e manipulação de sinais. Nesse contexto, como os métodos de paralelismo adotados são específicos do ambiente Linux, a aplicação será nativa deste sistema operacional, com foco na distribuição Ubuntu.

Responsabilidades

Cada membro da equipe de desenvolvimento será responsável por partes fundamentais do projeto. Onde, especificamente, Rafael Renó Corrêa irá implementar a aplicação servidora (*core*) e a *interface* cliente em C para o ambiente do terminal do Linux, e Lucas Ferreira Alves e Gabriel da Silva Henrique irão implementar a aplicação *Thin Client* em C++, baseada em ImGui, para *desktop* (Linux ou Windows) e *mobile* (Android).

Passos de Trabalho

A organização das atividades de implementação e modelagem será definida por dois ramos de desenvolvimento: *Core* e *Interface*. Nesse contexto, os passos de trabalho de cada ramo serão tratados separadamente, e posteriormente, as considerações a respeito dos prazos e dependências serão discutidas.

Assim sendo, quanto ao *Core*, são os passos de desenvolvimento a seguir necessários para a funcionalidade básica da aplicação servidora:

1) Estabelecer conexão persistente (`TCP/IPv4`) entre dois computadores e verificar a comunicação através do envio de mensagens;
2) Paralelizar a lógica de aceitação de novas conexões – utilizando as bibliotecas `<stdlib.h>` e `<unistd.h>`, de forma que a aplicação crie um novo processo filho para cada novo cliente, e exclua o processo criado uma vez que a conexão encerrar;
3) Dividir a lógica de comunicação cliente-servidor em duas *threads* simultâneas em cada processo – utilizando a biblioteca OpenMP – tanto para a aplicação cliente quanto para a aplicação servidora, ou seja:
    a) Uma *thread* para o recebimento de mensagens; e
    b) Outra *thread* para o envio de mensagens.
4) Realizar o tratamento dos sinais de interrupção (`SIGINT`), término (`SIGTERM`) e de encerramento de processo filho (`SIGCHLD`);
5) Implementar uma lista de PIDs dos processos filhos no contexto do processo pai, com a estrutura `struct leaf{ int UUID; char username[16]; int PID };` e as seguintes operações automáticas:
    a) `CREATE`: Quando uma nova conexão com um processo filho for estabelecida, deve ser acrescentado um novo nó na lista com o identificador único (`UUID`) do cliente, apelido (escolhido pelo cliente) e o `PID` do processo filho;
    b) `READ`: O nome do cliente e o `PID` do processo filho podem ser consultados a partir do UUID do cliente; e
    c) `DELETE`: O nó do cliente na lista de `PIDs` pode ser excluído a partir do `UUID` do cliente.
6) Definir uma rotina de encerramento gracioso, ou seja, um procedimento para terminar todos os processos e *threads* ativos, além de fechar todos os soquetes de comunicação, uma vez que o sinal de interrupção (CTRL+C) for acionado no servidor (no contexto do processo pai);
7) Implementar uma *thread* no processo pai para direcionar mensagens entre processos filhos – identificadas pelos `UUIDs` dos clientes remetente e destinatário). Nesse contexto, devem ser implementados dois protótipos:

a) Inicialmente, um fórum em que todos falam para todos; e

b) Posteriormente, comunicação privada entre somente dois clientes.

8) Realizar o tratamento de parâmetros de inicialização (por leitura de arquivo) para identificar o nome de cliente e palavra secreta, de forma que uma conexão possa ser estabelecida se, e somente se, dois clientes compartilharem a mesma chave secreta;

9) Escrever mensagens de *feedback* e de tratamento de erro para impressão no terminal;

10) Definir a *interface* de aplicação (API); e

11) Refinar o *software* em iterações sucessivas de modelagem.

Já quanto à *Interface*, são os passos de desenvolvimento a seguir necessários para a modelagem e funcionalidade da aplicação *Thin Client*:

1) Modelar o design básico da janela de *interface* do cliente com as seguintes funcionalidades:

   a) Na primeira janela, deve ser possível informar o **nome de cliente** (com no máximo 15 caracteres) e uma **chave secreta**. Com um **botão de confirmação** e um **botão de cancelamento**;

   b) Se confirmado, deve abrir uma segunda janela (de *feedback)* para demonstrar a tentativa de conexão, que dura exatamente um minuto; e

   c) Se bem-sucedida, deve abrir uma terceira janela (e fechar todas as outras), onde haverá um **campo de texto** para inserir uma mensagem a ser encaminhada, um **botão de confirmação** para enviar a mensagem, um **segundo campo de texto** para imprimir as mensagens recebidas e um **botão para encerrar a conexão**.

2) Implementar um protótipo básico da aplicação baseada em ImGui com os campos, botões e animações adequadas (porém sem as funcionalidades);

3) Acrescentar as funcionalidades aos botões e campos a partir da comunicação com a API do servidor; e

4) Refinar a aplicação em iterações sucessivas de testagem.

Prazos Estabelecidos

Cada item deve ser implementado em um prazo de ~~aproximadamente~~ uma semana. Entretanto, para a aplicação *core*, o 1º, 2º, 3º e 4º passos devem ser implementados no prazo de duas semanas. Posteriormente, do 5º ao 10º passo, cada item deve ser implementado em somente uma semana.

Assim sendo, para a *interface*, o 1º e 2º passos podem ser desenvolvidos a partir de 21/04/2025 em duas semanas, e o 3º passo – que depende da conclusão do 10º passo do *core* – pode ser implementado nas duas semanas seguintes à conclusão deste passo. Por fim, o 11º passo do *core* e o 4º passo da *interface* devem ser iterados até o prazo limite mínimo de 27/06/2025, ou até o prazo máximo de 11/07/2025.

Isto é, até o prazo de entrega limite mínimo em 27/06/2025, **o projeto deve ser implementado em no máximo oito semanas (ou dois meses) a partir de 21/04/2025**. Ou seja, como está descrito na Tabela 1:

Ramo	Passo	Prazo
Core	1º – 10º	21/04/2025 – 16/06/2025
	11º	16/06-2025 – 27/06/2025
Interface	1º – 2º	21/04/2025 – 05/05/2025
	3º	16/06/2025 – 23/06/2025
	4º	23/06/2025 – 27/06/2025

Tabela 1: Prazos para a conclusão do projeto.

Resultados Esperados

Uma vez concluída, a aplicação deve fornecer uma solução robusta com uma *interface* intuitiva para a comunicação segura entre muitos usuários, com baixa latência para o cliente e baixo custo em recursos computacionais na máquina servidora – ou, em termos práticos, uma instância do sistema deve suportar até 128 usuários simultâneos com latência inferior a 100ms (milissegundos) para cada mensagem transferida.