**Editor de Texto Colaborativo Distribuído com MPI e OpenMP**

Este documento descreve um projeto que implementa um editor de texto colaborativo simples, permitindo que múltiplos usuários editem o mesmo documento simultaneamente. Ele utiliza **MPI (Message Passing Interface)** para a comunicação entre os usuários (processos) e **OpenMP** para paralelizar a geração de dados internos para testes de desempenho.

**Funcionalidades Implementadas**

* **Múltiplos Usuários:** Suporte a múltiplos usuários conectados simultaneamente, onde cada usuário é um processo MPI distinto.
* **Consistência de Dados:** Mecanismo de bloqueio (lock) por linha, garantindo que apenas um usuário possa editar uma linha por vez. Tentativas de edição em linhas bloqueadas são recusadas, informando qual usuário já está editando.
* **Atualização em Tempo Real:** Alterações no documento são imediatamente propagadas para todos os usuários através de comunicação coletiva.
* **Comunicação Ponto-a-Ponto:** Usuários podem enviar mensagens privadas uns aos outros.
* **Comunicação Coletiva:** O servidor utiliza broadcasts para sincronizar o estado do documento e atualizações de linhas com todos os clientes.
* **Geração Automática de Dados:** Funcionalidade para gerar um volume de dados grande automaticamente, utilizando paralelismo OpenMP para acelerar o processo.
* **Logs de Alteração:** Cada processo mantém um arquivo de log detalhado, registrando as ações e comunicações realizadas.

**Requisitos**

Para compilar e executar este projeto, você precisará ter:

* Um compilador C/C++ que suporte OpenMP (geralmente gcc ou clang).
* Uma implementação de MPI (como **Open MPI** ou **MPICH**) instalada no seu sistema.

**Como Compilar**

1. **Salve o Código:** Salve o código-fonte fornecido em um arquivo com a extensão .c (por exemplo, editor\_colaborativo.c).
2. **Abra o Terminal:** Navegue até o diretório onde você salvou o arquivo.
3. **Compile o Código:** Utilize o compilador MPI (geralmente mpicc) com a flag para OpenMP (-fopenmp).
4. mpicc -fopenmp editor\_colaborativo.c -o editor\_colaborativo
   * mpicc: É o *wrapper* do compilador MPI para C. Ele já inclui as bibliotecas e flags necessárias para o MPI.
   * -fopenmp: Habilita o suporte a OpenMP no compilador, permitindo que as diretivas #pragma omp sejam processadas.
   * editor\_colaborativo.c: É o nome do seu arquivo fonte.
   * -o editor\_colaborativo: Define o nome do arquivo executável de saída.

Se a compilação for bem-sucedida, você terá um arquivo executável chamado editor\_colaborativo no mesmo diretório.

**Como Executar**

O programa deve ser executado utilizando o comando mpiexec (ou mpirun), que é o utilitário padrão para iniciar aplicações MPI. Você precisa especificar o número de processos (-np) que deseja rodar.

**Importante:** No design deste projeto, o processo com **rank 0** sempre atuará como o **servidor**, enquanto os demais processos (ranks 1, 2, ...) serão os **clientes/usuários**.

Para executar com **N usuários** (contando o servidor), você deve usar -np N. Recomenda-se usar N >= 2 para ter pelo menos um cliente interagindo com o servidor.

mpiexec -np 3 ./editor\_colaborativo

Este comando iniciará 3 processos:

* **Processo 0:** O servidor central que gerencia o documento e todos os bloqueios de linha.
* **Processo 1:** Um cliente/usuário ativo.
* **Processo 2:** Outro cliente/usuário ativo.

**Interagindo com o Programa**

Ao executar o comando acima, a saída de todos os processos será exibida no mesmo terminal. As mensagens do servidor e dos clientes se intercalarão.

* O **servidor (Rank 0)** iniciará o documento com conteúdo padrão e aguardará requisições dos clientes.
* Cada **cliente/usuário (Rank > 0)** apresentará um menu de opções no terminal:
  1. **Editar linha:** Permite ao usuário solicitar o bloqueio de uma linha para edição. Se o bloqueio for concedido (ou seja, a linha não está em uso por outro usuário), o cliente poderá digitar um novo texto para aquela linha. Essa alteração será então enviada ao servidor e propagada aos demais clientes.
  2. **Enviar mensagem pública (Chat):** Permite enviar uma mensagem de chat que será retransmitida pelo servidor para *todos* os outros clientes.
  3. **Gerar dados automáticos (OpenMP):** Esta opção usa OpenMP para preencher a cópia local do documento do cliente com conteúdo gerado aleatoriamente. Após a geração, esses dados são submetidos ao servidor, que os aplicará ao documento global e os transmitirá para todos os usuários. Isso é útil para testes de desempenho.
  4. **Enviar mensagem privada:** Permite ao usuário enviar uma mensagem direta e privada para um usuário específico, utilizando comunicação ponto-a-ponto entre os ranks dos processos.
  5. **Sair:** Encerra a execução do cliente.

Você precisará digitar o número da opção desejada e pressionar ENTER para interagir. Se as saídas ficarem muito misturadas, continue pressionando ENTER para ver o próximo menu ou as próximas mensagens.

**Monitorando Logs**

Cada processo criará um arquivo de log separado no mesmo diretório de execução. Os arquivos serão nomeados no formato log\_usuario\_<rank>.txt (por exemplo, log\_usuario\_0.txt para o servidor, log\_usuario\_1.txt para o primeiro cliente, etc.).

Você pode abrir esses arquivos em terminais separados para monitorar as ações e a comunicação em tempo real, o que é extremamente útil para depuração e para comprovar o funcionamento das funcionalidades.

Exemplo de como monitorar um log em tempo real (em um terminal separado):

tail -f log\_usuario\_0.txt

Isso mostrará as últimas entradas do log do servidor conforme elas são adicionadas.

**Estrutura do Código (Visão Geral)**

* Line struct: Define a estrutura de cada linha do documento, contendo o texto (char text[MAX\_LEN]) e o rank do usuário que a bloqueou (int locked\_by).
* PrivateMessage struct: Utilizada para encapsular mensagens ponto-a-ponto entre usuários, incluindo o remetente, o destinatário e o conteúdo da mensagem.
* log\_message, log\_edit\_action: Funções auxiliares para registrar eventos e ações específicas (como edições e bloqueios) em arquivos de log dedicados para cada processo.
* init\_document\_server: Inicializa o conteúdo inicial do documento no processo servidor.
* print\_document\_client: Exibe o estado atual do documento na interface de console do cliente, indicando linhas livres, editando ou bloqueadas.
* gerar\_dados\_automaticos\_omp: Função que utiliza **OpenMP** para paralelizar a criação de um grande volume de conteúdo para o documento, demonstrando o uso de múltiplos threads.
* broadcast\_full\_document, broadcast\_line\_update: Funções no servidor que utilizam **MPI\_Bcast** (comunicação coletiva) para enviar o estado completo do documento ou atualizações de linhas específicas para todos os clientes, garantindo a sincronização.
* server\_main\_loop: Contém a lógica principal do processo servidor, incluindo o tratamento de requisições de bloqueio, submissões de edição, retransmissão de mensagens de chat e recebimento de dados gerados automaticamente.
* send\_private\_message: Envia uma mensagem ponto-a-ponto para um usuário específico, utilizando MPI\_Send.
* client\_handle\_incoming\_messages: Gerencia a recepção assíncrona de mensagens e atualizações (do servidor ou de outros clientes) no lado do cliente, usando MPI\_Iprobe para verificar a chegada de mensagens sem bloquear.
* client\_main\_loop: Contém o menu de interação e a lógica principal para o processo cliente, incluindo o envio de requisições e a exibição de informações.
* main: O ponto de entrada do programa, responsável pela inicialização do ambiente MPI, configuração do OpenMP, abertura dos arquivos de log e direcionamento para a lógica do servidor ou cliente com base no rank do processo.