|  |  |
| --- | --- |
|  | **Departamento de Engenharias, Arquitetura e Computação**  Disciplina de Programação Paralela e Distribuída  **Profa.: Ivan L. Süptitz**  **Trabalho – Entrega em 03/07/2025** |

Integrantes: Arthur, Guilherme, Nícolas e João Antonio

**Introdução**

Este relatório descreve o desenvolvimento de um sistema colaborativo distribuído como parte da disciplina de Programação Paralela e Distribuída. O objetivo principal é aplicar, de forma prática, os conceitos de paralelismo e comunicação entre processos utilizando MPI e OpenMP. A proposta visa proporcionar uma experiência de programação realista, simulando ambientes modernos de trabalho colaborativo.

**Objetivo do Sistema**

O sistema desenvolvido consiste em um editor colaborativo de texto projetado para operar em um ambiente distribuído. Seu principal objetivo é possibilitar que múltiplos usuários (clientes), executando simultaneamente, interajam com um mesmo documento textual, realizando edições de forma concorrente, porém controlada, garantindo consistência e sincronização entre os processos.

O projeto adota tecnologias de paralelismo e comunicação interprocessual, utilizando MPI (Message Passing Interface) para a troca de mensagens entre processos distribuídos e OpenMP (Open Multi-Processing) para a execução paralela interna. O foco é didático, voltado à aplicação prática de conceitos de computação paralela e distribuída, simulações e testes de desempenho em sistemas com múltiplas entidades cooperativas.

**Descrição Geral do Funcionamento**

A aplicação é composta por um servidor (processo mestre de rank 0) e vários clientes (processos escravos com ranks > 0), todos instanciados no mesmo programa via mpirun. O servidor atua como gerente central, responsável por:

* Controlar o acesso às linhas do texto compartilhado.
* Propagar alterações feitas por clientes para os demais.
* Gerenciar o envio de mensagens privadas entre clientes.
* Monitorar e encerrar o sistema quando todos os clientes finalizarem.

Já os clientes são responsáveis por:

* Solicitar a edição de uma linha do texto.
* Atualizar a linha local e reportar a modificação ao servidor.
* Enviar mensagens privadas a outros clientes.
* Receber atualizações em tempo real e mensagens privadas enquanto interagem com o sistema.

**Arquitetura do Sistema**

**Modelo de Execução Distribuído**

A arquitetura do sistema segue o modelo mestre-escravo, com forte acoplamento de comunicação via MPI, mas mantendo paralelismo interno por cliente com OpenMP.

* **Mestre (rank 0):**
  + Controla o texto global e as permissões de edição.
  + Atua como um hub de mensagens privadas.
  + Detecta a saída dos clientes e determina o encerramento do sistema.
* **Escravos (ranks > 0):**
  + Interagem com o usuário via linha de comando.
  + Executam seções paralelas para interface e escuta de mensagens em background.
  + Atualizam localmente as linhas do texto conforme alterações são recebidas.

**Detalhamento das Funcionalidades**

**Texto Compartilhado**

O texto editável é armazenado como uma matriz de caracteres com tamanho fixo:

char texto[MAX\_LINHAS][MAX\_TAM\_LINHA];

* Cada linha é tratada como uma unidade de edição isolada.
* Ao iniciar, o servidor gera o conteúdo de cada linha utilizando múltiplas threads via OpenMP, com mensagens de debug indicando qual thread foi responsável por cada linha.

**Controle de Acesso às Linhas**

Para evitar condições de corrida, o sistema utiliza um vetor auxiliar linha\_em\_uso[MAX\_LINHAS], no qual cada índice indica se a linha correspondente está:

* Livre (valor 0)
* Ocupada por um processo específico (rank do cliente)

Antes de conceder permissão para edição, o servidor verifica se a linha solicitada está desocupada. Caso esteja em uso, a solicitação é negada com uma resposta adequada.

**Comunicação e Atualizações**

**1. Solicitação de Edição**

* O cliente envia uma mensagem com o número da linha desejada para edição.
* O servidor responde autorizando ou negando o pedido, conforme disponibilidade.

**2. Liberação da Linha**

* Após editar, o cliente envia a nova versão da linha ao servidor.
* O servidor então:
  + Atualiza sua cópia local.
  + Marca a linha como desocupada.
  + Propaga a atualização para todos os clientes com “MPI\_Isend”.

**3. Atualizações em Tempo Real**

* Cada cliente possui uma thread ativa (via #pragma omp section) dedicada a receber mensagens do servidor.
* Atualizações recebidas alteram a cópia local do texto, mantendo todos sincronizados com a versão mais recente.

**Mensagens Privadas**

Além da colaboração textual, o sistema oferece um canal privado de comunicação entre clientes:

* Um cliente envia uma mensagem privada ao servidor, indicando o destinatário.
* O servidor verifica a validade do destinatário e, se válido, encaminha a mensagem ao cliente de destino.

Este mecanismo simula canais privados de chat ou notificações internas entre colaboradores.

**Saída do Sistema**

* Ao escolher "Sair", o cliente envia uma mensagem com a tag “TAG\_SAIR”.
* O servidor decrementa o número de clientes ativos.
* Quando todos os clientes saem, o servidor encerra sua execução e finaliza o sistema globalmente.

**Detalhamento Técnico das Funções-Chave**

**No Servidor**

* “servidor\_loop”(int rank, int size): loop principal que escuta requisições e toma decisões conforme a tag recebida.
* “gerar\_texto\_com\_openmp”(): usa #pragma omp parallel for para gerar o conteúdo inicial.
* “log\_evento”(): padroniza logs com data/hora, tipo e origem do evento.
* “imprimir\_texto”(): mostra o texto completo e indica quais linhas estão em uso e por quem.

**No Cliente**

* “cliente\_loop”(int rank, int size): divide sua execução em duas seções paralelas:
  + Escuta por atualizações do servidor e mensagens privadas.
  + Interação com o menu textual, envio de mensagens e comandos de edição.
* “imprimir\_texto”(): visualiza a cópia local do texto.

**Tags de Comunicação (MPI)**

| **Tag Constante** | **Finalidade** |
| --- | --- |
| TAG\_SOLICITAR\_EDICAO | Cliente pede para editar uma linha. |
| TAG\_RESPOSTA\_EDICAO | Servidor responde à solicitação de edição. |
| TAG\_LIBERAR\_LINHA | Cliente envia nova linha e libera o acesso. |
| TAG\_ATUALIZACAO\_TEXTO | Servidor propaga linha modificada a todos os clientes. |
| TAG\_MSG\_PRIVADA\_SEND | Mensagem privada entre clientes, mediada pelo servidor. |
| TAG\_SAIR | Cliente informa que está encerrando participação. |
| TAG\_TEXTO\_INICIAL | Texto base enviado pelo servidor para inicializar os clientes. |

**Segurança e Consistência**

* **Sincronização explícita via servidor**: toda modificação deve passar pelo rank 0, que atua como autoridade do estado global.
* **Execução determinística**: apesar de concorrência interna, o sistema é projetado para garantir que duas edições simultâneas de uma mesma linha não ocorrem.
* **Logs centralizados**: cada evento relevante é registrado com horário, origem e descrição.

**Requisitos e Execução**

**Compilação**

Requer um compilador com suporte a MPI e OpenMP:

mpicc -fopenmp editor\_colaborativo.c -o editor

**Execução**

Execute com mpirun e, se necessário, certifique-se de que o script tenha permissão de execução:

chmod +x run.sh

./run.sh editor\_colaborativo

Ou diretamente:

mpirun -np 4 ./editor\_colaborativo

**Prints Código**

Texto

Descrição gerada automaticamente

Estabelece a estrutura fundamental do editor de texto colaborativo. Os includes importam as bibliotecas necessárias para MPI (comunicação entre processos), OpenMP (paralelização), e funções básicas de entrada/saída. As constantes definem os limites do sistema: 10 linhas máximas de texto, 256 caracteres por linha, e 200 caracteres para mensagens privadas. Os TAGs numerados (10-16) servem como identificadores únicos para diferentes tipos de mensagens MPI, permitindo que o servidor distingua entre solicitações de edição, atualizações de texto, mensagens privadas e comandos de saída. As duas estruturas principais, MensagemEdicao e MensagemPrivada, encapsulam os dados transmitidos entre processos: a primeira controla o acesso e modificação das linhas do texto, enquanto a segunda gerencia a comunicação direta entre clientes. Por fim, as variáveis globais texto, linha\_em\_uso e clientes\_ativos mantêm o estado compartilhado do documento e o controle de concorrência no servidor.

Texto

Descrição gerada automaticamente

Esta seção contém três funções utilitárias essenciais para o funcionamento do sistema. A função gerar\_texto\_com\_openmp() inicializa o documento usando paralelização OpenMP, onde cada thread processa diferentes linhas simultaneamente, criando conteúdo padrão que identifica qual thread gerou cada linha e inicializando o array de controle de uso das linhas. A função imprimir\_texto() exibe o estado atual do documento, com uma lógica especial para o servidor (rank 0) que mostra quais linhas estão sendo editadas e por qual processo, enquanto os clientes veem apenas o conteúdo das linhas. Por fim, log\_evento() implementa um sistema de logging robusto que registra todas as atividades do sistema com timestamp preciso no formato "YYYY-MM-DD HH:MM:SS", incluindo o tipo de evento, o rank do processo responsável e uma mensagem descritiva, garantindo rastreabilidade completa das operações e facilitando a depuração do sistema distribuído.

Texto

Descrição gerada automaticamente

Esta seção implementa o núcleo da função servidor\_loop(), que gerencia todas as operações do servidor. Inicialmente, o servidor gera o texto usando OpenMP, registra o evento no log e distribui o conteúdo inicial para todos os clientes através de MPI\_Send individual (há código comentado mostrando uma alternativa com MPI\_Bcast). O loop principal utiliza MPI\_Iprobe para verificar mensagens de forma não-bloqueante, permitindo que o servidor processe requisições de múltiplos clientes simultaneamente. Quando uma solicitação de edição (TAG\_SOLICITAR\_EDICAO) é recebida, o servidor valida se a linha está dentro dos limites válidos e se não está sendo usada por outro cliente. Se a linha estiver disponível, o servidor a reserva para o cliente solicitante e registra a operação no log; caso contrário, nega a solicitação informando que a linha está ocupada. Este mecanismo garante exclusão mútua no acesso às linhas, evitando conflitos de edição simultânea e mantendo a consistência do documento compartilhado.

Texto

Descrição gerada automaticamente

Esta seção complementa o tratamento das solicitações de edição e implementa o mecanismo de liberação e atualização de linhas no servidor. Quando um cliente solicita liberar uma linha (TAG\_LIBERAR\_LINHA), o servidor primeiro valida se a linha está dentro dos limites válidos e se o cliente realmente possui o lock dessa linha (verificando se linha\_em\_uso[linha] == cliente\_rank). Se a validação for bem-sucedida, o servidor atualiza o conteúdo da linha com o novo texto enviado pelo cliente, libera o lock zerando linha\_em\_uso[linha], e registra a operação no log. Em seguida, utiliza MPI\_Isend assíncrono para propagar a atualização para todos os outros clientes conectados, garantindo que todos vejam as modificações em tempo real. O uso de MPI\_Request permite que o servidor continue processando outras requisições sem bloquear na espera de confirmação das mensagens enviadas. Caso a validação falhe (cliente tentando liberar linha que não possui ou linha inválida), o servidor registra um erro no log mas continua operando normalmente, mantendo a integridade do sistema.

Texto

Descrição gerada automaticamente com confiança média

Esta seção finaliza o tratamento das principais operações do servidor, abordando o sistema de mensagens privadas e o controle de saída dos clientes. Quando uma mensagem privada (TAG\_MSG\_PRIVADA\_SEND) é recebida, o servidor primeiro registra no log o conteúdo e os participantes da comunicação, depois valida se o destinatário é um rank válido (maior que 0, menor que o tamanho total, e diferente do próprio servidor). Se válido, encaminha a mensagem usando MPI\_Send diretamente para o cliente destinatário e registra o encaminhamento; caso contrário, descarta a mensagem e registra o erro, protegendo o sistema contra tentativas de comunicação com processos inexistentes. Para solicitações de saída (TAG\_SAIR), o servidor simplesmente decrementa o contador clientes\_ativos e registra no log quantos clientes ainda permanecem conectados. Este mecanismo permite que o servidor monitore quando todos os clientes se desconectaram, possibilitando o encerramento ordenado do sistema. O design assegura que o servidor atue como um intermediário confiável para todas as comunicações, mantendo controle total sobre o fluxo de mensagens e o estado da aplicação distribuída.

Texto

Descrição gerada automaticamente

Esta seção implementa o sistema de recepção assíncrona de mensagens no cliente usando OpenMP. A primeira seção paralela funciona como um listener em background que monitora continuamente mensagens do servidor através de MPI\_Iprobe, verificando se há comunicações pendentes sem bloquear a execução. Quando detecta uma mensagem de atualização de texto (TAG\_ATUALIZACAO\_TEXTO), o cliente recebe a estrutura MensagemEdicao, atualiza sua cópia local do texto e exibe uma notificação em tempo real informando qual linha foi modificada. Da mesma forma, ao receber mensagens privadas (TAG\_MSG\_PRIVADA\_SEND), o sistema exibe imediatamente a mensagem recebida com identificação do remetente. Este design permite que o cliente mantenha sua interface de usuário responsiva enquanto simultaneamente processa atualizações do servidor, garantindo sincronização em tempo real entre todos os participantes do editor colaborativo.

Texto

Descrição gerada automaticamente

Esta seção implementa a interface interativa do usuário na segunda seção paralela do OpenMP. Enquanto a primeira seção monitora mensagens em background, esta thread gerencia o menu principal do cliente, apresentando quatro opções: editar linha, enviar mensagem privada, visualizar texto e sair. O código utiliza fgets() para capturar entrada do usuário de forma segura e sscanf() para validar se a opção digitada é um número válido. Quando o usuário escolhe editar uma linha (opção 1), o sistema primeiro exibe todo o texto atual para facilitar a seleção, permitindo que o usuário visualize o conteúdo antes de decidir qual linha modificar. Este design de interface baseada em menu com validação de entrada garante uma experiência de usuário robusta, evitando erros de input enquanto mantém o fluxo de interação simples e intuitivo no ambiente de terminal distribuído.

Texto

Descrição gerada automaticamente

Esta seção implementa o protocolo de edição colaborativa com controle de acesso exclusivo. O processo inicia quando o usuário solicita editar uma linha específica: primeiro valida se o número da linha está dentro dos limites válidos, depois envia uma requisição ao servidor através de TAG\_SOLICITAR\_EDICAO para obter permissão exclusiva. O servidor responde indicando se a linha está disponível ou ocupada por outro cliente. Se a permissão for concedida, o cliente exibe o conteúdo atual da linha, permite que o usuário digite o novo texto, remove quebras de linha com strcspn(), e envia a linha atualizada de volta ao servidor via TAG\_LIBERAR\_LINHA, liberando automaticamente o bloqueio. Caso a linha esteja ocupada por outro processo, o sistema informa o usuário e sugere tentar novamente mais tarde. Este mecanismo garante a integridade dos dados no ambiente colaborativo, evitando conflitos de edição simultânea e mantendo a consistência do texto compartilhado entre todos os clientes conectados.

Texto

Descrição gerada automaticamente

Esta seção finaliza as funcionalidades do menu do cliente, implementando o sistema de mensagens privadas e controles de sessão. Na opção 2, o cliente pode enviar mensagens privadas para outros processos: solicita o rank do destinatário com validação para garantir que seja um cliente válido (não o servidor nem ele mesmo), captura a mensagem do usuário, remove quebras de linha e envia tudo ao servidor via TAG\_MSG\_PRIVADA\_SEND para roteamento. A opção 3 simplesmente chama a função imprimir\_texto() para exibir o estado atual do documento. A opção 4 implementa o processo de saída graceful: registra a solicitação no log, envia uma mensagem TAG\_SAIR ao servidor para notificar a desconexão e define a flag sair = 1 que interrompe ambas as seções paralelas do OpenMP. O usleep(1000) ao final de cada iteração do menu evita consumo excessivo de CPU, criando uma pequena pausa entre verificações. Este design garante comunicação eficiente entre clientes através do servidor intermediário e permite saída controlada sem deixar processos órfãos.

Texto

Descrição gerada automaticamente

Esta seção implementa a função main que inicializa e coordena toda a aplicação distribuída. O código utiliza MPI\_Init\_thread() em vez do MPI\_Init() padrão para habilitar suporte a múltiplas threads, verificando se o ambiente MPI suporta MPI\_THREAD\_MULTIPLE - essencial para o funcionamento das seções paralelas OpenMP nos clientes. Após obter o rank e tamanho do comunicador, valida se há pelo menos 2 processos (1 servidor + 1 cliente mínimo) para funcionamento adequado. O roteamento de papéis é implementado através de uma simples verificação: o processo com rank 0 executa como servidor chamando servidor\_loop(), enquanto todos os outros processos (rank > 0) executam como clientes através de cliente\_loop(). Esta arquitetura mestre-escravo garante centralização do controle de acesso e sincronização, permitindo que múltiplos clientes colaborem de forma coordenada. O programa finaliza graciosamente com MPI\_Finalize(), liberando todos os recursos MPI alocados durante a execução.

**Prints Execução**

Texto

Descrição gerada automaticamente

Execução inicial de um editor colaborativo distribuído usando MPI e OpenMP. O sistema é composto por três processos: um servidor (P0) e dois clientes (P1 e P2). O servidor é responsável por gerar automaticamente o texto inicial utilizando múltiplas threads com OpenMP. Após gerar o conteúdo, ele envia esse texto para os dois clientes. Cada cliente, ao receber o texto, exibe um menu com opções para editar uma linha, enviar uma mensagem privada, visualizar o texto ou sair. Esse ambiente permite que múltiplos usuários interajam com um mesmo conteúdo de forma simultânea, simulando um editor colaborativo.

Texto

Descrição gerada automaticamente

Observamos os clientes interagindo com o editor colaborativo. O Cliente 1 (P1) escolhe a opção “Editar linha” e solicita a edição da linha 0. O servidor (P0) concede a permissão, e o cliente atualiza o conteúdo da linha com sucesso. Logo depois, o Cliente 2 (P2) também tenta editar a linha 0. No entanto, o servidor detecta que a linha já está sendo modificada por outro cliente e nega o acesso, informando que ela está ocupada. Esse mecanismo de controle garante que duas edições simultâneas não ocorram na mesma linha, preservando a consistência do texto compartilhado entre os usuários.

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

É possível ver a finalização do processo de edição no editor colaborativo. O Cliente 1 (P1), após obter permissão do servidor, altera o conteúdo da linha 0 para “teste” e envia a nova linha ao servidor. O servidor, então, realiza a atualização da linha em tempo real e propaga a modificação para todos os clientes conectados. Logo após, tanto o Cliente 1 quanto o Cliente 2 recebem a notificação de que a linha 0 foi atualizada, exibindo a mensagem:

\*\*\* ATUALIZAÇÃO em tempo real: Linha 00 = "teste" \*\*\*

Isso demonstra que o sistema mantém todos os clientes sincronizados com o conteúdo atualizado por meio da comunicação via MPI\_Isend, garantindo consistência e colaboração eficiente em tempo real.

Interface gráfica do usuário

Descrição gerada automaticamente

O sistema colaborativo demonstra a funcionalidade de mensagens privadas entre clientes. O Cliente 1 (P1) escolhe a opção "Mandar msg privada", informa o destino (P2) e envia a mensagem "ola". O servidor (P0) recebe essa mensagem e a encaminha ao Cliente 2 (P2). O Cliente 2 então recebe a mensagem com a seguinte notificação:

\*\*\* [P2] MENSAGEM PRIVADA de P1: ola \*\*\*

Esse recurso permite que os usuários troquem mensagens diretas durante a colaboração, mantendo a comunicação entre pares dentro do próprio ambiente do editor. A troca é mediada pelo servidor, que recebe e repassa as mensagens para o destino correto.

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

O Cliente 2 (P2) realiza uma nova edição, desta vez na linha 1. Ele solicita a alteração, recebe permissão do servidor e modifica o conteúdo da linha para “teste editando texto”. Em seguida, a linha atualizada é enviada ao servidor. O servidor, por sua vez, atualiza a linha no texto central e propaga a alteração em tempo real para todos os clientes conectados. Ambos os clientes, P1 e P2, recebem a mensagem de atualização com o novo conteúdo da linha 1, garantindo a sincronização do texto entre todos os participantes. Esse momento reforça o funcionamento colaborativo do sistema, onde múltiplos usuários podem editar e visualizar alterações em tempo real, sempre com controle de concorrência e consistência por parte do servidor.

Interface gráfica do usuário

Descrição gerada automaticamente

Temos um cenário com quatro clientes (P1 a P4**)** conectados ao servidor. O Cliente 3 (P3) edita a linha 0 e atualiza o conteúdo para “teste nicolas p1”. O servidor propaga essa atualização em tempo real para todos os clientes. Além disso, o Cliente 4 (P4) utiliza o recurso de mensagens privadas para se comunicar com o Cliente 1 (P1). Primeiro, envia um simples "ola", depois envia outra mensagem solicitando: “poderia editar a linha 2?”. Ambas as mensagens são enviadas por P4 e corretamente entregues a P1 com a identificação do remetente. Isso demonstra que o sistema permite comunicação eficiente entre os usuários, tanto para colaboração quanto para coordenação de ações no texto.

**Conclusão**

Este projeto demonstra, de forma prática e didática, conceitos fundamentais de computação paralela e distribuída, como:

* Gerência de concorrência em recursos compartilhados.
* Comunicação entre processos via MPI.
* Execução multithreaded com OpenMP.
* Coordenação entre múltiplas entidades autônomas sob controle central.

Além disso, simula uma funcionalidade cada vez mais presente em plataformas colaborativas modernas, tornando-se um excelente ponto de partida para experimentos mais sofisticados.

O sistema atendeu a todos os requisitos definidos para a atividade, como execução paralela via OpenMp, comunicação eficiente entre processos MPI e controle de acesso concorrente. O trabalho demonstrou, na prática, os desafios e soluções típicas da computação distribuída, servindo como base para projetos mais complexos, como editores com suporte a múltiplos arquivos, controle de versão ou replicação de servidores.