	<p>Departamento de Engenharias, Arquitetura e Computação</p> <p>Disciplina de Programação Paralela e Distribuída</p> <p>Prof^a.: Ivan L. Süptitz</p> <p>Trabalho – Entrega em 03/07/2025</p>
---	---

Integrantes: Arthur, Guilherme, Nicolás 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

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <mpi.h>
5  #include <omp.h>
6  #include <unistd.h>
7  #include <time.h>
8
9  #define MAX_LINHAS 10
10 #define MAX_TAM_LINHA 256
11 #define MAX_MSG_PRIVADA 200
12 #define LOG_BUFFER_SIZE (MAX_TAM_LINHA * 3)
13
14 #define TAG_SOLICITAR_EDICAO    10
15 #define TAG_RESPOSTA_EDICAO    11
16 #define TAG_ATUALIZACAO_TEXTO  12
17 #define TAG_MSG_PRIVADA_SEND   13
18 #define TAG_LIBERAR_LINHA      14
19 #define TAG_SAIR                15
20 #define TAG_TEXTO_INICIAL      16
21
22 typedef struct {
23     int linha;
24     char conteudo[MAX_TAM_LINHA];
25     int sucesso;
26     int rank_solicitante;
27 } MensagemEdicao;
28
29 typedef struct {
30     int remetente_rank;
31     int destino_rank;
32     char mensagem[MAX_MSG_PRIVADA];
33 } MensagemPrivada;
34
35 char texto[MAX_LINHAS][MAX_TAM_LINHA];
36 int linha_em_uso[MAX_LINHAS];
37 int clientes_ativos;
```

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 distinga 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.

```
38
39 void gerar_texto_com_openmp() {
40     printf("[Servidor] Gerando texto inicial com OpenMP...\n");
41     #pragma omp parallel for
42     for (int i = 0; i < MAX_LINHAS; i++) {
43         sprintf(texto[i], "Linha gerada automaticamente %d (thread %d)", i, omp_get_thread_num());
44         linha_em_uso[i] = 0;
45     }
46     printf("[Servidor] Geração de texto concluída.\n");
47 }
48
49 void imprimir_texto(int rank) {
50     printf("\n--- Texto Atual (P%d) ---\n", rank);
51     for (int i = 0; i < MAX_LINHAS; i++) {
52         if (rank == 0 && linha_em_uso[i] != 0) {
53             printf("[%02d]: %s (OCUPADA por P%d)\n", i, texto[i], linha_em_uso[i]);
54         } else {
55             printf("[%02d]: %s\n", i, texto[i]);
56         }
57     }
58     printf("-----\n");
59 }
60
61 void log_evento(const char* tipo, int rank, const char* mensagem_formatada) {
62     time_t timer;
63     char buffer_tempo[26];
64     struct tm* tm_info;
65
66     time(&timer);
67     tm_info = localtime(&timer);
68     strftime(buffer_tempo, 26, "%Y-%m-%d %H:%M:%S", tm_info);
69
70     printf("[%s] [P%d] %s: %s\n", buffer_tempo, rank, tipo, mensagem_formatada);
71     fflush(stdout);
72 }
```

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.

```

74 void servidor_loop(int rank, int size) {
75     char log_buffer[LOG_BUFFER_SIZE];
76     clientes_ativos = size - 1;
77
78     gerar_texto_com_openmp();
79     log_evento("Servidor", rank, "Texto inicial gerado.");
80     imprimir_texto(rank);
81
82     for (int i = 1; i < size; i++) {
83         MPI_Send(texto, MAX_LINHAS * MAX_TAM_LINHA, MPI_CHAR, i, TAG_TEXTO_INICIAL, MPI_COMM_WORLD);
84         snprintf(log_buffer, LOG_BUFFER_SIZE, "Texto inicial enviado para P%d.", i);
85         log_evento("Servidor", rank, log_buffer);
86     }
87
88     //MPI_Bcast(texto, MAX_LINHAS * MAX_TAM_LINHA, MPI_CHAR, 0, MPI_COMM_WORLD);
89     //log_evento("Servidor", rank, "Texto inicial broadcastado para clientes.");
90
91     MPI_Status status;
92     MensagemEdicao msg_edicao;
93     MensagemPrivada msg_priv;
94
95     while (clientes_ativos > 0) {
96         int flag;
97         MPI_Iprobe(MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &flag, &status);
98
99         if (flag) {
100             if (status.MPI_TAG == TAG_SOLICITAR_EDICAO) {
101                 MPI_Recv(&msg_edicao, sizeof(MensagemEdicao), MPI_BYTE, status.MPI_SOURCE, TAG_SOLICITAR_EDICAO, MPI_COMM_WORLD, &status);
102                 int linha = msg_edicao.linha;
103                 int cliente_rank = status.MPI_SOURCE;
104
105                 if (linha < 0 || linha >= MAX_LINHAS) {
106                     msg_edicao.sucesso = 0;
107                     snprintf(log_buffer, LOG_BUFFER_SIZE, "Solicitação de linha inválida (%d) de P%d.", linha, cliente_rank);
108                     log_evento("Servidor", rank, log_buffer);
109                 } else if (linha_em_uso[linha] != 0) {
110                     msg_edicao.sucesso = 0;

```

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.

```

111         snprintf(log_buffer, LOG_BUFFER_SIZE, "Linha %d negada a P%d (ocupada por P%d).", linha, cliente_rank, linha_em_uso[linha]);
112         log_evento("Servidor", rank, log_buffer);
113     } else {
114         linha_em_uso[linha] = cliente_rank;
115         msg_edicao.sucesso = 1;
116         snprintf(log_buffer, LOG_BUFFER_SIZE, "Linha %d concedida para P%d.", linha, cliente_rank);
117         log_evento("Servidor", rank, log_buffer);
118     }
119     MPI_Send(&msg_edicao, sizeof(MensagemEdicao), MPI_BYTE, cliente_rank, TAG_RESPOSTA_EDICAO, MPI_COMM_WORLD);
120
121 } else if (status.MPI_TAG == TAG_LIBERAR_LINHA) {
122     MPI_Recv(&msg_edicao, sizeof(MensagemEdicao), MPI_BYTE, status.MPI_SOURCE, TAG_LIBERAR_LINHA, MPI_COMM_WORLD, &status);
123     int linha = msg_edicao.linha;
124     int cliente_rank = status.MPI_SOURCE;
125
126     if (linha >= 0 && linha < MAX_LINHAS && linha_em_uso[linha] == cliente_rank) {
127         strcpy(texto[linha], msg_edicao.conteudo);
128         linha_em_uso[linha] = 0;
129
130         snprintf(log_buffer, LOG_BUFFER_SIZE, "Linha %d atualizada por P%d: \"%s\"", linha, cliente_rank, msg_edicao.conteudo);
131         log_evento("Servidor", rank, log_buffer);
132
133         MPI_Request request_atualizacao[size];
134         int num_requests = 0;
135
136         for (int i = 1; i < size; i++) {
137             MPI_Isend(&msg_edicao, sizeof(MensagemEdicao), MPI_BYTE, i, TAG_ATUALIZACAO_TEXTO, MPI_COMM_WORLD, &request_atualizacao[num_r
138             num_requests++;
139         }
140         snprintf(log_buffer, LOG_BUFFER_SIZE, "Atualização da linha %d iniciada para todos os clientes (via Isend).", linha);
141         log_evento("Servidor", rank, log_buffer);

```

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.

```
142 } else {
143     snprintf(log_buffer, LOG_BUFFER_SIZE, "Erro: P%d tentou liberar ou atualizar linha %d de forma inválida (não era o proprietário)", rank, linha);
144     log_evento("Servidor", rank, log_buffer);
145 }
146
147 } else if (status.MPI_TAG == TAG_MSG_PRIVADA_SEND) {
148     MPI_Recv(&msg_priv, sizeof(MensagemPrivada), MPI_BYTE, status.MPI_SOURCE, TAG_MSG_PRIVADA_SEND, MPI_COMM_WORLD, &status);
149
150     snprintf(log_buffer, LOG_BUFFER_SIZE, "Mensagem privada de P%d para P%d: \"%s\"", msg_priv.remetente_rank, msg_priv.destino_rank, msg_priv.texto);
151     log_evento("Servidor", rank, log_buffer);
152
153     if (msg_priv.destino_rank > 0 && msg_priv.destino_rank < size && msg_priv.destino_rank != rank) {
154         MPI_Send(&msg_priv, sizeof(MensagemPrivada), MPI_BYTE, msg_priv.destino_rank, TAG_MSG_PRIVADA_SEND, MPI_COMM_WORLD);
155         snprintf(log_buffer, LOG_BUFFER_SIZE, "Mensagem encaminhada de P%d para P%d.", msg_priv.remetente_rank, msg_priv.destino_rank);
156         log_evento("Servidor", rank, log_buffer);
157     } else {
158         snprintf(log_buffer, LOG_BUFFER_SIZE, "Destinatário inválido (P%d) para mensagem de P%d. Mensagem descartada.", msg_priv.destino_rank, msg_priv.remetente_rank);
159         log_evento("Servidor", rank, log_buffer);
160     }
161 }
162
163 } else if (status.MPI_TAG == TAG_SAIR) {
164     MPI_Recv(&msg_saida, sizeof(MensagemEdicao), MPI_BYTE, status.MPI_SOURCE, TAG_SAIR, MPI_COMM_WORLD, &status);
165     clientes_ativos--;
166     snprintf(log_buffer, LOG_BUFFER_SIZE, "P%d solicitou sair. Clientes ativos restantes: %d.", status.MPI_SOURCE, clientes_ativos);
167     log_evento("Servidor", rank, log_buffer);
168 }
```

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.

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.

```
238 int linha;
239 printf("[P%d] Número da linha para editar (0 a %d): ", rank, MAX_LINHAS - 1);
240 if (fgets(input_buffer, sizeof(input_buffer), stdin) == NULL || sscanf(input_buffer, "%d", &linha) != 1) continue;
241 if (linha < 0 || linha >= MAX_LINHAS) continue;
242
243 MensagemEdicao solicitacao;
244 solicitacao.linha = linha;
245 solicitacao.rank_solicitante = rank;
246 MPI_Send(&solicitacao, sizeof(MensagemEdicao), MPI_BYTE, 0, TAG_SOLICITAR_EDICAO, MPI_COMM_WORLD);
247 log_evento("Cliente", rank, "Solicitou edição da linha.");
248
249 MPI_Recv(&solicitacao, sizeof(MensagemEdicao), MPI_BYTE, 0, TAG_RESPOSTA_EDICAO, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
250
251 if (solicitacao.sucesso) {
252     printf("[P%d] ✓ Permissão concedida para linha %02d! conteúdo atual: \"%s\"\n", rank, linha, texto[linha]);
253     printf("[P%d] Novo conteúdo para linha %02d: ", rank, linha);
254
255     char nova_linha[MAX_TAM_LINHA];
256     fgets(nova_linha, MAX_TAM_LINHA, stdin);
257     nova_linha[strcspn(nova_linha, "\n")] = 0;
258
259     strcpy(solicitacao.conteudo, nova_linha);
260     solicitacao.linha = linha;
261     MPI_Send(&solicitacao, sizeof(MensagemEdicao), MPI_BYTE, 0, TAG_LIBERAR_LINHA, MPI_COMM_WORLD);
262     log_evento("Cliente", rank, "Enviou linha atualizada.");
263     printf("[P%d] ✓ Linha enviada para o servidor!\n", rank);
264 } else {
265     printf("[P%d] X Linha %02d está ocupada. Tente novamente depois.\n", rank, linha);
266 }
267
```

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.

```

268         } else if (opcao == 2) {
269             int destino;
270             printf("[P%d] Rank do destinatário (1 a %d): ", rank, size - 1);
271             if (fgets(input_buffer, sizeof(input_buffer), stdin) == NULL || sscanf(input_buffer, "%d", &destino) != 1) continue;
272             if (destino <= 0 || destino >= size || destino == rank) continue;
273
274             MensagemPrivada msg;
275             msg.remetente_rank = rank;
276             msg.destino_rank = destino;
277
278             printf("[P%d] Digite a mensagem: ", rank);
279             fgets(msg.mensagem, MAX_MSG_PRIVADA, stdin);
280             msg.mensagem[strcspn(msg.mensagem, "\n")] = 0;
281
282             MPI_Send(&msg, sizeof(MensagemPrivada), MPI_BYTE, 0, TAG_MSG_PRIVADA_SEND, MPI_COMM_WORLD);
283             log_evento("cliente", rank, "Mensagem privada enviada.");
284             printf("[P%d] ✓ Mensagem enviada!\n", rank);
285
286         } else if (opcao == 3) {
287             imprimir_texto(rank);
288
289         } else if (opcao == 4) {
290             log_evento("Cliente", rank, "Solicitando sair...");
291             MensagemEdicao sair_msg;
292             sair_msg.rank_solicitante = rank;
293             MPI_Send(&sair_msg, sizeof(MensagemEdicao), MPI_BYTE, 0, TAG_SAIR, MPI_COMM_WORLD);
294             sair = 1;
295         }
296         usleep(1000);
297     }
298 }
299
300 }

```

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.

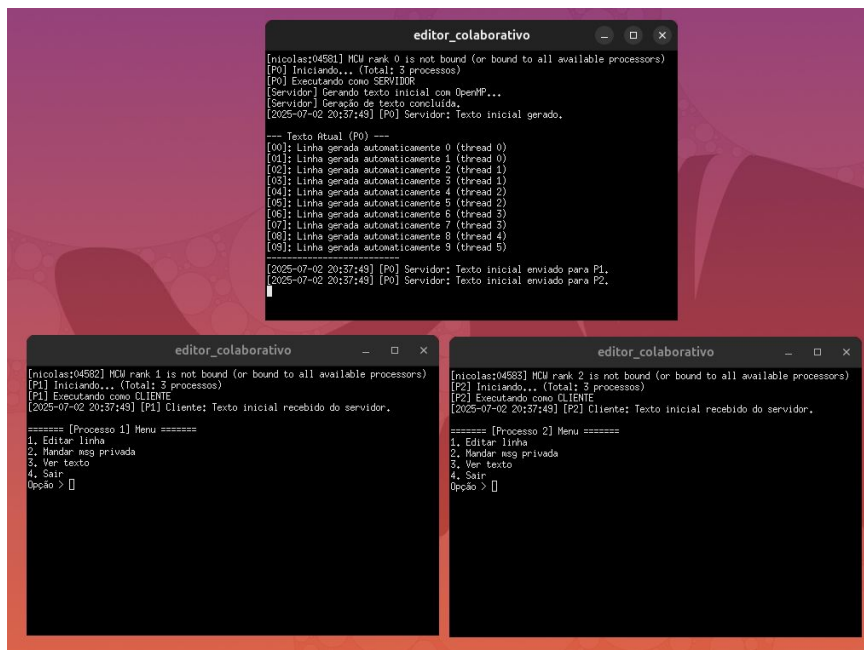
```

302 int main(int argc, char** argv) {
303     int rank, size;
304
305     //MPI_Init(&argc, &argv);
306     int provided;
307     MPI_Init_thread(&argc, &argv, MPI_THREAD_MULTIPLE, &provided);
308     if (provided < MPI_THREAD_MULTIPLE) {
309         printf("Erro: MPI não suporta múltiplas threads nesse ambiente.\n");
310         MPI_Finalize();
311         return 1;
312     }
313     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
314     MPI_Comm_size(MPI_COMM_WORLD, &size);
315
316     if (size < 2) {
317         printf("Este programa requer no mínimo 2 processos MPI (1 servidor e pelo menos 1 cliente).\n");
318         MPI_Finalize();
319         return 1;
320     }
321
322     printf("[P%d] Iniciando... (Total: %d processos)\n", rank, size);
323
324     if (rank == 0) {
325         printf("[P%d] Executando como SERVIDOR\n", rank);
326         servidor_loop(rank, size);
327     } else {
328         printf("[P%d] Executando como CLIENTE\n", rank);
329         cliente_loop(rank, size);
330     }
331
332     printf("[P%d] Finalizando...\n", rank);
333     MPI_Finalize();
334     return 0;
335 }
336

```

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



```
editor_colaborativo
[nicolas:04581] MCV rank 0 is not bound (or bound to all available processors)
[P0] Iniciando... (Total: 3 processos)
[P0] Executando como SERVIDOR
[Servidor] Gerando texto inicial com OpenMP...
[Servidor] Gerado de texto concluído.
[2025-07-02 20:37:49] [P0] Servidor: Texto inicial gerado.

--- Texto atual (P0) ---
[00]: Linha gerada automaticamente 0 (thread 0)
[01]: Linha gerada automaticamente 1 (thread 0)
[02]: Linha gerada automaticamente 2 (thread 1)
[03]: Linha gerada automaticamente 3 (thread 1)
[04]: Linha gerada automaticamente 4 (thread 2)
[05]: Linha gerada automaticamente 5 (thread 2)
[06]: Linha gerada automaticamente 6 (thread 3)
[07]: Linha gerada automaticamente 7 (thread 3)
[08]: Linha gerada automaticamente 8 (thread 4)
[09]: Linha gerada automaticamente 9 (thread 4)

[2025-07-02 20:37:49] [P0] Servidor: Texto inicial enviado para P1.
[2025-07-02 20:37:49] [P0] Servidor: Texto inicial enviado para P2.

editor_colaborativo
[nicolas:04582] MCV rank 1 is not bound (or bound to all available processors)
[P1] Iniciando... (Total: 3 processos)
[P1] Executando como CLIENTE
[2025-07-02 20:37:49] [P1] Cliente: Texto inicial recebido do servidor.

===== [Processo 1] Menu =====
1. Editar linha
2. Mandar msg privada
3. Ver texto
4. Sair
Opção > []

editor_colaborativo
[nicolas:04583] MCV rank 2 is not bound (or bound to all available processors)
[P2] Iniciando... (Total: 3 processos)
[P2] Executando como CLIENTE
[2025-07-02 20:37:49] [P2] Cliente: Texto inicial recebido do servidor.

===== [Processo 2] Menu =====
1. Editar linha
2. Mandar msg privada
3. Ver texto
4. Sair
Opção > []
```

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.

```
editor_colaborativo
[nicolas:04581] MCM rank 0 is not bound (or bound to all available processors)
[P0] Iniciando... (Total: 3 processos)
[P0] Executando como SERVIDOR
[Servidor] Gerando texto inicial com OpenMP...
[Servidor] Geração do texto concluída.
[2025-07-02 20:37:49] [P0] Servidor: Texto inicial gerado.

--- Texto Atual (P0) ---
[00]: Linha gerada automaticamente 0 (thread 0)
[01]: Linha gerada automaticamente 1 (thread 0)
[02]: Linha gerada automaticamente 2 (thread 1)
[03]: Linha gerada automaticamente 3 (thread 1)
[04]: Linha gerada automaticamente 4 (thread 2)
[05]: Linha gerada automaticamente 5 (thread 2)
[06]: Linha gerada automaticamente 6 (thread 3)
[07]: Linha gerada automaticamente 7 (thread 3)
[08]: Linha gerada automaticamente 8 (thread 4)
[09]: Linha gerada automaticamente 9 (thread 5)
-----
[2025-07-02 20:37:49] [P0] Servidor: Texto inicial enviado para P1.
[2025-07-02 20:37:49] [P0] Servidor: Texto inicial enviado para P2.
[2025-07-02 20:42:11] [P0] Servidor: Linha 0 concedida para P1.
[2025-07-02 20:42:24] [P0] Servidor: Linha 0 negada a P2 (ocupada por P1).
█

editor_colaborativo
===== [Processo 1] Menu =====
1. Editar linha
2. Mandar msg privada
3. Ver texto
4. Sair
Opção > 1

--- Texto Atual (sua cópia) ---
[00]: Linha gerada automaticamente 0 (thread 0)
[01]: Linha gerada automaticamente 1 (thread 0)
[02]: Linha gerada automaticamente 2 (thread 1)
[03]: Linha gerada automaticamente 3 (thread 1)
[04]: Linha gerada automaticamente 4 (thread 2)
[05]: Linha gerada automaticamente 5 (thread 2)
[06]: Linha gerada automaticamente 6 (thread 3)
[07]: Linha gerada automaticamente 7 (thread 3)
[08]: Linha gerada automaticamente 8 (thread 4)
[09]: Linha gerada automaticamente 9 (thread 5)
[P1] Número da linha para editar (0 a 9): 0
[2025-07-02 20:42:11] [P1] Cliente: Solicitou edição da linha.
[P1] ✓ Permissão concedida para linha 00! Conteúdo atual: "Linha gerada automaticamente 0 (thread 0)"
[P1] Novo conteúdo para linha 00: █

editor_colaborativo
4. Sair
Opção > 1

--- Texto Atual (sua cópia) ---
[00]: Linha gerada automaticamente 0 (thread 0)
[01]: Linha gerada automaticamente 1 (thread 0)
[02]: Linha gerada automaticamente 2 (thread 1)
[03]: Linha gerada automaticamente 3 (thread 1)
[04]: Linha gerada automaticamente 4 (thread 2)
[05]: Linha gerada automaticamente 5 (thread 2)
[06]: Linha gerada automaticamente 6 (thread 3)
[07]: Linha gerada automaticamente 7 (thread 3)
[08]: Linha gerada automaticamente 8 (thread 4)
[09]: Linha gerada automaticamente 9 (thread 5)
[P2] Número da linha para editar (0 a 9): 0
[2025-07-02 20:42:24] [P2] Cliente: Solicitou edição da linha.
[P2] ✗ Linha 00 está ocupada. Tente novamente depois.

===== [Processo 2] Menu =====
1. Editar linha
2. Mandar msg privada
3. Ver texto
4. Sair
Opção > █
```

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.

```
editor_colaborativo
[Servidor] Gerando texto inicial com OpenMP...
[Servidor] Geração de texto concluída.
[2025-07-02 20:57:49] [P0] Servidor: Texto inicial gerado.

--- Texto Atual (P0) ---
[00]: Linha gerada automaticamente 0 (thread 0)
[01]: Linha gerada automaticamente 1 (thread 0)
[02]: Linha gerada automaticamente 2 (thread 1)
[03]: Linha gerada automaticamente 3 (thread 1)
[04]: Linha gerada automaticamente 4 (thread 2)
[05]: Linha gerada automaticamente 5 (thread 2)
[06]: Linha gerada automaticamente 6 (thread 3)
[07]: Linha gerada automaticamente 7 (thread 3)
[08]: Linha gerada automaticamente 8 (thread 4)
[09]: Linha gerada automaticamente 9 (thread 5)

[2025-07-02 20:37:49] [P0] Servidor: Texto inicial enviado para P1.
[2025-07-02 20:37:49] [P0] Servidor: Texto inicial enviado para P2.
[2025-07-02 20:42:11] [P0] Servidor: Linha 0 concedida para P1.
[2025-07-02 20:42:24] [P0] Servidor: Linha 0 negada a P2 (ocupada por P1).
[2025-07-02 20:42:48] [P0] Servidor: Linha 0 atualizada por P1: "teste"
[2025-07-02 20:42:48] [P0] Servidor: Atualização da linha 0 iniciada para todos
os clientes (via Isend).
[]

editor_colaborativo
[03]: Linha gerada automaticamente 3 (thread 1)
[04]: Linha gerada automaticamente 4 (thread 2)
[05]: Linha gerada automaticamente 5 (thread 2)
[06]: Linha gerada automaticamente 6 (thread 3)
[07]: Linha gerada automaticamente 7 (thread 3)
[08]: Linha gerada automaticamente 8 (thread 4)
[09]: Linha gerada automaticamente 9 (thread 5)
[P1] Número da linha para editar (0 a 9): 0
[2025-07-02 20:42:11] [P1] Cliente: Solicitou edição da linha.
[P1] ✓ Permissão concedida para linha 0! Conteúdo atual: "Linha gerada automati
camente 0 (thread 0)"
[P1] Novo conteúdo para linha 00: teste
[2025-07-02 20:42:48] [P1] Cliente: Enviou linha atualizada.
[P1] ✓ Linha enviada para o servidor!

===== [Processo 1] Menu =====
1. Editar linha
2. Mandar msg privada
3. Ver texto
4. Sair
Opção >
*** [P1] ATUALIZAÇÃO em tempo real: Linha 00 = "teste" ***
[2025-07-02 20:42:48] [P1] Cliente: Linha 0 atualizada via servidor.
[]

editor_colaborativo
Opção >
*** [P2] ATUALIZAÇÃO em tempo real: Linha 00 = "teste" ***
[2025-07-02 20:42:48] [P2] Cliente: Linha 0 atualizada via servidor.
3

--- Texto Atual (P2) ---
[00]: teste
[01]: Linha gerada automaticamente 1 (thread 0)
[02]: Linha gerada automaticamente 2 (thread 1)
[03]: Linha gerada automaticamente 3 (thread 1)
[04]: Linha gerada automaticamente 4 (thread 2)
[05]: Linha gerada automaticamente 5 (thread 2)
[06]: Linha gerada automaticamente 6 (thread 3)
[07]: Linha gerada automaticamente 7 (thread 3)
[08]: Linha gerada automaticamente 8 (thread 4)
[09]: Linha gerada automaticamente 9 (thread 5)

===== [Processo 2] Menu =====
1. Editar linha
2. Mandar msg privada
3. Ver texto
4. Sair
Opção > []
```

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.


```
editor_colaborativo
[2025-07-02 20:37:49] [P0] Servidor: Texto inicial gerado.

--- Texto Atual (P0) ---
[00]: Linha gerada automaticamente 0 (thread 0)
[01]: Linha gerada automaticamente 1 (thread 0)
[02]: Linha gerada automaticamente 2 (thread 1)
[03]: Linha gerada automaticamente 3 (thread 1)
[04]: Linha gerada automaticamente 4 (thread 2)
[05]: Linha gerada automaticamente 5 (thread 2)
[06]: Linha gerada automaticamente 6 (thread 3)
[07]: Linha gerada automaticamente 7 (thread 3)
[08]: Linha gerada automaticamente 8 (thread 4)
[09]: Linha gerada automaticamente 9 (thread 5)

[2025-07-02 20:37:49] [P0] Servidor: Texto inicial enviado para P1.
[2025-07-02 20:37:49] [P0] Servidor: Texto inicial enviado para P2.
[2025-07-02 20:42:11] [P0] Servidor: Linha 0 concedida para P1.
[2025-07-02 20:42:24] [P0] Servidor: Linha 0 negada a P2 (ocupada por P1).
[2025-07-02 20:42:48] [P0] Servidor: Linha 0 atualizada por P1: "teste"
[2025-07-02 20:42:48] [P0] Servidor: Atualização da linha 0 iniciada para todos os clientes (via Isend).
[2025-07-02 20:43:57] [P0] Servidor: Mensagem privada de P1 para P2: "ola"
[2025-07-02 20:43:57] [P0] Servidor: Mensagem encaminhada de P1 para P2.

editor_colaborativo
[P1] Novo conteúdo para linha 00: teste
[2025-07-02 20:42:48] [P1] Cliente: Enviou linha atualizada.
[P1] ✓ Linha enviada para o servidor!

===== [Processo 1] Menu =====
1. Editar linha
2. Mandar msg privada
3. Ver texto
4. Sair
Opção >
*** [P1] ATUALIZAÇÃO em tempo real: Linha 00 = "teste" ***
[2025-07-02 20:42:48] [P1] Cliente: Linha 0 atualizada via servidor.
2
[P1] Rank do destinatário (1 a 2): 2
[P1] Digite a mensagem: ola
[2025-07-02 20:43:57] [P1] Cliente: Mensagem privada enviada.
[P1] ✓ Mensagem enviada!

===== [Processo 1] Menu =====
1. Editar linha
2. Mandar msg privada
3. Ver texto
4. Sair
Opção >

editor_colaborativo
3
--- Texto Atual (P2) ---
[00]: teste
[01]: Linha gerada automaticamente 1 (thread 0)
[02]: Linha gerada automaticamente 2 (thread 1)
[03]: Linha gerada automaticamente 3 (thread 1)
[04]: Linha gerada automaticamente 4 (thread 2)
[05]: Linha gerada automaticamente 5 (thread 2)
[06]: Linha gerada automaticamente 6 (thread 3)
[07]: Linha gerada automaticamente 7 (thread 3)
[08]: Linha gerada automaticamente 8 (thread 4)
[09]: Linha gerada automaticamente 9 (thread 5)

===== [Processo 2] Menu =====
1. Editar linha
2. Mandar msg privada
3. Ver texto
4. Sair
Opção >
*** [P2] MENSAGEM PRIVADA de P1: ola ***
[2025-07-02 20:43:57] [P2] Cliente: Mensagem privada recebida de P1.

```

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.

```
editor_colaborativo
[02]: Linha gerada automaticamente 2 (thread 1)
[03]: Linha gerada automaticamente 3 (thread 1)
[04]: Linha gerada automaticamente 4 (thread 2)
[05]: Linha gerada automaticamente 5 (thread 2)
[06]: Linha gerada automaticamente 6 (thread 3)
[07]: Linha gerada automaticamente 7 (thread 3)
[08]: Linha gerada automaticamente 8 (thread 4)
[09]: Linha gerada automaticamente 9 (thread 5)
-----
[2025-07-02 20:37:49] [P0] Servidor: Texto inicial enviado para P1.
[2025-07-02 20:37:49] [P0] Servidor: Texto inicial enviado para P2.
[2025-07-02 20:42:11] [P0] Servidor: Linha 0 concedida para P1.
[2025-07-02 20:42:24] [P0] Servidor: Linha 0 negada a P2 (ocupada por P1).
[2025-07-02 20:42:48] [P0] Servidor: Linha 0 atualizada por P1: "teste"
[2025-07-02 20:42:48] [P0] Servidor: Atualização da linha 0 iniciada para todos os clientes (via Isend).
[2025-07-02 20:43:57] [P0] Servidor: Mensagem privada de P1 para P2: "ola"
[2025-07-02 20:43:57] [P0] Servidor: Mensagem encaminhada de P1 para P2.
[2025-07-02 20:44:47] [P0] Servidor: Linha 1 concedida para P2.
[2025-07-02 20:45:11] [P0] Servidor: Linha 1 atualizada por P2: "teste editando texto"
[2025-07-02 20:45:11] [P0] Servidor: Atualização da linha 1 iniciada para todos os clientes (via Isend).
[]

editor_colaborativo
Opção > 3
--- Texto Atual (P1) ---
[00]: teste
[01]: Linha gerada automaticamente 1 (thread 0)
[02]: Linha gerada automaticamente 2 (thread 1)
[03]: Linha gerada automaticamente 3 (thread 1)
[04]: Linha gerada automaticamente 4 (thread 2)
[05]: Linha gerada automaticamente 5 (thread 2)
[06]: Linha gerada automaticamente 6 (thread 3)
[07]: Linha gerada automaticamente 7 (thread 3)
[08]: Linha gerada automaticamente 8 (thread 4)
[09]: Linha gerada automaticamente 9 (thread 5)
-----
===== [Processo 1] Menu =====
1. Editar linha
2. Mandar msg privada
3. Ver texto
4. Sair
Opção >
*** [P1] ATUALIZAÇÃO em tempo real: Linha 01 = "teste editando texto" ***
[2025-07-02 20:45:11] [P1] Cliente: Linha 1 atualizada via servidor.
[]

editor_colaborativo
[03]: Linha gerada automaticamente 3 (thread 1)
[04]: Linha gerada automaticamente 4 (thread 2)
[05]: Linha gerada automaticamente 5 (thread 2)
[06]: Linha gerada automaticamente 6 (thread 3)
[07]: Linha gerada automaticamente 7 (thread 3)
[08]: Linha gerada automaticamente 8 (thread 4)
[09]: Linha gerada automaticamente 9 (thread 5)
[P2] Número da linha para editar (0 a 9): 1
[2025-07-02 20:44:47] [P2] Cliente: Solicitou edição da linha.
[P2] ✓ Permissão concedida para linha 01! Conteúdo atual: "Linha gerada automati
camente 1 (thread 0)"
[P2] Novo conteúdo para linha 01: teste editando texto
[2025-07-02 20:45:11] [P2] Cliente: Enviou linha atualizada.
[P2] ✓ Linha enviada para o servidor!
===== [Processo 2] Menu =====
1. Editar linha
2. Mandar msg privada
3. Ver texto
4. Sair
Opção >
*** [P2] ATUALIZAÇÃO em tempo real: Linha 01 = "teste editando texto" ***
[2025-07-02 20:45:11] [P2] Cliente: Linha 1 atualizada via servidor.
[]
```

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.

The image displays four terminal windows titled 'editor_colaborativo', each showing a different view of the system's operation. The top-left window shows a list of 10 lines being generated automatically by threads 0 through 5. The top-right window shows a menu with options: 1. Editar linha, 2. Mandar msg privada, 3. Ver texto, 4. Sair. It also displays a message from Client 1 (P1) asking to edit line 2. The bottom-left window shows a message from Client 4 (P4) asking to edit line 2, followed by a confirmation message from the server. The bottom-right window shows a message from Client 4 (P4) asking to edit line 2, followed by a confirmation message from the server.

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.