Relatório - Trabalho Prático - Etapa 1: Sistema PIX Simulado

Disciplina: INF01151 - Sistemas Operacionais II

Grupo: Augusto Mattei Grohmann, Henrique Anderson Knorst, João Gabriel Eberhardt Pereira

1. Visão Geral

Implementação da Etapa 1 de um serviço distribuído de transferência de valores em `C`, usando UDP e `Pthreads`. O sistema simula um PIX, com foco na consistência de saldos e processamento concorrente de requisições.

2. Implementação dos Subserviços

Descoberta: O cliente envia broadcast UDP (`TYPE_DESC`); o servidor responde unicast (`TYPE_DESC_ACK`) e registra o cliente (protegido por `client_table_mutex`), inicializando o saldo **100**.

Processamento:
Servidor: Recebe `TYPE_REQ` e cria uma thread (`process_request`) por requisição. A thread valida o `seqn` (tratando duplicatas e fora de ordem, reenviando ACK antigo), processa transferências (verifica saldo, debita/credita), trata `value = 0` como consulta, atualiza `last_req` e os totais (`num_transactions`, etc., protegidos por `stats_mutex`), envia `TYPE_REQ_ACK` e enfileira o log para a interface. Utiliza bloqueio fino (`client_lock` por cliente) para permitir transferências concorrentes.

Cliente: A thread principal (rede) consome comandos da `input_thread_func`, envia `TYPE_REQ` com `seqn` incrementado, gera retransmissão (`MAX_RETRIES`) com timeout de 10 ms (`select` ou `pthread_cond_timedwait`) e valida o `seqn` do `TYPE_REQ_ACK` recebido.

Interface:
Servidor: Implementa o modelo Leitor/Escritor. A thread `interface_thread` (Leitor) bloqueia (`pthread_cond_wait`) até que threads de processamento (Escritores) adicionem logs formatados (linha única) à fila (`push_log`) e sinalizem (`pthread_cond_signal`).

Cliente: Implementa duas threads obrigatórias: `input_thread_func` lê `stdin` (`scanf`); a thread principal gerencia a rede e imprime logs recebidos da thread de entrada via buffer sincronizado (`resp_mutex`/`resp_cond`). Logs de descoberta e ACK seguem formato exato conforme a especificação.

3. Sincronização

Servidor: Modelo Leitor/Escritor (`log_mutex`/`update_cond`) para a interface. Mutexes separados para estatísticas (`stats_mutex`) e tabela de clientes (`client_table_mutex`). Bloqueio fino (`client_lock` por cliente) nas transferências para permitir paralelismo seguro.

Cliente: Uso de mutexes e variáveis de condição (`req_mutex`/`req_cond`, `resp_mutex`/`resp_cond`) para comunicação segura entre as threads de entrada, rede e saída.

4. Estruturas e Funções Chave

Estruturas: `packet` (formato de mensagem UDP), `client_data` (dados do cliente no servidor, incluindo `client_lock`).

Funções do Servidor: `main` (setup, dispatch), `process_request` (lógica da requisição), `interface_thread` (impressão de logs), `push_log` (enfileirar log).

Funções do Cliente: `main` (setup, descoberta, thread de rede), `input_thread_func` (leitura `stdin`), `output_thread_func` (impressão `stdout`).

5. Comunicação e Robustez

Uso de sockets UDP (`socket`, `bind`, `sendto`, `recvfrom`) com broadcast (`setsockopt`) para descoberta. Conversão de ordem de bytes (`htons`, `ntohl`) utilizada corretamente. O cliente implementa timeout (10 ms) e retransmissão. O servidor detecta e trata duplicatas e pacotes fora de ordem conforme o protocolo.

6. Problemas Encontrados

Bloqueio de broadcast por firewall; bugs de lógica (transferência para si, `value = 0`); desafios de concorrência (condições de corrida, deadlocks evitados com ordem de bloqueio); complexidade na sincronização inter-thread no cliente; garantia de conformidade exata dos logs.

7. Divisão de Tarefas

Todos os membros do grupo contribuíram em todas partes quando necessário, incluindo o relatório, mas a divisão principal:

Augusto Mattei Grohmann: construção do lado cliente da aplicação.

Henrique Anderson Knorst: construção do lado servidor da aplicação.

João Gabriel Eberhardt Pereira: construção de ambos os lados em conjunto com os outros.

8. Conclusão

A Etapa 1 foi implementada com sucesso, atendendo a todos os requisitos funcionais e de arquitetura, incluindo concorrência, sincronização avançada e robustez de rede. O sistema está pronto para a Etapa 2.