INF01151 - Sistemas Operacionais II - Trabalho Prático - Etapa 1

1 Especificação do Trabalho Prático

A proposta de trabalho prático é implementar um serviço distribuído de transferência de valores entre clientes, simulando um sistema de pagamento similar ao PIX. O objetivo do serviço é receber requisições de clientes diversos contendo um valor a ser transferido para outro cliente, processar cada requisição, atualizar os saldos dos clientes envolvidos e o saldo total do banco, e manter um histórico das transações. Cada requisição conterá um valor inteiro positivo e uma conta destino, que serão lidos pelo cliente a partir da entrada padrão. O servidor deverá processar as transações de forma concorrente/paralela, usando threads (uma thread para processar cada requisição recebida).

O principal critério de avaliação será a correta atualização dos saldos dos clientes, saldo total do banco, e a consistência do histórico de transações. Um critério secundário de avaliação será o tempo de resposta para o servidor receber, processar e responder um grande volume de requisições (p. ex.: 10.000+ transações) recebidas de clientes diversos.

A proposta deverá ser desenvolvida em duas etapas. A primeira etapa compreenderá funcionalidades que dependerão de tópicos como threads, processos, comunicação e sincronização para serem implementadas e a segunda compreenderá a distribuição do nó servidor para garantir tolerância a falhas. O projeto deverá executar em ambientes Linux, nos computadores dos laboratórios de ensino do INF-UFRGS, mesmo que tenha sido desenvolvido em outras plataformas/outros computadores. O projeto deverá ser implementado em C/C++, usando exclusivamente a API User Datagram Protocol (UDP) para comunicação inter-processos.

2 Descrição do Serviço e Funcionalidades Básicas

O projeto compreenderá dois programas distintos, sendo um cliente e um servidor. O servidor executará em uma única máquina, e receberá requisições de clientes diversos. Os clientes executarão em múltiplas estações (um cliente por estação).

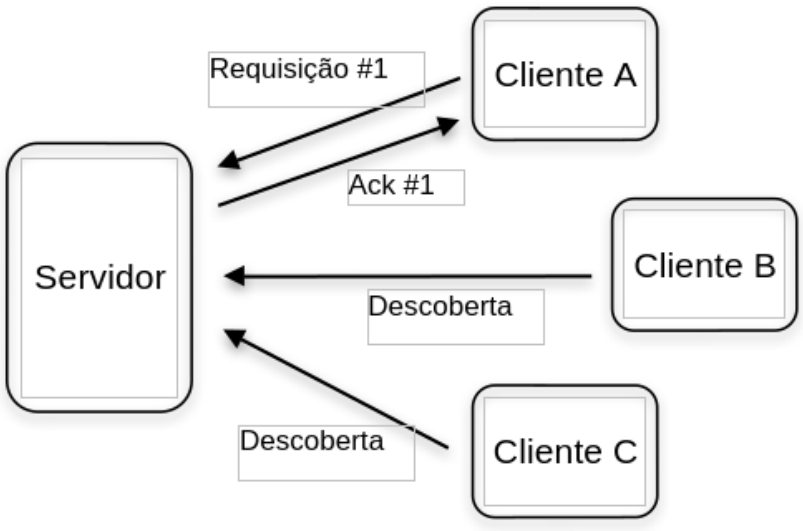

Figura 1: Dinâmica de funcionamento do serviço distribuído de agregação de dados.

A Figura 1 ilustra a dinâmica de funcionamento do serviço, que compreende duas fases: descoberta e processamento. Na fase de descoberta, o cliente recém inicializado envia na rede uma mensagem do tipo DESCOBERTA (em modo de difusão, ou broadcast), para "descobrir" em qual endereço o servidor está escutando por requisições. O servidor, ao receber uma mensagem de descoberta 1) responde ao cliente (em modo ponto-a-ponto, ou unicast), confirmando assim em qual endereço está escutando por requisições e 2) registra o novo cliente em uma tabela de participantes, inicializando seu saldo com um valor N de reais (p. ex: 100 reais).

Finalizada a fase de descoberta, o cliente passa para a fase de processamento. Nesta fase, o cliente envia um conjunto de mensagens do tipo REQUISIÇÃO, uma por vez. A requisição conterá o identificador da mensagem, o endereço do cliente destino e um valor inteiro a ser transferido. O servidor recebe cada mensagem, faz o processamento (i.e., verifica se o cliente origem possui saldo suficiente, debita o valor da conta do cliente origem, credita o valor na conta do cliente destino, atualiza o histórico de transações e o saldo total do banco), e envia uma resposta (acknowledgement, ou ack). A resposta enviada pelo servidor confirma que a mensagem de requisição foi recebida e processada, e informa qual o novo saldo do cliente origem. Caso o cliente origem não possua saldo suficiente, a transação não ocorre e o servidor informa ao cliente pagante que não possui saldo. O cliente somente prossegue com o envio da próxima requisição ao receber uma confirmação de que a requisição anterior foi processada. O cliente deverá aguardar um tempo e, caso não receba uma confirmação, deverá reenviar a requisição. Sugere-se fortemente à equipe organizar cada subserviço do sistema em módulos para implementá-las, sobretudo para facilitar o processo de desenvolvimento da Etapa 2 do Trabalho Prático.

- Subserviço de Descoberta, para identificar em qual endereço IPv4 (Internet Protocol v4) o servidor está esperando por requisições. A equipe pode assumir que o servidor estará executando no mesmo segmento físico e lógico de rede. Assim, sugere-se usar *sockets* UDP configurados com a opção broadcast para implementar as rotinas de descoberta. Na primeira parte do trabalho prático, este subserviço (i) operará de forma passiva (isto é, recebendo e respondendo mensagens em broadcast do tipo DESCOBERTA) no servidor e (ii) operará de forma ativa (isto é, enviando mensagens de descoberta) nos clientes;
- Subserviço de Processamento, para receber as requisições dos clientes (contendo o valor a ser transferido), processá-las (verificar saldo, debitar e creditar valores, atualizar histórico) e enviar as respostas (confirmar que a requisição foi recebida, processada, e o novo saldo do cliente origem). Este subserviço:
  1. operará de forma ativa (isto é, enviando mensagens de requisições) nos clientes e
  2. operará de forma passiva (isto é, recebendo e respondendo requisições) no servidor;
- Subserviço de Interface, para exibir as mensagens de envio (e reenvio) de requisições e recebimento das respostas (no cliente) e para exibir as mensagens de recebimento e resultado do processamento das requisições (no servidor), além de acusar mensagens recebidas fora de ordem/duplicadas.

O serviço deve garantir, ainda:

- Consistência nas estruturas de armazenamento: As estruturas de armazenamento de dados no serviço devem ser mantidas em um estado consistente. Por exemplo, as informações sobre os saldos dos clientes, saldo total, e o histórico de transações devem ser mantidas de forma consistente.
- Persistência de dados no servidor: As estruturas de armazenamento de dados no serviço devem ser mantidas somente em memória. Na Etapa 2 do Trabalho Prático, mecanismos de replicação serão usados para garantir a persistência consistente das estruturas em múltiplas estações servidoras.

3 Sobre as Requisições

Cada estação cliente deverá ler da entrada padrão (por exemplo, o stdin no C ou std::cin no C++) cada requisição, uma por vez, para enviá-las ao servidor. Cada requisição é formada por um número identificador da requisição (que será incrementado em um a cada nova requisição enviada), o endereço IP do cliente destino e um valor inteiro positivo (que corresponde ao valor que deverá ser transferido).

O primeiro número identificador de requisição que deverá ser usado por cada cliente é sempre 1. Este número deve ser incrementado em 1 a cada nova requisição enviada pelo cliente.

O servidor receberá a mensagem de requisição de um dado cliente e confirmará que o número de identificação recebido na mensagem é o próximo esperado. Caso positivo (o número de identificação recebido na mensagem é o próximo esperado), o servidor atualizará o último número de requisição recebido, processará a requisição, e responderá ao cliente. Caso negativo, o servidor deverá responder a requisição com uma mensagem de ACK com o último número de identificação de requisição recebida e processada (dado que se tratará de uma retransmissão de uma mensagem para a qual o cliente não recebeu um ack do servidor).

Por outro lado, caso o servidor receba uma mensagem do cliente com um número de identificação superior ao próximo identificador esperado, o servidor deverá responder a requisição com uma mensagem de ACK com o

último número de identificação de requisição recebida e processada, indicando assim que alguma requisição anterior foi perdida.

Caso o cliente não receba uma resposta do servidor até um tempo limite, o cliente deverá assumir que a mensagem foi perdida e deve reenviar a mensagem. Nenhuma nova mensagem de requisição deve ser enviada até o recebimento da confirmação da última requisição enviada. A equipe poderá assumir um valor de *timeout* de três vezes o RTT da rede local (medido com o comando ping), para melhorar o tempo de resposta. Se preferir, a equipe poderá usar o valor de *timeout* de 10 milissegundos.

4 Implementação do Sistema

DECISÃO DO GRUPO: Para as etapas do trabalho, o grupo poderá realizar o trabalho usando as linguagens C ou C++ . Sabendo que cada linguagem possui suas particularidades, o grupo deverá estar ciente de sua decisão e em quais implicações trará para o desenvolvimento do trabalho.

A proposta de trabalho prático está dividida em duas etapas, sendo que a segunda etapa irá contemplar funcionalidades adicionais. Portanto, considere uma implementação modular e com possibilidade de extensão, e o encapsulamento das funções de comunicação do cliente e do servidor em módulos isolados.
Para o funcionamento do serviço, o servidor deverá manter uma tabela de clientes participantes. Cada cliente é caracterizado pelo endereço IP (campo *address*), o número identificador da última requisição recebida e processada (campo *last_req*), e o valor do saldo atual do cliente (campo *balance*). Ao receber uma mensagem de descoberta de um novo cliente (via subserviço de descoberta), uma nova entrada deverá ser incluída na lista, com o campo last_req zerado e o campo balance inicializado com N reais. A Tabela 1 ilustra um exemplo dessa estrutura de dados.

Endereço IP (address)	**Último ID recebido (last_req)**	**Saldo atual (balance)**
10.1.1.1	0	162
10.1.1.2	2	96
10.1.1.3	6	42
10.1.1.4	6	100

Tabela 1: Estrutura de dados de clientes; a soma dos saldos é 162+96+42+100 = 400.

No exemplo da Tabela 1, considera-se que o cliente 10.1.1.4 acabou de ingressar no sistema com saldo inicial de 100 reais; o cliente 10.1.1.2, por sua vez, enviou duas requisições, a primeira transferindo 10 reais para outro cliente e a segunda transferindo 33 reais para outro cliente; por fim, o cliente 10.1.1.3 enviou seis requisições, transferindo valores diversos. Estas transações estão listadas no histórico da Tabela 2.

Tx #	**Origem (address)**	**ID no cliente (req_id)**	**Destino (address)**	**Valor (amount)**
1	10.1.1.2	1	10.1.1.3	10
2	10.1.1.2	2	10.1.1.1	33
3	10.1.1.3	1	10.1.1.1	5
4	10.1.1.3	2	10.1.1.2	7
5	10.1.1.3	3	10.1.1.1	9
6	10.1.1.3	4	10.1.1.2	12
7	10.1.1.3	5	10.1.1.1	15
8	10.1.1.3	6	10.1.1.2	20

Tabela 2: Histórico de transações processadas (total de 8; soma transferida = 111). Os contadores locais *req_id* por cliente alcançam 2 (10.1.1.2), 6 (10.1.1.3), 0 (10.1.1.1) e 0 (10.1.1.4), consistentes com *last_req*. A soma dos saldos permanece constante em 400.

O servidor deverá manter ainda uma tabela para guardar informações sobre o histórico de transações e o saldo total do banco (soma dos saldos de todos os clientes). Em resumo, o servidor deverá guardar a quantidade total de transações processadas (campo *num_transactions*), o valor total transferido entre clientes (campo *total_transferred*) e o saldo total do banco (campo *total_balance*). O histórico de transações em formato de *log* como detalhado na Tabela 2 não é necessário ser salvo em uma tabela, apenas informado no terminal. Porém, caso a equipe queira armazenar tal tabela, é possível.

Qtd. transações (num_transactions)	**Valor total transferido (total_transferred)**	**Saldo total do banco (total_balance)**
8	111	400

Tabela 3: Exemplo de estrutura de dados do histórico e saldo total mantida pelo servidor

No exemplo da Tabela 3, registra-se que o servidor processou 8 transações, o valor total transferido entre clientes foi 111 reais e o saldo total do banco (soma dos saldos de todos os clientes) é 400 reais.

Note que os valores de saldos e transações serão não-determinísticos, e dependerão essencialmente da ordem em que as requisições forem recebidas no servidor e se haverá perda de pacotes e retransmissões no sistema.

O servidor deverá implementar um esquema baseado no modelo leitor/escritor para gerenciar a leitura e escrita de dados na tabela que registra o histórico e saldo total. Isso significa que o acesso concorrente à tabela deverá ser controlado por primitivas de exclusão mútua. O subserviço de descoberta (primeiro escritor) ficará responsável por incluir entradas da tabela de clientes (sempre que um cliente ingressar do serviço). O subserviço de processamento (segundo escritor), por sua vez, deverá processar cada requisição recebida de cada cliente. Deverá ser obrigatoriamente implementada uma thread para processar cada requisição recebida. Já o subserviço de interface (leitor) deverá aguardar por atualizações na tabela, e escrevê-las na tela para o(a) usuário(a).

Observe que a escrita na tabela será uma operação não bloqueante, enquanto a leitura será uma operação bloqueante (o leitor ficará bloqueado até que uma nova atualização esteja disponível para leitura). Observe também que, enquanto a tabela estiver sendo lida, nenhum escritor poderá modificá-la.

5 Interface do Usuário

5.1 Servidor

O servidor deverá ser **OBRIGATORIAMENTE** iniciado via linha de comando de forma simples, com a porta UDP a ser usada para comunicação inter-processos sendo recebida como parâmetro:

```
$ ./servidor 4000
```

Após iniciar o servidor, este deverá **OBRIGATORIAMENTE** exibir uma mensagem na tela informando a data e hora atual, o número de transações processadas (0), o valor total transferido (0) e o saldo total do banco (0), NO EXATO FORMATO ABAIXO:

```
2025-09-11 18:37:00 num_transactions 0 total_transferred 0 total_balance 0
```

A cada nova requisição recebida, o servidor deverá OBRIGATORIAMENTE exibir na tela a data e hora atual, o endereço IP de origem da requisição, o identificador da requisição recebida, o endereço IP do cliente destino, o valor transferido, o número total de transações processadas (já incluindo a transação atual), o valor total transferido (já incluindo o valor da transação atual) e o saldo total do banco (já atualizado). Por exemplo:

- Assumindo que o servidor acabou de receber a Requisição #1 do cliente 10.1.1.2 transferindo 10 reais para o cliente 10.1.1.3:

```
2025-09-11 18:37:01 client 10.1.1.2 id_req 1 dest 10.1.1.3 value 10
num_transactions 1
total_transferred 10 total_balance 300
```

- Assumindo que o servidor acabou de receber a Requisição #1 do cliente 10.1.1.3 transferindo 8 reais para o cliente 10.1.1.2:

```
2025-09-11 18:37:02 client 10.1.1.3 id_req 1 dest 10.1.1.2 value 8
num_transactions 2
total_transferred 18 total_balance 300
```

- Assumindo que o servidor acabou de receber a Requisição #2 do cliente 10.1.1.2 transferindo 3 reais para o cliente 10.1.1.1:

```
2025-09-11 18:37:03 client 10.1.1.2 id_req 2 dest 10.1.1.1 value 3
num_transactions 3
total_transferred 21 total_balance 300
```

Em caso de requisição recebida em duplicidade de um determinado cliente, o servidor deverá reexibir a mensagem exibida ao receber a última mensagem do cliente, por exemplo:

- Assumindo que o servidor acabou de receber a Requisição #1 do cliente 10.1.1.3 transferindo 8 reais para o cliente 10.1.1.2, sendo que a tabela de clientes indica que a última requisição recebida de 10.1.1.3 foi #1:

```
2024-10-01 18:37:02 client 10.1.1.3 DUP!!  id_req 1 dest 10.1.1.2 value 8
num_transactions 2 total_transferred 18 total_balance 300
```

5.2 Cliente

O cliente também deverá ser OBRIGATORIAMENTE iniciado via linha de comando de forma simples, com a porta UDP a ser usada para comunicação inter-processos sendo recebida como parâmetro:

```
$ ./cliente 4000
```

Ao iniciar, o cliente deverá OBRIGATORIAMENTE exibir na tela uma mensagem informando a data e hora atual e o endereço IP do servidor (após a descoberta ser finalizada):

```
2024-10-01 18:37:00 server_addr 10.1.1.20
```

A partir de então, o cliente deverá OBRIGATORIAMENTE ler da entrada padrão (teclado) o próximo comando a ser processado, para ser enviado ao servidor. O comando deve seguir o formato: IP_DESTINO VALOR. Para verificação do saldo atual, o comando deve ser: IP_DESTINO 0.

Importante: o cliente não deve exibir nenhuma mensagem na tela solicitando o próximo comando a ser lido. O usuário deverá simplesmente digitar um comando e apertar ENTER. Assim, o cliente deverá criar uma mensagem de requisição e enviar ao servidor. Após receber cada resposta, o cliente deverá exibir uma mensagem na tela. Por exemplo:

- Assumindo que o cliente mandou a Requisição #1 transferindo 10 reais para o cliente 10.1.1.3:

```
2024-10-01 18:37:01 server 10.1.1.20 id_req 1 dest 10.1.1.3 value 10 new_balance
90
```

Caso o cliente não receba uma resposta a uma requisição e estoure o tempo de timeout, deverá reenviar a requisição. A interface do cliente deve ter uma thread para escrever as mensagens na tela, e outra thread para ler os comandos digitados pelo(a) usuário(a). Ao apertar CTRL+C (interrupção) ou CTRL+D (fim de arquivo), o processo cliente deverá encerrar, sinalizando ao manager que o(a) usuário(a) está saindo do serviço (similar a EXIT).

6 Formato de Estruturas

A equipe tem liberdade para definir o tamanho e formato das mensagens que serão usadas para troca de dados entre os processos executando em cada estação. Sugere-se a especificação de uma estrutura para definir as mensagens trocadas. Abaixo é apresentada uma sugestão de como implementar a estrutura para a troca de comandos e mensagens entre os processos.

```
struct requisicao {
        uint32_t dest_addr; // Endereço IP do cliente destino
        uint32_t value; // Valor da transferência
    };

    struct requisicao_ack {
        uint32_t seqn; // Número de sequência que está sendo feito o ack
        uint32_t new_balance; // Novo saldo do cliente origem
    };

    typedef struct __packet {
        uint16_t type; // Tipo do pacote (DESC | REQ | DESC_ACK | REQ_ACK )
        uint32_t seqn; // Número de sequência de uma requisição
    union {
        struct requisicao req;
        struct requisicao_ack ack;
        }
    } packet;
```

7 Descrição do Relatório

A equipe deverá produzir um relatório fornecendo os seguintes dados:

- Explicação e respectivas justificativas a respeito de:
  1. Como foi a implementado cada subserviço
  2. Em quais áreas do código foi necessário garantir sincronização no acesso a dados
  3. Descrição das principais estruturas e funções que a equipe implementou
  4. Explicas o uso das primitivas de comunicação inter-processos usadas
- Descrição dos problemas que a equipe encontrou durante a implementação e como estes foram resolvidos (ou não)
- Descrição de qual parte cada um dos integrantes do grupo atuou de forma ativa no desenvolvimento do trabalho

A nota será atribuída baseando-se nos seguintes critérios:

- Qualidade do relatório produzido conforme os itens acima;
- Correta implementação das funcionalidades requisitadas;
- Qualidade do programa em si (incluindo a implementação da interface exatamente como solicitado na especificação do trabalho, documentação do código, funcionalidades adicionais implementadas, etc.);
- Qualidade da apresentação (o que inclui domínio sobre o trabalho realizado por cada integrante da equipe).

8 Métodos de Avaliação

O trabalho deve ser feito em grupos de **TRÊS ou QUATRO INTEGRANTES**. As pessoas participantes da equipe devem estar claramente identificadas no relatório e na apresentação. Grupos com menos de 3 pessoas serão distribuídos ou reorganizados nos outros grupos. A avaliação do trabalho será pela análise da implementação, do relatório produzido e da apresentação e demonstração, incluindo o funcionamento correto e completo do sistema. A ausência de um(a) integrante da equipe no dia da apresentação implicará em conceito zero para a pessoa ausente (salvo por motivos excepcionais como por ex. de saúde, que deverão ser registrados via junta médica da UFRGS).

A apresentação dos trabalhos e demonstração prática em laboratório dos sistemas implementados será realizada presencialmente, conforme o cronograma da disciplina. O funcionamento correto e esperado dos sistemas implementados é um critério importante de avaliação, conforme mencionado no início deste documento. A participação de todos na demonstração é veementemente recomendada, tornando claro como cada participante atuou ativamente na produção do trabalho.

Faz parte do pacote de entrega os códigos-fonte da implementação, um tutorial de como compilar e executar os códigos e o relatório em um arquivo ZIP. A implementação deve estar funcional para demonstração durante a apresentação pela equipe, em laboratório. A compilação deverá ser feita via scripts automatizados (por ex., Makefile), de modo a facilitar o processo de avaliação do projeto submetido.

9 Dúvidas, Questionamentos e Sugestões

Alterações nas especificações do trabalho poderão ser realizadas em concordância com o professor. Adições e melhorias, como interface gráfica ou métodos adicionais, poderão ser realizadas desde que a funcionalidade básica seja mantida.

Dúvidas, questionamentos e sugestões podem ser enviados por e-mail ou pelo fórum do Moodle.

Professor Eder Scheid: `ejscheid@inf.ufrgs.br`
Mestrando João Nunes (Turma A): `jdmnunes@inf.ufrgs.br`
Mestrando Guilherme Soares (Turma B): `gmsoares@inf.ufrgs.br`