UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
DEPARTAMENTO DE INFORMÁTICA APLICADA

INF01151 – SISTEMAS OPERACIONAIS II N – Turma B
PROF. WEVERTON CORDEIRO
SEMESTRE 2024/2
TRABALHO PRÁTICO PARTE 1: THREADS, SINCRONIZAÇÃO E COMUNICAÇÃO

ESPECIFICAÇÃO DO TRABALHO

A proposta de trabalho prático é implementar um serviço distribuído de soma de números inteiros recebidos de clientes diversos. O objetivo do serviço é receber requisições de clientes diversos contendo um número a ser somado, processar cada requisição e somar o número recebido a uma variável acumuladora mantida pelo servidor, e exibir o resultado da soma após cada requisição recebida. Cada requisição conterá um número inteiro positivo, que será lido pelo cliente a partir da entrada padrão. O servidor deverá somar os números de todas as requisições recebidas de todos os clientes. O servidor deverá obrigatoriamente processar as requisições de forma concorrente/paralela, usando threads (uma thread para processar cada requisição recebida).

O principal critério de avaliação será a soma correta de todos os números enviados por todos os clientes nas requisições. Um critério secundário de avaliação será o tempo de resposta para o servidor receber, processar e responder um grande volume de requisições (10.000.000+ requisições) 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. 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.

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).

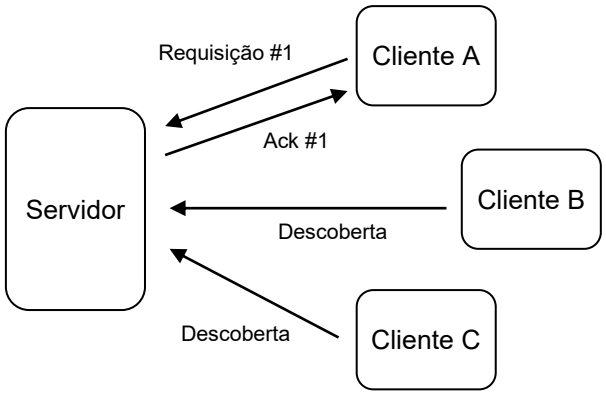

Figura 1. Dinâmica de funcionamento do serviço distribuído de agregação de dados.

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.

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 e um número inteiro. O servidor recebe cada mensagem, faz o processamento (soma o número contigo na mensagem de requisição a uma variável acumuladora mantida pelo servidor), 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 resultado parcial da soma após o processamento desta requisição. 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 número inteiro a ser somado), processá-las (somar o número inteiro recebido à variável acumuladora mantida no servidor) e enviar as respostas (confirmar que a requisição foi recebida, processada, e o resultado parcial da soma). Este subserviço (i) operará de forma ativa (isto é, enviando mensagens de requisições) nos clientes e (ii) 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 o resultado do processamento de todas as requisições 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 segunda parte do trabalho, mecanismos de replicação serão usados para garantir a persistência consistente das estruturas em múltiplas estações servidoras.

SOBRE AS REQUISIÇÕES

Cada estação cliente deverá ler da entrada padrão (**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) e outro número inteiro (que corresponde ao número que deverá ser somado pelo servidor).

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 servidor atualizará o último número de requisição recebido, processará a requisição, e responderá ao cliente. Caso negativo,

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 milisegundos.

IMPLEMENTAÇÃO DO SISTEMA

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 da soma enviado ao cliente em resposta à última requisição processada (campo *last_sum)*. Ao receber uma mensagem de descoberta de um novo cliente (via subserviço de descoberta), uma nova entrada deverá ser incluída na lista, com o os campos *last_req* e *last_sum* zerados. A Tabela 1 ilustra um exemplo dessa estrutura de dados.

Tabela 1. Exemplo de estrutura de dados de clientes mantida pelo servidor.

Endereço IP (address)	Última requisição recebida (last_req)	Último valor da soma acumulada enviado ao cliente (last_sum)
1.1.1.1	0	0
1.1.1.2	2	43
1.1.1.3	6	68

No exemplo da Tabela 1, considera-se que o cliente 1.1.1.1 acabou de ingressar no sistema; o cliente 1.1.1.2, por sua vez, enviou duas requisições, a primeira contendo o número 9 e a segunda contendo o número 34; por fim, o cliente 1.1.1.3 enviou seis requisições, com os seguintes valores: 7, 1, 3, 19, 33, 5.

O servidor deverá manter ainda uma tabela para guardar informações sobre a soma agregada. Em resumo, o servidor deverá guardar a quantidade de requisições recebidas de todos os clientes (campo *num_reqs*) e o valor total da soma acumulada (campo *total_sum*).

Tabela 2. Exemplo de estrutura de dados da soma agregada mantida pelo servidor.

Quantidade de requisições recebidas (num_reqs)	Valor total da soma acumulada (total_sum)
8	111

No exemplo da Tabela 2, registra-se que o servidor recebeu 8 requisições de todos os clientes, e o valor total da soma acumulada até o momento é 111.

O servidor deverá implementar um esquema baseado no modelo leitor/escritor para gerenciar a leitura e escrita de dados na tabela que registra a soma agregada. 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 (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.

INTERFACE DO USUÁRIO

SERVIDOR:

O servidor deverá ser 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 exibir uma mensagem na tela informando a data e hora atual, o número de requisições recebidas (0) e o valor da soma agregada (0), por exemplo:

```
2024-10-01 18:37:00 num_reqs 0 total_sum 0
```

A cada nova requisição recebida, o servidor deverá exibir na tela a data e hora atual, o endereço IP de origem da requisição, o identificador da requisição recebida, o número total de requisições recebidas (incluindo a requisição atual) e a soma total acumulada (incluindo o número contido na requisição atual). Por exemplo:

- Assumindo que o servidor acabou de receber a Requisição #1 do cliente 1.1.1.2 com o número 10

  ```
  2024-10-01 18:37:01 client 1.1.1.2 id_req 1 value 10 num_reqs 1 total_sum 10
  ```

- Assumindo que o servidor acabou de receber a Requisição #1 do cliente 1.1.1.3 com o número 8

  ```
  2024-10-01 18:37:02 client 1.1.1.3 id_req 1 value 8 num_reqs 2 total_sum 18
  ```

- Assumindo que o servidor acabou de receber a Requisição #2 do cliente 1.1.1.2 com o número 3

  ```
  2024-10-01 18:37:03 client 1.1.1.2 id_req 2 value 3 num_reqs 3 total_sum 21
  ```

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 1.1.1.3 com o número 8, sendo que a tabela de clientes indica que a última requisição recebida de 1.1.1.3 foi #1:

  ```
  2024-10-01 18:37:02 client 1.1.1.3 DUP!! id_req 1 value 8 num_reqs 2 total_sum 18
  ```

CLIENTE:

O cliente também deverá ser 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 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 1.1.1.20
```

A partir de então, o cliente deverá ler ***da entrada padrão*** (obrigatoriamente) o próximo número a ser somado, para ser enviado ao servidor. Importante: o cliente não deve exibir nenhuma mensagem na tela solicitando o próximo

número a ser lido. O usuário deverá simplesmente digitar um número 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 do cliente 1.1.1.2 com o número 10

```
2024-10-01 18:37:01 server 1.1.1.20 id_req 1 value 10 num_reqs 1 total_sum 10
```

Caso o cliente não receba uma resposta a uma requisição e estoure o tempo de timeout

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).

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 {
  uint16_t value;     // Valor da requsição
};

struct requisicao_ack {
  uint16_t seqn;      //Número de sequência que está sendo feito o ack
  uint16_t num_reqs;  // Quantidade de requisições
  uint16_t total_sum; // Valor da soma agregada até o momento
};

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

DESCRIÇÃO DO RELATÓRIO

A equipe deverá produzir um relatório fornecendo os seguintes dados:

- Explicação e respectivas justificativas a respeito de:
  - (A) Como foi implementado cada subserviço;
  - (B) Em quais áreas do código foi necessário garantir sincronização no acesso a dados;
  - (C) Descrição das principais estruturas e funções que a equipe implementou;
  - (D) Explicar 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).

A **nota será atribuída baseando-se nos seguintes critérios**: (1) qualidade do relatório produzido conforme os itens acima, (2) correta implementação das funcionalidades requisitadas, (3) qualidade do programa em si (incluindo uma interface limpa e amigável, documentação do código, funcionalidades adicionais implementadas, etc.) e (4) qualidade da apresentação (o que inclui domínio sobre o trabalho realizado por cada integrante da equipe).

MÉTODOS DE AVALIAÇÃO

O trabalho deve ser feito em grupos de **QUATRO INTEGRANTES**. As pessoas participantes da equipe devem estar claramente identificadas no relatório e na apresentação. A avaliação do trabalho será pela análise da implementação, do relatório produzido e da apresentação. 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.

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.

DÚVIDAS, QUESTIONAMENTOS E SUGESTÕES

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