**Processos, Threads, Concorrência e Paralelismo**

Eduarda Immianowsky[[1]](#footnote-1)

Filipi da Costa[[2]](#footnote-2)

**Resumo:** este artigo apresenta a resolução da primeira avaliação da disciplina de Sistemas Operacionais que tem por objetivo consolidar o aprendizado sobre conceitos de IPC, threads, concorrência e paralelismo.

**Palavras Chaves:** Palavra 1; Palavra 2.

1. **Introdução**

É necessário que os sistemas computacionais apresentem a capacidade de atender múltiplas requisições simultaneamente para garantia de um bom desempenho e escalabilidade. O uso de threads é o que permite uma execução paralela de tarefas dentro de um mesmo processo. Porém, criar uma nova thread para cada requisição se torna custoso em termos de recursos e desempenho, principalmente com um número alto de requisições. Para solucionar esse problema, a utilização de um pool de threads se torna mais eficaz. Isto é, um conjunto de threads pré-criadas que podem ser reutilizadas para processar requisições, tornando menos provável o overhead de criação e destruição de threads. Isso resulta em um sistema mais funcional e eficiente.

O objetivo deste projeto é a implementação de um servidor que utiliza dois sockets ou pipes nomeados para comunicação com clientes. Ele opera com dois tipos de conexões: uma para strings e outra para números. O servidor utiliza um pool de threads para gerenciar múltiplos clientes de maneira coerente.

1. **Inicialização**

Primeiramente, são criados dois sockets, um para strings e outro para números, e um pool de threads com duas threads. O servidor então utiliza a função select para monitorar conexões nos dois sockets simultaneamente, aguardando até que um deles tenha uma conexão pendente. Quando uma conexão é detectada, o servidor a aceita, estabelecendo uma conexão com o cliente em um dos pipes.

No processamento de dados, o servidor trata as solicitações de forma diferente dependendo do tipo de conexão. Se o cliente está conectado ao pipe de strings, o servidor converte os dados recebidos para maiúsculas. Se o cliente está conectado ao pipe numérico, o servidor adiciona 10 aos dados recebidos. Após o processamento, o servidor envia os dados de volta ao cliente.

A utilização de threads separadas permite que o servidor atenda múltiplos clientes ao mesmo tempo, aproveitando a concorrência para gerenciar várias conexões simultaneamente. Este código utiliza concorrência, sockets e comunicação entre processos em um ambiente Linux, permitindo ao servidor gerenciar múltiplos clientes de maneira eficiente.

1. **Criação dos sockets**

Os sockets locais são criados com as funções socket e bind. O servidor utiliza dois caminhos diferentes:

* /tmp/pipestr para strings;
* /tmp/pipenum para números.

Esses sockets usam o tipo AF\_UNIX para se comunicar com processos locais no mesmo sistema operacional.

O servidor também usa a função select para aguardar ambos os sockets ao mesmo tempo. Ela identifica qual pipe está pronto para aceitar uma nova conexão.

Basicamente o servidor chama select e aguarda até que um dos sockets tenha uma conexão pendente. Se um cliente tentar se conectar ao socket de strings (/tmp/pipestr), o servidor o aceita e processa como uma conexão de string. Por outro lado, se um cliente tentar se conectar ao socket numérico (/tmp/pipenum), o servidor o aceitará e processará como uma conexão numérica.

Em questão de aceitação de novas conexões, quando a função select detecta uma conexão, a função accept retorna um novo socket que representa a conexão específica com o cliente.

1. **Struct cliente\_data\_t**

Essa struct armazena duas informações:

* newsockfd: o socket do cliente;
* pipe\_type: indica qual tipo de pipe o cliente está usando (string ou numérica).

1. **Processamento de dados do cliente**

Após o servidor aceitar a conexão, é necessário processar o que foi enviado pelo cliente. Isso é feito pela função process\_client, que é adicionada ao pool de threads para ser executada usando concorrência.

Dentro da função process\_client, o servidor lê os dados usando read. O processamento dos dados varia conforme o tipo de pipe: se o cliente estiver conectado ao de strings, o buffer é transformado em maiúsculas, mas se o cliente estiver conectado ao numérico, o buffer é incrementado com uma constante (10). Por fim, o servidor escreve os resultados usando write.

1. **Pool de threads**

O servidor utiliza uma biblioteca de pool de threads chamada thpool, que é responsável por gerenciar as threads para executar as funções usando concorrência.

O servidor inicializa o pool de threads com duas threads usando a função thpool\_init(2). Quando um novo cliente se conecta, a função process\_client é adicionada ao pool, sendo executada em uma thread separada das outras. Isso permite que o servidor aceite novas conexões enquanto processa as conexões já existentes.

1. **Conexão do client ao server**

O código estabelece uma conexão ente cliente e servidor utilizando sockets Unix. Primeiro, prepara a estrutura sockaddr\_un para definir o endereço do servidor, configurando o tipo de socket e o caminho com base no tipo de pipe (string ou numérica). A conexão é então feita usando a função connect.

Se a conexão for bem-sucedida, o cliente solicita um dado ao usuário, envia-o para o servidor com a função write, e após isso lê a resposta do servidor com a função read.

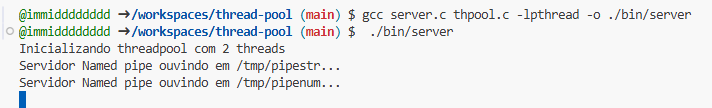
1. **Encerramento**

Ao encerrar, a conexão com o cliente é fechada (representada por newsockfd) e a memória alocada para o cliente (struct cliente\_data\_t) é liberada.

Se o servidor foi encerrado, os pipes (sockfd\_str e sockfd\_num) são fechados e o pool de threads é destruído utilizando a função thpool\_destroy.

1. **Resultados obtidos**

Seguindo as orientações de execução apresentadas no repositório, a biblioteca que executa o pool de threads não está pré-compilada e deve ser compilada junto com o projeto, como é demonstrado na Figura 1:



**Figura 1: Compilação e execução do server.**

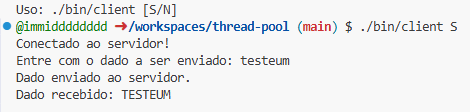
Depois, em outro terminal, é compilado e executado o client, como é visto na Figura 2:



**Figura 2: Compilação e execução do client.**

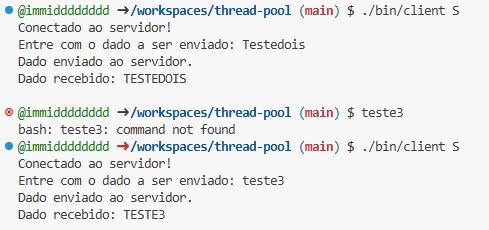
Recebemos a mensagem “./bin/client [S/N]”, que solicita qual pipe desejamos executar, S de string e N de numérico. Após a seleção é solicitado o dado a ser enviado ao servidor.

Na Figura 3, é possível observar a seleção de pipe S (string), e é enviado o dado “testeum”. Como informado anteriormente, o tipo de pipe string converte o dado para maiúsculas.



**Figura 3: Realizada a execução do tipo de pipe string.**

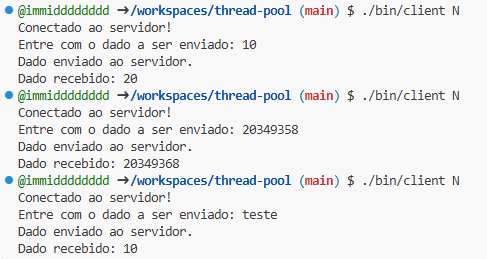
Outros testes do tipo de pipe string podem ser observados na Figura 4 abaixo:



**Figura 4: Testes do tipo de pipe string.**

No tipo de pipe numérico, após selecionarmos o tipo N (numérico), recebemos a mensagem que estamos conectados ao servidor. Podemos então entrar com o dado a ser enviado.

Na Figura 5 abaixo é demonstrado alguns testes realizados:



**Figura 5: Testes realizados no tipo de pipe numérico.**

1. UNIVALI – Universidade do Vale do Itajaí

   eduardaimmianowsky@edu.univali.br [↑](#footnote-ref-1)
2. UNIVALI – Universidade do Vale do Itajaí

   filipi\_costa@edu.univali.br [↑](#footnote-ref-2)