Arquivos distribuídos

Relatório de implementação do trabalho 2
INE5645 - Programação Paralela e Distribuída.

Giovane Pimentel de Sousa - 22202685
Guilherme Henriques do Carmo - 22201630
Isabela Vill de Aquino - 22201632

December 15, 2024

Introdução

O objetivo deste trabalho é a implementação de uma aplicação cliente/servidor distribuída utilizando sockets TCP para transferência de arquivos. A aplicação simula operações comuns como as realizadas por comandos Unix, como 'scp' e 'cp', mas com funcionalidades específicas, incluindo suporte para recomeço de transferências interrompidas e controle de múltiplas conexões simultâneas. Adicionalmente, buscou-se aplicar conceitos de programação paralela e distribuída para garantir a eficiência no processamento e atendimento de múltiplos clientes. Este relatório apresenta a arquitetura, os detalhes de implementação e as decisões técnicas tomadas durante o desenvolvimento do sistema.

Como executar

Servidor

A compilação segue o formato básico de qualquer aplicação Go. A execução do servidor segue o formato de aplicações Unix:

```
$ cd daemon && go build . # Este comando produzirá o binário remcp-serv.
$ ./remcp-serv daemon
```

Caso queira se saber qualquer saída mostrada pelo servidor como logs, conexões, erros em curso, etc., basta executar sem a flag daemon, que o servidor será executado utilizando o terminal como standard output.

Com a flag **daemon** o servidor será executado como um daemon, também ficando disponível para aceitar conexões de clientes. Como o daemon já estará sendo executado, já pode-se interrompê-lo com `Ctrl + C`.

Cliente

O cliente se comporta de maneira semelhante a comandos Unix como 'scp' ou 'cp', interpretando o IP para determinar a direção da transferência. Exemplos de execução:

```
$ cd remcp && go build . # Este comando produzirá o binário remcp.
$ ./remcp 127.0.0.1:/home/gio/Downloads/teste.txt /home/gio/arquivos_distribuidos/teste
$ ./remcp /home/gio/Downloads/teste.txt 127.0.0.1:/home/gio/arquivos_distribuidos/teste
```

- Caso o IP esteja no **primeiro** argumento, a transferência será **servidor => cliente**.
- Caso o IP esteja no **segundo** argumento, a transferência será **cliente => servidor**.

Implementação

A implementação foi realizada em Go 1.23 e explorou várias técnicas e conceitos relevantes para sistemas distribuídos, incluindo:

Arquitetura

A aplicação é dividida em dois componentes principais:

1. **Servidor (*remcp-serv*)**: Executa como um daemon, permitindo o atendimento simultâneo de múltiplos clientes, respeitando o limite máximo configurado.
2. **Cliente (*remcp*)**: Realiza transferências de arquivos, interpretando a direção com base nos argumentos fornecidos.

Configurações

As principais configurações, como número máximo de conexões simultâneas e velocidade de transferência, estão implementadas como constantes no código. Isso simplifica ajustes durante o desenvolvimento.

Controle de concorrência

Para garantir a integridade durante a execução paralela, mutexes foram utilizados nas regiões críticas. O servidor distribui a banda disponível igualmente entre os clientes conectados, ajustando dinamicamente a velocidade de transferência.

Acknowledgments (ACKs)

O uso de "acknowledgments" foi essencial para evitar problemas de mensagens malformadas. Sempre que uma mensagem é enviada entre cliente e servidor, o receptor confirma o recebimento antes de prosseguir.

Mecanismo de retry

Quando o número máximo de conexões é atingido, o cliente tenta se reconectar automaticamente ao servidor. O comportamento segue esta lógica:

- Até 5 tentativas são realizadas, com um intervalo de 5 segundos entre elas.
- Caso o limite persista, a conexão é encerrada com um erro.

Recomeço de transferências

Quando uma transferência é interrompida por qualquer circustância, elas podem ser retomadas da seguinte forma:

- Arquivos parciais (*.part*) são criados na pasta */tmp*, sendo ela utilizada como diretório padrão neste projeto.
- Durante a inicialização de uma transferência, o cliente verifica a existência de arquivos *.part* e informa ao servidor o ponto de recomeço. Assim o servidor só precisa enviar o restante dos dados, não o todo.

Fluxo de transferências

1. **Servidor => Cliente**: O cliente recebe os dados em partes, armazenando temporariamente até concluir o download.
2. **Cliente => Servidor**: A transferência ocorre de maneira direta, sem throttling ou controle adicional.

Conclusões

A implementação da aplicação proposta foi uma oportunidade para consolidar conceitos importantes de programação paralela e distribuída. Durante o desenvolvimento, enfrentamos desafios como gerenciamento de concorrência, controle de fluxo de dados e retomada de transferências, todos solucionados com técnicas adequadas para sistemas distribuídos.

Apesar de não termos utilizado padrões de projeto explícitos, conseguimos implementar soluções eficientes e atender aos requisitos do trabalho. Como aprimoramento futuro, consideramos a modularização de funcionalidades, como throttling e mecanismos de retry, para facilitar a manutenção e evolução do código. No geral, o trabalho cumpriu seus objetivos, destacando-se pela aplicação prática de conceitos teóricos em um cenário realista.