Relatório - Sistema de Backup de Arquivos

Pedro H. Kochinski
GRR
phk20@inf.ufpr.br

Vinicius Mioto
GRR20203931
vm20@inf.ufpr.br

***Resumo*—Esse relatório tem como objetivo apresentar as principais escolhas dos autores para desenvolver o sistema de *backup* de arquivos entre duas máquinas (cliente e servidor).**

***Palavras-chave*—redes, ethernet, backup de arquivos, sockets**

I. Introdução

O projeto consiste em um sistema de *backup* de arquivos desenvolvido usando C++ em ambiente UNIX. O sistema utiliza RawSockets para comunicação entre um Cliente e um Servidor, para isso é necessário uma conexão com cabo Ethernet entre as máquinas. Uma outra opção para testes é utilizar a interface *loopback* do computador, caso só exista uma máquina disponível.

Nas próximas seções estão descritas as principais funcionalidades e detalhes de implementações do sistema, assim como o funcionamento geral da execução.

II. Mensagens

Todas as mensagens enviadas e recebidas, por ambas as máquinas, são instâncias da classe *Message*. Para o envio, o *Controller* é responsável por receber todas as informações da mensagem e copiar para o *buffer* de envio. Por outro lado, no recebimento, os dados do canal serão atribuídos para um objeto da classe *Message*. Para leitura e escrita, todos os arquivos são tratados como binários.

As mensagens possuem os seguintes atributos:

Tabela I
Atributos da Classe Message

Campo	Descrição
Cabeçalho Ethernet	14 bytes
Marcador de início	01111110
Tamanho (dados)	6 bits
Sequência	6 bits
Tipo	4 bits
Dados	0 a 63 bytes
Paridade (dados)	1 byte

Existem diversas formas de limitar os campos para o tamanho desejado, uma delas é o uso de máscaras com operações lógicas *bit* a *bit*. Porém a técnica escolhida foi utilizar atributos com tamanho específico, permitido pela linguagem escolhida. Por exemplo, para definir o campo de "Marcador de início" usamos:

```
unsigned int initMarker : 6.
```

O cabeçalho Ethernet foi adicionado às mensagens para evitar erros causados pelos *bytes* que identificam o protocolo VLAN `0x88` e `0x81` (em base hexadecimal). Esses *bytes* podem causar problemas no momento de enviar e receber dados pelo cabo. Para evitar isso, existem diversas outras técnicas como adicionar *scapes* ou utilizar operações aritméticas nos bytes. Porém, acreditamos que adicionar o cabeçalho é extremamente eficaz e mantém o código simples.

III. Comunicação

A comunicação utiliza o protocolo "Para e Espera" (*Stop and Wait*), no qual o emissor envia um pacote de dados e espera pela confirmação do receptor antes de enviar o próximo pacote. Isso garante a integridade da transmissão, permitindo o reenvio de pacotes perdidos ou corrompidos.

As mensagens corrompidas são identificadas pelo receptor por meio do cálculo de paridade vertical, que envolve a contagem dos *bits* de valor 1, na prática é utilizada a operação lógica *XOR* no campo dos dados. Se alguma inconsistência for encontrada, o receptor deve enviar um aviso (NACK) para a origem. Assim, o pacote é reenviado até que seja recebida a confirmação de exatidão (ACK) da respectiva mensagem (ver figura 1).

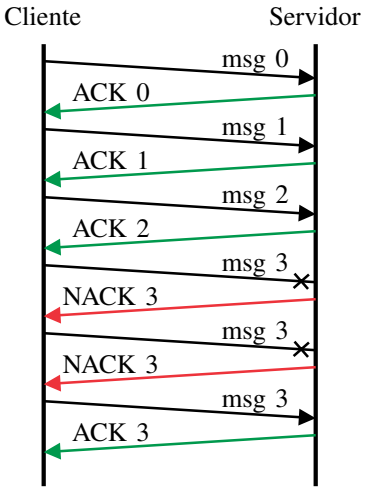

Fig. 1. NACK

De forma semelhante, as mensagens perdidas são gerenciadas através do uso de um mecanismo de *timeout*, cujo funcionamento é bastante simples. Após o envio da mensagem, o emissor aguarda até receber um ACK. Caso essa confirmação não seja recebida dentro do prazo de 1 segundo, consideramos que ocorreu um estouro do *timeout*. Nesse caso, a mensagem

é reenviada repetidamente até que a confirmação de exatidão seja finalmente recebida pela origem (ver figura 2).

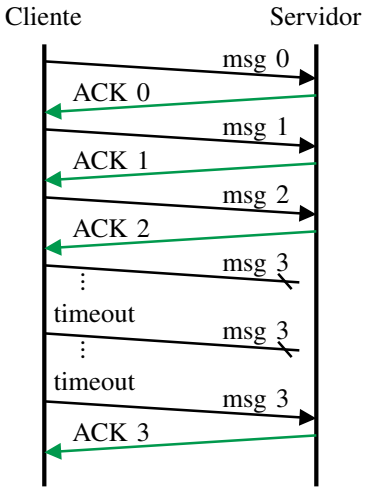

Fig. 2. Timeout

Em relação às mensagens perdidas e possíveis *timeouts*, existe um desafio adicional relacionado à perda de mensagens ACK ou NACK, especialmente quando o cabo é removido. Isso ocorre porque não há confirmação de recebimento dessas mensagens. Para solucionar esse problema, uma abordagem é enviar outro ACK para a mensagem em questão se a máquina de destino receber uma mensagem com a sequência imediatamente anterior à sequência atual. Dessa forma, o destino reconhece a mensagem e volta a aguardar a sequência atual, evitando assim possíveis falhas de comunicação.

IV. Instrução de Uso

Para compilar os arquivos, utilize o comando `make` em ambas as máquinas. Se desejar visualizar os detalhes das operações, utilize `make debug`. No cliente, execute o comando `./client`, enquanto no servidor, execute o comando `./server`. Após isso, uma tela de opções numéricas deve aparecer para o cliente, o qual deve escolher o respectivo número, pressionar `ENTER` e seguir as instruções.

V. Conclusão

O projeto foi testado com os computadores do Centro de Computação Científica e Software Livre (C3SL). Para as implementações iniciais, utilizamos a interface de *loopback*, com o objetivo de simplificar e agilizar os testes. O código fonte está em um repositório público no GitHub.