CI1058 - Redes de Computadores
Protocolo Kermit

Alexander Mion, Eduardo Kruger
Departamento de Informática
Universidade Federal do Paraná – UFPR
Curitiba, Brasil

I. Introdução

O trabalho, que está no link (https://github.com/EduardoFariaKruger/tp1-de-redes1) consistiu na implementação de um sistema cliente-servidor para um jogo distribuído em que o cliente movimenta um agente em um grid 8x8, buscando tesouros escondidos em posições sorteadas pelo servidor. Quando o jogador alcança um tesouro, o servidor envia ao cliente um arquivo correspondente ao conteúdo do tesouro (texto, imagem ou vídeo), que deve ser corretamente interpretado e exibido. A comunicação foi realizada por meio de raw sockets, respeitando os campos do protocolo proposto e o controle de fluxo do tipo stop-and-wait.

II. Decisões Técnicas

A. Gestão de Timeouts

Durante os testes, identificou-se que a função `recv()` pode travar indefinidamente caso não cheguem mensagens, o que ocorre, por exemplo, quando há problemas físicos na conexão (como o cabo de rede desconectado). Para mitigar isso, foram implementados dois tipos de timeout:

- **Timeout de socket:** Utilizando a opção `SO_RCVTIMEO`, foi possível definir um tempo máximo de espera por mensagem. Caso esse tempo seja ultrapassado, o programa identifica ausência de resposta e pode emitir mensagens de erro apropriadas ao usuário. Este timeout também permite detectar situações onde o cabo de rede foi desconectado, mesmo sem haver erro explícito na pilha de rede.
- **Timeout de aplicação:** Foi incorporado um segundo nível de timeout, medido com `time_t` e `difftime()`, que permite ao programa abortar operações que ficam presas mesmo com respostas inválidas chegando constantemente (como mensagens corrompidas ou lixo de rede). Assim, garante-se que o cliente não fique preso em loops sem progresso.

B. Tipo livre: REQUEST_ARQUIVO (REQ_ARQ)

Desde o início da implementação, priorizou-se uma abordagem modular para a transferência de arquivos. A primeira funcionalidade consolidada foi o envio de arquivos arbitrários pelo nome, via código `REQ_ARQ`. Essa decisão permitiu desacoplar o tipo do tesouro (texto, imagem, vídeo) da lógica principal de movimentação, tornando o sistema mais flexível para extensões futuras.

A partir dessa estrutura base, todo o fluxo de obtenção de tesouros foi construído de forma a aproveitar esse mecanismo, bastando identificar o tipo do arquivo e ajustar a resposta do servidor para os códigos 6, 7 ou 8 (com `ack` e nome).

C. Envio Explícito do Estado do Tabuleiro (ESTADO_TABULEIRO)

O protocolo original previa que, ao final do envio de um arquivo, o servidor retornasse um `OK_ACK`. No entanto, ao detectar que o jogador encontrou um tesouro, o servidor envia diretamente um `XXX_ACK_NOME` e inicia o envio do conteúdo, omitindo assim o estado atualizado do tabuleiro.

Para evitar que o cliente fique com um estado inconsistente, foi criado o tipo de mensagem `ESTADO_TABULEIRO`, enviada explicitamente após o término de um envio. Essa decisão garante a consistência visual e lógica da interface do cliente, independentemente do caminho seguido no fluxo de mensagens.

D. Enquadramento e Proteção contra TAG VLAN

Durante os testes práticos, foi identificado (com orientação de ex-alunos do C3SL) que certos padrões de bytes, associados a TAGs VLAN — poderiam corromper pacotes na recepção por determinadas placas de rede, que substituem automaticamente esses padrões por outros valores.

Para contornar o problema, implementou-se um sistema de enquadramento inspirado em técnicas estudadas em aula: inseriu-se o byte `0xFF` entre os trechos hexadecimais dos frames enviados, impedindo que o padrão indesejado ocorresse naturalmente. Na recepção, os bytes `0xFF` são removidos, reconstituindo o conteúdo original do pacote.

E. Validação de Espaço e Tipos de Arquivo

Outra decisão importante foi validar o espaço disponível no sistema de arquivos do cliente, antes de permitir o início da transferência. Utilizou-se a função `statvfs()` para estimar o espaço livre e, com base no tamanho do arquivo (obtido via `stat()`), evitar transferências que resultariam em falhas ou arquivos incompletos.

Além disso, foi implementada lógica de reconhecimento do tipo de arquivo com base na extensão, para que a exibição correta (via terminal ou aplicativo externo) fosse possível logo após o recebimento.