Redes 1 - Trabalho 1

Cauê Mateus Gonçalves Venturin Samonek[1], **Augusto Antonio Kolb Schiavini**[1]

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)

{cmgvs23,aaks23}@inf.ufpr.br

1. Informações Gerais

O trabalho 1 consiste na implementação de um jogo de Caça ao Tesouro usando dois computadores no formato cliente-servidor conectados por um cabo. Para isso, foi utilizada a linguagem C++, a biblioteca SMFL para disponibilizar uma interface gráfica e, para a conexão de rede, o raw sockets somado a um protocolo customizado com base no kermit. No jogo, um personagem deve caminhar por um grid 8x8 a procura de tesouros, os quais podem ser arquivos .txt, .jpg ou .mp4. A dinâmica geral funciona da seguinte forma: O cliente exibe a interface para o usuário e permite que ele mova o personagem pelo grid procurando tesouros; cada movimento feito pelo usuário deve ser comunicado com o servidor, que é responsável por validar os movimentos do usuário e escrevê-los como logs no terminal do servidor. Ao encontrar um tesouro, o servidor o envia para o cliente que exibe o arquivo automaticamente. Em caso de não haver espaço ou permissão de escrita adequada, a transferência do arquivo não é realizada e é emitido um alerta em ambos os computadores, porém de diferentes formas, avisando do erro ocorrido durante a tentativa de transferência.
Link Para O Repositorio Git: https://github.com/CaueSamonek/Redes_1_Trabalho_1

2. Interface e Interação

Usando a biblioteca SFML, é disponibiliza ao cliente uma interface gráfica com um grid 8x8 e um contador para a quantidade de tesouros encontrados. A inicialização (posicionamento dos tesouros e do player) é feita quando o cliente e o servidor tem uma conexão estabelecida. Com o início do jogo, o personagem (um pinguim!!) aparece na tela e poderá ser movido pelo grid utilizando as setas do teclado. A medida que o personagem se move, ele larga pequenos ovos, que vão demarcando as posições por onde já passou. Caso determinada posição contenha um tesouro, e este tesouro tenha sido transferido ao cliente com sucesso, ele é marcado como um pinguinzinho saindo do ovo, e um ovo rachado sem pinguim caso o arquivo não tenha sido transferido devido a algum erro.

Já o servidor, por outro lado, verifica se o pedido de movimento do cliente é válido, isto é, se o personagem se moverá ou não, imprimindo um log dos movimentos feitos. Ele avisa também quando algum arquivo que deveria ser transferido para o cliente deu erro na transmissão (de permissão de escrita ou de falta de espaço). Além disso, ele imprime uma lista com todos os tesouros existentes, se foram achados pelo cliente ou não, além de suas posições no grid. É importante notar que caso o tesouro não seja transferido por algum erro, ele não será marcado como ”achado” pelo servidor, mesmo que o cliente saiba de sua posição devido a marcações gráficas (ovo rachado).

3. Protocolo Utilizado

O protocolo utilizado é, na verdade, uma derivação do proposto no enunciado, sendo implementado com janela deslizante de tamanho 3 com Volta-N, além de alguns tipos

de mensagem terem sido modificados, sendo esses tipos e suas modificações descritos a seguir:

- Livre 1: Esse código foi utilizado como handshake entre o cliente e o servidor, fazendo com que as sequências de cada computador se sincronizem. Isso é utilizado quando algum dos lados ”cai”, e.g., o processo do servidor é morto e então executado novamente.
- Livre 2: Esse código foi utilizado como sinal de inicialização. Ele é enviado pelo cliente ao servidor para indicar um novo jogo, fazendo com que o servidor sorteie as posições dos tesouros e do jogador, atuando como reset caso o jogo já tenha sido iniciado.
- OK: Por questões de sincronismo entre o envio e recebimento de mensagens com as funções de desenho da biblioteca SFML, foi decidido remover a funcionalidade de ACK das mensagens do tipo OK, que agora apenas confirmam que o movimento foi realizado (mesmo que este tenha sido para a mesma posição, e.g., ficar batendo a cara na ”parede”).
- Tipo de Arquivo + ACK + Nome: Referente aos 3 tipos de mensagem de arquivo (Texto, Imagem e Video), a única mudança feita nesses 3 tipos é com relação ao seu funcionamento como ACK. Em nossa implementação, elas funcionam como mensagens de dados normais, sendo assim, as mensagens que servem para controle de fluxo ficam isoladas de forma atômica: ack, nack e handshake.

Além de tudo, vale mencionar também a forma com a qual foi implementada o timeout, sendo um loop na função de envio com uma verificação de quando tempo se passou desde a ultima resposta recebida, funcionando para toda a janela de uma vez, ou seja, quando o timeout ”estoura”, todas as mensagens da janela são reenviadas, e ao receber uma resposta, independente de qual mensagem da janela seja ela, o timer é reiniciado.

Apesar das alterações feitas no processamento de alguns tipos de mensagem, o frame continua o mesmo, com checksum sendo feito em cima dos campos de tamanho, sequência, tipo e dados, além da ordem de cada campo se manter a mesma.

3.1. Erros Silenciosos

Essa trecho do relatório serve para falar a respeito de 2 erros sutis que ocorriam em versões anteriores do trabalho e como eles foram corrigidos na versão final.

Devido ao ”desaparecimento” de alguns bytes em sequências específicas (bytes 0x81 e 0x88), sempre é realizado um pré-processamento de todas as mensagens, tanto antes do envio quanto logo após o recebimento, com a adição (para envio) ou remoção (para recebimento) de um byte 0xFF logo depois dos bytes conflituosos.

Além disso, há também o descarte de mensagens muito curtas (menos de 14 bytes) pela placa de rede, sendo assim, todas as mensagens menores que esse tamanho recebem um padding com zeros na área de dados, mesmo que a seção ”tamanho” da mensagem indique 0 ou um valor menor do que o real.