Caça ao Tesouro

Trabalho 1 - Redes de Computadores (2025/1)

Alunas:

Giovanna Fioravante Dalledone (GRR: 20232370)
Nadia Luana Lobkov (GRR: 20232381)

Link Para GitHub: https://github.com/nadialobkov/t1-redes

Implementação

Estrutura dos Pacotes

Os pacotes são implementados em forma de `struct` e contem os campos previstos no protocolo proposto.

```
struct pacote {
    // cabecalho do pacote
    uint8_t marcador;   // 8 bits para marcador de inicio
    uint8_t tam : 7;    // 7 bits para tamanho dos dados
    uint8_t seq : 5;    // 5 bits para campo de sequencia
    uint8_t tipo : 4;   // 4 bits para tipo da mensagem
    uint8_t checksum;   // 8 bits para checksum

    //dados
    uint8_t dados[127]; // vetor de bytes de dados
};
```

Usamos o tipo `uint8_t` (que corresponte a um espaço de 8 bits sem sinal) para poder manipular os dados de maneira mais segura e versátil. Para garantir que o pacote tenha exatamente 131 bytes, usamos `#pragma pack` para evitar o alinhamento de bytes dos dados.

Protocolo

As mensagens com tipos iguais a 3 e 14 estavam livres para a escolha de suas respectivas funcionalidades. Para o número 3 foi escolhido o tipo SYN que é utilizado na conexão inicial.

Ele é responsável pela sincroniação, ou seja, subir primeiro o cliente e depois o servidor não faz diferença por conta dele. O tipo número 14 é auxiliar e representa a ocorrência do timeout.

Além disso, um novo código de erro foi definido: `MOV_INV` (movimento inválido). Duas macros foram definidas para o caso do OK: OK `NORMAL` e OK `TESOURO`. No primeiro caso, a posição atual do jogador não contém nenhum tesouro e, por consequência, o OK Tesouro significa o oposto.

```
// tipos de mensagens
#define SYN     3   // conexao inicial
```

```
#define TIMEOUT 14  // tipo auxiliar que indica que houve timeout

// codigos de erros
#define SEM_PERM    0   // sem permissao de acesso
#define SEM_ESP     1   // espaco insuficiente
#define MOVE_INV    2   // movimento invalido no tabuleiro

// codigos para o OK
#define NORMAL      0   // posicao normal (vazia)
#define TESOURO     1   // posicao com tesouro
```

As funções de **enviar e receber** pacotes sofreram um versinonamento interessante. No início da implementação do trabalho, ambas as funções alocavam um ponteiro pacote para cada mensagem que deveria ser recebida, ou enviada. Além de muito estranho de manipular e visualizar, era muito fácil e perigoso liberar o ponteiro em um momento errado e acabar perdendo a informação apontada por ele. Como solução do problema, atualmente, ambas as funções mantém apenas um ponteiro pacote que é sobrescrito sempre que necessário. A função de enviar pacotes não tem retorno, enquanto a função que os recebe retorna o tipo do pacote recebido ou TIMEOUT se houve timeout.

Portanto, foi estabelecida a seguinte notação

- `pack_send`: pacote que contem a mensagem a ser enviada
- `pack_recv`: pacote por onde vai receber a mensagem

O lógica **timeout** foi implementada com o auxilio da biblioteca `signal`. A cada milissegundo é disparada uma função que conta os ticks de relógio e ao chegar no TIMEOUT_MAX uma flag global é setada para informar que houve timeout. A função que verifica isso é a de recebimento de pacotes. Quando acontece um timeout, a função para de receber pacotes e retorna o tipo TIMEOUT, assim, funções de **espera de pacotes** podem usar essa sinalização para **reenviar a sua mensagem**.

O **envio de ACKS e NACKS** acontece na verificação dos pacotes. A função `espera_pacote()` fica aguardando o recebimento de uma mensagem e assim que recebe, caso seu tipo não for TIMEOUT, faz a validação do checksum. Se houve erro, é enviado um NACK e em caso de sucesso é enviado um ACK. Enquanto não enviar um ACK ele ficará esperando pacotes válidos.

Na função que calcula o cheksum existe uma manipulação (`checksum = checksum & 0xFF`) que garante que o valor final contenha sempre 8 bits, pois realiza um AND bit a bit com 8 bits 1.

Estrutura do Trabalho

1. `Makefile`

Além de compilar os arquivos para gerar o executável, o makefile possue utilizades específicas.

1. Para compilar tudo

   ```
   make
   ```

2. Para compilar tudo e exibir impressões de depuração onde o tabuleiro é omitido para que a transferência de mensagens possa ser exibida e analisada na tela.

```
make debug
```

3. Para limpar os arquivos gerados

```
make clean
```

4. Limpar a pasta de arquivos (tesouros) encontrados pelo cliente;

```
make clean_cliente
```

---

2. `pacote.h`

Biblioteca de funções que lidam com os pacotes. Contem a estrutura do protocolo, assim como funcões de criação, destruição e impressão dos pacotes, escrita de mensagem, verificações de checksum, envio, recebimento e espera de pacotes.

---

3. `sock.h`

É a biblioteca responsável por lidar com socket `Raw Sockets` e manipular os dados dos arquivos - com manipulação entende-se obter extensão, exibir e enviar e receber dados.

A função que devolve a extensão abre e lê o cabeçalho do arquivo, que é único para cada tipo de extensão. Então, armazena esses valores em um vetor e compara com o cabecalho de imagem e vídeo, se não for nenhum dos dois é atribuído o tipo texto. Retorna o tipo do arquivo (imagem, texto ou vídeo) em caso de sucesso ou erro em caso de falha.

Para o arquivo ser exibido, um processo filho é executado. Nesse processo, com base na extensão obtida, um respectivo aplicativo é chamado para exibir o arquivo recebido. Nesse caso, os aplicativos são:

- Para imagens: `feh`
- Para vídeos: `ffplay`
- Para texto: `gedit`

OBS: O gedit emite warnings quando exibe o arquivo na tela.

Envio de arquivos

A função de enviar dados recebe o socket, os pacotes (de envio e recebimento, ambos previamente alocados) e o nome do arquivo a ser enviado. O arquivo é, então, quebrado em partes (sequencialização) para entrar no campo de dados da mensgem que só é enviada após o recebimento de um ACK.

O fluxo é:

1. Envia o nome do arquivo recebido -> espera ACK
2. Obtém tamanho do arquivo (com base no caminho recebido)
3. Escreve e envia o tamanho do arquivo -> espera ACK
4. Abre e percorre o arquivo de 127 em 127 bytes (tamanho do vetor de dados) e incializa o contador com 0

5. Enquantou houverem bytes a serem lidos no arquivo:
    - escreve os bytes lidos no pacote
    - envia o pacote
    - espera o ACK
6. Quando sai do laço, o arquivo foi todo enviado, então envia a mensagem de FIM do arquivo -> espera ACK

Recebimento de arquivo

A função de recebimento de dados segue uma loógica complementar a de envio.

1. Espera receber pacote com o nome do arquivo.
2. Cria arquivo com esse novo na pasta `/arq_cliente/`
3. Espera receber pacote com tamanho do arquivo.
4. Enquanto não receber pacote de fim do arquivo
    - recebe pacote de dados
    - escreve dados no arquivo
5. Recebe pacote fim e exibe arquivo na tela.

4. `timer.h`

É a biblioteca que administra o **timeout**, como mencionado anteriormente, ela tem um contador de ticks de relógio que é atualizado a cada milissegundo para verificar se já deu o tempo do timeout.

5. `jogo.h`:

A funções implementadas em `jogo.h` manipulam o tabuleiro, o jogador e as regras do jogo.
As estruturas definidas foram:

```
    struct tabuleiro_t {
        unsigned int posicoes[8][8];              //Matriz de posições
(são 64 posições disponíveis)
        struct coordenadas_t posicao_tesouro[8];  //Coordenadas x e y dos
tesouros
        struct coordenadas_t posicao_jogador;     //Posição atual do
jogador
        unsigned int tesouros;                    //Número de tesouros que
o jogador encontrou
    }


    struct jogador_t{
    unsigned int pos_x;                           //Noordenada x da
posição atual
    unsigned int pos_y;                           //Coordenada y da
```

```
posição atual
    unsigned int tesouros;                          //Número de tesouros que
o jogador encontrou
    unsigned int casas_percorridas;                 //Quantidade de casas
que o jogador já passou
    unsigned int mapa[8][8];                        //Mapa do jogador
    }
```

As maiores considerações a serem feitas sobre esse arquivo são as seguintes:

1. A navegação no tabuleiro ocorre depois de desativar o modo canônico do teclado, ou seja, as teclas pressionadas não são mais ecoadas na tela e são processadas sem o ENTER. Quando a leitura da movimentação termina, o modo canônico é reativado, para não causar prejuízos ao terminal.
2. Foram definidas macros para a movimentação, os valores são os mesmos dos recebidos no protocolo das mensagens e facilitam a movimentação do jogador.
3. A função de movimentar o jogador verifica se o movimento é possível e a função de atualizar o jogador imprime o movimento na tela.
4. O objetivo e deixar as estruturas independentes, assim o servidor teria apenas o tabuleiro e o cliente o jogador.

Observação: Atualmente o programa mantém os tesouros sempre nas mesmas posições, pois facilita os testes, contudo, caso o professor queira, podemos aleatorizar as posições para eles a cada nova rodada. Para tanto, basta substituir `srand(0)` *por* `srand(time=NULL)` *e incluir a biblioteca* `<time.h>`

6. `servidor.c`:

O servidor tem o tabuleiro e, por isso, é o responsável pelas informações gerenciais do jogo. Ele espera a sincronização do cliente para iniciar jogo. O servidor acessa os tesouros que estão na pasta `/tesouros/` para enviar ao cliente. O programa termina quando todos os tesouros são encontrados.

7. `cliente.c`:

O cliente é o jogador, então é ele quem solicita as movimentações e recebe os tesouros. Os tesouros ficam guardados na pasta `/arq_cliente/`.

8. `script.sh`:

Foi o script utilizado para criar duas máquinas virtuais, uma para o servidor e outra para o cliente.

Considerações Finais:

1. O envio e recebimento dos pacotes não trata o caso da sequência de bytes 0x88 e 0x81, portanto, os arquivos que contiverem tais bytes apresentarão erro. No repositório do git, é possível ver a branch em que foi tentado cortonar esse problema, porém, infelizmente, sem sucesso.
2. Quando o testamos o timeout com a remoção do cabo, na primeira remoção ocorre o comportamento esperado de envio de NACK e reenvio das mensagems. O problema ocorre quanto

testamos uma segunda remoção do cabo, nesse caso, o comportamento torna-se indefinifo com um erro aparente de envio de informações sem cabo.