

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Redes de Computadores

Trabalho prático 1

Batalha Naval

Aluno: Caíque Bruno Fortunato
Matrícula: 2013062731

Belo Horizonte, Maio de 2017

Introdução

Socket é um componente do sistema que permite a comunicação entre processos através da rede por meio dos protocolos de transporte, sendo também uma abstração computacional que meia diretamente a uma porta de transporte (TCP ou UDP) mais um endereço de rede.

Existem dois lados para a comunicação: a do cliente e a do servidor, que precisam estarem usando o mesmo protocolo. O TCP, que será utilizado neste trabalho, fornece uma conexão bidirecional, sequencial e com fluxo único de dados.

A comunicação de processos por meio de rede é um tópico da computação que está cada vez mais comum, uma vez que atualmente muitos recursos ficam armazenados em servidores e acessados por clientes. Sendo assim, navegadores web utiliza sockets para requisitar páginas, sistemas também usam o recurso para comunicar com banco de dados, entre várias outras aplicações.

O principal objetivo do trabalho é colocar em prática os conceitos listados acima através da construção de uma aplicação que simule um jogo de batalha naval onde o cliente será o usuário e o servidor será parte da aplicação e a comunicação será através do uso do socket que deverá funcionar com IPv4 e IPV6.

Nas próximas seções serão detalhados: descrição da implementação do jogo e da conexão de rede, testes e conclusão.

Estrutura do jogo Batalha Naval

Resumidamente o jogo de Batalha Naval é para dois jogadores onde eles precisam adivinhar em quais campos estão os navios do oponente, tendo como objetivo ser o primeiro a derrubar todas as embarcações.

No programa desenvolvido, o tabuleiro possui 10 linhas (identificadas por letras de A - Z) e 10 colunas (identificadas por números no intervalo de 0 a 9), totalizando 100 campos. No total, são 4 tipos de embarcações, sendo elas:

- 1 porta-avião que ocupa 5 campos;
- 2 navios-tanque que ocupam 4 campos;
- 3 contratorpedeiros que ocupam 3 campos;
- 4 submarinos que ocupam 2 campos.

No total são 30 espaços ocupados pelas embarcações. Um navio é afundado quando todos os quadrados forem adivinhados. Ganha quem atirar nos 30 campos primeiro.

Implementação

O programa foi implementado utilizando a linguagem C e suas respectivas bibliotecas padrão, entre elas *socket*, no ambiente Linux de arquitetura 64 bits.

Principais estruturas de dados e estratégias de implementação

Tanto o cliente quanto o servidor possuem implementados dois campos de batalha naval que são chamados de oceanos:

- **Navios:** São representados por números da seguinte maneira:
 - Porta avião: possui 5 campos preenchidos com o número 5;
 - Navio-tanque: possui 4 campos preenchidos com o número 4;
 - Contratorpedeiros: possui 3 campos preenchidos com o número 3;
 - Submarino: possui 2 campos preenchidos com o número 2.

- **Oceano:** Matriz de dimensões 10 x 10 que representa o posicionamento dos navios de acordo com a estratégia apresentada no tópico anterior. Vale ressaltar também dois campos:
 - Campo com 1: Recebeu ataque onde havia navio posicionado;
 - Campo com -1: Recebeu ataque onde havia navio posicionado.
- **Oceano rival:** Matriz de dimensões 10 x 10 que representa o que o cliente ou servidor sabe sobre seu adversário. Se ele atacou em uma posição e acertou é preenchido com 1, caso contrário com -1. Inicialmente ela é inicializada com 0.

----- Meu campo (-1 e 1: onde re
---- Batalha Naval ----

	0	1	2	3	4	5	6	7	8	9
A	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	4	0	0	0	2
C	0	3	3	3	0	4	0	0	0	2
D	0	0	0	0	0	4	0	0	0	0
E	3	3	3	0	0	4	0	0	0	0
F	0	0	2	2	0	0	0	0	3	0
G	0	-1	5	5	5	5	5	0	3	0
H	0	0	0	0	2	2	0	-1	3	0
I	0	0	0	0	0	0	0	0	0	0
J	0	2	2	0	4	4	4	4	0	0

Imagem 1: Exemplo de matriz com a tropa posicionada e com dois ataques sem sucesso.

Envio de mensagem

A troca de mensagens é realizada seguindo o padrão: junção dos vetores *mensagem* e *atq*, no primeiro existe a linha e coluna do ataque, exemplo: A7. Já o segundo possui quatro opções de letras: A para acerto, C para indicar que o jogo iniciou, Z para indicar que o jogo acabou e E indicando erro.

```
Aguarde... Posicionando o porta aviao
Aguarde... Posicionando navios tanque
Aguarde... Posicionando contra torpedeiros
Aguarde... Posicionando submarinos
Enviado: G1C
Recebido: A6E
Enviado: H7A
```

Exemplo: o cliente atacou em A6 e informa que o servidor errou o tiro em G1. O servidor informa que o tiro do cliente foi certo e ataca em H7, assim sucessivamente.

Implementação do cliente

O cliente precisa dos seguintes parâmetros para funcionar: < IP > < Porta>. Uma vez recebido os dados ele cria um socket na estrutura IPv6, faz a conversão do IP caso necessário (se a entrada for IPV4). Com o endereço IP, é realizada a montagem da estrutura de dados do endereço (sockaddr_in6 e hostent) e por fim é estabelecida a conexão com o servidor. Todas as etapas até o momento foram realizadas em sua grande maioria com funções da biblioteca *socket*.

O cliente inicializa o seu *oceano* e o *oceano rival* com 0 e preenche o primeiro com o posicionamento da frota que é lido de um arquivo de nome: posicionamento de extensão .txt. O arquivo deverá conter o padrão de posicionamento conforme a lógica da estrutura de dados (conforme a imagem 1, sem os ataques).

Uma vez que a conexão está estabelecida o cliente aguarda uma mensagem do servidor que será enviada com um ataque. Quando a mensagem for recebida o cliente contra-ataca e assim sucessivamente. Observe que caso o cliente pontue 30 pontos é enviado 'Z' para o servidor e a conexão será encerrada.

Principais funções

- ***mensagem_servidor***: Recebe a mensagem enviada pelo servidor chamando as funções *oceano_rival_ataque* e *oceano_recebe_ataque* além de realizar a pontuação dado a resposta do servidor..

- ***envia_mensagem_servidor***: Estrutura a resposta que será enviada para o servidor, ou seja, a posição que será atacada e a resposta de acerto ou erro referente a última tentativa do servidor.
- ***escolhe_campo_ataque***: Função correspondente a escolha do ataque por parte do cliente, lida através do teclado.
- ***oceano_recebe_ataque***: Recebe o ataque do servidor. Caso exista um número diferente de 0, 1 ou -1 o servidor acertou uma tropa e o campo é preenchido com 1. Caso seja 0 não existe navio no campo, ou seja erro, é marcado -1. Neste momento é gravado a mensagem que será enviada: A ou E.
- ***oceano_rival_ataque***: Atualiza o que se sabe sobre o servidor. Se eu ataco na posição A7 e acerto, por exemplo, o campo correspondente no oceano rival será preenchido com 1, caso contrário, com -1.
- ***importa_posicao***: Faz a importação dos dados da frota do cliente.

Implementação do servidor

O servidor possui duas variáveis do tipo *sockaddr_in6* para local e remoto, que será utilizado para comunicação com o cliente enquanto o local é do próprio servidor. Após a criação do socket, o local é preenchido com os dados no padrão IPv6. A função *bind()* é utilizada e espera o cliente através do *listen*.

O servidor precisa dos seguintes parâmetros para funcionar: < Porta>. Todas as etapas até o momento foram realizadas em sua grande maioria com funções da biblioteca *socket*.

O servidor inicializa o seu *oceano* e o *oceano rival* com 0 e preenche o primeiro com o posicionamento da frota que é preenchido automaticamente utilizando valores aleatórios.

Uma vez que a conexão com o cliente esteja estabelecida, o servidor escolher um campo inicial para atacar o cliente e passa a escutá-lo. Quando recebe a resposta do cliente é verificado se houve acerto ou se o cliente ganhou, caso contrário a mensagem recebida será analisada e um novo ataque será realizado e assim sucessivamente.

Principais funções

- ***mensagem_cliente***: Interpreta o que foi recebido do cliente e chama as funções de cabeçalho ***oceano_recebe_ataque*** e ***oceano_rival_ataque*** que são semelhantes com as do cliente já detalhadas na seção acima.
- ***cria_posições***: Chama a função *posiciona_navio* para criar as frotas e suas respectivas quantidades.
- ***posiciona_navio***: Posiciona os navios no oceano, respeitando os limites da matriz. Escolhendo assim uma linha, coluna e direção (horizontal ou vertical) aleatoriamente.
- ***escolhe_campo_ataque***: Escolhe um campo para atacar o cliente de maneira aleatória ou um campo vizinho caso exista acerto na jogada anterior.
- ***envia_mensagem***: prepara o vetor de acordo com o padrão de linha, coluna e acerto/erro.

Como foi tratado o IPv4 e IPv6

O IPv4 transfere endereços de protocolos de 32 bits enquanto no IPv6 os IPs trabalham com 128 bits. Embora todos os serviços estejam sendo migrados para o novo padrão, ainda existe o problema das técnicas de adaptação do IPv4 para IPv6. Até alguns anos atrás estima-se que cerca de 90% dos computadores usavam o padrão antigo.

Neste trabalho o programa utiliza IPv6 como padrão de comunicação entre o cliente e servidor por padrão. A técnica utilizada para não ter problema com as versões foi a seguinte: caso uma entrada do host do cliente seja IPv4 o programa irá converter para IPv6 (Ex: 127.0.0.1 → ::FFFF:127.0.0.1), assim a conexão pode ser estabelecida independente do tipo de IP utilizado.

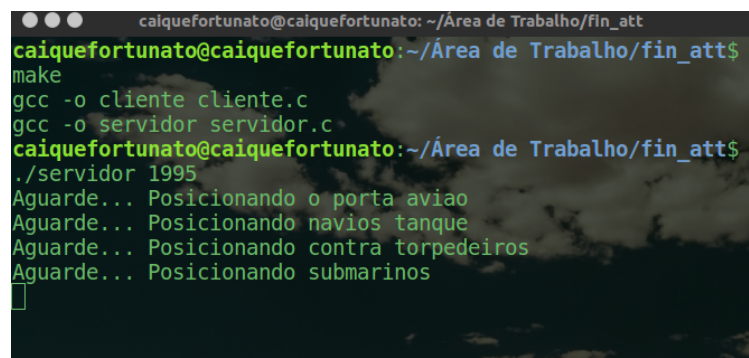
Testes

Os testes foram realizados em uma máquina com sistema operacional Linux com as seguintes configurações: 8GB de memória RAM, 1TB de HD, Processador Intel Core i5 5200U com 2.2GHz e distribuição Ubuntu 16.04 LTS.

Primeiro foi inicializado o servidor em uma porta aleatória: 1995 e o cliente com o IP local: 127.0.0.1 e a porta 1995. O servidor posicionou sua frota e enviou um ataque (posição B8) para o cliente que atirou em C6.

O cliente errou, foi atacado em B8 e clicou em P para ver a matriz. Depois atacou em H1 e acertou um navio, pontuou 1 ponto e foi atacado em F2.

A captura de tela dos passos pode ser visto abaixo, incluindo o comando make para compilar o programa.

A terminal window with a dark background and light green text. The window title is 'caiquefortunato@caiquefortunato: ~/Área de Trabalho/fin_att'. The user enters 'make', followed by 'gcc -o cliente cliente.c' and 'gcc -o servidor servidor.c'. Then they enter './servidor 1995'. The program outputs four status messages: 'Aguarde... Posicionando o porta aviao', 'Aguarde... Posicionando navios tanque', 'Aguarde... Posicionando contra torpedeiros', and 'Aguarde... Posicionando submarinos'. A cursor is visible on the line following the last message.

```
caiquefortunato@caiquefortunato: ~/Área de Trabalho/fin_att
caiquefortunato@caiquefortunato:~/Área de Trabalho/fin_att$ make
gcc -o cliente cliente.c
gcc -o servidor servidor.c
caiquefortunato@caiquefortunato:~/Área de Trabalho/fin_att$ ./servidor 1995
Aguarde... Posicionando o porta aviao
Aguarde... Posicionando navios tanque
Aguarde... Posicionando contra torpedeiros
Aguarde... Posicionando submarinos
```

Imagem 3: Iniciando o servidor


```

caiquefortunato@caiquefortunato:~/Área de Trabalho/fin_att$
./cliente 127.0.0.1 1995
Aguarde, posicionando navios...
Jogo comecou!

Voce foi atacado em: A3
Sua pontuacao: 0

Digite onde voce quer atacar
C6
Voce errou a ultima jogada!
Voce foi atacado em: B8
Sua pontuacao: 0

```

Imagem 4: Primeira interação do cliente

```

----- Meu campo (-1 e 1: onde recebi ataque) -----
----- Batalha Naval -----
  0  1  2  3  4  5  6  7  8  9
A  0  0  0 -1  0  0  0  0  0  0
B  0  0  0  0  0  4  0  0 -1  2
C  0  3  3  3  0  4  0  0  0  2
D  0  0  0  0  0  4  0  0  0  0
E  3  3  3  0  0  4  0  0  0  0
F  0  0  2  2  0  0  0  0  3  0
G  0  0  5  5  5  5  5  0  3  0
H  0  0  0  0  2  2  0  0  3  0
I  0  0  0  0  0  0  0  0  0  0
J  0  2  2  0  4  4  4  4  0  0
Digite onde voce quer atacar
H1
Voce acertou um navio!
Voce foi atacado em: F2
Sua pontuacao: 1
Digite onde voce quer atacar

```

Imagem 5: Segunda interação do cliente

Por se tratar de um loop que envolve sempre a troca de mensagens entre cliente e servidor, os outros passos serão omitidos da documentação uma vez que precisaria de no mínimo 30 interações para ganhar o jogo.

Conclusão

O primeiro trabalho prático foi essencial para compreender a comunicação de processos através de uma rede de computadores na prática. Foi necessário rever a teoria aprendida em sala de aula para aplicar e entender as funções do *socket* e principalmente conceitos sobre a camada de rede e transporte. Aprender a diferença entre TCP e UDP, IPv6 e IPv4 também foi construtivo para melhor compreensão e aprendizagem.

Aplicar todos esses conhecimentos em um jogo foi muito bom e divertido já que os testes foram feitos jogando, o que deixa o trabalho ainda mais interessante e motivador para programar.

As principais dificuldades encontradas foram na comunicação entre cliente e servidor além de alguns detalhes de implementação do jogo que não representaram grandes barreiras para o desenvolvimento.

Referências Bibliográficas

- 1. How to convert IPv4-mapped-IPv6 address to IPv4 (string format)?** Disponível em: <https://stackoverflow.com/questions/11233246/how-to-convert-ipv4-mapped-ipv6-address-to-ipv4-string-format?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa> Último acesso: 6 de maio de 2018.