Universidade Estadual de Campinas

Instituto de Computação - MC833 - A 2s2020

Atividade 5: Trabalho Final

Luana Felipe de Barros **RA: 201705**

Este trabalho tem por objetivo utilizar os conceitos aprendidos em aula, tais como comunicação TCP e UDP para realizar um jogo da velha.

Um servidor TCP é responsável por guardar uma lista de jogadores/clientes TCP. Para realizar essa tarefa, utilizei a função select para que pudesse ler do socket de escuta, e do socket de cada conexão TCP aceita. Utilizei um vetor de struct Client, no qual tinha informações sobre: IP, porta, valor do descritor do socket, variável indicadora da disponibilidade para jogar e uma variável para guardar os pontos obtidos nas partidas.

Tentei utilizar Fork anteriormente ao select, na tentativa de fazer o servidor concorrente. No entanto, como o processo filho é uma cópia do pai, percebi que as variáveis do pai que precisaria modificar no livro seriam copiadas para uma outra posição de memória exclusiva do filho, não acessando a variável original. Por isso, optei pela multiplexação.

Ocorreram alguns problemas na implementaçao do servidor TCP com select. Primeiramente, utilizei a chamada de sistemas read para ler dos descritores, mas logo percebi que esta era uma primitiva bloqueante. Assim, a atualização dos descritores do grupo de leitura era tardia, até que um cliente enviasse algo para o servidor, desbloqueando o read. Uma ideia para melhoria seria colocar um timeout para não bloquear para sempre.

No cliente, no mesmo arquivo tive a ideia de usar select novamente para ler do socket TCP e UDP. Assim, após a conexão TCP ser feita, um servidor UDP é iniciado, utilizando como porta de escuta a mesma porta local do cliente TCP. Escolhi esta maneira pois a lista disponibilizada com as informações dos jogadores apresenta a porta, e escolhendo por ela ficaria mais fácil de se iniciar uma comunicação UDP entre clientes.

No entanto, tive problemas ao utilizar select também, visto que só conseguiria fazer as verificações no cliente se ao menos um dos descritores forem setados. Porém, algumas chamadas bloquentes no código dificultavam esse processo. Então, utilizei o timeout no select, atualizando os valores a cada 1s.

No cliente existe um laço que fica perguntando para o servidor, caso um dos descritores do grupo de leitura seja setado, uma lista de usuários disponíveis. E em seguida, mostra na tela e pergunta o número de porta na qual o cliente deseja jogar com. Além disso, a opção 0 sai do programa e a opção 1 força continuar o laço, mostrando a lista de jogadores disponíveis novamente.

Inicalmente, nenhum jogador vai aparecer, já que apenas 1 está conectado ao servidor. Ao conectar um segundo cliente, este irá aparecer na tela do outro cliente, caso este atualize a lista. O que mais foi prejudicial nesta etapa foi o fato do servidor esperar qual a porta do jogador que você escolheu. O fato de ser bloqueante, afeta os outros clientes.

Quando um jogador seleciona uma porta de outro cliente, a porta é enviada para o servidor. O servidor então avisa o cliente desta porta qual é a porta na qual ele deve aceitar mensagens UDP, na finalidade de não receber mensagens de outros clientes. Então, apesar do servidor estar ouvindo desde o começo da execução, há uma verificação sobre qual porta ele deve aceitar as mensagens.

O cliente que selecionou a porta e a enviou ao servidor, inicia então um cliente UDP para tentar se comunicar com o outro jogador. Além disso, envia uma mensagem para o servidor avisando que não está mais disponível para jogar com outros. Envia então uma mensagem de Hello para o outro jogador. Utilizei connect no client UDP para enviar a mensagem.

O outro jogadador, por ter no select verificações no descritor sockudp, identifica quando chega uma mensagem UDP. Por conta dos problemas encontrados, não consegui a tempo realizar o jogo da velha. A intenção era simular e depois de fato começar a implementação do jogo, já que o foco seria a comunicação em si. Portanto, coloquei um sleep de 10 segundos nos dois jogadores, simulando quem ganha e quem vence. Este jogador mostra a mensagem de Hello recebida, e então avisa o servidor que também não está mais disponível para jogar com outros. Após o tempo de simulação, ambos jogadores enviam uma mensagem para o servidor do resultado da partida, que pode ser do tipo ‘won’ ou ‘lose’.

No servidor há verificação para vários tipos de mensagens TCP que recebe. Se receber “get” mostra a lista de jogadores disponíveis, exceto o jogador que a pediu. Se receber “playing”, indica que o jogador está jogando, e seta-o como indisponível. Se receber “won”, acrescenta um ponto na variavel do jogador. Caso receba “lose”, não modifica a pontuação. Nesses dois últimos tipos, o jogador é setado como disponível para jogar novamente.

Obtive problemas de sincronização, como se um jogador escolhe outro para iniciar uma partida, mas este está vendo a lista de jogadores e não atualiza, não há um mecanismo que interrompa esse evento e mostre que ele tem um jogo pendente.

Além disso, obtive erros utilizando o timeout. Dependendo de seu valor, as mensagens que chegam ficavam acumuladas. Também obtive erros com o read bloqueante, mesmo fazendo a verificação do número de bytes, caso o cliente terminasse (fechando a conexão inclusive), o servidor esporadicamente dava o erro read error: Connection reset by peer.

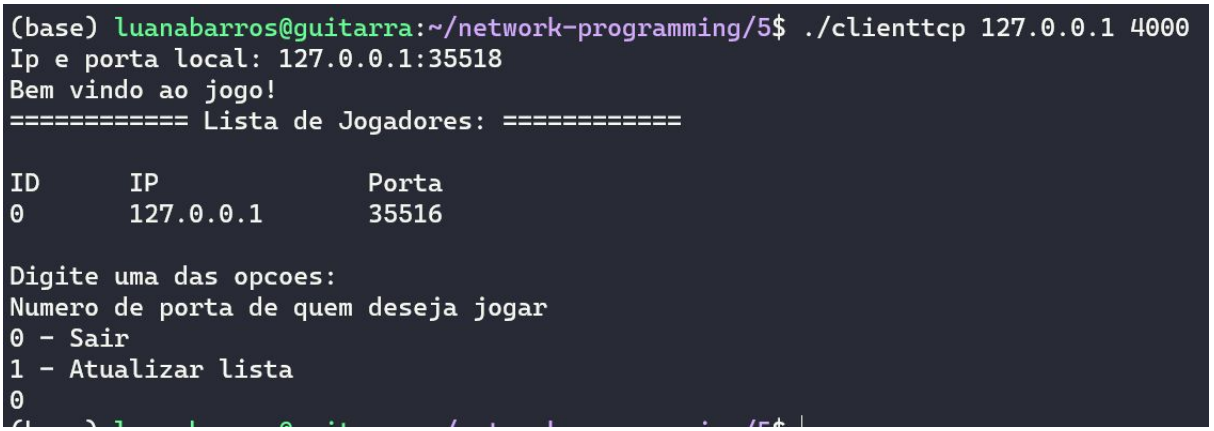
```
(base) luanabarros@guitarra:~/network-programming/5$ ./clienttcp 127.0.0.1 4000
Ip e porta local: 127.0.0.1:35514
Bem vindo ao jogo!
============ Lista de Jogadores: ============

ID      IP              Porta
0       127.0.0.1       35512

Digite uma das opcoes:
Numero de porta de quem deseja jogar
0 - Sair
1 - Atualizar lista
35512
Pedindo para jogar com: 0.0.0.0:35512
Estamos jogando...
Voce venceu
Nao tem jogadores disponiveis, tentando dnv...

============ Lista de Jogadores: ============

ID      IP              Porta
0       127.0.0.1       35512

Digite uma das opcoes:
Numero de porta de quem deseja jogar
0 - Sair
1 - Atualizar lista
0
```

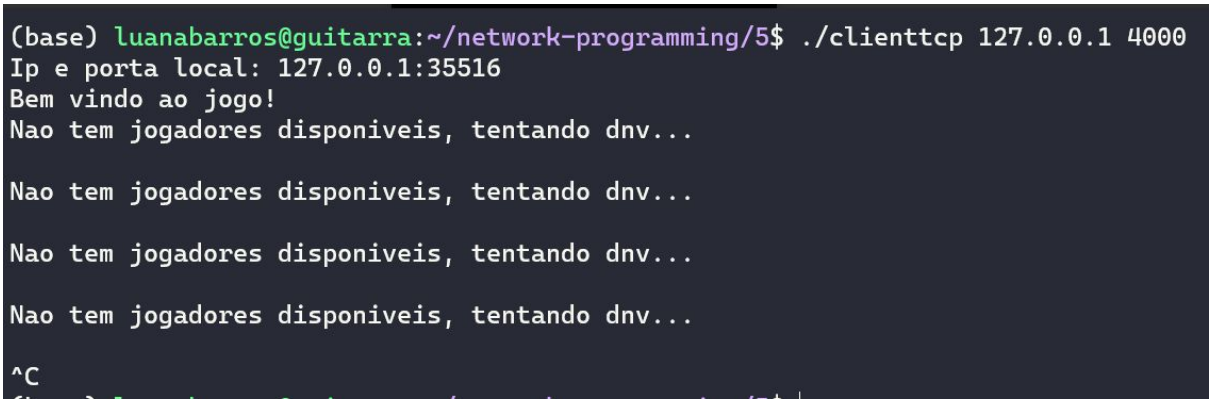
```
(base) luanabarros@guitarra:~/network-programming/5$ ./clienttcp 127.0.0.1 4000
Ip e porta local: 127.0.0.1:35512
Bem vindo ao jogo!
Nao tem jogadores disponiveis, tentando dnv...

============ Lista de Jogadores: ============

ID      IP              Porta
1       127.0.0.1       35514

Digite uma das opcoes:
Numero de porta de quem deseja jogar
0 - Sair
1 - Atualizar lista
1
O jogador aceitou jogar e enviou: Hello

Estamos jogando...
Voce perdeu
============ Lista de Jogadores: ============

ID      IP              Porta
35514NULL
Digite uma das opcoes:
Numero de porta de quem deseja jogar
0 - Sair
1 - Atualizar lista
1
============ Lista de Jogadores: ============

ID      IP              Porta
1       127.0.0.1       35514

Digite uma das opcoes:
Numero de porta de quem deseja jogar
0 - Sair
1 - Atualizar lista
0
```

```
(base) luanabarros@guitarra:~/network-programming/5$ ./selectserver 4000

ID      IP              Port    Free    Socket  Points
0       127.0.0.1       35512   1       4       0

ID      IP              Port    Free    Socket  Points
0       127.0.0.1       35512   1       4       0
1       127.0.0.1       35514   1       5       0

ID      IP              Port    Free    Socket  Points
0       127.0.0.1       35512   1       4       0
1       127.0.0.1       35514   1       5       0

ID      IP              Port    Free    Socket  Points
0       127.0.0.1       35512   1       4       0
1       127.0.0.1       35514   0       5       0

ID      IP              Port    Free    Socket  Points
0       127.0.0.1       35512   0       4       0
1       127.0.0.1       35514   1       5       1

ID      IP              Port    Free    Socket  Points
0       127.0.0.1       35512   0       4       0
1       127.0.0.1       35514   1       5       1

ID      IP              Port    Free    Socket  Points
0       127.0.0.1       35512   1       4       0
1       127.0.0.1       35514   1       5       1

ID      IP              Port    Free    Socket  Points
0       127.0.0.1       35512   1       4       0
1       127.0.0.1       35514   1       5       1

ID      IP              Port    Free    Socket  Points
0       127.0.0.1       35512   1       4       0
1       127.0.0.1       35514   1       5       1

ID      IP              Port    Free    Socket  Points
0       127.0.0.1       35512   1       4       0
1       127.0.0.1       35514   1       5       1
read error: Connection reset by peer
(base) luanabarros@guitarra:~/network-programming/5$
```

No entanto, em vários testes que fiz, para verificar se quando um cliente saísse, ele seria removido da fila, funcionou normalmente.

```
(base) luanabarros@guitarra:~/network-programming/5$ ./selectserver 4000

ID      IP              Port    Free    Socket  Points
0       127.0.0.1       35516   1       4       0

ID      IP              Port    Free    Socket  Points
0       127.0.0.1       35516   1       4       0
1       127.0.0.1       35518   1       5       0

ID      IP              Port    Free    Socket  Points
0       127.0.0.1       35516   1       4       0

ID      IP              Port    Free    Socket  Points
0       127.0.0.1       35516   1       4       0

ID      IP              Port    Free    Socket  Points
0       127.0.0.1       35516   1       4       0
^C
(base) luanabarros@guitarra:~/network-programming/5$
```





Por contas desses inúmeros erros, não consegui continuar com o problema proposto.

Como compilar:

servidor: `gcc selectserver.c -o selectserver -Wall`

cliente (tcp + servidor e cliente udp): `gcc clienttcp.c -o clienttcp -Wall`

`Execução:`

`servidor: ./selectserver porta`

`jogador 1: ./clienttcp 127.0.0.1 4000`

`jogador 2: ./clienttcp 127.0.0.1 4000`

e assim sucessivamente…

O jogador deve apenas escolher entre a porta, opção 0 (sair) e opção 1 (atualizar a lista).