

Exercício 5: Trabalho Final

# **Aluna:** Naomi Takemoto

# **RA:** 184849

# Instituto de Computação

# Universidade Estadual de Campinas

# Janeiro de 2021

# Instruções de execução

# Resumo

Segue abaixo o resumo do funcionamento do jogo:

1. Ao iniciar o cliente, o usuário coloca seu id na pela entrada padrão
2. O cliente se conecta ao servidor (usando protocolo TCP)
3. O cliente manda o seu player id (assumindo que ele é único)
4. O receber o player id o servidor coloca o cliente na lista de jogadores disponíveis para uma partida e manda em resposta a lista de jogadores disponíveis.
5. O cliente recebe a lista de jogadores disponíveis. Se a lista não estiver vazia escolhe um deles (exceto ele mesmo) como oponente e manda para o servidor o seu id e o do oponente.
6. O servidor recebe os ids dos jogadores e os retira da lista de jogadores disponíveis.
7. Os clientes oponentes se enfrentam
8. Se o cliente se desconecta (manda uma mensagem vazia para o servidor).
   1. O servidor retira o id do cliente da lista de jogadores disponíveis.
9. Se o cliente manda o resultado do jogo
   1. O servidor coloca o id do jogador na lista de jogadores disponíveis

# Funcionamento do programa

Para a implementação do cliente, optou-se por utilizar uma variável de estado que poderia assumir os seguintes valores:

**Tabela 1: estados do cliente**

|  |  |
| --- | --- |
| **Estado** | **Descrição** |
| PENDING\_LOGIN | O jogador tem que enviar ao servidor o seu id, para poder entrar na lista de jogadores disponíveis. |
| WAITING\_PLAYERS\_LIST | Depois de sair do estado acima, o jogador está na espera para que o servidor mande uma lista de jogadores disponíveis. |
| WAITING\_NEW\_PLAYER | O jogador (X) escolhe um dos jogadores disponíveis e manda o id para o servidor.  O servidor recebe o id do oponente e manda o convite para o jogador escolhido (Y) |
| WAITING\_ACCEPT | Se (Y) aceitar, ele manda uma mensagem de Accept para o servidor. O servidor retira X e Y da lista de jogadores disponíveis. Ambos X entra no estado PLAYER\_TURN e Y entra no estado WAITING\_OTHER\_PLAYER\_MOVE.  Se (Y) recusar, o servidor manda uma mensagem de Deny para X volta ao estado WAITING\_PLAYERS\_LIST. |
| PLAYER\_TURN | O X deve fazer sua jogada, que consiste em colocar um movimento válido na entrada padrão e enviar ao oponente via uma conexão UDP. |
| WAITING\_OTHER\_PLAYER\_MOVE | O Y espera uma o oponente fazer sua jogada. Quando ele recebe o movimento do oponente, se este for o primeiro turno ele deve calcular qual a sua peça ( x ou o ) baseado na escolha do oponente. |
| GAME\_OVER | X ou Y detecta o estado do jogo e manda o seu resultado pra o servidor. Neste estado o jogador pode pedir os Scores para o servidor:  Se pedir: vai par o estado WAITING\_SCORES  Caso contrário: volta o para o estado PENDING\_LOGIN. |
| WAITING\_SCORES | O cliente requisitou os scores e espera o resultado do servidor. Ao receber ele volta para o estado PENDING\_LOGIN. |

Nas tabelas abaixo são descritos os formatos de request e de response, que são strings. Cada token é separado por um espaço seguindo a convenção:

**<request/response name>:** <token1> <token1> …

O | é o operador lógico “ou”.

**Tabela 2:** **Formato das requests**

|  |  |
| --- | --- |
| **Request** | **Descrição** |
| **login:** self\_player\_id port | O cliente manda sou id para o servidor |
| **invite:** self\_player\_id opponent\_player\_id port | O cliente manda seu id e o do oponente |
| **accept:** self\_player\_id opponent\_player\_id yes | O cliente manda seu id e o do oponente e aceita a partida |
| **deny:** self\_player\_id opponent\_player\_id no | O cliente manda seu id e o do oponente e nega a partida |
| **uploadResult:** self\_player\_id DRAW | WIN | LOSS | O cliente manda seu id e o resultado de seu jogo. |
| **requestScore:** self\_player\_id | O cliente manda o seu id e pede os Scores. |
|  |  |

**Tabela 3:** **formato das responses**

|  |  |
| --- | --- |
| **Response** |  |
| availablePlayers: player\_id’s | O servidor retorna a lista de players disponíveis |
| accept: “yes” | “no” address port | O servidor informa que o oponente aceitou a partida, envia o endereço e a porta do oponente. |
| scores: | O servidor retorna os scores de todos os jogadores |
| Invitation: player\_id address port | O servidor manda para o jogador X o convite feito pelo jogador Y. |

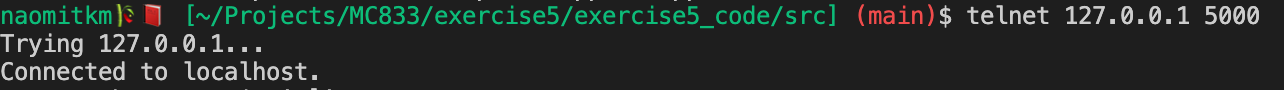
# Implementação do Servidor

Para que fosse possível lidar com diversos clientes sem o uso de threads ou processos filhos no servidor, foi utilizada a multiplexação de IO com o comando select, que permite o monitoramento de diversos file descriptors [2].

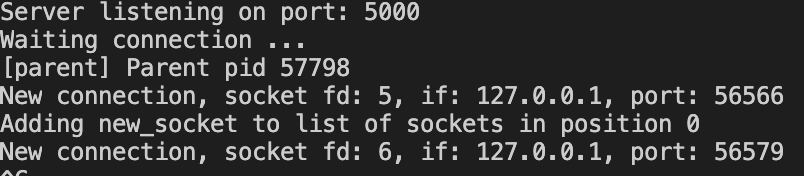
Para testar se o servidor fato conseguia suportar várias conexões TCP simultâneas, foi utilizado o comando telnet em diversas chamadas concorrentes (diferentes abas do terminal).

telnet localhost <#porta>

Neste exemplo em específico, foram criadas duas conexões (executando o comando abaixo em duas abas distintas do termninal)



Nos screenshots abaixo são mostradas a saída do servidor: os socket file descriptors associados a cada conexão telnet, IP e porta sendo utilizados.



# Implementação do Cliente

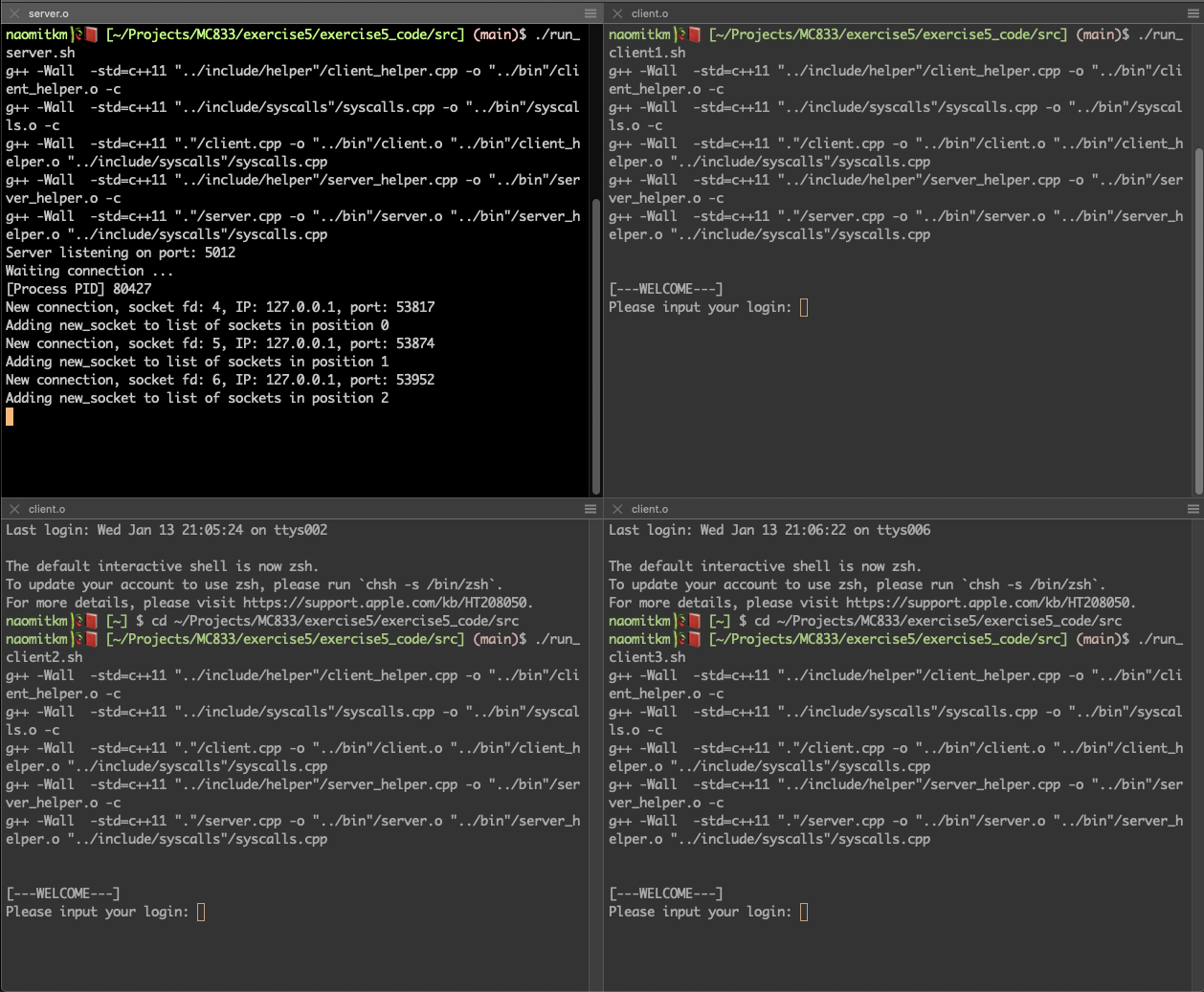
Assim como o servidor, o cliente utiliza a multiplexação de IO para possibilitar a comunicação não bloqueante:

1. Com o servidor
2. Com o oponente
3. Com o usuário, via entrada padrão

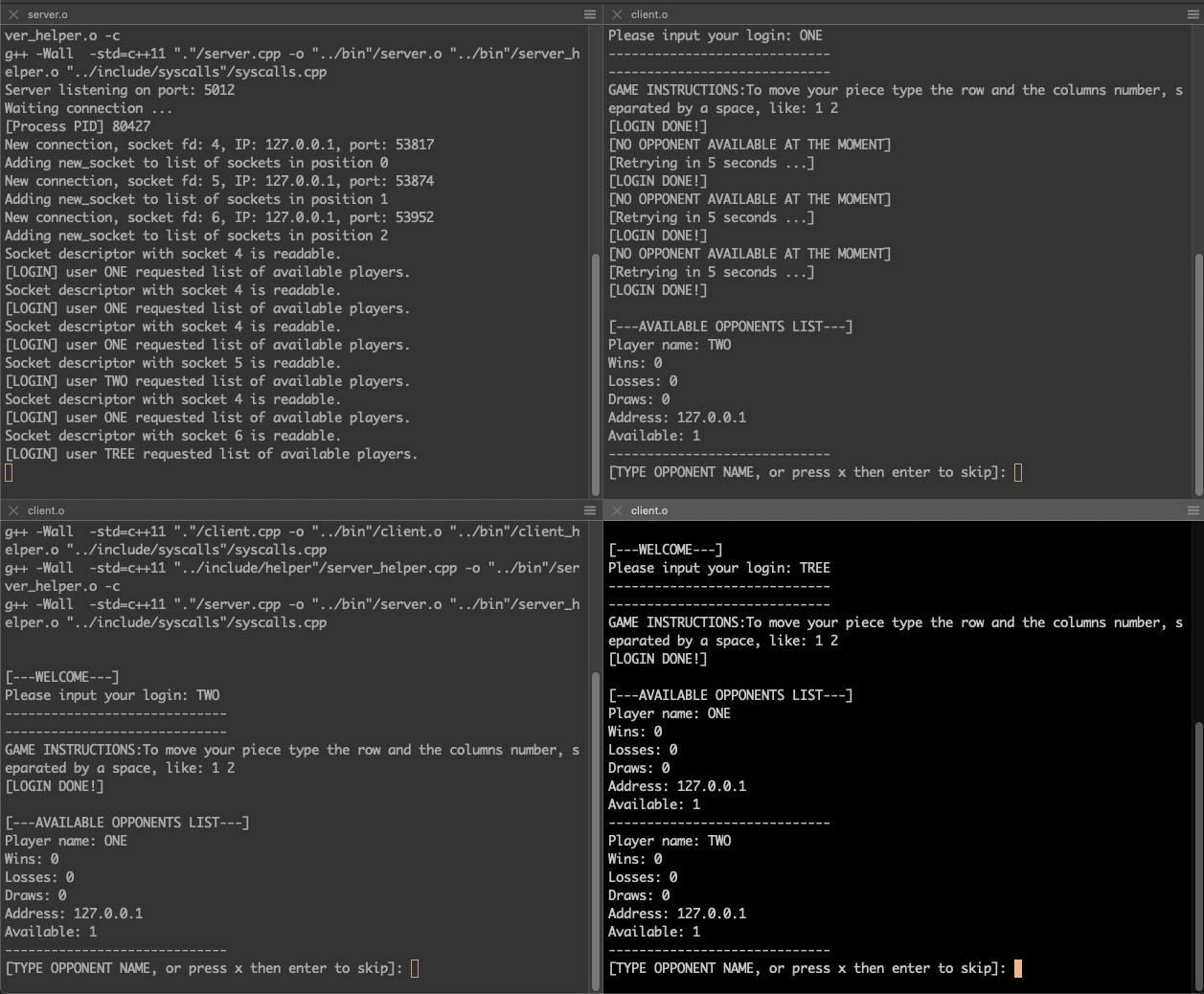
# Exemplo de execução:

1. Iniciar o servidor e os clientes: rodando os scripts
   1. run\_server.sh
   2. run\_client1.sh
   3. run\_client2.sh
   4. run\_client3.sh

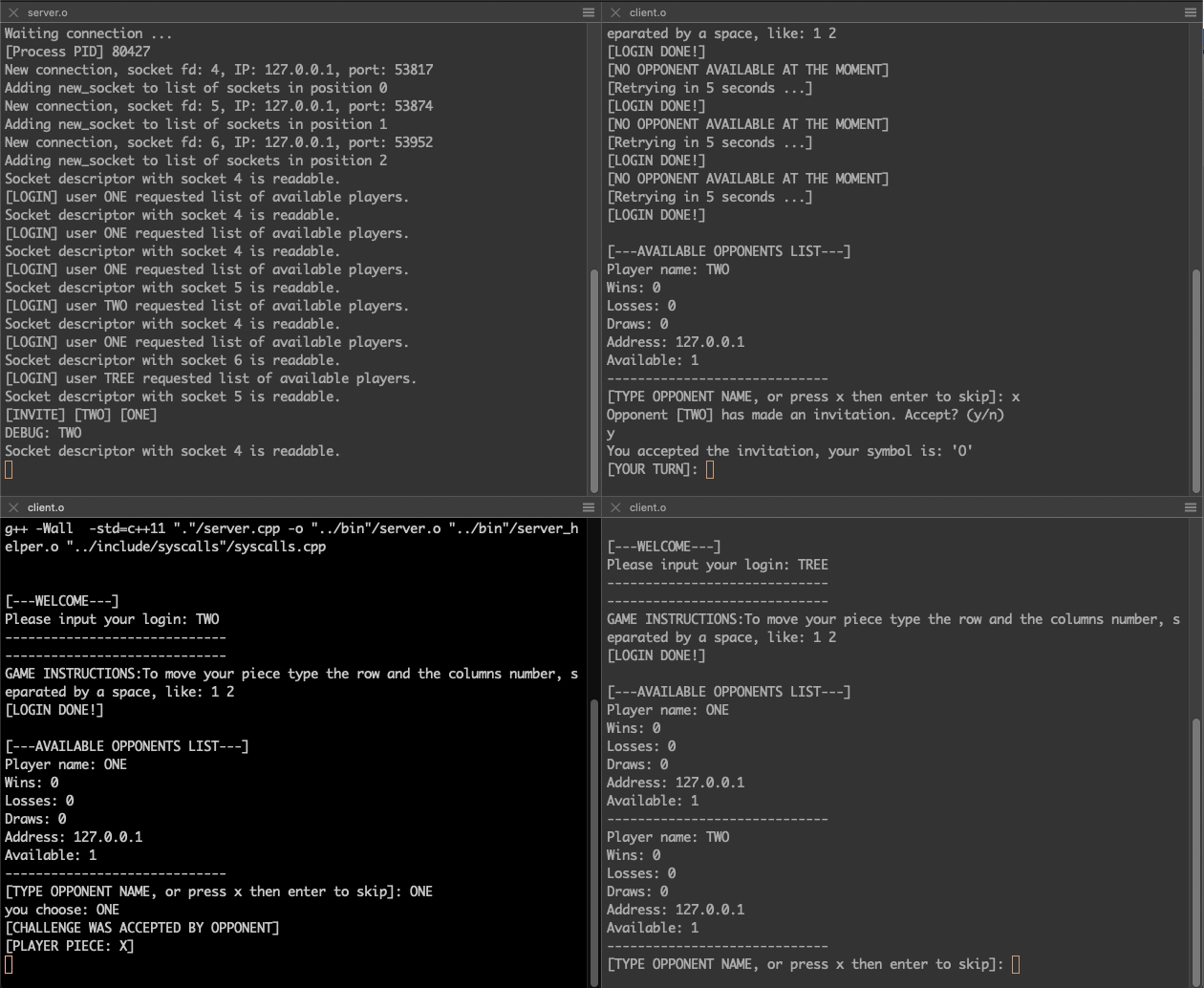
Da esquerda para direita, de cima para baixo, na figura a seguir, temos as saídas do servidor e dos clientes 1,2 e3;



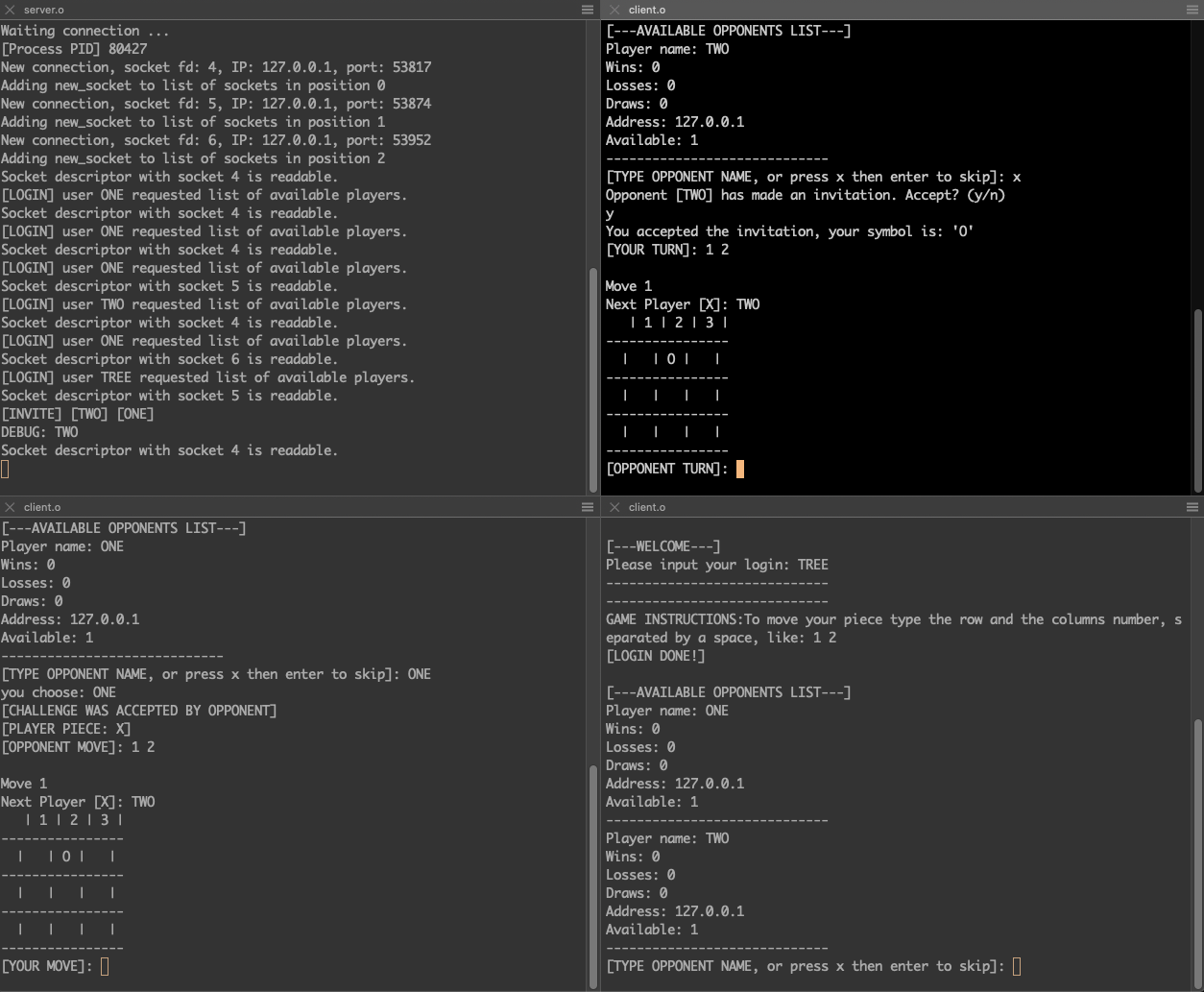
1. Cliente 1, 2 e 3 fazem login. O, quando o cliente 1 o faz, não há jogadores disponíveis então o cliente espera 5 segundos para fazer uma nova requisião da lista de jogadores para o servidor.



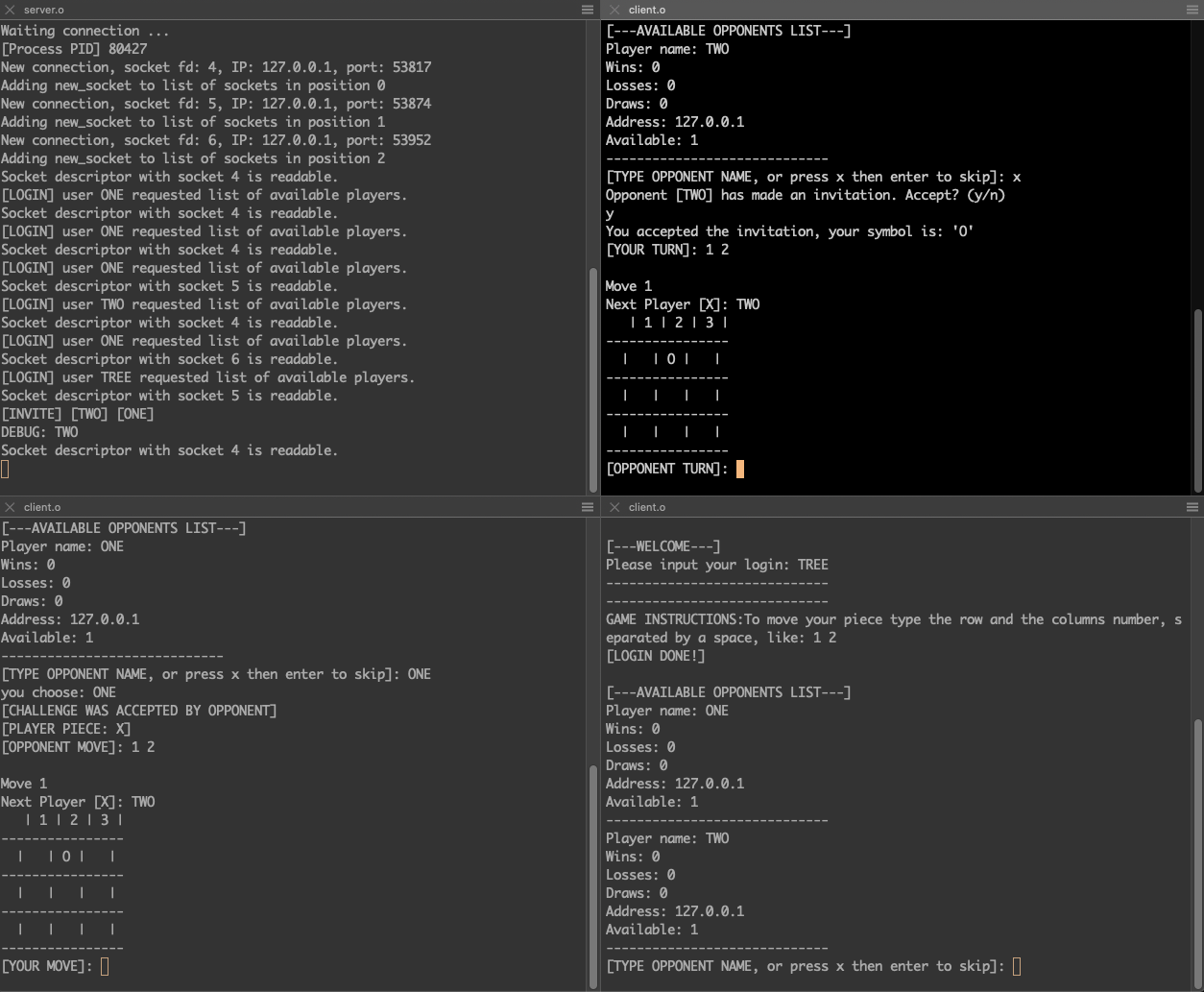
1. O cliente 2 (TWO) decide convidar o cliente 1 (ONE). O cliente 1 decide não convidar ninguém e aperta ‘x’ para pular esse passo. O cliente 1 recebe o convite, intermediado pelo servidor, e o jogo se inicia.



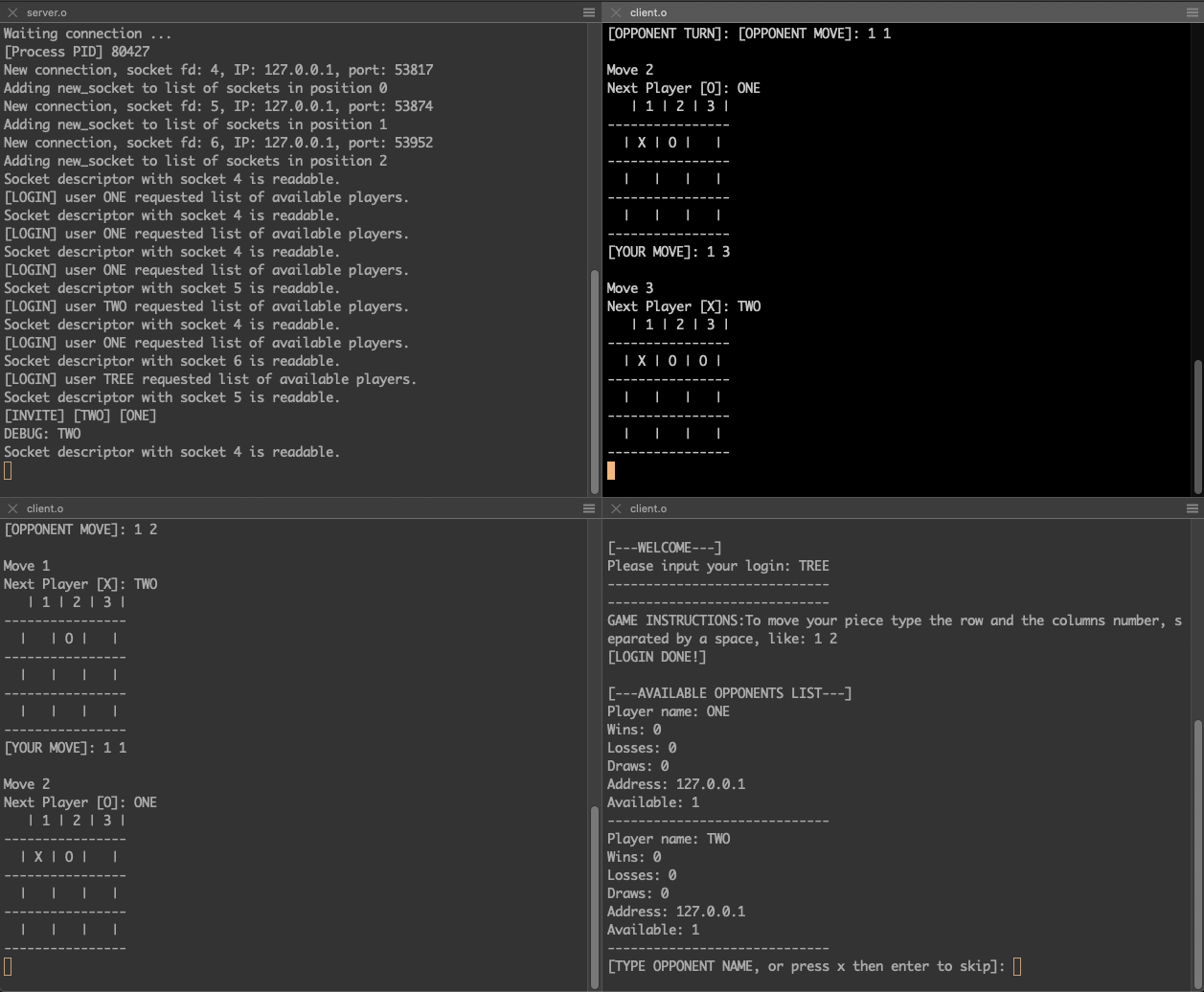
1. O cliente 1 faz a primeira jogada (linha 1, coluna 2)



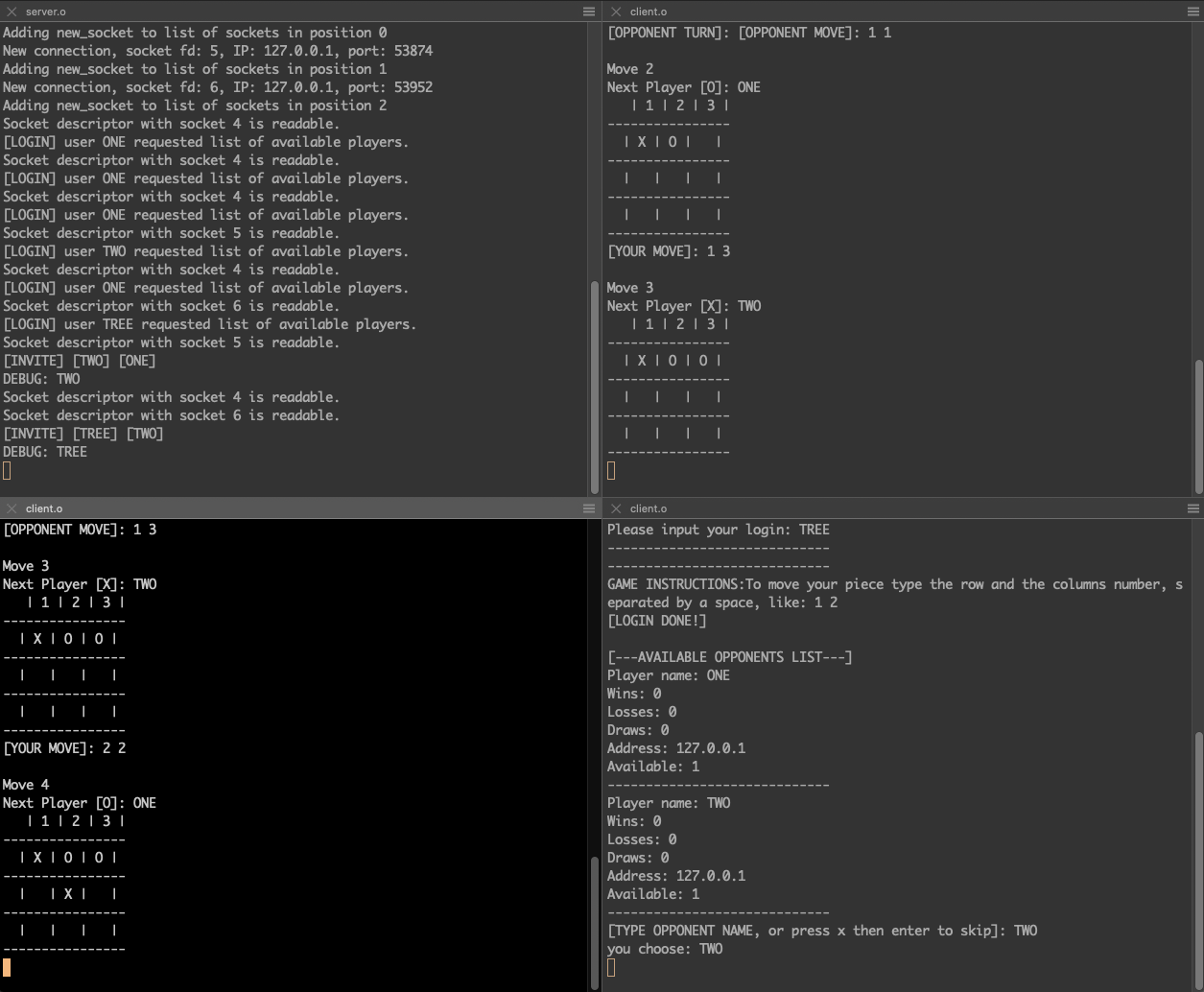
1. O cliente 2 faz a jogada 1 1



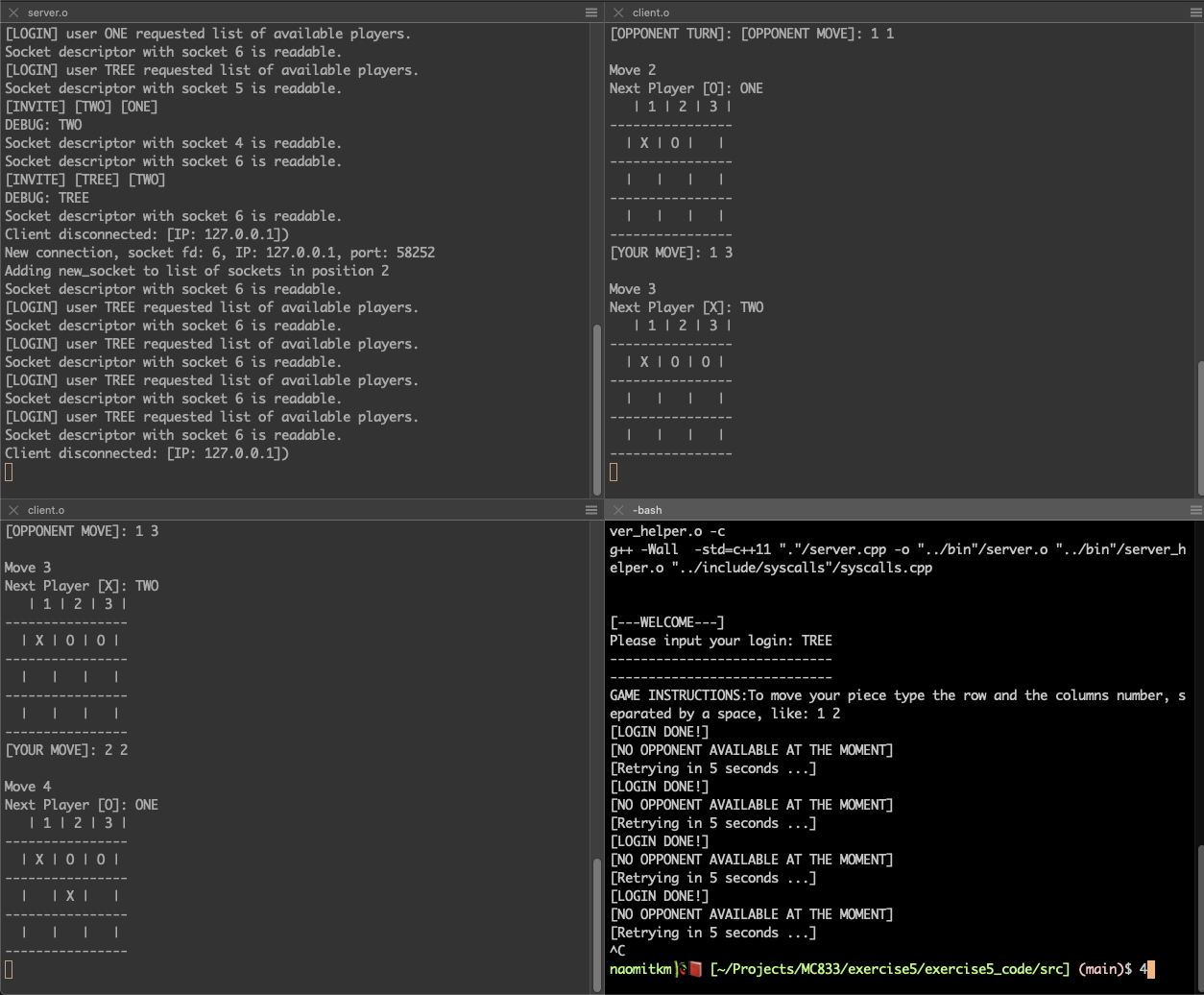
1. O cliente 1 move 1 3



1. O cliente 2 move 2 2, o cliente 3 manda um convite para o cliente 2, por intermédio do servidor.

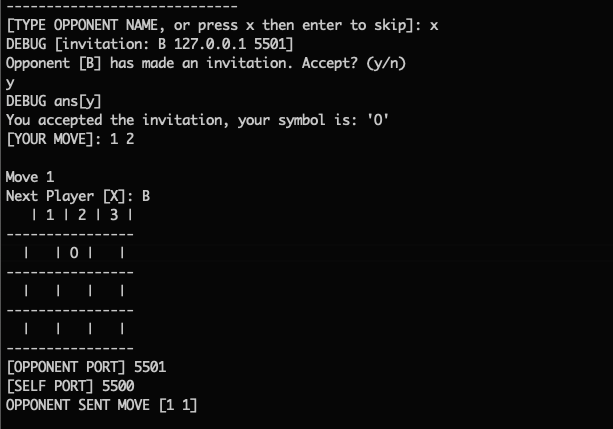


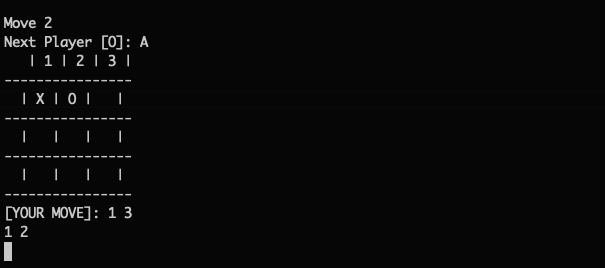
1. O cliente 3 se desconecta e conecta novamente, nenhum cliente está disponível para o jogo então de 5 em 5 segundos ele pede ao servidor para mandar a lista de jogadores disponíveis.





# 





# 

# 

# Referências

[1] Unix Network Programming, Volume 1: The Sockets Networking API (3rd Edition). Disponível em <https://www.amazon.com.br/Unix-Network-Programming-Sockets-Networking/dp/0131411551>.

[2] Socket Programming in C/C++: Handling multiple clients on server without multi threading. Disponível em <https://www.geeksforgeeks.org/socket-programming-in-cc-handling-multiple-clients-on-server-without-multi-threading/>.

[3] <https://www.geeksforgeeks.org/udp-server-client-implementation-c/>