



**Departamento de  
Informática**

Licenciatura em Engenharia Informática

Unidade Curricular: Sistemas Operativos

Docente: Paul Andrew Crocker

## ***Inter-Process Communication***

**Elementos do Grupo:**

Alessandra Delgado – 51713

Matilde Sequeira – 51752

5 de junho de 2025

# 1. Implementação do Servidor

Para este projeto foi implementado um servidor que contém a lógica do jogo do galo. Esta abordagem permite que o cliente não tenha que implementar a lógica do jogo, tendo apenas de tratar da informação de acordo com os protocolos recebidos pelo servidor, permitindo também, e de forma modular, a hospedagem de diferentes sessões de jogo para vários utilizadores, em sessões de *bash* diferentes. Para tal, optou-se pela criação de apenas uma *thread* principal para o servidor — *dispatcher* — que trata do despacho dos clientes para as sessões de jogo. Cada sessão de jogo é gerida por uma *thread* de sessão, criada a cada par de jogadores que se conecta ao servidor. A conexão e toda a comunicação com o servidor é feita através de uma *message-queue*, em que o servidor tem tipo de mensagem 1. Inicialmente, um cliente envia ao servidor o seu pid, e este será posto numa lista de espera. Depois de associado a uma sessão, a comunicação entre o servidor e o cliente será feita com a sua *thread* de sessão, cada uma com tipo de mensagem igual ao id da sessão. A utilização de uma *message-queue*, neste caso, possibilita a criação de apenas  $1 + N/2$  *threads* no lado do servidor (uma para despachar e uma para cada  $N$  clientes em sessões), enquanto que o uso de pipes necessitaria a criação de  $2 + 2N$  *threads*. Para uma utilização eficiente da *message-queue*, foi criado um *enum* que estabelece os protocolos das mensagens a serem trocadas entre o cliente e o servidor.

Para terminar o servidor, uma vez que `msgrcv()` é uma chamada de sistema bloqueante, foi criado um *handler* associado ao `SIGINT` para que este envie uma mensagem ao próprio servidor. O servidor ao receber a sua própria mensagem encerra de forma graciosa (espera que a ultima *thread* de sessão lhe mande um sinal via semáforo) ou abrupta, (se a ultima *thread* não terminar em cinco segundos, o servidor encerra, removendo a fila de mensagens). Para efeitos de sincronização e proteção de variáveis foram ainda utilizados um *mutex* (aplicado ao conjunto de sessões) e um semáforo (para a sincronização no *shutdown*, utilizando `sem_timedwait()`). Por fim, a lógica do jogo do galo foi implementada com auxílio de uma classe, em que são apenas utilizadas *bit masks* para guardar o tabuleiro e as jogadas dos jogadores, e uma *array* de caracteres que guarda o tabuleiro a ser mostrado no ecrã.

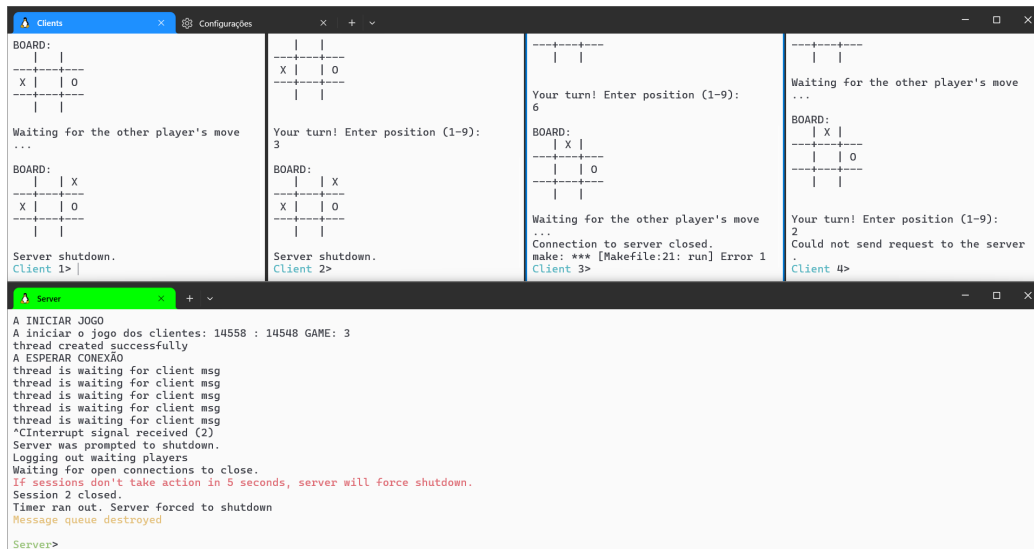
## 2. Implementação do Cliente

O cliente deste projeto comunica com o servidor através de uma *message queue* e é composto por uma única *thread*. Consiste num único loop, onde:

- Recebe a mensagem do servidor, e converte-a para string
- Processa a string. Se o cabeçalho (protocolo) for:
  - *BOARD*: extrai a informação do tabuleiro e exhibia no terminal;
  - *MOVE*: pede a posição ao jogador, e depois envia essa informação para o servidor;
  - *WAIT*: imprime a mensagem "*Waiting for the other player's move...*" no terminal;
  - *WIN*: imprime a mensagem "*Game over! You win!!*" no terminal do jogador que ganhou e termina o jogo;
  - *LOSE*: imprime a mensagem "*Game over! You lost.*" no terminal do jogador que perdeu e termina o jogo;
  - *DRAW*: imprime a mensagem *Game over!It's a draw!* no terminal dos dois jogadores quando há empate e termina o jogo;
  - *INVALID*: indica que houve um movimento invalido e volta ao início do *loop*;

### 3. Execução do Programa

Para compilar este projeto, são utilizados três **Makefiles**: um para o cliente, outro para o servidor e um principal, que invoca os outros dois **Makefiles** (*build* e limpeza) na diretoria raiz do projeto. Por serem duas componentes distintas (cliente e servidor), devem ser executadas em sessões de *bash* diferentes (pode usar o comando **make run**, mas deve estar nas diretorias raiz para cada um dos projetos).



```

Clients
---
BOARD:
| | |
| | |
X | | O
| | |
Waiting for the other player's move
...
BOARD:
| | X
| | |
X | | O
| | |
Server shutdown.
Client 1>

Your turn! Enter position (1-9):
3
BOARD:
| | X
| | |
X | | O
| | |
Server shutdown.
Client 2>

Your turn! Enter position (1-9):
6
BOARD:
| X |
| | |
| | O
| | |
Waiting for the other player's move
...
Connection to server closed.
make: *** [Makefile:21: run] Error 1
Client 3>

Waiting for the other player's move
...
BOARD:
| X |
| | |
| | O
| | |
Your turn! Enter position (1-9):
2
Could not send request to the server
.
Client 4>

Server
---
A INICIAR JOGO
A iniciar o jogo dos clientes: 14558 : 14548 GAME: 3
thread created successfully
A ESPERAR CONEXÃO
thread is waiting for client msg
thread is waiting for client msg
thread is waiting for client msg
thread is waiting for client msg
thread is waiting for client msg
^CInterrupt signal received (2)
Server was prompted to shutdown.
Logging out waiting players
Waiting for open connections to close.
If sessions don't take action in 5 seconds, server will force shutdown.
Session 2 closed.
Timer ran out. Server forced to shutdown
Message queue destroyed
Server>
```

Figura 3.1: Exemplo de execução, com duas sessões de jogo.