UNIVERSIDADE da MADEIRA
Faculdade de Ciências Exatas e da Engenharia

Sistemas Operativos

Projeto Prático, 2024/2025, 1º Semestre

Servidor Sudoku e Multi Jogadores

Docentes
Eduardo Marques | Fernando Martins
[eduardo.marques@staff.uma.pt | fernando.martins@staff.uma.pt]

1 Objetivos

O objetivo deste trabalho é conceber e implementar uma aplicação cliente/servidor para o jogo Sudoku. Pretende-se assim que os alunos pratiquem os conceitos aprendidos nas aulas teóricas e práticas na conceção e desenvolvimento de um sistema simples, empregando os mecanismos de concorrência, sincronização e comunicação existentes na linguagem C.

Os Sistemas Operativos, como gestores de recursos, pretendem-se melhor compreendidos através de uma simulação que terá muitas analogias com os mecanismos reais. O projeto permite a implementação de soluções diversas, de arquiteturas mais simples e algumas mais complexas e próximas da realidade. Os grupos devem, junto dos docentes, discutir as abordagens e opções possíveis, de forma a melhor compreenderem o esforço necessário para a implementação dos projetos.

2 Descrição

Sudoku é um jogo baseado na colocação lógica de números. O objetivo é preencher as células vazias de uma grelha 9x9 com números de 1 a 9, respeitando determinadas regras. A grelha está subdividida em nove regiões de 3x3, e algumas células já contêm números predefinidos como pistas iniciais. A partir destas pistas, o jogador deve deduzir os números que faltam, aplicando raciocínio lógico. Cada coluna, linha e região deve conter todos os números de 1 a 9, sem repetições. A resolução do quebra-cabeças requer raciocínio lógico e paciência. Uma grelha de exemplo pode ser visualizada na Figura 1[1].

Os jogadores (clientes) devem pedir jogos novos ao servidor e devem enviar as soluções completas e/ou parciais. Os jogos podem ser individuais ou em competição.

As aplicações a implementar pretendem avaliar as condições de acesso aos jogos em concorrência, gestão das comunicações, e contagem de estatísticas, como tempos de resolução,

[1] fonte imagem: https://pt.wikipedia.org/wiki/Sudoku

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Figura 1: Exemplo de um jogo inicial de Sodoku

número de verificações de solução parciais, número de jogadores simultâneos, entre outros.

Não é necessária a implementação de algoritmos de resolução diversos, bastando um algoritmo que teste todas as possibilidades até acertar. Podem ser usados outros algoritmos, mas não é o foco do projeto.

3 Arquitetura

O sistema a desenvolver deverá conter duas aplicações, a primeira (Servidor) que efetuará toda a gestão de jogos e análise estatística, e a segunda (Cliente) que receberá os jogos e terá de os resolver da forma mais eficiente, validando a solução (total ou parcial) junto do Servidor. A Figura 2 apresenta uma sugestão de arquitetura de ficheiros para o projeto.

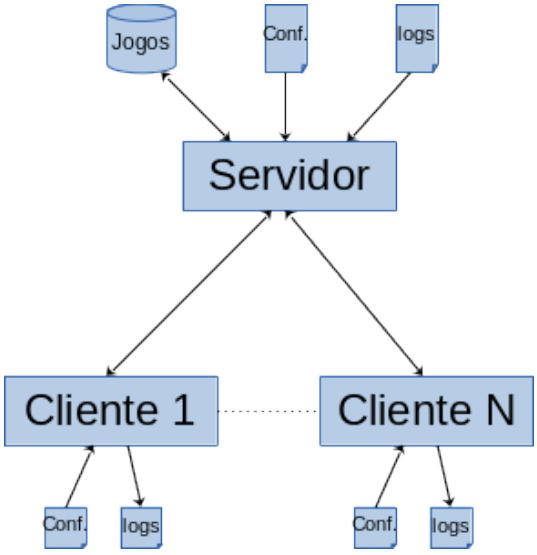

Figura 2: Sugestão de arquitetura de ficheiros para o projeto

O **Servidor** deverá ser lançado tendo como parâmetro o ficheiro de configuração. Os dados no ficheiro de configuração que deverão estar presentes, no mínimo, para o Servidor são os seguintes: ficheiro de jogos e de soluções (podem ser dois ficheiros).

O **Cliente** deverá ser lançado tendo como parâmetro o ficheiro de configuração. Os dados no ficheiro de configuração que deverão estar presentes, no mínimo, para o Cliente são os

seguintes: IP do Servidor, ID do Cliente.

Em ambas as aplicações, as configurações podem e devem incluir mais parâmetros, sendo valorizada a criação de código flexível e parametrizável sem ser necessário recompilar. Caso necessário podem ser indicados outros ficheiros. As aplicações devem ainda ter ficheiros de *logs*, onde todas as ações são guardadas para posterior consulta ou depuração, conforme necessário.

O **Servidor** deverá ter, no mínimo, o seguinte comportamento, no entanto, os grupos podem definir sequências e opções diversificadas. A lista apresentada serve apenas como fluxo simples e devem ter em atenção à construção das mensagens a enviar por cada aplicação.

- Inicializações diversas.
- Espera pela ligação de clientes.
- Para cada cliente:
  - Envia um tabuleiro
  - Recebe uma solução (completa ou parcial)
  - Verifica a solução
  - Devolve o cliente o resultado
- Fecha ligação com o cliente

Para o **Cliente** aplicam-se as mesmas recomendações deixadas para o servidor. A *interface* com o utilizador pode ter uma maior ou menor interatividade.

- Inicializações diversas.
- Liga-se ao servidor.
- Pede e recebe um tabuleiro
- Tenta resolver o tabuleiro
- Envia a solução completa ou parcial ao Servidor
- Recebe o resultado e continua resolução se incompleta
- Fecha ligação com o Servidor

O **Servidor** deverá ter métodos para gerir os pedidos de jogos dos **Clientes** e os pedidos de soluções. Esta gestão deverá ser feita com diferentes tarefas e sincronizada com diferentes políticas, como FIFO, prioridades e/ou barreira. A lógica de jogo pode ser definida de forma livre, principalmente na forma como devem ser feitas competições entre **Clientes**. Seguem alguns exemplos (não completos):

- `enviaJogo(IdCliente, idJogo, Tabuleiro)`
- `recebeSolucao(idCliente, idJogo, Solucao)`
- `respondeSolucao(idCliente, idJogo, "Certo/Errado")`
- ...

Deve ser dada aos utilizadores informação sobre o estado do **Servidor**, como clientes ligados, jogos ativos, jogos resolvidos, recordes de solução de jogos, e quaisquer outras medições consideradas relevantes e interessantes.

```
Estado actual => 3 Clientes
Jogos Resolvidos: 13
Jogos em Resolucao: 3
Jogos nao Resolvidos: 2
Recorde Jogo 1: Jogador 5 Tempo 5:35 Tempo medio 6:12 Maximo 8:23
...
```

O **Cliente** deve adotar um método de tentativa e erro para resolver o jogo. **Cada grupo de trabalho** deve definir os critérios para iniciar as tentativas, o número de tarefas paralelas, a estratégia de preenchimento do tabuleiro e o momento adequado para enviar a solução ao **Servidor**. Deve ser medida a eficiência da opção ou opções testadas. O envio da resposta para o **Cliente** pode ser apenas certo/errado ou incluir mais informações no caso de erro ou incompleta.

Não é importante a rapidez para encontrar uma solução, mas a forma conceptualizada, a sua boa implementação e medição comparativa (por exemplo, enviam para o Servidor 1, 2 ou mais números ou esperam até ter o Tabuleiro completo de números e, assim, ter tempos diferentes de resolução).

A verificação da solução no **Cliente** apenas deve ser feita para cada célula e evitar enviar soluções claramente erradas (por exemplo, 2 números iguais na mesma linha, coluna ou região). Seguem alguns exemplos de chamadas do lado do **Cliente** (não completos):

- `pedeJogo(IdCliente)`
- `enviaSolucao(idCliente, idJogo, Solucao)`
- `recebeResposta(idCliente, idJogo, "Certo/Errado")`
- ...

A interação com o utilizador pode ser mais complexa e a visualização deve dar aos utilizadores informação sobre o andamento da resolução do jogo e interações com o servidor.

```
ID jogo a decorrer : 25
    Hora de inicio > 10:02
   Tempo decorrido > 1:12
 Numero de Tarefas > 10
5 0 0 | 0 0 3 | 0 7 0
0 0 2 | 1 0 0 | 5 0 0
1 0 0 | 0 9 0 | 0 0 6
---------------------
0 5 0 | 0 0 6 | 0 4 2
0 6 0 | 0 0 0 | 0 8 0
3 2 0 | 4 0 0 | 0 9 0
---------------------
8 0 0 | 0 4 0 | 0 0 5
0 0 6 | 0 0 8 | 2 0 0
0 1 0 | 7 0 0 | 0 0 8
```

Em ambas as aplicações, recomenda-se o desenvolvimento de código modular e iniciar com uma abordagem simples e ir incrementando o número de funcionalidades para funções mais complexas.

Exemplo dos comandos para lançamento das aplicações:

```
$ servidorSudoku server.conf
$ clienteSudoku cliente1.conf
$ clienteSudoku cliente2.conf
```

4 Estrutura de Dados

As duas aplicações terão um conjunto de estruturas de dados para guardar diversas informações. Para os ficheiros de configuração fica a seguinte sugestão:

```
PARAMETRO1:dado1
PARAMETRO2:dado2
...
```

onde 'PARAMETRO' significa a designação do parâmetro e 'dado' é o seu valor. Uma sugestão para o ficheiro de jogos pode ser vista na caixa abaixo, mas podem ser usados outros formatos como *json*, *xml* ou outros formatos de texto.

```
IdJogo, Jogo, Solucao
1, 530070000600195000098000060800060003400803001700020006060000280000419005000080079,
534678912672195348198342567859761423426853791713924856961537284287419635345286179
2, 500003070002100500100090006050006042060000080320400090800040005006008200010700008,
594623871682174539173895426759386142468259783327417694835942715246598273918732568
.........
                                Visualizacao inicial do
                                  tabuleiro do jogo 2
                                5 0 0 | 0 0 3 | 0 7 0
                                0 0 2 | 1 0 0 | 5 0 0
                                1 0 0 | 0 9 0 | 0 0 6
                                ---------------------
                                0 5 0 | 0 0 6 | 0 4 2
                                0 6 0 | 0 0 0 | 0 8 0
                                3 2 0 | 4 0 0 | 0 9 0
                                ---------------------
                                8 0 0 | 0 4 0 | 0 0 5
                                0 0 6 | 0 0 8 | 2 0 0
                                0 1 0 | 7 0 0 | 0 0 8
```

Os dados sobre os acontecimentos das aplicações poderão ter o seguinte formato (para o **Servidor** e **Cliente** deverão ser mais detalhados):

```
IdUtilizador hora Acontecimento Descricao
25 10:12:35 3 "Solucao enviada"
...
```

onde o 'Acontecimento'[2] pode ser, por exemplo, um dos seguintes: pedido de jogo, envio de solução, solução completa, solução errada, entre outros.

5 Entrega e Avaliação do Projeto

O projeto será desenvolvido em grupos de 3 pessoas (exceção para os alunos trabalhadores-estudantes que, se preferirem, podem fazer individualmente). Os grupos podem ser atualizados até à segunda fase de entrega e, após esta data, não pode haver mais alterações. O projeto será dividido em três fases (obrigatórias) e as datas limite de cada fase serão anunciadas nas aulas e/ou na página da cadeira.

5.1 1ª Fase

A primeira fase deverá conter as bibliotecas para a gestão da informação nos ficheiros de texto. As aplicações **Servidor** e **Cliente** já devem carregar os parâmetros para a sua configuração e ainda exportar para um (ou vários) ficheiro(s) o registo dos eventos que ocorrem nas aplicações.

[2] Sugestão: codificar cada acontecimento como um inteiro.

Deve ainda já ter a função de verificação de solução (recebe uma solução e verifica se está correta, registando o resultado: correta, errada em x números).

A avaliação recairá sobre o código implementado e as estruturas de dados escolhidas para a configuração e para os registos em ficheiros. Na aula seguinte à data definida para esta fase os grupos deverão demonstrar o funcionamento do código. A avaliação desta fase tem um peso de **10%** na nota final da parte prática.

5.2 2ª Fase

A segunda fase envolve a implementação das bibliotecas para a comunicação entre o **Cliente** e o **Servidor**; e, a *interface* com o utilizador, onde será apresentado o estado corrente do jogo. Deverá ainda ser entregue um documento simples que descreva e fundamente (de forma simples) as opções seguidas, principalmente para o formato das mensagens e protocolo de comunicação entre o cliente e o servidor.

O documento deverá ainda conter uma descrição das funcionalidades a implementar, bem como a forma como se pretende resolver a questão da sincronização. O documento deverá ser conciso e explicar brevemente cada funcionalidade. A entrega desta fase será feita na plataforma Moodle, na área da disciplina.

A avaliação desta fase recairá sobre o código implementado, nomeadamente as estruturas do protocolo de comunicação e a *interface* com o utilizador já definido. Será ainda parte da avaliação o nível de maturidade da solução proposta para a sincronização das tarefas das aplicações. Na aula seguinte à data definida para esta fase os grupos deverão demonstrar o funcionamento do código e descrever rapidamente a sua visão do problema. A avaliação desta fase tem um peso de **20%** na nota final da parte prática.

5.3 3ª Fase

Por fim, a terceira fase, deverá agregar todas as bibliotecas anteriores e ainda conter o código com os mecanismos e políticas de sincronização. Nesta fase deverá ser ainda entregue a fundamentação da solução escolhida e as conclusões gerais do trabalho sob a forma de um relatório completo do projeto.

A avaliação desta fase tem um peso de **70%** na nota final da parte prática, divididos (não igualmente) entre o código desenvolvido, a apresentação/defesa do projeto e o relatório.

O relatório deve ser curto e ter, pelo menos uma página para as opções finais de formatos de texto e comunicação. Outra página deve explicar a lógica de jogo definida pelo grupo e como foram feitas as opções de concorrência (tarefas) e sincronização. O relatório ainda incluir uma resposta/reflexão às seguintes questões:

- Qual a forma de resolução de tabuleiros foi mais eficiente e porquê?
- A solução apresentada apresenta uma maior preocupação no uso justo/equilibrado dos recursos ou na eficiência geral do sistema?
- Descrevam, pelo menos, duas limitações da solução apresentada.

O relatório deve ser entregue via Moodle (que deve também conter em anexo a listagem do código-fonte, mas formatado de forma a reduzir ao máximo o número de folhas sem perder legibilidade). Os relatórios devem ainda conter a indicação clara na capa do(s) nome(s) do(s) aluno(s), número(s) mecanográfico(s) e curso(s), além dos dados do projeto.

As apresentações/discussões serão na semana após a da entrega do trabalho. O código e o relatório serão lidos pelo docente, e defendidos pelos alunos numa apresentação/discussão do projeto.

A apresentação do trabalho é por grupo, mas **a defesa e a nota final são individuais**. A não comparência de um elemento em qualquer uma das defesas implica nota "zero"nessa fase do projeto.

Terão de entregar um ficheiro compactado (ZIP ou RAR) contendo todo o código necessário para compilar e executar o programa. Junto com o código-fonte deve ser incluída uma *makefile* que compile corretamente as aplicações desenvolvidas no ambiente fornecido para as aulas ou em outro por acordo com o docente. Deve ainda ser acrescentado um ficheiro *README* de ajuda à compilação e execução do sistema para que um utilizador possa ler, compilar e visualizar o trabalho sem a ajuda dos seus autores.

6 Considerações Finais

A implementação do sistema poderá ser executada utilizando qualquer ambiente de desenvolvimento da linguagem C disponível. Recomenda-se, no entanto, a utilização do ambiente usado nos laboratórios.

Algumas considerações gerais:

- É preferível apresentar um projeto funcional, ainda que com funcionalidades incompletas, em vez de um projeto que implementa todas as funcionalidades, mas não funciona adequadamente;
- O relatório serve para descrever o que foi feito e, principalmente, fundamentar as opções tomadas. O relatório deve ainda conter os testes às aplicações e dados sobre as simulações executadas, não deixando de ser feita uma análise aos resultados obtidos;
- Evitem relatórios extensos, com erros ortográficos, com capas coloridas e/ou outros adereços complemente desnecessários, repetir partes do enunciado no relatório, etc. Lembrem-se que o que conta não é o número de páginas, mas sim a qualidade do conteúdo.