Relatório ...

Departamento de Engenharia Informática e de Sistemas (DEIS)

Bruno Alexandre Ferreira Pinto Teixeira (a2019100036)

...

Conteúdo

Lista de figuras

1 Introdução

Neste relatório apresentamos a arquitetura assim como o funcionamento do trabalho prático.
Durante as várias metas existiram algumas mudanças, no entanto todas as mudanças são descritas e apresentadas de forma a justificar o porquê dessas mudanças terem sido feitas.

2 Desenvolvimento

2.1 Arquitetura

Para este trabalho usamos uma arquitetura cliente servidor, em que o servidor é designado por árbitro e o cliente é uma interface para um jogador.
Cada jogador usa um cliente(shell), cliente este que interage com um servidor, ou seja, existem múltiplos clientes mas só um servidor.
O servidor é responsável pela criação dos jogos, pelo redirecionamento entre jogos e clientes e pela manutenção de toda a estrutura do programa.

Na figura seguinte mostramos um exemplo da arquitetura, com todos os pormenores, em que existem dois clientes, um arbitro e um jogo para cada cliente.

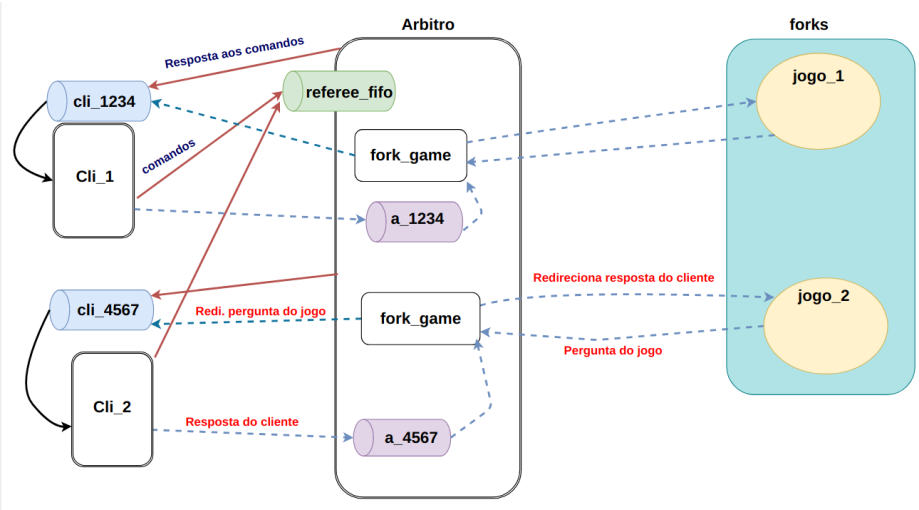

Figure 1: exemplo de caption de uma imagem

2.1.1 Estruturas

Consoante a realização do trabalho, decidimos fazer algumas alterações a algumas estruturas assim como acrescentar outras.

À estrutura **Games** foi-lhe retirado um campo, **pergunta**, pois não era utilizado.
A estrutura **Player** manteve-se, apenas foi adicionado o **player_trinco** sendo este um **mutex** para trabalhar com os respetivos dados e continua a ser usada

para identificar um jogador num array dinâmico de estruturas.

Foi tambem criada uma nova estrutura, **Mainframe**, que é usada para a passagem de informação para as threads. Esta estrutura contém tanto o array de jogadores como o array de jogos existentes, o tempo de espera (wait time) e o tempo do campeonato,assim como algumas flags.
A estrutura **User**, presente tanto no **referee** como no **client**, manteve-se igual à meta anterior.

2.1.2 Threads

Foram desenvolvidas três funções thread além da main thread.

Temos a **thread_register**, função usada para fazer a leitura do **pipe do arbitro** e proceder consoante a leitura, quer adicionar um novo jogador quer ler um comando do cliente. Esta thread é responsável também por lançar a **thread_timer** quando tem 2 jogadores já inscritos.

A função **thread_timer** cronometra o tempo de espera, antes de lançar um campeonato, e antes de cronometrar o tempo de duração do mesmo, escolhe de forma aleatória o jogo.

Esta thread lança uma thread chamada de **fork_game** para cada **jogador**.
A **thread_timer** notifica os jogadores sobre o inicio, o fim e o respetivo score do campeonato.

A thread **fork_game** é responsavel por realizar o **fork dos jogos** bem como executar o jogo(usando o **execl**). Nesta thread fazemos todo o processo de **reencaminhamento** entre o **cliente** e o seu respetivo **jogo**.

Após o fim do tempo do campeonato e do encerramento de todas as threads é recomeçado outro campeonato.

2.2 Funcionamento

O arbitro é iniciado e fica à espera de pelo menos dois jogadores para conseguir iniciar um campeonato.
Quando dois clientes são iniciados, é feita uma verificação caso o nome dos mesmos seja igual uma vez que não faz sentido terem ambos o mesmo nome.
Depois de todas as validações feitas, esses clientes são colocados num campeonato, campeonato este que é totalmente administrado pelo árbitro.
O campeonato consiste num jogo simples em que ganha quem tiver melhor pontuação no final do tempo.
Dentro deste processo existem várias comunicações internas que precisam de ser esclarecidas para que seja mais fácil entener o funcionamento do trabalho.

O cliente gere um **FIFO**, **FIFO** este que vai receber algumas informações sempre que se quiser comunicar com o cliente.
O arbitro também tem um **FIFO** próprio que é usado para receber comandos vindos do cliente, comandos pessoais, não relativos aos jogos.

Quando um jogador é inscrito, é criado um **FIFO** designado por (a_PID_DO_-JOGADOR) que é usado unica e exclusivamente para a comunicação entre cliente e jogo, sendo este processo de comunicação, um processo indireto.
Quando o campeonato começa, o arbitro lança algumas threads (explicadas anteriormente), sendo uma delas responsável pela comunicação entre jogo e cliente. Neste processo de comunicação, o jogo envia uma mensagem para a thread **fork_game** e a thread reencaminha a mensagem para o **FIFO** do cliente. Quando o cliente responde, é enviada uma mensagem para o **FIFO** (a_PID_DO_JOGADOR), a thread (**fork_game**) lê desse **FIFO** e redireciona a mensagem de volta para o jogo.

A inscrição e a saida de um jogador num campeonato só pode ser feita durante o tempo de espera entre campeonatos, ou seja, enquanto o campeonato não está a decorrer o jogador pode sair de maneira correta.

Quando o campeonato termina, os jogos são apagados enviando um sinal para cada jogo, as threads são terminadas e é recolhida a informação da pontuação de todos os jogadores, mostrando a todos os jogadores a sua pontuação e a melhor daquele campeonato.Quando o arbitro termina, todos os **FIFOS** são fechados e

eliminados.

3 Conclusão

Após a realização do trabalho, não foi encontrado nenhum **bug** que pusesse em causa a execução e a comunicação do **arbitro** e do **cliente**.
Foram testados vários cenários para que não houvesse problemas antes, durante e após a execução do **arbitro** e do **cliente**.

Uma vez que é usada memória dinâmica em vários pontos do trabalho, foi dada uma atenção extra para que não haja um **memory leak**.
No recomeço do campeonato verificamos tambem (com ajuda do **ps aux**) se todos os processos criados no campeonato anterior foram eliminados de maneira correta.

No final da execução do **arbitro** foi visto que todas as threads terminam de forma ordenada e correta e que todos os **FIFOS** são eliminados não deixando nenhum tipo de lixo no ambiente do utilizador.