Trabalho de Sistemas Operativos 2 (Windows API)

Eduardo Jesus, nº2023112187

ISEC, Maio de 2025

1 Introdução

Este texto descreve o desenho e implementação de uma solução para o problema descrito no ficheiro *relatorio.pdf*. Nesta solução, não considerámos o programa *painel*, tratando apenas dos três outros programas: *bot*, *jogoui* e *arbitro*. Aos últimos dois, vamos passar a referir de **cliente** e **servidor**, respetivamente, para o resto do relatório.

2 Design

O **servidor** é uma implementação de um servidor multithreaded para Windows, que utiliza *named pipes* para comunicação com clientes e **eventos**, **semáforos** e **memória partilhada** para a sincronização do estado do jogo. Uma das threads deste servidor gera aleatoriamente a próxima letra a entrar num *buffer* partilhado, do qual o processo cliente lê e atualiza o seu próprio estado.

O **servidor** utiliza três *threads*:

- **cli**, que é a interface de linha de comando
- **listen**, que gere mensagens ao servidor através de *named pipes*
- **game**, que trata de atualizar o *buffer* partilhado

O **cliente** inicialmente estabelece contacto com o **servidor** por meio de um pedido de *login*. Se este for aceite, então o cliente inicia a execução das suas *threads*:

- **listenUpdate**, que escreve o array de letras no *stdout*, apenas quando este sofrer mudanças. O tempo entre mudanças de estado é variável com o tempo

- **listenPipe**, que gere um *named pipe* de forma a receber notificações do servidor
- **cli**, que recebe input de consola

Adicionalmente, a funcionalidade **bot** é uma instância do programa **cliente** que em vez de um interface com a linha de comando utiliza um **dicionário** de palavras, também ele em memória partilhada com o **servidor**, para ir aleatoriamente selecionando palavras em intervalos de tempo fixos.

3 Estrutura de Dados - Gestão de Jogadores

Esta secção descreve a implementação de um sistema de gerenciamento de jogadores para o jogo, do lado do servidor.

3.1 Componentes Principais

3.1.1 Gerador de Identificadores de Jogadores

A estrutura `Player_ID_Generator` é responsável pela geração de identificadores únicos para os jogadores:

- **Propósito**: Gera IDs únicos utilizando um contador incremental
- **Funcionamento**: Inicia em 0 e incrementa para cada novo jogador registado
- **Implementação**: Utiliza um inteiro de 32 bits como estado interno

3.1.2 Estrutura do Jogador

A estrutura `Player` encapsula os dados essenciais de cada jogador:

- **ID único**: Identificador numérico gerado automaticamente
- **Pontuação**: Pontuação atual do jogador no jogo
- **Nome do pipe**: Caminho para o *named pipe* de comunicação
- **Handle de evento**: Identificador para notificações de atualização

3.2 Classe GameData

A classe `GameData` constitui o núcleo do sistema de gerenciamento, mantendo três estruturas de dados distintas para otimizar diferentes tipos de consultas:

3.2.1 Estruturas de Dados Internas

1. **id_map**: `std::map<int32_t, std::wstring>`
   - Mapeia ID do jogador para o nome do jogador
   - Permite consultas rápidas por identificador
2. **score_map**: `std::multimap<int32_t, std::wstring>`
   - Mapeia pontuação para nome do jogador
   - Ordenado por pontuação decrescente
   - Implementa automaticamente o sistema de classificação (*leaderboard*)
   - Utiliza `multimap` para permitir pontuações iguais
3. **name_map**: `std::map<std::wstring, Player>`
   - Mapeia nome do jogador para o objeto `Player` completo
   - Armazena todos os dados do jogador

3.2.2 Funcionalidades Implementadas

Registo de Jogadores O método `insert` executa as seguintes verificações e operações:

- Verifica se o nome do jogador já existe no sistema
- Confirma se existem vagas disponíveis no servidor (`MAX_PLAYERS`)
- Valida se o cliente criou o evento e *pipe* necessários para comunicação
- Adiciona o jogador simultaneamente nos três mapas para manter consistência

Remoção de Jogadores O sistema oferece dois métodos de remoção (`remove`):

- Remoção por nome do jogador
- Remoção por ID do jogador
- Ambos os métodos removem entradas de todos os três mapas
- Inclui tratamento para casos de inconsistência entre mapas

Atualização de Pontuação O método `update` permite modificar a pontuação dos jogadores:

- Suporte para atualização por nome ou ID
- Pontuação mínima estabelecida em 0 (não permite valores negativos)
- Atualiza o `score_map` removendo a entrada antiga e inserindo a nova

Sistema de Comunicação O sistema implementa vários métodos de comunicação:

- **broadcast**: Envia pacotes para todos os clientes conectados
- **send**: Envia pacote para um jogador específico
- **updateAllClients**: Notifica todos os clientes para atualizarem o seu estado

3.3 Características de Design

3.3.1 Redundância Intencional

A utilização de três mapas distintos cria redundância proposital para otimizar diferentes operações:

- Consultas rápidas por ID
- Ordenação automática por pontuação
- Acesso completo aos dados do jogador

3.4 Implementação do Servidor

3.4.1 Thread CLI

A thread CLI implementa a interface de linha de comando do servidor, permitindo comandos administrativos. Esta thread executa em paralelo com as outras threads do servidor e permite gestão em tempo real.

Os comandos administrativos disponíveis incluem:

- `listar` - Lista todos os jogadores conectados e suas pontuações
- `excluir <nome>` - Remove um jogador específico do jogo
- `acelerar` - Diminui o intervalo entre letras (mínimo 1000ms)

- `travar` - Aumenta o intervalo entre letras
- `bot <nome>` - Lança uma instância de bot com o nome especificado
- `encerrar` - Encerra o servidor graciosamente

3.4.2 Thread Listen

A thread listen é responsável por aceitar novas conexões de clientes:

3.4.3 Thread Game

A thread game gere o estado principal do jogo, incluindo a geração de novas letras:

3.5 Implementação do Cliente

3.5.1 Thread ListenUpdate

Esta thread monitoriza mudanças no buffer partilhado e atualiza o display:

3.5.2 Thread CLI do Cliente

Processa input do utilizador e envia comandos para o servidor.

3.6 Implementação do Cliente

Baseado na arquitetura do servidor, o cliente implementa três threads principais que interagem com o servidor e memória partilhada:

3.6.1 Thread ListenUpdate

Esta thread monitoriza continuamente a memória partilhada para mudanças no estado do jogo. Acede ao semáforo partilhado para ler o array de letras atual e atualiza o display local apenas quando há alterações. Esta thread garante que o cliente mantém uma vista consistente e atualizada do estado do jogo sem polling excessivo.

3.6.2 Thread ListenPipe

Responsável por receber notificações ativas do servidor através de um mecanismo de callback ou eventos. Esta thread processa mensagens de broadcast do servidor sobre:

- Novos jogadores que entraram no jogo

- Jogadores que saíram do jogo
- Palavras adivinhadas por outros jogadores
- Atualizações de pontuações
- Comandos administrativos (como expulsão)

3.6.3 Thread CLI do Cliente

Processa input do utilizador local e coordena a comunicação com o servidor:

- Lê comandos e palavras do utilizador via stdin
- Valida input localmente antes de enviar ao servidor
- Estabelece conexões ao named pipe do servidor para cada pedido
- Envia pacotes estruturados (LOGIN, LOGOUT, GUESS, SCORE)
- Processa respostas do servidor e atualiza estado local

3.7 Implementação do Bot

O bot representa uma implementação especializada do cliente que opera automaticamente. Em vez de interface humana, utiliza o dicionário em memória partilhada para:

- Ler continuamente o estado atual das letras disponíveis
- Selecionar aleatoriamente entre palavras existentes no dicionário
- Submeter tentativas em intervalos regulares fixos
- Funcionar como um cliente normal em termos de comunicação com o servidor

O bot é lançado como processo separado através do comando administrativo `bot <nome>`, utilizando `CreateProcess()` para iniciar uma nova instância do programa cliente com flag especial `-bot`. Desta forma, múltiplos bots podem coexistir no mesmo jogo, cada um com o seu próprio nome e pontuação.

3.8 Sincronização e Comunicação

3.8.1 Named Pipes

A comunicação entre cliente e servidor utiliza named pipes do Windows com as seguintes características:

- **LOGIN**: Cliente envia nome de utilizador para entrar no jogo
- **GUESS**: Cliente envia palavra formada com letras disponíveis
- **LOGOUT**: Cliente solicita desconexão do jogo
- **SCORE**: Cliente pede pontuação atual

O protocolo utiliza estruturas `Packet` para comunicação padronizada, com cada pacote contendo código de operação, ID de jogador e buffer de dados. O servidor responde com estruturas apropriadas dependendo do tipo de pedido.

3.8.2 Objetos de Sincronização

O sistema utiliza múltiplos objetos de sincronização do Windows para coordenação thread-safe:

- **Semáforo** (`semaphore_handle`): Controla acesso à memória partilhada com capacidade MAX_PLAYERS + 2, permitindo acesso simultâneo controlado
- **Mutex** (`data_handle`): Protege a estrutura GameData de acessos concorrentes durante operações de jogadores
- **Evento Manual** (`clear_handle`): Sinaliza quando o array de letras deve ser limpo após palavra correta
- **Evento Manual** (`quit_handle`): Coordena encerramento gracioso de todas as threads

3.8.3 Memória Partilhada

O sistema utiliza dois mapeamentos de ficheiro principais:

- **GameState**: Contém o array de letras atual e metadados do jogo, acessível por todos os clientes
- **Dictionary**: Contém o dicionário de palavras válidas partilhado entre servidor e bots

Ambos os mapeamentos utilizam alinhamento baseado na granularidade de alocação do sistema para otimização de performance.