isec
Politécnico de Coimbra

Licenciatura em Engenharia Informática

Sistemas Operativos 2

RELATÓRIO META2

TRABALHO PRATICO

BOLSA DE VALORES

Grupo:

Nome	Nº Aluno
Lucas Erardo Alves da Graça Monteiro	2022153271
Afonso Henrique Pena Pedroso da Silva	2021133861

Coimbra – 18 de Maio de 2024

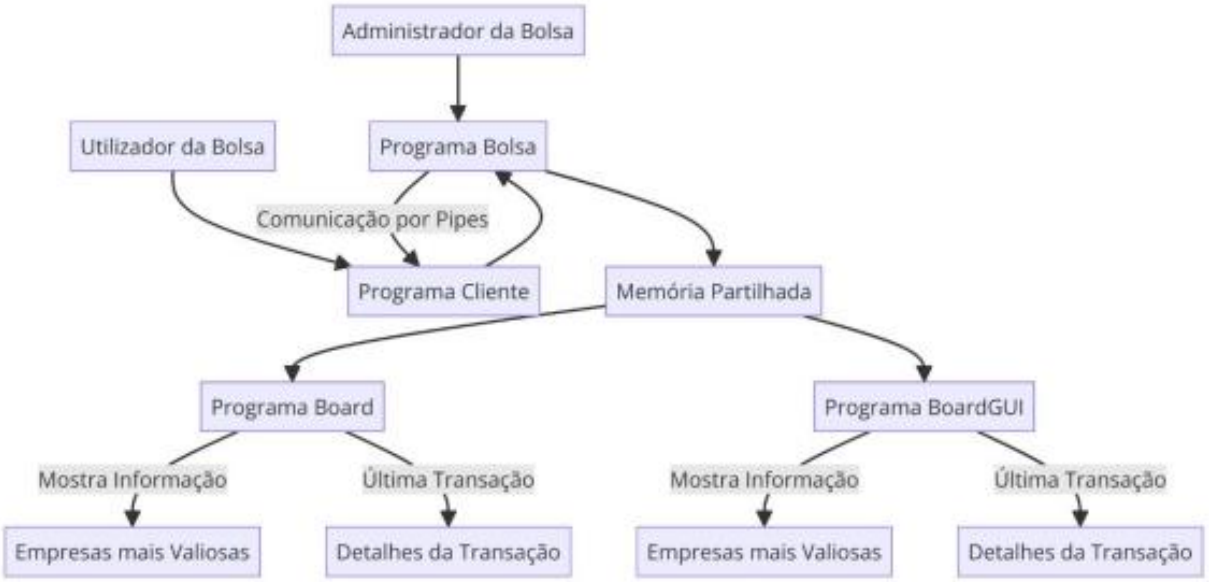
Índice

Introdução

O desenvolvimento de sistemas distribuídos e a comunicação entre processos são tópicos fundamentais em Sistemas Operativos II. Neste contexto, propõe-se a criação de uma bolsa de valores online simplificada, que servirá como um estudo prático para a aplicação de diversos conceitos e técnicas aprendidos. Este projeto consiste na implementação de vários programas que, juntos, formam uma bolsa de valores online. A funcionalidade esperada não é de elevado realismo, mas sim suficiente para explorar e aplicar os conhecimentos adquiridos durante o decorrer das aulas.

O principal objetivo deste trabalho é criar um sistema de bolsa de valores que simula, de forma simplificada, o funcionamento de uma bolsa real. Através deste sistema, os utilizadores podem comprar e vender ações de empresas, cujas informações e preços são geridos centralizadamente. O sistema também deve permitir o registo de novas empresas e a visualização de informações detalhadas sobre as transações e os participantes do mercado.

Arquitetura:



Estruturas de Dados

Constantes

As constantes definidas são usadas para especificar limites e nomes de recursos:

MAX	Numero mínimo de clientes permitidos
MAX_EMPRESAS	Número máximo de empresas.
TAM_CLIENTES	Número máximo de clientes.
NEMPRESAS_DISPLAY	Número de empresas a serem exibidas
TAM_STR	Tamanho das strings.
NOME_SM	Nome da memória compartilhada.
PIPE_NAME	Nome do pipe para comunicação.
SEM_O	Nome do semáforo.
NOME_MUTEX_IN e NOME_MUTEX_OUT	Nomes dos mutexes para controle de acesso.

Estruturas

Empresa

Representa uma empresa na bolsa de valores:

```
typedef struct {
    TCHAR nomeEmp[TAM_STR];
    DWORD valorAcao;
    DWORD numAcoes;
    TCHAR UltimaTransicao[TAM_STR];
} Empresa;
```

nomeEmp	Nome da empresa.
Valoração	Valor atual da ação.
numAcoes	Número de ações disponíveis para compra/venda.
UltimaTransicao	String da ultima tramsição

CarteiraAcoes

Representa a carteira de ações de um cliente:

```
typedef struct {
    Empresa emp[MAX_EMPRESAS];
    DWORD numEmpresasCarteira;
} CarteiraAcoes;
```

Emp	Array de empresas que o cliente possui na sua carteira.
numEmpresasCarteira	Numero de empresas que a carteira tem

Cliente

Representa um cliente na bolsa de valores:

```
typedef struct {
    TCHAR nome[TAM_STR];
    FLOAT saldo;
    TCHAR senha[TAM_STR];
    DWORD ligado
    DWORD continua;
    HANDLE hpipe;
    CarteiraAcoes carteira;
    TCHAR cmd[100
} Cliente;
```

Nome	Nome do cliente.
Saldo	Saldo disponível para transações.
Senha	Senha do cliente.
Ligado	Flag indicando se o cliente está ativo.
Continua	Flag para controle do processo.
hpipe	Handl para comunicação
Carteira	Carteira de ações do cliente.
Cmd	Armazenam comandos e dados para comunicação.

TDATA (Bolsa)

```
typedef struct {
    BOOL continua;
    HANDLE hMutex, hMutexC, hSem, hEv;
    Empresa empresas[MAX_EMPRESAS];
    DWORD* numEmpresas;
    DWORD pauseComando;
    DWORD instantePause;
    Cliente* clientes;
    DWORD idLoc;
    DWORD* nCLi;
    DWORD nCliente;
    HANDLE hPipes[TAM_STR];
} TDATA;
```

continua	Flag para indicar se o servidor continua executando
instantePause	Timestamp indicando o instante em que a pausa começou
hMutex, hMutexC, hSem, hEv	Handles para sincronização
empresas	Array de empresas
numEmpresas	Ponteiro para o número de empresas
pauseComando	Tempo de pausa entre comandos
idLoc	ID do cliente local
clientes	Array de clientes
nCLi	Ponteiro para o número de clientes

Em resumo:

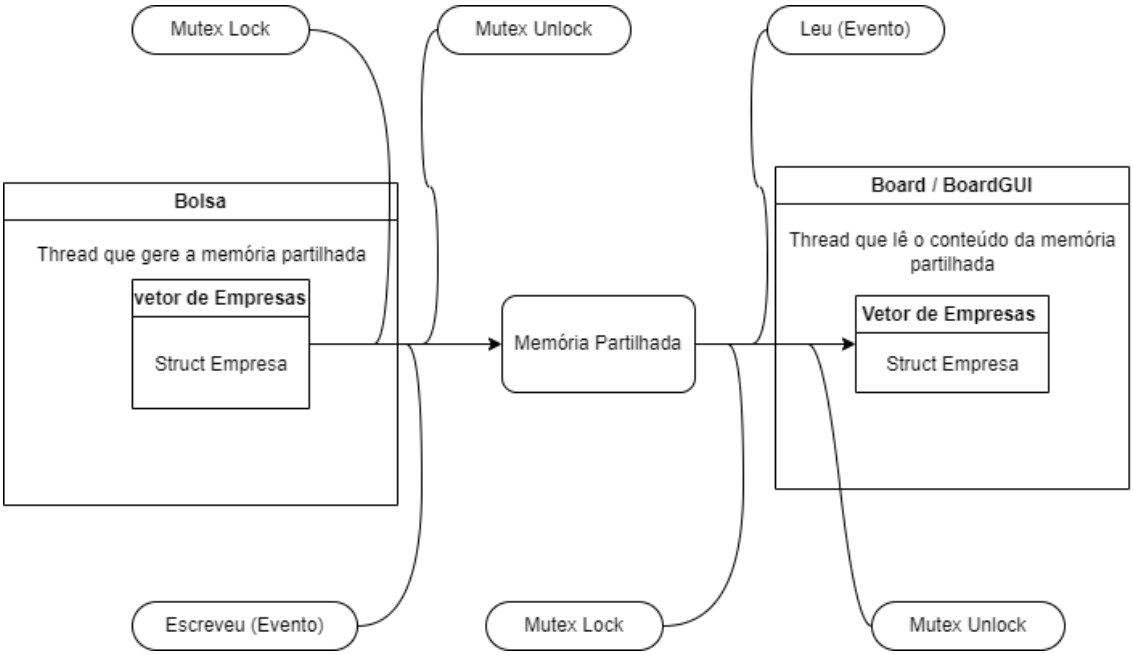
- A estrutura **Empresa** representa uma empresa na bolsa de valores. Cada empresa tem um nome, uma valoração atual por ação e um número de ações disponíveis para transação.

- A estrutura **CarteiraAcoes** representa a carteira de ações de um cliente. Esta estrutura contém um array de empresas que o cliente possui.

- A estrutura **Cliente** representa um cliente da bolsa de valores. Cada cliente tem informações pessoais, saldo disponível, credenciais de acesso, estado de conexão e uma carteira de ações.

A estrutura **TDATA (para a bolsa)** é semelhante à TDATA para clientes, mas inclui elementos adicionais específicos para a gestão centralizada das operações da bolsa.

Comunicação Bolsa – Board / BoardGUI



O mecanismo de comunicação entre dois componentes do sistema, "Bolsa" e "Board", é feito através do uso de memória partilhada e sincronização de eventos.

Estrutura e Sincronização

1. Mutex Locks e Unlocks:

- O mutex é usado para garantir que apenas um thread tenha acesso à seção crítica do código que manipula a memória compartilhada por vez. Isso previne condições de corrida.

2. Eventos (Events):

- Eventos são usados para sincronizar a comunicação entre a "Bolsa" e o "Board".
- SetEvent(td->hEv); sinaliza que um evento ocorreu, permitindo que o outro componente continue seu processamento.
- WaitForSingleObject(td->hEv, INFINITE); faz com que o thread espere até que o evento seja sinalizado.
- ResetEvent(td->hEv); reseta o estado do evento para não sinalizado, preparando-o para o próximo ciclo de comunicação.

Comunicação através de Memória Compartilhada

1. **Abertura e Mapeamento da Memória Compartilhada:**

- hMap = OpenFileMapping(FILE_MAP_WRITE, FALSE, _T("Empresa")); abre um objeto de memória compartilhada chamado "Empresa".

- pEmpresas = (Empresa*)MapViewOfFile(hMap, FILE_MAP_READ | FILE_MAP_WRITE, 0, 0, sizeof(Empresa)); mapeia essa memória no espaço de endereçamento do processo.

2. **Leitura e Escrita na Memória Compartilhada:**

- O mutex é usado para proteger o acesso à memória compartilhada.
- Se *td->numEmpresas == 0, isso indica que a lista de empresas não foi inicializada, então a função geraEmp(emp) é chamada para inicializar uma lista de empresas "vazia" para prevenir um deadLock e aparecimento de lixo no display do Board, e CopyMemory(pEmpresas, emp, sizeof(Empresa) * NEMPRESAS_DISPLAY); copia essas empresas para a memória compartilhada.
- Se a lista de empresas já estiver inicializada, CopyMemory(pEmpresas, &td->empresas, sizeof(Empresa) * NEMPRESAS_DISPLAY); copia os dados das empresas existentes para a memória compartilhada.

3. **Desmapeamento da Memória Compartilhada:**

- `UnmapViewOfFile(pEmpresas);` desmapeia a memória compartilhada.
- CloseHandle(hMap); fecha o handle do objeto de memória compartilhada.

Fluxo da Função comunicacaoBoard

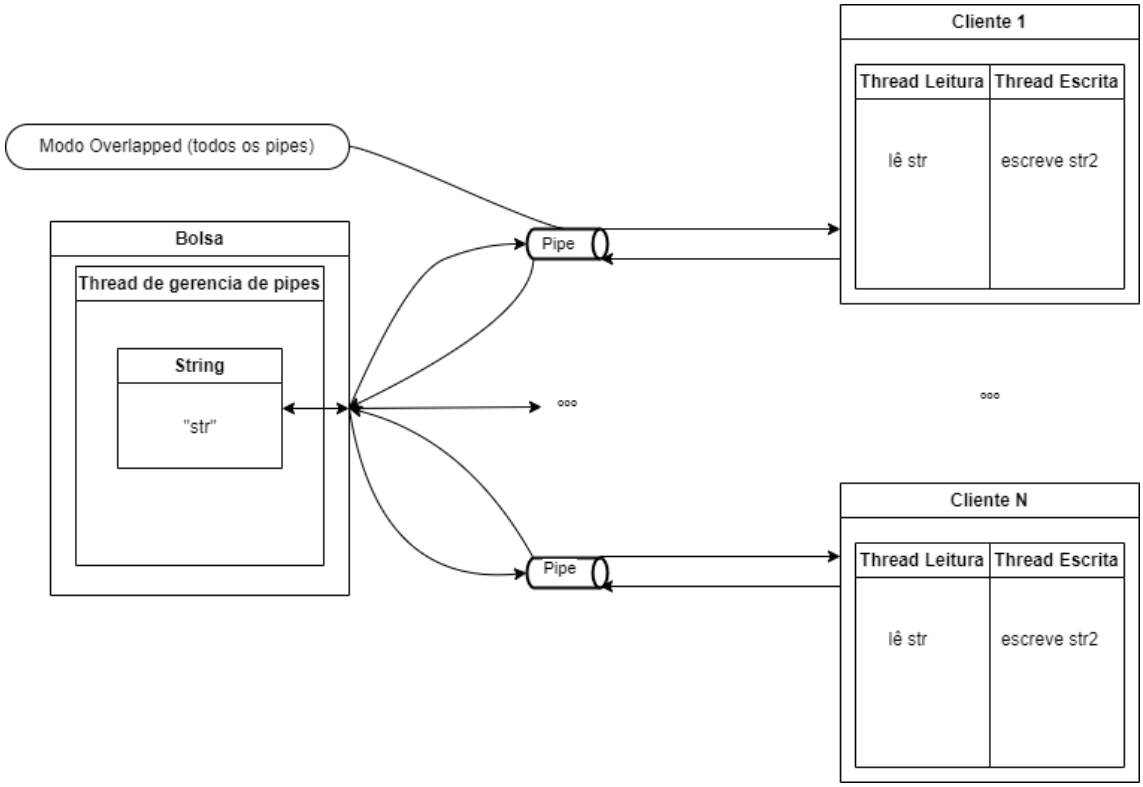
1. **Inicialização:**

- O código verifica se o array de ações está inicializado (isStockArrayInitialized) e o ordena (sortStocks).
- Se houver um comando de pausa (td->pauseComando), o thread pausa por esse período.

2. Sincronização e Atualização:

- Um evento é sinalizado (SetEvent) e aguardado (WaitForSingleObject).
- A memória compartilhada é aberta e mapeada.
- O acesso à memória é protegido por um mutex.
- Dependendo do estado de inicialização das empresas, a memória compartilhada é preenchida com novas empresas ou atualizada com empresas existentes.
- O mutex é liberado, e a memória compartilhada é desmapeada.
- O evento é resetado, e o thread dorme por um breve período antes de repetir o ciclo.

Comunicação Bolsa – Clientes



Este sistema envolve um servidor que gerencia múltiplos clientes através de pipes nomeados, utilizando operações de I/O sobrepostas.

Comunicação e Sincronização

1. **Pipes Nomeados:**

   - Pipes nomeados são usados para comunicação entre o servidor e os clientes. O servidor cria um pipe nomeado (CreateNamedPipe) e espera os clientes se conectarem (ConnectNamedPipe). - Cada cliente se conecta ao pipe nomeado do servidor (CreateFile) e se comunica através dele.

2. **I/O Sobreposto (Overlapped I/O):**

   - Tanto o servidor quanto os clientes utilizam I/O overlapped para operações de leitura e escrita não bloqueantes nos pipes. Isso é indicado pelo uso da estrutura `OVERLAPPED` e a flag FILE_FLAG_OVERLAPPED.

   - O I/O overlapped permite que o programa realize outras tarefas enquanto espera a conclusão das operações de I/O, melhorando a eficiência e a capacidade de resposta.

3. **Threads:**

   - O servidor cria uma thread separada (trataCli) para cada cliente conectado, para gerenciar a comunicação com aquele cliente.

   - Os clientes também criam uma thread separada (leMsg) para lidar com a leitura das respostas do servidor.

4. **Objetos de Sincronização:**

   - Mutex (CreateMutex): Usado para sincronizar o acesso a recursos compartilhados, como a lista de clientes conectados.

   - Semáforo (CreateSemaphore): Limita o número de conexões de clientes simultâneas ao servidor.

   - Eventos (CreateEvent): Usados para sinalizar a conclusão das operações de I/O overlapped.

Fluxo do Programa

1. **Inicialização do Servidor:**

   - O servidor inicializa os objetos de sincronização (mutex e semáforo).

   - Entra em um loop onde cria um novo pipe nomeado e espera por conexões de clientes.

2. **Conexão do Cliente:**

   - Quando um cliente se conecta, o servidor cria uma nova thread (trataCli) para gerenciar a comunicação com este cliente.

   - O cliente também cria uma thread (leMsg) para lidar com a leitura das respostas do servidor.

3. **Comunicação Cliente-Servidor:**

   - O cliente envia comandos para o servidor através do pipe nomeado que são strings.

   - O servidor lê o comando usando ReadFile com I/O overlapped. Se a operação de leitura não for concluída imediatamente, ele espera usando WaitForSingleObject e GetOverlappedResult.

- O servidor processa o comando e prepara uma resposta.
- O servidor envia a resposta de volta para o cliente usando WriteFile com I/O sobreposto.

4. **Processamento de Comandos:**

- Comandos incluem login, exit, listc, balance, buy, sell e listcarteira.
- Para comandos que requerem acesso a recursos compartilhados, como buy e sell, o servidor usa o mutex para garantir operações thread-safe.
- O servidor mantém informações de estado específicas para cada cliente (e.g., status de login, saldo de conta) para processar os comandos corretamente.

5. **Tratamento no Cliente:**

- A thread do cliente (leMsg) lê continuamente as respostas do servidor.
- Se a resposta indicar o fim da sessão (`exit`), o cliente fecha a conexão.

Justificativas de implementações próprias

Comandos extras:

Dos comandos extras temos **listarCarteira** do lado do Cliente que permite que o utilizador veja as empresas onde ele tem ações e quantas alções dessa respetiva empresa.

Do lado do servidor (Bolsa) temos comandos como **cls** que limpa a tela e o **read** que faz com que seja carregada uma lista de 10 empresas pré feitas de um ficheiro txt na memória para facilitar a defesa do trabalho.

Comunicação named pipes através de strings:

Comunicar entre cliente e servidor usando named pipes com strings como dados tem várias vantagens:

1. **Simplicidade e Clareza:**

- Utilizar strings como dados permite que a comunicação seja intuitiva e fácil de entender. Strings são um formato de dados humano-legíveis, o que facilita a depuração e o teste.

2. **Flexibilidade:**

- Como strings podem conter qualquer tipo de informação textual, é fácil alterar ou expandir o protocolo de comunicação sem alterar o tipo de dado subjacente.

3. Facilidade de Interpretação:

- Strings permitem a inclusão de metadados ou comandos de controle na própria mensagem, tornando a interpretação dos dados mais fácil e direta.

4. **Desenvolvimento Rápido:**

- O uso de strings simplifica a serialização e a desserialização dos dados, reduzindo a complexidade do código necessário para a comunicação entre o cliente e o servidor.

5. **Diagnóstico e Monitoramento:**

- Strings são mais fáceis de registar em logs e de monitorar, permitindo uma melhor auditoria e diagnóstico de problemas.

Utilização de um array vazio de empresas:

Esta decisão foi feita sobretudo para evitar um deadLock que aparecia pela exclusão mútua quando se iniciava o Board sem ter nenhuma empresa registrada o que fazia com que fosse impresso lixo na tela e o display fica-se desconfigurado.

Por isso utilizamos **evitamento** utilizando informação "extra" para fazer com que isso não acontecer.

ID	Descrição funcionalidade / requisito	Estado
1	Memória partilhada	Implementado
2	Named pipes	Implementado
3	Cliente	Implementado
4	Comandos do Administrador	Implementado
5	Comandos do Cliente	Implementado
6	Gravar constante no Registry	Implementado
7	Sincronização	Implementado
8	Board	Implementado
9	BoardGui	Implementado