ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO
POLITÉCNICO
DO PORTO

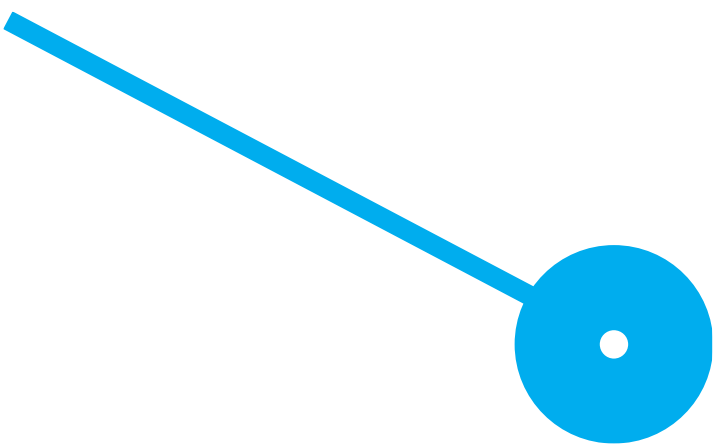
LEI

LICENCIATURA ENGENHARIA INFORMÁTICA
SISTEMAS OPERATIVOS

Trabalho Prático

Grupo A
José Magalhães – 8210125
Miguel Carvalho – 8210130

01/2024



Índice

Índice de Figuras

1 Introdução

O desenvolvimento de sistemas operativos eficientes e robustos é essencial para uma gestão eficaz dos recursos computacionais em ambientes críticos, como é o caso de satélites. Este relatório documenta o trabalho prático realizado na unidade curricular de Sistemas Operativos, cujo objetivo é simular um sistema operativo para um satélite, incorporando conceitos de multiprocessamento, comunicação e sincronização.

1.1 Âmbito / Contextualização

No âmbito desta disciplina, o trabalho proposto abrange a simulação de um sistema operativo de um satélite que gere duas unidades de computação, o CPU e a MEM, com ênfase na sincronização e comunicação entre os diferentes componentes. O propósito desta simulação está na necessidade de assegurar que os sistemas computacionais em satélites satisfaçam requisitos críticos, tais como segurança, robustez, fiabilidade e a capacidade de comunicação em tempo real. Ao enfrentar estes desafios, tivemos a oportunidade de aplicar conceitos teóricos adquiridos durante aulas práticas.

1.2 Objetivos

O principal objetivo deste projeto foi desenvolver um simulador de sistema operativo para um satélite, utilizando linguagem Java e aplicando conceitos já mencionados. O trabalho visa proporcionar uma compreensão prática da matéria abordada na unidade curricular, focando-se em situações reais de gestão de recursos computacionais em tempo real.

1.3 Estrutura do documento

Este relatório é composto por vários capítulos e subcapítulos de modo a facilitar a leitura e a interpretação dos conteúdos. A estrutura compreende os seguintes capítulos:

- **Introdução** – Este capítulo apresenta uma visão geral do trabalho.
- **Manual de compilação, configuração e utilização da aplicação** – Neste capítulo serão fornecidas orientações detalhadas sobre como compilar, configurar e utilizar a aplicação desenvolvida.
- **Descrição das funcionalidades implementadas** – Aqui, será realizada uma análise detalhada das funcionalidades desenvolvidas no sistema operativo. Cada componente, destacando as suas responsabilidades e contribuições para o bom funcionamento da aplicação.
- **Descrição e justificação da utilização de mecanismos de sincronização e comunicação entre módulos** – Este capítulo explora os mecanismos de sincronização e comunicação implementados no projeto. Serão identificados e explicados os métodos escolhidos para garantir a coordenação eficiente entre as diferentes partes do sistema.
- **Funcionalidades adicionais** – Neste último capítulo, serão apresentadas as funcionalidades adicionais propostas e implementadas para enriquecer o sistema operativo.

2 Manual de compilação, configuração e utilização da aplicação

Este trabalho foi desenvolvido na linguagem *JAVA* utilizando o *JDK (Java Development Kit) 21*, o *JRE (Java Runtime Environment) 1.8* e o *Visual Studio Code*.

Recomendamos a instalação e configuração prévia dessas ferramentas para facilitar a utilização da aplicação. O código-fonte completo está disponível no repositório do GitHub "https://github.com/TPsLEI/TP_SO". Qualquer documentação adicional ou informações relevantes também podem ser encontradas no mesmo repositório.

Para compilar e executar o projeto, é necessário seguir os seguintes passos:

1. Abrir o terminal na pasta do "projeto".
2. Executar os seguintes comandos, um de cada vez:
    - "javac -cp ./lib/miglayout.jar;./lib/miglayoutcore.jar;./lib/flatlaf.jar;./lib/json.jar;./lib/jfreechart.jar;./lib/jcommon.jar -d ./classes -encoding UTF-8 *.java"
    - "java -cp .;./lib/miglayout.jar;./lib/miglayoutcore.jar;./lib/flatlaf.jar;./lib/json.jar;./lib/jfreechart.jar;./lib/jcommon.jar;./classes Kernel"

Estes passos garantem a compilação e execução adequadas do projeto.

Outra opção, em vez de executar os comandos manualmente, é simplesmente executar o ficheiro *projeto.jar* com o comando "java -jar projeto.jar" no terminal da pasta "projeto".

Além disso, para facilitar os testes, são fornecidos dados de autenticação, com o username 'teste' e a password 'teste'. Este utilizador pode ser utilizado para verificar o correto funcionamento do sistema.

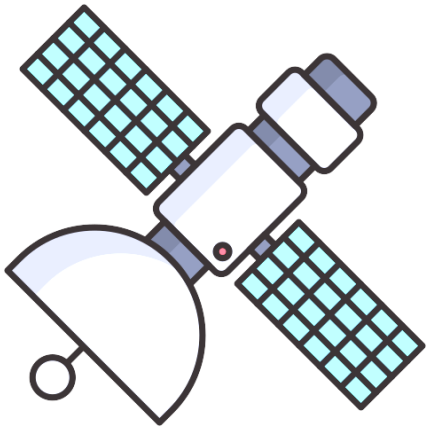


Figura 1 - Logo

3 Descrição das funcionalidades implementadas

3.1 Módulo *Kernel*

O núcleo essencial do sistema, desempenha um papel fundamental ao processar, controlar e validar as etapas fundamentais do sistema operativo de um satélite. Este módulo fica encarregue das verificações e controlo, desempenhando um papel crucial ao tomar as decisões essenciais para garantir o bom funcionamento do sistema.

A função principal do "*kernel*" é iniciar a aplicação e exibir a página de Login. Utilizando a classe *"SwingUtilities"*, garantimos que a criação da interface gráfica ocorre de forma segura, proporcionando assim uma boa experiência ao utilizador.

A função *"handleLogin"* é essencial para autenticar utilizadores. Ao receber informações de nome de utilizador e password, o "*kernel*" lê um ficheiro JSON que contém dados de utilizador. Se o nome de utilizador corresponder a um registo existente e a palavra-passe estiver correta, o utilizador é autenticado com sucesso, resultando na abertura de uma nova página do menu. Caso contrário, é fornecida uma mensagem de erro.

Também são utilizadas duas funções, *"exportMessages"* e *"exportLogs"* para exportar mensagens e logs para arquivos CSV, respetivamente. Este processo é facilitado pelo uso da biblioteca *Swing*.

Foi implementada um método que fica responsável por atualizar periodicamente a "*textBox*" do satélite, garantindo assim que as informações exibidas são sempre as mais recentes.

Por último, foi adicionada uma função que responde às mensagens recebidas.

3.2 Módulo MEM

O módulo MEM desempenha um papel fundamental no armazenamento e manipulação de dados, contribuindo para uma boa comunicação.

Na classe MEM, o método "*writeMessage*" foi desenvolvido para armazenar mensagens enviadas pelos utilizadores. A utilização do CompletableFuture possibilita a execução assíncrona dessas operações. Além disso, o método incorpora um log interno para registar as ações realizadas.

O método "*log*" é responsável por registar eventos significativos num arquivo de *log* compartilhado. A sincronização desse método garante que a escrita no arquivo seja thread-safe, prevenindo possíveis situações de competição (*race conditions*). Um *semaphore* foi utilizado para a exclusão mútua, garantindo a consistência dos registos.

A implementação também inclui a consideração da criação de diretórios necessários para os arquivos de *log*.

3.3 Módulo CPU

O módulo CPU desempenha um papel crucial na gestão, escalonamento e execução das tarefas do sistema. Este módulo é responsável por coordenar a comunicação entre o satélite e a estação, utilizando os recursos fornecidos pelas unidades de computação MEM e Middleware.

A classe CPU, que estende *Thread*, utiliza uma estrutura de dados do tipo "*LinkedBlockingQueue*" que atua como um canal de comunicação entre os diferentes componentes do sistema, permitindo uma troca segura e ordenada de informações.

No método "*run*", o CPU entra em *Loop* contínuo, aguardando constantemente por novas mensagens na fila de dados. Quando uma mensagem é recebida, o módulo CPU regista a hora atual, formata a mensagem com as informações do remetente e encaminha para a unidade MEM para um armazenamento assíncrono. Após este

processo, ocorre um atraso de 3 segundos antes da mensagem ser exibida, simulando o envio de dados em tempo real.

É relevante salientar que o método "*showMessageBox*" interage com o módulo Kernel para exibir uma caixa de mensagem com a resposta do satélite, representando a comunicação entre o mesmo e a estação.

3.4 Módulo Middleware

O módulo Middleware representa um componente crucial do sistema, desempenhando um papel essencial na comunicação entre as diferentes tarefas do sistema.

Este módulo utiliza uma "*LinkedBlockingQueue*" para facilitar a troca de informações entre os diversos componentes.

No desenvolvimento da classe Middleware, a fila de dados (*'dataQueue'*) é inicializada, estabelecendo a base para a comunicação entre os módulos. A interface gráfica (UI) é inicializada através da configuração do FlatDarkLaf, proporcionando assim uma aparência visual moderna e coerente. A classe também adota uma abordagem proativa ao iniciar um *listener* de mensagens, preparando-se para reagir a eventos de modificação no ficheiro de dados.

O método '*initMessageListener*' é responsável por iniciar uma thread que fica à espera de uma modificação no ficheiro de dados. Esta abordagem permite a deteção assíncrona de novas mensagens.

É utilizada também a função '*handleMessage*' que é essencial para o Middleware, recebendo mensagens provenientes de outras partes do sistema, inserindo-as na fila de dados (*'dataQueue'*), promovendo assim uma comunicação eficiente entre os módulos, permitindo que a informação seja devidamente encaminhada para a unidade de CPU e, posteriormente, para a memória (MEM).

4 Descrição e justificação da utilização de mecanismos de sincronização e comunicação entre módulos

No decorrer do desenvolvimento deste projeto, foram utilizados mecanismos de sincronização e comunicação entre módulos para garantir um bom funcionamento do sistema operativo.

Iniciou-se o processo com a utilização de uma "*LinkedBlockingQueue*" para facilitar a comunicação assíncrona entre os módulos, possibilitando assim uma troca eficiente de mensagens entre o Middleware, o MEM e o CPU. Na fase inicial, Middleware, MEM e CPU foram instanciados na MenuPage, estabelecendo assim a base para a interação dinâmica entre os componentes.

No Middleware, a "*LinkedBlockingQueue*" foi utilizada para receber as mensagens enviadas pela estação e direcioná-las para o CPU. Este, por sua vez, possui uma *thread* que permanece constantemente à espera de mensagens na *'dataQueue'*. Ao receber uma mensagem, o CPU formata e envia a mesma para o MEM, onde é processada e armazenada de forma assíncrona.

No MEM, foi necessário garantir acesso exclusivo ao método de escrita no arquivo CSV, levando assim a utilização do *"synchronized"*, assegurando assim que apenas uma thread por vez pode aceder a este módulo, eliminando assim a possibilidade de *race conditions* e garantindo a integridade das operações. O uso de um *"BufferedWriter"* na escrita do arquivo CSV reforçando assim a eficiência e segurança na manipulação de dados.

Foi introduzida uma espera de 3 segundos entre mensagens na *thread* do CPU, simulando assim um atraso realista no envio de dados em tempo real.

No Middleware, uma *thread* foi adiciona para monitorar alterações no arquivo CSV, informando o sistema sempre que uma mensagem é recebida e armazenada com sucesso.

A implementação de logs no MEM também foi cuidadosamente implementada, incorporando um *semaphore* para evitar sobreposições de *logs*.

Foi utilizado o CompletableFuture para facilitar a execução de operações de *log* em *threads* separadas, adotando assim uma abordagem de *multi-threading* para otimizar a eficiência do sistema.

5 Funcionalidades adicionais

Além das funcionalidades solicitadas, foram adicionadas funcionalidades extra à aplicação. Este capítulo destaca essas adições, contribuindo para uma experiência mais completa.

5.1 Interface

Durante o desenvolvimento deste projeto, foi adicionada uma interface gráfica (UI) para uma melhor interação e experiência do utilizador.

A UI foi implementada através da biblioteca Swing, com o apoio estético das bibliotecas **FlatDarkLaf** e **MigLayout**. Ao incluir está interface gráfica não só melhorou o aspeto visual do projeto, mas também permitiu uma comunicação mais dinâmica e eficiente entre o satélite e a estação. O envio de mensagens é representado de forma visualmente atrativa, melhorando assim a experiência do utilizador.

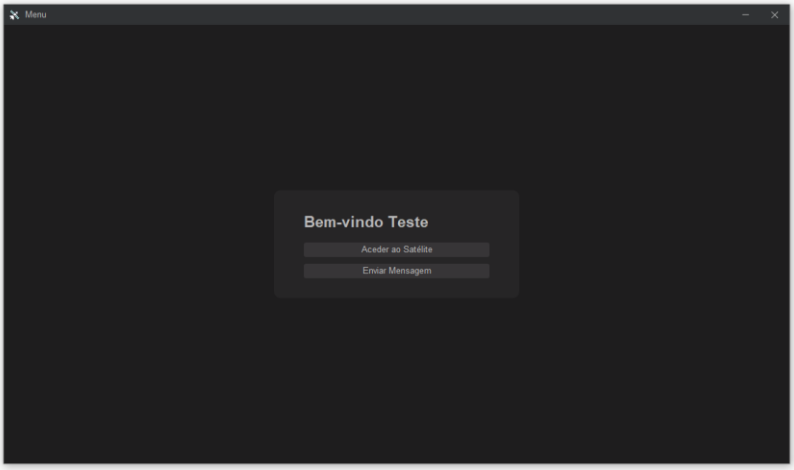


Figura 2 - Menu Page

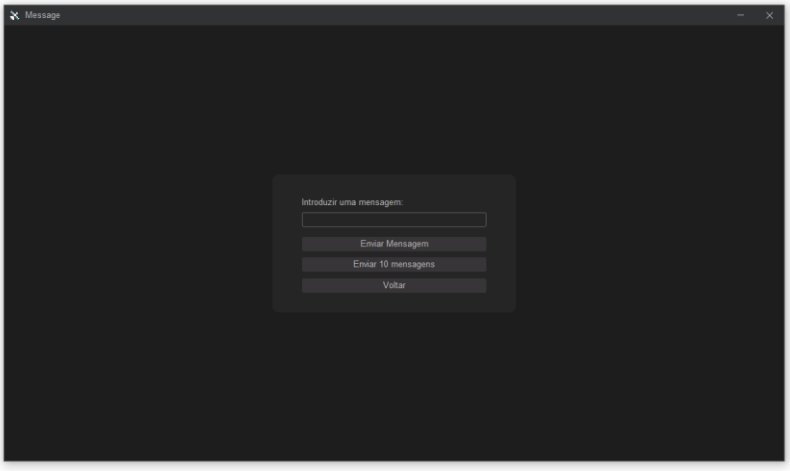


Figura 3 - Message Page

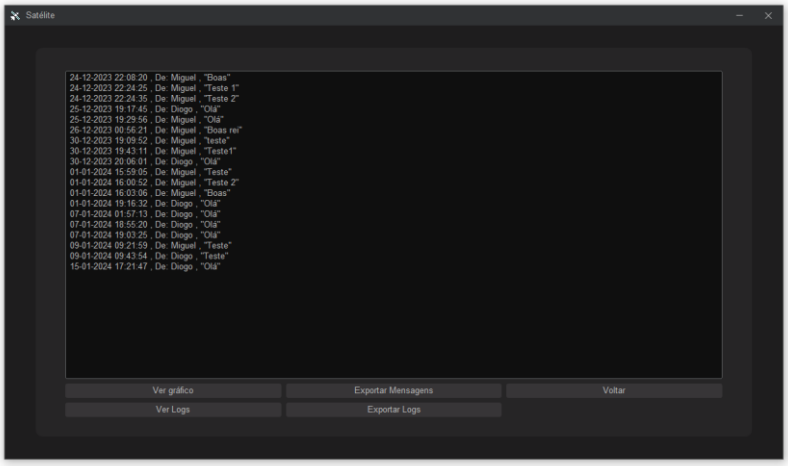


Figura 4 - Satellite Page

5.2 Módulo Login

Foi adicionado um formulário simples de Login com uma autenticação baseada num ficheiro JSON para adicionar alguma segurança ao sistema, assegurando assim que apenas utilizadores autenticados tenham permissão para interagir com as funcionalidades do sistema.

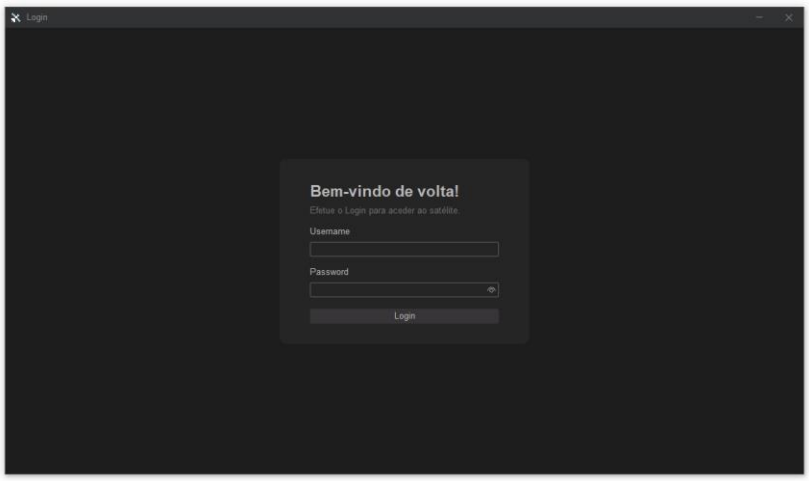


Figura 5 - Login Page

5.2.1 Ficheiro JSON

Para a leitura do ficheiro JSON foi utilizada uma biblioteca externa "***org json JSONObject***". Também é utilizada para fazer a comparação dos dados recebidos do formulário com os dados do ficheiro JSON.

5.3 Módulo Gráfico Informativo

Finalmente, foi adicionado um gráfico informativo que oferece uma visão detalhada do fluxo de mensagens. Cada barra do gráfico representa a quantidade diária de mensagens enviadas pela estação.

Este gráfico fornece uma ferramenta valiosa para análise e monitoramento, onde facilmente se pode identificar alguns padrões ou picos de atividade ao longo do tempo.

Para a implementação desta funcionalidade, foi utilizada outra biblioteca externa, "**JFreeChart**", fácil de utilizar, permitindo uma configuração rápida e eficiente de diversos tipos de gráficos.

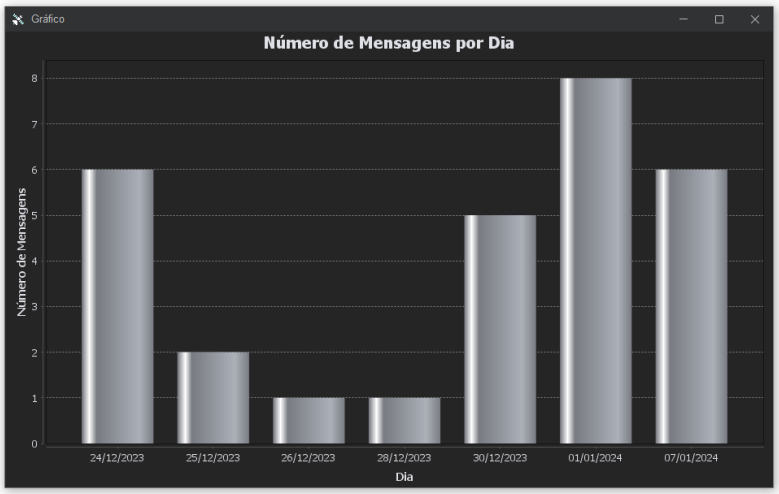


Figura 6 - Gráfico