Relatório do Trabalho Prático

Grupo AB

Miguel Tavares – 8220229

Bruno Ferreira - 8220191

Introdução

Este relatório documenta a implementação e funcionamento do trabalho prático, que envolve o desenvolvimento de um sistema de comunicação e execução de tarefas entre um utilizador e um satélite. A aplicação Java, intitulada "App", é composta por diversos módulos, incluindo um kernel, memória, CPU, middleware e interfaces gráficas.

Manual de Compilação, Configuração e Utilização

Compilação:

- Certifique-se de ter o JDK (Java Development Kit) instalado no seu sistema.
- Abra o terminal ou prompt de comando.
- Navegue até o diretório onde o código-fonte está localizado.
- Execute o seguinte comando para compilar os arquivos:
  javac *.java

Configuração:

- A aplicação não requer configuração adicional após a compilação.

Utilização:

- Após a compilação, execute o seguinte comando para iniciar o programa:
  java App
- O programa exibirá um menu interativo na consola.
- Utilize as opções do menu para interagir com o sistema de comunicação e execução de tarefas entre o utilizador e o satélite.

Outra opção é utilizar um IDE, como NetBeans, Visual Studio Code ou IntelliJ. Abra o projeto no IDE e execute-o a partir do ambiente integrado.

Manual de Utilização do Programa
Menu Principal:

O programa inicia exibindo um menu principal com as seguintes opções:

- 1.Enviar Mensagem para o Satélite
- 2.Receber Mensagem do Satélite
- 3.Adicionar Tarefa para o Satélite
- 4.Executar Tarefa do Satélite
- 5.Ver Tarefas em Memória (tarefas não executadas)
- 6.Fornecer Coordenadas do Satélite
- 7.Ver Percentagem de Memória Usada
- 8.Definições
- 9.Encerrar Programa

Siga as instruções do menu para interagir com as diversas funcionalidades do programa. Este manual fornece uma visão geral das opções disponíveis, orientando o utilzador em cada etapa.

2. Descrição das Funcionalidades Implementadas

2.1 Funcionalidades Implementadas

- **Envio de Mensagem para o Satélite:** Permite ao utilizador enviar mensagens para o satélite.
- **Recebimento de Mensagem do Satélite:** Possibilita receber mensagens enviadas pelo satélite, exibindo o conteúdo e a data de envio.
- **Adição de Tarefa para o Satélite:** Permite ao utilizador adicionar tarefas para o satélite, especificando nome, descrição e tempo estimado de execução (gerado de forma aleatória).
- **Execução de Tarefa do Satélite:** Executa uma tarefa previamente adicionada ao satélite, utilizando a CPU do kernel, através do middleware.
- **Visualização de Tarefas em Memória:** Mostra as tarefas armazenadas em memória que ainda não foram executadas.
- **Fornecimento de Coordenadas do Satélite:** Gera e exibe coordenadas aleatórias para o satélite.

- **Verificação da Percentagem de Memória Usada:** Exibe a percentagem de memória atualmente utilizada.
- **Alterar Tamanho da Memória:** Permite ao utilizador alterar o número de tasks possíveis de se alocar em memória.

2.2 Funcionalidades Implementadas de Forma Detalhada

Envio de Mensagem para o Satélite:

Implementação: É solicitado ao utilizador a entrada de uma mensagem. Essa mensagem é processada pela classe middleware, responsável pela comunicação entre o utilizador e o satélite. O middleware encaminha a mensagem para o satélite.

Recebimento de Mensagem do Satélite:

Implementação: O módulo de middleware recebe as mensagens do satélite e as armazena em uma fila de mensagens. O módulo de interface gráfica exibe as mensagens recebidas, mostrando o conteúdo e a data de envio.

Adição de Tarefa para o Satélite:

Implementação: A interface gráfica coleta os dados da nova tarefa. Esses dados são passados para o módulo de middleware, que os adiciona à fila de tarefas em memória. A tarefa de tempo estimado de execução gerado de forma aleatória pode ser estendida para incluir um intervalo específico, fornecendo ao utilizador uma estimativa mais realista do tempo necessário para a execução.

Execução de Tarefa do Satélite:

Implementação: A execução da tarefa do satélite é gerenciada pelo módulo de CPU. Quando uma tarefa é selecionada para execução, a CPU adquire um semáforo para garantir o acesso exclusivo. Em seguida, a tarefa é iniciada, e a CPU libera o semáforo após a execução. Caso não haja tarefas disponíveis, uma mensagem indicando a ausência de tarefas para execução é exibida.

Visualização de Tarefas em Memória:

Implementação: O módulo de interface gráfica consulta o módulo de memória para obter a lista de tarefas em espera. Essa lista é exibida ao utilizador, destacando as tarefas que ainda não foram processadas. Caso não haja tarefas na memória, uma mensagem informativa é apresentada indicando que a memória está vazia.

Fornecimento de Coordenadas do Satélite:

Implementação: O módulo de interface gráfica solicita ao módulo de satélite a geração de coordenadas aleatórias. Essas coordenadas são então exibidas ao utilizador. Quando solicitado pela interface gráfica, o módulo de interface gráfica chama o método provideSatelliteLocation do módulo de Middleware. Este método gera coordenadas aleatórias para a localização do satélite, que são então exibidas ao utilizador. Isso simula o fornecimento de coordenadas geográficas fictícias do satélite.

Verificação da Percentagem de Memória Usada:

Implementação: O módulo de interface gráfica obtém a informação da percentagem de memória utilizada a partir do módulo de memória. Essa informação é exibida ao utilizador para monitoramento. Caso a memória esteja completamente utilizada, uma mensagem de alerta é exibida.

Alterar Tamanho da Memória:

Implementação: O utilizador pode alterar o número de tarefas que podem ser alocadas em memória. Esta funcionalidade foi incorporada para oferecer uma flexibilidade adicional na gestão da memória, permitindo ajustar dinamicamente o tamanho da memória de acordo com as necessidades do sistema.

2.3 Descrição das classes

Nesta seção, será realizada uma descrição detalhada das classes envolvidas na implementação do sistema de comunicação e execução de tarefas entre um utilizador e um satélite.

App (Classe Principal)

Responsabilidade: A classe App é a classe principal que inicia a aplicação. Ela é responsável por criar instâncias dos diversos módulos e gerenciar a interação geral entre o utilizador e o satélite.

JanelaMemoria (Classe para Exibição de Informações de Memória)

Responsabilidade: Essa classe é responsável por exibir informações sobre a percentagem de memória em uma janela Swing.

Métodos Relevantes:

run(): Método que define o comportamento da thread ao ser iniciada, exibindo a janela com informações sobre a percentagem de memória por um determinado período de tempo.

A JanelaMemoria proporciona uma interface gráfica para monitorar a utilização de memória, utilizando a biblioteca Swing. O método run() pode ser personalizado para atender às necessidades específicas de exibição e atualização da janela.

Mensagem (Classe Que Representa Mensagens)

Responsabilidade: Representa uma mensagem que pode ser processada em uma thread separada.

Métodos Relevantes:

run(): Simula o processamento da mensagem, aguardando 5 segundos.

getRemetente(), getDestinatario(), getConteudo(), getDataEnvio(): Métodos para obter informações sobre a mensagem.

A classe Mensagem encapsula a estrutura e comportamento das mensagens do sistema. O método run() simula o processamento da mensagem e pode ser adaptado para refletir a lógica real de processamento.

MensagemWindow (Classe para Exibição de Informações de Mensagem)

Responsabilidade: Exibe informações detalhadas de uma mensagem em uma janela Swing.

Métodos Relevantes:

run(): Define o comportamento da thread ao ser iniciada, criando e exibindo uma janela com detalhes da mensagem por um período de tempo.

A MensagemWindow oferece uma visualização gráfica das mensagens, proporcionando ao utilizador uma maneira interativa de visualizar informações específicas das mensagens. O método run() controla o ciclo de vida da janela de mensagem.

Task (Classe Representando Tarefas)

Responsabilidade: Representa uma tarefa que pode ser executada em uma thread separada.

Métodos Relevantes:

run(): Contém a lógica da execução da tarefa, exibindo o progresso da tarefa e atualizando o status.

A classe Task encapsula a lógica e o comportamento de uma tarefa no sistema. O método run() representa a execução real da tarefa e pode ser personalizado conforme as exigências específicas de cada tipo de tarefa.

TaskWindow (Classe para Exibição de Informações de Tarefa)

Responsabilidade: Exibe informações detalhadas sobre a execução de uma tarefa em uma janela Swing.

Métodos Relevantes:

show(), updateStatus(status), close(), appendText(text): Métodos para exibir, atualizar, fechar a janela e adicionar texto à área de texto.

A TaskWindow proporciona uma interface gráfica para acompanhar o progresso e detalhes de execução de tarefas. Os métodos específicos fornecem funcionalidades para exibição e atualização dinâmica dos dados relacionados à tarefa.

TaskFileHandler (Classe para Leitura e Escrita de Tarefas em Arquivos)

Responsabilidade: Lida com a leitura e escrita de tarefas em arquivos.

Métodos Relevantes:

readTasksFromFile(filename, memory): Lê tarefas de um arquivo e as adiciona à memória.

saveTasksToFile(filename, tasks): Salva tarefas de uma fila em um arquivo.

O TaskFileHandler simplifica a manipulação de tarefas por meio da leitura e escrita em arquivos. Essa classe é crucial para a persistência de tarefas entre execuções do sistema, contribuindo para a consistência e confiabilidade do sistema.

MEM (Classe Representando a Memória)

Responsabilidade: Gerencia a alocação e liberação de tarefas em memória.

Métodos Relevantes:

allocateMemory(task): Aloca uma tarefa na memória.

getTasksInMemory(): Retorna a lista de tarefas em memória.

getUsedMemoryPercentage(): Retorna a percentagem de memória utilizada.

A classe MEM desempenha um papel fundamental na gestão da memória do sistema, garantindo a correta alocação e liberação de tarefas. Os métodos fornecem informações úteis sobre o estado atual da memória.

CPU (Classe Representando a Unidade Central de Processamento)

Responsabilidade: Gerencia a execução de tarefas do satélite.

Métodos Relevantes:

executeTask(task): Executa uma tarefa, adquirindo um semáforo para garantir acesso exclusivo durante a execução.

A classe CPU é responsável pela execução das tarefas do satélite. A utilização de semáforos garante uma execução segura e controlada, evitando conflitos de acesso às tarefas durante a execução.

Middleware (Classe de Comunicação Entre Módulos)

Responsabilidade: Facilita a comunicação entre diferentes módulos da aplicação.

Métodos Relevantes:

sendMessageToSatellite(message): Encaminha mensagens para o satélite.

receiveMessagesFromSatellite(): Recebe mensagens do satélite.

O Middleware desempenha um papel crucial na comunicação eficiente entre o Kernel e o Satellite. Ele coordena o envio e recebimento de tarefas e mensagens, garantindo que a integridade dos dados seja mantida por meio do uso de semáforos.

Kernel (Classe Representando o Núcleo do Sistema)

Responsabilidade: Representa o núcleo de um sistema computacional, composto por uma memória (MEM) e uma CPU. Controla o início e término do sistema.

Métodos Relevantes:

start(): Inicia o kernel, iniciando as threads da memória e da CPU.

stop(): Interrompe o kernel, interrompendo as threads da memória e da CPU.

getMemory(), setMemory(memory): Métodos para obter e definir a instância da memória associada ao kernel.

getCpu(), setCpu(cpu): Métodos para obter e definir a instância da CPU associada ao kernel.

isRunning(), setRunning(running): Métodos para verificar e definir o estado de execução do kernel.

Atributos Relevantes:

memory: Instância da classe MEM que representa a memória associada ao kernel.

cpu: Instância da classe CPU que representa a CPU associada ao kernel.

isRunning: Indica se o kernel está em execução ou não.

O Kernel atua como o ponto central do sistema, gerenciando a comunicação e execução entre a memória e a CPU. Ele fornece métodos para iniciar e interromper o sistema, bem como acessar e modificar as instâncias da memória e da CPU associadas. O controle de execução do kernel garante a estabilidade e a consistência do sistema como um todo.

2.4 Funcionalidades Não Implementadas

Até o momento, todas as funcionalidades especificadas foram implementadas. Nenhuma funcionalidade solicitada está ausente.

3. Descrição e Justificação da Utilização de Mecanismos de Sincronização e Comunicação entre Módulos

A aplicação utiliza diversos mecanismos de sincronização e comunicação entre seus módulos para garantir o correto funcionamento e a integridade dos dados. Algumas das principais justificações são:

Semáforos e Sincronização entre Threads: O uso de semáforos é fundamental para garantir a exclusão mútua em regiões críticas do código, especialmente ao lidar com a alocação e liberação de memória e execução de tarefas.

Fila de Mensagens (Queue): a utilização de uma fila para o armazenamento de mensagens e tasks promove a comunicação assíncrona e eficiente entre o middleware, a memória e a CPU, permitindo que operem de forma independente. Optamos pela fila (queue) devido à natureza assíncrona do sistema operativo. Em ambientes onde as tarefas são executadas com base no tempo em memória, a queue oferece uma solução eficaz, contribuindo para a previsibilidade e eficiência na troca de mensagens, mantendo uma ordem adequada nas operações. Essa escolha é especialmente valiosa em contextos onde a execução de tarefas é determinada pelo tempo em que estão em memória, tornando a fila de mensagens um componente crucial para uma comunicação eficiente e ordenada.

Ficheiros de Texto: A leitura e gravação em ficheiros de texto são empregues para persistência de dados, permitindo que informações como mensagens e tarefas sejam mantidas entre diferentes execuções do programa.

4. Enumeração das Funcionalidades Pedidas e Não Implementadas

A enumeração das funcionalidades foi apresentada na seção 2.2, onde foi destacado que todas as funcionalidades solicitadas foram implementadas com sucesso.

5. Exploração de Funcionalidades Avançadas e Possíveis Melhorias Futuras

A aplicação apresenta uma implementação sólida das funcionalidades requeridas. Algumas áreas para exploração e melhorias futuras incluem:

- **Interfaces Gráficas Melhoradas:** Explorar o uso de interfaces gráficas mais avançadas para proporcionar uma experiência de utilizador mais amigável.
- **Aprimoramento da Persistência de Dados:** Reforçar o sistema de leitura e gravação em arquivos para garantir maior segurança e integridade dos dados armazenados.

Este relatório apresenta uma visão geral da aplicação, suas funcionalidades, mecanismos de sincronização e justificações para escolhas de design. Certifique-se de seguir as instruções de compilação e execução para aproveitar todas as funcionalidades disponíveis.