Trabalho Prático

Licenciatura em Engenharia Informática

Sistemas Operativos

2023/2024

Grupo X

DANIEL LIMA FÁRIA, SÉRGIO LUÍS LOPES FÉLIX

8200113@estg.ipp.pt, 8200615@estg.ipp.pt

Instituto Politécnico do Porto, Escola Superior de Tecnologia e Gestão, Rua do Curral, Casa do Curral – Margaride, 4610-156 Felgueiras, Portugal

Índice

Índice de Figuras

Introdução

O seguinte trabalho foi desenvolvido no âmbito da unidade curricular Sistemas Operativos, com o objetivo de consolidar os conhecimentos obtidos ao longo das aulas lecionadas.

O objetivo geral visa simular o sistema operativo de um satélite, onde duas unidades de processamento, MEM e CPU, precisam ser coordenadas para garantir a comunicação em tempo real com uma estação terrestre. Inspirado em sistemas operativos de tempo real, como o Rodos, este projeto abrange conceitos fundamentais, incluindo compartilhamento de recursos, seção crítica, condições de corrida e técnicas de sincronização.

Manual de compilação

O seguinte projeto foi desenvolvido utilizando o Java 21, aproveitando as funcionalidades mais recentes da linguagem para oferecer uma implementação eficiente e moderna. Para compilar e executar o programa, é necessário utilizar o sistema de automação de compilação Gradle.

Abaixo segue as instruções para compilar e executar o projeto:

1. **Instalação do Gradle:**
   Como é necessário a ferramenta Gradle para prosseguir na execução do projeto é possível fazer o download do mesmo aqui e seguir as instruções indicadas.

2. **Compilação do Projeto:**
   Diriga-se até o diretório do projeto usando o terminal e execute o seguinte comando para compilar o projeto: **`gradle build`**.
   O comando irá baixar as dependências do projeto, compilar o código-fonte e gerar os artefactos necessários para a execução.

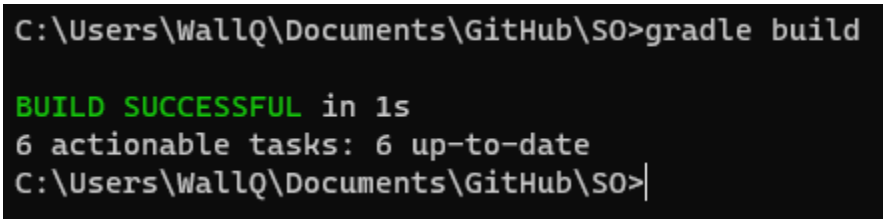

Figura 1 - Comando para compilar o projeto

3. **Execução do Projeto:**
   Após a conclusão bem-sucedida da etapa de compilação, você pode iniciar o programa com o seguinte comando: **`gradle run`**.
   Isso iniciará a aplicação, permitindo então simular o sistema operativo do satélite.

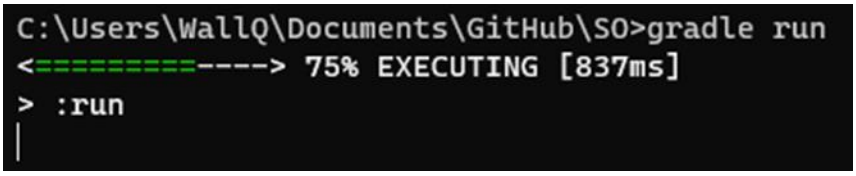

Figura 2 - Comando para executar a aplicação

Manual de configuração

O sistema operativo simulado para o satélite oferece algumas possibilidades para configuração através da classe Config, permitindo a personalização de parâmetros essenciais. Esta seção do manual detalha como ajustar as configurações por meio da classe Config no código-fonte.

```
public class Config {
    1 usage
    public static final int MAX_MEMORY = 1536;
    1 usage
    public static final int MAX_CONCURRENT_CONSUMERS = 5;
    1 usage
    public static final int NEXT_TASK_DELAY = 1500;
    1 usage
    public static final int VERIFY_TASK_EXECUTION_DELAY = 2500;
}
```

Figura 3 - Classe de configuração do programa

1. **Localização da Classe Config:**
   A classe Config está localizada no diretório do projeto, dentro do pacote **utils**. Abra o arquivo Config.java em um editor de código para acessar e modificar as configurações.

2. **Configurações na Classe Config:**
   Dentro da classe Config.java, você encontrará atributos que podem ser ajustados conforme necessário. Algumas configurações notáveis incluem:

   MAX_MEMORY: Tamanho máximo da unidade MEM. Ajuste esse valor para definir a capacidade de alocação de memoria para a execução das tarefas.

   MAX_CONCURRENT_CONSUMERS: Número máximo de consumidores simultâneos no Middleware.

   NEXT_TASK_DELAY: Tempo que a unidade da CPU deve esperar antes de executar a próxima tarefa.

3. **Atualização, compilação e execução:**
   Após realizar as alterações desejadas na classe Config.java, certifique-se de seguir as instruções no "Manual de Compilação" para compilar e executar o projeto utilizando o Gradle de forma a sortir as alterações efetuadas.

Funcionalidades implementadas

Middleware

O Middleware desempenha um papel crucial como ponte entre a aplicação e o sistema operativo, facilitando a comunicação eficiente e coordenada entre esses dois componentes do sistema simulado para o satélite. Uma característica fundamental do Middleware é o seu uso de semáforo, com um limite de 5 permissões, que age como um mecanismo de controle para regular o acesso concorrente ao sistema. Essa abordagem garante uma execução ordenada e eficaz das tarefas, evitando possíveis conflitos de acesso. Além disso, o Middleware mantém uma fila (queue) de tarefas a serem enviadas para execução, organizando-as de acordo com a ordem de chegada. Simultaneamente, mantém uma lista de tarefas já realizadas recebidas do sistema operativo, prontas para serem transmitidas de volta para a aplicação. Essa arquitetura inteligente do Middleware contribui significativamente para a sincronização e comunicação efetiva entre as unidades operativas, promovendo um funcionamento coordenado e eficiente do sistema.

Kernel

O Kernel desempenha um papel central na gestão e coordenação das unidades operativas CPU e MEM, exercendo controle fundamental sobre o sistema simulado do satélite. Encarregado de inicializar e finalizar as unidades de processamento e memória, o Kernel incorpora um Scheduler, que atua como intermediário para as tarefas transmitidas pelo Middleware. Este Scheduler organiza as tarefas em três filas de prioridade: alta, média e baixa. Ao receber uma nova tarefa, ela é atribuída à fila correspondente à sua prioridade. No processo de retornar uma tarefa invocada pela CPU, o Scheduler adota uma estratégia inteligente para evitar possíveis situações de *starvation*. Se o contador de *starvation* estiver em zero, o Scheduler envia uma tarefa de prioridade alta ao processador para execução imediata. Se o contador estiver entre 1 e 2, são enviadas duas tarefas consecutivas de prioridade média. Por fim, se o contador de *starvation* for igual ou superior a 3, são enviadas três tarefas consecutivas de prioridade baixa. Essa abordagem busca mitigar a possibilidade de *starvation*, garantindo uma distribuição equitativa das tarefas de diferentes prioridades. Além disso, o Kernel possui a capacidade de solicitar alocação de memória à MEM conforme necessário para a execução de tarefas pela CPU. Essa arquitetura robusta do Kernel é fundamental para a governança eficiente do sistema operativo simulado para o satélite.

CPU

O CPU, unidade central de processamento, desempenha um papel crucial na execução eficiente das tarefas no sistema operativo simulado para o satélite. Encarregado de verificar se o Kernel possui tarefas agendadas, o CPU inicia o processo de execução. Antes de realizar a execução, faz uma solicitação ao Kernel para requisitar à MEM a alocação de memória necessária para a tarefa em questão. Uma vez obtida a alocação de memória, o CPU transfere as tarefas do estado de espera para o estado de execução, adicionando-as à lista de tarefas em execução mantida pelo Kernel. Após a conclusão da execução, o CPU solicita ao Kernel que faça uma requisição à MEM para adicionar o resultado à tarefa, garantindo uma comunicação eficiente entre as unidades do sistema. Essa coordenação entre o CPU, Kernel e MEM é essencial para assegurar uma execução suave e eficaz das tarefas no contexto do satélite simulado.

MEM

A unidade MEM desempenha um papel crucial na gestão eficiente da memória no sistema operativo simulado para o satélite. Sua função principal é controlar o uso da memória no sistema, proporcionando operações essenciais, como alocação e liberação de memória. A MEM permite a alocação de espaço necessário para a execução das tarefas, garantindo uma distribuição eficiente dos recursos de memória disponíveis. Além disso, oferece a capacidade de liberar a memória utilizada após a conclusão das tarefas, contribuindo para a otimização do espaço. A MEM também fornece mecanismos para verificar o estado atual da memória, possibilitando uma gestão proativa dos recursos disponíveis. Uma funcionalidade adicional da MEM é a capacidade de modificar os resultados das tarefas, permitindo ajustes dinâmicos nas operações do sistema. Em resumo, a MEM desempenha um papel crucial na garantia da eficiência e da integridade do gerenciamento de memória no contexto do sistema simulado para o satélite.

Mecanismos de sincronização

A aplicação desenvolvida adota estratégias de sincronização para garantir a execução correta e segura em ambientes multi-threading. Estes mecanismos são essenciais para prevenir condições de corrida e assegurar que a manipulação de recursos partilhados seja efetuada de forma coordenada e consistente. Abaixo estão os principais mecanismos de sincronização empregues:

Semaphore no Middleware

A classe Middleware faz uso de semáforos para controlar o acesso concorrente à sua secção crítica, onde as tarefas são adicionadas à fila. O semáforo regula o número máximo de threads que podem aceder simultaneamente a recursos partilhados, prevenindo congestionamentos e assegurando uma execução coordenada.

```
public void sendTask(Task task) {
    try {
        System.out.println("[Middleware] Sending task to Kernel!");
        semaphore.acquire();
        tasks.add(task);
        kernel.receiveTask(Objects.requireNonNull(tasks.poll()));
    } catch (InterruptedException e) {
        Logger.getLogger(Middleware.class.getName()).log(Level.SEVERE,
        System.exit( status: 1);
    } finally {
        semaphore.release();
    }
}
```

Figura 4 - Utilização do semáforo na requisição de envio de tarefa para o Kernal

Synchronized no CPU

Na classe CPU, blocos synchronized são empregues para garantir a coerência nas operações que envolvem as listas de tarefas (tasks e tasksInExecution). Estas áreas críticas são sincronizadas para evitar condições de corrida durante a manipulação concorrente dessas estruturas de dados.

```
@Override
public void run() {
    try {
        while (kernel.isRunning()) {
            synchronized (kernel.tasks) {
                if (!kernel.tasks.isEmpty()) {
                    Task task = kernel.tasks.getTask();

                    if (task == null) continue;

                    if (kernel.allocateMemory(task)) {
                        Thread thread = new Thread(task);
                        long startTime = System.currentTimeMillis();

                        System.out.println("[CPU] Processing task!");
                        thread.start();

                        synchronized (kernel.tasksInExecution) {
                            kernel.tasksInExecution.add(task);
                        }
```

Figura 5 - Utilização da keyword synchronized em dois blocos

Estas estratégias de sincronização contribuem para a robustez da aplicação, assegurando que a execução concorrente de tarefas é coordenada e livre de condições indesejadas.

Comunicação entre módulos

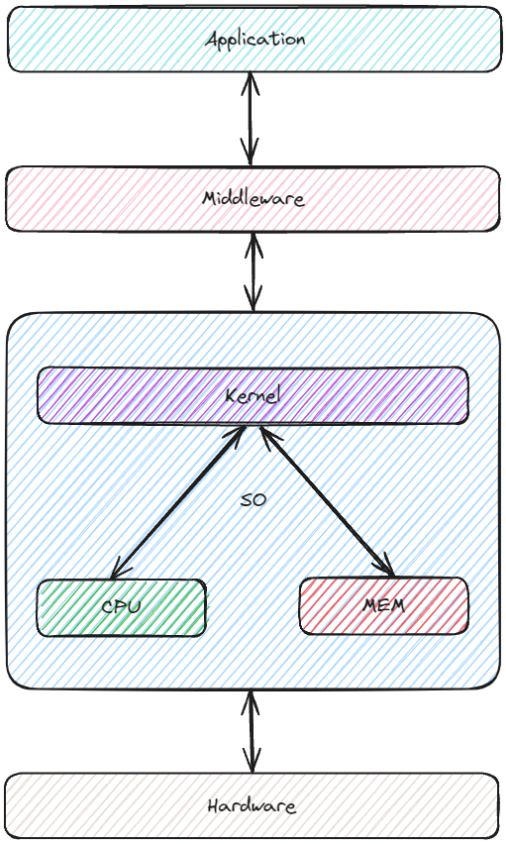

Figura 6 - Diagrama de comunicação de módulos

A ilustração acima demonstra a comunicação eficiente entre os módulos do sistema operativo simulado para o satélite. A comunicação é intermediada pelo Middleware, que atua como uma ponte essencial entre a aplicação e o Kernel. O Middleware possibilita o envio e recebimento de tarefas, bem como a ativação ou desativação do sistema operativo. A aplicação se comunica com o Kernel através do Middleware, enviando tarefas a serem executadas e recebendo os resultados após a conclusão. As requisições, tanto da aplicação quanto do Kernel, são gerenciadas no Middleware, que utiliza a aquisição de permissões de um semáforo para coordenar as operações. Na interação entre o Kernel e o CPU, o CPU permanece em escuta para tarefas em espera no Kernel, executando-as assim que estão disponíveis. Quanto à comunicação entre o Kernel e a MEM, o Kernel, ao receber uma solicitação do CPU para alocar memória para uma tarefa, realiza a requisição correspondente à MEM. Após a conclusão da execução da tarefa, o Kernel faz uma requisição à MEM para liberar a memória alocada e modificar o resultado da tarefa, garantindo uma gestão eficiente dos recursos de memória no sistema simulado. Essa arquitetura de comunicação entre módulos contribui para a sincronização harmoniosa e operação eficaz do sistema operativo simulado para o satélite.

Funcionalidades não implementadas

No tópico das funcionalidades não implementadas, um ponto que requer atenção é a unidade de memória (MEM), que é responsável pelo armazenamento e manipulação de dados e informações. Até o momento, observa-se que o armazenamento de dados na aplicação não é utilizado com um propósito específico, levantando a questão de sua relevância e necessidade. Além disso, a manipulação de dados pela MEM não está claramente definida em relação às expectativas do sistema operativo simulado para o satélite.

Outro aspecto a ser considerado é a funcionalidade do Middleware, que é responsável por gerir mensagens e objetos para definir um serviço de comunicação de satélite. Embora a implementação do Middleware esteja presente, a eficácia dessa camada intermediária em facilitar uma comunicação robusta entre as tarefas e os subcomponentes do sistema pode necessitar de uma avaliação mais aprofundada.

Por fim, salientar que não foram incorporadas no sistema operativo simulado para o satélite as funcionalidades avançadas, tais como a geração de gráficos para visualização das informações coletadas pelo sistema, a persistência de dados relevantes para a simulação ou até mesmo a sinalização gráfica de situações de competição como sugerido.

Conclusão

Em resumo, a realização deste trabalho prático foi uma mais-valia, pôr nos colocar a aprova de aplicar os conceitos teóricos aprendidos durante as aulas lecionadas.

O desenvolvimento do sistema operativo simulado para o satélite apresentou desafios significativos, com uma das principais dificuldades sendo a identificação dos pontos críticos da aplicação. O projeto exigiu uma abordagem cuidadosa para garantir a coordenação eficiente entre os diversos componentes. A arquitetura proposta enfatiza a prevenção de problemas como *race conditions* e *starvation*, incorporando conceitos sólidos, como o uso de semáforos para controlar o acesso concorrente.

Sendo assim, o resultado representa não apenas uma implementação eficaz, mas também uma compreensão aprofundada dos princípios subjacentes dos sistemas operativos.