

2023-2024

P.PORTO

**ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO**

SISTEMAS OPERATIVOS

TRABALHO PRÁTICO – ÉPOCA NORMAL

Trabalho elaborado por:

Grupo 19

8220169 – César Ricardo Barbosa Castelo

8220337 – Hugo Ricardo Almeida Guimarães

8220307 – Pedro Marcelo Santos Pinho

Índice

Índice de Figuras	2
Chave de Siglas	3
Introdução	4
1. Manual de compilação	5
2. Manual de Utilização	6
3. Funcionalidades Implementadas	10
• Middleware.....	10
• Kernel	10
• CPU.....	10
• Memória (MEM).....	11
• Agendador de tarefas (task scheduler)	11
4. Mecanismos de Sincronização Utilizados.....	12
5. Comunicação entre módulos.....	14
• Comunicação aplicação e Middleware	14
• Comunicação Middleware e kernel.....	14
• Comunicação Kernel e MEM.....	14
• Comunicação Kernel e CPU.....	14
6. Funcionalidades não implementadas.....	15
Conclusão	16

Índice de Figuras

Figura 1 - Comando para a compilação do programa.....	5
Figura 2 - Comando para a execução do programa.....	5
Figura 3 - Ficheiro de configuração.....	6
Figura 4 - Janela inicial do programa	6
Figura 5 - Consola com as tarefas recebidas	7
Figura 6 - Gráfico circular das tarefas	7
Figura 7 - Gráfico de barras das tarefas	8
Figura 8 - Gráfico de barras da memória usada	8
Figura 9 – Ficheiro com os registos da terra	8
Figura 10 – Ficheiro com os registos do satélite.....	9
Figura 11 - Conceção do buffer do middleware	10
Figura 12 - Exemplo da palavra-chave "Synchronized".....	12
Figura 13 - Semáforos do Middleware.....	13
Figura 14 - Conceção de como os semáforos funcionam	13
Figura 15 - Diagrama geral da comunicação entre os módulos	14

Chave de Siglas

API	Application Program Interface
CPU	Central Processing Unit

Introdução

Este trabalho foi realizado para o âmbito da disciplina de *Sistemas Operativos*, que funcionará como integrador dos conhecimentos adquiridos no decorrer das aulas. Ele consiste na elaboração de um programa em Java, que faça uso das técnicas de multiprocessamento, comunicação e sincronização aprendidas nas aulas para a construção de uma simulação de um sistema operativo de um satélite que gere duas unidades de computação (o CPU e a MEM) em tempo-real, recorrendo a mecanismos de sinalização e comunicação lecionados durante as aulas práticas.

A simulação proposta desempenha um papel crucial na consolidação do conhecimento teórico adquirido, proporcionando uma compreensão prática e aprofundada dos princípios dos Sistemas Operativos. Ao modelar a gestão em tempo-real de duas unidades de computação, o CPU e a MEM, o projeto visa demonstrar a aplicação eficaz de mecanismos de sinalização e comunicação, destacando a relevância prática desses conceitos.

1. Manual de compilação

Aviso: Todo este projeto foi construído em cima do java 17.0.8 LTS, qualquer versão superior ou inferior a esta pode resultar na não compilação do código.

Para compilar o projeto, primeiramente tem de se entrar dentro da pasta da aplicação a partir do terminal e inserir o comando presente na figura 1.

```
C:\Users\hugui\Documents\GitHub\so_grupoU>javac -encoding utf8 -cp .\lib\jfreechart-1.5.4.jar -d .\bin -sourcepath .\src .\src\*.java
Picked up _JAVA_OPTIONS: -Xmx512M
Note: src\TasksCircularChart.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
Note: src\TasksCircularChart.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

FIGURA 1 - COMANDO PARA A COMPILAÇÃO DO PROGRAMA

Este comando os seguintes parâmetros:

- **-encoding utf8** - Especifica a codificação de caracteres a ser usada durante a compilação.
- **-cp .\lib\jfreechart-1.5.4.jar** - Define o caminho para a biblioteca externa usada no programa, o *JFreeChart*¹.
- **-d .\bin** - Especifica a pasta de destino para os ficheiros compilados, que será na pasta *bin*
- **-sourcepath .\src** - Especifica onde procurar os ficheiros com o código do programa, que será na pasta *src*.
- **.\src*.java** - Lista de todos ficheiros com o código do programa a serem compilados, que serão todos os ficheiros com a extensão *.java*, na pasta *src*

Quando todos os ficheiros são compilados, o terminal irá mostrar umas notas que dizem que alguns ficheiros usam versões desatualizadas de algumas API's, mas é normal aparecer essas mensagens, pois o programa foi propositadamente construído em cima dessas versões.

Após todos os ficheiros estarem compilados, basta executar o comando presente na figura 2 para que o programa seja executado

```
C:\Users\hugui\Documents\GitHub\so_grupoU>java -cp .\lib\jfreechart-1.5.4.jar;.\bin\ .\src\application\App.java
```

FIGURA 2 - COMANDO PARA A EXECUÇÃO DO PROGRAMA

Este comando tem os seguintes parâmetros:

- **-cp .\lib\jfreechart-1.5.4.jar;.\bin** - Define onde procurar as classes e as bibliotecas externas durante a execução. Neste caso, está incluído o ficheiro JAR da biblioteca *JFreeChart* e a pasta onde as classes compiladas estão localizadas.
- **.\src\application\App.java** - Especifica a classe Java a ser executada. Neste caso, vai executar a classe *App* localizada na pasta *src\application\App.java*

¹ uma biblioteca em Java que facilita a exibição de gráficos.

2. Manual de Utilização

Antes da inicialização do programa existe um ficheiro de configurações dentro da pasta “Files”, nele constam todas as configurações que o programa necessita para poder executar como, por exemplo: os tempos de espera até à execução das próximas tarefas, tamanho da MEM, ou algumas mensagens. Todos esses valores podem ser alterados pelo utilizador, visto que o ficheiro foi guardado e é lido diretamente como .txt, logo, pode ser lido e reescrito facilmente.

```
TamanhoMem:500
StressTestNumeroDeTarefas:100
StressTestTempoMinimoTarefa:1000
StressTestTempoMaximoTarefa:10000
StressTempoEntreTarefas:200
MensagemRespostaTarefa:Mensagem de conclusão fictícia
TempoEsperaAteProximaTarefa:500
PeriodoVerificacaoTarefasEmExecucao:100
TempoAtualizacaoDoGrafico:200
```

FIGURA 3 - FICHEIRO DE CONFIGURAÇÃO

Quando o programa é executado irá aparecer a janela presente na Figura 4. Ela contém um botão de ligar e desligar, que irá ligar e desligar o sistema operativo do satélite, também tem uma secção de tarefas onde é possível gerar uma tarefa customizada, com: um nome; uma mensagem; memória que irá necessitar; tempo expectado para a tarefa e a prioridade da mesma. Para além disso, também tem um botão que efetua um teste de stress, que irá executar várias tarefas ao mesmo tempo, tarefas essas com valores aleatórios, que podem ser definidos no pelo ficheiro de configuração, este teste serve para testar como é que o CPU opera sobre muita pressão. Lembrando que estas funcionalidades funcionam apenas quando sistema operativo, está ligado.

A imagem mostra a interface gráfica do programa. No topo, há uma barra de título com o ícone do programa e botões de minimizar, maximizar e fechar. O conteúdo da janela é dividido em duas colunas principais. À esquerda, há dois botões: 'Ligar' (destacado em azul) e 'Desligar' (cinza). À direita, sob o título 'Tarefa', há cinco campos de entrada: 'Nome:' (campo de texto), 'Mensagem:' (campo de texto), 'Memória' (menu suspenso com o valor '5'), 'Tempo Expectado:' (menu suspenso com o valor '1 segundo') e 'Prioridade:' (menu suspenso com o valor 'Alta Prioridade'). Abaixo desses campos, há um botão 'Gerar Tarefa' (cinza). No rodapé da janela, há um botão 'Teste de Stress' (cinza).

FIGURA 4 - JANELA INICIAL DO PROGRAMA

Quando o satélite é ligado, irão aparecer quatro novas janelas. Uma dessas janelas é uma consola que representa o monitor da Terra, onde irão ser exibidas as tarefas enviadas para o satélite, bem como as respostas recebidas. As outras janelas são dois gráficos que têm quase os mesmos valores, mas exibidos de maneiras diferentes. Um deles é um gráfico circular que tem a quantidade de tarefas que estão em espera, em execução e as que já finalizaram, o segundo, é um gráfico de barras que contém apenas as tarefas que estão em execução bem como as tarefas que estão em espera na CPU. Este segundo gráfico, mesmo contendo os mesmos valores que o primeiro, é extremamente útil, pois como não tem as tarefas finalizadas, e como é um gráfico de barras, temos uma maior noção da quantidade de tarefas que estão a ser executadas em função das que estão em espera. O último gráfico é um gráfico com uma barra na horizontal que representa a quantidade de memória que está a ser usada pelas tarefas em execução.

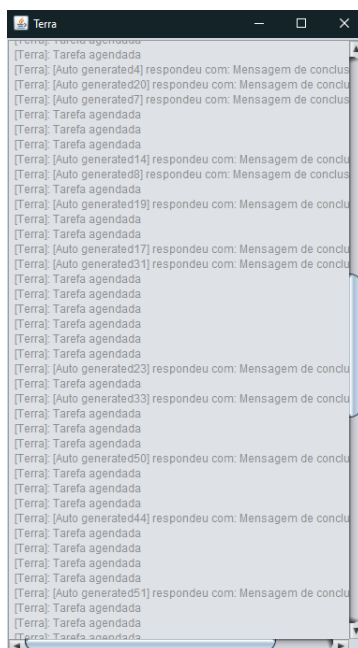


FIGURA 5 - CONSOLA COM AS TAREFAS RECEBIDAS

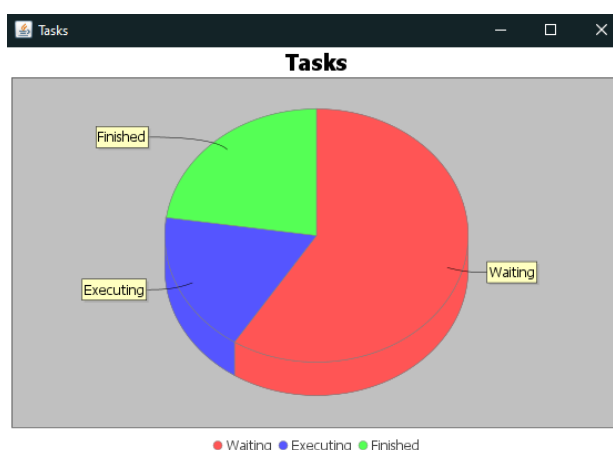


FIGURA 6 - GRÁFICO CIRCULAR DAS TAREFAS

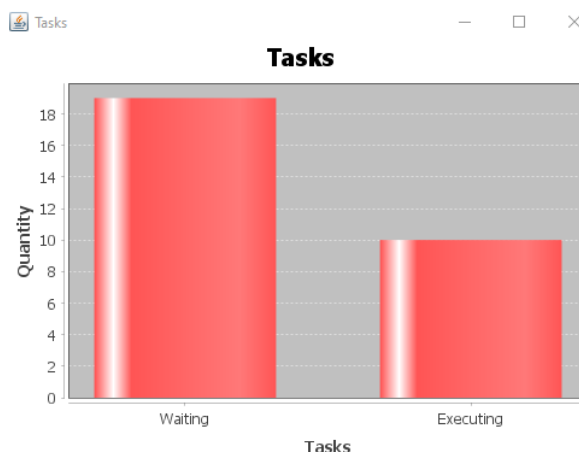


FIGURA 7 - GRÁFICO DE BARRAS DAS TAREFAS

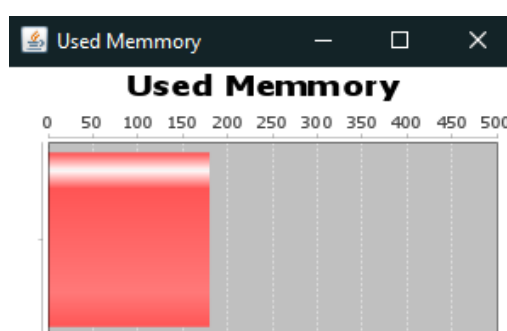
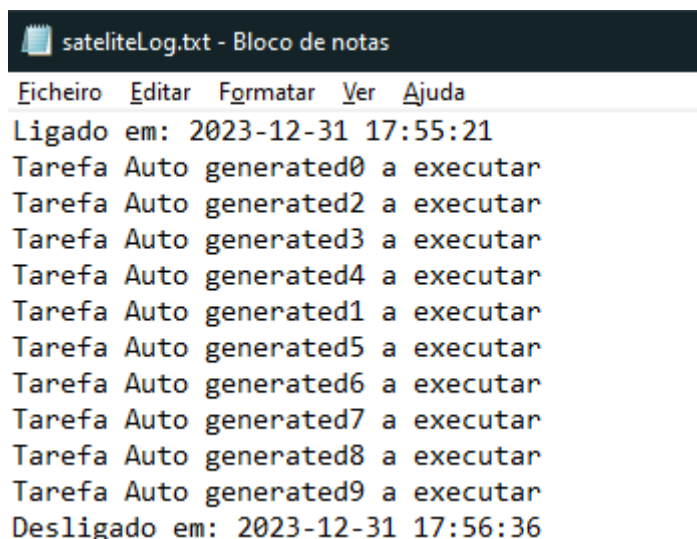


FIGURA 8 - GRÁFICO DE BARRAS DA MEMÓRIA USADA

Caso se queira saber o que aconteceu durante a execução do programa, com ele já encerrado, basta ir à pasta “.\Files”, lá contém dois ficheiros “.txt” que registam os eventos entre a Terra e o satélite. O ficheiro da Terra contém as tarefas que foram agendadas, e as mensagens que foram recebidas do satélite, enquanto o satélite tem as tarefas que lhe foram enviadas e as respostas que ele deu às mesmas, contém também a data e hora que este ligou e encerrou.

```
terraLog.txt - Bloco de notas
Ficheiro  Editar  Formatar  Ver  Ajuda
Tarefa agendada
Tarefa agendada
Tarefa agendada
Tarefa agendada
Tarefa agendada
[Auto generated1] respondeu com: Mensagem de conclusão fictícia
Tarefa agendada
[Auto generated5] respondeu com: Mensagem de conclusão fictícia
Tarefa agendada
[Auto generated0] respondeu com: Mensagem de conclusão fictícia
Tarefa agendada
[Auto generated2] respondeu com: Mensagem de conclusão fictícia
Tarefa agendada
...
```

FIGURA 9 – FICHEIRO COM OS REGISTOS DA TERRA



```
sateliteLog.txt - Bloco de notas
Ficheiro Editar Formatar Ver Ajuda
Ligado em: 2023-12-31 17:55:21
Tarefa Auto generated0 a executar
Tarefa Auto generated2 a executar
Tarefa Auto generated3 a executar
Tarefa Auto generated4 a executar
Tarefa Auto generated1 a executar
Tarefa Auto generated5 a executar
Tarefa Auto generated6 a executar
Tarefa Auto generated7 a executar
Tarefa Auto generated8 a executar
Tarefa Auto generated9 a executar
Desligado em: 2023-12-31 17:56:36
```

FIGURA 10 – FICHEIRO COM OS REGISTOS DO SATÉLITE

Quando o sistema operativo é desligado, ele vai demorar um bocado para o fazer, pois vai estar a apagar todas as tarefas que estão em espera, e vai esperar que todas as tarefas que estejam em execução terminem para assim poder desligar o sistema operativo em segurança.

3. Funcionalidades Implementadas

- **Middleware**

É um componente intermediário entre a aplicação e o sistema operativo, ele contém um buffer de cinco posições que armazena as tarefas que vão ser mandadas para o sistema operativo, e as tarefas que o sistema operativo já executou e que vão ser mandadas de volta para a aplicação com uma resposta. É reservado no buffer três posições para as tarefas que vão ser mandadas, a partir do método “*send()*”, e apenas duas posições para a leitura das tarefas já processadas, a partir do método “*receive()*”.

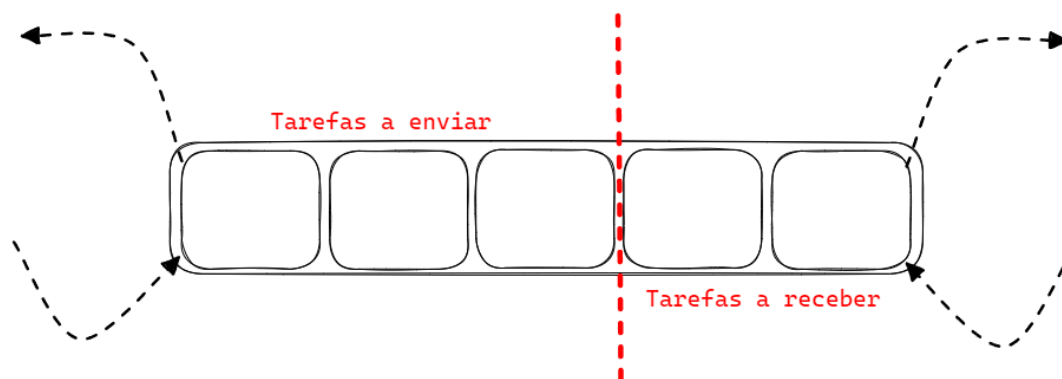


FIGURA 11 - CONCEÇÃO DO BUFFER DO MIDDLEWARE

- **Kernel**

O Kernel é o principal componente do sistema operativo, é ele que controla e supervisiona todos os componentes, é graças a ele que todos os componentes podem trabalhar de forma sincronizada.

É o Kernel que vai retirar as tarefas que vão ser executadas do buffer, que está no middleware, e colocá-las em espera, assim quando o CPU for executá-las, o kernel, que é usado como intermediário, reserva a memória necessária para a execução da tarefa, e quando a execução da tarefa acabar, o kernel vai colocá-las como terminadas e enviá-las outra vez para o buffer do middleware, que por sua vez vão para a aplicação, que é onde irão ser exibidas as respostas ao utilizador.

- **CPU**

A CPU é um componente que desempenha um papel crucial no sistema, sendo o componente responsável pela execução efetiva das tarefas, fazendo com que elas passem do estado de espera, para o estado de execução.

A classe CPU é projetada para ser executada como uma Thread² separada. Isso implica que a CPU opera de forma concorrente, permitindo a execução de tarefas simultaneamente com outras operações do sistema.

² Divisão ou subdivisão de um processo, que permite com que várias tarefas possam ser executadas concorrentemente

- Memória (MEM)

Este componente representa um gestor de memória que controla o uso de memória no sistema. É ele que permite alocar, desalocar e verificar o estado da memória. Além disso, inclui verificações para evitar alocações que excedam o tamanho máximo da memória ou deslocações que resultem em valores negativos de memória utilizada.

- Agendador de tarefas (task scheduler)

As tarefas em espera são guardadas numa estrutura chamada “*taskScheduler*”, ela implementa uma lógica de prioridade de tarefas, onde cada tarefa tem uma prioridade definida (alta prioridade ou baixa prioridade). As tarefas de alta prioridade são processadas antes das tarefas de baixa prioridade, isso é conseguido através do uso de duas filas: uma para tarefas de alta prioridade e outra para tarefas de baixa prioridade.

A estrutura de dados usada para as filas foi uma “*LinkedBlockingQueue*”, ela simplifica a implementação de operações seguras em ambientes concorrentes, evitando race conditions³, e permitindo que o código seja mais robusto e eficiente, pois, por usar uma estrutura baseada em *LinkedList* e não em *array*, permite com que não seja preciso fazer uma iteração caso seja removido ou adicionado um elemento em algum lugar que não seja na cauda. O uso de iterações em ambientes concorrentes é crítico, pois, podemos estar a iterar, e ao mesmo tempo em algum outro local, pode estar a ser removido ou adicionado um elemento, podendo gerar erros.

Um dos problemas da abordagem do agendador de tarefas, onde as tarefas de alta prioridade executam primeiro que as tarefas de baixa prioridade, é que podem existir problemas de fome (starvation), onde as tarefas de baixa prioridade vão ficar em constante espera, até não haver nenhuma tarefa de alta prioridade para ser executada. Para resolver este problema, foi criado um contador de fome (starvation count), onde a cada cinco tarefas de alta prioridade executadas, é obrigatório executar pelo menos uma tarefa de baixa prioridade, desta forma podemos reduzir esse problema não prejudicando as tarefas de alta prioridade que precisam de ser executadas o mais rapidamente possível.

³ Momento onde duas ou mais threads tentam aceder a um mesmo recurso ao mesmo tempo

4. Mecanismos de Sincronização Utilizados

No decorrer deste projeto foram utilizados vários mecanismos de sincronização, visto que cada condição de competição deve ser analisada cuidadosamente, e de acordo com essa análise, implementar o mecanismo de sincronização mais adequado.

O mecanismo de sincronização mais usado foi o uso da palavra-chave “*synchronized*”, tanto em métodos como em blocos de código, ela permite que quando um método ou um objeto declarado como *synchronized* é usado, apenas uma *thread* por vez pode executá-lo. Para poder saber quando utilizar essa palavra, foram definidas três questões que devem ser realizadas antes de as colocar, sendo elas:

O método ou objeto:

- Itera?
- Adiciona?
- Remove?

Se em alguma dessas questões a resposta for “sim”, então a palavra-chave “*synchronized*” deve ser utilizada, na figura a seguir segue um exemplo de um método onde a sua utilização pode ser vista.

```
protected synchronized void sendToMiddleware(Task task) {  
    try {  
        middleware.receiveSemaphore.acquire();  
    } catch (InterruptedException e) {  
        e.printStackTrace();  
    }  
  
    mem.deallocateMememory(task.getMemory());  
  
    synchronized (middleware.buffer) {  
        middleware.buffer.addRear(task);  
    }  
}
```

FIGURA 12 - EXEMPLO DA PALAVRA-CHAVE "SYNCRONIZED"

Na figura 12, está presente um método da classe *Kernel* que é chamado sempre que uma tarefa é concluída e pode ser feita uma ligação com a terra para devolver a resposta, essa ligação é feita com o *Middleware* como intermediário, onde ela é contida num buffer, antes de chegar à terra. A palavra *synchronized* é usada tanto no próprio método, para que apenas uma *thread* possa executar esse método por vez, como também dentro dele, para delimitar o uso do buffer do *Middleware*, assim temos a certeza de que quando a tarefa for para o buffer, apenas ela o pode modificar.

Outro dos mecanismos de sincronização foi a utilização de semáforos para poder controlar o acesso concorrente do buffer na classe “Middleware”, visto que o buffer está dividido em duas partes: as tarefas que vão ser enviadas para o *kernel*, e as tarefas que vão ser recebidas para serem enviadas para a aplicação, então, decidiu-se usar dois semáforos, o semáforo que envia, e o semáforo que recebe.

```
/**
 * Semáforo que controla a escrita no buffer, só pode ter 3 tasks para enviar
 * para o kernel no buffer
 */
protected Semaphore sendSemaphore;

/**
 * Semáforo que controla a leitura no buffer, só pode ter 2 tasks para enviar
 * para a aplicação no buffer
 */
protected Semaphore receiveSemaphore;
```

FIGURA 13 - SEMÁFOROS DO MIDDLEWARE

- **sendSemaphore:** Este semáforo controla a escrita no buffer, permitindo que até três tarefas sejam enviadas para o *kernel* simultaneamente. Quando uma tarefa é enviada, o semáforo é adquirido (sendSemaphore.acquire()), bloqueando outras *threads*, caso não haja mais permissões, até que uma permissão seja liberada (através de sendSemaphore.release());
- **receiveSemaphore:** Este semáforo controla a leitura no buffer, permitindo que até duas tarefas sejam recebidas pela aplicação simultaneamente. Quando uma tarefa é recebida, o semáforo é libertado (receiveSemaphore.release()), indicando que uma posição no buffer foi libertada para que outra tarefa possa ser.

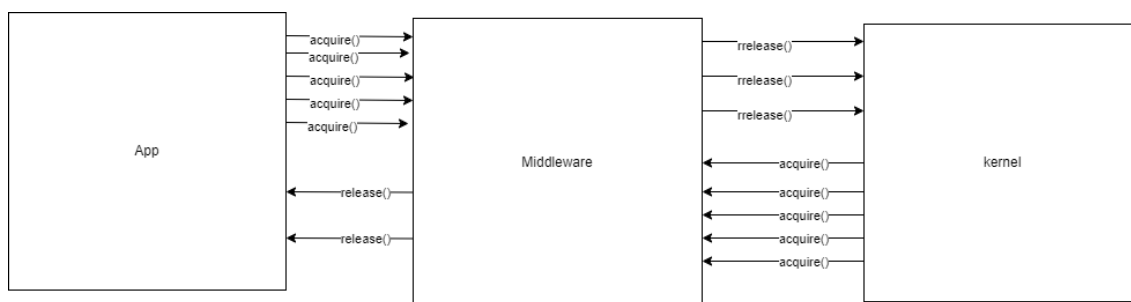


FIGURA 14 - CONCEÇÃO DE COMO OS SEMÁFOROS FUNCIONAM

5. Comunicação entre módulos

Toda a comunicação entre os módulos foi feita de acordo com a imagem a seguir, onde o CPU e a MEM apenas conseguiam interagir com o kernel, o kernel conseguia interagir entre a MEM, o CPU e o Middleware, e o Middleware apenas conseguia comunicar entre a aplicação e o kernel.

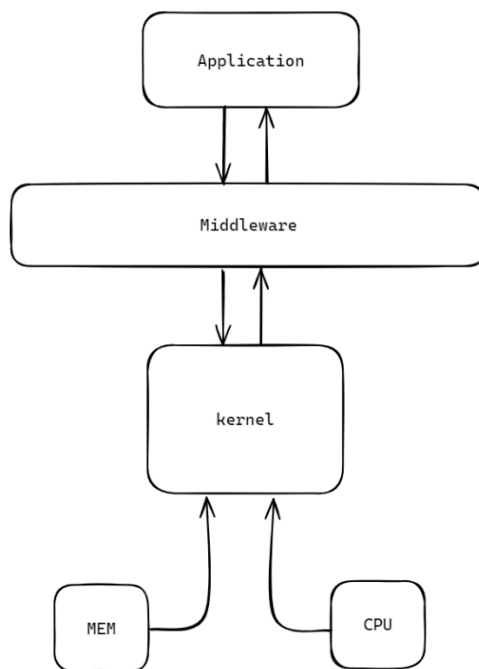


FIGURA 15 - DIAGRAMA GERAL DA COMUNICAÇÃO ENTRE OS MÓDULOS

- Comunicação aplicação e Middleware

A comunicação entre a aplicação e o Middleware foi feita a partir de um buffer, onde a aplicação tinha referência do Middleware, e chamava o método *"middleware.send()"* para enviar tarefas para o mesmo.

- Comunicação Middleware e kernel

A comunicação entre o Middleware e o Kernel ocorre de uma forma muito similar entre a aplicação e o Middleware, onde as tarefas têm de passar pelo buffer do Middleware primeiro.

- Comunicação Kernel e MEM

A comunicação entre o kernel e a MEM é feita a partir dos métodos implementados na classe MEM para a alocação e desalocação da memória, onde o kernel os chama sempre que necessário a partir de uma instância que o mesmo tem com a da classe MEM.

- Comunicação Kernel e CPU

A comunicação entre o kernel e o CPU ocorre de forma muito semelhante ao da MEM com o Kernel, onde o kernel guarda a referência da CPU, e sempre que precisar, pode chamar os métodos do CPU.

6. Funcionalidades não implementadas

Apesar de já terem sido implementadas muitas funcionalidades neste projeto, ainda existem muitas outras que foram pedidas, mas que por alguma razão, não conseguiram ser implementadas, sendo elas:

- **Armazenamento e Manipulação de Dados:** A MEM pode oferecer recursos para armazenar e manipular dados necessários para as operações do satélite.
- **Comunicação entre Tarefas:** Utilizar protocolos de comunicação no Middleware, de modo a facilitar a comunicação entre diferentes partes do sistema. Fazer com que gere mensagens e objetos para definir serviços de comunicação, um dos exemplos dados, foi a disponibilização da localização.

Conclusão

Achamos que a realização deste trabalho foi bem concebida tendo em conta o que nos foi proposto e também as nossas próprias exigências para o trabalho, entre elas, a constante busca por um programa funcional e coeso com aquilo que um sistema operativo é.

Ao longo da realização deste trabalho, confrontamo-nos com diversos desafios, sendo os principais: a identificação de situações de competição, que exigiram uma análise cuidadosa para compreender a natureza dos problemas e, subsequentemente, implementar soluções eficazes; a identificação do método mais apropriado de sinalização, considerando as complexidades inerentes ao problema em questão. Esta fase do projeto proporcionou uma oportunidade única de aplicar os conceitos teóricos aprendidos em sala de aula a um contexto prático e desafiador.

Em suma, este projeto desempenhou um papel crucial no nosso desenvolvimento académico e profissional. A abordagem prática contribuiu para sedimentar nosso conhecimento. A resolução de desafios específicos, como a competição por recursos e a escolha de métodos de sinalização, fortaleceu a nossa capacidade de análise e solução de problemas, que são competências essenciais na área de Sistemas Operativos.