
UNIVERSIDADE FEDERAL DA BAHIA

**Vitor Mattedi Carvalho - 216115242**

MATA82 – Programação em Tempo Real

# Simulador de um escalonador de tarefas para um sistema com múltiplos processadores

*Docente: Paul Denis Etienne Regnier*

**SALVADOR/BA**

**JULHO/2025**

# 1.Introdução

## 1.1 Escopo do Documento

Este documento é um relatório do processo de produção do projeto RTSim, um simulador de escalador de tempo discreto. Ele, no entanto, não é uma documentação do base de código, não contém instruções de uso e compilação em executável nem a documentação de outros sistemas que foram desenvolvidos para este projeto (como a psudo console engine). Estas informações estão disponíveis no github.

O projeto

O RTsim é um simulador de um escalonador em tempo discreto que surgiu da proposta de trabalho de programação em tempo real – MATA82 e começou a ser desenvolvido em maio de 2025.

A proposta era de fazer um simulador de tempo discreto para um escalonador de um sistema com um ou mais processadores.

Ele foi implementado em JavaScript, e requer o Node.JS para funcionar. Esta decisão de utilizar o JavaScript não é muito usual para este tipo de simulador, entretanto ele foi escolhido pois eu gostaria de disponibilizar o projeto na web também e com tal objetivo, ao implementar com JavaScript e corretamente modularizado, fazer a portabilidade para um ambiente web passa a ser relativamente mais simples, principalmente por causa de tecnologias como Xterm.js que simula um console VTI em uma página web. O resultado é que, hoje, é possível testar a aplicação neste site. Devido à natureza do simulador, desde visualização até os inputs necessários, o site não funciona muito bem com dispositivos mobile nem com dispositivos sem um teclado físico.

## 1.2 Escopo e Limitações do projeto

O projeto consegue simular um sistema multiprocessado e um conjunto de tarefas que podem ter prioridade, deadline, serem periódicas e chegam em tempos diferentes ao sistema.

Entretanto devido em grande parte a própria natureza de serem tarefas simuladas genéricas, não é possível simular o acesso à recurso compartilhado entre tarefas e, portanto, não é possível visualizar algoritmos de inversão de prioridade.

Além disso, o projeto tem um limite de tempos a serem simulados devido a ter que guardar, em memória, um snapshot do sistema a cada instante de tempo $\Delta t$ para poder visualizar o que aconteceu no sistema ao longo e o estado em qualquer t arbitrário. Entretanto, este limite é bem alto (Chegou a 16k na versão para web) e nunca foi testado até tão longe na versão para desktop. Possíveis soluções envolvem a utilização de memória externa e otimização de memória do projeto em si.

Por fim, o simulador simula *time slices* discretamente e assume que o tempo de sobrecarga em cada *time slice* é desprezível. Desta forma, todos os algoritmos disponíveis são preemptivos (apesar que alguns tradicionalmente não são como por exemplo hrrn).

# 2. O Escalonador

### 2.1 Funcionamento

O escalonador é responsável por atribuir as tarefas que estão disponíveis no sistema aos processadores do sistema. Isto é simulado ao chamar o método *tick* dentro do escalonador.

Este método é onde o escalonador, em ordem:

- Checa se novas tarefas chegaram ao processador neste instante de tempo.
- Verifica se tarefas terminaram ou foram finalizadas.
- Usa o algoritmo (FCFS, SJF, EDF, etc) escolhido para ordenar as tarefas a serem executadas neste *time slice*.
- Designa cada tarefa para cada CPU disponível.
- As tarefas selecionadas são por ordem do algoritmo a menos que: duas ou mais tarefas tem execução exclusiva no mesmo processador, neste caso, somente a tarefa mais bem colocada será executada
- Caso esteja habilitado, aplica um algoritmo de minimização de *core migration*, tentando melhor alocar as tarefas aos processadores previamente utilizados pois o custo é maior que um custo normal de troca de contexto.
- Executa as tarefas escalonadas.

### 2.2 Algoritmos de escalonamento

Para o projeto foi desenhado uma arquitetura onde o algoritmo utilizado para escalonar os processos é desacoplado do escalonador em si, facilitando a implementação de novos algoritmos. Cada algoritmo é implementado herdando a classe *SchedulerAlgorithms* e deve implementar, ao menos a função *sortTasks* que recebe as tarefas a serem organizadas e o tempo atual. Este método deve retornar um array com as tarefas passadas organizada em ordem de prioridade de acordo com o algoritmo.

As tarefas que serão executadas são as N primeiras (amenos no caso em que duas destas tenham execução única em um core).

Além disso, ao atualizar as classes *AlgoFactory* e *AlgorithmModels*, com novo algoritmo, o próprio simulador disponibiliza nas configurações disponíveis o novo algoritmo.

### 2.3 Algoritmos criados

Foram implementados 7 algoritmos de escalonamento diferentes. Os clássicos EDF, SJF, FCFS, RR e Por Prioridade dispensam introdução, mas além desses, temos também o PriorityRR e o HRRN.

Como descrito em 1.3 o escalonador simula cada time slice e aciona o algoritmo a cada instante para que ele selecione as tarefas a serem executadas, o que faz sentido em sistemas de tempo real onde tarefas mais importantes chegam ao processador constantemente.

O algoritmo HRRN tradicionalmente não é preemptivo, mas foi escolhido por utilizar ele de forma preemptiva onde a cada time slice ele calcula o *response ratio* de cada tarefa.

O algoritmo SJF, na realidade é um SRTF, ele ordena por tempo restante e não pelo *burst time*.

Por fim temos o PriorityRR, que é uma variação do Priority onde em caso de tarefas com a mesma prioridade, ele trabalha como o round robin, alternando entre as tarefas de mesma prioridade. Ele é utilizado em em sistemas operacionais de sistemas embarcados como FreeRTOS.

### 2.4 Context Switch & Core Migration

Cada vez que um processador precisa carregar, apagar ou trocar o contexto da tarefa atual isto, na vida real, tem um custo, e por isso essa estatística é mantida para cada tempo.

*Core migration* é quando uma tarefa estava sendo executada em um processador e passa a ser executada em outro, neste caso o custo de troca de contexto é ainda maior além disso tem várias outras penalidades como possíveis cache miss a menos que o cache seja compartilhado.

Desta forma, foi implementado um algoritmo de redução de core migration onde tentamos otimizar as tarefas escolhidas para os cores em que elas estavam sendo executadas e não de forma aleatória, entretanto caso não seja possível, a tarefa ainda é executa pois não devemos executar uma tarefa com menos prioridade apenas porque o custo é maior. Nestes casos ocorrerá o *core migration*.

Para reduzir estes *o core migrations*, distribuímos *os cores* que não estão sendo ocupados por tarefas com uma *hard task affinity* (pinToCore) pelos *soft task affinity* de cada tarefa, priorizando as tarefas com mais prioridade, exemplo se a tarefa t1(pri = 5) e tarefa t2 (pri = 4) ambas vão ser executadas e ambas têm apenas soft task afinnity e o core ao qual elas têm afinidade está disponível (nenhuma tarefa está pinada nele), a tarefa t1 executara sem *core migration* enquanto a tarefa t2 terá que sofrer o *core migration*.

### 2.5 Deadline & Período

As tarefas neste simulador podem ter deadline e serem periódicas.

Não há um reforço de que o período seja maior que o burst time, ou até deadline, fica a cargo do usuário simular uma tarefa com as características que ele queira incluindo criar uma task sem deadline que é periódica, apesar de não fazer muito sentido. Entretanto, o sistema permite adicionar noise ao sistema onde tarefas aleatórias podem chegar no processador durante a simulação (configurável) porém elas não podem ser periódicas.

As tarefas são identificadas por um ID assinalado pelo escalonador, entretanto tarefas periódicas são identificadas por ID#N onde o ID é comum a todas e o N é a instancia da tarefa.

Quando uma tarefa periódica existe, uma outra tarefa idêntica exceto pelo ID chega uma única vez no processador em *t* = *arrival time* + *period*. Como essa nova tarefa é idêntica, ela também é periódica e, portanto, mantém o período dela. Apesar de ser idêntica ela é outra tarefa e é tratada como tal pelos algoritmos de escalonamento e visualmente, como elas são a mesma tarefa, elas têm a mesma cor e isso pode gerar confusão sobre o que aconteceu. Nestes casos é necessário observar a tabela de tarefas no sistema que mostra o estado de cada tarefa naquele time slice.

Por fim, como o contexto é de sistemas de tempo real, tarefas que estouram o deadline, periódicas ou não, são marcadas como falhada e saem da lista de tarefas prontas.

# 3. O Simulador

### 3.1 Executando:

Para executar o programa, pode-se executar o executável disponível aqui diretamente e sem pre-requisitos, ou utilizando node.js, após clonar o repositório do programa, executar node main.js.

Em ambos os casos, é possível passar argumentos como -f <filename.json> e -s. O primeiro carrega uma lista de tarefas de um arquivo json como os disponíveis para teste tasks.json e fail.json. O segundo inicia a simulação diretamente, sem passar pelos menus.

### 3.2 Interação

Apesar de ser um programa para console ele é visual e interativo a partir de eventos. Na maioria das telas (menu, task manager, simulação) a interação é feita via as setas para navegação e enter ou espaço para selecionar, mas em algumas, teclas especificas fazem coisas especificas. Em geral há uma descrição do que cada tecla faz em cada tela.

### 3.3 Fluxo do programa

O programa funciona da seguinte forma: o usuário configura as tarefas iniciais a serem simuladas, configura as características do sistema como algoritmo, uso da redução de *core migration*, número de processadores (estes dois podem ser feito de forma e ordem independente) e depois começa a simulação com tais tarefas e configurações.

Caso um arquivo seja passado pela CLI, as tasks iniciais são as de tais arquivos. Caso não tenha sido passado nenhum arquivo, o programa verifica se há um arquivo tasks.json, se ele existe pergunta se o usuário quer carregá-los. Caso nenhum dos dois aconteçam, as tarefas iniciais são 10 tarefas aleatórias que não são periódicas.

Foram disponíveis dois arquivos de testes fail.json e pass.json, dois conjuntos de 4 tarefas) que são respectivamente (não escalonável e escalonável de acordo com Liu & Layland para 1 core. Simulando eles podem ver que o não escalonável precisa de mais cores

do que tarefas (para ser resistente a chegadas de tarefas aleatória) e mesmo o pass pode ter problemas (testado em 2 cores) caso cheguem tarefas mais prioritárias (2 ou mais pois ele é escalonável em 1 core) ou pode não atender uma tarefa aleatória menos prioritária

Na web não é possível passar um arquivo (por enquanto).

### 3.4 Configurações

O programa salva as configurações preferidas em um arquivo systemconfig.json, editar este arquivo mudara diretamente as configurações (em novas sessões) entretanto deve ser atentando os valores de máximo e mínimo de cada configuração.

As configurações podem ser modificadas na tela system config onde pode-se modificar:

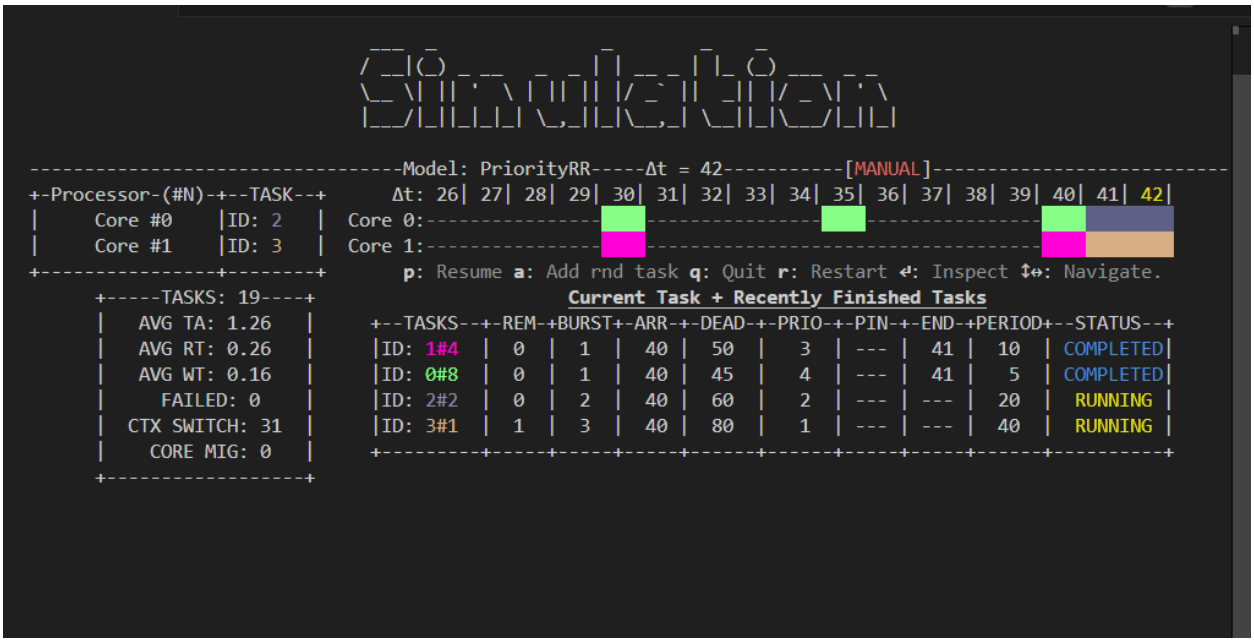
- O número de processadores
- O algoritmo de escalonamento desejado
- A chance de aparecerem novas tarefas aleatórias por time slice (uma forma de introduzir ruido a um sistema estável por exemplo e simular funcionamento esporádico de um sistema para ver se um conjunto de tarefas periódicas ainda assim são escalonáveis)
- Habilitar que estas tarefas possam chegar associadas a um core
- Máximo burst time dessas tarefas
- Permitir que elas cheguem com um deadline menor que o burst time (com certeza haverá falha)
- O tempo real de simulação (tempo para avançar automaticamente para o próximo time slice podendo ser pausado/continuado ou feito de forma manual na simulação)
- Ativar/desativar o algarismo de redução de core migration

Ao modificar alguma desta configurações o sistema as utilizara enquanto estiver rodando, porém, salvar (Ctrl + s) nesta tela, ela salva as configurações atuais para o arquivo de configuração e, desta forma, utilizará tais configurações por padrão na próxima vez que o programa for inicializado (exceto na web).

## 4. Tela de Simulação

### 4.1 Funcionamento

Nesta tela vamos de fato simular a passagem do tempo e teremos na tela o resultado da simulação para cada tempo discreto, podemos então ver o estado do sistema em cada time slice como visto abaixo:




### 4.2 Interpretação

Para cada time slice podemos ver a linha do tempo do que aconteceu nos últimos n time slices em cada core, o que cada core estava fazendo naquele tempo e o estado das tarefas do ponto de vista do escalonador naquele instante. Além disso temos uma tabela de estatísticas, também calculada para aquele instante de tempo.

Em cada delta t, estamos vendo o que aconteceu entre t e t.9999..., desta forma na linha do tempo temos, o número representa o instante à esquerda e antes do próximo instantes, por isso vemos um remaining duration = 0 (foi executado neste *time slice* e estará finalizado no próximo).

### 4.3 Interação

Por padrão o tempo avança sozinho a cada x milissegundos determinados pela configuração, porém o usuário pode utilizar as setas para navegar entres os instantes (pausando esse avanço automático), podendo assim visualizar o que aconteceu passo a passo e qual era o estado do sistema inteiro naquele instante.

Ao manualmente avançar a mais do que o tempo atual, também avançará o tempo.

Além disso o usuário pode apertar **r** para reiniciar a simulação, **q** para ir pro menu principal, **a** para manualmente adicionar uma nova tarefa aleatória (shift + a adiciona 10 de vez) e **enter** para escolher qual tabela ele quer ver, estando disponíveis (ordenada pelo algoritmo utilizado) a tabela de tarefas completa, as tarefas atuais (espera ou executando) as tarefas que já terminaram e variações para filtrar apenas tarefas recentes. Além disso é possível pausar e resumir o avanço automático a cada

É possível saber se estamos no tempo atual pela tag LIVE e pelo *time slice* atual estar em verde e não amarelo. Ao adicionar novas tarefas elas entram no simulador no tempo atual mesmo que o usuário esteja observando um tempo passado e no próximo *time slice* elas já estarão prontas e podem ser escalonadas.

## 5. Considerações Finais

O projeto acabou tendo proporções maiores do que eu incialmente previ e muitas das coisas que eu desenvolvi aqui fogem bastante do escopo da disciplina.

Apesar de não ser perfeito, principalmente para sistemas com múltiplos processadores, o simulador representa muito bem um sistema single core de tempo real (pelo menos de acordo com meus estudos nesta matéria e em ENGD33 STR para sistemas embarcados).

A visualização do projeto foi feita a partir do ponto de vista dos processadores e não das tarefas que não tenho certeza se é comum.

Por fim, eu pretendo continuar desenvolvendo este projeto e uma das coisas a serem implementadas é, justamente, possibilitar a visualização com um diagrama de Gantt. Além disso, pretendo melhorar a parte técnica do software reduzindo custos de memória e aprimorar a fidedignidade para sistemas reais multiprocessados.