UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE SISTEMAS ELETRÔNICOS

BRUNO LUIZ DEMARCHI
GUILHERME TURATTO

# TINY-OS

Trabalho para a disciplina de testes e verificação formal de software

Joinville
2023

# 1 INTRODUÇÃO

Cada vez mais as empresas, instituições e normas cobram a qualidade de software de um produto, a fim de garantir os requisitos de projeto, principalmente em aplicações de segurança, onde qualquer erro de programação pode ocasionar a perda de vidas, como em automóveis e aeronaves, por exemplo.

Para garantir a qualidade de um software, existem várias teorias e metodologias que podem ser utilizadas antes mesmo de começar o desenvolvimento desse software. Técnicas como análise de riscos e confiabilidade de um sistemas e árvore de falhas podem ser muito úteis para tentar estimar a probabilidade de falha de um sistema e o que irá ocorrer em caso de falhas. Entretanto, essas técnicas não ajudam diretamente no desenvolvimento do software do sistema.

Para desenvolver um software que apresenta menor chance de erros, teorias como a de verificação de modelos podem ajudar. Entretanto, essas teorias dependem do desenvolvedor criar um modelo que represente e atenda os requisitos do projeto. Isso normalmente é feito utilizando máquinas de estado, sejam elas utilizando a teoria de autômatos ou até mesmo redes de Petri. Existem diversas ferramentas que podem auxiliar na criação desses modelos, como o Nadzoru (UDESC, 2023), Uppaal, Tapaal, etc.

Com um modelo pronto, é possível elaborar algumas proposições baseadas na teoria de verificação de modelo e com isso garantir requisitos de software, como por exemplo, a ausência de *deadlocks*.

Este trabalho tem como objetivo construir o modelo de um sistema operacional simples, mostrado na figura 1, em dois software, Uppaal e Tapaal, criando algumas lógicas proposicionais para verificar o modelo com a ajuda do software e assim garantir que a máquina de estados do sistema está funcional.

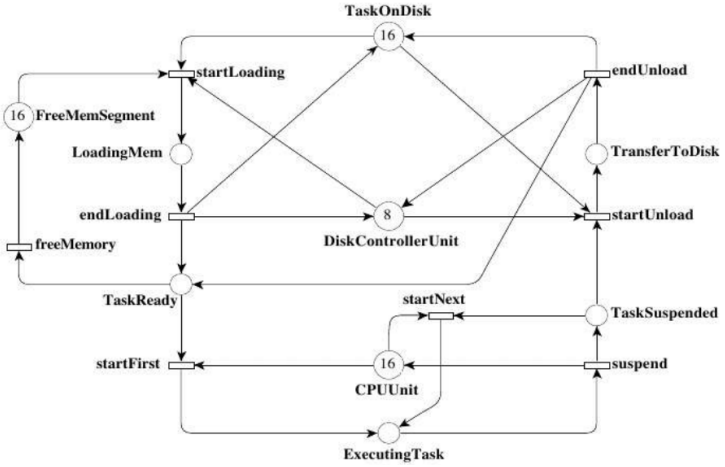


Figura 1

# 2 DESENVOLVIMENTO

Neste capítulo serão apresentadas máquinas de estado equivalentes ao sistema da figura 1 nos programas Uppaal e Tapaal. Ao final do capítulo, serão apresentadas as proposições geradas e também os resultados obtidos.

## 2.1 UPPAAL

A modelagem do sistema no Uppaal consiste em três templates de processos: tarefas, controladores de disco e CPUs. O modelo dos segmentos de memória foi omitido, pois, ao limitar o problema considerando M = T = MT, sempre haverá um segmento de memória disponível para cada tarefa existente.

As tarefas são descritas conforme os estados da Figura 2, sincronizando as transições de acordo com os estados e disponibilidade dos controladores de disco e CPUs. Variáveis globais são utilizadas para controlar o número de tarefas no disco e na fila de execução. A fila de execução também é definida globalmente, sendo operada pelas funções *enqueue(), dequeue()* e *front()*. Cada tarefa insere seu *id* na fila ao estar no estado *TaskReady* ou *TaskSuspended.*

Para modelar as transições que retornam uma tarefa para o disco, foi considerado um tempo de execução mínimo de 6 ciclos de clock (TET - Task Execution Time), necessário para que a tarefa termine sua execução. O ciclo de clock é informado pela CPU através dos canais *clk_tick[id],* existindo um canal para cada tarefa do sistema. Ao finalizar a execução, a tarefa informa os controladores de disco através do canal *T_finished***,** e aguarda as sinalizações do controlador para ser transferida para o disco ou retornar para a memória, em *TaskReady.*

Os controladores de disco, exibidos na Figura 3, além de carregar e descarregar as tarefas do disco para a memória, contam com uma transição probabilística para definir se uma tarefa finalizada irá retornar para o disco ou para a memória.

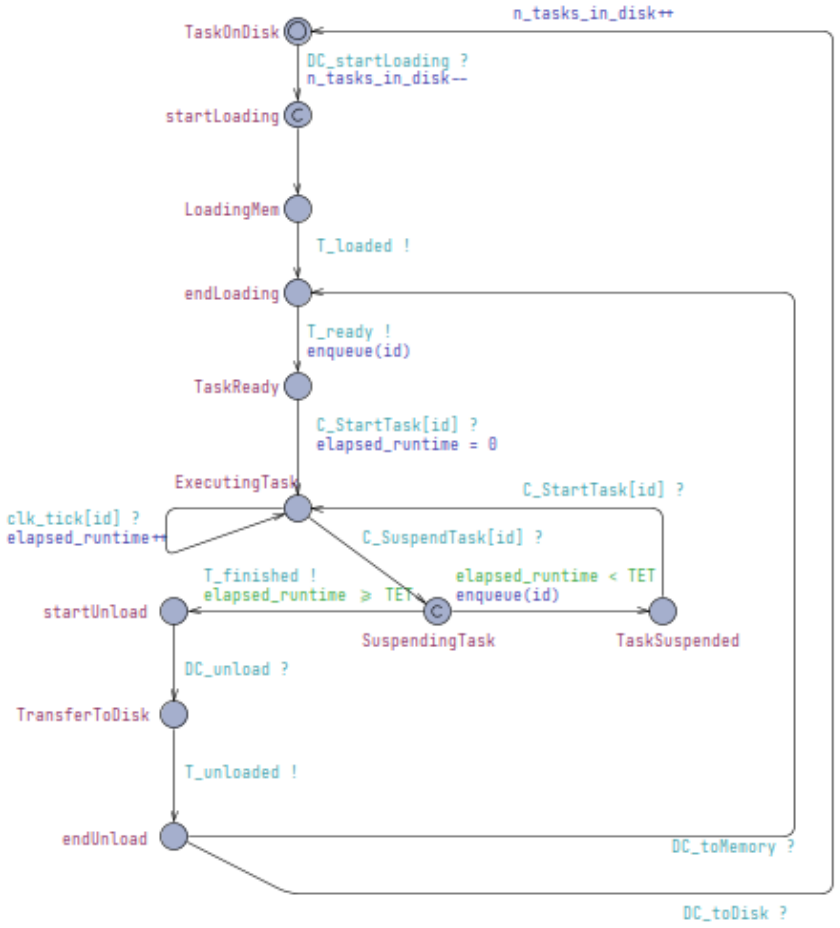


Figura 2

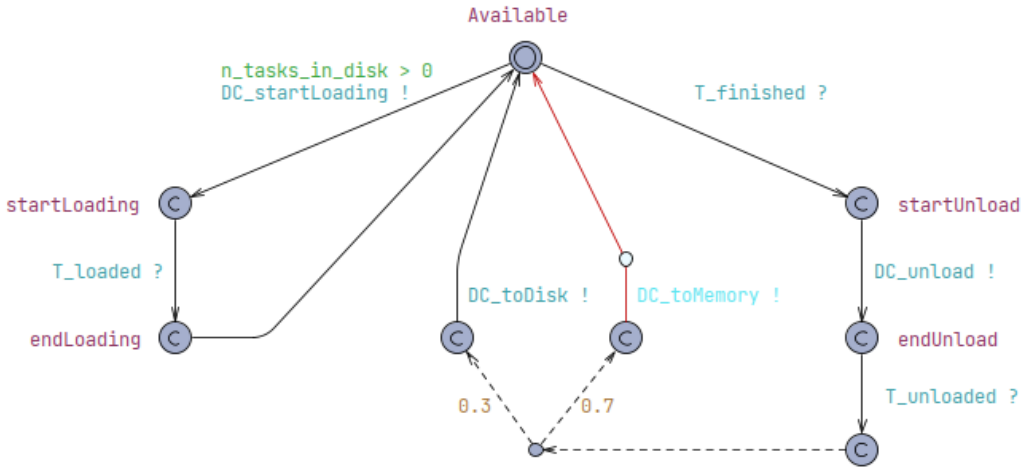


Figura 3

A modelagem das CPUs conta com a sinalização do clock, além de um quantum de execução para cada tarefa, estabelecido em 2 unidades de clock. Ao atingir o quantum estabelecido, a CPU suspende a tarefa em execução e inicia a próxima da fila. A Figura 4 apresenta a modelagem das CPUs.

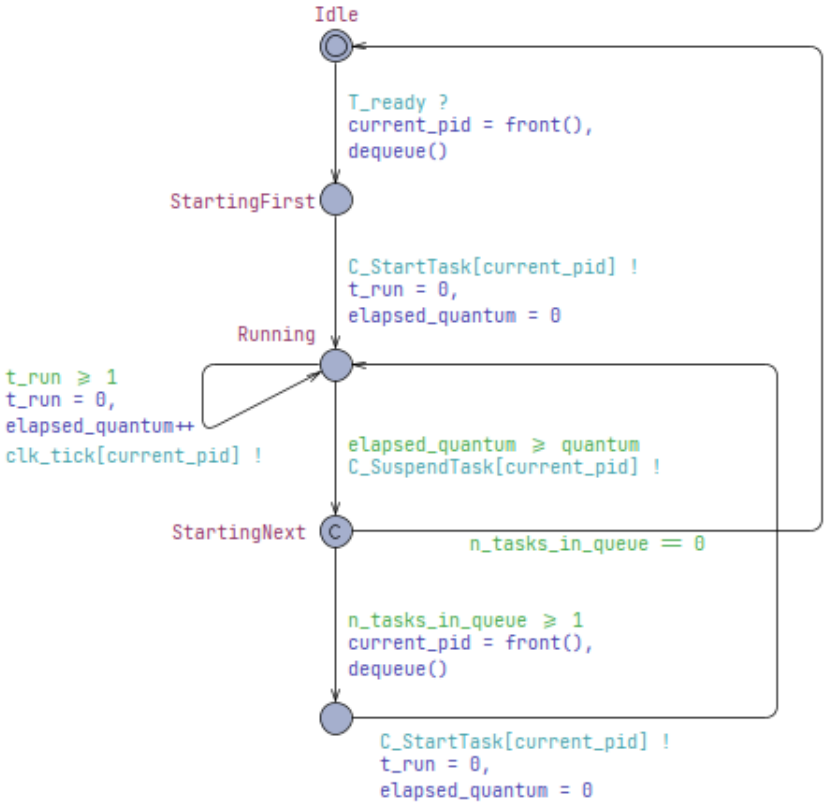


Figura 4

### 2.1.1 Verificação

A verificação de propriedades no Uppaal não prosseguiu devido ao grande número de combinações de estado e possíveis traces de execução, demandando tempo de processamento e memória. Ao tentar verificar a inexistência de deadlocks, foram utilizados mais de 4GB de RAM ao longo de 10 minutos de execução (Figura 5).

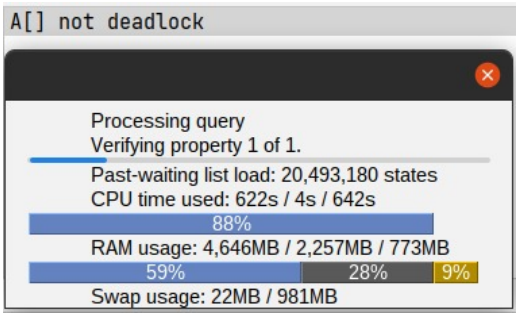


Figura 5

## 2.2 TAPAAL

Para tentar resolver as proposições que não foram possíveis com o Uppaal, o projeto foi desenvolvido também no programa Tapaal.

A Figura 6  mostra o projeto do Tiny-OS implementado no Tapaal.

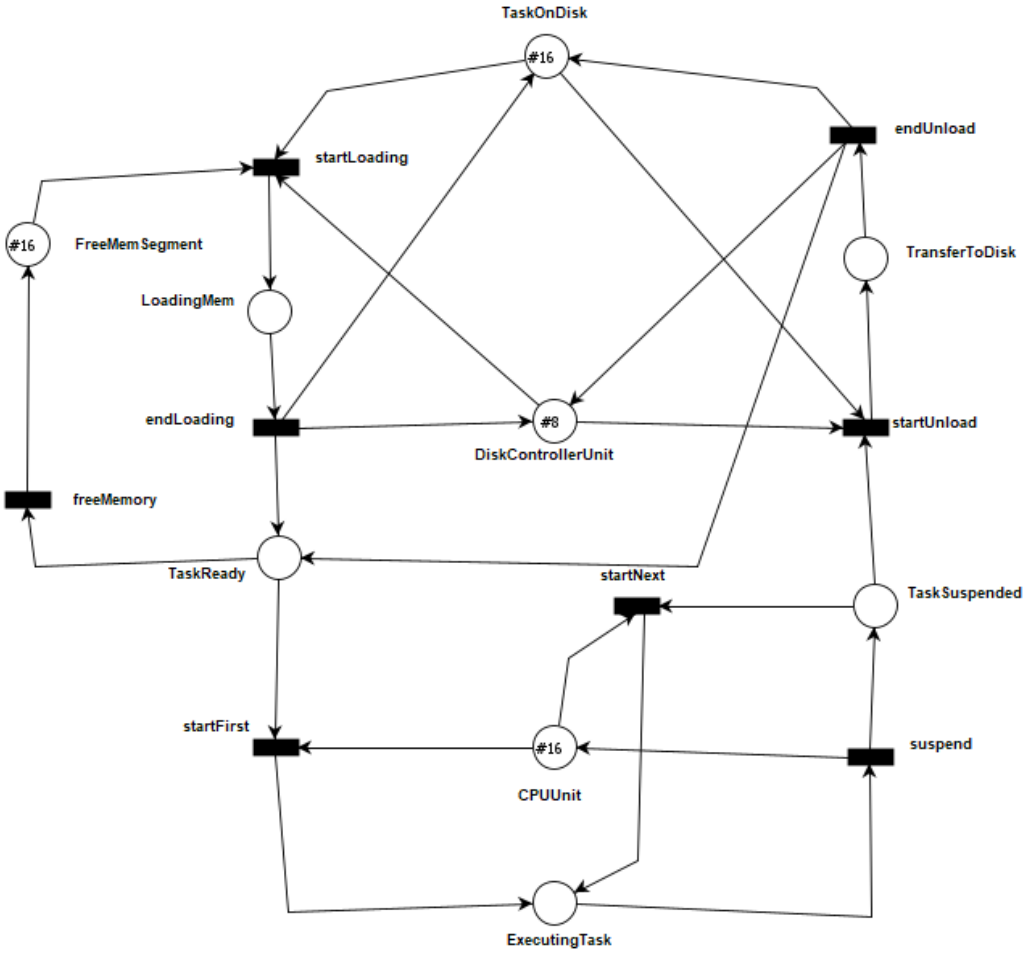


Figura 6

O Tapaal foi escolhido por permitir naturalmente modelar usando redes de Petri, que é a mesma rede que foi proposto o problema. Ou seja, foi bem mais simples modelar o problema nesse programa.

Esse programa possui o modo de simulação, onde é possível gerar transições aleatórias ou escolher quais transições irão acontecer, como é mostrado na Figura 7.

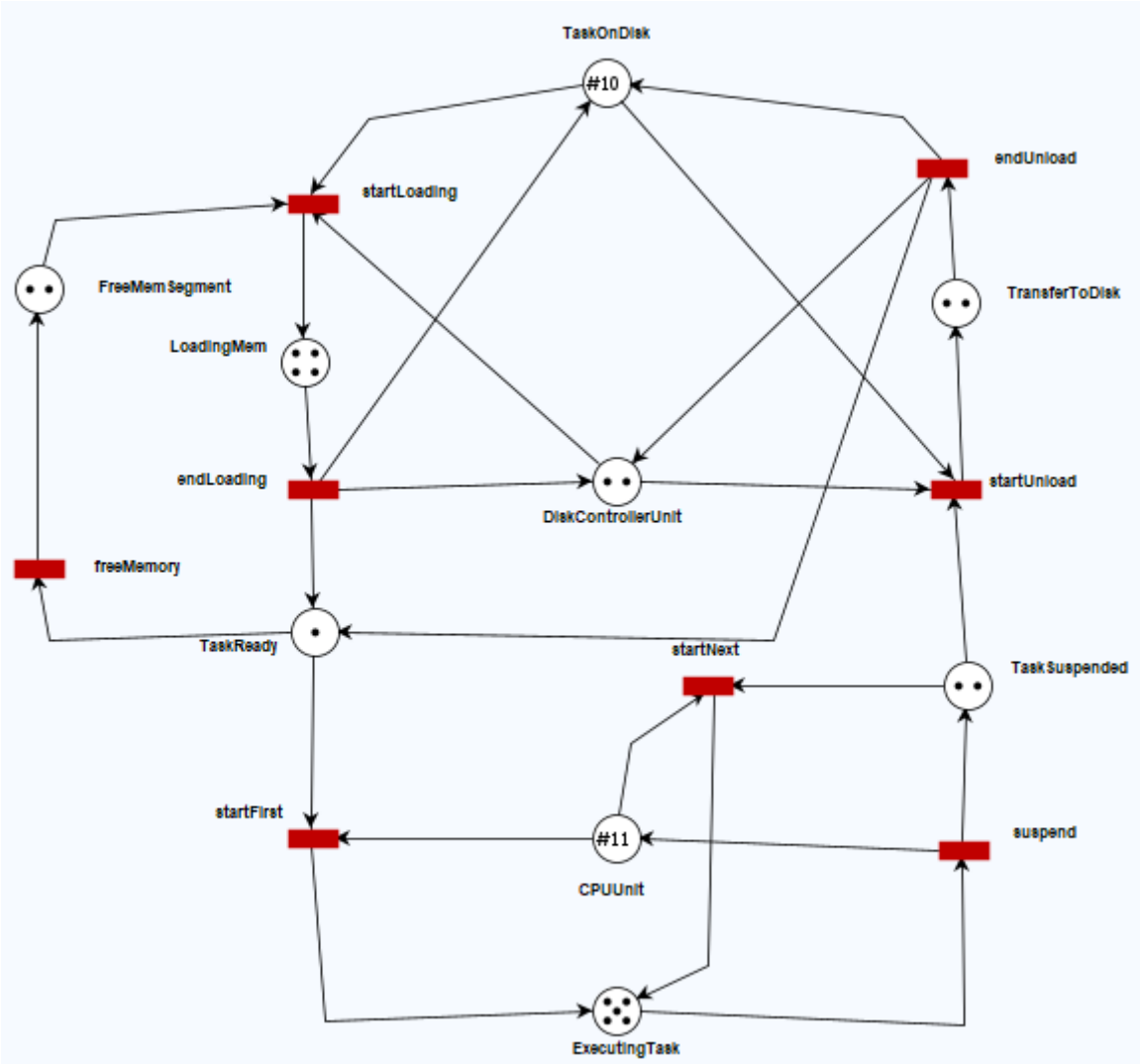


Figura 7

## 2.3 PROPOSIÇÕES

Foram criadas algumas proposições a fim de verificar se o sistema irá funcionar corretamente.

1. Não há deadlocks no sistema.
    a. AG (! deadlock) - Sempre, globalmente, não há deadlock.
2. Em qualquer estado do sistema, todas as transições podem eventualmente acontecer.
    a. AG ((EF tinyos.startLoading) and (EF tinyos.endLoading) and (EF tinyos.startUnload) and (EF tinyos.endUnload) and (EF tinyos.freeMemory) and (EF tinyos.startFirst) and (EF tinyos.suspend) and (EF tinyos.startNext))
3. Eventualmente, não haverão tarefas no disco.
    a. EF tinyos.TaskOnDisk = 0
4. Eventualmente, todas as tarefas estarão sendo executadas simultaneamente.
    a. EF tinyos.CPUUnit = 0

5. Eventualmente, haverão mais tarefas prontas do que o número de CPUs
    a. EF (tinyos.CPUUnit = 0 and tinyos.TaskReady != 0)

Resultado das proposições:

1. Verdadeiro, ou seja, não há deadlocks
2. Verdadeiro, ou seja, não há nenhum estado bloqueante
3. Falso
4. Verdadeiro, ou seja, todas as CPUs estão em uso
5. Falso, ou seja, o gargalo do sistema não é o número de CPUs.

# 3 CONCLUSÃO

A utilização de modelos permite o uso de teorias de verificação e validação que não seriam possíveis em projetos onde o software é feito sem muito projeto, testes e validações o que permite corrigir possíveis erros de lógica no software antes mesmo de começar a escrever o código. Isso foi possível verificar na prática durante o trabalho, onde, utilizando algumas proposições de safety, foi possível notar e corrigir diversos problemas na máquina de estado durante sua construção.

Apesar de tudo, ainda é difícil utilizar as técnicas comentadas durante o trabalho em alguns projetos de algumas empresas, visto que muitas empresas valorizam mais o "*time to market"* do que a qualidade de software, não se importando em lançar produtos com alguns bugs que podem ser corrigidos futuramente. Essa estratégia, apesar de parecer ser vantajosa no tempo de lançamento de produtos, muitas vezes faz com que o desenvolvimento/evolução do produto demore muito mais do que se o projeto fosse desenvolvido de maneira correta desde o início.

# REFERÊNCIAS

UDESC. **NADZORU**. Disponível em: https://www.udesc.br/cct/gasr/softwares/nadzoru. Acesso em: 05 set. 2023.