Diferentes Abordagens para Sistemas de Tempo Real

Fundamentos dos Sistemas de Tempo Real

Rômulo Silva de Oliveira

eBook Kindle, 2018

www.romulosilvadeoliveira.eng.br/livrotemporeal

Outubro/2018

Necessidade de Diferentes Abordagens 1/2

- Mercado para tempo real é amplo

- Sistemas de tempo real variam enormemente
  - Sistema de emergência em usina petroquímica
  - Controle de temperatura do freezer
  - Videogame

- **Diferentes abordagens são necessárias**

Necessidade de Diferentes Abordagens 2/2

- Diferentes tipos de criticalidade
  - Elevada criticalidade, exige certificação, ex: aviônica
  - Crítico, porém testes são suficientes, ex: relé de proteção elétrica
  - Não é crítico, tolera perdas eventuais de deadlines, ex: videogame

- Diferentes tipos de sistema operacional
  - Não usa SO, hardware simples, aplicação pequena (laço + trat. interrupções)
  - Microkernel, hardware médio sem MMU, aplicação multitarefa
  - Linux, hardware complexo, aplicação grande requer serviços de SO

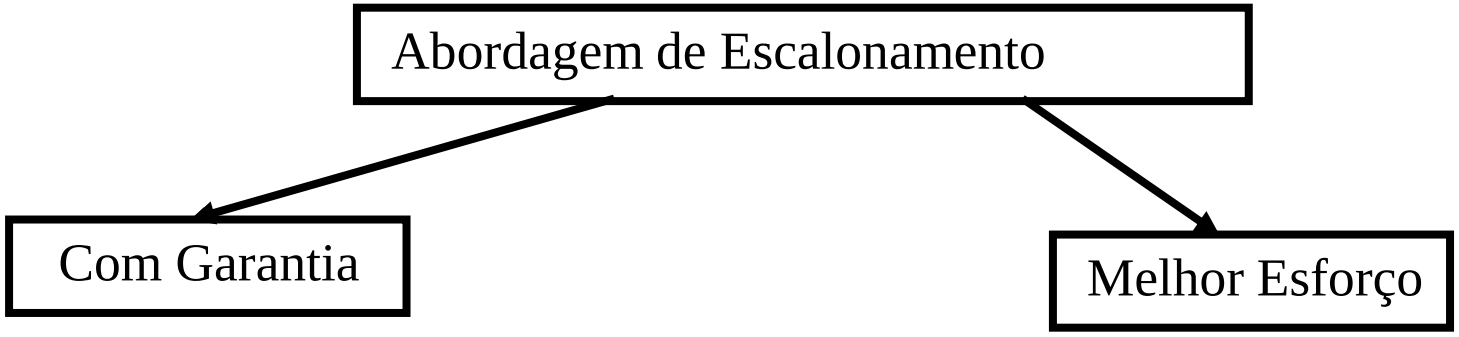
- Diferentes tipos de código fonte
  - Várias linguagens de programação
  - Design do software mais simples ou mais complexo
  - Código legado, bibliotecas compradas

Classificação das Abordagens na Academia 1/9

- Existem duas grandes abordagens de escalonamento para sistemas de tempo real na perspectiva acadêmica
  - Matemática de escalonamento

- **Sistemas com garantia (hard real-time)**

- **Sistemas com melhor esforço (soft real-time)**



Classificação Acadêmica: Com Garantia 2/9

- Deadlines são garantidos na construção do software
- Previsibilidade determinista
- Análise feita antes da execução
  - Carga precisa ser limitada e conhecida em projeto
    (Hipótese de Carga)
  - É Suposto um limite para faltas
    (Hipótese de Faltas)

- Para dar garantia precisa considerar o pior caso:
  - Do comportamento do software (fluxos de execução)
  - Do comportamento do hardware (tempos das instruções)

Classificação Acadêmica: Com Garantia 3/9

- Necessário conhecer o comportamento do programa no pior caso

- Isto significa
  - Pior fluxo de controle para cada tarefa (if, while)
  - Piores dados de entrada
  - Pior cenário de sincronização entre tarefas (exclusão mútua, etc)
  - Pior combinação de eventos externos (interrupções, sensores, etc)
  - Pior tudo

Classificação Acadêmica: Com Garantia 4/9

- Necessário conhecer o comportamento do hardware no pior caso

- Isto geralmente significa
  - Pior combinação de eventos externos (interrupções, sensores, etc)
  - Determinar os estados da memória cache
  - Determinar os estados do pipeline
  - Determinar o comportamento dos barramentos
  - Determinar o comportamento temporal seguro do hardware em relação ao pior caminho do software
  - Compor os estados em uma análise de pior caso
  - Sempre de forma segura (pessimista)

- As vezes o pior caso local não leva ao pior caso global

Classificação Acadêmica: Com Garantia 5/9

- Análise usualmente dividida em duas etapas

- **Tempo de Computação C**
  - Quanto tempo esta tarefa de software levaria para executar se estivesse sozinha no computador (única tarefa, nenhuma interrupção) ?
- Para garantia é necessário o WCET (Worst-Case Execution Time)

- **Tempo de Resposta R**
  - Quanto tempo esta tarefa de software leva para executar, considerando ela própria e todas as demais atividades do sistema ?
- Para garantia é necessário o WCRT (Worst-Case Response Time)
  - Inclui o WCET de todas as tarefas do sistema, de suas taxas de recorrência, de como são suas interações, de todos os overheads

Classificação Acadêmica: Com Garantia 6/9

- **Vantagens**
  - Determina previamente que todos os deadlines serão cumpridos
  - Necessário para aplicações críticas certificadas
  - Teoria serve de base para abordagens sem garantia

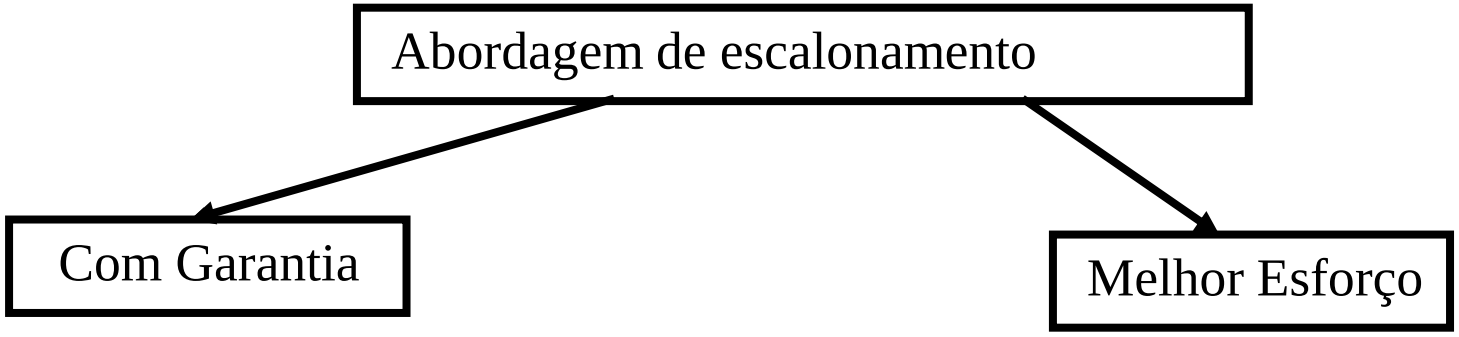
- **Desvantagens**
  - Necessário conhecer exatamente a carga
  - Necessário reservar recursos para o pior caso
  - Gera enorme sub-utilização do hardware (mais caro)
  - Difícil determinar o pior caso em soluções COTS (commercial off-the-shelf)

Classificação das Abordagens na Academia 7/9

- Existem duas grandes abordagens de escalonamento para sistemas de tempo real na perspectiva acadêmica
  - Matemática de escalonamento

- **Sistemas com garantia**

- **Sistemas com melhor esforço**



Classificação Acadêmica: Melhor Esforço 8/9

- Não existe garantia de que todos os deadlines serão cumpridos
- O “Melhor Esforço” será feito neste sentido

- Capaz de fornecer um previsão probabilista
  - Simulação, testes, etc

- Existe a possibilidade de Sobrecarga (*overload*)
- Sobrecarga:
  - Não é possível cumprir todos os deadlines
  - Não é uma falha do projeto
  - É uma situação natural uma vez que não existe garantia antecipada

Classificação Acadêmica: Melhor Esforço 9/9

- Questão fundamental: Como tratar a sobrecarga ?
  - Em sobrecarga ATRASA algumas tarefas
  - Em sobrecarga DIMINUI a precisão de algumas tarefas
  - Em sobrecarga NÃO EXECUTA algumas tarefas
  - Em sobrecarga AUMENTA O PERÍODO de algumas tarefas

- Vantagens desta abordagem
  - Não é necessário conhecer o pior caso
  - Sistemas mais baratos, não são projetados para o pior caso
  - Não é necessário conhecer a carga exatamente
- Desvantagens
  - A princípio qualquer deadline poderá ser perdido

Classificação das Abordagens na Prática 1/15

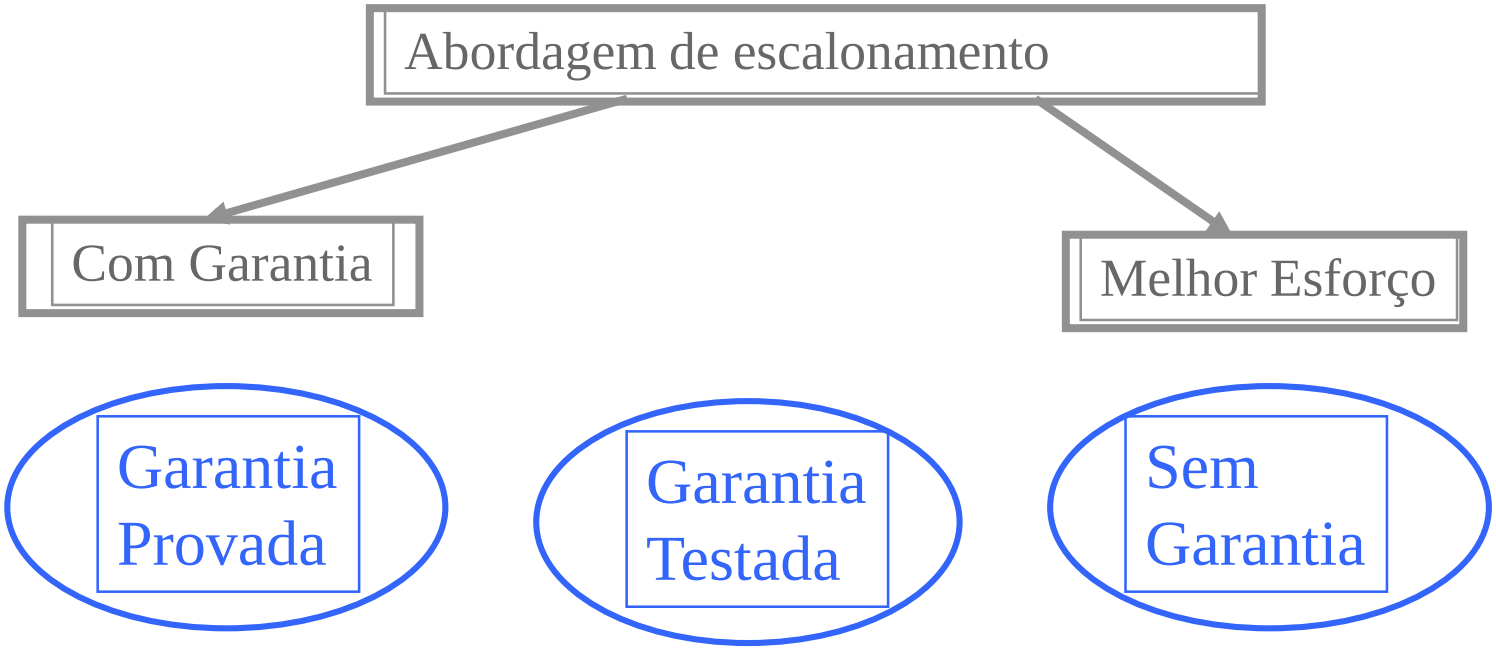
- Literatura acadêmica oferece várias maneiras para tratar sistemas de tempo real com garantia (hard real-time)

- Na prática a coisa é um pouco mais complicada
  - Entram aspectos econômicos
  - Entram aspectos do desenvolvimento como um todo e não só tempo real
  - Entram as limitações das propostas acadêmicas

- Uma classificação que incorpore as práticas da indústria é diferente da classificação acadêmica

Classificação das Abordagens na Prática 2/15

- A abordagem com garantia é dividida em 2 tipos
- **Garantia Provada** equivale ao **Hard Real-Time** da academia
- **Sem Garantia** equivale ao **Soft Real-Time**
- Surge a **Garantia Testada**, que tem um pouco dos dois



Classificação na Prática: Garantia Provada 3/15

- Hard Real-Time clássico inclui uma **Garantia Provada**
- Safety-critical applications
- Não tolera nenhuma perda de deadline
- A perda de um deadline representa uma falha do sistema
- Requer algum tratamento de exceção forte
  - Tolerância a faltas via replicação ativa
  - Tolerância a faltas passiva via propriedade construtiva (eletro-mecânica)
  - Reinicia
  - Desliga

Classificação na Prática: Garantia Provada 4/15

- Safety-critical applications
  - Não tolera nenhuma perda de deadline

- Tarefas críticas em satélites
- Tarefas críticas em aviões
- Tarefas críticas em carros

- Sistemas críticos em aviões é o grande motivador da área
- Começa a ser importante para carros autônomos

- **Necessita prova matemática de que jamais perde um deadline**
  - Certificação de agência fiscalizadora

Classificação na Prática: Garantia Provada 5/15

- Análise de escalonabilidade com garantia
- Ferramenta para determinar WCET
- Necessita arquitetura determinista, analisável
- Microcontrolador de pequeno ou médio porte
- Software simples, microkernel ou tudo na aplicação
  - Junta tudo, verifica o conjunto
- Certificação é o maior custo (30x mais caro que software comum)

- Tudo isto:
  - Restringe o espectro de processadores possíveis
  - Desenvolvimento é caro (ferramentas, design, verificação)
  - Justificável apenas para *safety-critical systems* com processo de certificação ou quando uma falha pode quebrar a empresa (freio do carro)

Classificação na Prática: Garantia Provada 6/15

- Muito difícil usar multicore
  - A não ser como um conjunto de monoprocessadores que estão por acaso no mesmo chip
  - Mesmo assim uma cache comum ou um barramento comum complica a análise
- Muito difícil usar processadores complexos
  - Não consegue analisar
- Novas tecnologias que poderão vir no futuro
  - Análise probabilista da escalonabilidade
  - Processadores projetados especialmente para serem analisados

Classificação na Prática: Garantia Testada 7/15

- Na prática, verificação dos requisitos temporais pode ser feita via teste e não via modelos matemáticos: **Garantia Testada**
- Não tolera nenhuma perda de deadline
- A perda de um deadline representa uma falha do sistema
- Requer algum tratamento de exceção forte
  - Tolerância a faltas via replicação ativa
  - Tolerância a faltas passiva via propriedade construtiva (eletro-mecânica)
  - Reinicia
  - Desliga

- **Verificação por teste (que jamais perde um deadline)**

Classificação na Prática: Garantia Testada 8/15

- Não tolera nenhuma perda de deadline
  - A perda de um deadline representa uma falha do sistema
- Mas não é geralmente *safety-critical*
  as vezes até é !!!

- Sistemas de geração/transmissão de energia elétrica
  - Relés de proteção, reguladores de tensão e frequência, etc.
- Inversores elétricos
- Muitas tarefas automotivas
- Equipamentos médicos
  - Safety-critical systems, mas são lentos, é fácil cumprir deadlines

- **Verificação por teste (que jamais perde um deadline)**

Classificação na Prática: Garantia Testada 9/15

- Necessita arquitetura quase determinista
- Código simples, quase determinista
  - Não utiliza algoritmos recursivos, por exemplo
- WCET obtido através de medições

- Não tem certificação
- Ênfase em testes de stress
  - Busca as condições nas quais deadlines poderiam ser perdidos
- Microkernel determinista ou tudo na aplicação

- Folgas grandes para as tarefas críticas

- Teoria de escalonamento hard real-time pode ser usada com valores aproximados como ferramenta auxiliar do desenvolvedor

Classificação na Prática: Garantia Testada 10/15

- Existe um trade-off no projeto
- Quanto mais safety-critical for a tarefa:
  - Mais determinista é o código
  - Mais simples é o processador (pode ter mais de um)
  - Maiores são as folgas
  - Mais rigorosos são os testes

- Prioridade por importância e não por período, deadline, etc:
  - Tarefas críticas recebem prioridade fixa mais alta
  - O tempo que sobra é para as demais tarefas
  - Folga delas corresponde a todo o resto do tempo do processador que elas próprias não usam
  - Melhor a tela travar um pouco do que o motor explodir

Classificação na Prática: Sem Garantia 11/15

- Muitas vezes o sistema é **Sem Garantia** temporal
- Chamado soft real-time system, best-effort

- Tolera a perda de deadlines se estas forem suficientemente raras

- O que é raro ?
- Depende da especificação do sistema:
  - Tarefa não pode perder x deadlines seguidos
  - Tarefa não pode perder mais que x deadlines em y ativações
  - Tarefa não pode perder mais que x deadlines em y segundos
  - Etc, a lista é grande

- Histograma com os tempos de resposta pode ajudar

Classificação na Prática: Sem Garantia 12/15

- A perda de um deadline não representa a falha do sistema
- A perda de um deadline isolado não requer tratamento de exceção
  - Não gera a falha do sistema

- Controle realimentado em aplicações industriais não críticas
  - A inércia da planta mascara a perda de um deadline (existem limites)
- Muitos exemplos no mundo industrial e doméstico
  - Controle de um forno industrial
  - Liga/desliga de chaves em fábrica (manufatura)
  - Linha branca
  - Controlador semafórico
  - Centrais telefônicas

- **Verificação por teste (perde deadlines de forma aceitável)**

Classificação na Prática: Sem Garantia 13/15

- Arquitetura qualquer, depende dos requisitos da aplicação
  - Desde de pequeno microcontrolador até PC multicore

- Testes principalmente em condições normais
- Em geral usa microkernel, mas
  pode ser usado desde nada até Linux, de tempo real ou não
  - Depende das funcionalidades da aplicação

- Design e testes dependem de quanto deadline pode perder sem isto ser percebido como uma falha

Classificação na Prática: Não é Tempo Real 14/15

- A grande maioria das aplicações computacionais não são sistemas de tempo real

- Exemplos:
  - Folha de pagamento
  - Compilador C

- Quanto mais rápido executar melhor
- Isto é desempenho !!!

Classificação das Abordagens na Prática 15/15

- Com garantia provada (Hard Real-Time System)
  - Não pode perder deadline nunca, necessário certificação
  - Teste de Utilização, Análise do Tempo de Resposta, Executivo Cíclico
  - Exemplo: sistemas aviônicos
- Com garantia testada (Quasi-Hard Real-Time System)
  - Não pode perder deadline nunca, não precisa certificação
  - Testes mostrando que não perde deadline
  - Exemplo: Inversor elétrico, relé de proteção elétrica
- Sem garantia (Soft Real-Time System, Best-effort)
  - Precisa estipular serviço mínimo (percentual perdido, atraso máximo, etc)
  - Testes mostrando que é bom o suficiente
  - Exemplo: Central telefônica, controle realimentado de processo lento
- Não é tempo real, requisito temporal é trivial
  - Folha de pagamentos

Exemplo: Controle Realimentado 1/2

- Controle realimentado de alguma variável física
  - Lê o sensor
  - Cálcula a ação de controle
  - Comanda o atuador
- Exemplo: velocidade, temperatura, vazão, etc
- Pode aparecer em qualquer um dos tipos de abordagens
- A criticalidade do requisito temporal define a abordagem, e não o fato de ser controle realimentado

Exemplo: Controle Realimentado 2/2

- Controle automático de altitude de um avião civil
  - Tempo real com garantia provada
  - Precisa certificar o sistema
- Controle de frequência (60 Hz) de uma turbina hidroelétrica
  - Tempo real com garantia testada
  - Não pode perder deadline nunca, mas apenas testes são usados
- Controle de temperatura de uma caldeira de hotel
  - Tempo real sem garantia
  - Perder alguns deadlines de vez em quando não é um problema
- Atualização do display de um rádio relógio a cada segundo
  - Requisito temporal é trivial
  - Esta tarefa nem é considerada como de tempo real

- A criticalidade do requisito temporal define a abordagem

Comentários sobre as Abordagens 1/4

- Existe uma vasta teoria de escalonamento **tempo real hard**
  - Muitos milhares de artigos
- Quase toda a teoria (matemática) de escalonamento tempo real visa sistemas de tempo real hard
  - Garantia para os deadlines provada matematicamente
- A demanda de prova formal limita o espaço de projeto do software
  - Apenas técnicas com pior caso razoável
    - Por exemplo, não pode usar tabela hash
- A demanda de prova formal limita o espaço de projeto do hardware
  - Tools para análise de wcet suportam poucos processadores
  - Caches, barramentos são problemas
- A demanda de prova formal aumenta custos de desenvolvimento
  - Subutilização do hardware, ferramentas mais caras, mais atividades

Comentários sobre as Abordagens 2/4

- Os custos e as limitações da prova formal a tornam aceitável somente em sistemas safety-critical
  - Principalmente se houver certificação

- Mercado restrito:
  - Aviões
  - Satélites
  - Carros autônomos (tende a crescer)

- Aplicação da teoria depende da evolução dos processadores
  - Processadores com tempo de execução determinista
    - Comercialmente improvável
  - Processadores com tempo de execução aleatório
    - Comercialmente improvável

Comentários sobre as Abordagens 3/4

- No outro extremo ...

- Métodos tradicionais de engenharia de software conseguem lidar com **sistemas de tempo real soft**, desde que:

- Acompanhados com testes de stress (garantia testada)
- Acompanhados com testes normais (sem garantia)

- Projetados levando em consideração os aspectos temporais
  - Impedimentos ao atendimento dos deadlines devem ser removidos do projeto
  - Por exemplo, grandes contenções causadas por mecanimos de sincronização
  - Algoritmos de escalonamento não apropriados

Comentários sobre as Abordagens 4/4

- **Sistemas de tempo real com garantia testada** demandam cuidados especiais

- Design do software e do hardware precisa levar em consideração a necessidade de determinismo
- Testes de stress são absolutamente necessários para a confiabilidade do produto
- Resultados teóricos válidos para sistemas críticos podem ser usados como heurísticas no projeto de sistemas firmes

- Não existe garantia formal para os deadlines
- Mecanismos para o tratamento de exceções temporais devem ser embutidos na aplicação
  - Tais como reiniciar ou levar para um estado seguro

Resumo

- Necessidade de diferentes abordagens
- Classificação das abordagens na academia
- Classificação das abordagens na prática
- Exemplo: Controle realimentado
- Comentários sobre as Abordagens