Sistemas Operacionais de Tempo Real

Fundamentos dos Sistemas de Tempo Real

Rômulo Silva de Oliveira

Edição do Autor, 2018

www.romulosilvadeoliveira.eng.br/livrotemporeal

Outubro/2018

Introdução 1/4

- Maioria das aplicações são construídas a partir dos serviços oferecidos por um microkernel ou kernel de sistema operacional
- O atendimento dos requisitos temporais da aplicação depende
  - Não somente do código da aplicação,
  - Mas também do comportamento do sistema operacional usado
- Muitas vezes os requisitos temporais da aplicação são tão rigorosos ou a plataforma de hardware é tão limitada que o sistema operacional é substituído
  - Por um executivo cíclico
  - Por um laço principal auxiliado por tratadores de interrupção

Introdução 2/4

- Qualquer programa de computador, seja de tempo real ou não, será mais facilmente construído se puder aproveitar os serviços de um sistema operacional
- O sistema operacional permite que o programador da aplicação utilize abstrações de mais alto nível
  - Processos
  - Threads
  - Arquivos
  - Segmentos de memória
  - Sockets de comunicação
- Sem precisar lidar com a gerência dos recursos básicos do hardware

Introdução 3/4

- Sistemas Operacionais de Propósito Geral são construídos com o objetivo de apresentar um bom desempenho médio
  - Distribuem os recursos do sistema de forma mais ou menos igualitária entre os processos, threads e usuários
- Mecanismos internos do kernel melhoram em muito o desempenho médio do sistema, mas geram grande variância
  - Tornam mais difícil mostrar que os requisitos temporais serão sempre cumpridos
  - Caches de disco, memória virtual, fatias de tempo do processador, etc
- Aplicações com requisitos de tempo real estão menos interessadas em uma distribuição justa dos recursos
  - Mais interessadas em atender requisitos temporais

Introdução 4/4

- **Sistemas Operacionais de Tempo Real – SOTR**
- São sistemas operacionais onde especial atenção é dedicada ao comportamento temporal das tarefas
- Além do aspecto temporal, também alguns serviços específicos são normalmente exigidos de um SOTR
  - Funcionalidade importante para aplicações de tempo real porém nem sempre necessária para as aplicações de propósito geral
- Existe uma dificuldade básica em definir o que é um SOTR
- O termo SOTR é usado largamente em páginas da Web, livros, artigos, produtos, etc
  - Entretanto, não existe consenso sobre o que isto significa exatamente
- Definição mais comum é
  "um sistema operacional apropriado para aplicações de tempo real"
  - Significa coisas muito diferentes para diferentes pessoas

Aspectos Funcionais dos SOTR 1/5

- Se um dado serviço existe no SOPG é por que ele é útil
  - É usado por alguma aplicação
- Como qualquer sistema operacional, um SOTR procura tornar a utilização do computador mais eficiente e mais conveniente
  - Facilidades providas por SOPG são bem vindas em um SOTR
- Com respeito à funcionalidade de um SOTR
  - É razoável supor que o mesmo deveria suprir os mesmos serviços que um SOPG
  - Em termos de serviços oferecidos, a princípio, mais é melhor
  - A aplicação de tempo real precisa dos mesmos serviços que as demais aplicações
  - Ela apenas coloca requisitos adicionais, de natureza temporal

Aspectos Funcionais dos SOTR 2/5

- Em sistemas embutidos (embedded systems) com sérias limitações de memória, energia e outras
  - SO deveria prover apenas os serviços realmente usados pela aplicação
  - No sentido de não gastar recursos do hardware para implementar serviços que jamais serão usados pela aplicação
  - A ênfase deste livro está nos aspectos de tempo real e não nos aspectos de redução de tamanho (footprint) dos sistemas operacionais
- Em SOPG um aspecto muito considerado é o sobrecusto (overhead)
  - A quantidade de recursos que o kernel consome ele próprio para prover os serviços que a aplicação necessita
  - Para SOPG, quanto menos sobrecusto melhor
- Não é diferente no caso dos SOTR
  - Queremos uma boa implementação do kernel, não importa tratar-se de um SOPG ou um SOTR
  - Teremos que buscar a diferença entre eles em algum outro aspecto

Aspectos Funcionais dos SOTR 3/5

- Existem alguns serviços adicionais e/ou melhorados que um SOTR deve prover
- Serviço de relógio (gettime)
  - Permite que o processo obtenha a hora e data correntes, preferencialmente conforme a UTC ou algo relacionado
- Temporização de intervalo (sleep)
  - Permite que o processo fique suspenso durante um determinado intervalo de tempo, especificado como parâmetro
- Temporização de instante (wake-up)
  - Permite que o processo fique suspenso até um determinado instante de tempo, especificado como parâmetro
- Monitoração temporal das tarefas (watch-dog)
  - Dispara a execução de um tratador de exceção caso o processo não sinalize que terminou determinado conjunto de tarefas até um instante de tempo especificado como parâmetro

Aspectos Funcionais dos SOTR 4/5

- Especialmente importante é a disponibilização de uma temporização (sleep)
  - Onde o parâmetro não seja o intervalo de tempo de espera
  - Mas sim o momento futuro de voltar a executar
  - Isto é essencial para a programação de tarefas realmente periódicas

- Deseja-se uma excelente <u>resolução</u> e uma excelente <u>precisão</u>
  - A resolução está associada com a granularidade dos parâmetros usados
  - Por exemplo, função “sleep()” recebe parâmetro em microssegundos
- O fato do parâmetro do “sleep()” usar como unidade de tempo o microssegundo não significa que a implementação do “sleep()” seja capaz de suspender o processo com a precisão de microssegundos
  - A precisão precisa ser boa também

Aspectos Funcionais dos SOTR 5/5

- Em termos de funcionalidade adicional dos SOTR, outro aspecto importante são os algoritmos usados na implementação de mecanismos de sincronização
  - Por exemplo o mutex
- Espera-se soluções mais apropriadas para sistemas de tempo real
  - No sentido de reduzir as inversões de prioridades causadas pelos bloqueios
- Espera-se que um SOTR ofereça opções em termos das políticas de funcionamento do mutex a nível de aplicação

Aspectos Temporais dos SOTR 1/2

- Tanto a aplicação como o SO compartilham os mesmos recursos do hardware
- Comportamento temporal do SO afeta o comportamento temporal da aplicação
- Por exemplo, considere a rotina do SO que trata as interrupções de timer
  - O projetista da aplicação pode ignorar completamente a função desta rotina
  - Mas não pode ignorar o seu efeito temporal
  - Isto é, a interferência que ela causa na aplicação, seu impacto nos tempos de resposta

- Solicitar um serviço ao SO através de chamada de sistema significa que o processador será ocupado pelo código do sistema operacional por algum tempo
  - Para executar esta chamada
- A capacidade da aplicação atender seus deadlines depende da capacidade do SO em fornecer o serviço solicitado em um tempo que não inviabilize aqueles deadlines

Aspectos Temporais dos SOTR 2/2

- Diversas técnicas populares em SOPG são especialmente problemáticas para as aplicações de tempo real
- O mecanismo de memória virtual (virtual memory) é capaz de gerar grandes atrasos, em função das faltas de páginas (page fault)
- Mecanismos tradicionais usados em sistemas de arquivos fazem com que o tempo para acessar um arquivo possa variar muito
  - Ordenar a fila do disco magnético para diminuir o tempo médio de acesso faz com que o tempo de acesso ao disco no pior caso seja maior
- Aplicações de tempo real procuram minimizar o efeito negativo destes mecanismos
- Por exemplo, pode-se desativar o mecanismo sempre que possível
- Ou ainda, usar o mecanismo apenas em tarefas sem requisitos temporais rigorosos
  - Acesso a disco somente feito por tarefas sem requisitos temporais

SOTR: O Ideal Impossível 1/1

- Com respeito aos aspectos temporais, um SOTR deveria ser completamente transparente, invisível, imperceptível para a aplicação
- O kernel não geraria nenhum tipo de interferência ou bloqueio sobre as tarefas da aplicação
  - Não existiria nenhum tipo de atraso devido ao kernel
  - O chaveamento de contexto seria instantâneo
  - Todas as chamadas de sistema demorariam exatamente o tempo do periférico
  - Nenhum atraso adicional seria causado pelas rotinas do kernel
  - As interrupções ficariam habilitadas todo o tempo
  - Em resumo, o kernel não afeta o comportamento temporal da aplicação
- Obviamente, este SOTR idealizado é impossível de ser implementado
  - A não ser que os serviços oferecidos por ele sejam nulos
  - A implementação de microkernel vai nesta direção

SOTR: O Ideal Possível porém Inexistente 1/2

- RTOS ideal possível
  - Minimiza seus efeitos sobre os tempos da aplicação
  - Deixa esses efeitos explícitos
  - Para que o desenvolvedor antecipe se os requisitos temporais da aplicação serão ou não atendidos
- Para tempo real crítico (hard real-time), é essencial demonstrar antes da execução que todos os deadlines serão cumpridos
  - Mesmo em cenários de pior caso
- Obter esta garantia de um kernel complexo é algo muito distante da prática usual na área de sistemas operacionais
- Tal garantia demanda mudanças no design e na própria filosofia de construção dos kernel
- Atualmente é possível fazer esta análise de pior caso somente em microkernel simples, com um conjunto bem limitado de serviços

SOTR: O Ideal Possível porém Inexistente 2/2

- Tradicionalmente o SO provê isolamento espacial entre as aplicações
  - Através da gerência de memória
- Agora surge a necessidade de “isolamento temporal” entre aplicações
  - Os requisitos temporais da aplicação X em execução continuam sendo cumpridos mesmo quando a aplicação Y começa a ser executada simultaneamente
- Pode-se supor que as aplicações X e Y compartilhem recursos (mesmo RTOS)
- Algum efeito mútuo vai haver entre elas
  - Bloqueios, interferências, jitters
- A análise de escalonabilidade idealmente iria informar o resultado
- O papel do SOTR no isolamento temporal está em permitir a análise de escalonabilidade do sistema
  - A qual vai informar sobre a existência ou não de isolamento
- Ainda existe um longo caminho até chegarmos em um SOTR ideal
  - Trata-se ainda de um sonho de desenvolvedor de aplicações de tempo real
  - É um ideal possível porém que inexiste atualmente

SOTR: A Realidade 1/1

- Na realidade, um SOTR em geral é apenas um sistema operacional com <u>bom comportamento temporal na maioria dos casos</u>
- São os sistemas operacionais que, quando comparados a outros, apresentam boas características de design para tempo real
- Embora com bom design, não é possível analisar um sistema baseado em kernel completo com respeito ao cumprimento dos deadlines
  - Devido a complexidade deste tipo de kernel
  - Seu design não é feito com este tipo de análise em mente
- Muitos sistemas operacionais de tempo real existentes estão nesta categoria
- Trata-se de uma evolução natural dos kernel existentes
- A grande maioria das aplicações de tempo real não são críticas e podem ser atendidas por este tipo de sistema operacional
- Mesmo aplicações de controle em malha fechada toleram atrasos eventuais no comando dos atuadores
  - Desde que isto não ocorra repetidamente por um longo período

Diferenças Construtivas entre SOPG e SOTR 1/14

- **Algoritmo de Escalonamento Apropriado**

- Do ponto de vista da análise de escalonabilidade, deseja-se um algoritmo para o qual a literatura ofereça métodos de análise e testes de escalonabilidade
- O escalonamento mais usado em sistemas de tempo real é aquele baseado em prioridades fixas preemptivas
- Este é o algoritmo implementado pela maioria dos sistemas operacionais de tempo real

Diferenças Construtivas entre SOPG e SOTR 2/14

- **Níveis de Prioridade Suficientes**

- Escalonamento baseado em prioridades é suportado pela maioria dos sistemas operacionais
  - Número de diferentes níveis de prioridade varia bastante
- Quando o número de níveis de prioridade disponíveis é menor do que o número de tarefas, e passa a ser necessário agrupar várias tarefas no mesmo nível, isto reduz a escalonabilidade do sistema
- O desejado é que o número de níveis de prioridade seja igual ou maior do que o número de tarefas no sistema

Diferenças Construtivas entre SOPG e SOTR 3/14

- **Sistema Operacional não Altera Prioridades das Tarefas**

- Muitos sistemas operacionais manipulam por conta própria as prioridades das tarefas
  - Por exemplo, o mecanismo de **envelhecimento** (***aging***)
- Muitos sistemas operacionais de propósito geral também incluem mecanismos que reduzem automaticamente a prioridade de uma thread na medida que ela consome tempo de processador
  - Diminui o tempo médio de resposta no sistema
- Em um SOTR esses mecanismos
  - Não contribuem para a qualidade temporal
  - Tornam mais complexa a verificação do cumprimento dos requisitos temporais
  - Devem ser evitados

Diferenças Construtivas entre SOPG e SOTR 4/14

- **Tratadores de Interrupções com Execução Rápida**

- Parte importante de qualquer sistema operacional é a execução dos tratadores de interrupção
- Eles são uma importante fonte de interferência sobre as demais tarefas
- Cada tratador de interrupção pode ser considerado como uma tarefa de prioridade mais alta do que todas as tarefas da aplicação.
- Em um SOTR todos os tratadores de interrupção sejam executados rapidamente
- O tratador de interrupção deve realizar apenas aquele processamento mínimo que é necessário imediatamente na ocorrência da interrupção
- O tratador de interrupção pode liberar uma thread de kernel para concluir o atendimento da interrupção recebida do periférico

Diferenças Construtivas entre SOPG e SOTR 5/14

- **Desabilitar Interrupções ao Mínimo**

- O tempo entre a sinalização de uma interrupção no hardware e o inicio da execução de seu tratador é normalmente chamado de latência do tratador de interrupção
- A latência no disparo de um tratador de interrupção inclui
  - O tempo que o hardware leva para desviar a execução para o código do tratador
  - Também é necessário incluir o tempo máximo que as interrupções podem ficar desabilitadas
- Por vezes, trechos de código do sistema operacional precisam executar com as interrupções desabilitadas
  - Deve ser minimizado

Diferenças Construtivas entre SOPG e SOTR 6/14

- **Emprego de Threads de Kernel**

- Em um SOTR é importante que todas as atividades do kernel ou microkernel sejam feitas ou por tratadores de interrupção rápidos
- Ou por threads de kernel as quais são liberadas pelos tratadores de interrupção
- Estas threads de kernel devem executar conforme a sua prioridade
  - Podendo ter prioridade menor que algumas tarefas de tempo real da aplicação
  - Conforme a atribuição de prioridades escolhida pelo desenvolvedor da aplicação
  - Por exemplo, driver do teclado versus tarefa de controle realimentado

Diferenças Construtivas entre SOPG e SOTR 7/14

- **Tempo de Chaveamento entre Tarefas Pequeno**

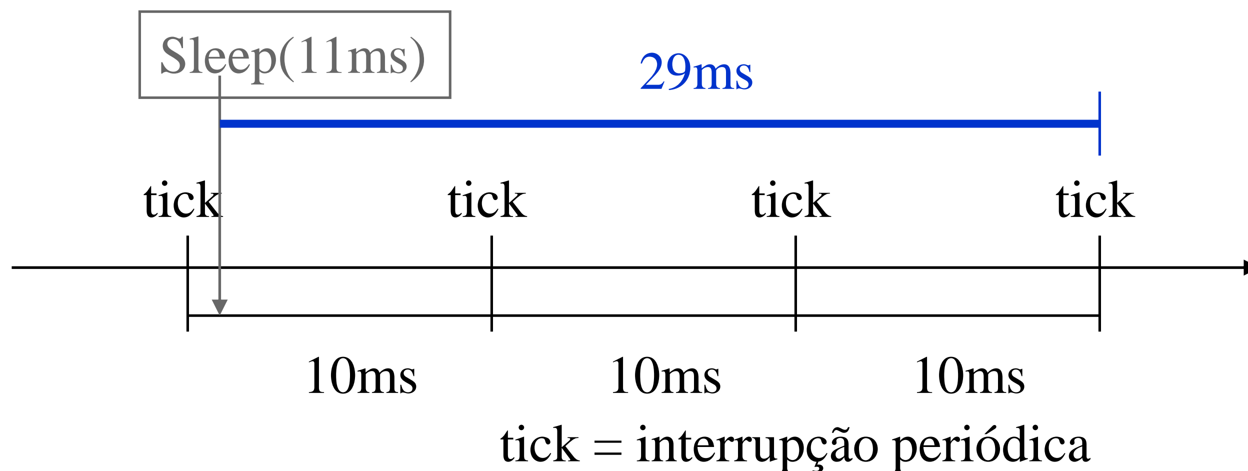
- Uma métrica muito citada no mercado de sistemas operacionais é o tempo para chaveamento de contexto entre duas tarefas
- Inclui salvar os registradores da tarefa que está executando e carregar os registradores com os valores da nova tarefa
- Em geral, esta métrica não inclui o tempo necessário para decidir qual tarefa vai executar
  - Depende do algoritmo de escalonamento utilizado
- O tempo de chaveamento de contexto soma-se ao tempo máximo de execução de cada tarefa, em um cenário de pior caso
  - Quanto menor, melhor

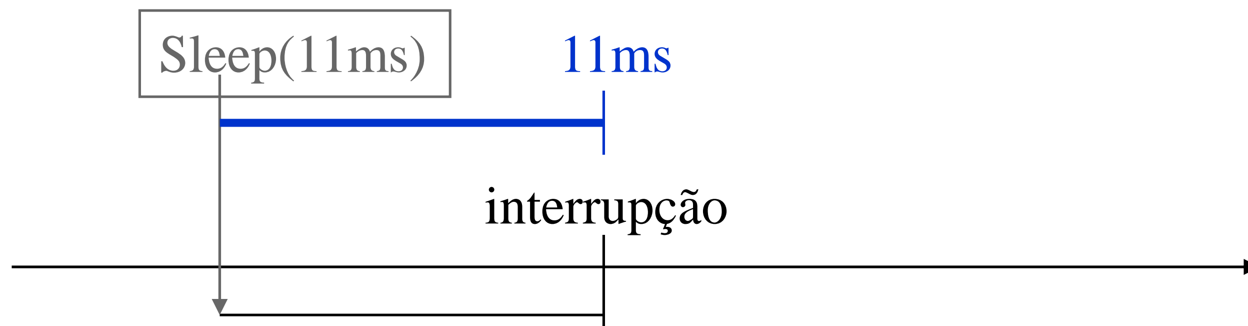
Diferenças Construtivas entre SOPG e SOTR 8/14

- **Emprego de Temporizadores com Alta Resolução**

- Tarefas da aplicação armam temporizações ao final das quais uma ação ocorre
- Essa ação pode ser assíncrona (por exemplo, o envio de um sinal Unix)
- Ou síncrona (por exemplo, a liberação da tarefa após uma chamada "sleep()")
- Temporizadores são utilizados na implementação de mecanismos de *time-out*, *watch-dog*, e também tarefas periódicas, entre outros usos
- Um SOTR utiliza temporizadores de alta resolução, baseados em interrupções aperiódicas
  - Relógio de hardware não gera interrupções periódicas mas é programado para gerar uma interrupção no próximo instante de interesse
- Supondo ser o final do "sleep()" o próximo instante de interesse, o temporizador em hardware seria programado para gerar uma interrupção exatamente no momento esperado pela tarefa em questão

Diferenças Construtivas entre SOPG e SOTR 9/14





Diferenças Construtivas entre SOPG e SOTR 10/14

- **Comportamento das Chamadas de Sistema no Pior Caso**

- Para as aplicações de tempo real, o tempo de execução no pior caso é mais relevante do que o tempo de execução no caso médio
- Em geral, a implementação das chamadas de sistema é feita de maneira a minimizar o tempo médio
- Aplicações de tempo real são beneficiadas quando o código que implementa as chamadas de sistema apresenta bom comportamento também no pior caso
- Na construção de um SOTR devem ser evitados algoritmos que apresentam excelente comportamento médio
  - Porém um péssimo comportamento de pior caso

Diferenças Construtivas entre SOPG e SOTR 11/14

- **Preempção de Tarefa Executando Código do Kernel**

- Um microkernel simples normalmente executa com interrupções desabilitadas e dentro dele existe, a cada momento, apenas um fluxo de execução
- Já um kernel completo é grande demais para executar com interrupções desabilitadas
  - É pensado como um programa concorrente
  - Vários fluxos de execução co-existem
- Um kernel não preemptivo é capaz de gerar grandes inversões de prioridade
  - Suponha que uma tarefa de baixa prioridade faça uma chamada de sistema
  - Enquanto o código do kernel é executado, ocorre a interrupção de hardware
  - Que deveria liberar uma tarefa de alta prioridade
  - A tarefa de alta prioridade terá que esperar até que a chamada de sistema da tarefa de baixa prioridade termine
- Um SOTR deve ser preemptivo
  - Ainda que em determinados momentos, interrupções precisem ser desabilitadas e a preempção desligada por pequenos intervalos de tempo

Diferenças Construtivas entre SOPG e SOTR 12/14

- **Mecanismos de Sincronização Apropriados**

- Aplicações de tempo real são majoritariamente construídas como programas concorrentes
- Threads precisam acessar variáveis compartilhadas
- Precisam de mecanismos de sincronização
- Um SOTR deve oferecer para as tarefas de aplicação mecanismos de sincronização apropriados para tempo real

Diferenças Construtivas entre SOPG e SOTR 13/14

- **Granularidade das Seções Críticas dentro do Kernel**

- Um kernel preemptivo é capaz de chavear imediatamente para a tarefa de alta prioridade quando a mesmo é liberada
- Entretanto, inversão de prioridade dentro do kernel ainda é possível
- Quando a tarefa de alta prioridade recém ativada faz uma chamada de sistema
  - Mas é bloqueada ao acessar, dentro do kernel, uma estrutura de dados compartilhada
- Manter uma granularidade fina para as seções críticas dentro do kernel
  - Aumenta a complexidade do código
  - Mas reduz o tempo que uma tarefa de alta prioridade precisa esperar até que a tarefa de baixa prioridade libere a estrutura de dados compartilhada

Diferenças Construtivas entre SOPG e SOTR 14/14

- **Gerência de Recursos em Geral**

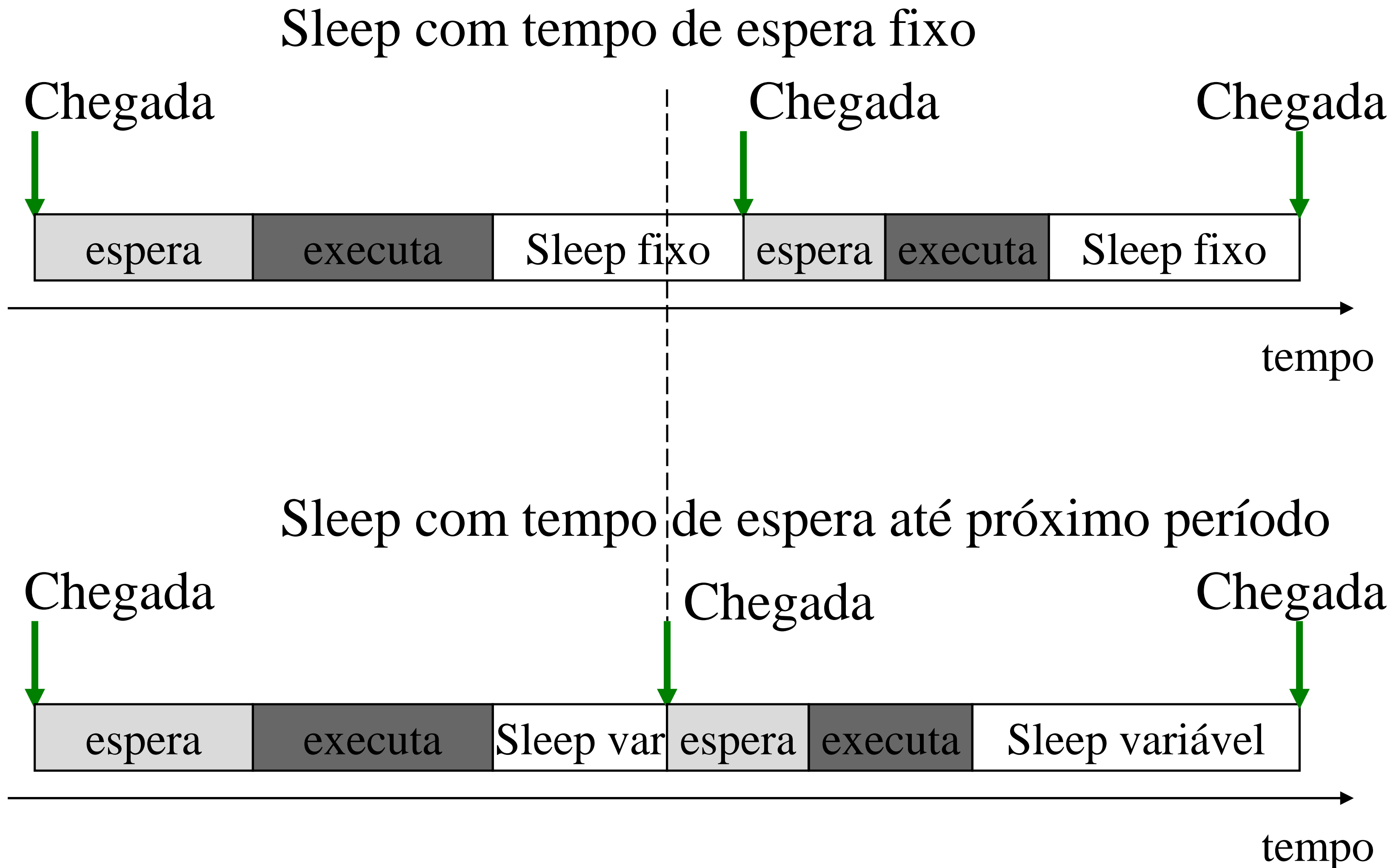
- Todos os sistemas operacionais desenvolvidos ou adaptados para tempo real mostram grande preocupação com a divisão do tempo do processador entre as tarefas
- Memória, periféricos, controladores, servidores também deveriam ser escalonados visando atender os requisitos temporais da aplicação
- Muitos sistemas ignoram isto
  - Tratam os demais recursos da mesma maneira empregada por um SOPG
- Todas as filas do sistema deveriam respeitar as prioridades das tarefas
  - Por exemplo, as requisições de ethernet deveriam ser ordenadas conforme a prioridade e não pela ordem de chegada

Cuidados do Desenvolvedor da Aplicação 1/2

- Por melhor que seja o design do SOTR usado, cabe ainda ao desenvolvedor da aplicação uma série de cuidados
- O correto uso das facilidades do SOTR é necessário

- Talvez o principal aspecto seja a atribuição de prioridades
- Uma razão frequente para a perda de deadlines é a inclusão no código da aplicação de seções críticas longas
- Implementação de tarefa periódica
  - Usar uma chamada de sistema apropriada para isto, caso o sistema operacional em questão ofereça uma
  - Uma outra possibilidade é usar uma chamada de sistema do tipo “sleep()” onde o parâmetro não é o tempo fixo de espera mas sim o instante absoluto no futuro quando a tarefa será liberada
  - Por exemplo, a função clock_nanosleep com a opção TIMER_ABSTIME no Linux

Cuidados do Desenvolvedor da Aplicação 2/2

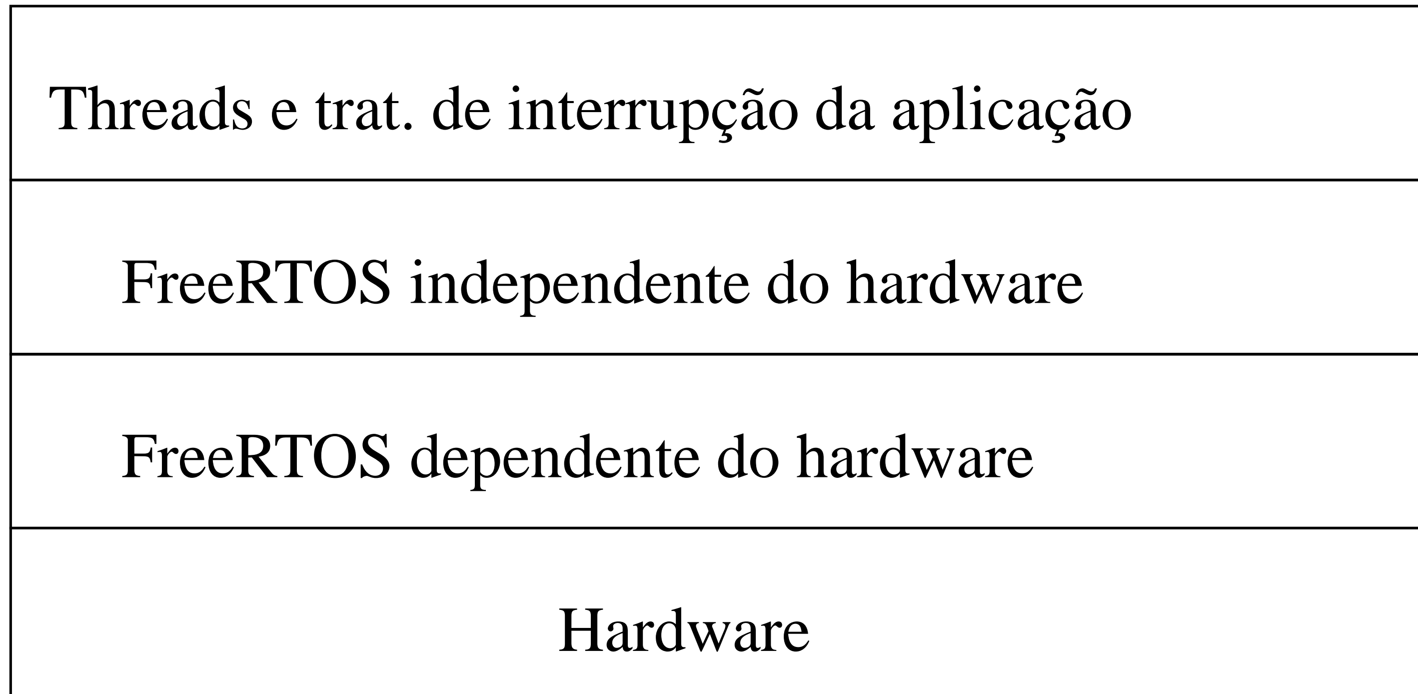


Microkernel Exemplo: FreeRTOS 1/5

- FreeRTOS(www.freertos.org) foi desenvolvido por Richard Barry em torno de 2003
- Mais tarde o desenvolvimento e manutenção continuaram através da empresa Real Time Engineers Ltd.
- Em 2017 a empresa Real Time Engineers Ltd. passou o controle do projeto FreeRTOS para a Amazon Web Services (AWS – aws.amazon.com)
- Existem portes do FreeRTOS para mais de 30 arquiteturas de processadores
- FreeRTOS foi criado para o mercado de aplicações embutidas ou embarcadas (*embedded*) de tempo real de pequeno porte
  - Podem incluir tarefas com diferentes níveis de criticalidade
- FreeRTOS é um microkernel que permite a execução de múltiplas threads (chamadas de tarefas na terminologia do FreeRTOS)
  - Escalonamento baseado em prioridades preemptivas
  - Diversos mecanismos de sincronização entre threads

Microkernel Exemplo: FreeRTOS 2/5

- É relativamente pequeno em tamanho
  - Configuração mínima cerca de 9000 linhas de código
- O código do FreeRTOS pode ser dividido em três subsistemas: gerência do processador, comunicação e sincronização entre threads e interfaceamento com o hardware



Microkernel Exemplo: FreeRTOS 3/5

- FreeRTOS foi projetado para ser configurável
  - Arquivos de configuração são usados para incluir no microkernel apenas as funcionalidades requeridas pelo projeto em questão
- No FreeRTOS é possível alocar memória estaticamente para todos os objetos gerenciados pelo microkernel
  - Threads, queues, semáforos, grupos de eventos, etc
  - Sem alocação dinâmica de memória evita as variâncias no tempo
- Em geral não existe proteção de memória no FreeRTOS
  - Embora alguns poucos portes do microkernel para arquiteturas com MPU (Memory Protection Unit) ofereçam este serviço
- FreeRTOS trata alocação de memória como um subsistema separado
  - O que permite diferentes soluções conforme o tipo de aplicação e a arquitetura de computador usada

Microkernel Exemplo: FreeRTOS 4/5

- Threads (tarefas na terminologia do FreeRTOS) são implementadas como funções C que normalmente executam para sempre
  - Existem chamadas de sistema para criar, destruir, suspender e continuar threads
  - Também é possível alterar a prioridade de uma thread já criada
- Threads com a mesma prioridade são escalonadas com fatias de tempo
  - Baseadas em uma interrupção periódica chamada ***tick interrupt*** (***tick period***)
  - Período é configurável, uma fatia de tempo é igual a um *tick period*
- Além de prioridades preemptivas, o escalonador pode ser configurado para usar apenas fatias de tempo
- No caso de prioridades preemptivas, threads com prioridades iguais executam com fatias de tempo
- É possível desligar o *tick period*
  - Mesma prioridade executam pela ordem de chegada na fila de aptos

Microkernel Exemplo: FreeRTOS 5/5

- Threads podem ficar bloqueadas a espera de eventos
- Eventos podem ser de natureza temporal
  - Quando a thread espera pela passagem do tempo
- Outros eventos estão relacionados com a sincronização com outras threads ou tratadores de interrupção
- FreeRTOS oferece uma variada gama de mecanismos de sincronização
- FreeRTOS também oferece temporizadores em software que podem ser programados pelas threads
  - Para disparar periodicamente (auto-reload timers)
  - Ou em um instante específico no futuro (one-shot timers)
- Quando o temporizador dispara, uma função C (timer callback function) previamente definida pela thread é chamada

Kernel Exemplo: Linux PREEMPT_RT 1/5

- Nos últimos 20 anos surgiram muitas variantes de tempo real do Linux
- Esta seção trata especificamente do PREEMPT_RT
- Patch aplicado no kernel do Linux
- Reduz os segmentos de código do kernel onde preempções não são possíveis
- Move grande parte do código dos tratadores de interrupções para threads do kernel
- Uma excelente fonte de informações atualizadas sobre o PREEMPT_RT é o
  “Real-Time Summit”,
  organizado pela Linux Foundation (www.linuxfoundation.org)
- Também existe uma página sobre a história do PREEMPT_RT no site da Linux Foundation

Kernel Exemplo: Linux PREEMPT_RT 2/5

- Ao longo dos anos, muitas das capacidades originalmente encontradas apenas com a inclusão do patch PREEMPT_RT foram incorporadas na versão oficial do Linux
- Não será dada relevância especial sobre o que já foi incorporado ao kernel oficial e o que ainda não
- O objetivo aqui é descrever as capacidades do Linux para tempo real
  - O que implica em usar o que o kernel oficial oferece e também o patch
- A aplicação do patch PREEMPT_RT ao kernel do Linux requer
  - Que o seu código fonte seja obtido
  - Descompactado
  - E aplicado ao código do kernel
  - Existem scripts Linux que automatizam o processo
  - A configuração do kernel deve ser ajustada

Kernel Exemplo: Linux PREEMPT_RT 3/5

- Nenhuma interface de programação para aplicações (API - Application Program Interface) é disponibilizada pelo patch
- Aplicações de tempo real devem utilizar a API padrão do Posix (Portable Operating System Interface)
  - Conjunto de padrões especificado pela IEEE Computer Society buscando gerar compatibilidade entre diferentes sistemas operacionais
- PREEMPT_RT é suportado por diversas arquiteturas de computador
  - x86, x86_64, ARM, MIPS, Power Architecture, etc
- Da mesma forma que o Linux, o patch PREEMPT_RT é disponibilizado como software livre

Kernel Exemplo: Linux PREEMPT_RT 4/5

- O patch PREEMPT_RT inclui o modelo de preempção:
- **Fully Preemptible Kernel (RT)**
- Todo o código do kernel torna-se preemptável
  - Com a exceção de algumas seções críticas
- Tratadores de interrupção são parcialmente transformados em threads de kernel
- Vários mecanismos de sincronização internos do kernel são reprojetados para reduzir as inversões de prioridade
  - Introdução de “sleeping spinlock” e “rt_mutex”
- Seções longas que executavam com preempção desligada foram reprogramadas como uma sequência de seções menores

Kernel Exemplo: Linux PREEMPT_RT 5/5

- O patch PREEMPT_RT transforma o Linux em um kernel mais apropriado para tempo real
- A implementação das temporizações é alterada, permitindo que temporizadores Posix no espaço de usuário operem com alta resolução
- Conversão de parte do código dos tratadores de interrupção em threads de kernel (threaded interrupt handlers), as quais são preemptáveis
- Alterações nos mecanismos de sincronização dentro do kernel
- Tornou preemptáveis seções críticas dentro do kernel
- Spin-locks são usados para proteger seções críticas dentro do kernel
  - Quando a seção crítica é longa e/ou muito disputada, spin-lock gera atrasos
  - Maioria dos spin-locks convertidos em rt_mutex (sleeping spinlock)

Considerações Finais 1/6

- A <u>escolha de um SOTR</u> não é trabalho simples
- Fatores a considerar:
  - Diferentes SOTR possuem diferentes abordagens de escalonamento
  - Desenvolvedores de SOTR publicam métricas diferentes
  - Desenvolvedores de SOTR não publicam todas as métricas e todos os dados
  - As métricas fornecidas foram obtidas em plataformas diferentes
  - O conjunto de ferramentas para desenvolvimento de aplicações que é suportado varia
  - Ferramentas para monitoração e depuração das aplicações variam
  - As linguagens de programação suportadas em cada SOTR são diferentes
  - O conjunto de periféricos suportados por cada SOTR varia
  - O conjunto de plataformas de hardware suportados varia em função do SOTR
  - Cada SOTR possui um esquema próprio para a incorporação de novos tratadores de dispositivos (device-drivers)
  - Diferentes SOTR possuem diferentes níveis de conformidade com os padrões
  - A política de licenciamento e o custo associado variam

Considerações Finais 2/6

- O comportamento temporal da aplicação tempo real depende tanto da aplicação em si quanto do sistema operacional
- A seleção do SOTR a ser usado depende fundamentalmente dos requisitos temporais da aplicação em questão
- Não existe um SOTR melhor ou pior para todas as aplicações
- A diversidade de aplicações de tempo real existente gera uma equivalente diversidade de sistemas operacionais de tempo real
- Cada abordagem de SOTR tem suas vantagens e desvantagens
  - A escolha faz parte do espaço de projeto da aplicação

Considerações Finais 3/6

- O **FreeRTOS** apresenta um excelente determinismo temporal
- Consequência da sua simplicidade e da elegância de seu design
- A gama de serviços oferecidos é limitada
  - Basicamente a criação de threads e mecanismos de sincronização
  - Qualquer outro serviço deverá ser provido pela própria aplicação

Considerações Finais 4/6

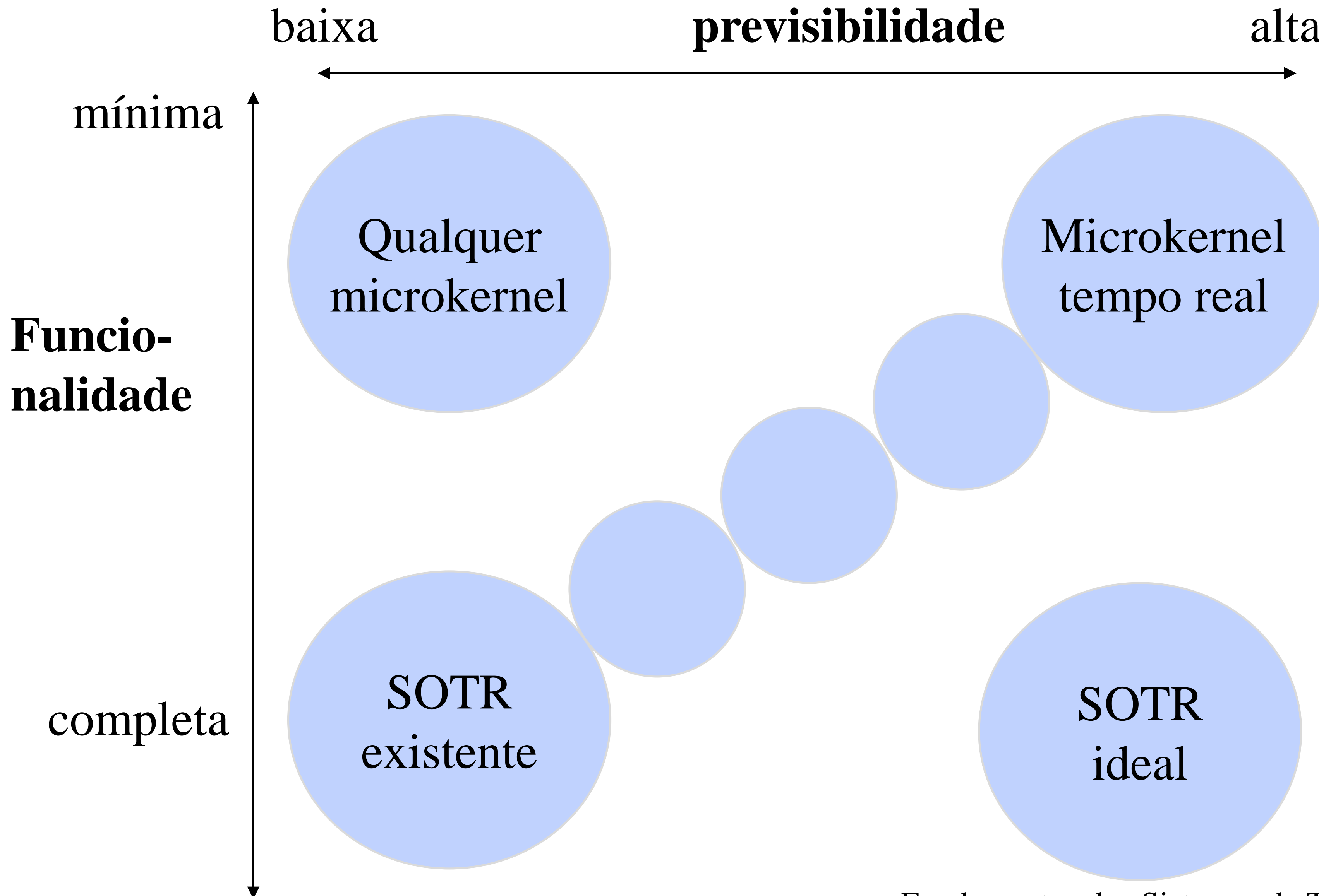
- Existem diversas variantes de Linux para tempo real
  - Seguindo diversas abordagens
- O ***patch*** **PREEMPT_RT** teve uma rápida evolução nos últimos anos e ganha cada vez mais mercado
  - Várias partes do *patch* foram integradas no kernel Linux padrão (*mainline*)
  - Mas o *patch* ainda existirá por muitos anos
- Existe uma compensação (*tradeoff*) fundamental:
  melhor comportamento para aplicações de tempo real tipicamente reduz o desempenho médio (*throughput*)
- Dado o enorme espectro de aplicações que utilizam Linux, não é simples encontrar o balanço ideal
- O *patch* PREEMPT_RT permanece como aquele "algo mais"
  - que aplicações de tempo real precisam
  - mas a grande maioria das aplicações de propósito geral não precisa

Considerações Finais 5/6

- Uma busca rápida na Internet mostrará centenas de SOTR
  - Como kernel ou microkernel
  - Comercial ou software livre
  - Para tempo real crítico ou não

- Em uma dimensão temos a funcionalidade oferecida pelo SOTR
- Na outra dimensão, temos o determinismo temporal
- Funcionalidade mínima com boa previsibilidade temporal é possível
  - Microkernel de tempo real
- Um kernel completo vai oferecer muito menos determinismo que um microkernel

Considerações Finais 6/6



Resumo

- Introdução
- Aspectos Funcionais dos SOTR
- Aspectos Temporais dos SOTR
- SOTR: O Ideal Impossível
- SOTR: O Ideal Possível porém Inexistente
- SOTR: A Realidade
- Diferenças Construtivas entre SOPG e SOTR
- Cuidados do Desenvolvedor da Aplicação
- Microkernel Exemplo: FreeRTOS
- Kernel Exemplo: Linux PREEMPT_RT
- Considerações Finais