Aspectos Construtivos dos Sistemas Operacionais de Tempo Real

Rômulo Silva de Oliveira
Departamento de Automação e Sistemas
Universidade Federal de Santa Catarina
DAS-UFSC

Caracterização

- Sistemas computacionais de tempo real:
  - Submetidos a requisitos de natureza temporal
  - Resultados devem estar corretos lógica e temporalmente
  - Requisitos definidos pelo ambiente físico
- Aspectos temporais
  - NÃO estão limitados a uma questão de maior ou menor desempenho
  - Estão diretamente associados com a funcionalidade
- Sistemas em geral:
  - "Fazer o trabalho usando o tempo necessário"
- Sistemas de tempo real:
  - "**Fazer o trabalho usando o tempo disponível**"

Conceitos básicos

- **Tarefa** (task)
  - Segmento de código cuja execução possui atributo temporal próprio
  - Exemplo: método em OO, sub-rotina, trecho de um programa
- **Deadline**
  - Instante máximo desejado para a conclusão de uma tarefa
- Tempo real crítico (**hard real-time**)
  - Falha temporal pode resultar em conseqüências catastróficas
  - Necessário garantir requisitos temporais em projeto
  - Exemplo: industria petroquímica, mísseis
- Tempo real não crítico (**soft real-time**)
  - Requisito temporal descreve apenas comportamento desejado
  - Exemplo: multimídia, vídeo game

Previsibilidade

- **Previsibilidade** ("predictability")
  - Está associada a capacidade de poder antecipar, saber antes da execução, se os processamentos serão executados dentro de seus prazos
- Associada a uma previsão determinista
  - todos os deadlines serão respeitados
- ou a uma antecipação probabilista
  - baseadas em estimativas, probabilidades são associadas a deadlines definindo as possibilidades dos mesmos serem respeitados
- Previsibilidade determinista implica considerar todos os níveis do sistema computacional:
  - linguagens
  - sistemas operacionais
  - arquitetura do computador, etc

Análise de escalonabilidade tempo real

- Extensa teoria criada para tempo real
  - principalmente nos últimos 15 anos
- Existem muitos modelos matemáticos para
  - Descrever um dado sistema
  - Calcular o tempo máximo de resposta de cada tarefa
- Processo para analisar:
  - Encontrar na literatura um modelo que seja capaz de representar as particularidades do sistema em questão
  - Aplicar as equações definidas para aquele modelo

Análise de escalonabilidade tempo real

- Por exemplo (Audsley, 1992)
  - Tarefas periódicas ou esporádicas
  - Prioridade fixa preemptiva (qualquer política)
  - D <= P
  - Relações de exclusão mútua e precedência
  - Release jitter

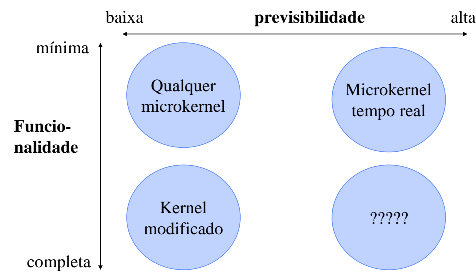
$$W_i = C_i + B_i + \sum_{j \in HP(i)} \left\lceil \frac{J_j + W_i}{P_j} \right\rceil \times C_j$$

$$R_i = J_i + W_i$$

Sistemas operacionais

- Aplicações de tempo real são mais facilmente construídas
  - se puderem aproveitar os serviços de um sistema operacional
- Comportamento temporal do SO
  afeta o comportamento temporal da aplicação
  - Exemplo: tratador de interrupções do timer
- Entretanto
  - Não é corrente o uso da teoria de tempo real na modelagem e análise de sistemas operacionais complexos
  - Apenas pequenos núcleos

Sistemas operacionais



Objetivo

- Objetivo: análise dos aspectos construtivos dos sistemas operacionais
- De que maneiras o sistema operacional impacta o tempo de resposta das tarefas de tempo real ?
- Como podem ser modeladas as contribuições do sistema operacional para o tempo de resposta das tarefas da aplicação ?

1 - Algoritmo de Escalonamento Apropriado

- Primeiro aspecto sempre lembrado é o algoritmo de escalonamento usado
- Deseja-se um algoritmo para o qual a literatura ofereça um método de análise e testes de escalonabilidade
- Para aplicar a teoria citada no artigo
  - SO deve oferecer prioridades preemptivas
- Isto não é um problema
  - Este é o algoritmo implementado pela maioria dos sistemas operacionais de tempo real
  - Basta que cada tarefa no sistema tenha uma prioridade fixa
  - Tanto as tarefas da aplicação como as do sistema

2 - Níveis de Prioridade Suficientes

- Número de diferentes níveis de prioridade varia bastante
  - Por exemplo, POSIX exige no mínimo 32 níveis
- Quando o número de níveis de prioridade disponíveis é menor do que o número de tarefas
  - É necessário agrupar várias tarefas no mesmo nível
  - Diminui a escalonabilidade do sistema
- Ideal é que o número de níveis de prioridade seja igual ou maior que o número de tarefas no sistema

2 - Níveis de Prioridade Suficientes

- Exemplo: Três tarefas T1, T2 e T3
- Três níveis de prioridade:
  - Ninguém interfere com T1
  - T1 interfere com T2
  - T1 e T2 interferem com T3
- Dois níveis de prioridade (T2 e T3 juntas):
  - Ninguém interfere com T1
  - T1 e T3 interferem com T2
  - T1 e T2 interferem com T3

3 - Alteração das Prioridades pelo Sistema Operacional

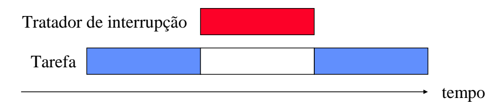
- Muitos sistemas operacionais manipulam por conta própria as prioridades das tarefas
  - Por exemplo, o mecanismo de envelhecimento (*aging*)
  - reduzem automaticamente a prioridade de uma *thread* na medida que ela consome tempo de processador
- Esses mecanismos fazem sentido em um SOPG
  - quando o objetivo é estabelecer um certo grau de justiça
- No contexto de tempo real
  - preocupação com o atendimento dos deadlines
  - tornam maior o esforço de modelagem
  - devem ser evitados

4 - Tratadores de Interrupções

- Interrupções são uma importante fonte de interferência sobre as demais tarefas
- Cada tratador de interrupção é considerado como tarefa de prioridade mais alta do que tarefas normais, apresentando as propriedades
  - Tempo máximo de execução
  - Período ou intervalo mínimo entre ativações

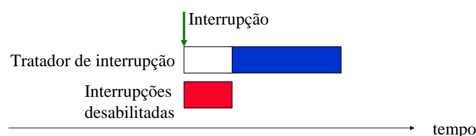
4 - Tratadores de Interrupções

- Embora difícil, o estabelecimento de um intervalo mínimo entre ativações de tratadores é uma necessidade matemática para a análise do sistema (teclado, disco, etc)



5 - Latência dos Tratadores de Interrupções

- Tempo entre a sinalização de uma interrupção no hardware e o inicio da execução de seu tratador
- Atraso no reconhecimento da interrupção pode ser modelado como um *release jitter* associado com cada pseudo-tarefa "tratador de interrupção"



5 - Latência dos Tratadores de Interrupções

- Interrupções podem ter prioridades na arquitetura
  - Pseudo-tarefa tratador de interrupção recebe interferência quando é atrapalhada por um tratador de interrupção mais prioritária
  - Sofre *release jitter* quando atrasa em função de uma tarefa de prioridade mais baixa desabilitar interrupções
- Em alguns trabalhos a prioridade dos tratadores de interrupção também é definida por deadline monotônico, etc.

6 - *Threads* de Kernel

- Sistemas operacionais incluem *threads* de kernel
  - Execução periódica ou esporádica
  - Responsáveis pelas tarefas de manutenção
  - Escrever entradas da *cache,* contabilizações, etc
- Essas *threads* de kernel devem fazer parte do esquema global de prioridades
- Solução paliativa, adotada em alguns sistemas:
  - "anotar trabalhos a serem feitos" em listas
  - depois executá-los em determinador momentos (chamada de sistema ou uma interrupção de hardware)
  - Esta solução de projeto dificulta a modelagem

7 - Implementação das Filas

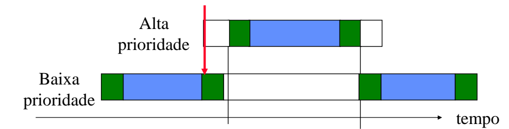
- Tradicionalmente são utilizadas listas encadeadas na implementação das filas
- O processamento de listas encadeadas introduz *overhead* em cenários de pior caso
  - como na liberação simultânea de várias tarefas
- Minimizado através da substituição da tradicional implementação da fila por algoritmos específicos

8 - Tempo de Chaveamento entre Tarefas

- Métrica muito citada no mercado de sistemas operacionais para tempo real
- Na modelagem o tempo de chaveamento pode ser somado ao tempo máximo de execução de cada tarefa
  - Necessariamente, cada ativação de cada tarefa deverá carregar o seu contexto e salva-lo depois

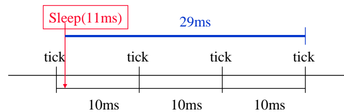
8 - Tempo de Chaveamento entre Tarefas

- Se a tarefa em questão é preemptada por outra, a interferência que ela sofre da outra incluirá o tempo de chaveamento de contexto



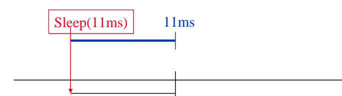
9 - Temporizadores de Alta Resolução

- Temporizadores são utilizados na implementação de *time-out*, *watch-dog*, tarefas periódicas, etc
- Nos SOPG são implementados via interrupções periódicas geradas por relógio de hardware
  - Por exemplo, a cada 10ms uma interrupção é gerada
  - Essa implementação pode gerar erros grosseiros



9 - Temporizadores de Alta Resolução

- SOTR utilizam temporizadores de alta resolução
  - Baseados em interrupções aperiódicas
  - HW gera interrupção no próximo instante de interesse
- Temporizador de alta resolução é esporádico, intervalo mínimo entre ativações depende da dinâmica do sistema

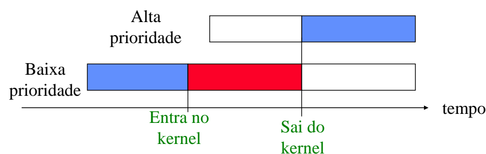


10 - Comportamento das Chamadas de Sistema no Pior Caso

- Em geral, implementação das chamadas de sistema é feita para minimizar o tempo médio
- Aplicações de tempo real são beneficiadas quando o código das chamadas de sistema apresenta bom comportamento também no pior caso
- Maior problema:
  - Dependência que pode haver entre o tempo de execução da chamada de sistema e o estado do SO quando a chamada é feita
  - Quanto mais complexo for o SO, mas difícil será calcular este tempo

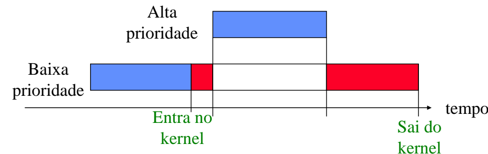
11 - Preempção de Tarefa Executando Código do Sistema

- Um kernel não preemptivo é capaz de gerar grandes inversões de prioridade
  - Corresponde a um bloqueio



11 - Preempção de Tarefa Executando Código do Sistema

- Kernel com pontos de preempção reduz o problema
- Obviamente, um **SOTR deve ser preemptivo**
  - ainda que em determinados momentos interrupções precisem ser desabilitadas em função das estruturas de dados acessadas



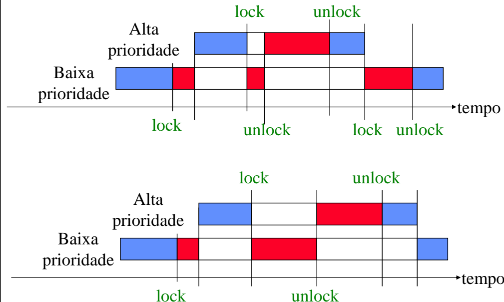
12 - Mecanismos de Sincronização Apropriados

- A literatura de tempo real é rica em mecanimos de sincronização apropriados para tempo real
  - Como mutexes que incorporam herança de prioridades, *priority ceiling*, etc
- Tais mecanismos
  - reduzem a inversão de prioridades dentro do kernel
  - reduzindo o tempo de bloqueio sofrido por tarefas que fazem chamadas de sistema

13 - Granularidade das Seções Críticas dentro do Kernel

- Kernel preemptivo é capaz de chavear imediatamente para a tarefa de alta prioridade quando a mesma é liberada
- Inversão de prioridade dentro do kernel ainda é possível
  - quando a tarefa de alta prioridade faz chamada de sistema
  - mas é bloqueada
  - pois uma estrutura de dados compartilhada
  - foi anteriormente alocada por uma tarefa de mais baixa prioridade
- Mesmo mecanismos apropriados de sincronização não conseguem evitar esta situação
- Granularidade fina para as seções críticas dentro do kernel reduz o tempo de bloqueio

13 - Granularidade das Seções Críticas dentro do Kernel



14 - Gerência de Recursos em Geral

- Processador é apenas um recurso do sistema
- Memória, periféricos, controladores, servidores também deveriam ser escalonados visando atender os requisitos temporais da aplicação
- **Todas as filas** do sistema deveriam respeitar as prioridades das tarefas
- Por exemplo, as requisições de disco
  - deveriam ser ordenadas conforme a prioridade
  - e não para minimizar o tempo de acesso
- Recurso acessado por FCFS gera inversão de prioridade (por toda a fila)

15 - Tratadores de Dispositivos (*Device-Drivers*)

- Tarefa de alta prioridade **é suspensa** quando ocorre uma interrupção de periférico
- Processador passa a executar o tratador de interrupção incluído em ***device-driver*** associado com o periférico em questão

15 - Tratadores de Dispositivos (*Device-Drivers*)

- Sistemas não limitam o tempo de execução desse tratador: tarefa com tempo de execução possivelmente longo e alta prioridade
  - sua alta prioridade é decorrente **não** de uma política de prioridades explícita, **mas** de um efeito colateral da construção do SO
  - Exemplo: Disco e rede no Linux
- Tratadores de interrupção associados com *device-drivers* simplesmente devem liberar *threads* de kernel

16 - ????????

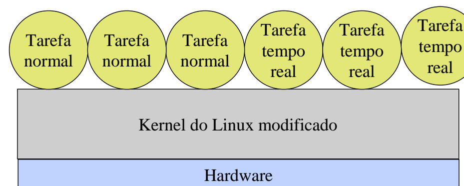
- ????????????

Linux para tempo real

- Existem muitas variações de Linux para tempo real
  - Código aberto, livre
  - Sistema robusto, completo
- Cada variação muda o sistema de alguma forma para melhorar sua previsibilidade temporal
- Cada variação ataca um ou mais aspectos listados
- São *patches* que alteram o kernel de alguma forma
- Maioria das vezes afeta estilo de programação

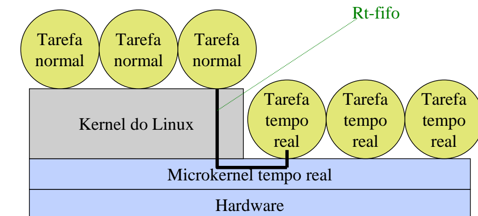
Linux modificado

- Kernel convencional com melhorias:
  - Escalonador apropriado
  - Redução da inversão de prioridades por bloqueio
  - Redução das seções não-preemptivas



Linux para tempo real

- Abordagem bastante eficaz:
  - Microkernel tempo real suportanto o kernel do Linux



Linux para tempo real

- Tarefas normais usam recursos típicos
  - Interface gráfica de usuário
  - Memória virtual
  - Sistema de arquivos
  - TCP/IP
- Tarefas de tempo real executam funções que exigem excelente comportamento temporal
  - Controle de processos industriais
  - Amostragem de variáveis físicas
  - Precisão na ordem de microssegundos em PC comum
  - Exemplo: Máquina de corte a laser

Conclusões

- Na literatura de tempo real
  - existe ampla teoria para a modelagem de um sistema operacional completo
- Para que isto seja possível, são necessárias certas disciplinas de projeto
- Difícil aplicar a modelagem em sistemas operacionais antigos/existentes/que não foram feitos para isto
- Seria viável com sistema operacional completo, desde que construído com algum cuidado ?

Conclusões

- **<u>Questão:</u>** Em alguns anos, existirão sistemas operacionais completos (Linux,Windows), onde
  - equações permitirão
    o <u>cálculo do tempo máximo de resposta</u>
  - mesmo que tarefas utilizem livremente
    <u>os serviços do sistema operacional</u>

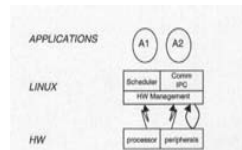
<u>???</u>

Linux de Tempo Real

Abordagem *Nested OS*
(RTAI)

RTAI (Real time Application Interface)

- Linux é um sistema operacional convencional
  - Camada mais baixa gerencia o hardware e lida com interrupções do processador e periféricos
  - Escalonamento preocupa-se com justiça, prioridades, fatias de tempo, podendo lidar com soft real-time
  - Permite comunicação entre processos



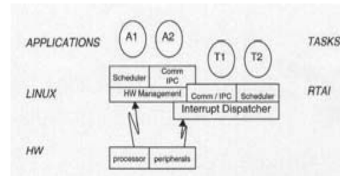
RTAI (Real time Application Interface)

- RTAI é um pequeno executivo desenvolvido no DIAMP (Dipartimento di Ingegneria Aerospaziale del Politecnico di Milano)
- Não é um Sistema Operacional completo
- Torna o kernel do Linux completamente preemptável
- RTAI considera o Linux como uma tarefa de background que executa quando nenhuma atividade de tempo real ocorre

RTAI (Real time Application Interface)

- RTAI é basicamente um dispatcher baseado em interrupções
- As interrupções do processador continuam gerenciadas pelo Linux
- RTAI intercepta interrupções de periféricos e, se necessário, redireciona elas para o Linux
- As funções Cli () e Sti () do RTAI manipulam flags que registram as interrupções que ocorrem
- RTAI registra todas as interrupções e sinaliza elas em um momento apropriado

RTAI (Real time Application Interface)



RTAI (Real time Application Interface)

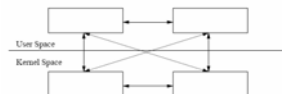
- A arquitetura de software do RTAI inclui:
  - Interface com o Linux HW Management (HAL): basicamente uma estrutura de dados
  - Três componentes básicos (dispatcher, scheduler, fifo's)
  - Uma interface usadas em tarefas de usuário para inicializar e disparar os componentes
- Para o Linux esses elementos aparecem como módulos
  - Um módulo é um componente de kernel, carregável dinamicamente, o qual executa no espaço do kernel.
  - Ele tem os mesmos privilégios e direitos de acesso

Recursos do RTAI

- Semáforos
- Mailboxes
- Memória compartilhada
- FIFOs
- Módulo POSIX
  - Mutexes
  - Variáveis condição
  - Filas de mensagens
  - Pthreads
- Suporta Uniprocessor e SMP
- Suporta Floating-Point para tarefas em espaço de usuário e de tempo real hard
- Implementado como patch do kernel, baixo nível de intrusão

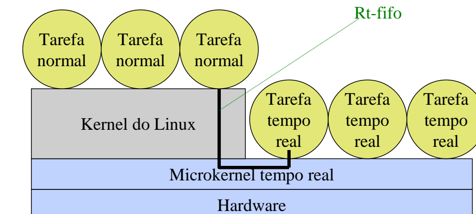
Recursos do RTAI: LXRT

- LX (Linux) RT (Real-Time) module
- Implementa serviços para tornar as funções dos escalonadores disponíveis para processos Linux
- Torna possível compartilhar memória, enviar mensagens usar semáforos:
  - Linux<->Linux
  - Linux<->RTAI
  - RTAI<->RTAI



Resumo RTAI

- Abordagem bastante eficaz:
  - Microkernel tempo real suportando o kernel do Linux



Resumo RTAI

- Tarefas normais usam recursos típicos
  - Interface gráfica de usuário
  - Memória virtual
  - Sistema de arquivos
  - TCP/IP
- Tarefas de tempo real executam funções que exigem excelente comportamento temporal
  - Controle de processos industriais
  - Amostragem de variáveis físicas
  - Precisão na ordem de microssegundos em PC comum
  - Exemplo: Máquina de corte a laser

Linux de Tempo Real

Abordagem *Reduz Latência* (PREEMPT_RT)

Patch PREEMPT_RT

- Existem muitas patches que alteram o kernel do Linux de alguma forma
- Cada variação muda o sistema de alguma forma para melhorar sua previsibilidade temporal
- Cada variação ataca um ou mais aspectos listados
- Principal preocupação: reduzir latência
  - Tempo entre evento e início da execução do código que trata aquele evento

Patch PREEMPT_RT

- Objetivos
  - Resposta rápida em multiprocessadores convencionais de médio porte
  - Mesmo código base do kernel do Linux
  - Fazer 20% do trabalho que seria necessário para conseguir então atender 80% das aplicações de tempo real hoje
- Não são objetivos
  - Garantia para tempos de resposta (hard real-time)
  - Resposta em tempo real para todos os serviços
- Além disto:
  - Uso de mecanismos convencionais dentro do kernel
  - Suporte ao POSIX
  - Escalabilidade e desempenho além de determinismo temporal

Patch PREEMPT_RT

- Ponto chave: minimizar a quantidade de código que não é preemptável
- Ao mesmo tempo que minimiza a quantidade de código que precisa ser modificada para prover um maior nivel de preempção
- O patch PREEMPT_RT aproveita as capacidades SMP do kernel do Linux para aumentar o nível de preempção sem re-escrever o kernel completamente

Patch PREEMPT_RT

- Seções críticas dentro do Kernel que não eram preemptáveis agora o são
- Tratadores de interrupção podem ser preemptados
  - Quase todos os tratadores de interrupção executam no contexto de um processo no ambiente do PREEMPT_RT

Patch PREEMPT_RT

- Seqüências de código com “interrupções desabilitadas” podem ser preemptadas
  - Usa a capacidade SMP do kernel do Linux para tratar corridas com outros tratadores de interrupções
  - Maioria dos tratadores de interrupções executam em contexto de processo
  - Qualquer código que interage com um tratador de interrupção deve estar preparado para lidar com um tratador de interrupções executando concorrentemente em alguma outra CPU

Patch PREEMPT_RT

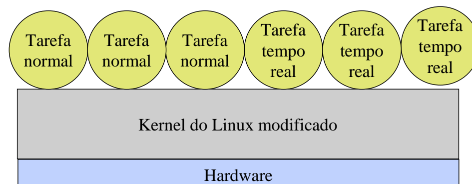
- Herança de prioridade implementada para spinlocks e semáforos dentro do kernel
- Operações postergadas
  - Mudanças geraram alguns problemas em casos particulares
  - Em alguns casos, operações que necessitam de sincronização precisam ser postergadas até que o sistema esteja em estado seguro
- Outras medidas para reduzir a latência
  - Considera peculiaridades dos processadores, como instruções de máquina especiais (x86 MMX/SSE)

Patch PREEMPT_RT

- Linux está fazendo um grande progresso no que diz respeito ao tempo de latência
- Objetivos técnicos modestos, procurando maximizar a utilidade
- Dezenas de microsegundos como latência para interromper/escalonar
- Latências similares para algumas operações operações e chamadas de sistema
- Tenta usar a mesma base de código do kernel
- Simplicidade, escalabilidade e desempenho pouco prejudicados
- Sem garantia para as latências, ferramentas de software podem ajudar ?

Resumo Patch PREEMPT_RT

- Kernel convencional com melhorias:
  - Escalonador apropriado
  - Redução da inversão de prioridades por bloqueio
  - Redução das seções não-preemptivas



Resumo Patch PREEMPT_RT

- Possibilidades futuras nesta abordagem
  - E/S determinística
  - Sistema de arquivos determinístico (Flash memory)
  - Protocolos de rede (UDP)
  - Priority Inheritance nos mecanismos de bloqueio
  - Gerência de memória ?