

SISTEMAS DE TEMPO REAL

Universidade de Araraquara – UNIARA
Prof. Rodrigo D. Malara

Adaptado de Bruno Tadeu Caetano, Fellipe Gustavo Forte, Luis Fernando de Oliveira Jacintho, Jair Jonko Araujo

Conteúdo

Introdução

Classificações

Sistemas operacionais e tempo real

Escalonamento

Modelos de escalonamento

Exemplos de SOTR

Introdução

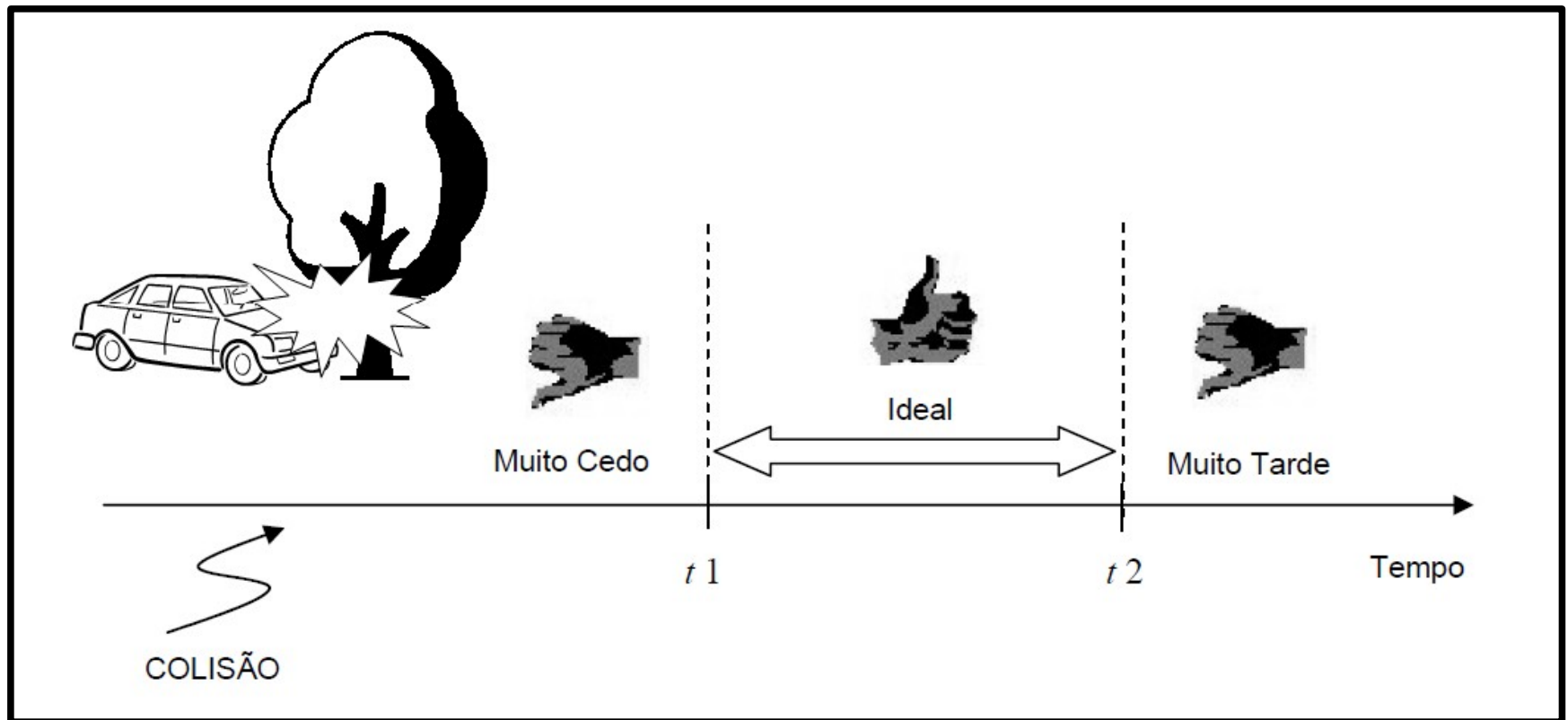
- Computação Convencional (não-tempo-real)
 - Programa P é executado com um entrada i , gerando uma saída o em um tempo t .
 - $o = P(i), t$
 - Menor t : maior performance
- Computação de tempo-real
 - Computação convencional com uma imposição a t
 - $o = P(i), t < D$
 - D recebe o nome de deadline. O programa deve obrigatoriamente ser executado antes do deadline.

Introdução

- Sistemas de tempo real (STR)
 - Sistemas reativos ao ambiente
 - Comportamento previsível atendendo características temporais impostas pelo ambiente ou pelo usuário, mesmo com recursos limitados.
 - Algoritmos de escalonamento bem definidos
- Objetivo
 - Executar o aplicativo no tempo correto.

Introdução

Airbag



Classificação

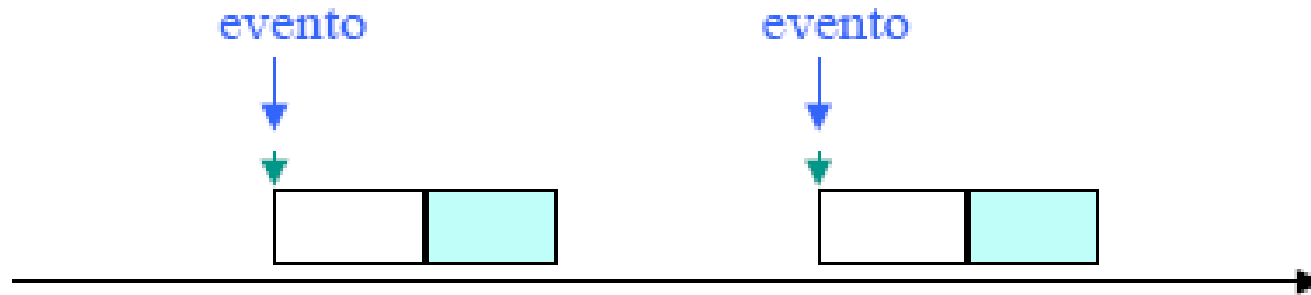
- Em geral apresentam duas classificações
 - Reativos
 - O escalonamento é dirigido pela interação com seu ambiente.
 - Embutidos
 - Fazem partes de sistemas maiores não computacionais
- Críticos (*Hard real time*)
 - *Falha temporal pode resultar em consequências catastróficas*
 - *Necessário garantir requisitos temporais em projeto*
 - *Exemplo: usina nuclear, indústria petroquímica, mísseis*
- Não-críticos (*Soft real time*)
 - *Elevada disponibilidade*
 - *Elevada integridade*

Sistemas operacionais de tempo real

- Permitem especificar, verificar e implementar sistemas ou programas que, mesmo com recursos limitados, apresentam comportamentos previsíveis e atendem restrições temporais impostas pelo usuário ou pelo ambiente.
- **Sistemas operacionais de tempo real (SOTR)**
 - Sistemas que auxiliam, um escalonamento de tarefas que cumpre com deadlines.
 - O elemento chave que diferencia um SOTR de um SO convencional é o escalonador de processos.

Conceitos Básicos

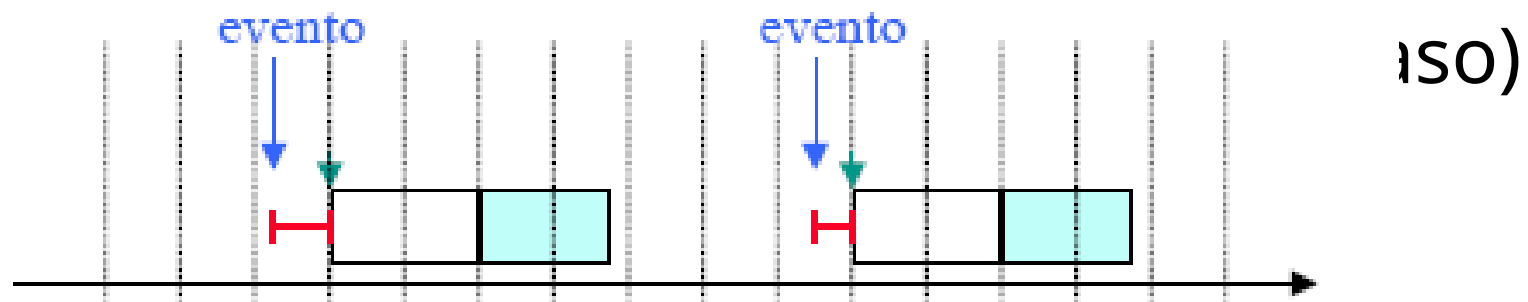
- Tarefas disparadas por Eventos (Event-Triggered)
 - Evento externo gera interrupção e



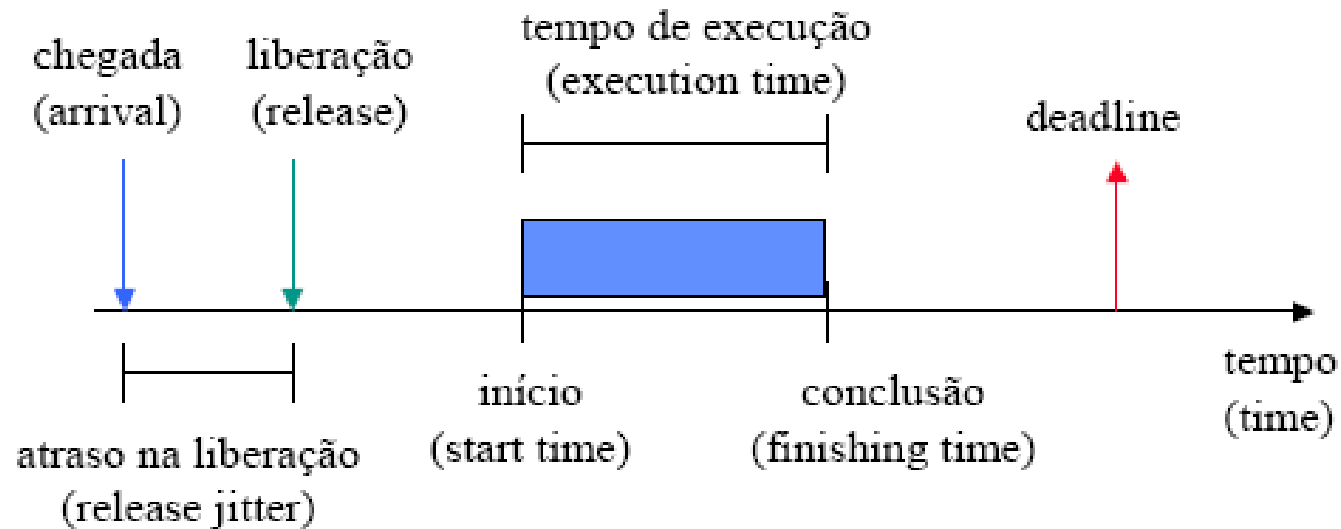
- Funciona bem com carga pequena
- Pode falhar no caso de carga pesada

Conceitos Básicos

- Eventos disparados de tempo em tempo (Time-Triggered)
- Interrupção de relógio a cada T milisegundos (tick)
- A cada tick alguns sensores e atuadores são acessados
- Não existem interrupções além das do relógio



Propriedades Temporais das Tarefas



- $\text{Folga} = \text{Deadline} - \text{Liberação} - \text{Tempo de execução}$
- $\text{Atraso} = \text{MAX}(0, \text{Conclusão} - \text{Deadline})$
- $\text{Tempo de resposta} = \text{Conclusão} - \text{Chegada}$

Tarefas Periódicas

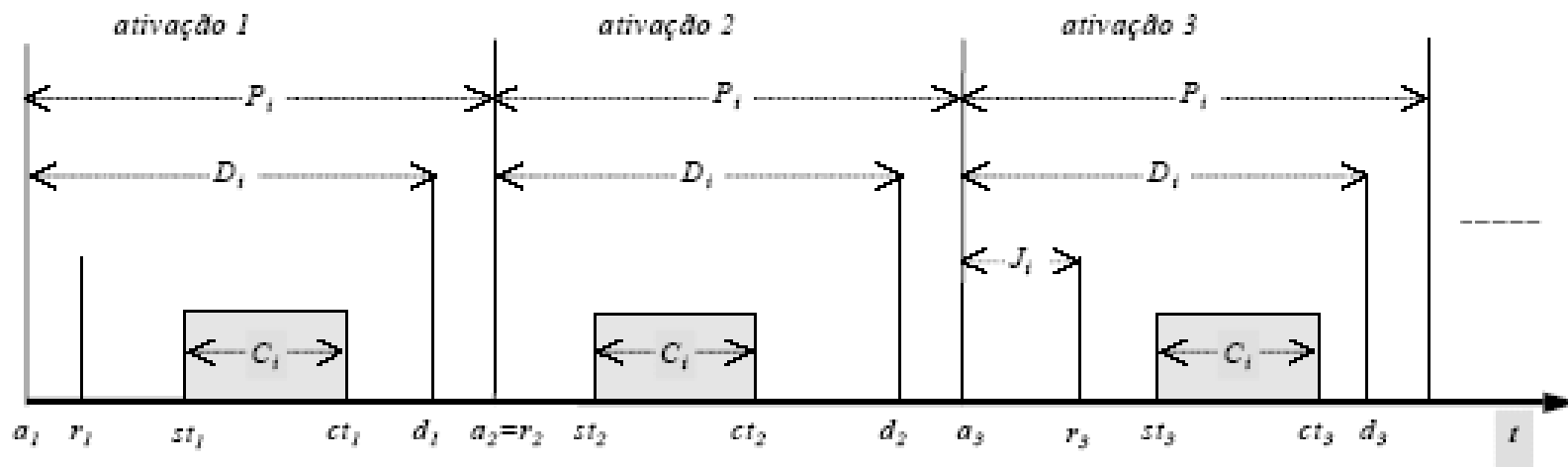


Figura 2.1: Ativações de uma tarefa periódica

- Tarefa é ativada a cada P unidades de tempo
- Instantes de chegada podem ser calculados a partir do inicial
- Exemplo: controle de processo via laço de realimentação

Tarefas Esporádicas

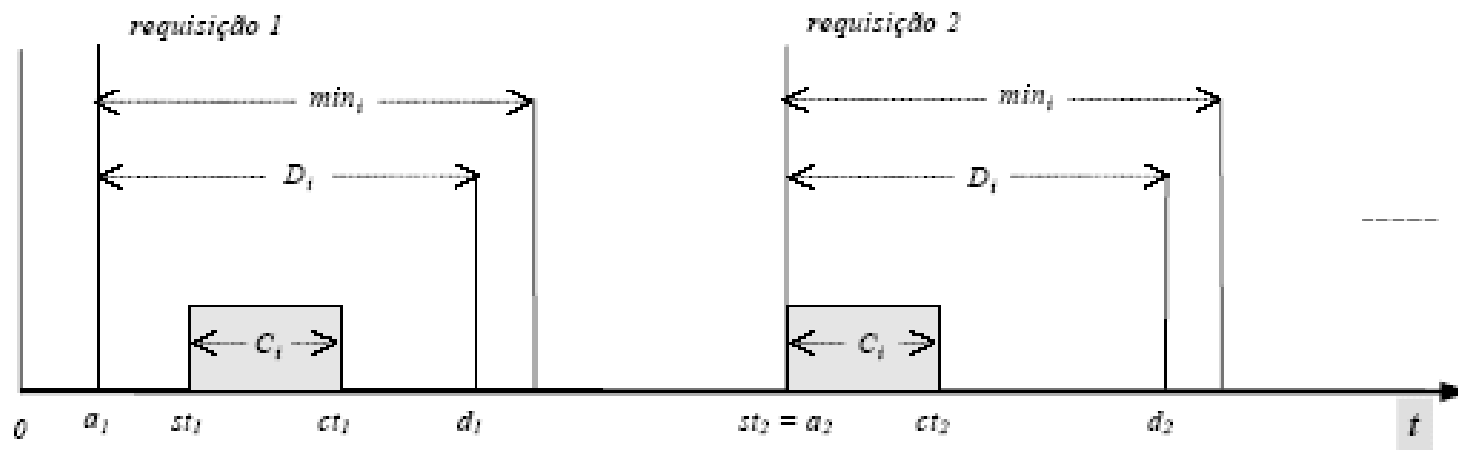


Figura 2.2: Ativações de uma tarefa esporádica

- Instantes de chegada não são conhecidos
- Existe um intervalo mínimo de tempo entre chegadas
- Exemplo: atendimento a botão de alarme
 - Aparecimento de objeto em tela de radar

Conceitos Básicos

- Relações de exclusão mútua entre tarefas
 - Tarefas A e B apresentam exclusão mútua quando NÃO podem executar simultaneamente
 - Exemplos:
 - Estrutura de dados compartilhada
 - Arquivo
 - Controlador de periférico

Escalonamento

- Ordem de execução que processos em **pronto** utilizarão o processador.
- Uma escala é dita ótima se sua organização é a melhor possível no atendimento das restrições temporais.
- Problema: Criar essa ordem.
- Propriedades relacionadas ao tempo devem ser levadas em conta.

Escalonamento

- *Release time*
 - Tempo em que um processo está pronto para ser processado
- *Deadline - prazo*
 - Tempo máximo para que um processo complete sua execução
- *Tempo de Execução no Pior Caso (Worst case execution time)*
 - Tempo máximo estabelecido em tempo de projeto, para a completa execução de um processo/tarefa.
- Período
 - Intervalo em que determinado processo deve se repetir
- Prioridade

Modelos de Escalonamento

- Modelos de escalonamento para tempo real
[Liu e Layland, 1973]
 - *Rate Monotonic* (RM)–Período menor→maior prioridade
 - Monoprocessado, com prioridade fixa e preemptivo
 - Escala produzida *off-line* ou em tempo de projeto
 - Prazo Final mais Cedo (*Earliest Deadline First* - EDF)
 - Processo Prazo Final mais Cedo deve executar primeiro
 - Orientado a prioridade com atribuição dinâmica
 - Escala produzida *on-line* ou em tempo de execução
 - *Deadline Monotonic* (DM)–Deadline menor→maior prioridade
 - Similar ao RM

Modelos de Escalonamento - RM

□ Rate Monotonic

- Premissas

- As tarefas são periódicas e independentes
- O deadline de cada tarefa coincide com seu período ($D_i = P_i$)
- O tempo de computação C_i de cada tarefa é conhecido e constante (*Worst Case Execution Time*)
- O tempo de troca de contexto entre as tarefas é assumido como nulo.

- Prioridades decrescem em função do aumento de período.

- Quanto menor a prioridade numérica mais frequente a execução do processo

Modelos de Escalonamento - RM

□ Rate Monotonic - Quantum 40ms

<i>Tarefas periódicas</i>	<i>Período</i>	<i>Tempo de Computação</i>	<i>Prioridade RM</i>
A	100 ms	20 ms	1
B	150 ms	40 ms	2
C	350 ms	100 ms	3

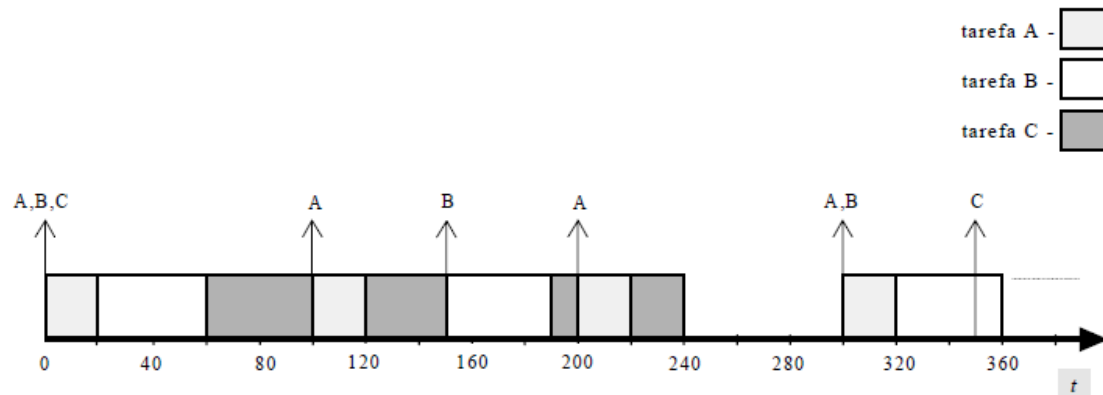


Figura 2.5: Escala RM produzida a partir da Tabela 2.1

Modelos de Escalonamento - RM

□ Rate Monotonic - Quantum 40ms

Tarefas periódicas	Período	Tempo de Computação	Prioridade RM
A	100	20	1
B	150	40	2
C	350	100	3

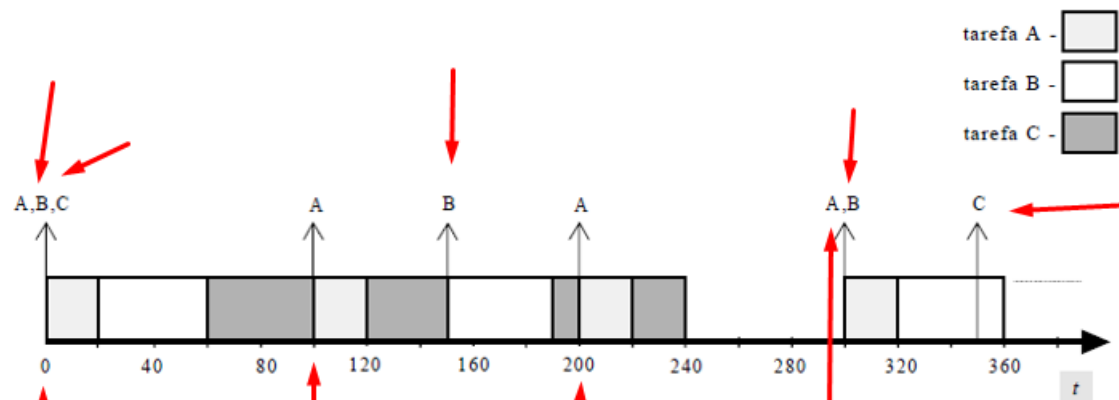


Figura 2.5: Escala RM produzida a partir da Tabela 2.1

Modelos de Escalonamento - RM

□ Vantagens

- Simplicidade
- Rate Monotonic Scheduling é ótimo para a classe de problemas na qual ele se encontra inserido

□ Desvantagens

- A maioria dos problemas não possuem tarefas totalmente independentes.
- Pode causar deadlock e problemas de inversão de prioridades.

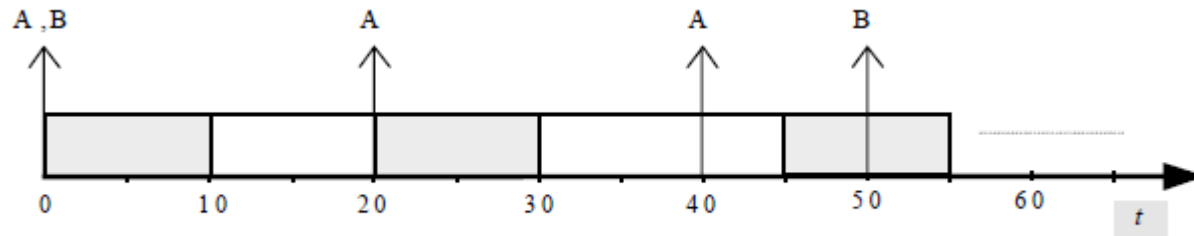
Modelos de Escalonamento - EDF

- EDF *Earliest Deadline First* (EDF)
 - Processos com prazo final mais cedo devem executar primeiro
- Premissas:
 - As tarefas são periódicas e independentes
 - O deadline de cada tarefa coincide com seu período ($D_i = P_i$)
 - O tempo de computação C_i de cada tarefa é conhecido e constante (*Worst Case Execution Time*)
 - O tempo de troca de contexto entre as tarefas é assumido como nulo.
- Ordenação ocorre segundo seus *deadlines* absolutos

Modelos de Escalonamento - EDF

□ EDF

<i>Tarefas periódicas</i>	<i>Período</i>	<i>Tempo de Computação</i>	<i>Deadline</i>
A	20 ms	10 ms	20 ms
B	50 ms	25 ms	50 ms



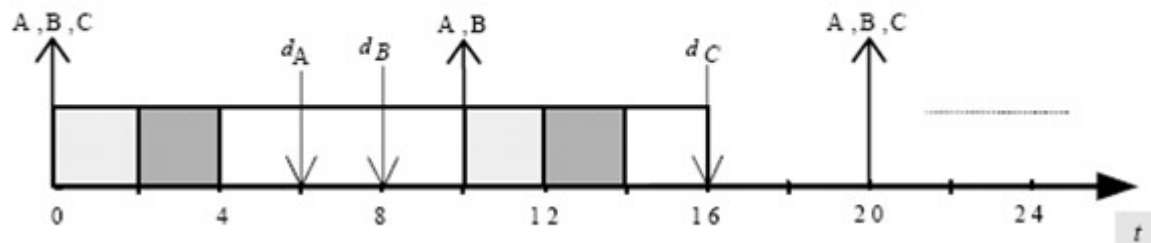
Modelos de Escalonamento - DM

- DM ou Deadline Monotonic (DM)
 - Premissas:
 - As tarefas são periódicas e independentes
 - O *deadline* de cada tarefa menor ou igual seu período ($D_i \leq P_i$)
 - O tempo de computação C_i de cada tarefa é conhecido e constante (*Worst Case Execution Time*)
 - O tempo de chaveamento entre as tarefas é assumido como nulo.
 - Prioridade baseada nos *deadlines* relativos das tarefas.

Modelos de Escalonamento - DM

□ DM



<i>Tarefas periódicas</i>	<i>Deadline</i>	<i>Período</i>	<i>Tempo de Computação</i>	<i>Prioridade DM</i>
A	6 ms	10 ms	2 ms	1
B	8 ms	10 ms	2 ms	2
C	16 ms	20 ms	8 ms	3

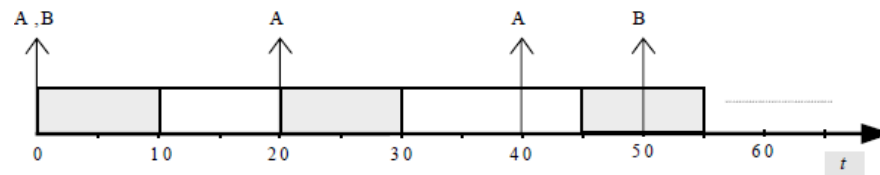


Modelos de Escalonamento

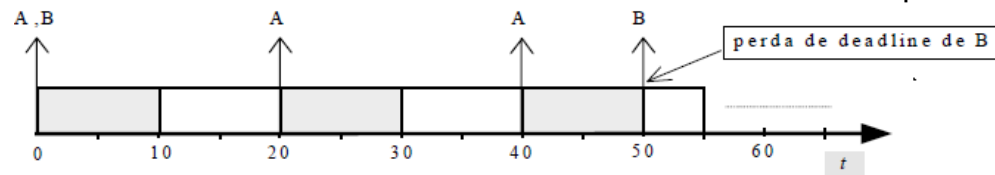
Qual é o melhor?

<i>tarefas periódicas</i>	C_i	P_i	D_i
tarefa A	10	20	20
tarefa B	25	50	50

tarefa A - 
tarefa B - 



(a) Escalonamento EDF



(b) Escalonamento RM

Ausência de um SOTR

Sistemas microcontrolados

Software sem SO de Tempo Real

- Superloop com processos background/foreground:
- **Background:** A aplicação consiste em um loop infinito que chama algumas funções para realizar as operações desejadas.
- **Foreground:** Rotinas de tratamento de interrupção tratam eventos assíncronos .
- Rotina de Tratamento de Interrupção ou Interrupt Service Routine (ISR)

Software sem SO de Tempo Real

0 SUPER-LOOP

```
int main(void)
{
    init_hardware();

    while (1) {
        I2C_ReadPkg();
        UART_Receive();
        USB_GetPacket();
        Audio_Play();
        ADC_Convert();
        LCD_Show();
        ...
    }
}
```

BACKGROUND

```
void USB_ISR(void)
{
    Limpa interrupção;
    Lê pacote;
}

void UART_ISR(void)
{
    Limpa interrupção;
    Trata byte recebido;
}
```

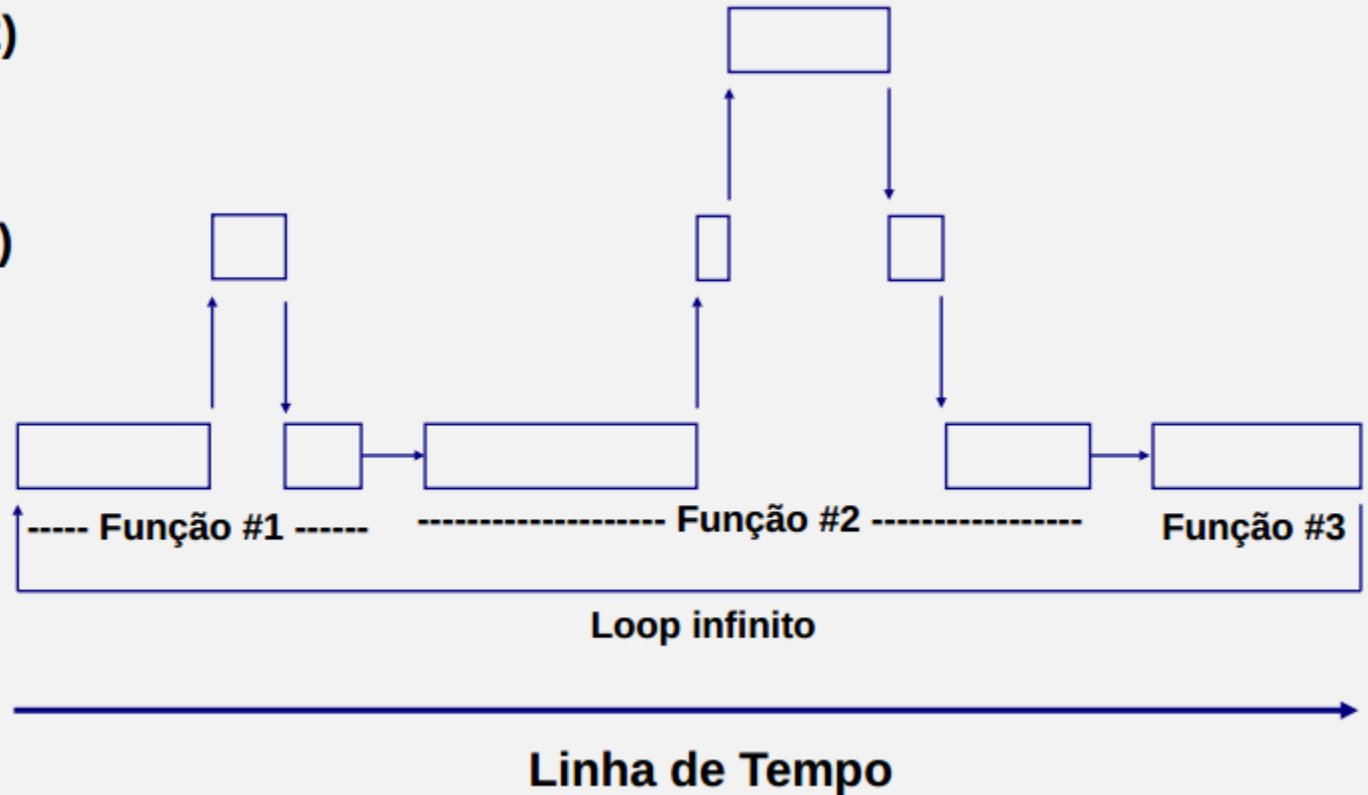
BACKGROUND

Software sem SO de Tempo Real

Foreground (ISR2)

Foreground (ISR1)

Background



Vantagens do Super-Loop

- Fácil e rápido de desenvolver.
- Solução ótima em projetos pequenos e com requisitos modestos de restrições de tempo.
- Não requer treinamento ou conhecimento de API's específicas de um sistema operacional.
- Não consome recursos adicionais comparado à solução com um sistema operacional.

Deficiências do Super-Loop

- Difícil garantir que uma operação irá ser executada dentro das restrições de tempo.
- Todo o código em background tem a mesma prioridade.
- Se uma das funções demorar mais do que o esperado, todo o sistema será impactado.

Deficiências do Super-Loop

```
while (1) {  
    ADC_Read();  
    UART_Receive();  
    USB_GetPacket();  
    ...  
}  
  
void ADC_Read (void) {  
    configure_ADC();  
    while (conv_rdy == 0) {  
        ;  
    }  
    ...  
}
```

Delays e outras rotinas podem impactar todas as funções rodando em background.

Deficiências do Super-Loop

- Tarefas de alta prioridade precisam ser colocadas em foreground (ISR).
- ISRs muito longas podem impactar o tempo de resposta do sistema.
- É difícil coordenar a execução de rotinas em background e em foreground.

Deficiências do Super-Loop

- Qualquer alteração em determinada parte do código pode impactar o tempo de resposta de todo o sistema.
- Difícil de garantir as restrições de tempo da aplicação.
- Sentimento de “medo” para alterar o código.

Exemplos de SOTR

VxWorks 6

Windows CE.NET

RTLinux

VxWorks 6

- Desenvolvido pela Wind River
 - Fundada em 1983
 - Comprada pela Intel em 2009
 - Vendida a TPG Capital em 2018
 - Primeira Sonda da NASA com VxWorks lançada em 1995
- Usado frequentemente em sistemas que envolvem robótica, aviação, sistemas de controle médico, simuladores aeroespaciais e controle bélico
 - NASA, Departamento de Defesa dos EUA, Agência Espacial Europeia, BMW, Nissan

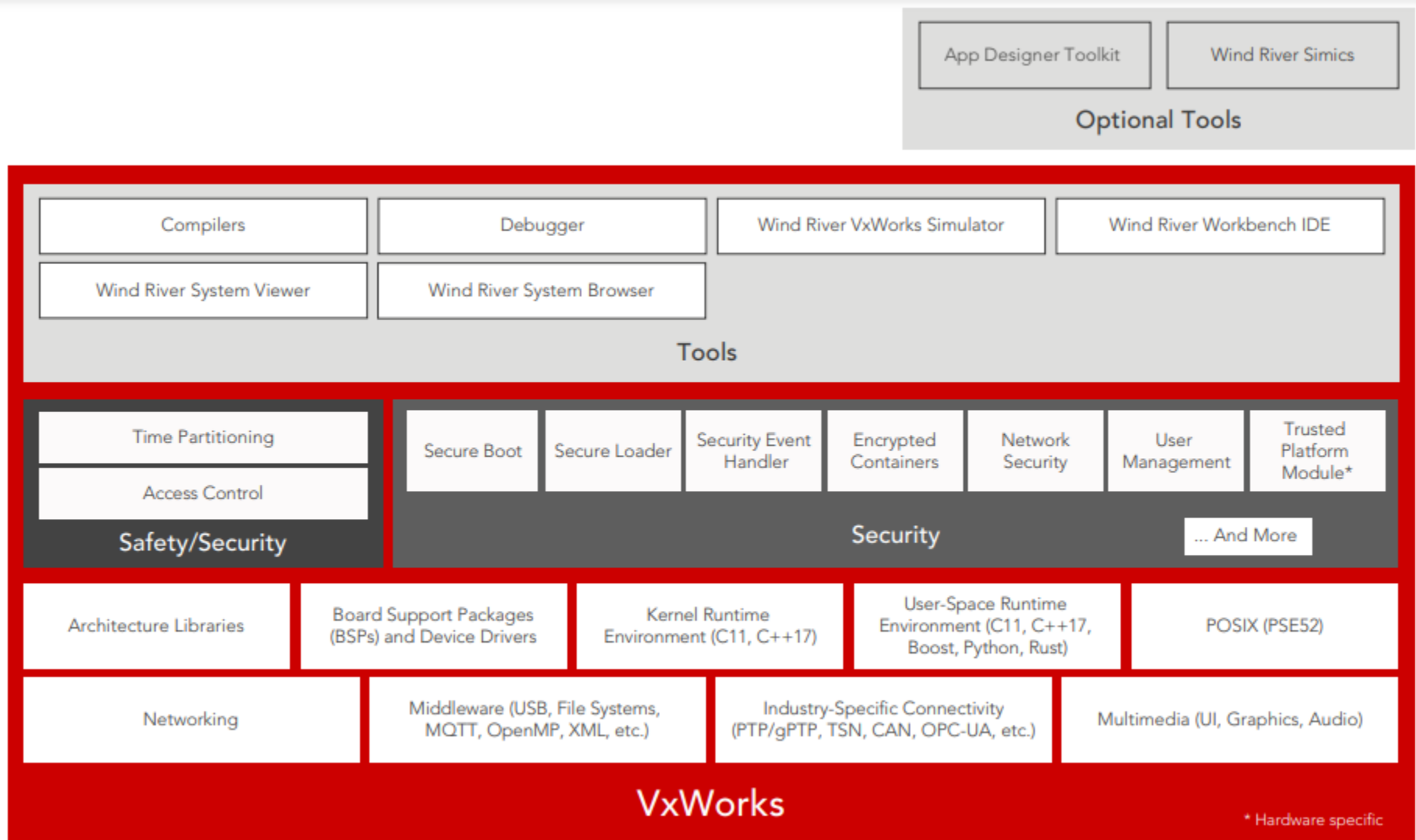
VxWorks6

- VxWorks é um RTOS hard real-time para aplicações embarcadas
- Ele é um RTOS determinístico, baseado em prioridades e preemptivo com baixa latência e mínimo jitter
- O seu Microkernel provê suporte multitarefas, interrupção por software e hardware, mecanismos de comunicação entre tarefas (memória compartilhada, troca de mensagens, semáforos)
- <https://www.windriver.com/inspace/>

VxWorks6

- VxWorks suporta processadores de 32 e 64 bits, multi-core incluindo Intel®, Arm®, Power Architecture® e RISC-V.
- Seu abrangente suporte multi-núcleo permite configurações para processamento simétrico (SMP) e assimétrico (AMP) e afinidade de CPU (BMP ou bound multiprocessing)
- Possui um Sistema de arquivos tolerante a falhas (HRFS – Highly Reliable File System)
- <https://resources.windriver.com/vxworks/vxworks-product-overview>

VxWorks6



VxWorks

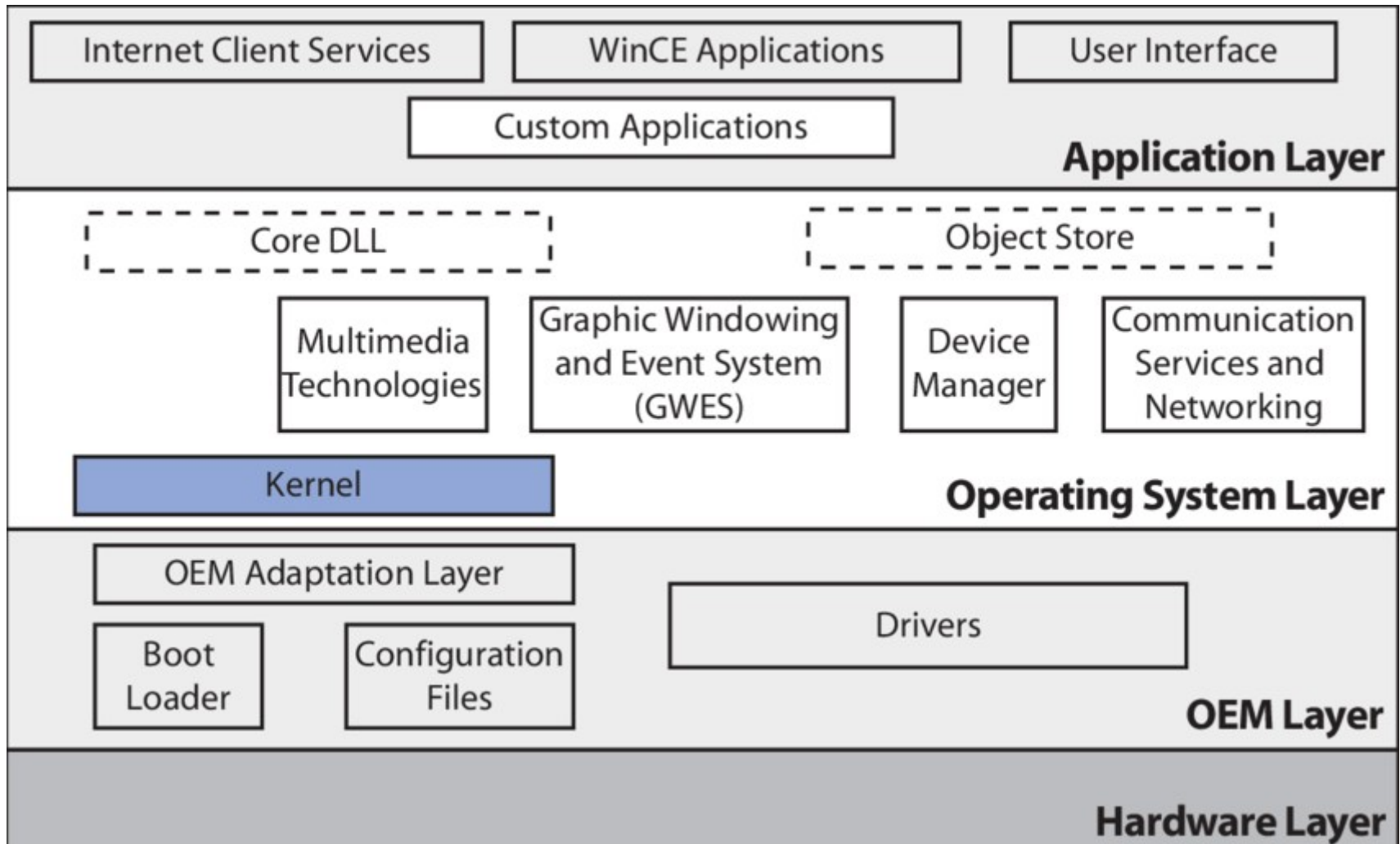
□ Limitações:

- Limite de tarefas é definido pela quantidade de memória disponível;
- Não permite uso de armazenamento secundário como memória (swapping)
- Compatibilidade limitada com outros SOTR, baseando-se apenas no padrão POSIX
 - Portabilidade entre sistemas baseados em UNIX

Windows CE versão .NET

- Sistema aberto, escalonável, de 32-bits que integra recursos de tempo real com tecnologias já conhecidas como a API dos sistemas operacionais *Windows*, usada como interface de programação de aplicativos
- Projeto modular da arquitetura
 - Isola cada processo impedindo assim a interferência em outro processo em caso de falhas ou mau funcionamento do sistema

Windows CE versão .NET



Windows CE versão .NET



Windows CE versão .NET

□ Camada de hardware:

- é onde o sistema operacional faz interação com a plataforma, na qual tem relação direta com a camada de hardware (OEM), que trata dos Controladores de Dispositivos (*Devices Drivers*), o inicializador do sistema operacional (*Boot Loader*) e os arquivos de configuração além da Camada de adaptação do OEM (BIOS).

□ Camada de adaptação do OEM:

- Para Arquiteturas computacionais alternativas
- responsável pelo carregamento do sistema operacional na memória e o processo de inicialização

Windows CE versão .NET

- **Camada do Sistema Operacional:**
 - contém o *kernel* do SOTR, além dos principais serviços de multimídia, comunicação e gerenciamento de dispositivos.

- Camada de Aplicação:
 - Modo 'User'
 - aloca as interfaces gráficas com usuário e aplicativos, além de serviços de *Internet*.

Windows CE versão .NET

- *Kernel* usa um sistema de paginação baseado em memória-virtual para gerenciar e alocar memória para os programas.
- bastante flexível
 - construído sobre um conjunto de módulos, cada um possui uma funcionalidade específica
 - sua versão mais compacta requer apenas 200 KB de ROM.

Windows CE versão .NET

- Sincronização
 - regiões críticas, *mutex*, eventos e semáforos.
- Escalonador
 - baseado no *Round-robin* com fatia de tempo (*time-slice* ou *quantum*) ajustável pelo projetista do sistema. Possui 256 níveis de prioridade e pode ter 32000 processos executando simultaneamente.

Comparação com VxWorks

SOTR	Hardware suportado	Suporte a SMP	Políticas de Escalonamento	Proteção de memória	Prioridades
VxWorks 6	Motorola 68k/CPU32, ColdFire, PowerPC, Intel x86/Pentium/IA-32, ARM/StrongARM, Hitachi, SuperH, SPARC, i960, DSP, xScale e MIPS.	Sim.	Prioridade fixa e escalonamento round-robin (time sharing).	Não possui.	256 níveis de prioridades. Limite de tarefas executando limitado a quantidade de memória disponível.
Windows CE versão .NET	x86, MIPS, SH.	Não.	Prioridade fixa, round-robin (time sharing).	Sim, inclusive com alocação dinâmica de memória.	256 níveis de prioridades. 32 processos executando.

RTLinux

- Funciona sob o kernel do Linux
- É um RTOS hard real time
- Tarefas em tempo real devem rodar com o mínimo de latência e o mínimo jitter.
- O sistema deve ser simples, previsível e o mais perto possível da máquina.
- Permite ao programador escrever seu próprio escalonador.
- "Controlling a laser with Linux is crazy, but everyone in this room is crazy in his own way. So if you want to use Linux to control an industrial welding laser, I have no problem with it if your using PREEMPT_RT." -- Linus Torvalds

RTLinux

- Impossibilidade do Kernel do Linux ser usado como RTOS
- Escalonamento imprevisível - depende da carga do sistema.
- Resolução grosseira do temporizador (10 ms)
- Kernel não preemptivo – A interrupção de relógio não funciona quando o kernel está em execução.
- Desativação de interrupções usadas para sincronização de baixa granularidade.
- Uso de memória virtual
- Reordenação de solicitações por eficiência (por exemplo, para E/S de disco)

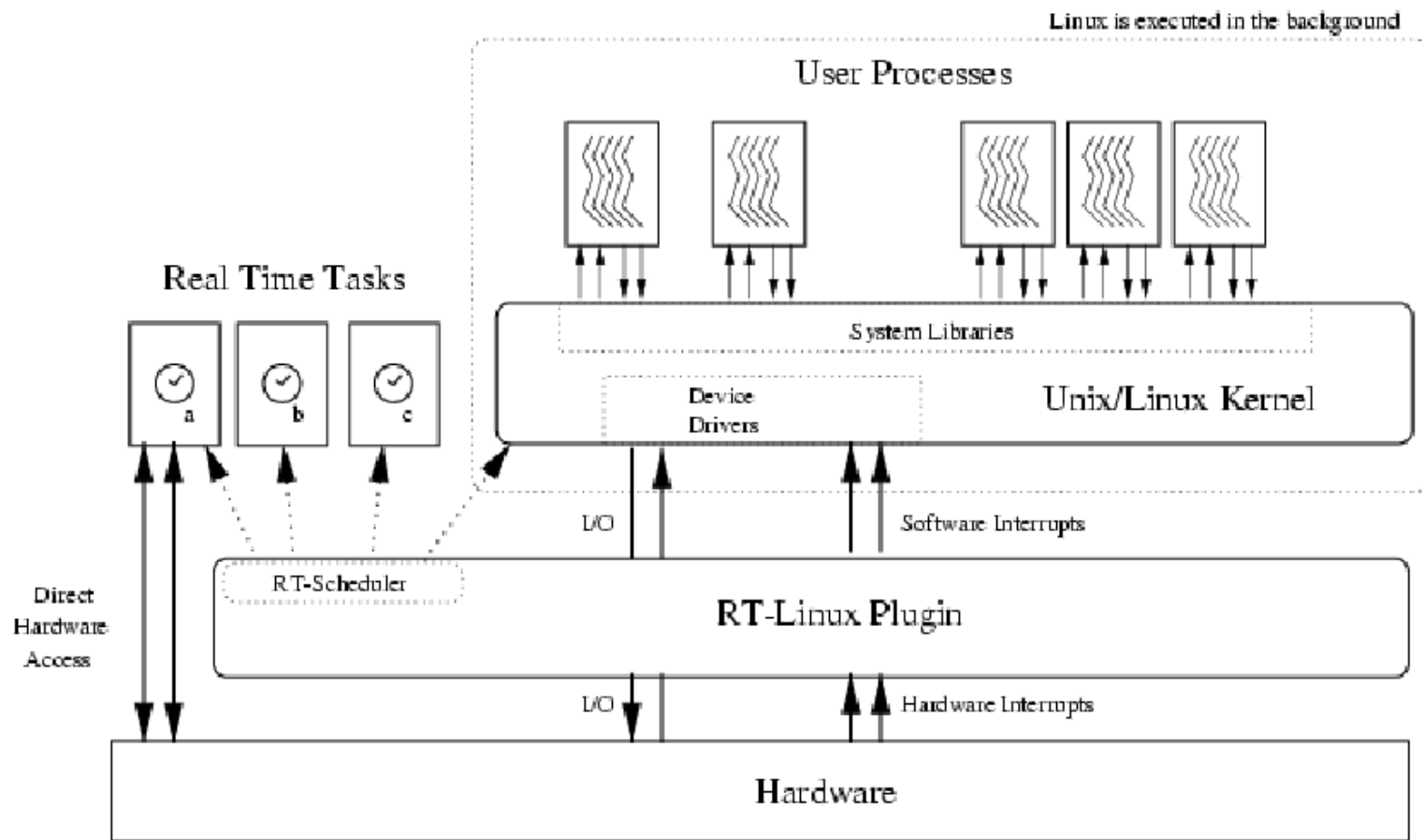
RTLinux

- Inspirado por MERT (Bell Labs 1978) - um conceito de máquina virtual (VM) completo.
- RT Linux usa o conceito de VM limitada a emulação de interrupções.
- Linux se torna uma tarefa para este sistema operacional.
- Uma camada de software de emulação entre o kernel Linux e o hardware do controlador de interrupção.
- Impede a desativação de interrupções pelo Linux.

RTLinux

- No RTLinux
- O kernel é considerado uma camada entre o kernel padrão do Linux e o hardware.
- O kernel do Linux vê a camada de tempo real como o próprio hardware.
- Kernel do RTLinux atribui prioridade mais baixa para o Kernel padrão do Linux.
- RTLinux não é preemptivo.
- A inicialização de uma tarefa em tempo real informa ao Kernel seu deadline e seu período.

RTLinux



RTLinux kernel



Estudo de Caso

Estudo de Caso

- Controle de Tráfego Aéreo
 - São aplicações importantes que consistem em rastrear todas as aeronaves em um espaço aéreo monitorado e garantir que cada uma delas mantenha uma distância segura de separação uma das outras.
 - Radar rastreia a posição da aeronave no espaço.
 - Comunicação.
 - Interação com o operador.

Estudos de Caso

