Implementação de Tarefas em Sistemas Pequenos

Fundamentos dos Sistemas de Tempo Real

Rômulo Silva de Oliveira

Edição do Autor, 2018

www.romulosilvadeoliveira.eng.br/livrotemporeal

Outubro/2018

Sumário

- Executivo Cíclico
- Tratadores de Interrupções
- Laço Principal com Tratadores de Interrupções
- Microkernel Simples

Introdução

- Tempo de resposta de uma tarefa de tempo real está diretamente associado com a forma adotada para implementá-la
  - Depende da **organização dos fluxos de execução**
  - Do **design do software do sistema**
- Implementação de tarefas em sistemas computacionais simples
  - Executivo cíclico
  - Laço principal com interrupções
  - Microkernel simples

Executivo Cíclico 1/19

- Os sistemas mais simples podem ser construídos de tal forma que existe apenas um único fluxo de controle no sistema
- Todo o sistema consiste de um grande laço que sempre é repetido periodicamente
- Tipo de solução chamada de **Executivo Cíclico** (***Cyclic Executive***)
- Trata-se de **escalonamento dirigido por tempo (*clock-driven scheduling*)**

Executivo Cíclico 2/19

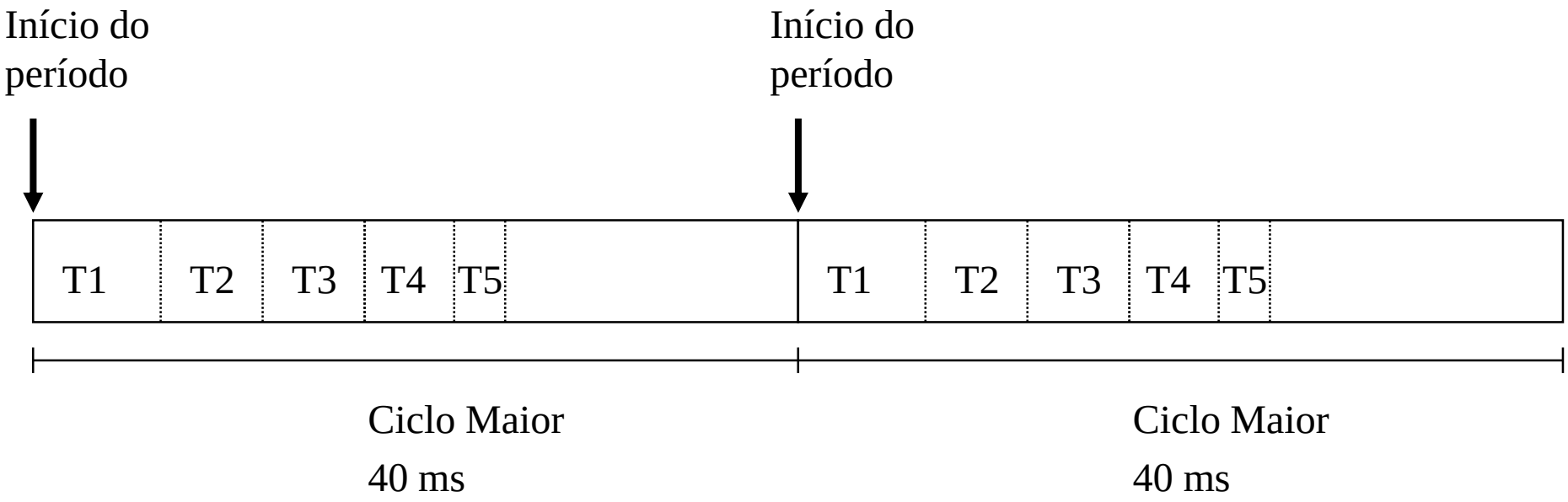
- O período de repetição do laço controlado através de um temporizador em hardware (*timer*)

```
CicloMaior = 40 ms
While( true ) {
      Espera_próximo_ciclo_maior_iniciar( );
      funcao_tarefa_1( );
      funcao_tarefa_2( );
      funcao_tarefa_3( );
      funcao_tarefa_4( );
      funcao_tarefa_5( );
}
```

Executivo Cíclico 3/19

- Exemplo com 5 tarefas, cada uma mapeada para uma função
- O tempo de execução de cada tarefa pode ser diferente
- Porém todas elas executam com o mesmo período
  - 40ms no caso do exemplo
- É necessário garantir que a soma dos tempos de execução no pior caso de todas as tarefas seja menor que 40ms
- O período de tempo no qual todas as execuções se repetem é chamado de **Ciclo Maior** (***major cycle***)

Executivo Cíclico 4/19



Executivo Cíclico 5/19

- Tarefas podem ter períodos diferentes
- Por exemplo, a tarefa que executa a estratégia de controle de um motor elétrico precisará executar com maior frequência (menor período) do que a tarefa que atualiza o display
- Suponha sistema com 5 tarefas, mas cujos períodos e tempos de execução no pior caso sejam:

Tarefa τi	Período Pi	Tempo de computação no pior caso Ci
τ1	20	8
τ2	20	7
τ3	40	4
τ4	40	3
τ5	80	2

Executivo Cíclico 6/19

- Períodos diferentes podem ser acomodados no executivo cíclico
- Através da divisão do ciclo maior em um número inteiro de ciclos menores
- Dentro de cada **Ciclo Menor** (***minor cycle***)
  apenas algumas tarefas executam
- A cada ciclo maior tudo se repete

Executivo Cíclico 7/19

- O código abaixo utiliza 4 ciclos menores dentro do ciclo maior

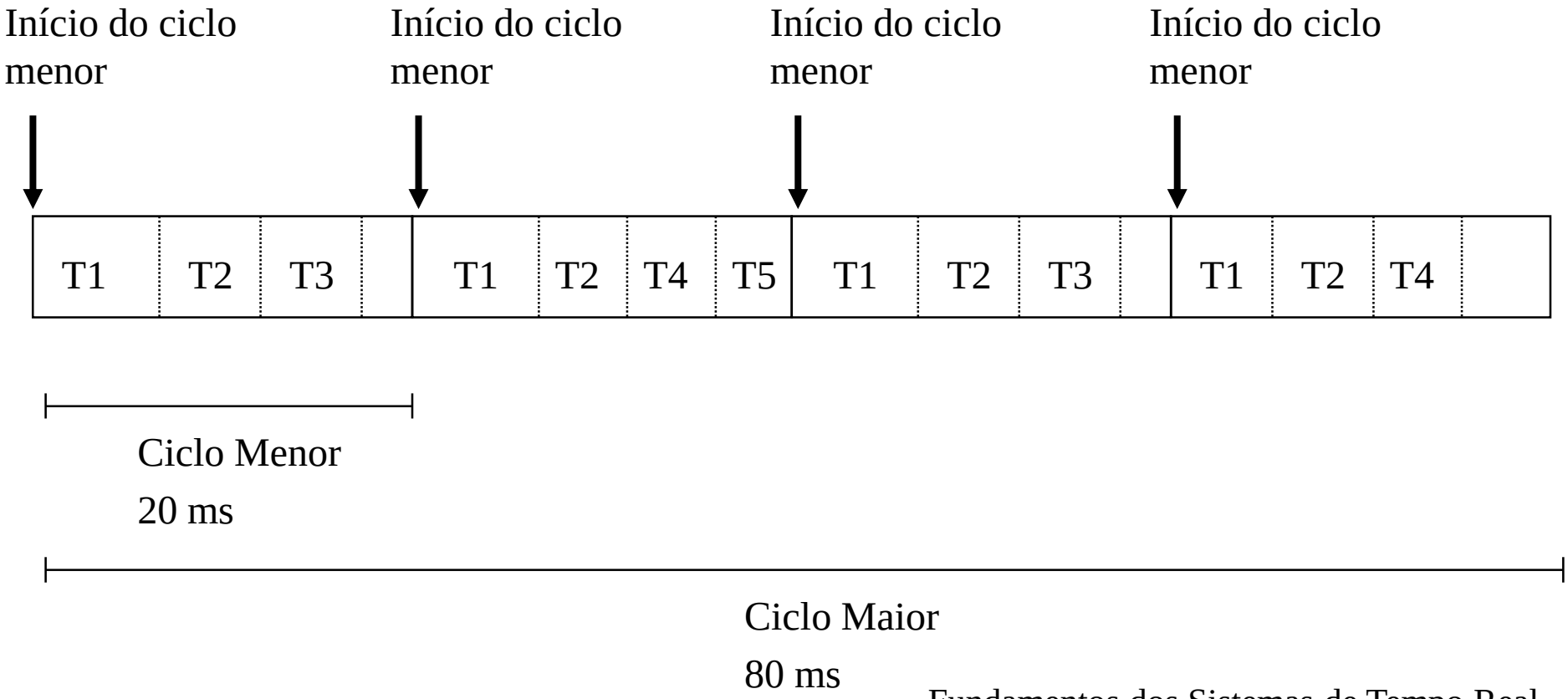
```
CicloMenor = 20 ms
While( true ) {
     Espera_próximo_ciclo_menor_iniciar( );
     funcao_tarefa_1( );
     funcao_tarefa_2( );
     funcao_tarefa_3( );
     Espera_próximo_ciclo_menor_iniciar( );
     funcao_tarefa_1( );
     funcao_tarefa_2( );
     funcao_tarefa_4( );
     funcao_tarefa_5( );
     Espera_próximo_ciclo_menor_iniciar( );
     funcao_tarefa_1( );
     funcao_tarefa_2( );
     funcao_tarefa_3( );
     Espera_próximo_ciclo_menor_iniciar( );
     funcao_tarefa_1( );
     funcao_tarefa_2( );
     funcao_tarefa_4( );
}
```

Executivo Cíclico 8/19

- O período de cada tarefa é respeitado
- A tarefa $\tau 1$ é executada em todos os ciclos menores
  - Executada uma vez a cada 20ms, seu período
- A tarefa $\tau 5$ é executada em apenas um dos ciclos menores
  - Apenas uma vez a cada ciclo maior, pois seu período é 80ms
- Tipicamente o valor do ciclo maior corresponde ao mínimo múltiplo comum dos períodos das tarefas
  - **Hiperperíodo** (***Hyperperiod***)
- Uma duração conveniente para o ciclo menor é o máximo divisor comum dos períodos das tarefas
  - Ciclo menor (20ms) foi escolhido desta forma
  - Entretanto, esta não é a única solução possível

Executivo Cíclico 9/19

- A soma dos tempos de execução no pior caso das tarefas contidas em cada ciclo menor não pode ultrapassar 20ms
- A computação em cada ciclo menor totaliza 19ms, 20ms, 19ms e 18ms



Executivo Cíclico 10/19

- Uma tarefa com tempo de computação grande pode dificultar esta divisão
- Neste caso, pode ser necessário adaptar as tarefas
- Uma tarefa com período Pk e tempo de computação Ck talvez possa ser dividida em duas partes
- Duas novas tarefas com períodos Pk e
  tempo de computação Ck/2 cada uma delas
- Este tipo de divisão facilita a organização dos ciclos menores

Executivo Cíclico 11/19

- A função *Espera_próximo_ciclo_menor_iniciar( )* representa a espera pelo final do ciclo menor corrente
- Pode usar um laço que fica constantemente lendo um relógio de tempo real no hardware
- Ou usar uma interrupção de temporizador em hardware (*timer*) previamente programado
- De qualquer forma, o tempo gasto nesta função deve ser somado ao tempo total de execução do ciclo menor

Executivo Cíclico 12/19

- É usual sobrar algum tempo de computação em cada ciclo menor
- No caso do exemplo anterior, temos sobras de 1ms no primeiro e terceiro ciclos menores, e uma sobra de 2ms no quarto ciclo menor
- Este tempo pode ser aproveitado para a execução de tarefas que não são de tempo real e aproveitam o tempo restante
- Tempos previstos no executivo cíclico são para o pior caso
- No caso típico as tarefas executam em menos tempo
  - Sobra ao final de cada ciclo menor será ampliada com este tempo adicional
- Todo tempo de processamento reservado para uma tarefa porém não utilizado por ela é chamado de **tempo ganho** (***gain time***)
- Pode ser usado por outras tarefas

Executivo Cíclico 13/19

- Caso o tempo alocado para o ciclo menor tenha terminado, mas ainda existe código de tarefa para executar, é dito que ocorreu um ***overrun*** do ciclo menor
- Uma abordagem simples é continuar a execução das tarefas
  - Na esperança de que as folgas existentes nos ciclos menores em sequência recomponham a corretude temporal do sistema
- No caso de sistemas críticos, a ocorrência do *overrun* representa uma grave **falta temporal** (***timing fault***)
- Algum tipo de tratamento de falta será necessário
  - Sinalizar a ocorrência da falta no painel do equipamento
  - Enviar mensagem para o fabricante
  - Reinicializar o equipamento
  - Mudar para um modo de segurança onde ações físicas perigosas do equipamento são suspensas

Executivo Cíclico 14/19

- Planejamento da execução das tarefas em ciclos menores e ciclo maior pode ser feita manualmente no caso de sistemas pequenos e simples
- Como a escala de execução é construída em tempo de projeto, algoritmos mais complexos também podem ser usados
- Exemplo: meta-heurísticas tais como Busca Tabu, Algoritmos Genéticos e Recozimento Simulado (*Simulated Annealing*)

- Podem existir objetivos secundários
  - Distribuir as folgas uniformemente entre os ciclos menores favorece o tratamento de *overrun*
  - Minimizar a variação do intervalo de tempo entre términos consecutivos de uma mesma tarefa

Executivo Cíclico 15/19

- A maior vantagem do executivo cíclico é a sua simplicidade
- As tarefas são implementadas como simples funções
- Facilitando a gerência do processador em sistemas pequenos e simples
  - Sistema operacional não é usado devido às limitações do hardware

- Fácil verificar se todas as tarefas cumprem os seus respectivos requisitos temporais
- Basta inspecionar a escala de execução gerada

Executivo Cíclico 16/19

- Executivo cíclico apresenta diversas limitações
- Períodos das tarefas devem ser múltiplos do tempo de ciclo menor
- É difícil incorporar tarefas com períodos longos
- Tarefas que atendem a situações de emergência e precisam executar imediatamente não são facilmente posicionadas na escala de execução

- O executivo cíclico funciona bem quando todas as tarefas possuem um deadline relativo igual ao período
- Tarefas esporádicas, ou tarefas periódicas com deadline relativo menor que o período, são difíceis de serem acomodadas
- Quando feito, geram ociosidade no sistema

Executivo Cíclico 17/19

- A programação das tarefas deve ser muito cuidadosa, pois nenhuma tarefa pode ficar bloqueada esperando por algum evento
- Considere o exemplo de um teclado.
- Tarefa responsável por ler o teclado é posicionada em um ou mais ciclos menores da escala de execução
- Quando ela executa, ela acessa diretamente o controlador do teclado e verifica se uma tecla foi acionada ou não
- Caso afirmativo, a tecla pode ser lida e processada
  - Talvez copiada para um array de caracteres na memória
- Não existe algo como uma função *scanf()* que a tarefa possa chamar e ficar bloqueada até que algo seja teclado

Executivo Cíclico 18/19

- Quando uma tarefa espera pela realização de uma operação de entrada ou saída a mesma é chamada de **entrada e saída síncrona**
- É desta forma que a maioria dos programas são construídos
- Forma mais simples de programar, mais legível, menos sujeita a erros

- No caso do executivo cíclico é necessário empregar operações de **entrada e saída assíncronas**
  - Tarefa não fica bloqueada esperando por elas
- Para saber se uma dada operação de entrada ou saída comandada antes foi concluída, é necessário ler alguma variável ou registrador
- Programação com entrada e saída assíncrona é mais difícil
  - Mais sujeita a erros
  - Manutenção mais custosa

Executivo Cíclico 19/19

- Executivo cíclico puro não inclui tratadores de interrupções
  - No máximo um tratador de interrupção para o temporizador no hardware que sinaliza o início de um ciclo menor
- Ações urgentes não podem ser realizadas por tratadores de interrupções
- Todos os sensores precisam ser amostrados por tarefas, como previsto na escala de execução

- Resposta aos eventos externos urgentes bem mais lenta do que seria com tratadores de interrupções

Sumário

- Executivo Cíclico
- Tratadores de Interrupções
- Laço Principal com Tratadores de Interrupções
- Microkernel Simples

Mecanismo de Interrupções 1/13

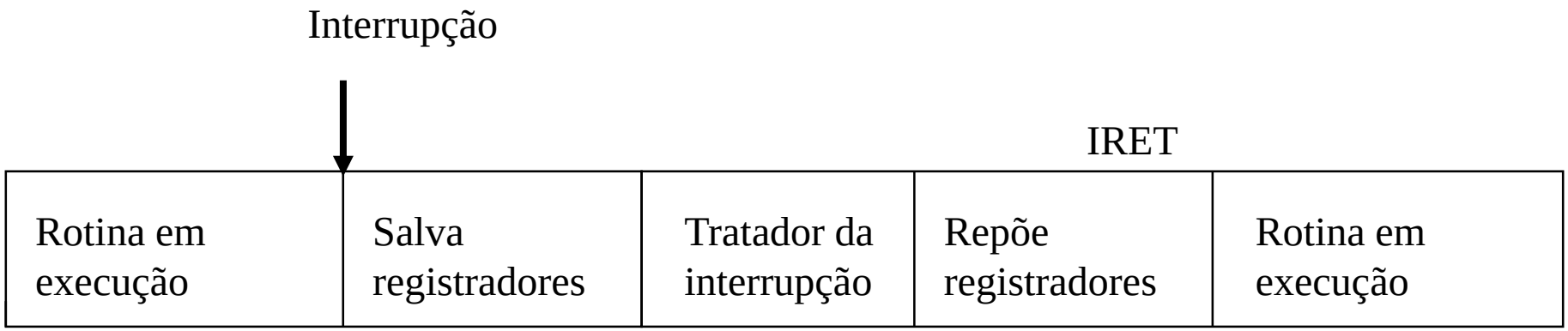
- O **mecanismo de interrupções** é um recurso presente em processadores de qualquer porte
- Ele permite, por exemplo, que um controlador de periférico (ethernet, timer, teclado, etc)
  acione um sinal elétrico no barramento de controle do computador e, chame a atenção do processador

Mecanismo de Interrupções 2/13

- Quando uma interrupção acontece
  - Processador termina a instrução de máquina em andamento
  - Desvia a execução para uma rotina específica
  - O **tratador de interrupção** (***interrupt handler***)
- O tratador de interrupção realiza as ações necessárias em função da ocorrência daquela interrupção.
  - Trata-se de uma rotina que não é chamada como uma função normal
  - É executada quando ocorre uma interrupção
- Quando a rotina que realiza o tratamento da interrupção termina
  - Processador volta a executar as instruções de máquina seguintes àquela quando a interrupção aconteceu

Mecanismo de Interrupções 3/13

- Tratador de interrupções



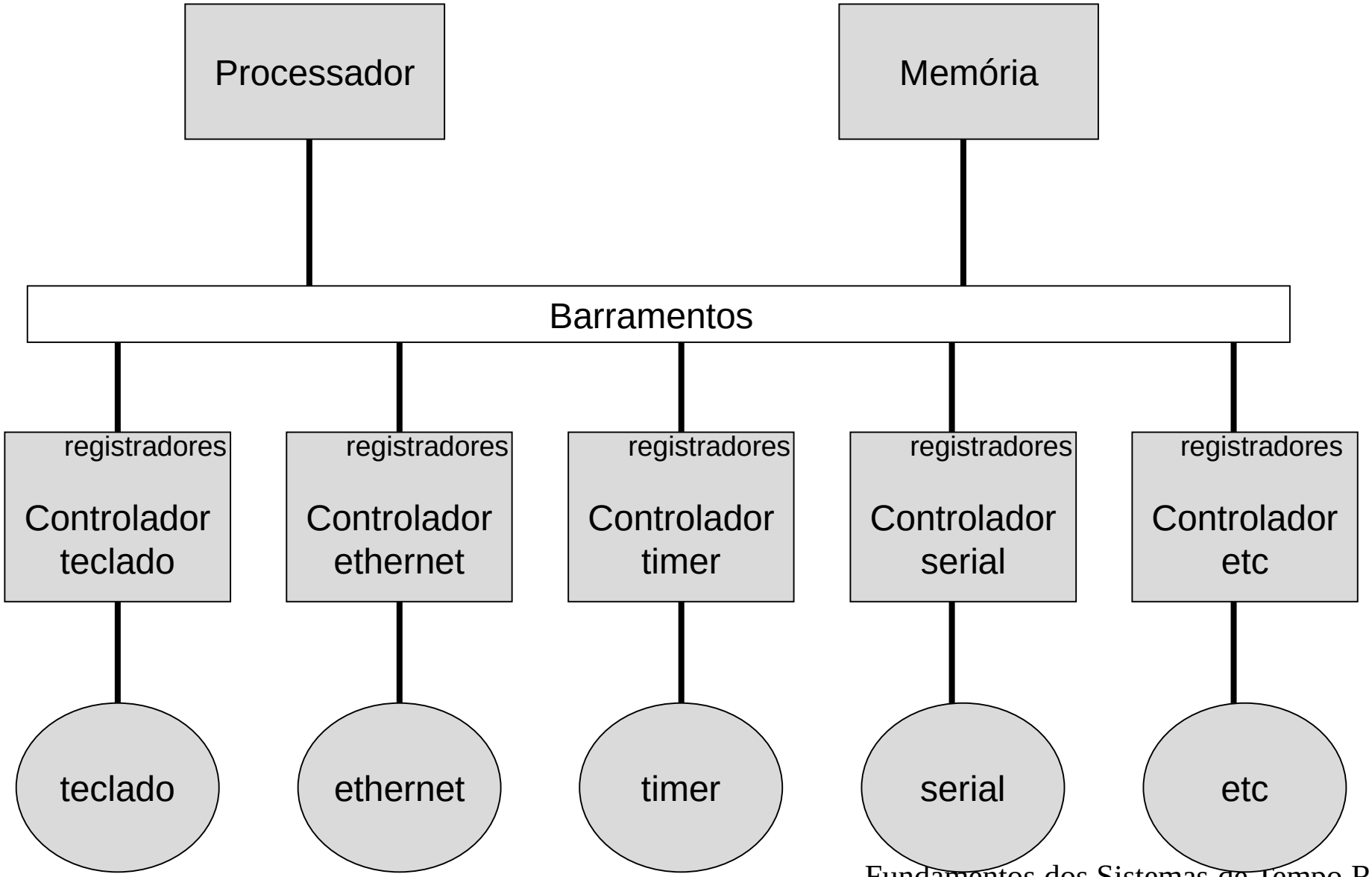
Mecanismo de Interrupções 4/13

- Durante a execução do tratador de interrupção é necessário preservar o conteúdo dos registradores associados com a rotina interrompida
- A maioria dos processadores salvam automaticamente os principais registradores quando ocorre uma interrupção
  - Tratador da interrupção salva os demais registradores
- Ao final, o tratador deve repor os valores que ele salvou
- Instrução de máquina “retorno de interrupção” (IRET)
  - Repõe o conteúdo original nos registradores salvos automaticamente
  - Faz o processador retomar a execução da rotina interrompida

Mecanismo de Interrupções 5/13

- Computadores incluem uma variada gama de periféricos os quais são comandados através de um componente eletrônico chamado de **controlador de periférico** (***peripheral device controller***)
- O processador é capaz de ler e escrever registradores do controlador de periférico e, desta forma, enviar comandos ou receber informações
- Cabe ao controlador de periférico executar os comandos através de acionamentos eletrônicos, elétricos e mecânicos apropriados
- Controladores de periféricos utilizam interrupções para alertar ao processador que algo aconteceu
  - Recepção ou transmissão de um pacote de dados
  - Acionar de uma tecla
  - Etc

Mecanismo de Interrupções 6/13



Mecanismo de Interrupções 7/13

- A forma mais simples de identificar a origem de uma interrupção é associar a cada controlador um tipo diferente de interrupção
- A maioria dos processadores identifica diferentes **tipos de interrupções** através de um número, tipicamente entre 0 e 255
- Cada tipo de interrupção é associado com uma causa diferente
- Por exemplo,
  - o controlador do teclado gera interrupções do tipo 20
  - o controlador ethernet gera interrupções tipo 30
  - Etc
- É possível associar um tratador específico para cada tipo de interrupção, chamado automaticamente

Mecanismo de Interrupções 8/13

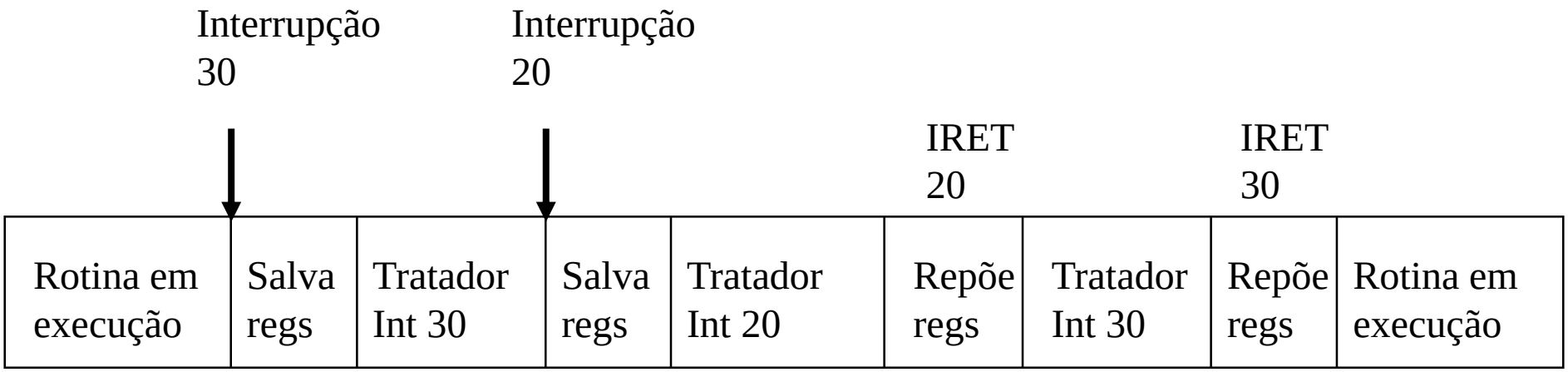
- O endereço de um tratador de interrupção é muitas vezes chamado de **vetor de interrupção** (***interrupt vector***)
  - Ele “aponta” para a rotina de atendimento da interrupção

- Cada tipo de interrupção possui associado um vetor de interrupção

- O processador utiliza uma tabela na memória com os vetores de todos os tipos de interrupção definidos
- Cada computador define os tipos de interrupções associados com cada periférico a sua maneira

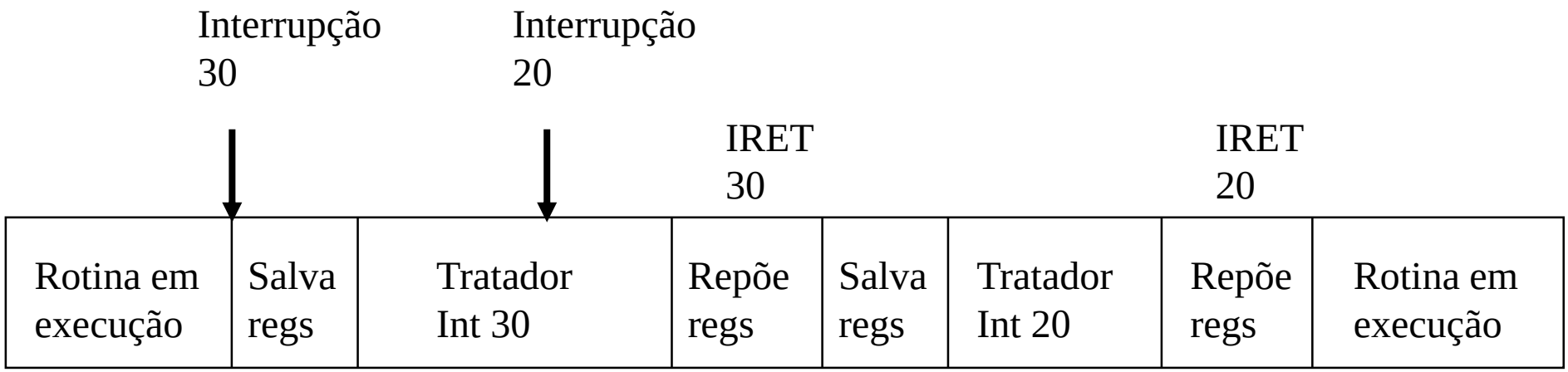
Mecanismo de Interrupções 9/13

- Em muitos processadores os diferentes tipos de interrupção possuem diferentes prioridades de atendimento
- Em alguns sistemas é permitido que tratadores de interrupções sejam eles próprios interrompidos pela ocorrência de tipos mais prioritários



Mecanismo de Interrupções 10/13

- Em muitos sistemas os tratadores de interrupção sempre executam completamente antes de outra interrupção ser reconhecida



Mecanismo de Interrupções 11/13

- Nem sempre interrupções podem ser atendidas.
- Suponha que um programa está alterando variáveis globais as quais também são acessadas pelo tratador de interrupções
  - Variáveis globais poderiam ficar inconsistentes
- Interrupções podem ser desabilitadas temporariamente através de instruções de máquina específicas, tais como **Desabilita Interrupção** (**DI** - ***Disable Interrupt***) e **Habilita Interrupção** (**EI** - ***Enable Interrupt***)
- Caso ocorra uma interrupção e as mesmas estão desabilitadas, a mesma não é perdida, ela fica pendente
  - Interrupções que podem ser desabilitadas são chamadas de **Interrupções Mascaráveis**
- Um tipo especial de interrupção que nunca pode ser desabilitada é chamada de **Interrupção Não Mascarável** (**NMI – *Non-Maskable Interrupt***)
  - As NMI são associadas com eventos críticos no sistema, por exemplo, com uma iminente falta de energia

Mecanismo de Interrupções 12/13

- As **Interrupções de Software** (***Software Interrupts***) são geradas pela execução de uma instrução de máquina específica, usualmente representada por INT
- Ela tem como parâmetro o número do tipo de interrupção a ser gerada

- O efeito é semelhante a uma interrupção de hardware do mesmo tipo, porém seu momento de ocorrência é controlado pelo programa sendo executado

- Ao contrário de uma interrupção de periférico, a qual em geral não é possível prever no programa o seu momento de ocorrência

Mecanismo de Interrupções 13/13

- Existe uma terceira classe de interrupções, geradas pelo próprio processador:
  Interrupções de Proteção ou **Exceções** (***Exceptions***)
- São interrupções geradas pelo processador quando o mesmo detecta algum tipo de erro ou de violação dos mecanismos de proteção
- Por exemplo, uma divisão por zero ou o acesso a uma posição de memória que na verdade não existe
- A sequência de atendimento a essa classe de interrupções é igual ao descrito anteriormente
- As exceções são tipadas da mesma forma que as interrupções de periféricos e de software

Sumário

- Executivo Cíclico
- Tratadores de Interrupções
- Laço Principal com Tratadores de Interrupções
- Microkernel

Laço Principal com Tratadores de Interrupções 1/8

- Executivo cíclico é uma forma simples e efetiva de resolver o problema de escalonamento quando todas as tarefas são periódicas e possuem um deadline relativo igual ao período

- Tarefas esporádicas, que podem chegar a qualquer momento, são mais difíceis de acomodar

- Tarefas com deadline relativo menor que o período também dificultam a construção da escala de execução

Laço Principal com Tratadores de Interrupções 2/8

- Suponha um sistema onde existe um sensor de pressão que deve ser amostrado
  - Caso a pressão no duto ultrapasse um certo limiar, a tarefa que abre a válvula de saída deve ser executada
  - A válvula precise ser aberta (a tarefa executada) em no máximo 200ms (seu deadline relativo ao instante de chegada)
  - O tempo de execução no pior caso da tarefa em questão é 40ms

- Esta tarefa pode ser acomodada em um executivo cíclico, desde que a cada intervalo de 200ms sejam reservados 40ms para ela
  - Caso o ciclo menor seja de 200ms, em todo ciclo menor devem ser reservados 40ms para esta tarefa, não importa se ela executa ou não
  - Embora a execução da tarefa seja um evento raro, 20% do tempo total do processador (40/200) precisa ser reservado para ela

Laço Principal com Tratadores de Interrupções 3/8

- Para sistemas onde existem tarefas esporádicas, com deadlines relativos menores que o intervalo mínimo entre chegadas, um design mais eficiente é o **Laço Principal com Tratadores de Interrupções**
- Todas as atividades que não são de tempo real, ou são periódicas com um deadline relativo igual ao período, são colocadas no laço principal que executa periodicamente
- Tarefas no laço não possuem deadline ou possuem um deadline relativo maior ou igual ao período do laço
- Também assume-se implicitamente que o período de tais tarefas é igual ao período do laço
  - Pois elas são executadas uma vez dentro do laço

Laço Principal com Tratadores de Interrupções 4/8

- É possível acomodar facilmente tarefas com períodos múltiplos inteiros do período do laço
  - Basta colocar um contador que é incrementado a cada execução do laço e executar a tarefa em questão a cada X execuções do laço principal

- Soluções mais complexas para o código do laço principal podem ser pensadas

- Quando a organização do código do laço principal torna-se complexa, é um claro indicativo de que o design do software deve ser alterado

Laço Principal com Tratadores de Interrupções 5/8

- Tarefas de tempo real mais exigentes são implementadas como tratadores de interrupção
- Todos os tratadores de interrupções tem prioridade sobre as tarefas do laço principal
  - Exceto quando interrupções são desabilitadas
  - Isto pode ocorrer quando tarefas do laço principal precisam acessar variáveis globais compartilhadas com os tratadores de interrupções

Laço Principal com Tratadores de Interrupções 6/8

- No caso da tarefa responsável por abrir a válvula de saída no duto, é possível usar um dispositivo de hardware que gera uma interrupção sempre que a pressão ultrapassa o limiar desejado
  - Acionando assim a tarefa em questão

- Também é possível usar interrupções de um temporizador em hardware (*timer*) para ativar a tarefa periodicamente
  - Quando a mesma então amostra a pressão no duto e decide se deve ou não abrir a válvula de saída
- Várias tarefas periódicas podem ser associadas à interrupção do *timer*, desde que seus períodos sejam múltiplos inteiros do período das interrupções do *timer*
  - Ou computador ofereça vários temporizadores no hardware

Laço Principal com Tratadores de Interrupções 7/8

- Caso o laço principal seja periódico,
  o período do laço principal deve comportar
  o tempo de execução no pior caso de todas as tarefas no laço
  e também o tempo de execução no pior caso
  dos tratadores de interrupções
  - Todas as interrupções podem ocorrer durante um ciclo do laço
- Por exemplo, se o período do laço principal for 100ms e
  porta de comunicação serial pode gerar uma interrupção a cada 2ms,
  ao longo de um ciclo do laço principal podem ocorrer 50 interrupções
  da porta serial
- Serão necessárias 50 execuções da tarefa associada com o tratador de
  interrupções da porta serial

Laço Principal com Tratadores de Interrupções 8/8

- A desvantagem desta abordagem construtiva em relação ao executivo cíclico é que agora não existe mais uma escala de execução fixa para inspeção
- Não é possível prever quando ocorrerão interrupções de dispositivos como teclado, ethernet, porta serial, etc
- Natureza esporádica destas interrupções faz com que a escala de execução realmente observada varie completamente de ciclo para ciclo do laço principal

- Qualquer análise do tempo de resposta das tarefas deve ser feita com cuidado
- Mesmo os tratadores de interrupção podem ser atrapalhados
- Interrupções são por vezes desabilitadas
- Diferentes interrupções podem ocorrer simultaneamente
  - Gerando uma situação de prioridade entre elas

Sumário

- Executivo Cíclico
- Tratadores de Interrupções
- Laço Principal com Tratadores de Interrupções
- Microkernel Simples

Microkernel Simples 1/25

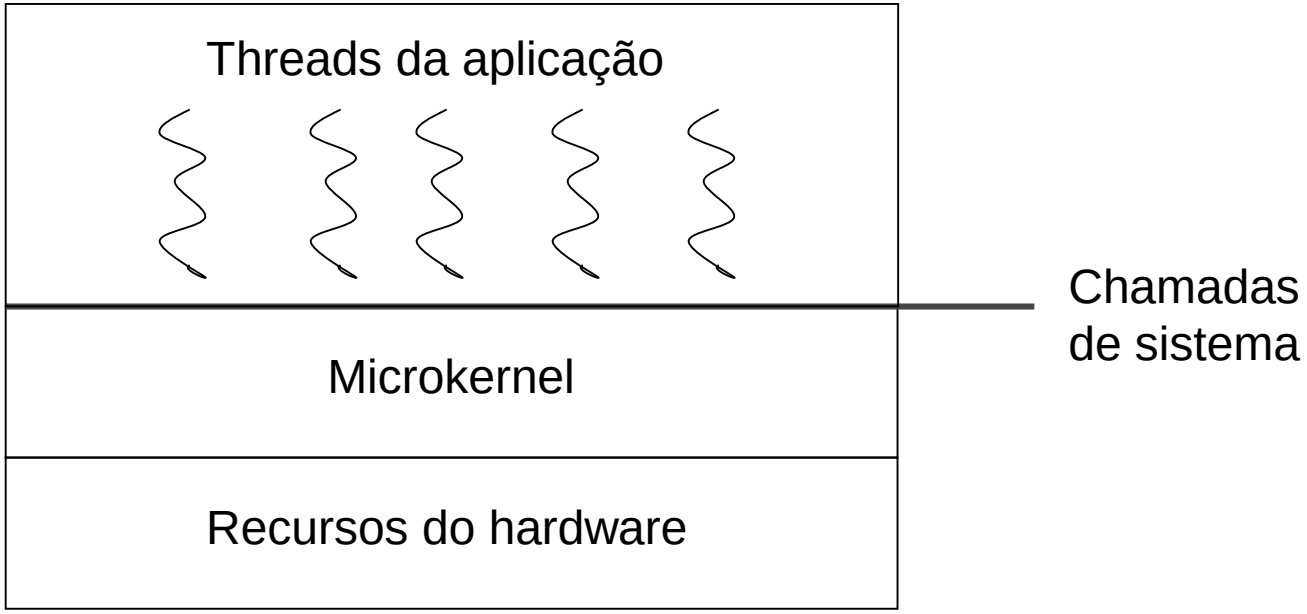
- Na medida que a complexidade da aplicação aumenta
  - Com um número maior de tarefas para executar
  - Mais periféricos para gerenciar
- passa a ser vantajoso um sistema operacional multitarefa

- Caso mais simples:
  Microkernel que só gerencia o processador

- Exemplo FreeRTOS (www.freertos.org)

Microkernel Simples 2/25

- Microkernel oferece apenas serviços de gerência do processador
  - Não inclui sistema de arquivos ou gerência de memória sofisticada
- **Multiprogramação** (***multiprogramming***) cria a abstração ***thread***
- Microkernel cria a ilusão que as threads executam simultaneamente



Microkernel Simples 3/25

- Threads solicitam serviços ao microkernel através de **chamadas de sistema** (***system calls***)
- Chamadas de sistema são implementadas como interrupções de software cujo tratador faz parte do microkernel
- Thread desejando ler um caractere do teclado deve colocar em um registrador específico o código numérico da operação e gerar a interrupção de software que aciona o microkernel em questão

- Quando a aplicação é programada em uma linguagem de mais alto nível, como C ou C++,
  as chamadas de sistema ficam escondidas dentro das bibliotecas da linguagem
  - São as rotinas da biblioteca que fazem as chamadas de sistema

Microkernel Simples 4/25

- Uma thread da aplicação requer um serviço do microkernel e faz uma chamada de sistema na forma de uma interrupção de software
- Neste momento, o microkernel entra em execução, atendendo esta chamada de sistema
- Muitas vezes a chamada não pode ser atendida imediatamente
  - Uma leitura de teclado ou uma chamada do tipo “sleep”
- Quando uma thread não pode continuar a executar pois está esperando por algum evento:
  o microkernel coloca outra thread para executar

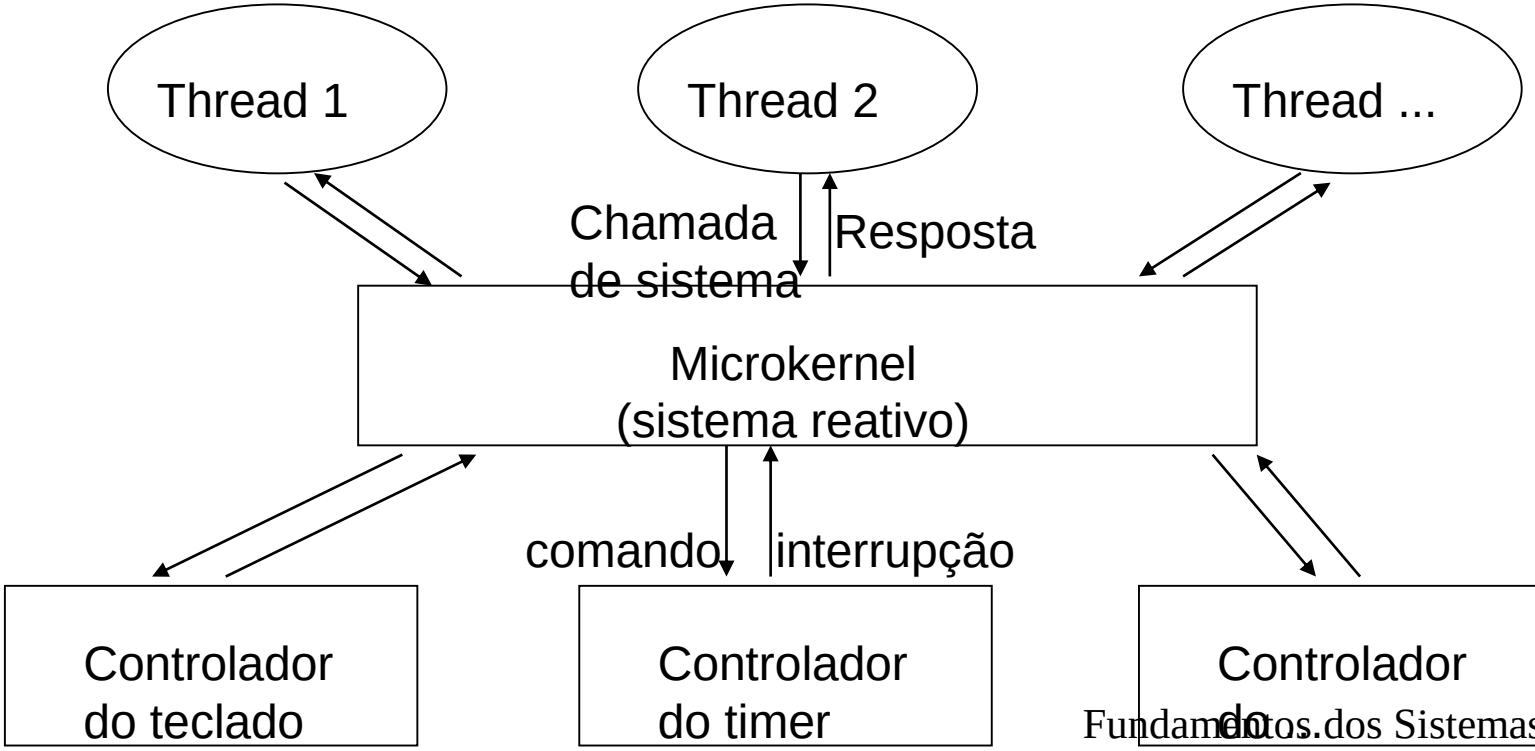
- Desta forma, o microkernel sempre volta a não fazer nada

Microkernel Simples 5/25

- Em algum momento no futuro o evento esperado acontecerá: o tempo de *sleep* passou, um caractere foi teclado, etc
- Neste momento, o controlador do periférico em questão gera uma interrupção e aciona o tratador, que faz parte do microkernel
- O microkernel volta a executar
  - Faz o tratamento dos dados que for necessário
- Agora a thread que fez a chamada de sistema pode continuar sua execução
- Na existência de várias threads aptas a executar e apenas um processador, o microkernel precisará escolher quem executa antes
- Uma delas é colocada para executar
  - O microkernel volta a não fazer nada

Microkernel Simples 6/25

- Microkernel é um **sistema reativo** (***reactive system***)
  - Não faz nada até que ocorra um evento
  - Chamada de sistema via interrupção de software
  - Controlador de algum periférico requer atenção via interrupção de hardware
  - Microkernel reage ao evento, volta a não fazer nada



Microkernel Simples 7/25

- Um microkernel simples contém apenas os *device-drivers* para os periféricos mais fundamentais do sistema

- Tipicamente um temporizador em hardware (*timer*) está presente
  - Usado para serviços do tipo “sleep” e por vezes pelo algoritmo de escalonamento do processador

- Muitos incluem um mecanismo básico de comunicação de dados com o mundo exterior
  - Por exemplo uma porta serial ou um controlador ethernet

Microkernel Simples 8/25

- Outra abordagem é o fornecedor do microkernel oferecer uma gama de *device-drivers* para vários periféricos
  - Porém apenas incluir no código do microkernel, no momento de sua compilação, aqueles *device-drivers* que o programador da aplicação realmente necessita
- Também é muito comum utilizar um microkernel em plataforma de hardware parcialmente projetada pelo desenvolvedor da aplicação
- O caso de um **sistema computacional embutido** ou **embarcado** (***embedded computer system***) em máquinas ou equipamentos
- É comum a existência de **periféricos especializados** (***custom peripherals***) para os quais não existem *device-drivers* previamente programados
  - Cabe ao desenvolvedor da aplicação também programar o código que fará este periférico especializado funcionar

Microkernel Simples 9/25

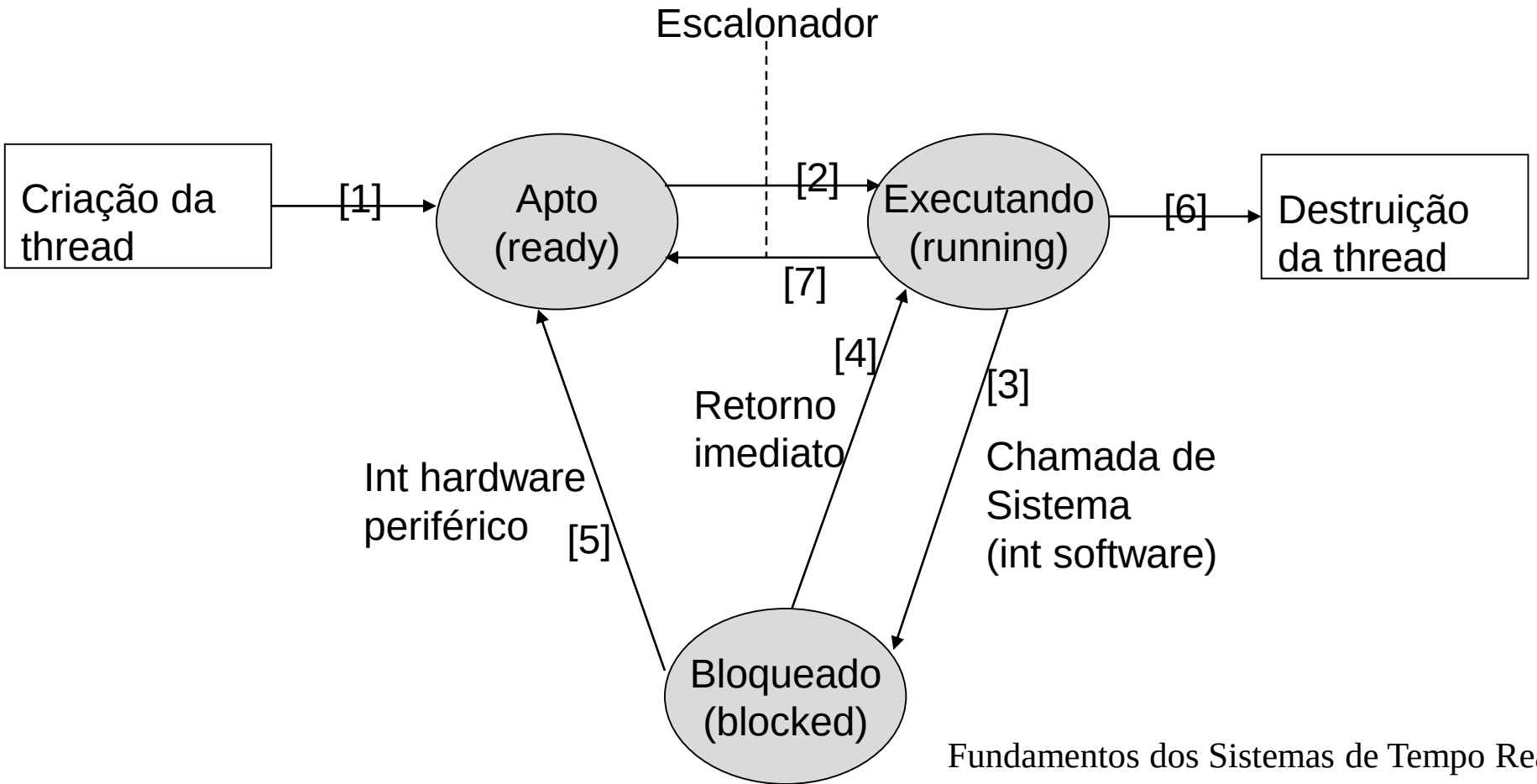
- Uma solução de projeto popular é implementar o *device-driver* do periférico especializado como uma thread da aplicação
- Outras threads da aplicação podem solicitar serviços do periférico especializado através de mecanismos de comunicação entre threads
- Thread que comanda o periférico especializado acessa diretamente os registradores do seu controlador
- É necessário tratar as interrupções de hardware geradas pelo controlador do periférico especializado
- Alguns microkernels permitem que a aplicação instale seus próprios tratadores
- Outros oferecem chamada de sistema do tipo “espera pela ocorrência de uma interrupção do tipo X”
  - O microkernel apenas recebe as interrupções de hardware e libera threads que porventura estejam esperando a ocorrência de tais interrupções

Microkernel Simples 10/25

- **Estados de uma Thread**
- Threads podem ser criadas e destruídas dinamicamente
  - através de chamadas de sistema
- A primeira thread precisa ser criada na inicialização do microkernel
  - Neste momento ainda não existe ninguém para fazer chamadas de sistema

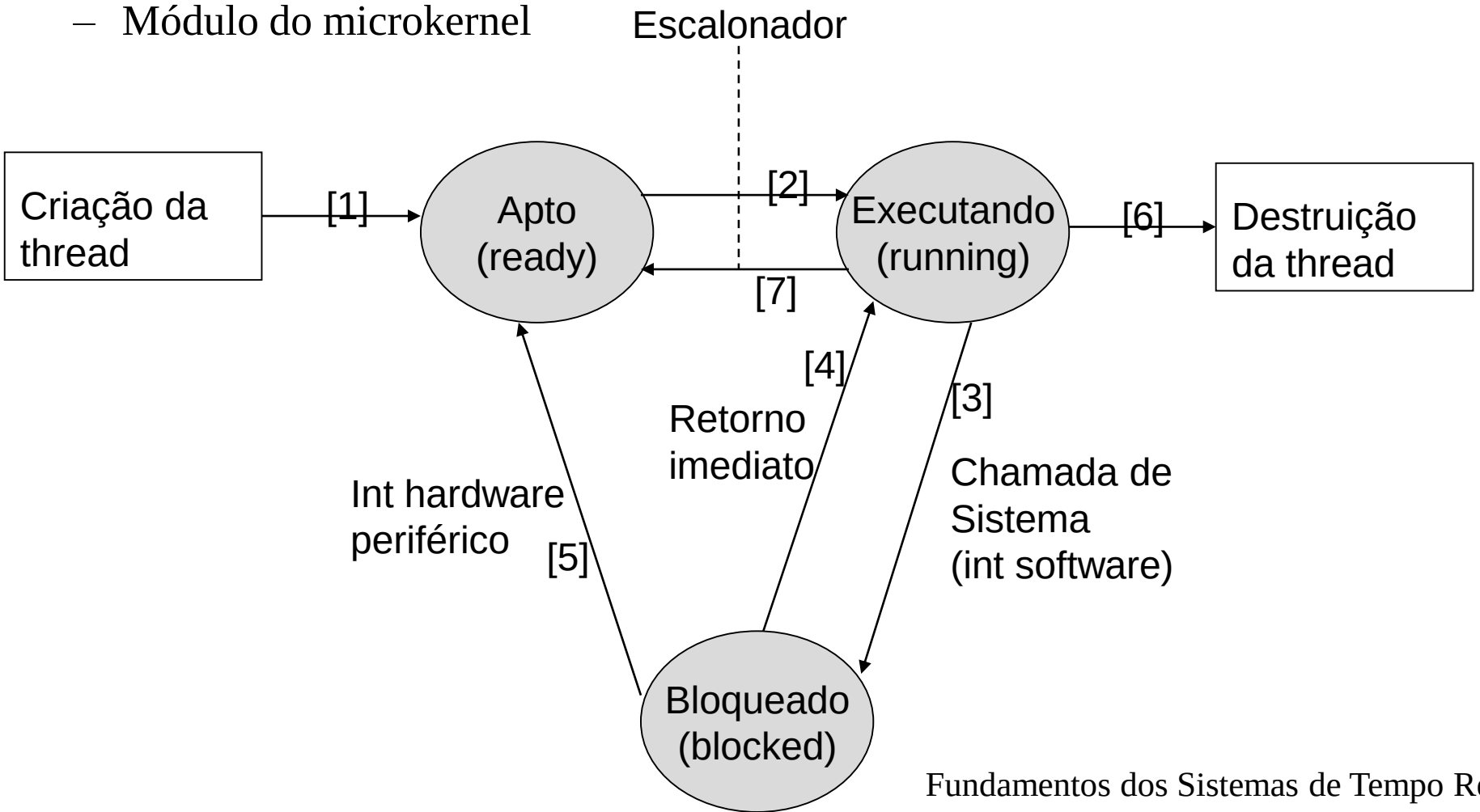
Microkernel Simples 11/25

- Após ser criada, uma thread precisa do processador
- Podem existir dezenas de threads competindo
  - Fila é chamada de **Fila de Aptos** ou **Fila de Prontos** (***Ready Queue***)



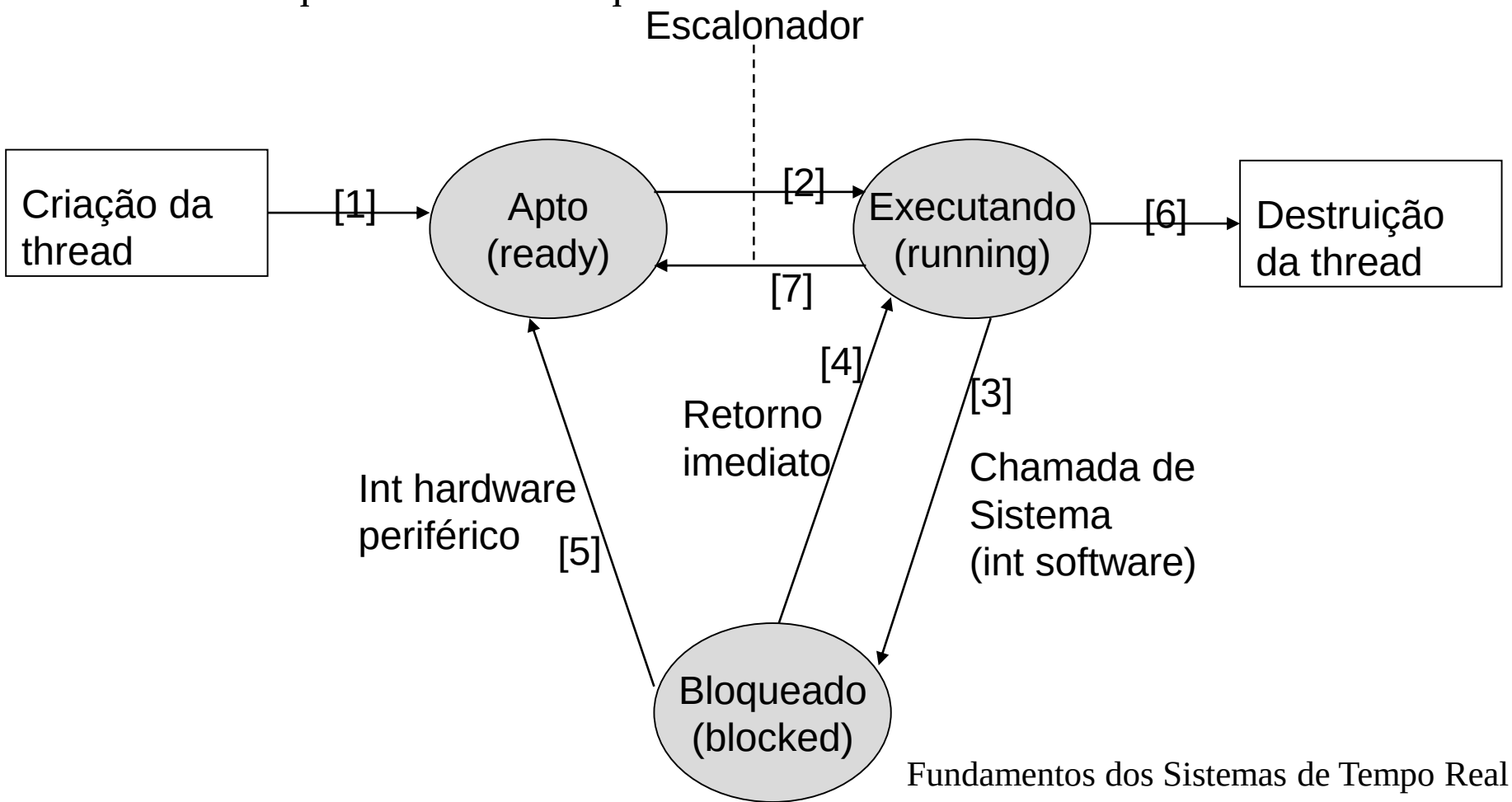
Microkernel Simples 12/25

- Em algum momento no futuro a thread é escolhida para ser executada
- Decisão cabe ao **escalonador** (***scheduler***)
  - Módulo do microkernel

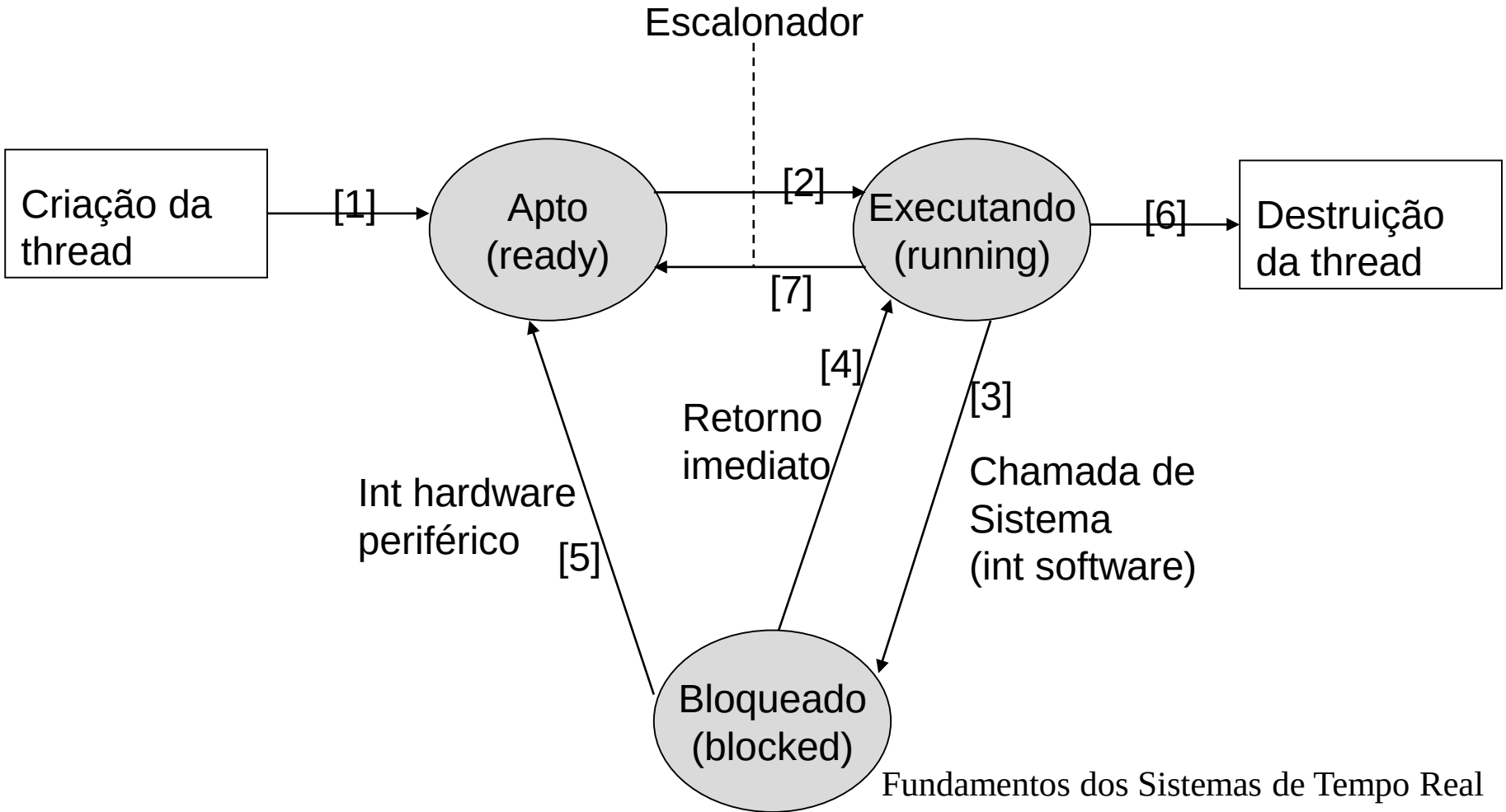


Microkernel Simples 13/25

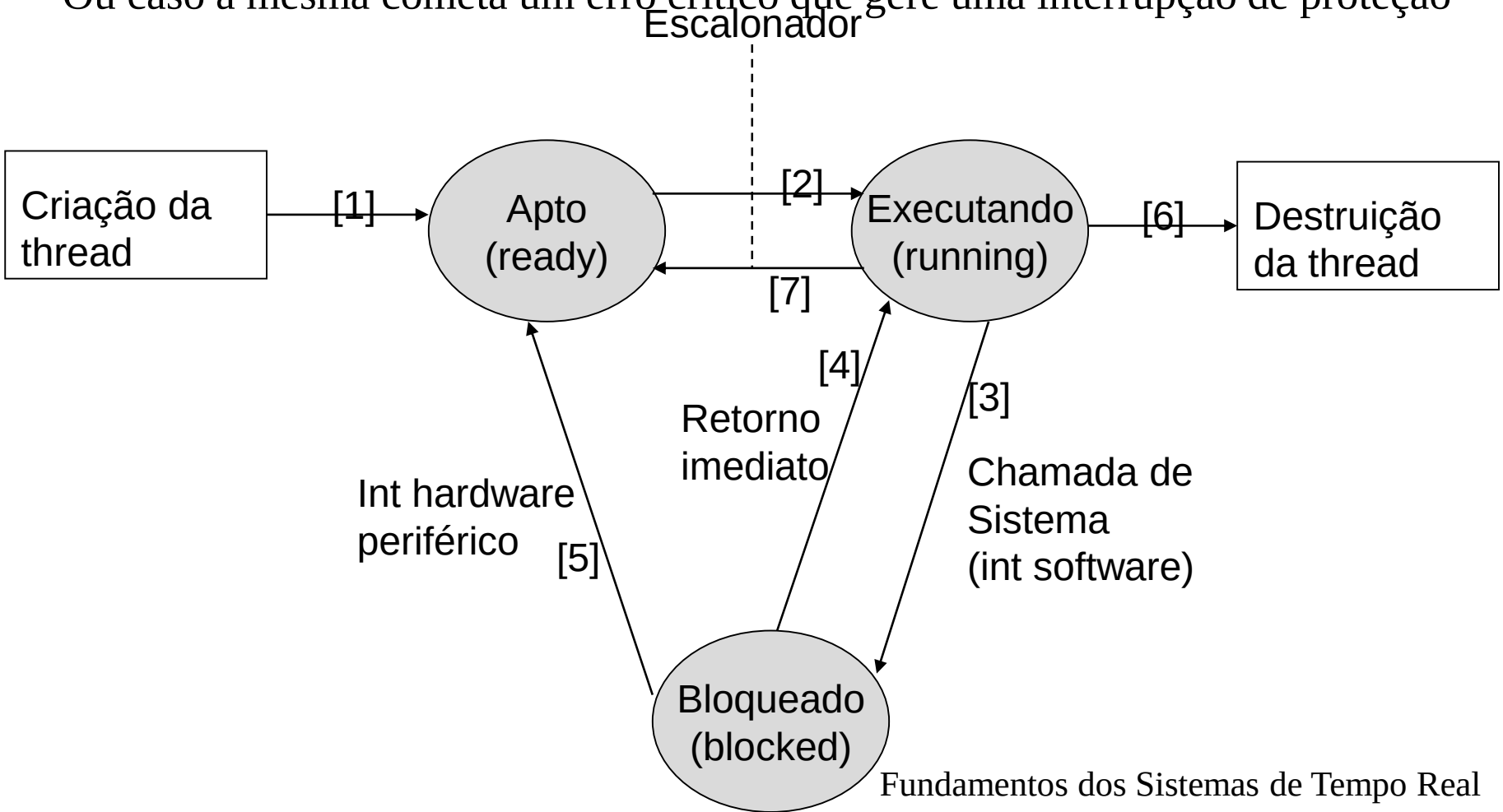
- Thread executa até fazer chamada de sistema
  - Recepção da porta serial, sleep, etc
  - Thread em questão ficará bloqueada ou retorna imediatamente



- Após algum tempo, a ação solicitada pela thread é concluída
  - Thread que estava bloqueada é liberada (*released*)

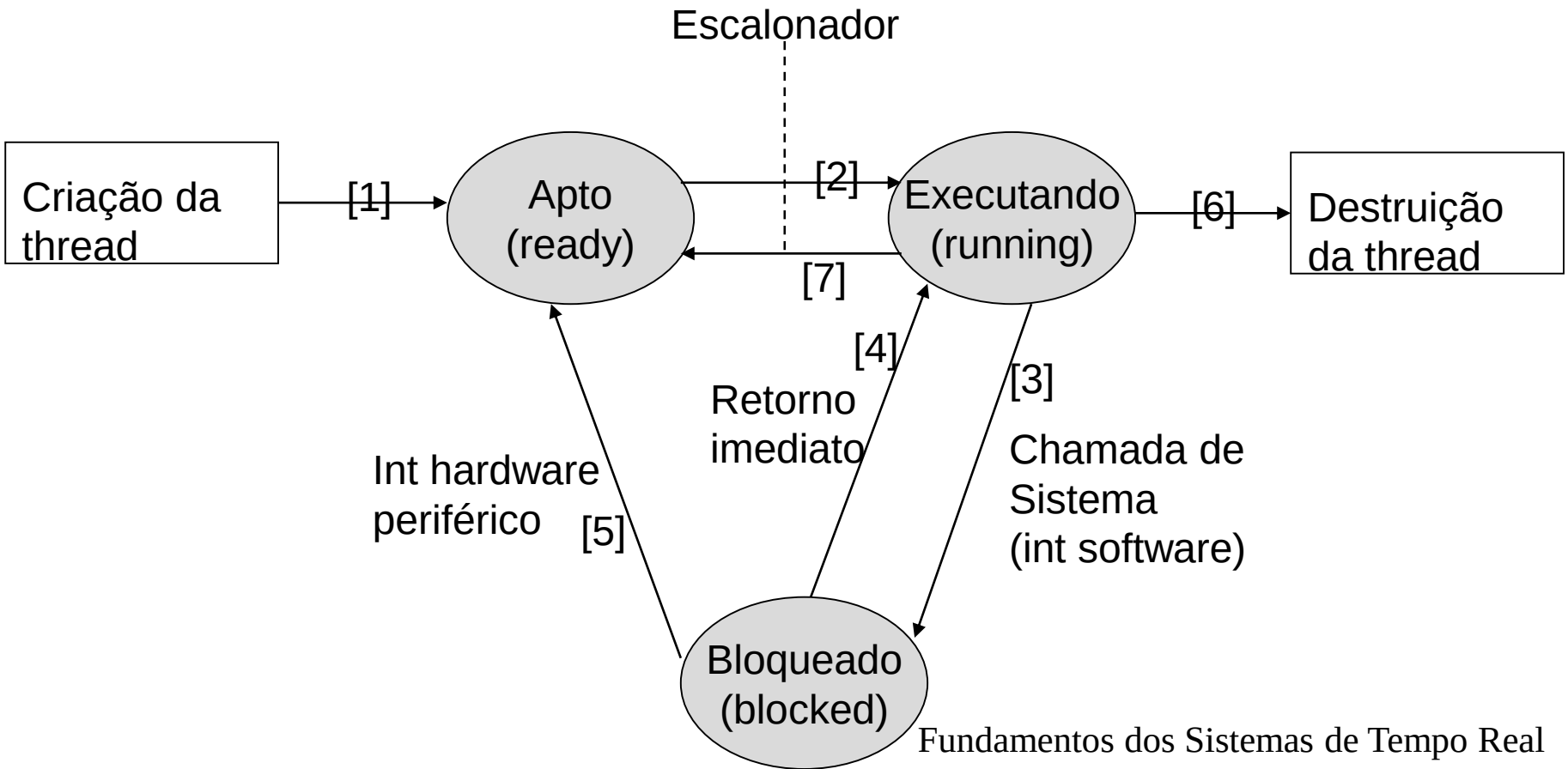


- Thread termina através de uma chamada de sistema
  - Pode ser destruída por outra thread através de chamada de sistema
  - Ou caso a mesma cometa um erro crítico que gere uma interrupção de proteção



Microkernel Simples 16/25

- Quando uma thread torna-se apta, cabe ao escalonador decidir
  - thread em execução prossegue
  - ou deve ceder lugar para a thread recém tornada apta



- **Chaveamento de Contexto entre Threads**
- A operação de suspender a thread em execução (passar a thread do estado de executando para o estado de apto) e
  colocar uma outra thread para executar (passar a thread do estado de apto para executando)
  é chamada de **troca de contexto** (***context switch***).
  - Na multiprogramação isto acontece o tempo todo

- Quando uma thread tem sua execução suspensa é necessário salvar o seu contexto de execução
- Para que uma thread possa passar a executar, ocupando o processador, é necessário recarregar o seu contexto
  - Contexto de execução é formado principalmente pelo conteúdo dos registradores

Microkernel Simples 18/25

- O microkernel precisa manter as informações relativas ao contexto de execução de cada thread
  - Para efetuar os chaveamentos
- Microkernel mantém uma estrutura de dados chamada de **Bloco Descritor de Thread** (**TCB - *Thread Control Block***)
  - Conteúdos dos registradores quando a thread não estiver executando
  - Número **identificador da thread** (**TID** – ***Thread Identification***)
  - Prioridade da thread usada pelo escalonador para definir a ordem de execução
  - Etc
- Tipicamente as filas de threads esperando por recursos são representadas dentro do microkernel como listas encadeadas dos respectivos TCBs

Microkernel Simples 19/25

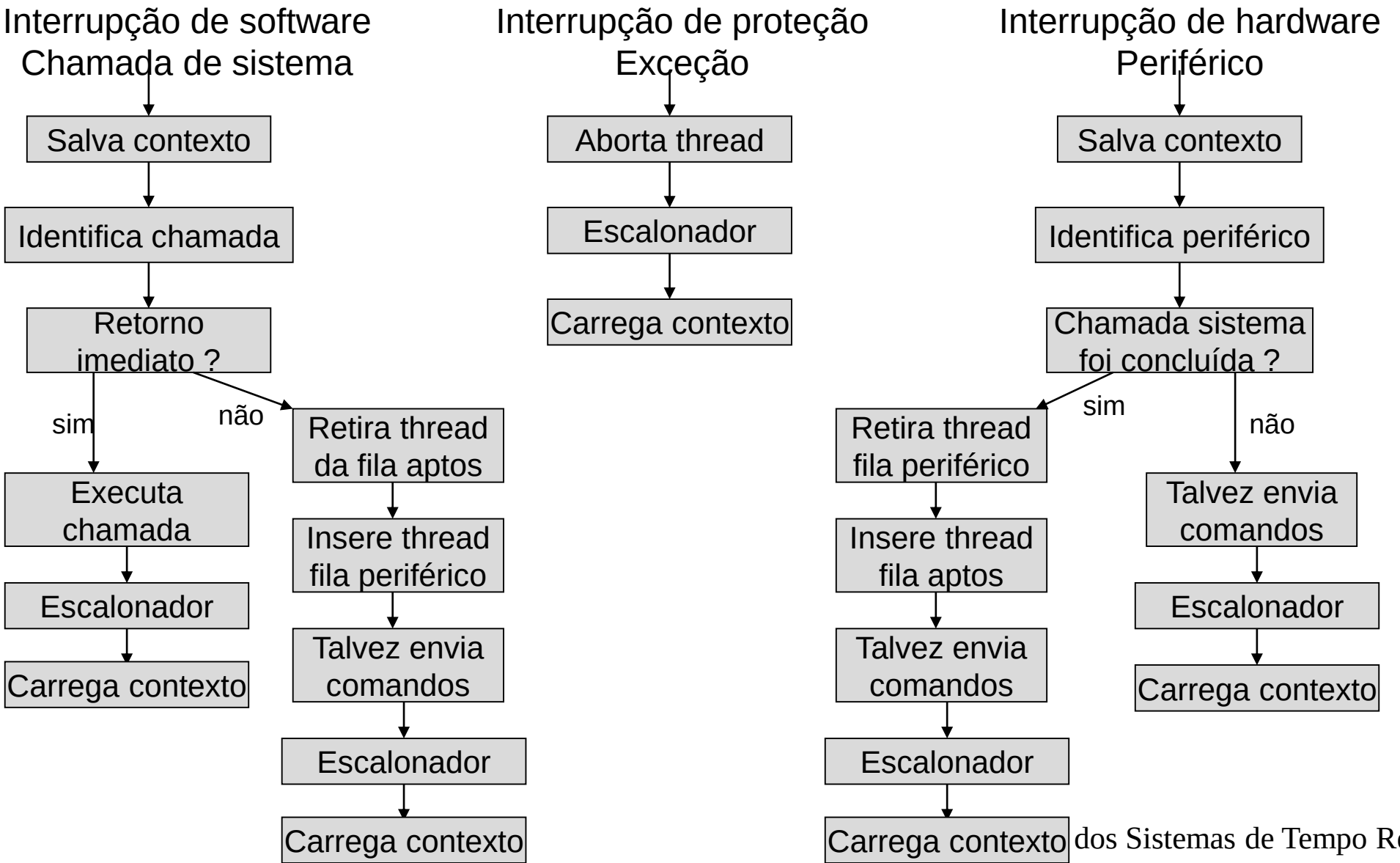
- **Design do Microkernel**
- O microkernel é um programa de computador
  - Normalmente escrito na linguagem C
  - Com algumas pequenas partes em linguagem de montagem (*assembly language*)

- Quando o computador é energizado ou reinicializado o microkernel começa a executar e pode inicializar as suas estruturas de dados
- A partir deste momento, não é o microkernel que executa, mas as threads da aplicação
- O microkernel não faz nada até que ocorra um evento
- Ele reage ao evento, e depois volta a não fazer nada
  - Sistema reativo

Microkernel Simples 20/25

- Os eventos que acionam o microkernel são as interrupções
- Interrupção de software, sinalizando que a thread em execução requer um serviço através de chamada de sistema
- Interrupção de hardware, associada com um periférico que requer atenção
- Interrupção de proteção, pois a thread em execução cometeu alguma falta e gerou uma exceção

Microkernel Simples 21/25



Microkernel Simples 22/25

- O microkernel sempre inicia salvando o contexto da thread interrompida, a não ser quando a mesma será destruída
- Microkernel sempre chama o escalonador para escolher a próxima thread a executar e então carrega o seu contexto de execução
- Para evitar a situação inconveniente de não existir nenhuma thread para executar, muitos definem uma **thread ociosa** (***idle thread***)

- O fluxograma supõe que interrupções são desabilitadas automaticamente na ocorrência de uma interrupção

Microkernel Simples 23/25

- **Comparação entre as Formas de Implementar Tarefas**
- Existem várias formas possíveis para implementar uma tarefa de tempo real
- No caso de um executivo cíclico, tarefas de tempo real aparecem como funções chamadas no código do executivo
- No caso de laço principal com tratadores de interrupção, as tarefas de tempo real podem aparecer como funções no laço principal ou como tratadores de interrupção
- Quando um microkernel simples é usado, tarefas são em geral threads da aplicação executando fora do microkernel
  - Excepcionalmente, tarefas de tempo real podem ser associadas com tratadores de interrupção que ficam dentro do microkernel

Microkernel Simples 24/25

- Complexidade crescente é acompanhada por um **sobrecusto** (***overhead***) de implementação crescente
- O executivo cíclico é preferido em computadores pequenos, onde recursos como processamento e memória são escassos
- Já um kernel completo consome, ele próprio, um bom conjunto de recursos computacionais
- Laço com tratadores de interrupção e microkernel simples são formas intermediárias

Microkernel Simples 25/25

- Maior complexidade da solução também implica em maior flexibilidade e comodidade para o programador da aplicação
- O único serviço que o executivo cíclico provê é o escalonamento do processador
  - Tudo mais é programado pelo desenvolvedor da aplicação
- Com um laço principal e tratadores de interrupções o desenvolvedor tem mais flexibilidade para compor o sistema
- Ao usar um microkernel simples o desenvolvedor da aplicação tem agora acesso a uma gama de serviços
  - os quais podem ser acessados através de chamadas de sistema
- Serviços oferecidos por um microkernel estão basicamente associados com a gerência do processador
- A melhor solução depende da aplicação em questão

Sumário

- Executivo Cíclico
- Tratadores de Interrupções
- Laço Principal com Tratadores de Interrupções
- Microkernel Simples