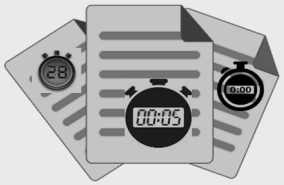
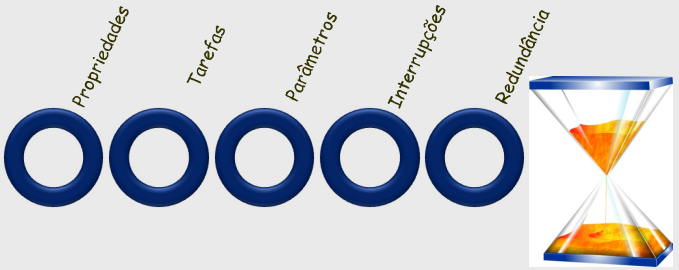
PCS5761 Especificação de Sistemas de Tempo Real

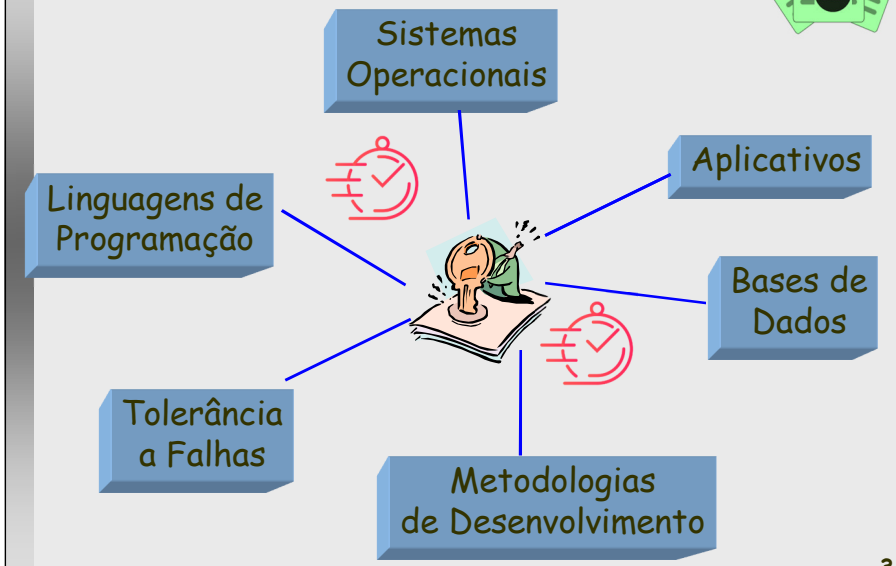
Prof. Dr. Jorge Rady de Almeida Jr.

Software para Tempo Real





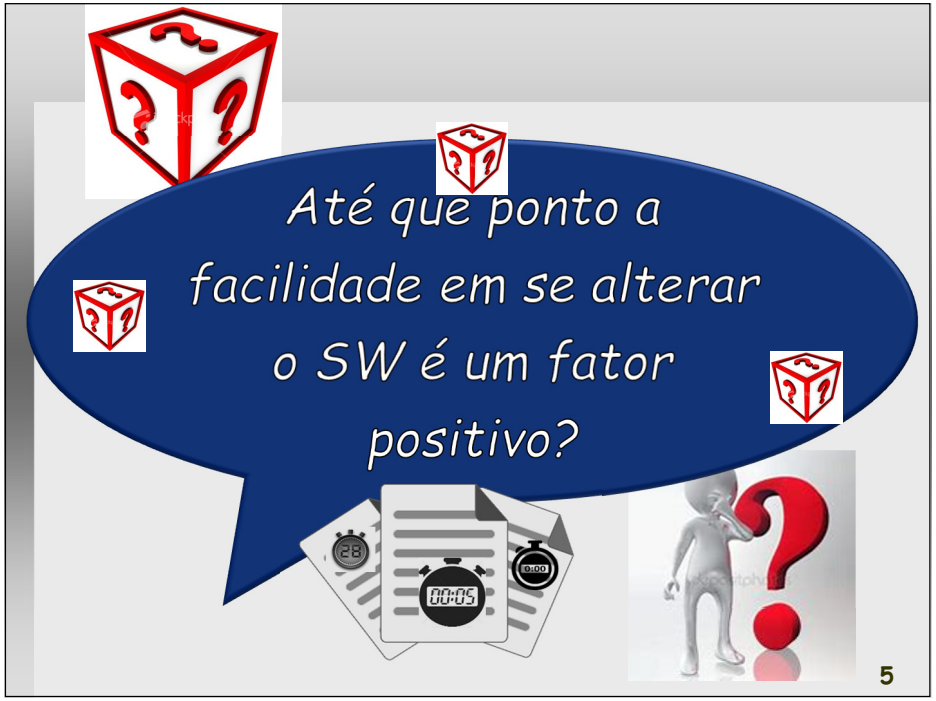
Software de Tempo Real



Software de Tempo Real

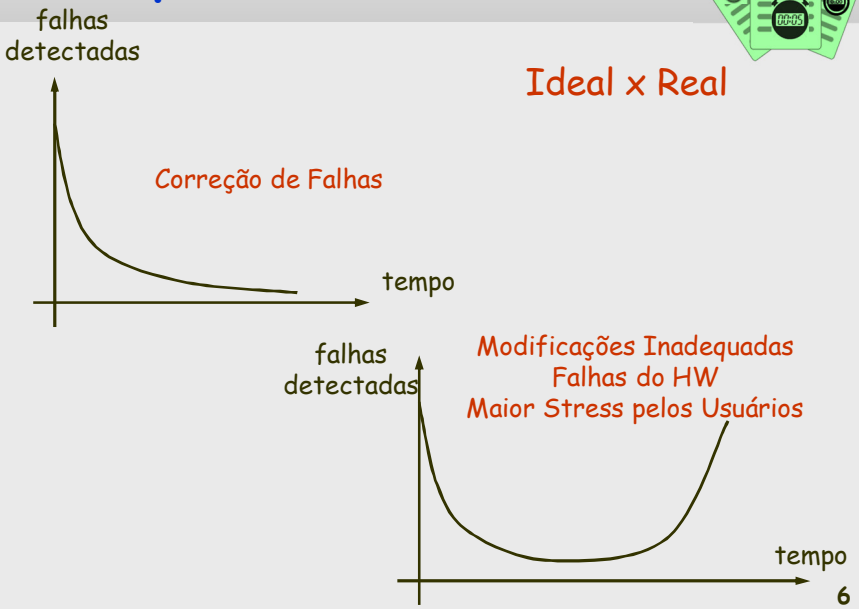
- Tempo de desenvolvimento maior do que HW

Pode ser projetado a partir de outros sistemas já existentes

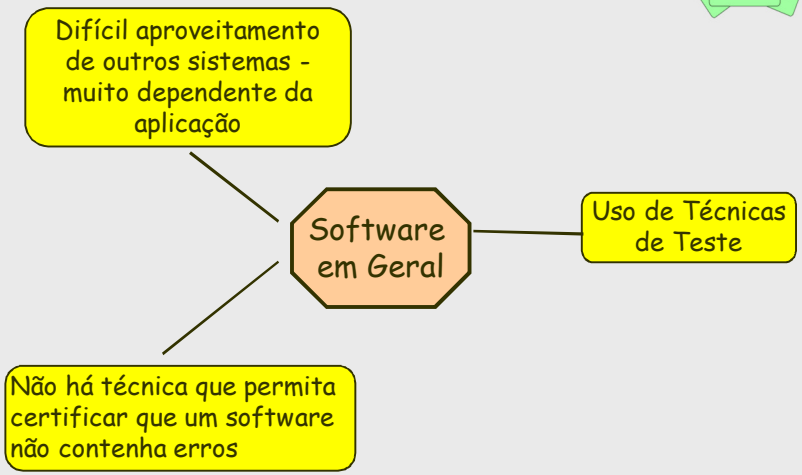


Propriedades do SW

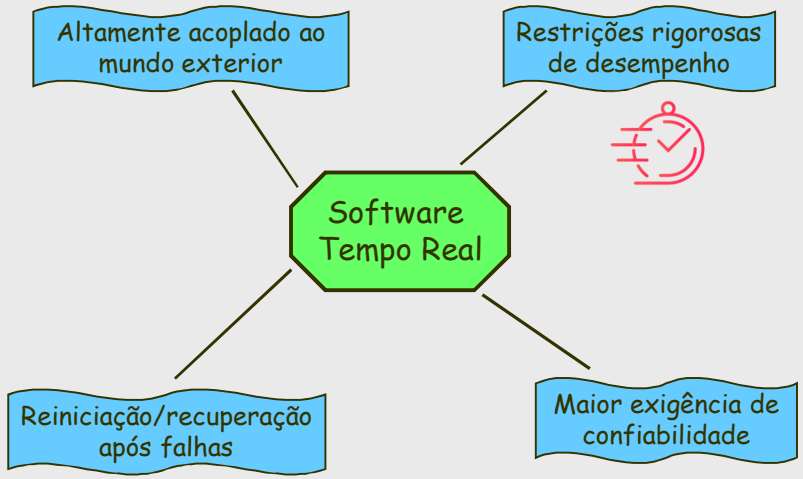
Ideal x Real



Software em Geral



Software de Tempo Real



Software de Tempo Real - Tarefas

- Coordenação entre processos
- Processamento de interrupções
- Tratamento de E/S
- Implementação de restrições temporais
- Atualização, consulta, . . . precisa de bancos de dados
- Garantia de precisão de resultados

Há algumas decisões a serem tomadas

Software de TR - Parâmetros

Tempo de Resposta → Ação

evento

Complexidade de Algoritmos

Taxa de Transferência de Dados

Software de TR - Características

- Latência de Interrupção
- Latência do Barramento
- Desempenho dispositivos de E/S
- Hardware em geral
- Tempo de Acesso à Memória
- Implementação SW
- Tamanho de buffers

Tempo de Resposta → Ação

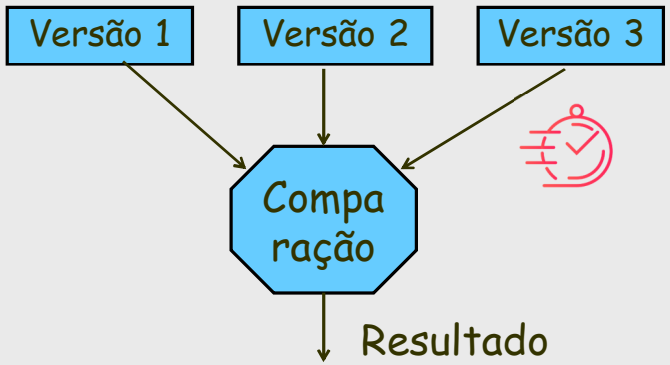
evento

Complexidade de Algoritmos

Taxa de Transferência de Dados

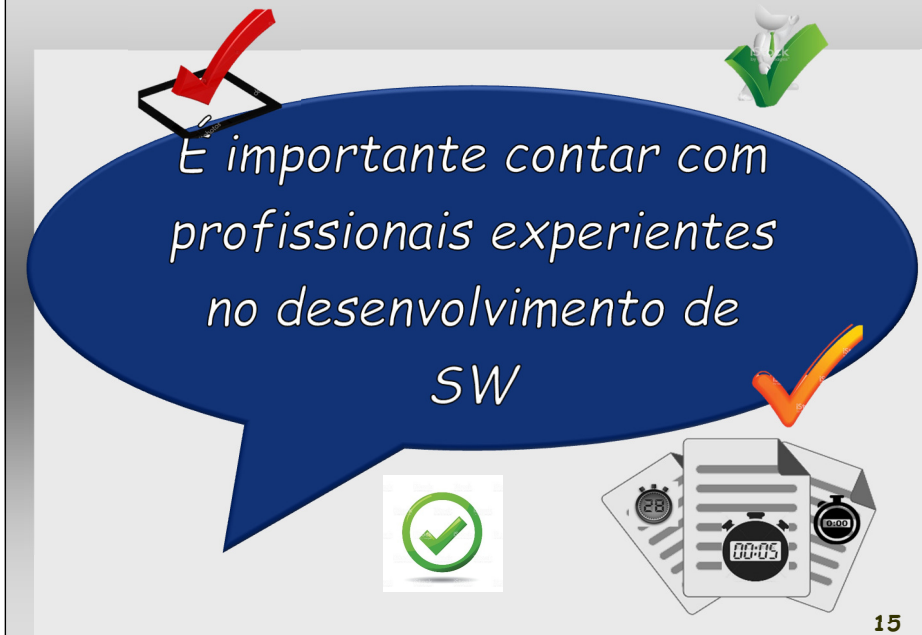
Redundância em SW para TR

- Programação N-Versões

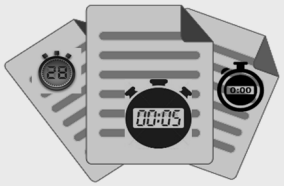


Redundância em SW para TR

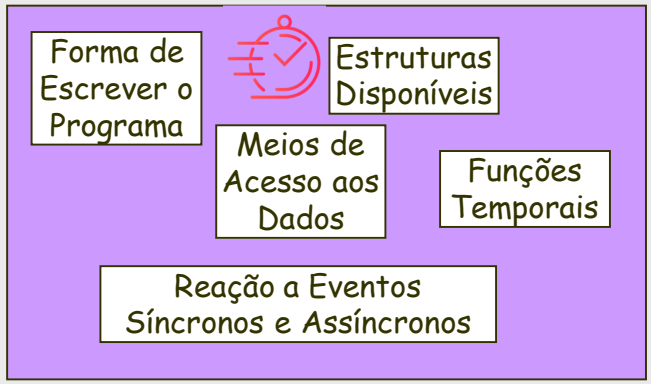
- Alternativas caso haja desacordo das saídas
  - Novo processamento
  - Restart
  - Transição para um estado pré-definido
  - Confiança em uma das versões
- N-Versões: sucesso desta técnica depende
  - Especif. Inicial: erros se propagam p/todas as versões
  - Independ. equipes de projeto: evitar erros comuns
  - Orçamento: muito mais caro/demorado produzir e manter N-Versões



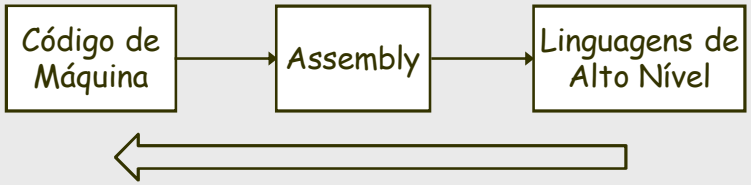
Linguagens de Programação e Sistemas de Tempo Real



Linguagens de Programação e STR



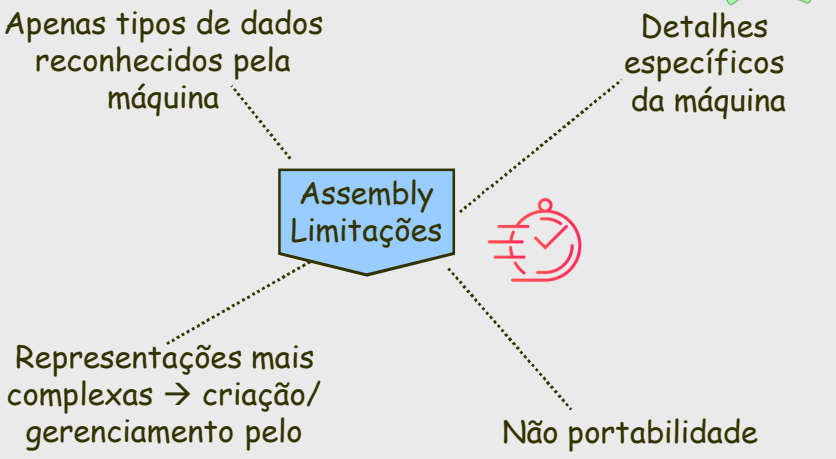
Linguagens de Programação e STR



Linguagens de Programação e STR

- Assembly
  - Linguagem inicialmente utilizada
  - Microcomputadores não suportavam linguagens de alto nível
  - Aproveitamento altamente eficiente de recursos

Linguagens de Programação e STR



Linguagens de Programação e STR

- Alto Nível
  - Aumento de poder das máquinas
  - Compiladores mais eficientes: velocidade obtida é suficiente à maioria das aplicações de TR
  - Podem não suportar todas necessidades de STR: complementação com assembly e suporte de SO
- Interface com Assembly
  - Situações críticas: parte do código em assembly
  - Integração: chamadas / passagem de dados

Linguagens de Programação e STR

Ling. Alto Nível

- Grande variedade de estruturas de dados/controle
- Cuidam de detalhes de "baixo nível" para o programador
- Menor custo de desenvolvimento
- Funcionamento em máquinas diferentes - programa de tradução
- Facilidade de escrita
- Escritas muito mais rapidamente

Linguagens de Programação e STR

Ling. Alto Nível - Interpretador

- Interpreta/executa o programa comando a comando
- Ferramentas de debug (execução passo a passo, breakpoints, traces, . . .)
- Linguagem com compilador e interpretador
  - Programa desenvolvido com o interpretador
  - Programa correto e terminado – compilado → código objeto final

Linguagens de Programação e STR

Linguagens de Programação e STR

Experiência do Programador

- Programa assembly eficiente: conhecimento da máquina e seu código
- Programador médio não produz código mais eficiente do que um bom compilador

Não há informação que permita afirmar que linguagem/compilador seja superior a outro

Utilização de testes / programas de benchmark

Selecionar benchmark mais próximo da aplicação

Testes: aspectos específicos

Linguagens de Programação e STR

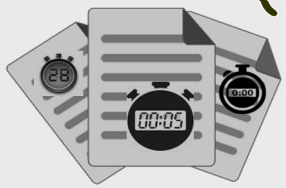
- Algumas funções de timer
  - Get_Time
  - Get_Duration
  - Set_Time (t)
  - Set_Duration (t)
  - Delay_by (Δt)
  - Delaty_until (t)
  - Set_Alarm (Δt)

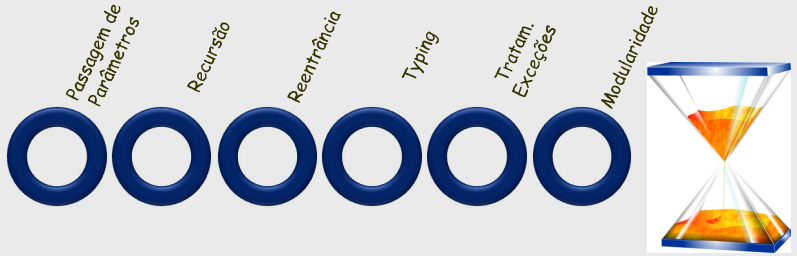
Critérios de Seleção de Linguagens para STR

- Portabilidade
- Familiaridade
- Qualidade de Compiladores
- Experiência
- Simplicidade
- Disponibilidade
- Eficiência
- Propriedade de Leitura
- Flexibilidade

Utilizar linguagens de programação de baixo nível? De alto nível?

Características (Tempo Real) das Linguagens de Programação



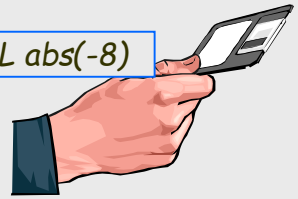


Passagem de Parâmetros

- **Call by Value**: valor do parâmetro copiado no parâmetro de chamada da rotina

```
int abs (int x)
   {if (x < 0)
      return (-x);
   else
      return (x);
   }
```

CALL abs(-8)



- Usa pilha do processador - **rapidez**
- Não usar call by value para passar uma estrutura (pode estourar a pilha)

Passagem de Parâmetros

- **Call by Reference** (by address): endereço do parâmetro é passado pela rotina acionadora
  - Acionamento de rotinas - **mais demorado** - instruções indiretas

```
PROGRAM TEST
REAL X, Y, Z
X = 4.0
Y = 3.0
CALL MEDIA (X, Y, Z)
END
```


Recursão

- Uma rotina chamar a si própria
  - Ex. cálculo do MDC

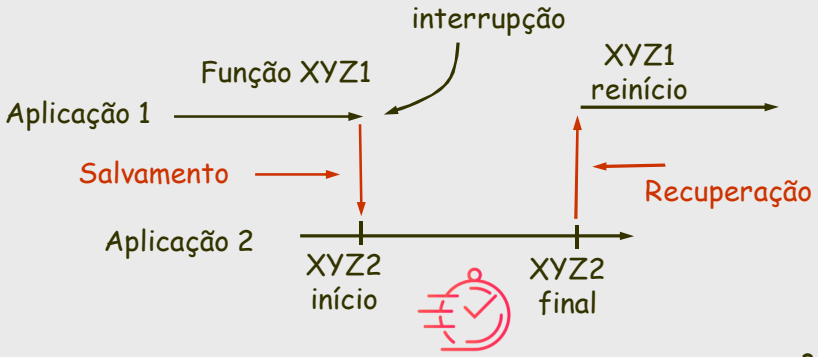
```
procedure mdc (x, y: integer);
begin
  if (y = 0) then
    writeln (x)
  else
    mdc (y, (x mod y))
end;
```

Recursão

- Característica ruim para TR: alocação de pilha (passagem de parâmetros / armazenam. de variáveis locais)
  - Tempo para realizar essas operações
  - Difícil previsão do tamanho da pilha/memória
- Alternativa: representar loops de controle

Reentrância

- Uma mesma função interrompida e utilizada em outra aplicação
- Salvamento/recuperação estado da execução da rotina antes/após reentrância



Typing

- Linguagens tipadas
  - Cada variável/cte: tipo específico
  - Declaradas antes do uso
- Linguagens fortemente tipadas (strong typing)
  - Proíbem mistura de diferentes tipos em operações/atribuições
  - Previnem
    - Truncamento/ arredondamento indesejáveis
    - Gasto de tempo

Typing

```
int i, j;
float k, l, m;
j = i * k + m;
j = (int)((float)i * k + m)
```

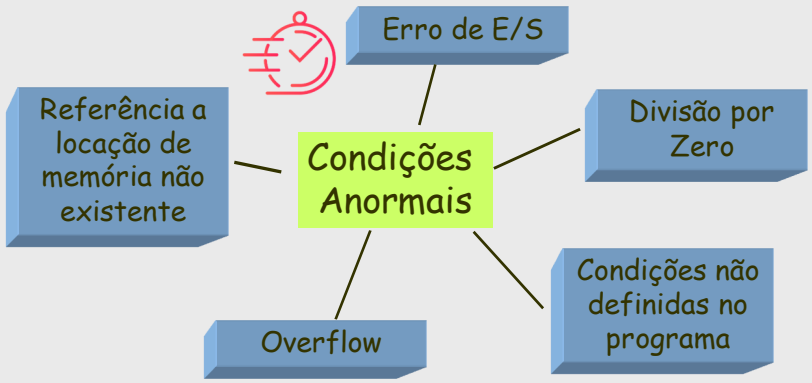
rebaixamento — promoção

```
int a;
float x;
a := x;
```
✗

```
int a;
float x;
a := int(x);
```
✓

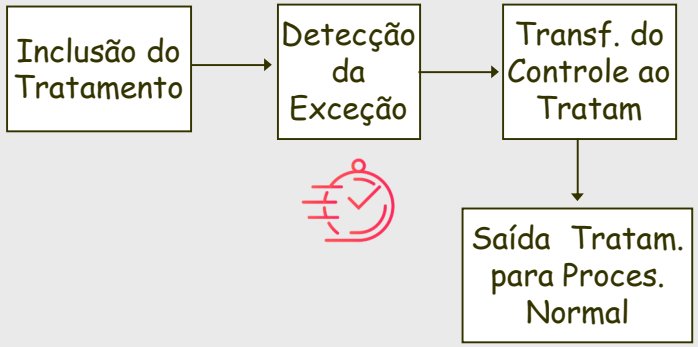
Tratamento de Exceções

- Tratamento de erros/condições anormais na execução do programa
- Exceção - código específico para seu tratamento



Tratamento de Exceções

- Seqüência usual de ações



Consome tempo

Modularidade

Divisão em módulos: altamente recomendável - ponto de vista eng. SW

Aplicações de tempo real: pode não ser totalmente indicada

Overhead associado às chamadas de subrotinas

Analisar as opções disponíveis considerando cada aplicação

Geração de Código

OTIMIZADORES AUTOMATIZADOS???

Técnicas de Otimização

Técnicas de Otimização

Identidades Aritméticas

- Usar identidades para eliminar código
- Eliminar multiplicar por 1 ou somar com 0

Redução pela Força

- Usar macroinstrução mais rápida disponível para fazer um cálculo
- Multiplicação, substituir um inteiro por inteiros potência de 2: shifts no lugar da multiplicação
- Divisão leva mais tempo: substituir a multiplicação pelo recíproco: x/2 por x*0,5

Técnicas de Otimização

Eliminação de Sub-Expressões Comuns

- Evitar cálculo de mesma subexpressão mais de uma vez

```
x := y + a * b;        t := a*b;
y := a * b + z;        x := y + t;
                       y := t + z;
```

Funções Intrínsecas

- Qdo possível, usar fções intrínsecas (exponencial, ...) no lugar de fções simples
- Substituir chamada de funções por código corrido (elimina necessidade de passagem de parâmetros)

Técnicas de Otimização

Reunião de Constantes

```
x := 2.0 * y * 4.0;          x := 8.0 * y;

z:= π/2 + y                  pi_div2 := π/2;
                             z := pi_div2 + y;
```

Otimização Invariante de Loop

- Mover cálculos para fora do loop

```
x := 100;                    x := 100;
while (x > 0) do             t := y + z;
   x := x - (y + z);         while (x > 0) do
                                x := x - t;
```


Técnicas de Otimização

Eliminação de Indução de Loop

- Variável de loop altera seu valor: haver incremento/decremento de alguma constante

```
for i := 1 to 10 do          for j := 2 to 11 do
  a[i+1] : = 1                 a[j] := 1;
```

Uso de Registradores

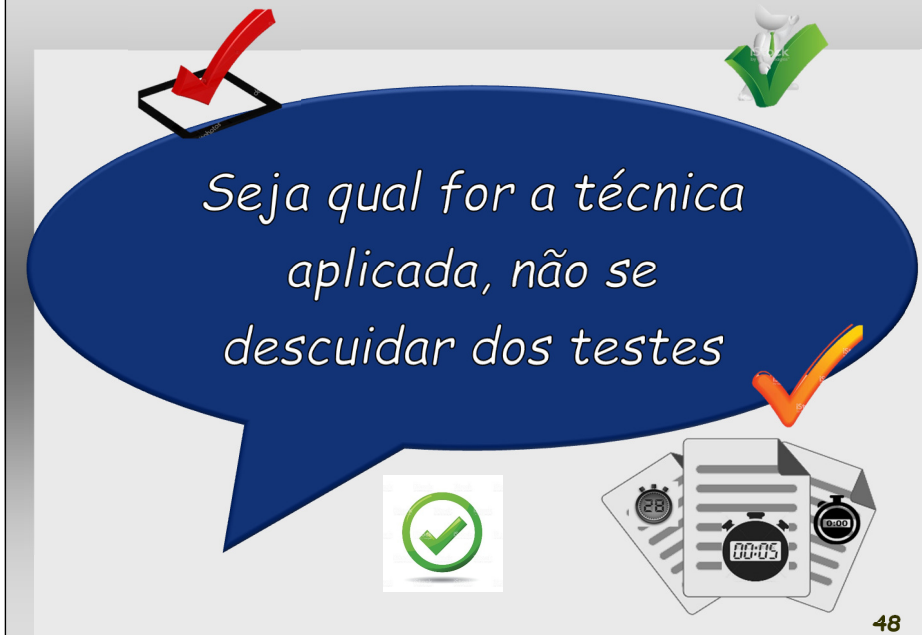
- Quando disponível: fazer operações sobre registradores (em vez da memória)

Técnicas de Otimização

Remoção de Código Morto ou Não Alcançável

- Código que nunca pode ser alcançado no fluxo normal de controle

```
debug = False          #ifdef debug
. . . . .                    {
if (debug)                    . . . . . .
  {                           }
   . . . . .                 #endif
  }
```

Técnicas de Otimização

Otimização do Fluxo de Controle

```
goto label1;                goto label2;
label0: y = 1;              label0: y = 1;
label1: goto label2;        label1: goto label2;
```

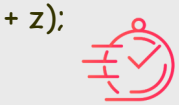
Propagação de Constantes

- Substituir comandos de atribuição de variáveis por comandos de atribuição de constantes

```
x := 100;                   x := 100;
y := x;                     y := 100;
LOAD R1, 100                LOAD R1, 100
STORE x, R1                 STORE x, R1
LOAD R1,x                   STORE y, R1
STORE y, R1
```


Técnicas de Otimização

Eliminação de Armazenamento Morto



```
t := y + z;                 x: = func (y + z);
x := func(t);
```

Eliminação de Variáveis Mortas

```
x := y + z;                 x: = y
x := y;
```

Curto Circuito de Código Booleano: testar cada expressão em separado

```
if (x>0) AND (y>0) then         if (x>0) then
    z := 1;                       if (y>0) then
                                      z := 1;
```


Técnicas de Otimização

Desenrolar Loop: reduzir overhead

```
for i = 1 to 6 do
  a[i] := a[i] * 8
```

```
a[1] := a[1] * 8
. . . . . .
a[6] := a[6] * 8
```

Junção de Loops

```
for i := 1 to 100 do
  x[i] := y[i] * 8;

for i := 1 to 100 do
  z[i] := x[i] + y[i];
```

```
for i = 1 to 100 do
  begin
    x[i] := y[i] * 8;
    z[i] := x[i] + y[i];
  end;
```


Técnicas de Otimização

Eliminação de Cross Jump

```
case x of
  0:  x := x + 1;
  1:  x := x * 2;
  2:  x := x + 1;
  3:  x := 2;
end;
```

```
case of
  0,2:  x := x + 1;
  1:  x := x * 2;
  3:  x := 2;
end;
```

Técnicas de Otimização

Outras Otimizações

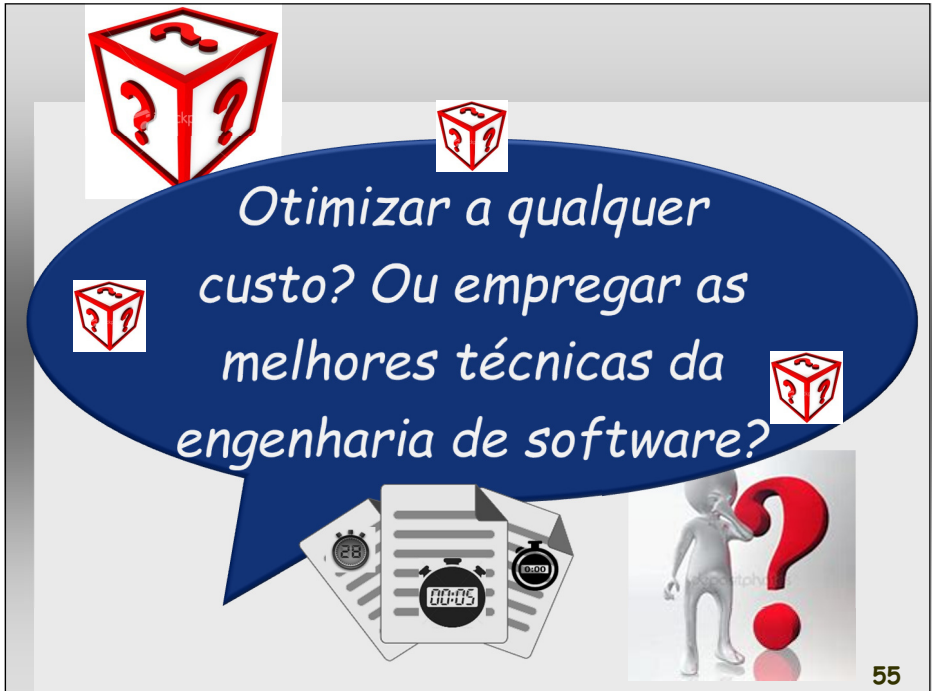
- Arranjar séries de ***IFs***: mais provável a falhar em primeiro lugar
- Arranjar séries de ***ANDs***: mais provável a falhar em primeiro lugar
- Arranjar séries de ***ORs***: mais provável a dar certo em primeiro lugar

Técnicas de Otimização

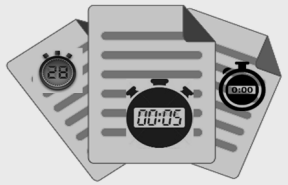
Outras Otimizações

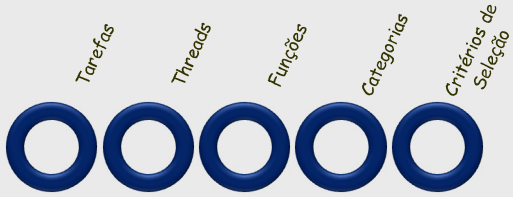
- Substituir testes de funções monotônicas por testes em seus parâmetros

  if($e^x < e^y$) then if ($x < y$) then
- Reunir procedimentos/dados mais usados: maximizar a localidade de referência

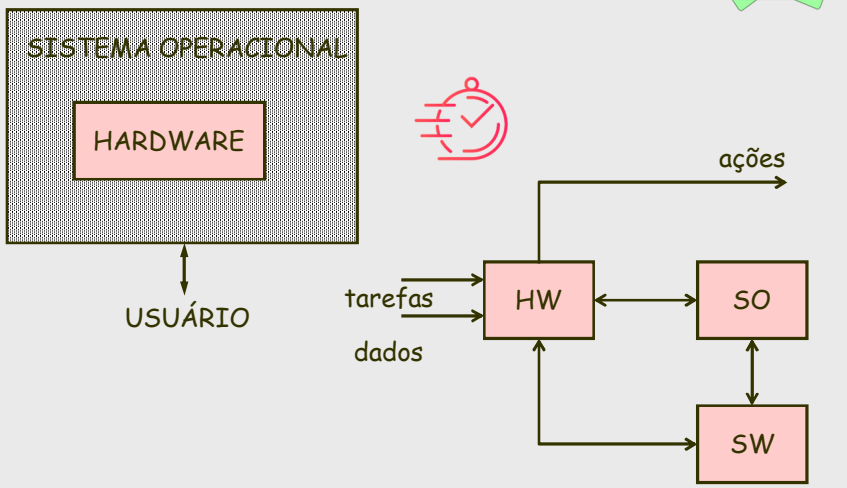


Sistemas Operacionais para Tempo Real

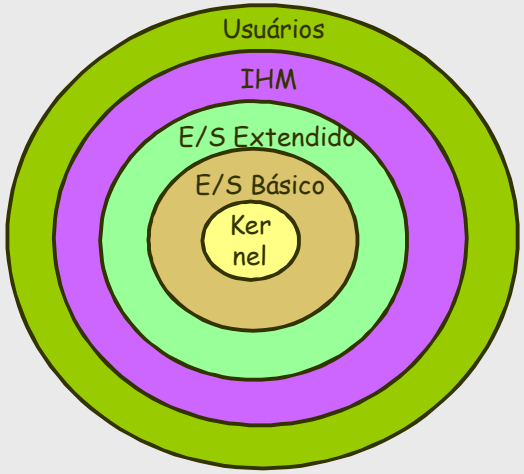




Sistemas Operacionais para TR



Sistemas Operacionais para TR



Sistemas Operacionais para TR

Atender Tarefas e Threads

Tarefas

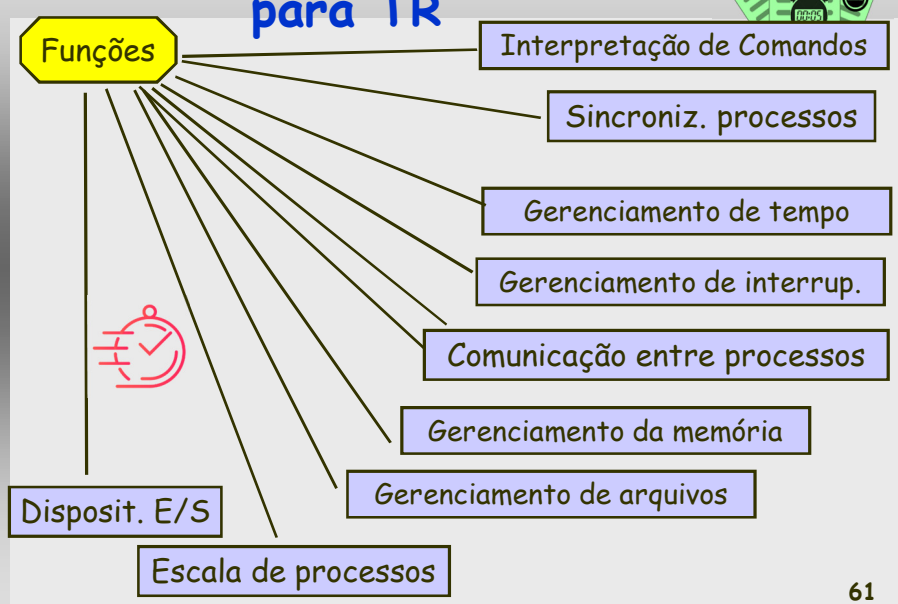
- Tbém chamadas de processos no contexto de SO
- Incluem: espaço de endereçamento, conjunto de arquivos abertos, de regras de acesso e de registradores do processador
- Tempo de chaveamento entre tarefas: tempo necessário para mudar tal conjunto

Sistemas Operacionais para TR

Threads

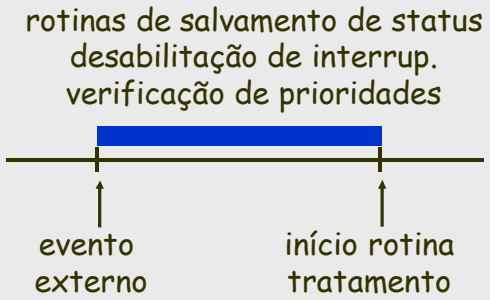
- "Tarefas leves"
- Únicos atributos particularizados: registradores do processador
- Demais atributos: herdados da tarefa hospedeira
- Por ex., todas as threads de uma mesma tarefa compartilham o mesmo espaço de endereçamento

Sistemas Operacionais para TR



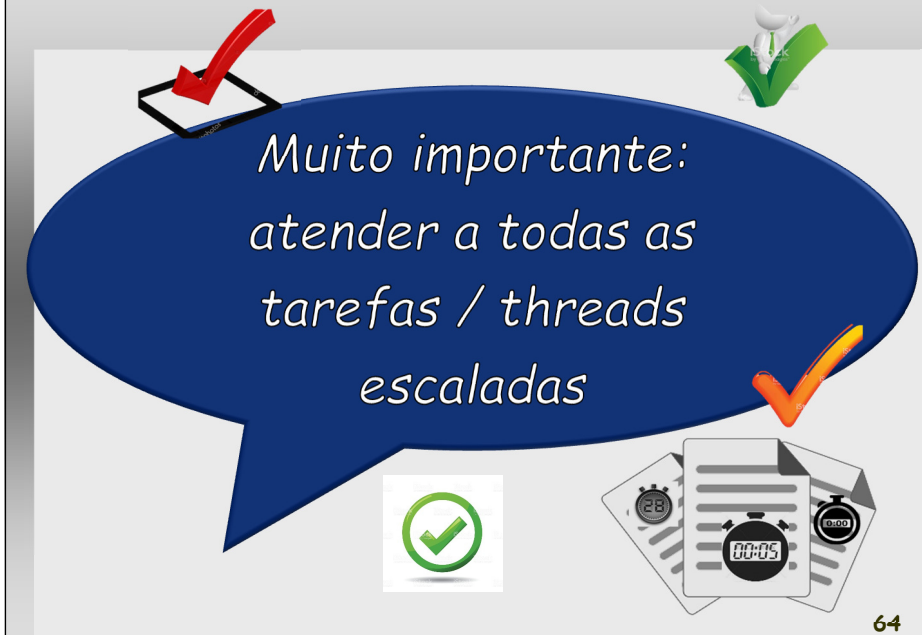
SO – Exigências TR

- **Algoritmos de Escalação**
  - Prioridades/preempção
- **Tratamento/Latência de Interrupções**
  - Não violar preempção de processos

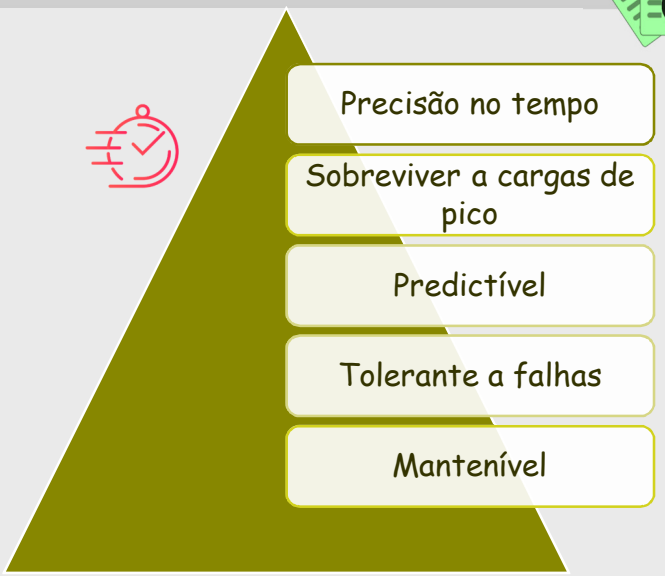


SO – Exigências TR

- **Tempos Tratam. Processos**
  - Criação/ Execução
  - Suspensão/ Reativação
  - Fila de Mensagens
  - Alocação de Memória
- **Escalabilidade**
  - Todos processos atender seus deadlines



SO – Exigências TR



Tipos de Processos

Independente: não se comunica ou não se sincroniza com outros processos

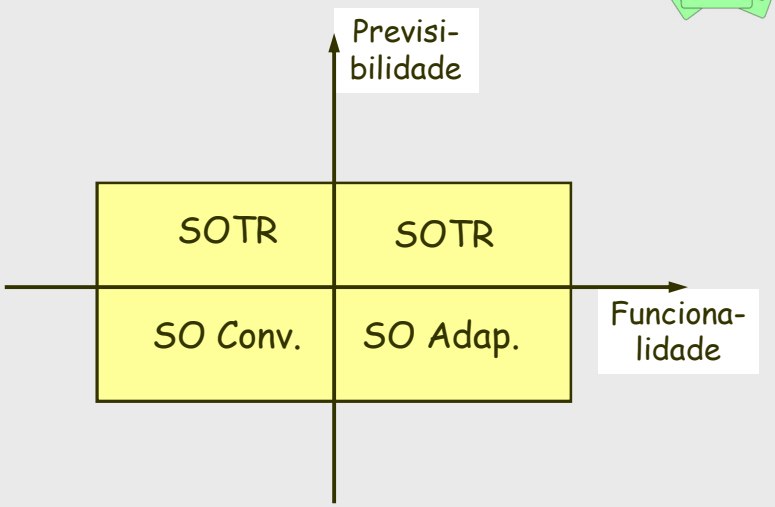
Cooperativo: se comunica e sincroniza regularmente suas atividades

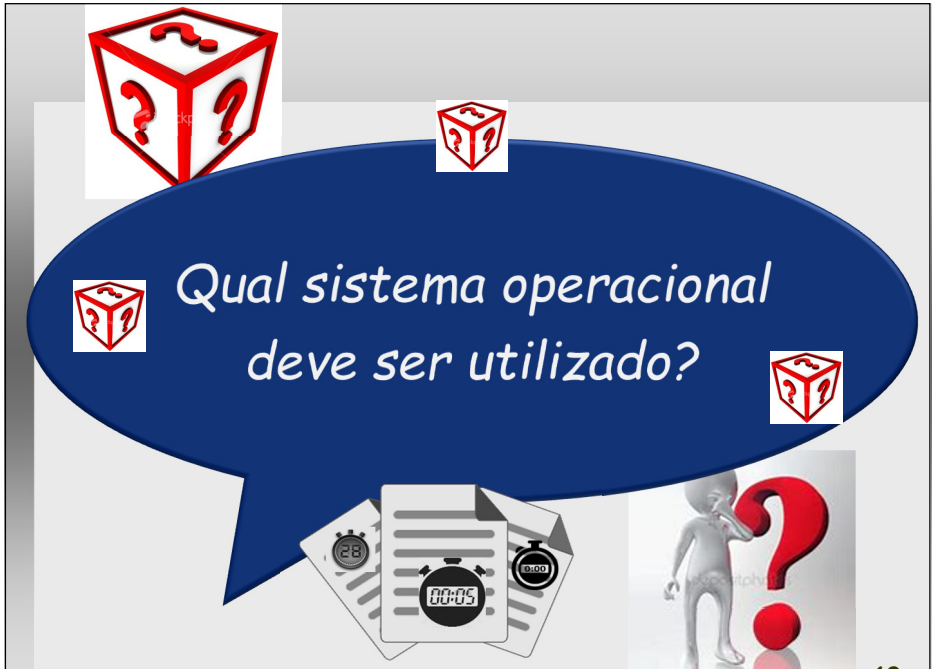
Competidor: processos competem pelo uso de recursos (memória, periféricos, processador) → exige comunicação e sincronização

RTOS – Comprar ou Desenvolver?

- Compra de COTS
  - Disponibilidade imediata
  - Ferramentas de depuração
  - Portabilidade
  - Muitas funções não necessárias
- Desenvolvimento
  - Tempo de projeto
  - Mais simples e rápido
  - Menor overhead

Sistemas Operacionais





Um critério para seleção de RTOS

- Tempo mínimo de latência de interrupção (m1)
- Nº máx. processos suportados simult.(m2)
- Memória necessária para suportar o SO (m3)
- Nº mecanismos disp. p/ escala de processos (m4)
- Nº métodos disp. p/ comum. entre processos (m5)
- Suporte pós-venda (m6)

Real-Time Systems Design and Analysis, Phillip Laplante, IEEE Press

Um critérios para seleção de RTOS

- SW disponível para desenvolver aplicações (m7)
- Nº de processadores suportados (m8)
- Disponibilidade de código fonte (m9)
- Tempo para chaveamento de contexto (m10)
- Custo (m11)
- Nº de outros SO compatíveis (m12)
- Nº de protocolos de rede suportados (m13)

$$M = \sum_{i=1}^{13} w_i m_i$$

w_i: importância para a aplicação

Critérios para seleção de RTOS

Table 3.10 Summary data for real-time operating system A[a]

Criterion	Description	Rating	Comment
m_1	Minimum interrupt latency	*	CPU dependent
m_2	Number of tasks supported	0.5	32-Thread priority levels
m_3	Memory requirements	0.7	ROM: 60 K
m_4	Scheduling mechanism	0.25	Preemptive
m_5	Intertask synchronization mechanism	0.5	Direct message passing
m_6	Software support (warranty)	0.5	Paid phone support
m_7	Software support (compiler)	1	Various
m_8	Hardware compatibility	0.8	Various
m_9	Royalty free	0	No
m_{10}	Source available	1	Yes
m_{11}	Context switch time	*	NA
m_{12}	Cost	0.7	Approximately $2500
m_{13}	Supported network protocols	1	Various

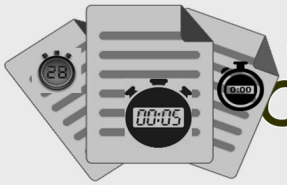
[a]Some of the specific details have been deliberately omitted to preserve the identity of the product.

Critérios para seleção de RTOS

Table 3.11 Decision table for inertial measurement system

Criterion	Description	Weight, w_1	A	B	C	D	E
m_1	Minimum interrupt latency	1	0.5	0.8	1	0.5	1
m_2	Number of tasks supported	0.1	0.5	0.5	0.5	1	1
m_3	Memory requirements	1	0.7	0.2	0.5	1	0.9
m_4	Scheduling mechanism	0.5	0.25	0.5	0.25	1	0.25
m_5	Intertask synchronization mechanism	1	0.5	1	0.5	1	1
m_6	Software support (warranty)	1	0.5	0.5	1	0.8	1
m_7	Software support (compiler)	1	1	0.75	1	1	0.5
m_8	Hardware compatibility	0.1	0.8	0.5	0.2	1	0.2
m_9	Royalty free		0	1	1	1	1
m_{10}	Source available	1	1	1	0	0.4	1
m_{11}	Context switch time	1	0.5	0.5	0.5	1	0.5
m_{12}	Cost	0.4	0.5	0.5	0.1	0.1	0.7
m_{13}	Supported network protocols	0.5	1	1	1	1	0.6
M			5.66	5.80	5.24	6.94	6.73

Here the metric M in equation 3.10 is computed for five candidate RTOS (A through E). From the last row it can be seen that RTOS D is the best fit in this case.



Alguns Sistemas Operacionais para Tempo Real

Classificação dos Sist. Oper.

- Kernels Especializados
- Extensões de Sistemas Operacionais Comerciais
- Kernels de pesquisa

Kernels Especializados

- Altamente especializados para a aplicação
- QNX, PDOS (Eyring Corporation), VRTX32 (Ready Systems), VxWorks (Wind River Systems)

Rápido chaveamento de contexto

Pequeno tamanho – funcionalidade mínima necessária

Resposta rápida a interrupções

Minimiza tempos com interrupção desabilitada

Kernels Especializados

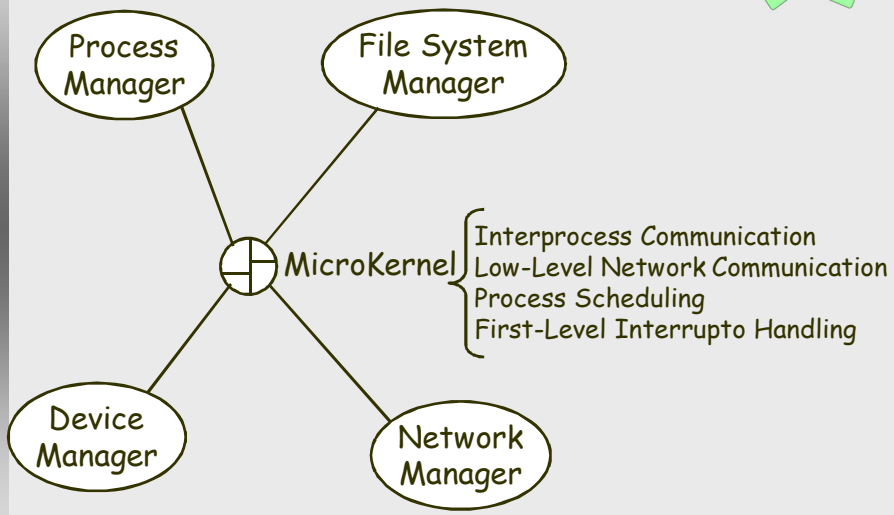
Mecanismo de escalamento com prioridades

Alarmes e timeouts especiais

Mecanismo de fila

Primitivas para atrasar, suspender e retomar a execução de processos

QNX



QNX

Cada processo tem prioridade
- 0 (menor) .....31 (maior)

Processos com mesma prioridade
- FIFO com preempção
- Round Robin com preempção ou time-slice
- Adaptive – deslocamento de prioridade

Disponibilidade de tabelas com tempos de latência de interrupção e de escala

QNX Neutrino: aplicações menores //
QNX Tradicional: aplicações maiores

Extensões de Sist.Oper. Comerciais

- UNIX → RT-UNIX, MACH → RT-MACH
- Comitê definindo normas para Sistemas Operacionais para Tempo Real - POSIX

Timers

Escalamento de prioridades

Memória compartilhada

Semáforos

Comunicação entre processos

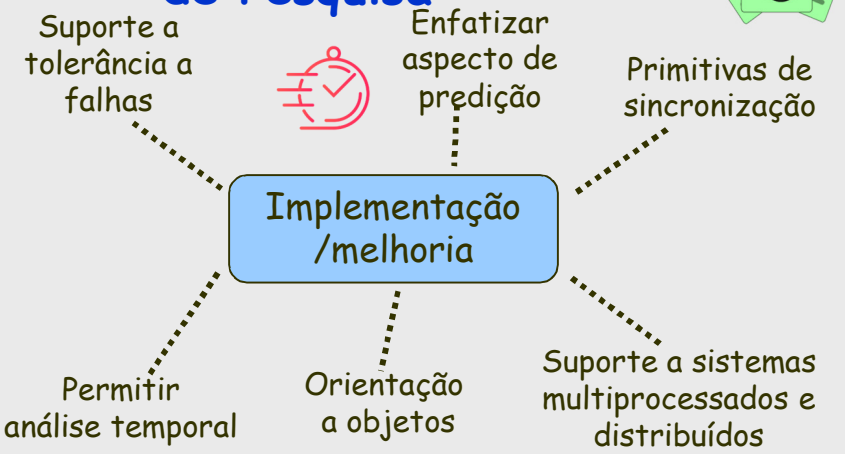
Notificação assíncrona de eventos

E/S síncrona/assíncrona

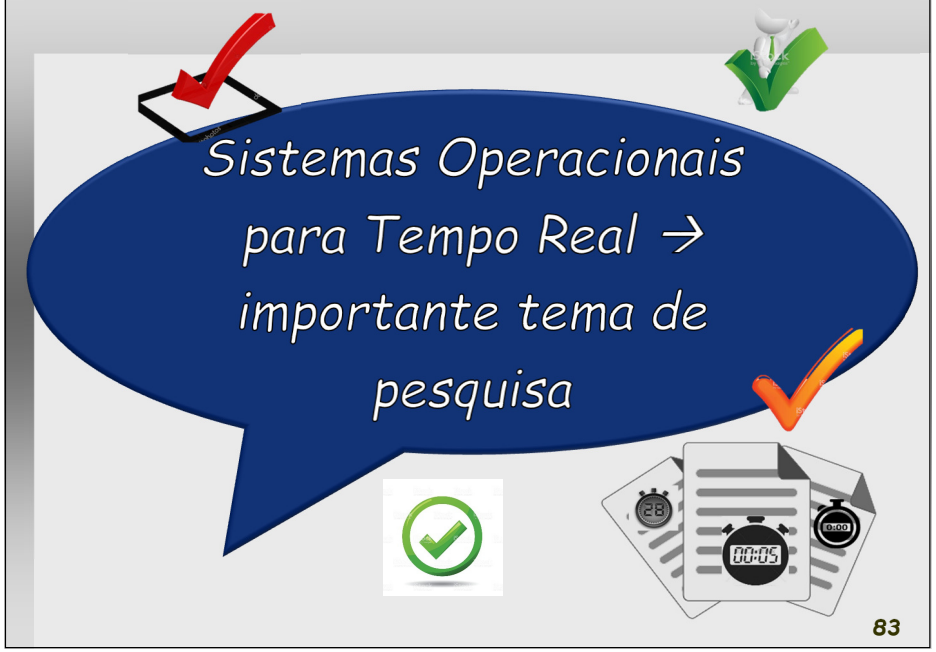
Real-Time Linux

- Extensão Linux – Microkernel coexiste com Kernel Linux
- Usa serviços "sofisticados" do Kernel Linux
- Microkernel usa o Kernel como tarefa de mais baixa prioridade (completamente preemptível)
- Interrupções: recebidas pelo Microkernel → passadas ao Kernel Linux quando não há processos de TR
- Microkernel: tarefas básicas – escalonamento baseado em prioridades, alocação estática de memória

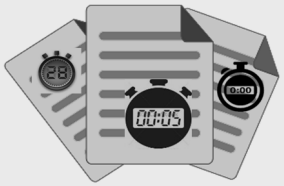
Sistemas Operacionais de Pesquisa

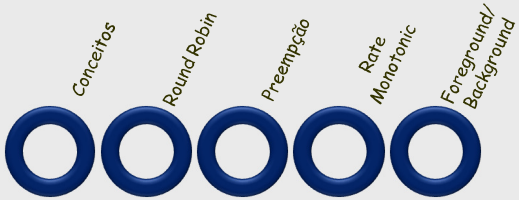


- Exemplos: MARS, SPRING, ARTS, HARTOS, . .



Tempo Real Escala de Processos





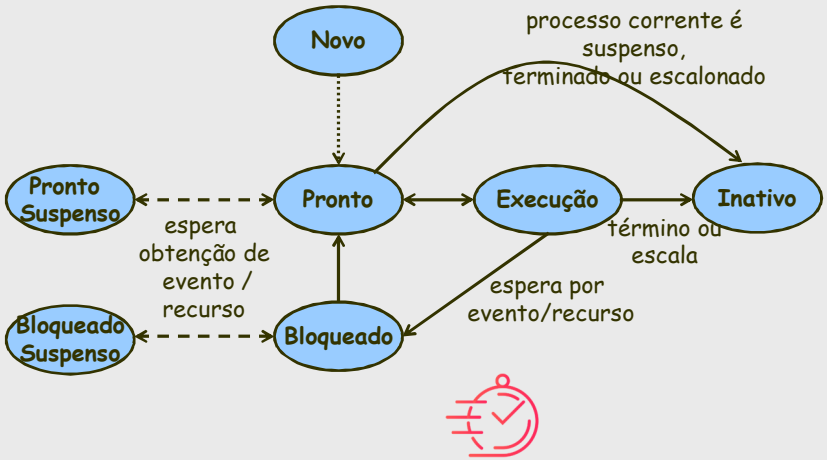
Escala de Processos

- Longo Prazo
  - Determinar novos processos no sistema – criação de processos
- Médio Prazo
  - Decidir que processos devem ocupar a memória principal/secundária – gerenciamento da memória
- Curto Prazo
  - Processos na memória principal
  - Decidir qual processo será executado

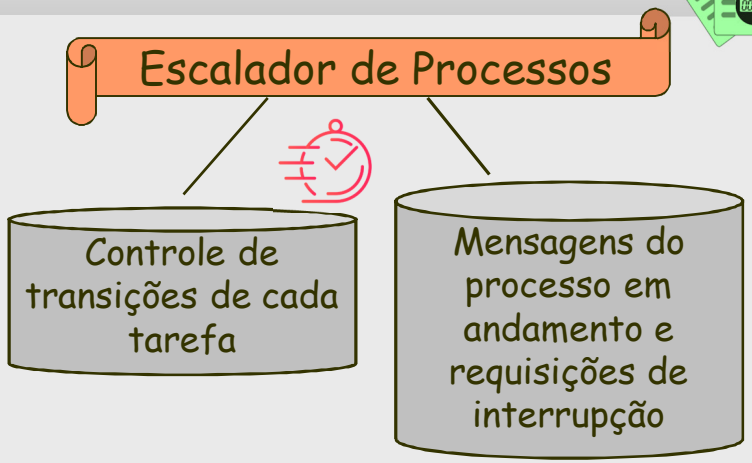
Escala de Processos

- Troca de contexto entre processos
  - Parar execução do processo corrente
  - Salvar contexto do processo que sai
  - Obter/restaurar contexto do processo que entra
  - Iniciar/reiniciar esse processo
- Tempo de Retorno – Turnaround Time
  - Tempo total desde a entrada na fila até sua finalização
- Melhor política de escalonamento depende
  - Tipo dos processos
  - Critério de otimização desejado

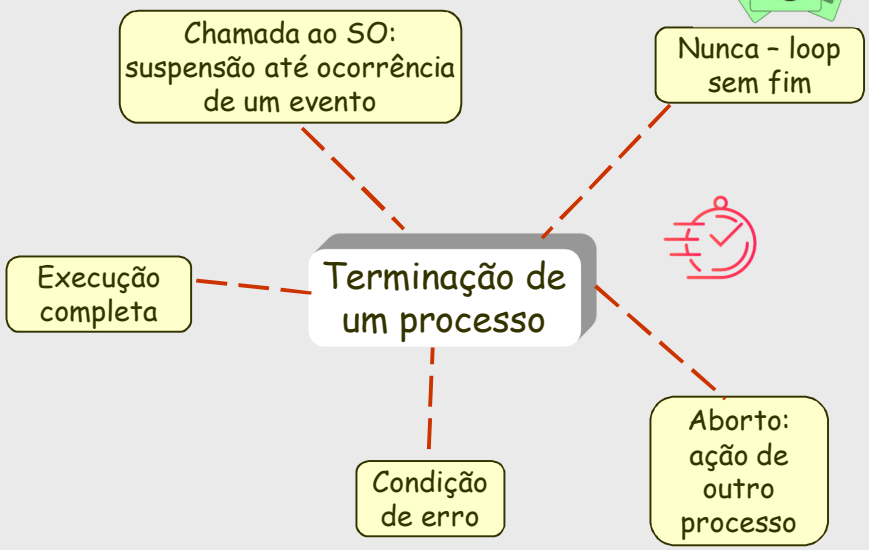
Escala de Processos em TR



Escala de Processos em TR

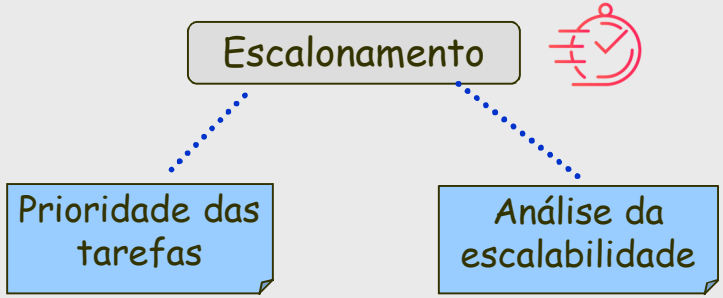


Processos em STR



Escala de Processos em TR

- Habilidade do escalador em controlar execução de processos → ***chave para eficiência de um Sistema Operacional de Tempo Real***



Escala de Processos em TR

Prioridades	
Fixas	**Variáveis**
Definidas quando da criação do processo	Definidas pelo escalador – recálculo periódico
Função da importância no sistema	Função do comportamento do sistema
Problemas no ciclo de vida do sistema	Algoritmo complexo

Escala de Processos em TR

Níveis de Prioridade

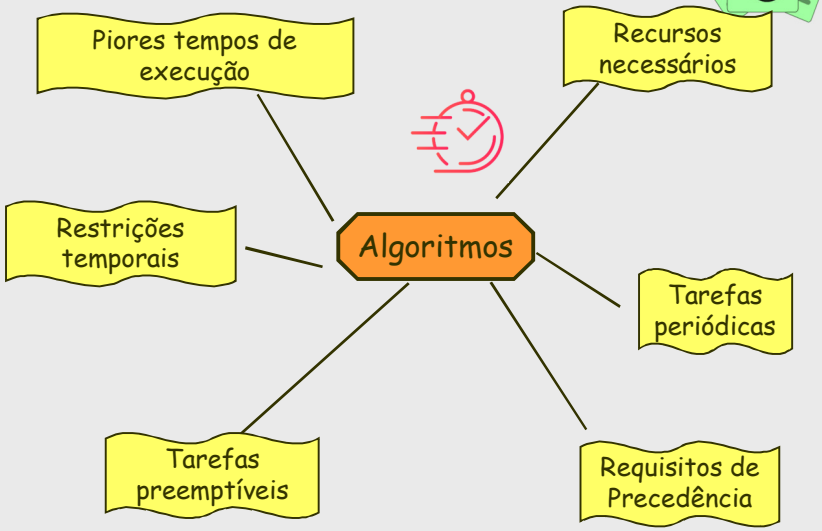
Nível 1: processos atendem eventos sinalizados por interrupções

Prioridades relativas dentro de cada nível

Nível 2: processos repetitivos - escalados a intervalos regulares

Nível 3: processo com tempo de espera não crítico

Escala de Processos em TR



Tipos de Escalonadores

- Modo Não Preemptivo
  - Processo só libera a CPU a seu critério
  - FCFS – First Come First Served (ou FIFO – First In First Out)
  - SJF – Shortest Job First
    - Se houver empate usa FCFS
- Modo Preemptivo
  - Escalonador pode retirar o processo da CPU
  - SRTF – Shortest Remaining Time First
    - Para o caso de chegar um processo com tempo menor restante que o processo atual
  - Round Robin

Tipos de Escalonamento em TR

Round Robin

Preemptivo

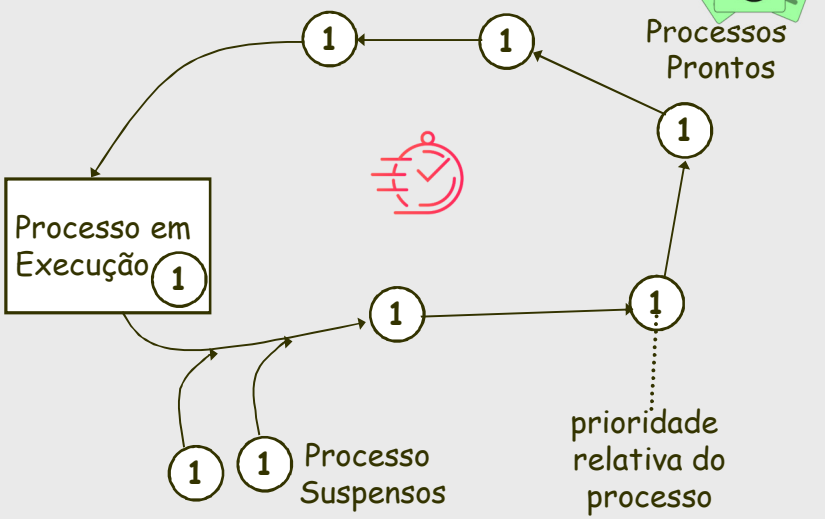
Preempção Baseada em Prioridade

Rate Monotonic

Não Preemptivo

Foreground/ Background

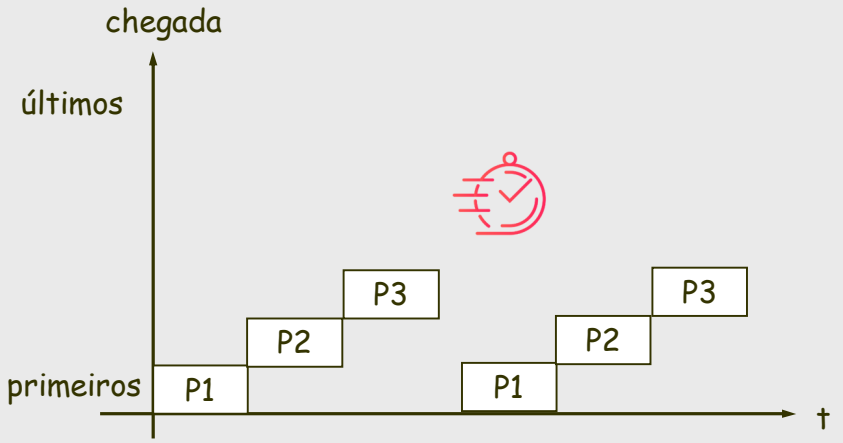
Escala Round Robin



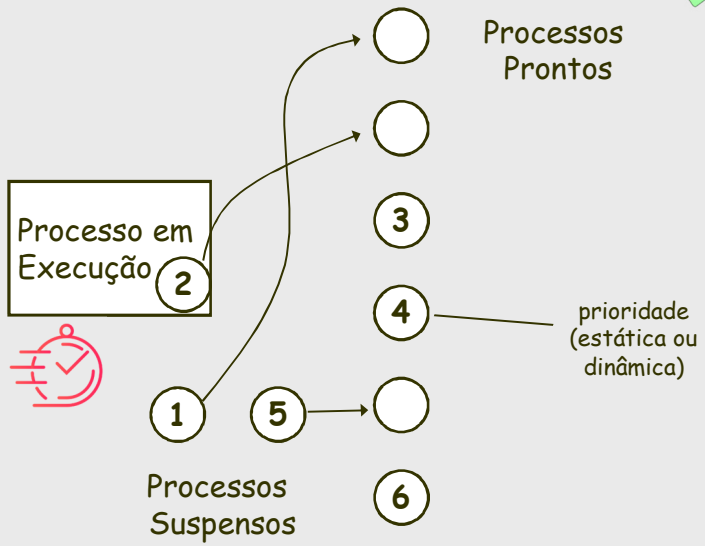
Escala Round Robin

- Todos processos têm mesma prioridade
- Lista de tarefas ready: fila FIFO
- Cada processo tem time-slice igual
- Um processo removido da lista de suspensos e colocado no fim da lista ready pode ter de esperar muito tempo até ser atendido

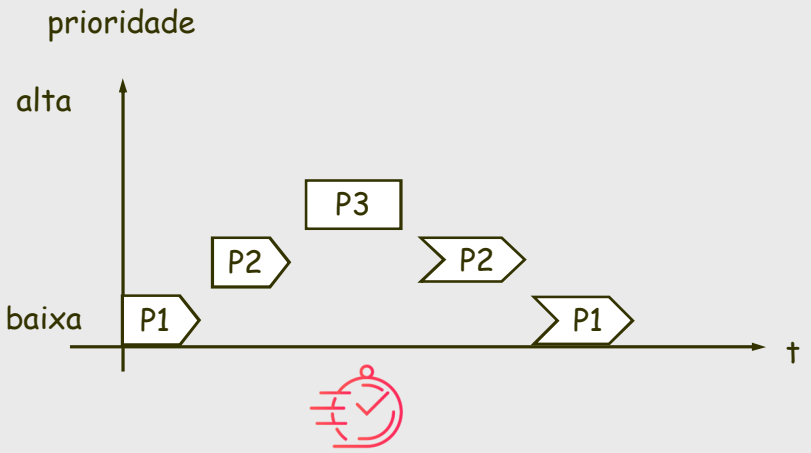
Escala Round Robin



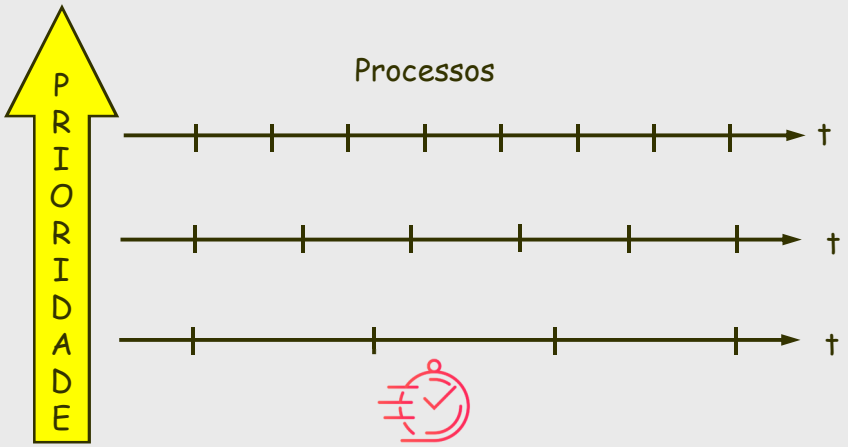
Preempção Baseada em Prioridades



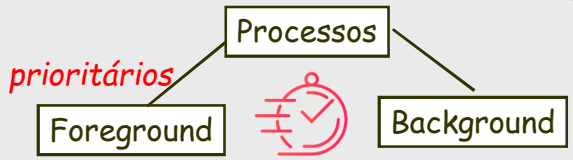
Preempção Baseada em Prioridades



Sistemas Rate-Monotonic

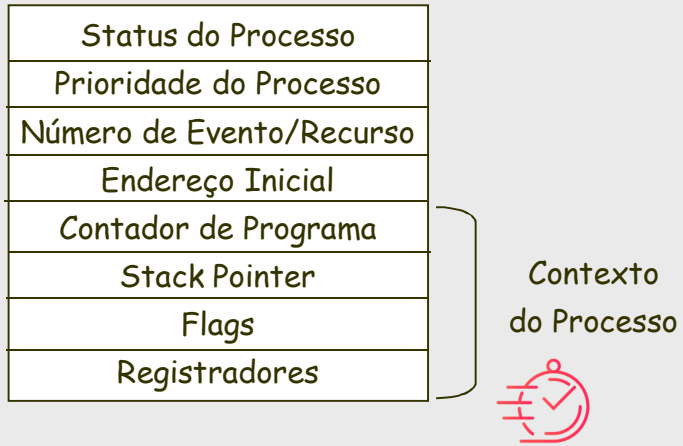


Foreground / Background



- Processos rodando em background
- Processos aceitos por interrupção: foreground
- Foreground: mais prioritárias
- Background: menos críticos (contadores, timers, . . .)
- É considerado Múltiplas Filas
- Cada fila tem seu próprio algoritmo
- Retroalimentação: processo pode ser trocado de fila

Bloco de Controle de Processos



Pode ser utilizada uma combinação de técnicas de escala de processos

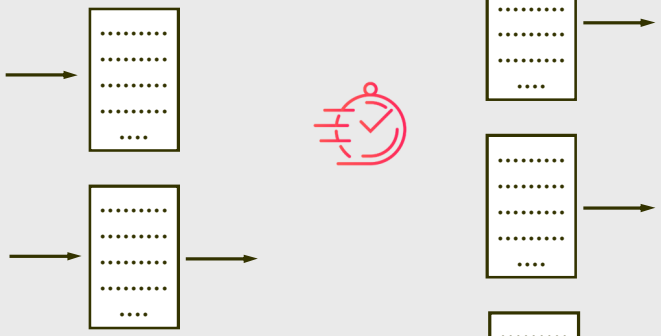
Transferência de Dados entre Processos

Problemas

Soluções

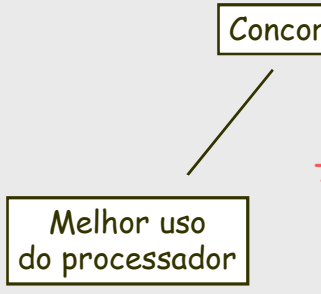
Transf. de Dados entre Processos

Problema do Produtor-Consumidor



- Produtor sobrescrever novos dados antes que o consumidor os leia
- Consumidor ler os mesmos dados várias vezes antes que o produtor os atualize

Transf. de Dados entre Processos



Transf. de Dados entre Processos

- Problema de Atualização Perdida

Processo A	Tempo	Processo B
READ (P)	t1	
	t2	READ (P)
WRITE (P)	t3	
	t4	WRITE (P)

Transf. de Dados entre Processos

- Problema de Dependência não Completada: processo utiliza informação atualizada por outra transação ainda não completada

Processo A	Tempo	Processo B
	t1	WRITE (P)
READ (P)	t2	
	t3	abortado

Transf. de Dados entre Processos

BLOQUEIO

Garantia que outros processos não alterem dados de forma indevida

Transf. de Dados entre Processos

Processo A/B	X	S
X	Não	Não
S	Não	Sim

Transf. de Dados entre Processos

- Problema da atualização perdida

Transação A	Tempo	Transação B
READ (P)	t1	
Bloqueio S em P		
	t2	READ (P)
		bloqueio S em P
WRITE (P)	t3	
requer bloqueio		Problema de
X em P		Deadlock
WAIT		
WAIT	t4	WRITE (P)
WAIT		requer bloqueio
		X em P
		WAIT
		WAIT

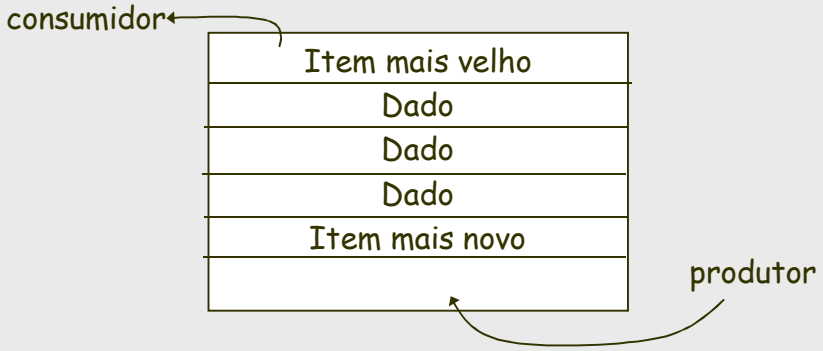
Transf. de Dados entre Processos

- Problema de dependência não completada

Transação A	Tempo	Transação B
	t1	WRITE (P)
		bloqueio X em P
READ (P)	t2	
requer bloqueio		
S em P		
WAIT		
WAIT	t3	compromissada
		ou abortada
		libera P
Resume READ (P)	t4	
Bloqueio S em P		

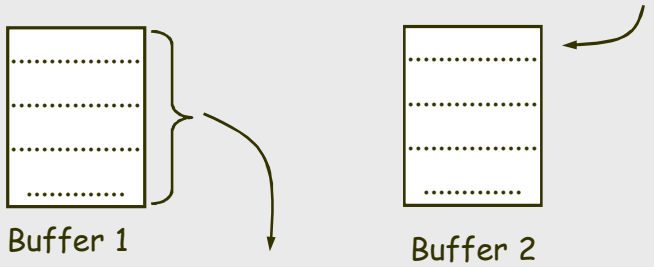
Transf. de Dados entre Processos

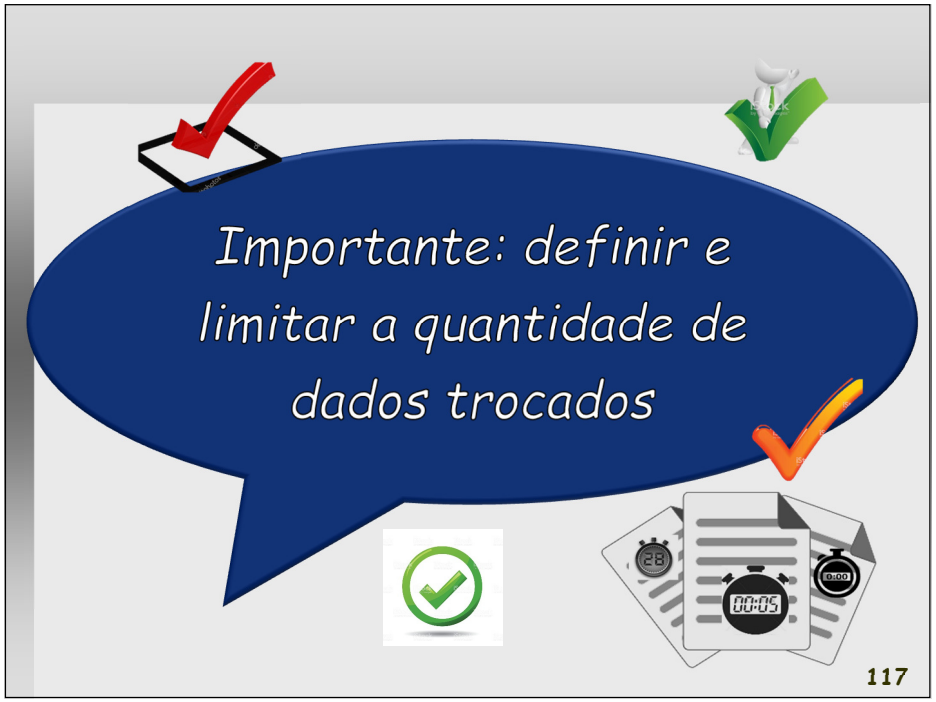
- Canais de dados entre processos

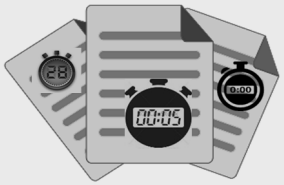


Transf. de Dados entre Processos

- Double Buffer
  - Qdo buffer i estiver completo: dados transferidos de uma só vez



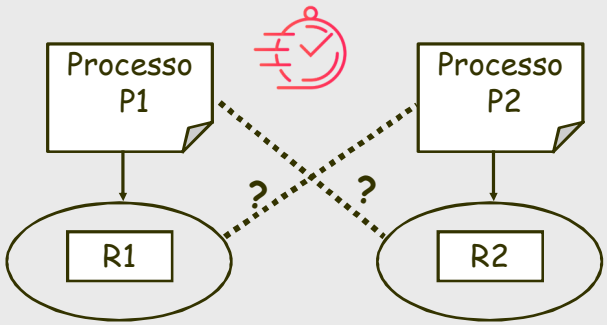




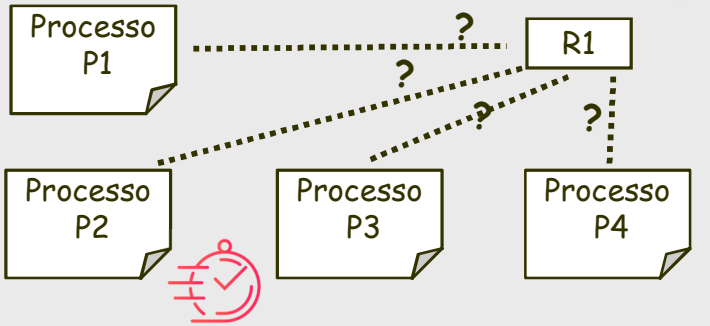
Deadlock

Deadlock

- Processos competindo por recursos finitos
- Processamento não pode ter seqüência: 2 ou mais processos aguardando recursos já alocados
- **Deadlock** → todos processos afetados suspensos indefinidamente



Deadlock



- Política de alocação ineficaz: processo nunca consegue acesso ao recurso → **Adiamento Indefinido**

Condições para Deadlock

Sem Exclusão Mútua: apenas um processo pode acessar um recurso de cada vez

Hold and Wait: processos retendo recursos enquanto aguardam por outros recursos

Sem Preempção: recurso só é liberado voluntariamente

Espera Circular: cada processo segura recursos requisitados pelo próximo processo na fila (cadeia)

Prevenir Deadlock

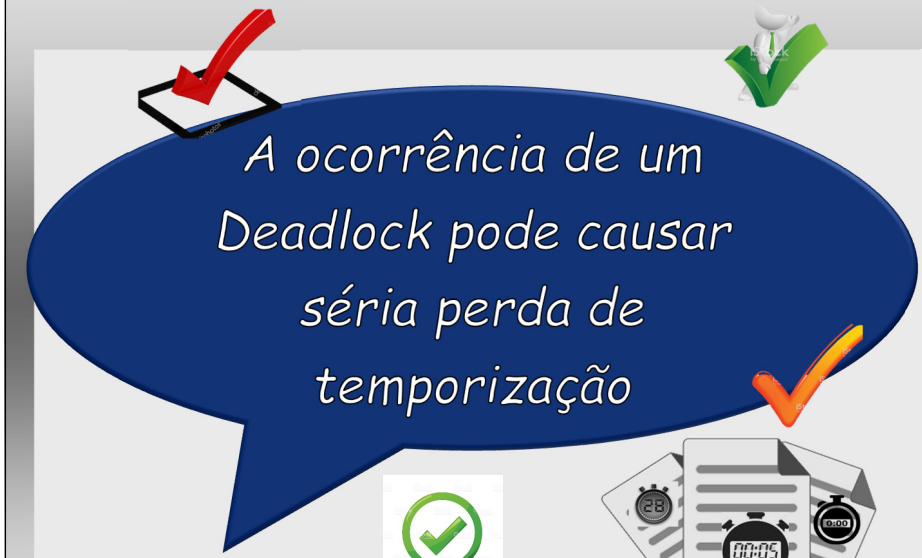
Sem Exclusão Mútua: dispositivos compartilháveis ou uso de buffers

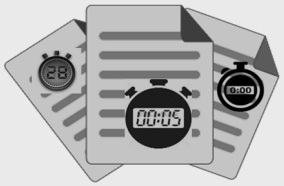
Hold and Wait: processos requisitariam todos dispositivos em seu início de execução

4 condições

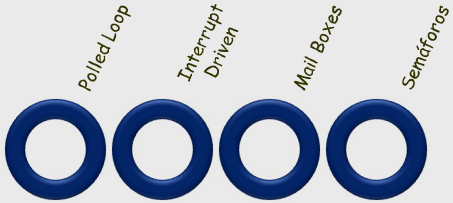
Com Preempção: processo liberar recursos que estiver utilizando, se tentar acessar outro recurso não disponível

Espera Circular: imposição de ordem aos tipos de recursos (disco:1, impressora:2, . . .)





Mecanismos de Sincronização



Polled Loop

- Processos posicionam/verificam variáveis compartilhadas → flags

Processo P1 /*processo em espera*/

. . . . . .

while flag1 = down do
  null
end

. . . . . .

end P1

Processo P2

. . . .

flag1 = up

. . . . .

end P2

Polled Loop

- Desvantagem: tempo de processamento para ficar verificando uma condição
- Opção: suspender processo em espera se a condição não for satisfeita

Processo P1 /*processo em espera*/

. . . . . .

while flag1 = down do
  suspend
end

. . . . . .

end P1

Processo P2

. . . . . .

flag1 = up

resume P1

. . . . . .

end P2

Interrupt Driven

- Processos escalados por interrupções
- Contexto e chaveamento de contexto
  - Salvar/restaurar informação suficiente para processo original ser retomado
  - Em tempo real: minimizar quantidade de informação salva (registradores, program counter, variáveis especiais)

Mensagens ou Mail Boxes

Local de memória que dois ou mais processos usam para passar status do processo

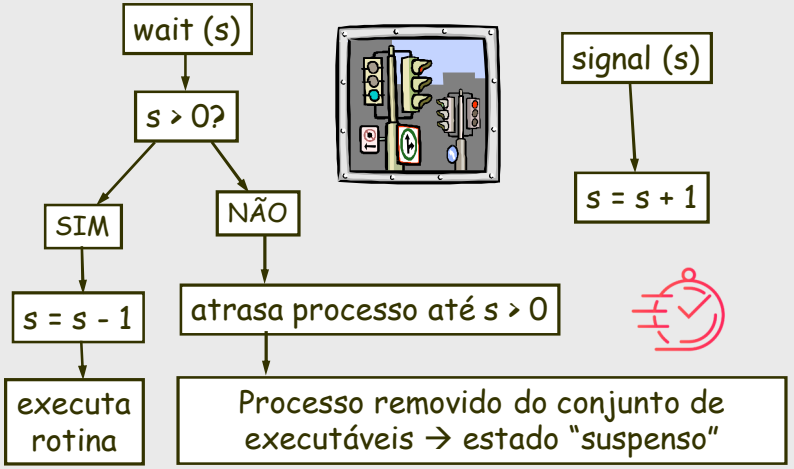
Mensagens ou Mail Boxes

Assíncrono: gerador da mensagem não espera confirmação da recepção

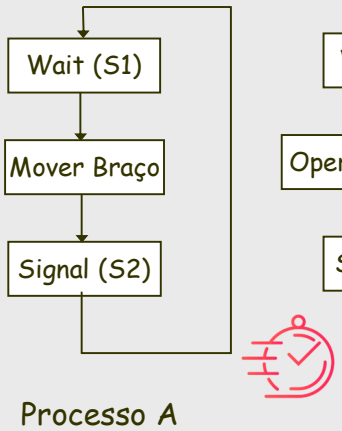
Síncrono: gerador da mensagem espera confirmação da recepção

Semáforos

- Variável inteira não negativa: só alterável por dois procedimentos - wait e signal



Semáforos

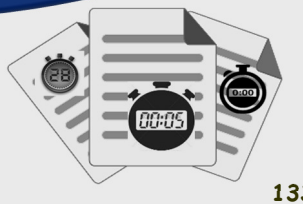


Semáforos

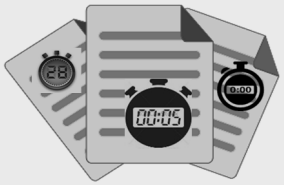
- Exclusão Mútua

```
Processo P1                    Processo P2
  loop                           loop
    wait (s)                       wait (s)
    <seção crítica>                seção crítica
    signal (s)                     signal (s)
    seção não crítica              seção não crítica
  end                            end
end P1                         end P2
```

Novamente, pode ser utilizada uma combinação de mecanismos de sincronização



Análise de Desempenho

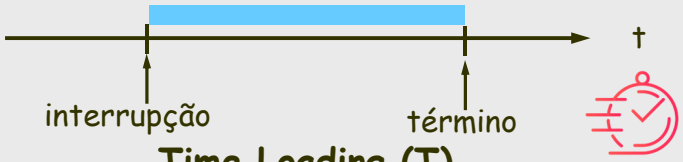


Conceitos

Medição

Análise de Desempenho

Tempo de Resposta



Time Loading (T)

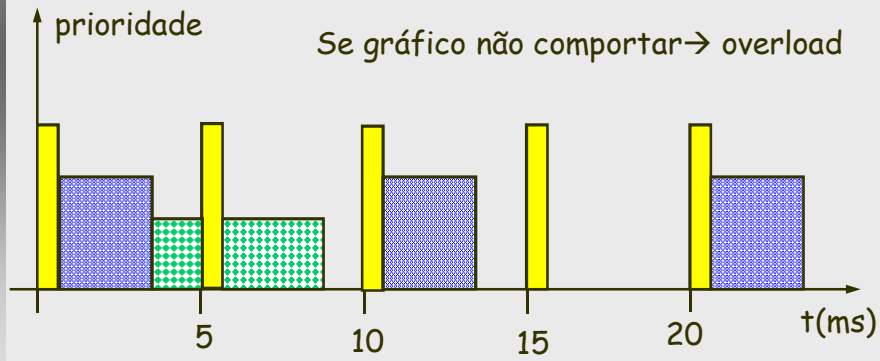
Tempo de Execução

$$T = \sum_{i=1}^{n} \frac{A_i}{T_i}$$

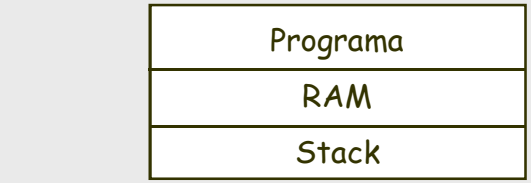
Cycle Time
(tempo mínimo entre ocorrências)

Análise de Desempenho

- Sistema com os seguintes time-loading
  - 16,5% para 5 ms → 0,8ms
  - 30 % para 10 ms → 3,0 ms
  - 13 % para 40 ms → 5,2 ms



Análise de Desempenho



Memory Load Programa

$M_T = M_P.P_P + M_R.P_R + M_S.P_S$

Memory Load Total

% Memória Alocada ao Programa

Análise de Desempenho

- Cada Mi deve ser menor do que 100%
- Exemplo:
  - Memória de 100MB
    - 64MB Programa – 75% load
    - 24MB RAM – 25% load
    - 12MB Stack – 50% load

$$M_T = 0{,}75.\frac{64}{100} + 0{,}25.\frac{24}{100} + 0{,}5.\frac{12}{100} = 60\%$$

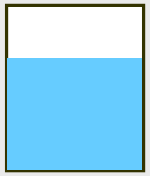
$$M_P = \frac{U_P}{T_T}$$

U_P: Número de locações usadas na área de programa

T_T: Número de locações total na memória

Análise de Desempenho

Memória



Memory Loading

% ocupada

Análise de Desempenho

- Reduzindo o Memory Loading

  - Variáveis locais → Stack
  - Variáveis globais → RAM
  - Buscar melhor distribuição das áreas
  - Reutilizar variáveis globais
- Exemplo
  - discrim := b*b – 4*a*c e
    raiz := (-b + sqrt(discrim)) * 0,5/a

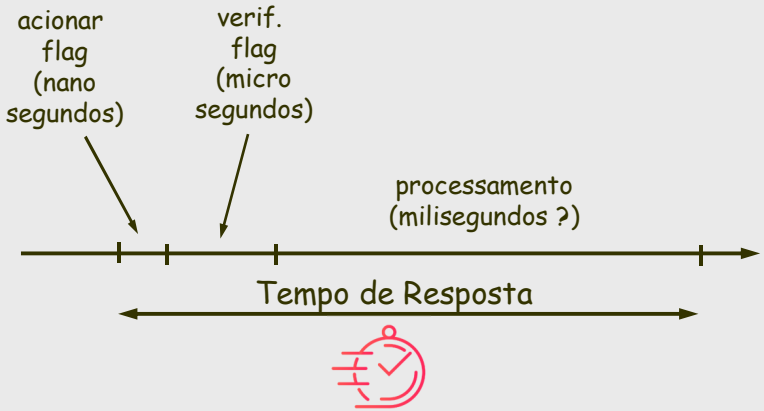
    X
  - raiz := (-b + sqrt(b*b – 4*a*c) * 0,5/a
  - Poupa uma variável pto flutuante → 4 bytes de mem.

Um ótimo desempenho absoluto pode não ser a solução real-time, mas . . .

. . .

Tempo de Resposta

- Polled Loop: 3 componentes

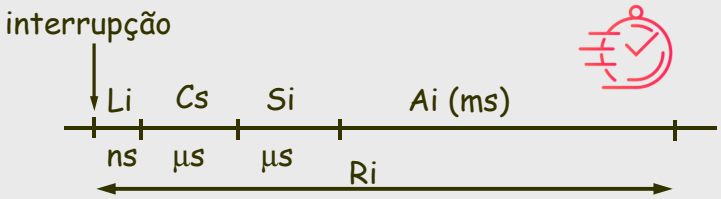


Tempo de Resposta

- Sistemas com Interrupção

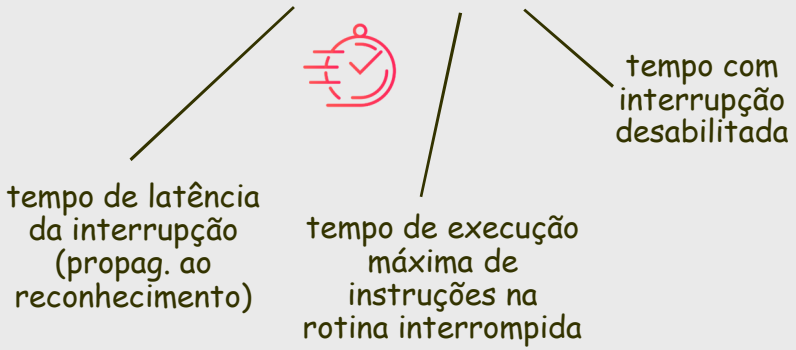
$$Ri = Li + Cs + Si + Ai$$

Ri = tempo para a tarefa i
Li = latência da interrupção
Cs = tempo para salvar contexto
Si = tempo para escalação
Ai = tempo para resposta



Tempo de Resposta

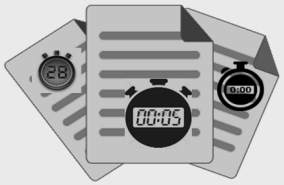
- Determinação de Li
  - Tarefa de maior prioridade interrompe a de menor prioridade: $Li = Lp + \max(L_I, L_D)$



Tempo de Resposta

- Reduzir tempo de resposta
  - Saber como o tempo não útil está sendo consumido
  - Computação realizada na taxa mais baixa possível
- Medição do Time Loading
  - Analisador lógico
  - Contagem de instruções
  - Simuladores
  - Execução próprio programa

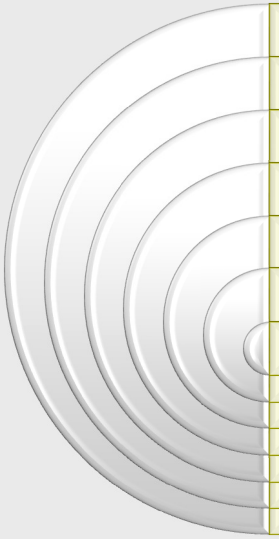
Tempo Real
Testes



Conceitos

Checklist

Testes de SW - Conceituação



- Visam identificar falhas no código do SW
- Não conseguem provar sua correção
- Não ser encarados como pessoais
- Alta probabilidade de detectar erros
- Não serem redundantes
- Validação: o SW corresponde aos requisitos do cliente?
- Verificação: o SW implementa corretamente as funções especificadas?

Testes de SW - Conceituação

- Técnicas de teste
  - Caminhos independentes
  - Partições equivalentes
  - Valores limites
  - Error guessing

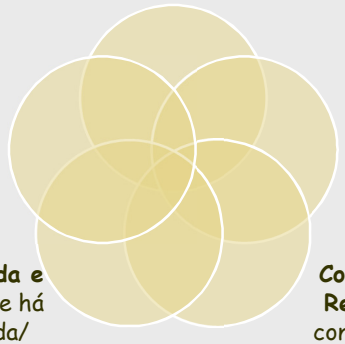
Testes de SW - Conceituação

- Testes de Unidade → Integração (top down / bottom up) → Sistema (funcionalidade, recuperação, segurança, estresse, desempenho)
- Caixa Preta x Caixa Branca
- Checklist
- Walkthrough
- Simulação

Checklist

Retorno de Rotinas: o retorno deve ocorrer sem perdas - continuidade assegurada

Controle de Fluxo de Programas: verificação de estruturas de controle, . . .



Tratamento de Interrupções: procedimento adequado de controle de interrupções

Testes de Entrada e Saída: verificar se há testes de entrada/ saída das rotinas

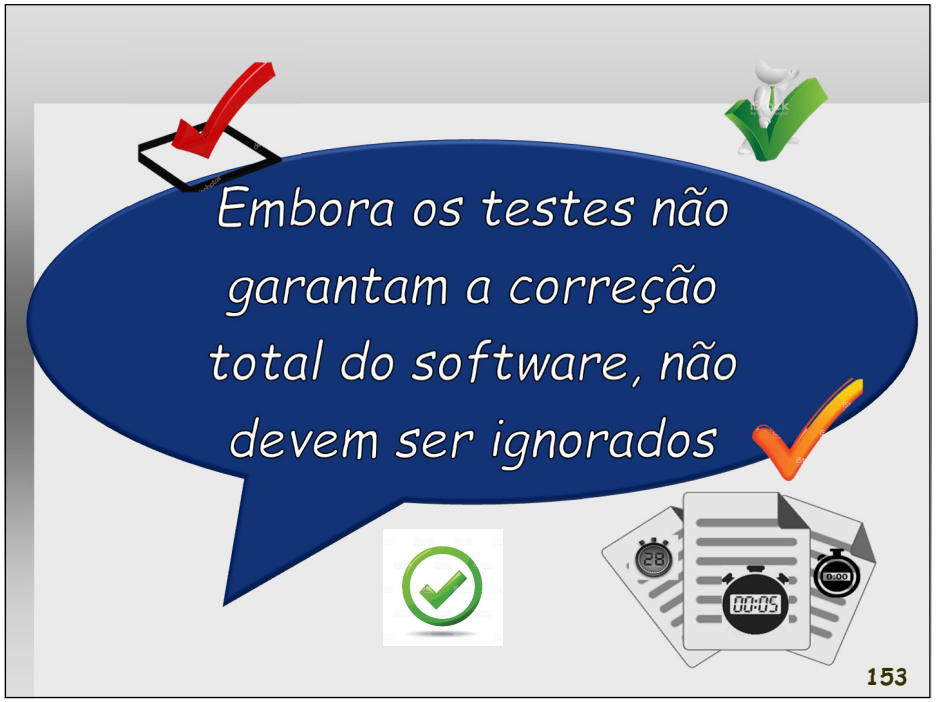
Controle de Laços Repetitivos: laços conseguem atingir a instrução de saída

Checklist

- **Código Fonte não Utilizado**: influência em manutenções
- **Utilização de Variáveis/Constantes**: proteção, definição, coerência de tipos
- **Comentários do Código Fonte**: melhorar a compreensão do código
- **Legibilidade de Código**: influência em manutenções

Walkthrough

- Reuniões formais
- 3 a 5 pessoas
- Papeis: defensor, relator
- Duração de até 2 horas
- . . .


Embora os testes não garantam a correção total do software, não devem ser ignorados
153