Projeto de Sistemas Embarcados Microcontrolados
Departamento de Engenharia Elétrica
Universidade de Brasília

Introdução ao desenvolvimento de firmwares

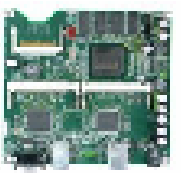
Prof. Geovany A. Borges
gaborges@ene.unb.br

Firmwares de sistemas embarcados

- O firmware é responsável por:
  - Fazer inicialização básica do hardware (e.g., BIOS):
    - Exemplo: Verificação da memória disponível;
    - Exemplo: Configuração padrão E/S;
  - Repassar o controle para um sistema operacional (*bootloader*)
  - Implementar as funcionalidades do sistema embarcado:
    - Firmware estruturado: um único thread e interrupções por hardware.
    - Firmware baseado em kernel: várias tarefas e serviços em execução.

Modalidades de desenvolvimento

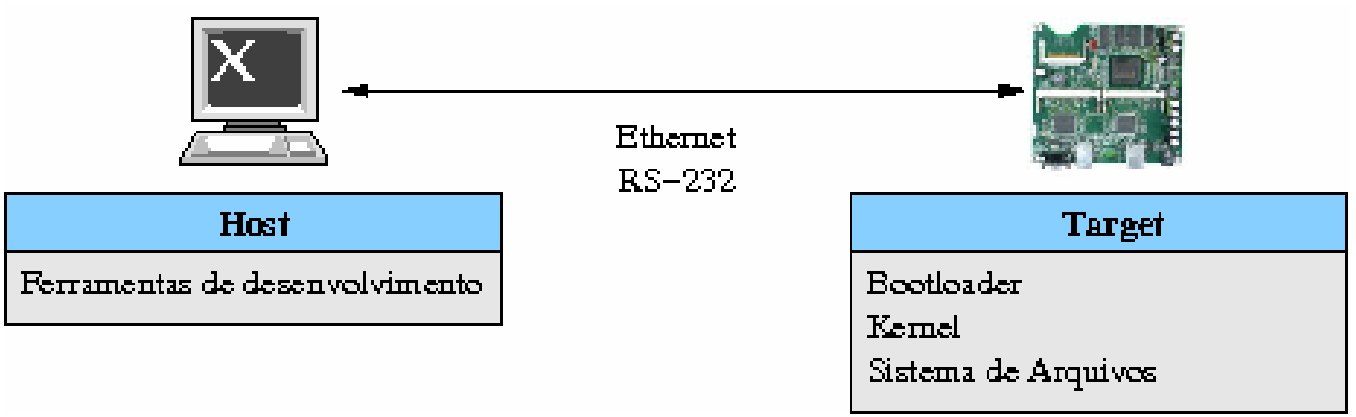
- TARGET: para sistemas capazes de armazenar e executar as ferramentas de desenvolvimento



Target
Bootloader Kernel Sistema de Arquivos Ferramentas de desenvolvimento

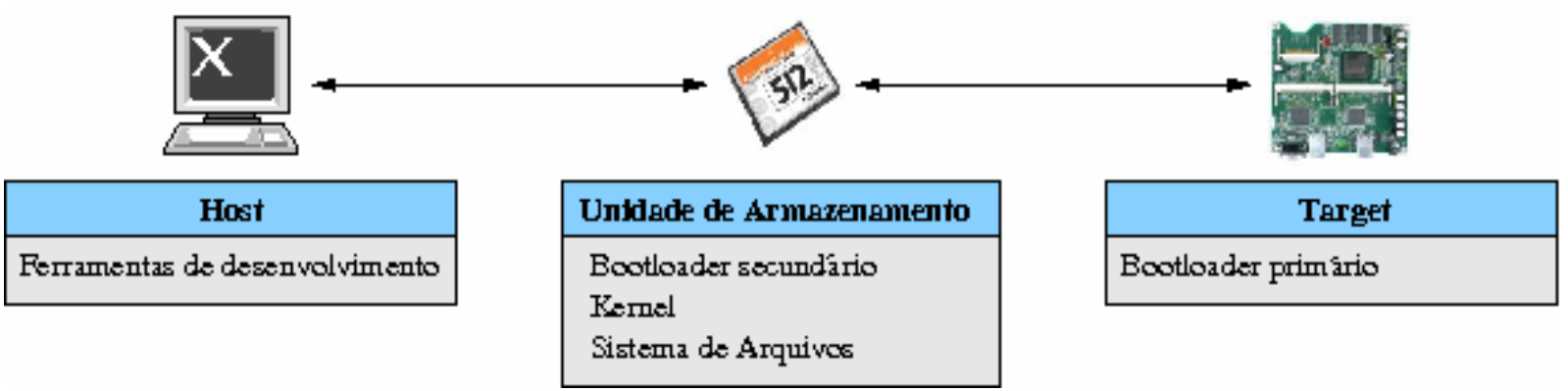
Modalidades de desenvolvimento

- HOST e TARGET: ferramentas de desenvolvimento cruzado em um microcomputador hospedeiro



Modalidades de desenvolvimento

- HOST, TARGET e unidade de armazenamento removível: indicado para estágios iniciais de configuração/otimização do kernel



Ciclo de desenvolvimento

- Requisitos: coleta de informações com o cliente.
  - Funcionalidades, interface com usuário, desempenho, custo, tamanho/peso, consumo...
- Especificação: requisitos colocados na linguagem do projetista.
  - Interfaces E/S, tempos de resposta, comunicação, periféricos, interdependência entre as partes...

Ciclo de desenvolvimento

- Definição da arquitetura de hardware/software: esboço na forma de diagrama de blocos dos componentes de hardware e de software do sistema.
  - Hardware: Processador e interconexão com periféricos
  - Software: Componentes de software, módulos, bibliotecas.
- Escolha das ferramentas de desenvolvimento
  - Hardware: kit de desenvolvimento, emuladores
  - Software: sistema operacional, linguagem de programação, bootloaders, IDE
  - Ferramentas de projeto de esquemáticos/PCB
  - Ferramentas de documentação

Ciclo de desenvolvimento

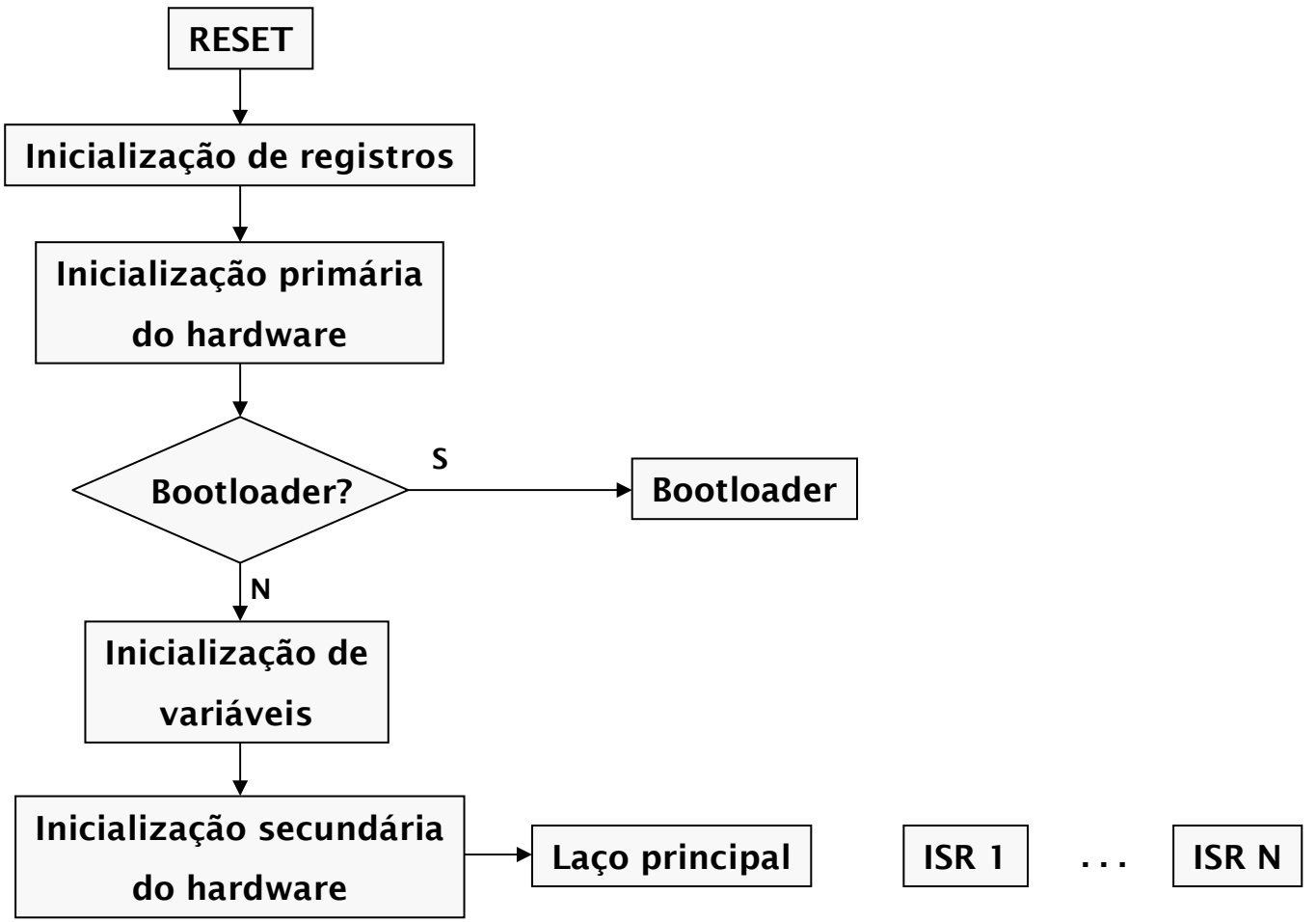
- Implementação: uso de ferramentas de auxílio ao projeto
  - Formalismos: Unified Modeling Language (UML), FSM
  - Implementação de Hardware/Software
- Testes: verificação de adequação do sistema conforme as especificações
- Integração de sistemas: implantação do sistema e avaliação acerca da satisfação dos requisitos

Tópicos desenvolvidos para a disciplina

- Introdução à linguagem C
  - Consultar documento *linguagemC.pdf*
- Desenvolvimento com o µC ATMega8
  - Consultar documento *desenvolvimentoATmega8.pdf*
- Arquiteturas de firmwares
  - Firmware estruturado
  - Firmware baseado em Kernel

Firmware estruturado

- Organização genérica



Firmware estruturado

- Operações atômicas em regiões críticas
  - Região crítica: segmento de código que não pode ser interrompido.
  - Exemplos:
    - Variáveis globais compartilhadas entre diferentes níveis de interrupção
    - Funções não reentrantes
    - Registros de E/S que possuem validade temporal (e.g, contadores de temporizadores)
    - Gerenciamento por softawre de handshake
    - Acesso a uma seqüência de registros de um dispositivo de hardware
    - Conversão A/D “simultânea” de dois ou mais canais sem S-H externo
  - Solução para firmwares estruturados: desabilitar globalmente a ocorrência de interrupções.

Firmware estruturado

- Operações atômicas em regiões críticas
  - Exemplo: interrupções iniciando com bit I desabilitado

```
volatile unsigned int Counter1msTicks = 0;
volatile unsigned char DAOutput;

SIGNAL (SIG_OUTPUT_COMPARE1A)
{
   // Decrementa Counter1msTicks a cada interrupção se diferente de 0.
   if(Counter1msTicks!=0){
     --Counter1msTicks;
   }
}

SIGNAL (SIG_2WIRE_SERIAL)
{
   do{
     send_twi_digital_to_analogic(DAOutput);
     delay(5);
   }
   while(!receive_twi_analogic_to_digital_device());
}
```

Vetor 7: maior prioridade

Vetor 18: menor prioridade

Firmware estruturado

- Operações atômicas em regiões críticas
  - Exemplo: interrupções iniciando com bit I desabilitado

```
void delay(unsigned int time_ms) // limitado em até 65535 ms.
{
   unsigned int counter = 1;

   ENTER_CRITICAL();
   Counter1msTicks = time_ms;
   LEAVE_CRITICAL();
   while(counter>0){
     sleep_mode();
     ENTER_CRITICAL();
     counter = Counter1msTicks;
     LEAVE_CRITICAL();
   }
}
```

Firmware estruturado

- Operações atômicas em regiões críticas
  - Exemplo: soluções para as macros de acesso
    - Solução 1

```
#define ENTER_CRITICAL() cli()
#define LEAVE_CRITICAL() sei()
```

    - Solução 2

```
#define LOCAL_CRITICAL_CONTROL() volatile char cSREG
#define ENTER_CRITICAL() cSREG = SREG; cli()
#define LEAVE_CRITICAL() SREG = cSREG

void delay(unsigned int time_ms)
{
   unsigned int counter = 1;
   LOCAL_CRITICAL_CONTROL();

   ENTER_CRITICAL();
```

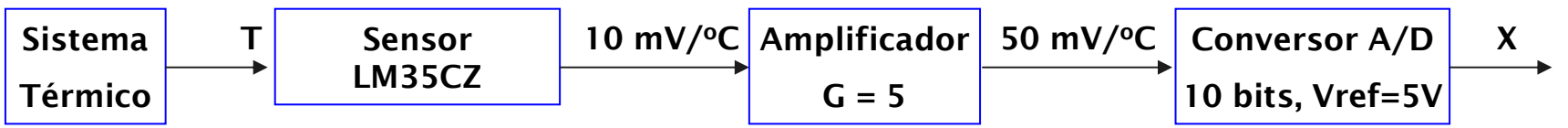
.

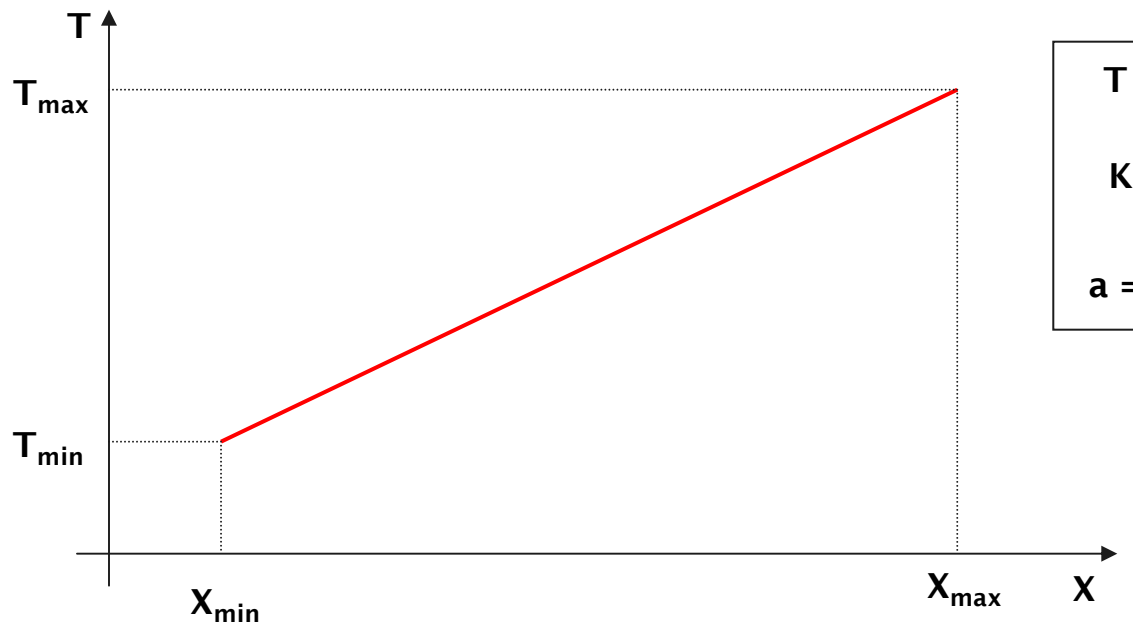
.

.

Firmware estruturado

- Quantização e overflow em operações
  - Considera-se o problema da obtenção da grandeza medida pela leitura de um conversor A/D





$$T = K \cdot X + a$$

$$K = \frac{(T_{max} - T_{min})}{(X_{max} - X_{min})}$$

$$a = T_{max} - K \cdot X_{max} = T_{min} - K \cdot X_{min}$$

$$T = K \cdot (X - X_{min}) + T_{min}$$

Firmware estruturado

- Quantização e overflow em operações
  - Considera-se o problema da obtenção da grandeza medida pela leitura de um conversor A/D (ponto flutuante)
    - Solução 1 : $T = K \cdot (X - X_{min}) + T_{min}$

```
#define Xmax 1023
#define Xmin 0
#define Tmax 99.90234375  /* Temperatura para Xmax */
#define Tmin 0.0           /* Temperatura para Xmin */

float convert_temperature(unsigned int X)
{
  return (Tmax - Tmin)*(X - Xmin)/(Xmax - Xmin) + Tmin;
}
```

Problema de quantização

Firmware estruturado

- Quantização e overflow em operações
  - Considera-se o problema da obtenção da grandeza medida pela leitura de um conversor A/D (ponto flutuante)
    - Solução 2 : $\mathbf{T = K \cdot (X - X_{min}) + T_{min}}$

```
#define Xmax 1023
#define Xmin 0
#define Tmax 99.90234375   /* Temperatura para Xmax */
#define Tmin 0.0           /* Temperatura para Xmin */

float convert_temperature(unsigned int X)
{
  return ((Tmax - Tmin)*(X - Xmin))/(Xmax - Xmin) + Tmin;
}
```

Firmware estruturado

- Quantização e overflow em operações
  - Considera-se o problema da obtenção da grandeza medida pela leitura de um conversor A/D (ponto fixo)
    - Solução 1 : $T = K \cdot (X - X_{min}) + T_{min}$

```
#define Xmax 1023
#define Xmin 0
#define Tmax 100 /* Temperatura aproximada para Xmax */
#define Tmin 0   /* Temperatura aproximada para Xmin */

unsigned int convert_temperature(unsigned int X)
{
  return ((Tmax - Tmin)*(X - Xmin))/(Xmax - Xmin) + Tmin;
}
```

Possível problema de overflow

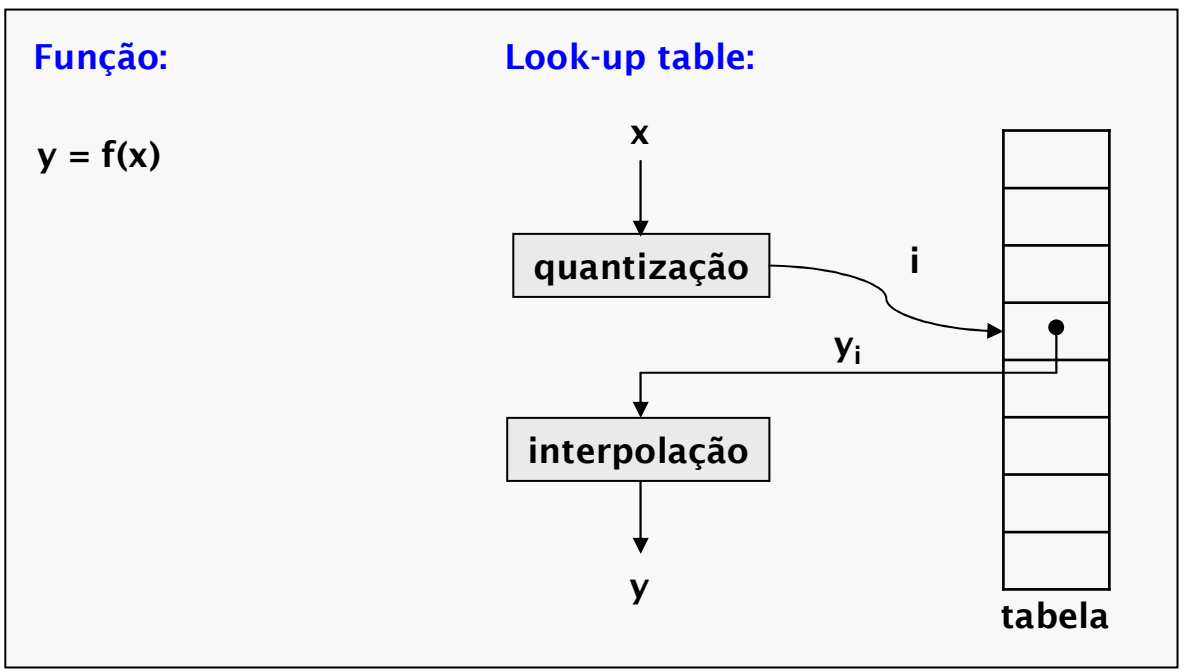
Firmware estruturado

- Quantização e overflow em operações
  - Considera-se o problema da obtenção da grandeza medida pela leitura de um conversor A/D (ponto fixo)
    - Solução 2 : $T = K \cdot (X - X_{min}) + T_{min}$

```
#define Xmax 1023
#define Xmin 0
#define Tmax 100 /* Temperatura aproximada para Xmax */
#define Tmin 0   /* Temperatura aproximada para Xmin */

unsigned int convert_temperature(unsigned int X)
{
  unsigned long int aux;

  aux = (X - Xmin);
  aux = ((Tmax - Tmin)*aux); aux /=(Xmax - Xmin);
  return ((unsigned int)(aux)) + Tmin;
}
```

Firmware estruturado

- Look-up tables (LUTs)
  - Em algumas situações, em vez lugar de avaliar uma função em tempo de execução, é preferível armazenar resultados indexados em uma tabela e acessá-los quando necessário.



Firmware estruturado

- Look-up tables (LUTs)
  - Situações para as quais usa-se LUTs:
    - Cálculo trigonométrico com inteiros (e.g., geração de uma onda senoidal através de conversor D/A);
    - Leitura A/D de variáveis de característica não-linear (e.g., leitura de termopares, RDTs);
    - Máquina de inferência Fuzzy (e.g, controle Fuzzy);
    - Conversão de código Gray para decimal (e.g., leitura de codificadores ópticos absolutos);
    - Leitura de interfaces de entrada (e.g., conversão de scancodes de teclados em ASCII ou UNICODE).
  - Exercício: geração de sinal PWM para acionamento de motores monofásicos

Firmware estruturado

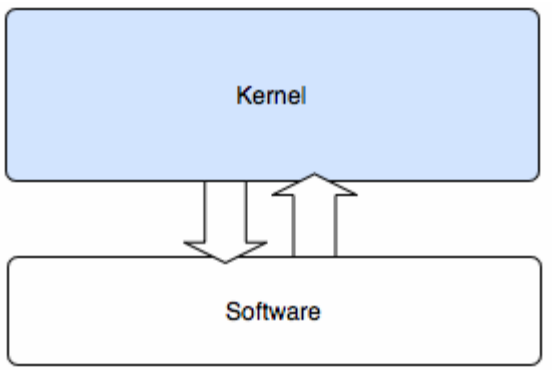
- Tópicos não abordados, mas importantes:
  - Manutenção do tempo
  - Medição de período/freqüência
  - Leitura de dados por conversão A/D
  - Máquinas de estado
  - FIFOs
  - Listas encadeadas

Firmware baseado em *Kernel*

- O que vem a ser um kernel ?
- Funcionalidades mínimas providas por um kernel:
  - Gerenciamento de memória
  - Gerenciamento de processos e comunicação entre processos (IPC)
  - Recursos de E/S
- Níveis de abstração (processadores avançados):
  - Espaço usuário e espaço kernel
  - Abstração dos dispositivos de E/S (UNIX e Linux)
  - Abstração da camada de hardware: reduz dependência do hardware, tais como unidade de gerenciamento de memória, controladores de interrupção, temporizador, DMA, controlador de tarefas, ...

Firmware baseado em *Kernel*

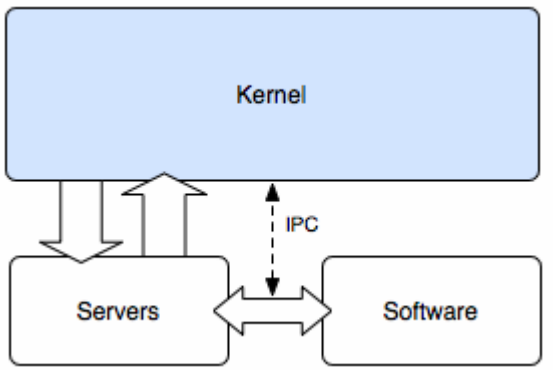
- Classificação de kernels: (wikipedia)
  - Kernel monolítico:
    - Todos os serviços estão no kernel



    - Aspectos:
      - (-) menor estabilidade: a falha de um serviço derruba o sistema;
      - (+) se bem feito, alcança elevado nível de desempenho;
    - Exemplos: Linux, Windows 95-98-Me

Firmware baseado em *Kernel*

- Classificação de kernels: (wikipedia)
  - Microkernel:
    - Serviços básicos:
      - gerenciamento de memória
      - processos e IPC
    - Extensão por meio de servidores
    - Aspectos:
      - (+) maior estabilidade: a falha de um serviço não derruba o sistema;
      - (+) menor footprint e fácil de manter;
      - (-) em geral, se sobrecarregam com mais facilidade;
    - Exemplos: QNX, Minix, AmigaOS

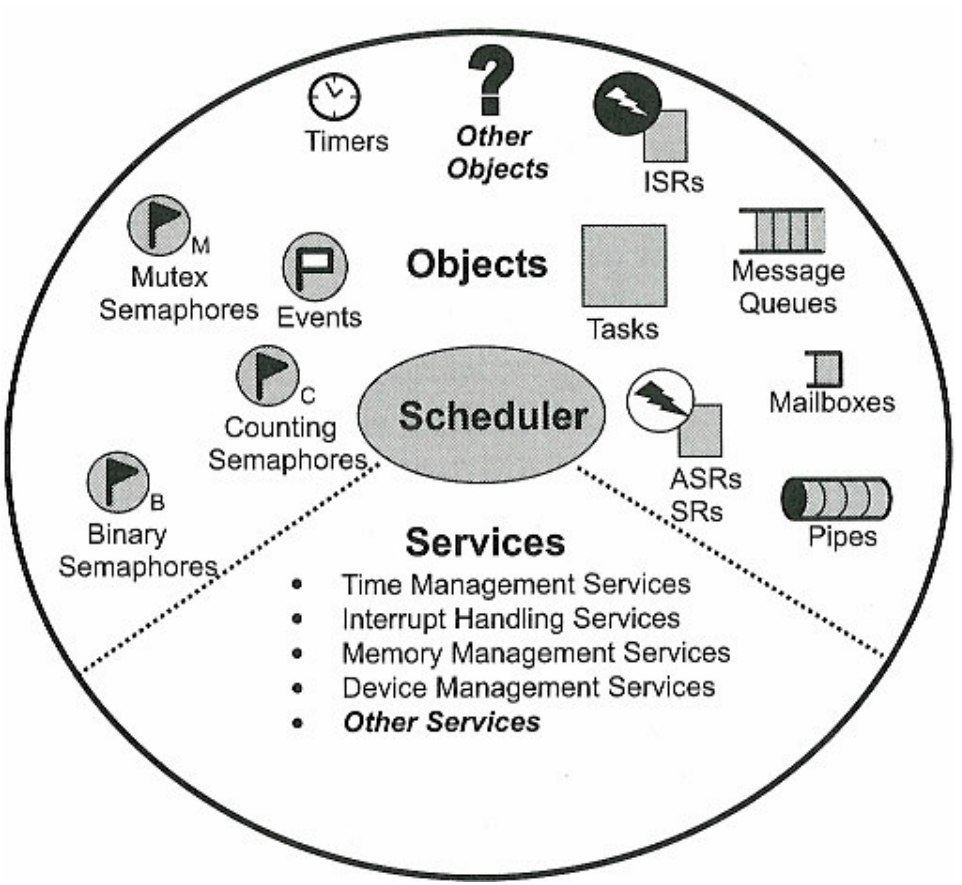


Firmware baseado em *Kernel*

- Classificação de kernels: (wikipedia)
  - Nanokernels:
    - Controlam o fluxo de interrupções de hardware, redirecionando-as para outros sofwares.
    - Usados como forma de virtualização do hardware, permitindo que vários sistemas operacionais co-existam ao mesmo tempo
- Limitações de uso de *kernels* clássicos para sistemas embarcados:
  - Tamanho de memória
  - Não-determinismo de algumas funções
  - Alto jitter para comutação de tarefas
  - Escalonadores buscam maximizar o número de tarefas por período de execução

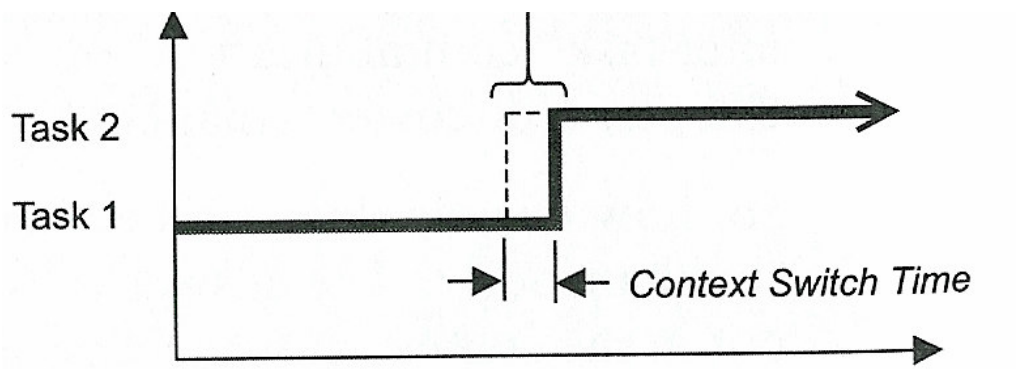
Firmware baseado em *Kernel*

- Sistemas operacionais de tempo real (RTOS)



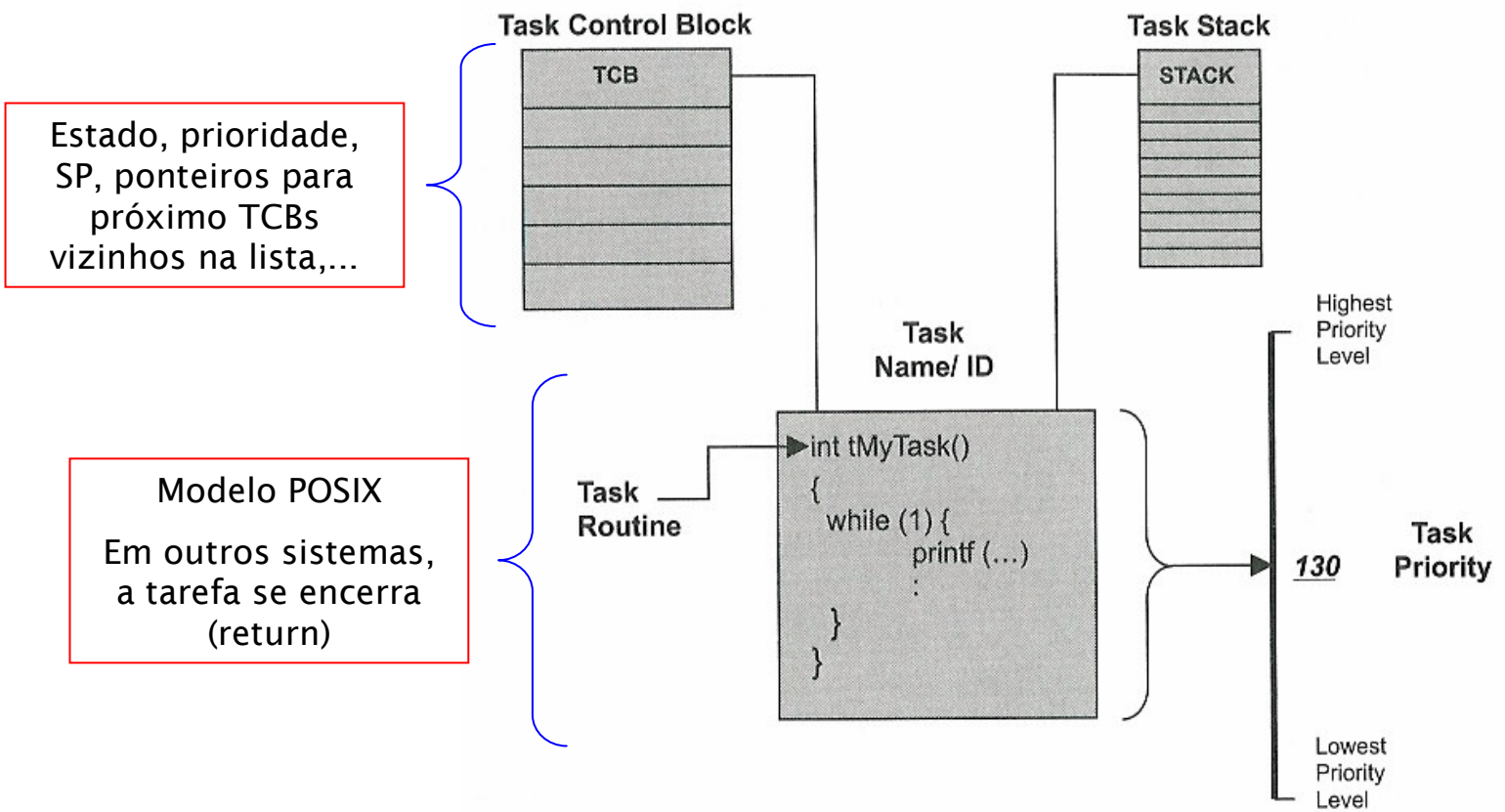
Firmware baseado em *Kernel*

- Sistemas operacionais de tempo real (RTOS)
  - Escalonador
    - Associado a uma interrupção de temporizador de alta prioridade e periodicidade constante.
    - Responsável por:
      - Atualização dos estados das tarefas
      - Aplicação da política de escalonamento
      - Execução das tarefas
      - Chaveamento de contexto



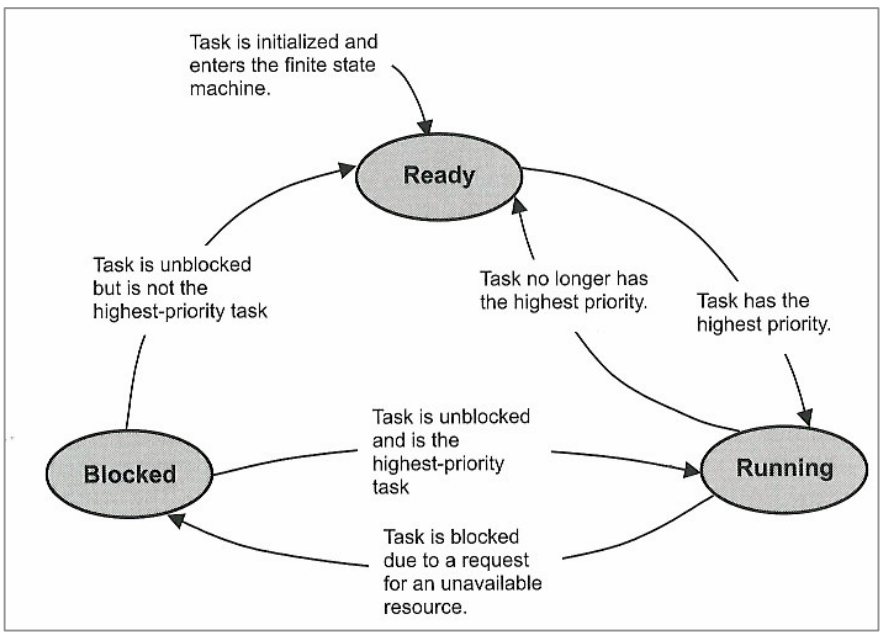
Firmware baseado em *Kernel*

- Sistemas operacionais de tempo real (RTOS)
  - Tarefas: modelo e estruturas associadas

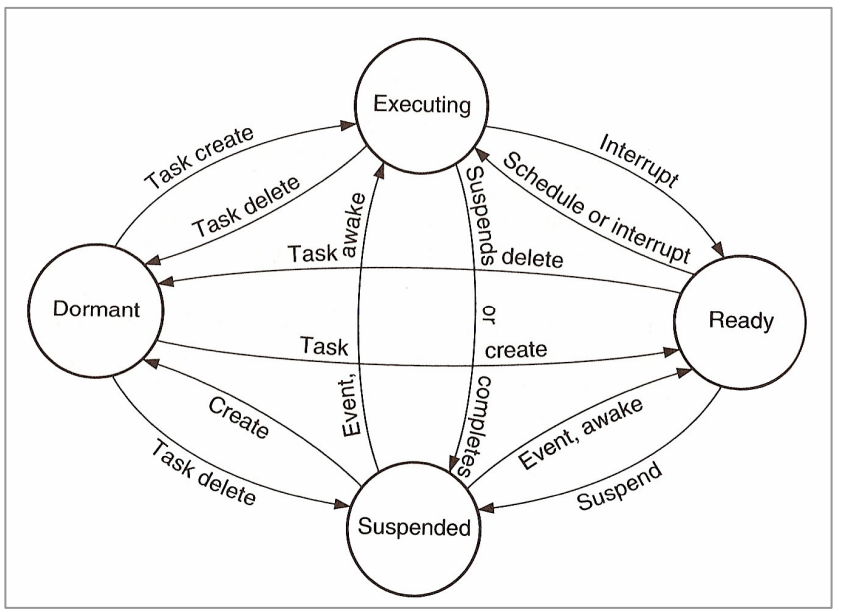


Firmware baseado em *Kernel*

- Sistemas operacionais de tempo real (RTOS)
  - Tarefas: estados



[Li & Yao, 2003]



[Laplante, 1993]

Pode ser diferente conforme o sistema e o número de processadores

Firmware baseado em *Kernel*

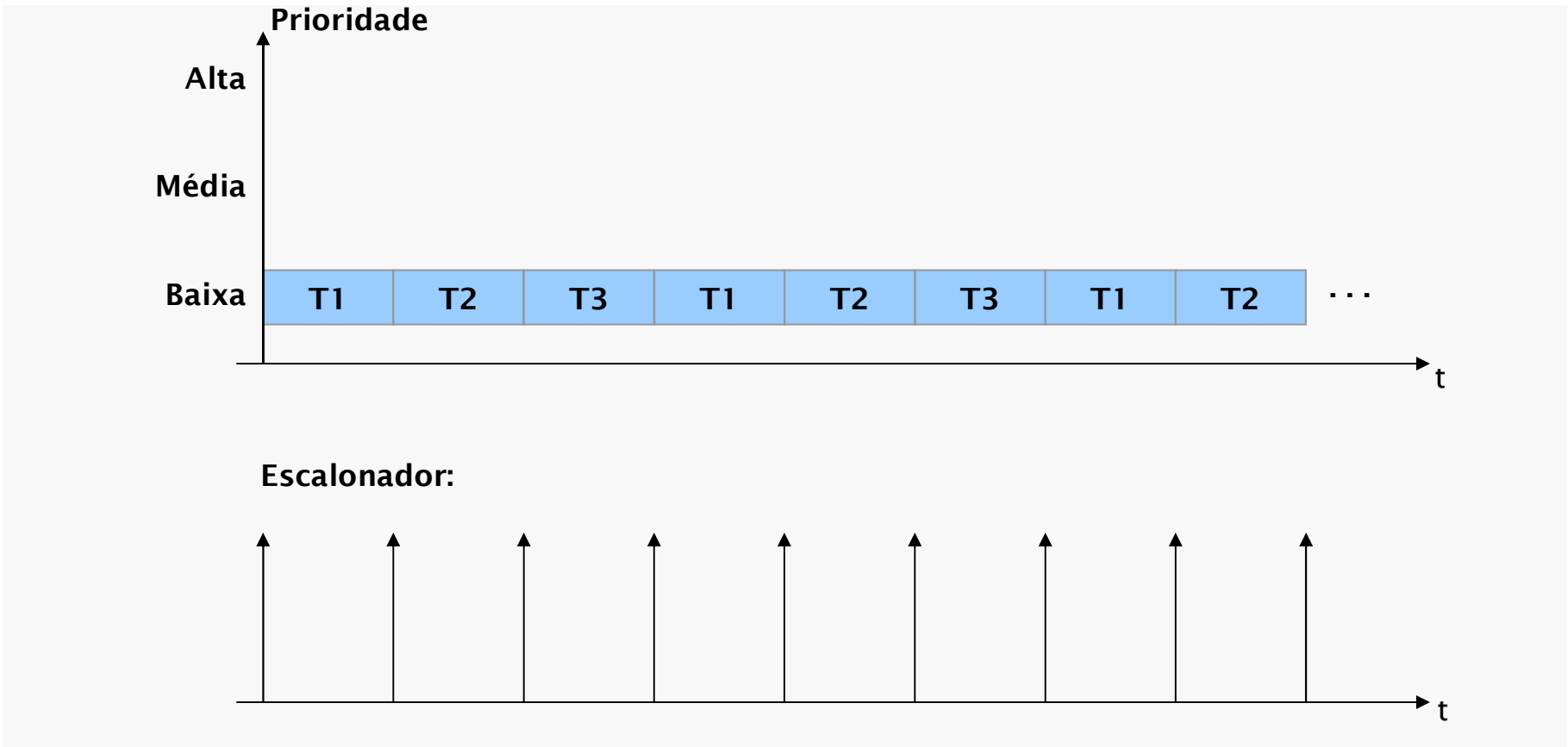
- Sistemas operacionais de tempo real (RTOS)
  - Tarefas: exemplo

```
void tarefa1(void)
{
   for(;;){
     printf("Tarefa 1");
   }
}
```

```
void tarefa2(void)
{
   for(;;){
     printf("Tarefa 2");
   }
}
```

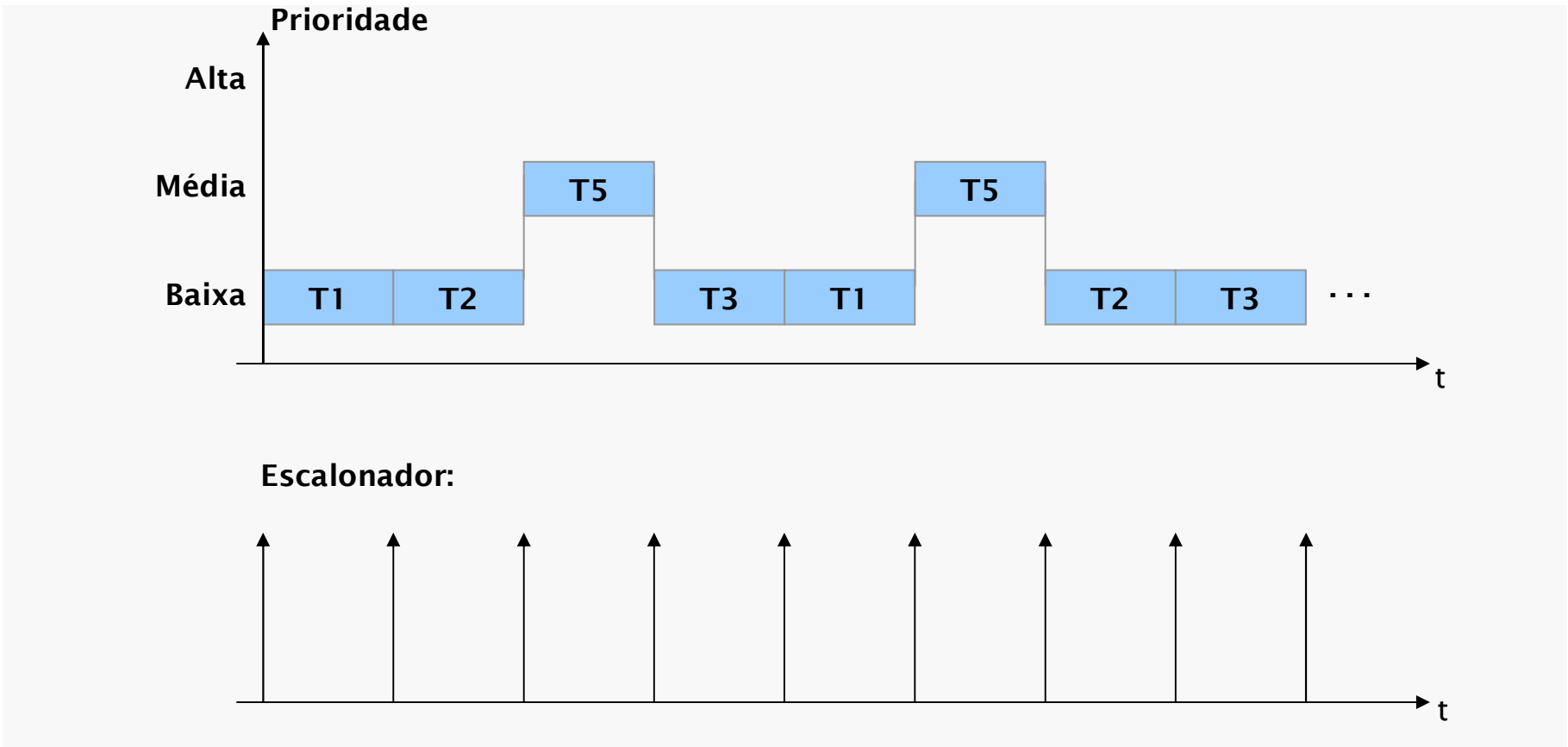
Firmware baseado em *Kernel*

- Sistemas operacionais de tempo real (RTOS)
  - Políticas de escalonamento: round-robin



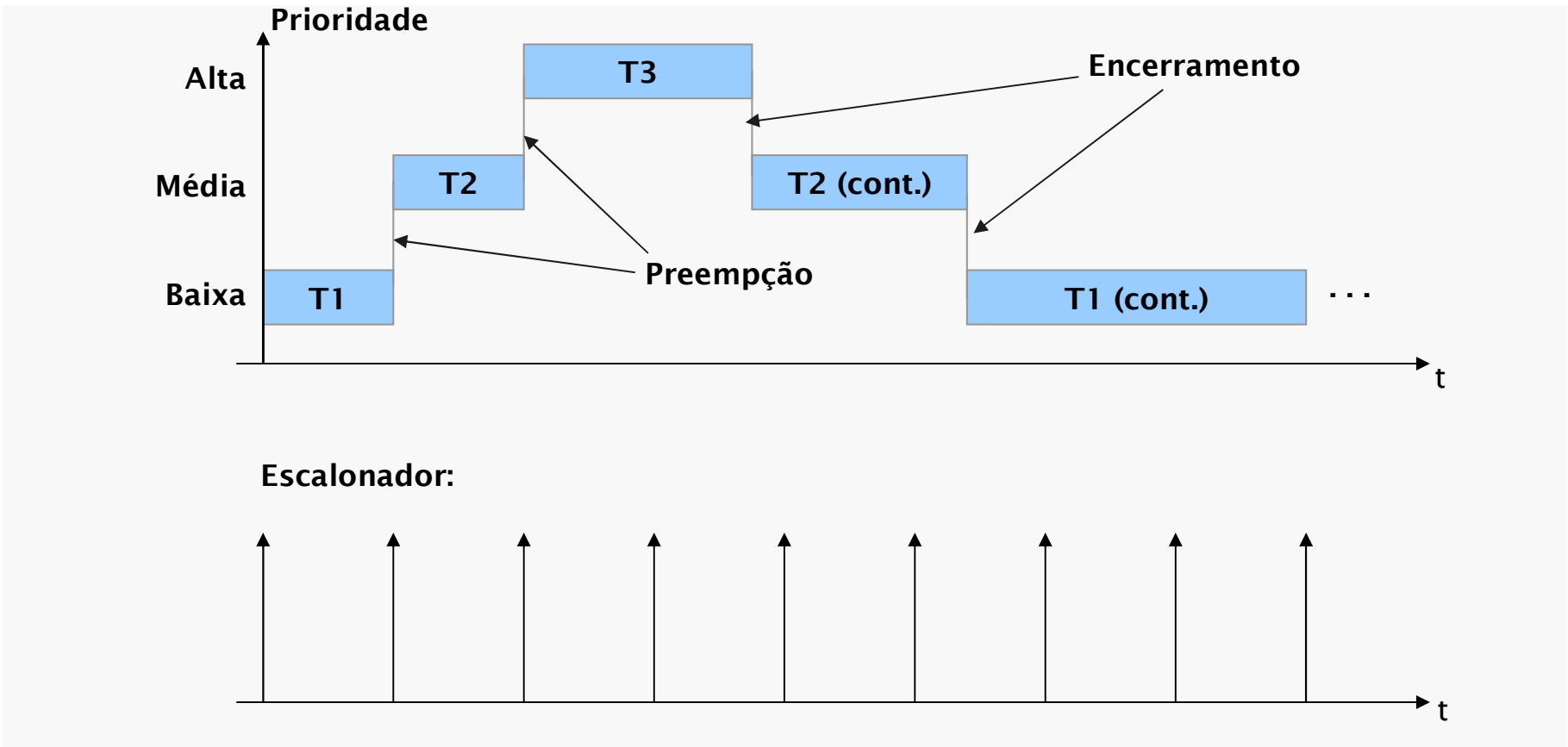
Firmware baseado em *Kernel*

- Sistemas operacionais de tempo real (RTOS)
  - Políticas de escalonamento: round-robin com prioridade



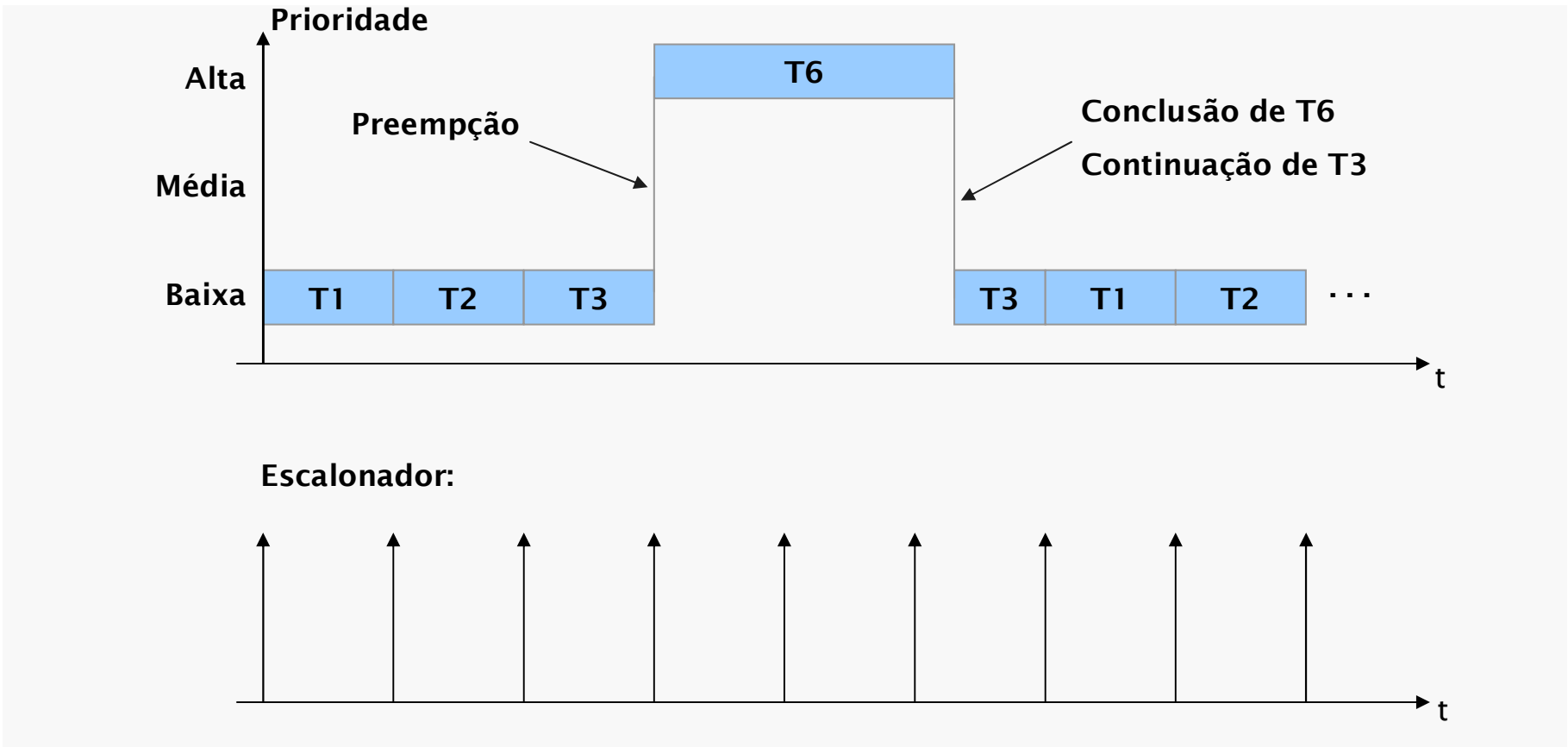
Firmware baseado em *Kernel*

- Sistemas operacionais de tempo real (RTOS)
  - Políticas de escalonamento: preemptivo



Firmware baseado em *Kernel*

- Sistemas operacionais de tempo real (RTOS)
  - Políticas de escalonamento: round-robin e preemptivo



Firmware baseado em *Kernel*

- Sistemas operacionais de tempo real (RTOS)
  - Políticas de escalonamento: exemplo (revisitado)

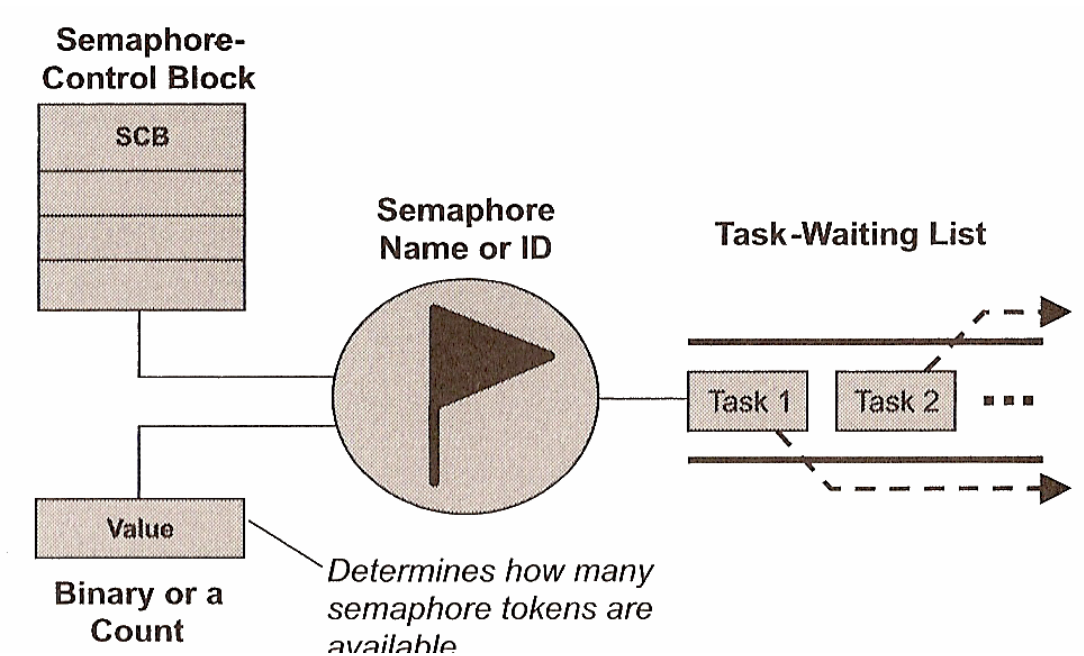
```
void tarefa1(void)
{
   for(;;){
     printf("Tarefa 1");
   }
}
```

```
void tarefa2(void)
{
   for(;;){
     printf("Tarefa 2");
   }
}
```

    - Que padrões de saída espera-se ter quando...
      - (a) escalonamento round-robin
      - (b) escalonamento round-robin com prioridade (P2 > P1)

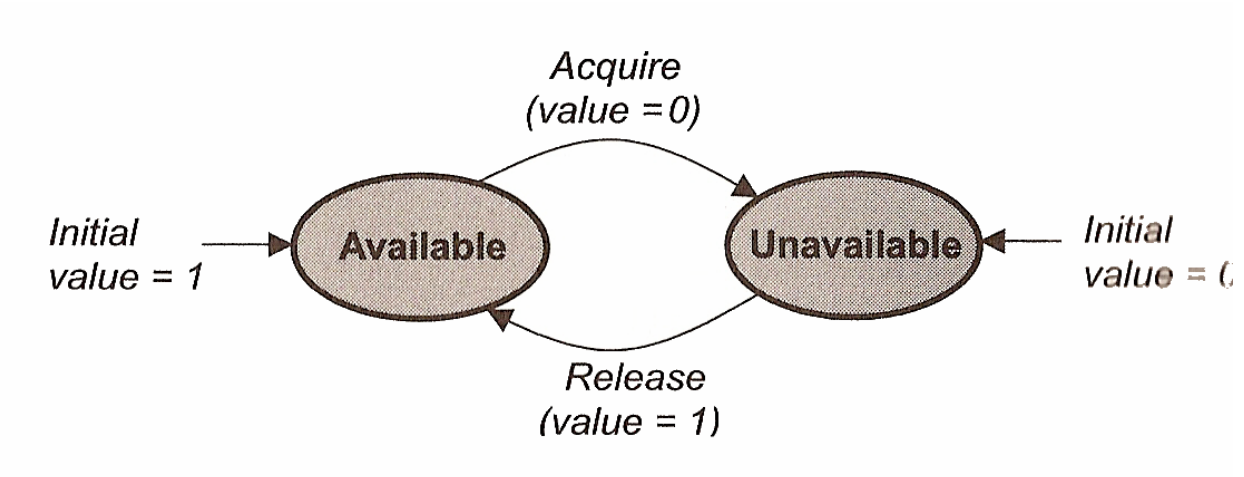
Firmware baseado em *Kernel*

- Sistemas operacionais de tempo real (RTOS)
  - Semáforos
    - Objetos que permitem lidar com o acesso a recursos compartilhados
    - Estrutura de um semáforo:



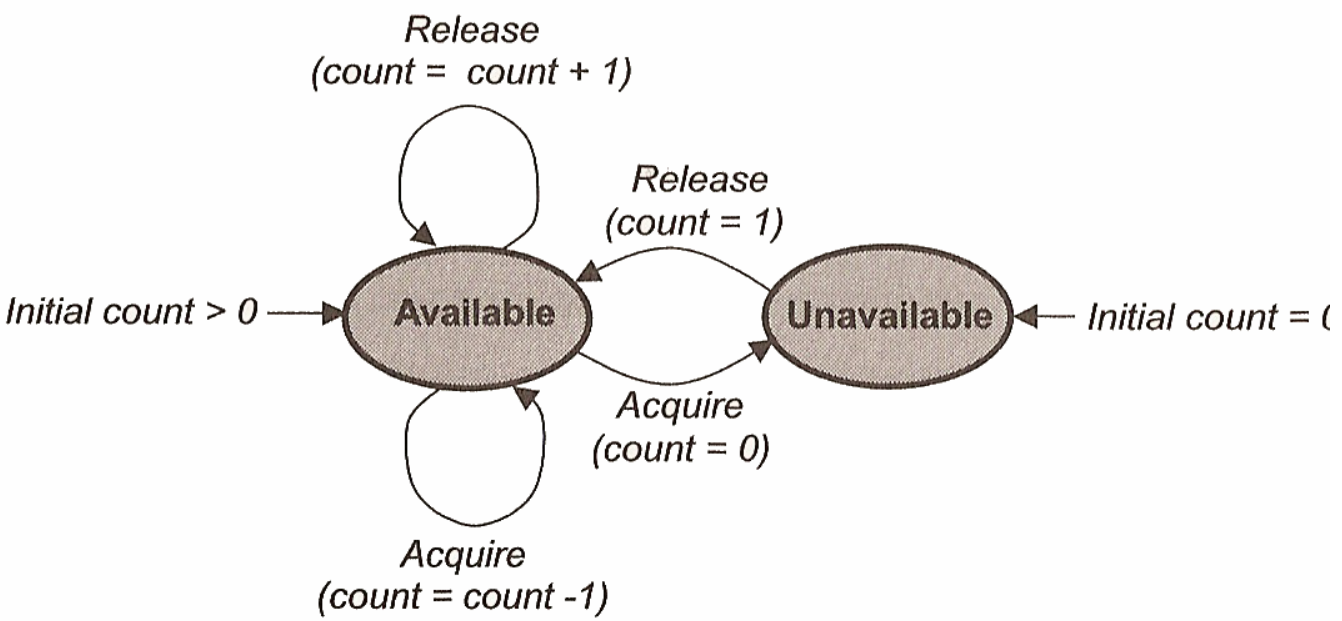
Firmware baseado em *Kernel*

- Sistemas operacionais de tempo real (RTOS)
  - Semáforos
    - Semáforo binário:



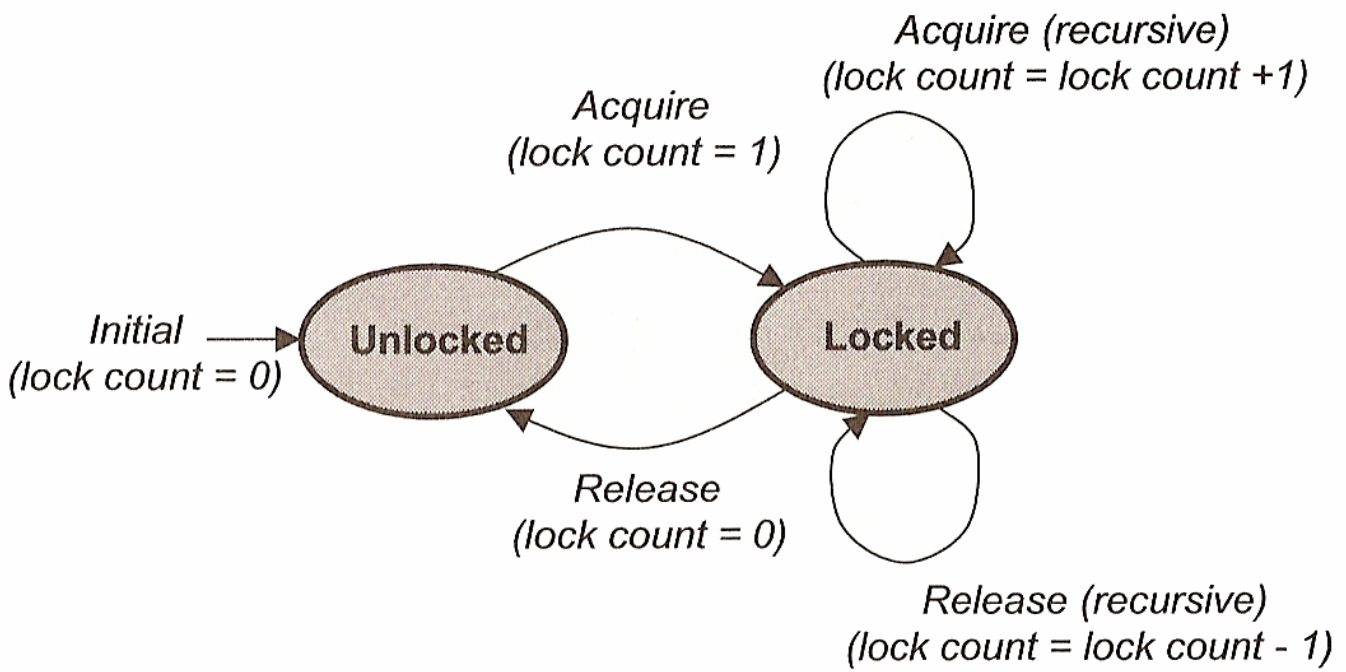
Firmware baseado em *Kernel*

- Sistemas operacionais de tempo real (RTOS)
  - Semáforos
    - Semáforo com contador:



Firmware baseado em *Kernel*

- Sistemas operacionais de tempo real (RTOS)
  - Semáforos
    - Mutex:



Firmware baseado em *Kernel*

- Sistemas operacionais de tempo real (RTOS)
  - Semáforos
    - Semáforos: implementações com bloqueio [Laplante, 1993]

```
_inline void Wait(unsigned char Status)
{
   while(Status==TRUE);
   Status = TRUE;
}

_inline void Signal(unsigned char Status)
{
   Status = FALSE;
}
```

Firmware baseado em *Kernel*

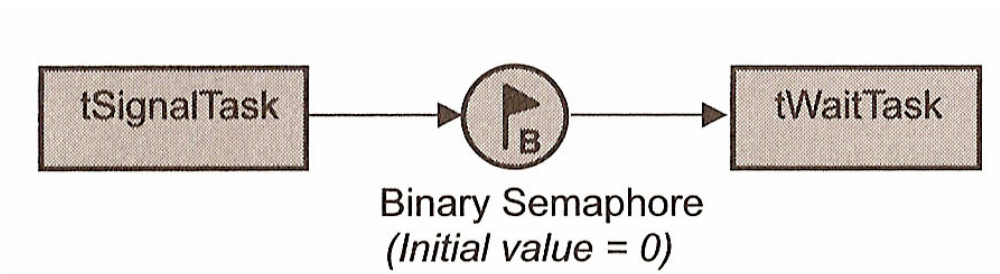
- Sistemas operacionais de tempo real (RTOS)
  - Semáforos
    - Semáforos: implementações sem bloqueio [Stallings, 2001]

```
type semaphore = record
                   count: integer;
                   queue: list of process
                 end;

wait(s):
    s.count := s.count − 1;
    if s.count < 0
      then begin
        place this process in s.queue;
        block this process
      end;

signal(s):
    s.count := s.count + 1;
    if s.count ≤ 0
      then begin
        remove a process P from s.queue;
        place process P on ready list
      end;
```

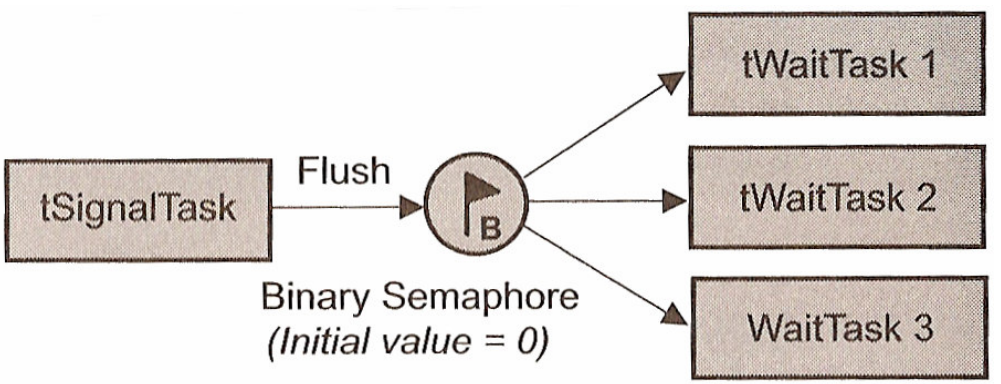
```
type binary semaphore = record
                          value: (0,1);
                          queue: list of process
                        end;

waitB(s):
    if s.value = 1
      then
        s.value = 0
      else begin
        place this process in s.queue;
        block this process
      end;

signalB(s):
    if s.queue is empty
      then
        s.value := 1
      else begin
        remove a process P from s.queue;
        place process P on ready list
      end;
```

Firmware baseado em *Kernel*

- Sistemas operacionais de tempo real (RTOS)
  - Semáforos
    - Emprego de semáforos
      - Sincronismo espera e sinaliza:

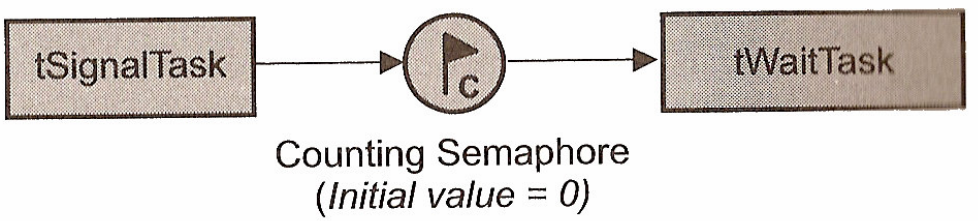


      - Sincronismo espera e sinaliza de múltiplas tarefas:

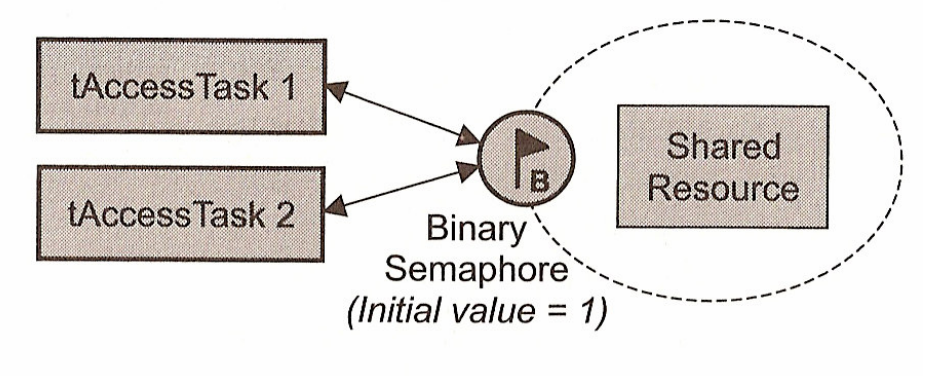


Firmware baseado em *Kernel*

- Sistemas operacionais de tempo real (RTOS)
  - Semáforos
    - Emprego de semáforos
      - Sincronismo com contagem de sinalização:

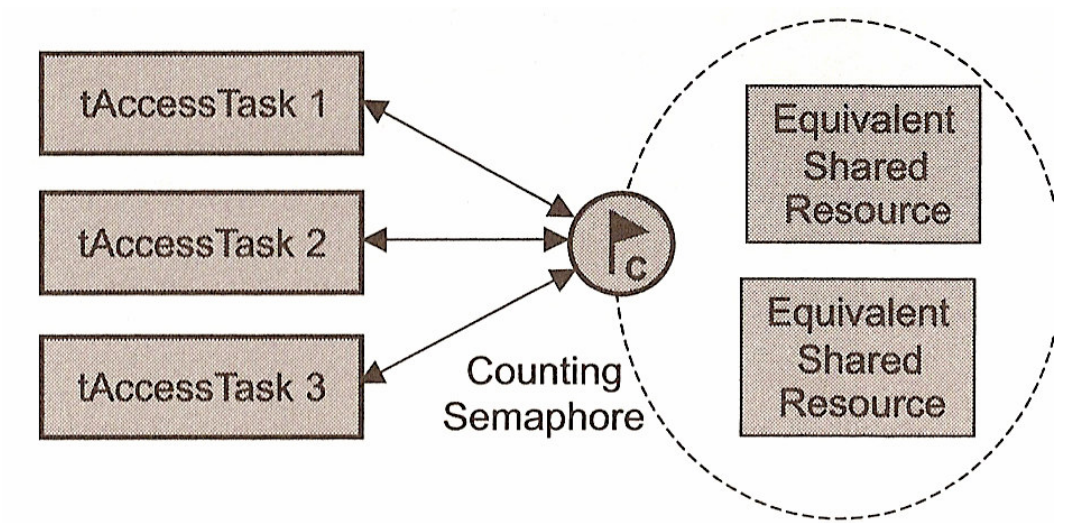


      - Acesso a recurso compartilhado:



Firmware baseado em *Kernel*

- Sistemas operacionais de tempo real (RTOS)
  - Semáforos
    - Emprego de semáforos
      - Múltiplo acesso a recurso compartilhado:



Firmware baseado em *Kernel*

- Sistemas operacionais de tempo real (RTOS)
  - Semáforos
    - Exemplo: recurso stdout compartilhado

```
void tarefa1(void)
{
   for(;;){
     printf("Tarefa 1");
   }
}
```

```
void tarefa2(void)
{
   for(;;){
     printf("Tarefa 2");
   }
}
```

Sem semáforo binário

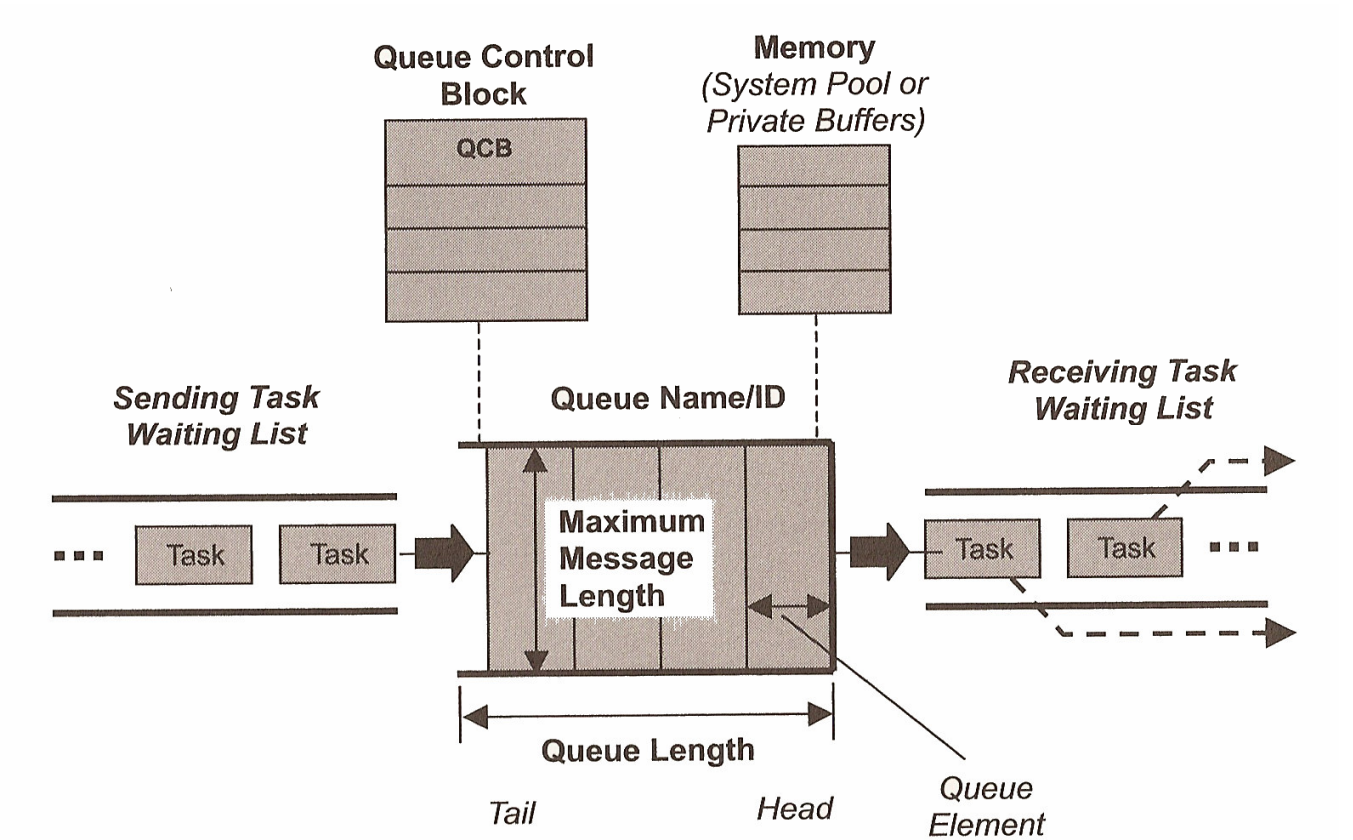
```
void tarefa1(void)
{
   for(;;){
     wait(S);
     printf("Tarefa 1");
     signal(S);
   }
}
```

```
void tarefa2(void)
{
   for(;;){
     wait(S);
     printf("Tarefa 2");
     signal(S);
   }
}
```

Com semáforo binário

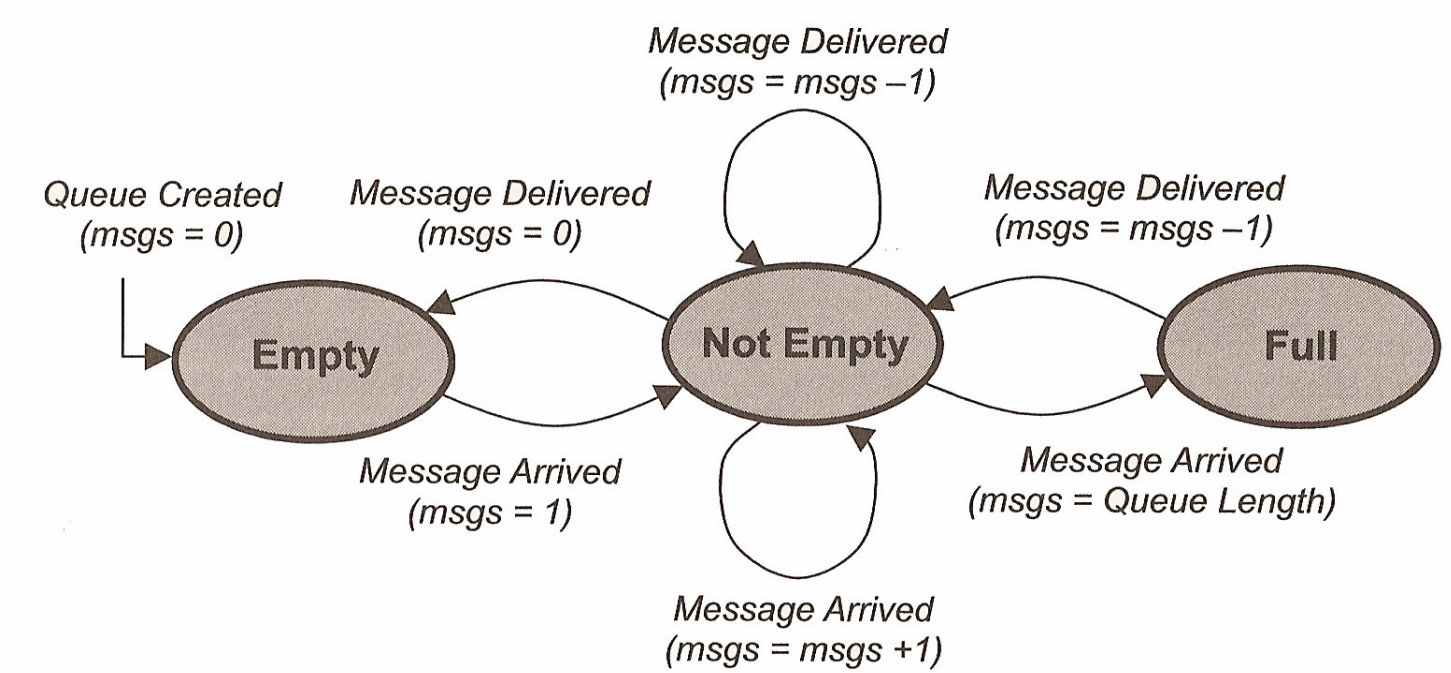
Firmware baseado em *Kernel*

- Sistemas operacionais de tempo real (RTOS)
  - *Message queues*
    - Permite a transferência de dados entre tarefas/processos
    - Estrutura:



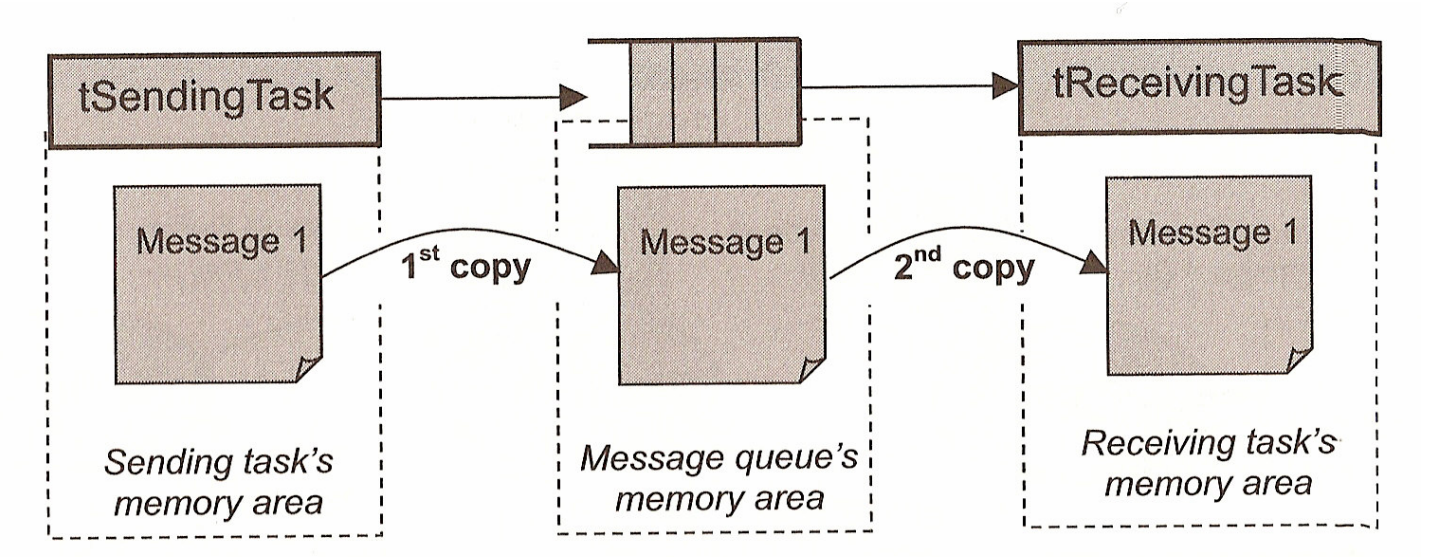
Firmware baseado em *Kernel*

- Sistemas operacionais de tempo real (RTOS)
  - *Message queues*
    - Estados:



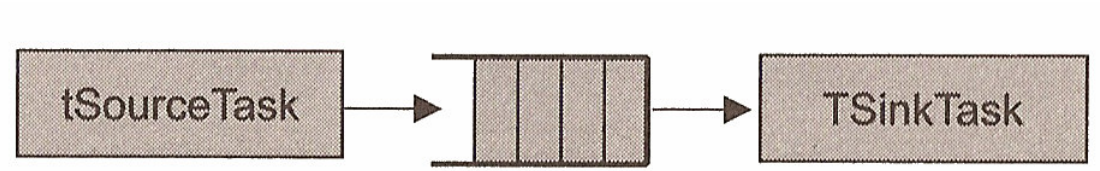
Firmware baseado em *Kernel*

- Sistemas operacionais de tempo real (RTOS)
  - *Message queues*
    - Procedimento de uso:

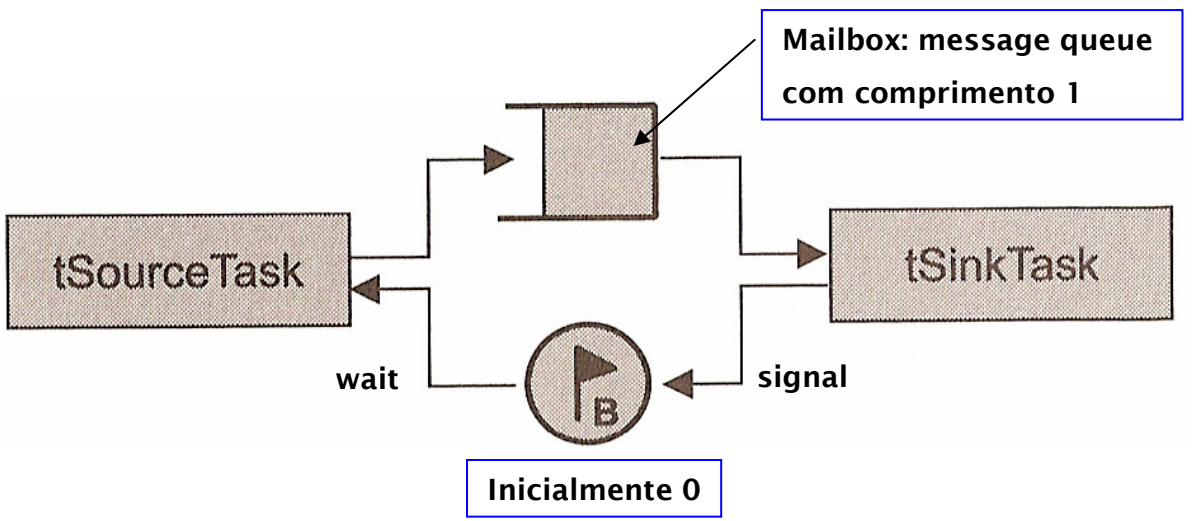


Firmware baseado em *Kernel*

- Sistemas operacionais de tempo real (RTOS)
  - *Message queues*
    - Emprego de *message queues*:
      - Comunicação em sentido único sem inter-travamento:

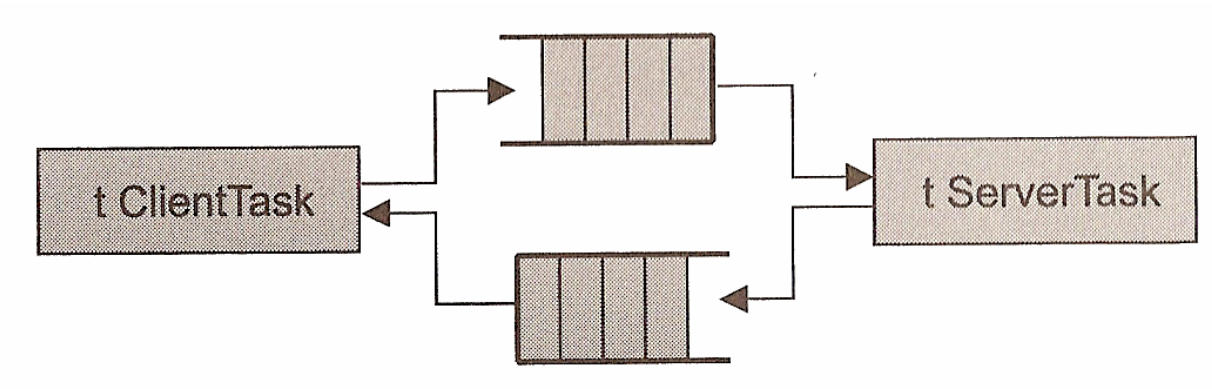


      - Comunicação em sentido único com inter-travamento:



Firmware baseado em *Kernel*

- Sistemas operacionais de tempo real (RTOS)
  - *Message queues*
    - Emprego de *message queues*:
      - Comunicação em sentido duplo com inter-travamento:



Firmware baseado em *Kernel*

- Sistemas operacionais de tempo real (RTOS)
  - *Rate-monotonic schedulers*: a prioridade de uma tarefa é determinada em função de sua freqüência
  - Outros tópicos:
    - Processos
    - Objetos de sincronismo: eventos
  - RTOS para microcontroladores AVR:
    - FreeRTOS: http://www.freertos.org/
    - AvrX: http://www.barello.net/avrx/

Referências

[Li & Yao, 2003] LI, Q.; YAO, C. *Real-Time Concepts for Embedded Systems*. CMPBooks, 2003.

[Laplante, 1993] Laplante, P.A. *Real-Time Systems Design and Analysis – an engineer's handbook*, IEEE Computer Society Press, 1993;

[Stallings, 2001] William Stalling, *Operating Systems: Internals and Design Principles*, Prentice Hall, 2001.