Universidade Federal de Santa Catarina
Centro Tecnológico – CTC
Departamento de Engenharia Elétrica

"Escalonamento de Tarefas em Sistemas Embarcados"

Prof. Eduardo Augusto Bezerra

Eduardo.Bezerra@ufsc.br

Curso preparado utilizando parcialmente material didático do programa universitário da Renesas - https://www.renesas.com/

Florianópolis, novembro de 2010.

“Escalonamento de Tarefas em Sistemas Embarcados”

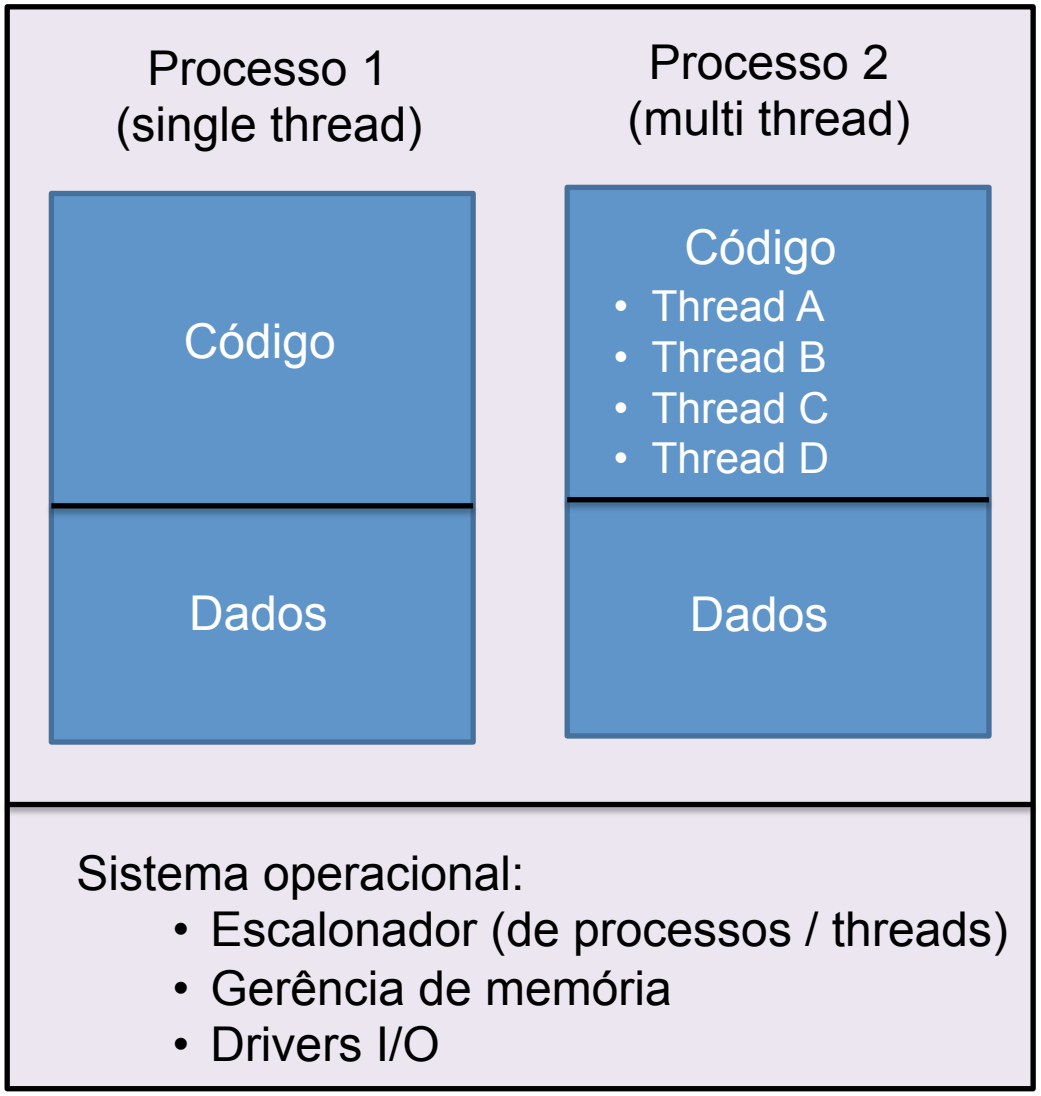
- Agenda:
  - Conceitos básicos
  - Políticas de escalonamento
  - Escalonador
  - Sistemas operacionais embarcados comerciais

Sistemas operacionais e tarefas (tasks)

Tarefa (task) – Programa/Processo
- Contexto
- Código
- Dados
- Memória, I/O

Função (método)
- Contexto
- Código
- Dados
- Não tem memória própria. Ex. ao usar *new*, aloca memória para o processo, e não para a função

Thread (pertence a um processo)
- Contexto
- Código
- Não possui dados privados



Processos vs Threads

- Processo – informação não é visível para outros processos (nada é compartilhado)
- Thread – compartilha espaço de endereçamento e código com outras threads (também conhecido por “processo leve”)
- Efeito secundário: tempo para chaveamento de contexto pode variar
  - Chaveamento de processos necessita troca de grande quantidade de informação
  - Chaveamento de threads necessita bem menos informação (PC, ponteiro para pilha, estado da CPU, outros registradores)

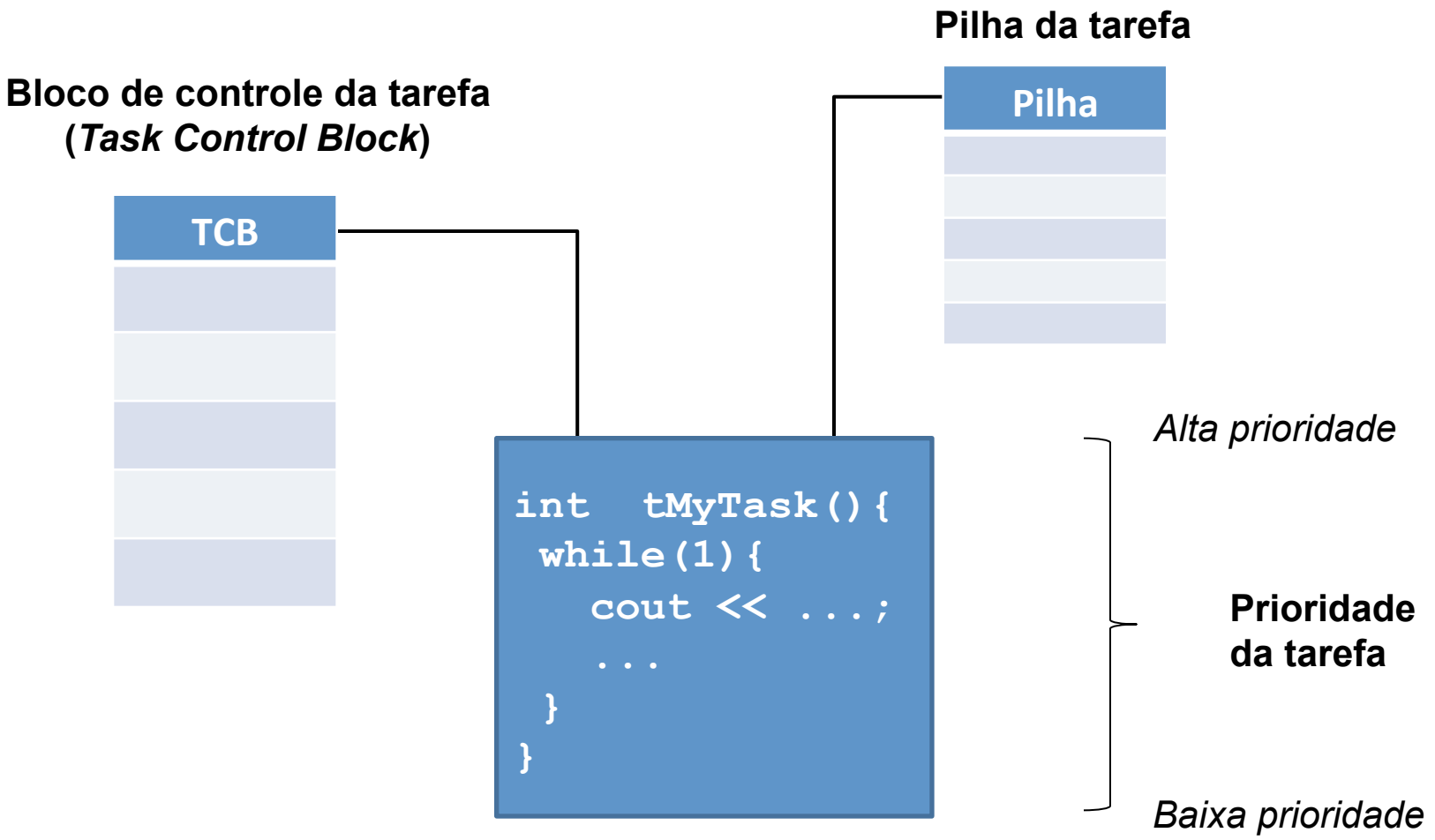
Gerência da execução de tarefas em SOs

Sistemas operacionais e tarefas

- Sistemas simples e sequenciais:
  - *main()* executa *funções()* para realizar tarefas simples (ex. laço infinito onde é realizada a leitura de um sensor de temperatura, e um LCD é atualizado sempre que a temperatura for alterada).
- Em sistemas de maior complexidade são utilizadas mais tarefas para tratamento de eventos.
- Projeto do sistema: funções que precisam ser executadas periodicamente são modeladas e implementadas como **tarefas**.

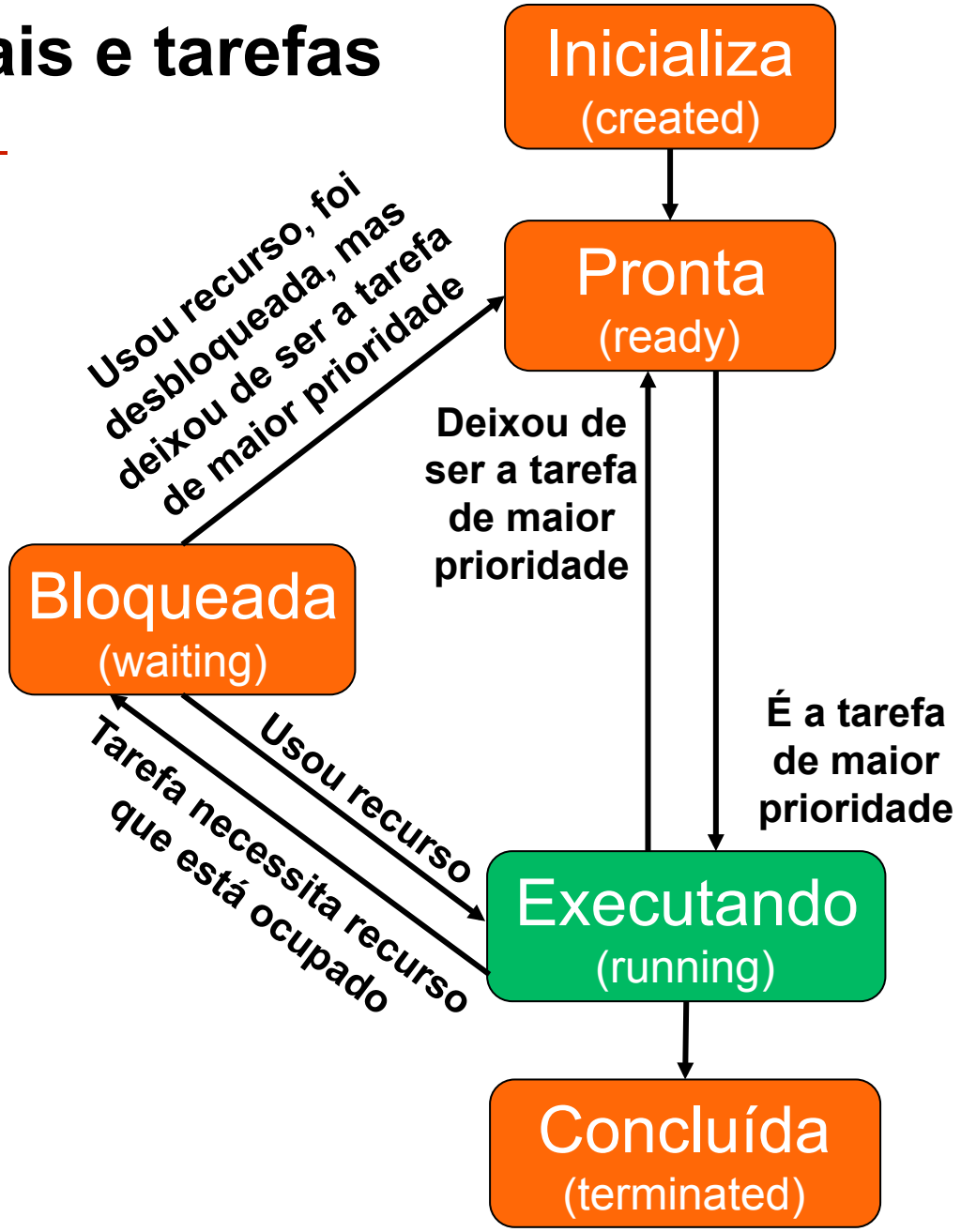
Sistemas operacionais e tarefas

- Tarefa (*task*) é uma unidade de execução independente e concorrente com outras tarefas, podendo ser "escalonável"



Sistemas operacionais e tarefas

FSM típica para representação dos estados de uma tarefa



Regras para escalonamento

- Funções que precisam executar periodicamente são projetadas como tarefas.
- Se existir uma tarefa “pronta” para execução, então deve ser executada.
- *Run to completion – RTC* – Estratégia onde a tarefa em execução é concluída antes de iniciar a execução de outra tarefa.
- Se existir mais de uma tarefa pronta para execução, executar primeiro a de maior prioridade.

Escalonamento de processos

- Escalonamento - abordagens para compartilhamento do processador:
  - Escalonamento **não-preemptivo** – as tarefas não são interrompidas. Uma determinada tarefa, assim que iniciada, será executada até seu término (*Run To Completion*, RTC).
  - Escalonamento **preemptivo** – tarefas podem ser preemptadas (interrompidas) por outras tarefas, seguindo regras de escalonamento pré-definidas.

Obs. A palavra "preemptivo" não existe em dicionários da língua portuguesa.

Escalonamento não-preemptivo

Projeto do comportamento do escalonador

- **Manter tabela com informação da tarefa:**
  - Ponteiro para o início da tarefa
  - A "duração" da tarefa
  - Intervalo de tempo ("delay") para próxima execução
    - Decrementar esse timer frequentemente
    - Recarregar com "duração" em caso de overflow
  - Informação se está "pronta" para execução
  - Informação se está "habilitada"
- **Usar timer periodicamente para atualizar "delay" (um tick por milisegundo) - ISR**
- **Escalonador executa tarefa se "ready" = 1**

Inicialização da tabela da tarefa

```
#define MAX_TASKS  5   // nr. máximo de tarefas
typedef struct {
  int period;        // duração da tarefa em ticks
  int delay;         // intervalo de tempo para próxima ativação
  int ready;         // binário: 1 = "executar agora"
  int enabled;       // tarefa ativa?
  void (* task)(void);     // endereço (ponteiro) da função
} task_t;
task_t GBL_task_table[MAX_TASKS];   // Tabela global de tarefas

void init_Task_Table(void) {
  // Inicializa as entradas de todas as tarefas com 0
  int i;
  for (i=0 ; i<MAX_TASKS ; i++) {
    GBL_task_table[i].delay = 0;
    GBL_task_table[i].ready = 0;
    GBL_task_table[i].period = 0;
    GBL_task_table[i].enabled = 0;
    GBL_task_table[i].task = NULL;
  }
}
```

Inicialização do temporizador (*ticks*)

```
// Configura timer B0 para gerar uma interrupção a cada 1 ms
// UPDATE
// tb0 default = 65536 (timer gera tick a cada 3,2768 ms)
// se tb0 for carregado com 20000, gera tick a cada 1,0000 ms

init_Task_Timers(); // Inicializa todas tarefas
tb0 = 20000;        // um tick a cada 1 ms
DISABLE_INTS
tb0ic = 1;          // Timer B0 overflow
ENABLE_INTS
tb0st = 1;          // Inicia timer B0
```

Atualização da tabela da tarefa a cada tick

```
// A cada tick do timer
//     - Dec intervalo de tempo para próxima ativação (delay)
//     - Se delay == 0, marcar tarefa como Ready para executar e
//       recarregar delay com duração da tarefa (Period)
//
// Certificar que tabela de interrupção possuirá o endereço desse ISR
#pragma INTERRUPT tick_timer_intr
void tick_timer_intr(void) {
static char i;
for (i=0 ; i<MAX_TASKS ; i++) {
   if ((GBL_task_list[i].task != NULL) &&   //Se for escalonada
       (GBL_task_list[i].enabled == 1) &&
       (GBL_task_list[i].delay != 0)      ) {
          GBL_task_list[i].delay--;          // Decrementa
          if (GBL_task_list[i].delay == 0){
             GBL_task_list[i].ready = 1;
             GBL_task_list[i].delay = GBL_task_list[i].period;
          } // if delay == 0
       } // if
} // for
```

API do RTC - *Run to completion* – ver arquivo rtc.[c|h]

Tarefas executam até terminar, para só então devolver o controle para CPU

Init_RTC_Scheduler(void)

- **Inicializa timer B0 para uso por todas as tarefas**

Add Task(*task*, *time period*, *priority*)

- ***task*: endereço da tarefa (nome da função, sem parênteses)**
- ***time period*: duração da tarefa (em ticks)**
- ***priority*: valores menores representam mais altas prioridades. ID da tarefa**
- **Habilita a tarefa automaticamente**
- **Retorna: 1 – carregou com sucesso, 0 – falha na carga da tarefa**

Remove Task(*task*)

- **Remove a tarefa do escalonador**

Run Task(*task number*)

- **Informa escalonador que a tarefa pode executar quando for possível**

Run RTC Scheduler()

- **Executa o escalonador!**
- **Nunca retorna**
- **Antes de chamar essa função é necessário pelo menos uma tarefa escalonada**

Enable_Task(*task_number*) e Disable_Task(*task_number*)

- **Ativa ou limpa flag de enable, definindo se tarefa pode ou não executar**

Reschedule_Task(*task_number*, *new_period*)

- ***Altera a duração da tarefa. Também inicializa (reset) o timer com esse valor.***

Execução do escalonador

```
void Run_RTC_Scheduler(void) {  // Sempre executando
  int i;
  GBL_run_scheduler = 1;
  while (1) {       // Laço infinito; verifica cada tarefa
    for (i=0 ; i<MAX_TASKS ; i++) {
      // Se essa for uma tarefa escalonada
      if ((GBL_task_list[i].task != NULL) &&
          (GBL_task_list[i].enabled == 1) &&
                (GBL_task_list[i].ready == 1)     ) {
                  GBL_task_list[i].task(); // Executar
                  GBL_task_list[i].ready = 0;
                break;
      } // if
    } // for i
  } // while 1
}
```

Adicionando uma tarefa

```
int addTask(void (*task)(void), int time, int priority){
  unsigned int t_time;
  /* Verifica se a prioridade é válida */
  if (priority >= MAX_TASKS || priority < 0) return 0;
  /* Verifica se sobre-escreve uma tarefa escalonada */
  if (GBL_task_list[priority].task != NULL) return 0;
  /* Escalona a tarefa */
  GBL_task_list[priority].task = task;
  GBL_task_list[priority].ready = 0;
  GBL_task_list[priority].delay = time;
  GBL_task_list[priority].period = time;
  GBL_task_list[priority].enabled = 1;
  return 1;
}
```

Removendo uma tarefa

```
void removeTask(void (* task)(void))
{
  int i;

  for (i=0 ; i<MAX_TASKS ; i++) {
    if (GBL_task_list[i].task == task) {
      GBL_task_list[i].task = NULL;
      GBL_task_list[i].delay = 0;
      GBL_task_list[i].period = 0;
      GBL_task_list[i].run = 0;
      GBL_task_list[i].enabled = 0;
      return;
    }
  }
}
```

Habilitando / desabilitando uma tarefa

```
void Enable_Task(int task_number)
{
  GBL_task_list[task_number].enabled = 1;
}

void Disable_Task(int task_number)
{
  GBL_task_list[task_number].enabled = 0;
}
```

Reescalonando uma tarefa

```
// Altera a duração da tarefa e inicializa (reset) o contador

void Reschedule_Task(int task_number, int new_period)
{
  GBL_task_list[task_number].period = new_period;
  GBL_task_list[task_number].delay = new_period;
}
```

Disparando o sistema “Round Robin”

Antes de executar o escalonador RTC é necessário adicionar a(s) tarefa(s) - funções a serem executadas concorrentemente:

```
addTask(flash_redLED, 25, 3);
addTask(sample_ADC, 500, 4);
```

O último comando a ser executado na função *main()* deve ser:

```
Run_RTC_Scheduler(); // nunca retorna
```

Escalonamento preemptivo

Introdução

- Escalonamento não-preemptivo:
  - Rotinas de interrupção – *foreground*
  - Tarefas (funções) – *background*
  - Limitação – funções precisam ser projetadas de forma a executar até o fim sem liberar a CPU. Problema, como aproveitar melhor o tempo da CPU enquanto tarefas esperam por recursos?
- Compartilhamento eficiente da CPU
  - PCs, servidores – Possibilitar que diversos usuários compartilhem os recursos de um computador
  - Sistemas embarcados – Simplificar projeto dos programas, possibilitando divisão do projeto em diversos componentes (tarefas)

Escalonamento preemptivo

- Kernel:
  - Compartilha processador entre várias tarefas (threads/processos).
  - Força a troca da CPU da tarefa A para B, continuando tarefa A mais tarde (preempted).
  - Simplifica comunicação entre tarefas.
  - Parte central (núcleo) de sistemas operacionais.

Preemptivo vs não-preemptivo

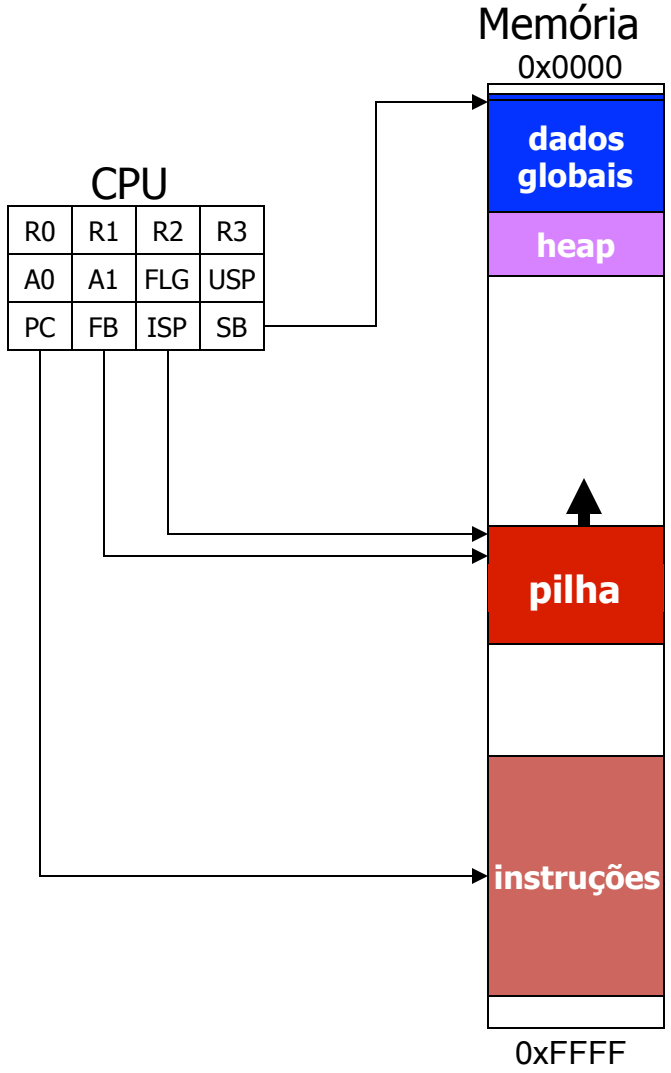
- **Kernel não-preemptivo / sistema multitarefa cooperativo**
  - **As tarefas precisam, explicitamente, liberar a CPU**
  - **Eventos assíncronos são gerenciados por ISRs**
  - **ISR sempre retorna para tarefa interrompida**
  - **Tempo de resposta do sistema, a nível de tarefa, será o tempo de resposta da tarefa mais lenta**

- **Kernel preemptivo**
  - **A cada escalonamento a tarefa de mais alta prioridade recebe a CPU**
  - **Quando uma tarefa de mais alta prioridade fica pronta, a tarefa em execução é interrompida e colocada na fila de tarefas prontas**
  - **Tempo de resposta máximo é menor do que no sistema não-preemptivo**
  - **Dados compartilhados, normalmente, precisam de semáforos**

Quais informações de estado devem ser mantidas?

Task (processo ou thread) em execução possui informação sobre seu estado (contexto):

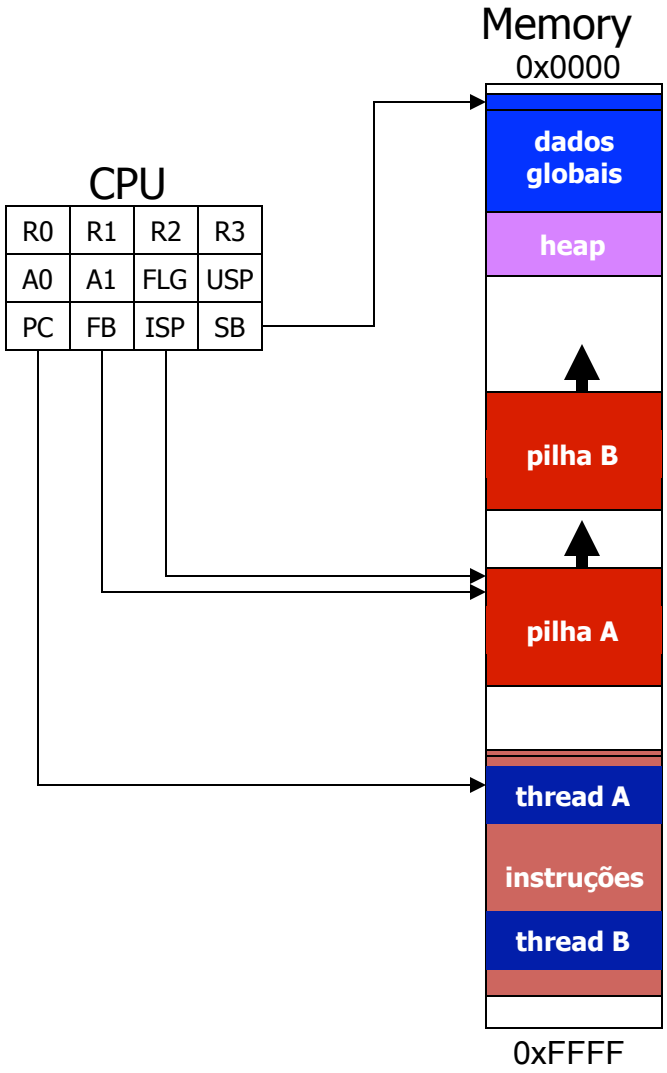
- Instrução atual – apontada pelo PC
- Pilha – apontada pelo *stack pointer*
  - Parâmetros, variáveis locais, endereços de retorno
- Outros estados da CPU
  - Valores de registradores (tudo que é compartilhado e que pode ser afetado por outros processos) – regs de uso geral, ponteiro para pilha, ...
  - Flags de status (zero, carry, ...)
- Outras informações
  - Arquivos abertos, info gerência de memória, número do processo, ...



Mantendo o estado para múltiplas tarefas

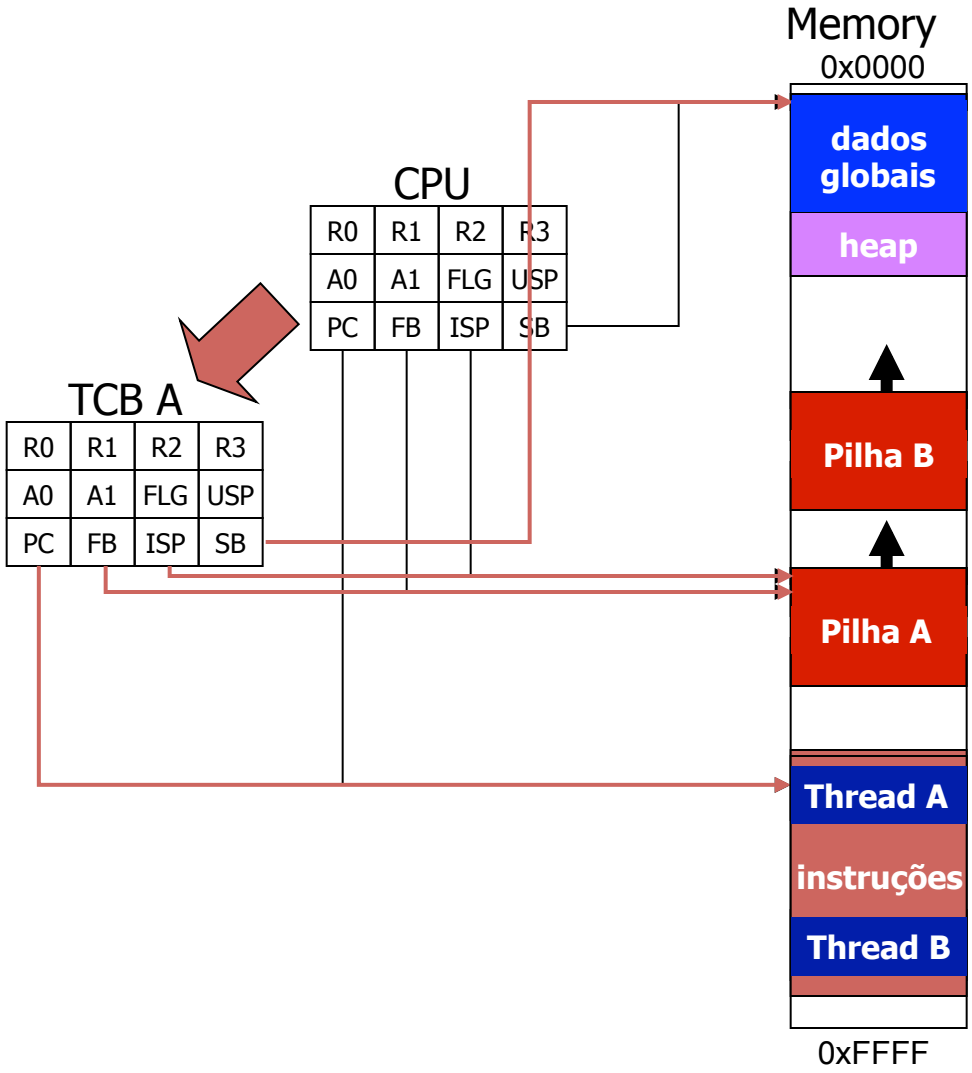
- **Informações da task são armazenadas em um bloco de controle TCB – (task/*thread control block*)**

- **Exemplo: chaveando da task A para a task B**
  - **Uma pilha para cada task**
  - **Variáveis globais compartilhadas pelas tasks**



Passo 1 – copiar estado da CPU na TCB A

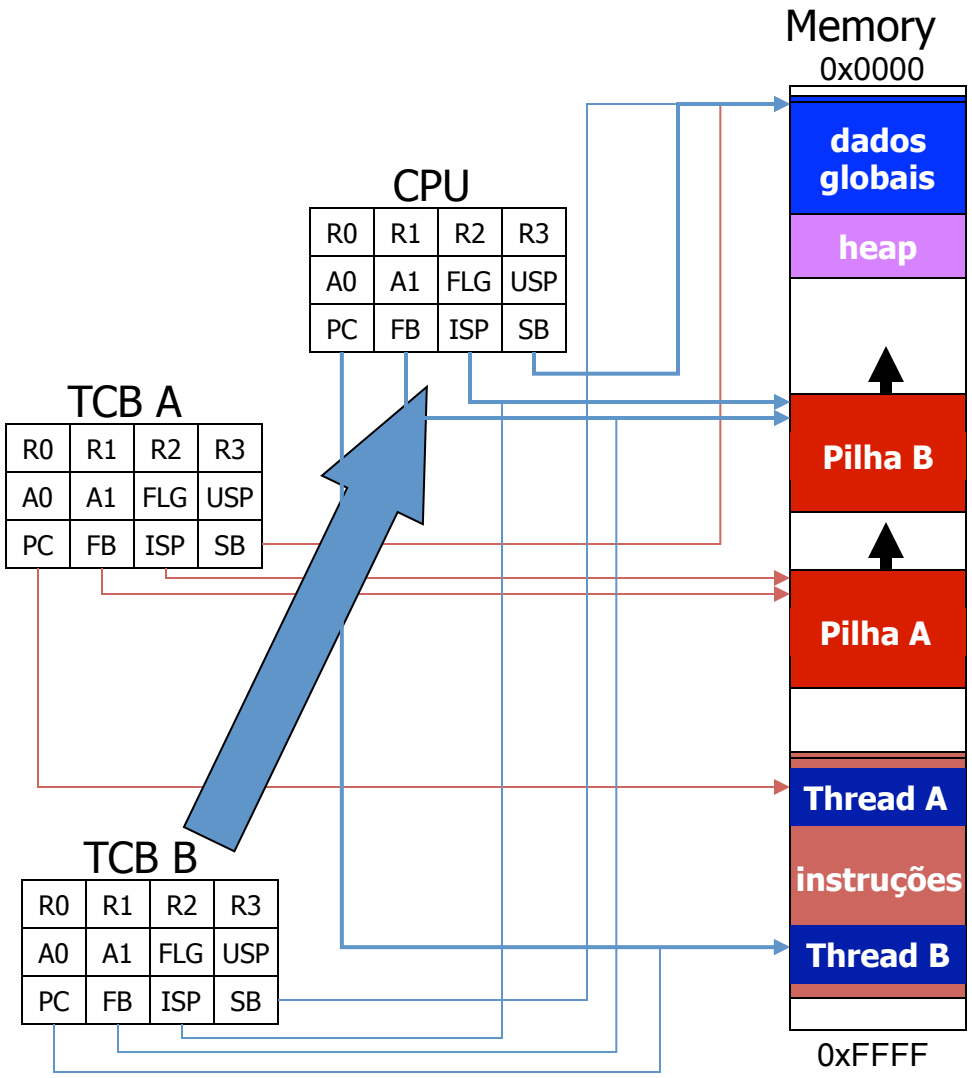
- **CPU inicialmente está executando a task A, logo é preciso salvar o contexto da task A na TCB A.**



TCB = *task/thread control block*

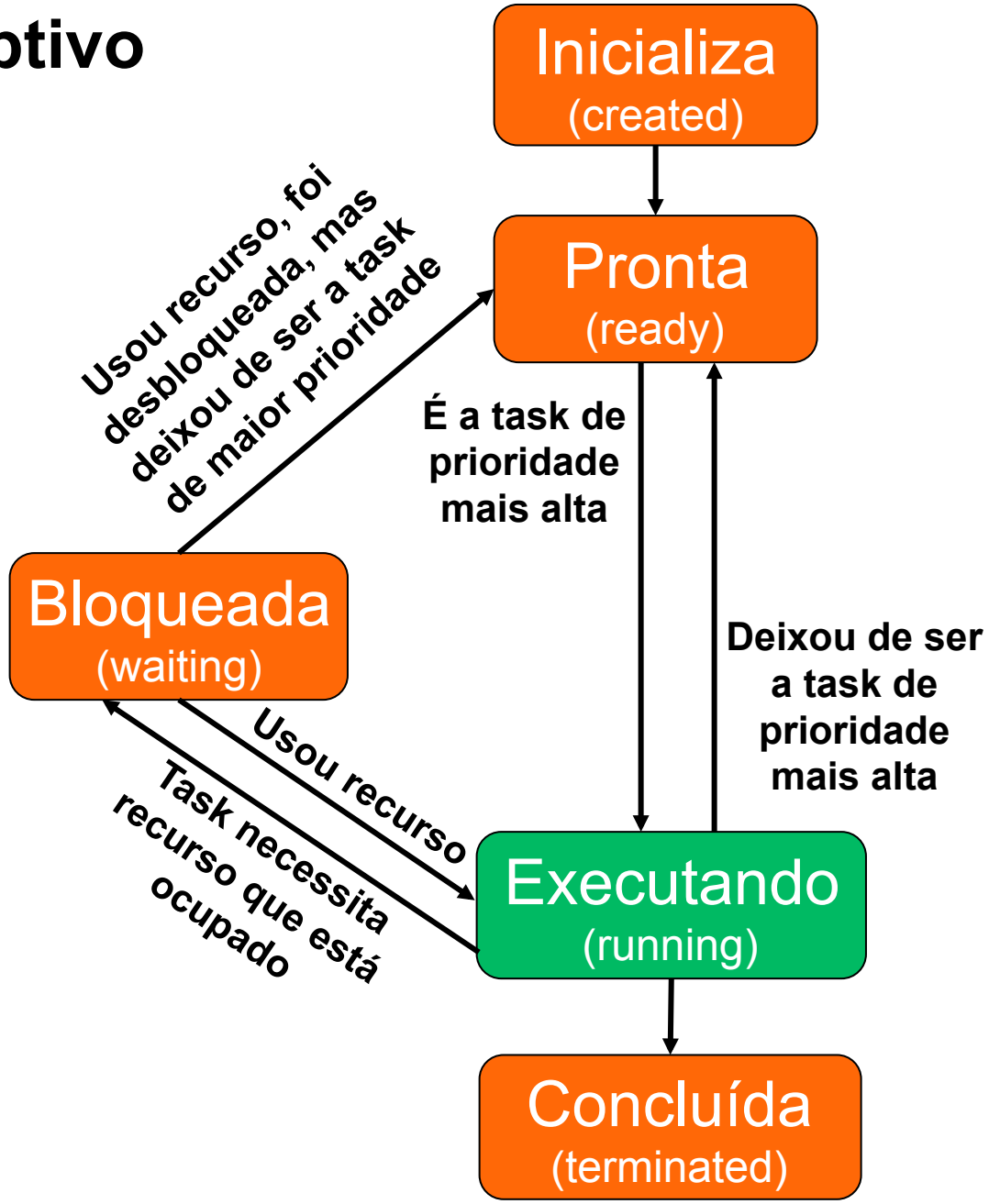
Passo 2 – recarregar o estado anterior da CPU, que se encontra armazenado na TCB

- CPU é recarregada com o estado anterior da task B, no ponto exato onde a execução da task foi interrompida (preempted).
- Esse chaveamento de contexto é realizado pelo *dispatcher* (expedição)
- Código, normalmente, escrito em assembly para acessar recursos não acessíveis em C



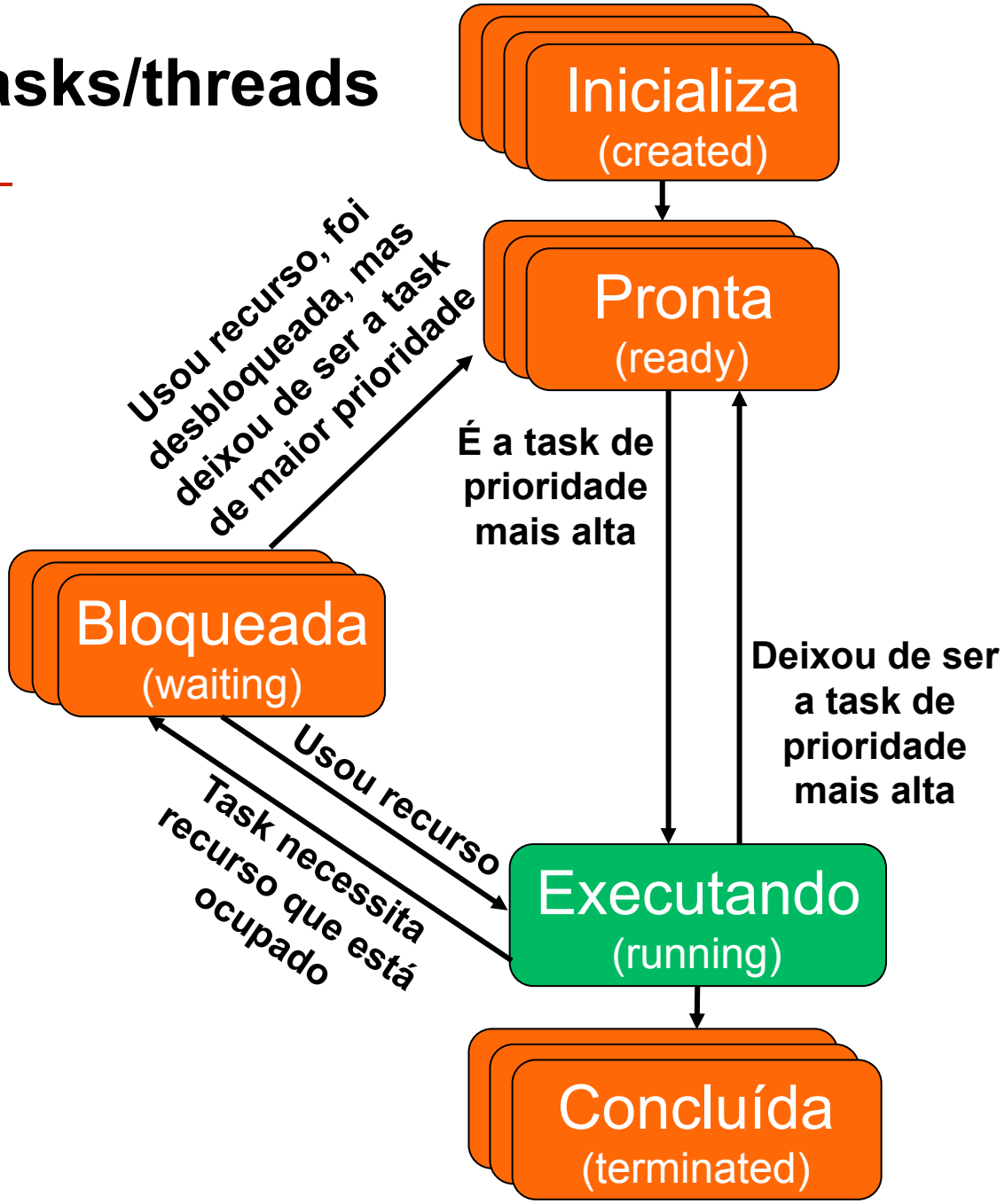
Escalonamento preemptivo
(estados de uma task/thread)

FSM típica para representação dos estados de uma task



Filas de estados de tasks/threads

- Criação de uma fila para cada estado (exceto “executando”)
- Blocos de controle de tasks são armazenados nas filas apropriadas
- Kernel move tasks nas filas/ registradores conforme necessário

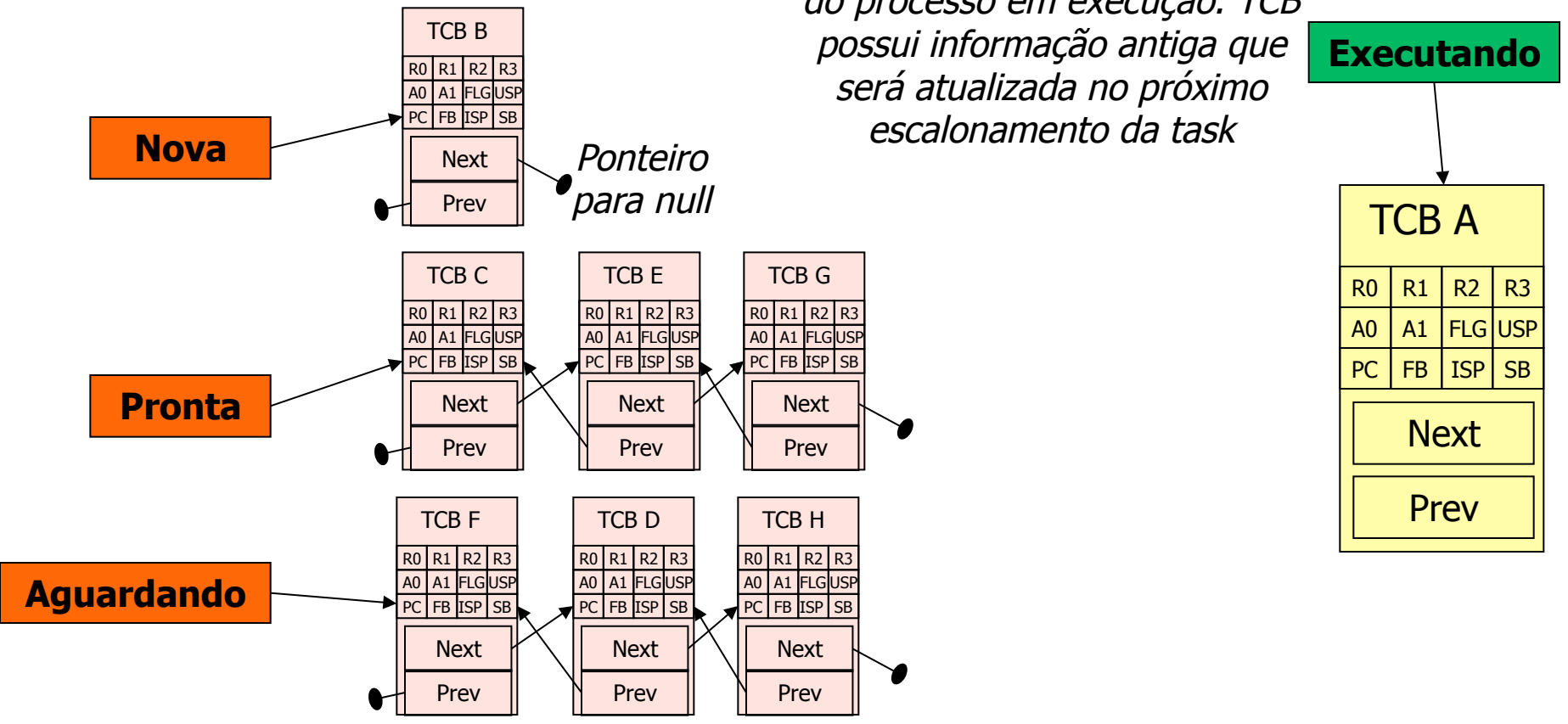


Controle de estados de tasks

- **Escalonador do OS controla tasks e seus estados**
  - **Para cada estado, OS mantém uma fila de TCBs para todos os processos do estado**
  - **Move TCBs de uma fila para outra, conforme as alterações de estados das tasks**
  - **Escalonador seleciona para execução uma das tasks "prontas", a de mais alta prioridade**

- **Trocas de estado de uma task – o que força isso?**
  - **OS recebe um tick do timer, forçando a decisão do que será feito a seguir**
  - **Uma task voluntariamente libera a CPU**
  - **Uma task precisa de informação/recurso que ainda não está pronta**

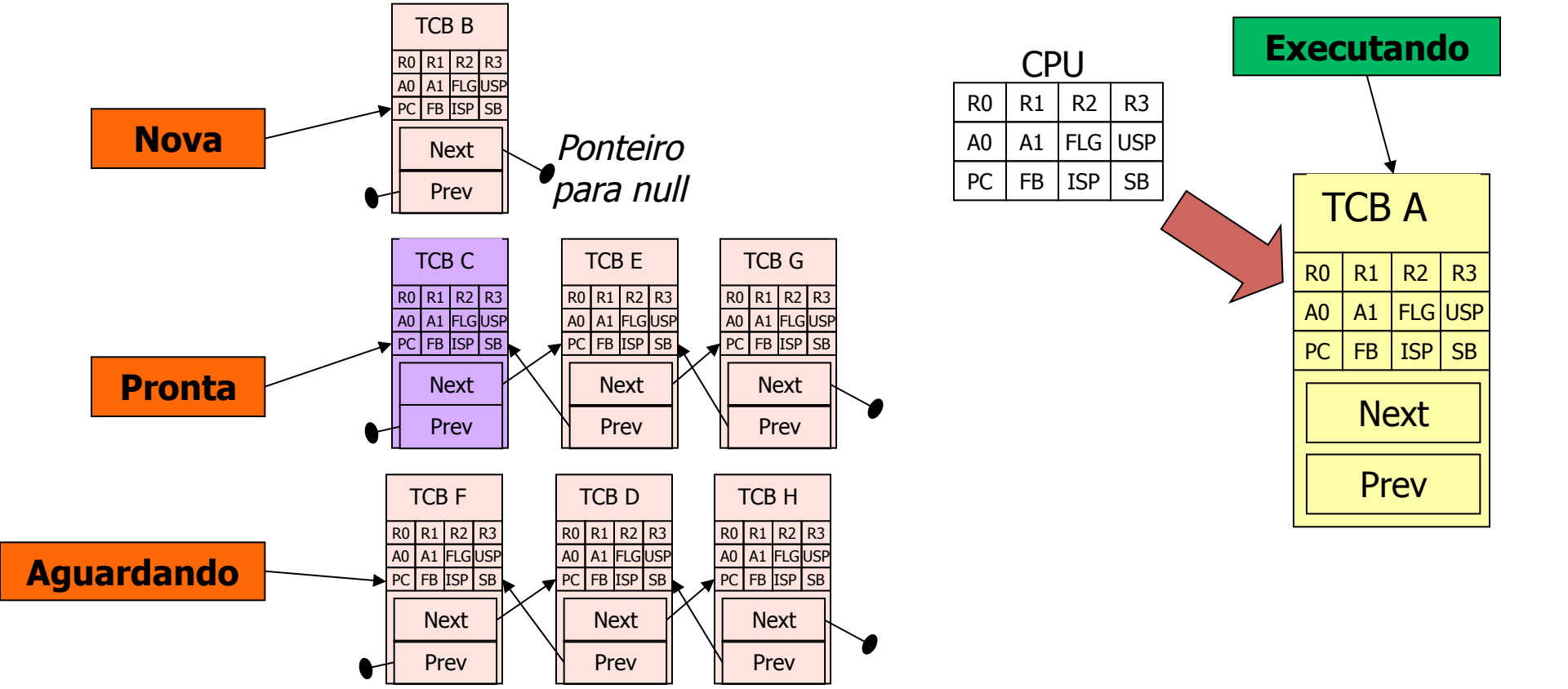
Estruturas de dados do escalonador



- Incluir ponteiros "*next*" e "*prev*" nos TCBs para construção de uma lista duplamente encadeada
- Criar um ponteiro para cada fila ("*Nova*", "*Pronta*", "*Aguardando*")
- Criar um ponteiro ("*Executando*") para a TCB da task em execução

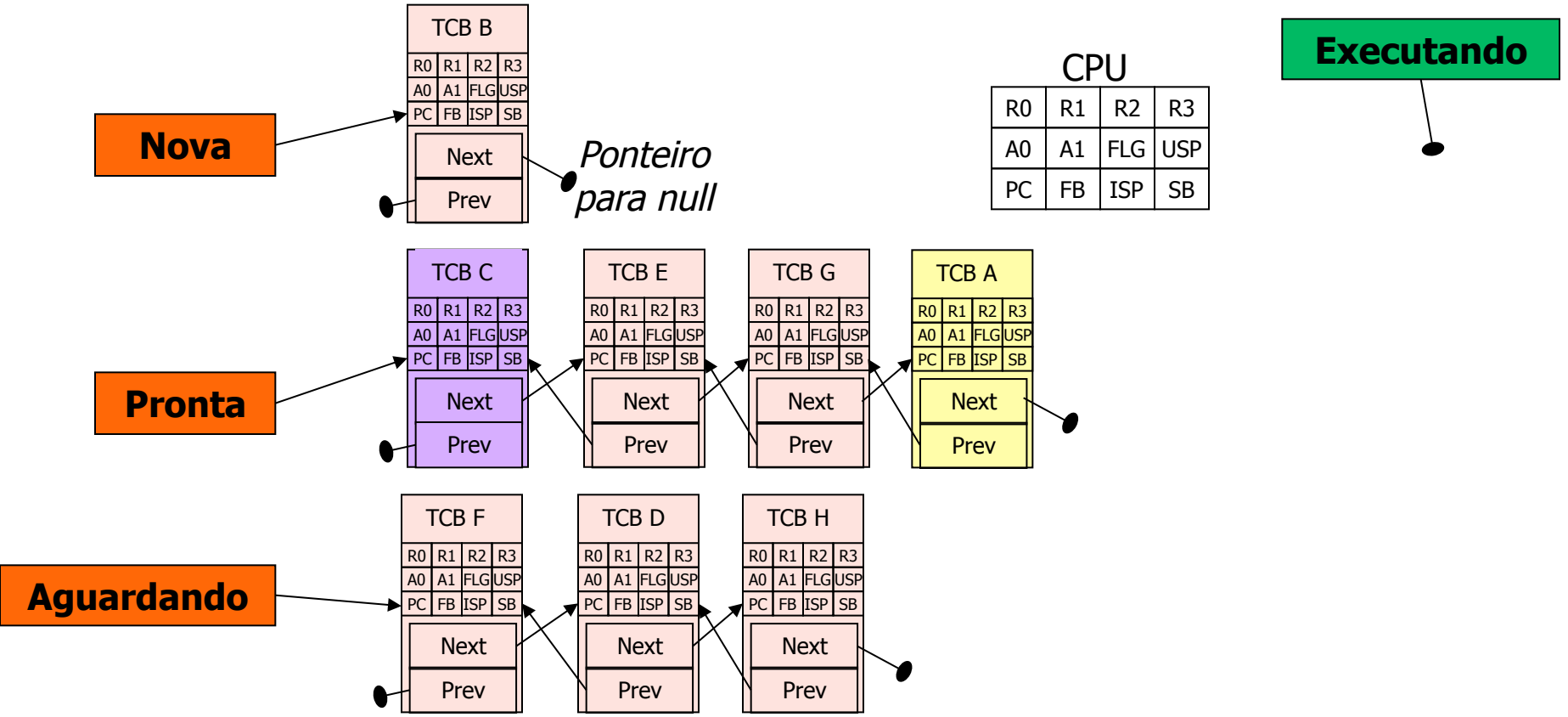
Exemplo de chaveamento de contexto

- **Task A em execução, e escalonador decide chavear para Task C. Task A ainda poderia continuar executando, mas possui prioridade menor do que C**
- **Salvar estado da CPU na TCB A**



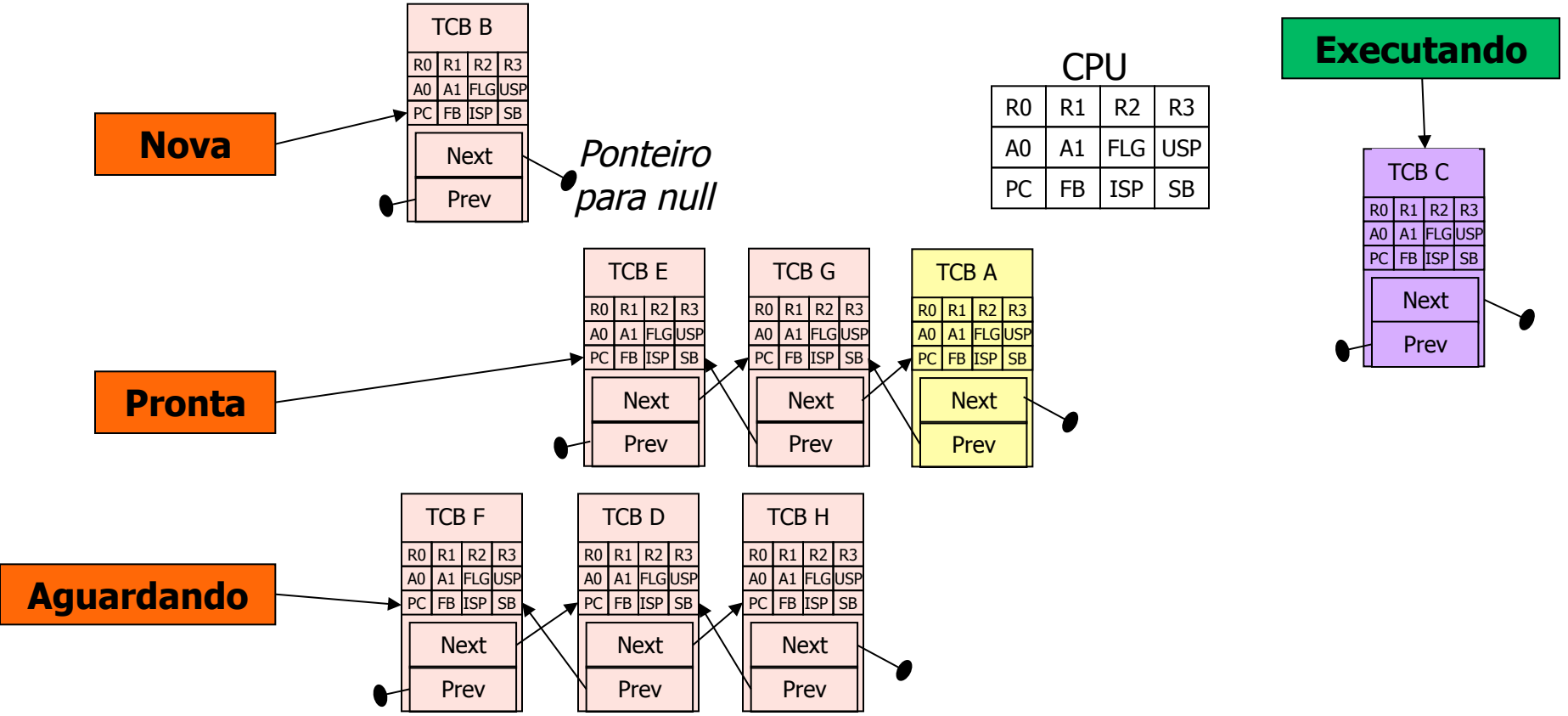
Exemplo de chaveamento de contexto

- **Modificar os ponteiros necessários para inserir TCB A na fila de tasks prontas**



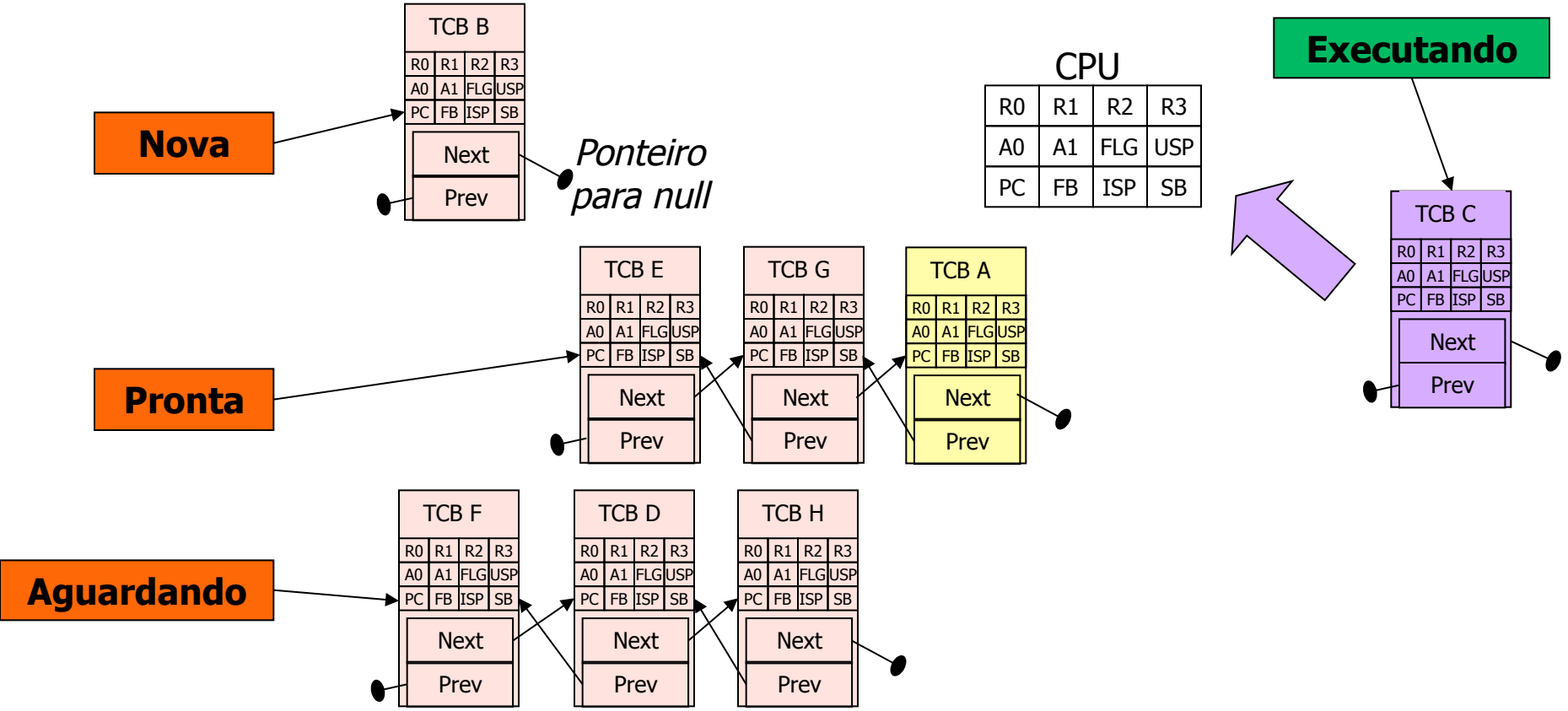
Exemplo de chaveamento de contexto

- **Remover Task C da fila da fila de prontas e marcar como próxima a ser executada**



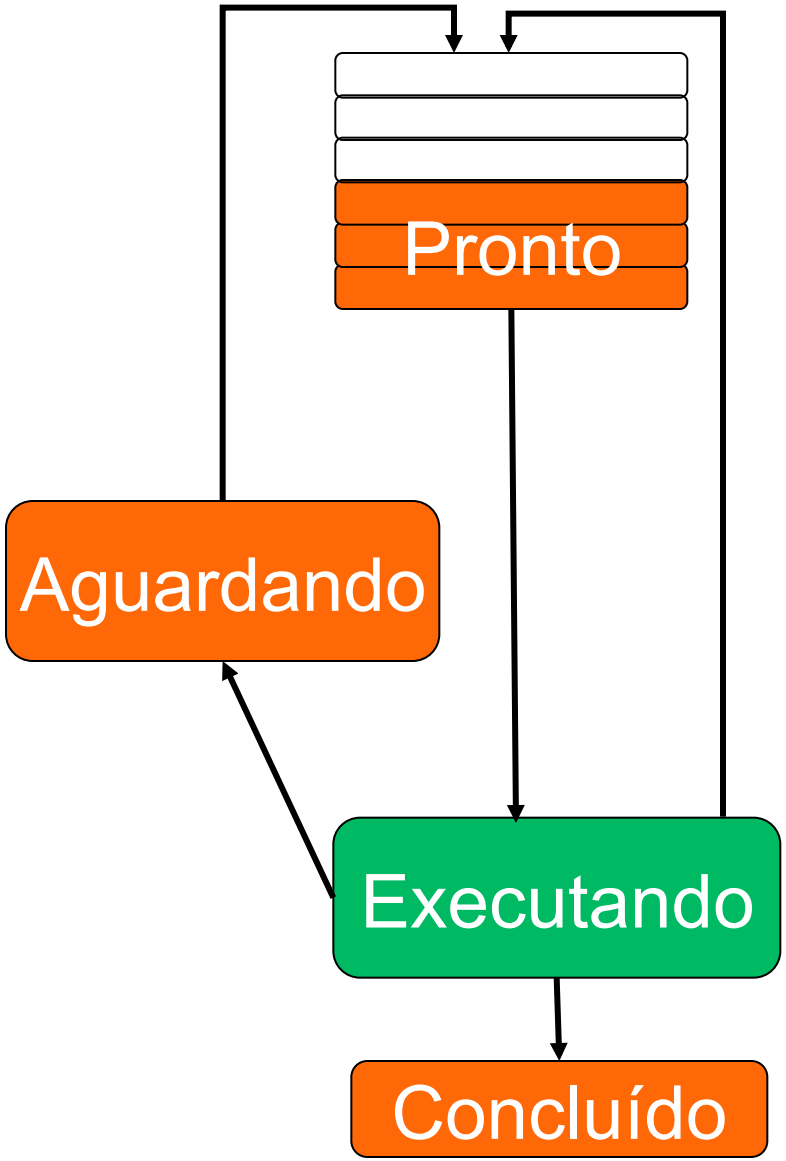
Exemplo de chaveamento de contexto

- **Copiar as informações de estado da Task C na CPU e continuar sua execução**



Escolha do algoritmo de escalonamento

- Escolha da task “pronta” a ser executada
- Critério em comum:
  - Uso da CPU – fração de tempo em que a CPU permanece ocupada
  - Throughput – número de tarefas finalizadas por unidade de tempo
  - Tempo total – intervalo entre a submissão de uma tarefa a CPU até a execução total (com todos escalonamentos)
  - Tempo de espera – tempo que a tarefa permanece aguardando na fila de espera



Outros algoritmos de escalonamento utilizados em sistemas embarcados

- First-Come, First Served (FCFS)
  - Todas as filas funcionam como FIFOs tradicionais, sem prioridade
  - Problemas: alto atraso médio, não preemptivo
- Round Robin: incluindo tempo compartilhado no FCFS
  - No final do tick, tarefa em execução é colocada no fim da fila de tarefas “prontas”
  - Problemas: Também possui alto atraso médio
- Shortest Job First (SJF)
  - SJF é comprovadamente o ideal para minimizar o tempo médio de espera
  - Problema: Como determinar a duração do próximo job? Pode ser realizada uma previsão baseada na duração do job anterior?

RTOS – Real Time Operating System

- Sistemas operacionais tradicionais (não tempo real)
  - Difícil prever o tempo de resposta
  - Difícil garantir que uma tarefa irá sempre terminar antes do seu prazo máximo (deadline)
- Sistema Operacional de Tempo Real - RTOS
  - Facilidade para determinar que uma tarefa irá executar antes de atingir seu deadline
  - Projetado para tarefas periódicas
- Exemplos de escalonadores usados em RTOS
  - Rate Monotonic Scheduling (RMS) – Quanto menor o período do processo, maior a prioridade
  - Earliest Deadline First (EDF)

Compartilhamento de dados entre processos/ threads

- **Memória compartilhada**
  - Ideal para situações com baixo custo para comunicação
  - Regiões críticas – região utilizada apenas por um processo por vez
  - Semáforos utilizados para acesso exclusivo a recursos compartilhados

- **Troca de mensagens**
  - Ideal para situações com alto custo para comunicação (sistemas distribuídos)
  - Processo “produtor” gera mensagens, e processo “consumidor” recebe mensagens
  - Bloqueante, não-bloqueante, broadcast

Exemplos de Sistemas Operacionais Embarcados

μC/OS-II

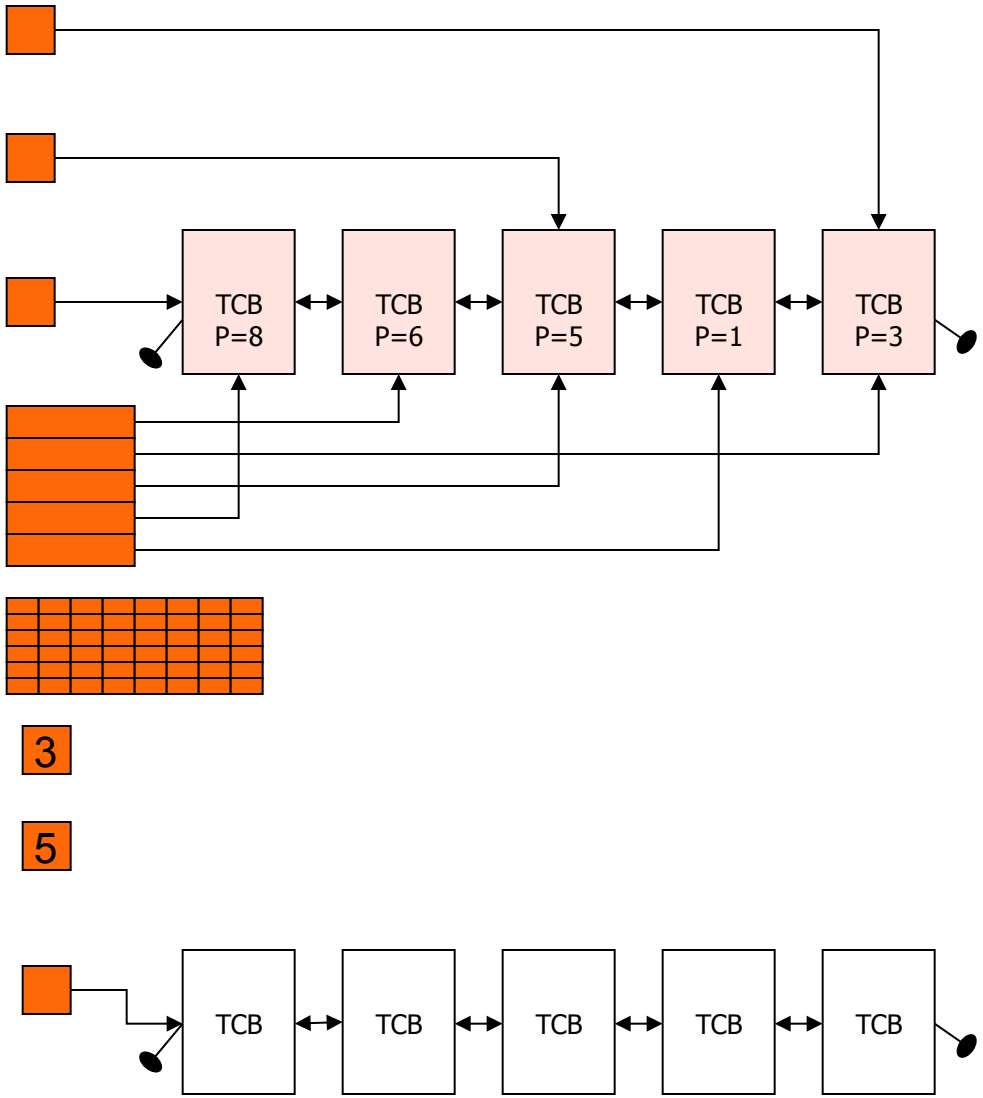
- **Kernel de tempo real**
  - **Portável, escalável, RTOS preemptivo**
  - **Porte para cerca de 90 processadores**
- **Autor: Jean J. Labrosse, Micrium, *http://ucos-ii.com***
- **Implementação**
  - **Estado da CPU não é armazenado em TCBs, e sim na pilha da própria thread**
  - **A "TCB" armazenada na pilha gerencia também os limites de memória da própria pilha**
  - **TCB também monitora eventos, mensagens, e atrasos**

TCB do μC/OS-II (armazenada na pilha)

```
typedef struct os_tcb {
   OS_STK *OSTCBStkPtr;          /* Pointer to current top of stack */
        void   *OSTCBExtPtr;     /* Pointer to user definable data for TCB
                                    extension */
   OS_STK *OSTCBStkBottom;       /* Pointer to bottom of stack - last
                                    valid address */
   INT32U OSTCBStkSize;          /* Size of task stack (in bytes) */
   INT16U OSTCBOpt;              /* Task options as passed by
                                     OSTaskCreateExt() */
   INT16U OSTCBId;               /* Task ID (0..65535) */
   struct os_tcb *OSTCBNext;     /* Pointer to next TCB in the TCB list */
   struct os_tcb *OSTCBPrev;     /* Pointer to previous TCB in list */
   OS_EVENT *OSTCBEventPtr;      /* Pointer to event control block */
   void *OSTCBMsg;               /* Message received from OSMboxPost() or
                                    OSQPost() */
   INT16U OSTCBDly;              /* Nbr ticks to delay task or, timeout
                                    waiting for event */
   INT8U OSTCBStat;              /* Task status */
   INT8U OSTCBPrio;              /* Task priority (0 == highest,
                                    63 == lowest) */
   BOOLEAN OSTCBDelReq;          /* Indicates whether a task needs to
                                    delete itself */
} OS_TCB;
```

TCB do μC/OS-II (armazenada na pilha)

- OSTCBCur – Ponteiro para TCB da tarefa em execução
- OSTCBHighRdy – Ponteiro para TCB de mais alta prioridade pronta para execução
- OSTCBList – Ponteiro para lista duplamente encadeada de TCBs
- OSTCBPrioTbl[OS_LOWEST_PRIO + 1] – Tabela de ponteiros para criação de TCBs, por prioridade
- OSReadyTbl – Tabela codificada de tarefas prontas para execução
- OSPrioCur – Prioridade da tarefa
- OSPrioHighRdy – Prioridade da tarefa pronta de mais alta prioridade
- OSTCBFreeList – Lista de OS_TCBs livres, usada para criar novas tarefas



Dispatcher para o μC/OS-II

```
_OSCtxSw:
PUSHM R0,R1,R2,R3,A0,A1,SB,FB
MOV.W _OSTCBCur, A0;OSTCBCur->OSTCBStkPtr = Stack pointer
STC    ISP, [A0]   ;Call user definable OSTaskSwHook()
JSR    _OSTaskSwHook
;OSTCBCur = OSTCBHighRdy
MOV.W _OSTCBHighRdy, _OSTCBCur;OSPrioCur = OSPrioHighRdy
MOV.W _OSPrioHighRdy, _OSPrioCur
;Stack Pointer = OSTCBHighRdy->OSTCBStkPtr
MOV.W _OSTCBHighRdy, A0
LDC    [A0], ISP ;Restore all processor registers from the
new task's stack
POPM   R0,R1,R2,R3,A0,A1,SB,FB
REIT
```

VxWorks da empresa WindRiver (Intel, Julho 2009)

VxWorks, RTOS para sistemas embarcados, com versões para diversas arquiteturas: x86, MIPS, PowerPC, ColdFire, Intel i960, ARM, StrongARM

Principais funcionalidades:

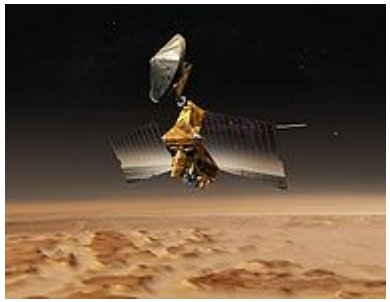
- **Kernel multitarefa com escalonamento preemptivo e round-robin com tempo de resposta rápido**
- **Proteção de memória para isolar kernel de aplicações do usuário**
- **Suporte a SMP**
- **Comunicação entre processos rápida e flexível incluindo TIPC**
- **Framework para manipulação de erros**
- **Semáforos para exclusão mútua com herança de prioridade**
- **Filas de mensagens locais e distribuídas**
- **Certificado de conformidade com POSIX PSE52**
- **Sistema de arquivos**
- **Pilha de rede IPv6**
- **Simulador VxSim**

Alguns sistemas embarcados usando VxWorks

- Honda Robot *ASIMO*
- *KUKA* industrial robots
- Airbus A400M Airlifter (in development)
- Boeing 787 airliner (in development)
- Boeing 747-8 airliner (in development)
- BMW iDrive system
- Linksys WRT54G wireless routers (versions 5.0 and later)
- Xerox *Phaser* and other Adobe PostScript-based computer printers
- The Experimental Physics and Industrial Control System (EPICS)
- Apache Longbow attack helicopter
- ALR-67(V)3 Radar Warning Receiver used in the F/A-18E/F Super Hornet
- Siemens VDO automotive navigation systems
- Siemens AG MRI measurement control units
- AMX Controls System Devices
- External RAID controllers designed by LSI Corporation and used in IBM System Storage's DS3000 and DS4000 (formerly FAStT) plus some storage systems from Silicon Graphics, Sun Microsystems/StorageTek, Teradata, Dell, Sepaton, BlueArc and several other companies worldwide

VxWorks em missões espaciais

- **Deep Space Program Science Experiment (DSPSE) - Clementine**
  Clementine foi lançada em 1994 e utilizou o RTOS VxWorks 5.1 em uma CPU baseada no MIPS. O sistema era responsável pelo rastreamento de estrelas e por algoritmos de processamento de imagens. Na época o uso de sistemas RTOS comerciais em veículos espaciais era considerado como um experimento.
- **Mars Reconnaissance Orbiter**
- ***Phoenix* Mars Lander**
- ***Deep Impact* space probe**
- **James Webb Space Telescope (em desenvolvimento)**
- ***Sojourner* Mars Pathfinder rover**
- ***Spirit* and *Opportunity* Mars Exploration Rovers**
- **Stardust**



Mars Reconnaissance Orbiter

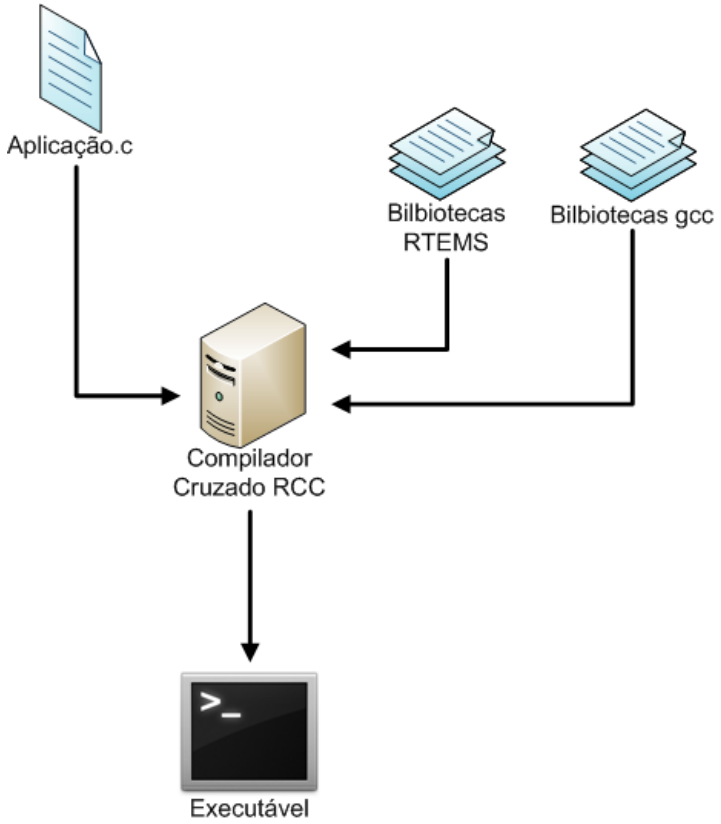
RTEMS - Real-Time Executive for Multiprocessor Systems

RTEMS, RTOS para sistemas embarcados, com versões para diversas arquiteturas: ARM, Atmel AVR, Blackfin, ColdFire, TI C3x/C4x DSPs, H8/300, Intel (80386, Pentium, e superiores), Lattice Mico32, 68k, Renesas M32C, Renesas M32R, MIPS, Nios II, PowerPC, Renesas SuperH, SPARC

Principais funcionalidades:

- **Kernel multitarefa com escalonamento preemptivo dirigido por eventos**
- **Escalonamento rate-monotonic opcional**
- **Sistemas multiprocessados homogêneos e heterogêneos**
- **Comunicação entre processos e sincronização**
- **Alocação dinâmica de memória**
- **UDP/TCP**
- **DNS, FTP, NFS, FTP, HTTPD, Telnetd, RPC, Corba**
- **In-memory file system (IMFS), MS-DOS FAT32 (16 e 12), NFS**
- **Gnu debugger gdb**
- **Debug via Internet**
- **Debug via porta serial**

RTEMS – Geração de executável



Outros sistemas operacionais embarcados

- Windows CE (Microsoft)
- Android (Google)
- Symbian OS (Nokia)
- Green Hills Software (INTEGRITY e velOSity RTOS)
- QNX Inc. (QNX Neutrino system)
- LynuxWorks (LynxOS RTOS)
- Mentor Graphics (Nucleus RTOS)
- Microsoft (Windows CE e Windows NT Embedded)
- Linux RTAI
- Diversos baseados em Linux