

Fundamentos de Sistemas Operacionais de Tempo Real

Criando seu próprio escalonador de tarefas

Marcelo Barros de Almeida
marcelobarrosalmeida@gmail.com

Licenciamento Creative Commons

A obra "Fundamentos de Sistemas Operacionais de Tempo Real - Criando seu próprio escalonador de tarefas" de Marcelo Barros de Almeida foi licenciada com uma Licença Creative Commons - Atribuição - Uso Não-Comercial - Partilha nos Mesmos Termos 3.0 Não Adaptada.

Com base na obra disponível em <http://code.google.com/p/basicrtos/>

Podem estar disponíveis permissões adicionais ao âmbito desta licença através do contato direto ao autor via email marcelobarrosalmeida@gmail.com.



Marcelo Barros de Almeida

marcelobarrosalmeida@gmail.com

- Formação:

- Engenheiro eletrônico (UNIFEI, 1996), mestre (UFMG, 1998), doutor (UFMG, 2002)

- Atualmente:

- Engenheiro P&D (Smar Equip. Industriais LTDA)
- Professor do Barão de Mauá (RP)

- Detalhes:

- <http://jedizone.wordpress.com>
- <http://www.twitter.com/marcelobarros>
- http://linuxabordo.com.br/wiki/index.php?title=Marcelo_Barros
- <http://lattes.cnpq.br/0711663486251657>



smar

Sumário

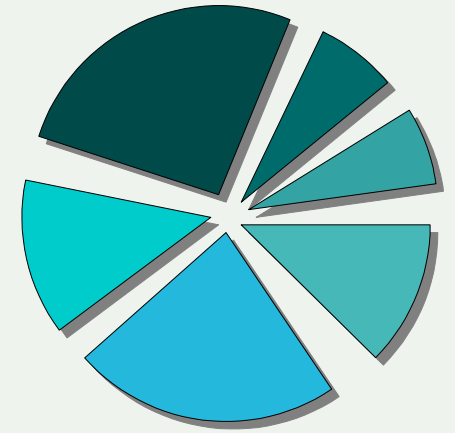
- Objetivos
- Dividindo o tempo do processador
- Troca de contexto
- O escalonador
- Adicionando tarefas
- Inicializando o sistema

Objetivos

- Interrupt driven x Multitasking
- Explorar os princípios de um RTOS
- Descrever via um exemplo didático: Basic RTOS
 - Número fixo de tarefas (mesma prioridade)
 - Time slice diferentes
 - Requer apenas um timer
 - Não preemptivo
 - ~1150 bytes de flash, ~128 bytes de RAM
 - Fontes: <http://code.google.com/p/basicrtos/>

Como dividir o tempo do processador ?

- Time slicing
 - Tempo de tarefa
 - Tempo do RTOS
 - Escalonador, interrupções, timers, etc
- Estratégia:
 - Definição de um tick para o sistema
 - Interrupção periódica de timer de alta prioridade
 - Avaliação periódica das tarefas em execução



Dividindo o tempo do processador

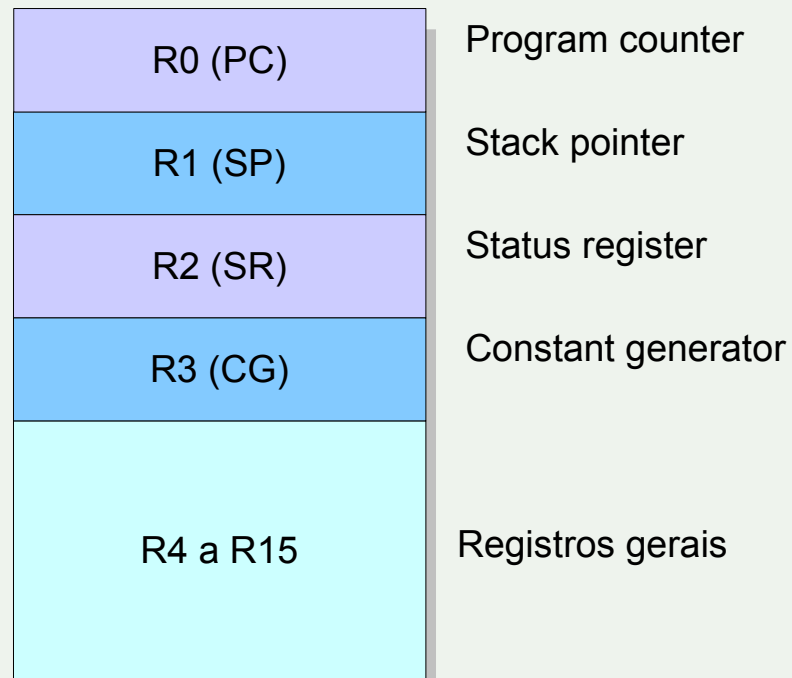
Implementando no MSP430/GCC um timer cíclico:

```
static interrupt (WDT_VECTOR) BRTOS_Scheduler(void);

static void BRTOS_ConfigureClock(void)
{
    WDTCTL = WDT_MDLY_0_5;          /* configuring interval timer */
    usTicksPerSecond = 1000/0.5; /* 2k ticks per second      */
}
```

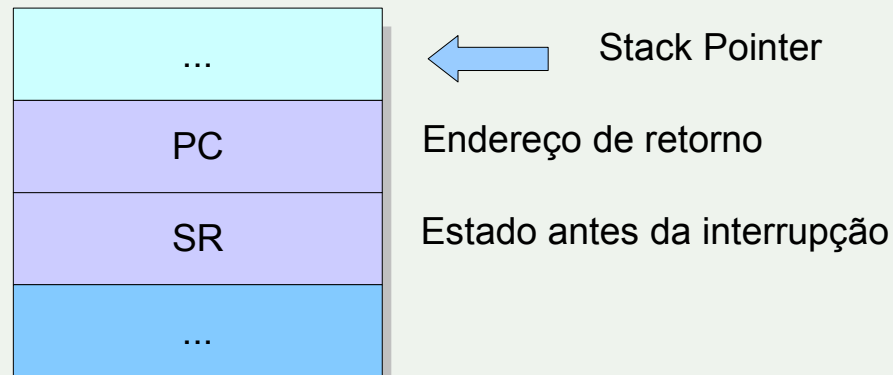
Troca de contexto

- Como compartilhar um só conjunto de registros ?



Troca de contexto

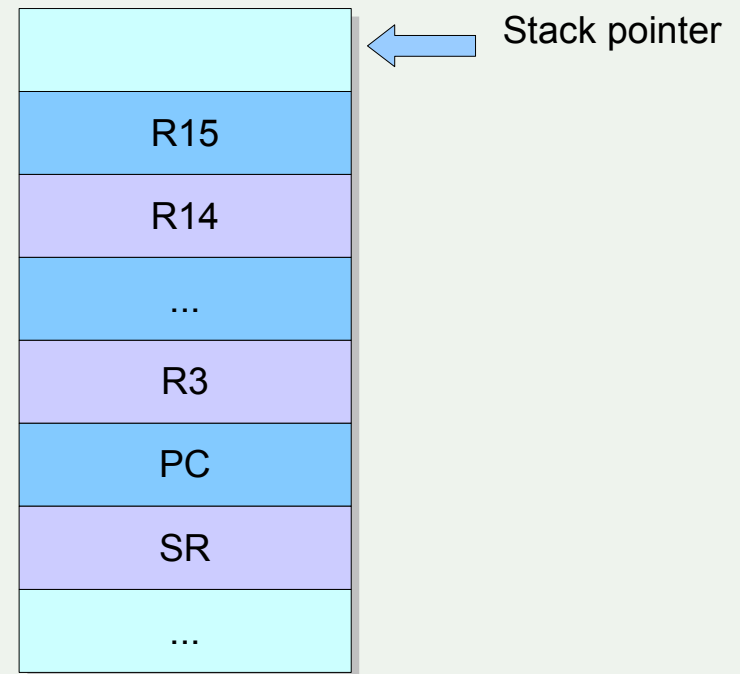
- O que acontece no momento da interrupção do temporizador ?
- Para o MSP430:
 - PC e SR salvos na pilha, automaticamente
 - A instrução reti recupera os registros da pilha, ao sair
- Todo o resto do trabalho precisa ser feito para a troca de contexto



Troca de contexto

- Salvando o restante dos registros, no contexto da tarefa corrente (não mudamos o SP ainda)

```
#define SaveContext() \
    __asm__ __volatile__ \
    ( "push R3\n" \
      "push R4\n" \
      "push R5\n" \
      "push R6\n" \
      "push R7\n" \
      "push R8\n" \
      "push R9\n" \
      "push R10\n" \
      "push R11\n" \
      "push R12\n" \
      "push R13\n" \
      "push R14\n" \
      "push R15" )
```



Troca de contexto

- Falta salvar o SP atual antes de usar um outro contexto
- Informações sobre a tarefa: Task Block Control (TCB)

```
#define SaveStackPointer() \  
    asm("mov.w R1,%0" : "=m" \  
        (asBrtosTasks[ucCurrentTask].pusStackPtr))
```

```
typedef struct {  
    pfTaskEntry    pfEntryPoint;        /* task entry point    */  
    unsigned char  ucPriority;           /* task priority       */  
    unsigned char  ucTaskState;         /* current task state  */  
    unsigned short usTimeSlice;         /* desired time slice  */  
    unsigned short *pusStackBeg;        /* stack beginning     */  
    unsigned short *pusStackPtr;         /* stack pointer       */  
    unsigned short usSleepTicks;        /* count sleep ticks   */  
    unsigned short usTicks;             /* count slice ticks    */  
} BRTOS_TCB;
```

Troca de contexto

- O novo contexto é o do escalonador
- O escalonador decide qual a próxima tarefa a ser executada
- A nova tarefa tem o seu contexto restaurado e ganha o controle do processador

O escalonador

```
NAKED( BRTOS_Scheduler )
{
save_context_entry:

    SaveContext();
    SaveStackPointer();
    RestoreSchedStackPointer();

dont_save_context_entry:

    BRTOS_ProcessTimers();
    BRTOS_SleepTasks();
    ucCurrentTask = BRTOS_RoundRobin(ucCurrentPriLevel);

    if(ucCurrentTask == BRTOS_NO_TASK_TO_RUN)
    {
        GoToLowPowerMode3();
        goto dont_save_context_entry;
    }

    SaveSchedStackPointer();
    RestoreStackPointer();
    RestoreContext();
    EnableInterrupts();
    ReturnFromInterrupt();
}
```

O escalonador

- Processar timers pode ser feito em uma tarefa também
- Round Robin não é a única forma de escalonar, talvez você não queira ser "justo" sempre
- Adicionar prioridades pode melhorar o nível de controle do sistema
- Uma tarefa nula pode ser interessante quando o sistema estiver ocioso
- Economizar energia pode ser um requisito

Adicionando tarefas

- Em geral, é um processo simples:
 - Definir uma função para a tarefa
 - Reservar espaço para a pilha da tarefa
 - Especificar prioridade e slice de tempo
- As tarefas entram no vetor de TCBs, na partida do sistema
- Alguns cuidados essenciais:
 - Criar o contexto inicial da tarefa na pilha dela
 - Criar um contexto para o caso de a tarefa retornar
 - Alinhe a área do stack

Adicionando tarefas

```
#define TASK_STACK_SIZE 50

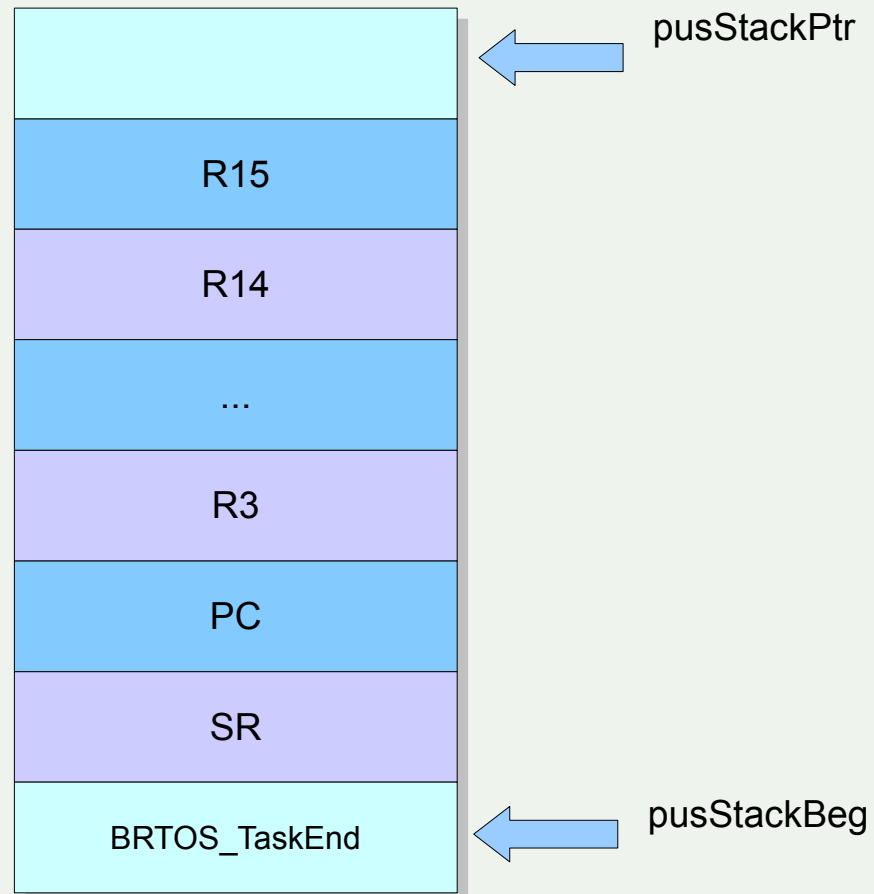
unsigned short usStack[TASK_STACK_SIZE/2];

void task(unsigned long ulArgc)
{
    while(1)
    {
        int i = 0;
        while(i < 50)
        {
            i = i + 1;
        }
        BRTOS_Sleep(5);
    }
}

void BRTOS_Application_Initialize(void)
{
    BRTOS_CreateTask(task,
                    (unsigned short)&usStack[TASK_STACK_SIZE/2-1],
                    10,
                    BRTOS_TASK_PRIORITY_1);
    /* ... other tasks ... */
}
```

Adicionando tarefas

- Criando o contexto inicial



Adicionando tarefas

```
asBrtosTasks[n].pfEntryPoint = entry_point;
asBrtosTasks[n].pusStackBeg  = (unsigned short *) stack_addr;
asBrtosTasks[n].pusStackPtr  = (unsigned short *) stack_addr;
asBrtosTasks[n].usTimeSlice  = MSEC_TO_TICKS(time_slice);
asBrtosTasks[n].ucPriority    = pri;
/* more initialization here */

/* if task returns some day, prepare stack */
*(asBrtosTasks[n].pusStackPtr) = (unsigned short) BRTOS_TaskEnd;
asBrtosTasks[n].pusStackPtr    -= 1;

/* status register and program counter */
*(asBrtosTasks[n].pusStackPtr) = (unsigned short) 0;
asBrtosTasks[n].pusStackPtr    -= 1;
*(asBrtosTasks[n].pusStackPtr) = (unsigned short) entry_point;
asBrtosTasks[n].pusStackPtr    -= 1;

/* prepare for first RestoreContext():
   dummy NUM_REGS_IN_CONTEXT regs */
for(i = 0 ; i < NUM_REGS_IN_CONTEXT ; i++)
{
    *(asBrtosTasks[n].pusStackPtr) = (unsigned short) 0;
    asBrtosTasks[n].pusStackPtr    -= 1;
}
```

Adicionando tarefas

- One more thing ...

```
static void BRTOS_TaskEnd(void)
{
    EnterCriticalSection();
    asBrtosTasks[ucCurrentTask].ucTaskState =
        BRTOS_TASK_STATE_TERMINATED;
    LeaveCriticalSection();
    while(1);
}
```

Inicializando o sistema

- Basicamente, é preciso:
 - Inicializar as variáveis de controle do RTOS
 - Definir as tarefas (não existe suporte à criação dinâmica de tarefas)
 - Chamar o escalonador

```
int main(void)
{
    SaveSchedStackPointer();
    BRTOS_Initialize();
    /* call user initialization (create threads here) */
    BRTOS_Application_Initialize();
    /* the stack pointer should point to the first available
       task stack, and the context should be subtracted. This way,
       the calling of GoToScheduler will not fail
    */
    asBrtosTasks[0].pusStackPtr += NUM_REGS_IN_CONTEXT;
    RestoreStackPointer();
    GoToScheduler();
}
```

Comentários, dúvidas ?



marcelobarrosalmeida@gmail.com

<http://twitter.com/marcelobarros>

<http://code.google.com/p/basicrtos/>