Desenvolv. de Sistemas Embarcados em Tempo Real

Prof. Hermano Cabral

Departamento de Eletrônica e Sistemas — UFPE

15 de agosto de 2024

Plano de Aula

Tema central

- ChibiOS — sistema operacional

Plano de Aula

Tema central

- ChibiOS — sistema operacional

Objetivos

- Conhecer as características de threads e temporizador virtual do ChibiOS.

Introdução

- O kernel do ChibiOs é sua parte portável e implementa as funcionalidades esperadas de um sistema de tempo real:
  - Threads e temporizadores virtuais

Introdução

- O kernel do ChibiOs é sua parte portável e implementa as funcionalidades esperadas de um sistema de tempo real:
  - Threads e temporizadores virtuais
  - Semáforos, mutexes, variaveis de condição e eventos

Introdução

- Threads são essenciais em um sistema de tempo real.

Introdução

- Threads são essenciais em um sistema de tempo real.
- Podemos ver uma thread como uma CPU virtual com seus próprios registradores e pilha.

Introdução

- Threads são essenciais em um sistema de tempo real.
- Podemos ver uma thread como uma CPU virtual com seus próprios registradores e pilha.
- As seguintes ações podem ser feitas em uma thread:
  - Criação
  - Encerramento
  - Sincronismo ao término das threads

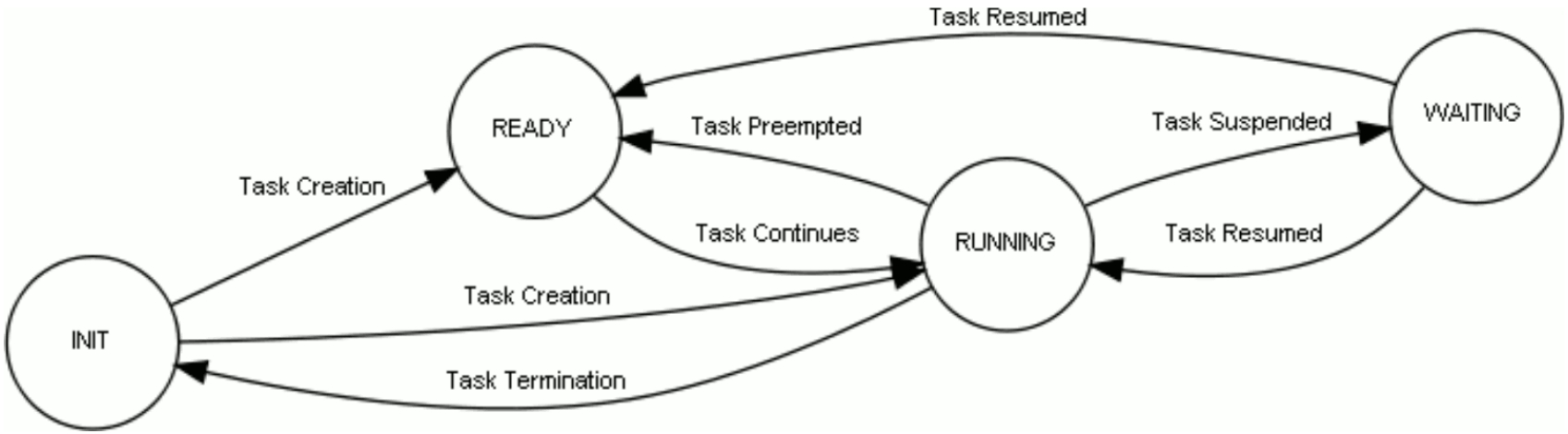
Introdução

- Threads são essenciais em um sistema de tempo real.
- Podemos ver uma thread como uma CPU virtual com seus próprios registradores e pilha.
- As seguintes ações podem ser feitas em uma thread:
  - Criação
  - Encerramento
  - Sincronismo ao término das threads
- Todo programa tem pelo menos uma thread, que é a da função main()

Introdução

- Threads são essenciais em um sistema de tempo real.
- Podemos ver uma thread como uma CPU virtual com seus próprios registradores e pilha.
- As seguintes ações podem ser feitas em uma thread:
  - Criação
  - Encerramento
  - Sincronismo ao término das threads
- Todo programa tem pelo menos uma thread, que é a da função main()
- A documentação para threads no CibiOS está em http://chibiforge.org/doc/21.11/rt/group__threads.html

Kernel RT - Threads



Estados de uma thread

- Uma thread pode estar em um dos seguintes estados:
  - Prontidão: a thread está pronta para continuar execução, mas ainda não chegou sua vez
  - Executando: a thread está com a CPU executando suas instruções
  - Esperando: a thread está esperando algum evento acontecer para poder executar

Criação de uma thread

- Antes de criarmos uma thread, devemos definir duas coisas:
  - A função a ser executada
  - Uma área de trabalho para a thread (inclui sua pilha)

Criação de uma thread

- Antes de criarmos uma thread, devemos definir duas coisas:
  - A função a ser executada
  - Uma área de trabalho para a thread (inclui sua pilha)
- A função deve ser definida da seguintes forma:
  - `static THD_FUNCTION(nome_da_funcao, arg) {}`
  - O argumento arg é do tipo `void*`

Criação de uma thread

- Antes de criarmos uma thread, devemos definir duas coisas:
  - A função a ser executada
  - Uma área de trabalho para a thread (inclui sua pilha)
- A função deve ser definida da seguintes forma:
  - `static THD_FUNCTION(nome_da_funcao, arg) {}`
  - O argumento arg é do tipo `void*`
- A área de trabalho deve ser definida da seguinte forma:
  - `static THD_WORKING_AREA(nome_area, tamanho_area);`
  - O tamanho da área é em bytes

Criação de uma thread

- Após definidas a função e a área de trabalho, podemos criar e iniciar a thread através da função `chThdCreateStatic()`:
  - `chThdCreateStatic(area_trabalho, sizeof(area_trabalho), prioridade, nome_thd, argumento);`

Criação de uma thread

- Após definidas a função e a área de trabalho, podemos criar e iniciar a thread através da função `chThdCreateStatic()`:
  - `chThdCreateStatic(area_trabalho, sizeof(area_trabalho), prioridade, nome_thd, argumento);`
- A função `chThdCreateStatic()` retorna uma variável do tipo `thread_t*` apontando para a thread recém-criada.

Criação de uma thread

- Após definidas a função e a área de trabalho, podemos criar e iniciar a thread através da função `chThdCreateStatic()`:
  - `chThdCreateStatic(area_trabalho, sizeof(area_trabalho), prioridade, nome_thd, argumento);`
- A função `chThdCreateStatic()` retorna uma variável do tipo `thread_t*` apontando para a thread recém-criada.
- A prioridade é um valor entre 2 e 255, onde quanto maior o valor, maior é a prioridade.
  - `NORMALPRIO` é a prioridade da thread da função `main()`

Kernel RT - Threads

```
static THD_WORKING_AREA(wa_blinky, 128);
static THD_FUNCTION(blinky, arg) {
    while (1) {
        palTogglePad(LED_PORT, LED_PIN);
        chThdSleepMilliseconds(LED_PERIODO/2);
    }
}

int main(void) {
    halInit();
    chSysInit();

    palSetPadMode(LED_PORT, LED_PIN, PAL_MODE_OUTPUT_PUSHPULL);
    palClearPad(LED_PORT, LED_PIN);

    chThdCreateStatic(wa_blinky, sizeof(wa_blinky), NORMALPRIO + 1,
                      blinky, NULL);

    while (1) {}
}
```

Exemplo

- Acima temos um programa que usa uma thread para piscar o led da placa.

Kernel RT - Threads

```
static THD_WORKING_AREA(wa_blinky, 128);
static THD_FUNCTION(blinky, arg) {
    while (1) {
        palTogglePad(LED_PORT, LED_PIN);
        chThdSleepMilliseconds(LED_PERIODO/2);
    }
}

int main(void) {
    halInit();
    chSysInit();

    palSetPadMode(LED_PORT, LED_PIN, PAL_MODE_OUTPUT_PUSHPULL);
    palClearPad(LED_PORT, LED_PIN);

    chThdCreateStatic(wa_blinky, sizeof(wa_blinky), NORMALPRIO + 1,
                      blinky, NULL);

    while (1) {}
}
```

Exemplo

- Acima temos um programa que usa uma thread para piscar o led da placa.
- Observe o uso da função `chThdSleepMilliseconds()` para parar a thread por um certo intervalo de tempo.

Criação de uma thread

- Como sugestão, modifique o programa anterior para ter 2 threads piscando o mesmo led, uma com um período de 5 vezes um valor-base, e outra 7 vezes esse valor.

Criação de uma thread

- Após definidas a função e a área de trabalho, podemos criar e iniciar a thread através da função `chThdCreateStatic()`:
  - Isto é apenas um convite para encerrar, e não um encerramento incondicional.

Criação de uma thread

- Após definidas a função e a área de trabalho, podemos criar e iniciar a thread através da função `chThdCreateStatic()`:
  - Isto é apenas um convite para encerrar, e não um encerramento incondicional.
- A thread deve usar a função `chThdShouldTerminateX()` para saber se alguém pediu para ela encerrar.

Criação de uma thread

- Após definidas a função e a área de trabalho, podemos criar e iniciar a thread através da função `chThdCreateStatic()`:
  - Isto é apenas um convite para encerrar, e não um encerramento incondicional.
- A thread deve usar a função `chThdShouldTerminateX()` para saber se alguém pediu para ela encerrar.
- A thread que solicitou o encerramento pode esperar pelo término da thread passando um ponteiro para ela para a função `chThdWait()`.

Exemplo

- O programa ao lado usa este sincronismo para encerrar a execução da thread do led após 5 segundos

```
static THD_FUNCTION(blinky, arg) {
    while (!chThdShouldTerminateX()) {
        chThdSleepMilliseconds(LED_PERIODO/2);
        palTogglePad(LED_PORT, LED_PIN);
    }

    chThdExit(0);
}

int main(void) {
    thread_t* thd = 0;

    halInit();
    chSysInit();

    palSetPadMode(LED_PORT, LED_PIN, PAL_MODE_OUTPUT_PUSHPULL);
    palClearPad(LED_PORT, LED_PIN);

    thd = chThdCreateStatic(wa_blinky, sizeof(wa_blinky),
        NORMALPRIO + 1, blinky, NULL);

    chThdSleepMilliseconds(4999);
    chThdTerminate(thd);
    chThdWait(thd);

    while (1) {}
}
```

Criação de uma thread

- Esse sincronismo entre threads se aplica apenas ao início e término de uma thread.
  - Semáforos e mutexes oferecem mecanismos de sincronismo mais detalhados.

Criação de uma thread

- Esse sincronismo entre threads se aplica apenas ao início e término de uma thread.
  - Semáforos e mutexes oferecem mecanismos de sincronismo mais detalhados.
- ChibiOS tem variações das funções já vistas (veja documentação)
  - `chThdSleep()`, `chThdSleepSeconds()`, `chThdSleepUntil()` e outras variantes

Criação de uma thread

- Esse sincronismo entre threads se aplica apenas ao início e término de uma thread.
  - Semáforos e mutexes oferecem mecanismos de sincronismo mais detalhados.
- ChibiOS tem variações das funções já vistas (veja documentação)
  - `chThdSleep()`, `chThdSleepSeconds()`, `chThdSleepUntil()` e outras variantes
- Outras funções ocasionalmente importantes são:
  - `chThdSetPriority()`: muda a prioridade da própria thread
  - `chThdGetPriorityX()`: retorna a prioridade atual da thread
  - `chThdGetSelfX()`: retorna um ponteiro para a própria thread

Kernel RT — temporizadores virtuais

Introdução

- O ChibiOS oferece um tipo de temporizador baseado no seu tick periódico.

Kernel RT — temporizadores virtuais

Introdução

- O ChibiOS oferece um tipo de temporizador baseado no seu tick periódico.
- A resolução de tempo é da ordem de milisegundos, ao invés de microsegundos do GPT.

Introdução

- O ChibiOS oferece um tipo de temporizador baseado no seu tick periódico.
- A resolução de tempo é da ordem de milisegundos, ao invés de microsegundos do GPT.
- Outra diferença importante é que a quantidade de temporizadores virtuais não é limitada.

Introdução

- O ChibiOS oferece um tipo de temporizador baseado no seu tick periódico.
- A resolução de tempo é da ordem de milisegundos, ao invés de microsegundos do GPT.
- Outra diferença importante é que a quantidade de temporizadores virtuais não é limitada.
- Uma desvantagem é que os temporizadores são apenas do tipo *one-shot*.

Introdução

- O ChibiOS oferece um tipo de temporizador baseado no seu tick periódico.
- A resolução de tempo é da ordem de milisegundos, ao invés de microsegundos do GPT.
- Outra diferença importante é que a quantidade de temporizadores virtuais não é limitada.
- Uma desvantagem é que os temporizadores são apenas do tipo *one-shot*.
- A documentação sobre temporizadores virtuais encontra-se em `http://chibiforge.org/doc/21.11/rt/group__time.html`.

Operação

- O uso básico consiste em:
  - Inicializar uma variável de temporizador virtual do tipo `virtual_timer_t` usando a função `chVTObjectInit()`.

Operação

- O uso básico consiste em:
  - Inicializar uma variável de temporizador virtual do tipo `virtual_timer_t` usando a função `chVTObjectInit()`.
  - Chamar `chVTSet()` para iniciar a contagem do timer e informar a função de callback a ser chamada.

Operação

- O uso básico consiste em:
  - Inicializar uma variável de temporizador virtual do tipo `virtual_timer_t` usando a função `chVTObjectInit()`.
  - Chamar `chVTSet()` para iniciar a contagem do timer e informar a função de callback a ser chamada.
- Caso desejemos parar o temporizador antes de sua conclusão, chamamos a função `chVTReset()`.

Kernel RT — temporizadores virtuais

```
void chVTObjectInit(virtual_timer_t *vtp);
void chVTReset(virtual_timer_t *vtp);
void chVTSet(virtual_timer_t *vtp, sysinterval_t delay,
    vtfunc_t vtfunc, void *par);
systime_t chVTGetSystemTime(void);
bool chVTIsSystemTimeWithin(systime_t start, systime_t end);
```

Exemplo

- As assinaturas dessas funções estão acima.

Operação

- Os parâmetros da função chVTSet() são:
  - um ponteiro para um objeto do tipo `virtual_timer_t`.
  - o delay em unidades de ticks.
  - um ponteiro para a função callback com a assinatura `void (*vtfunc_t)(void *)`.
  - um ponteiro para dados (pode ser nulo).

Kernel RT — temporizadores virtuais

Operação

- As versões para uso em callbacks tem um “I” ao final do nome.
    - Por exemplo, `chVTSetI()`.

Unidade de tempo

- O parâmetro de intervalo de tempo passado para setar o temporizador tem unidades de ticks, e não de segundos.

Unidade de tempo

- O parâmetro de intervalo de tempo passado para setar o temporizador tem unidades de ticks, e não de segundos.
- Em geral usamos funções para conversão entre segundos e ticks:
  - TIME_MS2I() e TIME_S2I(), para converter de tempo para ticks

Unidade de tempo

- O parâmetro de intervalo de tempo passado para setar o temporizador tem unidades de ticks, e não de segundos.
- Em geral usamos funções para conversão entre segundos e ticks:
  - TIME_MS2I() e TIME_S2I(), para converter de tempo para ticks
  - TIME_I2S() e TIME_I2MS() para converter de ticks para tempo

Kernel RT — temporizadores virtuais

Observações

- A função de callback é chamada dentro do contexto de uma interrupção.

Observações

- A função de callback é chamada dentro do contexto de uma interrupção.
- Assim, se quisermos rearmar o temporizador, devemos usar a função chVTSetI() na função de callback.

Exemplo

- Um exemplo do uso de temporizadores virtuais é para acender e apagar o led da placa.

```
void vt_cb(void *arg) {
    chSysLockFromISR();
    palTogglePad(LED_PORT, LED_PIN);
    chVTSetI((virtual_timer_t*) arg, LED_PERIODO/2, vt_cb, arg);
    chSysUnlockFromISR();
}

int main(void) {
    virtual_timer_t vt;

    halInit();
    chSysInit();

    palSetPadMode(LED_PORT, LED_PIN, PAL_MODE_OUTPUT_PUSHPULL);
    palClearPad(LED_PORT, LED_PIN);

    chVTObjectInit(&vt);
    chVTSet(&vt, TIME_MS2I(LED_PERIODO/2), vt_cb, (void*) &vt);

    while (1) {}
}
```