Desenvolv. de Sistemas Embarcados em Tempo Real

Prof. Hermano Cabral

Departamento de Eletrônica e Sistemas — UFPE

22 de agosto de 2024

Plano de Aula

Tema central

- ChibiOS — sistema operacional

Plano de Aula

Tema central

- ChibiOS — sistema operacional

Objetivos

- Conhecer os mecanismos de sincronização de mutex, variáveis de condição e semáforos 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
  - Mutexes, semáforos, variaveis de condição e eventos

Introdução

- Mutexes são um tipo de mecanismo de sincronismo usado para "proteger" uma determinada região de código.

Introdução

- Mutexes são um tipo de mecanismo de sincronismo usado para "proteger" uma determinada região de código.
- Ele garante o acesso exclusivo à região de código sob sua guarda
  - Se duas threads tentarem obter acesso ao mesmo tempo, o mutex garante que apenas uma é bem sucedida

Operação

- No ChibiOS, o procedimento para usar um mutex é:
  - Criar uma variável do tipo apropriado e inicializá-la.
  - Travar o mutex antes de entrar na região crítica.
  - Destravá-lo ao final da região crítica.

Operação

- Para criarmos um mutex, usamos a macro `MUTEX_DECL(name)`
  - Isto cria a variável name do tipo `mutex_t` inicializada de modo apropriado

Operação

- Para criarmos um mutex, usamos a macro `MUTEX_DECL(name)`
  - Isto cria a variável name do tipo `mutex_t` inicializada de modo apropriado
  - A função `chMtxObjectInit(mutex_t *mp)` também pode ser usada para inicializar um mutex

Operação

- Para obter e liberar um mutex usamos as funções `chMtxLock(mutex_t *mp)` e `chMtxUnlock(mutex_t *mp)`.

Operação

- Para obter e liberar um mutex usamos as funções `chMtxLock(mutex_t *mp)` e `chMtxUnlock(mutex_t *mp)`.
- A função chMtxLock() só retorna quando a thread for bem sucedida em obter o mutex
  - Se pelo menos uma thread estiver tentando obter o mutex, exatamente uma será bem sucedida.

Operação

- Para obter e liberar um mutex usamos as funções `chMtxLock(mutex_t *mp)` e `chMtxUnlock(mutex_t *mp)`.
- A função chMtxLock() só retorna quando a thread for bem sucedida em obter o mutex
  - Se pelo menos uma thread estiver tentando obter o mutex, exatamente uma será bem sucedida.
- Uma outra função para se obter o mutex é `chMtxTryLock(mutex_t *mp)`.
  - Essa função retorna de imediato um valor booleano para indicar o sucesso ou não em obter o mutex.

Kernel RT - Mutex

```
void chMtxLock(mutex_t *mp);
void chMtxUnlock(mutex_t *mp);
bool chMtxTryLock(mutex_t *mp);

void chMtxLockS(mutex_t *mp);
void chMtxUnlockS(mutex_t *mp);
bool chMtxTryLockS(mutex_t *mp);

void chMtxUnlockAll(void);
void chMtxUnlockAllS(void);
void chMtxObjectInit(mutex_t *mp);
```

Operação

- As assinaturas das funções são as acima.

Observações

- Nenhuma dessas funções deve ser usada no contexto de uma interrupção.
  - Isso significa que não devemos usar mutexes em callbacks do ChibiOS.

Observações

- Nenhuma dessas funções deve ser usada no contexto de uma interrupção.
  - Isso significa que não devemos usar mutexes em callbacks do ChibiOS.
- Para o desempenho, é importante que a região crítica seja o mais curta possível

Exemplo

- Um exemplo de uso de mutex é a reserva de uso de uma interface de comunicação, para evitar problemas como o do lado.

```
static THD_FUNCTION(send1, arg) {
    uint8_t contador = 0;
    int i;

    while (!chThdShouldTerminateX()) {
        for (i=0; i<10; i++) {
            sdPut(&SD1, contador);
            contador += 2;
        }
        chThdSleepMilliseconds(3);

        for (i=0; i<10; i++) {
            sdPut(&SD1, contador);
            contador += 2;
        }
        chThdSleepMilliseconds(2);
    }

    chThdExit(0);
}
```

Exemplo

- Um exemplo de uso de mutex é a reserva de uso de uma interface de comunicação, para evitar problemas como o do lado.
- Existem duas threads como as do lado, ambas tentando enviar blocos de 10 bytes ao mesmo tempo.

```
static THD_FUNCTION(send1, arg) {
    uint8_t contador = 0;
    int i;

    while (!chThdShouldTerminateX()) {
        for (i=0; i<10; i++) {
            sdPut(&SD1, contador);
            contador += 2;
        }
        chThdSleepMilliseconds(3);

        for (i=0; i<10; i++) {
            sdPut(&SD1, contador);
            contador += 2;
        }
        chThdSleepMilliseconds(2);
    }

    chThdExit(0);
}
```

```
f5 ec f7 ee f9 fb fd f0 ff f2 01 f4 03 f6 f8 fa
fc 05 fe 07 00 09 02 0b 0d 0f 11 13 04 15 06 17
08 0a 0c 0e 10 12 19 14 1b 16 1d 1f 21 23 18 25
1a 27 1c 29 1e 2b 20 22 24 2d 26 2f 28 31 2a 33
35 37 39 3b 3d 2c 3f 2e 30 32 34 41 43 36 45 38
47 3a 49 3c 4b 3e 4d 4f 51 53 40 42 44 46 55 48
```

Exemplo

- O resultado do código anterior pode ser visto acima.

Kernel RT - Mutex

Exemplo

- Com um mutex, garantimos que os blocos de bytes transmitidos estarão em ordem.

```
static THD_FUNCTION(send1, arg) {
    uint8_t contador = 0;
    int i;

    while (!chThdShouldTerminateX()) {
        chMtxLock(&serial_mtx);
        for (i=0; i<10; i++) {
            sdPut(&SD1, contador);
            contador += 2;
        }
        chMtxUnlock(&serial_mtx);
        chThdSleepMilliseconds(3);

        chMtxLock(&serial_mtx);
        for (i=0; i<10; i++) {
            sdPut(&SD1, contador);
            contador += 2;
        }
        chMtxUnlock(&serial_mtx);
        chThdSleepMilliseconds(2);
    }

    chThdExit(0);
}
```

```
f1 f3 f5 f7 f9 fb fd ff 01 03 f0 f2 f4 f6 f8 fa
fc fe 00 02 05 07 09 0b 0d 0f 11 13 15 17 04 06
08 0a 0c 0e 10 12 14 16 19 1b 1d 1f 21 23 25 27
29 2b 18 1a 1c 1e 20 22 24 26 28 2a 2d 2f 31 33
35 37 39 3b 3d 3f 2c 2e 30 32 34 36 38 3a 3c 3e
41 43 45 47 49 4b 4d 4f 51 53 40 42 44 46 48 4a
```

Exemplo

- Os bytes transmitidos agora estão em ordem, como pode ser visto acima.

Introdução

- Mutexes são muito usados em conjunção com variáveis de condição.

Introdução

- Mutexes são muito usados em conjunção com variáveis de condição.
- O par variável de condição e mutex combinam a proteção do mutex com uma condição lógica e a sua sinalização pela variável de condição.

Operação

- De forma similar ao mutex, para criarmos uma variável de condição usamos a macro `CONDVAR_DECL(name)`.
  - Isto cria a variável name do tipo `condition_variable_t` inicializada de modo apropriado.
  - A função `chCondObjectInit(condition_variable_t *mp)` também pode ser usada para inicializar uma variável de condição.

Kernel RT — Variáveis de condição

Operação

- As principais funções são a `chCondWait(condition_variable_t *mp)` e a `chCondSignal(condition_variable_t *mp)`.

Kernel RT — Variáveis de condição

Operação

- As principais funções são a `chCondWait(condition_variable_t *mp)` e a `chCondSignal(condition_variable_t *mp)`.
- A função `chCondWait()` libera o último mutex adquirido pela thread e espera que a função `chCondSignal()` seja chamada por outra thread, quando então readquire o mutex
  - Ela só retorna quando a thread for sinalizada
  - A função `chCondSignal()` age como uma interrupção

Kernel RT — Variáveis de condição

```
msg_t chCondWait(condition_variable_t *cp);
msg_t chCondWaitTimeout(condition_variable_t *cp,
    sysinterval_t timeout);
void chCondSignal(condition_variable_t *cp);
void chCondObjectInit(condition_variable_t *cp);

msg_t chCondWaitS(condition_variable_t *cp);
msg_t chCondWaitTimeoutS(condition_variable_t *cp,
    sysinterval_t timeout);
void chCondSignalI(condition_variable_t *cp);
```

Operação

- As assinaturas das funções são as mostradas acima.

Kernel RT — Variáveis de condição

```
chMtxLock(&mtx);

/* Código opcional protegido por mutex */

while (!condition)
    chCondWait(&cond1);

/* Código da região crítica com condição
   satisfeita */

chMtxUnlock();
```

Operação

- O código acima é o uso típico de variáveis de condição.

Exemplo

- Um exemplo de uso de variável de condição é um buffer onde várias threads podem armazenar ou retirar elementos.

```
void queue_write(elem_t e) {
    chMtxLock(&_qmtx);

    while (_qsize >= QUEUE_SIZE)
        chCondWait(&_queue_full);

    *wr = e;
    if (++wr >= &_queue[QUEUE_SIZE])
        wr = &_queue[0];
    _qsize++;

    chCondSignal(&_queue_empty);

    chMtxUnlock(&_qmtx);
}
```

Exemplo

- Um exemplo de uso de variável de condição é um buffer onde várias threads podem armazenar ou retirar elementos.
- A função ao lado é usada para armazenar elementos no buffer.

```
void queue_write(elem_t e) {
    chMtxLock(&_qmtx);

    while (_qsize >= QUEUE_SIZE)
        chCondWait(&_queue_full);

    *wr = e;
    if (++wr >= &_queue[QUEUE_SIZE])
        wr = &_queue[0];
    _qsize++;

    chCondSignal(&_queue_empty);

    chMtxUnlock(&_qmtx);
}
```

Kernel RT — Variáveis de condição

Exemplo

- A função complementar para retirar elementos do buffer é mostrada ao lado.

```
elem_t queue_read(void) {
    elem_t e;

    chMtxLock(&_qmtx);

    while (_qsize == 0)
        chCondWait(&_queue_empty);

    e = *rd;
    if (++rd >= &_queue[QUEUE_SIZE])
        rd = &_queue[0];
    _qsize--;

    chCondSignal(&_queue_full);
    chMtxUnlock(&_qmtx);

    return e;
}
```

Introdução

- ChibiOS implementa semáforos do tipo contadores que podem ser sinalizados, aguardados e resetados.

Introdução

- ChibiOS implementa semáforos do tipo contadores que podem ser sinalizados, aguardados e resetados.
- Este tipo de semáforo é inicializado com um valor positivo.

Introdução

- ChibiOS implementa semáforos do tipo contadores que podem ser sinalizados, aguardados e resetados.
- Este tipo de semáforo é inicializado com um valor positivo.
- Cada vez que uma thread tenta adquirir o semáforo, é verificado se o contador é positivo:
  - Se for, a thread é permitida continuar execução

Introdução

- ChibiOS implementa semáforos do tipo contadores que podem ser sinalizados, aguardados e resetados.
- Este tipo de semáforo é inicializado com um valor positivo.
- Cada vez que uma thread tenta adquirir o semáforo, é verificado se o contador é positivo:
  - Se for, a thread é permitida continuar execução
  - Se não, a thread é colocada para dormir aguardando o contador ficar positivo novamente

Introdução

- Se um semáforo for sinalizado, seu contador é incrementado:

Introdução

- Se um semáforo for sinalizado, seu contador é incrementado:
  - Se o contador não é positivo, pelo menos uma thread está aguardando o semáforo, e uma delas será escolhida para ser ativada.

Operação

- A operação básica com um semáforo consiste na sua declaração com um valor inicial para o contador através da função `SEMAPHORE_DECL(nome, n)`.

Operação

- A operação básica com um semáforo consiste na sua declaração com um valor inicial para o contador através da função `SEMAPHORE_DECL(nome, n)`.
- Em seguida, qualquer thread que deseje adquirir o semáforo executa `chSemWait()` ou `chSemWaitTimeout()`.

Operação

- A operação básica com um semáforo consiste na sua declaração com um valor inicial para o contador através da função `SEMAPHORE_DECL(nome, n)`.
- Em seguida, qualquer thread que deseje adquirir o semáforo executa `chSemWait()` ou `chSemWaitTimeout()`.
- Se o valor de retorno for `MSG_OK`, a espera foi bem sucedida.

Operação

- A operação básica com um semáforo consiste na sua declaração com um valor inicial para o contador através da função `SEMAPHORE_DECL(nome, n)`.
- Em seguida, qualquer thread que deseje adquirir o semáforo executa `chSemWait()` ou `chSemWaitTimeout()`.
- Se o valor de retorno for `MSG_OK`, a espera foi bem sucedida.
- Para sinalizar (ou liberar) o semáforo, a thread executa `chSemSignal()`.

Kernel RT — Variáveis de condição

```
msg_t chSemWait(semaphore_t *sp);
msg_t chSemWaitTimeout(semaphore_t *sp, sysinterval_t timeout);
void chSemSignal(semaphore_t *sp);

void chSemObjectInit(semaphore_t *sp, cnt_t n);
msg_t chSemWaitS(semaphore_t *sp);
msg_t chSemWaitTimeoutS(semaphore_t *sp, sysinterval_t timeout);
void chSemSignalI(semaphore_t *sp);
```

Operação

- As assinaturas das funções são as mostradas acima.

Operação

- Podemos também resetar o semáforo através da função `chSemReset()`.

Operação

- Podemos também resetar o semáforo através da função `chSemReset()`.
- Neste caso, as threads que estiverem esperando serão ativadas, porém recebendo o valor de retorno `MSG_RESET`.

Observações

- Não se deve usar a função `semWait()` em uma interrupção nem em callbacks.

Observações

- Não se deve usar a função `semWait()` em uma interrupção nem em callbacks.
- Por outro lado, a função semSignall() pode ser usada em interrupções e callbacks.