LABORATÓRIO RTOS

Prática 04: Comunicação entre Tasks

Prof. Francisco Helder

20 de maio de 2025

1 Comunicação entre Tasks

Uma QUEUE pode conter um número finito de dados de tamanho fixo. O número máximo de dados que uma QUEUE pode conter é chamado de "comprimento". Tanto o comprimento quanto o tamanho de cada dados são definidos quando a QUEUE é criada.

Normalmente, as QUEUES são usadas como buffers First In First Out (FIFO), onde os dados são gravados no final da QUEUE e removidos no começo da QUEUE. Também é possível gravar no começo de uma QUEUE.

Gravar dados em uma QUEUE faz com que uma cópia byte por byte dos dados seja armazenada na própria QUEUE. Ler dados de uma QUEUE faz com que a cópia dos dados seja removida da QUEUE. A Figura 1 demonstra dados sendo gravados e lidos de uma QUEUE e o efeito sobre os dados armazenados na QUEUE de cada operação.

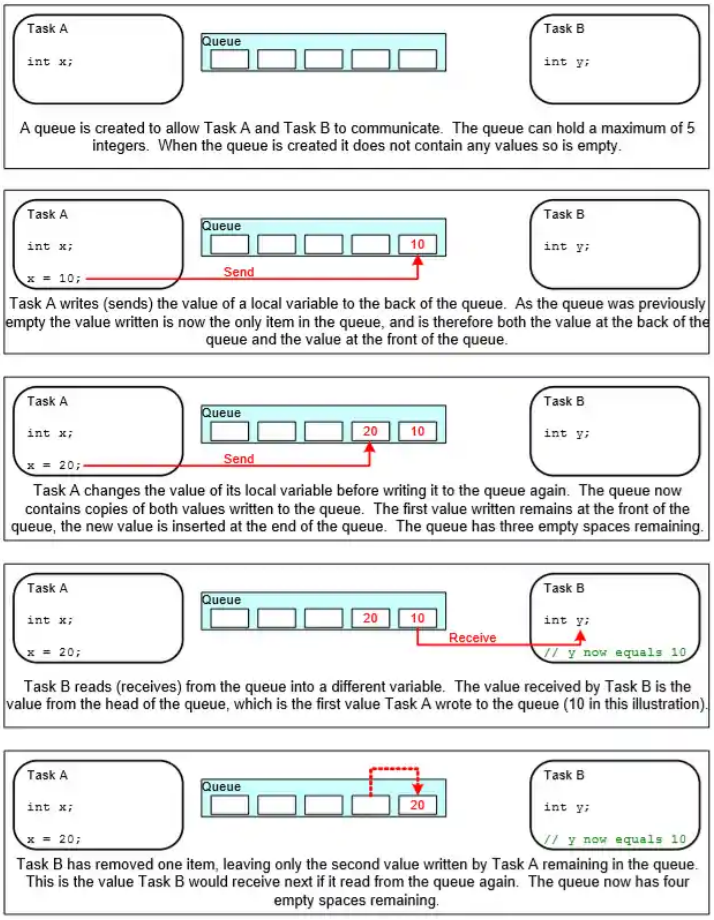

Figura 1: Um exemplo de sequência de gravações e leituras na QUEUE.

QUEUES são objetos com características próprias, e que não são de propriedade ou atribuídos de nenhuma Task específica. Qualquer Task pode gravar na mesma QUEUE e qualquer Task pode ler da mesma QUEUE. Uma QUEUE com vários escritores é muito comum, enquanto uma QUEUE com vários leitores é bastante rara.

1.1 Bloqueio em Leituras de QUEUES

Quando uma Task tenta ler de uma QUEUE, ela pode opcionalmente especificar um tempo de "bloqueio". Este é o tempo que a Task deve ser mantida no estado BLOCKED para aguardar

que os dados estejam disponíveis na QUEUE, caso a QUEUE já esteja vazia. Uma Task que está no estado BLOCKED aguardando que os dados fiquem disponíveis em uma QUEUE é automaticamente movida para o estado READY quando outra Task ou interrupção coloca dados na QUEUE. A Task também será automaticamente movida do estado BLOCKED para o estado READY se o tempo de bloqueio especificado expirar antes que os dados fiquem disponíveis.

As QUEUES podem ter vários leitores, então é possível que uma única QUEUE tenha mais de uma Task bloqueada aguardando dados. Quando este for o caso, apenas uma Task será desbloqueada quando os dados ficarem disponíveis. A Task que for desbloqueada será sempre a Task de maior prioridade que estava aguardando dados. Se as Tasks bloqueadas tiverem prioridade igual, será a Task que estiver aguardando dados por mais tempo que será desbloqueada.

1.2 Bloqueio na Escrita da QUEUE

Assim como ao ler de uma QUEUE, uma Task pode opcionalmente especificar um tempo de bloqueio ao gravar em uma QUEUE. Nesse caso, o tempo de bloqueio é o tempo máximo que a Task deve ser mantida no estado BLOCKED para esperar que o espaço fique disponível na QUEUE, caso ela já esteja cheia.

As QUEUES podem ter vários escritores, então é possível que uma QUEUE cheia tenha mais de uma Task bloqueada esperando para concluir uma operação de envio. Quando esse for o caso, apenas uma Task será desbloqueada quando o espaço na QUEUE ficar disponível. A Task que for desbloqueada sempre será a Task de maior prioridade que estava esperando por espaço. Se as Tasks bloqueadas tiverem prioridade igual, será a Task que estiver esperando por espaço por mais tempo que será desbloqueada.

2 Prática de Comunicação entre Tasks

Nosse prática iremos iniciar duas Tasks, uma para acender o led e outra para ler dois botões SW1 e SW2. Quando o botão SW1 for pressionado, o led deverá acender. Quando o botão SW2 for pressionado, o led deverá ser apagado. Estas Tasks deverão se comunicar via QUEUE. Aproveite o projeto das práticas anteriores e utilize na prática de QUEUE. No arquivo main.c adicione o include das funções de QUEUE do FreeRTOS:

```
#include "queue.h"
```

Crie uma estrutura para armazenar as informações dos leds. Usaremos esta estrutura como elemento do queue para as Tasks dos leds e dos botões se comunicarem:

```
1 typedef struct {
2   unsigned int num;
3   unsigned int status;
4 } LedInfo;
```

Reimplemente a Task que irá manipular o led. Esta Task deverá simplesmente esperar um elemento do QUEUE, e quando receber este elemento, deverá acionar o led de acordo com as informações recebidas:

```
void taskLed(void *pvParameters){
  xQueueHandle ledQueue = (xQueueHandle) pvParameters;
  LedInfo led;
  for (;;) {
    xQueueReceive(ledQueue, &led, portMAX_DELAY);
    drvLedsSet(led.num, led.status);
  }
}
```

Crie agora a Task que irá manipular os botões. Esta Task será responsável por ler os botões SW1 e SW2 a cada 100ms. Se o botão SW1 for pressionado, ela irá enviar uma mensagem via QUEUE para acender o led. Se o botão SW2 for pressionado, ela irá enviar uma mensagem via QUEUE para apagar o led.

```
void taskButton(void *pvParameters){

  xQueueHandle ledQueue = (xQueueHandle) pvParameters;
  LedInfo led;
  led.status = DRV_LEDS_OFF;
  led.num = DRV_LEDS_ORANGE;

  for (;;) {
    if (drvButtonsSW1Pressed() && led.status == DRV_LEDS_OFF) {
      led.status = DRV_LEDS_ON;
      xQueueSendToBack(ledQueue, &led, portMAX_DELAY);
    }else if (drvButtonsSW2Pressed() && led.status == DRV_LEDS_ON) {
      led.status = DRV_LEDS_OFF;
      xQueueSendToBack(ledQueue, &led, portMAX_DELAY);
    }

    vTaskDelay(100/portTICK_RATE_MS);
  }
}
```

Por último, crie a função main() para inicializar o hardware, criar o QUEUE e as duas Tasks, então compile e teste:

```
int main(void){

  xQueueHandle ledQueue;

  /* init hardware */
  drvCpuClockInit();
  drvLedsInit();
  drvButtonsInit();

  /* init led queue */
  ledQueue = xQueueCreate(1, sizeof(LedInfo));

  /* create leds task */
  xTaskCreate(taskLed, (signed char *)"TaskLed",
        configMINIMAL_STACK_SIZE, (void *)ledQueue, 1, NULL);

  /* create buttons task */
  xTaskCreate(taskButton, (signed char *)"TaskButton",
        configMINIMAL_STACK_SIZE, (void *)ledQueue, 1, NULL);

  /* start the scheduler */
  vTaskStartScheduler();

  /* should never reach here! */
  for(;;);
}
```

pratica 1:

Nesta atividade vamos criar uma Task para ler o Botão e acionar um led em outra Task. A Task do botão deve ser acionado por interrupção

pratica 2:

Crear uma Task de leitura do ADC e comunicar, via QUEUE, com uma Task de acionamento do led. O seu ADC deve está conectado à um potenciômetro.

pratica 3:

Cria uma Task para ler o ADC a cada 300ms, e três Task, uma TASK que acende o led AZUL se a leitura do ADC ultrapassar um certo valor X1 de tensão, outra Task que acende o led amarelo se a leitura for estiver entre X1 e X2, e terceira acende um led vermelho quando o valor ficar abaixo de X2. A comunicação entre as Tasks dos leds e a Task do ADC deverá ser via QUEUE, somente um led pode está ligado.