

LABORATÓRIO RTOS



Prática 04: Comunicação entre Tarefas

Prof. Francisco Helder

5 de julho de 2024

1 Comunicação entre Tarefas

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.

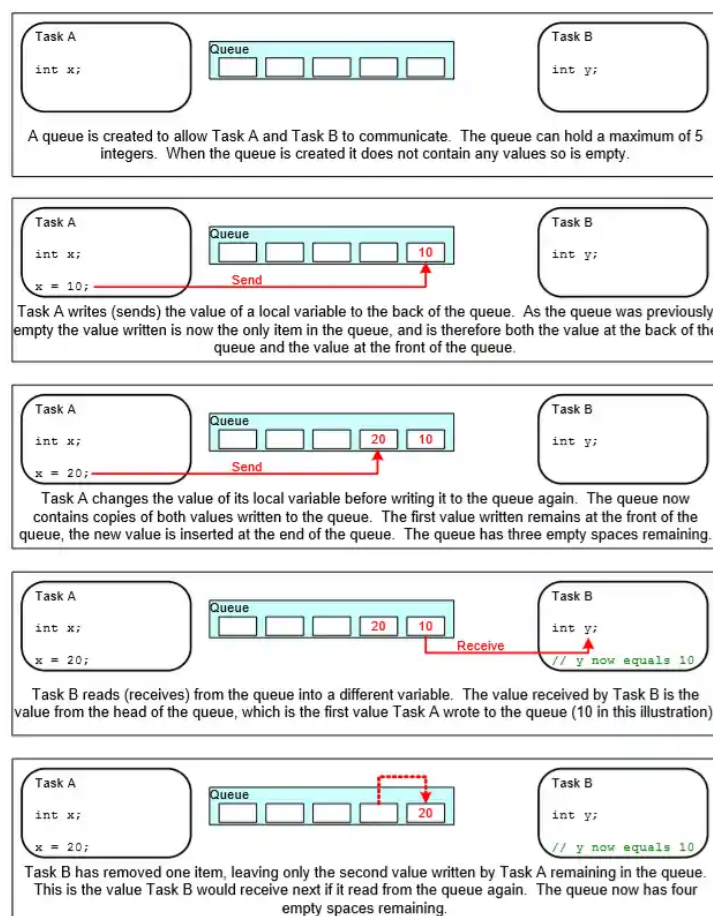


Figura 1: Um exemplo de sequência de gravações e leituras na QUEUE.

QUEUES são objetos com características próprias, e que não são de propriedade ou atribuídos de nenhuma tarefa específica. Qualquer tarefa pode gravar na mesma QUEUE e qualquer tarefa 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 tarefa tenta ler de uma QUEUE, ela pode opcionalmente especificar um tempo de “bloqueio”. Este é o tempo que a tarefa deve ser mantida no estado Blocked para aguardar

que os dados estejam disponíveis na QUEUE, caso a QUEUE já esteja vazia. Uma tarefa que está no estado Blocked aguardando que os dados fiquem disponíveis em uma QUEUE é automaticamente movida para o estado Ready quando outra tarefa ou interrupção coloca dados na QUEUE. A tarefa 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 tarefa bloqueada aguardando dados. Quando este for o caso, apenas uma tarefa será desbloqueada quando os dados ficarem disponíveis. A tarefa que for desbloqueada será sempre a tarefa de maior prioridade que estava aguardando dados. Se as tarefas bloqueadas tiverem prioridade igual, será a tarefa 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 tarefa pode opcionalmente especificar um tempo de bloqueio ao gravar em uma QUEUE. Nesse caso, o tempo de bloqueio é o tempo máximo que a tarefa 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 tarefa bloqueada esperando para concluir uma operação de envio. Quando esse for o caso, apenas uma tarefa será desbloqueada quando o espaço na QUEUE ficar disponível. A tarefa que for desbloqueada sempre será a tarefa de maior prioridade que estava esperando por espaço. Se as tarefas bloqueadas tiverem prioridade igual, será a tarefa que estiver esperando por espaço por mais tempo que será desbloqueada.

2 Prática de Comunicação entre Tarefas

Nossa prática iremos iniciar duas tarefas, 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 tarefas 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 tarefas dos leds e dos botões se comunicarem:

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

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

```
1 void taskLed(void *pvParameters){
2     xQueueHandle ledQueue = (xQueueHandle) pvParameters;
3     LedInfo led;
4     for (;;) {
5         xQueueReceive(ledQueue, &led, portMAX_DELAY);
6         drvLedsSet(led.num, led.status);
7     }
8 }
```

Crie agora a tarefa que irá manipular os botões. Esta tarefa 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.

```
1 void taskButton(void *pvParameters){
2
3     xQueueHandle ledQueue = (xQueueHandle) pvParameters;
4     LedInfo led;
5     led.status = DRV_LEDS_OFF;
6     led.num = DRV_LEDS_ORANGE;
7
8     for (;;) {
9         if (drvButtonsSW1Pressed() && led.status == DRV_LEDS_OFF) {
10             led.status = DRV_LEDS_ON;
11             xQueueSendToBack(ledQueue, &led, portMAX_DELAY);
12         } else if (drvButtonsSW2Pressed() && led.status == DRV_LEDS_ON) {
13             led.status = DRV_LEDS_OFF;
14             xQueueSendToBack(ledQueue, &led, portMAX_DELAY);
15         }
16
17         vTaskDelay(100/portTICK_RATE_MS);
18     }
19 }
```

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

```
1 int main(void){
2
3     xQueueHandle ledQueue;
4
5     /* init hardware */
6     drvCpuClockInit();
7     drvLedsInit();
8     drvButtonsInit();
9
10    /* init led queue */
11    ledQueue = xQueueCreate(1, sizeof(LedInfo));
12
13    /* create leds task */
14    xTaskCreate(taskLed, (signed char *)"TaskLed",
15               configMINIMAL_STACK_SIZE, (void *)ledQueue, 1, NULL);
16
17    /* create buttons task */
18    xTaskCreate(taskButton, (signed char *)"TaskButton",
19               configMINIMAL_STACK_SIZE, (void *)ledQueue, 1, NULL);
20
21    /* start the scheduler */
22    vTaskStartScheduler();
23
24    /* should never reach here! */
25    for(;;);
26 }
```

3 Atividade Prática

Nesta atividade vamos criar uma tarefa para ler o ADC e acionar um led dependendo do valor da leitura realizada. A comunicação entre a tarefa de leitura do ADC e a tarefa de acionamento do led será realizada através de um QUEUE. O seu ADC deve está conectado à um potenciômetro. Cria uma tarefa para ler o ADC a cada 300ms, acender o led AZUL se a leitura do ADC ultrapassar 20.000, e apagar o led AZUL se a leitura for menor ou igual à 20.000. A comunicação entre a tarefa do led e a tarefa do ADC deverá ser via QUEUE.