Relatório: Introduction to the use of FreeRTOS

Beatriz Silva - 85276

Diogo Vala – 64671

2.1

Quando a tarefa LedFlash tem uma prioridade mais alta que a tarefa InterfTask, é assegurada a execução da primeira, através de preempção, sendo a segunda apenas executada quando a primeira está idle. Ao trocar as prioridades, a tarefa LedFlash fica no estado ready durante a execução da tarefa InterfTask e apenas resume quando esta está idle.

2.2

O módulo vTaskDelay utilizado neste programa garante apenas periodicidade relativamente ao instante em que a função vTaskDelay foi chamada anteriormente. Isto significa que rotinas de interrupção, diferentes percursos no código e outras tarefas podem interferir com o tempo de execução da tarefa em curso e, por sua vez, o instante em que vTaskDelay é novamente chamada.

Uma alternativa mais robusta é a utilização do módulo vTaskDelayUntil, que garante que a tarefa é desbloqueada tendo em conta o instante de tempo em que foi previamente desbloqueada e não relativamente ao instante em que a função vTaskDelayUntil foi chamada. Para este efeito é utilizada uma variável que guarda o instante de tempo (em forma de ticks) da última ativação da tarefa, sendo o novo instante de ativação: $* pxPreviousWakeTime + xFrequency$. Esta função atualiza automaticamente o valor $pxPreviousWakeTime$ após cada chamada, garantindo uma periodicidade absoluta.

2.3

a)

Para a passagem de valores entre tarefas utilizaram-se duas variáveis globais, uma para guardar as amostras obtidas pela ADC na tarefa Acq, a serem passadas para a tarefa Proc, e outra para guardar a média calculada na tarefa Prof, a enviar para a tarefa Out. Para evitar condições de corrida à memória, escolheu-se utilizar dois semáforos binários.

A escolha de semáforos binários advém de serem ideais para uma simples sincronização entre tarefas, sem se ter de libertar o semáforo no fim da execução da tarefa, ao contrário dos mutexes. Isto permite que uma tarefa liberte um semáforo e que a outra tarefa obtenha esse semáforo, sem o libertar de novo.

Neste programa, o primeiro semáforo, xSemaphoreProc controla o acesso à variável que armazena as amostras. A tarefa Acq realiza a amostragem, armazena o valor na variável global e liberta o semáforo com xSemaphoreGive. Por sua vez, a tarefa Proc obtém o semáforo com xSemaphoreTake e acede à variável para a processar. Quando obtém cinco amostras, a tarefa Proc calcula a média, guarda-a na segunda variável global e liberta o segundo semáforo, xSemaphoreOut. A tarefa Out obtém este semáforo e imprime o valor da variável global.

A execução das três tarefas é feita em cascata, sendo a tarefa Acq periódica, através do uso de vTaskDelayUntil e estando as outras tarefas dependentes da execução desta. As duas tarefas subsequentes têm um intervalo de tempo de $100ms - t_{acq}$, onde t_{acq} é o tempo de execução da tarefa Acq, para executarem.

As prioridades estão definidas por ordem decrescente, tendo a tarefa Acq a maior prioridade, seguida da tarefa Proc e, finalmente, a tarefa Out.

É também importante mencionar que o tempo de espera da função xSemaphoreTake é infinito, através do uso de $portMAX\_DELAY$, que garante que as tarefas ficam perpetuamente bloqueadas enquanto o respetivo semáforo não for libertado.

b)

A utilização de Queues é semelhante à utilização de semáforos, do ponto de vista de organização do código, sendo a tarefa Acq periódica através de vTaskDelayUntil, como no ponto anterior. As Queues funcionam como buffers em RAM, onde são armazenadas variáveis, assim como o estado da Queue. São, assim, semelhantes a FIFOs, mas é também possível enviar uma variável para a frente da fila.

Esta ferramenta permite que as tarefas sejam executadas à medida que existem novas variáveis disponíveis para processamento.

No contexto deste programa, criaram-se duas Queues, uma para armazenar as amostras obtidas na tarefa Acq, a enviar à tarefa Proc, e outra para armazenar as médias calculadas na tarefa Prov, a enviar à tarefa Out. A primeira Queue foi criada com espaço para uma variável do tipo $uint16\_t$ e a segunda Queue foi criada com espaço para uma variável do tipo $xResult\_t$, que é uma estrutura que tem como conteúdo duas variáveis do tipo $uint16\_t$.

Na tarefa Acq, é realizada a amostragem da ADC e esse valor é copiado para a Queue xSamplesQueue, através da função xQueueSend. É feito um cast para ponteiro void ao ponteiro para a amostra. Isto é necessário porque as Queues são ferramentas capazes de enviar qualquer tipo de dados, logo estas são armazenadas com o tipo void.

Subsequentemente, a tarefa Proc realiza xQueueReceive e copia a variável da Queue xSamplesQueue para uma variável automática, sendo feito o cast de ponteiro void para ponteiro para uint16_t, de modo a preservar o tipo de variável em questão. O tempo de espera na função xQueueReceive é infinito, através de $portMAX\_DELAY$, ficando a tarefa indefinidamente à espera de uma nova variável. É de notar que a função xQueueReceive elimina a variável que estava a armazenada na Queue, ao contrário de, por exemplo, a função xQueuePeek, que apenas realiza uma cópia.

Quando, na tarefa Proc, são processadas cinco amostras, é realizada a média e a estrutura que armazena este valor, assim como o seu valor decimal é copiada para a Queue xAveragesQueue, de novo com cast para ponteiro void. Na tarefa Out, é feito o cast oposto e a estrutura é armazenada numa variável automática para ser impressa.

extra)

Como trabalho extra, realizou-se de novo o exercício com a utilização de TaskNotify. Esta ferramenta permite enviar uma notificação diretamente de uma tarefa para outra, sem utilizar um objeto intermédio, como nos pontos anteriores, o que permite otimizar a utilização de recursos em memória e ciclos de CPU, já que não é necessária memória para guardar o estado do objeto e não são necessários passos intermédios aquando da sua utilização, como desligar interrupções. A notificação é feita com a alteração direta de um valor no array de notificações a tarefas, que indica que a tarefa passa a estar pendente.

Para utilizar notificações a tarefas é necessário saber identificar a tarefa em questão. Para esse feito, atribui-se um Task Handle a cada tarefa a notificar, no momento de criação da mesma.

De forma análoga aos semáforos, são chamadas as funções xTaskNotifyGive e ulTaskNotifyTake para a sincronização entre as tarefas. É utilizada a função ulTaskNotifyTake por ser indicada como substituição direta de um semáforo binário.