LABORATÓRIO RTOS

Prática 07: Gerenciamento de Memória

Prof. Francisco Helder

23 de agosto de 2024

1 Gerenciamento de Mamória

O kernel tem que alocar RAM dinamicamente cada vez que uma tarefa, fila ou semáforo é criado. As funções da biblioteca padrão malloc() e free() podem ser usadas, mas também podem sofrer de um ou mais dos seguintes problemas:

1. Nem sempre estão disponíveis em pequenos sistemas embarcados;
2. Sua implementação pode ser relativamente grande, então ocupa espaço valioso no código;
3. Raramente são thread-safe;
4. Não são determinísticas, com quantidade de tempo necessária para executar as funções sendo diferente de chamada para chamada;
5. Podem sofrer de fragmentação de memória;
6. Podem complicar a configuração do vinculador.

Diferentes sistemas embarcados têm requisitos variados de alocação e temporização de RAM, então um único algoritmo de alocação de RAM só será apropriado para um subconjunto de aplicativos. O FreeRTOS, portanto, trata a alocação de memória como parte da camada adcional (em oposição a parte da base de código principal). Isso permite que aplicativos individuais forneçam sua implementação específica quando apropriado.

Quando o kernel requer RAM, em vez de chamar malloc() diretamente, ele chama pvPortMalloc(). Quando a RAM está sendo liberada, em vez de chamar free() diretamente, o kernel chama vPortFree(). pvPortMalloc() tem o mesmo protótipo que malloc(), e vPortFree() tem o mesmo protótipo que free(). O FreeRTOS vem com cinco implementações de exemplo de pvPortMalloc() e vPortFree(), todas documentadas na paǵina do freeRTOS.

Os cinco exemplos são definidos nos arquivos heap_1.c, heap_2.c e heap_3.c, heap_4.c e heap_5.c respectivamente – todos localizados no diretório FreeRTOS
Source
Portable
MemMang. O pool de memória original e o esquema de alocação de blocos usados por versões muito antigas do FreeRTOS foram removidos devido ao esforço e à compreensão necessários para dimensionar os blocos e pools.

É típico que pequenos sistemas embarcados criem apenas tarefas, filas e semáforos antes que o scheduler seja iniciado. Nesse caso, a memória só é alocada dinamicamente antes que o aplicativo comece a executar qualquer funcionalidade real em tempo real e, uma vez alocada, a memória nunca mais é liberada. Isso significa que o esquema de alocação escolhido não precisa considerar questões mais complexas, como determinismo e fragmentação, e pode considerar apenas atributos como tamanho do código e simplicidade.

2 Monitorando e Ajustando o Heap

pratica 1:

Nesta prática os alunos devem estudar as implementações heap_4.c e heap_5.c do FreeRTOS, e aprender a monitorar e ajustar o heap da aplicação.

pratica 2:

Crie uma tarefa para monitorar o estado do heap a cada 1 segundo com a função xPortGetFreeHeapSize(), enviar o tamanho livre do heap pela porta serial, e acender um led se o tamanho livre for menor que 10% do tamanho total.