

Curitiba, 19, fevereiro de 2024.

**Disciplina:** Sistemas Computacionais

**Professor:** Jhonatan Geremias

**Estudantes:** Gustavo Furini e Leonardo Nervino

## TDE I – TRABALHO DISCENTE EFETIVO

### Semáforos FreeRTOS

#### Descrição da Atividade:

Nesta atividade o estudante deverá efetuar a leitura do capítulo 4 do “Manual de Referência do FreeRTOS” disponibilizada no Canvas. Este manual foi disponibilizado na versão que está sendo utilizada em aula, o que vai auxiliar o desenvolvimento desta atividade. O capítulo 4 refere-se à utilização da estrutura Semáforos no FreeRTOS. O estudante deve diferenciar o uso de semáforos binários de semáforos mutexes. Posteriormente deverá implementar o exercício prático com a utilização do semáforo mutex conforme o roteiro da atividade.

#### Entrega:

O TDE1 deverá ser entregue até o dia **04/04/2024** no Canvas. O estudante deverá entregar um arquivo “.pdf” contendo as respostas do roteiro de atividade item 1. Para facilitar a implementação foi disponibilizado um projeto do Visual Studio para o TDE, sugiro fortemente que utilizem este projeto, pois vai facilitar o desenvolvimento. Os estudantes poderão realizar esta atividade em equipe, sendo até três integrantes em cada equipe. A equipe deverá implementar o código apenas dentro do arquivo “example.c”. Para entrega do arquivo do projeto encaminhar apenas o arquivo “example.c” onde foi codificado, este deve conter o nome dos estudantes e curso adicionados no cabeçalho do arquivo como comentário. Ambos os arquivos devem ser postados individualmente no Canvas até a data limite de entrega, atenção não compactar os arquivos.

#### Roteiro da Atividade:

1. Efetue a leitura do material de apoio (Manual de Referência do FreeRTOS v9.0.0 disponibilizado no Blackboard) e responda:
  - a. Responda com suas palavras para que serve a estrutura semáforo?

**R:** Um semáforo é uma estrutura de dados que tem como objetivo coordenar o acesso concorrente a recursos compartilhados por threads ou

processos. Ele garante que apenas uma entidade por vez possa acessar o recurso protegido, evitando assim conflitos de acesso conhecidos como condições de corrida.

b. Qual a diferença entre os semáforos binários e o mutexes?

**R:** A principal diferença entre semáforos binários e mutexes reside no fato de que os semáforos binários têm um valor máximo de 1, indicando se um recurso está disponível ou ocupado. Por outro lado, mutexes são semáforos binários específicos usados para garantir que apenas uma thread por vez possa acessar uma seção crítica de código, assegurando a exclusão mútua entre as threads.

c. Descreva para que é utilizado as funções `xSemaphoreTake()` e `xSemaphoreGive()`.

**R:** A função `xSemaphoreTake()` tem o objetivo de adquirir um semáforo. Se o semáforo estiver disponível (ou seja, se o contador for maior que zero), ele será reduzido e a função será concluída. Se o semáforo não estiver disponível, a função pode bloquear o chamador até que o semáforo esteja disponível. Por outro lado, a função `xSemaphoreGive()` é responsável por liberar um semáforo que foi previamente adquirido. Ela incrementa o contador do semáforo, indicando que o recurso protegido está novamente disponível para outros threads e processos.

## 2. Implemente um programa utilizando a estrutura semáforo no FreeRTOS conforme a especificação abaixo:

- Implementar um programa no FreeRTOS utilizando semáforos para controlar o uso de uma variável global, esta variável deve simular o uso do recurso de um display;
- Criar uma estrutura semáforo do tipo mutex, para tal utilizar a função `xSemaphoreCreateMutex()`;
- Deverá ser criado três tarefas diferentes, todas as tarefas devem escrever uma mensagem na variável display e posteriormente imprimir a mensagem do display na tela;
- Cada tarefa deve garantir o uso exclusivo do acesso a variável display utilizando a estrutura do semáforo, para isso utilizar as tarefas `xSemaphoreTake()` e `xSemaphoreGive()`;

- e. A criação das três tarefas deve ser realizada no main\_, todas as tarefas criadas devem ter a mesma prioridade, passar por parâmetro na criação da tarefa um número id que permite identificar a tarefa;
- f. A primeira tarefa (vTask1) deve imprimir no display a data atual obtida do sistema. O código abaixo permite obter a data atual do sistema, lembrando que é necessário importar a biblioteca "time.h":

```
//Obtendo o tempo em segundos
time(&segundos);
//Converter unidade segundos para o tempo local
datetimeNow = localtime(&segundos);
//Obtendo o dia
int dia = datetimeNow->tm_mday;
//Obtendo o mês
int mes = datetimeNow->tm_mon + 1;
//Obtendo o ano
int ano = datetimeNow->tm_year + 1900;
```

- g. A segunda tarefa (vTask2) deve imprimir no display a hora atual obtida do sistema. O código abaixo permite obter a hora atual do sistema, lembrando que é necessário importar a biblioteca "time.h":

```
//Obtendo o tempo em segundos
time(&segundos);
//Converter unidade segundos para o tempo local
datetimeNow = localtime(&segundos);
//Obtendo a hora
int hour = datetimeNow->tm_hour;
//Obtendo os minutos
int min = datetimeNow->tm_min;
//Obtendo os segundos
int sec = datetimeNow->tm_sec;
```

- h. A terceira tarefa (vTask3) deve imprimir no display o nome da cidade onde você se encontra e uma temperatura obtida aleatoriamente. O código abaixo permite obter um número randômico do tipo float, a temperatura chegando até o valor máximo definido na constante maxTemp.

```
//Obtendo um número float aleatório, o valor máximo definido na constante maxTemp
float temp = ((float)rand() / (float)(RAND_MAX)) * maxTemp;
//Adiciona conteúdo na variável display
sprintf(display, "Task %ld - Curitiba %.2f°C\n", idTarefa,temp);
//Imprime o conteúdo do display na saída do console
vPrintString(display);
```