Modelagem do Sistema Operacional de Tempo Real FreeRTOS para um Desenvolvimento de Sistemas de Tempo Real Fidedignos

S. Galvão, D. Déharbe, V. Medeiros Júnio, A. Martins Moreira

[1]Departamento de Informática e Matemática Aplicada
Programa de Pós-graduação em Sistemas e Computação
Universidade Federal do Rio Grande do Norte
Campo Universitário, Lagoa Nova
59078-970, Natal, RN, Brasil

{stepgalvao,david,junior,anamaria}@consiste.dimap.ufrn.br

Abstract. *This meta-paper describes the style to be used in articles and short papers for SBC conferences. For papers in English, you should add just an abstract while for the papers in Portuguese, we also ask for an abstract in Portuguese ("resumo"). In both cases, abstracts should not have more than 10 lines and must be in the first page of the paper.*

Resumo. *Este meta-artigo descreve o estilo a ser usado na confecção de artigos e resumos de artigos para publicação nos anais das conferências organizadas pela SBC. É solicitada a escrita de resumo e abstract apenas para os artigos escritos em português. Artigos em inglês deverão apresentar apenas abstract. Nos dois casos, o autor deve tomar cuidado para que o resumo (e o abstract) não ultrapassem 10 linhas cada, sendo que ambos devem estar na primeira página do artigo.*

1. Introdução

A tecnologia da informação é uma característica intrínseca da vida quotidiana moderna, a qual está fortemente dependente dos softwares presentes em quase todas as tarefas da sociedade atual. Entretanto, para proporcionar uma maior comodidade e segura aos seus usuários, a qualidade do sistema tornou-se um requisito indispensável. Devido a isso, o desenvolvimento de sistemas com qualidade e segurança deixou de ser uma ostentação para tornar-se uma necessidade na vida moderna.

Um exemplo de sistemas presentes na sociedade atual que necessitam de um certo grau de confiabilidade e segurança são os sistemas de tempo real. Esse tipo de sistemas são geralmente embarcados e possuem como sua principal característica um rigoroso tempo de resposta aos eventos externos do ambiente controlado por ele [Li and Yao 2003]. Devido isso e a sua facilidade de trabalhar em ambientes com uma limitada quantidade de recursos, esses sistemas possuem uma larga utilização podendo a sua funcionalidade varia do controle de eletrodomésticos até a realização de operações críticas como o controle de aparelhos hospitalares e o controle de transportes (aeronaves, trens e automóveis).

Foi devida essa grande necessidade do desenvolvimento de sistemas com qualidade que a sociedade brasileira de computação, com a finalidade de intensificar as

pesquisar nessa área, classificou como um dos cinco grandes desafios da computação "o desenvolvimento tecnológico de qualidade", no qual está presente o tópico de "desenvolvimento e adaptação de tecnologia e instrumento em geral de apoio para à implementação e à avaliação de software fidedigno por construção"[S.B.C 2006].

Ligado a esse tópico está também o desafio proposto pelo o pesquisador britânico Jim, em [Woodcock 2008], de especificar formalmente o sistema operacional de tempo real FreeRTOS[FreeRTOS],o qual serve como uma ferramenta base para o desenvolvimento de sistemas de tempo real. Assim, através da especificação dessa ferramenta será possível proporcionar a mesma toda a segurança, confiabilidade e coerência dos métodos formais, e com isso transformando essa ferramenta em uma base para o desenvolvimento de sistemas de tempo real fidedigno.

Entretanto, a especificação de um sistema operacional de tempo real é uma tarefa árdua e praticamente impossível de ser realizada em um único trabalho, pois os formalismos existentes possuem muitas restrições o que impede a especificação de sistemas complexos, principalmente, os que possuem alocação dinâmica de memória. Assim, o esforço relatado nesse trabalho pretende apenas inicializar a resolução dos desafios comentados acima, os quais teram sua completa solução formada pela união de esforços de vários trabalhos.

Para essa inicialização, foi criada, utilizando-se o método B [Schneider 2001], uma modelagem abstrata do sistema, na qual, algumas das suas principais propriedades são especificadas e comprovadas. O método B servirá assim, como a abordagem utilizada para o rigoroso desenvolvimento da ferramenta, pois ele inspirado em técnicas de especificação, como VDM e Z, e na teoria de refinamento proporciona um rígido desenvolvimento de sistemas, que se inicia na modelagem abstrata de um sistema vai até o nível de algoritmo, podendo este último ser estendido para o nível de assemblagem do microprocessador alvo [Dantas et al. 2008].

Assim para iniciar essa especificação, primeiramente foi desenvolvido um projeto de modelagem com a finalidade de estudar o sistema e identificar os seus principais comportamentos e características. Esse estudo foi feito através do material de apoio do sistema junto com a verificação do seu código fonte, onde foi averiguado o comportamento interno dos seus estados. Após esse estudo, foram escolhidas estrategicamente algumas das principais funcionalidades e características do sistema para serem especificadas nessa modelagem inicial, sendo as demais abstraídas para serem tratados em futuros refinamento da especificação.

Com isso, esse trabalho prossegue demonstrado como é realizada a modelagem aqui proposta e como ela pretende iniciar a resolução do desafio de especificar formalmente o sistema operacional de tempo real FreeRTOS contribuindo assim para criação de uma ferramenta capaz de construir sistemas de tempo real mais fidedignos.

Para cumprir o seu objetivo, esse artigo foi dividido da seguinte forma. inicialmente tem-se com uma pequena introdução ao FreeRTOS onde são comentados alguns conceitos relevantes para esse trabalho. Em seguida, na sessão 3 o método B é rapidamente comentado. Após isso tem-se uma sessão explicando como foi feito o projeto da modelagem aqui descrita. E por fim, na sessão 5 é relatada como foi feita essa modelagem inicial do sistema e na sessão 6 tem-se as conclusões e possíveis continuações

desse trabalho.

2. FreeRTOS

O FreeRTOS é um sistemas operacional de tempo real pequeno, simples e de fácil uso. O seu código fonte, feito em *C* com partes em assembler, é aberto e possui aproximadamente 2.242 linhas de código, que são basicamente distribuídas em quatro arquivos (task.c, queue.c, croutine.c e list.c). Além disso, outra característica marcante desse sistema esta na sua portabilidade, sendo o mesmo oficialmente portável para 17 arquiteturas diferentes, entre elas a PIC, ARM, Zilog Z80 e PC, arquiteturas bastante utilizadas pelas indústrias de microeletrônica.

Por ser um sistema operacional, o FreeRTOS comporta-se como uma camada abstrata localizada entre a aplicação e o hardware, como na figura 1. Essa camada tem como principal objetivo esconder da aplicação detalhes do hardware que será utilizado. Assim, o funciona do FreeRTOS, pode ser comparado ao de uma ferramenta, igual a da figura 2, que fornece para os seus usuários serviços de gerenciamento de tarefa, comunicação e sincronização entre tarefas, controle de memória, gerenciamento do tempo e controle dos dispositivos de entrada e saída, tornando assim o desenvolvimento das aplicações de tempo real mais simples e prático, pois o usuário necessita apenas procupar-se com as funcionalidades do sistema deixando os demais detalhes de *hardware* para o FreeRTOS.

[Acho que poderia ser demonstrado um exemplo de como é feita uma aplicação utilizando o FreeRTOS]

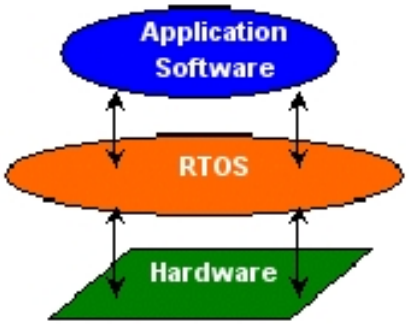

Figure 1. Figura da esquerda

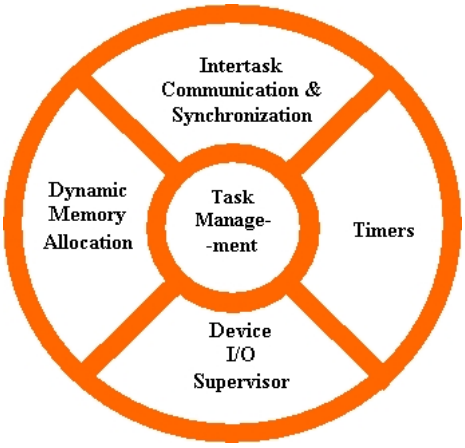

Figure 2. Figura da direita

Em seguida tem-se o detalhamento dos principais serviços disponibilizados pelo FreeRTOS que serão relevantes para a continuidade do trabalho, sendo que maiores informações sobre os demais podem ser encontradas em [FreeRTOS].

2.1. Gerenciamento de Tarefa

Para entender-se o funcionamento do gerenciador de tarefas do FreeRTOS é necessário primeiro entender-se o conceito de tarefa. Para o FreeRTOS, tarefas são unidades de trabalho independentes que compõem um sistema multitarefa. Nele, uma tarefa é composta por: uma pilha de contexto, que serve para armazenar o ambiente de execução (estados

dos registradores) no momento que uma tarefa é interrompida; uma prioridade que varia de zero até uma constante de prioridade máxima definida pelo o usuário e serve para indicar a importância da tarefa para o sistema; e um estado, que demonstra a atual situação da tarefa, podendo a mesma estar em execução, pronta, suspensa ou bloqueada.

O kernel do FreeRTOS, além de ser multitarefa, suporta três tipos de configurações diferentes de gerenciamento, podendo o mesmo ser preemptivo, cooperativo ou híbrido. Na configuração preemptivo, a tarefa em execução pode ser interrompida a qualquer momento pelo escalonador e consequentemente substituída por uma outra tarefa de acordo com a política de escalonamento. Diferentemente, na configuração cooperativa, a tarefa em execução não é interrompida pelo escalonador, pois ela é quem decide o momento de sua pausa seguida da sua substituição, que será feita de acordo com a política de escalonamento. E por fim, na configuração híbrida, o escalonador funciona hora como preemptivo e hora como cooperativo, sendo que um grupo de tarefas faz o escalonamento cooperativo entre si e as demais realizam o gerenciamento preemptivo.

A política de escalonamento do FreeRTOS, assim como a maioria dos sistemas operacionais de tempo real, é baseada na prioridade. Isso indica que tarefas com maiores prioridades têm sempre a preferência sobre as tarefas de menores prioridades do mesmo estado. Uma conseqüência dessa política e a propriedade de que a tarefa que está em execução deve possuir prioridade maior ou igual às tarefas de estado pronto, pois a tarefa em execução nada mais é do que a tarefa que tem preferência de entrar em execução entre as tarefas de estado pronto.

Um exemplo dessa política de escalonamento no modo preemptivo pode ser visto na figura 3, na qual as tarefas de menores prioridades são interrompidas pelo escalonador, quando tarefas de maiores prioridades assumem o estado pronto. Outro fator importante no funcionamento de escalonador que vale a pena ressaltar, é que tarefas de mesma prioridade dividem o tempo de execução entre si. Com isso, se houver duas tarefas prontas e com a prioridade máxima, elas irão dividir tempo de processamento em frações iguais entre si.

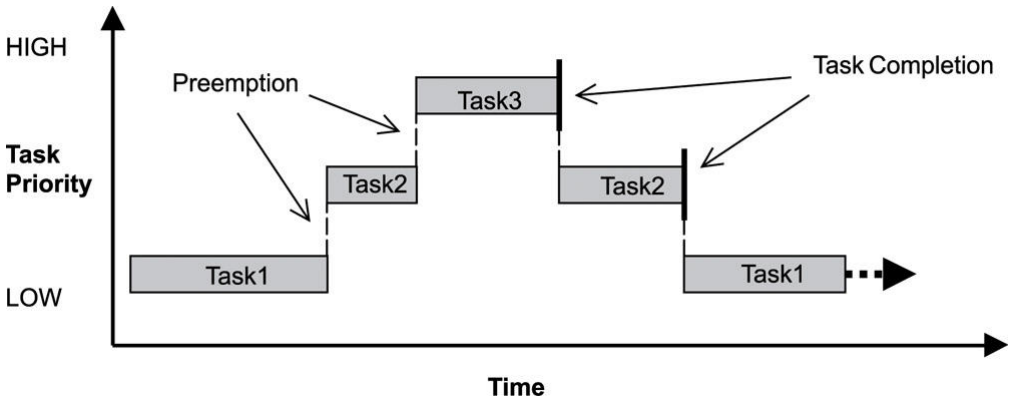

Figure 3. Funcionamento do escalonador do FreeRTOS

2.2. Gerenciamento da comunicação e sicronização entre tarefas

A comunicação entre tarefas no FreeRTOS é feita através de uma estrutura denominada fila de mensagens. Filas de mensagens são estruturas primitivas de comunicação entre

tarefas que funcionam como um túnel no qual as tarefas enviam e recebem mensagem. No FreeRTOS, as filas de mensagens possuem tamanho fixo, calculado através do tamanho e da quantidade de mensagens que ela pode armazenar. O seu funcionamento ocorre da seguinte forma, uma tarefa enviar uma mensagem para uma fila, essa verifica se existe espaço na sua estrutura e caso exista, a mensagem é adcionada a fila, caso contrário, a tarefa é bloqueada até que seja liberado um lugar na fila. O mesmo ocorre quando uma tarefa quer receber uma mensagem da fila, só que, nesse caso, a fila verifica primeiramente se existe alguma mensagem no seu repositório, caso afirmativo, a mensagem é passada para a tarefa e, caso contrário, a tarefa é bloqueada até a uma mensagem ser adicionada na fila.

Os mecanismos usados para sincronização entre tarefas no FreeRTOS são os semáforos. Eles funcionam como uma chave que libera, ou não, o uso de um determinado recurso compartilhado por duas ou mais tarefas. Assim quando uma tarefa deseja acessar um recurso compartilhado, ela primeiramente deve solicitar o semáforo do recurso, caso este esteja disponível, a tarefa tem a permissão de utilizar o recurso e, em seguida, disponibilizar o semáforo novamente, caso contrário, a tarefa é bloqueada até que o semáforo seja liberado. No FreeRTOS os semáforo são implementados como uma fila de mensagens com um único item. Assim, quando a fila estiver vazia, indica que o semáforo está em uso e, quando a fila estiver cheia, indica que o semáforo está liberado. Desse modo, as tarefas que utilizam uma fila de mensagens como semáforo não estão preocupadas com o conteúdo da mensagem, mas apenas com o estado da fila, se ela está cheia ou vazia, indicando a disponibilidade ou não do semáforo.

3. O método B

[Esta sessão deve obrigatoriamente falar sobre conceitos de substituição, modularização e refinamento, como também utilizar um exemplo simples que esteja ligado ao contexto do artigo]

4. Projeto da modelagem

Antes de iniciar a modelagem abstrata de qualquer sistema é necessário primeiramente conhecer as suas principais características e o seu comportamento interno e externo. Assim, com o intuito de adquirir essas informações e realizar o planejamento da especificação do sistema, foi desenvolvido um projeto da modelagem do FreeRTOS, no qual as principais características do sistema são listadas e, em seguida, estrategicamente selecionadas para fazerem parte dessa modelagem inicial.

A identificação das características do sistema foi feita através do levantamento bibliográfico sobre o mesmo. Nele foi possível constatar que o FreeRTOS é composto principalmente por cinco conceito fundamentais, tarefa, fila de mensagens, co-rotina, semáforo e mutex, sendo que cada um desses conceitos possuem uma vasta quantidade de funcionalidades, as quais servem como interfaces de interação entre os conceitos e os usuários do sistema, pois é através delas que os conceitos são instanciados e manuseados.

Para selecionar os principais conceitos, junto com as suas funcionalidades, foram levados em consideração a importância e complexidade do mesmo. Com isso, os conceitos selecionados para fazerem parte dessa modelagem inicial foram o conceito de tarefa e o conceito de fila de mensagens. O conceito de tarefa foi escolhido devido a sua grande

importância para o sistema, pois além dele ser o conceito base do sistema é através dele que são implementadas a maioria das funcionalidades e características do kernel, sendo com isso impossível iniciar uma especificação do sistema sem esse conceito. E, por fim, a escolha do conceito de fila de mensagem está liga principalmente a sua simplicidade e importância, pois esse conceito trata-se de uma estrutura de dados simples, do tipo FIFO (First in first out), e através dele são implementados os conceitos de semáforo e fila de mensagem, servindo assim como um conceito crucial para a especificação dos demais. A seguir tem-se como o FreeRTOS trata cada um desses conceitos.

4.1. Tarefa

Na sessão 2.1, foi dito que uma tarefa é composta por uma prioridade, por um estado, e uma pilha de contexto. Assim, no FreeRTOS, a implementação de uma tarefa é feita como demonstra o código a seguir, no qual primeiramente é criada a variável *pxCurrentTCB*, que armazena uma referência para a tarefa em execução, e as listas de tarefas *ReadyTasksLists*, *pxDelayedTaskList* e *xSuspendedTaskList*, que armazenam respectivamente referências para tarefas no estado pronto, bloqueado e suspenso. Após isso, é criado um bloco de controle reponsável pela representação de uma tarefa para o sistema. Assim esse bloco é formado principalmente pela variável *xGenericListItem*, que é colocada nas listas e variável de estados, indicando qual é o estado da tarefa, pela variável *uxPriority* que armazena a prioridade da tarefa e pela variável *pxTopOfStack* que aponta para o começo da pilha de contexto de uma tarefa.

```
..
tskTCB * volatile pxCurrentTCB = NULL;
static xList pxReadyTasksLists[ configMAX_PRIORITIES ];
static xList * volatile pxDelayedTaskList;
static xList xSuspendedTaskList;
..
typedef struct tskTaskControlBlock {
  volatile portSTACK_TYPE *pxTopOfStack;
  xListItem xGenericListItem;
  unsigned portBASE_TYPE uxPriority;
} tskTCB;
..
```

4.1.1. Funcionalidades

As funcionalidades que manuseiam o conceito de tarefa junto com o kernel do sistema podem ser divididas em dois grupos, gerenciamento de tarefa e controle do kernel. A seguir tem-se a listagem e descrição de cada uma das funcionalidades desses grupos escolhidas para fazerem parte da modelagem inicial.

Gerenciamento de Tarefa

Nesse grupo estão presentes as funcionalidades responsáveis por criar, excluir e manusear o conceito de tarefa. Ele é formado por 10 funcionalidades, sendo que 8 delas foram escol-

hidas para fazerem parte dessa especificação inicial. A lista completa das funcionalidades escolhidas junto com suas descrições pode ser vista a seguir:

- **xTaskCreate:** cria uma nova tarefa
- **xTaskDelete:** remove uma tarefa do FreeRTOS
- **uxTaskPriorityGet:** informa a prioridade de uma tarefa
- **vTaskSuspend:** coloca uma tarefa no estado "suspenso"
- **vTaskResume:** retorna uma tarefa suspensa colocando-a no estado pronto
- **vTaskPrioritySet:** altera a prioridade de uma tarefa
- **vTaskDelay:** bloqueia uma tarefa por uma determinada quantidade de tempo, levando em consideração o instante em que o método foi chamada. Essa funcionalidade não é recomendável para criação de tarefas cíclicas
- **vTaskDelayUntil:** bloqueia uma tarefa por uma determinada quantidade de tempo, levando em consideração o instante em que essa foi desbloqueada. Essa funcionalidade é recomendável para criação de tarefas cíclicas

Controle do Kernel

As funcionalidades presentes no conjunto de controle do kernel são responsáveis por iniciar, finalizar, suspender, retornar e fornecer informações úteis sobre as atividades do kernel. Com isso, as principais funcionalidades desse grupo escolhidas para fazerem parte dessa especificação inicial foram:

- **xTaskGetCurrentTaskHandle:** retorna uma referência para a tarefa em execução
- **xTaskGetSchedulerState:** informa o atual estado do escalonador (executando, suspenso ou não inicializado).
- **uxTaskGetNumberOfTasks:** informa a atual quantidade de tarefas tarefas do FreeRTOS.
- **xTaskGetTickCount:** informa o tempo decorrido apartir da inicialização do escalonador
- **vTaskStartScheduler:** esse método primeiramente cria a tarefa ociosa e, em seguida, inicia as atividades do escalonador
- **vTaskEndScheduler:** finaliza as atividades do escalonador colocando-o no estado não inicializado. Para isso, ele primeiramente exclui todas as estruturas (tarefas, filas de mensagens, co-rotinas, etc) criadas no FreeR TOS e em seguida finaliza o escalonador.
- **vTaskSuspendAll:** suspende temporariamente todas atividades do escalonador colocando-o no estado suspenso.
- **xTaskResumeAll:** retorna todas as atividades do escalonador quando este encontra-se no estado suspenso.

4.2. Fila de mensagem

O conceito de fila de mensagem é implementado como demonstra o código abaixo. Ele é formado principalmente por três variáveis. A primeira variável *pcHead* aponta para o início de uma área de memória alocada na criação da fila para armazenadas as mensagem enviadas à fila. Na segunda, *xTasksWaitingToReceive*, são armazenadas referências

paras tarefas que aguardam pela chegada de uma mensagem na fila, ou seja, as tarefas que foram bloqueadas ao tentar ler a fila quando ela estava vazia. E por último, na terceira variável *xTasksWaitingToSend*, são armazenadas referências para as tarefas que aguardam para enviar uma mensagem para a fila, ou seja, as tarefas que foram bloqueadas ao tentaram enviar uma mensagem para a fila quando esta estava cheia. Além disso, na implementação desse conceito existem também duas variáveis adicionais *uxItemSize* e *uxLength*. A primeira armazenar o tamanho da mensagem da fila e a outra armazenar a quantidade de mensagem que a fila pode receber.

```
..
typedef struct QueueDefinition{
  signed portCHAR *pcHead;
  ..
  xList xTasksWaitingToReceive;
  xList xTasksWaitingToSend;
  ..
  unsigned portBASE_TYPE uxItemSize;
  unsigned portBASE_TYPE uxLength;
  ..
} xQUEUE;
```

4.2.1. Funcionalidades

As funcionalidades responsáveis por manusear esse conceito podem ser agrupadas em um único conjunto denominado de gerenciamento de fila de mensagem. Nesse grupo estão presentes as funcionalidades responsáveis pela criação, exclusão e gerenciamento do conceito de fila de mensagens. Assim as funcionalidades desse conjunto inicialmente selecionadas para fazerem parte dessa modelagem inicial são:

- **xQueueCreate:** cria uma nova fila de mensagem.
- **xQueueSend:** enviar uma mensagem para uma fila de mensagens.
- **xQueueSendToBack:** envia uma mensagem para o final de uma fila de uma fila de mensagens.
- **xQueueSendToFront:** envia uma mensagem para o início de uma fila de mensagem.
- **xQueueReceive:** ler e retirar a mensagem localizada no início da fila de mensagens.
- **xQueuePeek:** ler sem retirar a mensagem localizada no início da fila de mensagens.
- **vQueueDelete** - Exclui uma fila de mensagens do FreeRTOS.

5. Modelagem

Como visto na sessão 3, a especificação de sistemas através do método B pode ser feita utilizando-se dois mecanismos importantes, modularização e refinamento. Assim, com o intuito de iniciar a modelagem do FreeRTOS, foi feita primeiramente, utilizando-se o mecanismo de modularização, uma especificação abstrata dos conceitos e funcionalidades propostos pelo projeto de modelagem. Nessa especificação abstrata o principal objetivo

foi modelar o comportamento do sistema em relação à característica de estado de uma tarefa e suas permutações. Com isso, nessa primeira etapa da especificação inicial as funcionalidades e conceitos do FreeRTOS propostos na sessão 4 foram especificados apenas com informações relevantes para os estados de uma tarefa e suas permutações.

Feito o comportamento do sistema em relação ao estado de uma tarefa, foi necessário então, utilizando-se do conceito de refinamento, detalhar a especificação abstrata da etapa inicial para que essa suporte também a característica de prioridade de uma tarefa. Assim, foi adicionado à especificação abstrata da etapa inicial informações relevantes ao comportamento do sistema em relação à característica de prioridade. A seguir, tem-se em detalhe como foi realizada e especificada a primeira etapa da modelagem inicial acompanhada de seu refinamento.

5.1. Modelagem da característica de estado de uma tarefa

A modelagem abstrata dos conceitos e funcionalidades do sistema baseada na característica de estado de uma tarefa pode, assim como no projeto de modelagem, dividida em dois grupos, grupo de tarefa e grupo de fila de mensagens. No grupo de tarefa, como o próprio nome já diz, estão presentes os modulos que especificam o conceito de tarefa e suas funcionalidades e no grupo de fila de mensagens estão os módulos responsáveis pelo conceito de fila de mensagens e suas funcionalidades.

5.1.1. Tarefa

O conceito de tarefa, por preocupar-se somente com a característica de estado, nessa modelagem inicial foi especificado através do módulo $Task\_Core$ como demonstra a figura 4. Nela uma tarefa é formada pelo conjunto abstrato $TASK$, que simboliza o conjunto universo de todas as tarefas que podem ser criadas no FreeRTOS e pela variável $tasks$ que armazenada o conjunto de tarefas que estão sendo gerenciadas pelo sistema. Além disso, para representar os estados de uma tarefa, foram criadas as variáveis $running$, $ready$, $blocked$ e $suspended$, as quais representam respectivamente os estados executando, pronto, bloqueado e suspenso. Assim para uma tarefas estar no estado pronto, ela obrigatoriamente deve possuir uma referências na variável $ready$, ocorrendo o mesmo para os demais estado.

Uma conseqüência direta dessa especificação é a restrição de que as variáveis de estados não devem possuir referências comuns entre si, pois, em um determinado instante, uma tarefa só pode possuir um estado. Essa restrição é especificada pelo $INVARIANT$ da máquina através das sete últimas definições da figura 4, garantido assim que nessa especificação nenhuma tarefa possa possuir dois estados ao mesmo tempo.

Operações básicas

Após a especificação do conceito de tarefa, foram criadas, também na máquina $Task\_Core$, as operações básicas que manipulam as características de uma tarefa, nesse caso o estado, e servem como suporte para a especificação das funcionalidades que utilizam esse conceito. Ao todo, foram criadas 12 operações, das quais iremos observar

$$
\begin{array}{lll}
\textbf{SETS} & \textbf{INVARIANTS} & running \in TASK \wedge \\
\quad TASK; & \quad tasks \in \textbf{FIN}(TASK) \wedge & running \in tasks \wedge \\
\quad \ldots & \quad tasks \neq \emptyset \wedge & running \notin ready \wedge \\
 & & running \notin blocked \wedge \\
\textbf{VARIABLES} & blocked \in \textbf{FIN}(TASK) \wedge & running \notin suspended \wedge \\
\quad tasks, & blocked \subset tasks \wedge & \\
\quad blocked, & & ready \cup blocked = \emptyset \wedge \\
\quad running, & ready \in \textbf{FIN}(TASK) \wedge & blocked \cup suspended = \emptyset \wedge \\
\quad ready, & ready \subset tasks \wedge & suspended \cup ready = \emptyset \wedge \\
\quad suspended & & \\
\quad \ldots & suspended \in \textbf{FIN}(TASK) \wedge & tasks = \{running\} \cup suspended \\
 & suspended \subset tasks \wedge & \qquad \cup blocked \cup ready \\
 & \ldots &
\end{array}
$$

Figure 4. Modelagem do conceito tarefa

apenas a operação t_create, responsável pela criação de uma nova tarefa, ficando para o leitor interessado o trabalho de visitar o site do projeto e verificar as demais operações.

Na especificação da operação t_create, demonstrada pela figura 5, uma nova tarefa é criada através da substituição ANY, onde é definido que a nova tarefa deve pertencente ao conjunto abstrato $TASK$ e não deve estar "contida" no conjuto $tasks$, sendo, em seguida, armazenada na variável $task$. Após definida a nova tarefa, essa é colocada no conjunto de tarefas gerenciadas pelo FreeRTOS $task$ e, em seguida, fixada com o estado pronto através da inserção da mesma ao conjunto de tarefas com o estado pronto $ready$.

$$
\begin{array}{ll}
result \longleftarrow t\_create(priority) = & \ldots \\
\textbf{PRE} & \textbf{THEN} \\
\quad priority \in PRIORITY \wedge & \quad tasks := \{task\} \cup tasks \parallel \\
\quad running = TASK\_NULL & \quad ready := \{task\} \cup ready \parallel \\
\textbf{THEN} & \quad result := task \\
\quad \textbf{ANY} & \textbf{END} \\
\quad\quad task & \textbf{END}; \\
\quad \textbf{WHERE} & \\
\quad\quad task \in TASK \wedge & \\
\quad\quad task \notin tasks & \\
\quad \ldots &
\end{array}
$$

Figure 5. Especificação da operação *t_create*

Funcionalidades

Feita a modelagem das operações básicas do conceito de tarefa, resta apenas especificar as funcionalidades que se utilizam desse conceito. A especificação dessas funcionalidades é feita através da máquina $Task\_Core$, na qual estão prensentes as funcionalidades dos conjuntos controle de tarefa e controle do kernel da sessão 4.1.1. Entretanto aqui

demonstraremos apenas a especificação da funcionalidade $xTaskCreate$, a qual utiliza-se da operação básica t_create para criar uma nova tarefa que sera gerenciada pelo FreeRTOS. Assim, a especificação dessa funcionalidade foi feita como demonstra a figura 6, onde são identificados dois comportamentos possíveis para a função: ou a tarefa é criada através da chamada ao método t_create e uma referência para a nova tarefa é retornada, junto com a mensagem de sucesso; ou, a tarefa não é criada e uma referência nula, junto com a mensagem de erro, é retornada.

$result, handle \longleftarrow$
$xTaskCreate(pvTaskCode, pcName,$
$usStackDepth, pvParameters, uxPriority) =$
PRE
$pvTaskCode \in TASK\_CODE \wedge$
$pcName \in NAME \wedge$
$usStackDepth \in NATURAL \wedge$
$pvParameters \subset PARAMETER \wedge$
$uxPriority \in PRIORITY \wedge$
THEN
...
...
CHOICE
$handle \leftarrow$
$\mathbf{t\_create}(uxPriority) \parallel$
$result := pdPASS$
OR
$result := errMEMORY \parallel$
$handle := TASK\_NULL$
END

END;

Figure 6. Modelagem da funcionalidade $xTaskCreate$

5.1.2. Fila de mensagens

Independente de qualquer abstração, para iniciar-se a especificação do conceito de fila de mensagens é necessário que essa possua um conjunto capaz de armazenar as mensagens que serão enviadas para a fila. Com base nisso e na necessidade de preocupar-se como a característica de estado de uma tarefa, o conceito de fila de mensagens foi especificado pelo módulo $Queue\_Core$ como demonstra a figura 7. Nela, inicialmente tem-se os conjuntos abstratos $QUEUE$ e $ITEM$, que simbolizam respectivamente o conjunto universo das filas de mensagens e suas mensagens. Em seguida, a variáveis $queues$ é criadas para armazenar as filas de mensagens que estão sendo gerenciadas pelo FreeRTOS. E por fim, são criadas as relações $queue\_items$, $queue\_sending$ e $queue\_receiving$, responsáveis por ligar uma fila de mensagens a um conjunto de itens, a um conjunto de tarefas que esperam para enviar uma mensagem para a fila e a um conjunto de tarefa que aguardam para ler uma mensagem da fila, respectivamente.

Assim como ocorreu na modelagem do conceito de tarefa, após especificar-se o conceito de fila de mensagens foram criadas também, na máquina $Queue\_Core$, algumas operações básica responsáveis por manipular as características do conceito. Com isso foram implementadas o total de seis operações, nas quais são feitas as inclusões e exclusões de mensagens e tarefas nos conjuntos que formam o conceito de fila de mensagens. E para finalizar essa especificação abstrata, após a criação das operações básicas, as funcionalidades que utilizam o conceito de fila de mensagens (funcionalidades do grupo gerenciamento de fila de mensagens) foram especificadas pela máquina $Queue$.

SETS	**INVARIANTS**
$QUEUE$;	$queues \in \mathbf{POW}(QUEUE) \wedge$
$ITEM$;	$queue\_items \in QUEUE \nrightarrow \mathbf{POW}(ITEM) \wedge$
...	$queue\_receiving \in QUEUE \nrightarrow \mathbf{POW}(TASK) \wedge$
VARIABLES	$queue\_sending \in QUEUE \nrightarrow \mathbf{POW}(TASK) \wedge$
$queues$,	
$queue\_items$,	$queues = \mathbf{dom}(queue\_items) \wedge$
$queue\_receiving$,	$queues = \mathbf{dom}(queue\_receiving) \wedge$
$queue\_sending$	$queues = \mathbf{dom}(queue\_sending)$

Figure 7. Modelagem do conceito fila de mensagem

5.2. Modelagem da característica de prioridade de uma tarefa

Para adicionar a característica de prioridade de uma tarefa à modelagem inicial, foi necessário apenas fazer o refinamento do conceito de tarefa e suas operações básicas, ou seja, o refinamento da máquina $Task\_Core$. Essa simplicidade é consequência direta da modularição realizada na primeira etapa da especificação, pois as demais máquinas que se utilizam do conceito de tarefa fazem esse uso através do módulo $Task\_Core$, sendo necessário apenas o refinamento do mesmo para que a característica de prioridade seja adicionada a modelagem. Esse refinamento é feito através da máquina $Task\_CoreR$ no qual os estados e operações do módulo $Task\_Core$ são refinados de acordo com os comportamentos exigidos pela característica de prioridade.

5.2.1. Refinamento do conceito de tarefa

A nova especificação do conceito de tarefa ficou refinada como demonstra a figura 8. Nela foram adicionadas a definição do conjunto de prioridade, $PRIORITY$, que até o momento tratava-se de um conjunto abstrato, a variável $t\_priority$, usada para ligar uma tarefa a sua prioridade e a restrição presente nas seis últimas linhas do $INVARIANT$ a qual especifica que, após inicializado o funcionamento do sistema a tarefa em execução deve possui prioridade maior ou igual às tarefas de estado pronto.

PROPERTIES	**INVARIANTS**
$PRIORITY =$	$t\_priority \in TASK \nrightarrow PRIORITY \wedge$
$0..(configMAX\_PRIORITIES - 1)$	$\mathrm{dom}(t_priority) = tasks \wedge$
...	$\forall tt.(tt \in ready \wedge$
VARIABLES	$ready \neq \emptyset \wedge$
...	$running \neq TASK\_NULL \wedge$
$t\_priority$	$tt \neq running \Rightarrow$
...	$t\_priority(running) \geq t\_priority(tt)$
	)

Figure 8. Refinamento do conceito tarefa

5.2.2. Refinamento das operações básicas

Após o refinamento do conceito de tarefa, foi necessário fazer também o refinamento das suas operações básicas. Assim todas as operações presentes na máquina $Task\_Core$ foram refinadas na máquina $Task\_CoreR$. Um exemplo desse refinamento está na operação t_create que foi refinada como demonstra a figura 9. Através dela percebe-se que ao ser criada uma tarefa é necessita agora relacionar essa com uma prioridade, sendo em seguida, essa relação é armazenada na variável $t\_priority$. As demais operações seguem o mesmo raciocínio e procuram adicionar o conceito de prioridade ao seu funcionamento respeitado as restrições impostas pelo $INVARIANT$ da máquina abstrata e do seu refinamento.

$result \longleftarrow t\_create(priority) =$
PRE
$priority \in PRIORITY \wedge$
$running = TASK\_NULL$
THEN
ANY
$task$
WHERE
$task \in TASK \wedge$
$task \notin tasks$
...
...
THEN
$tasks := \{task\} \cup tasks \parallel$
$t\_priority := t\_priority \cup \{task \mapsto priority\} \parallel$
$ready := \{task\} \cup ready \parallel$
$result := task$
END
END;

Figure 9. Refinamento da operação *t_create*

6. Conclusões e Trabalhos futuros

Como conclusões desse trabalho têm-se além da inicialização do desafio proposto por Jim de especificar formalmente o sistema operacional de tempo real FreeRTOS, um ponta pé para a criação de uma ferramenta capaz de construir sistemas de tempo real mais fidedigno, pois com a especificação do FreeRTOS esse sistema terá a garantia, comprovada pro sua especificação, que realmente cumpre as funcionalidades propostas por ele.

Entretanto além comprovar formalmente as propriedades do FreeRTOS e servir como base para uma modelagem formal do sistema, essa especificação também pode ser usadas como entrada para outras técnicas de verificação como a criação de caso de teste apartir de uma especificação do sistema, onde a partir da especificação são gerados casos de teste para verificar o correto funcionamento determinadas funcionalidades do sistema.

Contudo, como esse trabalho trata-se apenas de uma introdução para uma completa modelagem do sistema, ele possui várias vertentes para a sua continuação, sendo a mais natural dessas vertentes o refinamento até o nível de algoritmo dessa especificação, o que proporcionaria uma modelagem capaz de ser sintetizada para uma alguma linguagem de programação convencional como C.

Outra possibilidade de continuação desse trabalho seria a especificação do sistema usando outro tipo de formalismo, no qual o mais recomendadopara aproveitar

a especificação aqui inicializada seria o Event B, considerado como uma extensão do método B, para a especificação com propriedades concorrentes.

References

Dantas, B., Déharbe, D., Galvão, S., Moreira, A. M., and Jr., V. M. (2008). Proposta e avaliação de uma abordagem de desenvolvimento de software fidedigno por construção com o método b. In *Anais do XXXV Seminário Integrado de Software e Hardware (SEMISH 2008)*.

FreeRTOS. Freertos:designed for microcontrollers. `http://www.freertos.org`, último acesso em Fevereiro de 2008. Site de informações sobre o sistema.

Li, Q. and Yao, C. (2003). *Real Time Concepts for Embedded Systems*. CMP Books.

S.B.C (2006). Grandes desafios da pesquisa em computação no brasil 2006-2016. Technical report, Sociedade Brasileira de Computação.

Schneider, S. (2001). *The B-Method: An Introduction*. Palgrave.

Woodcock, J. (2008). Grand challenge in software verification. Slides da sua aprensentação no Simpósio Brasileiro de Métodos Formais - SBMF 2008.