RTKPIC18 – Real Time Kernel PIC18FXX8

Pedro Leite , Ricardo Marau

***Resumo* – Este artigo descreve sucintamente um executivo (*kernel*) tempo-real para os microprocessadores da família PIC18FXX8, desenvolvido pelos autores no âmbito da disciplina de Sistemas de Tempo-Real, opção do 5º ano da LEET e LECT. Nele são abordados aspectos gerais da implementação deste tipo de executivos, como sejam, escalonabilidade das tarefas no acesso ao processador, a preempção de tarefas e os tempos de activação. São também abordados aspectos específicos desta família de microprocessadores, como a mudança de contexto das tarefas e a gestão temporal.**

Foram realizadas análises de carga do executivo num sistema multitarefas com um PIC18F258. O espaço ocupado pelo kernel na memória de código é de 2900 *words* e na memória de dados de 39 bytes mais 31 por tarefa. A resolução temporal do kernel é de 2ms.

I. INTRODUÇÃO

Os sistemas de tempo-real são sistemas computacionais que devem reagir a eventos dentro de restrições temporais precisas. Como consequência, o correcto comportamento destes sistemas não depende apenas dos resultados computacionais, mas também do instante temporal em que esses resultados são produzidos. Uma reacção que ocorre demasiado tarde pode ser inútil ou mesmo perigosa [1].

Nesse sentido, a utilização de um kernel tempo-real pode constituir uma ajuda particularmente útil em aplicações relativamente complexas que contenham várias tarefas, permitindo ter explicitamente em conta as respectivas restrições temporais e escalonando-as por forma a que essas restrições sejam cumpridas.

As actividades básicas que um kernel tempo-real tem de executar são:

1) Gestão de tarefas; incluí várias funções de suporte ao sistema, como a criação e terminação de tarefas, escalonamento de tarefas, mudanças de contexto, etc. Em particular, o escalonamento é uma peça fundamental para a pontualidade do sistema dado que determina qual a próxima tarefa a utilizar o processador. Dois tipos comuns são os escalonamentos baseados em prioridades fixas e em prioridades dinâmicas. Exemplos do primeiro tipo são o escalonamento *Rate Monotonic* (RM), no qual a prioridade é atribuída de forma inversamente proporcional ao período de reactivação das tarefas, e o escalonamento *Deadline Monotonic* (DM), semelhante ao anterior mas em que a prioridade é atribuída de acordo com uma *deadline* relativa que pode ser mais curta que o período. No que diz respeito ao segundo tipo, um exemplo comum é o escalonamento *Earliest Deadline First* (EDF), que atribuí a maior prioridade à tarefa com a deadline absoluta mais próxima.
2) *Handling* de interrupções; serviço às interrupções geradas por dispositivos periféricos. Note-se que as interrupções podem causar interferências no funcionamento do sistema e, como tal, o respectivo tratamento deve ser executado de acordo com políticas adequadas que permitam quantificar tais interferências. Uma forma habitual é a utilização de *handlers* muito curtos, que apenas activam tarefas aperiódicas onde é efectuado o respectivo processamento. Estas tarefas são depois tidas em conta na escalonabilidade sistema, em conjunto com as restantes tarefas periódicas.
3) Sincronização e comunicação de processos; Serviços que permitem uma correcta comunicação entre as tarefas, incluíndo sincronização no acesso a recursos partilhados (regiões críticas), e.g. através de semáforos ou *lock flags*.

Neste artigo descreve-se sucintamente um kernel tempo-real simples, o RTKPIC18, que é multitarefa, preemptivo e com vários tipos de escalonamento. Nesta versão não existem mecanismos específicos para sincronização de tarefas nem atendimento de tarefas aperiódicas, sendo contudo possível estabelecer fases relativas e desse modo cumprir relações de precedência. Este kernel foi desenvolvido de raíz pelos autores, para os microcontroladores da família PIC18FXX8 da Microchip. O trabalho foi realizado no âmbito da disciplina de Sistemas de Tempo-Real [2], do 5º ano da Lic. em Engª de Electrónica e Telecomunicações da Universidade de Aveiro.

II. AMBIENTE DE DESENVOLVIMENTO

Conforme referido, o *kernel* RTKPIC18 foi desenvolvido para os microcontroladores da família PIC18FXX8, tendo sido escrito em linguagem C e compilado usando o PICC-18 v8.20PL4 da HI-TECH Software [3]. Todo o desenvolvimento do *kernel*, desde a fase de projecto, implementação, teste, *debug* e simulação foi efectuado recorrendo a um IDE (*Integrated Development Environment*) da HI-TECH Software, o HI-TIDE v1.0PL4.

III. CARACTERIZAÇÃO DO SISTEMA

Ao desenvolver uma aplicação baseada no RTKPIC18, o utilizador deve começar por definir as tarefas que pretende implementar, bem como a respectiva caracterização temporal, isto é, o periodo de activação e a *deadline*

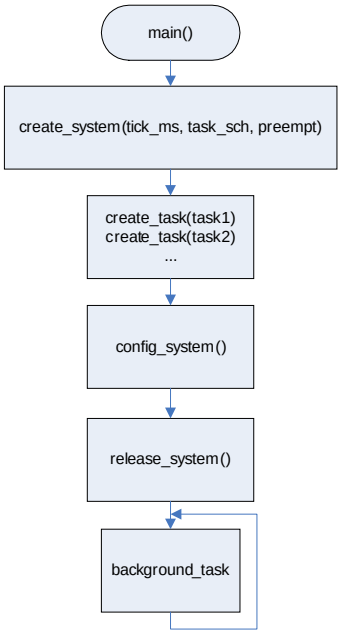

Fig. 1: Criação de um sistema

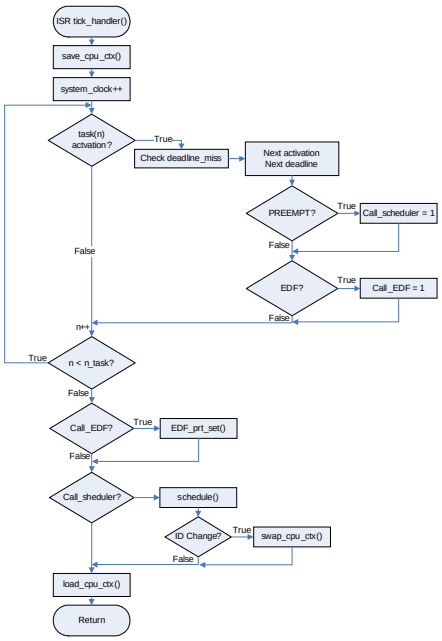

Fig. 2: ISR tick_handler

relativa. Após escrito o código relativo às várias tarefas, é necessário construir o sistema completo (fig. 1). Para tal, o *kernel* disponibiliza a função *create_system*, que cria e inicializa as estruturas internas, e que permite definir o período do *tick*, o tipo de escalonamento das tarefas (RM, DM ou EDF) e a modalidade de preempção (PREEMPT ou NOPREEMPT).

Seguidamente são criadas as várias tarefas utilizando a função *create_task*. A criação de uam tarefa consta de associar ao respectivo código as estruturas dinâmicas necessárias tais como o t*ask control block (tcb)* e a memória para salvaguarda do contexto. A função *config_system* permite atribuir as prioridades às tarefas, no caso de prioridades fixas.

O arranque do sistema, efectua-se através da primitiva *release_system* que inicia a contagem do tempo (em *ticks*) por parte do sistema, permitindo assim respeitar as fases relativas estabelecidas para as tarefas (instante da primeira activação). Seguidamente, o sistema entra num ciclo infinito onde é executada a tarefa de background. Este ciclo é interrompido para a execução das tarefas definidas pelo utilizador.

O controlo e gestão temporal das tarefas utiliza uma interrupção periódica, o *tick*, que define a resolução do sistema. Esta interrupção é servida pelo *tick_handler* (fig. 2) que determina as activações e suspensões das tarefas, e é gerada a uma cadência de 2ms por um *timer* de *hardware* modular (*timer*2). Desta forma, para garantir a precisão da gestão temporal do sistema, é necessário que as interrupções não sejam desactivadas por períodos superiores a 2ms. No *tick_handler* é feito um *software postscaler* que permite a obtenção de um *tick* com períodos entre 2 e 65534 ms em intervalos de 2ms.

VI. CARACTERIZAÇÃO DAS TAREFAS

A criação e caracterização das tarefas é efectuada ainda antes do arranque do sistema. São permitidas no máximo 13 tarefas às quais é adicionada a tarefa de *background*. As tarefas são caracterizadas pelo período de activação, por um *offset* referente à primeira activação (fase inicial) e ainda pela sua *deadline* relativa.

Num *kernel* multitarefa as várias tarefas concorrem pelo processador podendo encontrar-se em vários estados relativamente à sua execução. O respectivo diagrama de estados é ilustrado na fig. 3. Existem 3 estados possíveis:

- *IDLE*, estado em que são colocadas as tarefas quer quando acabadas de criar, quer entre a terminação de uma instância e a activação da instância seguinte;
- *READY*, estado em que são colocadas as tarefas assim que activadas ou quando sofrem preempção, enquanto esperam por receber tempo de processador;
- *RUN*, estadp em que é colocada a tarefa correntemente em execução.

As características estáticas e dinâmicas de cada tarefa, incluíndo as características temporais, a prioridade e o estado, são salvaguardadas numa estrutura de dados referida por *task control block*. Por sua vez, os vários *tcb's*

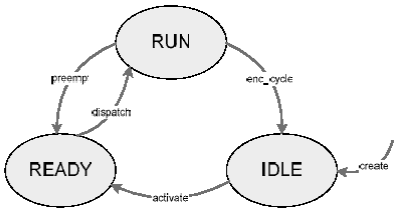

Fig. 3: Diagrama de estados das tarefas

```
void nome_tarefa(void)
{
        (...) /* definição de variáveis (não inicializar) */

        task_init();

        (...) /* inicialização de variáveis estáticas */
        while(1)
        {
                (...) /* inicialização de variáveis locais */
                (...) /* código periódico */

                end_cycle();
        }
}
```

Fig. 4: Modelo periódico de execução das tarefas.

estão agrupados numa tabela gerida pelo *kernel*.

O modelo computacional subjacente ao kernel é um modelo periódico, em que as tarefas são constituídas por um ciclo infinito, onde a última instrução desse ciclo é a chamada ao sistema *end_cycle* (fig. 4). Assim, as tarefas nunca terminam, no sentido de fazerem *return*, mas são reactivadas periodicamente. Com a primitiva *end_cycle* o *kernel* manipula a *stack* de funções do microcontrolador PIC para que o retorno seja efectuado para a próxima tarefa escalonada.

No início de cada tarefa, a função *task_init* permite ao *kernel* inicializar o contexto dessa tarefa. Após a execução desta função em todas as tarefas o sistema fica preparado para arrancar, com todas as tarefas alinhadas à entrada do respectivo ciclo.

O contexto das tarefas refere-se às respectivas variáveis dinâmicas bem como a todo o conjunto de parâmetros que lhes permite concorrer independentemente no acesso ao processador. Nestes parâmetros estão incluídos o *program counter*, *status* e todo um conjunto de registos utilizados pelo processador. O contexto de cada tarefa é guardado e manipulado pelo *kernel* num bloco de memória chamado *task CPU context* (*tcpuctx*).

Um dos aspectos específicos do microcontrolador PIC18FXX8 é não possuir *stack* para variáveis locais. Este facto requer uma abordagem diferença da usual na salvaguarda do contexto das tarefas para permitir preempção, normalmente efectuada num *stack*, quer privado da tarefa quer global. Neste microcontrolador, o espaço para variáveis dinâmicas (locais) é reservado estaticamente, na fase de linkagem, sendo sobreposto para funções independentes. Este facto leva à possível destruição do conteúdo das variáveis entre sucessivas invocações da função, o que não é admissível no caso da preempção de tarefas.

No entanto verifica-se que se uma função invocar outra, o *linker* reserva espaços disjuntos para as respectivas variáveis locais. Mais ainda, o *assembler* disponibiliza uma directiva (FNCALL) que indica ao *linker* que uma determinada função é chamada por uma outra, mesmo que isso não aconteça. Assim, graças a esta directiva, consegue-se que sejam atribuídas zonas disjuntas para as variáveis locais e contexto das várias tarefas, permitindo que os respectivos valores se mantenham intactos quando ocorre preempção.

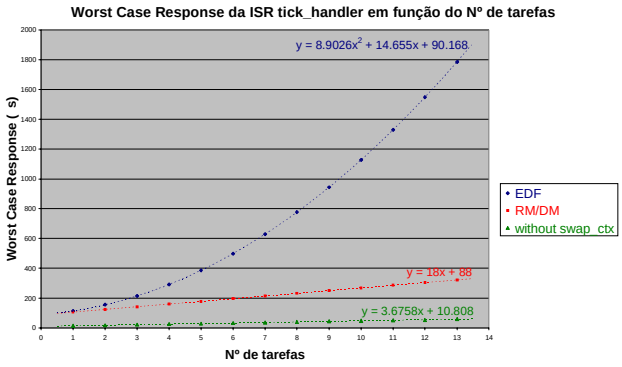

Fig. 5: Resposta temporal da ISR *tick_handler*

VI. CARACTERIZAÇÃO QUANTITATIVA

Memory footprint:

Em termos de memória de código, o *kernel* utiliza 2900 *words* de um total 32*Kwords* disponível (8.8%).

No que diz respeito à memória de dados, o *kernel* utiliza 39 bytes + 31 bytes por tarefa, de um total de 1536 bytes de memória RAM. Assim, utilização da RAM variará entre 101 bytes (6.6%) para 1 tarefa de utilizador mais *background*, e 473 bytes (31%) para o máximo de 13 tarefas de utilizador e *background*.

Tempo de resposta do *tick_handler:*

O tempo de resposta do *tick_handler* depende do tipo de escalonamento utilizado (fig. 5). Para um escalonamento baseado em prioridades fixas, o overhead computacional do sistema é muito menor do que o escalonamento baseado em prioriades dinâmicas. No entanto sabe-se que neste último caso o aproveitamento do CPU é melhor e resulta num menor número de preempções. Este facto estabelece um compromisso que depende da aplicação.

V. CONCLUSÕES

Neste artigo descreveu-se o kernel RTKPIC18, um kernel tempo-real que permite efectuar a gestão de várias tarefas periódicas de acordo com vários algorítmos de escalonamento (RM, DM e EDF), com e sem preempção. Apresentou-se a respectiva estrutura interna bem como a sua utilização e caracterização quantitativa.

Um aspecto relativamente importante que poderá ser objecto de desenvolvimento futuro é a adição de mecanismos mais eficientes de sincronização entre as tarefas no acesso a recursos partilhados e de atendimento de tarefas aperiódicas.

REFERÊNCIAS

[1] Giorgio C. Buttazzo, *Hard Real-Time Computing Systems*, Kluwer Academic Publishers, 1997.

[2] http://sweet.ua.pt/~lda/str/str.htm - Página da disciplina de Sistema de Tempo Real

[3] http://www.htsoft.com – Home page da HI-TECH Software