PICMiK – PIC Micro Kernel

André Quinta, Rudolfo Andrade

***Resumo* – Este artigo descreve um pequeno kernel para o microcontrolador PIC16F876, da Microchip, construído pelos autores no âmbito da disciplina de Sistemas de Tempo Real do 5º ano das LEET e LECT. O kernel PICMiK, não preemptivo, tem a particularidade de utilizar muito poucos recursos, 96 bytes de RAM e 1064 words de código, o que é fundamental para ser aplicado ao microcontrolador referido.**

Um dos objectivos ao construir este kernel é o de disponibilizar suporte multitarefa para as aplicações desenvolvidas pelos participantes no Concurso Micro-Rato da UA, no qual tem vindo a ser utilizada uma plataforma baseada no referido microcontrolador.

I. INTRODUÇÃO

Um sistema de tempo real é aquele em que a resposta a um dado estímulo tem não só de ser logicamente correcta mas também produzida em tempo útil. Por isso, quando temos uma aplicação que usa várias tarefas concorrentes, não basta termos um processador rápido, é necessário escalonar as tarefas e gerir os respectivos acessos a recursos partilhados de forma a minimizar os atrasos e controlar as interferências mútuas.

Uma das aplicações típicas de sistemas deste tipo é no controlo em malha fechada de processos físicos, em que existem várias tarefas peródicas que amostram variáveis do ambiente e determinam actuações adequadas. Um exemplo é o controlo de pequenos robôs autónomos e móveis, em que diferentes comportamentos podem ser facilmente encapsulados em tarefas periódicas, automaticamente activadas pelo kernel [3].

Este artigo descreve um kernel de tempo-real que permite executar várias tarefas periódicas em que a respectiva activação é controlada de forma transparente pelo próprio kernel. A utilização de um kernel deste tipo representa uma simplificação substancial do processo de desenvolvimento da aplicação, retirando ao programador a complexidade da gestão temporal das tarefas.

Como muitas das opções tomadas no desenvolvimento do kernel são em muito determinadas pelas características do hardware, este artigo começa por descrever alguns aspectos relevantes do microcontrolador usado, o PIC16F876 da Microchip. Seguidamente, são discutidos aspectos da arquitectura do kernel PICMiK, nomeadamente a composição das estruturas de dados utilizadas para suportar a informação relativa a cada tarefa, os estados porque passa a execução das tarefas, e os escalonadores utilizados. São também apresentadas as funções utilizadas e disponibilizadas pelo kernel. Finalmente faz-se uma análise dos resultados obtidos com os vários escalonadores. Por último faz-se uma alusão a algumas melhorias que poderão ser introduzidas como trabalho futuro. Resta referir que este trabalho foi realizado no âmbito da disciplina Sistemas de Tempo Real, opção de 5º ano das LEET e LECT.

II. O MICROCONTROLADOR PIC16F876

O microcontrolador alvo é o PIC16F876 da Microchip [1], o qual tem vindo a ser amplamente utilizado em pequenos robôs móveis e autónomos, quer no Concurso Micro-Rato da UA quer noutras competições de robótica. Este microcontrolador possui uma arquitectura do tipo *Harvard* com 8 K*words* de memória livre para programa e 384 bytes para dados, divididos em 4 bancos. Possui ainda dois *timers* de 8 bits e um de 16 bits. Uma das principais restrições advém da utilização de um *stack* fechado e não acessível pelo programa, com 8 níveis, apenas para os endereços de retorno das funções invocadas. Por outro lado, as estruturas de dados em memória estão limitadas a um banco, ou seja, 96 bytes.

III. ANÁLISE DO KERNEL

A. Arquitectura

A principal opção tomada face ao tipo de stack utilizado foi a de não suportar preempção de tarefas. Desta forma é possível construir um kernel mais simples, com menores requisitos de memória e inerentemente livre de contenção no acesso a recursos partilhados (zonas críticas). Por outro lado, a não-preempção causa bloqueios às tarefas de maior prioridade, pela incapacidade de estas interromperem a execução de tarefas de menor prioridade que entretanto tenham iniciado execução. Estes bloqueios poderão ser pouco significativos se as tarefas da aplicação forem curtas relativamente ao período da tarefa mais rápida.

As tarefas podem estar num dos seguintes quatro estados, conforme ilustra a Fig. 1:

- **CREATED** – Estado da tarefa quando é criada;
- **IDLE** – Estado de uma tarefa enquanto espera pela reactivação periódica;
- **READY-TO-RUN** - Estado de uma tarefa após a reactivação periódica, à espera de ser executada;
- **RUNNING** – Estado de uma tarefa quando está a ser executada.

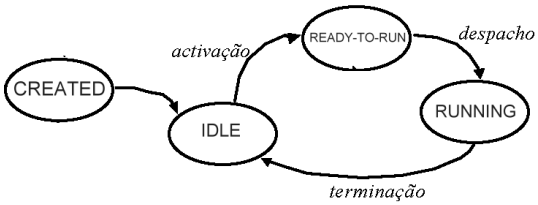

Fig. 1 - Diagrama de estados de uma tarefa

Toda a informação relativa a cada tarefa está guardada numa tabela de TCB's (*Task Control Block*s).

Cada TCB é uma estrutura com 6 campos:

- **Period** – período de reactivação cíclica da tarefa;
- **Deadline** – *deadline* relativa da tarefa, isto é, prazo dentro do qual a tarefa deve terminar após a sua activação;
- **Tickcounter** – *timer* de *software* que conta o tempo para a próxima activação da tarefa; quando uma tarefa é criada este *timer* é inicializado com o valor da fase relativa (primeira activação), nas reactivações subsequentes o *timer* é inicializado com o valor do período.
- **Aux** – *timer* semelhante ao anterior mas que conta o tempo que falta para ser atingida a deadline;
- **Status** – Estado da tarefa;
- **Func** – Contém o endereço do código da tarefa.
- **Next_element** – Apontador para o TCB da tarefa READY-TO-RUN com prioridade imediatamente inferior; só tem significado para tarefas que estejam naquele estado.

O último campo do TCB permite realizar uma lista ligada e ordenada de todas as tarefas que estejam prontas a executar (READY-TO-RUN), cujo primeiro elemento é o mais prioritário. Esta lista simplifica o despacho de tarefas, já que a próxima a ser executada é sempre a que está no topo da lista. Existe uma variável global que aponta para a tarefa no topo da lista. O fim da lista é indicado pelo valor **FIMLISTA**.

Para escalonar as tarefas construíram-se quatro escalonadores diferentes, dos quais os três primeiros seguem critérios baseados em prioridades fixas e o quarto em prioridades dinâmicas:

- **RM (Rate Monotonic)** – As tarefas mais prioritárias são as que têm menor período;
- **DM (Deadline Monotonic)** – As tarefas mais prioritárias são as que têm menor deadline;
- **FP (generic Fixed Priorities)** – O utilizador é que define a prioridade de cada tarefa ao criá-la. A ordem de criação estabelece a prioridade.
- **EDF (Earliest Deadline First)** – EDF ordena as tarefas conforme o tempo que falta para a *deadline* de cada tarefa;

O algorítmo de escalonamento deve ser escolhido em tempo de projecto, sendo a aplicação compilada já com um determinado escalonador.

A gestão de tempo é efectuada com recurso a uma interrupção periódica, o TICK, gerada pelo timer de 16 bits do PIC16F876. A rotina de atendimento do TICK actualiza os *timers* das tarefas, determina as mudanças de estado de IDLE para READY-TO-RUN e detecta a perda de *deadlines*.

B. Funções

O manuseamento das estruturas do kernel PICMiK, bem como os serviços fornecidos ao utilizador, são efectuados dentro de um conjunto de funções do sistema. A função *init_system( )* deverá ser a primeira a ser invocada e faz a inicialização de todas as variáveis e estruturas de dados do sistema. Seguidamente devem ser criadas as tarefas através da função *create_task( ).* Após esta fase inicial, deverá invocar-se a função *start_all( )* que dá início à gestão temporal das tarefas tendo em conta as respectivas fases relativas expressas nos instantes das primeiras activações.

De seguida a função *main()* entra num ciclo infinito onde está constantemente a verificar se existe alguma função pronta a executar (topo da lista ligada com valor diferente de FIMLISTA). Se existir, é invocada a função *dispatch( )* que lança em execução a primeira tarefa da lista (Fig. 2), de forma semelhante à invocação de uma qualquer função. Quando a tarefa termina retorna à função *dispatch( )* que actualiza o estado da tarefa para IDLE e termina.

Fig. 2 - Estrutura de uma aplicação.

Sempre que ocorre um TICK a execução das tarefas é interrompida pela respectiva rotina de serviço, a função *tick( ).* Esta função actualiza, ou seja decrementa, todos os campos *tickcounter* e *Aux* das tarefas READY-TO-RUN ou RUNNING. Caso o campo *Aux* seja igual a zero é activado o bit MISSDEADLINE (Fig. 3).

Sempre que ao actualizar o estado de uma tarefa ela passe de IDLE para READY-TO-RUN, isto é, for activada, a função *tick( )* invoca a função de escalonamento *scheduler( ).* Esta função insere a nova tarefa na lista ordenada de tarefas READY-TO-RUN de acordo com a respectiva prioridade quer fixa (RM, DM e FP) quer dinâmica (EDF). Neste último caso, o valor instantâneo da prioridade é o que consta no campo *Aux* do

TCB, isto é, o tempo que falta para a *deadline* expirar. Note-se ainda que a função *tick( )* neste caso deverá necessariamente actualizar os *timers* de todas as tarefas antes de invocar a função de escalonamento.

Finalmente, o *kernel* disponibiliza ainda a função d*eadline_status( )*, que permite à própria tarefa verificar se ultrapassou ou não a respectiva *deadline*.

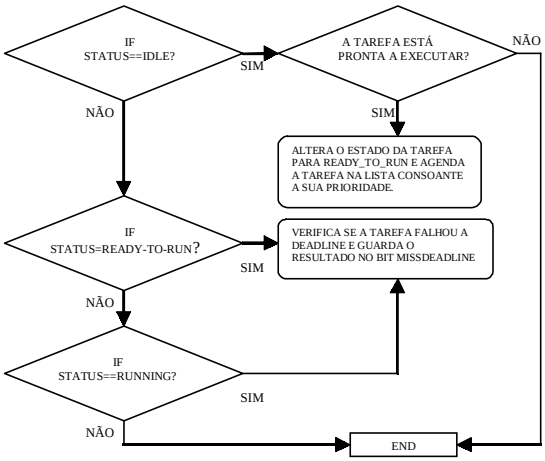

Fig. 3 - Função Tick( ) - (em EDF a prioridade instantânea é indicada no campo Aux do TCB)

IV. UTILIZAÇÃO DE RECURSOS E DESEMPENHO

Em termos de ocupação de memória, cada TCB ocupa 8 bytes, utilizando parâmetros de 1 byte apenas, com excepção do endereço do código que ocupa 2 bytes. Por sua vez, as varáveis internas do kernel ocupam 24 bytes. A distribuição pelos bancos de memória é a seguinte: 16 bytes no banco 0 e, limitando o número de tarefas a 9, 80 bytes no banco 1. No total, são utilizados 96 bytes, isto é 25% da RAM do PIC.

Por sua vez, o código do kernel ocupa no máximo, correspondente ao escalonador EDF, 1064 *words*, isto é 13% das 8 *Kwords* de espaço de código.

Em termos de *overhead*, foram feitas medições de pior caso do tempo de execução da função *tick( )*, utilizando tarefas activadas em simultâneo, isto é, em fase, com um período igual a 10 TICKs. Os tempos medidos para os vários escalonadores, com 20MHz de clock, estão representados na Fig. 4. Ao contrário do habitual, e devido à implementação baseada em lista ordenada, os escalonadores EDF, RM e DM não apresentam diferenças significativas. Apenas o escalonador de prioridades fixas genéricas apresenta tempos mais baixos o que se deve a utilizar um código ligeiramente mais simples. O pior caso absoluto medido foi de 720µs, correspondendo a 7,2% de *overhead* com um TICK de 10ms.

Em termos de resolução temporal, com um TICK de 10ms e parâmetros de 1 byte é possível especificar períodos de activação, bem como instantes de primeira activação, de 10ms a 2.5s. Também com um TICK desta dimensão, os bloqueios causados pela não preempção não deverão ser significativos já que as tarefas típicas no controlo reactivo de pequenos robôs são curtas e deverão executar em bastante menos do que 10ms.

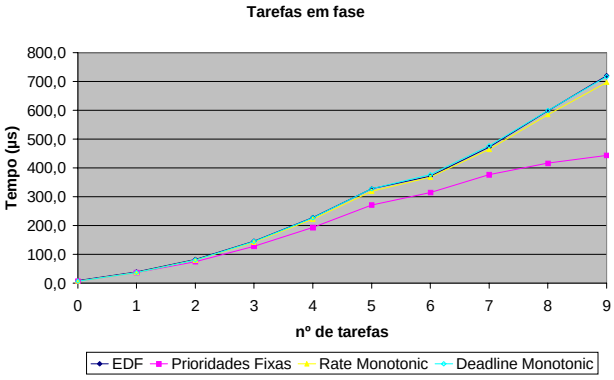

Fig. 4 – Tempo de execução da função tick( ) usando tarefas em fase.

V. CONCLUSÃO

Este artigo descreveu a estrutura e funcionamento de um micro kernel *multitasking*, sem preempção, para o microcontrolador PIC16F876 da Microchip. Este trabalho foi realizado no âmbito da opção de 5º ano da LEET e LECT intitulada Sistemas de Tempo-Real.

O kernel desenvolvido foi baptizado com o nome PICMiK e utiliza apenas uma pequena fracção dos recursos do microcontrolador, sendo adequado a suportar aplicações com um máximo de 9 tarefas periódicas e com um tempo de execução curto relativamente ao menor período.

O kernel PICMiK está neste momento disponibilizado aos participantes do Concurso Micro-Rato 2004 esperando-se que venha a ser utilizado por algumas equipas. Como trabalho futuro poderá introduzir-se o suporte à definição de pontos de preempção nas tarefas, de modo a permitir conciliar tarefas longas com tarefas curtas e rápidas.

REFERÊNCIAS

[1] *Microcontrollers PIC16F87X Datasheet*. The Microchip Worldwide Site. Disponível em http://www.microchip.com.

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

[3] Página do *ReTMiK – Real-Time Micro-rato Kernel*, http://sweet.ua.pt/~lda/retmik/retmik.htm