PicMiK - Pic Micro Kernel

Breve manual

Luís Almeida, 2004/04/21

1- Introdução

O kernel PicMik foi desenvolvido em finais de 2003 no âmbito da disciplina de Sistemas de Tempo-Real [1], pelos alunos Rudolfo Andrade e André Quintas. A motivação para o desenvolvimento do PicMik foi a de fornecer uma estrutura de software que facilite o desenvolvimento de múltiplos comportamentos cíclicos para pequenos robôs, como os que concorrem ao Concurso Micro-Rato da UA. No fundo, o PicMik segue na linha do anterior RetMik [2], um kernel multi-tarefa disponibilizado aos participantes do Concurso Micro-Rato desde 1998 e destinado à plataforma Kit188.
O kernel PicMik é simples e permite gerir a execução concorrente de até 9 tarefas, activando-as aos ritmos especificados pelo utilizador. Cada tarefa está associada a uma função escrita em C, que representa o respectivo código. A execução dessa função é despoletada automaticamente pelo kernel, de forma completamente transparente para o programador. A utilização do kernel adequa-se a situações do tipo: "pretende-se executar a função A todos os 40ms, a função B todos os 500ms, a função C todos os 90ms e a função D todos os 40ms mas sempre 10ms depois da função A". O programador apenas tem que fornecer o código de cada função e especificar o período e a fase com que pretende que a função seja executada.
Os recursos do processador utilizados plo kernel são, aproximadamente 1064Words de código, 16 bytes no Bank0 e 80 bytes no Bank1.
Este breve manual fornece uma breve explicação das funções disponibilizadas pelo kernel PicMik bem como sobre o processo de geração de aplicações.

2 - A livraria *picmik.c*

Esta livraria contém as funções do kernel (ou executivo) PicMik que suportam a criação e gestão de tarefas independentes e periódicas. Foi escrito para a plataforma PIC16F876, inteiramente em C, utilizando o compilador da Hi-Tech. Basicamente, este kernel permite transformar normais funções de C em tarefas periódicas cuja activação e execução concorrente é controlada pelo próprio kernel e completamente transparente para o utilizador.
As tarefas não admitem preempção o que quer dizer que quando uma tarefa inicía execução, executa até ao final, podendo apenas ser interrompida por interrupções ao processador. A execução das tarefas é efectuada por ordem das respectivas prioridades, que são fixas, atribuídas automaticamente de forma decrescente, de acordo com a ordem da criação das tarefas. Para melhores resultados sugere-se criar primeiro as tarefas com períodos menores e por último as que tiverem períodos mais longos.
As várias tarefas podem comunicar através de variáveis globais. A resolução temporal do *kernel* (tempo que demora um *tick*) é um parâmetro de entrada da função de inicialização *init_system()*.
Ao criar-se cada tarefa é necessário fomecer dois parâmetros que descrevem o respectivo período e o instante da primeira activação para permitir controlar a fase de activação entre tarefas com o mesmo período.

2.1 - Funções disponíveis

void init_system (int tick_in_ms);

Inicía o *kernel*, i.e., as respectivas estruturas internas bem como as interrupções de relógio e o respectivo *handler*. O sistema usa o *timer1* do Pic16 de forma que é necessário ter cuidado nas aplicações para não desactivar ou reprogramar este timer.
O parâmetro de entrada *tick_in_ms* especifica a duração dos *ticks* em múltiplos de 10 milisegundos (tipicamente usa-se o valor 1, i.e. tick = 10ms). Este parâmetro define a resolução temporal de todo o sistema o que quer dizer que os períodos, fase inicial e *deadline* são expressos em número de ticks.

unsigned char create_task (TASK1 *task_descript);

Cria uma tarefa deixando-a em estado latente, i.e., não é activada enquanto estiver neste estado.
O parâmetro de entrada é um ponteiro para uma estrutura do tipo TASK1 (*task_descriptor*) que possui os seguintes campos:

```
typedef struct {
  void (*func_entry)();
      ponteiro    para    a    função
      associada à tarefa
  char period;
      período de activação em ticks
```

```
char deadine;
    prazo para execução em ticks
    (tipicamente igual ao período)
char first;
    instante da primeira activação
    relativa ao "acordar da tarefa"
    pelo start_all, em ticks >=0
} TASK1;
```

O valor de saída é o identificador da tarefa acabada de criar (valor entre 0 e 8).

void start_all (void);

Conforme dito atrás, quando uma função é criada é colocada no estado latente, i.e., com a execução periódica desactivada (a tarefa nunca executa enquanto estiver neste estado). Esta função *start_all* permite iniciar sincronamente a execução periódica das tarefas. Isto não quer dizer que as tarefas iniciem execução logo após *start_all*. Quer dizer, sim, que a contagem do tempo (em *ticks*) para a primeira activação de todas as tarefas (que estavam em estado latente) começa assim que se chama a função *start_all*.

void execute_system (void);

Esta função mantém o sistema em funcionamento, lançando as tarefas em execução nos instantes apropriados. Deve ser a última linha de comando dentro da função *main()*.

unsigned char deadline_status (unsigned char tid);

Devolve uma informação binária sobre o cumprimento da *deadline* da tarefa *tid* no instante de chamada desta função. Se o valor devolvido for 1, houve perda de deadline, i.e. a tarefa demorou mais tempo a executar desde a respectiva activação periódica do que o especificado em *deadline*.

long get_abs_ticks(void);

Esta macro devolve o valor do contador absoluto de ticks, desde o arranque do sistema (*init_system()*). Permite medir intervalos de tempo com a resolução de ticks. Por exemplo, para saber se passaram 4 minutos desde um determinado instante, com um ticks de 10ms, lê-se o contador nesse instante e depois vai-se lendo o contador até que a diferença entre o valor lido agora e o lido inicialmente seja de 24000 ticks. Covém ter em atenção que o próprio processo de recarga do timer1 do Pic pode levar a pequenos erros pelo que convém dar um intervalo de segurança.

2.2 - Utilização do kernel *ReTMiK*

Para utlizar o *kernel* retmik deverá ter em atenção o seguinte:

- não usar o *timer1* do µP.
- em geral, não inibir as interrupções durante a execução das tarefas já que isso levaria à suspensão da contagem temporal do sistema e a erros nos períodos de activação das tarefas. Contudo, a inibição pode ser utilizada durante pequenos intervalos, por exemplo para garantir o acesso atómico a estruturas (ou variáveis) partilhadas.
- incluir o ficheiro picmik.*h* com as definições e declarações necessárias à utilização do sistema.
- escrever o código das tarefas sem ciclos globais. A repetição periódica das tarefas é completamente controlada pelo sistema.
- por seu lado, o *main()* deverá começar por chamar a função *init_system()*, criar as tarefas necessárias chamando a função *create_task()*, fazer outras inicializações, disparar a activação periódica das tarefas com *start_all()* e entrar na função de execução sistema *execute_system()*. Repare-se que o processamento necessário deve ser todo executado ao nível das tarefas.
- como o kernel não efectua preempção entre tarefas, i.e., as tarefas não se podem interromper umas às outras, é necessário ter cuidado com o respectivo tempo de execução. Note-se que uma tarefa muito longa, enquanto estiver em execução, não permite que mais nenhuma tarefa execute. Assim, convém tentar evitar tarefas que possam ser muito longas (vários ticks de duração). Em particular, é necessário que o maior tempo de execução de entre todas as tarefas seja mais curto do que o menor período de activação. Caso contrário, a tarefa mais longa poderá não permitir a execução da tarefa mais rápida (menor período). Para os algorítmos típicos usados nos Micro-Ratos não deverão ser necessários cuidados especiais. Apenas o de evitar a utilização das funções de escrita de strings na porta série, que poderão facilmente demorar muitos milisegundos.

3 - Geração de código

O programa de controlo do robot, depois de escrito em C, deverá ser compilado usando o compilador da Hi-Tech incluindo no projecto o ficheiro **picmik.c** (e outras livrarias eventualmente necessárias, e.g. *robot.c*). Não esquecer de fazer o *include* do ficheiro **picmik.h**
O ficheiro resultante inclui a aplicação com o próprio kernel integrado. E está pronto a carregar, por exemplo com o WinPicLoader, e a executar...

[1] http://sweet.ua.pt/~lda/str/str.htm
[2] http://sweet.ua.pt/~lda/retmik/retmik.html