Modelo de Concorrência e Mecanismos de Gerência de Tarefas do TinyOS

Pedro Rosanes

28 de novembro de 2010

1 Sequência de Inicialização

O principal componente do TinyOS responsável por botar o sistema no ar é o *MainC*. Para isso, ele inicializa o escalonador de tarefas, os componentes de hardware e software. Primeiro é configurado o sistema de memória e escolhido o modo de processamento. Com esses pré-requisitos básicos estabelecidos é inicializado o escalonador, para permitir que as proximas etapas possam postar tarefas.

O segundo passo é inicializar o hardware como um todo, permitindo a operabilidade da plataforma. Alguns exemplos são configuração de pinos de entrada e saida, calibração do clock e dos LEDs. Como esta etapa exige códigos específicos para cada tipo de plataforma, o *MainC* se liga ao componente *PlataformC* que implementa o programa necessário.

O terceiro passo trata o software. Além de configurar os aplicativos básicos do sistema, como o temporizador, também é preciso inicializar os programas do usuário. Portanto, se um componente do usuário precisa ser inicializado basta amarrá-lo ao *SoftwareInit*. Assim o TinyOS se responsabiliza por executar este código.

Por ultimo, quando tudo é concluido, o MainC sinaliza que a inicialização concluiu, através do sinal *Boot.booted()*. Isso permite que os componentes rodem. E finalmente, o TinyOS entra no seu laço principal, no qual o escalonador espera por tarefas, e as executa. É importante notar que durante todo este processo, as interrupções do sistema ficam desabilitadas.

2 Mecânismos de Gerência de Tarefas

Tasks, ou tarefas, têm duas propriedades importantes. Elas não são preemptivas entre si, e são executadas de forma adiada. Isso significa que ao postar uma tarefa, o fluxo de execução continua, sem desvio, e ela só será processada depois. As *tasks* básicas não tem parâmetro, apesar de ser possível fazê-las receber parâmetros, criando uma interface e amarrando-a ao componente do escalonador.

O responsável por gerenciar e escalonar tarefas no *TinyOS* é o componente *TinySchedulerC*. O escalonador padrão adota uma política *First-in First-out* para agendar as tarefas. Ele também cuida de parte do gerenciamento de energia, pois bota a CPU em um estado de baixo consumo se não há nada para ser executado.

É possível mudar a política de gerenciamento de tarefas substituindo o escalonador padrão. Para isto, basta adicionar uma configuração com o nome *TinySchedulerC* no diretório da aplicação e amarrá-la ao componente responsável pela implementação. Qualquer novo escalonador tem de aceitar a interface das *tasks* padrões, e garantir a execução de todas as tarefas, sem permitir *Starvation*

3 Modelo de Concorrência

O *TinyOS* mantém os problemas de concorrência bem simples, qualquer possível condição de corrida é detectada em tempo de compilação. Para que isso seja possível, o código em nesC é dividido em dois tipos:

Código Assíncrono Código alcançável a partir de pelo menos um tratador de interrupção.

Código Síncrono Código alcançavel somente a partir de *tasks*.

Eventos e comandos que podem ser sinalizados ou chamados a partir de um trador de interrupção são códigos assíncronos. Eles podem interromper outros eventos, comandos e *tasks*. Por isso devem ser marcados como *async* no código fonte. O problema aparece quando variáveis compartilhadas são acessadas por esse tipo de código. Para contornar isso, deve-se usar o comando *atomic* ou *Power locks*.

O comando *atomic* permite que um trecho de instruções possa ser executado sem ser interrompido. Dois fatos importantes surgem com o seu uso, primeiro a ativação e desativação de interrupções consome ciclos de CPU. Segundo, longos trechos atômicos podem atrasar outras interrupções, portanto é preciso tomar cuidado ao chamar outros componentes a partir desses blocos.

Algumas vezes é preciso usar um certo hardware por um longo tempo, sem compartilhá-lo. Como a necessidade de atomicidade não está no processador e sim no hardware, pode-se conceder sua exclusividade a somente um 'usuário' através de *Power locks*. Para isso, primeiro é feito um pedido através de um comando, depois quando o recurso desejado estiver disponível, um evento é sinalizado. Assim não há bloqueio de execução, como em semáforos. Existe a possibilidade de requisição imediata. Nesse caso nenhum evento será sinalizado: se o recurso não estiver protegido, ele será imediatamente cedido, caso contrário, o comando retornará falso. *Power Locks* têm três sub-componentes: Um abitrador que gerência as prioridades dos pedidos, um gerênciador de energia e um configurador que ajusta o hardware de acordo com o cliente.

Referências

[1] Tutorial 6, Boot Sequence. http://docs.tinyos.net/index.php/Boot$_S$*equence*

[2] P. Levis e C. Sharp. TEP 106: Schedulers and Tasks. http://www.tinyos.net/tinyos-2.x/doc/html/tep106.html

[3] P. Levis. TEP 107: TinyOS 2.x Boot Sequence. http://www.tinyos.net/tinyos-2.x/doc/html/tep107.html

[4] P. Levis e D. Gay. TinyOS Programming, capítulo 11, 2009.