Implementação de contadores de tempo em threads simultâneas usando o kernel Elevescall no MARS

Carolina Adilino, Ana Julia Botega

Departamento de Informática e Estatística – Universidade Federal de Santa Catarina
Caixa Postal 5094 – 88035-972 – Florianópolis – SC – Brasil

Resumo. *Esse trabalho visa apresentar o programa desenvolvido pelos autores, cujo objetivo é implementar dois contadores de tempo, um que seja incrementado a cada segundo, e outro a cada 3 segundos. Para que sejam implementados de maneira simultânea, serão usadas duas threads: uma para cada contador. Para realização desta concorrência, será utilizado o kernel Elevescall, disponibilizado pela disciplina. O resultado será mostrado no Digital Sim Lab, ferramenta disponibilizada pelo MARS.*
Palavras-chave: *Assembly, MIPS, concorrência, contador*

1. Objetivo

Implementar um programa em Assembly para o MARS que utilize um escalonador de tarefas de um pequeno kernel - Elevescall para lançar duas threads, uma que irá implementar um contador de 1 segundo e a outra que irá implementar um contador de 3 segundos.

2. Materiais e métodos

Primeiramente, o código do kernel foi disponibilizado pela disciplina e consiste em 3 arquivos: kernel.asm, labels.asm e prog.asm. Este documento não entrará em detalhes sobre o funcionamento destes. A única alteração necessária se mostra nas linhas sob o título “configuração”. Aqui foram adicionados os nomes das threads de cada arquivo (contador1seg e contador3seg), e a quantidade de threads que serão lançadas.

.eqv thread_v p1, p2, p3, contador1seg, contador3seg

.eqv max_th 5

Agora indo para os arquivos nomeados thread_1 e thread_2, é necessário utilizar a diretiva .globl, já que essa está relacionada ao ligador (linker) do MARS. A fase do ligador, em uma compilação de código normal, é a responsável por unir diversos arquivos em um só arquivo de código de máquina para a plataforma alvo.

O código dos contadores é simples, começa carregando o valor de zero no registrador $t0, esse vai ser nosso contador. Em seguida, iniciamos o loop principal *loop_1s*. É aqui que fazemos uma chamada de sistema com o comando 30, para usar a base de relógio do computador e permitir que seja lido o valor do relógio a cada 1 ms. Esse valor é salvo no registrador $t1. O próximo loop chama *espera1*, e irá fazer a chamada de sistema novamente, e salvar o novo tempo no registrador $t2. Através de uma subtração, dos valores de $t1 e $t2, temos a diferença de tempo milissegundos do início do *loop_1s* e do início do loop *espera1*. Usando a instrução *blt (branch if less than)* esse valor é comparado com 1000 (no caso do contador de 1 segundo) e com 3000 (no caso do contador de 3 segundos). Caso a diferença entre o início do primeiro e do segundo loop seja menor que esse valor, o programa pula para *espera1* e repete esse procedimento.

Quando um segundo (ou 3, no caso do programa *thread_2*) tiver passado, ao invés de fazer esse pulo, o programa segue em frente somando 1 ao registrador $t0. Como iremos usar apenas um dígito do Digital Sim Lab, mostra-se necessário comparar esse valor com 10 e, caso seja igual ou maior que, é substituído por 0. O próximo passo é mover o valor de $t0 para $a0, para que seja passado como argumento do procedimento que iremos chamar, *mostra_display_direito* (ou esquerdo, em *thread_2*).

Nesse bloco de código, carregamos o endereço do display direito do Digital Sim Lab e o endereço da tabela de 7 segmentos que foi carregada na memória previamente. Em $t1, é salvo o valor da soma da tabela com nossa variável $a0, e depois é salvo no endereço $t5 através de uma instrução store byte. Vale notar que no código do arquivo *thread_2*, nosso contador de 3 segundos, está o endereço do display esquerdo do Digital Sim Lab.

Voltando ao bloco principal de código com uma instrução *jr $ra*, o último comando é um jump para o início do código, assim, repetindo o procedimento como um contador efetivo.

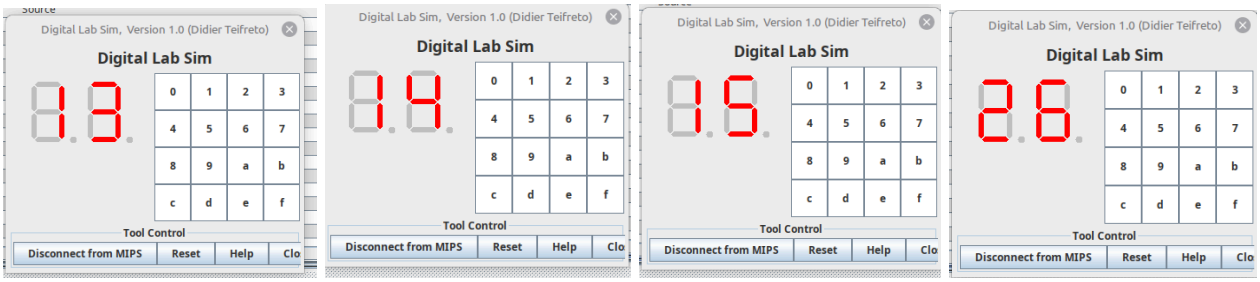

Fig 1, 2, 3 e 4 - exemplo de funcionamento do programa

3. Resultados e discussões

A execução do programa foi bem-sucedida, com todas as etapas ocorrendo de maneira esperada. Durante a fase de desenvolvimento, o valor de todos os registradores foi acompanhado e os resultados obtidos pelo código foram conferidos.

5. Conclusões

Esse trabalho foi desenvolvido com o objetivo principal de estudar a capacidade dos alunos de desenvolver códigos concorrentes em assembly usando o kernel Elevescall. Além disso, aprender a fazer o acompanhamento de tempo através de contadores também se mostrou útil para a disciplina, pois a lógica pode ser reaproveitada para o trabalho final. Em conclusão, tais metas foram alcançadas com êxito, de modo que, tanto a concorrência quanto os contadores, foram implementados e executados de modo satisfatório.