Uma biblioteca de threads $N \times M$ *

Lucas Correia Villa Real†, Gerson Geraldo H. Cavalheiro

Programa Interdisciplinar de Pós-Graduação em Computação Aplicada
Centro de Ciências Exatas e Tecnológicas
Universidade do Vale do Rio dos Sinos
São Leopoldo – RS – Brasil
{lucasvr, gersonc}@exatas.unisinos.br

Palavras-chave: threads, alto desempenho, Linux, paralelismo.

Introdução

No contexto de processamento de alto desempenho, é desejável a existência de recursos que possibilitem a exploração máxima das arquiteturas existentes, dado a necessidade da realização de cálculos freqüentemente pesados e repetitivos. Para explorar os recursos de arquiteturas multi-processadas, é comum a divisão das partes críticas dos programas, submetendo-os a fluxos independentes de execução: eles recebem um dado de entrada, executam muitas operações sobre eles e produzem um dado de saída.

O recurso padrão disponível no GNU/Linux para a criação de novos processos é através da chamada `fork()`. No entanto esta chamada apresenta um overhead muito grande, pois existe o custo de criação e manipulação de um processo comum. O kernel Linux suporta desde 1996 uma implementação de *threads* baseada no padrão especificado pelo POSIX 1003.1c[IEE 94], intitulada LinuxThreads. Diferente da chamada `fork()`, a implementação LinuxThreads usa a chamada `clone()`, uma função posicionada logo acima da chamada de sistema `sys_clone()`, para realizar a criação de novos fluxos de execução. Esta biblioteca implementa o modelo de *threads* conhecido como 1×1, onde para cada *thread* de usuário criada existe uma *thread* de sistema para gerenciá-la.

Este artigo apresenta uma solução para a criação de processos leves que utiliza o modelo de *threads* adotado pela LinuxThreads como base para a implementação de *processadores virtuais*, que têm como objetivo escalonar e executar tarefas criadas pelo usuário, vistas neste ambiente como *threads* de usuário. Este modelo de descrição de *threads* também é conhecido como um modelo $N \times M$, onde para N processadores virtuais (implementados sob a forma de *threads* de sistema) existem associadas M tarefas de usuário. O artigo apresenta ainda algumas medidas de desempenho obtidas em operações básicas desta solução (criação e sincronização de tarefas), comparando-os com os tempos consumidos para as mesmas operações sobre a biblioteca LinuxThreads.

*Apoio: CNPq, UNISINOS
†ITI-CNPq

Threads Anahy

A chamada `clone()` realiza a criação de um novo processo, permitindo que o processo filho compartilhe recursos do seu contexto de execução com o processo pai, como a tabela de descritores de arquivos, espaço de endereçamento de memória e manipuladores de sinais. Isto permite obter um modelo mais adequado às necessidades de aplicações que requerem um processamento de alto desempenho, pois os custos para manter os vários fluxos de execução da aplicação serão reduzidos [TAN 2001].

No entanto, ainda assim estes fluxos de execução são caros quando deseja-se extrair o máximo de desempenho de uma arquitetura multi-processada. Tomando o kernel Linux como exemplo: as *threads* criadas desta forma são representadas por entidades independentes no kernel, e nele são escalonadas *junto* com processos normais do sistema. Somando ao fato de que o escalonador presente na série 2.4 do Linux tem complexidade na ordem de $O(n)$, torna-se cara a manutenção de muitas *threads* neste sistema.

O modelo de *threads* N×M implementado – *threads* Anahy – baseia-se no uso de *threads* POSIX para implementar processadores virtuais. Estes processadores virtuais consistem em fluxos de execução responsáveis pelo escalonamento de tarefas criadas pelo usuário, apresentando o mesmo papel de processadores reais na arquitetura base.

As tarefas Anahy consistem de uma estrutura semelhante à utilizada pelas *threads* LinuxThreads: um identificador único da tarefa, atributos de execução, um ponteiro para a função que será executada pela tarefa, um ponteiro para dados de entrada e outro para armazenar os dados por ela produzidos [DAL 2003, CAV 2003].

O armazenamento lógico (em memória) de tarefas seguindo o modelo proposto por Anahy envolve a dependência de dados entre elas, como é visto na Figura 1: tarefas criadas para produzir dados para outras são criadas como *filhas* delas, de forma a serem sincronizadas pela tarefa pai e entregar-lhes o resultado de seu cálculo. Caso uma tarefa crie várias outras, estas serão vistas como *irmãs*. Este modelo de armazenamento implica na criação e manutenção de tarefas em uma árvore binária, onde a complexidade computacional para o escalonamento de tarefas é da ordem de $O(n/2)$. No entanto, esta busca ocorre em um espaço limitado, pois cada processador virtual possui a visão de uma determinada região desta árvore, a qual contém as tarefas às quais ele se encontra oferecendo suporte à execução. Cada processador virtual conta ainda com uma pilha de execução, que funciona como um registro de ativação: cada tarefa (mapeada na árvore) a ser executada é inserida no topo da pilha, sendo removida ao término de sua execução. Assim, para encontrar a tarefa que será sincronizada, basta realizar a busca sobre a sub-árvore gerada a partir da tarefa que encontra-se no topo desta pilha.

Desempenho

O fato de não precisar criar uma entidade em kernel para cada tarefa Anahy permite que o tempo para criação de uma *thread* fique limitado apenas às rotinas da biblioteca Anahy, que encontra-se em execução em espaço de usuário. Logo, o tempo de criação de uma *thread* Anahy indica o *overhead* que a biblioteca enfrenta para alocar suas estruturas e inserir a tarefa na árvore, conforme a política ilustrada na Figura 1. Da mesma forma, o tempo levado para a criação de uma *thread* implementada pela LinuxThreads equivale ao

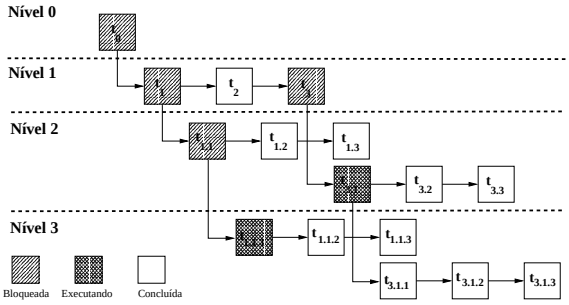

Figura 1: Grafo de Dependências entre Tarefas de uma Aplicação

tempo de alocação de suas estruturas em espaço de usuário, além da chamada `clone()` e de sua inserção na lista de processos do kernel Linux.

Esta seção mostra os testes de desempenho realizados, visando obter um paralelo da implementação da biblioteca de *threads* Anahy com as LinuxThreads, focado nas operações básicas de manipulação de *threads*. Além disso, é apresentado o tempo necessário para realizar a sincronia com as tarefas (*join*) em ambas bibliotecas. Vale salientar que neste caso o ambiente Anahy verifica se a tarefa já foi executada; caso tenha sido, o resultado produzido pela tarefa é retornado; caso não tenha sido, a tarefa é executada; e caso esteja em execução (e exista mais de um processador virtual em uso), o processador virtual aguarda até que o outro execute a tarefa para então realizar o *join*. A implementação de *threads* da LinuxThreads, por sua vez, apenas retorna o resultado gerado pela tarefa, ou aguarda pelo término de sua execução.

As medidas de desempenho apresentadas na Tabela 1 mostram o tempo consumido para a criação de *threads* com a biblioteca de *threads* Anahy sobre tarefas (funções) vazias, com a média obtida sobre 4000 iterações. Nestas medidas foi utilizado o kernel 2.4.21 do Linux, com a Glibc 2.3.1 (a biblioteca LinuxThreads está agora integrada à ela), em uma arquitetura UP (*uni-processor*), composta por processadores Pentium 4 de 2.4 Mhz, com 512 MBs de RAM. Para efeito de comparação, os tempos obtidos nesta mesma arquitetura com a biblioteca LinuxThreads para a operação `pthread_create` foi de 0,0556μs, com desvio-padrão de 0,0378μs. A operação `pthread_join` custou 0,0065μs, com desvio-padrão de 0,0055μs.

Os mesmos testes foram realizados sobre uma arquitetura XEON 2.4 Mhz biprocessada e com *Hyper-Threading*, com 2.5 Gb de RAM. Os resultados obtidos no XEON são vistos na Tabela 2. Nota-se a estabilidade mantida nos tempos de criação e sincronização das *threads* Anahy, mesmo com os gastos com a manutenção dos processadores virtuais. Os tempos obtidos na arquitetura SMP para a biblioteca LinuxThreads para a operação `pthread_create` foi de 0,0597μs, com desvio-padrão de 0,1540μs, e de 0,0063μs para a operação `pthread_join`, com desvio-padrão de 0,0081μs.

Nota-se que a soma dos tempos da criação da *thread* e da sincronização com seu resultado determina o tempo total gasto para criá-la, executá-la e recuperar os dados produzidos pela tarefa. Neste caso, o ambiente Anahy apresentou um ganho considerável sobre a biblioteca LinuxThreads, que pode ser justificado em parte por não haver o gasto

Tabela 1: Tempos de criação e sincronização de *threads* em arquitetura UP (medidas em μs)

	1 Proc.Virtual		2 Proc.Virtuais		3 Proc.Virtuais		4 Proc.Virtuais	
	Média	DesvPad	Média	DesvPad	Média	DesvPad	Média	DesvPad
athread_create	0,0027	0,0037	0,0028	0,0021	0,0029	0,0088	0,0030	0,0051
athread_join	0,0034	0,0196	0,0031	0,0142	0,0031	0,0139	0,0029	0,0015

Tabela 2: Tempos de criação e sincronização de *threads* em arquitetura SMP (medidas em μs)

	1 Proc.Virtual		2 Proc.Virtuais		3 Proc.Virtuais		4 Proc.Virtuais	
	Média	DesvPad	Média	DesvPad	Média	DesvPad	Média	DesvPad
athread_create	0,0045	0,0036	0,0050	0,0039	0,0059	0,0041	0,0052	0,0041
athread_join	0,0050	0,0038	0,0048	0,0044	0,0049	0,0037	0,0046	0,0037

com comunicação com o kernel, pois a biblioteca Anahy mantém seu código em espaço de usuário, permitindo acesso imediato para a manipulação das *threads*.

Conclusão

Este artigo apresentou uma biblioteca de *threads* de usuário implementada sobre *threads* de sistema. Os resultados vistos mostram que a biblioteca de *threads* em Anahy consegue implementar com bom desempenho *processos leves*, validando a proposta de obter uma implementação de *threads* voltada para o processamento de alto desempenho.

Atualmente as *threads* Anahy encontram-se em estágio de adaptação ao conjunto de normas especificadas pelo padrão POSIX para *threads*.

Referências

[CAV 2003] CAVALHEIRO, G. G. H.; VILLA REAL, L. C.; DALL'AGNOL, E. C. Uma Biblioteca de Processos Leves para a Implementação de Aplicações Altamente Paralelas. In: WSCAD 2003, 2003, São Paulo, Brasil. **Anais...** [S.l.: s.n.], 2003.

[DAL 2003] DALL'AGNOL, E. C. et al. Portabilidade na Programação para o Processamento de Alto Desempenho. In: WSCAD 2003, 2003, São Paulo, Brasil. **Anais...** [S.l.: s.n.], 2003.

[IEE 94] IEEE Computer Society. **American National Standards Institute: ieee standard for information technology: portable operating system interface (posix). part 1, system application program interface (api) – amendment 1 – realtime extension [c language]**. Silver Spring: [s.n.], 1994.

[TAN 2001] TANEMBAUM, A. (Ed.). **Modern Operating Systems, Second Edition**. [S.l.]: Prentice Hall, 2001.