Tópicos em Ciência da Computação

Plano de Estudos Depuração de sistemas paralelos e distribuídos

Giuliano Mega
Orientador: Fábio Kon
Yoshiko Wakabayashi

:: Introdução

O processo de depuração constitui, sem sombra de dúvidas, uma das atividades mais fundamentais no desenvolvimento de qualquer sistema de software; seja ele de pequeno, médio ou de grande porte. Sistemas de software, principalmente conforme crescem em tamanho e complexidade, costumam apresentar erros e comportamentos errôneos cujas fontes podem variar consideravelmente conforme o caso. Erros de lógica e/ou hardware são freqüentemente a causa dos problemas e, em todos os casos, é tanto de interesse dos desenvolvedores do software quanto dos usuários que esses comportamentos errôneos sejam identificados e corrigidos o quanto antes. É nesse cenário em que as ferramentas de depuração, velhas conhecidas de desenvolvedores no mundo todo, despontam como peça essencial.

Conforme dito anteriormente, as fontes de comportamento errôneo num sistema podem variar consideravelmente caso a caso. Além disso, variam também de sistema para sistema os aspectos de interesse numa análise de depuração, bem como a maneira pela qual os dados para essa análise são coletados. Num programa seqüencial, por exemplo, erros podem ser detectados através da análise de valores de variáveis ou pela inspeção da pilha de execução, que são acessíveis trivialmente através do acoplamento de um depurador local ao processo que se deseja depurar.

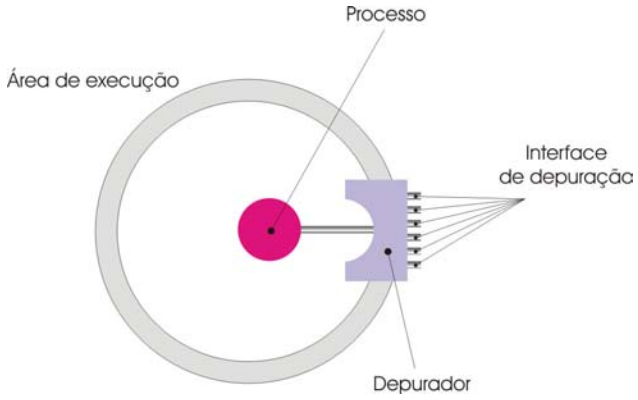

fig 1. Depuração no caso mais simples

A figura acima se aplica também a programas compostos por múltiplas linhas (*threads*) de execução, mas que executam numa única máquina (embora nesse caso haja múltiplas pilhas de execução independentes). A **área de execução** representa uma abstração onde programas que nela residem executam sem possibilidade de interferência por parte do usuário (pense nela como uma situação normal de execução numa estação de trabalho). Um dos papéis do depurador é, portanto, prover meios para que essas informações de tempo de execução (referidas doravante como informações de estado do programa), antes inacessíveis ao desenvolvedor (ou acessíveis por meios intrusivos, como o entrelaçamento de instruções de impressão no código-fonte do aplicativo), sejam disponibilizadas de maneira ampla, limpa, precisa e, em alguns casos, automática. Poderíamos, dentro da nossa abstração de **área de execução**, dizer que o papel do depurador é prover acesso a essa área especial, coletando e exportando as informações de execução e interagindo com os processos nela contidos (algo conhecido também conhecido como *trace-debug*).

Esse papel bastante geral do depurador se complica, obviamente, na medida em que novos aspectos e paradigmas são levados em consideração. Tomemos como exemplo o foco deste projeto de estudos – depuradores para aplicações distribuídas. Existem uma série de conceitos que surgem apenas no contexto de sistemas distribuídos – e que devem ser tratados por qualquer candidato a depurador de aplicações distribuídas – mas que inexistem ou são mais simples em sistemas centralizados (que executa num único nodo). Um exemplo de tal conceito é o de um *thread* distribuído. O conceito *thread* distribuído surge em sistemas que utilizam mecanismos de chamada remota de procedimentos, como por exemplo aplicações baseadas em *middleware* **CORBA**. Um *thread* distribuído ocorre quando um processo, **A,** executa uma chamada síncrona e bloqueante de um método remoto, digamos, num nodo arbitrário e diferente do nodo onde se localiza **A**.

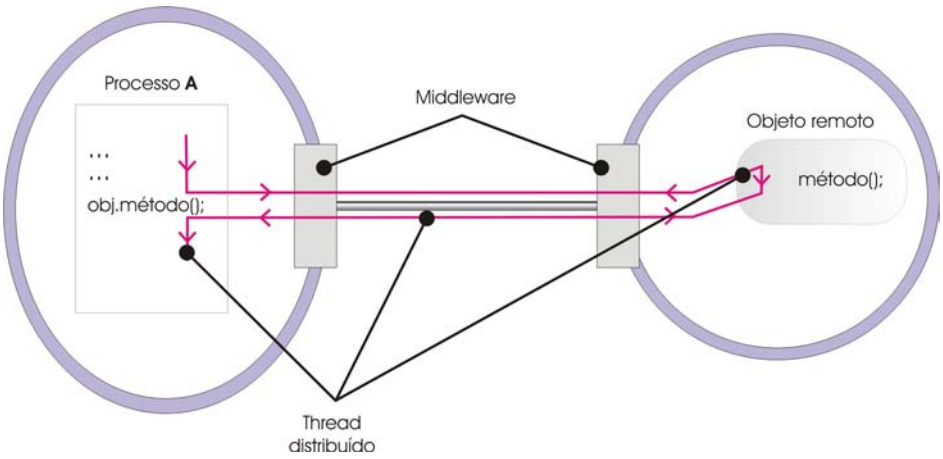

fig 2. *Thread* distribuído

Suponha que seja de nosso interesse rastrear esse *thread* distribuído. Note que esse é um conceito de alto nível. Os processos locais não têm conhecimento do *thread* distribuído e, mesmo se acoplássemos um depurador convencional ao processo cliente local, seria muito difícil determinar se uma dada chamada cria ou não um *thread* distribuído através da informação obtenível pelo depurador local (na máquina local teríamos, no máximo, acesso ao código que simula a chamada remota através de *sockets*, por exemplo). Mais do que isso, não existe sequer um consenso a respeito de como um *thread* distribuído se desenrola, em baixo nível, nos diferentes **ORB**s. O **ORB** no lado servidor, por exemplo, poderia despachar uma chamada de método num *thread* completamente diferente do *thread* que recebe a requisição, tornando o nosso *thread* distribuído uma entidade abstrata composta por três *threads*. Logo se vê que tentar rastrear em baixo nível algo tão simples quanto um *thread* distribuído é, para dizer um mínimo, difícil.

É bastante oportuno introduzir, neste ponto da discussão, dois pilares do nosso projeto: não-intrusão e generalidade. Os dois caminham de mãos dadas, já que a intrusão tende a quebrar o encapsulamento e requerir técnicas específicas e acopladas a implementações. Alcançar esses dois objetivos depende fortemente do quanto é possível levar a cabo esse paradigma de depuração de alto nível que, em muitos momentos, mistura-se com a depuração de baixo nível. É natural que nos venha à cabeça neste momento a pergunta: como observar conceitos abstratos, como o do *thread* distribuído, sem ter conhecimento de como esse *thread* se desenrola e nem da implementação do *middleware* envolvido? Pois muito bem, esse é o objetivo do nosso estudo.

:: Assuntos em pauta

O depurador distribuído para sistemas distribuídos (é isso mesmo, não é pleonasmo) tem como foco a plataforma Eclipse; isto é, pretendemos produzir, até o final do meu mestrado, um *plug-in* funcional para o Eclipse que implemente não só o rastreio de *threads* distribuídos e controle de fluxo em aplicações remotas, mas também uma série de outras funcionalidades muito mais complexas e de extrema importância, tais como *replay* de execução, detecção de predicados atômicos globais (estáveis, instáveis, fracos e fortes), bem como formas de visualização da execução de programas distribuídos através de representações gráficas e análise *post-mortem* (talvez).

Inicialmente, este depurador será voltado à depuração de sistemas que utilizem *middleware* **CORBA,** a começar por software Java. Isso envolve um estudo bastante aprofundado da especificação **CORBA** e suas possibilidades (que muito possivelmente serão exploradas ao limite para atingir ao máximo o nosso objetivo de generalidade e baixa intrusão) e das ferramentas de depuração voltadas a aplicações não-distribuídas, como a arquitetura de depuração Java e depuradores C++.

Temos, portanto, em pauta:

- Estudo da factibilidade da implementação de facilidades de rastreio em implementações **CORBA**.
- Estudo da plataforma de depuração Java (**JPDA**) e de alternativas equivalentes para C++ e outras linguagens (inicialmente só nos importa Java).

- Implementação do rastreio de *threads* distribuídos *Java-Java* e quem sabe *Java-C++*.
- Estudo acessório de modelos formais de sistemas distribuídos e algoritmos que nos auxiliem na direção correta.

É claro que a própria falta de maturidade no estudo do problema nos impede de fazer afirmações por demais categóricas a respeito do que vai ser estudado, mas é possível dizer que o foco vai ser nesses assuntos. O estudo envolve, obviamente, uma vasta pesquisa em resultados já publicados.

:: Resultados esperados

Como sugerido anteriormente, o plano de estudo consiste numa dissecação sistemática de problemas cuja solução deve resultar na construção incremental do depurador (difícil avaliar o ponto dessa construção até o fim do semestre). Os subprodutos dessas soluções - relatórios técnicos e informações diversas - serão então compostos num único relatório que servirá como base para o meu trabalho e como relatório desta disciplina. Esperamos até lá dispor de informações suficientes para que seja possível avaliar e definir um escopo mais preciso para o depurador. Esperamos também evoluir substancialmente em sua implementação.

:: Bibliografia

1. LYNCH, Nancy. *Distributed Algorithms,* Morgan Kauffmann Publishers, Inc., 1996.
2. HENNING, Michi e VINOSKY, Steve. *Advanced CORBA Programming with C++,* Addison-Wesley, 1999.
3. BROSE Gerald et. al. *Java Programming with CORBA*. John Wiley & Sons, 2001.
4. OMG. *The common object request broker:Architecture and specification*. OMG, Fevereiro de 1998.
5. LAMPORT, Leslie. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558-565, Julho de 1978.
6. DUCHIEN, Laurence e SEINTURIER Lionel, Reflection and Debug for CORBA Applications, Relatório Técnico para o CNAM-CEDRIC 99-10, Junho 1999, 15 páginas.
7. MAXIMILIÁN, Otta. A Distributed Debugger Framework Applicable in Java/CORBA Environment. *European Research Seminar on Advances in Distributed Systems*, Maio 2001, 6 páginas.
8. TARAFDAR, Ashis e GARG, Vijay K. Predicate Control for Active Debugging of Distributed Programs. Relatório Técnico ECE-PDS-1998-002, Parallel and Distributed Systems Laboratory, ECE Dept. University of Texas at Austin, 1998. disponível em http://www.maple.ece.utexas.edu.
9. NETZER, Robert H. B. Optimal Tracing and Replay for Debugging Shared-Memory Parallel Programs. *Proceedings of the Workshop on Parallel and Distributed Debugging,* páginas 1-10, ACM/ONR, 1993.
10. TOMLINSON, Alexander I. e GARG, Vijay K. Detecting Relational Global Predicates in Distributed Systems. *Proceedings of the Workshop on Parallel and Distributed Debugging,* páginas 21-31, ACM/ONR, 1993.
11. Michel et. al. Detecting Atomic Sequences of Predicates in Distributed Computations. *Proceedings of the Workshop on Parallel and Distributed Debugging,* páginas 36-42, ACM/ONR, 1993.
12. Sun Microsystems. *Java Platform Debugger Architecture.*
    `http://java.sun.com/j2se/1.4.2/docs/guide/jpda`.