Injeção de Falhas via Depuradores

Luis Claudio Rodrigues Gonçalves
lclaudio@conectiva.com.br
Conectiva S/A
R. Tocantins, 89
Curitiba – PR

Taisy Silva Weber
taisy@inf.ufrgs.br
PPGC – UFRGS
Av. Bento Goncalves, 9500
Porto Alegre – RS

Resumo

Esta trabalho apresenta a injeção de falhas via depuradores, uma abordagem para validação das técnicas de tolerância a falhas utilizadas em aplicações. São discutidos alguns dos prós e contras e limitações desta abordagem. Também apresenta comentários sobre uma ferramenta em desenvolvimento, chamada FIDe.

Palavras–chave: Injeção de Falhas, Depuradores, Ptrace, Chamadas de Sistema.

Abstract

This paper presents fault injection via debuggers, an approach to validate fault tolerance techniques used on applications. Some of the pros and cons along with the bounds of this approach are also discussed. Also presents comments about a tool, under development, called FIDe.

Keywords: Fault Injection, Debuggers, Ptrace, Syscalls.

1. Introdução

Em função da crescente utilização de sistemas computacionais em tarefas críticas, a dependabilidade destes sistemas torna–se mais importante a cada dia. Tarefas como gerência de equipamentos médicos em uma U.T.I., controle de tráfego aéreo, controle de temperatura em caldeiras e outras tantas podem colocar em risco o meio ambiente e, mesmo, vidas humanas se operando de forma incorreta.

Dada a natureza não determinística das falhas e a possibilidade de que elas aconteçam não somente no software, mas também no hardware, as técnicas da Engenharia de Software para a elaboração de bons programas não são suficientes. Neste momento entram em cena as técnicas de tolerância a falhas.

As técnicas de tolerância a falhas visam, em grande parte dos casos, não a eliminação da falha mas seu mascaramento. Em suma, busca–se que, mesmo na presença de falhas o sistema tenha um comportamento dentro de sua especificação.

A implementação de mecanismos de tolerância a falhas, porém, também está sujeita a falhas e a erros de projeto. O teste e validação destes mecanismos é dificultado pela já citada natureza não determinística das falhas. Uma determinada falha a ser tolerada pode nunca acontecer durante o período de testes e mesmo durante o período de utilização do sistema. Nas aplicações críticas, porém, a comprovação *in loco* do funcionamento ou não de um determinado mecanismo não é uma alternativa plausível.

Uma das formas de validar os mecanismos de tolerância a falhas utilizados é a Injeção de Falhas. A idéia básica é a de levar a aplicação em teste a perceber uma falha, dentro de seu conjunto de falhas a serem toleradas, e então disparar os mecanismos de tratamento relativos a esta. Se a falha for injetada em um nível de abstração inferior ao nível do sistema em teste, o sistema não consegue diferenciá–la de uma falha real. A injeção de falhas pode ser realizada de três formas distintas: via hardware, software ou simulação.

2. Injeção de Falhas

A injeção de falhas visa testar o comportamento do sistema sob falhas. Não substitui de forma

alguma as técnicas de Engenharia de Software, teste e especificação formal. Na verdade, a injeção de falhas só possui sentido quando a aplicação em questão se propõe a tolerar um modelo de falhas definido, isto é, a injeção de falhas é interessante em sistemas que apresentem características de tolerância a falhas.

O elemento injetor de falhas deve gerar a menor intrusão possível no sistema em teste. O distúrbio causado pelo injetor no comportamento do sistema deve resumir-se às falhas por ele injetadas. Porém, a carga do sistema, a temporização de eventos, o escalonamento dos processos no sistema e uma série de outras variáveis podem ser alteradas em função da presença do injetor. Isto traz à tona o princípio da intrusão de Heisenberg [PFI 98].

As técnicas de injeção de falhas, geralmente, são baseadas em sistemas de *hardware*, *software* ou de simulação. As técnicas de *hardware* injetam falhas físicas no sistema em teste. Métodos de simulação se utilizam de um modelo simulado do sistema alvo. A injeção de falhas via *software* permite emular falhas e erros de *hardware* [CAR 98].

A injeção de falhas por *hardware* utiliza equipamentos extras para introduzir falhas no sistema físico em estudo. Dependendo das falhas e de suas localizações, estes métodos podem ser: a) injeção de falhas por *hardware* com contato, caso em que se produz alterações de corrente e voltagem em função de contato físico direto do injetor com o sistema alvo e b) injeção de falhas por *hardware* sem contato, sendo as alterações geradas pela produção de algum fenômeno físico como interferência eletromagnética, vindo de uma fonte externa sem contato com o sistema.

A injeção de falhas por simulação toma por base um modelo de simulação do sistema em questão, onde são inseridas as falhas. Por ser uma simulação, pode-se controlar a temporização, o tipo de falhas e os componentes afetados no modelo com um maior ou menor grau de precisão, dependendo do nível de abstração do simulador. Convém ressaltar que a injeção de falhas e os resultados obtidos com este método são tão precisos e confiáveis quanto o modelo utilizado.

Técnicas de injeção de falhas por *software* alteram o estado do hardware e do *software* do sistema em teste utilizando ferramentas especiais de *software*, que levam o sistema a comportar-se como se uma falha real de hardware tivesse ocorrido. Não necessitam de *hardware* especial e podem ser utilizadas para atingir sistemas operacionais ou aplicações específicas.

2.1 Injeção de Falhas por Software

Se o alvo é uma aplicação, o injetor de falhas pode ser inserido em vários locais: na própria aplicação (necessita da disponibilidade do código fonte ou objeto), entre a aplicação e o sistema operacional (em bibliotecas, *wrappers* ou mesmo via depuradores) ou no próprio sistema operacional. Se o alvo é o sistema operacional, o injetor de falhas deve ser embutido no mesmo, visto que é bastante difícil colocar uma camada entre o sistema operacional e o *hardware* [PRA 96].

A injeção de falhas, se realizada em um nível de abstração inferior ao do sistema alvo, torna-se transparente e indistingüível de falhas reais. Um exemplo simples é a injeção de falhas no sistema operacional para o teste de uma determinada aplicação[LEI 98].

Assim como a instrumentação de um código para injetar falhas pode inserir erros não planejados no mesmo, a mesma instrumentação de código, enquanto alteração do código fonte ou objeto, pode alterar características do sistema como temporização, utilização de memória e outras, mascarando falhas. É importante, então, avaliar e minimizar a intrusão do injetor no sistema em estudo.

Outra forma de inserir falhas a nível de aplicação é a utilização de um injetor entre a aplicação e o sistema operacional. Geralmente estes métodos consistem em alteração de bibliotecas utilizadas pelo sistema, *wrappers* ou *stubs* que filtram, traduzem ou redirecionam requisições feitas ao sistema pela aplicação. Pode-se ainda utilizar depuradores de código (*debuggers*) ou mesmo técnicas de depuração para executar o sistema passo a passo, até um ponto de parada (*breakpoint*) ou até uma chamada do sistema. Estas técnicas de depuração beneficiam-se de características dos processadores dos nodos, de características dos sistemas operacionais em uso ou de ambos [ROS 96].

A utilização de técnicas de depuração para a injeção de falhas é particularmente interessante por sua grande flexibilidade e uma certa portabilidade, se baseada em sistemas POSIX. Pode–se, por exemplo, interceptar o retorno de chamadas do sistema e alterar o valor de registradores, simulando–se falhas arbitrárias ou alterar um *buffer* de dados retornado por uma chamada de sistema, simulando–se uma falha de memória ou de acesso a disco. Em sistemas de tempo real, porém, a intrusão gerada pelo depurador ainda é um problema em estudo.

A inserção de falhas no sistema operacional, seja no seu núcleo ou em um de seus módulos auxiliares, possui como principal agravante o fato de ser intimamente ligada a características do mesmo, dificultando a portabilidade. Em contrapartida, as falhas injetadas desta forma chegam à aplicação indistingüíveis de falhas reais, dada a diferença de nível das camadas de abstração onde a falha é injetada e onde a falha é colhida. A localização do injetor no sistema operacional também é intimamente ligada ao tipo de sistema, sua estrutura (monolítico ou *microkernel*), a disponibilidade ou não de seu código fonte e ao tipo de falha a ser injetada.

Dependendo da construção do injetor e do tipo de falhas a ser injetada, o mesmo pode atingir apenas a aplicação em teste ou o sistema como um todo, inclusive alterando o comportamento do próprio sistema operacional. Uma ferramenta para simulação de falhas de disco, por exemplo, deve ser bem projetada caso se deseje armazenar *logs* de monitoração no próprio disco do nodo em teste.

Em todos estes casos é de grande valia que se possa especificar detalhadamente que falhas injetar e quando injetá–las, para que se possa levar protocolos e aplicações a estados difíceis de se atingir.

2.2 Injeção de Falhas via Depuradores

A injeção de falhas utilizando–se depuradores de código, ou técnicas de depuração, constitui uma poderosa ferramenta para experimentação e análise de dependabilidade. Beneficia–se, principalmente, pela ausência da necessidade de código fonte do sistema em teste e pelas facilidades advindas do controle sobre a aplicação em si. Execução passo a passo, alteração de conteúdos de memória e alteração do valor de registradores são exemplos das facilidades mencionadas acima.

Os depuradores podem basear–se em características do processador, características do sistema operacional ou ambos.

A utilização de características de um determinado processador torna as técnicas de depuração intrinsecamente ligadas ao mesmo e, portanto restritas a este. No caso de sistemas operacionais onde o acesso ao *hardware* é realizado via chamadas de sistema, pode existir a necessidade de alterações no código fonte do sistema operacional para que se possa obter acesso a recursos extras de um determinado processador.

Os sistemas unix, os unices, em geral oferecem uma chamada de sistema para a depuração de programas, chamada *ptrace*. Depuradores como o *gdb* e ferramentas como *strace*, ambos da GNU, são baseados nesta chamada [LKI 98].

A chamada *ptrace* possibilita execução do programa passo a passo, até uma chamada de sistema ou até um ponto de parada, leitura ou escrita de valores nas áreas de código, dados, estruturas do sistema operacional relativas ao processo ou na pilha [TLK 98]. Toda esta funcionalidade pode ser utilizada para a criação de ferramentas para injeção de falhas.

3. O Injetor de Falhas

Um injetor de falhas baseado em técnicas de depuração encontra–se em desenvolvimento, como parte integrante da dissertação de mestrado do autor. Estudos preliminares, realizados em ambiente Linux, revelaram ser absolutamente factível e relativamente simples a criação de tal ferramenta. Esta ferramenta visa, inicialmente, instrumentalizar pesquisas sobre *recovery* em bancos de dados realizadas pelo grupo de tolerância a falhas da UFRGS.

A ferramenta FIDe (Fault Injection via Debugging) trabalhará inicialmente com um conjunto de falhas definido pelos pesquisadores envolvidos no projeto de *recovery*. Em função

da definição de um conjunto de falhas, a forma de injetar cada falha em especial será estudada. Algumas destas falhas podem ser injetadas como alterações nos registradores de entrada de uma determinada chamada de sistema, alterações nos dados resultantes, alterações nos *flags* de indicação do resultado ou mesmo a não realização da chamada em si. Outra possibilidade seria a alteração dos parâmetros de entrada da chamada passados via pilha [PHR 96].

Foi escolhida a execução de processos até uma chamada de sistema em função de dois motivos básicos: 1) praticamente todo acesso a dispositivos e todas as operações básicas do sistema são realizadas via chamadas de sistemas e 2) neste caso não se faz necessário nenhum estudo do código objeto da aplicação para a escolha local de inserção dos pontos de parada, diminuindo tanto a dificuldade da inserção de falhas quanto o perigo da inserção de pontos de parada em locais inacessíveis ou da alteração inadvertida de trechos de código.

Como exemplo de operações que utilizam chamadas de sistema podemos citar leitura de arquivo, escrita em vídeo, duplicação de processo (*fork*), término de programa, dentre outras [SYS 93].

O FIDe deve apresentar uma interface para a configuração do cenário de falhas a ser gerado, dentro do conjunto especificado. Temporização da injeção de uma determinada falha, inserção de uma falha na *n–ésima* ocorrência de uma determinada chamada de sistema ou a cada *n* ocorrências da mesma são algumas das características previstas para o FIDe. A idéia de uma ferramenta para gerar conjunto de falhas também existe. Desta forma, a geração de cenários de falhas, reais ou mesmo caóticos, torna–se possivel. Tanto melhor, torna–se passível de repetição [HSU 97].

Um detalhe interessante é o fato de que esta abordagem é interessante para a injeção de falhas em aplicações mas inapropriada para injeção de falhas no núcleo de um sistema operacional, por exemplo. Apesar disto, em função das características citadas anteriormente, esta ferramenta não é interessante apenas para a injeção de falhas mas também para a criação de cenários de testes para aplicações comuns. A principal vantagem do FIDe em relação aos depuradores convencionais é a possibilidade de geração e reutilização dos cenários de falhas, podendo–se repetir de forma automática um dado experimento para várias aplicações diferentes.

4. Bibliografia

[CAR 98] CARREIRA, João; MADEIRA, Henrique; SILVA, João G. **Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers**. Transactions on Software Egineering, vol 24, no. 2, Fevereiro, 1998.

[HSU 97] HSUEH, Mei–Chen; TSAI, Timothy K.; IYER, Ravishankar K. **Fault–Injection Techniques and Tools**. Computer, vol. 30, no. 4, Abril, 1997.

[LEI 98] LEITE, Fábio O. **Injeção de Falhas de Comunicação no Sistema Operacional Linux**. Trabalho Individual, CPGCC/UFRGS, 1998.

[LKI 98] BECK, M.; BÖHME, H.; DZIADZKA, M.; KUNITZ, U.; MAGNUS, R.; VERWORNER, D.. **Linux Kernel Internals**. Second Edition. Addison–Wesley. England, 1998.

[PRA 96] PRADHAN, Dhiraj. **Fault–Tolerant Computer System Design**. Prentice–Hall, 1996.

[PFI 98] PFISTER, Gregory F.. **In Search of Clusters**. Prentice–Hall, 1998.

[PRH 96] Route9. **Smashing the Stack for Fun and Profit**. Phrack 46, arquivo 14. Ezine disponível via WWW em http://www.phrack.org. Agosto, 1996

[ROS 96] ROSENBERG, Jonatham B.. **How Debuggers Work: Algorithms, Data Structures, and Architecture**. John Wiley & Sons. New York, 1996.

[SYS 93] LANKESTER, Branko. **SYSCALL_ENT.C**. .Versão 1.3. Arquivo integrante do código fonte da ferramenta Strace, disponível via WWW em http://www.leidenuniv.nl/~wichert/strace. Agosto, 1996

[TLK 98] CARD, Rémy; DUMAS, Éric; MÉVEL, Franck. **The Linux Kernel Book**. John Wiley & Sons. England, 1998.