Short Communication

Como depurar código fonte DLL feita em linguagem diferente do software executável principal

How to debug DLL source code made in language different from the main executable software

Gustavo Marino Botta[(1)]; Luiz Fernando Resende dos Santos Anjo [(2)].

[(1)]Estudante; Universidade Federal do Triângulo Mineiro; Uberaba, MG; gustavomarino@iftm.edu.br;
[(2)]Professor; Universidade Federal do Triângulo Mineiro; Uberaba, MG; luizfernando@civil.uftm.edu.br;

RESUMO: Parte do software EPANET é desenvolvida em linguagem Delphi (ambiente gráfico) e parte em linguagem C (biblioteca DLL de cálculos hidráulicos). Em ambas as partes, é necessário fazer a depuração do código fonte para verificação de erros de programação. A DLL não é depurada no ambiente Delphi, pois o código já está compilado de forma externa e em outra linguagem. A solução encontrada foi depurar a DLL na IDE DEV-C++ utilizando o executável principal como software chamador.

Termos de indexação:Biblioteca de Ligação Dinâmica, Depuração, Linguagens de Programação.

ABSTRACT*: Part of EPANET software is developed in Delphi language (graphical environment) and partly in C language library (DLL hydraulic calculations). In both parts, it is necessary to debug the source code to check for programming errors. The DLL is not debugged in the Delphi environment, since the code is already compiled externally and in another language. The solution was to debug the DLL in the IDE DEV-C++ using the main executable software as caller.*

Index terms*: Debugging, Dynamic Link Library, Programming Languages.*

INTRODUÇÃO

O autor pretende implantar outro método de cálculo no software EPANET. O EPANET é um software que disponibiliza seu código fonte e realiza cálculos hidráulicos. Neste software, parte dele é desenvolvido em linguagem Delphi e parte em linguagem C.

Na integração destas duas linguagens é necessário passar pelo processo de depuração em ambas as linguagens para que seja possível testar o código por etapas, a fim de auxiliar na detecção de erros de programação.

Como objetivo, pretende descrever como depurar o código fonte DLL feito em linguagem C diferente do software executável principal, feito em linguagem Delphi.

REVISÃO TEÓRICA

O EPANET é um software que simula sistemas de tubulação de distribuição de água e é de domínio público,podendo ser copiado e distribuído livremente. Sendo um programa desenvolvido para Windows, o EPANET realiza simulação ao longo do período de movimento da água dentro de redes de tubulação pressurizada. (EPANET, 2013). Conforme Rossman (2000), o software EPANET utiliza como método de cálculo o método gradiente.

O projeto de pesquisa do autor deste artigo propõe a implantação de outro método de cálculo no software EPANET, denominado como Modelo Dinâmico Inercial Rígido (MDIR) que foi elaborado por Anjo (2008) na sua Tese de Doutorado. Este modelo considera os efeitos da inércia no escoamento da água, e despreza os efeitos da elasticidade e da compressibilidade que possam ocorrer.

Ao examinar o código fonte do EPANET foi verificado que o mesmo possui um ambiente gráfico desenvolvido em linguagem Delphi, onde é possível montar a rede de tubulação que é composta de tubos, nós (junções), bombas, válvulas e tanques de armazenamento ou reservatórios. Conforme Leão (2003), o Delphi é uma ferramenta para desenvolvimento de aplicações com interface gráfica e baseada na linguagem *Object Pascal*.

Na análise do código fonte também foi possível verificar que o EPANET utiliza uma biblioteca externa no formato de vinculação dinâmica (*Dynamic Link Libraries* – DLL) e que os cálculos hidráulicos e de qualidade da água são todos feitos nesta biblioteca, cujo nome é epanet2.dll e foi escrita na linguagem C.

Segundo a Microsoft Developer Network (2014a), existem duas formas de vincular bibliotecas: a vinculação estática e a vinculação dinâmica.

Na vinculação estática, as bibliotecas que serão utilizadas no programa são compiladas e anexadas ao código executável deixando o mesmo com tamanho maior e não permitindo que outros programas acessem esta biblioteca que foi embutida no executável.

Na vinculação dinâmica a biblioteca fica armazenada externamente ao arquivo executável principal como, por exemplo, em um arquivo com

extensão DLL. No módulo executável é necessário somente ter os dados que consigam localizar esta função externa, não sendo necessário que a mesma esteja embutida no código executável. Esta localização será feita somente no tempo de execução do programa e, caso não encontre o arquivo referenciado, é neste momento (*run-time*) que será acusado o erro.

A vinculação dinâmica fornece uma maneira para um aplicativo chamar uma função que não é parte do seu código executável. O código executável para a função está localizado em uma DLL, que contém uma ou mais funções que são compiladas, vinculadas e armazenadas separadamente dos processos que irão usá-las.

O uso da vinculação dinâmica, em vez da estática, oferece várias vantagens: economia de memória, economia de espaço em disco e atualização facilitada.

Quando se está desenvolvendo um algoritmo em um software é importante ter como testá-lo passo a passo para verificar se não houve erro. Este processo é chamado de *debugging* (depuração). Segundo Warner & Goldsman (1996), existem duas categorias de *bugs*: erros de gramáticas e erros de lógica.

Os erros de gramáticas são os erros de ponto-e-vírgulas faltando, comandos digitados de forma errada e outros erros de sintaxe. Estes erros são normalmente detectados no momento da compilação, pois o compilador da linguagem faz a verificação desteserros de sintaxe.

O segundo tipo de erros, que são os erros de lógica de programação, não são detectados pelo compilador. O programa executará normalmente, porém não dará o retorno desejado, como, por exemplo, o resultado de uma fórmula complexa. Neste caso, o erro pode se encontrar em um cálculo equivocado no momento da programação e o valor que foi designado para uma variável ficar errada por este motivo. Isto caracteriza erro humano, mas a linguagem de programação pode auxiliar na localização destes tipos de erros usando a técnica de depuração. “Uma grande parte do tempo gasto em programas de computador é gasto em depuração.” (WARNER & GOLDSMAN, 1996).

PARTE EXPERIMENTAL

O código fonte do EPANET foi obtido no site da EPA (2013), na seção downloads. O item EPANET 2 source code files contém o código fonte em delphi e seus respectivos componentes adicionais, possui também o código fonte em C da DLL: epanet2.dll.

Inicialmente, não se havia noção que os cálculos hidráulicos estavam todos estruturados na DLL, então a primeira linha de pesquisa foi analisar o código fonte em Delphi. Foram instalados os componentes adicionais, que são usados no projeto Epanet2w.dpr e feita a compilação do projeto que ocorreu com sucesso.

Ao fazer a depuração do código fonte Delphi para ir conhecendo as etapas dos cálculos hidráulicos, percebeu-se que o mesmo referenciava uma biblioteca externa no formado DLL.

Como a DLL já estava compilada, a mesma não interferiu em nada na compilação do projeto em Delphi. O erro só ocorreria em tempo de execução se a biblioteca não fosse encontrada.

Neste ponto surgiu um novo problema: a necessidade de compilar o código fonte da DLL, pois como os cálculos estão todos nesta biblioteca dinâmica, será nela que a implantação do MDIR terá que ser feita.

Esta biblioteca cujo nome é epanet2.dll foi escrita na linguagem C. No site da Epanet.de Hydraulic Network Analysis (2014) foi possível baixar o projeto para construir a DLL do Epanet na ferramenta de desenvolvimento DEV-C++, pois o código fonte foi disponibilizado, mas a junção dos mesmos em DLL estava inicialmente documentada somente para outros compiladores que não eram de uso e conhecimento do autor deste artigo.

Com isto, a DLL conseguia ser compilada e gerada o arquivo externo: epanet2.dll. O executável Delphi referente ao modo gráfico também compilava e executava normalmente. Neste ponto tinha-se a certeza que seria possível adaptar o código fonte, uma vez que o executável e a biblioteca DLL foram compilados corretamente. No entanto, para uma melhor análise das alterações efetuadas no código fonte, era necessário depurar os dois códigos (Delphi e DLL).

A depuração do Delphi ocorreu com sucesso permitindo passar etapa por etapa no código fonte e analisar cada linha de código. O mesmo não aconteceu com a DLL,pelo fato da mesma já estar compilada, o Delphi não adentrava ao código fonte em C, ele somente recebia o resultado sem passar pela depuração linha a linha.

No item abaixo será descrito como foi conseguido a depuração da DLL.

RESULTADOS E DISCUSSÃO

Conforme instruções da Microsoft Developer Network (2014b), a depuração da DLL pode ser obtida iniciando-se a depuração do projeto que cria o executável que chama a DLL ou o projeto que cria a própria DLL.

Seguindo esta linha de orientação, no primeiro item foi conseguida somente a depuração do código fonte do executável em Delphi, mas ao passar pela DLL a depuração não adentrava ao código em C da DLL. O segundo item não foi realizado, pois o código em C não roda no Ambiente Integrado de Desenvolvimento (IDE - *Integrated Development Environment*) do Delphi.

Segundo Leão (2003), “Você não pode, no entanto, executar uma DLL a partir do ambiente de desenvolvimento do Delphi, selecionando o item Run do menu Run (embora esse item esteja habilitado), pois uma DLL não é uma aplicação (a menos que você defina um aplicativo como Host, a ser especificado na caixa de diálogo Run Parameters, exibida quando se seleciona o item Parameters do menu Run).”

Utilizando esta linha de raciocínio de Leão, foi feito um experimento abrindo o código fonte da DLL, em linguagem C, na IDE DEV-C++ e tentado localizar um executável chamador. Foi obtido sucesso, onde o executável principal rodava e quando passava pelas rotinas da DLL a mesma era depurada.

Para conseguir este resultado foi utilizado a seguinte configuração no DEV-C++: no menu *Debug*, no item Parâmetros, no campo aplicação local foi especificado o localização do arquivo executável principal, veja **figura 1**. Desta forma a aplicação principal foi executada sem depuração e quando chamava a DLL, a mesma era depurada através da utilização do comando menu *Debug*, item *Debug* ou a tecla de atalho F8.

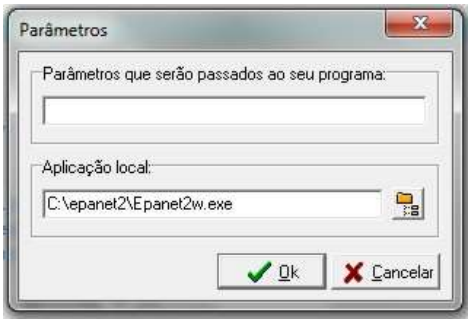

Figura 1. Tela parâmetros do debug.
Fonte: do autor (2014).

CONCLUSÕES

Quando se utiliza código fonte do executável principal em uma linguagem diferente do código fonte de uma biblioteca externa, é possível fazer a depuração dos códigos bastando se atentar à IDE que consegue ler cada um dos códigos.

No caso específico deste artigo, o ambiente gráfico do EPANET, feito em *Object Pascal,* será depurado usando a IDE do Delphi. O código da DLL em linguagem C será executado normalmente pela chamada do aplicativo principal, mas sem passar pelo depurador.

E para depurar a DLL (linguagem C) utiliza-se a IDE DEV-C++, informando qual é o aplicativo principal que faz a chamada à DLL. Neste caso, a aplicação principal será executada sem depuração e a DLL será depurada.

REFERÊNCIAS

ANJO, L. F. R. S. **Modelo Hidráulico para Transitórios Lentos em Conduto Forçado**. 2008. 154 p. Tese Doutorado, Universidade Estadual de Campinas, Campinas.

EPA - United States Environmental Protection Agency. **EPANET - Software That Models the Hydraulic and Water Quality Behavior of Water Distribution Piping Systems**, 2013. Disponível em: < http://www.epa.gov/nrmrl/wswrd/dw/epanet.html >. Acesso em: 03 nov. 2013.

EPANET. de Hydraulic Network Analysis. **EPANET and C - Build the EPANET toolkit DLL**. Disponível em: < http://epanet.de/developer/index.html >. Acesso em: 20 fev. 2014.

LEÃO, M. **Borland Delphi 7 Curso Completo**. Rio de Janeiro: Axcel Books, 2003.

MICROSOFT Developer Netword. **DLLs**. Disponível em : < http://msdn.microsoft.com/library/1ez7dh12(v=vs.90).aspx >. Acesso em: 08 mar. 2014a.

_____. **How to: Debug Native DLLs**. Disponível em : < http://msdn.microsoft.com/library/c91k1xcf.aspx >. Acesso em: 08 mar. 2014b.

ROSSMAN, L. A. **Epanet 2 Users Manual**. Disponível em: <http://nepis.epa.gov/Adobe/PDF/P1007WWU.pdf>. Acesso em: 20 fev. 2014.

WARNER, S. L.; GOLDSMAN, P. **Delphi 2 em exemplos**. São Paulo: Makron Books, 1996.