Linear: Linearizador de Estruturas Complexas

Rogério Drummond *Carlos Hoyos*

Relatório Técnico IC 97-05

Maio de 1997

Linear

Linearizador de Estruturas Complexas [1]

Rogério Drummond
rog@ahand.unicamp.br

Carlos Hoyos
hoyos@ahand.unicamp.br

Laboratório A-HAND
Instituto de Computação
Unicamp
Rua Roxo Moreira, 1076
13083–591 Campinas, SP

Maio de 1996

RESUMO

Apresentamos um linearizador de estruturas complexas. Ele usa uma estratégia diferente da utilizada pelo XDR da Sun Microsystem e seu desempenho chega a ser cinco vezes mais rápido. Em adição, suporta estruturas circulares e referências a partes de estruturas.

Mesmo quando o Linear é usado com o suporte a circularidade ligado, ele é, em geral, mais rápido que o XDR para a mesma estrutura sem circularidade.

ABSTRACT

Presents a marshalling system called *Linear*. It is based on a different approach than the Sun Mycrosystem's XDR, and can be five times as fast as XDR. In addition, the Linear system supports circularity of refferences which is not supported by XDR as well as refferences to parts of a bigger structure.

Most often, the Linear system is faster than XDR even when circularity is being verified.

INTRODUÇÃO

Aplicações que utilizam estruturas de dados complexas que devem ser armazenadas em disco para uso posterior são bastante comuns. Todo editor se enquadra nesta categoria, seja ele um editor de textos, de gráficos, de diagramas ou um CAD.

Os elementos de uma estrutura complexa, como uma árvore ou lista, são alocados individualmente conforme são necessários. Como resultado, a estrutura fica espalhada na mémória do programa. Para ser armazenada em disco, ela deve ser *linearizada*. Nesse processo, os apontadores que contêm endereços de memória passam a ter deslocamentos (do inglês *offsets*) relativos ao começo

[1] Este trabalho é parcialmente financiado pelo PROTEM/CNPq, processo número 680089/94–2 e 380371/95-2 e pela FAPESP, processo número 94/5179-6.

do arquivo. O processo inverso de *deslinearização* corresponde a colocar os dados linearizados numa estrutura em memória equivalente à original.

Esse mesmo problema também ocorre quando dois processos distribuídos necessitam trocar estruturas de dados complexas. Um cliente pode passar como parâmetro uma árvore e receber uma lista como valor de relatório. Essas estruturas têm que ser linearizadas antes de serem enviadas pela rede.

O programador pode construir para cada estrutura funções que a linearizam/deslinearizam. Isso no entanto é tedioso e a cada alteração dos tipos envolvidos na estrutura as funções devem ser reescritas. Obviamente, o ideal é uma ferramenta que a partir das definições dos tipos gere automaticamente essas funções. Isso foi o que a Sun Microsystem desenvolveu a mais de 8 anos atrás com o sistema XDR[Sun88].

SOLUÇÕES CONHECIDAS PARA LINEARIZAÇÃO/DESLINEARIZAÇÃO

Muitos linearizadores existem atualmente e em geral seguem o padrão XDR da Sun. Este estabelece uma linguagem universal de definição de tipos e uma representação binária desses tipos independente de linguagem, sistema operacional e arquitetura de *hardware*. A linguagem de definição de tipos do XDR abrange praticamente o universo de tipos da linguagem C.

O problema de linearização de estruturas com suporte a múltiplas plataformas e independência de linguagens já foi objeto de um longo estudo pelo grupo de Carnegie Mellon University associado ao Projeto Gandalf[HeN86, Lom87]. O DCE[XXX] segue a linha do XDR e fornece facilidades de suporte ao programador que simplifica o seu uso. O CORBA[XXX], por outro lado estendeu a linguagem de tipos de forma a melhor acomodar linguagens como C++. A IDL do CORBA, no entanto é só uma proposta normativa. As suas implementações, em geral também seguem os princípios do XDR.

Quando o XDR lineariza uma estrutura, ele a transforma num padrão de representação (binária) universal que pode ser facilmente traduzido para qualquer plataforma de hardware e sistema operacional. Para que isso seja possível, os dados tem que satisfazer o "mínimo denominador comum" entre todas as plataformas suportadas. Alguns processadores, RISCs em particular, têm fortes restrições de alinhamento, exigindo por exemplo que todo número inteiro comece um endereço múltiplo de 4. A estratégia do XDR é traduzir (linearizar) os dados de uma plataforma específica para a representação universal. O processo inverso (deslinearização) não dependente da plataforma origem, somente da plataforma destino.

O XDR utiliza uma função de linearização e outra de deslinearização para cada tipo de dado utilizado. Para tipos pré-definidos pelo padrão XDR, essas funções são pré-definidas. Para novos tipos, tais como estruturas, uniões e enumerados, novas funções devem ser definidas pelo usuário, manualmente ou com o aplicativo *rpcgen*. Ele lê um arquivo (com extensão ".x") contendo as definições dos tipos que serão linearizados e gera um arquivo com as definições dos tipos na linguagem de programação (C ou C++) para ser incluído nos arquivos-fonte que utilizam esses tipos e um arquivo com linearizadores e deslinearizadores específicos. Esse arquivo deve ser compilado e ligado ao código objeto final.

Para cada tipo e para cada plataforma envolvida, XDR define funções específicas. O processo de linearização/deslinearização envolve a chamada de uma função para cada tipo ou sub-tipo envolvido. Um programa só pode receber ou enviar estruturas para as quais ele conhece o código dos linearizadores de tipos necessários.

Esse sistema de linearização entretanto apresenta alguns aspectos restritivos:

- a representação universal aumenta consideravelmente o tamanho dos dados;

- é necessária uma função para linearizar e outra para deslinearizar cada tipo de dado;
- não é possível linearizar estruturas de dados que contenham circularidades;
- não é possível linearizar estruturas de dados que contenham referências a estruturas parciais.

ESTRATÉGIA DO SISTEMA LINEAR

Diferente do XDR, o *Linear* usa funções linearizadora e deslinearizadora universais. As informações sobre os tipos são armazenadas numa tabela e não em código executável. Como no RPCGen, o usuário especifica um arquivo ".x", descrevendo os tipos envolvidos expressos na linguagem de tipos do XDR. O programa "linprep" analisa o arquivo ".x" e gera a tabela de tipos. As funções de linearização e deslinearização (*linear* e *unlinear*) percorrem a tabela em paralelo com a navegação da estrutura (em memória ou linearizada). Essa estratégia se mostrou mais rápida que o código gerado pelo XDR nos testes realizados. Embora as descrições de tipos sejam interpretadas durante a execução, as funções do Linear não são geradas mecanicamente e podem ser otimizadas.

O Linear introduz o conceito de tipo *compacto.* Todo tipo básico (char, int, float, enum e suas variações short, long, unsigned, ...) é compacto. Todo vetor ou estrutura composta de campos compactos também é um compacto. Membros compactos em seqüência numa estrutura são considerados como um elemento compacto. Em resumo, um compacto é uma seqüência contígua de bytes que não contém apontadores.

O conceito de "compacto" não precisa ser conhecido do usuário do Linear, sendo usado apenas para otimização. Cada compacto pode ser copiado como um bloco de memória ignorando o seu conteúdo. O programa "linprep" quando analisa as definições de tipos no arquivo ".x" determina os tipos "compacto" dentro destas estruturas e na medida do possível, descreve os tipos em termos de sub-tipos compactos. A descrição detalhada dos componentes de um compacto pode ser incluída na tabela para deslinearizações que necessitem dessas informações.

O maior problema com o XDR é não aceitar estruturas auto referenciáveis. O Linear suporta tais estruturas com desempenho aceitável. Dependendo da complexidade da estrutura o Linear é mais rápido do que o XDR na mesma estrutura sem circularidade.

ANÁLISE DE DESEMPENHO

Mostramos nesta seção comparações de desempenho entre o XDR e o Linear. A estrutura básica a ser linearizada é uma árvore com três variações.

Em cada caso geramos árvores uniformes de altura de 1 a 15. O tempo de linearização e deslinearização foi verificado numa SPARC Classic onde medimos o *user time* e o *system time*. São traçadas 4 curvas, correspondendo aos tempos de execução do Linear para linearizar e deslinearizar a árvore com e sem circularidade. Os gráficos de tempo de execução mostram os tempos como porcentagem do tempo gasto pelo XDR para amesma operação.

O Linear pressupõe que as estruturas não são circulares a menos que explicitamente parametrizado neste sentido. Neste caso, toda referência é potencialmente à uma sub-estrutura já linearizada e deve ser verificada. Como o objetivo dos testes é comparar os tempos com XDR, formamos para o Linear as mesmas estruturas (árvores) sem referências circulares reais. No entanto, ele foi parametrizado como se a estrutura fosse circular. A diferença dos tempos entre o Linear com e sem circularidade mostra o tempo extra causado pelo tratamento da circularidade.

A primeira estrutura (no1) é um nó de árvore binária contendo um inteiro como membro extra. A segunda (no2) adiciona um vetor de 10 caracteres à primeira (no1). A terceira adiciona uma sub-estrutura com vários campos.

Para árvores grandes, o desempenho do Linear nessas três estruturas é respectivamente cerca de 50%, 35% e 20% do tempo do linearizador XDR. As medidas foram tomadas em função do *timer* do Solaris e os tempos de CPU em modo usuário que mede somente o tempo de execução.

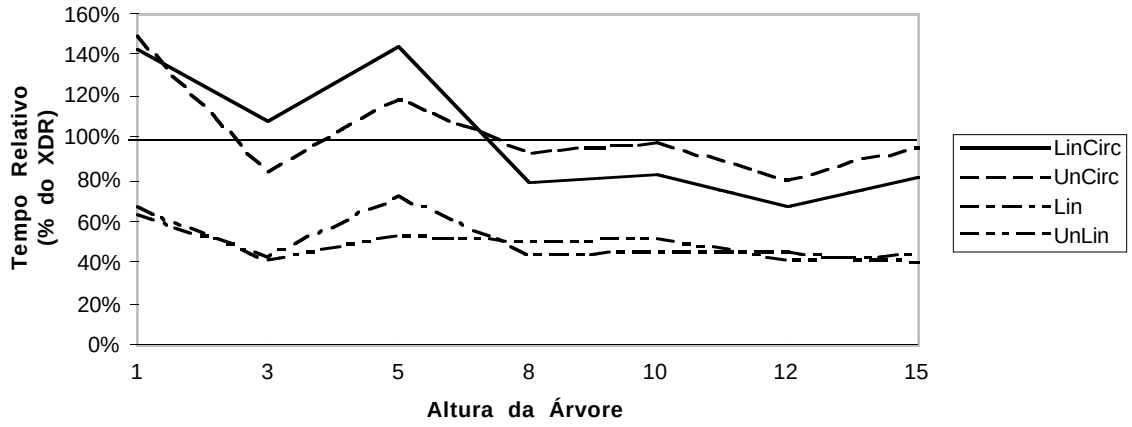
O tamanho das árvores em memória é descrito na tabela que se segue.

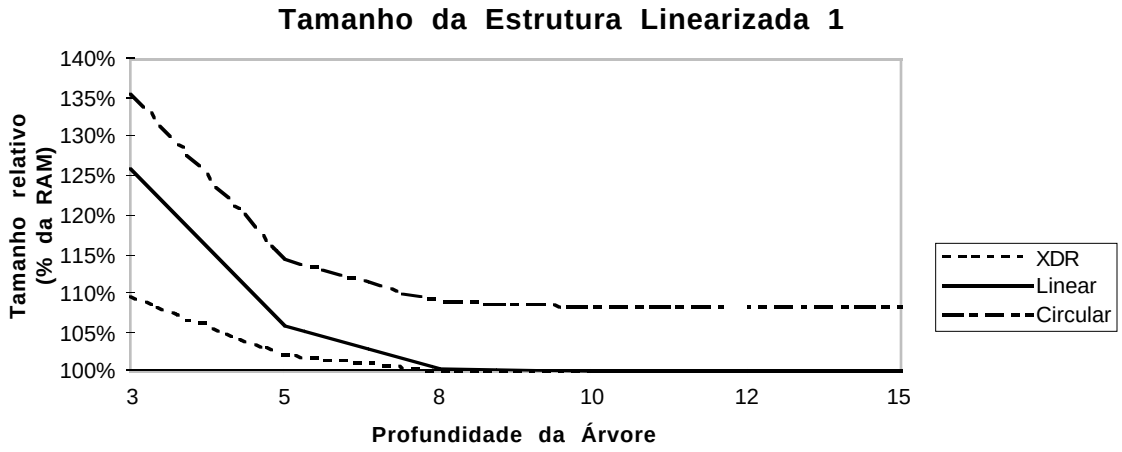
Tamanho dos Dados em Memória

Estrutura Básica	Altura da Árvore						
	1	3	5	8	10	12	15
no1	12	84	372	3.060	12.276	49.140	393.204
no2	24	168	744	6.120	24.552	98.280	786.408
no3	52	364	1.612	13.260	53.196	212.940	1.703.884

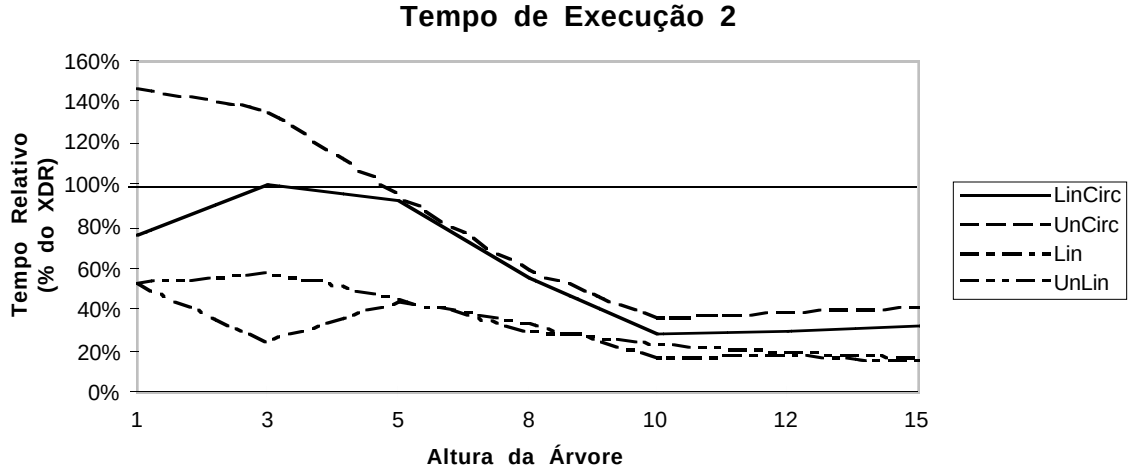
```
typedef struct BinTreeNode {
       BinTreeNode   *left;
       BinTreeNode   *right;
       int           x;
} no1;
```

Tempo de Execução 1

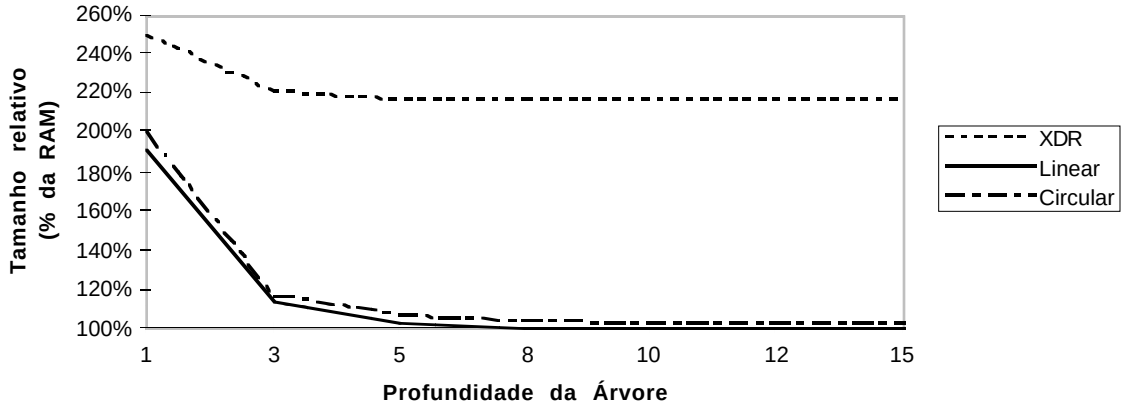





```
typedef struct BinTreeNode {
       BinTreeNode   *left;
       BinTreeNode   *right;
       int           x;
       char          name[10];
} no2;
```



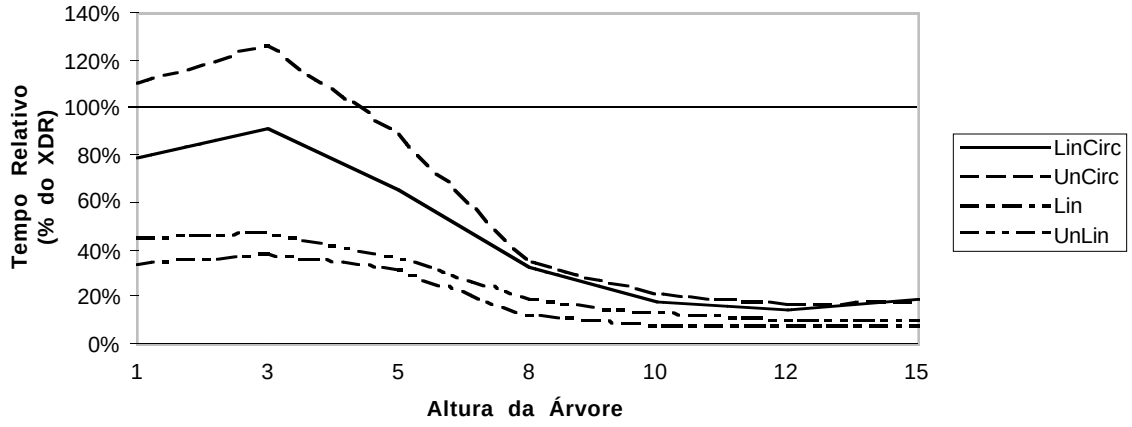
Tamanho da Estrutura Linearizada 2

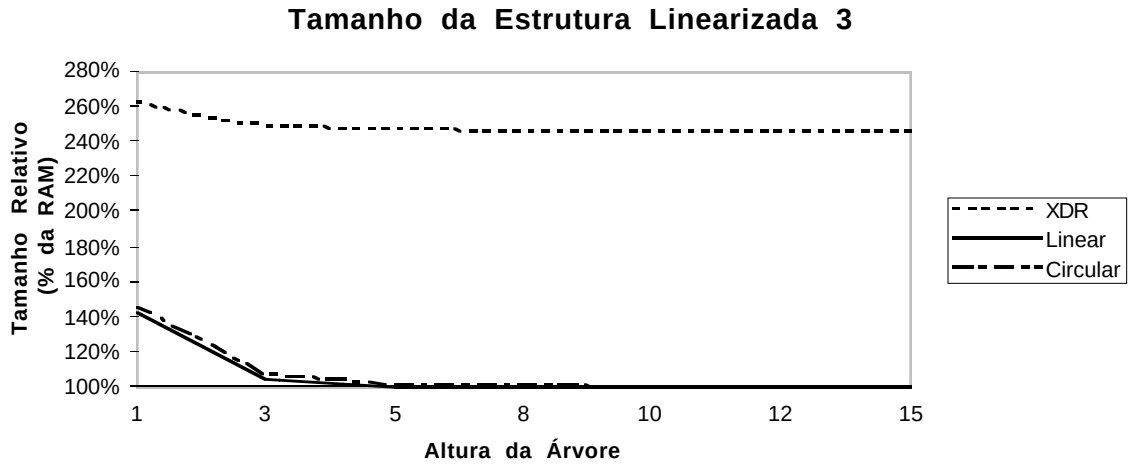



```
typedef struct {
       int    a;
       char   b;
       char   c[15];
       int    d;
       long   e;
} InternalData;

typedef struct BinTreeNode {
       BinTreeNode   *left;
       BinTreeNode   *right;
       int           x;
       char          name[10];
       InternalData  id;
} no3;
```

Tempo de Execução 3





SISTEMA LINEAR – ESTRUTURA INTERNA

Estruturas Auto-Referenciáveis: circularidades e referências a estruturas parciais

Além do alto desempenho para estruturas simples, o sistema Linear trata também eficientemente de problemas de *circularidades* e referências para *estruturas parciais.*

Circularidades acontecem quando elementos de uma estrutura de dados complexa referenciam partes da mesma estrutura já referenciados anteriormente, como em listas duplamente ligadas, listas circulares, árvores costuradas e grafos de uma forma geral. O modelo de linearização XDR entra em laço infinito quando tenta processar tais estruturas. Linear mantém informações necessárias durante a fase de linearização e deslinearização de forma que quaisquer níveis de circularidade são resolvidos.

Já *referências a estruturas parciais* acontecem quando há referências a partes de uma estrutura, não apenas ao seu início. O modelo XDR não suporta tais referências, tratando cada referência ao mesmo bloco como referências a blocos distintos; a linearização trata cada referência como a um elemento distinto, criando uma estrutura muito diferente da original. Linear resolve esse problema verificando se a referência acontece para dentro de uma região de memória já referenciada para um bloco de memória que contém uma região anteriormente referenciada.

Linearização e Deslinearização: *linear* e *unlinear*

Antes de analisar como a resolução de auto-referências é feita pelo Linear, serão apresentadas brevemente os processos de linearização e deslinearização.

Linearização (*linear*):

Uma estrutura complexa pode ser vista como um grafo. Cada nó desse grafo pode conter dados de tipo compacto e dados de tipo não compacto. Tipos compactos são elementos internos ao nó e que podem ser tratados na corrente invocação da função *linear*. Contudo, tipos não compactos são referências para outros nós. A idéia é percorrer o grafo a partir de um nó raiz e tratar todos os seus "filhos" recursivamente. O percurso é feito em profundidade e *in-order*, i.e., tipos são tratados à medida em que vão surgindo, sendo que os não-compactos implicam em recursão.

O processo de linearização começa a partir do nó raiz e recursivamente processa toda a estrutura. Para cada nó processado, a linearização concatena uma identificação do tipo da estrutura linearizada e também a própria estrutura linearizada ao resultado parcial dos nós já processados.

Tipos compactos são simplesmente concatenados ao resultado parcial com sua identificação. A sua definição de tipo no arquivo-tabela especifica seu tamanho. Tipos não compactos provocam a concatenação dos resultados das chamadas recursivas à função *linear*.

Deslinearização (*unlinear*):

O processo de deslinearização trata recursivamente cada nó a partir do nó raiz. Como esse nó tem seu tipo especificado (um parâmetro da função *unlinear* especifica o tipo do nó raiz), é possível, com o auxílio do arquivo tabela, determinar quais são seus elementos compactos e quais são seus nós "filhos", ou seja, referências para dados não compactos. Assim acontece da mesma forma para qualquer nó, pois o arquivo com a informação linearizada contém a indicação de qual é o tipo da próxima estrutura.

Cada nó da estrutura a recuperar pode ou não ser compacto. Em caso afirmativo, basta alocar o espaço correspondente e copiar a informação do arquivo com a informação linearizada. Em caso de um não-compacto, há recursão.

Como no caso da linearização, deve-se contornar as circularidades e auto-referências. Contudo, o arquivo linearizado e a tabela de definições são suficientes para resolver o problema.

Resolução de Auto-Referências

O problema de auto-referências no processo de linearização consiste em sua detecção e no armazenamento de informações para sua correta deslinearização. Os casos de auto-referências podem ser descritos em três situações:

a) a estrutura sendo linearizada contém referência a outra estrutura já linearizada;
b) a estrutura sendo linearizada contém referência a uma parte de uma estrutura já linearizada;
c) a estrutura sendo linearizada contém referência a uma estrutura não totalmente linearizada.

Cada situação acima requer um tratamento especial.

Para o caso (a), simplesmente concatena-se ao resultado parcial de linearização uma indicação de que o que se segue é uma referência a uma estrutura já linearizada e a referência para tal estrutura. É relevante notar que o sistema de linearização deve manter a posição de todas as estruturas já linearizadas.

Já para o caso (b), concatena-se ao resultado parcial de linearização uma indicação de que o que se segue é uma referência a parte de uma estrutura já linearizada, a própria referência à estrutura com o devido deslocamento em relação ao seu início e o tamanho desta parte. O sistema de linearização deve manter, portanto, além da posição de todas estruturas linearizadas, o tamanho de cada estrutura.

Finalmente, para o caso (c), deve-se concatenar ao resultado parcial da linearização uma indicação de que o que se segue é uma referência a uma estrutura contendo outra estrutura previamente linearizada. Concatena-se também o tamanho do que ainda não fora linearizado e o resultado da linearização do que ainda não havia sido linearizado; em seguida acrescenta-se uma referência ao resto da estrutura previamente linearizada. Além disso, deve-se atualizar todas as referências anteriores para as sub-estruturas como sendo referências para uma parte de uma estrutura maior, com o devido deslocamento. Logo, é necessário também manter além da posição para cada estrutura linearizada, o seu tamanho.

Já no processo de deslinearização, cuidados devem ser tomados para colocar as referências corretas das estruturas já deslinearizadas. Novamente temos as três situações acima mencionadas. A detecção desses três casos ocorre facilmente, porque o processo de linearização marcou corretamente cada estrutura como nova, como referência a outra estrutura já linearizada, como parte de uma estrutura já linearizada ou ainda como parte de uma estrutura ainda não totalmente linearizada.

No primeiro caso, estamos deslinearizando uma estrutura já deslinearizada: basta inserir uma referência para esta estrutura. Logo, deve-se manter durante todo o processo de deslinearização as posições de todas as estruturas previamente deslinearizadas.

Para o segundo caso, deslinearização de uma referência ao interior de uma estrutura já deslinearizada, a solução também é simples: põe-se a devida referência com o deslocamento correspondente a sub-estrutura.

E para o último caso, referência para uma estrutura com partes recuperadas e partes ainda não recuperadas, simetricamente à solução de linearização, deve-se recuperar as partes ainda linearizadas e atualizar as devidas referências.

DESENVOLVENDO APLICAÇÕES COM O SISTEMA LINEAR

A primeira versão do Sistema Linear já foi implementada e é composta por duas partes: o aplicativo *linprep* e a biblioteca *liblinear.a*, que contém as funções *linear* e *unlinear*.

O usuário deve criar um arquivo de definições de tipos na linguagem XDR.

O arquivo de definições é processado pelo aplicativo *linprep*, que como o próprio nome sugere, faz uma preparação para o processo de linearização, gerando dois arquivos. Primeiramente um arquivo ".h", com as definições em ANSI C dos tipos dos dados a serem linearizados. Este deve ser incluído (*#include*) nos arquivos que fizerem referências aos tipos nele descritos.

O outro arquivo gerado é uma tabela de tipos internos.

Após a utilização de *linprep*, basta utilizar as funções de linearização quando forem necessárias e ligar o projeto com a biblioteca linear.

Seguem os protótipos das funções *linear* e *unlinear*.

```
int linear(int outFd, int stFd, char * rootObj, int rootType, int flags);
```

outFd: arquivo de definição de tipos (.x) que irá conter as estruturas linearizadas

stFd: arquivo tabela (.tab) que irá conter as definições dos tipos a serem linearizados

rootObj: apontador para a raiz da estrutura a ser linearizada

rootType: tipo previamente definido (em .x) do apontador para a raiz da estrutura a ser linearizada

flags: parâmetros adicionais para linearização, tais como suporte a circularidades

```
int unlinear(int inFd, int stFd, char **rootObj, int rootType, int flags);
```

inFd: arquivo que contém as estruturas linearizadas

stFd: arquivo tabela (.tab) que contém as definições dos tipos linearizados

rootObj: apontador para a raiz da estrutura a ser recuperada

rootType: tipo do apontador para a raiz da estrutura a ser recuperada

flags: parâmetros adicionais para recuperação, tais como suporte a circularidades

EXTENSÕES

A versão atual do Linear será estendida para suportar:

- diferentes *front-ends* , i.e., especificação de tipos em outras linguagems de definição além de XDR;
- linearização e deslinearização para múltiplas plataformas;
- controle de versões das estruturas linearizadas;
- geração do arquivo tabela a partir do compilador CmD [Tel93, Gon94];
- compactação das estruturas linearizadas.

O aplicativo *linprep* será estendido de forma a aceitar diferentes *front-end*s, ou seja, além de XDR outras linguagens de definição como IDL (CORBA) poderão ser utilizadas para especificar os tipos envolvidos no processo de linearização. Seu *back-end* poderá variar, gerando arquivos de definições para C++ e CmD. Dessa forma, o sistema Linear poderá linearizar e deslinearizar objetos.

Tipos de dados serão insensíveis a mudança de *hardware*. Como parâmetro do *linprep*, serão especificadas as plataformas (sistema operacional e arquitetura de *hardware*) a que se deseja exportar o arquivo linearizado. Assim vários arquivos tabela serão gerados, um para cada plataforma. Existirão versões de *linear* e *unlinear* para as plataformas mais conhecidas. O funcionamento multi-plataforma se dará da seguinte forma: a função *linear* lineariza uma estrutura de dados para as plataformas escolhidas, e a função *unlinear* deslineariza para a corrente plataforma, não importando para qual plataforma o corrente arquivo se encontra. As informações suficientes de qual a plataforma utilizada para a linearização constam no arquivo tabela.

O controle de versões no Linear consistirá em uma filtragem e expansão dos tipos de dados contidos no arquivo linearizado. O processo de linearização se dará de maneira convencional. Já o processo de deslinearização será distinto. A função *unlinear* receberá como parâmetro adicional um segundo arquivo tabela contendo os novos tipos dos dados que estão linearizados. Cada tipo poderá ser redefinido, e em caso de tipos agregados (*struct*s e *enum*s), campos poderão ser eliminados e outros poderão ser acrescentados. Tipos redefinidos precisarão de funções de conversão. Já campos filtrados simplesmente deixarão de constar na estrutura deslinearizada em memória. Os dados correspondentes a esse campo serão descartados. Para novos membros, valores *default* serão definidos no arquivo tabela adicional para especificar o valor quando da deslinearização do correspondente agregado.

A geração do arquivo tabela em CmD dispensa o uso do *linprep*. Durante a compilação de um programa, a tabela de tipos do compilador é suficiente para a criação do arquivo tabela. Assim, durante a compilação de um programa em CmD, o compilador pode criar automaticamente tudo que for necessário para linearizar e deslinearizar as classes envolvidas.

A compactação das estruturas linearizadas visa minimizar o tempo de acesso a disco e tempo de transferência na rede. Considerando que estes fatores são predominantes quando comparados ao tempo de processamento em CPU, seria preferível linearizar toda a estrutura de dados, compactá-la e só depois disso armazená-la em disco.

CONCLUSÃO

O Linear foi incorporado no produto *Undelete for Unix*, desenvolvido no Laboratório A-HAND em 1994. Recentemente, em função da sua utilização no Projeto ASAP (financiado pelo Protem/CC, CNPq), melhoramos seu código e vamos incorporá-lo dentro do compilador para CmD. Como o CmD é uma linguagem que suporta a noção de objetos distribuídos, é essencial uma ferramenta de linearização de estruturas. Com o bom desempenho do Linear e nosso domínio do seu código ele é a escolha mais adequada. Com a incorporação de novos *fornt-ends*, em especial para C++, ele poderá ser útil ao desenvolvimento em geral.

REFERÊNCIAS

[Gon94] Objetos distribuídos
Celso Gonçalves Jr.
Tese de mestrado, DCC-IMECC-UNICAMP, agosto 1994.

[HeN86] D. GANDALF: Software Development Environments,
Habermann, A. N. e Notkin
IEEE Transactions on Software Engineering, vol. 12, nº. 12, Dez 1986, pp. 1117-1127.

[Lom87] IDL: Sharing Intermediate Representations
Lomb, D. A.
ACM Transactions on Programming Languages and Systems, Vol. 9, nº. 3, Jul 1987, pp. 297-318.

[Tel93] A linguagem de programação Cm (versão 2.0x)
Alexandre P. Teles
Tese de mestrado, DCC-IMECC-UNICAMP, outubro 1993.

[Sun88] Network Programming Guide
Unix Programming Documentation, Sun Microsystems, 1988.