BOLETIM DO

# plurix®

NCE

7

Fev.Mar.Abr.
Ano 2 — Nº 7
1989

Núcleo de Computação Eletrônica
Universidade Federal do Rio de Janeiro

## EDITORIAL

Este nº 7 do Boletim do PLURIX já estava fechado quando soubemos pelo jornal que a SEI havia liberado o licenciamento do System V.3. Esta liberação nos deixou perplexos não pelo ato em si, que devido a fortes pressões já sabíamos que viria, mas pela rapidez de sua resolução, como se fosse para contribuir deliberadamente para a inviabilização das alternativas nacionais. Esta liberação tem fortes implicações para aqueles que no Brasil tentam (ou tentavam) criar um sistema operacional segundo padrões internacionais mas, ao mesmo tempo, com tecnologia nacional. Uma análise mais aprofundada da situação ficará para o Boletim do PLURIX nº 8.

Neste número 7 mostramos, na seção Padronização, o processo do X/OPEN, ao qual um sistema operacional com filosofia UNIX pode ser submetido para receber um selo de certificação de conformidade. Este processo abre possibilidade para que um SOFIX desenvolvido independentemente da AT&T possa, uma vez obtido o selo do X/OPEN, ser aceito internacionalmente como sistema padronizado. Em particular, o SOX da Cobra já está sendo submetido aos testes do X/OPEN e, em breve, exibirá este selo.

Na seção Perspectiva fazemos uma análise das possibilidades de definição e aceitação das chamadas "Application Binary Interfaces" (ABI). A definição e, principalmente, a aceitação de ABI's para uma determinada arquitetura de computadores permitirá que um software para um SOFIX possa ser distribuído como hoje se faz com o software para MS-DOS. As ABI's permitirão a compatibilidade dos programas em nível da linguagem objeto. Entretanto, as perspectivas para que isto venha realmente a ocorrer não são muito animadoras.

Neste número inauguramos uma nova seção: Avaliação de Desempenho. Desde o início do desenvolvimento do PLURIX temos nos preocupado com o desempenho do sistema. Temos aproveitado inúmeros *benchmarks* reconhecidos internacionalmente, como também temos construído os nossos próprios. Gostaríamos de compartilhá-los com os nossos leitores de forma que os mesmos possam ser utilizados nos mais diversos sistemas. Com isso, comparações podem ser feitas no sentido de apontar, para cada tipo de aplicação, quais são os melhores SOFIX disponíveis.

Na seção Mundo PLURIX apresentamos dois artigos. O primeiro descreve as técnicas de otimização de compiladores que estão sendo implementadas no compilador "C" do PLURIX. Os primeiros resultados são promissores e este otimizador estará disponível junto com a Versão 3.0 do PLURIX. O segundo artigo descreve as informações adicionais que os módulos objeto requerem para que possam fazer uso, em sua total potencialidade, do depurador simbólico do PLURIX. É importante lembrar que, para se obter o máximo de eficiência no desenvolvimento de software, compilador, otimizador, *assembler*, editor-ligador (*linkage editor*) e depurador simbólico devem fazer parte de um todo coerente, como é o caso no PLURIX.

Finalmente gostaria de chamar a atenção para uma preocupação corrente nos E.U.A. que traz ensinamentos para nós. O IEEE (Institute of Electrical and Electronics Engineers) publicou um documento intitulado "U.S. Supercomputer Vulnerability" onde, abandonando a sua posição supostamente imparcial, alerta os americanos para o perigo que representam os japoneses por estarem superando os americanos na área de supercomputadores. O documento afirma que, embora os americanos não estejam percebendo, uma crise real existe e que a base tecnológica americana está ameaçada. Afirma ainda que para superar esta crise será necessária uma solução envolvendo uma cooperação integrada, englobando indústria, universidades e governo. E continua dizendo que aparentemente tal solução vai requerer uma coordenação por parte do governo que praticamente nunca foi adotada em tempo de paz, e que o perigo é real e não está limitado aos supercomputadores, mas é apenas o primeiro passo no processo de total domínio tecnológico japonês. Sugere, finalmente, a criação de uma nova agência civil ligada diretamente ao governo federal para coordenar este esforço.

Muitas Universidades americanas, porém, não compartilham deste prognóstico. Afirmam que o importante não são os supercomputadores em si mas a forma de usá-los, isto é, software. E reagem à proposta que devem dar preferência aos equipamentos de fabricantes americanos, alegando que devem utilizar o melhor supercomputador que existe e que, atualmente, é o japonês. Concordam os signatários do documento do IEEE que os japoneses ainda estão mais atrasados em software mas ponderam que, com a tendência para sistemas abertos, os japoneses em breve terão à sua disposição todo o software do mundo e, assim, a ameaça japonesa poderá se concretizar.

O Departamento de Defesa americano, *"which sees national security in every byte"*, proibiu, através do Legislativo, a importação de qualquer supercomputador estrangeiro (leia-se obviamente japonês) em 1988. Enquanto isso, o Departamento de Comércio americano procura enquadrar os japoneses nos regulamentos, supostamente não respeitados por eles, do GATT (General Agreement on Tariffs and Trade).

Se nesta novela trocarmos os atores, a menos da parte em que a indústria, universidades e governo cooperam, o enredo nos parecerá bastante familiar. E onde se encontrará o preceito absoluto de que somente com abertura do mercado às tecnologias mais avançadas é que se pode desenvolver um país? Este acesso imediato às tecnologias mais avançadas, quando verdadeiramente possível, é realmente fundamental? Ou será que não existem objetivos estratégicos mais importantes? Quanto ao que é sempre recomendado para o "bem" do Brasil parece-nos um caso típico de "faça o que eu digo mas não faça o que eu faço". Lamentavelmente, como no caso do software, começamos a fazer o que eles dizem.

PLURIX é Marca Registrada do NCE/UFRJ
UNIX é Marca Registrada da AT&T

O Futuro Incerto das ABI's

O Selo de Conformidade do X/OPEN

Avaliação de Desempenho do PEGASUS/PLURIX

PERSPECTIVA

# O futuro incerto das "Application Binary Interfaces"

Os computadores que se utilizam do sistema operacional MS-DOS e similares têm uma inegável vantagem sobre aqueles que se utilizam de um SOFIX: disponibilidade de software. Esta disponibilidade se caracteriza não só pela existência dos mais variados tipos de software aplicativo e de suporte mas, principalmente, pela facilidade com que se compra um software (em disquete, por exemplo) imediatamente pronto para ser executado. A adapatação do mesmo, se requer alguma, é trivial e praticamente qualquer usuário pode fazê-la.

Esta facilidade se deve a três razões principais que se realimentam positivamente. A primeira razão é a enorme quantidade de IBM-PC e clones que existe atualmente no mundo. Esta base instalada dá a escala necessária para viabilizar praticamente qualquer desenvolvimento de software. A segunda razão é que o sistema operacional MS-DOS se tornou um padrão de fato (produto de um único fabricante que domina praticamente 100% do mercado) para os IBM-PC e clones. E a terceira razão se deve ao fato de a arquitetura dos IBM-PC e seus clones ser baseada em microprocessadores da INTEL, cujos modelos mais recentes e de maior desempenho (80286, 80386) continuam capazes de executar as mesmas instruções de máquina dos modelos anteriores (8086, 8088) (compatíveis em nível de linguagem de máquina).

Os SOFIX de maneira geral (incluindo o UNIX) têm-se mantido já há alguns anos como os sistemas operacionais do futuro. Este futuro, entretanto, está custando a chegar, justamente pela menor disponibilidade de software comparativamente a um MS-DOS. Não que estes softwares aplicativos e de suporte não existam. O problema está no fato de que no momento existem inúmeros SOFIX (mais de 30, com certeza) sendo executados nos mais diversos processadores (micros a *mainframes*) e nas mais diversas arquiteturas (mesmo quando usam um mesmo processador), o que torna praticamente impossível comprar um software desenvolvido para um SOFIX de um fabricante e executá-lo em qualquer outro sem uma custosa adaptação (transporte). As *software houses* no exterior mantêm equipes de desenvolvimento, suporte e manutenção para cada SOFIX julgado importante. O custo dessas equipes é altíssimo, o que se reflete no custo do software e numa menor disponibilidade. Os custos seriam bem menores se houvesse uma certa padronização. Esta diminuição de custos e aumento da disponibilidade de software é a grande promessa (difícil de ser cumprida, por sinal) das *Application Binary Interfaces* (ABI's). Com a definição e a aceitação de uma ABI, pelo menos para uma mesma família de processadores, os softwares em linguagem objeto seriam intercambiáveis da mesma forma como acontece hoje com os do MS-DOS.

Uma *Application Binary Interface* (ABI) pode ser definida como um conjunto de padrões, para uma específica arquitetura de computadores, com os quais uma comunidade de usuários e fabricantes concorda inteiramente e que, se respeitada por aqueles que desenvolvem software, torna os seus produtos transportáveis, sem necessitar quaisquer modificações, entre plataformas de hardware que utilizam essa arquitetura específica.

Lamentavelmente, as perspectivas para que se cumpra a promessa dessas ABI's não são das mais animadoras. Gostaríamos de comentar algumas das dificuldades da definição e da aceitação de uma ABI, de forma a mostrar aos nossos leitores que, embora devam ter todo nosso apoio, não nos surpreenderemos se elas nunca vierem realmente a ser utilizadas.

Em primeiro lugar, se é desejado compatibilidade a nível do módulo objeto, é obviamente necessário começar com a compatibilidade em nível do módulo fonte. Ainda não se chegou a este acordo. Existem inúmeras implementações de SOFIX diferentes. A tendência é de se reduzir este número. O esforço do X/OPEN, por exemplo, é digno de nota (veja neste número "O processo de verificação e certificação do X/OPEN"). Mas, mesmo assim, teremos num futuro próximo pelo menos dois padrões: o da AT&T e o da OSF (Open Software Foundation).

Suponhamos que a padronização em nível de módulo fonte seja resolvida. Qual o formato do módulo objeto? Basicamente quase todos os SOFIX utilizam o formato COFF (Common Object File Format). O COFF, porém, é complexo e tão flexível que tem-se comentado que há módulos no formato COFF de um fabricante que não são lidos corretamente por um outro.

Suponhamos que todos os SOFIX entendam um formato objeto universal. Os SOFIX serão capazes de executá-los? Mais alguns problemas! As chamadas ao sistema operacional (*system calls*) são codificadas através de rotinas que trocam o modo do processador de usuário para supervisor a fim de que tais serviços (E/S, por exemplo) possam ser executados com segurança num ambiente multiusuário. Essas chamadas usam um número como identificação e este número estará no módulo objeto. Assim, um SOFIX pode fazer um "write" ser identificado pelo número 17, enquanto em outro este mesmo "write" é identificado por 42. Em nível de módulo fonte, esta diferença é irrelevante, pois cada um deles será compilado corretamente pelo compilador específico de cada sistema.

Suponhamos que as *system calls* correspondentes em cada SOFIX tenham os mesmos números. Agora é preciso passar os parâmetros para elas. Em geral isto é feito para garantir um melhor desempenho, através dos registradores do processador. Mas quais registradores? Um SOFIX usa os registradores 0, 1, 2, 3 e apresenta o valor de retorno no 0. Outros usam o 4, 5, 6, 7 e o valor de retorno no 7.

Suponhamos que não somente os parâmetros passados pelas *system calls* sejam entendidos corretamente, mas também os das diversas rotinas (como

está organizada a "stack"?), além dos tamanhos dos inteiros (16 ou 32 bits?), tratamento dos sinais, etc. O programa pode ser executado? Depende da gerência de memória. Computadores que usam um mesmo processador podem ter gerências de memória diferentes. Pode acontecer que uma gerência de memória não permita que determinadas partes do programa (principalmente dados) possam ser alocados nos endereços estipulados no módulo objeto.

Suponhamos que tudo o que foi mencionado até agora seja compatível. Teremos ainda que nos preocupar com o meio físico para transportar o módulo objeto. Qual será, para um SOFIX, a formatação padrão de um disquete? É comum encontrar-se computadores que possuem unidades de disquete cujos SOFIX se comunicam através de linha de comunicação porque os disquetes são incompatíveis! No caso de fitas magnéticas, o formato "tar", embora deixando muito a desejar (não tem multivolume, não tem controle de erro, etc), é considerado um padrão de fato.

Suponhamos que se tenha conseguido padronizar até o meio físico de transporte. A padronização até este ponto, utilizando o SVID (System V Interface Definition), foi o que a Motorola realmente fez junto com fabricantes que se utilizam de família de microprocessadores 68020 e 68030. Foi publicado o "MC68030 UNIX Binary Portability Standard", por exemplo. É preciso, porém, agora, que os fabricantes utilizem este padrão, isto é, que a proposta de padronização seja aceita.

Um dos maiores fabricantes que se utilizam dos processadores da Motorola e do UNIX, por exemplo, é a Apple. O seu sistema operacional A/UX poderia ser um dos primeiros candidatos a se utilizar do padrão. A Apple, porém, quer manter o seu diferencial no mercado e se mostra reticente em apoiar um padrão binário diferente do seu, alegando que já possui a sua própria ABI através do seu Mac Toolbox.

A situação é um pouco menos crítica nos processadores da INTEL e deverá estar mais bem equacionada, em breve, para os processadores RISC. A SUN, com o SPARC, a Motorola com o 88000, etc. já estão definindo padrões binários para UNIX antes que cada implementador defina o seu. Se estes padrões vão ter sucesso, só o tempo dirá.

Se ainda formos levar em consideração aspectos gráficos (X/Windows, por exemplo) e de comunicação (protocolos, etc.), vemos que a situação não é realmente muito animadora em termos de *Application Binary Interfaces*.

Como conclusão, temos a recomendar que usuários de SOFIX não comprem software *off the shelf*, a não ser que se possua exatamente o computador para o qual ele tenha sido gerado. É mais garantido comprar software para SOFIX somente através de *software houses* que dominem a tecnologia do produto que vendem e que sejam indicadas por fabricantes de hardware, que detêm o total controle da tecnologia do sistema operacional. A prática comum no Brasil de se comprar software sem que haja alguém no país que tenha acesso aos módulos fonte do mesmo, se funciona para o MS-DOS, pode ter efeitos desastrosos tanto para usuários quanto para fornecedores de sistemas operacionais com filosofia UNIX. (NF)



# O processo de verificação e certificação do X/OPEN

Já faz alguns anos que a comunidade de usuários de sistemas operacionais com filosofia UNIX (SOFIX) sente a necessidade do estabelecimento de um padrão para tais sistemas. Esforços têm sido feitos desde 1984 através da criação de grupos de interesse e consórcios de fabricantes, no sentido do estabelecimento de tal padrão (veja "Padronização: As Diversas Entidades e suas Propostas", Boletim do PLURIX nº 1, ago/out 87). Muitas propostas já foram publicadas (veja Boletim do PLURIX nºs 2, 3 e 4), mas apenas mais recentemente estão se tornando disponíveis conjuntos de programas ("suites") de teste capazes de verificar a conformidade de um SOFIX a um determinado padrão.

Uma das "suites" mais comuns é a SVVS (System V Verification Suite) que, como o nome explicita, testa a conformidade de um sistema operacional ao padrão definido pelo SVID (System V Interface Definition). Esta "suite", porém, como era de se esperar, é de propriedade da AT&T e está disponível apenas para empresas que tenham licenciado o UNIX. Para as empresas que desenvolveram os seus SOFIX independentemente do UNIX da AT&T, existem algumas "suites" disponíveis no mercado internacional e que podem ser adquiridas sem restrições. Um exemplo é a "suite" de teste da empresa americana Perennial, que vem sendo usada no Brasil já há algum tempo pela COBRA nos testes de conformidade do sistema operacional SOX.

A simples afirmação de um fabricante que seu sistema operacional é capaz de executar uma "suite" de teste sem erros não garante ao usuário que o sistema seja padronizado. Para dar esta garantia é necessário que um organismo internacional independente de fabricante, utilizando critérios determinados, defina uma "suite" padrão, estabeleça um procedimento para certificação dos sistemas operacionais e crie uma caracterização externa visível para que usuários, mesmo leigos, possam constatar que determinado sistema operacional segue o padrão estabelecido. A obtenção de um selo de conformidade do X/OPEN, conferido ao término de um processo bem sucedido de certificação, vem justamente cumprir estes objetivos.

Em 12 de setembro de 1988 o consórcio de empresas X/OPEN anunciou o seu processo de verificação e certificação. Como padrão foi definido o Ambiente Comum de Aplicações (Common Applications Environment), publicado no Guia de Portabilidade do X/OPEN (veja "O Padrão X/OPEN", Boletim do PLURIX nº 3 fev/abr 88). A promessa do

# PADRONIZAÇÃO

X/OPEN é que qualquer software aplicativo, que possua o selo de conformidade, para poder ser executado em qualquer sistema operacional, que também possua o selo de conformidade, requer apenas uma simples recompilação. Não custa lembrar que a compatibilidade de software entre SOFIX se faz sempre em nível de código fonte, embora existam já tentativas para padronização de código objeto (veja neste número "O Futuro Incerto das ABI'S").

As vantagens de se trabalhar com sistemas padronizados já foram discutidas em outros artigos, mas não custa relembrar: maior facilidade na escolha de hardware e software; redução na dependência de um fabricante específico; diminuição nos custos de treinamento, manutenção e suporte; integração mais fácil e rápida de sistemas provenientes de fabricantes diferentes; e, finalmente, alongamento do ciclo de vida dos sistemas por retardar o processo de obsolescência.

Existem três "suites" de teste do X/OPEN para sistemas operacionas. Elas estão orientadas para os Guias de Portabilidade 1 (1985), 2 (1987) e 3 (1988), respectivamente. A "suite" para o Guia de Portabilidade 2, por exemplo, é composta de 1.292 testes da linguagem "C", 1.578 testes do sistema operacional, 685 testes de internacionalização e 190 testes de acesso a arquivos seqüenciais indexados (ISAM), dando um total de 3.745 testes. Para o Guia de Portabilidade 3 prevê-se, aproximadamente, 4.200 testes.

O sistema que for capaz de executar sem interrupção todos os testes de uma determinada "suite" sem que produza qualquer erro e sem intervenção do operador, ganha o direito ao selo de conformidade tipo "PLUS" seguido do número do ano da publicação do Guia de Portabilidade a que se refere a "suite". Em geral, as características dos sistemas descritos pelos Guias de Portabilidade mais antigos representam um subconjunto das características descritas nos mais modernos. Assim, sistemas que apresentam total conformidade com padrão X/OPEN podem ostentar selos "PLUS 85", "PLUS 87" ou "PLUS 88".

Um sistema operacional, entretanto, não requer uma conformidade total para receber um selo de conformidade. Um sistema operacional não precisa possuir todas as caracteríistias de internacionalização ou de acesso seqüencial indexado, por exemplo, para poder executar corretamente a grande maioria dos aplicativos com selo de conformidade. Assim existem selos "BASE 85", "BASE 87" e "BASE 88".

Há selos também para caracterizar a conformidade de um componente específico do sistema operacional ou de um software aplicativo ("COMPONENT 88""ou "APPLICATION 88", por exemplo).

Há três formas de se ter acesso a estas "suites" de teste. Os membros do X/OPEN (anuidade US$400.000) têm acesso gratuito e podem executá-las livremente sem supervisão de terceiros. Os não-membros do X/OPEN têm acesso às mesmas através dos centros de teste que a UniSoft (única empresa autorizada até o momento) montou nos E.U.A., Europa e Japão. Não-membros também poderão se utilizar dos serviços de centros nacionais de teste, assim que estes vierem a ser estabelecidos.

Uma vez aprovado no "exame", o sistema operacional pode utilizar o selo X/OPEN a que tem direito mediante o pagamento nominal de US$8.50, para sistemas mono-usuário, e US$25 para sistemas multi-usuário. Não há qualquer cobrança pelo uso do selo para aplicativos que tenham passado nos testes.

A COBRA é o primeiro fabricante brasileiro a submeter o seu sistema operacional ao crivo do X/OPEN. Não sendo membro do X/OPEN, o teste do SOX será efetuado pela UniSoft num período de três a cinco dias. As taxas a serem desembolsadas são bastante pequenas (US$7.500, pelos primeiros dias, e US$1.250, por dia adicional) se compararmos com as vantagens que um selo destes pode proporcionar.

É ainda difícil dizer se o padrão definido pelo X/OPEN é o melhor para os SOFIX. Existem outros, como o POSIX, que ainda estão em elaboração, além dos padrões do NIST (National Institute of Standards and Technology, antigo National Bureau of Standards), e da ISO (International Standards Organization) que precisam ser respeitados. O X/OPEN afirma que todos os padrões que forem relevantes para a caracterização do Ambiente Comum de Aplicações serão oportunamente incorporados ao Guia de Portabilidade e passarão a constar das "suites" de teste dando, assim, um caráter dinâmico ao processo de verificação e certificação.

A criação de um processo de verificação e certificação internacional e independente de fabricante abre uma grande oportunidade para os SOFIX nacionais. A aprovação nos testes do X/OPEN, por exemplo, dará aos sistemas nacionais um passaporte para o mercado internacional e um acesso menos controvertido ao mercado nacional. Estará garantido ao usuário nacional ou estrangeiro a utilização de um sistema que segue um padrão internacional, com a vantagem de que a tecnologia de tais sistemas permanece sob o controle de brasileiros. O selo de conformidade dará a garantia qualitativa, já a quantitativa terá que ser pesquisada pelo próprio usuário, devendo recair a sua escolha no sistema que proporcione o melhor desempenho para a sua aplicação. (NF)

# A avaliação do desempenho do PEGASUS/PLURIX

A avaliação do desempenho de qualquer sistema computacional é sempre um processo difícil. Isto se deve ao fato de haver um grande número de variáveis envolvidas: o processador utilizado, o tipo de gerência de memória, a organização dos arquivos em disco, a linguagem escolhida e o compilador utilizado são algumas delas.

Poderia-se acreditar que o melhor tipo de avaliação é aquela que executa a aplicação final do usuário nos diversos sistemas em avaliação. Mas esta avaliação, além de ser restrita, pode apresentar procedimentos incorretos, que favoreçam um sistema em detrimento de outro.

A avaliação realizada no PEGASUS/PLURIX procurou determinar o desempenho global do sistema, da maneira como se apresentaria ao usuário. Para isto, foi escolhida uma série de programas escritos em linguagem "C", encontrados na literatura especializada [1,2,3].

Como o PLURIX é um sistema similar ao UNIX da AT&T, a escolha de programas escritos em uma linguagem de alto nível permite, para efeito de comparação, a portabilidade destes programas para qualquer ambiente do tipo UNIX. O uso do *assembler* se mostra inconveniente por estar distante das aplicações do usuário, e ser dependente do processador utilizado.

O UNIX é um sistema operacional multiusuário e multitarefa. Desta forma, os diversos processos ocupam simultaneamente partes distintas da memória principal. No caso de haver pouca memória disponível, o disco poderá se tornar um recurso crítico, devido às trocas de contexto entre os diversos processos que competem pelo uso da memória e à excessiva utilização de arquivos temporários alocados desnecessariamente em disco. Estas situações devem ser evitadas, quando possível, sob pena de o tempo de uso do disco mascarar completamente o tempo de uso dos demais recursos do sistema.

O PLURIX, no caso de haver bastante memória, permite que os programas de uso mais freqüente fiquem permanentemente residentes na memória, juntamente com os arquivos temporários do sistema. Estas facilidades diminuem a utilização de disco, permitindo um melhor desempenho do sistema como um todo.

Antes de apresentarmos os programas de avaliação utilizados, é conveniente esclarecermos alguns pontos que devem ser observados, para podermos realizar medidas não tendenciosas em um ambiente UNIX.

Nas medidas realizadas, os programas foram executados em ambiente multiusuário para que o *overhead* normal do sistema pudesse ser levado em conta. Se utilizássemos o modo monousuário (ou administrador), poderíamos obter resultados não confiáveis.

Para medida do tempo de execução foi utilizada a rotina "time" do UNIX, que apesar de introduzir um pequeno *overhead*, oferece um resultado bastante apurado. Esta chamada discrimina os tempos de usuário, supervisor e real gastos pelo programa em execução (veja ao lado).

Cada programa foi executado mais de uma vez, e sem outros usuários no sistema. Os tempos mostrados são a média dos diversos resultados obtidos.

Todo cuidado deve existir para o uso de opções de otimização e declaração de variáveis em registrador [4]. Compiladores mais "espertos" podem modificar ou até mesmo suprimir trechos do programa original. Um caso típico é o compilador retirar um *loop* que não altere o valor final de nenhuma variável.

Na comparação de desempenho entre vários computadores, é comum normalizarmos os resultados em relação a um deles. Neste caso, deve-se utilizar a média geométrica do valor da comparação (razão entre tempos de execução), pois a utilização da média aritmética pode levar a uma interpretação errada dos resultados [5].

Para que o leitor possa avaliar corretamente os resultados apresentados, deve-se indicar as características do hardware da máquina avaliada: memória instalada, memória em disco, velocidade do processador e tipo de processador. Isto permite, por exemplo, explicar diferenças entre computadores do mesmo tipo, mas com configurações diferentes.

Os valores de saída apresentados pela função "time" de um sistema operacional tipo UNIX são os seguintes:

— o tempo de usuário ("user") é o tempo gasto pelo processo executando instruções não privilegiadas;

— o tempo de sistema ("sys") é o tempo gasto executando comandos privilegiados (p.ex., "syscall"), e mais algum *overhead* em nível de sistema operacional (a soma destes tempos costuma ser apresentada também como tempo de processador);

— o tempo real ("real" ou "elapsed") é o tempo decorrido entre o início e o término da execução do programa (não é a soma dos tempos de usuário e sistema. A diferença é o tempo perdido executando operações de E/S e aguardando processamento nas filas dos sistema).

Os ponto mais importantes para avaliação no UNIX são a interface com o usuário ("shell"), os dutos de comunicação entre processos ("pipe"), o acesso aos arquivos em disco, as chamadas ao sistema ("syscall") e a capacidade de processamento de funções de usuário. Os programas escolhidos para avaliação do PEGASUS/PLURIX procuram exercitar estas funções, de modo que o desempenho obtido na execução destes programas estará perto daquele rendimento obtido por um usuário comum.

Os programas utilizados foram chamados de "syscall", "funcall", "piper", "shell", "sieve" e "disktest", que executam os seguintes procedimentos:

— o programa "syscall" realiza diversas chamadas ao sistema, para uma rotina que devolve o número de identificação do processo;

— o programa "funcall" realiza diversas chamadas a uma sub-rotina de usuário;

— o programa "piper" exercita a comunicação entre processos utilizando dutos;

— o programa "shell" é um conjunto de diversos

# AVALIAÇÃO DE DESEMPENHO

utilitários de uso comum no UNIX;

— o programa "sieve" calcula uma seqüência de números primos;

— o programa "disktest" realiza uma série de acessos a arquivos em disco.

Estes mesmos programas também foram utilizados para avaliar o desempenho do PEGASUS/PLURIX operando com mais de um processador, com bons resultados [6].

Os resultados obtidos com a execução destes programas podem ser encontrados nas tabelas I e II. Na tabela I é feita uma comparação com supermicros, e na tabela II com minis e superminis. Os resultados mostram que o desempenho do PEGASUS/PLURIX é equivalente ao VAX-780, e superior aos demais computadores apresentados, entre eles um sistema desenvolvido pela AT&T.

O bom desempenho do PEGASUS pode ser atribuído a sua arquitetura, com controladores inteligentes de E/S e memória cache; e ao cuidado de projeto do sistema, que procurou alcançar uma solução integrada de hardware e software, adaptando um às necessidades do outro.

A avaliação realizada procurou mostrar as potencialidades do sistema PEGASUS/PLURIX e, considerando o fato de este não ser um produto comercial, mostrou excelentes resultados. O transporte do sistema operacional PLURIX para outras arquiteturas permitirá que este mesmo conjunto de programas seja executado em computadores comerciais. Com isto poderemos obter resultados da comparação com outros sistemas de ambientes UNIX que venham a ser lançados no mercado.

Os resultados das avaliações certamente auxiliarão o usuário final na escolha de um sistema adequado às suas necessidades.

Cópias destes programas poderão ser obtidas com o autor, no endereço deste boletim, ou nas referências abaixo relacionadas. Esperamos poder contar com os resultados da execução destes programas obtidos pelos leitores nos seus respectivos sistemas para divulgação neste boletim. Sugestões de novos programas também são aceitas, e esperamos em breve publicar maiores detalhes sobre programas de avaliação para aplicações científicas. **(Gabriel Pereira da Silva).**

## BIBLIOGRAFIA

(1) Hendricks, S.; Denney, M.; Halliday, D. "Performance Tests on Unix Micros", UNIQUE, Denville, NJ, 3(7): 20-28, Summer 1984;

(2) Hinnant, D.F. "Benchmarking Unix Systems", Byte, Peterborough, NH, McGraw Hill, (8): 132-5, 400-9, Aug. 1984;

(3) Gilbreath, J. & Gilbreath, G. "Erathostenes Revisited", Byte, Peterborough, NH, McGraw Hill, 9(2): 283-326, Jan. 1983;

(4) Grehan, Richard et alli "Introducing the New Byte Benchmarks", Byte, NH, McGraw-Hill, 13(6): 2329-66, Jun. 1988;

(5) Fleming, P.J. & Wallace, J.J. "How to not Lie with Statistics: The Correct Way to Summarize Benchmarks Results", Comm. of the ACM, New York, 30(3): 218-21, Mar. 1986;

(6) Silva, Gabriel P. "Avaliação de um Ambiente Unix com Múltiplos Processadores", Anais do II Simpósio Brasileiro de Arquitetura de Computadores — SBC, Águas de Lindóia, SP, Vol. 2: 11.b.6, Set. 1986

## TABELA I

| Nome do Progr. | AT&T 3B2 | ALTOS 586-30 | ZILOG 11+ | PEGASUS 32.1 |
|---|---|---|---|---|
| Sieve | 7.1 (7.0) | 7.0 (6.9) | 5.0 (4.9) | 3.1 (3.0 + 0.0) |
| Syscall | 3.5 (3.4) | 5.5 (4.8) | 4.0 (4.0) | 1.5 (0.1 + 1.4) |
| Funcall | 48.0 (47.8) | 37.0 (36.8) | 15.0 (14.9) | 18.9 (18.8 + 0.1) |
| Piper | 10.2 (5.3) | 11.0 (6.5) | 9.0 (3.9) | 10.5 (0.1 + 4.6) |
| Disktest | 30.3 (12.5) | 70.0 (15.1) | 30.0 (10.7) | 33.8 (0.2 + 15.1) |

**AT&T 3B2** = 2 Mb de memória, 32 Mb de disco, CPU 32100, Unix System V;
**ALTOS 586-30** = 512 Kb de memória, 30 Mb de disco, CPU 8086, Xenix Versão 7;
**ZILOG 11+** = 512 Kb de memória, 52 Mb de disco, CPU Z8000, Zeus Versão III
**PEGASUS** = 4 Mb de memória, 85 Mb de disco, CPU 68020 @ 12 Mhz, Plurix Versão 1.0.
Tempos em segundos: real (user + sys)

## TABELA II

| Nome do Progr. | VAX-780 | VAX-750 | PDP-11/70 | PEGASUS 32.1 |
|---|---|---|---|---|
| Sieve | 1.7 (1.5 + 0.1) | 2.4 (1.7 + 0.1) | 2.3 (1.6 + 0.1) | 1.6 (1.5 + 0.0) |
| Syscall | 4.8 (0.4 + 4.0) | 7.0 (0.8 + 6.2) | 8.0 (0.2 + 7.5) | 3.8 (0.3 + 3.5) |
| Shell | 3.3 (0.3 + 1.8) | 3.8 (0.4 + 1.5) | 4.0 (0.2 + 1.7) | 3.6 (0.3 + 2.8) |
| Piper | 3.2 (0.1 + 1.2) | 4.6 (0.2 + 2.1) | 3.1 (0.0 + 3.4) | 4.5 (0.0 + 2.0) |

**VAX-780** = 4 Mb de memória, 256 Mb de disco, Unix 4.1 BSD;
**VAX-750** = 2 Mb de memória, 121 Mb de disco, Unix 4.1 BSD;
**PDP-11/70** = 1 Mb de memória, 400 Mb de disco, Zeus Versão III;
**PEGASUS** = 4 Mb de memória, 85 Mb de disco, CPU 68020 @ 12 Mhz, Plurix Versão 1.0.
Tempos em segundos: real (user + sys)

# "PLC2", um otimizador para o compilador "PLC"

O compilador "PLC" para a linguagem "C" do PLURIX foi descrito no boletim nº 4 de maio/ junho/ julho de 1988, e está disponível desde a versão 2.0. Para complementá-lo, estamos iniciando a concepção e implementação do otimizador de código "PLC2", para melhorar o código objeto gerado pelo compilador.

O nome "otimizador" na realidade não é totalmente adequado, pois em geral não conseguimos obter o código "ótimo". Um nome mais realista seria um "melhorador" de código, mas o termo "otimizador" já está incorporado à tradição da informática, e será este que usaremos.

A existência de um otimizador de código objeto não deve ser interpretado como a indicação de que o código gerado por um compilador seja de má qualidade; inevitavelmente, o compilador "PLC", durante a sua complexa tarefa de traduzir as construções de uma linguagem de alto nível como "C", não consegue se preocupar simultaneamente com todos os detalhes e idiossincrasias de uma linguagem de máquina (ou simbólica) de um processador. Uma vez concluída a tradução, o otimizador pode se concentrar exclusivamente nestes detalhes e tentar aproveitar ao máximo as características do processador.

Um outro motivo que contribui para que o compilador não gere um código melhor é o fato de que ele realiza a tradução em apenas um passo. Isto significa que em certas situações ele tem de tomar decisões que envolvem trechos do programa que ainda não foram vistos. Portanto, de certo modo, o otimizador pode ser visto como um "segundo passo" do compilador.

Apesar da prioridade de otimização ser o aumento da velocidade, o tamanho do programa objeto em geral também diminui, pois, normalmente as transformações que aumentam a velocidade (como por exemplo, substituir um par de instruções por uma única instrução) também reduzem o tamanho.

Pretendemos incorporar no "PLC2" diversas fases com o objetivo de melhorar o código. Entre elas citamos:

1. No começo de cada função, o compilador gera instruções para guardar os conteúdos dos registros (tanto inteiros como flutuantes) que serão utilizados durante a função. No final da função, ela gera as instruções complementares para restaurar os valores destes mesmos registros. No entanto, como a compilação é realizada em apenas um passo, durante a geração das instruções de guarda dos registros, ainda não sabemos quantos, nem quais registros serão utilizados pelo programa, forçando a geração de instruções que (possivelmente) são desnecessárias. O otimizador analisa quais são os registros utilizados, e remove as instruções supérfluas.

2. Na segunda fase, todos os desvios (instruções de "jump" condicionais ou incondicionais) do programa objeto são analisados, procurando-se (entre outros) por desvios cujo destino é um outro desvio e desvios condicionais ao redor de desvios incondicionais. Estas construções são simplificadas, e ao fazer isto, encontramos instruções que nunca serão executadas (e são removidas), seqüências de instruções iguais (da qual apenas um "exemplar" é conservado e as outras substituídas por desvios). Finalmente, são procuradas as instruções geradas por malhas (tais como "for", "while" e "do"), e realizadas melhorias.

3. Nesta fase, são procuradas certas seqüências de instruções que podem ser substituídas por outras seqüências com menos instruções ou instruções mais rápidas. Esta fase é tradicionalmente denominada de "peep-hole" na literatura, pois lembra a visão restrita e localizada proporcionada por um buraco de fechadura.

4. Nesta fase, são examinadas todas as cadeias definidas pelo programa, à busca de cadeias repetidas. Se houver cadeias repetidas, apenas um exemplar de cada é mantido, e o programa é alterado de acordo. Esta fase, cuja velocidade é normalmente de complexidade quadrática, será implementada de forma linear, utilizando-se uma tabela de busca "hash".

5. Nesta última fase, é realizada uma análise simplificada de "flow-graph", na qual é criada e mantida uma tabela dos conteúdos dos registros, que é atualizada à medida em que percorremos o código objeto. Para cada instrução que carrega um registro com um certo valor, consultamos a tabela para verificar se o registro já contém o dado valor, e eliminar a instrução redundante.

Como sabemos, o compilador "PLC" pode (com uma opção adequada) incluir comentários na sua saída em linguagem simbólica, contendo trechos do programa fonte em "C". O objetivo disto é facilitar a associação de um comando do programa original com o respectivo código gerado. O otimizador altera bastante o programa objeto, e com isto esta associação não é mais tão imediata. No entanto, consideramos que, mesmo assim, o otimizador deve permitir a conservação destes comentários em sua saída.

Embora (com restrições) seja possível a utilização de um depurador simbólico (como o "wsdb") também em programas otimizados, não pretendemos (pelo menos inicialmente) permitir o uso do "PLC2" com as pseudodiretivas com as informações para o depurador.

Através de análises preliminares, estimamos que com a utilização do otimizador, o código do kernel do PLURIX deverá ficar da ordem de 10% menor, e a sua velocidade 5% maior (utilizando a chamada ao sistema "getpid" para realizar as comparações). Outras previsões para pequenos programas de teste indicam casos em que a velocidade pode aumentar em até 25%.

Quando o otimizador "PLC2" estiver concluído e sendo plenamente utilizado, poderemos realizar medidas, e verificar se as previsões são corretas. De qualquer modo, esperamos que seja mais uma ferramenta poderosa do PLURIX. **(PS)**

# Extensões do arquivo objeto para uso com o depurador simbólico

Um depurador simbólico (veja "WSDB — O Depurador Simbólico do PLURIX", Boletim do PLURIX nº 4, mai/jul 88) é um depurador que possui uma interface de alto nível com o usuário, ou seja, este se comunica com o depurador com a mesma abstração com que o programa foi escrito.

Por outro lado, a interface entre o sistema e o depurador é essencialmente de baixo nível, exigindo assim que o depurador simbólico exerça, entre outras, a função de "tradutor" na comunicação usuário/sistema durante uma depuração. Nesta "tradução" estão incluídas as transformações de números de linhas de módulos ou nomes de símbolos em endereços e vice-versa. É importante destacar que estas transformações não são automáticas e só serão possíveis mediante a existência de determinadas informações no arquivo objeto (arquivo que contém o código executável do programa).

Algumas destas informações são inerentes ao arquivo objeto, como os tamanhos das diversas partes do arquivo (usado para localizar cada uma delas, já que são seqüenciais) e a data e hora de criação (usada para verificar a correspondência com um arquivo 'core'). Outras informações só são colocadas no arquivo objeto caso sejam requisitadas pelo usuário através da opção '-w', utilizada na compilação/link-edição do programa. Estas informações consistem de duas tabelas: a tabela de números de linhas e a tabela de símbolos para o depurador.

A tabela de números de linhas permite que o usuário acompanhe a depuração pelos módulos fonte. Ela contém, para cada módulo, a correspondência entre os números de suas linhas e os endereços dos respectivos códigos gerados. Somente as linhas geradoras de código constam desta tabela. O início de cada módulo é marcado pela linha zero e pelo índice do respectivo módulo na tabela de símbolos.

A tabela de símbolos para o depurador é uma tabela de símbolos estendida não só por conter mais símbolos que a tradicional, como também por conter mais informações a respeito de cada símbolo. A disposição dos símbolos nesta tabela é importante para o depurador. Eles devem ser colocados numa ordem tal que seja possível identificar a que módulo pertence cada rotina e cada variável estática, e a que rotina pertence cada parâmetro e cada variável automática.

As informações acerca de cada símbolo são:
a) nome;
b) classe:
começo e fim de arquivo, começo e fim de função, começo e fim de bloco, símbolo externo, símbolo estático, símbolo automático, parâmetro, label, nome de estrutura, de união ou enumeração, membro de estrutura, de união ou enumeração.
c) valor:
varia de acordo com a classe e pode ser:
— endereço (começo de função, símbolo externo, símbolo estático e label);
— "offset" na pilha (parâmetro e símbolo automático);
— "offset" a partir de endereço (membro de estrutura);
— valor (membro de enumeração).
d) tipo:
— básico: void, char, uchar, short, ushort, long, ulong, int, uint, float, double, extd, struct, union, enum, membro de enum.
— derivado: nenhum, pointer, função, array.

Alguns símbolos exigem informações complementares. Os vetores possuem as dimensões. Já as estruturas, as enumerações e as uniões e os seus respectivos membros possuem uma correspondência.

Todas estas informações são utilizadas para transformar um depurador em simbólico. A tabela de símbolos possibilita a utilização da sintaxe da linguagem fonte na especificação dos símbolos. Já a tabela de números de linhas possibilita a referência a números de linhas de módulos, evitando a utilização de endereços. (SMBF)

BOLETIM DO plurix

Publicação Trimestral

**Editor**
Newton Faller

**Editor Assistente**
Geni Hirata

**Consultor Técnico**
Pedro Salenbauch

**Colaboradores**
Gabriel Pereira da Silva
Gustavo Peixoto de Azevedo
Newton Faller
Pedro Salenbauch
Rafael Peixoto de Azevedo
Silvia Maria Barbosa Figueira

**Publicação**
Universidade Federal do Rio de Janeiro
Centro de Ciências Matemáticas e da Natureza
Núcleo de Computação Eletrônica
Caixa Postal 2324
CEP 20001 — Rio de Janeiro RJ
Tel.: (021) 290-3212
Telex: (21) 37466

**Correspondência**
Boletim do PLURIX
NCE/UFRJ — Caixa Postal 2324
CEP 20001 — Rio de Janeiro RJ

**Produção Gráfica**
ASB Arte Gráfica e Editora
Tel.: (021) 253-5801
Telex: 2131889 ASBG BR

*Os artigos do Boletim do PLURIX podem ser reproduzidos livremente desde que citada a fonte.*

*Os artigos assinados são de responsabilidade de seus autores.*

BOLETIM DO plurix

IMPRESSO