UM AMBIENTE PARA PROGRAMAÇÃO DE APLICAÇÕES DISTRIBUÍDAS EM REDES DE WORKSTATIONS

Antônio Marinho Pilla Barcellos, João Frederico Lacava Schramm
Carlos Alberto Teixeira Jr., Gustavo Gerhardt, Cláudio Fernando Resin Geyer
{marinho, schramm, catj, gerhardt, geyer}@inf.ufrgs.br

CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
INSTITUTO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Av. Bento Gonçalves, 9500 Bloco IV
Caixa Postal 15064 CEP 91501 Porto Alegre, RS
Tel.: (052) 3368399 (052) 3391355 ramal 6165 -- Fax: (052) 3365576

RESUMO

O sistema operacional de rede heterogêneo HetNOS é composto por um conjunto de camadas de software dispostas sobre SOs "nativos" de forma a compor uma plataforma para programação distribuída. O ambiente é formado pela linguagem de comandos do *shell* do HetNOS, *hsh*, bem como pela interface de chamadas do sistema (acessadas através de uma biblioteca de funções). Em ambos os níveis de interação com o usuário, o conjunto de máquinas "integradas" pelo HetNOS é visto, até certo ponto, como uma "máquina virtual única".

O *hsh* apresenta boa parte das funções vistas em interpretadores de comandos mais tradicionais. É possível lançar, monitorar e terminar processos em qualquer máquina da rede, tal como realizado na máquina local. O Núcleo Distribuído do HetNOS emprega um esquema simbólico de identificação de processos, global e independente de localidade. Aplicações distribuídas são divididas em processos seqüenciais, que interagem por mensagens. Não há conexões, portas, etc., sendo o mecanismo de comunicação fortemente influenciado pelo esquema de identificação de processo, que é global. Este trabalho apresenta o ambiente HetNOS para programação de aplicações distribuídas, descrevendo suas principais características e comparando o mesmo com trabalhos similares como a linguagem SR e as bibliotecas PVM e P4.

ABSTRACT

The HetNOS network operating system is a set of software layers laid over "native" operating systems to provide a distributed programming platform. The environment is composed of the HetNOS shell command language as well as the system calls interface (accessed through a procedure library). In both levels of interaction with users, the set of machines "glued" by HetNOS are, to some degree, seen as a single virtual machine.

The HetNOS command interpreter, namely *hsh*, implements most functions present in more traditional command interpreters. It is possible to spawn, monitor, and terminate processes in any host in the network just like in the local case. The HetNOS Distributed Kernel uses a symbolic, global, location independent. process identification scheme. Distributed applications are split into sequential processes, which interact to each other by message exchange. There are neither connections nor ports, being the communication mechanism strongly influenced by the process identification scheme. This work presents the HetNOS environment for distributed programming, by describing its main characteristics and comparing it with the SR distributed language and the P4 and PVM libraries.

1. INTRODUÇÃO

A construção de aplicações paralelas e/ou distribuídas, atualmente, envolve duas alternativas principais [BAL89]. A primeira delas é utilizar uma linguagem completamente nova, específica à construção desse tipo de aplicações, enquanto a outra é empregar uma linguagem tradicional (como C), acrescida de bibliotecas com primitivas para processamento paralelo-distribuído.

A linguagem em questão pode apresentar paralelismo ou distribuição de forma implícita, tal como Prolog Paralelo [GUE91], ou explícita, tal como SR [AND82, AND93]. No modelo implícito, o paralelismo não é especificado pelo programador, e sim detectado pelo compilador (técnica freqüentemente utilizada em processamento vetorial). No explícito, a linguagem fornece meios (construções) para que programadores convertam uma especificação de um problema em um programa paralelo ou distribuído.

Apesar de serem uma forma mais natural de expressar algoritmos distribuídos, as linguagens não têm encontrado a aceitação inicialmente esperada. Usuários experientes geralmente apresentam restrições à utilização de novas linguagens. As incertezas associadas à manutenção da linguagem (atualização em relação a novos equipamentos, portabilidade, etc.), geram uma certa insegurança quanto à essa abordagem. Muitas vezes o processo de aprendizado ou adaptação de um programador a uma nova linguagem representa um custo demasiado alto para o pessoal treinado em linguagens mais tradicionais como C ou Pascal.

A outra abordagem possível consiste na utilização de uma linguagem familiar, com a adição de bibliotecas para **comunicação em rede**, tal como *sockets* ou TLI *(Transport Layer Interface)* [STE90]. O programador continua a usar a linguagem que domina, com sintaxe e semântica preservadas, precisando apenas aprender a lidar com as funções disponíveis. Outra vantagem dessa abordagem é o seu excelente desempenho, pois tais primitivas de comunicação apresentam estreita relação com o sistema operacional, sendo usualmente fornecidas com o mesmo.

Entretanto, as interfaces de funções *(Application Program Interfaces*, ou simplesmente API's) atualmente disponíveis são pouco amigáveis, pois exigem que o usuário lide com detalhes de baixo-nível, tal como protocolos de comunicação e estruturas de dados com endereços de máquinas e portas. Adicionalmente, o usuário deve identificar a localização dos múltiplos processos na rede, abrir diversas sessões remotas para monitorar a execução de seus processos, enfrentando dificuldades para a depuração de aplicações distribuídas.

O presente trabalho descreve as principais virtudes e problemas da **interface de programação** do HetNOS [BAR93a, BAR93b, EMI92, GER93, PER93, SCH92, ZAR93], uma abordagem **intermediária** entre a simplicidade e o desempenho das bibliotecas e a abstração das linguagens. Demonstra-se seus aspectos através da descrição dos princípios de programação, do ambiente de desenvolvimento e de suas principais primitivas. O HetNOS é uma evolução sobre o esquema com bibliotecas, pois utiliza um núcleo distribuído e um conjunto de módulos servidores para oferecer, através de bibliotecas de funções de alto-nível, uma gama de serviços a programas de usuários. Entre as vantagens propiciadas pelo ambiente, estão: a possibilidade de fazer diferentes aplicações cooperarem em diferentes momentos (não é um único código dividido em processos); a inexistência de módulos extra de programa fonte a integrar, tal como esqueletos de clientes e servidores; e a facilidade no desenvolvimento de aplicações devido ao ambiente de programação familiar e à simplicidade das primitivas.

Inicialmente, é apresentada a arquitetura interna do HetNOS (descrita em detalhe em [BAR93a]). A seguir, são analisados os princípios do modelo de programação no HetNOS, incluindo paradigma de comunicação, gerência de processos e exemplos de primitivas disponíveis. Então, é mostrado o ambiente de desenvolvimento disponível ao programador. Finalmente, compara-se o HetNOS com trabalhos relacionados, incluindo a implementação de um algoritmo distribuído em diversas abordagens.

2. ARQUITETURA DO SISTEMA

O HetNOS pode ser denominado "sistema operacional de rede" por duas razões: por ser um conjunto de camadas de software dispostas sobre um grupo de sistemas operacionais (ditos "nativos") [GEI90a], e porque o grau de transparência de localidade oferecido aos usuários é limitado [TAN85]. A heterogeneidade está presente no HetNOS porque o mesmo permite a execução de processamento cooperativo entre máquinas de arquiteturas distintas e sistemas operacionais semelhantes.

O HetNOS está estruturado em camadas hierárquicas (vide figura 1). Cada camada é composta por um ou mais módulos, sendo que cada módulo é implementado como um processo independente, que não compartilha memória com outros módulos. Por esta razão, a comunicação é implementada com a cópia de dados através de canais de comunicação. Tal organização estruturada é mais genérica e portável porque limita a interação entre os módulos à especificação das interfaces (mensagens aceitas e possíveis respostas). Por outro lado, apresenta desempenho inferior aos sistemas monolíticos.

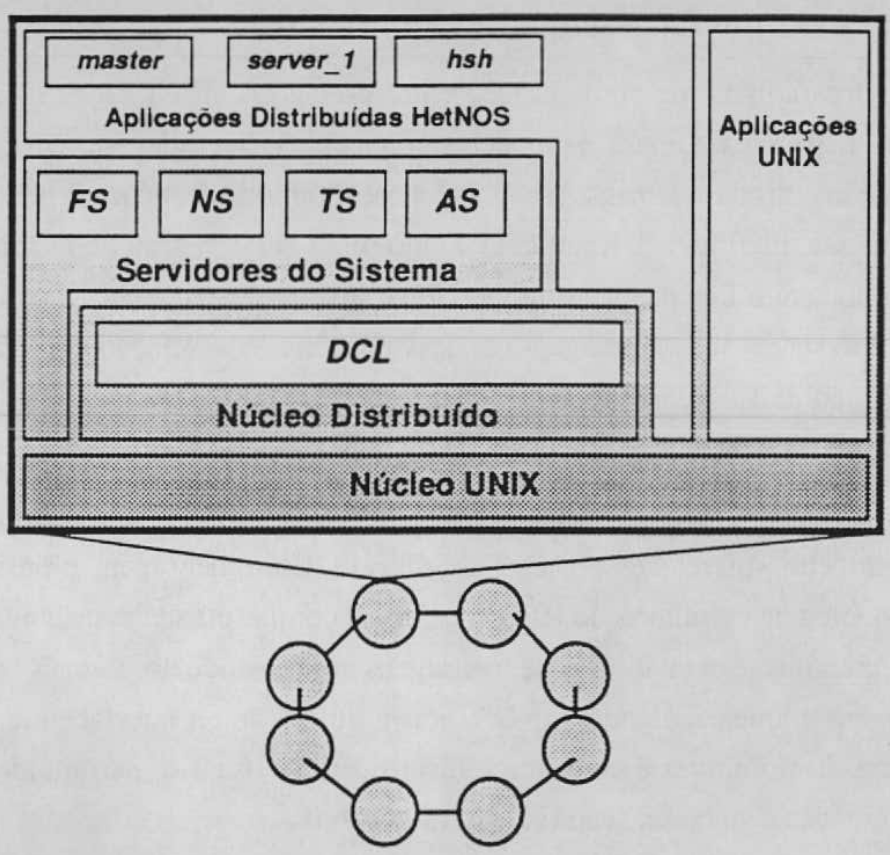

Figura 1 - Estrutura do Sistema HetNOS

A estrutura ilustrada na figura 1 representa a estrutura do sistema HetNOS em um nodo individual, bem como a topologia lógica de interligação dos elementos, um anel lógico. Uma cópia funcionalmente idêntica do sistema executa em cada nodo da rede, sendo o controle e os dados distribuídos entre cada uma das instâncias. Detalhando melhor a estrutura do sistema, as quatro camadas do sistema são: (**1**) o núcleo Unix (sistema nativo), (**2**) o Núcleo Distribuído (DCL- *Distributed Computing Layer*), (**3**) os Servidores do Sistema *(FS-File Server, NS-Name Server, TS-Type Server e AS-Authorization Server)*, (**4**) além das aplicações de usuários.

Este trabalho enfoca a **interface de programação** distribuída do ambiente HetNOS. Por esta razão, a descrição da arquitetura de software do HetNOS, incluindo sua estrutura interna de funcionamento e os argumentos que justificam as decisões de projeto adotadas, fogem ao escopo deste trabalho (vide [BAR93b]).

3. MODELO DE PROGRAMAÇÃO NO HetNOS

Neste item, a filosofia de programação no HetNOS é descrita. O conjunto de primitivas disponíveis para comunicação e gerência distribuída de processos é mostrado no item 4.

Mensagens como Paradigma de Comunicação

O paradigma de comunicação entre processos (IPC) é decisivo no projeto da aplicação distribuída. Dentre os modelos conhecidos, destacam-se, em ordem crescente de abstração: "troca de mensagens", "chamada remota de procedimento" e "memória compartilhada distribuída". Entretanto, como regra geral, o grau de abstração e sofisticação do ambiente é inversamente proporcional ao seu desempenho. Como cada uma das opções apresenta suas vantagens e desvantagens em relação às demais [BAL89, AND91], não existe consenso sobre qual seja o melhor paradigma. Tal análise foge ao escopo deste trabalho.

O HetNOS adota o mecanismo de "troca de mensagens", devido à flexibilidade e ao desempenho oferecidos por essa abordagem. Com mensagens é possível implementar os modelos de comunicação RPC e memória compartilhada distribuída. Dos três modelos de comunicação citados, o de mensagens corresponde ao de mais baixo nível. Essa desvantagem é amenizada no HetNOS, pois a utilização da interface é bastante simples. Uma gama de primitivas é colocada à disposição do usuário, permitindo diversos tipos de comunicação e sincronização (vide item PRIMITIVAS).

Ao se interligar sistemas Unix, utilizando as API's de rede hoje disponíveis, um dos maiores problemas encontrados é a necessidade de lidar com endereços de *hosts* e portas de comunicação. As dificuldades impostas pelas interfaces *sockets* ou TLI podem ser exemplificadas através dos fatores envolvidos na escolha do protocolo de comunicação, TCP ou UDP, se considerado o conjunto TCP/IP. TCP obriga o usuário a gerenciar conexões entre processos, oferecendo um fluxo confiável de bytes, enquanto UDP está orientado a mensagens, mas não é confiável. Pode ocorrer perda, adulteração ou embaralhamento na ordem das mensagens, freqüentemente obrigando o usuário a criar seu próprio mecanismo de verificação.

Outra alternativa de IPC, mais alto-nível, como chamada remota de procedimento (ex.: RPC/XDR [SUN90]), resolve satisfatoriamente grande parte das aplicações cliente-servidor, mas não soluciona adeqüadamente problemas onde o grau de paralelismo é intenso, quando há interações assíncronas entre diversos processos. Adicionalmente, há problemas como a ausência de comunicação multiponto (o **modelo** de RPC- não suas implementações- é inerentemente síncrono e ponto-a-ponto), a dificuldade em lidar com falhas e outras condições excepcionais, dificuldades na conversão de dados, passagem de ponteiros, etc.

O HetNOS apresenta a flexibilidade que RPC não fornece, ao mesmo tempo que exime o usuário da gerência associada à troca de mensagens com *sockets* ou TLI. Processos enviam e recebem mensagens através de sua identificação, que é única no sis-

tema como um todo. Ao identificar o destino de uma mensagem, um processo designa apenas o **nome** de um ou mais processos, não a sua localização. Não há estabelecimento ou encerramento de conexões, nem a necessidade de gerenciar portas de comunicação.

As primitivas do HetNOS oferecem a confiabilidade necessária (como TCP), sem no entanto implicar a perda de fronteiras entre mensagens (como UDP). Todas suas primitivas de comunicação são confiáveis, embora estejam previstas implementações de versões não-confiáveis, porém mais eficientes.

No sentido de simplificar o uso das primitivas, mensagens no HetNOS, ao invés de tipos estruturados de dados, são "pacotes de tamanho variável contendo uma seqüência de bytes quaisquer terminados por um código zero final" (uma *string*). Afora o limite máximo no tamanho da mensagem, usualmente um valor bastante razoável, a única restrição aplicável é quanto à existência de zeros no decorrer da mensagem.

Para transmitir dados numéricos, o HetNOS oferece duas possibilidades, que correspondem a duas classes de primitivas:

- funções que formatam e enviam cadeias de caracteres com representações ASCII (uso de parâmetros variáveis, à semelhança de `printf()`);
- funções que enviam vetores de bytes de forma codificada (com *byte-stuffing* para o código zero), de forma transparente para o usuário.

A simplicidade e a flexibilidade que decorrem do uso de *strings* como mensagens, nesse caso, compensam a perda no desempenho.

Gerência de processos

Outro aspecto crucial em termos de programação distribuída é a gerência de processos. Com *sockets*, TLI ou RPC, por exemplo, processos são criados através de primitivas como `rexec()`. Neste caso, a máquina destino precisa ser especificada, sendo considerado para execução o ambiente da máquina remota, incluindo o arquivo executável, de dados, *path*, etc. A chamada é extremamente inconveniente em termos de segurança, sendo usualmente necessário: **(a)** fornecer a senha no próprio código fonte e executável; ou **(b)** preparar um arquivo texto de configuração contendo a senha. Além do mais, é necessário gerenciar conexões adicionais para redirecionamento de E/S e sinais. Para que a saída de um processo apareça na console associada ao "processo pai", é necessário ler da conexão remota e escrever na saída padrão todo conteúdo lido (usualmente dedica-se processos locais para tal).

No HetNOS, processos são criados, listados, sincronizados e removidos considerando a rede de estações como uma "máquina virtual". Tais operações ocorrem de

acordo com a filosofia Unix, entretanto de forma simplificada e distribuída. Mesmo quando um processo é criado em um *host* remoto, diferente daquele em que o "pai" reside, o processo filho herda todos os atributos cabíveis do processo pai, incluindo *tty* associada, UID (identificador de usuário), GID (identificador de grupo), variáveis de ambiente (como *path*), etc.

Isto permite que comandos e ferramentas do HetNOS, bem como demais aplicações de usuário, utilizem serviços de gerência distribuída de processos. A influência na *interface de comandos* é significativa, pois um usuário pode disparar um processo remoto, acompanhar seu andamento, abortá-lo, etc. sem precisar saber em qual *host* o mesmo estava sendo executado.

A identificação de processos exerce papel fundamental na implementação de aplicações distribuídas, segundo o modelo de programação considerado. No HetNOS, processos são identificados não com valores inteiros cíclicos, mas sim com **nomes**. Nomes são rótulos, que neste contexto podem ser definidos como uma seqüência de caracteres pertencentes a um sub-conjunto do código ASCII ('A'..'Z', 'a'..'z', '0'..'9', '_' e '-'). Ao criar um novo processo, o usuário o *batiza* escolhendo-lhe um nome. Este identificador é informado através da primitiva de criação do HetNOS, que por sua vez verifica se o nome é válido (é único na rede, inicia por caracter, etc.)

As primitivas que acessam serviços do HetNOS e que necessitam identificar um outro processo fornecem como parâmetro um apontador para uma *string* com o seu nome. A identificação é global à rede e independente de localidade. Exemplificando, o processo que executa uma chamada do sistema do tipo `h_kill("proc")` não precisa saber em que máquina o processo **`proc`** está rodando. É de responsabilidade do HetNOS localizar o processo em questão e, caso existente, eliminá-lo. Todas as questões relacionadas à morte do processo devem ser tratadas transparentemente, como no caso local.

Nas situações específicas em que é necessário denominar o *host* em que um dado processo está sendo executado, o seu nome é sucedido de um "sufixo indicador de localidade", composto pelo sinal de arroba (`@`) mais o nome do *host* (p.ex., **`proc@host`** identifica o processo **`proc`** no nodo **`host`**). A princípio, apenas os módulos do sistema necessitarão de tal recurso, visto que, para processos de usuários, a identificação é independente de localidade.

O esquema de identificação é **global** e, se por um lado, permite que diferentes aplicações se comuniquem, por outro poderia provocar um problema de interferência entre identificadores de diferentes usuários. Há uma série de nomes que são fortes candidatos a identificar processos de uma aplicação distribuída, tais como **`servidor`**, **`cliente`**, **`coordenador`**, **`calculador`**, **`mestre`**, **`escravo`**, etc. Como estes nomes estariam

freqüentemente ocupados por outros usuários, seria necessário utilizar nomes estranhos, não naturais. Por esta razão, nomes referem-se apenas aos processos pertencentes ao próprio usuário. O esquema de identificação conta com um "sufixo identificador de proprietário", elemento opcional utilizado para indicar o usuário a quem pertence um processo (o *default* são os processos do próprio usuário). A sintaxe é **`nome_processo#nome_usuário`**, **#** significando "de". Portanto, para referenciar processos de demais usuários, basta adicionar o sufixo ao final do identificador.

Exemplificando, o rótulo **`pink#floyd`** refere-se ao processo **`pink`** do usuário **`floyd`**, sendo opcional o uso do sufixo **`#floyd`** quando se tratar do próprio usuário **`floyd`**. O conjunto de nomes correspondente aos módulos do HetNOS (**`DCL`**, **`FS`**, **`NS`**, **`TS`** e **`AS`**), bem como **`ANY`** e **`HetNOS`**, é reservado e não pode ser requisitado por usuários.

Usuários podem escrever servidores e oferecer seus serviços à comunidade usuária, bastando "publicar" o nome do serviço. Quem acessa o serviço precisa conhecer apenas o nome do processo servidor e do usuário responsável pelo mesmo (caso não seja o próprio HetNOS que providencie o serviço).

4. PRIMITIVAS

Este item descreve as principais primitivas da interface de programação. O tipo de dado mais utilizado como argumento para o conjunto de primitivas do HetNOS é a *string*. Para facilitar a programação, as rotinas que utilizam uma ou mais *strings* como argumento possuem uma função correspondente que aceita um conjunto variável de argumentos, sendo este definido dinamicamente segundo uma *string* de formato. Estas funções, cujo comportamento se assemelha ao da família **`printf()`**, possuem o nome original da função mais um sufixo "**`_v`**".

Comunicação entre Processos

Para a programação distribuída por mensagens são oferecidas diversas opções além das primitivas básicas de envio e recepção, *send* e *receive*. Quanto ao envio de mensagens, há primitivas síncronas, assíncronas, "descartáveis" e multiponto (todas confiáveis). No envio síncrono o processo origem é bloqueado até que a mensagem seja lida, enquanto no assíncrono o processo é liberado mesmo que o destino não esteja pronto para receber (neste caso a mensagem é armazenada em *buffer* do sistema para posterior leitura). Já a primitiva para envio "descartável" não usa *buffers*, pois caso o processo destino não esteja pronto para receber a mensagem a mesma é **descartada**. A primitiva de envio assíncrono multiponto libera o processo assim que a mensagem tiver sido entregue a todos os *hosts* destinos.

Quanto à recepção de mensagens, elas podem ser bloqueantes/não-bloqueantes ou multiponto. Recepção bloqueante ou síncrona suspende a execução do processo receptor até que uma mensagem "aceitável" seja lida. A primitiva para recepção não-bloqueante não suspende o processo, retornando mesmo que nenhuma mensagem tenha sido lida. O HetNOS não oferece o que se convencionou chamar "recepção assíncrona". Neste modo de comunicação, o pedido de recepção de mensagem continua pendente quando o mesmo não pode ser imediatamente satisfeito (quando a mensagem requisitada chega, o processo é interrompido e um *handler* acionado). Na recepção multiponto um processo (receptor) informa um conjunto de processos, selecionando-se a primeira mensagem oriunda de um deles.

Finalmente, há um par de primitivas dedicado à transmissão assíncrona de "vetores de bytes", que pode ser utilizado para transmissão de grandes conjuntos de bytes (como *bit-maps*) ou, nos casos em que o hardware do *host* origem e do destino são compatíveis, para cópias de estruturas de dados pela rede.

A tabela 1 resume o conjunto básico de primitivas de comunicação atualmente disponíveis no HetNOS.

Tabela 1 - principais primitivas de comunicação

PRIMITIVA	DESCRIÇÃO
`h_send(dest, msg)`	envia assincronamente mensagem *msg* ao processo *dest*
`h_receive(orig, msg)`	requisita o recebimento síncrono de mensagem de *orig*
`h_sync_send(dest, msg)`	envia sincronamente mensagem *msg* ao processo *dest*
`h_multi_send(lst_dst, msg)`	envia assincronamente mensagem *msg* ao conjunto de processos nominado pela lista *lst_dst*
`h_multi_receive(lst_org, msg)`	requisita o recebimento síncrono da primeira mensagem que vier de um dos processos nominados por *lst_org*
`h_send_bytes(dest, dados, tam)`	envia assincronamente um conjunto de bytes situado a partir de *dados* e com tamanho *tam*
`h_receive_bytes(orig, dados, max)`	recebe sincronamente um conjunto de bytes a partir de *dados* e com tamanho máximo *max*

A título de ilustração, um pequeno trecho de programa mostra como um conjunto de processos, dividido em *mestre* e *escravos*, poderia comunicar-se. Um processo *mestre* (figura 2) distribue serviço a seus *escravos* (figura 3), preparando e enviando assincronamente uma mensagem diferente a cada um deles. Após, espera o resultados enviados pelos *escravos*, e imprime um valor final. Conforme citado anteriormente, as funções "_v" tomam um número variável de argumentos, dependendo da descrição de

formato fornecida. O formato aceito por estas funções é basicamente o mesmo de **printf()**, mas o código "**%i**" foi acresentado às funções de recepção para que um *label* possa ser ignorado.

```
mestre() { /* prog_mestre.c */
   h_init();
   ...
   /* criação de processos escravos */
   ...
   /* distribue tarefas */
   for (i = 0; i < num_esc; i++)
      h_send_v("%s_escravo%d", meunome, i, "v1=%f v2=%f", f1*i, f2);
   /* colhe resultados */
   for (s = 0.0, i = 0; i < num_esc; i++) {
      h_receive_v("ANY", "%i: v3= %f", &f3);
      s += f3;
   }
   printf("Resultado: %f\n", s);
   h_exit(0);
}
```

Figura 2 - código processo *mestre*

```
escravo() { /* prog_escr.c */
...
   h_init();
   h_get_parent_name(nome_pai);
   h_receive_v(nome_pai, "%i: v1= %f v2= %f", &f1, &f2);
   h_send_v(nome_pai, "v3= %f", f1 + f2);
   h_exit(0);
}
```

Figura 3 - código processo *escravo*

Gerência de Processos

O modelo de programação no HetNOS envolve particionamento de uma aplicação em múltiplos processos a serem distribuídos nas máquinas de uma rede.

A criação de processos no Unix está baseada nas primitivas **fork()**, para "duplicação" de um processo em "pai" e "filho", e **exec()**, para que um processo passe a executar outro programa [BAC86]. Neste esquema, os processos são criados sempre localmente.

O HetNOS estende esse esquema do Unix de forma a permitir que usuários criem processos sobre "uma máquina virtual" composta por diversos *hosts* rodando o HetNOS. Há transparência de localidade para todas as suas primitivas de gerência de processos, pois o usuário não precisa conhecer a localização da máquina onde o processo será criado ou reside. Como sistema operacional de rede, as operações do HetNOS sobre processos são de certa forma limitadas pelo sistema operacional nativo, pois não

há controle sobre suas tabelas internas. Dessa forma, não há como migrar um processo entre duas máquinas.

A tabela 2 resume o conjunto básico de primitivas de gerência de processos atualmente disponíveis no HetNOS. Todas as funções são confiáveis e síncronas, ou seja, em caso de erro um código especial é retornado ao usuário. Cada função possui uma equivalente "`_v`".

Tabela 2 - principais primitivas de gerência de processos

PRIMITIVA	DESCRIÇÃO
`h_init()`	inicializa um determinado processo junto ao núcleo distribuído do HetNOS
`h_fork(filho)`	cria um processo local através da duplicação, e batiza o processo com o nome indicado por *filho*
`h_loc_exec(filho, comando)`	cria um novo processo local, batiza o mesmo como *filho* e carrega o executável indicado por *comando*
`h_rem_exec(filho, host, comando)`	idem `h_loc_exec()`, mas na máquina *host*
`h_exec(filho, comando)`	idem `h_loc_exec()`, mas escolhe a máquina com menor carga na rede (retorna um apontador para o nome do *host* escolhido)
`h_wait(processo)`	espera a morte do processo indicado por *processo*
`h_kill(lista_processos)`	termina um conjunto de processos (e seus filhos) de acordo com a lista *lista_processos* - diferente do `kill()` do Unix, pois ainda não há esquema de controle distribuído de sinais
`h_terminate()`	indica término do processo HetNOS, embora este possa continuar sua execução no Unix
`h_exit(código)`	idem `h_terminate()`, mas o processo também termina no Unix - com a chamada `exit(código)`

Continuando o exemplo das figuras 2 e 3, o processo *mestre* poderia criar o conjunto de *escravos* conforme o código ilustrado na figura 4.

5. AMBIENTE DE DESENVOLVIMENTO

O HetNOS fornece um esquema simplificado de segurança a seus usuários. Como qualquer sistema do tipo "*add-on*", tal esquema depende da segurança oferecida pelos sistemas operacionais nativos em questão [GEI90a]. Cada usuário precisa abrir uma sessão no HetNOS antes de acessar os seus serviços, o que só pode ser obtido se o mesmo tiver sido previamente cadastrado por um super-usuário (do HetNOS).

Através de uma console virtual (p.ex., uma janela do ambiente gráfico do sistema nativo) é executada a aplicação de *login* do HetNOS, denominada `hlogin`. Esta apli-

cação solicita a entrada do *username* e da senha do usuário, requisitando então ao HetNOS a verificação da autenticidade do usuário. Em caso positivo, uma sessão é iniciada com a apresentação de uma mensagem de boas vindas e a execução de um interpretador de comandos, o **hsh**. Este último apresenta um sinal de prontidão na tela.

```
...
/* criação dos escravos pelo mestre */
for (i = 0; i < num_esc; i++)
    if (h_exec_v("%s_escravo%d", meunome, i,
    "prog_escr %d", i)==NULL) {
        printf("não consegui criar escravo, erro %s\n", h_error);
        h_exit(1);
    }
...
```

Figura 4 - criação de processos

A partir do **hsh** o usuário entra comandos HetNOS e Unix. Para executar uma determinada aplicação HetNOS, um usuário emprega o comando interno **h**, na forma **h *nome_processo nome_executável args_aplicação***. Exemplificando, a aplicação *mestre x escravo* apresentada pode ser acionada como **h mestre prog_mestre**. O processo executado pode descobrir seu nome através de uma variável externa definida na biblioteca de rotinas do HetNOS, o mesmo valendo para o nome do processo pai. Múltiplas ativações de uma mesma aplicação não são um problema na medida que o usuário atribui diferentes nomes, tais como **mestre1**, **mestre2** e **mestre3**. De uma forma geral, se uma aplicação precisa criar um conjunto de processos filhos, então deve escolher nomes que tomem como base seu próprio nome, como **mestre1_escravo1**, **mestre1_escravo2** e **mestre1_escravo3**.

O interpretador de comandos fornece uma série de facilidades comuns a outros *shells*, tal como definição de variáveis de ambiente, *aliases*, histórico de comandos e execução em *background*. À semelhança de outros interpretadores de comandos, o **hsh** possui uma série de comandos embutidos, como por exemplo **hkill**, **hps** e **hsend** - para eliminar, listar ou enviar mensagem a processos, respectivamente.

O **hsh** permite que o usuário, via linha de comando, interaja com seus processos (através de mensagens), facilitando a depuração. Processos podem enviar mensagens de depuração ao *shell* do usuário, podendo estas serem recebidas com o comando **hreceive** em um instante qualquer. O estado dos processos na rede pode ser continuamente monitorado (e sua localização pode ser ignorada pelo usuário).

Os serviços do HetNOS só podem ser acessados de suas sessões, mas não necessariamente de uma aplicação executada a partir da linha de comandos do **hsh**. A primitiva **h_login()** abre uma sessão, enquanto **h_logout()** encerra uma sessão

(invocada pelo comando interno **hexit** do **hsh**). A partir da chamada **h_login()** bem sucedida, todas operações do HetNOS podem ser acessadas pelo processo e por seus processos filhos locais e remotos.

Uma das principais vantagens do ambiente de trabalho do HetNOS é a "máquina virtual" composta pelo conjunto de *hosts* interligados. Processos são executados em diversas máquinas, distribuídos pelo usuário ou pelo sistema. O acompanhamento de cada um dos diversos processos em execução é facilitado pelo comando **hps** e pelo redirecionamento transparente de *tty's* remotas, assim como é simples eliminar alguns dos processos envolvidos com **hkill**. Em um processamento cooperativo em larga-escala, *processos escravos* podem ser dinamicamente adicionados ou removidos via linha de comando.

6. TRABALHOS RELACIONADOS

Trabalhos relacionados podem ser examinados sob dois ângulos: linguagens ou ambientes para programação distribuída. Como exemplo de linguagem, toma-se SR *(Synchronizing Resources)*, uma das melhores opções em termos de programação distribuída. Como ambiente de programação, existem diversos exemplos cujo objetivo principal é explorar a capacidade ociosa de *workstations*, e através do processamento paralelo, obter computação de alto-desempenho. CPS [RIN93], P4 [BUT92] e PVM [GUE90b] são representantes significativos dessa classe.

A linguagem SR oferece mecanismos de alto-nível para que programadores possam implementar algoritmos paralelos. Seu principal elemento é o *resource*, que pode ser comparado aos módulos em outras linguagens. A principal vantagem do SR sobre o HetNOS é a utilização do paradigma de programação orientado a objetos. A definição da especificação de um *resource* determina os objetos que podem ser "exportados" e os objetos "importados" de outros *resources*. A facilidade de uso de mecanismos de troca de mensagens do tipo RPC e *rendezvous* pode ser considerada outra vantagem de SR sobre o HetNOS.

Por outro lado, o HetNOS oferece um ambiente para programação distribuída mais flexível e com mais recursos para controle de processos. Não é preciso definir a localização de objetos, como ocorre no SR. Outra vantagem diz respeito à interação entre aplicações, pois no HetNOS diferentes programas executáveis podem se comunicar, enquanto que no SR a interação está restrita a um único código. Um exemplo clássico é a programação de um servidor para prestar serviços à aplicações de diferentes usuários.

O HetNOS pode ser comparado também com bibliotecas de suporte à programação distribuída como P4 e PVM.

O P4 é uma biblioteca de funções e macros desenvolvida para rodar sobre máquinas paralelas. Aplicações escritas em C ou Fortran são estruturadas em processos, que se comunicam através de mensagens. Uma vantagem de P4 sobre o HetNOS é o desempenho das primitivas de comunicação, que mostrou-se bem superior nos testes realizados (vide tabela 4). Entretanto, o desempenho da criação de processos no P4 é bem menor que o HetNOS (vide tabela 3), mesmo considerando que a gerência distribuída de processos neste último é mais completa.

Uma desvantagem menor do P4 é o consumo de memória principal e secundária, devido ao tamanho dos executáveis gerados (pois é preciso incluir em cada processo uma cópia do seu sistema de comunicação). Já as aplicações do HetNOS apresentam executáveis menores, porque há um núcleo distribuído com serviços que são compartilhados por todos os processos de usuários (há apenas uma cópia). Outra desvantagem do P4 é não permitir a comunicação entre diferentes aplicações, estando restrita a comunicação aos módulos de um mesmo programa (tal como SR).

Assim como o P4, o PVM *(Parallel Virtual Machine)* é uma biblioteca para linguagens C e Fortran que fornece suporte para a programação de aplicações distribuídas. Os processos componentes da aplicação estão espalhados em uma máquina virtual (conjunto de máquinas que o PVM trata como uma única máquina). Em cada nodo da máquina virtual existe um *daemon* que controla os processos da máquina local e se comunica com os *daemons* remotos. HetNOS e PVM associam identificadores únicos globais a cada processo, embora os do PVM sejam inteiros e os do HetNOS, rótulos. No HetNOS o próprio usuário pode determinar o nome (cadeia de caracteres) de seus processos, enquanto que no PVM identificadores são inteiros gerados dinamicamente (evitando que a identificação possa ser determinada em tempo de compilação). O controle de processos é semelhante ao do HetNOS, onde uma gama de operações se encontra disponível, como por exemplo disparar processos de modo transparente de localidade (ou indicando a localidade) na máquina virtual, terminar processos, listar os processos ativos distribuídos e até mesmo adicionar e retirar *hosts* da máquina virtual. Entretanto, o PVM fornece o conceito de grupo de processos, ainda não disponível no HetNOS.

A primitiva para envio de mensagens é assíncrona; comunicação síncrona precisa ser simulada por um *send* seguido de um *receive* bloqueante. Além disso, o PVM possui uma primitiva que recebe mensagens podendo bloquear ou não o programa invocador desta primitiva. Além das primitivas de *send* e *receive* implementadas pelo PVM, o

HetNOS conta uma primitiva "descartável" que pode ser usada para eficientemente distribuir cargas de processamento entre um grupo de *tasks*.

Possivelmente a principal vantagem do PVM sobre os demais sistemas seja seu desempenho, que foi melhor tanto na gerência distribuída de processos como nos mecanismos de IPC (vide tabelas 3 e 4).

Uma vantagem importante do HetNOS sobre os três outros sistemas analisados é a herança de atributos entre processos remotos, destacando-se as variáveis de ambiente e o redirecionamento de *tty*. No PVM, p.ex., a saída padrão das *tasks* é gravada em um arquivo em um diretório de temporários, sendo necessário continuamente monitorar o conteúdo do arquivo (através de comandos como `tail`).

O desempenho e a facilidade de programação foram avaliados através da implementação de um algoritmo distribuído em SR, P4, PVM e HetNOS. Tal algoritmo, denominado "*waves*" [RAY90], possui paralelismo de *fina granulosidade* e foi originalmente criado para rodar em uma malha de *transputers*.

O problema consiste em construir uma árvore com uma raiz cobrindo completamente uma malha quadrada de processos. Nesta malha, cada elemento pode comunicar-se somente com os seus vizinhos. O algoritmo inicia com a escolha de um processo raiz, e ao término, cada processo comunica-se apenas com o seu processo pai e com os seus filhos. A figura 5 mostra um exemplo para uma malha 3x3 (9 processos).

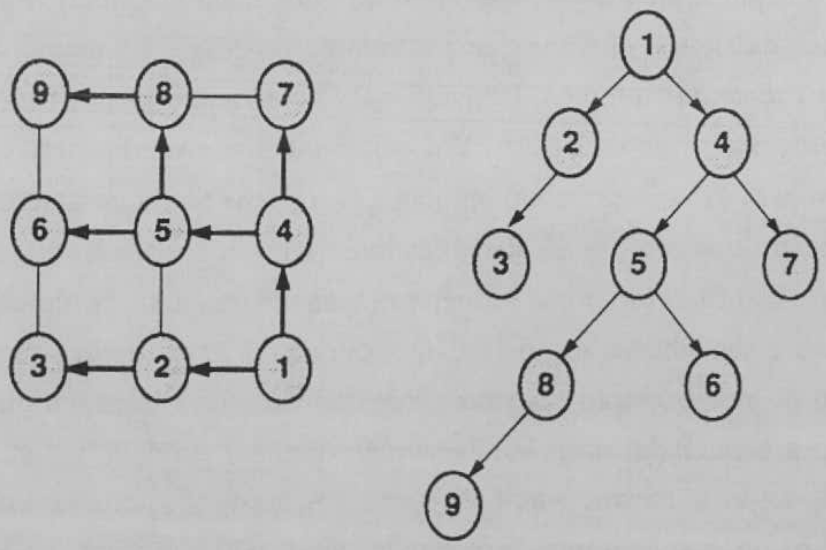

Figura 5 - algoritmo "waves"

O desempenho do algoritmo "*waves*" foi avaliado experimentalmente com uma configuração de hardware idêntica para HetNOS, SR, PVM e P4, formada por estações de trabalho SUN 4 (SPARC 1+, IPCs e SLCs) com sistema SunOS 4.1.1 interligadas por uma rede Ethernet 10 Mbits/s. Realizaram-se medições com malhas de 4, 9 e 16 elemen-

tos, ou seja, grades de 2x2, 3x3 e 4x4 elementos (nas malhas com 9 e 16 máquinas, dois e três segmentos de rede foram utilizados, respectivamente).

As primitivas foram separadas em dois grupos: comunicação e criação de processos. Quanto à eficiência da criação de processos, tomou-se o tempo necessário à criação do conjunto de processos *escravos* remotos. Os resultados colhidos são resumidos na tabela 3.

Tabela 3 - comparação no desempenho da gerência de processos

Dimensão da Malha	Tempo (em segundos) para criação dos elementos			
	PVM	HetNOS	P4	SR
4 (2x2)	0,4467	1,7533	10,3733	13,5800
9 (3x3)	0,9100	4,1566	25,0033	31,0700
16 (4x4)	1,6500	8,3133	40,0333	54,0466

Foi avaliado também o desempenho do mecanismo de comunicação entre processos. Como este é um algoritmo para processamento paralelo, de granularidade fina, a comunicação entre processos representa a parte crítica do tempo de execução total do algoritmo (ressaltando as diferenças no desempenho dos sistemas de comunicação). O tempo medido foi aquele gasto na composição de uma árvore, sendo os resultados apresentados na tabela 4.

Tabela 4 - comparação no desempenho da comunicação

Dimensão da Malha	Tempo (em segundos) para composição da árvore			
	PVM	P4	SR	HetNOS
4 (2x2)	0,1083	0,2826	0,3023	1,8993
9 (3x3)	0,2310	0,5336	0,6103	8,1152
16 (4x4)	0,4213	1,0860	0,9843	17,5967

7. CONCLUSÃO

O presente trabalho demonstra a apropriabilidade do sistema operacional de rede HetNOS como ambiente de suporte à programação distribuída de aplicações em C para o ambiente Unix. O aspecto inovativo apresentado pelo HetNOS é sua interface de alto nível, onde processos são identificados com *strings* e as primitivas e chamadas do sistema suportam operações sobre essas cadeias. Expansão de meta-caracteres pode ser inserida, de forma a tornar a interface de gerência de processos semelhante à gerência de arquivos.

Apresentou-se o modelo de programação paralela e/ou distribuída adotado pelo HetNOS, que consiste na divisão de uma aplicação distribuída entre diversos processos sequenciais executados sobre uma máquina virtual distribuída. Tais processos, que po-

dem também não pertencer à mesma aplicação, interagem entre si através de mensagens. Processos são identificados por nomes globais atribuídos por usuários, associando conteúdo semântico a processos.

Entretanto, é sensivelmente fraco o desempenho dos mecanismos IPC do atual protótipo do HetNOS. Embora *performance* nunca tenha sido o objetivo principal do sistema, a prioridade estabelecida para o futuro próximo é atingir taxas de comunicação semelhantes às do PVM. O fraco desempenho da versão atual pode ser facilmente explicado pela sua estrutura em anel lógico [BAR93b] (conforme ilustrado na figura 1). Duas alternativas estão sendo atualmente testadas: **(a)** emprego de UDP para comunicação direta entre *hosts* e **(b)** mudança para uma topologia mista, possivelmente hierárquica. Adicionalmente, estuda-se o uso de memória compartilhada, reimplementando-se a troca local de mensagens (naturalmente, sem alterar a interface).

Outro ponto de trabalho futuro é o desenvolvimento de um ambiente gráfico para o HetNOS. Primeiramente, o ambiente será utilizado para controle de processos na máquina virtual distribuída, com uma interface orientada a ícones e janelas. Em um segundo momento, o ambiente será expandido de forma a suportar a especificação gráfica de programas paralelos e distribuídos.

Pontos de pesquisa futura incluem também tolerância a falhas, depuração de software distribuído, balanceamento de carga (controlando o funcionamento de *tasks*) e mecanismos de controle de processos, para incluir manipulação de sinais e grupos de processos no HetNOS.

BIBLIOGRAFIA

[AND 82] ANDREWS, G.R. The Distributed Programming Language SR-Mechanisms, Design and Implementation. **Software-Practice & Experience**, Chichester, v.12, p.719-753, 1982.

[AND 91] ANDREWS, G. Paradigms for Process Interaction in Distributed Programs. **ACM Computing Surveys**, New York, v.23, n.1, Mar. 1991.

[AND 93] ANDREWS, G.; OLSSON, R.A. **The SR Programming Language: concurrency in practice.** Benjamin/Cummings, Redwood, 1993. 344p.

[BAC 86] BACH, M. **The Design of the Unix Operating System.** Prentice-Hall, Englewood Cliffs, 1986. 471p.

[BAL 89] BAL, H.; STEINER, J.; TANENBAUM, A. Programming Languages for Distributed Computing Systems. ACM Computing Surveys, New York, v.21, n.3, Sep. 1989.

[BAR 93a] BARCELLOS, A.M.P. O Sistema Operacional de Rede Heterogêneo HetNOS. CPGCC-UFRGS: Dissertação de Mestrado, abr. 1993. 213p.

[BAR 93b] BARCELLOS, A.M.P. Projeto do Sistema Operacional de Rede Heterogêneo HetNOS. In.: Seminário Integrado de Software e Hardware, 20 (XX SEMISH). **Anais.** Florianópolis, SC. set. 1993. Rio de Janeiro, SBC, 1993. p.718-732

[BUT 92] BUTLER, R.; LUSK, E. **User's Guide to the P4 Programming System.** Argonne National Laboratory, Oct. 1992. 37p.

[COR 91] CORBIN, J. The Art of Distributed Applications. Springer-Verlag, 1991.

[EMI 92] EMIR DA SILVA, M. Um Servidor de Nomes para um Sistema Operacional Distribuído Heterogêneo. Porto Alegre: II-UFRGS: **Trabalho de Conclusão**, jul. 1992. 117p.

[GEI 90a] GEIHS, K.; HOLLBERG, U. Retrospective on DACNOS. Communications of the ACM, New York, v.33, n.4, p.439-448. Apr. 1990.

[GEI 90b] GEIST, A. et alli. **PVM 3 User's Guide and Reference Manual.** Oak Ridge National Laboratory, May 1993. 108p.

[GER 93] GERMANO, A. Um Servidor de Autorização para o HetNOS. Porto Alegre: II-UFRGS: **Trabalho de Conclusão**, jul. 1993.

[GUE 91] GEYER, C.F.R. Une Contribution a L'Etude du Parallelisme OU en Prolog sur des Machines sans Mémoire Commune. Université Joseph Fourier, Grenoble, 1991.

[LEF 89] LEFFLER, S.J.; McKUSICK, M.K.; KARELS, M.J.; QUARTERMAN, J.S. **The Design and Implementation of the 4.3BSD UNIX Operating System.** Addison-Wesley, Reading, 1989. 471p.

[PER 93] PEROZZO, C.R. Gerência de Recursos e Controle de Acesso Distribuídos para o HetNOS. Porto Alegre: II-UFRGS: **Trabalho de Conclusão**, dez. 1993.

[RAY90] RAYNAL, M.; HELARY, J.M. **Synchronization and Control of Distributed Systems and Programs.** John Wiley & Sons, Chichester, 1990. 124p.

[RIN 93] RINALDO, F.J.; FAUSEY, M.R. Event Reconstruction in High-Energy Physics. **Computer**, Los Alamitos, v.26, n.6, Jun. 93.

[SCH 92] SCHRAMM, J.F.L. Primitivas de Comunicação Multi-Ponto para um Sistema Operacional Distribuído. Porto Alegre: II-UFRGS: **Trabalho de Conclusão**, dez. 1992.

[STE 90] STEVENS, W.R. **UNIX Network Programming.** Prentice-Hall, Englewood Cliffs, 1990. 772p.

[SUN 90] SUN Microsystems, Inc. **Network Programming Manual**. Revision A, 1990. 297p.

[TAN 85] TANENBAUM, A.S.; van RENESSE, R. Distributed Operating Systems. **Computing Surveys**, New York, v.17, n.4, p.419-470, Dec. 1985.

[ZAR 93] ZARDO, M. Um Servidor de Tipos para o HetNOS. Porto Alegre: II-UFRGS: **Trabalho de Conclusão**, jul. 1993.