AMU - Ambiente de Multiprocessamento para Unix

Elgio Schlemer[1]
elgios@ulbra.tche.br

Roland Teodorowitsch[2] (orientador)
roland@ulbra.tche.br

Universidade Luterana do Brasil (ULBRA) - Campus Canoas
Centro de Ciências Naturais e Exatas - Departamento de Informática
Rua Miguel Tostes, 101 - Bairro São Luís - CEP 92.420-280 - Canoas/RS - Brasil

Resumo

Este trabalho descreve a implementação de um ambiente de programação que transforma um conjunto de máquinas Unix ligadas através de uma rede em um multiprocessador virtual, de forma semelhante ao que é feito pela biblioteca PVM. O ambiente, implementado sobre o sistema operacional Linux, é composto de dois módulos: um processo chamado *amd* e uma biblioteca chamada *am_lib*. O processo *amd* será executado em cada máquina da rede e responderá a todos os pedidos de comunicação destinados à máquina. Aplicações que desejarem utilizar os serviços do processo *amd* deverão utilizar as funções da biblioteca *am_lib*. Tanto o processo de comunicação, quanto a biblioteca, foram implementados em C, usando chamadas de RPC e chamadas de sistema básicas do Unix, o que garante a sua portabilidade. Pela sua simplicidade e tamanho relativamente reduzido, o Ambiente de Multiprocessamento para Unix (AMU) torna-se uma ferramenta bastante didática.

Abstract

This work describes the implementation of a programming environment that turns a group of Unix machines interconnected by a network into a virtual multiprocessor, like the PVM library does. The environment, implemented on the top of the Linux operating system, is composed of two modules: a process called *amd* and a library called *am_lib*. The *amd* process will execute on each machine of the network and will answer all communications requests for that machine. Applications that would use the services of the *amd* process must call the functions of the *am_lib* library. The communication process and the library were implemented in C, using RPC calls and basic Unix system calls. This ensures the portability of the environment. Due to its simplicity and relatively small size, the Unix Multiprocessing Environment is a very didactic tool.

1 Introdução

Sempre foi uma das grandes preocupações de técnicos em computação executar o maior número possível de tarefas no menor período de tempo e custo necessários. Muitas descobertas

[1] Elgio Schlemer foi aluno da Ulbra e atualmente é professor nas áreas de Arquitetura e Sistemas Operacionais
[2] Roland Teodorowitsch é professor da Ulbra na área de Sistemas Operacionais

contribuíram de forma significativa para isso, tais como processadores mais rápidos, hierarquias de memórias, discos maiores e mais eficientes e redes cada vez mais rápidas, possibilitando a troca de mensagens entre as máquinas de forma eficiente [2]. Este fato, aliado ao baixo custo dos computadores pessoais e das redes para interligá-los, possibilitou o surgimento dos Sistemas Distribuídos.

Dividir uma aplicação em várias tarefas e designar uma máquina diferente para executar cada uma delas passou a ser viável e compensador. Surgiram, ao longo do tempo, diversos programas para facilitar este tipo de programação. Um exemplo é a biblioteca PVM (*Parallel Virtual Machine*) [1], que oferece um grande conjunto de funções para facilitar, principalmente, a comunicação entre processos executando em diferentes máquinas.

Considerando o crescente emprego deste tipo de programação, foi implementado o "Ambiente de Multiprocessamento para Unix" (AMU). Este ambiente permite o desenvolvimento de aplicações paralelas sobre uma rede Unix como se houvesse um multiprocessador disponível. Suas funções básicas são: iniciar a execução de processos em outras máquinas da rede e controlar estes processos, enviando e recebendo mensagens dos mesmos.

O AMU foi implementado em C sobre o sistema operacional Linux [4], compatível com o Unix. São utilizadas chamadas de RPC (*Remote Procedure Call*) [5] e chamadas básicas de sistema do Unix [6], o que garante a sua portabilidade. Pela sua simplicidade e tamanho relativamente reduzido, o AMU torna-se ainda uma ferramenta bastante didática, cobrindo todos os detalhes da implementação de primitivas de troca de mensagens em sistemas operacionais modernos. Seu objetivo não é substituir, nem mesmo competir, com bibliotecas de troca de mensagens consagradas, como a PVM e a MPI [3], mas apresentar uma forma simples de implementar a troca de mensagens, explorando os recursos disponíveis em sistemas operacionais compatíveis com Unix.

A seguir, algumas características do AMU serão apresentadas com mais detalhes. Na seção 2, será apresentada uma visão geral do AMU. Na seção 3, serão apresentadas as funções oferecidas pela biblioteca. Por fim, serão apresentadas algumas conclusões.

2 O Ambiente

Na visão de seu usuário, o AMU é uma ferramenta com a qual é possível desenvolver uma aplicação paralela sobre uma rede de máquinas rodando Unix. Para tanto, o AMU disponibiliza aos seus usuários dois componentes básicos: um processo chamado *amd* e uma biblioteca chamada *am_lib*. A figura 1 mostra como estes componentes interagem transformando um conjunto de processos, rodando em diferentes máquinas da rede, em uma aplicação paralela.

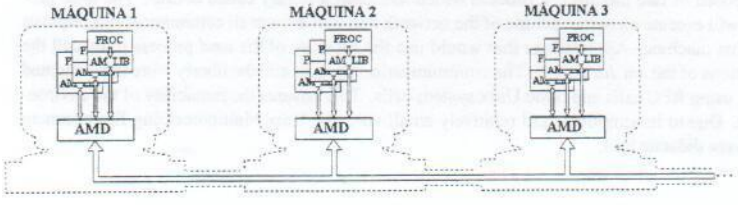

Figura 1: O Ambiente de Multiprocessamento para Unix

O processo *amd* é um processo residente (*daemon*) executado automaticamente em cada máquina da rede que faz parte do ambiente. Este processo é responsável por qualquer comunicação executada entre os processos da máquina. Ao ser executado em uma máquina, ele registra-se na máquina como um serviço de RPC, ficando responsável pelo atendimento de chamadas.

Uma chamada RPC funciona de forma semelhante a uma chamada de processo, podendo conter parâmetros e retornando uma resposta. No caso do AMU, toda vez que algum serviço for requisitado, uma cadeia de caracteres é construída como parâmetro pela função que atende aquele serviço, sendo que o primeiro caracter determina qual o tipo de serviço e os caracteres restantes são os parâmetros necessários à execução deste serviço. Como exemplo, podemos citar o serviço de envio de mensagens. A aplicação chama a função *am_send()*, fornecendo dois parâmetros: o identificador do processo para o qual se deseja enviar a mensagem e a mensagem que será enviada. Internamente, a função *am_send()* monta uma cadeia de caracteres com a seguinte composição:

$$2\#myid\#idproc\#msg \quad (1)$$

Onde: "2" indica ao processo servidor da máquina destino que trata-se do serviço "send"; *myid* será substituído pelo identificador do processo atual, para que o receptor saiba de quem veio a mensagem; *idproc* é o identificador do processo que recebe a mensagem (aquele que o usuário passou como parâmetro na chamada de função) e *msg* é a mensagem que o usuário deseja enviar, também passado como parâmetro na chamada de função. Cada parâmetro do serviço é separado pelo caracter "#".

Quando o parâmetro chega ao processo servidor pela chamada RPC, este testa o primeiro caracter para selecionar o serviço adequado e depois retira as informações necessárias. Ao terminar o serviço, uma cadeia de caracteres de retorno é construída como resposta à invocação. No caso do *am_send()*, esta resposta será apenas uma condição de sucesso ou código de erro, em outras funções, como o *am_receice()*, a resposta será a mensagem recebida e informações do processo que a enviou.

3 As Funções da Biblioteca do AMU

Aplicações que desejarem utilizar os serviços do processo *amd* deverão utilizar as funções da biblioteca *am_lib*. Esta biblioteca, que deverá ser incluída na aplicação, contém uma série de rotinas que o usuário utiliza para inicializar o ambiente, cadastrar uma máquina no sistema, executar um processo em uma máquina, remover uma máquina do sistema, encerrar a execução de um processo, receber mensagens, enviar mensagens, entre outras. Onze serviços básicos foram implementados e estão disponíveis na biblioteca.

Dois são para iniciar e encerrar a máquina virtual, *am_init()*, e *am_exit()*. Para troca de mensagens entre os processos do ambiente, implementou-se as funções *am_send()* e *am_receive()*, responsáveis, respectivamente, por enviar e receber mensagens. As mensagens são enviadas como cadeia de caracteres. Se o usuário deseja enviar inteiros ou outro tipo de dado, deverá "empacotá-los" em uma cadeia de caracteres e desempacotá-los em uma cadeia de caracteres. O recebimento de mensagens é bloqueante. Se a mensagem não chegou ou mesmo nunca chegará, o processo que executou a chamada ficará bloqueado.

Para manipulação do sistema, o AMU oferece os serviços *am_addhost()*, para inserir novas máquinas no ambiente, *am_startproc()*, para iniciar processos de forma remota, *am_deletehost()*, para remover uma máquina do ambiente e *am_killproc()* para matar um processo remoto.

Alguns serviços para informação são oferecidos, como: *am_hostname()* para identificar uma máquina pelo nome através de seu identificador, *am_myhost()* para obter o identificador da máquina atual e *am_mytid()* para obter o identificador do processo atual.

Todas as máquinas e processos do sistema são identificados por um identificador único. No caso de processos, o identificador é gerado a partir do identificador da máquina no qual ele está rodando e do seu numero de identificacao de processo (PID).

Sempre que houver comunicação entre processos, o identificador do processo destino deve ser fornecido. Como este identificador é gerado levando-se em conta o identificador da máquina onde o processo está hospedado, a função saberá para qual máquina deverá enviar a solicitação

do serviço sem precisar consultar servidores de nomes, frequentemente empregados em grandes bibliotecas de comunicação.

4 Conclusão

A gama de recursos da biblioteca do AMU não é, obviamente, tão ampla quanto as outras bibliotecas consagradas. Mesmo assim diversas aplicações clássicas podem ser implementadas com relativa facilidade. Seu principal objetivo foi criar um ambiente relativamente simples e pequeno, usando chamadas básicas de sistema do Unix, em que diversos conceitos relacionados com sistemas distribuídos e programação paralela possam ser praticados.

Desta forma foi criada uma estrutura básica em cima da qual é possível desenvolver novos projetos que explorem conceitos relacionados à execução distribuída de processos e à troca de mensagens entre eles.

Este trabalho, inicialmente desenvolvido pelo autor como Trabalho de Conclusão de Curso, enquanto aluno da Universidade Luterana do Brasil, teve como motivação incentivar o uso deste tipo de programação entre os alunos de informática, bem como incentivar o uso do Linux como plataforma de desenvolvimento. O AMU constituiu um exemplo prático de uso das chamadas de sistema disponíveis nos sistemas operacionais da família Unix para comunicação entre processos executando na mesma máquina (IPC - *InterProcess Communication*) ou em máquinas diferentes (RPC). E como tal, vem sendo utilizado na disciplina de Projeto de Sistemas Operacionais do curso de Bacharelado em Informática da Ulbra.

Além dos alunos usarem a biblioteca para desenvolver programas distribuídos e explorar a programação por troca de mensagens, eles também podem explorar o AMU de várias outras formas, como, por exemplo, construir pequenos serviços de balanceamento de carga, rotinas mais sofisticadas de troca de mensagens que permitam enviar outros tipos de dados além de cadeias de caracteres, uso de um padrão para troca de parâmetros entre diferentes arquiteturas, etc.

O AMU pode, ainda, servir como base para outros trabalhos, como, por exemplo, a implementação de processos leves (*threads*), o que permitiria que a um rocesso remoto atendesse várias requisições simultaneamente. Quando o processo servidor, por exemplo, recebesse uma requisição, rapidamente um processo leve é criado para atendê-la, liberando o servidor para receber novas requisições.

Referências

[1] GEIST, Al et al. **PVM: Parallel Virtual Machine**. Cambridge, Massashussets: MIT Press, 1994.

[2] KITAJIMA, João Paulo F. W. **Programação Paralela Utilizando Mensagens**. Porto Alegre: Instituto de Informática da UFRGS, 1995.

[3] MPI FORUM. **The MPI Message Passing Interface Standard**. Knoxville: University of Tennessee, 1994.

[4] PURCELL, John; ROBSINSON, Amanda. **Linux: The Complete Reference**. Linux Systems Labs, 1996.

[5] SUN. **Network Programming Guide**. Sun, 1990.

[6] VAHALIA, Uresh. **UNIX Internals: The New Frontiers**. New Jersey: Prentice Hall, 1996.