OMNI
Sistema de Suporte a Aplicações Distribuídas*

Rogério Drummond[†] Cassius Di Cianni[‡]
drummond@dcc.unicamp.br cassius@dcc.unicamp.br

Projeto A_HAND
Departamento de Ciência da Computação
Universidade Estadual de Campinas
Caixa Postal 6065, CEP 13081-970
Campinas, SP, Brasil

Setembro de 1992

Sumário

O sistema OMNI oferece facilidades para a criação e comunicação entre processos distribuídos numa rede heterogênea de computadores tendo UNIX como sistema operacional. Vários conceitos presentes no UNIX (como, por exemplo, envio de sinais) foram estendidos ao seu equivalente distribuído. Processos comunicam-se através de portas que podem ser com ou sem conexão. Objetos do sistema (como portas, por exemplo) podem ser referenciados através de nomes simbólicos, oferecendo total transparência quanto à sua localização. O sistema será utilizado para dar suporte a três linguagens de programação distribuída em desenvolvimento no Projeto A_HAND, a saber: Cm, *LegoShell* e CO^2.

Abstract

The OMNI system provides facilities for creation and communication of distributed processes in an heterogeneous network of computers running UNIX. Several concepts present in the UNIX system (such as the sending of signals, for example) have been extended to their distributed equivalent. Processes communicate via ports, which can be connection or connectionless. System objects (such as ports, for example) may be referenced by symbolic names, thus providing total transparency with regard to their actual location. The system will be used to support three distributed programming languages currently under development at the A_HAND Project: Cm, *LegoShell* and CO^2.

*Este trabalho foi parcialmente financiado pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) processo no 501680/91-8 e pela Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) processo no 91/1823-0

[†]PhD (*Cornell University*, 1986) Áreas de interesse: ambientes de desenvolvimento de software, sistemas distribuídos, *groupware*.

[‡]BSc em Ciência da Computação (Unicamp, 1990); mestrando em Ciência da Computação (DCC-Unicamp). Áreas de interesse: sistemas distribuídos, *groupware*, engenharia de software.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

1 Introdução

O sistema **OMNI** encontra-se em desenvolvimento dentro do Projeto A_HAND[1] [11, 12], no Departamento de Ciência da Computação da UNICAMP com o objetivo de facilitar a criação e a comunicação entre processos distribuídos numa rede heterogênea de computadores e estações de trabalho, onde o sistema operacional é alguma variante do UNIX. A rede sobre a qual **OMNI** opera é a Internet [6], o que permite que (em princípio) processos em qualquer parte do globo possam comunicar-se através dele.

O sistema constitui-se basicamente de uma biblioteca de funções escritas na linguagem C [21] e de um conjunto de programas (os *daemons* **OMNI**) que executam em cada máquina da rede. Um pequeno conjunto de comandos a nível de *shell* (como `om_ps`, similar ao `ps`, por exemplo) é também oferecido.

A motivação inicial para o desenvolvimento do **OMNI** foi a de oferecer um mecanismo comum de suporte à programação distribuída para três linguagens atualmente em desenvolvimento no Projeto A_HAND:

Cm [9, 15, 16, 31], uma extensão da linguagem C orientada a objetos, com verificação forte de tipos, dotada de polimorfismo paramétrico e amplo suporte a programação modular. Uma versão do compilador Cm já se encontra plenamente operacional. A linguagem está sendo estendida para oferecer também, entre outras facilidades, suporte à programação distribuída, a fim de permitir que programas Cm possam trocar informações através de portas de comunicação.

LegoShell [8], uma linguagem gráfica de configuração de computações distribuídas, que permite interligar portas de comunicação de programas Cm ou mesmo de programas executáveis quaisquer (os descritores de arquivo padrão de uma programa qualquer podem ser encarados como portas de comunicação).

CO² [13, 14], uma linguagem de comandos (*shell*) voltada à prototipagem rápida de programas distribuídos. CO² engloba toda a funcionalidade da *LegoShell* em uma linguagem textual, fornecendo ainda muitos outros recursos que não podem ser satisfatoriamente oferecidos por uma linguagem gráfica, como comandos condicionais ou de repetição, por exemplo.

Apesar de ter sido inicialmente concebido para dar suporte às três linguagens acima, o **OMNI** é uma ferramenta geral, podendo ser utilizado de forma completamente independente dessas linguagens. Na verdade, vários outros subprojetos do A_HAND deverão utilizá-lo, como por exemplo o sistema de teleconferência Phone+ [32], e o SISTRAC [5] (Sistema de Suporte a Trabalho Cooperativo).

Os principais objetivos que o sistema pretende alcançar são:

- Oferecer facilidades para a criação de processos distribuídos, bem como troca de sinais entre eles.
- Possibilitar a troca de mensagens entre processos através de portas de comunicação. Dois tipos de porta serão suportados: conectáveis e não conectáveis.
- Permitir que objetos do sistema (como processos ou portas de comunicação) possam ser referenciados através de nomes simbólicos, que são cadeias de caracteres atribuídas a eles pelo usuário.
- Transparência total quanto à localização dos objetos manipulados pelo sistema.
- Nível satisfatório de segurança.
- Portabilidade para qualquer sistema UNIX disponível comercialmente.

Em suma, o **OMNI** procura estender os conceitos presentes no sistema UNIX para o seu equivalente distribuído, a fim de dar a ilusão ao usuário que ele está programando sobre um sistema operacional realmente distribuído, apesar disso não ser verdade. Sendo assim, muitas das primitivas **OMNI** tem nomes (e funcionalidade) iguais aos de chamadas ao sistema operacional acrescidas do prefixo "`om_`", como `om_kill`, por exemplo.

As funções do **OMNI** são bastante básicas e gerais. Elas tem um grande número de parametros que aumentam a complexidade de sua utilização. Antecipamos a elaboração de conjuntos de macros e bibliotecas de funções para uso específico em certos nichos de aplicação que simplificariam o uso do sistema.

No Cm, por exemplo, temos portas de dados e portas de serviço. Estas portas estão num nível de abstração acima do **OMNI**. O sistema de execução do Cm incluirá uma biblioteca de funções que, utilizando o **OMNI**, oferece facilidades mais próximas das abstrações do Cm [20].

Neste artigo mostraremos as potencialidades do **OMNI**. Sua facilidade de uso ficará a cargo das macros e funções adicionais que podem ser inferidas pelo leitor.

[1] Ambiente de desenvolvimento de *software* baseado em Hierarquias de Abstração em Níveis Diferenciados.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

2 Sistemas Existentes

Muito já foi feito na área de sistemas distribuídos. A maioria dos esforços nesse campo podem ser classificados em duas categorias: aqueles relacionados ao desenvolvimento de ***linguagens e ambientes de programação distribuída*** e os voltados à criação de ***sistemas operacionais distribuídos***.

Os aspectos relacionados a linguagens de programação distribuída foram amplamente analisados pelos autores das linguagens às quais o OMNI deverá dar suporte. Linguagens como CSP [18] (*Communicating Sequential Processes*), DP [17] (*Distributed Processes*) e ambientes de programação como MP [23, 22] (que inclui a linguagem Darwin [24]) foram estudados e chegou-se à conclusão que a melhor plataforma para o desenvolvimento das linguagens do A_HAND seria algo similar a algum dos vários sistemas operacionais distribuídos existentes[2]. Entretanto, a restrição de atrelar as linguagens a um sistema operacional pouco difundido ou ainda experimental era forte demais (além do fato de ser difícil obter acesso a algum deles). Sendo assim, a alternativa mais lógica foi criar uma ***camada*** ao redor do UNIX que o fizesse parecer um sistema distribuído. Esta é a idéia básica por trás do OMNI.

Existem atualmente várias propostas de sistemas operacionais distribuídos, como por exemplo Chorus [2, 27], Mach [1, 19], V [3, 4] ou Sprite [7, 25], para citar apenas alguns dos mais representativos. Esses sistemas tem em comum a maioria dos seguintes objetivos:

1. Prover mecanismos de criação de processos distribuídos.
2. Oferecer métodos adequados de comunicação entre processos.
3. Implementar um sistema de arquivos distribuído.
4. Prover transparencia quanto a localização dos objetos manipulados pelo sistema.
5. Fornecer uma maneira uniforme de referenciar os objetos do sistema.
6. Possibilitar que processos distribuídos compartilhem memória virtual.
7. Permitir a migração de processos de uma máquina a outra.
8. Oferecer um nível satisfatório de segurança.

Apesar de não ser essencial para sistemas distribuídos, é também desejável que seja oferecido:

9. Capacidade de criação de "processos leves"[3], que são processos cuja criação é muito mais barata (em termos de tempo e espaço) e que compartilham o mesmo espaço de endereçamento do "processo pesado".

O sistema OMNI procura oferecer a maior quantidade possível dos recursos acima enumerados, a saber, 1, 2, 4, 5 e 8. É impossível para o OMNI prover o compartilhamento de memória virtual, pois para isso seria necessário que ele fizesse parte do núcleo do sistema operacional. Quanto à migração de processos, OMNI não se propõe a implementá-la, mas caso seja viável desenvolver um mecanismo de migração que funcione sobre o UNIX (mesmo que apenas para certa classe de processos "bem comportados"), não existe nada no OMNI que impessa a incorporação de tal mecanismo. A criação de processos leves é possível através do uso da biblioteca de ***Lightweight Processes*** [29] ou dos novos recursos oferecidos por ***Multithreads*** [26]. Quanto à implementação de um sistema de arquivos distribuído, isso já é feito pelo NFS [28] (*Network File System*).

As facilidades oferecidas pelo OMNI serão agora explicadas através da análise dos tres principais módulos constituintes do sistema: o *Servidor de Nomes*, o *Gerenciador de Processos* e as *Portas de Comunicação*.

3 O Servidor de Nomes

Todos os objetos no sistema OMNI possuem uma identificação OMNI (a qual chamaremos de agora em diante de OMNIid), que é uma estrutura de dados capaz de identificar univocamente o objeto dentro da rede. A OMNIid é única no tempo e no espaço, ou seja, nunca dois objetos distintos podem possuir OMNIids iguais, mesmo que

[2] Para maiores detalhes sobre tal estudo, consulte as referências fornecidas sobre as linguagens.

[3] *Lightweight processes*, às vezes também chamados de *threads*.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

eles nunca cheguem a coexistir simultaneamente. Esta OMNIid é utilizada sempre que se deseja efetuar uma operação sobre o objeto.

O Servidor de Nomes é responsável por associar objetos a nomes simbólicos descritos por cadeias de caracteres. No OMNI, as informações sobre um objeto cadastrado no Servidor possibilitam a determinação da OMNIid do objeto. Esta OMNIid pode ser recuperada fornecendo-se o seu nome simb'lico. Estes serviços são implementados de forma totalmente distribuída. Cada máquina possui um *daemon* que armazena os nomes cadastrados por processos locais a ela. O Servidor de Nomes oferece os seguintes serviços:

- `ns_putName(name, context, type, info, infoSize, attributesAndPermissions, cachePol, handle)`
- `ns_getInfo(name, context, type, searchScope, timeOut, cacheUse, maxInfoSize, info)`
- `ns_getAllInfo(name, context, type, searchScope, timeOut, cacheUse, maxInfoSize, maxArraySize, infoArray)`
- `ns_removeName(handle)`
- `ns_removeProcNames(context, site, unixProcId)`
- `ns_removeUsrNames(context, site, usrId)`
- `ns_purgeName(name, context, site)`
- `ns_purgeCache(context, scope, id)`
- `om_getId(name, OMNIid, timeOut)` (implementada pelas camadas superiores)

É possível limitar o escopo da busca pelo nome efetuada pelo Servidor (*searchScope*) a apenas uma máquina específica, ao invés da rede toda. É também possível especificar um intervalo de tempo (*timeout*) durante o qual um processo está disposto esperar pelo cadastramento de um nome pelo qual ele procurou, mas que ainda não foi registrado. Nesse caso, o processo fica bloqueado até que o nome seja cadastrado ou o tempo expire. Isto é útil para sincronizar a utilização e a criação de portas entre processos.

O Objetivo principal do Servidor de Nomes é oferecer independência entre os módulos de um programa distribuído. No desenvolvimento de um projeto grande, os serviços e objetos distribuídos podem ter nomes a eles atribuídos durante a fase de especificação. O desenvolvimento das partes desse projeto podem ser realizadas de forma independente por um grupo de programadores. A ligação (*binding*) entre os objetos pode ser realizada antes da sua execução de forma simbólica, sem a necessidade de se saber a identificação do objeto no sistema.

3.1 Atributos de um Nome Simbólico

Todo nome cadastrado junto ao Servidor possui dois atributos: *visibilidade* e *unicidade*:

Visibilidade pode assumir dois valores: *local* ou *global*. Um nome local só pode ser referenciado por processos residentes na mesma máquina onde se encontra o objeto possuidor do nome. Um nome global pode ser referenciado de qualquer máquina da rede.

Unicidade pode assumir três valores: *único local*, *único global* ou *não necessariamente único*. Se um nome é único local, o sistema garante que, na máquina onde reside o objeto possuidor do nome, não existe nenhum outro objeto com o mesmo nome. Se ele é único global, não existe nenhum outro objeto em toda a rede que possua aquele mesmo nome. Se ele é não necessariamente único, podem haver outros objetos com nome igual ao dele.

3.2 Contextos e Tipos de Nomes

Muitas vezes deseja-se que um mesmo nome tenha significados diferentes em situações diferentes. Pode-se, por exemplo, querer que uma impressora tenha o mesmo nome que uma estação de trabalho. A noção de *contexto* permite a definição de espaços de nomes independentes. Dois nomes iguais em contextos diferentes são únicos em cada um de seus contextos.

Todo objeto cadastrado no Servidor tem um tipo associado. O tipo é interpretado pelo usuário do Servidor (em geral, as camadas superiores do OMNI) e orienta como devem ser interpretadas as informações associadas ao objeto.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

3.3 Permissões de Acesso

No sistema OMNI, a OMNIid de um objeto geralmente é encarada como uma *capacidade* (*capability*) para o objeto, ou seja, o simples conhecimento da OMNIid de um certo objeto por parte de um processo capacita tal processo a manipular o objeto. Por exemplo, se um processo conhece a OMNIid de uma porta de comunicação, ele está automaticamente capacitado a enviar mensagens a essa porta. Uma vez que as OMNIids dos objetos são geralmente associadas a um nome simbólico e cadastradas junto ao Servidor, este deve restringir o acesso a elas, do contrário qualquer processo que soubesse o nome simbólico de um objeto seria automaticamente capaz de manipulá-lo.

A fim de prover um mecanismo de controle de acesso, o Servidor associa a todo nome a identificação do usuário e do grupo ao qual o usuário pertencia no instante do cadastramento. A partir daí, sempre que um processo fizer uma consulta, ele será classificado em um (e somente um) dos quatro grupos abaixo, de acordo com o usuário e o grupo ao qual pertence e da máquina onde reside:

Dono Pertence ao usuário que cadastrou o nome.

Grupo Pertence ao grupo do usuário que cadastrou o nome.

Domínio Reside numa máquina situada no mesmo domínio Internet da máquina onde o nome foi cadastrado.

Outros Qualquer outro processo.

4 Gerenciamento de Processos

Processos no sistema OMNI procuram estender conceitos presentes no UNIX para ambientes distribuídos. O módulo de Gerenciamento de Processos oferece serviços para a criação de processos, envio e recebimento de sinais:

- `om_forkExec(forkExecStructure, OMNIids)`
- `om_createProc(path, argv, envp, site, OMNIprocId [, parametros para redirecao], NULL)`
- `om_wait(OMNIprocId, statusp)`
- `om_signal(sig, func)`
- `om_kill(OMNIprocId, sig)`
- `om_killpg(OMNIpgrpId, sig)`
- `om_getpid(OMNIprocId)`
- `om_getppid(OMNIpprocId)`
- `om_getpgrp(OMNIpgrpId)`
- `om_setpgrp(OMNIpgrpId)`
- `om_ptrace(request, OMNIprocId, addr, data [, addr2])`

4.1 Criação de Processos

Um processo no sistema OMNI é criado através da primitiva `om_forkExec` ou `om_createProc`. Ambas executam `fork`, todas as redireções necessárias e um `exec` na máquina especificada, retornando a OMNIid do processo recém criado e de quaisquer portas para as quais algum descritor de arquivo tenha sido redirecionado (se for o caso).

Nestas duas funções é necessário especificar o nome da máquina que se deseja atingir. Entretanto, em quase todas as demais utiliza-se uma identificação OMNIou um nome simbólico, garantindo a transparência de localidade. Sugere-se que processos locais sejam também criados através delas e referenciados pela sua identificação OMNI, para obter total transparência.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

4.2 Envio de Sinais

Sinais em UNIX são trocados entre processos pela chamada ao sistema operacional `kill` [30]. O envio de um sinal é similar a uma interrupção de *hardware*. A primitiva `om_kill` substitui funcionalmente `kill`, usando-se a identificação OMNI em lugar do número do processo.

O sistema estendeu o conceito de grupos de processos existente no UNIX para processos distribuídos. É possível criar um novo grupo através da primitiva `om_setpgrp` ou enviar um sinal a um grupo utilizando-se `om_killpg`, muito similares a suas respectivas versões UNIX.

4.3 Recebimento de sinais

No UNIX, a maneira padrão de se receber um sinal é registrando-se (via `sigvec` [30] ou `signal`[4] [30]) uma função que atuará como um tratador assíncrono para esse sinal específico.

Quase todos os sinais enviados por `om_kill` podem ser tratados usando-se normalmente `signal`. No entanto, o término ou interrupção da execução de um processo acarreta o envio de um sinal (SIGCHLD) ao seu pai. Como este pode se localizar numa máquina diferente, faz-se necessário que o OMNI transporte o sinal e ative seu tratador, que deve ser registrado pela primitiva `om_signal`.

UNIX oferece a chamada ao sistema `wait` [30], para bloquear um processo até a morte (ou mudança de estado) de um de seus filhos. OMNI oferece a equivalente `om_wait` para esperar esses eventos de um processo criado por `om_createProc` ou `om_forkExec`.

4.4 Controle de Acesso

Um usuário não pode criar processos em qualquer máquina. Somente máquinas que conheçam o usuário permitirão que se crie nelas um processo. A política implementada pelo Gerenciador de Processos é muito semelhante àquela utilizada pelo comando UNIX `rlogin` [30] para criar sessões remotas em diferentes máquinas da rede. Em resumo, um processo pertencente a um usuário pode criar outros numa máquina remota se o usuário é capaz de nela estabelecer uma sessão `rlogin` sem fornecer nenhuma senha.

OMNI é mais seguro que `rlogin`, pois utiliza criptografia para autenticar a identidade dos processos e dos *daemons* OMNI.

5 Portas

Para que o sistema seja realmente útil, é necessário prover um mecanismo de comunicação entre os processos distribuídos. No OMNI, tal comunicação se dá através de *Portas*, que podem ser *Conectáveis* ou *Não Conectáveis*.

5.1 Portas Conectáveis

No sistema OMNI, processos podem criar portas conectáveis a fim de se comunicarem. Um processo pode possuir um número arbitrário de portas de *saída* e de *entrada*. Podemos interconectar uma porta de saída com outra de entrada, fazendo com que tudo que for escrito por um processo em sua porta de saída possa ser lido pelo outro em sua porta de entrada, tornando transparente a distribuição dos processos. Note que um processo só pode ler e escrever em suas próprias portas.

Portas Conectáveis OMNI estendem o conceito de *pipes* [30] do UNIX para o seu equivalente distribuído. Assim como a *shell* (ou qualquer outro programa) é capaz de redirecionar a saída de um processo para a entrada de outro através de um *pipe* sem que os próprios processos envolvidos se dêem conta disso, o sistema OMNI permite que um processo interconecte portas pertencentes a dois outros sem que estes precisem se preocupar com o estabelecimento dessa conexão. Não é necessário (como no caso de *pipes*) que esses processos sejam filhos daquele que os interconectou.

Este esquema alia a flexibilidade de *sockets* [28], que permitem a interconexão de processos distribuídos quaisquer, à simplicidade e transparência de *pipes*, que permitem que um processo interconecte outros dois. Este recurso permite a construção de diferentes computações distribuídas a partir das diferentes formas de interconexão entre as portas de vários processos. Esta facilidade será amplamente utilizada pela *LegoShell*.

[4] Normalmente usa-se `signal`, de interface mais amigável e mais portável.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

As funções do sistema que manipulam Portas Conectáveis são:

- `om_createConPort(name, attribPermsAndSemantics [, charsOrSize], OMNIportId)`
- `om_destroyConPort(OMNIportId)`
- `om_connect(OMNIport1Id, OMNIport2Id)`
- `om_read(OMNIportId, buffer, bufferSize, more)`
- `om_write(OMNIportId, message, messageSize)`
- `om_disconnect(OMNIportId)`
- `om_conHandler(OMNIportId, handlerFunction)`
- `om_createMbox(site, bufferSize, inName [, inAttribAndPerms], outName [, outAttribAndPerms], OMNIMboxId, OMNIinPortId, OMNIoutPortId)`
- `om_createBcast(site, bufferSize, inName [, inAttribAndPerms], outName [, outAttribAndPerms], OMNIBcastId, OMNIinPortId, OMNIoutPortId)`
- `om_createSpecCon(specConStruct, OMNIids)`
- `om_destroyCon(OMNIconId)`

OMNI oferece primitivas para *criar* e *destruir* (`om_createConPort/om_destroyConPort`) e para *interconectar* portas (`om_connect`). A conexão entre duas portas OMNI recebe o nome de *pipe*, numa analogia ao *pipe* do UNIX.

`om_createConPort` tem como parâmetros o nome simbólico[5] da futura porta (que será cadastrado no Servidor de nomes juntamente com sua OMNIid), o tipo (entrada ou saída), visibilidade, unicidade e demais características. Ela retorna a identificação OMNI da porta, usada em futuras operações de leitura e escrita.

A conexão entre duas portas é feita através de suas identificações OMNI, a fim de manter a transparência de localidade. Através do Servidor de Nomes, pode-se descobrir a identificação OMNI de uma porta a partir do seu nome simbólico.

Uma porta pode também ser conectada a um arquivo. Nesse caso, a leitura ou escrita na porta é mapeada diretamente a uma leitura ou escrita no arquivo.

5.1.1 Conectores Especiais

Um conceito novo introduzido pelo OMNI (derivado da *LegoShell*) é a idéia de conectores especiais.

É possível em UNIX criarmos uma cadeia (*pipeline*) de processos ligados por *pipes*, onde a saída de cada processo liga-se à entrada do seguinte. Além disso, pode-se fazer mais de um processo ler/escrever num único *pipe*. O sistema permite tal arranjo, mas provê pouco ou nenhum suporte para delimitar os dados dentro do *pipe*. Os processos envolvidos é que seriam os responsáveis por implementar regras que evitassem essa mistura. Isso, aliado ao fato de que *pipes* UNIX são incapazes de interligar processos remotos, restringe bastante a utilização prática desse recurso.

OMNI implementa *conectores especiais* que procuram sanar os problemas apontados. Podemos ligar quantas portas quisermos (pelo menos uma) tanto à entrada quanto à saída de um deles[6], conforme mostrado na figura 1. O sistema suporta dois tipos de conectores especiais:

Broadcast As portas de todos os processos C_i, $1 \leq i \leq m$ ligadas à saída do conector K recebem uma cópia de todos os dados enviados por cada uma das portas dos processos P_j, $1 \leq j \leq n$ ligadas à entrada de K.

Mailbox Somente uma porta pertencente a algum processo C_i, $1 \leq i \leq m$ ligada à saída de K recebe um dado que entrou por uma porta pertencente a algum processo P_j, $1 \leq j \leq n$ ligada à entrada de K. Quando um processo lê um dado de uma porta conectada ao *mailbox*, ele consome esse dado, impedindo que ele seja lido novamente em outra porta.

---

[5]Em todas as funções do sistema que criam objetos com nomes é possível especificar `NULL` no lugar do nome, fazendo com que o objeto seja criado sem nome nenhum.

[6]Somente portas de conectores especiais podem estar conectadas simultaneamente a mais de uma porta. Portas pertencentes a processos só podem, num dado instante, estar conectadas a no máximo uma porta.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

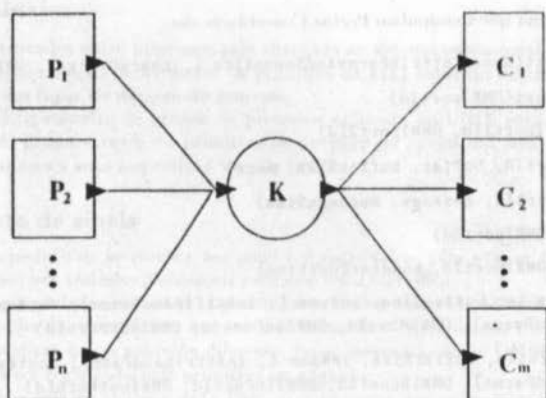

Figura 1: Conector especial K interligando n processos produtores $P_1, P_2, \ldots, P_n$ e m processos consumidores $C_1, C_2, \ldots, C_m$.

Com esses dois tipos de conectores, é possível generalizar conexões unidimensionais (*pipelines*) para conexões bidimensionais, mas resta ainda uma questão: como um conector especial interpreta e separa os dados que recebe? Essa pergunta será respondida a seguir.

5.1.2 Semântica dos Dados

A semântica da primitiva `om_read` é muito parecida à da chamada ao sistema operacional `read` (o mesmo ocorre entre `om_write` e `write`). A diferença é que uma lê (ou escreve) de uma porta, ao passo que a outra o faz de um descritor de arquivo. Tal semântica, entretanto, acarreta um sério problema: dados escritos (ou lidos) através das portas carecem de qualquer estrutura, sendo apenas uma seqüência de *bytes*. Isso pode constituir um problema ao interligarmos duas portas e certamente *é* um grande problema quando ligamos várias portas a um conector especial, pois o conector não distingüiria o início e fim de mensagens, acabando por combinar nas saídas diferentes trechos de mensagens recebidas.

Este problema pode ser resolvido se, ao criarmos uma porta, especificarmos que os dados a trafegar por ela possuem uma certa estrutura. Ou seja, devemos atribuir alguma *semântica* a esses dados. As várias semânticas suportadas pelo sistema são:

Stream É uma seqüência de *bytes* sem estrutura alguma.

Tamanho fixo Dados têm tamanho fixo em *bytes*, como estruturas na linguagem C[7] [21].

Caractere separador *Bytes* de valor especificado delimitam o término de cada dado.

Tamanho Variável Uma escrita numa porta (mais especificamente uma ativação da primitiva `om_write`), envia à porta um bloco indivisível de dados, não importando o tamanho desse bloco. Quando um processo ler esses dados, lerá o bloco inteiro. Note que esta semântica requer o uso das primitivas `om_read` e `om_write` e portanto não está disponível para processos comuns (processos comuns serão explicados na seção 5.4).

Herdada Semântica implicitamente herdada da porta à qual foi conectada. Todas as portas de conectores especiais são deste tipo.

[7]Precauções devem ser tomadas ao ligar máquinas heterogêneas, quanto ao alinhamento e ordenação de *bytes*.

OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

Estes recursos serão a base para a implementação de portas de comunicação tipadas pelas linguagens em desenvolvimento no Projeto A_HAND. Nelas será possível declarar o tipo dos dados que trafegam entre uma porta e outra e efetuar uma verificação de tipos no momento da conexão. Este verificador fará parte do sistema de execução dessas linguagens (inicialmente Cm). Com isto, o conceito de verificação de tipos será estendido para fora do escopo das linguagens de programação, abrangendo também os programas constituídos pela interconexão de programas mais simples.

Note que é possível especificar a semântica dos dados mesmo sem conectores especiais envolvidos.

5.2 Portas Não Conectáveis

A abordagem de portas conectáveis descrita acima mostra-se inadequada em situações onde o estabelecimento de uma conexão provoca um *overhead* desnecessário. Por exemplo, no modelo cliente servidor, a criação de duas conexões (uma para a requisição do serviço, outra para a resposta) pode ser evitada. Devido a problemas desse tipo, OMNI introduziu o conceito de portas não conectáveis, inspirando-se parcialmente no sistema Chorus. Assim como as conectáveis, elas possuem nomes simbólicos e identificações OMNI, mas não exigem conexão para o envio de mensagens. As funções que as manipulam são:

- `om_createConLessPort(name, attributesAndPermissions, OMNIportId)`
- `om_destroyConLessPort(OMNIportId)`
- `om_send(OMNIportId, message, messageSize)`
- `om_receive(OMNIportId, message, maxMsgSize, more)`
- `om_conLessHandler(OMNIportId, handlerFunction)`
- `om_createPortGroup(name, attributesAndPermissions, joinAndExitPermissions, OMNIportGrpId)`
- `om_insertPort(OMNIportGrId, OMNIportId)`
- `om_removePort(OMNIportGrId, OMNIportId)`
- `om_destroyPortGroup(OMNIportGrpId)`

A primitiva `om_createConLessPort` cria uma porta não conectável. Diferentemente das conectáveis, só existem portas não conectáveis de *entrada*. Um processo pode enviar mensagens a qualquer porta, mas só pode receber mensagens em suas próprias portas. Para enviar, ele deve utilizar a primitiva `om_send`, que recebe como parâmetro a porta destino. Para receber, ele deve se valer de `om_receive`, passando como argumento a porta onde recebeu a mensagem.

5.2.1 Grupos de Portas

OMNI provê a facilidade de agregarmos zero ou mais portas não conectáveis em *grupos*. Grupos de portas possuem um nome simbólico e uma identificação OMNI, e podem ser referenciados exatamente da mesma maneira que uma porta. Equivalem a conectores especiais para portas conectáveis.

Existem duas políticas possíveis para o tratamento de mensagens enviadas a um grupo:

Broadcast Todas as portas pertencentes ao grupo recebem uma cópia da mensagem.

Mailbox Somente uma das portas pertencentes ao grupo recebe a mensagem.

Um uso interessante para grupos ocorre quando o usuário utiliza portas não conectáveis para implementar o modelo cliente-servidor. A existência de grupos possibilita serviços mais persistentes, pois se uma das portas do grupo deixa de funcionar, o serviço continua sendo oferecido pelas outras. Além disso, simplifica a reconfiguração dos serviços.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

5.3 Recebimento Assíncrono de Mensagens

Tanto a primitiva `om_read` para Portas Conectáveis como `om_receive` para Portas Não Conectáveis podem ser usadas de forma a bloquear ou não o requisitante caso não haja mensagens a ler.

É possível também, através de `om_conHandler` e `om_conLessHandler`, especificar uma função tratadora para o recebimento de mensagens em uma porta. A função é automaticamente chamada a cada mensagem recebida, com os parâmetros apropriados. Na linguagem Cm, o recebimento (assíncrono) de uma mensagem é mapeado para uma *exceção* cujo tratador é uma função especificada pelo sistema de execução da linguagem através de `om_comLessHandler`.

5.4 Suporte a Processos Comuns

Processos comuns são aqueles que não fazem chamada a nenhuma primitiva OMNI. Apesar disso, o sistema permite que eles se comuniquem através de portas OMNI pois, quando é criado um processo, pode-se redirecionar seus descritores padrão (entrada, saída ou saída de erro) para portas de comunicação do sistema.

Para Portas Conectáveis, quaisquer dos três descritores padrão pode opcionalmente ser redirecionado. No caso de Portas Não Conectáveis, isso pode ser feito somente para a entrada padrão (não existem Portas Não Conectáveis de saída).

Processos comuns lêem e escrevem em suas portas utilizando as chamadas `read` e `write`, e não as primitivas OMNI. Ou seja, entendem seus dados como uma seqüência de *bytes*. As semânticas atribuíveis às Portas Conectáveis (seção 5.1.2) minimizam este problema, mas deve ter-se em mente que processos comuns não são capazes de tratar seus dados com a mesma elegância que processos OMNI. Em particular, não distinguem fronteiras entre duas mensagens recebidas.

Como podemos ver, OMNI permite interligar de forma distribuída praticamente *qualquer* processo, inclusive comandos UNIX, como `ls`, `grep` ou `sort`.

6 Exemplos

A seguir são apresentados alguns exemplos de computações distribuídas utilizando o sistema OMNI. O primeiro deles ilustra o uso de Portas Conectáveis e o segundo mostra a utilização de Portas Não Conectáveis.

6.1 *Mergesort*

Mergesort é um programa que ordena um arquivo executando duas ordenações paralelas de partes do arquivo e depois combinando essas ordenações parciais em um único arquivo.

Conforme mostrado na figura 2, o programa consiste em ligar um arquivo a um conector especial tipo *mailbox*, que irá ler seu conteúdo e fornecê-lo sob demanda para os dois programas *sort* que executam paralelamente em máquinas distintas. O resultado da ordenação de cada um deles é então passado ao programa *merge*, que os combina para produzir o arquivo ordenado.

O código que implementa o programa *merge* é mostrado na figura 3. O código do programa *mergesort* se encontra na figura 5. O programa *sort* é o próprio comando `sort` disponível no UNIX, que pode ser usado graças à capacidade do sistema de utilizar processos comuns (veja seção 5.4).

Na figura 5, a primeira chamada a `om_createProc`, cria o processo comum "*sort*" na máquina "enterprise" e redireciona sua entrada e saída padrão para portas conectáveis com semantica de caractere separador ('\n'). A saída padrão de erro foi redirecionada para o arquivo "/dev/null". Como não havia necessidade, nenhum nome simbólico foi atribuído às portas. Note que na criação do processo *merge*, nenhum descritor padrão foi redirecionado, pois *merge* não é um processo comum e cria suas próprias portas.

`om_getId` consulta o Servidor de Nomes a fim de obter a OMNIid das portas cujo nome foi fornecido. Note que é especificado um intervalo de tempo durante o qual o processo se dispõe a esperar pela criação do nome, pois *merge* pode não ter ainda criado as portas no instante em que *mergesort* tenta obter suas OMNIids. Caso isso aconteça, *mergesort* fica bloqueado esperando a criação do nome por no máximo `TIME_OUT` segundos.

O programa *merge* cria três portas, duas de entrada e uma de saída, e compara as linhas de texto recebidas em cada uma das portas de entrada, escrevendo na de saída aquela que for lexicograficamente menor. Note que as portas de *merge* possuem nomes simbólicos e uma série de atributos: visibilidade global, seus nomes não são

Compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

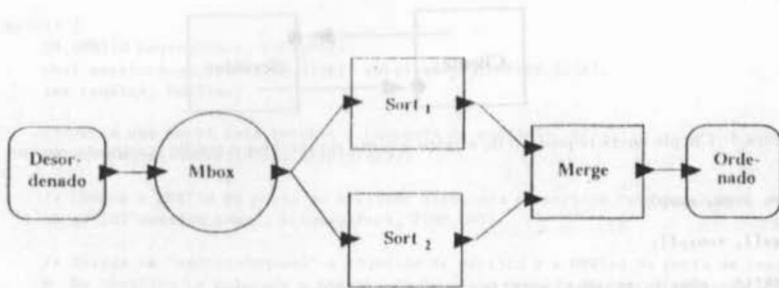

Figura 2: Diagrama mostrando a interconexão de processos, *mailbox* e arquivos no programa distribuído *mergesort*.

```
main()
{
    OM_OMNIid input[2], outId;
    char inBuffer[2][BUFFER_SIZE];
    int port, nBytes[2];

    /*  Cria duas portas de entrada e uma de saida. */
    om_createConPort("mergeIn1", OM_INPUT | OM_GLOBAL | OM_NOTUNIQUE | OM_OWN | OM_SEPCHAR,
        "\n", &input[0]);
    om_createConPort("mergeIn2", OM_INPUT | OM_GLOBAL | OM_NOTUNIQUE | OM_OWN | OM_SEPCHAR,
        "\n", &input[1]);
    om_createConPort("mergeOut", OM_OUTPUT | OM_GLOBAL | OM_NOTUNIQUE | OM_OWN | OM_STREAM,
        &outId);

    /*  Le as primeiras linhas das portas de entrada. */
    nBytes[0] = om_read(&input[0], inBuffer[0], BUFFER_SIZE, NULL);
    nBytes[1] = om_read(&input[1], inBuffer[1], BUFFER_SIZE, NULL);

    do {
        /* Compara as duas linhas e coloca o indice da menor delas em "port". */
        ...

        /*  Escreve a menor linha na porta de saida. */
        om_write(&outId, inBuffer[port], nBytes[port]);

        /*  Enquanto os dados em uma das portas nao acabarem, continue lendo mais dados. */
    } while(nBytes[port] = om_read(input[port], inBuffer[port], BUFFER_SIZE, NULL));

    /*  Le o resto dos dados da porta que sobrou e os escreve na porta de saida. */
    port = !port;
    while((nBytes[port] = om_read(input[port], inBuffer[port], BUFFER_SIZE, NULL))
        om_write(&outId, inBuffer[port], nBytes[port]);

    /*  Desconecta a porta de saida. */
    om_disconnect(&outId);
}
```

Figura 3: Código do programa *merge*, que combina dois textos ordenados em um só.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

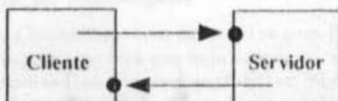

Figura 4: Cliente envia requisição de serviço à porta do servidor e recebe a resposta em sua porta.

```
main(argc, argv, envp)
int argc;
char *argv[], *envp[];
{
   OM_OMNIid  mboxIn, mboxOut, sort1In, sort1Out, sort2In, sort2Out,
              mergeIn1, mergeIn2, mergeOut;
   char *args[2];  /* Contem os argumentos de chamada dos processos. */

   /* Cria um conector especial tipo mailbox na maquina "galileu". */
   om_createMbox("galileu", MBOX_BUFFER_SIZE, NULL, &mboxIn, &mboxOut);

   /* Conecta a entrada do mailbox com um arquivo nao ordenado. */
   om_connFile(&mboxIn, "unsorted_file", O_RDONLY, OM_NOMODE);

   /* Cria um processo executando o programa sort na maquina "enterprise". */
   om_createProc("sort", args, envp, "enterprise", NULL,
      OM_STDIN, OM_CONNPORT, NULL, OM_SEPCHAR, "\n", &sort1In,
      OM_STDOUT, OM_CONNPORT, NULL, OM_SEPCHAR, "\n", &sort1Out,
      OM_STDERR, OM_FILE, "/dev/null", O_WRONLY, OM_NOMODE,
      NULL);

   /* Cria mais uma instancia de sort na maquina "excelsior". */
   om_createProc("sort", args, envp, "excelsior", NULL,
      OM_STDIN,  OM_CONNPORT, NULL, OM_SEPCHAR, "\n", &sort2In,
      OM_STDOUT, OM_CONNPORT, NULL, OM_SEPCHAR, "\n", &sort2Out,
      OM_STDERR, OM_FILE, "/dev/null", O_WRONLY, OM_NOMODE,
      NULL);

   /* Conecta a saida do mailbox com a entrada de cada um dos 2 sorts. */
   om_connect(&mboxOut, &sort1In);
   om_connect(&mboxOut, &sort2In);

   /* Cria um processo que executa o programa merge na maquina "stargazer". */
   om_createProc("merge", args, envp, "stargazer", NULL, NULL);

   /* Obtem as OMNIids de cada uma das 3 portas do programa merge. */
   om_getId("mergeIn1", &mergeIn1, TIME_OUT);
   om_getId("mergeIn2", &mergeIn2, TIME_OUT);
   om_getId("mergeOut", &mergeOut, TIME_OUT);

   /* Conecta a saida dos sorts com cada uma das entradas do merge. */
   om_connect(&sort1Out, &mergeIn1);
   om_connect(&sort2Out, &mergeIn2);

   /* Conecta a saida do merge com o arquivo "sorted_file". */
   om_connFile(&mergeOut, "sorted_file", O_WRONLY, 0666);
}
```

Figura 5: Código do programa *mergesort*

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

```
main() {
    OM_OMNIid requestPort, replyPort;
    char serviceRequest[BUFFER_SIZE], serviceReply[BUFFER_SIZE];
    int reqSize, repSize;

    /* Cria uma porta para receber a resposta do servidor. */
    om_createConLessPort(NULL, &replyPort);

    /* Obtem a OMNIid da porta do servidor associada ao servico "service_name". */
    om_getId("service_name", &requestPort, TIME_OUT);

    /* Coloca em "serviceRequest" a requicao de servico e a OMNIid da porta de resposta.
     * Em "reqSize" e colocado o tamanho da mensagem contida em "serviceRequest". */
    ...
    /* Envia a requisicao ao servidor. */
    om_send(&requestPort, serviceRequest, size);

    /* Recebe a resposta da execucao do servico. */
    repSize = om_receive(&replyPort, serviceReply, BUFFER_SIZE, NULL);
}
```

Figura 6: Código do programa cliente.

```
void serviceHandler(servicePort, serviceRequest, reqSize)
    OM_OMNIid *servicePort;
    char *serviceRequest;
    int reqSize;
{
    OM_OMNIid replyPort;
    char serviceReply[BUFFER_SIZE];
    int repsize;

    /* Recebe a OMNIid da porta do servidor, a requisicao de servico e o tamanho da requisicao.
     * Executa o servico. Atribui a "replyPort" a OMNIid da porta de resposta extraida de
     * "serviceRequest". Coloca a resposta em "serviceReply" e o tamanho dela em "size". */
    ...
    /* Responde a requisicao. */
    om_send(&replyPort, serviceReply, size);
}

main() {
    OM_OMNIid servicePort;

    /* Cria a porta onde sera fornecido o servico "service_name". */
    om_createConLessPort("service_name",
        OM_GLOBAL | OM_UNIQUE | OM_OWN | OM_GROUP | OM_DOMAIN, &servicePort);

    /* Registra a funcao "serviceHandler()" como  tratadora da porta "servicePort". */
    om_conLessHandler(&servicePort, serviceHandler);

    /* O Servidor agora pode ficar processando o que quiser. Quando chegar uma
     * requisicao ele tera sua execucao desviada para a funcao tratadora. */
    ...
}
```

Figura 7: Código do programa servidor.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

necessariamente únicos e possuem permissão de acesso apenas para processos do mesmo usuário dono de *merge*. As portas de entrada possuem semântica de caractere separador '\n' e a de saída, de *stream*.

6.2 Cliente e Servidor

Este é um exemplo de utilização de portas não conectáveis para implementar o modelo cliente-servidor. Conforme ilustrado na figura 4, o cliente envia uma requisição de serviço à porta do servidor acompanhada da identificação OMNI da sua própria porta. O servidor então atende a requisição e envia a resposta à porta que o cliente lhe forneceu. O código que implementa o cliente é mostrado na figura 6. O do servidor na figura 7.

Como podemos observar, o cliente constrói uma mensagem contendo a requisição de serviço e a identificação OMNI da sua própria porta, onde receberá a resposta do servidor, e envia essa mensagem utilizando om_send. Ele então lê de sua porta usando om_receive, ficando bloqueado até que a resposta chegue.

O servidor cria a porta onde será fornecido o serviço e registra para ela uma função tratadora, que será invocada sempre que for enviada uma mensagem à porta, permitindo que o servidor possa ser interrompido assincronamente para atender às requisições.

Como pudemos observar, o uso de portas normalmente requer a *serialização* das mensagens, que podem constituir estruturas razoavelmente complexas. Esse problema pode ser sanado através do uso de um serializador automático de estruturas, como por exemplo o XDR [28] (*eXternal Data Representation*) ou o linear [10], este último em desenvolvimento no Projeto A_HAND.

7 Conclusão

O sistema OMNI cria a abstração de objetos distribuídos identificando-os através de um mecanismo uniforme (OMNIid). Eles têm localização transparente e podem ser referenciados através de nomes simbólicos.

Dentre os objetos suportados encontram-se processos, portas, conectores especiais e grupos de portas. Na medida do possível, constituem extensões naturais de seus equivalentes centralizados do UNIX.

Com o sistema OMNI, não há necessidade de se utilizar mecanismos padrão do UNIX como sockets (ou TLI [28]) ou mesmo RPC [28], uma vez que são excedidos em funcionalidade pelas facilidades do OMNI.

O sistema aqui descrito não pretende oferecer diretamente facilidades de orientação a objetos como as encontradas em alguns sistemas recentes. Elas serão fornecidas pelo conjunto das linguagens Cm, *LegoShell* e CO^2 que, utilizando o OMNI, oferecerão abstrações de mais alto nível. Será suportado também um mecanismo de verificação de tipos comum a pelo menos estas três linguagens.

Agradecimentos

Os autores gostariam de agradecer a Carlos A. Furuti, Bill W. Coutinho, Maurício Fernández e outros tantos membros do Projeto A_HAND que, de uma forma ou de outra, contribuíram para a concepção do sistema OMNI, bem como para a elaboração deste artigo.

Referências

[1] Mike Accetta, Robert Baron, David Golub, Richard Rashid, Avadis Tevanian, and Michael Young. Mach: A new kernel foundation for UNIX development. Technical report, Department of Computer Science, Carnegie Mellon University, August 1986.

[2] François Armand, Michel Gien, Frédéric Herrmann, and Marc Rozier. Revolution 89 or "Distributing UNIX Brings it Back to its Original Virtues". In *Workshop on Experiences with Distributed (and Multiprocessor) Systems*, pages 153–174, Ft. Lauderdale, FL, USA, October 1989.

[3] David R. Cheriton. The V kernel: A software base for distributed systems. *IEEE Software*, April 1984.

[4] David R. Cheriton. The V distributed system. *Communications of the ACM*, 31(3), March 1988.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

[5] Leírton S. de Castro. SISTRAC: Sistema de Suporte a Trabalho Cooperativo. Tese de mestrado, Departamento de Ciencia da Computação da Universidade Estadual de Campinas, 1991.

[6] Defense Advanced Research Projects Agency, Information Processing Techniques Office, RFC 791. *Internet Program Protocol Specification*, September 1981.

[7] Fred Douglis and John Ousterhout. Process migration in the Sprite operating system. In *Proceedings of the 7th International Conference on Distributed Computing Systems*, September 1987. Reprinted by the Computer Society Press of the IEEE.

[8] Rogério Drummond. *LegoShell* linguagem de computações. *III Simpósio Brasileiro de Engenharia de* Software, pages 1-13, Recife, PE, outubro 1989.

[9] Rogério Drummond e Fábio Q. B. da Silva. Linguagem Cm: Manual de referencia. *Anais da IV Reunião de Trabalho do Projeto ESTRA*, páginas 175-210. SID Informática, outubro 1988.

[10] Rogério Drummond e Marcelo A. H. de Souza. Linear - linearizador de estruturas complexas. Relatório técnico, Projeto A_HAND. DCC - IMECC - UNICAMP, 1992.

[11] Rogério Drummond e Hans Liesenberg. A_HAND Ambiente de desenvolvimento de *software* baseado em Hierarquias de Abstração em Níveis Diferenciados. *IV Encontro de Trabalho do Projeto Ethos*, Petrópolis, RJ, abril 1987. Revisto e reimpresso como relatório técnico do Projeto A_HAND em outubro de 1987.

[12] Rogério Drummond e Hans Liesenberg. Requisitos para um ambiente de desenvolvimento de programas. *I Encontro IBM de Ciência e Tecnologia em Informática*, Rio de Janeiro, RJ, novembro 1987.

[13] Mauricio Fernández e Rogério Drummond. A_HAND Ambientes e Linguagens. Relatório técnico, Projeto A_HAND. DCC- IMECC - UNICAMP, 1991. Capítulo "Sobre uma linguagem de prototipagem para ambiente Unix".

[14] Mauricio Fernández e Rogério Drummond. Linguagem de Comandos para Desenvolvimento de *Software*. Proposta de tese de mestrado do primeiro autor submetida ao Departamento de Ciencia da Computação da Unicamp, outubro 1991.

[15] Carlos A. Furuti. Introdução a Cm. Relatório técnico, Projeto A_HAND. DCC - IMECC - UNICAMP, setembro 1991.

[16] Carlos A. Furuti. Um compilador para uma linguagem de programação orientada a objetos. Tese de mestrado, Departamento de Ciencia da Computação da Universidade Estadual de Campinas, julho 1991.

[17] Per Brinch Hansen. Distributed Processes: A Concurrent Programming Concept. *Communications of the ACM*, November 1978.

[18] C.A.R. Hoare. Communicating Sequential Processes. *Communications of the ACM*, August 1978.

[19] Michael B. Jones and Richard F. Rashid. Mach and Matchmaker: Kernel support for object-oriented distributed systems. In *OOPSLA'86 Proceedings*. Association for Computing Machinery, 1986.

[20] Celso G. Junior. Suporte para a programação em sistemas distribuídos. Proposta de tese de mestrado submetida ao Departamento de Ciência da Computação da Unicamp, abril 1992.

[21] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice-Hall, Inc., 1978.

[22] Jeff Magee and Naranker Dulay. MP: A Programming Environment for Multicomputers. Submited for publication.

[23] Jeff Magee, Naranker Dulay, and Jeff Kramer. Constructive Communication in MP. Submited for publication.

[24] Jeff Magee, Naranker Dulay, and Jeff Kramer. Structuring Parallel and Distributed Programs. Submited for publication.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

[25] John K. Ousterhout, Andrew R. Cherenson, Frederick Douglis, Michael N. Nelson, and Brent B. Welch. The Sprite network operating system. *IEEE Computer*, February 1988.

[26] M. L. Powel, S. R. Kleiman, S. Barton, D. Shah, D. Stein, and M. Weeks. SunOS multi-thread architecture. Technical report, Sun Microsystems Inc., 1991.

[27] M. Rozier, V. Abrossimov, F. Armand, I. Boule, M. Gien, M. Guillemont, F. Herrmann, C. Kaiser, S. Langlois, P. Léonard, and W. Neuhauser. Overview of the CHORUS Distributed Operating System. Technical Report CS/TR-90-25, Chorus systèmes, April 1990.

[28] Sun Microsystems. *Network Programming Guide*, March 1990.

[29] Sun Microsystems. *Programming Utilities & Libraries*, March 1990. Chapter 2 "Lightweight Processes".

[30] Sun Microsystems. *SunOS Reference Manual*, March 1990.

[31] Alexandre P. Teles. Extensão da linguagem Cm. Proposta de tese de mestrado submetida ao Departamento de Ciência da Computação da Unicamp, abril 1992.

[32] Lídia A. R. Yamamoto e Rogério Drummond. Ferramentas para *Groupware*. Relatório técnico, Projeto A_HAND, DCC - IMECC - UNICAMP, 1991. Capítulo "Sistema de Teleconferencia".

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor