Sistema Gerenciador de Processamento Cooperativo

Este exemplar corresponde a redação final da tese devidamente corrigida e defendida pela Sra. Ivonne Martínez Carrazana e aprovada pela Comissão Julgadora.

Campinas, 16 de março de 1993

.....
Prof. Dr. Nelson C. Machado

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação, UNICAMP, como requisito parcial para obtenção do Título de MESTRE em Ciência da Computação.

Universidade Estadual de Campinas
Instituto de Matemática, Estatística e Ciência da Computação
Departamento de Ciência da Computação

Sistema Gerenciador de Processamento Cooperativo

Ivonne Martínez Carrazana

Orientador: Prof. Dr. Nelson C. Machado
Coorientador: Prof. Dr. Célio C. Guimarães

16 de março de 1993

Dissertação apresentada ao Departamento de Ciência da Computação
como parte dos requisitos exigidos para a obtenção
do título de Mestre em Ciência da Computação

A Branca de Neve (Carol).

Aos outros seis anões:

Atchim (Chris), Dengoso (Valéria),
Dunga (Silvinha), Mestre (Inés),
Zangado (Marquinhos), e

especialmente A Feliz (Cecil),
exemplo de atitude perante a vida.

Soneca.

Conteúdo

Lista de Figuras

Agradecimentos

Ao Prof. Dr. Nelson C. Machado e ao Prof. Dr. Célio C. Guimarães, pelas orientações.

À CAPES e à FAEP que forneceram apoio financeiro para realização do Mestrado.

À Universidad de La Habana, pela confiança e apoio que me proporciona há anos.

A minha família que soube estar presente, apesar da distância.

A Paulo, Atta, Lincoln e Nabor, pelas diversões em comum.

A Furuti, pela permanente disposição a ajudar.

A todos os colegas que fizeram valiosas correções e sugestões do texto.

À colônia de amigos estrangeiros no Brasil.

À turma toda dos colegas da Pós, pela solidaridade.

Ao Brasil, pela oportunidade oferecida.

o o o

Resumo

Este trabalho apresenta o projeto e a implementação de um Sistema Gerenciador de Processamento Cooperativo (SGPC). A utilização do SGPC facilita a exploração da capacidade de paralelismo de uma rede local. Foram desenvolvidos um **Servidor de Processamento** e um pacote de rotinas que serve de interface para o servidor.

A utilização do sistema pode ser feita em um de dois níveis: seguindo o modelo Cliente/Servidor através de Chamadas Remotas de Procedimentos (RPCs), ou através de subrotinas de biblioteca que se encarregam de gerar as RPCs e oferecem ao usuário um ambiente mais amigável.

Em ambos os níveis se realizam execuções remotas com seleção automática da máquina hospedeira, e garante-se a transparência destas operações para os usuários do sistema. A escolha das máquinas de destino das migrações baseia-se no nível de ociosidade de suas CPUs.

O sistema visa facilitar o desenvolvimento de aplicações paralelas com baixo custo de implementação, aumentar a velocidade de processamento global de tais aplicações, e colaborar no aumento do nível de *conectividade* da rede, permitindo o aproveitamento dos recursos das estações ociosas.

Abstract

A Cooperative Processing Management System design and its implementation are presented. The system provides facilities to exploit the inherent parallelism of a local area network. A Processing Server was developed, as well as a set of library functions which provide a high level interface to the processing server.

It is possible to use the system at two levels: following a Client/Server model, through Remote Procedure Calls (RPCs), or making calls to subroutines in a library. These routines generate the RPCs, providing a friendlier interface with the system.

At both levels remote executions are performed with automatic selection of the host, in a transparent way. Target machines are selected according to their level of idle processing capability.

The system is intended to improve the development of parallel applications with low implementation cost, to increase the global processing speed, and to improve the network connectivity level, making it possible to utilize the resources of idle workstations.

Capítulo 1

Introdução

A pesquisa por equipamentos melhores e mais rápidos sempre foi uma área de grande interesse em computação, visando resolver problemas cada vez mais complexos.

Algumas aplicações importantes em engenharia e ciência exigem quantidades de tempo tão consideráveis que são impraticáveis de serem implementadas nos computadores disponíveis. Os requisitos computacionais destes problemas estão na classe de supercomputadores e de arquiteturas concorrentes.

O crescimento no desempenho dos monoprocessadores, embora contínuo e acelerado, vai-se aproximando de limites físicos insuperáveis. Entretanto, a demanda por maior poder de processamento continua sempre crescendo.

As empresas fabricantes de computadores têm observado que não basta melhorar somente o desempenho dos processadores individuais. É necessário também criar novas organizações de computadores, que empreguem multiprocessadores [Kuc86] (processamento paralelo), em vez de um único processador (processamento seqüencial).

Para que seja possível utilizar, de forma simultânea, e efetivamente mais de um processador, realizam-se pesquisas sobre paralelismo nas seguintes áreas:

- algoritmos e aplicações;
- linguagens e compiladores;
- arquiteturas;
- ambientes de programação.

Algoritmos e Aplicações: Nesta linha, desenvolvem-se algoritmos capazes de dividir os passos computacionais entre um número de processadores, e de serem mapeados em arquiteturas concorrentes, explorando adequadamente seus recursos, um exemplo é o sistema HeNCE (*Heterogeneous Network Computing Environment*) [Beg91a] [Beg91b]. Trabalha-se para desenvolver pacotes de aplicações e rotinas de biblioteca utilizando algoritmos paralelos para problemas padrões.

Linguagens e Compiladores: Nesta área o trabalho está orientado à criação de novas linguagens que permitam expressar, numa forma bem estruturada, algoritmos para processamento paralelo. A linguagem **Occam** é um exemplo neste sentido, e embora tenha sido projetada para gerar código para os **Transputers**, sua aplicabilidade tem chegado mais longe [Tay82]. Estas linguagens ajudam a isolar o programador dos detalhes de *hardware*, e até do sistema operacional na sua utilização concorrente dos recursos. Procura-se construir compiladores capazes de explorar efetivamente as facilidades disponíveis em diferentes arquiteturas, visando compilar eficientemente programas já escritos em versões seqüenciais da linguagem.

Arquiteturas (inclui o *hardware* e os sistemas operacionais [Tan87]): Para conseguir maior poder de processamento pode-se trabalhar, na área de arquitetura, em dois níveis diferentes [Fat83]:

- a nível de processador;
- a nível de sistema

No primeiro caso, ao tentar conseguir maior desempenho desenvolvendo inovações a nível de processador, a pesquisa se concentra em introduzir paralelismo ou concorrência dentro de uma CPU, com superposição de diferentes fases de busca, decodificação e execução (*pipelining*), assim como na execução simultânea de mais de uma instrução em diferentes unidades dedicadas.

Já no segundo caso, o esforço para melhorar o desempenho a nível de sistema tenta explorar a execução concorrente de diferentes partes da tarefa. Neste sentido, uma execução concorrente pode ser realizada por um único processador que divida seu tempo entre as diferentes partes (às vezes aparentando aos usuários a existência de mais de um processador), ou pelo emprego real de vários processadores (aqui é muito importante a organização dos processadores em relação à arquitetura e/ou topologia).

Ambientes de programação: Trabalha-se no desenvolvimento de ambientes de programação efetivos para usar o *software* e os recursos de *hardware*; estes ambientes podem estar ligados a linguagens de programação, como é o caso de **Linda**. São desenvolvidas ferramentas que facilitam a distribuição de módulos para seu processamento por diferentes processadores interligados [Beg91b] [Dru92] [Che84] [Che88].

1.1 Técnicas para Melhorar o Desempenho

Flynn [Fly66] introduziu uma classificação das arquiteturas segundo a simultaneidade no fluxo de instruções e de dados, que tem tido uma ampla aceitação [Sie79, Fat83]. Segue abaixo tal classificação:

- **SISD** *(Single Instruction Single Data)*
- **MISD** *(Multiple Instruction Single Data)*
- **SIMD** *(Single Instruction Multiple Data)*
- **MIMD** *(Multiple Instruction Multiple Data)*

No caso de **SISD** trata-se da arquitetura tradicional de von-Neumann, que é muito propensa a gargalos. Esforços têm sido feitos para melhorar o desempenho ainda dentro desta categoria, por exemplo: técnicas de *pipeline*, arquiteturas de tipo Harvard, e de tipo RISC [Hes87].

Alguns autores apresentam a técnica de utilizar diferentes unidades especializadas de processamento, sendo que todas elas constituem-se em um único processador central (utilizadas em *pipeline*), como arquiteturas de tipo **MISD**. Porém, a velocidade de uma máquina com este tipo de processador central está limitada pelo número de segmentos de *pipeline* inerentes a uma operação particular.

Os processadores usados em supercomputadores ganharam velocidade nos últimos 20 anos com técnicas de *pipeline*, instruções *vetorizáveis*, múltiplas unidades funcionais e componentes muito rápidos. No entanto, essas máquinas são onerosas, e as melhorias no desempenho tornam-se cada vez mais difíceis de serem obtidas [Col86]. Isto faz com que os esforços sejam conduzidos às novas organizações.

No caso de **SIMD**, trabalha-se com vários processadores individuais fortemente acoplados *(tightly coupled)* [Sie79], em geral com controle comum, que executam todos as mesmas instruções sobre diferentes dados. Como exemplos temos os vetores *sistólicos (systolic arrays)* [Kun85], projetados para operar em paralelo sobre um programa, e os processadores para processamento *vetorial*, por exemplo os dos supercomputadores da série FPS [Mil87]. Embora sendo muito eficiente, este tipo de organização fica limitado a aplicações razoavelmente especializadas [Yew86].

Sistemas de computadores paralelos prometem desempenho mais elevado, não alcançado antes em grandes configurações, e melhor relação preço/desempenho que os computadores seqüenciais em pequenas configurações [Les85].

Com a existência de mais de um processador no sistema, começa-se a falar em termos de "topologias", e não somente de arquiteturas.

As arquiteturas de tipo **MIMD** podem aparecer, dependendo de seu nível de *acoplamento*, em dois grandes grupos: *forte* e *fracamente acopladas* [Fat83]. A comunicação entre os processos em cada nó realiza-se, em geral, através de memória compartilhada e/ou de troca de mensagens.

Como exemplos de topologias de tipo *MIMD fortemente acopladas* temos as numerosas versões de arquiteturas de *hipercubo*: o NCube [Bar86] [Col86] [Pal86], o Cubo Cósmico [Sei85], assim como os **transputers** [Bar83], entre outras. Em geral, estes sistemas são chamados de multiprocessadores.

Nas topologias **MIMD fracamente acopladas** podemos localizar sistemas baseados em uma hierarquia de processadores mestre-escravos e as redes, que se apresentam em uma ampla variedade segundo seu modo de operação (síncrono, assíncrono, combinado), sua estratégia de controle (centralizada, distribuída), seu método de comutação (*circuit switching*, *packet switching*, combinado), sua topologia (estática, dinâmica), e seu alcance (LAN, WAN), entre outras [Fen81, Mok85, Mad88].

Alguns autores falam ainda de topologias *moderadamente acopladas*, colocando nesta categoria os sistemas de microcomputadores distribuídos; outros falam de multicomputadores contrapondo-os aos multi-processadores [Pak83].

De fato, as semelhanças entre as arquiteturas paralelas e distribuídas fazem com que ambas possam ser usadas para desenvolvimentos de programação paralela.

O Sistema objeto deste trabalho poderia ser classificado como MIMD *fracamente* acoplado.

1.1.1 Sistemas Distribuídos e seus Níveis de *Conectividade*

Quando os computadores surgiram a comunicação entre eles era impraticável. Quando evoluíram para sistemas multiusuários [Tho78], a comunicação limitou-se a protocolos de terminal-computador.

O uso e a demanda de comunicação aumentaram e surgiram o correio eletrônico e as redes; a evolução continuou e novos usos foram criados, como *log-in's* remotos e protocolos de execução remota. Hoje em dia se trabalha em sistemas de suporte a trabalho cooperativo [Cas91], e se fala em termos de *groupware*. A evolução deu lugar à idéia do desenvolvimento de Sistemas Operacionais Distribuídos, e exemplos destes são os sistemas operacionais MACH [Jon86] e Sprite [Dou87, Ous88].

Os sistemas distribuídos existentes [Mag89] variam desde aqueles que somente provêem interligação de sistemas autônomos até aqueles que provêem uma linguagem e um ambiente completo para a escrita e desenvolvimento de programas distribuídos [Wri88]. Os primeiros têm maior flexibilidade e acesso direto às facilidades do Sistema Operacional, mas são complexos de utilizar. Os últimos têm uso mais simples, mas tendem a ocultar e impedir o acesso a muitas facilidades do Sistema Operacional.

É comum que diferentes computadores em uma LAN *(Local Area Network)* possam executar programas distintos concorrentemente, aparentando um computador grande e rápido. Quando está operando sobre um único programa de grande porte, no entanto, este grupo de computadores não provê vantagens em desempenho, no caso geral.

Uma LAN pode ser uma rota para alcançar um ambiente de processamento multiusuário distribuído, aproveitando uma característica própria das LAN's, que é a habilidade para de um ponto da rede comunicar-se com qualquer outro. O que pode ser conseguido nesta comunicação define o nível de *conectividade (connectivity)* da rede.

Qualquer rede de computadores autônomos é por si própria, potencialmente, uma máquina paralela. Sistemas de computadores em rede podem prover aos usuários diferentes níveis de *conectividade*:

nível mínimo: permite compartilhar arquivos, correio eletrônico, assim como compartilhar outros recursos de *hardware* como discos e impressoras (neste nível um programa de aplicação só pode executar em um único computador da rede).

nível superior: além das possibilidades do nível mínimo, diferentes computadores da rede podem executar diferentes porções de código que juntas fazem um programa de aplicação. Por sua vez, este nível superior pode manifestar-se de duas formas:

- cada módulo independente executa completamente sobre um nó da rede;
- uma aplicação sendo executada parcialmente sobre um nó, e parcialmente sobre outro.

O *software* atual de redes oferece aos projetistas as ferramentas necessárias para implementar sistemas que suportem a distribuição dos módulos para seu processamento em rede e a utilização compartilhada dos recursos, incluindo: sistema operacional em rede, sistema de arquivos em rede, e ferramentas de desenvolvimento de aplicações sobre a rede.

Está ficando cada vez mais importante contar com ferramentas para a programação distribuída, o que contribui para o enriquecimento do conceito de redes abertas *(open networks)*.

O conceito e a prática do processamento paralelo têm sido desenvolvidos na área dos grandes computadores, e a idéia de colocar poder de processamento paralelo sobre um *desktop* é ainda muito nova. As estações de trabalho *(workstations)* paralelas e as redes de estações de trabalho convencionais têm um potencial enorme para isto.

É necessário desenvolver árduo trabalho em relação à *conectividade*, uma vez que este aspecto está limitando a produtividade das máquinas *desktop* nos modelos de computação existentes. Hoje as máquinas requerem otimização no acesso e uso da rede, e conseqüentemente as estações de trabalho estão caminhando para um ponto onde uma aplicação possa ser distribuída ao longo da rede, ao invés de residir unicamente dentro de uma estação [Run88].

1.2 Exploração da Concorrência

A maneira de conseguir explorar a concorrência é altamente dependente do problema a ser resolvido, da arquitetura, e da organização do sistema de processamento que vai ser empregado. Por sua vez, os recursos de *software* disponíveis (como sistema operacional e linguagens) também influem grandemente no que pode ser conseguido neste sentido [Hes87]. Entretanto, é possível apresentar, em termos gerais, os requisitos que devem ser satisfeitos para conseguir explorar a concorrência:

- **Segmentação do problema em tarefas**: primeiro é necessário verificar se o problema tem paralelismo; se isto não for possível, então não há como explorar concorrência. Uma vez

identificado o paralelismo deve-se dividir a tarefa em processos paralelos, tomando decisões quanto à *granularidade.*

- **Arbitragem dos recursos e das tarefas:** para uma arbitragem adequada é desejável:
  - **informações atualizadas sobre os recursos:** manter um *pool* dos recursos existentes e de suas características;
  - **informações atualizadas sobre as tarefas:** manter um retrato das tarefas em execução no ambiente distribuído e daquelas sobre o sistema base (não ligadas à ferramenta de distribuição);
  - **identificar os pedidos de recursos feitos pelas tarefas:** estes podem ser iniciados assíncronamente por estímulos internos e/ou externos;
  - **determinar os recursos que podem satisfazer um pedido:** garantir que cumpram requerimentos imprescindíveis e, opcionalmente, outras características desejáveis;
  - **testar o estado dos recursos:** determinar seu nível de disponibilidade;
  - **re-arbitragem:** possibilidade de interromper tarefas já iniciadas e migrá-las para outros sistemas [The86].

- **Alocação dos recursos às tarefas:** distribuir efetivamente os recursos entre as tarefas, obtendo uma configuração destas nos nós de processamento. Esta alocação pode ser estática ou dinâmica [Fat83, Bar86], mas a existência de re-arbitragem exige alocação dinâmica.

- **Coordenação e interação entre as tarefas:** sustenta-se nos mecanismos de comunicação disponíveis. Pode ser muito simples, limitada a dependências externas (ordem da seqüência entre as tarefas, subordinação de umas ao sucesso de outras), ou pode estender-se até a comunicação entre processos em execução.

1.3 Escopo do Trabalho

O trabalho aqui apresentado tem como plataforma de *hardware* um conjunto de *workstations* da *Sun*[1] interligadas através de uma rede **Ethernet**. Toda a rede é gerenciada pelo Sistema Operacional *SunOS*[2] [Sun90a].

O Sistema Gerenciador de Processamento Cooperativo (SGPC) desenvolvido situa-se na área de Ambientes de Programação, relacionando-se com o desenvolvimento de um ambiente que facilite ao usuário a exploração da capacidade de paralelismo inerente à arquitetura de rede.

O objetivo do trabalho de dissertação consiste no desenvolvimento de um sistema para distribuir módulos (que residem em disco como arquivos executáveis) entre estações de trabalho, possibilitando a utilização do tempo ocioso (*idle*) destas. Com a utilização desta ferramenta pode-se conseguir um *"processamento cooperativo"* na rede; não só a utilização compartilhada de periféricos

[1] Sun Microsystems, Inc.
[2] Sun Operating System

(como discos e impressoras), mas também, e fundamentalmente, da capacidade de processamento das estações.

Segmentar um problema em tarefas possíveis de serem executadas concorrentemente é altamente dependente da aplicação. Isto faz com que seja muito difícil, ou impossível, automatizá-lo completamente de uma maneira razoável e eficiente. O SGPC pressupõe que o usuário tem uma idéia clara do paralelismo em seu trabalho e da maneira de obter o algoritmo do mesmo. O SGPC também pressupõe que os módulos possíveis de executar em paralelo residem em arquivos executáveis independentes.

Os usuários poderão iniciar módulos independentes que serão migrados, de maneira transparente e dinâmica, para serem executados em algumas das máquinas no conjunto disponível, de preferência naquelas com maior disponibilidade de recursos para manipular cada tarefa.

Com a utilização do SGPC passa-se de um ambiente de estações de trabalho distribuídas a um pseudo ambiente de multiprocessadores, fazendo uso, de maneira transparente de uma capacidade de processamento distribuída.

O fato dos módulos serem executados local ou remotamente, bem como a existência de migrações são transparentes para as aplicações. O SGPC pode decidir que a "melhor" estação disponível para executar um módulo seja a própria estação onde o processo foi ativado, implicando numa chamada local. Ainda que a rede esteja inoperante ou não existam estações disponíveis para execução cooperativa, a tarefa do usuário deve prosseguir de modo normal (obviamente, com todos os módulos sendo alocados à estação local).

Segundo a linha de desenvolvimento seguida pelo *SunOS* para implementar os serviços de rede, o SGPC baseia-se no desenvolvimento de um Servidor. Este Servidor de Processamento (ServProc) foi implementado como um *"daemon"*[3]; uma cópia deste deve existir em cada estação participando do pseudo ambiente de multiprocessadores.

Os Servidores nas diversas estações operam concorrentemente, trabalhando para conseguir uma distribuição que equilibre o uso das estações de trabalho, isto é, para distribuir o processamento de maneira que se faça uma boa utilização dos recursos, diminuindo o tempo total de execução das aplicações. O funcionamento integrado de vários Servidores de Processamento (ServProcs) constitui o Sistema Gerenciador de Processamento Cooperativo.

O SGPC encarrega-se da arbitragem e da alocação de recursos a módulos, e para tanto:

- **mantém informações atualizadas sobre as estações na rede**: algumas dessas informações são fixas, determinadas pela configuração (por exemplo, a arquitetura das máquinas), e outras são variáveis no tempo (quantidade de processos executando, medida da ociosidade e disponibilidade de cada máquina, entre outras);
- **mantém informações atualizadas sobre os módulos em execução**: algumas dessas informações podem ser fixas, conhecidas no momento de iniciar a tarefa (localização do arquivo, parâmetros da linha de comandos), e outras são variáveis no tempo (como o estado da execução do módulo, e o tempo de execução);

[3]Processo servidor que fica residente e trabalha em estreita vinculação com o *kernel*

- **determina e aloca as estações hospedeiras**: estas estações devem cumprir os requisitos de capacidade de processamento dos pedidos;
- **re-aloca estações para os módulos**: em caso de condições excepcionais que justifiquem a suposição de que ocorreu uma queda na estação hospedeira, ou outras decisões de novos encaminhamentos.

O SGPC supõe, nesta sua primeira versão, que em tempo de execução não há comunicação entre os módulos migrados. O sistema não oferece mecanismos para o intercâmbio de informações entre os módulos em execução em diferentes máquinas.

1.3.1 Apresentação dos Módulos do Sistema

Os serviços prestados pelo ServProc podem ser usufruídos pelas aplicações usuárias em dois níveis diferentes:

- Sistema Gerenciador de Processamento Cooperativo (SGPC);
- Sistema Gerenciador de Processamento Cooperativo Estendido (SGPC Estendido);

No primeiro caso, a aplicação faz uso do SGPC interagindo diretamente com o Servidor de Processamento Local[4] segundo as convenções do Modelo Cliente/Servidor (Seção 2.1.1). Neste caso a aplicação é o Cliente e deve conhecer o protocolo para interagir com o ServProc. O Apêndice C contém uma listagem deste protocolo, e na Seção 3.1 são apresentados seus serviços.

Embora uma aplicação possa comunicar-se diretamente com um Servidor Remoto, isto não é recomendado. A comunicação deve ser sempre com o Servidor Local e este, se necessário, comunica-se com o Servidor Remoto.

No segundo caso, SGPC Estendido, o SGPC foi acrescentado de uma interface de *maior* nível de abstração, chamada de *Highest Level do RPC* na documentação da *Sun*. Alguns dos Servidores padrões do Sistema Operacional em rede da *Sun* apresentam serviços com este tipo de interface (conforme seção 3R do Manual de Referência). Esta interface poupa as aplicações de conhecerem as convenções do Modelo Cliente/Servidor, permitendo que interajam com rotinas simples em uma biblioteca, e com um *front end C*.

Para este SGPC Estendido foi desenvolvido um conjunto de rotinas estruturadas para a comunicação com a ferramenta, a **InterfPC**. Estas rotinas ficam em uma biblioteca à disposição dos usuários, e através delas poderão ser solicitados os serviços de *Processamento Cooperativo*. Cada aplicação que use o SGPC Estendido incluirá dentro de seu código objeto, através da ligação, uma cópia da *InterfPC*. O Apêndice D apresenta uma listagem desta *interface*, e na Seção 3.8 é detalhada sua operação.

A Figura 1.1, ilustra as interações entre os diferentes componentes do SGPC em operação. Elas incluem interações entre ServProcs e interações entre aplicações e Servidores de Processamento.

[4] *ServProc* residente na Estação Local

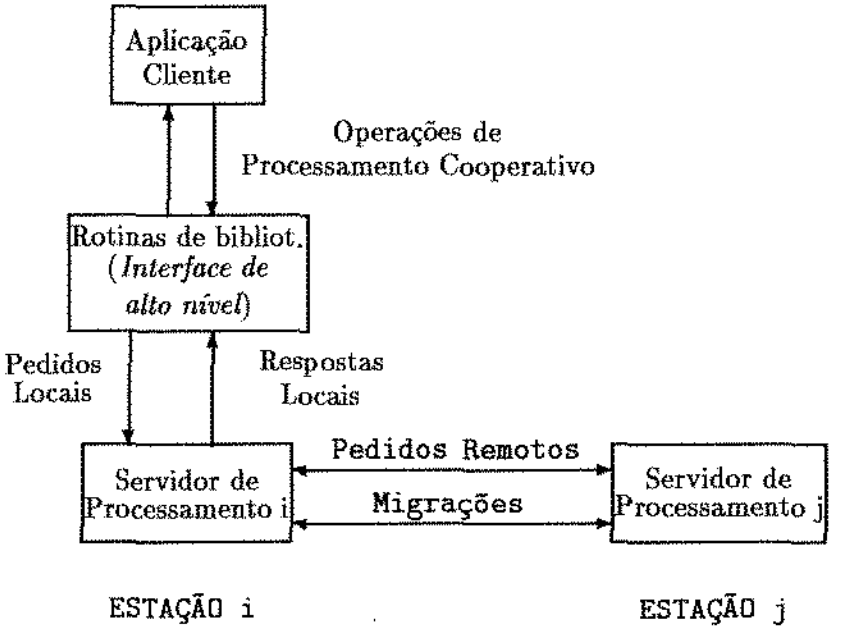

Figura 1.1: Interações entre Servidores de Processamento e Aplicações

O Capítulo 4 inclui um exemplo de aplicação, um programa *make* voltado para realizar a compilação de um conjunto de módulos ao longo da rede. Este *make distribuído* encaminha as compilações para diferentes estações, conseguindo que sejam realizadas em paralelo, com a conseguinte economia de tempo.

Cada aplicação em uma estação terá a sua disposição um **Servidor de Processamento** (que poderá usar diretamente), e um **conjunto de rotinas** em uma biblioteca, através das quais terá acesso indireto aos serviços de *Processamento Cooperativo*.

Ao longo do trabalho chamam-se de *operações* as facilidades oferecidas pela InterfPC (Seção 3.8), e de *serviços* as facilidades oferecidas pelo ServProc (Seção 3.1).

Para cada operação descrita na Seção 3.8, deve-se ter um procedimento remoto do ServProc que a implementa. Todas as operações disponíveis para a aplicação através da *InterfPC* são mapeadas em serviços do protocolo do ServProc.

O funcionamento integrado de vários ServProcs (o que constitui o SGPC), exige que interajam. Por este motivo no protocolo do ServProc estão incluídos além dos serviços propriamente destinados às aplicações usuárias, serviços para a interação entre ServProcs. Uma descrição da arquitetura dos ServProcs, e de sua interação no SGPC é apresentada no Capítulo 2.

O SGPC desenvolvido visa garantir um melhor aproveitamento da capacidade de processamento esparsa entre as estações de trabalho da rede em que opera. O SGPC colabora na exploração de uma *capacidade de processamento distribuída* diminuindo o "isolamento" entre as diferentes estações na

rede e conseguindo aumentar o nível de *conectividade* da mesma.

O trabalho exigiu um estudo bastante profundo do sistema operacional e dos serviços da rede de estações da *Sun* como um todo, pois as informações que o SGPC tem que manipular para alcançar o seu objetivo são muito variadas e amplas. A implementação do sistema faz uso intensivo de *System Calls*, Funções de Biblioteca, *Network Information System*, e *Remote Procedure Calls*, entre outras [Sun89, Sun90a, Sun90b].

Capítulo 2

Arquitetura do SGPC

Neste capítulo apresentam-se detalhes da arquitetura do Sistema Gerenciador de Processamento Cooperativo (**SGPC**). O capítulo enfoca as técnicas utilizadas, descreve problemas, e apresenta as soluções encontradas.

Na fase de projeto do Sistema foram definidos:

- os serviços do Sistema Operacional que suportam o SGPC;
- os módulos e submódulos da implementação, e suas interfaces;
- as estruturas de dados que guardam as informações das estações e dos módulos, em cada servidor;
- as estruturas de dados utilizadas pelos usuários nas suas interações com o Sistema;
- as restrições impostas aos módulos candidatos a migrações;
- as características do contexto de execução dos módulos;
- o caminho a ser percorrido pelos pedidos e pelas migrações dos módulos;
- a política para a interação entre o Cliente e o Servidor Local;
- a política para a interação entre Servidores de Processamento.

Estes pontos serão abordados ao longo do trabalho. Uma breve explicação das características principais dos serviços do *SunOS* utilizados como plataforma computacional aparece na Seção 2.1. A Seção 2.2 refere-se ao segundo tópico da lista acima, incluindo as interrelações entre os módulos.

O momento e a maneira de extrair e propagar os dados no SGPC são abordados na Seção 2.3, que trata da organização e do fluxo das informações. Nela são apresentados também os caminhos percorridos pelos pedidos de execução de módulos, e as transações e interações que isto pressupõe entre os componentes do SGPC.

Na Seção 2.4 é explicado em detalhe o Ciclo de Vida que percorrem os módulos submetidos às migrações, esclarecendo os diferentes estágios por que passam.

É deixada para o Capítulo 3 a apresentação dos detalhes das estruturas de dados manipuladas pelo Sistema (as quais o usuário deve conhecer para conseguir interagir diretamente com o mesmo), assim como das estruturas de dados internas ao ServProc.

Ao longo do presente capítulo são apresentadas ainda as políticas para as interações entre os diferentes ServProcs residentes nas distintas estações.

2.1 Plataforma Computacional

O SGPC foi desenvolvido sobre uma rede de estações de trabalho *(workstations)* da **Sun** interligadas com uma rede **Ethernet**, e com o Sistema Operacional **SunOS (UNIX)** como plataforma de programação [Sun90a, Sun90b]. A programação foi realizada nas linguagens C [Ker78] e RPC[1] [Sun90a]. Esta última foi utilizada na descrição do protocolo de serviço RPC aceita pelo compilador de protocolos *rpcgen* também empregado.

A combinação *hardware-software* da rede *Sun* provê uma plataforma flexível para a integração das diferentes máquinas e para a integração de novos *softwares*, sem interferir no já existente. O Sistema Operacional, entre outras coisas, oferece um conjunto extensível de protocolos para o intercâmbio de dados.

Os serviços de rede do *SunOS* que basicamente suportam o SGPC, e que devem estar disponíveis para sua instalação e uso são os seguintes:

RPC (Remote Procedure Call): Biblioteca de procedimentos que podem ser chamados nos programas. Estes serviços permitem associar um processo *(caller)* com outro processo *(server)* em outro sistema. O processo *server* executa para o *caller* uma chamada de procedimento;

XDR (eXternal Data Reference): Especificação para a representação *portável* e padronizada dos dados;

portmapper : Meio utilizado por todos os serviços baseados em RPC para registrar, de forma padronizada, sua existência em uma máquina e sua correspondência com os canais de comunicação. Padroniza ainda o acesso do Cliente, ajudando-o a localizar o número de porta dos programas RPC em que está interessado;

NIS (Network Information Service): Serviço de informações da rede com os dados replicados e distribuídos de sua configuração. Permite o acesso a estas informações de maneira independente das localizações relativas dos clientes e dos servidores;

NFS (Network File System): Servidor que permite montar diretórios através da rede e tratá-los como se fossem locais; permite compartilhar arquivos e diretórios, provendo acesso remoto transparente aos sistemas de arquivos, mesmo sobre sistemas heterogêneos.

[1] extensão da linguagem XDR, e muito similar a C

Novos serviços da rede são acrescentados ao *SunOS* através de processos servidores. O Servidor de Processamento (ServProc) desenvolvido neste trabalho é também acrescentado ao sistema da rede por este mecanismo.

As Chamadas Remotas de Procedimentos (RPCs), utilizadas na implementação do Sistema, são um paradigma de comunicação de alto nível que permite às aplicações da rede realizar chamadas de procedimentos de uma classe especializada, ocultando os detalhes dos mecanismos subjacentes na rede. Estes procedimentos estão preparados para serem executados em máquinas remotas da própria rede. Ao programar com RPC, os programas são projetados para executar dentro de um modelo Cliente/Servidor (Seção 2.1.1).

Os serviços da XDR são usados implicitamente pelos do RPC para fazer as transferências dos dados entre as estações remotas em um formato padrão através da rede. O ambiente RPC/XDR é inerentemente extensível; novos serviços da rede podem ser facilmente acrescentados construindo-se sobre estas bases.

O serviço de *portmapper* é empregado pelos ServProcs para se auto-registrarem como serviço residente na estação local (3.2). Isto é feito como parte da sua auto-instalação; nesta instalação, o próprio ServProc também se auto-configura.

O código do SGPC não contém quaisquer informações dependentes da configuração de *hardware*, sendo que está isolado das características específicas da rede local, e das estações que a compõem. No Sistema Operacional *SunOS* existem os mecanismos necessários para encontrar o conjunto de estações ligadas na rede local. Estes mecanismos são usufruídos pelo SGPC que, por sua vez, os faz disponíveis (indiretamente) para suas aplicações usuárias, através de seus serviços.

Como conseqüência da generalidade do projeto do SGPC, as aplicações dos usuários podem também manter-se isoladas da configuração específica da rede local e das estações em particular.

Os serviços do **NIS** são muito empregados ao longo do SGPC, com o objetivo de conhecer dinamicamente a configuração da rede e das estações interligadas.

Os serviços de **NFS** são importantes nas estações de destino, para garantir que as referências ao sistema de arquivos do módulo em execução sejam satisfeitas da mesma forma que seriam em sua estação origem.

2.1.1 Modelo Cliente/Servidor

As interações com o ServProc seguem o paradigma mais comumente utilizado na construção de aplicativos distribuídos: o Modelo Cliente/Servidor. Nesse modelo, processos clientes solicitam a execução de serviços a processos servidores, os quais recebem requisições, e devolvem resultados aos processos clientes. Isto implica no estabelecimento de um protocolo de comunicação entre esses processos. No nosso caso, o Servidor também pode assumir o papel de Cliente. Isto acontece quando um Servidor se faz Cliente de outro Servidor nas interações entre Servidores de Processamento.

A implementação do modelo Cliente/Servidor, no caso, é feita através do mecanismo de RPCs. O processo designado Cliente realiza uma chamada de procedimento segundo as convenções normais,

e como conseqüência desta chamada o Sistema Operacional envia uma requisição para um outro processo chamado Servidor. O Servidor, em geral, encontra-se em execução em outra estação de trabalho na rede, mas como caso particular pode encontrar-se na própria estação do Cliente.

A comunicação entre o Cliente e o Servidor é feita através dos parâmetros e do resultado do procedimento remoto. O Sistema Operacional constrói e envia remotamente mensagens com estes dados para efetivar a comunicação.

Nas Figuras 2.1 e 2.2 mostra-se esquematicamente o Modelo Cliente/Servidor em um computador simples e em um Sistema Distribuído, respectivamente.

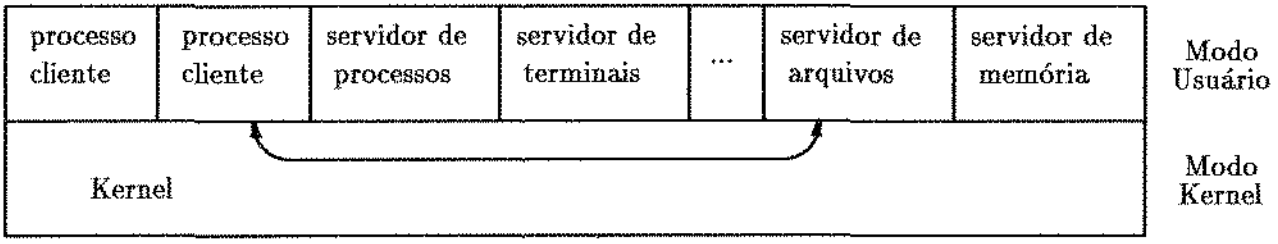

Figura 2.1: Modelo Cliente/Servidor em um Monoprocessador

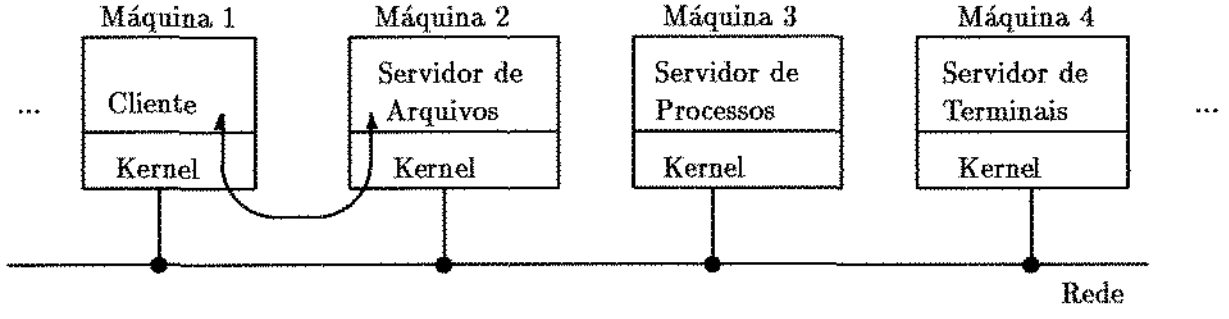

Figura 2.2: Modelo Cliente/Servidor em um Sistema Distribuído

Sob o ponto de vista do Cliente, enviar uma mensagem a um Servidor e ficar esperando pela resposta é um processo muito similar ao de chamar um procedimento e esperar que este finalize. Alguns Sistemas Operacionais Distribuídos exploram esta similaridade fazendo com que a comunicação entre processos adquira a forma de RPCs. A beleza desta ótica faz com que nem o Cliente, nem o procedimento Servidor tomem conhecimento de que estão sendo usadas mensagens. Eles vêem chamadas comuns a procedimentos locais.

As convenções de uma chamada a um procedimento remoto são semelhantes às de uma chamada

a um procedimento local. Esta semelhança está na seqüência lógica de controle entre os dois processos (*caller* e *called*), mas a ligação e o intercâmbio de dados (parâmetros e resultados) são conseguidos de maneira diferente.

Com RPC, o Cliente faz uma chamada de procedimento, a qual provoca o envio de requisições ao Servidor, na medida que seja necessário. Quando uma requisição chega, o Servidor chama uma rotina *dispatch* que escalona os pedidos e dispara o procedimento encarregado de realizar o serviço que foi requisitado. Finalmente, envia-se de volta a resposta, e a chamada de procedimento retorna ao Cliente.

No SGPC todas as comunicações entre os Servidores de Processamento e, em cada estação, entre as aplicações clientes e seus ServProcs são feitas através de RPCs. Este modelo garante que as diferentes chamadas não tenham que preocupar-se com o fato do outro processo na comunicação estar ou não na mesma máquina.

Usualmente o Cliente (produto de uma chamada remota) fica bloqueado até que o Servidor envie uma resposta. Entretanto, o procedimento remoto pode impedir o bloqueio do Cliente, indicando que não enviará resposta.

As seções a seguir neste capítulo apresentam os diferentes aspectos do projeto do SGPC.

2.2 Interação entre os Módulos do *SGPC*

O SGPC é formado por um conjunto de um ou mais Servidores de Processamento executando em diferentes estações na rede local em questão. A quantidade de ServProcs cooperando no SGPC pode variar dinamicamente.

Nesta seção discutiremos estes Servidores, com ênfase em sua funcionalidade e interfaces. A implementação dos ServProcs será analisada detalhadamente no Capítulo 3.

Durante a operação do SGPC deve existir uma cópia do módulo Servidor de Processamento residente em cada estação, para prover os Serviços do Sistema (não existe hierarquia entre as diferentes cópias). Cada servidor deve instalar-se, e ficar executando em cada uma das estações, coletando informações, e pronto para servir como *"Servidor de Processamento"*. O processo de instalação das cópias dos ServProcs é abordado na Seção 3.2.

Cada Servidor de Processamento atende a pedidos locais, gerados a partir de aplicações clientes na própria estação, e a pedidos remotos, provenientes de outros Servidores de Processamento executando em outras estações (Seção 3.1).

Todos os Servidores de Processamento têm capacidades similares; qualquer um dispõe de um retrato das condições de todas as estações e pode fazer uma avaliação do uso destas. Estas informações possibilitam a escolha da estação para processar o módulo (Seção 3.3).

A *migração* de módulos acontece entre duas estações em cooperação através do Servidor de Processamento. Diz-se que a "**estação origem**" do módulo (isto é, a que recebe localmente o pedido de execução) *"exporta"* o módulo. Analogamente, diz-se que a "**estação hospedeira**" ou *estação de destino* (a que recebe e executa o módulo) *"importa"* o módulo. Portanto, um módulo que migra é exportado pela estação origem e importado pela estação hospedeira. Como caso particular a **estação de origem** e a **estação de destino** podem coincidir, mas o usuário não precisa ter conhecimento deste fato.

Os pedidos das aplicações, dependendo da sua natureza, implicarão na realização de algumas ou de todas as transações abaixo enumeradas:

- pedido da aplicação ao ServProc Local;
- transformação do pedido em requisição entre ServProcs Remotos;
- lançamento da execução de um módulo em um ServProc Remoto;
- devolução de resultado pelo módulo requisitado ao módulo que fez a requisição.

A Figura 2.3 apresenta um modelo protótipo para esquematizar as transações entre os diferentes módulos no SGPC em funcionamento.

Estação de Origem		Estação de Destino (hospedeira)	
Aplicação (inclui InterfPC)	Servidor Local (ServProc)	Servidor Remoto (ServProc)	Módulo Migrado
1	2	3	4

Figura 2.3: Modelo Protótipo para Transações entre Módulos

A coluna 1 representa o código objeto de uma aplicação usuária que faz uso do SGPC, atuando como seu Cliente. As interações destas aplicações serão apenas com o módulo do ServProc Local representado na coluna 2.

Os módulos das colunas 1 e 2 residem na Estação Local (de Origem) do ponto de vista da aplicação.

A coluna 3 representa o código objeto de um ServProc residente em uma Estação Remota (de Destino) do ponto de vista da aplicação.

A migração de módulos dos usuários entre estas duas estações, e qualquer intercâmbio de informações entre elas, será baseado, em interações entre os dois ServProcs (Local e Remoto) representados nas colunas 2 e 3.

A quarta e última coluna é usada para representar o código objeto em execução de um módulo migrado para a estação com o referido ServProc Remoto; para o módulo migrado esta é a Estação de Destino. O lançamento da execução do módulo migrado, assim como o recebimento do sinal gerado por seu término, ficam sob a responsabilidade do ServProc da coluna 3.

As transações entre os módulos representados nas colunas 1 e 2, e entre os módulos representados nas colunas 3 e 4 são chamadas de Transações Locais. As transações entre estações de trabalho distintas, isto é, entre ServProcs representados nas colunas 2 e 3 são chamadas de Transações Remotas.

Quando, para a satisfação de um pedido, realiza-se alguma Transação Remota, dizemos que a sua satisfação foi Remota; caso contrário, a satisfação foi Local. Os pedidos feitos pelas aplicações podem satisfazer-se Local ou Remotamente, segundo a sua natureza. Seguem-se alguns exemplos de pedidos e as transações que provocam.

2.2.1 Exemplo de Serviço de Satisfação Local

Se uma aplicação (coluna 1) está interessada em conhecer o conjunto de estações na rede (exemplo da Figura 2.4), ela primeiro deve ter estabelecido comunicação com o ServProc residente na sua estação (Local), fazendo-se seu Cliente. Uma vez estabelecida a ligação, a aplicação pode fazer um pedido ao Servidor Local (coluna 2) através de uma chamada remota de procedimento:

... clnt_call(cl_user, LISTAHOSTS, ...); ...

com esta chamada invoca-se o serviço de nome LISTAHOSTS() (Seção 3.1.2), que localiza as informações requisitadas em estruturas de dados locais ao próprio Servidor, e as devolve à aplicação Cliente.

Finalmente a aplicação deve destruir o *handler* que a associa como Cliente do Servidor.

Neste caso, para conseguir satisfazer o pedido foram suficientes transações locais. □

2.2.2 Exemplo de Serviço de Satisfação Remota

Quando uma aplicação (coluna 1) deseja executar um programa fazendo uso do Processamento Cooperativo (exemplo da Figura 2.5), ela faz um pedido ao ServProc Local (coluna 2) (após estabelecer sua ligação como cliente deste), chamando o procedimento **EXECMOD()** (Seção 3.1.3) do ServProc Local:

... clnt_call(cl_user, **EXECMOD**, ...); ...

este procedimento, depois de selecionar a estação hospedeira para o módulo, contacta o ServProc Remoto (coluna 3) correspondente (se fazendo previamente seu cliente), e chama seu procedimento remoto EXECREMMOD() (Seção 3.1.3):

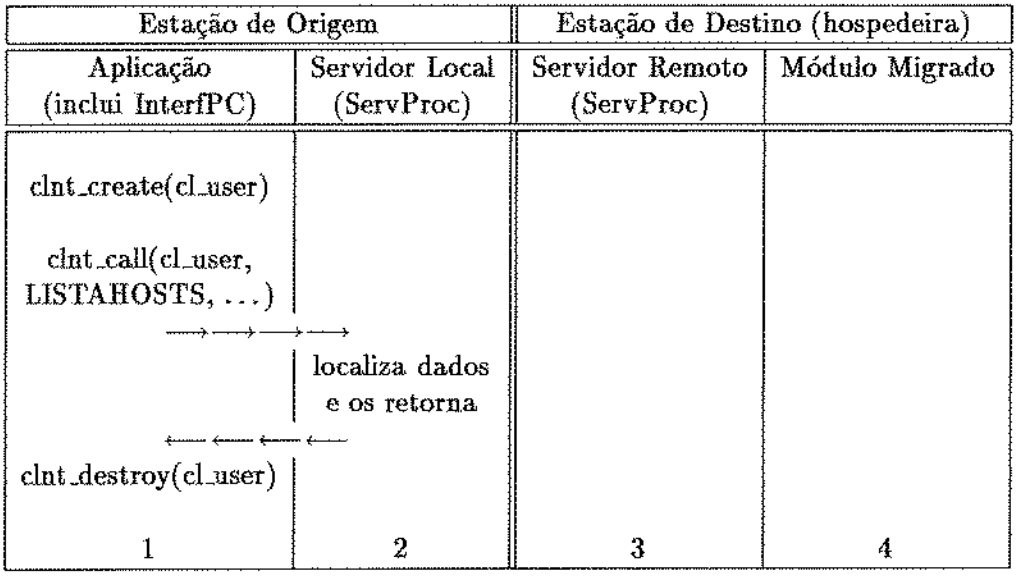

Figura 2.4: Pedido de Serviço de Satisfação Local

```
... clnt_call(cliente, EXECREMMOD, ...); ...
```

este último é encarregado de pedir a execução do programa (coluna 4) na sua estação, e de devolver ao cliente remoto (coluna 2) a confirmação da iniciação do módulo.

O serviço **EXECMOD()** do servidor da estação origem (coluna 2), por sua vez, retorna à aplicação (coluna 1) o identificador do módulo recém iniciado.

Tanto o Servidor na Origem quanto a aplicação devem destruir os *handlers* clientes respectivos.

Neste caso, para conseguir satisfazer o pedido foram necessárias tanto transações locais quanto remotas. □

No primeiro caso apresentado a satisfação do pedido do usuário é conseguida totalmente na estação local, embora esta satisfação envolva informações de estações remotas. No segundo exemplo, pelo contrário, a satisfação do pedido do usuário é conseguida remotamente, e para isto, são realizados todos os tipos de transações listados anteriormente. Os detalhes destas transações são apresentados no Capítulo 3.

Existem ainda requisições entre Servidores Remotos originadas primariamente nos servidores, e que só envolvem transações do tipo de requisições remotas. Isto ocorre, por exemplo, quando se deseja verificar a Identidade de um ServProc Remoto, e quando se informa a uma estação origem do término da execução de um módulo migrado a partir dela.

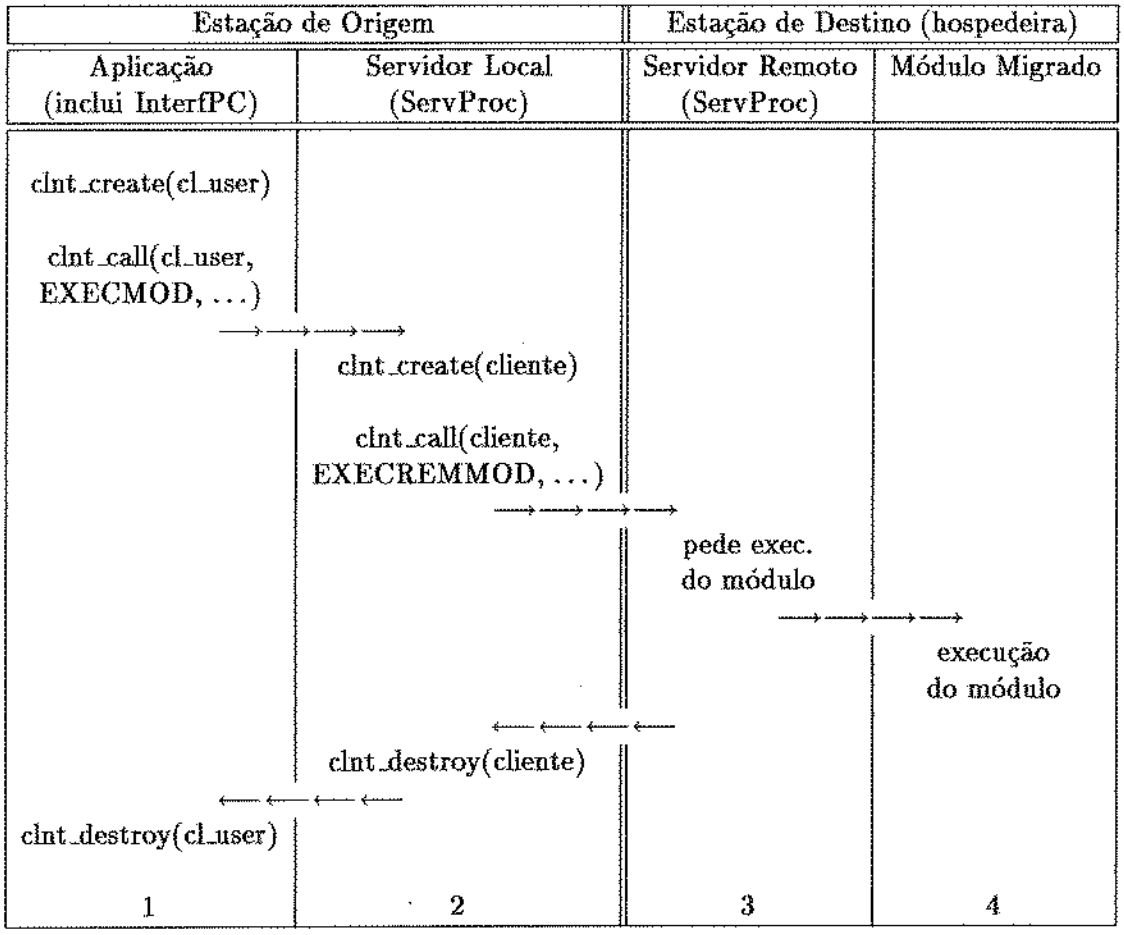

Figura 2.5: Pedido de Serviço de Satisfação Remota

2.3 Organização e Fluxo das Informações

Cada Servidor de Processamento deve manter informações atualizadas sobre os recursos existentes na estação local, sobre a utilização atual destes recursos, e sobre os módulos em execução na estação. Além disso, o Servidor deve possuir conhecimento sobre os módulos em execução em outras estações a seu pedido.

Na escolha da estação para executar um módulo específico em determinado momento, é necessário avaliar a carga de trabalho das diversas estações. Os Servidores devem também acompanhar a execução dos módulos migrados. Para isto, o servidor armazena dados em tabelas que representam o estado das estações na rede e dos módulos em execução. A manutenção destas tabelas exige a transferência e distribuição de informações relativas ao estado de cada estação participando da cooperação.

Cada Servidor opera tanto como um produtor quanto um consumidor de informações de estado. Como produtor destas informações, o Servidor interroga o estado da Estação Local e constrói mensagens de estado que são distribuídas pela rede. Como consumidor das informações de estado, o Servidor requisita e espera o recebimento de mensagens com informações de estado de outros Servidores de Processamento remotos, validando-as e registrando-as em estruturas de dados locais.

Portanto, as informações de estado são coletadas de maneira descentralizada pelo Servidor Local, e precisam ser centralizadas para possibilitar a tomada de decisões, assim como para satisfazer pedidos de informações de estado dos usuários finais. As informações coletadas pelo SGPC foram catalogadas nas seguintes categorias:

informações fixas das estações: dados relativos às estações que não se modificam durante a operação do Servidor (informações estáticas). São dados coletados pelo Servidor Local durante sua iniciação, e enviados a outros servidores sob sua requisição;

informações variáveis das estações: dados relativos às estações, e que mudam em função do uso que se esteja fazendo delas (informações dinâmicas). Estes dados são periodicamente coletados na estação local e distribuídos às demais estações;

informações fixas dos módulos: dados relativos aos módulos migrados que são conhecidos no momento em que se pede a sua execução (informações estáticas). Estes dados são obtidos na estação de origem do módulo e enviados à estação de destino no momento da exportação;

informações variáveis dos módulos: dados relativos aos módulos migrados e que se modificam ao longo do tempo em função de seu progresso (informações dinâmicas). Estes dados são coletados localmente na estação de destino do módulo e enviados a sua estação de origem.

As aplicações usuárias do SGPC podem ter acesso às informações das estações e dos módulos através de serviços que fornecem estas informações (Seções 3.1.2 e 3.1.4).

Nesta primeira versão, optou-se por coletar e enviar sob demanda as informações de caráter dinâmico. Na fase do projeto do sistema foi avaliada uma variante, na qual estas informações pudessem ser enviadas, antecipadamente ao momento em que fossem requisitadas. Nesta última

variante, o envio das informações variáveis seria feito com uma freqüência periódica, disparada por um alarme. Os testes realizados evidenciam que o tráfego na rede aumenta notavelmente, razão pela qual, na versão atual, foi adotada a variante de envio apenas sob requisição; outros fatores que justificam essa opção são sua simplicidade, e a possibilidade de adoção, no futuro, de outra variante sem prejuízos.

2.4 Ciclo de Vida dos Módulos Submetidos a Migração

Como foi mostrado no segundo exemplo da Seção 2.2, a satisfação do pedido de execução de um módulo pode ser conseguida remotamente, e neste caso são realizados todos os tipos de transações enumerados. O percurso destas transações passa pela aplicação, pelo ServProc Local e possivelmente por um ServProc Remoto. Retomando aquele exemplo:

clnt_call(cl_user, **EXECMOD**, ...)
⇓
clnt_call(cliente, **EXECREMMOD**, ...)
⇓
fork()
⇓
execve()

Acrescentamos acima as chamadas às primitivas *fork()* e *execve()* do sistema operacional, que possibilitam a execução do módulo do usuário.

Durante a ocorrência destas transações (pedido da aplicação ao ServProc Local, transformação do pedido em uma requisição entre ServProcs Remotos, lançamento da execução do módulo migrado na Estação Remota) o módulo passa por diferentes estágios. A Figura 2.7 mostra um diagrama de estado representando o Ciclo de Vida dos Módulos. Neste diagrama, cada nó representa um dos possíveis estados nos quais o módulo pode encontrar-se, e os arcos representam as transições ou mudanças de estado. Os estados no Ciclo de Vida de um módulo são os seguintes:

não_migrado: estado inicial. Mantém-se do instante em que o ServProc Local aceita o pedido de sua execução até o instante em que este contacta o ServProc Remoto escolhido, enviando-lhe o pedido de execução remota. Neste estado o módulo é reconhecido apenas na Estação Origem;

migrado: estado intermediário, transitório. Marca o início da existência do módulo também na Estação de Destino. Mantém-se do instante em que se faz o pedido de execução remota por parte do Servidor Local até o recebimento das primeiras informações de estado, indicando que o módulo está em andamento;

em_progresso: estado intermediário que vai do recebimento das primeiras informações que indicam o início da execução do módulo na Estação de Destino até o recebimento da informação que assinala o término dessa execução;

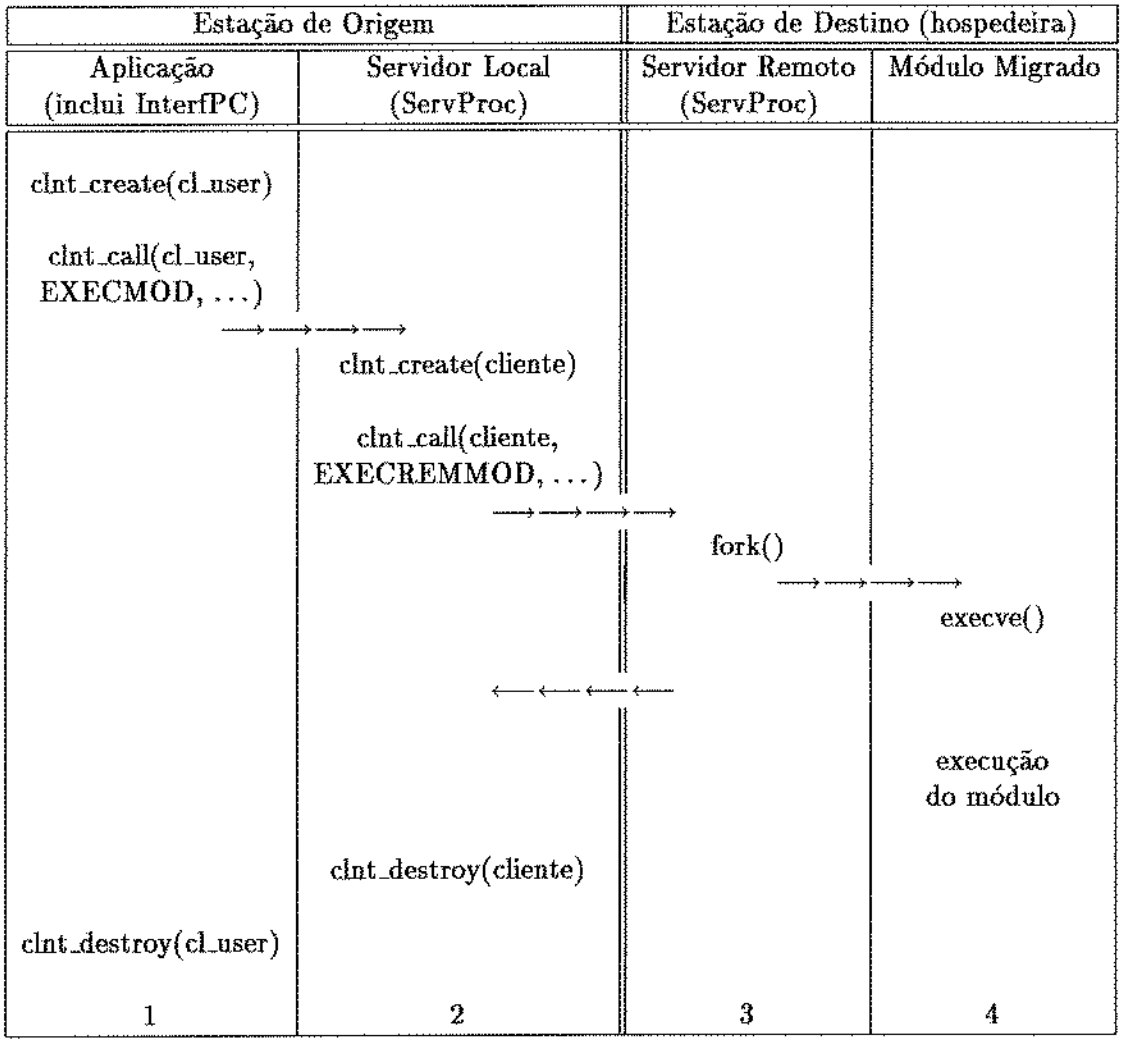

Figura 2.6: Execução Remota de um Módulo Migrado

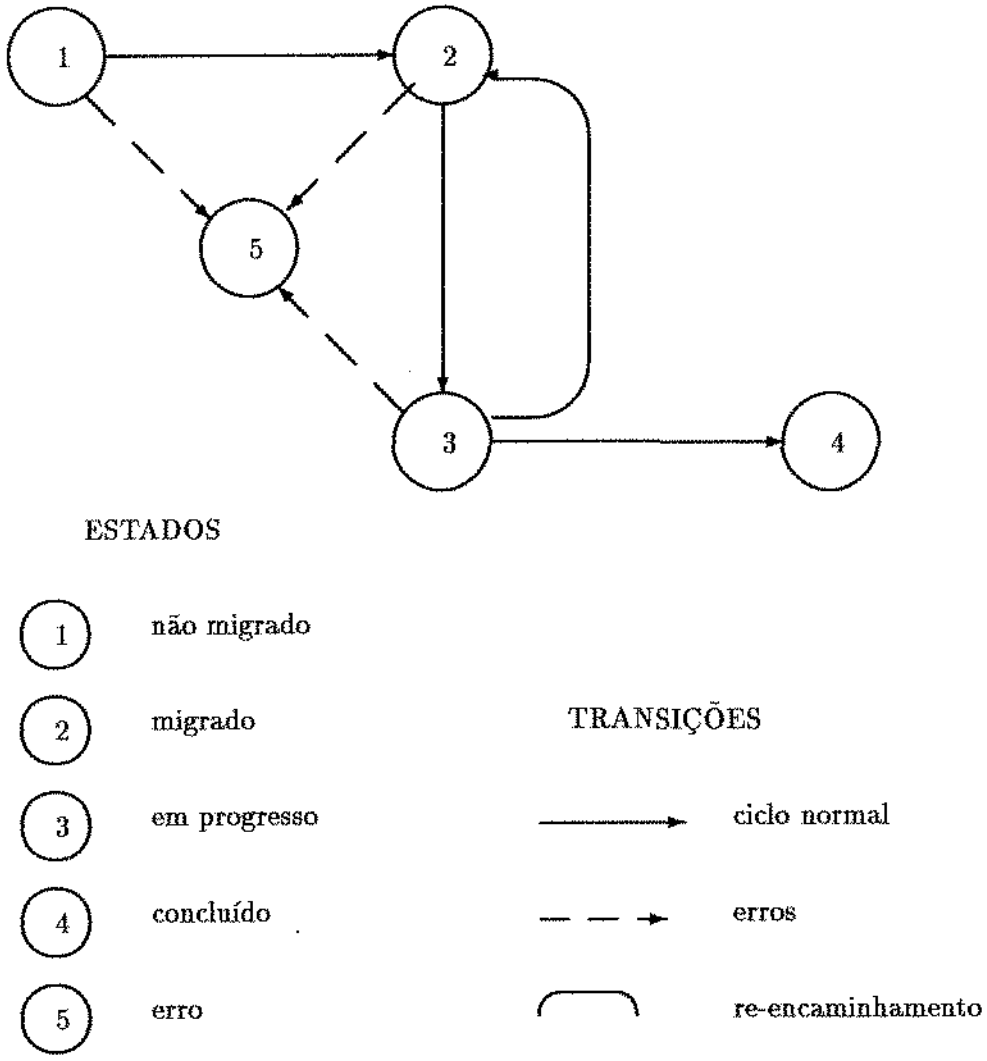

Figura 2.7: Ciclo de Vida dos Módulos

concluído: estado final. Estabelece-se a partir do momento que se recebe a informação do ServProc hospedeiro indicando o fim da execução do módulo e dura até que o módulo desapareça totalmente da jurisdição do SGPC, isto é, até finalizar seu Tempo de Vida Estendido;

erro: estado final, excepcional (fora da seqüência normal do Ciclo de Vida de um Módulo). Indica que aconteceu alguma condição de erro com a execução desse módulo, e que foi perdido o controle sobre ele. Por exemplo, se aconteceu uma queda no ServProc ou na Estação que estavam hospedando o módulo.

A seqüência normal de estados no Ciclo de Vida de um módulo aparece ilustrada na Figura 2.7 pelas transições representadas com as flechas de traço reto contínuo (⟶). A transição do estado (3) ao (2) traçada como arco contínuo aconteceria unicamente no caso de repetidos encaminhamentos do módulo.

O tempo de estadia de um módulo no SGPC está marcado por alguns momentos importantes como, por exemplo, os horários de Migração e de Início do módulo. A Figura 2.8 esquematiza o Tempo de Vida de um módulo no SGPC. Seguem-se definições destes Tempos:

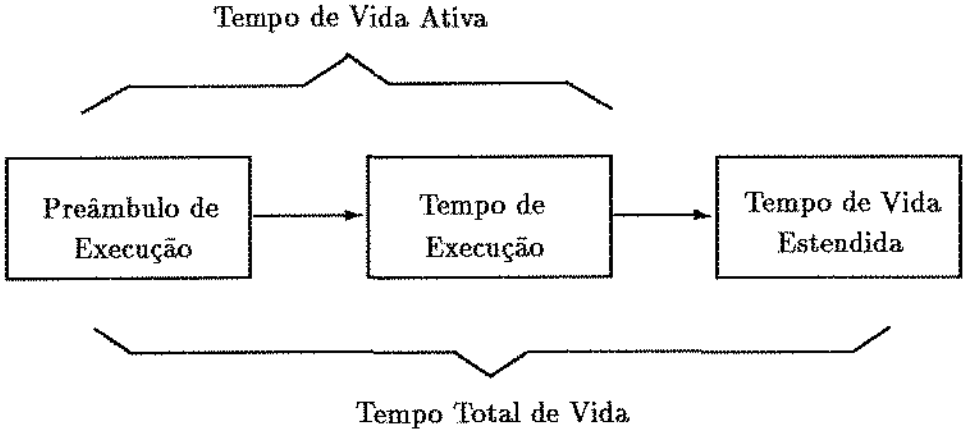

Figura 2.8: Tempos de Vida dos Módulos

- **Preâmbulo de Execução:** Tempo que transcorre entre o reconhecimento do pedido de execução de um módulo (por EXECMOD() ou EXECMODHOST(), Seção 3.1.3) e o começo de sua execução na Estação de Destino. Nesse tempo o módulo passa pelos estados **não_migrado** e **migrado.**
- **Tempo de Execução:** Período que vai do início da execução do módulo (com *execve()*) na Estação de Destino até o fim de sua execução (sua morte como filho do ServProc pai), momento em que é recebido o sinal SIGCHLD (Seção 3.6). Nesse tempo o módulo está no estado **em_progresso.**

- **Tempo de Vida Estendida**: Período que se inicia após a finalização da execução do módulo e dura até que ele desapareça do controle do SGPC. Neste tempo o módulo já não é mais conhecido pelo ServProc na Estação de Destino, mas as informações em relação a ele são retidas e mantidas por um tempo adicional na sua Estação de Origem. Nesse tempo o módulo (não ativo) aparece em um estado final **concluído**, se seu Ciclo de Vida no SGPC desenvolveu-se normalmente, ou **erro** se aconteceu alguma condição errônea. O fim do Tempo de Vida Estendida coincide com o fim do Tempo Total de Vida.

- **Tempo de Vida Ativa**: Período que inclui o tempo prévio à Execução do Módulo (Preâmbulo) e o Tempo de Execução. Nesse período o módulo passa pelos estados **não_migrado, migrado** e **em_progresso**, e é considerado ativo porque ele continua evoluindo e passando por transições de estado, isto é, não se encontra em um estado final.

- **Tempo Total de Vida**: Período em que o Módulo é conhecido no SGPC. Inicia-se com o recebimento do pedido de execução e se estende até o seu desaparição, com a finalização de seu Tempo de Vida Estendida. Nesse tempo o módulo deve ter completado um Ciclo de Vida, passando pelos estados **não_migrado, migrado, em_progresso**, e **concluído** (ou **erro**, devido a alguma situação excepcional).

O SGPC deve ter o controle dos módulos em andamento sob sua responsabilidade. Esse controle pressupõe a aquisição de informações de estado destes módulos. Para exercer este controle, cada ServProc atribui um Identificador a cada módulo, durante seu Tempo Total de Vida.

O fato do SGPC ser formado por um conjunto de ServProcs independentes, coordenados apenas pelo intercâmbio de algumas informações, faz com que as atividades de controle estejam também distribuídas. Em particular, para cada módulo migrado apenas dois Servidores no conjunto tomam conhecimento da existência desse módulo: os servidores executando nas Estações de Origem e de Destino da migração.

Enquanto o módulo se encontra no Preâmbulo da sua execução, apenas o servidor local sabe da sua existência. Enquanto o módulo permanece em seu Tempo de Execução, ambos os ServProcs sabem da sua existência, mas é o ServProc de Destino (hospedeiro) o que tem o controle direto sobre a referida execução; o ServProc Origem só é informado de seu estado. Uma vez terminada a execução, o ServProc de Origem é informado deste fato, e as informações relativas a esse módulo são apagadas do ServProc de Destino. No ServProc de Origem as informações do módulo sobrevivem ao término da sua execução, daí o Tempo de Vida Estendida.

Logo, cada Servidor tem que manter-se informado, controlando os módulos migrados a partir dele e para ele, isto é, os seus módulos exportados e importados, segundo o estágio em que se encontrem no seu Tempo Total de Vida.

É o Servidor de Origem o encarregado de atribuir a cada módulo o seu Identificador único dentro do SGPC. Este Identificador de Módulo é útil para as próprias tarefas internas de controle do SGPC, e para a identificação do módulo por parte da aplicação usuária.

No próximo capítulo abordaremos os ServProcs, sua interface, e sua implementação.

Capítulo 3

Servidor de Processamento

Neste capítulo é apresentado em detalhe o módulo Servidor de Processamento (ServProc). O ServProc é o coração do Sistema Gerenciador de Processamento Cooperativo (SGPC). É através da interação entre vários ServProcs que se realiza a avaliação do nível de carga de trabalho relativo das diversas estações, assim como o encaminhamento de módulos dos programas aplicativos entre as estações para serem executados. Ao longo do capítulo são enfocadas as técnicas utilizadas e descritos problemas e as soluções encontradas.

Um Servidor ou Processo Servidor (também chamado de *daemon*) é um processo que implementa serviços. É muito comum que, através destes, sejam providos serviços de rede. Esta é a maneira recomendada pela *Sun* para acrescentar novos serviços à rede. Esses serviços, na realidade, são uma coleção de programas, cada um deles com diversos procedimentos remotos. Cada programa destes representa uma versão diferente do conjunto de serviços.

Um Servidor trabalha em estreita vinculação com o *kernel* do Sistema Operacional, mas seu código não está contido nele. A decisão de implementar este mecanismo de servidores sem que eles façam parte do próprio *kernel* evita o crescimento exagerado deste último, além de facilitar o acréscimo de serviços por parte do usuário.

O conjunto de todos os procedimentos que implementam este tipo de serviço, seus parâmetros e resultados, isto é, tudo o que se necessita para um usuário poder explorar um Servidor, são documentados, segundo as normas, em uma especificação chamada "Protocolo de Serviço RPC do Programa Servidor". Este protocolo é a definição da interface do programa escrito na linguagem RPC. A descrição do Protocolo do ServProc pode ser consultada no Apêndice C.

A criação do SGPC foi auxiliada pelo compilador de protocolos *rpcgen*. Este aceita como entrada a definição de uma interface do programa remoto escrita na linguagem RPC, e gera programas na linguagem fonte C.

A saída do *rpcgen* inclui *stubs* do cliente e do servidor do Serviço RPC, podendo conter, além disso, rotinas XDR. O *stub* do cliente responsabiliza-se pelo *empacotamento* dos parâmetros, pelo envio da requisição, e pelo *desempacotamento* dos resultados devolvidos pelo servidor.

O *stub* do servidor desempacota os parâmetros e faz a chamada local do procedimento desejado. Finalmente os resultados são empacotados e enviados de volta para o cliente.

As rotinas XDR são úteis no *empacotamento* e *desempacotamento* de parâmetros e resultados em um formato independente da arquitetura dos computadores envolvidos.

Os esqueletos de programas gerados pelo *rpcgen* para suportar o RPC nas versões atuais do *SunOS* podem incluir, opcionalmente, código para os mecanismos de transporte TCP (*Transmission Control Protocol*) e UDP (*User Datagram Protocol*). Na geração do código para o SGPC optou-se por manter o transporte com o protocolo TCP devido a sua maior confiabilidade. Este último inclui políticas de retransmissão e *timeout*.

Os programas gerados pelo *rpcgen* não são os de uso definitivo. Estes programas gerados foram acrescentados com o código particular do Sistema Gerenciador de Processamento Cooperativo.

Além de todo o código específico acrescentado, foram feitas modificações nas partes de código geradas automaticamente. Estas modificações incluem:

- adequações para que os ServProcs também façam o papel de Clientes (este é o caso das transações entre Servidores Remotos);
- adequações para que alguns procedimentos remotos não devolvam resultados, e para que os respectivos Clientes não fiquem bloqueados esperando por resposta do procedimento remoto que lhe dá serviço;
- inclusão da liberação de memória alocada pelas rotinas XDR em chamadas repetitivas.

Os serviços oferecidos pelos ServProcs são abordados na Seção 3.1. Nestes serviços pode-se diferenciar dois grupos: os preparados para que as aplicações usufruam o SGPC, e os preparados para garantir as interligações entre ServProcs. Estes últimos possibilitam a integração do SGPC a partir de um conjunto de ServProcs.

Cada estação que oferece os serviços do Processamento Cooperativo executa um processo ServProc que fica residente nela a partir do momento de sua instalação. Este processo de Instalação é detalhado na Seção 3.2.

O critério considerado no processo de escolha da estação de destino é abordado na Seção 3.3. As informações e as tarefas necessárias para manter o controle dos módulos migrados são o tema da Seção 3.4.

As características dos módulos submetidos a migração são apresentadas na Seção 3.5. Nessa seção também são apresentadas as condições de *environment* em que são executados os módulos migrados.

Os módulos em execução são monitorados até seu término. As providências tomadas para a percepção desse término são abordadas na Seção 3.6.

O tratamento dado pelos ServProcs e pelo SGPC a condições de erro que podem ocorrer ao longo do funcionamento do Sistema é discutido como ponto final do capítulo, na Seção 3.7. Uma listagem completa dos códigos de erro e suas mensagens explicativas pode ser consultada no Apêndice E.

Na seção final do Capítulo (3.8) são apresentadas as rotinas de biblioteca que constituem a interface de maior nível do SGPC Estendido.

3.1 Serviços do ServProc

O SGPC amplia os serviços de rede do *SunOS*. Ajustando-se às normas de implementação de novos serviços, a sua ferramenta base é um Servidor, o ServProc. Nesta seção abordaremos os serviços oferecidos por este ServProc.

Segundo o tipo do cliente, os procedimentos remotos do SGPC podem ser divididos em dois grandes grupos:

- os preparados para suportar serviços para as aplicações Clientes;
- os preparados para serem usados na intercomunicação entre *Servidores de Processamento*.

Os projetados pensando em outros ServProcs como clientes têm como objetivo viabilizar a integração dos diversos ServProcs em um todo, chamado de SGPC. Através destes serviços materializa-se o intercâmbio de informações de configuração, de informações de estado, e o que é mais importante, a migração dos módulos. Na apresentação de cada serviço será esclarecido para que tipo de cliente foi projetado o mesmo.

Os serviços no ServProc, segundo a sua função, estão incluídos em algum dos seguintes grupos:

- identidade dos ServProcs;
- informações sobre condições errôneas;
- informações sobre as estações;
- execução de módulos;
- informações sobre os módulos.

A seguir serão discutidos, em subseções separadas, os serviços segundo a sua função. Em todos os casos, as informações retornadas pelos serviços vem acompanhadas da Identidade (*stamp*) do ServProc que faz o envio, isto é, do ServProc que prestou o serviço. É responsabilidade do Cliente verificar que as identidades original e corrente coincidam. Nos casos em que os ServProcs fazem o papel de clientes, eles garantem esta verificação. Com o uso do SGPC Estendido também é garantida a verificação da Identidade para todos os serviços.

Em todos os casos também é fornecido, com os dados de retorno, um código de erro. Os serviços de erro são tratados na Seção 3.7.

3.1.1 Serviços sobre a Identidade do ServProc

Os serviços neste grupo são:

- **int ENVIAIDSERVPROC(stamp) = 1**
- **stamp PEDEIDSERVPROC(void) = 2**

O primeiro destes serviços foi concebido com o objetivo de que um ServProc, na sua instalação, faça saber aos outros ServProcs de sua existência, informando-lhes sua Identidade. De retorno recebe a confirmação ou não de que o *stamp* enviado foi recebido.

Os clientes (tanto aplicações usuárias como outros ServProcs) podem também fazer um pedido explícito da identificação de um servidor remoto. Isto é conseguido através do segundo destes serviços. Este segundo serviço é empregado pelos ServProcs em sua instalação, na troca de identificações inicial que permite a integração do SGPC como um todo. A existência destes Identificadores e seu intercâmbio garantem ainda futuras detecções de falhas (Seção 3.7).

A Identidade de um ServProc está associada ao horário em que acontece sua instalação:

```
struct stamp

    {
      cadeia nome_host;
      long   sec;
      long   usec;
    } /* stamp */
```

Este *stamp* é criado pelo próprio servidor na sua instalação a partir do horário do sistema local e do nome lógico da estação.

3.1.2 Serviços de Informações sobre as Estações

Os serviços incluídos neste grupo são:

- **ids_host LISTAHOSTS(void) = 3**
- **inf_host PEDEINFHOST(cadeia) = 4**

O primeiro destes (concebido para ser usado por aplicações usuárias) retorna uma listagem das estações na rede. Esta listagem é devolvida na seguinte estrutura:

struct ids_host

```
{
  stamp  ident_serv;
  int    cod_erro;
  int    quantidade;
  cadeia lista_nomes[MAX_HOSTS];
}  /* ids_host */
```

Na lista de nomes (apontadores para cadeias de caracteres terminadas em NULL) de estações devolvida aparecem todas as estações detectadas na rede. Isso não significa que todas elas estejam preparadas para receber migrações de módulos.

O segundo destes serviços é o que propriamente oferece informações sobre as características e estado das estações, e entre elas, se a estação pode ou não colaborar no processamento cooperativo. Este procedimento recebe como parâmetro o nome (apontador para a cadeia de caracteres) lógico da estação de interesse. Este serviço foi concebido para ser utilizado tanto por aplicações usuárias quanto por outros ServProcs.

A pedido das aplicações, o Servidor Local localiza e envia estas informações a partir dos dados locais a ele. Isto pressupõe a obtenção prévia dos dados de caráter dinâmico, a partir da estação remota de interesse. As informações das estações, acessíveis através deste serviço são refletidas na seguinte estrutura:

struct inf_host

```
{
   stamp   ident_serv;
   int     cod_erro;
   cadeia  nome;
   cadeia  arch;
   int     tem_server;
   stamp   ident_serv_host;
   int     porc_cpu_idle;
   int     quant_mods_impors;
   int     quant_usuarios;
   int     quant_processos;
}  /* inf_host */
```

O campo *.ident_serv* identifica o Servidor de Processamento que envia toda esta estrutura de informações. O contacto que envia o Identificador original é o primeiro estabelecido entre o Servidor Local e o Servidor Remoto de referência; esta identificação é obtida nesse momento e

usada para validar o remetente de todos os contactos posteriores. A não correspondência entre este identificador e outro que supostamente identifica o mesmo servidor é tomada como sinal de que no servidor original aconteceu uma queda (Seção 3.7).

O campo *.arch* representa a arquitetura de aplicação, e indica quais códigos executáveis podem executar sobre a máquina dada. Duas máquinas têm Arquiteturas de Aplicação iguais se elas podem executar os mesmos programas de aplicação (independentemente de ter ou não outras diferenças). Esta condição é considerada como requerimento indispensável no momento de selecionar a Estação de Destino para o módulo executável (Seção 3.3).

O dado do campo *.tem_server* é o indicador de que se pode contar com essa estação para empréstimo de sua capacidade de processamento. Os campos que restam refletem informações dinâmicas, e só contém informações significativas no caso de existir um ServProc na estação que as colete e envie. Estas informações mostram o estado corrente da estação em questão.

.porc_cpu_idle: é um relato de utilização da unidade de processamento central (CPU). O valor representa a porcentagem de tempo que ela tem ficado ociosa (*idling*). Esta porcentagem é calculada do intervalo de tempo transcorrido desde o último relato da mesma. Este valor é considerado na escolha automática da estação de destino no momento de migrar um módulo;

.quant_mods_impors: representa a quantidade de módulos importados pela estação a pedido da atual. Este valor é considerado no critério atual para a seleção da estação de destino;

.quant_usuários: contém a quantidade de usuários correntes da estação. Este valor inclui os usuários trabalhando propriamente na estação física, aqueles trabalhando em terminais ligadas à estação, e até usuários em outras estações e terminais na rede que executaram *login's* remotos acessando esta;

.quant_processos: reflete a quantidade de processos executando na estação.

3.1.3 Serviços para a Execução de Módulos

Nos serviços de execução de módulos está a razão de ser do sistema como um todo. Eles permitem a execução dos módulos em outras estações aproveitando melhor a capacidade de processamento esparsa pela rede toda. Este aproveitamento consegue-se com a migração dos módulos para execução entre as estações, depois de ter sido avaliado o nível de utilização de cada estação.

Os serviços neste grupo são:

id_mod EXECMOD(mod) = 5: recebe o pedido para a execução de um módulo com escolha automática da estação de destino;

id_mod EXECMODHOST(mod_host) = 6: recebe o pedido para a execução de um módulo com a indicação da estação de destino desejada;

id_mod_migrado EXECREMMOD(inf_mod) = 7: recebe o pedido de execução remota de um módulo que está sendo exportado para esta estação.

A execução de um módulo começa com a chamada por parte da aplicação cliente de um serviço de pedido de execução. Os serviços para este tipo de pedido são **EXECMOD()** e **EXECMODHOST()**. Com o recebimento destes pedidos os módulos começam seu Ciclo de Vida pelo estado **não_migrado**. É neste momento que se diz que eles estão no Preâmbulo das suas execuções. Estes dois primeiros serviços estão destinados a serem usados pelas aplicações usuárias.

Os dois procedimentos para pedidos de execução pelas aplicações diferem em que o primeiro destes implica que o Sistema fique totalmente encarregado da decisão sobre qual será a **estação hospedeira** do módulo, enquanto no segundo existe um parâmetro a mais através do qual o usuário pede a **estação hospedeira** de sua preferência.

Uma discussão dos aspectos sobre a escolha automática da estação de destino pode ser vista na Seção 3.3. Como caso particular se pode tomar a decisão de que não acontecerá migração, sendo que a execução será local.

Em estes dois primeiros procedimentos de pedido de execução tem que ser dado como entrada um parâmetro com a descrição que permita localizar e manipular o módulo. Esta estrutura contém as informações que normalmente dão-se na linha de comandos da ordem interativa de execução.

```
struct mod

    {
       cadeia nome;
       cadeia path;
       cadeia argv;
       cadeia envp;
       int    re_avalia;
       int    re_executavel;
    } /* mod */
```

- *.nome*: nome que identifica o módulo (nome de arquivo com o programa executável);
- *.path*: caminho através do sistema de arquivos que permite localizar o diretório que contém o arquivo executável;
- *.argv*: cadeia de caracteres com os argumentos (parâmetros) a serem passados ao módulo ao iniciar sua execução;
- *.envp*: cadeia de caracteres com as variáveis de *environment* desejadas acompanhadas de seus valores.

Através da informação em *.re_avalia* o cliente estabelece se o Servidor Local deve provocar uma coleta de dados novos através de todo o SGPC antes de selecionar a estação de destino. Isto implica também em não fazer depender a escolha, de migrações anteriores realizadas pelo ServProc Local.

O campo *.re_executável* contém uma indicação dada pelo usuário segundo a qual o SGPC vai ou não executar novamente o módulo em caso de quedas que interrompam sua execução. Este indicador deve ser posicionado segundo o programa na sua execução provoque ou não efeitos colaterais.

O parâmetro de tipo *mod_host* do serviço **EXECMODHOST()** inclui, além destes campos, um outro campo *.nome* com a identificação da estação de destino desejada.

Nestes dois serviços para pedido de execução coincide a informação devolvida no retorno: o Identificador do Módulo para o SGPC.

```
struct id_mod

    {
       stamp  ident_serv;
       int    cod_erro;
       int    ident_modulo;
    } /* id_mod */
```

O valor do campo *.ident_modulo* serve para identificar, de maneira única, o módulo dentro do SGPC. Este identificador deve ser conservado pela aplicação para sua utilização posterior em pedidos de informações sobre o módulo correspondente (Seção 3.1.4). Internamente este identificador é um índice de entrada na tabela interna de módulos exportados do ServProc de Origem.

O serviço **EXECREMMOD()** é propriamente o pedido remoto de execução. É este o serviço de maior importância nos ServProcs, sendo o agente principal na migração dos módulos. Está concebido para seu uso por outros ServProcs agindo como clientes. Este serviço é chamado no corpo da implementação dos dois anteriores, uma vez que tenha sido decidida que a execução vai ser remota.

O pedido deste serviço faz com que o módulo transite para o estado **migrado**, ainda no Preâmbulo de sua execução. O procedimento que implementa este serviço recebe como parâmetros, entre outros, os dados necessários para:

- localizar o arquivo executável no sistema de arquivos;
- identificar o módulo unicamente dentro do SGPC;
- impor o ambiente de execução similar ao da Estação de Origem.

Para conseguir a execução do módulo o ServProc faz uso das primitivas *fork()* e *execve()*. O identificador do processo em execução, atribuído pelo Sistema Operacional, constitui o identificador do módulo no destino (campo *.id_processo* de *id_mod_migrado*).

O êxito no lançamento da execução do módulo marca o início de seu Tempo de Execução, e como conseqüência o módulo passa ao estado **em_progresso**.

A estrutura (**struct** *id_mod_migrado*) devolvida por **EXECREMMOD()** é a seguinte:

```
struct id_mod_migrado

    {
       stamp  ident_serv;
       int    cod_erro;
       int    id_processo;
    } /* id_mod_migrado */
```

3.1.4 Serviços de Informações sobre os Módulos

Os serviços neste grupo são:

ids_mods LISTAMODS(void) = 8: retorna quantos e quais são os módulos migrados através deste ServProc;

inf_mod PEDEINFMOD(int) = 9: recebe o pedido de informações sobre o estado do módulo que se corresponde com o Identificador dado como parâmetro. A execução do módulo deve ter sido invocada previamente, através de **EXECMOD()** ou **EXECMODHOST()**; os quais devolvem o identificador do módulo para o SGPC. O identificador do módulo pode também ser obtido com uma chamada prévia a **LISTAMODS()**;

int ENVIAINFMOD(inf_mod) = 10: informa à estação de origem de um módulo sobre o estado deste na estação de destino.

O primeiro dos serviços acima (projetado para ser usado por aplicações usuárias) fornece a listagem dos módulos ativados através deste ServProc. O serviço retorna estes dados em uma estrutura da forma:

```
struct ids_mods

    {
       stamp  ident_serv;
       int    cod_erro;
       int    quantidade;
       int    lista_ids[MAX_MODS];
    } /*  ids_mods   */
```

A lista obtida por este serviço inclui todos os módulos no sistema (enquanto estejam em seu Tempo de Vida) que tenham a estação local como origem e/ou destino de sua migração.

O SGPC encarrega-se, de maneira automática, de manter sob vigilância os módulos em andamento. Os serviços **PEDEINFMOD()** e **ENVIAINFMOD()** são utilizados pelos próprios ServProcs (nas estações de origem e de destino) para manterem-se informados do estado dos módulos migrados através dos mesmos.

Por sua vez as aplicações usuárias podem utilizar também o serviço **PEDEINFMOD()** para informar-se do estado dos módulos de seu interesse.

No pedido de informações de estado de um módulo deve-se dar como parâmetro de entrada o Identificador do Módulo desejado. Este identificador pode ser obtido com o serviço que retorna a lista de módulos em andamento, ou pode ser obtido e conservado desde o momento em que se fez o pedido de sua execução (Seção 3.1.3).

As informações do módulo em execução retornadas por **PEDEINFMOD()** e enviadas por **ENVIAINFMOD()** (**struct inf_mod**) incluem as seguintes:

```
cadeia host_origem;
int    ident_modulo;
tempo  horario_migracao;
tempo  horario_reconh_termino;

cadeia host_destino;
int    migrado;
tempo  horario_inicio;
tempo  horario_termino;

int         id_processo;
estado_exec estado;
int         cod_term;
tempo       ru_utime;
tempo       ru_stime;
```

Onde *estado* pode ser um dos seguintes:

```
enum estado_exec {nao_migrado, migrado, em_progresso,
                  concluido, erro};
```

Os campos *.nome*, *.path*, *.argv*, *.envp*, *.re_avalia*, e *re_executável* são preenchidos diretamente a partir dos dados fornecidos pelo cliente que fez o pedido de execução do módulo.

O campo *.host_destino* é preenchido ao ser escolhida a estação que vai hospedar o módulo em sua execução. A partir deste dado e do dado do *.host_origem* é possível determinar se se trata de um módulo *exportado* ou *importado*. A existência do campo *.migrado* foi decidida por conveniência, embora exista redundância nas informações refletidas.

Existem ainda os identificadores do módulo no SGPC e na sua Estação Hospedeira, *.ident_modulo* e *.id_processo*, respectivamente. São também mantidos alguns horários importantes no Ciclo de Vida do Módulo: *.horário_migração*, *horário_reconh_termino*, *horário_inicio*, e *horário_termino*, preenchidos os dois primeiros pela Estação Origem, e os dois últimos pela Estação de Destino.

Com o término da execução de um módulo são refletidas em *.ru_utime* e *.ru_stime* as quantidades de tempo em que o módulo utilizou o processador na sua execução (discriminadas em tempo passado em modo usuário e tempo passado em modo sistema).

O código de terminação (campo *.cod_term*) corresponde ao valor de *status* de saída do processo executado, oferecido pelo Sistema Operacional.

É através do serviço **ENVIAINFMOD()** que o servidor de origem fica sabendo quando termina a execução de um módulo que exportou, e o valor de retorno deste serviço é utilizado para conferir se o envio das informações do estado do módulo chegaram a seu destino.

As informações de um módulo são retidas na sua Estação de Destino até que a estação de origem do módulo seja informada do término de sua execução. Entretanto, as informações dos módulos são retidas na sua Estação de Origem ainda depois de terminada sua execução. Um módulo termina seu Tempo Total de Vida no sistema apenas quando a aplicação que provocou sua execução toma conhecimento de sua finalização. Neste caso dá-se por terminado o Tempo de Vida Estendido do módulo.

3.2 Instalação do ServProc

O trabalho do SGPC depende em grande medida da interação entre os Servidores de Processamento residentes nas estações participantes da cooperação. O ServProc é capaz de se auto-configurar e de estabelecer comunicação com outros ServProcs em outras estações. Este deve ficar coletando informações e pronto para servir como *"Servidor de Processamento"*.

O Servidor é responsável pelo armazenamento e manipulação das informações referentes às estações e aos módulos, e mais, por garantir a satisfação dos serviços descritos na Seção 3.1. Para conseguir isto, cada ServProc tem primeiro que se auto-instalar, inteirando-se de quais outros Servidores de Processamento existem na rede, e fazendo com que os outros saibam da sua existência.

A Figura 3.1, ilustra o processo de iniciação do Servidor de Processamento e sua posterior latência como *"daemon"*.

Não necessariamente todas as estações na rede participam desta colaboração. É condição necessária para qualquer uma delas fazer parte do conjunto de estações na colaboração, que estejam executando o "daemon" que implementa o Servidor de Processamento. Por sua vez, para que a instalação dos ServProcs possa dar certo, é necessário que estejam presentes alguns serviços do Sistema Operacional considerados básicos para estes Servidores. Uma listagem destes serviços aparece no Apêndice A, e uma referência sobre a utilidade de cada um encontra-se na Seção 2.1, que apresenta a Plataforma Computacional.

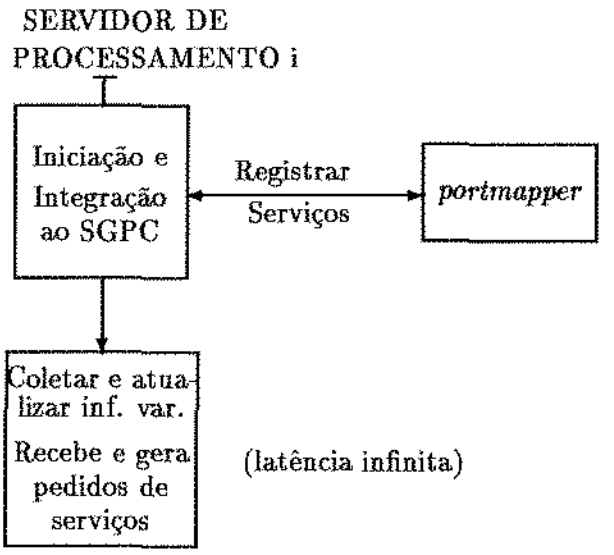

Figura 3.1: Iniciação e Tarefas de um Servidor de Processamento

3.2.1 Tarefas do Processo de Instalação

A seguir apresentam-se as tarefas que o ServProc cumpre no processo de instalação:

- averigua o nome do domínio NIS da rede;
- averigua nome do *host* corrente;
- cria identidade (*stamp*) de si mesmo;
- coleta informações de caráter fixo da estação corrente;
- extrai as informações de estado da estação corrente;
- desliga-se do terminal de controle;
- estabelece o uso do mecanismo de mensagens do *kernel*;
- determina quais são as estações existentes na rede local;
- localiza outros ServProcs e pede suas Identidades;
- demanda de ServProcs Remotos informações de caráter fixo e de estado;
- envia sua Identidade para os outros ServProcs instalados.

Identidade do ServProc

A Identidade de um ServProc está associada ao horário em que acontece sua instalação (Seção 3.1.1). Este dado é utilizado na verificação dos contactos com ele, e através do mesmo são detectadas possíveis quedas dos ServProcs (Seção 3.7).

Terminal de Controle

O ServProc vai ser executado em *background*, logo não deve ter interações diretas com o terminal. Por este motivo o ServProc se desliga do terminal de controle, excluindo-se da *sessão* do terminal que o invocou. Esta provisão tem conseqüências diretas sobre a interação dos módulos a serem executados (Seção 3.5.2).

Sob o Sistema Operacional, todo processo é membro de uma *sessão*, e os processos em sua criação se fazem pertencer à *sessão* de seu pai. Um processo tem a possibilidade de alterar sua pertinência a uma *sessão*.

Por sua vez, um terminal pode ser associado a uma *sessão*, e neste caso este é o terminal de controle (*controlling terminal*) da sessão. Desta maneira, o grupo de processos na sessão será também o grupo de processos associado ao terminal (*tty's process group*). Por definição diz-se que estes processos executam em *foreground*, e como tais podem realizar operações de entrada e de saída interativas através desse terminal.

Cada processo pode ou não ser membro de um grupo de processos associado a um terminal. Se um processo deseja não ter terminal de controle associado, este tem que provocar sua saída do grupo de processos associados a um terminal de controle. Isto implica na criação de uma nova *sessão*, e esse processo passa a ser o líder da mesma.

Se um processo pertence a uma sessão que não tem terminal de controle associado, diz-se que executa em *background*, e como tal lhe é proibido realizar entradas e saídas interativas a partir de qualquer terminal.

O ServProc consegue desligar-se do terminal de controle com a seguinte seqüência de instruções:

```
for (i = getdtablesize() - 1 ; i >= 0 ; --i) close(i);

open("/dev/null", O_RDONLY);

dup2(0, 1);
dup2(0, 2);

i = open("/dev/tty", O_RDWR);
if (i > 0)
  { ioctl(i, TIOCNOTTY, 0);
    close(i);
  }
```

Utilização do Mecanismo de Mensagens do *Kernel*

Como conseqüência da execução do fragmento de programa que desliga ao ServProc do terminal de controle, este não poderá mais (por via direta) mostrar mensagens na tela. Entretanto, às vezes o ServProc necessita mostrar algumas mensagens em situações excepcionais. Para dar solução a este problema o ServProc faz uso do mecanismo padrão de mensagens do *kernel* realizando chamadas à função *syslog()* sempre que seja necessário.

Utilização do NIS

Os serviços do *Network Information System* (NIS) são utilizados para conhecer dinamicamente a configuração de estações interligadas na rede.

Com a averiguação de quais são as estações da rede local, é guardada uma quantidade de informações mínima de cada uma delas, inclusive das que no momento não tem o ServProc instalado, porque poder-se-ia instalá-lo depois. Desta maneira, no Servidor em instalação ficam preparadas as condições mínimas para a posterior integração de novos ServProcs ao SGPC.

Integração ao SGPC

Um Servidor de Processamento executando em uma estação pode ficar sabendo da existência de Servidores de Processamento em outras estações de duas maneiras diferentes:

- ao tentar contactar o suposto Servidor com um pedido de converter-se em seu Cliente;
- ao receber um pedido de serviço de outro ServProc.

A primeira forma é tentada pelo Servidor na sua instalação, e o possível êxito dessa ligação constataria a existência do Servidor Remoto. A segunda forma pode acontecer eventualmente quando o servidor recebe um pedido de serviço propriamente dito.

Os Servidores podem iniciar-se nas diferentes estações em momentos distintos. Essa instalação pode acontecer em tempo de iniciação da máquina ou em qualquer momento posterior. O SGPC está habilitado para incorporar novos Servidores de Processamento em qualquer momento.

No momento da instalação o Servidor que está sendo instalado averigua quais outros ServProcs já existem instalados. Por sua vez os restantes ficam sabendo da existência deste novo através de pedidos de serviços que ele faz aos mesmos.

Em resumo, o conhecimento da existência de todos em relação a todos os outros consegue-se com uma combinação de perguntas de uns aos outros e de comunicação de todos aos demais. Esta integração de ServProcs está acompanhada do intercâmbio de suas Identidades.

Na própria instalação faz-se o pedido das informações (**struct** *inf_host*) das outras estações, sendo que são guardadas localmente. Para esta transferência são chamados os procedimentos remotos PEDEINFHOST() (Seção 3.1.2) de cada ServProc.

Outras Tarefas da Instalação

O processo de Configuração estabelece as condições para que eventualmente seja chamado o procedimento interno do ServProc **fim_de_modulo()** para a detecção do término da execução dos módulos que foram *"importados"* por este Servidor (Seção 3.6).

Como última tarefa da instalação o ServProc se auto-registra ante o Serviço de RPC e do *portmapper* garantindo o acesso aos procedimentos remotos que implementa. Uma vez terminado o processo de instalação o Servidor fica residente como *daemon*, na espera de pedidos de serviços.

Na Figura 3.2 mostra-se um exemplo de rastro deixado por um ServProc ao ser instalado. A partir deste ponto este tipo de rastro será chamado de Histórico.

```
Nome Dominio: dcc.unicamp.br

      Estacao          Ident_Serv        Arq.    Idle  Quant Users  Procss

 0:           s2a <        -1><     -1> <desc.> < -1>   <   -1>    < -1>
 1:           s2c <        -1><     -1> <desc.> < -1>   <   -1>    < -1>
 2:           s2i < 725557827>< 331777> < sun4> <100>   <    1>    < 19>
 3:           s3a <        -1><     -1> <desc.> < -1>   <   -1>    < -1>
 4:           s2e < 725557803>< 906741> < sun4> < 99>   <    1>    < 20>
 5:           s3b <        -1><     -1> <desc.> < -1>   <   -1>    < -1>
 6:           s2d < 725557768>< 240827> < sun4> <100>   <    1>    < 19>
 7:           s2h <        -1><     -1> <desc.> < -1>   <   -1>    < -1>
 8:           s2g <        -1><     -1> <desc.> < -1>   <   -1>    < -1>
 9:           s2f <        -1><     -1> <desc.> < -1>   <   -1>    < -1>
10:         tiete <        -1><     -1> <desc.> < -1>   <   -1>    < -1>
11:           s2b <        -1><     -1> <desc.> < -1>   <   -1>    < -1>
12: dcc.unicamp.br <        -1><     -1> <desc.> < -1>   <   -1>    < -1>
13:       marumbi < 725557738>< 781618> < sun4> < 46>   <    1>    < 19>

Quantidade de Estacoes: 14    Estacao Atual[2]: s2i

******** Ate aqui instalacao **********
```

Figura 3.2: Histórico dos Contactos com outros ServProcs na Instalação

3.3 Seleção da Estação Hospedeira

A escolha de uma estação para agir como estação hospedeira de um módulo migrado requer a satisfação de algumas condições imprescindíveis:

- existência do Servidor de Processamento executando na estação;
- correspondência entre suas arquiteturas de aplicação (Seção 3.5.1).

A escolha automática da estação de destino para uma migração garante que esta cumpre esses requerimentos imprescindíveis. Uma vez determinado o conjunto de estações que cumpre os requisitos necessários, passa-se fazer a seleção de uma delas para a migração.

Nesta seleção são consideradas características de estado. Os dados de estado que no momento o SGPC mantém de cada estação são os seguintes:

- porcentagem de tempo ocioso da CPU;
- quantidade de módulos importados;
- quantidade de usuários *"logados"*;
- quantidade de processos executando.

A avaliação do nível de ociosidade de uma estação, no caso do trabalho apresentado, tem sido simplificada ao uso da informação sobre a porcentagem de tempo ocioso (*idle*) de sua CPU no último intervalo de tempo.

Os critérios para a avaliação do nível de utilização da CPU a partir dos dados do estado da estação são suscetíveis de serem reconsiderados. No caso, optou-se por uma variante simples. No entanto, a informação selecionada para basear a escolha é uma medida muito significativa do grau de ociosidade do processador central da estação.

A própria escolha de uma estação para a execução de um módulo leva em consideração este nível de ociosidade, além das migrações anteriores (ainda ativas) realizadas por este servidor. Cada ServProc mantém uma estrutura de dados auxiliar (*hosts_ordenados*) onde aparecem as estações ordenadas segundo a porcentagem de tempo em que a CPU tem ficado ociosa. Esta estrutura de dados mantém de cada estação seu índice e essa porcentagem:

```
struct inf_sel_host

    {
      int  indice_host;
      int  porc_cpu_idle;
    }  /*  inf_sel_host  */
```

A atribuição de estações para a execução de módulos é feita seguindo esta ordem. Como dado auxiliar é considerada a quantidade de módulos que já tem sido migrados para cada estação, fazendo com que não se repitam as seleções.

Na Figura 3.3 mostram-se estas informações, assim como as mesmas vão sendo modificadas. O valor entre colchetes é o índice que identifica a estação (Figura 3.2), e os dois valores entre os sinais de menor e maior representam a porcentagem de tempo ocioso e a quantidade de módulos importados, respectivamente.

Nessa figura pode-se apreciar que as estações neste vetor encontram-se ordenadas segundo suas porcentagens de tempo ocioso, e como a quantidade de módulos importados é atualizada na medida que as estações são selecionadas para as execuções dos módulos.

No momento de selecionar a estação de destino para um módulo, considera-se o valor dado pelo usuário no campo *.re_avalia* (Seção 3.1.3). De acordo com este valor se determina a obtenção ou não de valores atualizados de porcentagem de tempo ocioso, e a reordenação das estações a selecionar, segundo estes valores novos.

Entretanto, as aplicações usuárias também terão a possibilidade de fazer a seleção da estação de destino. Para isto devem utilizar o serviço **EXECMODHOST()**. Nessa seleção pode-se considerar outros dados de estado.

3.4 Controle dos Módulos Migrados

Todo ServProc mantém duas estruturas de dados com as informações dos seus Módulos Exportados e Importados. Os momentos de atualização destes dados são mostrados na Figura 3.4.

Como foi apontado na Seção 2.4, a cada módulo submetido para execução remota é associado um identificador. O ServProc na Estação Origem do módulo é o encarregado de criar o Identificador do Módulo. Desta maneira, para cada módulo em andamento existe um Identificador único ao longo de sua existência no SGPC.

Com o pedido de uma aplicação para executar um módulo, é inserida uma entrada com suas informações na estrutura de dados da Estação de Origem que reflete os Módulos Exportados. Com a migração do módulo são transferidas as informações armazenadas na fonte em relação a este para a Estação de Destino. O ServProc cria, nesta última, uma entrada para o módulo dado em sua estrutura de dados que reflete os Módulos Importados. Enquanto o módulo está em execução (estado *em_progresso*) o controle direto deste é exercido pelo ServProc na Estação Hospedeira.

Com o fim da execução do módulo termina seu Tempo de Vida Ativa. O ServProc na Estação Hospedeira apaga de sua estrutura de dados que contém os Módulos Importados todo o relativo ao módulo terminado, mas estes dados são mantidos ainda na estrutura de Módulos Exportados (Tempo de Vida Estendida), dando oportunidade para a recuperação de seu *status* pelas aplicações.

As Figuras 3.5 e 3.6, respectivamente, mostram fragmentos da história de um ServProc que age na Origem (s2i) de uma migração e de um outro ServProc que age no Destino (s2d) da própria migração.

```
                ⋮

Hosts segundo sua porc_cpu_idle: [indice]<porc_cpu_idle><quant_mods_impors>

[ 2]:<99><0>  [ 6]:<99><0>  [ 4]:<98><0>  [13]:<49><0>  [ 1]:<-1><0>
[ 5]:<-1><0>  [ 0]:<-1><0>  [ 7]:<-1><0>  [ 8]:<-1><0>  [ 9]:<-1><0>
[10]:<-1><0>  [11]:<-1><0>  [12]:<-1><0>  [ 3]:<-1><0>

Servico EXECMOD: Nome: <cc>  Path:</bin>  Argv:< -c make.c -o make.o>

Hosts segundo sua porc_cpu_idle: [indice]<porc_cpu_idle><quant_mods_impors>

[ 2]:<99><1>  [ 6]:<99><0>  [ 4]:<98><0>  [13]:<49><0>  [ 1]:<-1><0>
[ 5]:<-1><0>  [ 0]:<-1><0>  [ 7]:<-1><0>  [ 8]:<-1><0>  [ 9]:<-1><0>
[10]:<-1><0>  [11]:<-1><0>  [12]:<-1><0>  [ 3]:<-1><0>

                ⋮

Servico EXECMOD: Nome:<cc>  Path:</bin>  Argv:< -c parse.c -o parse.o>

Hosts segundo sua porc_cpu_idle: [indice]<porc_cpu_idle><quant_mods_impors>

[ 2]:<99><1>  [ 6]:<99><1>  [ 4]:<98><0>  [13]:<49><0>  [ 1]:<-1><0>
[ 5]:<-1><0>  [ 0]:<-1><0>  [ 7]:<-1><0>  [ 8]:<-1><0>  [ 9]:<-1><0>
[10]:<-1><0>  [11]:<-1><0>  [12]:<-1><0>  [ 3]:<-1><0>

Chamada a EXECREMMOD de:<s2d> com </bin/cc>  < -c parse.c -o parse.o>

                ⋮

Servico EXECMOD: Nome:<cc>  Path:</bin>  Argv:< -c tstring.c -o tstring.o>

Hosts segundo sua porc_cpu_idle: [indice]<porc_cpu_idle><quant_mods_impors>

[ 2]:<99><1>  [ 6]:<99><1>  [ 4]:<98><1>  [13]:<49><0>  [ 1]:<-1><0>
[ 5]:<-1><0>  [ 0]:<-1><0>  [ 7]:<-1><0>  [ 8]:<-1><0>  [ 9]:<-1><0>
[10]:<-1><0>  [11]:<-1><0>  [12]:<-1><0>  [ 3]:<-1><0>

Chamada a EXECREMMOD de:<s2e> com </bin/cc>  < -c tstring.c -o tstring.o>

                ⋮
```

Figura 3.3: Histórico de um ServProc na Seleção da Estação Hospedeira

Estação de Origem		Estação de Destino (hospedeira)	
Aplicação (inclui InterfPC)	Servidor Local (ServProc)	Servidor Remoto (ServProc)	Módulo Migrado
	inserir em expors (**EXECMOD()** e **EXECMODHOST()**)		
		inserir em impors (**EXECREMMOD()**)	
			execução
		apagar de impors (*fim_de_modulo()* chama a **ENVIAINFMOD()**)	
	apagar de expors (**PEDEINFMOD()**)		
1	2	3	4

Figura 3.4: Controle dos Módulos durante seu Ciclo de Vida

```
                ⋮

Servico EXECMOD: Path:</bin>  Argv:< -c parse.c -o parse.o>

                ⋮

Chamada a EXECREMMOD de:<s2d> com </bin/cc>  < -c parse.c -o parse.o>

                ⋮

Servico PEDEINFMOD[1]
E[ 1]
 Nome:<cc>   Path:</bin>   Argv:< -c parse.c -o parse.o>
 Envp:<...>
 Re_executavel:<0>
 Host_Origem:<s2i>    Ident_modulo:<1>
 Horario_Migracao: < 725577925> secs.  < 614424> usecs.
 Horario_Reconh_Termino: <        -1> secs.  <     -1> usecs.
 Host_Destino:<s2d>
 Migrado:<1>          Id_processo:<1304>
 Horario_Inicio: <        -1> secs.  <     -1> usecs.
 Horario_Termino: <        -1> secs.  <     -1> usecs.
 Estado:<2>           Cod_term:<-1>
 (user mode) Secs:<        -1>  uSecs:<     -1>
 (system mode) Secs:<        -1>  uSecs:<     -1>

                ⋮

Servico ENVIAINFMOD
E[ 1]
 Nome:<cc>   Path:</bin>   Argv:< -c parse.c -o parse.o>
 Envp:<...>
 Re_executavel:<0>
 Host_Origem:<s2i>      Ident_modulo:<1>
 Horario_Migracao: < 725577925> secs.  < 614424> usecs.
 Horario_Reconh_Termino: < 725577929> secs.  < 485826> usecs.
 Host_Destino:<s2d>
 Migrado:<1>            Id_processo:<1304>
 Horario_Inicio: < 725577924> secs.  < 474989> usecs.
 Horario_Termino: < 725577928> secs.  < 238564> usecs.
 Estado:<3>             Cod_term:<0>
 (user mode) Secs:<         1>  uSecs:< 900000>
 (system mode) Secs:<         0>  uSecs:< 530000>

                ⋮
```

Figura 3.5: Histórico de um ServProc Agindo na Origem de uma Migração

```
                ⋮

Servico EXECREMMOD: Nome:<cc>   Path: </bin>  Args: < -c parse.c -o parse.o>

I[ 0]
 Nome:<cc>   Path:</bin>   Argv:< -c parse.c -o parse.o>
 Envp:<...>
 Re_executavel:<0>
 Host_Origem:<s2i>      Ident_modulo:<1>
 Horario_Migracao: < 725577925> secs.  < 614424> usecs.
 Horario_Reconh_Termino: <        -1> secs.  <     -1> usecs.
 Host_Destino:<s2d>
 Migrado:<1>            Id_processo:<1304>
 Horario_Inicio: <        -1> secs.  <     -1> usecs.
 Horario_Termino: <        -1> secs.  <     -1> usecs.
 Estado:<2>             Cod_term:<-1>
 (user mode) Secs:<        -1>  uSecs:<     -1>
 (system mode) Secs:<        -1>  uSecs:<     -1>

                ⋮

SIGCHLD do processo <1304>. impors[0]   Status:<0>

I[ 0]
 Nome:<cc>   Path:</bin>   Argv:< -c parse.c -o parse.o>
 Envp:<...>
 Re_executavel:<0>
 Host_Origem:<s2i>      Ident_modulo:<1>
 Horario_Migracao: < 725577925> secs.  < 614424> usecs.
 Horario_Reconh_Termino: <        -1> secs.  <     -1> usecs.
 Host_Destino:<s2d>
 Migrado:<1>            Id_processo:<1304>
 Horario_Inicio: < 725577924> secs.  < 474989> usecs.
 Horario_Termino: < 725577928> secs.  < 238564> usecs.
 Estado:<3>             Cod_term:<0>
 (user mode) Secs:<         1>  uSecs:< 900000>
 (system mode) Secs:<         0>  uSecs:< 530000>

Se informou a Origem

                ⋮
```

Figura 3.6: Histórico de um ServProc Agindo no Destino de uma Migração

3.5 Ambiente de Execução

No SGPC a unidade de código que é manipulada para ser submetida a migração e execução é o Módulo. A *granularidade* deste módulo é a de um arquivo executável independente. Existem algumas características do ambiente de execução dos módulos que serão detalhadas nos pontos que seguem.

- arquitetura de aplicação;
- interação;
- variáveis de ambiente;
- re-execução.

3.5.1 Arquitetura de Aplicação

A arquitetura de aplicação define quais códigos objeto podem ser executados em uma estação.

Cada estação de trabalho tem uma Arquitetura de Aplicação definida. Por sua vez, os códigos objeto gerados pelos diversos tradutores devem ser executados em uma máquina com uma arquitetura de aplicação específica. É condição necessária para um programa ser executado sobre uma estação que as Arquiteturas de Aplicação de ambos se correspondam.

O SGPC certifica-se de que os módulos sujeitos a migração sejam encaminhados para estações com a arquitetura de aplicação adequada para hospedá-los.

A aplicação usuária não tem que se preocupar com este aspecto, o SGPC garante que a estação de destino selecionada tenha a arquitetura de aplicação adequada; ainda no caso extremo que não exista outra estação na rede com as características necessárias, sempre será possível executar o módulo na própria estação de origem.

3.5.2 Interação

Normalmente existirão usuários trabalhando nas estações de destino escolhidas; estes usuários não devem ser perturbados em seu trabalho por causa dos módulos importados, e ainda mais, não precisam sequer tomar conhecimento da existência desses outros processos executando em seu próprio sistema. Logo, o SGPC tem que tomar emprestada a capacidade de processamento alheia sem interferir no trabalho dos usuários correntes nas estações de destino dos módulos.

O módulo migrado não pode fazer uso, na sua execução, de recursos interativos da Estação de Destino. Portanto, lhe é proibido realizar entradas interativas a partir do teclado. Analogamente, o uso do processador emprestado não inclui o uso do monitor de vídeo.

Em resumo, o SGPC deve garantir que os módulos migrados não realizem entradas nem saídas com o terminal da estação de destino.

Como foi apresentado na Seção 3.2, com a instalação do ServProc este se desliga do terminal de controle. Desta maneira o ServProc passa a ser líder de uma *sessão*.

Todos os módulos que o ServProc põe em execução são seus filhos, e como tais eles são também membros da *sessão* do servidor pai. Os mecanismos de herança do sistema operacional garantem que estes módulos (filhos) também não tenham terminal de controle associado, pelo que estão vetados de fazer operações de entrada e saída interativas. Os módulos migrados ficam em *background* durante a sua execução; isto garante parte da não interferência no trabalho do usuário na estação.

Entretanto estes módulos podem ter em seu código instruções de entrada e/ou saída interativas. A execução de uma instrução de entrada interativa em um módulo provocaria seu bloqueio (garantido pelo sistema operacional para processos executando em *background*).

No caso de saídas para a tela do terminal, optou-se por capturá-las e redirigí-las para arquivos que posteriormente podem ser consultados. O ServProc toma providências para que as possíveis operações dos módulos que tentem escrever tanto para a saída padrão (*stdout*) quanto para a saída de erros padrão (*stderr*) sejam redirigidas para arquivos:

```
f_saida = freopen(arq_saida, "w", stdout);
f_err   = freopen(arq_saida_err, "w", stderr);
```

Os nomes para estes arquivos são gerados automaticamente, garantindo que sejam únicos.

Outra versão mais elaborada do SGPC poderia tentar reproduzir todas as entradas e saídas interativas na estação de origem do módulo.

3.5.3 Variáveis de Ambiente

Outra característica conseguida pelo SGPC é a criação das variáveis de ambiente adequadas para o módulo. Na execução de cada programa, o sistema operacional põe à disposição deste um vetor de cadeias chamado de *environment*. Por convenção estas cadeias são da forma 'nome=valor', e o uso delas depende do programa em particular.

No SGPC a aplicação tem a possibilidade de fornecer através dos parâmetros do pedido de execução (Seção 3.1.3) o *environment* de execução que deseja (campo *.envp* da estrutura *mod*). Este *environment* é passado ao módulo na chamada de sua execução (parâmetro *envp* da primitiva *execve()*).

Entretanto, no SGPC Estendido existe uma facilidade adicional na criação do *environment*. O usuário tem a possibilidade de oferecer ou não esta informação, mas se o *environment* não é dado explicitamente o sistema reproduz automaticamente no destino as variáveis da origem com seus valores (Seção 3.8). O SGPC Estendido consegue isto fazendo uso do vetor de cadeias de *environment*, disponível através da variável *environ*, toda vez que as rotinas do SGPC Estendido fazem parte do processo usuário (foram ligadas com ele).

O SGPC Estendido garante, então, automaticamente as ações necessárias para impor as mesmas variáveis de *environment* (com seus valores) que existiam no ambiente do programa solicitante.

3.5.4 Re-Execução

Este aspecto é de interesse apenas no caso que fosse necessário começar a execução de um módulo mais de uma vez. No percorrer normal dos acontecimentos a execução do módulo deve chegar com êxito a seu término, desta forma não existe necessidade de recomeçar sua execução.

Embora o percorrer mais natural não envolva interrupções na execução de um módulo, sempre existe a possibilidade de que aconteçam condições excepcionais que impeçam seu término normal. A ocorrência de quedas de um servidor ou estação (Seção 3.7) são exemplos de condições excepcionais deste tipo.

O SGPC toma providências, ainda nestes casos, para conseguir a execução com êxito dos módulos. O sistema pode lançar a execução do módulo novamente, mas estas novas execuções não podem ser realizadas arbitrariamente. Existem programas que não obteriam os mesmos resultados em execuções repetidas como conseqüência dos próprios efeitos de suas execuções anteriores.

Pelo motivo assinalado acima, o SGPC possibilita à aplicação usuária indicar se o módulo pode ou não ser executado mais de uma vez (campo *.re_executavel* da estrutura *inf_mod*). A aplicação apenas tem que ocupar-se de dar esta informação no momento do pedido de execução. O SGPC se ocupa de detectar o término normal ou não da execução do módulo, e se necessário (e permitido), recomeçar sua execução.

3.6 Finalização de Módulos

A finalização da execução de um módulo (lançada a pedido de uma aplicação) dá término à satisfação do requerimento do usuário. O SGPC detecta quando isto acontece. O fato de que os módulos sejam lançados a executar como filhos do ServProc na Estação de Destino do módulo possibilita que o próprio Servidor seja informado do término do módulo através do mecanismo de sinais do Sistema Operacional.

O ServProc prepara as condições para que seu procedimento interno **fim_de_modulo()** seja chamado quando termina a execução de um módulo *importado* e executado localmente (processo filho do Servidor Local). A instrução a seguir estabelece (no momento da instalação do servidor) o procedimento a ser chamado automaticamente, com o término da execução de um módulo:

```
signal(SIGCHLD, fim_de_modulo);
```

Este procedimento de execução assíncrona é disparado através do mecanismo de sinais do sistema operacional. No caso é utilizado o sinal SIGCHLD:

SIGCHLD: sinal emitido quando um processo muda de estado, em especial enviado a seu processo pai quando termina sua execução.

No momento da sinalização deste evento é registrado o horário de término da execução do módulo (campo *.horário_termino*).

Quando termina a execução de um módulo as informações de seu estado devem registrar sua condição de **concluído**. O Servidor se encarrega também de captar o código de saída dessa ter-

- **inf_erro INFERRO(int) = 11:** dado como parâmetro um código de erro retorna uma estrutura com o próprio código e uma mensagem explicativa:

```
struct inf_erro

  {
    stamp  ident_serv;
    int    cod_erro;
    cadeia msg_erro;
  }  /* inf_erro */
```

Quedas (*Crashs*)

Como é conhecido, todo sistema é suscetível a falhas. Por alguma causa externa ou interna pode acontecer uma queda de um ServProc executando em uma estação, e até mesmo a queda da própria estação.

No SGPC foram tomadas precauções para a detecção de condições de exceção, assim como para a recuperação de algumas situações.

Cada ServProc em execução tem um Identificador único, criado por ele mesmo durante sua instalação. No momento em que se estabelece o primeiro contacto entre dois ServProcs, ambos se enviam seus respectivos Identificadores. Em qualquer tentativa de contacto posterior (iniciada por um destes ServProcs) em que se espera achar aquele outro como servidor realiza-se uma checagem de sua identidade. No caso de não ser satisfatória a verificação, o ServProc que fez o pedido assume que o Servidor original daquela estação não existe mais e toma providências em relação aos módulos migrados entre essas duas estações.

Se existem módulos importados no ServProc corrente que foram exportados pelo ServProc considerado em queda, a execução local destes módulos é cancelada.

Se o ServProc corrente exportou algum módulo para a estação em que aconteceu a queda, este tenta recuperar a situação recomeçando a execução do módulo. Esta decisão de executar ou não novamente fica determinada pelo campo *.re_executável* da estrutura de informações do módulo dada pelo usuário. Este dado indica se a execução do módulo tem ou não efeitos colaterais (*idempotência*). No caso negativo são posicionados o código e a mensagem de erro de forma que indiquem esta situação.

Outra possibilidade de considerar queda em um ServProc ocorre no caso de não se conseguir contactar com nenhum ServProc em uma estação remota, sendo que se conhecia que existia este executando nela. As provisões tomadas no caso são similares às de não coincidência na Identidade do Servidor de Processamento Remoto.

3.8 Interface de Alto Nível

Nesta seção é apresentado o módulo de Interface com as aplicações do SGPC, **InterfPC**. Este módulo é formado por um conjunto de subrotinas que oferecem outro nível de acesso às facilidades implementadas nos ServProcs.

O conjunto de operações na InterfPC habilita o usuário a ter acesso às informações referentes às estações e aos módulos, assim como a solicitar a execução de programas. A listagem completa da interface destas operações é apresentada no Apêndice D. O SGPC expandido por esta interface é chamado de Sistema Gerenciador de Processamento Cooperativo Estendido (SGPC Estendido).

A implementação das subrotinas da biblioteca baseia-se na existência de um Servidor de Processamento executando na estação local, e implementado com as convenções estabelecidas para agir como Servidor no paradigma Cliente/Servidor.

A comunicação entre Cliente e Servidor, neste caso, é feita através de chamadas remotas de procedimentos (RPCs), e como tal, o pacote de rotinas vai fazer o papel de Cliente desse Servidor Local, e vai fornecer às aplicações usuárias as operações, sem que elas tenham que mexer com as convenções do lado do Cliente. Desta maneira, as rotinas servem como intermediárias entre o Servidor e a aplicação usuária do Processamento Cooperativo, ocultando o modelo Cliente/Servidor segundo o qual são implementados os programas RPC.

Chamamos de *operações* às facilidades oferecidas pela InterfPC (utilizadas pelas aplicações usuárias do SGPC Estendido). Chamamos de *serviços* às facilidades oferecidas pelos ServProcs (utilizados por outros ServProcs, pelas rotinas da *Interface de Alto Nível*, e por aplicações usuárias desse nível).

Para a utilização da InterfPC, inclui-se o código objeto da biblioteca no processo de ligação do programa da aplicação usuária.

A aplicação usuária uma vez ligada junto com esta biblioteca está em condições de utilizar as operações do Processamento Cooperativo. Para isto, a aplicação deve cumprir algumas exigências quanto ao protocolo de uso, como, por exemplo, restringir o uso das operações ao tempo de vida da "Sessão de Processamento Cooperativo".

Os procedimentos desta biblioteca podem ser divididos nos seguintes grupos:

Ligação ao Servidor de Processamento Cooperativo: estabelecem o início e o fim de Sessões de Processamento Cooperativo;

Operações de Informações sobre as Estações: dão a possibilidade de que as aplicações tenham acesso às informações que o SGPC tem sobre quais são as estações na rede, e às informações de configuração e uso daquelas participando do Processamento Cooperativo;

Operações de Execução de Módulos: permitem que as aplicações iniciem a execução de módulos com o emprego de múltiplas estações da rede. Esta execução apresenta mecanismos automáticos de escolha de estações;

Operações de Informações sobre os Módulos: permitem o acesso das aplicações às informações que o SGPC manipula sobre os módulos migrados através do mesmo;

Diagnóstico de erros: mapeia códigos de erro em mensagens de erro.

Nas rotinas da InterfPC não se oferecem operações sobre a Identidade dos Servidores de Processamento, uma vez que o uso destes ServProcs fica totalmente transparente para as aplicações usuárias deste nível. Entretanto, o pacote de rotinas nesta biblioteca toma conta das verificações necessárias para manter a integridade do sistema. A InterfPC checa a Identidade dos ServProcs ao longo da Sessão de Processamento Cooperativo.

No que resta da seção serão apresentadas as operações que permitem abrir e encerrar uma Sessão de Processamento Cooperativo, assim como as operações que podem ser chamadas dentro de uma "Sessão".

Ligação ao ServProc

Neste grupo estão incluídos dois procedimentos para o estabelecimento e destruição da ligação entre a aplicação usuária e o SGPC. Eles delimitam o tempo em que é válido o *handler* do cliente. Estes procedimentos são:

- **void clnt_proc_coop()**: inicialização do processo que faz o pedido como processo administrado pelo SGPC . É necessário chamar este procedimento antes de fazer uso de qualquer outra subrotina deste pacote;
- **void fim_proc_coop()**: fim do uso do SGPC Estendido; libera a memória associada com as estruturas de dados utilizadas.

O emprego destas duas operações delimitam o tempo de vida do que se chama uma "Sessão de Processamento Cooperativo".

O uso da operação **clnt_proc_coop()** cria internamente a chave que dá acesso às demais operações de Processamento Cooperativo; esta operação estabelece os vínculos entre a aplicação e a InterfPC, e entre a InterfPC e o ServProc Local. Nestes vínculos a biblioteca das rotinas que constituem a InterfPC age como intermediária entre a aplicação e o Servidor. Para conseguir isto a biblioteca se faz Cliente deste Servidor Local com o uso do serviço **clnt_create()**.

A aplicação que faz o chamado desta operação fica gerenciada aos efeitos do Processamento Cooperativo pelo Servidor de Processamento da estação local.

A operação **fim_proc_coop()** faz uso do serviço **clnt_destroy()** e provoca o encerramento da "Sessão de Processamento Cooperativo". Como resultado do uso desta operação, se desliga a aplicação do seu Servidor gerenciador, e a partir de então esta não pode mais chamar operações do SGPC Estendido.

O usuário deve chamar esta função quando não vai mais utilizar as operações do SGPC Estendido, embora este último esteja preparado para desligá-lo automaticamente com o término da aplicação usuária. Para isso faz uso da função *on_exit()*.

As operações restantes nos outros grupos implementam chamadas aos procedimentos remotos do Servidor Local. Para isto, fazem uso do serviço **clnt_call()**, agindo como cliente do ServProc. As estruturas de dados dos parâmetros e dos resultados destas operações podem ser consultadas na Seção 3.1.

Operações de Informações sobre as Estações

As operações contidas neste grupo são:

- **ids_host listahosts(/*void*/)**: pedido das informações sobre quais são todos os *hosts* na rede;
- **inf_host pedeinfhost(/* cadeia nome_host */)**: pedido das informações associadas a uma estação. Inclui configuração fixa e dados de estado.

Estes pedidos são transferidos pela biblioteca ao Servidor Local. Este, por sua vez, contacta com ServProcs remotos, se for necessário, e devolve as informações requisitadas.

Operações para a Execução de Módulos

As operações neste grupo são:

- **id_mod execmod(/* mod dados_modulo */)**: pedido de execução cooperativa; implica na escolha automática da estação de destino da migração;
- **id_mod execmodhost(/* mod_host dados_modulo */)**: pedido de execução cooperativa com *host*. É similar à anterior, exceto que a aplicação indica a estação de destino desejada.

Em ambas as operações a aplicação usuária tem a opção de fornecer ou não uma cadeia com as variáveis de *environment* e seus valores. Colocando a cadeia vazia no campo *.envp* o SGPC Estendido encarrega-se automaticamente de estabelecer um *environment* similar ao que tinha o processo que pede a execução do módulo (Seção 3.5).

O SGPC Estendido consegue isto fazendo uso do vetor de cadeias de *environment*, disponível através da variável *environ*, toda vez que as rotinas do SGPC Estendido fazem parte do processo usuário (foram ligadas com ele).

Operações de Informações sobre os Módulos

As operações no grupo apresentado nesta seção permitem que as aplicações possam obter as informações sobre os módulos em execução, em particular dos módulos "exportados" e "importados" pelo Servidor Local.

Estas operações são:

- **ids_mods listamods(/* void */)**: pede os Identificadores de todos os módulos migrados através da Estação Local (inclui Importados e Exportados);
- **inf_mod pedeinfmod(/* int id_modulo */)**: pedido do estado de um módulo em andamento (pode ser importado ou exportado da estação local).

Diagnóstico de Erros

Todas as estruturas de dados retornadas pelas operações do SGPC Estendido incluem um campo *.cod_erro* com um código de erro que reflete o sucesso ou não da operação. Em caso de que este valor seja diferente de ERRNO (0), é sinal da ocorrência de alguma condição errônea. Com a ajuda da seguinte operação podem ser obtidas as mensagens de erro que se correspondem com esses códigos.

- **inf_erro inferro(/* int */)**.

No próximo capítulo são apresentados exemplos de utilização do Sistema Gerenciador de Processamento Cooperativo. Alguns destes exemplos estão acompanhados de medições de tempos de execução, e de comparações dos tempos empregados em variantes distribuídas e não distribuídas das mesmas tarefas.

Capítulo 4

Exemplos de Aplicações

O SGPC oferece serviços que possibilitam distribuir execuções ao longo da rede. Estes serviços são úteis para aplicações que precisem disparar a execução em lote de vários programas, e especialmente quando esses programas vão fazer um uso intensivo da CPU.

Neste capítulo apresentam-se exemplos práticos de utilização do SGPC. Primeiro, na Seção 4.1, é apresentada uma versão de *make distribuído*. São mostrados exemplos de compilações realizadas através deste (Seção 4.1.2), assim como medições de tempo de compilações. São apresentadas ainda comparações dos tempos de execução de variantes distribuídas e não distribuídas das mesmas tarefas (Seção 4.1.3).

Na Seção 4.2 apresenta-se um exemplo de programa que faz uso intensivo do processador. Este exemplo calcula os números primos em um intervalo. Na própria seção são mostrados os tempos de execuções locais e distribuídas para a solução desse problema.

4.1 *Make Distribuído*

Na elaboração deste exemplo de aplicação, trabalhou-se a partir do código fonte de um programa *make*. Foram introduzidas modificações para que os pedidos de execução dos comandos que constituem as regras fossem encaminhados através do SGPC. Este *make* modificado tem sido utilizado na compilação de conjuntos de arquivos fonte em C, fazendo com que as compilações sejam realizadas de maneira paralela.

4.1.1 Características do *Make*

O *make* é um utilitário que mantém, atualiza e gera programas e arquivos relacionados. Este utilitário utiliza um arquivo chamado de *makefile* que contém entradas descrevendo como atualizar *targets* em relação a dependências. As dependências por sua vez podem ser *targets*, de maneira que o *make* verifica recursivamente cada *target*.

Depois de ter processado as dependências, o *make* decide construir um *target* se ele não existe,

ou reconstruí-lo se é mais velho que alguma de suas dependências. Na construção do *target* é executada uma lista de comandos do sistema operacional associados com este, chamada **regra**.

O *make* do qual partimos no desenvolvimento deste exemplo foi o *gymake*, desenvolvido por Greg Yachuk, da Informix Software Inc. [Yac], e colocado em domínio público. Segundo seu autor, é uma implementação muito próxima ao *make* existente nos sistemas *Sun*.

Primeiro descreveremos alguns aspectos da sintaxe do arquivo de descrição das dependências (*makefile*).

Os *makefiles* podem conter uma mistura de entradas de comentários, entradas de definições de macros, entradas de *include*, e entradas *target*. São estas últimas as que interessam para nossos fins.

Uma entrada *target* no *makefile* tem o seguinte formato:

```
<nome de arquivo> ... : [ <nome de arquivo> ...]
        [ <regra> ]

              ⋮
```

Os nomes de arquivos que aparecem no inicio da linha são os chamados **targets**; os nomes de arquivos após os dois pontos definem as **dependências**. É a partir destes arquivos que o *target* é reconstruído com a execução das **regras** (comandos) das linhas seguintes.

Modificações realizadas no programa *make*

A versão original deste *make* estava concebida para fazer cumprir todas as regras seqüencialmente. Para o exemplo aqui apresentado ele foi modificado para suportar a execução das regras concorrentemente.

O *make* modificado foi utilizado na compilação de um conjunto de arquivos, de forma que as compilações fossem realizadas de maneira paralela.

Ao invés de, como no *make* usual, as regras resultarem na submissão de programas diretamente ao sistema operacional para execução, no *make distribuído* são geradas chamadas ao serviço **EXECMOD()** no SGPC para execução concorrente daquele mesmo programa.

Assim, por exemplo, cada chamada ao compilador C foi substituída pela chamada ao serviço **EXECMOD()**, sendo o compilador C o módulo a ser executado:

```
r_execmod_1 = (id_mod *) lexecmod_1(e_execmod_1, cl_user);
```

As modificações feitas no *make* tem como objetivo apenas sua utilização como um exemplo da distribuição das execuções para diferentes estações, e sua realização em paralelo. Uma versão definitiva de um *make distribuído* precisaria introduzir modificações na sintaxe do arquivo *makefile*, através da qual fosse possível indicar quais regras podem ser executadas em paralelo e quais não. Na versão utilizada para o exemplo supõe-se que *todas* as regras podem ser executadas em paralelo.

Foram também criadas, no *make* modificado, novas estruturas de dados para manter um registro dos comandos lançados e uma indicação de seu estado terminado ou não.

Nas seções seguintes apresentam-se dados de compilações utilizando este *make distribuído.*

4.1.2 Compilações Distribuídas de Conjuntos de Arquivos

O arquivo *makefile* utilizado neste exemplo é mostrado na Figura 4.1. Com a utilização deste *makefile* são feitas as compilações do próprio código fonte do *make*. Fragmentos deste exemplo tem sido apresentados ao longo do trabalho.

```
# makefile.ex1    Makefile para teste

make.o:     make.c
            /bin/cc -c make.c -o make.o

parse.o:    parse.c
            /bin/cc -c parse.c -o parse.o

tstring.o: tstring.c
            /bin/cc -c tstring.c -o tstring.o
```

Figura 4.1: Exemplo de *makefile*

A execução do *make* para a reconstrução destes três arquivos *targets* provoca o lançamento de três compilações, e para cada uma delas é preciso decidir em que estação realizar o processamento:

```
make -f makefile.ex1 make.o parse.o tstring.o
```

A história das transações consta no Apêndice F: O histórico criado pelo ServProc Local (na estação s2i – Apêndice F.1) mostra a instalação do próprio servidor, o recebimento dos pedidos de execução dos comandos, a decisão tomada quanto à escolha das estações de destino para cada, e finalmente o recebimento da indicação do término dessas execuções.

Na instalação, o ServProc (na estação s2i) detecta a existência de ServProcs em outras três estações (s2d, s2e e marumbi). Na ordem de seleção de estações de destino estabelecida logo após a instalação, as estações aparecem na seqüência s2i, s2d, s2e e marumbi, segundo suas porcentagens de tempo de CPU ocioso.

O primeiro pedido de execução recebido é o da compilação do arquivo *make.c*, sendo que a própria estação local (s2i) é selecionada para a execução do módulo. Assim sendo, nas tabelas de módulos exportados e importados do ServProc Local são inseridos (nas suas primeiras entradas) os dados relativos a este módulo.

O segundo pedido de execução recebido é o da compilação do arquivo *parse.c*, sendo que desta vez é selecionada a estação s2d para a execução do módulo. Para realizar a migração do módulo, é contactado o ServProc nessa estação de destino e chamado seu serviço **EXECREMMOD()**. As informações do módulo são refletidas na tabela de módulos exportados do ServProc de origem.

O terceiro pedido de execução recebido é o da compilação do arquivo *tstring.c*. Neste caso, é selecionada a estação s2e como destino da migração e contacta-se o serviço **EXECREMMOD()** de s2e. As informações do módulo são refletidas também na tabela de módulos exportados do ServProc Local em sua primeira entrada livre.

Quando concluídas as execuções dos módulos nos destinos, o ServProc de origem em s2i é informado através do serviço **ENVIAINFMOD()**. Com as informações recebidas por esta via, é feita a atualização da entrada correspondente ao módulo exportado que terminou. Os novos dados refletem o Horário do Término e de seu Reconhecimento pela origem, assim como o código de saída do módulo executado remotamente.

Neste exemplo, o ServProc em s2i (a origem) é comunicado primeiro do término do módulo migrado para s2e, e logo depois do término do módulo migrado para s2d. Finalmente termina a execução do módulo que se fez localmente. O término deste último é reconhecido com o recebimento do sinal **SIGCHLD**.

Por sua vez, nos arquivos de história gerados pelos ServProcs nas estações de destino (s2d e s2e) pode-se apreciar os recebimentos dos pedidos remotos.

No histórico do ServProc residente na estação s2d (Apêndice F.2), pode-se constatar o recebimento do pedido da compilação remota do arquivo *parse.c*. Este pedido é recebido através do serviço **EXECREMMOD()**. Os dados desse módulo são refletidos então na tabela de módulos importados deste ServProc de destino. No término da execução do módulo o ServProc Local é informado através do sinal **SIGCHLD**, e este por sua vez se encarrega de informar ao ServProc de origem (em s2i) através do serviço **ENVIAINFMOD()**.

O histórico do ServProc residente na estação s2e é semelhante ao da estação s2d e por isto não foi incluído no Apêndice F.

4.1.3 Dados de Tempos de Compilações Distribuídas

Para a realização de outros testes, com o objetivo de realizar medições de tempo, foram criados arquivos com programas artificiais em C com diferentes tempos de compilação. Nesta seção apresentamos os resultados obtidos em compilações de arquivos que demoram aproximadamente 28 segundos cada uma em uma estação de trabalho **Sun SPARCstation 1+**. Estes 28 segundos correspondem ao tempo de compilação do arquivo fonte com a utilização direta do compilador C [Sun89], e referem-se a tempo de uso do processador (tempo de CPU).

Este arquivo fonte tem 4900 linhas de código. Para realizar compilações múltiplas foram criadas seis cópias do mesmo (cod28segs.1.c até cod28segs.6.c). É de assinalar que no sistema utilizado (rede do DCC/IMECC) todos os arquivos estão localizados remotamente e são acessados através de um servidor de arquivos.

O conteúdo do arquivo *makefile* utilizado nas compilações, *makefile.cod28segs*, é o seguinte:

```
cod28segs.1.o : cod28segs.1.c
            /bin/cc -c cod28segs.1.c -o cod28segs.1.o
cod28segs.2.o : cod28segs.2.c
            /bin/cc -c cod28segs.2.c -o cod28segs.2.o
cod28segs.3.o : cod28segs.3.c
            /bin/cc -c cod28segs.3.c -o cod28segs.3.o
cod28segs.4.o : cod28segs.4.c
            /bin/cc -c cod28segs.4.c -o cod28segs.4.o
cod28segs.5.o : cod28segs.5.c
            /bin/cc -c cod28segs.5.c -o cod28segs.5.o
cod28segs.6.o : cod28segs.6.c
            /bin/cc -c cod28segs.6.c -o cod28segs.6.o
```

Foram realizadas compilações de conjuntos de um, três e seis arquivos cod28segs, e variantes destas compilações com o compilador C diretamente, através do *make* original, através do *make distribuído* com o ServProc em uma única estação, e através do *make distribuído* com o ServProc em três e seis estações.

Nas diferentes variantes de compilações foram medidos tempos de dois tipos:

- de utilização do processador;
- de *relógio de parede* ou real.

O primeiro destes reflete o tempo que o programa (compilador C) esteve utilizando o processador, enquanto o segundo representa o intervalo desde o "pedido" da compilação até seu término. Este segundo valor é sempre maior que o primeiro devido à concorrência pelo uso do processador por vários processos, o que o torna também muito mais variável. É, entretanto, o de maior relevância para o usuário porque é o que determina a demora em obter o resultado desejado.

As medições de tempo foram todas realizadas com a ajuda do comando /usr/bin/time. Este recebe como argumento um outro comando e oferece após o término deste último um sumário do tempo de sua execução. O diagnóstico assim obtido inclui três tempos:

- o real desde o começo até o término do comando;
- o de uso do processador pelo sistema;
- o de uso do processador pelo comando.

Cada compilação foi realizada no mínimo três vezes e em cada caso foram calculados valores de tempo mínimo, máximo e médio. Estes tempos variam significativamente segundo o método de compilação empregado. Todos os tempos são dados em segundos.

Compilação de três arquivos de 28 segundos cada

As compilações com medições de tempo foram realizadas nas seguintes variantes:

- **A:** Um arquivo independente com chamada direta ao compilador C:

  /usr/bin/time cc -c cod28segs.c -o cod28segs.o

- **B:** Os três arquivos em uma chamada ao *make* original:

  /usr/bin/time makeorig -f makefile.cod28segs cod28segs.1.o cod28segs.2.o cod28segs.3.o

- **C:** Os três arquivos em uma chamada ao *make distribuído* com o ServProc instalado em uma única estação (todas as compilações foram locais):

  /usr/bin/time makedistr -f makefile.cod28segs cod28segs.1.o cod28segs.2.o cod28segs.3.o

- **D:** Os três arquivos em uma chamada ao *make distribuído* com vários ServProcs instalados (as compilações migraram para três estações diferentes):

  /usr/bin/time makedistr -f makefile.cod28segs cod28segs.1.o cod28segs.2.o cod28segs.3.o

Os tempos reais das compilações dos três arquivos, nas diferentes variantes, são mostrados na tabela da Figura 4.2.

	Mínimo	Máximo	Médio
A	42.40	48.00	44.33
B	82.70	87.90	84.87
C	112.90	116.20	113.90
D	47.00	67.80	56.78

Figura 4.2: Tempos Reais da Compilação de um Conjunto de Três Arquivos

O caso das compilações todas locais através do *make distribuído* é, como seria de esperar, o de pior desempenho. O tempo aumenta consideravelmente com relação ao do *make* convencional, devido ao *overhead* adicional, natural neste caso, do SGPC. Note-se entretanto que, mesmo neste caso, o tempo total é inferior ao triplo do tempo da compilação isolada.

O resultado do uso do *make distribuído* com migrações reais das compilações evidencia melhora no desempenho.

O tempo de compilação de um módulo independente é maior ao migrar esta do que ao ser feita localmente, mas o tempo de compilação do conjunto é, naturalmente, melhor que o tempo necessário para a compilação seqüencial na mesma máquina: Cada uma das compilações foi realizada em uma máquina diferente, com paralelismo real, e não competindo pelo mesmo processador. A simultaneidade faz com que o tempo de compilação do conjunto melhore.

É interessante notar que a compilação d 3 arquivos utilizando o *make original*, na mesma máquina, requer tempo da ordem de dois vezes o tempo de compilação isolado, ao invés de cerca de três vezes, como sería de se esperar. Isto pode ser explicado se levamos em conta uma série de detalhes do sistema operacional e de rede utilizados: reentrância do compilador, tempo de carga deste, maneira utilizada pelo *make* para invocar as cópias do compilador, etc.

Compilação de seis arquivos de 28 segundos cada

As compilações com medições de tempo foram realizadas nas seguintes variantes:

- **A:** Um arquivo independente com chamada direta ao compilador C:

  ```
  /usr/bin/time cc -c cod28segs.c -o cod28segs.o
  ```

- **B:** Os seis arquivos em uma chamada ao *make* original:

  ```
  /usr/bin/time makeorig -f makefile.cod28segs cod28segs.1.o cod28segs.2.o cod28segs.3.o
                                               cod28segs.4.o cod28segs.5.o cod28segs.6.o
  ```

- **C:** Os seis arquivos em uma chamada ao *make distribuído* com o ServProc instalado em uma única estação (todas as compilações foram locais):

  ```
  /usr/bin/time makedistr -f makefile.cod28segs cod28segs.1.o cod28segs.2.o cod28segs.3.o
                                                cod28segs.4.o cod28segs.5.o cod28segs.6.o
  ```

- **D:** Os seis arquivos com uma chamada ao *make distribuído* com vários ServProcs instalados (as compilações migraram para três estações diferentes):

  ```
  /usr/bin/time makedistr -f makefile.cod28segs cod28segs.1.o cod28segs.2.o cod28segs.3.o
                                                cod28segs.4.o cod28segs.5.o cod28segs.6.o
  ```

- **E:** Os seis arquivos com uma chamada ao *make distribuído* com vários ServProcs instalados (as compilações migraram para seis estações diferentes):

  ```
  /usr/bin/time makedistr -f makefile.cod28segs cod28segs.1.o cod28segs.2.o cod28segs.3.o
                                                cod28segs.4.o cod28segs.5.o cod28segs.6.o
  ```

Os tempos reais das compilações dos seis arquivos, nas diferentes variantes, são mostrados na tabela da Figura 4.3.

	Mínimo	Máximo	Médio
A	42.40	48.00	44.33
B	167.80	170.10	168.63
C	367.80	745.30	555.17
D	90.70	107.50	97.96
E	79.50	173.80	122.50

Figura 4.3: Tempos Reais da Compilação de um Conjunto de Seis Arquivos

Novamente o pior caso é quando todas as compilações são realizadas localmente com o emprego do *make distribuído*. O detrimento no desempenho se faz sentir em maior proporção com seis módulos ao que com os três apresentados anteriormente.

É de assinalar que estes exemplos de compilações fazem uso relativamente intenso do sistema de arquivos (em maior medida o de seis compilações) o que introduz demoras que não dependem apenas do processamento na CPU. Note-se que cada compilação resulta na escrita em disco de um arquivo objeto de tamanho considerável.

É de supor que, nestes casos de programas com considerável atividade de entrada/saída, uma série de outros fatores tornam-se importantes e mesmo dominantes à medida que aumenta o número de execuções paralelas: no sistema utilizado todas as escritas em disco competem por um único servidor de arquivos central e todo o tráfego passa por uma única rede *Ethernet* de 10 Mbps.

Novamente também o desempenho melhora com a distribuição real das compilações. Ao aumentar o número das estações de destino de três para seis é mantida a melhora em relação às variantes locais. Mas não se pode afirmar invariavelmente que a melhoria é proporcional à quantidade de estações. Provavelmente devido aos fatores acima identificados e contrariamente ao que inicialmente se poderia esperar, em média é pior executar os seis módulos em seis estações ao invés de executar os seis módulos em três estações (dois em cada).

Note-se ainda que, a medida que aumenta o número de estações concorrentes, vai-se tornando mais crítica a característica de "pior caso" inerente ao processo: o tempo total é determinado pela estação que mais tempo leva para terminar, e quanto maior o número de estações usadas em paralelo, maior a probabilidade de se ter exportado módulos para estações já bastante "ocupadas".

O tráfego na rede é também um fator alheio ao uso puro do processador que afeta os tempos totais de execução.

Os testes realizados com outros códigos fontes mostraram resultados similares. Na Seção 4.2 a seguir apresenta-se um exemplo de aplicação em que predomina o uso intensivo do processador.

4.2 Aplicações com Uso Intensivo do Processador

Os requisitos computacionais de alguns problemas podem exigir um uso intensivo do processador. Neste caso, dizemos que este limita a aplicação (*CPU Bound Application*). Existem, ainda, aplicações que são limitadas devido a um grande número de transações de Entrada/Saída (*Input/Output Bound Application*).

O SGPC desenvolvido neste trabalho visa aumentar a velocidade de processamento global em benefício das aplicações com maior demanda por poder de processamento. Este seria o caso de muitas aplicações em engenharia e ciência que exigem quantidades de tempo consideráveis.

Para a realização de testes com o objetivo de efetuar medições de tempo de alguma aplicação que faça uso intensivo do processador, foi selecionado o problema de calcular números primos.

Implementou-se um programa básico que calcula os números primos em um intervalo. Este programa recebe como argumentos os valores mínimo e máximo dos extremos do intervalo. O algoritmo empregado no cálculo dos números primos é bem ineficiente e *"naive"* pois se queria apenas um exemplo simples de programa *CPU Bound*. Aplicações deste tipo, bem mais sofisticadas, serão sugeridas nas conclusões.

Foi desenvolvido um programa principal (Figura 4.4) que, para calcular os números primos em um intervalo, faz uma divisão deste em três subintervalos e chama três instâncias do programa básico. Este programa principal faz uso do SGPC fazendo com que o cálculo dos números primos em cada intervalo constitua um módulo independente possível de ser migrado para sua execução.

Nesta seção, apresentamos as medições de tempo obtidas no cálculo dos números primos. São apresentados os dados de tempo das variantes, utilizando diretamente o programa básico para o intervalo maior, e para cada um dos três intervalos por separado. Também apresentamos os dados referentes à utilização do SGPC com uma e três estações como destino das migrações. Todos os tempos, em segundos, foram medidos pelo comando **/usr/bin/time** empregado nos exemplos da Seção 4.1.3.

```
#include <rpc/rpc.h>
#include "pcoop.h"

                :

  mod     modulo1, modulo2, modulo3;
  id_mod  *r_execmod1, *r_execmod2, *r_execmod3;
  int     ident1, ident2, ident3;
  inf_mod *r_pedeinfmod1, *r_pedeinfmod2, *r_pedeinfmod3;

                :

   cl_user = clnt_create(hostatual, PROCCOOP, PROCCOOPVERS, "tcp");
   if (cl_user == NULL)
     { printf("\Nao se pode contactar ao ServProc\n");
       _exit(1);
     }
  strcpy(modulo1.path, "primos1");
  strcpy(modulo1.argv, "1 4750\0");
  strcpy(modulo1.envp, "\0");
  modulo1.re_avalia = 0;
  modulo1.re_executavel = 1;

  strcpy(modulo2.path, "primos2");
  strcpy(modulo2.argv, "4750 6800\0");

                :

  strcpy(modulo3.path, "primos3");
  strcpy(modulo3.argv, "6800 8375\0");

                :

  r_execmod1 = (id_mod *) lexecmod_1(modulo1, cl_user);
  if (r_execmod1)
    { if (r_execmod1->cod_erro == 0)
        { ident1 = r_execmod1->ident_modulo;
          printf(\"Identif Modulo1: <%d>\n", ident1);
        }
      else { ident1 = NOEXISTE; }
    }
  else { printf("\nProblemas com o servico EXECMOD"); }

                :
```

Figura 4.4: Aplicação Usuária do SGPC para o Cálculo de Números Primos (Parte I)

```
                    ⋮

r_execmod2 = (id_mod *) lexecmod_1(modulo2, cl_user);
if (r_execmod2)
  { if (r_execmod2->cod_erro == 0)
      { ident2 = r_execmod2->ident_modulo;
        printf(\"Identif Modulo2: <%d>\n", ident2);
      }
    else { ident2 = NOEXISTE; }
  }
else { printf("\nProblemas com o servico EXECMOD"); }

r_execmod3 = (id_mod *) lexecmod_1(modulo3, cl_user);

                    ⋮

r_pedeinfmod1 = (inf_mod *) lpedeinfmod_1(&ident1, cl_user);

                    ⋮
```

Figura 4.4: Aplicação Usuária do SGPC para o Cálculo de Números Primos (Parte II)

Para a realização dos testes apresentados, selecionou-se um intervalo e uma divisão do mesmo de forma que o cálculo para cada subintervalo demora aproximadamente 30 segundos. Os tempos reais médios obtidos para cada uma destas variantes são mostrados na Figura 4.5.

Intervalos		Tempos
Mínimo	Máximo	(segs)
1	4750	29.6
4750	6800	30.3
6800	8375	30.1
1	8375	90.70

Figura 4.5: Tempos Reais de Execução do Cálculo dos Números Primos por Intervalos

Os tempos de execução reais para o cálculo dos números primos no intervalo 1 até 8375 foram calculados com três abordagens distintas, utilizando:

- **B:** O programa básico:

  /usr/bin/time primos 1 8375

- **C:** O SGPC com uma única estação como destino das migrações (todas as execuções para o cálculo nos subintervalos foram locais):

  /usr/bin/time main_primos

- **D:** O SGPC com três estações como destino das migrações (a execução para o cálculo de cada subintervalo migrou para uma estação diferente):

  /usr/bin/time main_primos

Foi realizado ainda um teste adicional com estes programas, onde foram provocadas migrações de seis módulos para seis estações diferentes como destino. Para isto, foi executado o programa básico seis vezes com duas cópias para cada subintervalo. No caso, o objetivo foi obter medições de tempo que mostraram a variação no desempenho com o aumento do número de estações de destino com uma aplicação *CPU Bound*.

Estes tempos são apresentados na tabela da Figura 4.6, sendo que a última linha (E) corresponde ao teste adicional.

	Mínimo	Máximo	Médio
B	89.90	90.20	90.70
C	92.80	96.40	95.17
D	33.9	34.0	33.97
E	35.6	43.9	40.00

Figura 4.6: Tempos Reais de Execução do Cálculo dos Números Primos do Intervalo Maior

Os resultados confirmam novamente o esperado: a execução local de todos os módulos através do SGPC é a pior opção, embora a desvantagem em tempo não seja muito significativa. Nos casos D e E confirma-se também a notável economia de tempo com o aproveitamento da capacidade de migrações do SGPC.

Diferentemente dos testes realizados com aplicações que fazem uso intensivo de operações de entrada e saída com arquivos, a melhora no desempenho nas aplicações *CPU Bound* é diretamente proporcional à quantidade de estações de destino. No caso específico do teste adicional, observa-se uma melhoria no desempenho muito superior ao sêxtuplo do tempo de execução do programa básico.

Capítulo 5

Conclusões

O objetivo deste trabalho de dissertação foi o desenvolvimento de uma ferramenta prática que permite a utilização de uma rede de computadores como um recurso computacional simples. Foram realizados o projeto e a implementação de um Sistema Gerenciador de Processamento Cooperativo (SGPC).

O SGPC obtido provê um ambiente que facilita ao usuário a exploração da capacidade de paralelismo inerente à arquitetura de rede. No desenvolvimento do sistema foram implementados:

- um servidor de processamento cooperativo (oferece serviços), e
- uma biblioteca de rotinas para processamento cooperativo (oferece operações).

O Servidor de Processamento (ServProc) foi desenvolvido como um *"daemon"*. As interações com este seguem o paradigma do Modelo Cliente/Servidor através de Chamadas Remotas de Procedimentos (RPCs).

Os serviços prestados por este ServProc podem ser usufruídos pelas aplicações usuárias em dois níveis diferentes:

- Sistema Gerenciador de Processamento Cooperativo (SGPC);
- Sistema Gerenciador de Processamento Cooperativo Estendido (SGPC Estendido);

Em ambos os níveis realizam-se execuções remotas com seleção automática da máquina hospedeira, e garante-se a transparência destas operações para os usuários do sistema.

O funcionamento integrado de vários ServProcs constitui o Sistema Gerenciador de Processamento Cooperativo (SGPC). O SGPC é extensível, sendo que a quantidade de ServProcs cooperando pode variar dinamicamente.

Os Servidores nas diversas estações operam concorrentemente, trabalhando para conseguir uma distribuição que equilibre o uso das estações de trabalho. A escolha das máquinas de destino das migrações baseia-se no nível de ociosidade da capacidade de processamento destas.

Na utilização do SGPC, as aplicações interagem diretamente com o Servidor de Processamento Local, segundo as convenções do Modelo Cliente/Servidor.

A utilização do SGPC visa trazer os seguintes benefícios:

- Aproveitar a provável ociosidade de sistemas na rede;
- Utilização transparente da capacidade de processamento esparsa na rede;
- Aumentar a velocidade de processamento global;
- Melhorar a relação custo/desempenho em relação à utilização de um computador mais rápido e mais custoso;
- Eliminar a dependência de aplicações em rede de uma configuração específica;
- Escolher automaticamente a estação destino da migração;
- Prover execução remota, com baixo custo de implementação para as aplicações;
- Garantir transparência para as aplicações quanto à execução local ou remota e quanto às migrações;
- Aumentar o nível de *conectividade* da rede;
- Facilitar o desenvolvimento de aplicações paralelas.

Com a utilização do SGPC passa-se de um ambiente de estações de trabalho distribuídas a um pseudo ambiente de multiprocessadores, fazendo uso da capacidade de processamento distribuída, e dando lugar a um *"processamento cooperativo"*.

Para obter o SGPC Estendido, o SGPC foi ampliado com uma interface de maior nível, chamada de *Highest Level do RPC* na documentação da *Sun*. As aplicações neste nível interagem com rotinas em uma biblioteca e com um *front end C*. Estas rotinas dão acesso indireto aos serviços de *Processamento Cooperativo*.

O SGPC Estendido acrescenta ainda, outros benefícios para o usuário, entre estes:

- Oculta o mecanismo de RPC;
- Isola do modelo cliente/servidor;
- Interação simples com C *front end*;
- Garante a checagem de condições excepcionais transparentemente;
- Estabelece automaticamente as variáveis de ambiente.

As medidas realizadas com um *make distribuído* simplificado e com uma aplicação *CPU Bound* mostraram a viabilidade da utilização prática e as vantagens que o SGPC pode oferecer aos usuários da rede. Obviamente, por razões práticas, as medidas limitaram-se a programas relativamente rápidos em que dificilmente se justificaria o uso do SGPC. Mas fica claro o potencial do sistema para aplicações paralelizáveis em que os tempos reais de cada passo sejam da ordem de dezenas de minutos ou horas.

Os experimentos com o *make distribuído* mostraram ainda a pesada influência, na arquitetura utilizada, da centralização das operações de entrada/saída devido à utilização de uma rede do tipo *Ethernet* e de um servidor de arquivos central. Nestes casos, os ganhos representados pelo paralelismo são inferiores e saturam com um certo número de estações cooperantes, se comparados às aplicações *CPU Bound*.

5.1 Possíveis Trabalhos Futuros

Trabalhos posteriores, a partir da ferramenta desenvolvida, podem dirigir-se ao enriquecimento da mesma, assim como a sua utilização por aplicações usuárias. As seções seguintes enfocam as possíveis evoluções nestes dois sentidos.

5.1.1 Evolução do SGPC

O sistema apresentado pode ser um ponto de partida no desenvolvimento de uma ferramenta mais elaborada. Alguns aspectos que podem ser abordados neste sentido são apresentados a seguir:

- **Permitir interação nos módulos migrados, encaminhando-a à estação de origem:** Na implementação atual do SGPC, os módulos migrados não podem realizar entradas e saídas interativas a partir de qualquer terminal. Isto foi decidido como uma solução para não perturbar os usuários trabalhando nas estações de destino. No caso de saídas para a tela do terminal, estas são capturadas e redirigidas para arquivos. Outra versão mais elaborada do SGPC poderia tentar reproduzir todas as entradas e saídas interativas na estação de origem do módulo. Isto aumentaria a flexibilidade do SGPC, permitindo às aplicações se beneficiarem de E/S interativas;

- **Desenvolver uma interface interativa para uso do SGPC:** A utilização do SGPC neste momento é feita através de programas (tanto com o SGPC quanto com o SGPC Estendido). Seria desejável implementar uma interface que aceite pedidos interativos para executar programas e que estes possam ser executados remotamente, ficando a interface encarregada da interação com o SGPC;

- **Acrescentar facilidades de comunicação entre os módulos:** Na versão atual do SGPC, não é oferecido nenhum mecanismo para o intercâmbio de informações entre os módulos em execução em diferentes máquinas. Qualquer mecanismo de comunicação entre estes processos em execução, deve garantir que a aplicação usuária continue, sem consciência de qual é a

estação em que está sendo executado cada módulo. Facilidades neste sentido beneficiariam as possibilidades de interação entre tarefas paralelizadas;

- **Melhorar os critérios de seleção da estação hospedeira:** A seleção automática da estação hospedeira de um módulo migrado baseia-se, no momento, na porcentagem de tempo ocioso de sua CPU. A escolha de uma estação para a execução de um módulo leva em consideração este nível de ociosidade, além das migrações anteriores realizadas por este servidor. Tanto o critério de avaliação do nível de utilização da CPU de uma estação, quanto o critério de seleção da estação hospedeira podem ser reconsiderados. No caso, optou-se por uma variante simples. Melhorar a escolha da estação de destino deve garantir uma maior disponibilidade desta, para a execução mais rápida da tarefa;

- **Realizar rearbitragem dinâmica:** Trata-se da possibilidade de interromper tarefas já iniciadas e migrá-las para outros sistemas. Na presente implementação do SGPC, este executa novamente o módulo, apenas no caso de aconteçerem condições excepcionais (quedas de um servidor ou estação) que impeçam seu término normal. Uma versão posterior do SGPC poderia provocar a parada da execução do módulo, ainda que esta não tenha sido interrompida, e recomeça-la em uma nova estação de destino. Esta decisão poderia ser tomada ante um atraso inesperado na execução do módulo, que permita supor que sua execução, em outra estação, chegaria ao término mais rápido. O ganho na velocidade da chegada ao término desse módulo em particular, poderia ocasionar o término mais rápido de toda a aplicação;

- **Migração múltipla:** Em caso de exceso de estações ociosas disponíveis em relação ao número de módulos a migrar, um mesmo módulo podería também ser migrado para mais de uma estação, estabelecendo-se uma "corrida". Seriam utilizados os resultados da primeira estação em completar o módulo e abortadas as execuções dos demais "competidores".

5.1.2 Aplicações Usuárias

As aplicações usuárias que podem tirar maior proveito do SGPC são aquelas com as seguintes características:

- *CPU Bound*;
- paralelizáveis;
- consumidoras de grande quantidade de tempo.

Os exemplos de aplicações apresentados no Capítulo 4 podem ser submetidos a uma maior elaboração.

Na versão de *Make Distribuído* apresentada (Seção 4.1) se supõe que todas as regras podem ser executadas em paralelo. Uma versão definitiva precisaria introduzir modificações na sintaxe do arquivo *makefile*, com estruturas de controle do paralelismo. Se poderia então ter pontos em que se espera o término de certos processos paralelos antes de ser permitido prosseguir.

O algoritmo empregado no exemplo do cálculo de números primos da Seção 4.2 é bem ineficiente. Outra possibilidade de solução a este problema seria utilizar o Teste de Primalidade de Rabin. Este método é probabilístico, seu resultado pode rejeitar o número como primo ou simplesmente concluir que existe uma probabilidade de sê-lo. A utilização de multiples instâncias deste algoritmo, com a execução em paralelo destas, permitiria concluir, sem demora adicional e com uma probabilidade bem maior, que o número pode ser primo (no caso de todas as instâncias concordarem).

Note-se que existe um considerável conjunto de aplicações importantes que apresenta as características necessárias para se beneficiarem do SGPC. Uma linha de trabalho em perspectiva é ilustrada a seguir.

Na área de teste de *software* a Análise de Mutantes – um dos critérios da técnica de teste baseada em erros – demanda, para sua aplicação efetiva, uma alta capacidade de processamento para a execução de grande número de mutantes gerados. Uma direção de pesquisa e de trabalho conjunto é integrar o SGPC ao ambiente **PROTEUM** em desenvolvimento no ICMSC-USP [Del92].

Uma aplicação típica poderia ser nos algoritmos de busca em cadeias (atualmente objeto de intensa pesquisa devido ao interesse no projeto "Genoma Humano"). A busca em uma cadeia poderia ser feita em paralelo em suas diversas subcadeias.

Bibliografia

[Bar83] Barron,I.; Cavill,P. & May,D. Transputer does 5 or more MIPS even when not used in parallel. Electronics, Nov 17, 1983, pp 109-115

[Bar86] Barhen,J. & Palmer,J.F. The Hypercube in Robotics and Machine Intelligence. Computers in Mechanical Engineering, March 86

[Beg91a] Beguelim,A.; Dongarra,J.; Geist,A.; Marchek,R. & Sunderam,V. Heterogeneous Network Computing Environment. version 1.3 Technical Report, Oak Ridge National Laboratory, Engineering Physics and Mathematics Division, Mathematical Sciencies Section, 1991.

[Beg91b] Beguelim,A.; Dongarra,J.; Geist,A.; Marchek,R. & Sunderam,V. A User's Guide to PVM Parallel Virtual Machine, version 2.3 Technical Report, Oak Ridge National Laboratory, Engineering Physics and Mathematics Division, Mathematical Sciencies Section, July 1991.

[Cas91] Castro,L.S. SISTRAC: Sistema de Suporte a Trabalho Cooperativo. Tese de Mestrado, 1991. Departamento de Ciência da Computação. UNICAMP.

[Che84] Cheriton,D.R. The V Kernel: A software base for distributed system. IEEE Software, April 1984.

[Che88] Cheriton,D.R. The V distributed system. Communications of the ACM, 31 (3), March 1988.

[Col86] Colley,S. & Palmer,J. A Microprocessor-based Hypercube Supercomputer. IEEE Micro, Oct 86, pp 6-16

[Del92] Delamaro,M.E. PROTEUM: Um Ambiente de Teste Baseado na Análise de Mutantes. Exame de Qualificação de Mestrado, 1992. ICMSC/USP.

[Dou87] Douglis,F. & Ousterhout,J. Process Migration in the Sprite Operating System Proceedings of 7th International Conference on Distributed Computing Systems, Sept 1987. Reprintred by the Computer Society Press of the IEEE, Aug 87, pp 18-25

[Dru92] Drumond,R. & Di Cianni, C. OMNI. Sistema de Suporte a Aplicações Distribuídas. VI Simposio Brasileiro de Engenharia de Software, Gramado, RS, Nov 1992. Projeto A_HAND. Departamento de Ciência da Computação. UNICAMP.

[Fat83] Fathi,E.T. & Krieger,M. Multiple Microprocessor Systems: What, Why and When. IEEE Computer, Vol 16, No. 3, March 83, pp 23-32

[Fen81] Feng,T. A Survey of Interconnection Networks. IEEE Computer, Dec 81, pp 12-27

[Fly66] Flynn,M.J. Very High Speed Computing Systems. Proc. IEEE, Vol 54, No. 12, Dec 66, pp 1901-1909

[Hes87] Hess,J.W. Alternative computer architectures reduce bottlenecks. EDN, Nov 87, pp 271-278

[Jon86] Mach and Matchmaker: Kernel support for object-oriented distributed systems. Association for Computing Machinary, 1986.

[Ker78] Kernighan,B.W. & Ritchie,D.M. The C Programming Language. Prentice-Hall, 1978.

[Kuc86] Kuck,D.J.; Davidson,E.S.; Lawrie,D.H. & Sameh,A.H. Parallel Supercomputing Today and the Cedar Approach. Science, Vol 231, Feb 86, pp 967-974

[Kun85] Kung.H.T. Systolic Arrays. Yearbook of Science and Technology. McGraw-Hill 1985.

[Les85] Lesser,L. Parallel Processing on Personal Computers. Trabuco Circle, Mission Viejo, CA.

[Mad88] Madron,T.W. Local Area Networks: the second generation. John Wiley & Sons, Inc., 1988

[Mag89] Magee,J.; Kramer,J. & Sloman,M. Constructing Distributed Systems in Conic. IEEE Transactions on Software Engineering, Vol 15, No 6, June 1989.

[Mil87] Miles,D.; Carlile,B.R. & Groshong,J. Parallel Programming on FPS T Series. Inter. Supercomputer Conference, Jan 87.

[Mok85] Mokhoff,N. Local and Global Networks. Special Report. Computer Design, Vol 24, No. 11, Sept 85, pp 49

[Ous88] Ousterhout,J.K.; Cherenson,A.R.; Douglis,F.; Nelson,M.N. & Welch,B.B. The Sprite network operating system. IEEE Computer, Feb 1988.

[Pak83] Paker,Y. Multi-microprocessor Systems. Academic Press, Inc., 1983

[Pal86] Palmer,J.F. A VLSI Parallel Supercomputer. HYPERCUBE Multiprocessors, 1986

[Run88] Runyon,S. From Systems to standards, the pace quickens in networking. Electronics, Vol 61, No. 8, April 88, pp 67-100

[Sei85] SeitzÇ.L. The Cosmic Cube. Comm. of the ACM, Vol 28, No. 1, Jan 85, pp 22-33

[Sie79] Siegel,H.J. Interconnection Networks for SIMD Machines. IEEE Computer, June 79, pp 57-65

[Sun89] Sun Microsystems Inc. C Programmer's Guide. 1989.

[Sun90a] Sun MicroSystems Inc. Network Programming Guide. Sun Release 4.1 Systems Services Overview. 1990.

[Sun90b] Sun MicroSystems Inc. Reference Manual. 1990.

[Tan87] Tanembaun, Andrew S. Operating Systems: Design and Implementation. Prentice-Hall International Editions, 1987

[Tay82] Taylor,R. & Wilson,P. Process-oriented language meets demands of distributed processing. Electronics, Nov 82, pp 89-95

[The86] Theimer,M. Preemptable Remote Execution Facilities for Loosely-Coupled Distributed Systems. PhD thesis, Stanford University, 1986

[Tho78] Thompson, K. Unix Time-Sharing System: UNIX Implementation. The Bell System Technical Journal, Vol 57, No. 6, Jul-Aug 78

[Wri88] Wright,M. Networking Software. Special Report. EDN, March 88, pp 102-108

[Yac] Yachuk, G. Código fonte do *gymake*. Informix Software Inc.

[Yew86] Yew,P.C. Architecture of the Cedar Parallel Supercomputer. Center for Supercomputing Research and Development, Aug 86.

Apêndice A

Serviços Básicos da Plataforma Computacional

RPC (Remote Procedure Call): Biblioteca de procedimentos que podem ser invocados nos programas. Estes serviços permitem que um processo *(caller)* tenha um processo em outro sistema *(server)* que execute para ele uma chamada de procedimento como se fosse feita pelo *caller*;

XDR (eXternal Data Reference): Especificação para a representação *portável* e padronizada dos dados;

portmapper : Meio utilizado por todos os serviços baseados em RPC para, de maneira padronizada, registrar sua existência em uma máquina e a sua correspondência com os canais de comunicação. Padroniza ainda o acesso do Cliente, ajudando-o a localizar o número de porta dos programas RPC em que está interessado;

NIS (Network Information Service): Serviço de informação da rede com os dados replicados e distribuídos da sua configuração. Permite o acesso a esta informação de maneira independente das localizações relativas dos clientes e dos servidores;

NFS (Network File System): Servidor que permite montar diretórios através da rede e tratá-los como se fossem locais; permite compartilhar arquivos e diretórios, provendo acesso remoto transparente aos sistemas de arquivos, mesmo sobre sistemas heterogêneos.

Apêndice B

Módulos e Arquivos do SGPC

SGPC

pcoop.x :
Especificação do Protocolo de Serviço RPC do **ServProc**.
Contém a definição da interface do servidor escrita na linguagem RPC.
Entrada para o gerador de *stubs* de programas RPC: *rpcgen*.
Uma listagem deste protocolo é dada no Apêndice C;

pcoop.h :
Código fonte C gerado por *rpcgen*.
Header com as definições de dados e funções do **ServProc**;

pcoop_svc.c :
Código fonte C gerado por *rpcgen*.
Contém o *stub* do servidor responsável por desempacotar os parâmetros, fazer a chamada local do procedimento desejado, e empacotar e enviar os resultados de volta para o *stub* do cliente;

pcoop_clnt.c :
Código fonte C gerado por *rpcgen*.
Contém o *stub* do Cliente do **ServProc** responsável pelo *empacotamento* dos parâmetros, pelo envio da requisição, e pelo *desempacotamento* dos resultados devolvidos pelo servidor ;

pcoop_xdr.c :
Código fonte C gerado por *rpcgen*.
Contém rotinas XDR responsáveis pelo *empacotamento* e *desempacotamento* de parâmetros e resultados em um formato independente da arquitetura dos computadores envolvidos;

pcoop_modif1.c :
Código fonte C, com as definições de dados globais acrescentadas aos *stubs* do servidor.
Incluído com *#include* em *pcoop_svc.c*;

pcoop_modif2.c :
Código fonte C da Inicialização do **ServProc**.
Incluído com *#include* em *pcoop_svc.c*;

pcoop_proc.c :
Código fonte C da implementação dos procedimentos remotos que constituem os serviços do **ServProc**;

pcoop :
Código executável do **ServProc**.

SGPC Estendido

interfpc.c :
Código fonte C da implementação das operações da **InterfPC**;

interfpc.h :
Código fonte C.
Header com as definições de dados e funções da *InterfPC*. Para ser usado com *#include* nas aplicações usuárias do SGPC Estendido;

libsgpc.a :
Biblioteca com o código objeto da **InterfPC**.
Deve ser ligada com a aplicação usuária do SGPC Estendido.

ERROS

erros.msg :
Códigos e texto das mensagens de erro;

erros.h :
Código fonte C com o *header* para incluir nas aplicações usuárias.

Apêndice C

Protocolo de Serviço RPC do ServProc

```
/*
 * pcoop.x : file de E p/ rpcgen(1)(444),
 *     Protocolo de Servico RPC p/ o Servidor de Processamento
 *          Cooperativo (ServProc)
 *   Implementacao dos procedimentos remotos en pcoop_proc.c
 */

/* ESTRUTURAS DE DADOS PARA INFORMACOES DOS HOSTS */

const LONGSTRING = 80;
const MAX_HOSTS = 70;
const MAX_MODS_IMPORS = 20;
const MAX_MODS_EXPORS = MAX_MODS_IMPORS;
const MAX_MODS = 40; /* MAX_MODS_IMPORS + MAX_MODS_EXPORS */
const MAX_ENV = 2500;

typedef char cadeia[LONGSTRING];
typedef char cadeia_grande[MAX_ENV];

  struct stamp
     { cadeia nome_host;
       long   sec;
       long   usec;
     };
```

```
struct tempo
   { long   sec;
     long   usec;
   };

struct ids_host
  {
     stamp  ident_serv;
     int    cod_erro;
     int    quantidade;
     cadeia lista_nomes[MAX_HOSTS];
  };

struct  inf_host
   {
     stamp  ident_serv;
     int    cod_erro;
     cadeia nome;
     cadeia arch;
     int    tem_server;
     stamp  ident_serv_host;
     int    quant_mods_impors;
     int    porc_cpu_idle;
     int    quant_usuarios;
     int    quant_processos;
   };

struct mod
   {
     cadeia nome;
     cadeia path;
     cadeia argv;
     cadeia envp;
     int    re_avalia;
     int    re_executavel;
   };
```

```
struct mod_host
   {
     cadeia host;
     cadeia nome;
     cadeia path;
     cadeia argv;
     cadeia envp;
     int    re_executavel;
   };

struct id_mod
   {
     stamp  ident_serv;
     int    cod_erro;
     int    ident_modulo;
   };

struct ids_mods
   {
     stamp  ident_serv;
     int    cod_erro;
     int    quantidade;
     int    lista_ids[MAX_MODS];
   };

enum estado_exec {nao_migrado, migrado, em_progresso,
                  concluido, erro};

struct inf_mod
   {
     stamp  ident_serv;
     int    cod_erro;
     cadeia nome;
     cadeia path;
     cadeia argv;
     cadeia envp;
     int    re_avalia;
     int    re_executavel;

     cadeia host_origem;
     int    ident_modulo;
     tempo  horario_migracao;
     tempo  horario_reconh_termino;
```

```
        cadeia host_destino;
        int    migrado;
        tempo  horario_inicio;
        tempo  horario_termino;

        int         id_processo;
        estado_exec estado;
        int         cod_term;
        tempo       ru_utime;
        tempo       ru_stime;
      };

  struct id_mod_migrado
      {
        stamp  ident_serv;
        int    cod_erro;
        int    id_processo;
      };

  struct inf_erro
      {
        stamp  ident_serv;
        int    cod_erro;
        cadeia msg_erro;
      };

program PROCCOOP {
   version PROCCOOPVERS {

     int  ENVIAIDSERVPROC(stamp) = 1;

     stamp PEDEIDSERVPROC(void) = 2;


     ids_host LISTAHOSTS(void) = 3;

     inf_host PEDEINFHOST(cadeia) = 4;


     id_mod EXECMOD(mod) = 5; /* ID do mod. E en expors[] */

     id_mod EXECMODHOST(mod_host) = 6;

     id_mod_migrado EXECREMMOD(inf_mod) = 7; /* ID proceso no destino */
```

```
    ids_mods LISTAMODS(void) = 8;

    inf_mod PEDEINFMOD(int) = 9;

    int ENVIAINFMOD(inf_mod) = 10;


    inf_erro INFERRO(int) = 11;

        } =1;  /*  PROCCOOPVERS */
} = 0x20000099;  /* PROCCOOP */
```

Apêndice D

Interface de Alto Nível do SGPC: InterfPC

```
/*
 * interfpc.h : Subrotinas de Higest Level, interface
 *     com o Servidor de Processamento Cooperativo
 *
 */

#define LONGSTRING 80
#define MAX_HOSTS 70
#define MAX_MODS_IMPORS 20
#define MAX_MODS_EXPORS MAX_MODS_IMPORS
#define MAX_MODS 40
#define MAX_ENV 2500

typedef char cadeia[LONGSTRING];
bool_t xdr_cadeia();


typedef char cadeia_grande[MAX_ENV];
bool_t xdr_cadeia_grande();


struct stamp {
cadeia nome_host;
long sec;
long usec;
};
```

```
typedef struct stamp stamp;
bool_t xdr_stamp();


struct tempo {
long sec;
long usec;
};
typedef struct tempo tempo;
bool_t xdr_tempo();


struct ids_host {
stamp ident_serv;
int cod_erro;
int quantidade;
cadeia lista_nomes[MAX_HOSTS];
};
typedef struct ids_host ids_host;
bool_t xdr_ids_host();


struct inf_host {
stamp ident_serv;
int cod_erro;
cadeia nome;
cadeia arch;
int tem_server;
stamp ident_serv_host;
int quant_mods_impors;
int porc_cpu_idle;
int quant_usuarios;
int quant_processos;
};
typedef struct inf_host inf_host;
bool_t xdr_inf_host();


struct mod {
cadeia nome;
cadeia path;
cadeia argv;
cadeia envp;
int re_avalia;
```

```
int re_executavel;
};
typedef struct mod mod;
bool_t xdr_mod();


struct mod_host {
cadeia host;
cadeia nome;
cadeia path;
cadeia argv;
cadeia envp;
int re_executavel;
};
typedef struct mod_host mod_host;
bool_t xdr_mod_host();


struct id_mod {
stamp ident_serv;
int cod_erro;
int ident_modulo;
};
typedef struct id_mod id_mod;
bool_t xdr_id_mod();


struct ids_mods {
stamp ident_serv;
int cod_erro;
int quantidade;
int lista_ids[MAX_MODS];
};
typedef struct ids_mods ids_mods;
bool_t xdr_ids_mods();


enum estado_exec {
nao_iniciado = 0,
migrado = 1,
em_progresso = 2,
concluido = 3,
erro = 4,
};
```

```
typedef enum estado_exec estado_exec;
bool_t xdr_estado_exec();


struct inf_mod {
stamp ident_serv;
int cod_erro;
cadeia nome;
cadeia path;
cadeia argv;
cadeia envp;
int re_avalia;
int re_executavel;
cadeia host_origem;
tempo horario_migracao;
tempo horario_reconh_termino;
cadeia host_destino;
tempo horario_inicio;
tempo horario_termino;
int ident_modulo;
int migrado;
int id_processo;
estado_exec estado;
int cod_term;
tempo ru_utime;
tempo ru_stime;
};
typedef struct inf_mod inf_mod;
bool_t xdr_inf_mod();


struct id_mod_migrado {
stamp ident_serv;
int cod_erro;
int ident_mod_migrado;
};
typedef struct id_mod_migrado id_mod_migrado;
bool_t xdr_id_mod_migrado();


struct inf_erro {
stamp ident_serv;
int cod_erro;
cadeia msg_erro;
```

```
};
typedef struct inf_erro inf_erro;
bool_t xdr_inf_erro();


extern int  clnt_proc_coop();
extern void fim_proc_coop();
extern ids_host listahosts();
extern inf_host pedeinfhost(/* cadeia nome_host */);
extern id_mod execmod(/* mod dados_modulo */);
extern id_mod execmodhost(/* mod_host dados_modulo */);
extern ids_mods listamods();
extern inf_mod pedeinfmod(/* int id_modulo */);
extern inferro(/* int codigo */);
```

Apêndice E

Códigos e Mensagens de Erros do SGPC

/* Códigos e mensagens de erro do SGPC */

```
#define NOERR    0

#define ENOIDENT 1 /* Nao e possivel conferir a identidade de
                      um ServProc remoto, pois nao se consigue
                      obter sua identidade atual */

#define EIDENT   2  /* Se verificou que existe um novo ServProc
                       em uma estacao remota, quando se tinha
                       contactado anteriormente com outro nela.
                        Sua identidade atual nao coincide com
                       a anterior */

#define EREXECREM 3 /* nao se consegue relancar a execucao
                       de um modulo do usuario na mesma
                       estacao remota em que ele estava executando,
                       depois de ter acontecido uma queda do
                       ServProc que originalmente residia nela */

#define ECLIENT   4 /* Nao se consegue criar um novo cliente de
                       um ServProc remoto do qual se garantia
                       a existencia */

#define ESERVIC   5 /* Nao se consegue obter um servico de um
                       ServProc remoto do qual se e cliente */
```

```
#define EHOST      6 /* o nome de host dado pelo usuario nao
                        e valido */

#define EMAXMODS   7 /* tentativa de exceder a quantidade maxima de
                        modulos a executar */

#define EFORK      8 /* nao se conseguiu criar um filho para
                        a execucao do modulo do cliente */

#define EESPECMOD  9 /* erro na especificacao do modulo
                        dada pelo cliente */

#define EEXECMOD  10 /* erro na tentativa de execucao
                        do modulo */

#define EIDMOD    11 /* o ID de mod dado pelo cliente nao
                        se corresponde com nenhum modulo
                        existente */

#define ENAOEXISTE 12 /* nao existe esse codigo de erro */

#define EFILEERRO 13 /* nao esta presente o arquivo de erros */
```

Apêndice F

Histórico das Transações de ServProcs

F.1 ServProc Origem das Migrações (estação s2i)

```
Nome Dominio: dcc.unicamp.br

      Estacao          Ident_Serv        Arq.    Idle  Quant Users  Procss

 0:            s2a <        -1><     -1> <desc.> < -1>   <   -1>    < -1>
 1:            s2c <        -1><     -1> <desc.> < -1>   <   -1>    < -1>
 2:            s2i < 725577871><  80923> < sun4> < 99>   <    1>    < 19>
 3:            s3a <        -1><     -1> <desc.> < -1>   <   -1>    < -1>
 4:            s2e < 725577853>< 817781> < sun4> < 98>   <    1>    < 20>
 5:            s3b <        -1><     -1> <desc.> < -1>   <   -1>    < -1>
 6:            s2d < 725577829>< 949047> < sun4> < 99>   <    1>    < 20>
 7:            s2h <        -1><     -1> <desc.> < -1>   <   -1>    < -1>
 8:            s2g <        -1><     -1> <desc.> < -1>   <   -1>    < -1>
 9:            s2f <        -1><     -1> <desc.> < -1>   <   -1>    < -1>
10:          tiete <        -1><     -1> <desc.> < -1>   <   -1>    < -1>
11:            s2b <        -1><     -1> <desc.> < -1>   <   -1>    < -1>
12: dcc.unicamp.br <        -1><     -1> <desc.> < -1>   <   -1>    < -1>
13:        marumbi < 725577810>< 943867> < sun4> < 49>   <    1>    < 19>

Quantidade de Estacoes: 14    Estacao Atual[2]:s2i

******** Ate aqui instalacao **********
```

```
Hosts segundo sua porc_cpu_idle: [indice]<porc_cpu_idle><quant_mods_impors>

[ 2]:<99><0>  [ 6]:<99><0>  [ 4]:<98><0>  [13]:<49><0>  [ 1]:<-1><0>
[ 5]:<-1><0>  [ 0]:<-1><0>  [ 7]:<-1><0>  [ 8]:<-1><0>  [ 9]:<-1><0>
[10]:<-1><0>  [11]:<-1><0>  [12]:<-1><0>  [ 3]:<-1><0>

Servico EXECMOD: Path:</bin/cc>  Argv:< -c make.c -o make.o>

Hosts segundo sua porc_cpu_idle: [indice]<porc_cpu_idle><quant_mods_impors>

[ 2]:<99><1>  [ 6]:<99><0>  [ 4]:<98><0>  [13]:<49><0>  [ 1]:<-1><0>
[ 5]:<-1><0>  [ 0]:<-1><0>  [ 7]:<-1><0>  [ 8]:<-1><0>  [ 9]:<-1><0>
[10]:<-1><0>  [11]:<-1><0>  [12]:<-1><0>  [ 3]:<-1><0>

E[ 0]
 Nome:<cc>   Path:</bin>   Argv:< -c make.c -o make.o>
 Envp:<...>
 Re_executavel:<0>
 Host_Origem:<s2i>        Ident_modulo:<0>
 Horario_Migracao: < 725577925> secs.  < 285363> usecs.
 Horario_Reconh_Termino: <        -1> secs.  <     -1> usecs.
 Host_Destino:<s2i>
 Migrado:<0>              Id_processo:<1360>
 Horario_Inicio: < 725577925> secs.  < 285882> usecs.
 Horario_Termino: <        -1> secs.  <     -1> usecs.
 Estado:<2>               Cod_term:<-1>
 (user mode) Secs:<        -1>  uSecs:<     -1>
 (system mode) Secs:<        -1>  uSecs:<     -1>

I[ 0]
 Nome:<cc>   Path:</bin>   Argv:< -c make.c -o make.o>
 Envp:<...>
 Re_executavel:<0>
 Host_Origem:<s2i>        Ident_modulo:<0>
 Horario_Migracao: <        -1> secs.  <     -1> usecs.
 Horario_Reconh_Termino: <        -1> secs.  <     -1> usecs.
 Host_Destino:<s2i>
 Migrado:<0>              Id_processo:<1360>
 Horario_Inicio: < 725577925> secs.  < 285882> usecs.
 Horario_Termino: <        -1> secs.  <     -1> usecs.
 Estado:<2>               Cod_term:<-1>
 (user mode) Secs:<        -1>  uSecs:<     -1>
 (system mode) Secs:<        -1>  uSecs:<     -1>
```

```
                ⋮

Servico EXECMOD: Path:</bin/cc>  Argv:< -c parse.c -o parse.o>

Hosts segundo sua porc_cpu_idle: [indice]<porc_cpu_idle><quant_mods_impors>

[ 2]:<99><1>  [ 6]:<99><1>  [ 4]:<98><0>  [13]:<49><0>  [ 1]:<-1><0>
[ 5]:<-1><0>  [ 0]:<-1><0>  [ 7]:<-1><0>  [ 8]:<-1><0>  [ 9]:<-1><0>
[10]:<-1><0>  [11]:<-1><0>  [12]:<-1><0>  [ 3]:<-1><0>

Chamada a EXECREMMOD de:<s2d> com </bin/cc>  < -c parse.c -o parse.o>


E[ 1]
 Nome:<cc>   Path:</bin>   Argv:< -c parse.c -o parse.o>
 Envp:<...>
 Re_executavel:<0>
 Host_Origem:<s2i>          Ident_modulo:<1>
 Horario_Migracao: < 725577925> secs.  < 614424> usecs.
 Horario_Reconh_Termino: <        -1> secs.  <     -1> usecs.
 Host_Destino:<s2d>
 Migrado:<1>                Id_processo:<1304>
 Horario_Inicio: <          -1> secs.  <     -1> usecs.
 Horario_Termino: <          -1> secs.  <     -1> usecs.
 Estado:<2>                 Cod_term:<-1>
 (user mode) Secs:<          -1>  uSecs:<     -1>
 (system mode) Secs:<          -1>  uSecs:<     -1>

                ⋮

Servico EXECMOD: Path:</bin/cc>  Argv:< -c tstring.c -o tstring.o>

Hosts segundo sua porc_cpu_idle: [indice]<porc_cpu_idle><quant_mods_impors>

[ 2]:<99><1>  [ 6]:<99><1>  [ 4]:<98><1>  [13]:<49><0>  [ 1]:<-1><0>
[ 5]:<-1><0>  [ 0]:<-1><0>  [ 7]:<-1><0>  [ 8]:<-1><0>  [ 9]:<-1><0>
[10]:<-1><0>  [11]:<-1><0>  [12]:<-1><0>  [ 3]:<-1><0>

Chamada a EXECREMMOD de:<s2e> com </bin/cc>  < -c tstring.c -o tstring.o>
```

```
E[ 2]
 Nome:<cc>   Path:</bin>   Argv:< -c tstring.c -o tstring.o>
 Envp:<...>
 Re_executavel:<0>
 Host_Origem:<s2i>        Ident_modulo:<2>
 Horario_Migracao: < 725577925> secs.  < 948633> usecs.
 Horario_Reconh_Termino: <        -1> secs.  <     -1> usecs.
 Host_Destino:<s2e>
 Migrado:<1>              Id_processo:<2328>
 Horario_Inicio: <         -1> secs.  <     -1> usecs.
 Horario_Termino: <        -1> secs.  <     -1> usecs.
 Estado:<2>               Cod_term:<-1>
 (user mode) Secs:<        -1>  uSecs:<     -1>
 (system mode) Secs:<        -1>  uSecs:<     -1>

                :

Servico ENVIAINFMOD

E[ 2]
 Nome:<cc>   Path:</bin>   Argv:< -c tstring.c -o tstring.o>
 Envp:<...>
 Re_executavel:<0>
 Host_Origem:<s2i>        Ident_modulo:<2>
 Horario_Migracao: < 725577925> secs.  < 948633> usecs.
 Horario_Reconh_Termino: < 725577928> secs.  < 690837> usecs.
 Host_Destino:<s2e>
 Migrado:<1>              Id_processo:<2328>
 Horario_Inicio: < 725577928> secs.  < 485494> usecs.
 Horario_Termino: < 725577931> secs.  < 113963> usecs.
 Estado:<3>               Cod_term:<0>
 (user mode) Secs:<          0>  uSecs:< 700000>
 (system mode) Secs:<          0>  uSecs:< 500000>

Hosts segundo sua porc_cpu_idle: [indice]<porc_cpu_idle><quant_mods_impors>

[ 2]:<99><1>  [ 6]:<99><1>  [ 4]:<98><0>  [13]:<49><0>  [ 1]:<-1><0>
[ 5]:<-1><0>  [ 0]:<-1><0>  [ 7]:<-1><0>  [ 8]:<-1><0>  [ 9]:<-1><0>
[10]:<-1><0>  [11]:<-1><0>  [12]:<-1><0>  [ 3]:<-1><0>
```

```
Servico ENVIAINFMOD

E[ 1]
 Nome:<cc>   Path:</bin>   Argv:< -c parse.c -o parse.o>
 Envp:<...>
 Re_executavel:<0>
 Host_Origem:<s2i>        Ident_modulo:<1>
 Horario_Migracao: < 725577925> secs.  < 614424> usecs.
 Horario_Reconh_Termino: < 725577929> secs.  < 485826> usecs.
 Host_Destino:<s2d>
 Migrado:<1>              Id_processo:<1304>
 Horario_Inicio: < 725577924> secs.  < 474989> usecs.
 Horario_Termino: < 725577928> secs.  < 238564> usecs.
 Estado:<3>               Cod_term:<0>
 (user mode) Secs:<         1>  uSecs:< 900000>
 (system mode) Secs:<         0>  uSecs:< 530000>


Hosts segundo sua porc_cpu_idle: [indice]<porc_cpu_idle><quant_mods_impors>

[ 2]:<99><1>  [ 6]:<99><0>  [ 4]:<98><0>  [13]:<49><0>  [ 1]:<-1><0>
[ 5]:<-1><0>  [ 0]:<-1><0>  [ 7]:<-1><0>  [ 8]:<-1><0>  [ 9]:<-1><0>
[10]:<-1><0>  [11]:<-1><0>  [12]:<-1><0>  [ 3]:<-1><0>

SIGCHLD do processo <1360>. impors[0]   Status:<0>


I[ 0]
 Nome:<cc>   Path:</bin>   Argv:< -c make.c -o make.o>
 Envp:<...>
 Re_executavel:<0>
 Host_Origem:<s2i>        Ident_modulo:<0>
 Horario_Migracao: <        -1> secs.  <     -1> usecs.
 Horario_Reconh_Termino: <        -1> secs.  <     -1> usecs.
 Host_Destino:<s2i>
 Migrado:<0>              Id_processo:<1360>
 Horario_Inicio: < 725577925> secs.  < 285882> usecs.
 Horario_Termino: < 725577930> secs.  < 759467> usecs.
 Estado:<3>               Cod_term:<0>
 (user mode) Secs:<         3>  uSecs:< 150000>
 (system mode) Secs:<         0>  uSecs:< 700000>
```

```
Provem do proprio host


E[ 0]
 Nome:<cc>   Path:</bin>   Argv:< -c make.c -o make.o>
 Envp:<...>
 Re_executavel:<0>
 Host_Origem:<s2i>          Ident_modulo:<0>
 Horario_Migracao: < 725577925> secs.  < 285363> usecs.
 Horario_Reconh_Termino: < 725577930> secs.  < 759467> usecs.
 Host_Destino:<s2i>
 Migrado:<0>                Id_processo:<1360>
 Horario_Inicio: < 725577925> secs.  < 285882> usecs.
 Horario_Termino: < 725577930> secs.  < 759467> usecs.
 Estado:<3>                 Cod_term:<0>
 (user mode) Secs:<          3>  uSecs:< 150000>
 (system mode) Secs:<          0>  uSecs:< 700000>


Hosts segundo sua porc_cpu_idle: [indice]<porc_cpu_idle><quant_mods_impors>

[ 2]:<99><0>  [ 6]:<99><0>  [ 4]:<98><0>  [13]:<49><0>  [ 1]:<-1><0>
[ 5]:<-1><0>  [ 0]:<-1><0>  [ 7]:<-1><0>  [ 8]:<-1><0>  [ 9]:<-1><0>
[10]:<-1><0>  [11]:<-1><0>  [12]:<-1><0>  [ 3]:<-1><0>
```

F.2 ServProc Destino de Migrações (estação s2d)

```
Servico EXECREMMOD: Path: </bin/cc>  Args: < -c parse.c -o parse.o>

Hosts segundo sua porc_cpu_idle: [indice]<porc_cpu_idle><quant_mods_impors>

[ 6]:<99><1>  [13]:<49><0>  [ 2]:<-1><0>  [ 3]:<-1><0>  [ 4]:<-1><0>
[ 5]:<-1><0>  [ 0]:<-1><0>  [ 7]:<-1><0>  [ 8]:<-1><0>  [ 9]:<-1><0>
[10]:<-1><0>  [11]:<-1><0>  [12]:<-1><0>  [ 1]:<-1><0>


I[ 0]
 Nome:<cc>   Path:</bin>   Argv:< -c parse.c -o parse.o>
 Envp:<...>
 Re_executavel:<0>
 Host_Origem:<s2i>        Ident_modulo:<1>
 Horario_Migracao: < 725577925> secs.  < 614424> usecs.
 Horario_Reconh_Termino: <        -1> secs.  <     -1> usecs.
 Host_Destino:<s2d>
 Migrado:<1>              Id_processo:<1304>
 Horario_Inicio: <        -1> secs.  <     -1> usecs.
 Horario_Termino: <        -1> secs.  <     -1> usecs.
 Estado:<2>               Cod_term:<-1>
 (user mode) Secs:<        -1>  uSecs:<     -1>
 (system mode) Secs:<        -1>  uSecs:<     -1>

SIGCHLD do processo <1304>. impors[0]   Status:<0>

I[ 0]
 Nome:<cc>   Path:</bin>   Argv:< -c parse.c -o parse.o>
 Envp:<...>
 Re_executavel:<0>
 Host_Origem:<s2i>        Ident_modulo:<1>
 Horario_Migracao: < 725577925> secs.  < 614424> usecs.
 Horario_Reconh_Termino: <        -1> secs.  <     -1> usecs.
 Host_Destino:<s2d>
 Migrado:<1>              Id_processo:<1304>
 Horario_Inicio: < 725577924> secs.  < 474989> usecs.
 Horario_Termino: < 725577928> secs.  < 238564> usecs.
 Estado:<3>               Cod_term:<0>
 (user mode) Secs:<         1>  uSecs:< 900000>
 (system mode) Secs:<         0>  uSecs:< 530000>
```

```
Se informou ao Origem

Hosts segundo sua porc_cpu_idle: [indice]<porc_cpu_idle><quant_mods_impors>

[ 6]:<99><0>  [13]:<49><0>  [ 2]:<-1><0>  [ 3]:<-1><0>  [ 4]:<-1><0>
[ 5]:<-1><0>  [ 0]:<-1><0>  [ 7]:<-1><0>  [ 8]:<-1><0>  [ 9]:<-1><0>
[10]:<-1><0>  [11]:<-1><0>  [12]:<-1><0>  [ 1]:<-1><0>
```