Modelo de Referência para Especificação Formal de Sistemas Operacionais Distribuídos

Tânia Saraiva de Melo Pinheiro*

Paulo Roberto Freire Cunha**

Departamento de Informática

Universidade Federal de Pernambuco

Caixa Postal 7851, 50739 Recife - PE - Brasil

Resumo

A especificação formal de um modelo de referêcia que se propõe a ser uma ferramenta para projeto e especificação de sistemas operacionais distribuídos, denominado RM-DOS, é introduzida neste trabalho. Com o RM-DOS, apresenta-se uma modularização para sistemas operacionais na qual cada módulo possui funções de gerenciamento de recursos similares e uma interface bem definida para se comunicar com os demais. A linguagem para especificação utilizada para descrever formalmente essa modularização é LOTOS.

Abstract

In this paper we introduce the formal specification of a reference model, called RM-DOS, proposed as a tool for design and specification of distributed operating systems. RM-DOS proposes a modularization for operating systems with each module grouping similar resource management functions and also a well definded interface for communicating with the others. The formal specification language used to describe the modularization is LOTOS.

*M.Sc. em Computação (UFPE 1990). Sistemas Distribuídos, Sistemas Operacionais, Programação Concorrente.

**Ph.D. em Computação (Waterloo 1981). Prof. Adjunto (UFPE). Redes de Computadores, Sistemas Distribuídos, Metodologias de Programação.

1 Introdução

O projeto de grandes sistemas constituídos pela união de vários componentes de pequeno porte dá origem às arquiteturas que permitem o processamento distribuído, e que motivam o desenvolvimento de aplicações utilizando dados espalhados por várias máquinas. Entretanto, a distribuição dos dados pode comprometer a eficiência da aplicação caso os programas que a implementam não sejam também distribuídos. Deve-se, ainda, tomar cuidado para que a execução de partes de um programa em máquinas diferentes ocorra coordenadamente, para que não haja prejuízo da integridade do sistema como um todo.

Os sistemas operacionais distribuídos foram desenvolvidos para gerenciar a utilização de vários computadores interligados, dando ao usuário a idéia de estar acessando apenas uma máquina mais poderosa [TR85]. A enorme gama de conceitos que envolvem estes sistemas faz com que o projeto ou, simplesmente, o estudo de algum deles seja uma tarefa árdua.

A constatação desta dificuldade motivou o desenvolvimento do Modelo de Referência para Projeto e Especificação Formal de Sistemas Operacionais Distribuídos, denominado RM-DOS (**R***eference* **M***odel for the Design and Specification of* **D***istributed* **O***perating* **S***ystems*).

2 Modelo de Referência

A necessidade de se dispor de um modelo de referência para projeto de sistemas operacionais distribuídos pode ser percebida fazendo-se uma analogia com redes de computadores. Devido à complexidade das Redes de Computadores, dentre outros motivos, utiliza-se o modelo de referência RM-OSI da ISO [DZ83] para comparar duas delas, o que só é feito após ambas terem sido apresentadas de acordo com este modelo. Além disso, é recomendado que o projeto de uma nova rede tenha como base o modelo RM-OSI, mesmo que não se pretenda segui-lo em sua totalidade.

Enquanto que o *software* de uma Rede corresponde, basicamente, a protocolos de comunicação, o *software* de um sistema distribuído engloba, além destes protocolos, todas as funções inerentes aos sistemas operacionais. O escalonamento de processos e o sistema de arquivos são exemplos de tais funções. Assim, a necessidade de se dispor de um modelo de referência para projetá-los e analisá-los é ainda maior que aquela verificada em Redes de Computadores.

O modelo de referência RM-DOS [Pin90] é uma tentativa de suprir esta necessidade. Um sistema operacional distribuído abstrato é modularizado e especificado formalmente, usando a técnica de descrição formal LOTOS [ISO87,Bri86].

O RM-DOS foi desenvolvido a partir de uma análise de vários sistemas operacionais distribuídos já implementados. Sua primeira etapa consistiu da determinação de uma modularização desejável para estes sistemas, além da identificação das possíveis interações entre os módulos. Em seguida o modelo foi especificado formalmente utilizando a linguagem de especificação LOTOS. Neste trabalho apresenta-se, resumidamente, a especificação formal da modularização desenvolvida.

3 Modularização

A determinação dos serviços de um sistema operacional distribuído (SOD), para sua posterior modularização, teve como base a própria definição de sistemas operacionais [Tan87]: "um sistema operacional é um programa que possui as funções básicas de prover aos usuários uma máquina virtual e de gerenciar os recursos disponíveis".

A máquina virtual citada refere-se à interface do sistema com seus usuários[1], o que corresponde às chamadas do supervisor. Dentre as características desejáveis a uma interface externa, destacam-se a semelhança com aquelas dos sistemas operacionais centralizados e o suporte à programação concorrente [Pin90]. É função do gerenciamento de recursos implementar estas chamadas.

Os aspectos básicos da modularização do gerenciamento de recursos de um sistema operacional distribuído são apresentados na sequência.

Gerenciamento de Recursos

O gerenciamento de recursos foi modularizado conforme ilustrado a seguir. Neste esquema, o gerenciamento de entrada/saída é dividido de acordo com o tipo de periférico considerado, devido à grande importância dos dispositivos de comunicação e de memória secundária (que armazenam o sistema de arquivos). Este trabalho propõe que qualquer atividade (função) de um SOD pode ser enquadrada em pelo menos um destes módulos gerenciadores apresentados.

- *Gerenciamento de Recursos*
  - *Gerenciamento de Processadores*
  - *Gerenciamento de Memória*
  - *Gerenciamento de Entrada/Saída*
    - *Controle de Comunicacão*
    - *Controle de Arquivos*
    - *Controle de Entrada/Saída*

Uma vez determinados os módulos básicos, vejamos como projetá-los. Existe uma grande tendência em se desenvolver sistemas distribuídos em camadas, colocando-se a maioria de suas funções sob o controle de camadas superiores (mais abstratas) e deixando-se o núcleo o menor possível. O tratamento comum é seguir o MODELO DE SERVIDORES no qual a maioria dos serviços são implementados como processos servidores, que estão no mesmo nível de abstração dos processos usuários [Che88]. A preocupação agora, consiste em determinar camadas genéricas o bastante para que o RM-DOS seja abrangente como desejado.

[1] Por usuários entenda-se todos os elementos que utilizam o sistema, incluindo programas aplicativos (compiladores, editores de texto, interpretador de comandos, etc.) e os usuários propriamente ditos.

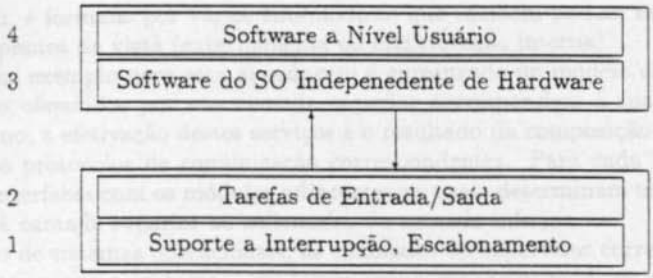

Figura 1: Camadas de um Sistema Operacional

Observando-se sistemas já desenvolvidos, comumente identifica-se a presença de quatro camadas, conforme apresentado na Figura 1. Apesar de nem todos os sistemas seguirem exatamente este modelo, sua abrangência é satisfatória [Pin90] e, portanto, foi o adotado pelo RM-DOS.

A camada mais alta contém processos tão relacionados com o sistema operacional que, muitas vezes, são considerados como seus membros. Exemplos típicos são as bibliotecas de rotinas, que "expandem" a interface externa, e o interpretador de comandos. Os processos dos usuários também encontram-se nesta camada.

A terceira camada contém o gerenciamento de recursos independente de máquina e é formada por processos servidores. As tarefas de entrada/saída, ou *device drivers*, correspondem ao baixo nível do controle de recursos — aquele dependente de máquina. Quanto à camada inferior, esta recebe sinais do *hardware* e ativa as tarefas responsáveis por tratá-los. Suporta também o escalonamento de processos e a comunicação local entre eles. As camadas três, dois e um da Figura 1 foram denominadas **Servidores**, **Tarefas** e **Núcleo**, respectivamente.

Uma vez determinado que os sistemas serão estruturados em camadas e que eles compõem-se de cinco módulos funcionais referentes ao gerenciamento de recursos[2], manteve-se o mesmo princípio para concluir que cada um destes módulos deve ser estruturado em camadas. A estruturação dos módulos de gerenciamento de recursos de acordo com as três camadas referentes ao sistema operacional (três camadas inferiores da Figura 1, bem como as possíveis interações entre os módulos é detalhadamente apresentada em [Pin90] e esquematizada na Figura 2.

4 Especificação Formal

Existem dois pontos de vista básicos através dos quais se pode visualizar a especificação de um *software*: como um examinador externo, observando apenas sua interface com seu ambiente; ou como projetista, estruturando-o em módulos que se mostrem necessários para que efetive a funcionalidade (aparência externa) esperada [BB87]. Cada Módulo,

[2]Gerenciamento de processadores, gerenciamento de memó ria, controle de comunicação controle de arquivos e controle de entrada/saída.

Gerentes / Camadas		Gerenciamento de Processadores	Gerenciamento de Memoria	Gerenciamento de Entrada/Saida		
				Controle de Comunicacao	Controle de Arquivos	Controle de Ent./Saida
Servidores		Alocacao de Processadores (AProc)	Gerenciamento de Espaco (GEsp)	Comunicacao Transparente (ComT)	Sistema de Arquivos (SArq)	Servicos de Ent./Saida (SES)
Executor	Tarefas	Dados	Tarefa Memoria	Tarefa Comunicacao	Tarefas Memoria Secundaria	Tarefas E/S
	Nucleo	Escalonador	Rotina Memoria	Comunicacao Local	Rotinas Memoria Secundaria	Rotinas E/S

Figura 2: Modularização de Sistemas Operacionais Distribuídos

por sua vez, é formado por vários sub-módulos que também podem ser observados sob estes dois pontos de vista (externamente ou estruturação interna).

Um bom exemplo para esta abordagem é encontrado no modelo de referência OSI. Os serviços oferecidos por sua camada superior correspondem à sua interface com o meio externo; a efetivação destes serviços é o resultado da composição de suas camadas através dos protocolos de comunicação correspondentes. Para cada uma delas, estão definidas interfaces com os módulos adjacentes as quais determinam os serviços que são prestados à camada superior ou solicitados da camada inferior.

No caso de sistemas operacionais, as chamadas do supervisor correspondem à interface do sistema com o ambiente. Processos servidores são os módulos que implementam as chamadas do sistema, utilizando os serviços do núcleo. Para cada um destes módulos de um SOD também se pode definir interfaces através das quais se presta e se solicita serviços.

O RM-DOS consiste de uma sucessão hierárquica de módulos de *software*, todos analisados tanto do ponto de vista de sua interface externa como internamente. A linguagem de especificação utilizada para o seu detalhamento formal é LOTOS.

4.1 LOTOS

LOTOS é uma técnica de descrição formal que foi inicialmente desenvolvida visando a especificação dos protocolos e serviços de redes de computadores; em particular daqueles definidos pelo RM-OSI da ISO. Com este trabalho, comprova-se que LOTOS também é adequada à especificação de sistemas operacionais distribuídos.

Um sistema é especificado em LOTOS definindo-se as relações temporais entre as interações que constituem seu comportamento externamente observável. A especificação destas sincronizações (relações temporais) utiliza uma versão simplificada da linguagem, denominada LOTOS Básico. A comunicação de valores é suportada pela extensão da linguagem básica, denominada LOTOS Completo, que permite a descrição de estruturas de dados e expressões aritméticas. Esta última é baseada na teoria formal de tipos abstratos de dados e a maioria dos conceitos utilizados são oriundos desta técnica.

Uma especificação LOTOS descreve um sistema por meio de uma hierarquia de definições de processos. Um processo é uma entidade capaz de realizar ações internas e não observáveis e de interagir com outros processos através de ações externamente observáveis. A unidade atômica de interação entre processos é denominada **evento**, que corresponde a uma comunicação síncrona que pode ocorrer entre processos capazes de efetuar este evento. Eventos são atômicos, no sentido de que eles ocorrem instantaneamente, sem consumir tempo; ocorrem em um ponto de interação denominado **porta**, podendo, ou não, envolver troca de valores.

A Tabela 1 relaciona as possíveis expressões de comportamento de LOTOS, utilizando todos os operadores. Os símbolos `C1` e `C2` representam expressões de comportamento; `p1`, `p2`, ..., `pn` representam eventos (porta ou ação) observáveis; e `i` um nome reservado de LOTOS para representar um evento interno não observável[3]. Qualquer

[3]Este evento foi utilizado pelo RM-DOS para especificar o comportamento de rotinas do núcleo que são ativadas por interrupção.

NOME DA EXPRESSÃO		SINTAXE
inação		*stop*
prefixo de ação	não observável	`i; C1`
	observável	`p1; C1`
escolha		`C1 [] C2`
composição paralela	caso geral	`C1 |[p1,p2,...,pn]| C2`
	sem sincronização	`C1 ||| C2`
	sincronização plena	`C1 || C2`
hiding		`hide p1,...,pn in C1`
instanciação de processos		`ABC[p1,...,pn]`
terminação com sucesso		`exit`
composição sequencial		`C1 >> C2`
desabilitação		`C1 [> C2`

Tabela 1: Sintaxe das Expressões de Comportamento de LOTOS

expressão de comportamento válida possui um dos formatos apresentados na Tabela 1.

Em LOTOS Básico, o nome de um evento coincide com o nome da porta. Entretanto, quando se utiliza tipos de dados, ou seja, quando se utiliza LOTOS Completo, o evento é formado pelo nome da porta seguido por uma lista de zero ou mais valores oferecidos no momento da interação. Por exemplo, `p1!true!7` representa a ação observável que oferece o valor booleano `true` e o número natural 7 através da porta p1. Assim, é possível se expressar um número infinito de ações observáveis, uma vez que os valores oferecidos nas portas podem variar infinitamente, como é o caso dos naturais, por exemplo.

4.2 Modelo de Referência para Especificação Formal

O MODELO DE ESPECIFICAÇÃO foi construído baseado na modularização de sistemas operacionais distribuídos esquematizada na Figura 2. Conforme apresentado nesta Figura, o sistema foi dividido em cinco unidades (gerentes) que prestam diferentes tipos de serviços de gerenciamento, de acordo com o tipo de dispositivo considerado. Cada um destes gerentes[4], por sua vez, foi estruturado em até três camadas hierárquicas. As duas inferiores compõem o executor e a camada superior corresponde a um processo servidor.

O nível mais abstrato da especificação em LOTOS do RM-DOS é apresentado a seguir[5] . Nesta etapa, apresenta-se não apenas a especificação de um SOD em seu nível mais abstrato, mas também aquela referente à sua interação com o ambiente (Figura 1). O objetivo é mostrar como unir a especificação do SOD com possíveis especificações

[4]Processadores, memória, comunicação, arquivos e entrada/saída.

[5]O símbolo > que aparece na primeira coluna faz parte da metalinguagem que apresenta as especificações em LOTOS e é usado para diferenciar texto de especificações.

de aplicações, considerando-se o fato de que LOTOS também pode ser utilizado para especificá-las [BB87,Fer89,Mou89].

```
> specification Estacao : noexit                    (* especificação do comportamento
>                                                        de uma estação *)
> behaviour
>    ProcessosUsuarios[cs] || SOD[cs]               (* interação através da porta cs *)
> where
>    process ProcessosUsuarios[cs] : noexit :=          (*processos usuarios*)
>       ProcessoUsuario[cs] ||| ProcessosUsuarios[cs]
>    where ...  endproc
>
>    process SOD[cs] : noexit  :=                   (* Sistema Operacional Distribuído*)
>       ( Servidores[cs] || Executor[cs] ) ||| SOD[cs]
>    where
>       process Servidores[cs] :=                       (*processos servidores*)
>              AlocacaoDeProcessadores[cs]
>          ||| GerenciamentoDeEspaco[cs]
>          ||| ComunicacaoTransparente[cs]
>          ||| SistemaDeArquivos[cs]
>          ||| ServicosDeEntradaSaida[cs]
>       where ...  endproc
>
>       process Executor[cs] :=                         (*executor do SOD*)
>       ...
>       endproc
>    endproc
> endspec
```

Informalmente, os processos desta especificação possuem o seguinte comportamento:

- **specification Estacao**
  Os processos usuários interagem com o sistema operacional por meio de chamadas do sistema. A invocação destas chamadas é feita através da porta cs (Chamadas do Sistema).

- **process ProcessosUsuarios**
  Uma característica desejável de um SOD, cofirmada pelo largo uso do modelo de servidores, é que a comunicação entre os processos seja feita por troca de mensagens, quer eles sejam processos usuários ou integrantes do próprio sistema operacional.

  Como as primitivas oferecidas pelo sistema para suportar a comunicação entre processos também são chamadas do sistema (ou são tratadas como se o fossem), a comunicação entre processos usuários também é realizada através da porta `cs`.

- **process SOD**
  O gerenciamento de recursos de um SOD foi dividido em cinco módulos e, cada um destes, estruturado em três camadas (Figura 2). A camada superior, contendo os servidores, é independente de máquina e as duas inferiores fazem parte do Executor do sistema. O `process SOD` especifica a interação dos servidores entre si e com o executor.

  Outra característica do processo `SOD` é a recursão especificada. A execução paralela sem sincronismo com ele próprio expressada por
  `... ||| SOD`
  especifica que este SOD suporta multiprogramação — uma nova chamada do sistema, por exemplo, pode ser recebida e tratada em qualquer instante de tempo. A importância desta característica fica mais clara quando se considera o exemplo de uma chamada do sistema referente à criação de processos: um novo processo é criado mesmo antes do término daquele que o gerou.

Um aspecto muito importante da especificação apresentada é o largo uso do paralelismo sem sincronismo, modelado pelo operador `|||`. Ao invés de se ter

```
> ProcessoUsuario[cs]  |[cs]|  ProcessosUsuarios[cs]
```

o que seria lógico dado que esses processos interagem através de `cs`, utilizou-se

```
> ProcessoUsuario[cs]  |||  ProcessosUsuarios[cs]
```

A justificativa para esta inovação está em uma das funções do núcleo (camada inferior do executor). Ele é o responsável pela comunicação local entre os processos, ou seja, é nele que, de fato, se processam as trocas de mensagens. A solicitação de uma chamada do sistema (por um processo usuário `P`, por exemplo) que conceitualmente é realizada enviando-se uma mensagem ao processo servidor reponsável por tratá-la, na prática é tratada através da sincronização da porta `cs` de `P` com a de mesmo nome do núcleo. Este, por sua vez, efetiva a troca de mensagens fazendo com que, dentre outras coisas, sua porta `cs` seja sincronizada com aquela do processo servidor destinatário.

Caso se estivesse especificando uma aplicação qualquer, ela provavelmente teria o formato

```
> AlocacaoDeProcessadores[cs] |[cs]| GerenciamentoDeEspaco[cs]
```

No caso de sistemas operacionais, entretanto, ela gera problemas pois, não apenas se especifica módulos que utilizam primitivas de comunicação mas também aqueles que as implementam. Não se pode confundir a utilização de um *send*, por exemplo, com sua "efetivação/implementação". Caso o operador com sincronismo fosse utilizado, não haveria como especificar que a comunicação local é implementada pelo executor.

Descendo-se um nível de abstração, a próxima etapa consiste na especificação dos componentes de SOD: os servidores e o executor.

4.2.1 Servidores

A expressão de comportamento de todos os servidores é muito semelhante. Como o RM-DOS organiza sistemas operacionais por funções é possível identificar algumas características comuns à especificação de cada um dos servidores básicos. São elas:

- Os servidores não prestam apenas os serviços determinados pelas chamadas do sistema: algumas outras funções não estão disponíveis aos usuários pois se referem a serviços prestados exclusivamente a outros processos servidores.
- Todos possuem um componente com funções locais e outro com funções globais. O primeiro recebe as chamadas do sistema e se encarrega de todo tratamento que pode ser feito localmente. O segundo, quando necessário, trata da interação com servidores semelhantes, alocados em outras máquinas.
- Para a comunicação entre os módulos que compõem os processos servidores, se define uma porta interna. Esta representa a possibilidade destes componentes poderem interagir de forma direta[6], em vez de terem que se comunicar por mensagens que, depois de enviada por um deles, passaria pelo núcleo antes de chegar ao componente destino.

Apesar das semelhanças, cada servidor possui suas características particulares. A seguir, a expressão comportamento do servidor `Comunicação Transparente` é apresentada. Este foi escolhido para ser descrito por ser o servidor mais rico em detalhes. A especificação dos demais servidores pode ser encontrada em [Pin90].

```
> process ComunicacaoTransparente[cs] :=
>
> hide int in                                  (*porta de interação entre os sub-módulos*)
> ( Seletor[cs,int]                                        (*gerenciamento local*)
>   |[int]|                                   (*interação entre os sub-módulos*)
>   ProtocolosDeAltoNivel[cs,int])                         (*gerenciamento global*)
> where
>
```

[6]Compartilhamento de variáveis, chamadas de procedimento, etc.

```
>    process Seletor[cs,int]          (*suporte aos serviços oferecidos pelo servidor*)
>    :=
>    ( (*caso 1: chamadas do sistema*)
>      (*formato da mensagem recebida: cs !destino ?processo-origem ?chamada*)
>      cs !ComT ?pedinte:pid ?chamada:chsist[EcsComT(chamada)];
>      ( [Esend(chamada)]                                   (*é chamada send?*)
>        => ChamadaSend[cs,int](pedinte,chamada)
>      [] [...] => ...)          (*demais chamadas do sistema tratadas pelo módulo
>                          ComunicacaoTransparente*)
>    )
>    |||
>    ( (*caso 2: serviços oferecidos aos outros gerenciadores e ao executor*)
>      (*formato da mensagem recebida*)
>      (*cs !destino ?processo-origem ?serviço-solicitado*)
>      cs !ComT ?pedinte:pid [not EPusu(pedinte)]
>         ?servico:servicos[EcsComT(chamada)] ;
>      ( [Exxx(servico)]                                    (*é serviço zzz?*)
>        => ServicoXxx[cs,int](pedinte,servico)
>      [] [...] => ...))                                    (*demais serviços oferecidos
>                                                  pelo módulo Comunicação Transparente*)
>    where ...
>    endproc
>
>    process ProtocolosDeAltoNivel[cs,int] :=
>       int ?... ; ... ; ...
>    endproc
> endproc
```

A estrutura do evento que ocorre na porta cs normalmente consiste de uma tupla de três elementos. O primeiro deles identifica o processo destinatário da mensagem trocada e, o segundo, identifica o processo que a enviou; ambos são do tipo pid (identificação de processo), o qual é definido em [Pin90]. O último elemento da tupla trocada na porta cs corresponde à solicitação do serviço propriamente dita; seu tipo varia conforme o evento considerado (chamada do sistema ou serviço oferecido a outros servidores do próprio SOD).

A interação entre processos paralelos com troca de valores ocorre quando eles oferecem a mesma estrutura em uma porta comum [BB87]. O recebimento de uma chamada do sistema é especificado pelo evento

```
> cs !ComT ?pedinte:pid ?chamada:chsist[EcsComT(chamada)]
```

onde:

- `!ComT` indica apenas mensagens destinadas ao módulo `ComunicacaoTransparente` são esperadas (esta constante é definida em [Pin90]);
- `?pedinte:pid` especifica o recebimento da identificação do pedinte;
- `?chamada:chsist [EcsComT(chamada)]` indica o recebimento de uma chamada do sistema, verificando se ela pertence ao conjunto daquelas referentes ao gerenciamento de processadores (`EcsComT` abrevia "É chamada do sistema referente ao servidor ComT?").

A ocorrência do evento citado na porta `cs` habilita a expressão de comportamento

```
> [Esend(chamada)] => ChamadaSend[cs,int](pedinte,chamada)
> [] ...
```

A chamada do sistema `send` é apenas um exemplo de possíveis chamadas referentes ao gerenciamento de comunicação. Dependendo da chamada solicitada, um diferente processo é ativado para tratá-la (`ChamadaSend, ...`). O recebimento da solicitação de um serviço, dentre aqueles oferecidos a outros servidores e ao executor, é especificado pelo evento

```
> cs !ComT ?pedinte:pid [not EPusu(pedinte)]
>    ?servico:servicos[EcsComT(chamada)]
```

onde:

- `!ComT` indica que apenas mensagens para o módulo `ComunicacaoTransparente` são esperadas.
- `?pedinte:pid [not EPusu(pedinte)]` não apenas especifica o recebimento da identificação do processo pedinte mas também a restrição de que estes serviços não estão disponíveis aos processos usuários (`EPusu` abrevia "É Processo usuário?").
- `?servico:servicos[EcsComT(chamada)]` indica o recebimento de uma solicitação de serviço, verificando se ele pertence ao conjunto de serviços oferecidos por `ComT` (`EServicoComT` abrevia "É serviço oferecido pelo servidor `ComT`").

A escolha que segue este evento é semelhante à do caso das chamadas do sistema, ou seja, de acordo com o serviço solicitado, um diferente proceso é selecionado para tratá-lo.

Quando se utiliza o modelo de servidores para implementar sistemas operacionais distribuídos, a comunicação entre processo é feita por meio do mecanismo de passagem de mensagens. Durante o desenvolvimento do RM-DOS, identificou-se a existência de

dois níveis de troca de mensagens: processos e computadores. Cada um deles é tratado por um diferente sub-módulo de `ComunicacaoTransparente`, os quais são o `Seletor` e o `ProtocolosDeAltoNivel`, respectivamente.

O sub-módulo `Seletor` suporta as mensagens a nível de processos usuários que correspondem àquelas trocadas entre processos usuários e servidores. Há uma grande variedade de primitivas que podem ser determinadas e cada sistema utiliza um conjunto diferente. Elas podem ser síncronas ou assíncronas, com ou sem *buffer*, etc [AS83,Cun81, Pra84,TR85].

O `Seletor` recebe mensagens para serem enviadas tanto a processos locais quanto a remotos. Caso o destino seja remoto, a mensagem é encaminhada ao sub-módulo `ProtocolosDeAltoNivel` através da porta `int`, o qual é responsável pelo suporte às mensagens em nível de computadores. As características das mensagens deste nível são determinadas pelo protocolo de comunicação utilizado. Como não se detém a sistemas específicos, o RM-DOS segue a recomendação OSI.

As sete camadas deste modelo são agrupadas em protocolos de alto nível — as quatro superiores[7] — e protocolos de baixo nível — as três inferiores[8]. Os protocolos de alto nível são independentes de máquina e costumam ser suportados por processos em nível dos usuários (processos servidores). Aqueles de baixo nível, por outro lado, dependem do equipamento utilizado e, portanto, devem ser suportados pelo executor. Assim, o módulo `ProtocolosDeAltoNivel`, como o próprio nome sugere, trata apenas das camadas superiores do protocolo de comunicação— toda comunicação remota é feita através deste sub-módulo, que interage com os demais servidores através da porta `cs`. Os protocolos de baixo nível são suportado pela `TarefaComunicacao`, parte integrante do executor.

4.2.2 Executor

O executor de um sistema operacional distribuído corresponde às duas camadas inferiores da modularização apresentada neste trabalho. Nada mais natural que sua expressão de comportamento seja formada pela composição de dois processos, cada um, deles responsável pelas funções atribuídas a uma destas camadas.

```
> process Executor[cs] :=
>    hide comlocal in
>    ( TarefasDeES[comlocal]                    (*camada dois do RM-DOS*)
>      |[comlocal]|
>      Nucleo[cs,comlocal] )                    (*camada um do RM-DOS*)
> where ... endproc
```

No executor, define-se a porta `comlocal` (`COM`unicacao `LOCal`) com o mesmo objetivo de comunicação interna entre sub-módulos verificado com a definição da porta `int` dos processos servidores.

[7] Aplicação, apresentação, sessão e transporte.

[8] Rede, enlace e física.

A expressão de comportamento do módulo TarefasDeES consiste, basicamente, da composição paralela de todas as tarefas de entrada/saída. Quanto ao Núcleo, este, conforme identificado em [Pin90], possui três funções que são escalonar processos, suportar a comunicação local e suportar as interrupções. A especificação formal proposta em RM-DOS para o núcleo é apresentada a seguir.

```
> process Nucleo[cs,comlocal] :=
>    hide esc,indicamsg in
>    ( SuporteAInterrupcoes[indicamsg]
>      |[indicamsg]|
>      ( ComunicacaoLocal[cs,comlocal,indicamsg,esc]
>        |[esc]|
>        Escalonador[esc] ))
> where ... endproc
```

De uma maneira geral, o núcleo funciona da seguinte forma. A ocorrência de uma interrupção é indicada ao módulo ComunicacaoLocal (através da porta indicamsg) que formula uma mensagem e a entrega ao módulo responsável por trata-la, cuja escolha depende do tipo da interrupção ocorrida: de *software* ou de *hardware*. A interrupção de *hardware* é gerada nos dispositivos de entrada/saída e, neste caso, o módulo ComunicacaoLocal envia a mensagem à tarefa de entrada/saída adequada (utilizando a porta comlocal).

A interrupção de *software* é normalmente decorrente de uma chamada do sistema (*read*, *exec*, *send*, etc.) ou da solicitação de um serviço do núcleo (escalonar, supender a execução de um processo, dentre outras). No primeiro caso, o módulo ComunicacaoLocal entrega a mensagem ao servidor responsável por tratá-la (utilizando a porta cs). No caso dos serviços do núcleo, ele mesmo trata, interagindo com o Escalonador através da porta esc.

As especificações dos três sub-processos, SuporteAInterrupcao, ComunicacaoLocal e Escalonador juntas, contém cerca de 80 linhas, sem considerar-se a definição dos tipos de dados utilizados. Elas possuem muitos detalhes relevantes uma vez que descrevem o comportamento de procedimentos que acessam o *hardware* diretamente, que efetuam a comunicação local entre processos e também que acessam o *hardware* diretamente, sendo, em alguns casos, ativadas por interrupção.

Analogamente ao fato do nível mais abstrato do RM-DOS ter especificado a interação do SOD com o ambiente dos processos usuários, desejou-se que seu nível mais refinado descrevesse a interação do sistema com o *hardware*. Para tal, uma das etapas finais da especificação consiste da descrição do comportamento do *hardware*, do ponto de vista do sistema operacional.

5 Estudo de Casos

A apresentação de exemplos é fundamental para que se possa verificar a coerência do RM-DOS. Com este objetivo, partes relevantes dos sistemas operacionais MINIX,DIOS

e V são descritos tanto informalmente como formalmente, de acordo com o modelo de referência proposto, em [Pin90].

Com o exemplo do sistema operacional MINIX, enfatiza-se a especificação dos módulos servidores, além da utilização do RM-DOS para projetar sistemas não distribuídos. A modularização de sistemas é o enfoque dos outros dois exemplos: DIOS e V. No primeiro deles, aborda-se os módulos de todo o sistema operacional, ou seja, tanto os servidores como o executor. No último, V, enfatiza-se o executor, módulo fundamental de qualquer sistema.

6 Conclusões

A modularização de sistemas operacionais distribuídos baseou-se na estruturação das funções referentes ao gerenciamento de recursos de um sistema. Uma vez identificadas as funções de um SOD, definiu-se um módulo para cada conjunto de funções equivalentes, tentanto-se manter a interface entre eles a menor possível.

Não foi objetivo definir qual o melhor algoritmo para alocar processos em processadores remotos, por exemplo, mas sim, definir qual o módulo que implementará esta função. Desta forma, permite-se o projeto de sistemas flexíveis, pois, uma melhoria em um dos seus módulos não não afeta os demais, caso sua interface com o ambiente seja mantida.

A utilização de LOTOS Completo mostrou que esta linguagem é adequada à especificação do comportamento de sistemas operacionais, mesmo daqueles módulos dependentes de máquina. Além disto, com a utilização de tipos de dados, o que é relativamente pouco comum ao se trabalhar com LOTOS, foi possível observar o grande poder de expressão desta linguagem.

Acredita-se que a coerência do modelo possa ser avaliada com a sua utilização para especificar o comportamento de sistemas operacionais reais. Alguns exemplos podem ser encontrados em [Pin90].

Referências

[AS83] G. R. Andrews e F. B. Schneider. Concepts and notations for concurrent programming. *Computing surveys*, 15(1):1–43, Março de 1983.

[BB87] T. Bolognesi e E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems*, (14):25–29, 1987.

[Bri86] E. Brinksma. A tutorial on LOTOS. Em *Proceedings of IFIP Workshop "Protocols Specification Testing and Verification V"*, páginas 171–194, Amsterdam, 1986.

[Che88] D. A. Cheriton. The V Distributed System. *Communications of the ACM*, 31(3):314–333, Março de 1988.

[Cun81] P. R. F. Cunha. *Design and analysis of message oriented programs.* Tese de doutorado, Universidade de Waterloo, Ontario, 1981.

[DZ83] J. D. Day e H. Zimmermann. The OSI Reference Model. *Proceedings of the IEEE*, 71(12):1334–1340, Dezembro de 1983.

[Fer89] C. A. G. Ferraz. *Um Estudo para o Desenvolvimento de Protótipos de Especificações LOTOS através de Programação Funcional.* Tese de mestrado, Universidade Federal de Pernambuco, Dezembro de 1989.

[ISO87] ISO. *LOTOS - A Formal Description Technic Based on Temporal Ordering of Observational Behaviour.* ISO, 1987. ISO/TC 97/SC 21.

[Mou89] M. T. S. Moura. *Desenvolvimento Estruturado de Especificações LOTOS.* Tese de mestrado, Universidade Federal de Pernambuco, Dezembro de 1989.

[Pin90] T. S. M. Pinheiro. *Modelo de Referência para Projeto e Especificação de Sistemas Operacionais Distribuídos.* Tese de mestrado, Universidade Federal de Pernambuco, Março de 1990.

[Pra84] T. W. Pratt. *Programing Languages, Design and implementation.* Prentice-Hall, 1984.

[Tan87] A. S. Tanenbaum. *Operating Systems - Design and Implementation.* Prentice-Hall, 1987.

[TR85] A. S. Tanenbaum e R. V. Renesse. Distributed Operating Systems. *Computing Surveys*, 17(4):419–470, Dezembro de 1985.