José A.M. QUEIROZ*

SUMÁRIO

Este trabalho apresenta o kernel distribuído para suporte da linguagem Constructor. Nele são descritas as funções do kernel, as estruturas de dados e as primitivas do kernel. Considerações são feitas sobre uma implementação do ambiente Constructor na UFPE.

ABSTRACT

This work describes the distributed kernel Constructor. It also gives the kernel functions, the data structures, and the kernel primitives. An intended implementation of the Constructor environment is presented.

* Engenheiro Civil (EE/UFPE - 1973) e Mestre em Ciência da Computação (PGCC/UFRGS-1984); Sistemas Distribuídos e Sistemas Operacionais Distribuídos; Professor Assistente do Departamento de Informática da UFPE, Analista de Sistemas do Núcleo de Processamento de Dados da UFPE e membro do Grupo de Redes de Computadores e Sistemas Distribuídos da UFPE; REDIS/UFPE - Caixa Postal 7474 - Recife - PE; Fone: (081) 271 2510; TELEX (081) 1267.

INTRODUÇÃO

A construção de um sistema distribuído envolve o projeto de diversos elementos que, em seus funcionamentos, trabalham cooperativamente para que recursos dispersos sejam utilizados na prestação de diversos serviços. O modelo de arquitetura de sistema distribuído (QUE 84) se baseia no modelo de objeto (JON 78) onde a interação entre processos é feita através da invocação de operações definidas por outros processos (AND 82).

A base para implementação do modelo de sistema distribuído é constituída por um conjunto de sistemas de computação autônomos, interconectados através de um subsistema de comunicação. Além do nível correspondente ao subsistema de comunicação, o modelo envolve três níveis: kernel, sistema operacional e aplicações. O kernel de cada nodo oferece, entre outras coisas, facilidades para sincronização e intercomunicação de processos e facilidades para atendimento de interrupções. Além disso, ele trata da comunicação com o subsistema de comunicação. O kernel distribuído é constituído pelo consórcio de todos os kernels dos nodos componentes interconectados através do subsistema de comunicação. Acima dele, o restante do sistema distribuído (nível de sistema operacional e nível de aplicações) é implantado por processos que interagem através das funções implementadas pelo kernel.

Cada novo componente possui uma cópia do kernel que fornece os mecanismos necessários para que os conceitos de operações e objetos sejam implementados. A interação entre processos pode ocorrer para processos em um mesmo nodo (interação local) ou em nodos distintos (interação remota). O kernel deve fornecer mecanismos para que processos interajam do mesmo modo, independentemente da localização dos mesmos e independentemente de serem processos fornecidos pelo sistema ou por usuário.

O kernel componente é constituído de um conjunto de rotinas e estruturas de dados que implementam as funções básicas (primitivas) do sistema executadas de forma não interrompível. Os processos se comunicam com o kernel através de instruções especiais ('traps'), tipo chamadas do supervisor, e, através delas, solicitam serviços. O processador é passado ao kernel após um 'trap'. Os dispositivos e o relógio pedem a atenção do kernel através de interrupções. Quando o kernel não está em execução e uma interrupção ocorre, o processador é passado para ele. Durante a maior parte do seu tempo o kernel deverá estar envolvido na execução de facilidades para interação de processos. Para interações remotas ele se utilizará dos serviços do subsistema de comunicação.

A definição do kernel busca atingir metas que o tornem tão pequeno e simples quanto possível. Esta minimização de tamanho e de complexidade aumenta a probabilidade de se obter uma implementação correta para ele. Os kernels pequenos, simples e também práticos oferecem as condições de segurança para que o software que vai ser suportado possa ser mais confiável (POP 78). O kernel deve comportar todo o software que se relacione com a segu-

ça do sistema. Funções não importantes para a segurança do sistema podem ser implementadas por processos em um nível acima do kernel.

FUNÇÕES DO KERNEL

A tarefa principal a ser realizada pelo kernel é a de implementar os conceitos de programação da linguagem Constructor (BUZ 84). Com isto um ambiente de programação Constructor poderá ser fornecido aos usuários, permitindo a definição de processos para serem executados nos diversos componentes do sistema distribuído. Cada processo terá para si sua própria unidade de processamento virtual. Estas unidades virtuais são como as unidades reais; a diferença entre elas é apenas a menor velocidade de execução das unidades virtuais que são implementadas pela multiplexação das unidades reais, o que resulta numa taxa de execução variável para os processos.

As funções que o kernel executa para implementar um ambiente Constructor e para atender às necessidades dos processos, dando a cada um a impressão de possuir sua própria unidade de processamento, se dividem entre: funções para implementação e gerenciamento de processos, funções para sincronização e intercomunicação de processos, funções para gerenciamento de áreas de armazenamento para passagem de parâmetros, e funções para manipulação de interrupções e execução de operações de entrada/saída.

Implementação e Gerenciamento de Processos

Os processos são entidades de execução independentes que podem interagir com outros processos através de operações. Os processos executam, ao menos conceitualmente, de forma concorrente e têm acesso exclusivo às suas variáveis locais.

Cada processo é representado no kernel por um registro descritor. O registro descritor contém informações necessárias para colocar o processo em execução; estas informações incluem o contador de instruções, os registradores gerais e de endereçamento, e outras. Isto permite que o processador seja multiplexado pelos diversos processos que concorrem por ele.

Ao serem inicializados os processos são colocados em uma fila de processos aptos ('ready') para receberem o processador. Sempre que acaba uma tarefa, o kernel deve selecionar um processo na fila de aptos para receber o processador. O escalonamento faz com que um processo seja retirado da fila de aptos e que seu ambiente de execução, que estava guardado no seu descritor, seja restaurado. A fila de aptos é organizada de acordo com a prioridade e com o tempo de entrada dos processos. O processo escalonado assume o controle do processador e fica em execução ('running') até que ele necessite de um serviço do kernel ou seja interrompido. A solicitação, por um processo, de serviços do kernel faz com que o estado atual do solicitante seja salvo e que ele seja bloqueado ('blocked') a espera do atendimento do serviço solicitado. Quando o serviço solicitado for concluído o processo que estava bloqueado passará para a fila de aptos. Quando um processo em execução é

interrompido, ou pelo fim de sua fatia de tempo ou por uma interrupção de dispositivo, o estado atual do processo é salvo e ele é colocado na fila de aptos. Um processo em execução pode decidir por seu próprio encerramento e isto fará com que seu descritor seja marcado como correspondente a processo terminado. O término normal de um programa Constructor ocorrerá quando terminarem todos os processos. Mecanismos para que processos exerçam controle sobre outros processos fazem com que novos estados sejam incorporados aos que foram descritos.

Sincronização e Comunicação entre Processos

Embora independentes, os processos devem interagir para executar as suas tarefas, as quais, em geral, são partes (sub-tarefas) de uma tarefa maior, comum a todos, a ser executada de modo cooperativo.

Os processos se comunicam e sincronizam através de operações. Um processo que deseja interagir com outro pode fazê-lo através da invocação de uma operação que este outro implemente. Através dos parâmetros da invocação o processo invocante envia informações para o processo invocado. Esta invocação pode ser uma invocação síncrona feita através de um comando de chamada CALL, ou uma invocação assíncrona feita através de um comando de envio SEND. A operação invocada estará implementada em um processo, embutida em um comando de seleção SELECT. Este comando, cada vez que for executado, selecionará uma operação para atender, dentre um conjunto de operações pendentes. Se a invocação atendida tiver sido feita por um CALL, uma resposta será enviada para o processo invocante. O processo invocado pode passar informações para o processo invocante através dos parâmetros de saída da invocação. Complementando este mecanismo para sincronização e comunicação entre processos, uma facilidade é fornecida para que diversas invocações síncronas (CALLS) possam ser disparadas concorrentemente, o comando de invocação concorrente CONC. O comando CONC permite que um processo possa invocar concorrentemente diversas operações que podem estar implementadas por diversos processos. O processo invocante será bloqueado depois que todas as invocações forem feitas e, voltará a ficar apto e concorrer pelo processador depois que todas as respostas às invocações feitas sejam recebidas.

Os processos que interagem podem estar localizados em nodos processadores diferentes. Para um processo a interação com um outro é feita de forma independente da localização desse outro. Quem distingue entre invocação local e invocação remota é o kernel, o qual, no caso de interações remotas lança mão do subsistema de comunicação.

A cada operação implementada está associada uma lista com invocações pendentes para a operação. Uma invocação pendente indica qual o processo que fez a invocação e onde estão os parâmetros atuais da invocação. Estas listas são mantidas pelo kernel.

A invocação de uma operação faz com que o kernel insira um novo elemento na lista de invocações pendentes para a operação, contendo: identificação do processo, tipo de invocação e ponteiro para área de parâmetros. Após, é verificado se o processo que implementa

a operação invocada está bloqueado e se a invocação corrente pode fazer com que esse processo seja continuado. Se isto ocorrer o kernel acorda o processo, colocando-o na fila de aptos. Se a invocação foi feita por um comando CALL, o processo invocante é bloqueado e um novo processo é escalonado. Se a invocação foi feita por comando SEND, o processo invocante não é bloqueado e continua executando. A invocação feita por comando CALL dentro do comando CONC origina ações que são uma mistura das ações correspondentes aos comandos SEND e CALL. O processo que fez invocações concorrentes só será bloqueado depois que todas as suas chamadas CALL forem disparadas. Para este processo ser colocado novamente na fila de aptos é necessário que ele receba todas as respostas das operações que foram invocadas.

Para que um processo atenda a uma invocação de operação é necessário que o seu comando de seleção SELECT que implementa a operação seja executado. A execução do comando SELECT está dividida em duas partes: na primeira delas é feita a seleção de uma operação para executar e na segunda os comandos que implementam a operação selecionada são executados, de acordo com os parâmetros da invocação escolhida para ser atendida. Durante a fase de seleção uma operação pode ou não ser escolhida para execução. Se não há seleção de uma operação e o comando possui a cláusula ELSE, os comandos dessa cláusula são executados e a execução do SELECT é concluída. Caso uma operação não tenha sido selecionada e o comando SELECT não possua cláusula ELSE, este processo deve ser bloqueado. Quando o bloqueio do processo é feito são passados como argumento desse bloqueio uma máscara que indica as operações que, se invocadas, podem fazer com que o comando seja completado, e uma informação que indica se a alteração de variáveis globais do recurso podem tornar verdadeiras uma ou mais expressões de sincronização. A execução do comando SELECT é retardada até que uma operação que ele implemente possa vir a ser selecionada. Quando uma operação é selecionada para ser executada, ela é removida da lista de invocações pendentes da operação. A operação é então executada de acordo com os parâmetros atuais da invocação selecionada. Quando a execução da operação termina, também termina a execução do comando SELECT. Aí, então, o kernel verifica o tipo de invocação que foi atendido. Se a invocação foi feita por comando SEND a área alocada para passagem de parâmetros é devolvida. A resposta de uma invocação feita por comando CALL faz com que o processo invocante seja despertado caso a invocação tenha sido de um comando CALL isolado ou caso todas as invocações de um comando CONC já tenham sido atendidas.

O processo que esteve bloqueado por causa de uma invocação síncrona (CALL ou CONC), depois que é escalonada e assume o processador deve recuperar os resultados da execução das operações que ele invocou. Se os resultados se encontram em áreas que foram alocadas dinamicamente, estas áreas devem ser devolvidas. Isto completa a execução de um comando CALL ou CONC, e o processo continua executando a partir do seu próximo comando.

Gerenciamento de Áreas de Armazenamento de Parâmetros de Operações

Os processos de um programa Constructor trocam informações entre si através dos parâme-

tros que são passados quando operações são invocadas e quando operações são executadas com produção de resultados (resposta). Para que estes parâmetros sejam passados entre os processos é necessário que áreas para armazenamento dos valores desses parâmetros sejam usadas. Estas áreas são preenchidas quando invocações de operações que passam parâmetros são realizadas, e são atualizadas depois que operações que produzem resultados são executadas. O fornecimento dessas áreas de armazenamento de parâmetros é feito pelo kernel.

Dois tipos de áreas são supridos: áreas permanentes e áreas dinâmicas. As áreas permanentes são aquelas alocadas para o processo quando da sua inicialização. Cada processo, ao ser inicializado, é associado a uma área fixa para passagem de parâmetros. Esta área será utilizada para os parâmetros das chamadas CALL-simples (não embutidas em comandos CONC). As áreas dinâmicas são alocadas durante a execução dos processos de acordo com a necessidade. A alocação de uma área dinâmica é solicitada quando é executado um comando SEND ou um comando CALL embutido em comando CONC, que necessitem passar parâmetros.

O kernel possuirá uma área para atender às solicitações de alocação. A alocação de áreas é feita durante a inicialização de um processo, alocação de área permanente, ou quando solicitado por um processo, alocação de área dinâmica. Se uma área for alocada por um comando SEND ou por um comando CALL-embutido, ela ficará associada temporariamente ao processo.

Manipulação de Interrupções e Execução de Operações de Entrada/Saída

Os mecanismos específicos para atendimento de interrupções e operações de E/S são dependentes de máquina.

Como a linguagem Constructor é destinada também ao desenvolvimento de software básico, é conveniente que os mecanismos utilizados para sincronização e intercomunicação de processos possam ser usados para o atendimento de interrupções e execução de operações de E/S. O kernel deverá possibilitar que as ocorrências de interrupções possam ser mapeadas em invocações de operações e o compilador deverá possibilitar que variáveis do programa possam ser alocadas em posições absolutas de memória associadas a dispositivos de E/S. Por motivos de segurança estes recursos da linguagem só deverão ser permitidos dentro de módulos especiais.

Para permitir que interrupções possam ser transformadas em chamadas de operações deve existir no kernel uma rotina que transforme ocorrência de interrupções em invocação de operações. O endereço dessa rotina deve ser colocado em todas as entradas associadas a interrupções. Esta rotina deve ter acesso a uma tabela que indique, para cada interrupção, qual é a operação a ser invocada. Quando esta rotina recebe o controle é sinal que ocorreu uma interrupção para qual existe operação associada. Após descobrir que interrupção ocorreu, a rotina simula a ocorrência de uma invocação SEND, sem parâmetros, para a operação correspondente.

Para implementar operações de E/S em um dispositivo, por exemplo, o programador associa

variáveis aos registradores de estado, de comando, e de dados desse dispositivo e associa uma operação às interrupções respectivas. Atribuindo valores às variáveis associadas aos registradores de comando e de dados consegue-se iniciar a operação do dispositivo.

Quando acabar a operação de E/S comandada (isto é, quando acontecer a interrupção do dispositivo), irá ocorrer a invocação da operação associada, a qual poderá obter o resultado da operação de E/S executada consultando os valores das variáveis correspondentes aos registradores de estado e de dados do dispositivo.

Convém observar que a existência destes recursos possibilita ao programador da linguagem de alto nível Constructor o domínio dos sistemas de interrupções e de E/S.

ESTRUTURAS DE DADOS DO KERNEL

O kernel para fornecer um ambiente de execução Constructor deve manter estruturas de dados que permitam a implantação de processos e de operações. Estas estruturas auxiliarão o kernel no fornecimento das unidades de processamento virtuais para todos os processos do sistema componente. Além disso, através delas, será fornecido o suporte para que processos interajam e para que interrupções sejam atendidas.

As principais estruturas de dados são: os descritores de processos, as áreas de parâmetros, as listas de invocação de operações, as filas de processos, o indicador de urgência e a tabela de mapeamento de interrupção em operação.

PRIMITIVAS DO KERNEL

As funções básicas primitivas do kernel são relacionadas a seguir. Elas são: primitiva de inicialização, primitivas para manipulação de áreas para parâmetros de operação, primitivas correspondentes aos comandos de interações e primitivas para controle de processos.

A primitiva de inicialização de processos *inicproc* é invocada durate a fase de inicialização do sistema. Para que um processo seja completamente criado é necessário que sejam alocadas para ele um registro descritor de processo, as cabeças de filas de operações e a área permanente para passagem de parâmetros. O registro descritor é então inicializado e passa a representar o processo perante o kernel.

As primitivas para manipulação de áreas para parâmetros de operações *allocate* e *deallocate* são usadas para solicitar e devolver, respectivamente, áreas para armazenamento de parâmetros de operações. Durante a inicialização dos processos a primitiva *allocate* é invocada para fornecer áreas permanentes para armazenamento de parâmetros de operações. Quando os processos estiverem em execução, a primitiva *allocate* será invocada quando um comando SEND ou CALL-embutido for executado ou quando uma invocação remota para uma operação local for recebida pelo kernel, para que seja fornecida uma área dinâmica para armazenamento de parâmetro de operações. A primitiva *deallocate* é invocada para que uma área dinâmica previamente alocada seja devolvida ao kernel.

As primitivas correspondentes aos comandos de interação são: a primitiva *invoke* , a primitiva *endconc*, a primitiva *delay*, a primitiva *invrem* e a primitiva *reply*. A primitiva *invoke* é usada pelos comandos CALL e SEND para solicitar que o kernel encaminhe a invocação de uma operação ao processo que a implementa. A primitiva *endconc* é utilizada para sinalizar a finalização de um comando CONC. A primitiva *delay* é invocada para retardar a execução de um comando SELECT por não ser possível fazer uma seleção de operação a executar e o comando não possuir cláusula ELSE. Quando uma operação é selecionada para execução, a invocação premiada com a seleção deve ser removida da lista de invocações pendentes e isto é feito através da invocação da primitiva *invrem*. A primitiva *reply* é invocada depois que a execução de uma operação é concluída. Ela é usada para avisar que a invocação da operação foi atendida.

As primitivas para controle de processos são: a primitiva *suspend*, a primitiva *activate*, a primitiva *restart*, a primitiva *kill* e a primitiva *quit*. Estas primitivas permitem que processos exerçam controle sobre outros processos (processos controlados).

SIMULAÇÃO DO KERNEL DISTRIBUÍDO EM AMBINETE CENTRALIZADO

Visando um estreitamento na relação com esses novos conceitos de programação, buscou-se implementar um ambiente onde experiências possam ser realizadas. O ambiente escolhido foi o do sistema centralizado DEC-10 da UFPE. Neste ambiente estão sendo simulados os processos e o kernel distribuído. Os processos e o kernel estão programados em Pascal e se utilizam das Facilidades de Comunicação Inter-Processos (IPCF-Inter-Process Communication Facility) do DEC-10 para simular as facilidades de comunicação do sistema. O IPCF permite que processos se comuniquem uns com os outros através do envio e da recepção de pacotes de informações.

Cópias do kernel componente são carregadas no sistema e interconectadas através do IPCF, compondo, assim, o kernel distribuído. Sobre este kernel são clocados os processos que simulam as interações processo-kernel que são realizadas durante a execução de um programa distribuído.

Com este experimento podem ser testados os novos conceitos introduzidos na linguagem Constructor.

CONCLUSÕES

O kernel projetado pode ser implementado com relativa facilidade, tornando disponível ferramentas poderosas para a construção de sistemas distribuídos. Estas ferramentas podem ser utilizadas a partir de versões mais simples do sistema Constructor, que possam ser mais facilmente implementáveis (o sistema Constructor na sua versão mais completa requererá, certamente, um enorme esforço para o seu desenvolvimento). Este kernel pode ser aproveitado a nível de programação assembler, independentemente de qualquer suporte de linguagem de alto nível.

A disponibilidade de uma implementação do sistema, por mais simplificada que fosse a versão implementada, possibilitaria o desenvolvimento de vários tipos de experiências orientadas a aplicações distribuídas (mesmo que o equipamento não fosse fisicamente distribuído, as aplicações seriam logicamente distribuídas).

O trabalho de implementação está se desenvolvendo em um equipamento tradicional (não distribuído). Ainda assim o trabalho é válido pois permite o desenvolvimento de experiências com esse novo estilo de programação baseado no conceito de objetos e operações remotas.

REFERÊNCIAS

(AND 82) ANDREWS, Gregory R. The Distributed Programming Language SR-Mechanisms, Design and Implementation. *Software - Practice and Experience*, 12(8):719-53, Aug. 1982.

(BUZ 84) BUZIN, Paulo F.W.K. *Uma Abordagem para a Concepção e Implementação de Sistemas Distribuídos*. Porto Alegre, PGCC-UFRGS, 1984.

(JON 78) JONES, Anita K. The Object Model: A Conceptual Tool for Structuring Software. In: ADVANCED COURSE ON OPERATING SYSTEMS, MÜnche, Mar. 29-Apr.6, 1978. Berlin, Springer-Verlag, 1978. Cap. 2A. p.8-16. (Lecture Notes in Computer Science, 60)

(POP 78) POPEK, Gerald J. & KLINE, Charles S. Issues in Kernel Design. In: ADVANCED COURSE ON OPERATING SYSTEMS, MÜnche, Mar. 29-Apr.6, 1978. Berlin, Springer-Verlag, 1978. Cap. 3B. p.210-27. (Lecture Notes in Computer Science, 60)

(QUE 84) QUEIROZ, José A.M. de. *Um Modelo de Sistema Distribuído e Proposta de Kernel para a REDURGS*. Porto Alegre, PGCC-UFRGS, 1984.