15. ESTUDO DE CASOS

Neste capítulo vamos apresentar dois exemplos de sistema operacional distribuído: Amoeba e Mach.

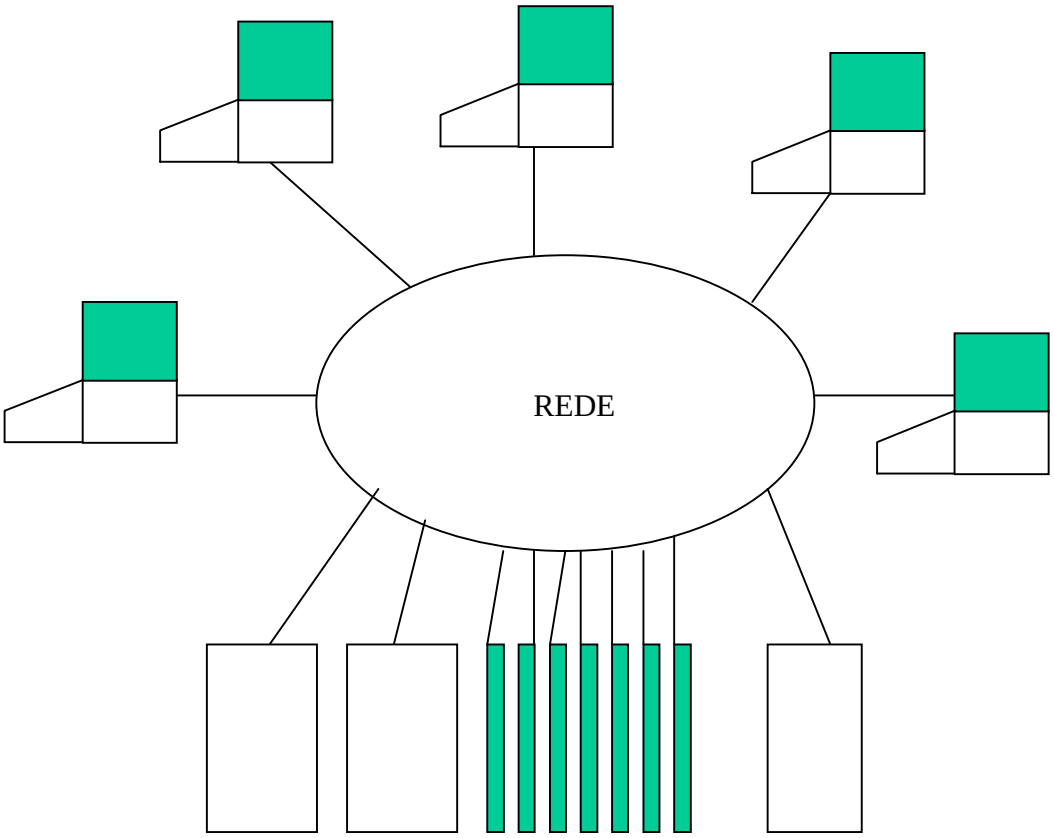
15.1 Amoeba

O sistema Amoeba é originário da Universidade de Vrije em Amsterdam em um projeto de pesquisa em computação paralela e distribuída em 1981 (Andrew S. Tanenbaum e estudantes de doutorado). Várias versões foram desenvolvidas desde então, a versão aqui descrita será a 5.2.
A abordagem do projeto Amoeba foi diferente de outros projetos em sistemas operacionais distribuídos, adicionar novas características a sistemas existentes (rede, sistema de arquivos compartilhado) para torná-los mais distribuídos. O Amoeba optou por desenvolver o sistema desde o zero sem se importar com compatibilidade e usando idéias novas para sua implementação. A meta importante do projeto era construir um sistema operacional distribuído transparente, ou seja, o usuário usa o Amoeba como se fosse um sistema operacional timesharing tradicional, a diferença é que as ações realizadas pelo sistema envolvem a participação de várias máquinas, incluindo servidores de processos, servidores de arquivos, servidores de diretórios, etc, sem que o usuário saiba disto. No Amoeba não existe o conceito de máquina própria, ou seja, quando o usuário entra ele o faz no sistema como um todo e não em uma máquina específica. Todos os recursos pertencem ao sistema como um todo e são gerenciados desta forma. Uma segunda meta do sistema é fornecer um ambiente apropriado para programação paralela e distribuída.
Como metas básicas de projeto o Amoeba apresenta:

1. Transparência de Rede – Todos os acessos a recursos devem ser transparentes de rede. Em particular, a existência de um sistema de arquivos para o sistema como um todo, e os processos executam em processadores escolhidos pelo sistema sem o conhecimento do usuário.
2. Gerência de recursos baseado em Objetos – O sistema foi projetado para ser baseado em objetos. Cada recurso é considerado como um objeto e todos os objetos, independente do seu tipo, são acessados por um esquema de nomes uniforme. Os objetos são gerenciados por um servidor e o acesso a eles é feito enviando mensagens para os servidores, mesmo localmente.
3. Servidores a nível de usuário – O sistema foi construído, tanto quanto possível, como uma coleção de servidores executando, no nível de usuário, em cima de um microkernel padrão que roda em todas as máquinas do sistema. Atenção particular foi dada ao problema de proteção, o microkernel do Amoeba utiliza um modelo uniforme para acesso aos recursos através de capacidades.

Arquitetura do Amoeba. O Amoeba foi projetado com duas asserções em mente: 1) Os sistemas terão um grande número de CPUs e, 2) Cada CPU terá dezenas de megabytes de memória. Provavelmente, no futuro, a maioria das instalações será compatível com estas asserções. A idéia da arquitetura do sistema é suportar a necessidade de incorporar um grande número de CPUs de uma forma não complexa. Suponha que você possa prover para

cada usuário um sistema multiprocessador com 10 ou 100 processadores, o que você faria? Provavelmente a solução seria dar um destes sistemas para cada usuário. Entretanto, talvez esta não seria a melhor forma e nem efetiva de gastar o orçamento, principalmente porque a maior parte do tempo os processadores estariam ociosos para a maioria dos usuários, enquanto que alguns precisariam executar programas necessitando um grande poder computacional. Desta forma, o Amoeba é baseado no modelo *pool de processadores* mostrado anteriormente. Ele é baseado no modelo mostrado na figura abaixo.



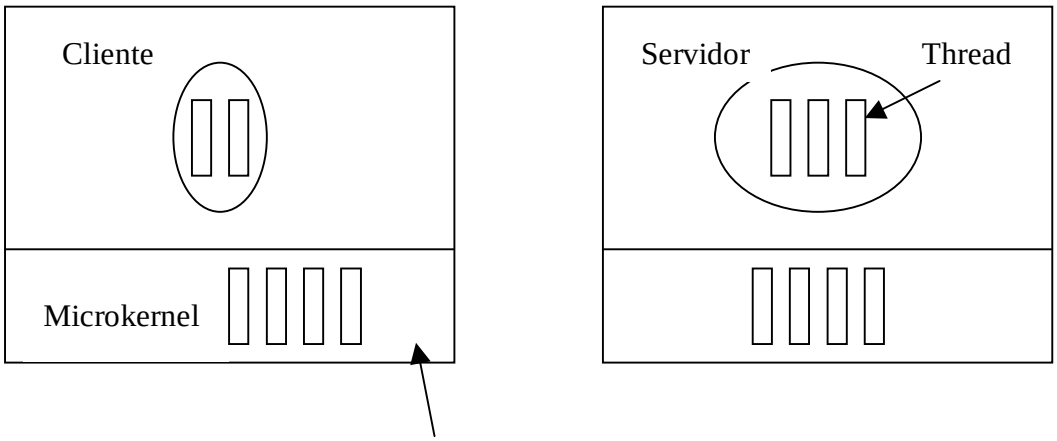
Neste modelo todo o poder computacional está localizado em um ou mais *conjuntos de processadores*. Um conjunto de processadores é composto por um número de CPUs sendo que cada uma com a sua memória local e conexão em rede. As CPUs podem ser de qualquer tipo, diferentes arquiteturas. Quando o usuário entra com um comando, o sistema operacional escolhe dinamicamente um ou mais processadores para executar o comando. Quando o comando é terminado, os processos são terminados e os recursos devolvidos para o conjunto de processadores. Os processadores por sua vez podem ser compartilhados, executando mais de um processo por vez.

Um segundo elemento na arquitetura do Amoeba é o *terminal*, é através dele que o usuário acessa o sistema (tipico: terminal X, tela grande bit-mapped e mouse). A idéia aqui é usar terminais baratos e concentrar o poder computacional no conjunto de processadores,

embora seja possível executar programas no terminal. Por exemplo, ao invés de comprar 100 estações de alto desempenho para 100 usuários, poderíamos comprar 50 processadores de alto desempenho, que compõe o conjunto de processadores, e 100 terminais X pelo mesmo preço. Uma vez que o conjunto de processadores são alocados só quando necessário, um usuário ocioso bloqueia apenas um terrminal X e não uma estação.

Outro componente importante são os servidores. Na maioria das vezes estes servidores, por características específicas, executam em processadores determinados. Os servidores fornecem serviços. Um serviço é uma definição abstrata daquilo que o servidor está preparado para fazer para os seus clientes.

O microkernel Amoeba. O modelo de software adotado pelo Amoeba apresenta duas partes fundamentais: o microkernel, que executa em cada processador e uma coleção de servidores que fornecem a maioria das funções tradicionais de um sistema operacional, como mostra a figura abaixo.



O microkernel Amoeba executa em todas as máquinas do sistema (conjunto de processadores, terminal ou servidor). As funções básicas do microkernel são:

1. Gerenciamento de processos e threads
2. Fornecer suporte para gerenciamento de memória (alocação/lib segmentos)
3. Suporte a comunicação (ponto a ponto, em grupo)
4. Tratamento de I/O de baixo nível (device driver)

Os servidores Amoeba. Tudo que não é feito pelo kernel é feito pelos servidores. Desta forma, é possível minimizar o tamanho do kernel e melhorar a flexibilidade. O Amoeba é baseado no modelo cliente-servidor. Um conceito central no projeto de software é o

conceito de objeto, que é visto como um tipo de dados abstrato. Os objetos são gerenciados por servidores, arquivos, diretórios, segmentos de memória, janelas, processadores, discos e fitas são exemplos de objetos. Quando um processo cria um objeto (arquivo), o servidor que gerencia o objeto (servidor de arquivos) retorna para o cliente uma capacidade (criptografada) para aquele objeto. Para usar o objeto, a capacidade associada ao mesmo deve ser apresentada. Todos os objetos no sistema, software e hardware, são protegidos, tem nome e são gerenciados por capacidades.
Para usar um servidor, o usuário utiliza um procedimento *stub* (biblioteca) que organiza os parâmetros, envia a mensagem e bloqueia até que a resposta volte. Este mecanismo esconde todos os detalhes de implementação do usuário. Alguns servidores importantes do Amoeba são:

- servidor *bullet* : servidor de arquivos, responsável pelo gerenciamento, criação, leitura/escrita, deleção de arquivos. Os arquivos são imutáveis. Facilita a replicação automática e evita muitas condições de corrida que são próprias da replicação de arquivos que podem mudar durante o processo de replicação.
- servidor de diretórios (soap server) : responsável pelo gerenciamento de diretórios e nomes de caminhos (path names) e pelo mapeamento dos mesmos em capacidades. Na leitura de um arquivo o processo requisita ao servidor de diretórios para encontrar o nome, este retorna a capacidade do arquivo em caso de sucesso.
- outros servidores são responsáveis por funções como: tratamento de replicação de um objeto, iniciar processos, monitorar servidores para falhas e comunicação com o mundo exterior.

Objetos e Capacidades no Amoeba

Todos os servidores e serviços do Amoeba tem como base um conceito básico que é o objeto. Um objeto é o encapsulamento de um conjunto de dados sobre os quais operações bem definidas podem ser realizadas (tipo de dados abstrato). Os objetos são passivos. Eles não possuem processos ou métodos ou outra entidade ativa que faz coisas. Cada objeto é gerenciado por um processo servidor. Para realizar uma operação em um objeto, um cliente executa uma RPC com o servidor e especifica o objeto, a operação e opcionalmente parâmetros necessários. O servidor realiza o trabalho e devolve uma resposta. As operações são síncronas, o cliente bloqueia até que o servidor responde. Os clientes não conhecem a localização dos objetos e dos servidores.

Capacidades. Os objetos são nomeados e protegidos de uma forma uniforme, por recibos especiais chamados de *capacidades*. Para criar um objeto o cliente executa um RPC com o servidor apropriado. O servidor executa o serviço e devolve a capacidade do objeto, para executar outras operação no objeto o cliente deve apresentar a capacidade. Uma capacidade é um número binário com o seguinte formato:

Porta do Servidor	Objeto	Ac.	Verificador

Porta do servidor (server port) : permite ao kernel localizar a máquina na qual o servidor requisitado esta localizado. É o endereço lógico do servidor sendo assim está associado a um servidor ou grupo de servidores (48 bits).
Objeto : usado pelo servidor para identificar o objeto específico em questão. Por exemplo, no caso de arquivos funciona um i-node no UNIX (24 bits).
Direitos de Acesso : informa ao servidor quais as operações são permitidas para o dono desta capacidade (8 bits).
Verificador : este campo é usado para validar a capacidade (48 bits).

Proteção dos Objetos. O algoritmo básico para a proteção dos objetos é o seguinte: Quando o objeto é criado, o servidor pega randomicamente um campo *Verificador* e armazena na nova capacidade e nas suas tabelas internas. O bits de *Direitos de Acesso* são inicialmente todos ligados (1). Esta é a capacidade do *dono* que é retornada pelo servidor para o cliente. Quando a capacidade é enviada para o servidor, para que o mesmo realize alguma operação, ele verifica o campo *Verificador*.
Na criação de uma capacidade restrita, o cliente passa a capacidade para o servidor e junto os novos direitos (mascara de bits). O servidor pega o campo *Verificador* original de suas tabelas e faz um XOR com os novos direitos (que são um sub-conjunto dos direitos originais) e executa uma função de um caminho sobre o resultado. Desta forma é criada uma nova capacidade para o mesmo objeto porém com direitos diferentes e com o campo check também diferente. Esta nova capacidade é retornada para o chamador que pode então enviar esta capacidade para outro processo. Quando o servidor recebe uma capacidade restrita ele a identifica pelos direitos porque pelo menos 1 bit esta desligado (capacidade do dono todos bits ligados). O servidor busca o número randômico original das suas tabelas faz um XOR com os direitos atuais e passa o resultado pela função. Se o resultado é igual ao campo check, a capacidade é aceita como válida.
Este mecanismo leva a um esquema uniforme de nomeação e proteção que também é completamente transparente a localização. Não existem listas de acesso para autenticação, portanto sem overhead administrativo.

Operações Padrões

Apesar de muitas operações nos objetos dependerem do tipo de objeto existem algumas operações que aplicam-se a maioria dos objetos.

Age -	Coleta capacidades perdidas de objetos que não são mais acessíveis
Copy -	Faz uma duplicata do objeto e retorna uma capacidade para a copia
Destroy -	Deleta o objeto e libera seu espaço
Getparams	
Setparams -	Trabalha com os parametros do servidor
Info -	Retorna um string que descreve o objeto (tipo, etc)
Status -	Informações sobre o servidor (memória disponível)
Restrict -	Produz uma nova capacidade restrita para o objeto

Gerência de processos no Amoeba

Um processo no Amoeba é basicamente um espaço de endereçamento e uma coleção de threads que executam neste espaço.

Processos. Um processo é um objeto no Amoeba. Quando um processo é criado, o processo pai recebe uma capacidade para o filho. Através desta capacidade, o filho pode ser suspenso, reiniciado, sinalizado e destruído. A criação de processos no Amoeba é diferente do UNIX (muito overhead, FORK, EXEC) e é mais parecido com o MS-DOS (com a imagem de memória que se quer desde o início) só que é permitido ao processo pai continuar em paralelo com o filho e assim por diante. O gerenciamento de processos é tratado em 3 níveis:

- servidores de processos (nível mais baixo) : são threads do kernel que executam em cada uma das máquinas. Para criar um processo em uma determinada máquina, outro processo executa um RPC com o servidor de processos da máquina desejada e fornece as informações necessárias.
- conjunto de procedimentos de biblioteca que fornecem uma interface mais conveniente para programas de usuário.
- servidor de execução que é a maneira mais simples para criar um processo, utilizado para realizar a maior parte do trabalho na determinação de onde executar um novo processo.

As chamadas de gerenciamento de processos utilizam uma estrutura de dados chamada *descritor de processo* para fornecer informações sobre o processo a executar. Esta estrutura está descrita a seguir. Um campo do descritor fornece informação sobre que arquitetura (x86, Sparc, etc) o processo pode executar. Outro campo contém a capacidade do dono do processo, quando o processo termina esta capacidade é usada para fazer RPCs informando o evento. O descritor também inclui descritores para todos os segmentos do processo que juntos formam o espaço de endereçamento do mesmo, assim como descritores para as todas as threads do processo. O descritor de thread, dependente de arquitetura, contém no mínimo o PC e o SP.

Os procedimentos importantes da interface de baixo nível dos processos incluem *exec*, *getload*, *stun*. O primeiro procedimento tem 2 parâmetros, a capacidade do processo servidor e um descritor de processo, sua função é realizar um RPC com o processo servidor especificado pedindo para o mesmo executar o processo. Se a chamada é completada com sucesso, uma capacidade para o novo processo é retornada para o chamador. O segundo procedimento retorna informações a respeito da velocidade da CPU, carga atual e quantidade de memória livre, é usado pelo servidor de execução para determinar o melhor lugar para executar um processo novo. O terceiro procedimento pode ser usado pelo pai de um processo para suspende-lo tirando-lhe a consciência. Existem duas formas de *stun* : normal e emergência. A diferença básica diz respeito ao que acontece se o processo esta bloqueado em 1 ou mais RPCs. No normal, o processo envia uma mensagem para o servidor que ele esta esperando para informar o evento (eu preciso que termine o trabalho e me envie uma resposta). Na emergência, o processo é parado instantaneamente.

Threads. O modelo de threads suportado pelo Amoeba é simples, quando um processo inicia ele tem uma thread. Durante a execução, o processo pode criar e terminar outras threads, o número de threads é dinamico. Quando uma nova thread é criada, os parâmetros da chamada especificam o procedimento a executar e o tamanho da pilha inicial. É possível para uma thread criar variáveis que são visíveis para ela e todos os seus procedimentos mas não para outras threads (glocal). Existem três métodos usados para sincronizar threads: signals, mutexes e semáforos. 1)Signals são interrupções assíncronas enviadas de uma thread para outra no mesmo processo, 2)Mutex é um semáforo binário, podendo estar no estado bloqueado ou livre e, 3)Semáforos, considerados semáforos contadores, mais lentos que o mutex.
Todas as threads são gerenciadas pelo kernel. A vantagem é que quando uma thread faz um RPC, o kernel pode bloquear aquela thread e escalonar uma outra pronta do mesmo processo. O escalonamento da threads é feito por prioridades, onde as threads do kernel tem prioridade mais alta do que as de usuário. O escalonamento pode ser escolhido para ser preemptivo ou não (executa até o fim) conforme o desejo do processo.

Gerência de memória no Amoeba

O Amoeba tem um modelo de memória extremamente simples. Um processo pode ter um número qualquer de segmentos que podem estar localizados onde ele quiser dentro do espaço de endereçamento virtual do processo. Os segmentos não sofrem swap nem paginação, sendo que um processo tem que estar inteiramente na memória para executar. Mesmo com a utilização de hardware MMU, cada segmento é armazenado contiguamente na memória. Isto é devido a possibilidade de fazer RPC mais rápido. Outra razão é a simplicidade, ou seja, o fato de não ter paginação ou swapping faz com que o sistema seja mais simples e o kernel menor e mais fácil de gerenciar.

Segmentos. Os processos tem disponível algumas chamadas para gerenciar os segmentos. Chamadas que permitem criar, destruir, ler e escrever segmentos. Quando um segmento é criado, o chamador recebe de volta uma capacidade para o segmento que é utilizada para a leitura e escrita e outras chamadas que envolvem o segmento. Um segmento pode mudar de tamanho durante a execução do processo.

Segmentos mapeados. No Amoeba os espaços de endereçamento virtual são construídos a partir de segmentos. Quando um processo inicia, ele deve ter pelo menos 1 segmento. Entretanto, uma vez executando, um processo pode criar segmentos adicionais e mapeá-los no espaço de endereçamento em qualquer endereço virtual não usado. Um processo pode também desfazer um mapeamento ou especificar um conjunto de endereços virtuais que devem ser desmapeados. Quando isto acontece, uma capacidade é retornada assim o segmento continua ser acessado. Um segmento pode ser mapeado para dois ou mais processos ao mesmo tempo.

Comunicação no Amoeba

Duas formas de comunicação são suportadas pelo Amoeba: RPC, através de mensagens ponto a ponto do tipo *request reply* e comunicação em grupo, usando *broadcasting* ou *multicasting* ou ainda simulado pelo kernel.

Chamada Remota de Procedimento (RPC). Cliente envia uma mensagem de requisição para um servidor e espera pela resposta. As requisições são definidas para cada servidor e configuram a interface do mesmo que é composta por rotinas, *stubs*, que empacotam os parâmetros em mensagens e chamam as primitivas do kernel para enviar a mensagem. Para o RPC ser realizado é necessário que o cliente conheça o endereço do servidor, para isto cada thread pode escolher uma *porta* (número de 48 bits) que é usada como endereço para as mensagens enviadas para a thread.

Primitivas:

1. *get_request (&header, buffer, bytes)* - indica a disposição do servidor para receber de uma porta
2. *put_reply* - executada pelo servidor quando o mesmo tem uma resposta para enviar
3. *trans* - envia uma mensagem do cliente para o servidor e espera resposta

As duas primeiras são usadas para servidores, enquanto que a terceira é usada para clientes. Todas são chamadas de sistema, ou seja, elas chamam o kernel diretamente. Quando uma mensagem é transmitida pela rede, ela contém um cabeçalho e um corpo. O cabeçalho é uma estrutura fixa de 32 bytes. No caso de o servidor fazer uma chamada de sistema do tipo *get_request (&header, buffer, bytes)* o primeiro parâmetro diz ao kernel onde colocar o cabeçalho a ser recebido. O servidor antes de fazer a chamada deve inicializar o campo *Porta* do cabeçalho para que o kernel saiba qual porta cada servidor esta escutando. Quando a mensagem chega, o servidor é desbloqueado e verifica o cabeçalho. O campo *Signature* é para autenticação. Alguns campos são utilizados para a comunicação cliente/servidor: o campo *Private part* é usado para conter informações da capacidade do objeto, o campo *Command* é usado para indicar que tipo de operação deve ser realizada no objeto. Após completar seu trabalho o servidor chama a primitiva *put_reply(&header, buffer, bytes)* para enviar resposta para o cliente.
O cliente por sua vez quando precisa algum serviço de um servidor executa uma RPC através da chamada *trans(&header1, buffer1, bytes1, &header2, buffer2, bytes2),* que bloqueia o cliente até que a resposta do servidor chegue.
Para resolver problemas de segurança, o Amoeba utiliza criptografia. Cada porta é na realidade um par de portas: *get-port* (privada, servidor) e *put-port* (pública, cliente). O mapeamento é feito por uma função *one-way* (put-port=F(get-port). Quando o servidor executa uma chamada get-request ele coloca a put-port correspondente em uma tabela de escuta, quando uma mensagem chega do cliente a sua put-port é procurada na tabela. As get-port não circulam na rede. A semântica suportada no Amoeba para RPC é *at-most-once*.

Comunicação em grupo no Amoeba. Um grupo no Amoeba consiste de um ou mais processos que cooperam para executar alguma tarefa ou fornecer algum serviço, podendo os processos serem membros de vários grupos ao mesmo tempo. Os grupos são fechados, por uma questão de transparência.

Primitivas : As operações para comunicação em grupo no Amoeba são listadas na tabela a seguir.

Chamada	Descrição
CreateGroup	Criar novo grupo e setar parâmetros
JoinGroup	Tornar o chamador um novo membro do grupo
LeaveGroup	Retira chamador do grupo
SendToGroup	Envia mensagem para todos membros do grupo
ReceiveFromGroup	Bloqueia até receber uma mensagem do grupo
ResetGroup	Inicia recuperação após quebra

Quando o último membro de um grupo retira-se do mesmo, o grupo é destruído. No envio de mensagens aos membros do grupo o Amoeba suporta ordenação global, assegurando a recepção ordenada de mensagens (consistência sequencial). A ultima chamada fornece a possibilidade de recuperação em caso de quebra.

Implementação. A idéia principal da comunicação em grupo é uma comunicação broadcast confiável. No Amoeba a comunicação em grupo funciona da seguinte forma: quando uma aplicação inicia uma máquina é eleita como *sequenciadora*, se esta máquina quebra os membros restantes elegem outra. Uma seqüência para conseguir broadcast confiável segue:

1. O processo usuário executa um trap para o kernel, passando uma mensagem.
2. O kernel aceita a mensagem e bloqueia o processo.
3. O kernel envia uma mensagem ponto-a-ponto para o sequenciador.
4. Quando o sequenciador pega a mensagem, ele aloca o próximo numero de seqüência disponível, coloca o mesmo no cabeçalho e faz um broadcast da mensagem.
5. Quando o kernel enviante recebe o broadcast, ele desbloqueia o processo para continuar sua execução.

Quando o kernel executa 3 ele inicia um timer para controlar possíveis retransmissões. As possibilidades aqui são: 1) recebeu broadcast antes do timer, 2) timer esgotou. Uma terceira possibilidade pode acontecer quando um broadcast retorna antes do timer mas é um broadcast errado.
Quando o sequenciador recebe um pedido de broadcast ele verifica se a mensagem é uma retransmissão, informando que o broadcast já foi feito se positivo. Se é uma mensagem nova, atribui um numero e armazena a mensagem num buffer.
Quando o kernel recebe um broadcast, ele compara o numero de seqüência com o recebido anteriormente (mais recente), se for maior (1 maior) nenhum broadcast foi perdido. Senão ele pede o broadcast perdido para o sequenciador.

Servidores do Amoeba

A maioria dos serviços do sistema operacional no Amoeba são implementados como processos servidores. Os servidores seguem um modelo único com o objetivo de conseguir uniformidade e simplicidade. O modelo e alguns exemplos de servidores são descritos nesta seção. Todos os servidores padrão no Amoeba são definidos por um conjunto de procedimentos stub. Servidores novos são definidos em AIL (Amoeba Definition Language). Os procedimentos stub são gerados por um compilador AIL a partir da definição dos stubs e colocados em bibliotecas para utilização por parte dos clientes.

Servidor Bullet. No Amoeba o sistema de arquivos é uma coleção de processos servidores e portanto independente do kernel. Diferentes usuários podem usar diferentes sistemas de arquivos. O sistema de arquivos padrão do Amoeba é composto de 3 servidores, o servidor bullet, o servidor de diretório e o servidor de replicação.
De uma forma geral, um processo cliente pode criar um arquivo através da chamada create e o servidor bullet responde enviando uma capacidade que pode ser usada nas chamadas subsequentes para leitura.
O servidor bullet foi projetado para ser muito rápido, rodando em máquinas com muita memória principal e grandes discos. A organização é diferente dos servidores de arquivos convencionais, uma particularidade é que os arquivos são imutáveis. O modelo conceitual neste caso é que o cliente cria um arquivo completo na sua própria memória e depois transmite (uma RPC) o mesmo para o servidor bullet que armazena e retorna uma capacidade. Este processo se repete nas modificações, que implicam na leitura do arquivo antigo e criação de um novo arquivo.
É possível também a leitura de um arquivo em partes, neste caso dois tipos de arquivos são introduzidos: uncommitted, que estão no processo de criação e podem ser modificados e committed, que são permanentes e não podem ser modificados. A opção neste caso é feita durante o create. Um arquivo uncommitted permite crescimento, mas não pode ser lido.

Interface do servidor bullet. A tabela abaixo mostra operações suportadas pelo servidor.

Chamada	Descrição
Create	Cria um novo arquivo, podendo optar por commit
Read	Ler todo ou parte de um arquivo como especificado
Size	Retorna o tamanho de um arquivo
Modify	Escreve n bytes de um uncommitted arquivo
Insert	Insere ou adiciona n bytes em um arquivo uncommitted
Delete	Deleta n byte de um arquivo uncommitted

Na operação read é fornecida a capacidade do arquivo. Além destas operações o bullet também suporta outras 3 especiais para o administrador do sistema, estas operações permitem transferir dados da cache para o disco, compactar o disco e recuperar arquivos com defeito.

Implementação. O servidor mantém uma tabela de arquivos com uma entrada para cada arquivo, similar a tabela de i-nodes do UNIX. Toda a tabela é lida para a memória quando o servidor bullet é carregado e é mantida durante a execução do servidor. De uma forma geral, cada entrada da tabela contém 2 apontadores e 1 tamanho. Os apontadores, um é o endereço em disco e o outro o endereço em memória (se o arquivo estiver na cache). Os arquivos são armazenados de forma contígua. Quando um cliente quer ler um arquivo ele envia a capacidade do mesmo para o servidor bullet, o servidor extrai o numero do objeto da capacidade e usa o mesmo como índice da tabela de arquivos, a entrada selecionada contém o numero randomico usado como *verificador* da capacidade que é então usado para verificar a validade da capacidade. Se a mesma é válida o arquivo é carregado para a memória, o espaço em cache é gerenciado usando LRU (least recently used). Se a capacidade de um arquivo é perdida ele não pode mais ser acessado, no entanto permanece no sistema. A solução adotada aqui utiliza timers para arquivos uncommitted e contadores para arquivos committed.

Servidor de diretório. A função principal deste servidor é nomeação, fornecer um mapeamento de nomes de arquivos (ASCII) para capacidades. Os processos podem criar 1 ou mais diretórios os quais podem conter várias linhas, sendo que cada linha descreve um objeto e contém o nome do objeto e sua capacidade. Entre as operações fornecidas pelo servidor estão, criação deleção de diretórios, inclusão deleção de linhas e procura de nomes em diretórios. Além de o servidor de diretório ser usado para armazenar pares <arquivo, capacidade> ele pode suportar um modelo mais geral, como por exemplo: 1) nomear qualquer tipo de objeto que é descrito por uma capacidade, não existe nenhum requisito especial que diga que um objeto tem que ser de um tipo ou todos serem gerenciados por um único servidor. Quando uma capacidade é carregada, o seu servidor é localizado po broadcast. 2) Uma linha pode conter um conjunto de capacidades, que servem para cópias do objeto e podem ser gerenciadas por diferentes servidores. 3) Cada linha pode conter vários campos, formando diferentes domínios de proteção com diferentes direitos de acesso (exemplo UNIX).
No Amoeba, cada usuário tem o seu próprio diretório raiz que contém capacidades para os subdiretórios privados do usuário assim como para diretórios públicos. Alguns diretórios da raiz são similares ao UNIX (bin, dev, etc).

Interface. As chamadas principais no caso do servidor de diretórios são mostradas na tabela a seguir.

Chamada	Descrição
Create	Criar um novo diretório
Delete	Remover um diretório ou uma entrada de um diretório
Append	Adicionar uma nova entrada em um diretório
Replace	Modificar uma entrada do diretório
Lookup	Retorna conjunto de capacidades referentes a um nome
Getmasks	Retorna os direitos de acesso de uma entrada
Chmod	Modificar direitos em uma entrada

No caso de deleção de uma entrada, não significa que o objeto está sendo destruído. Se uma capacidade é removida de um diretório o objeto continua a existir, a capacidade pode ser colocada em outro diretório. Para remover o objeto ele deve ser explicitamente destruído.

Implementação. Visto que o servidor de diretório é um componente crítico no Amoeba, ele foi implementado com características tolerantes a falhas. A estrutura de dados básica é um array de capacidades armazenada em uma partição do disco. Este array não usa o servidor bullet porque ele deve ser atualizado frequentemente e o overhead é grande. Quando um diretório é criado, o numero do objeto colocado na capacidade é um índice para o array. Os servidores de diretório normalmente existem em pares, cada um com o seu próprio array de pares de capacidades (em discos diferentes), para prevenir quando uma das partições do disco é afetada. Os servidores comunicam-se para manter sincronizados.

Servidor de replicação. Objetos gerenciados pelo servidor de diretório podem ser replicados automaticamente usando o servidor de replicação (lazy replication). Quando um arquivo ou outro objeto é criado inicialmente uma cópia é feita, então o servidor de replicação pode ser acionado para fazer cópias. O servidor executa em background examinando partes do diretório periodicamente, sempre que ele encontra uma entrada de diretório que deveria ter *n* capacidades mas tem menos, ele contata os servidores importantes e prepara cópias adicionais. O servidor de replicação também é responsável pelo mecanismo de envelhecimento (aging) e de coleta de lixo (garbage collection) usado pelo servidor bullet e outros.

Servidor de execução. Quando o usuário entra um comando no terminal, é necessário determinar: 1) em que tipo de arquitetura o processo vai executar (x86, sparc, 68xxx) e, 2) qual processador deveria ser escolhido (carga, memória). Cada servidor de execução gerencia 1 ou mais conjuntos de processadores, representados por um diretório *pooldir*, contendo subdiretórios para cada uma das arquiteturas suportadas. Os subdiretórios contém capacidades para acesso aos servidores de processo em cada uma das máquinas do conjunto.
Quando o interpretador de comandos (shell) deseja executar um programa, ele olha em */bin* para encontrar *programa*. Se *programa* está disponível para várias arquiteturas, ele não será um arquivo simples mas um diretório com um executável para cada uma das arquiteturas. O shell neste caso executa RPC com o servidor de execução enviando para ele todos descritores de processo disponíveis e pedindo para ele escolher uma arquitetura e uma CPU específica. O servidor, olha o seu *pooldir* para ver o que esta disponível, e a seleção é feita da seguinte forma: 1) interseção dos descritores de processos (binários) com conjunto de processadores, escolher candidatos; 2) verifica quais candidatos tem memória suficiente para executar o programa; 3) para os que sobraram, estima o poder de computação que pode ser gasto para o novo programa.

Servidor de boot. Este servidor é usado para fornecer um grau de tolerância a falhas para o sistema através da verificação se todos os servidores que deveriam estar executando realmente estão, tomando medidas corretivas se eles não estão. Ele tem um arquivo de configuração onde servidores que devem sobreviver a quebras são incluídos, cada entrada indica a freqüência da verificação e como ela deve ser feita. Enquanto o servidor responde corretamente o servidor de boot não realiza nenhuma ação mas, se o servidor falha na

resposta um certo numero de vezes ele é declarado como morto e, neste caso, o servidor de boot tenta recupera-lo. Este servidor pode ser replicado.

Servidor TCP/IP. Mesmo usando o protocolo FLIP internamente, o Amoeba em certas comunicações necessita conversar em TCP/IP. Para estabelecer uma conexão, o processo Amoeba faz RPC com o servidor TCP/IP para conseguir um endereço TCP/IP, o processo é bloqueado até que a conexão seja estabelecida ou recusada. Como resposta o servidor devolve uma capacidade para usar a conexão.