15.2 Mach

O projeto Mach é originário da Universidade Carnegie-Mellon e tem como base um sistema chamado RIG (Rochester Intelligent Gateway) da Universidade de Rochester em 1975. Na sua primeira versão (1986) o Mach foi desenvolvido para um VAX 11/784 com 4 processadores, sendo compatível com UNIX. Embora tivesse facilidades de rede nesta época o Mach foi concebido para uma única máquina ou multiprocessador ao invés de um sistema operacional distribuído para uma coleção de máquinas em uma LAN. Em 1988, a versão Mach 2.5 era grande e monolítica devido a quantidade de código Unix Berkeley no kernel, este código foi removido do kernel e colocado em espaço de usuário. A partir daí o que permaneceu foi um microkernel Mach 3, e um emulador BSD Unix.

Metas do Mach. As principais metas do sistema Mach são as seguintes:

1. Prover uma base para construção de outros sistemas operacionais : para suportar emulação binária do UNIX e outros SO's, o Mach permite redirecionamento de chamadas de SO para uma biblioteca de emulação que depois são enviadas para servidores de SO no nível de usuário.
2. Suporte de espaços de endereçamento esparsos e grandes.
3. Permitir acesso transparente aos recursos de rede : para suportar programas distribuídos, Mach adota um modelo de comunicação independente de localização a partir da utilização de portas. Entretanto, todo o tratamento de rede fica a cargo dos servidores de rede que ficam no nível de usuário.
4. Explorar paralelismo a nível de sistema e de aplicação: Mach foi projetado para executar em multiprocessador de forma que tanto threads do kernel quanto de usuário poderiam executar em qualquer processador.
5. Possibilitar portabilidade para grande coleção de máquinas: código dependente de máquina foi isolado tanto quanto possível. Por exemplo, código que trata de memória virtual foi dividido em parte independente e parte dependente de máquina.

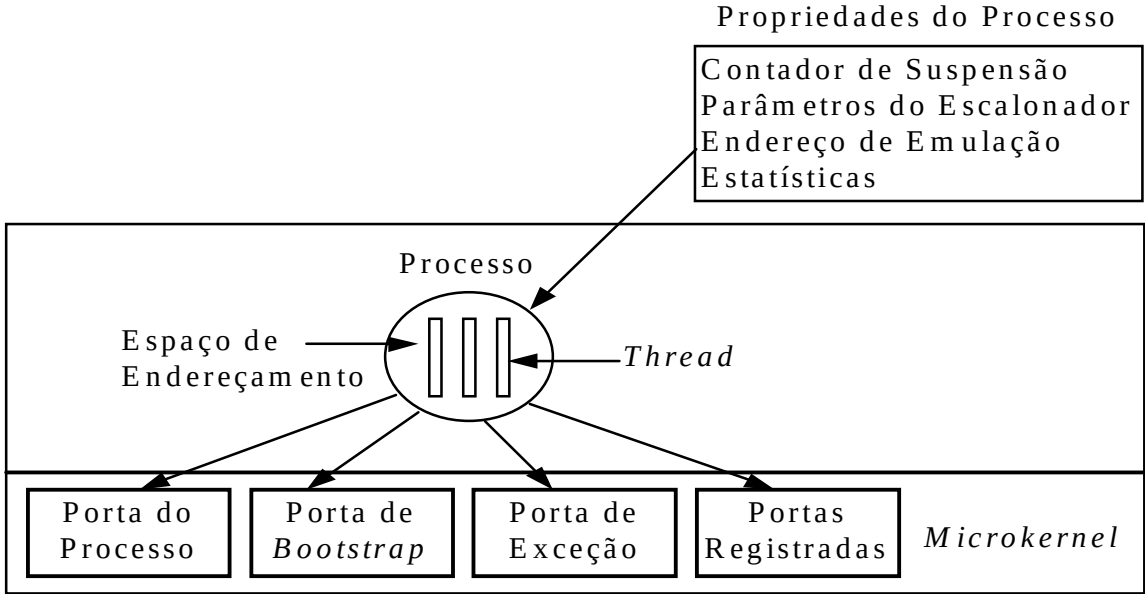
Abstrações do Mach. Como outros microkernels, o Mach kernel, é composto por serviços de gerência de processos, gerência de memória, comunicação e de E/S. Sistema de arquivos e outras funções de SO são tratados no espaço de usuário. A idéia é fornecer os mecanismos necessários para que o sistema funcione, deixando para o nível de usuário as políticas. O kernel é responsável por 5 principais abstrações:

1. Processos – Um processo Mach (task) é um ambiente de execução e é composto de um espaço de endereçamento protegido e uma coleção de capacidades, gerenciadas pelo kernel, usadas para acessar portas.
2. Threads – Processos podem conter várias threads, sendo que threads de um mesmo processo podem executar em paralelo em um multiprocessador.
3. Objetos de memória – Cada região de um espaço de endereçamento virtual de um processo Mach corresponde a um objeto de memória. Operações como busca e armazenamento de dados são parte destes objetos.
4. Portas – Uma porta em Mach é um canal de comunicação unicast e unidirecional com uma fila associada.

5. Mensagens – Uma mensagem no Mach pode conter além de dados, direitos de acesso a portas (capabilities). O kernel utiliza técnicas de gerência de memória para transferir dados de forma eficiente entre os processos (tasks).

Processos e Threads no Mach

Um processo no Mach é composto de um espaço de endereçamento, uma coleção de threads (entidades ativas) e portas. A figura a seguir ilustra um processo Mach.



As portas associadas a um processo apresentam funções especiais:

> *Porta do processo*: usada para o processo se comunicar com o kernel, requisitando serviços.
> *Porta de Bootstrap*: usada na inicialização do processo. O processo inicial utiliza esta porta para conhecer os nomes das portas do kernel que fornecem serviços essenciais.
> *Porta de exceção*: usada para processar (receber sinais) exceções (divisão por zero).
> *Portas registradas*: usadas para permitir a comunicação do processo com servidores padrões, como servidor de nomes.

Outras propriedades do processo indicam o estado do mesmo (suspenso, pronto), parâmetros de escalonamento (processadores, prioridades), endereço de emuladores e estatísticas de utilização de recursos. Diferente do UNIX um processo Mach não tem uid, gid, mascara de sinais, diretório raiz, diretório de trabalho, descritor de arquivo. Estas informações são todas gerenciadas pelo emulador do SO.

O Mach fornece um número pequeno de primitivas para gerência de processos, a maioria delas é realizada através do envio de mensagens para o kernel via porta do processo. A tabela a seguir mostra algumas delas. As primitivas são válidas para processos (task_) e threads (thread_).

Chamada	Descrição
Create	Criação de novo processo, com herança de certas propriedades
Terminate	Termina um processo especificado
Suspend	Incrementa o contador de suspensão
Resume	Decrementa o contador de suspensão, se 0 desbloqueia o processo
Priority	Atribui prioridade para threads (atual ou futuras)
Assign	Informa que processador as novas threads devem executar
Info	Retorna informação estatística
Threads	Retorna uma lista das threads do processo

Na chamada create para processo é especificado um protótipo do processo que serve como base para a criação do novo. O processo filho será uma cópia do protótipo, podendo herdar o seu espaço de endereçamento ou não. Inicialmente, o filho recebe porta do processo, porta bootstrap, porta de exceção e não tem nenhuma thread.

Threads. As threads são as entidades ativas no Mach. Cada thread pertence a um processo. Todas as threads de um processo compartilham o mesmo espaço de endereçamento e recursos. Threads podem ter recursos privados como porta da thread, usada para serviços do kernel específicos de threads. As threads no Mach são gerenciadas pelo kernel sendo que a criação e deleção de threads envolve estruturas de dados do kernel. Em um sistema com uma única CPU as threads usam compartilhamento de tempo (timesharing) enquanto que em multiprocessadores várias threads podem estar ativas ao mesmo tempo. Assim como os processos as threads podem estar prontas para execução ou bloqueadas (mecanismo similar aos processos). Uma variedade de primitivas são oferecidas, algumas delas são mostradas na tabela abaixo.

Chamada	Descrição
Fork	Cria uma nova thread executando o mesmo código do pai
Exit	Termina a thread chamadora
Join	Suspende o chamador até o término de uma thread especificada
Detach	Informa que a thread não vai esperar por nenhuma outra
Yield	Libera a CPU voluntariamente
Self	Retorna o identificador da thread

Outras chamadas permitem nomear threads, controlar o numero de threads por programa e o tamanho da pilha. Sincronização entre threads é feita através de variáveis de condição (signal, wait, broadcast), usadas para permitir que uma thread fique bloqueada em uma condição e seja acordada quando a condição acontecer, e mutexes (lock, trylock, unlock). Mutexes funcionam como semáforos binários e fornecem os meios para exclusão mútua, mas não carregam informação.

Implementação de Threads no Mach. Várias implementações de threads C estão disponíveis no Mach. A primeira fazia tudo em espaço do usuário dentro de um único processo. Esta abordagem compartilha todas as threads C em uma única thread do kernel, e pode ser usada em sistemas que não fornecem suporte a threads a nível de kernel (UNIX). As threads neste caso executam como co-rotinas, ou seja, o escalonamento é não pre-emptivo. Esta abordagem apresenta como problema o fato de que se uma thread executa uma chamada de sistema bloqueante, todo o processo bloqueia. Este problema pode ser resolvido a partir da não chamada de funções bloqueantes ou usando primitivas, como SELECT, para informar quando existe risco de bloqueio.

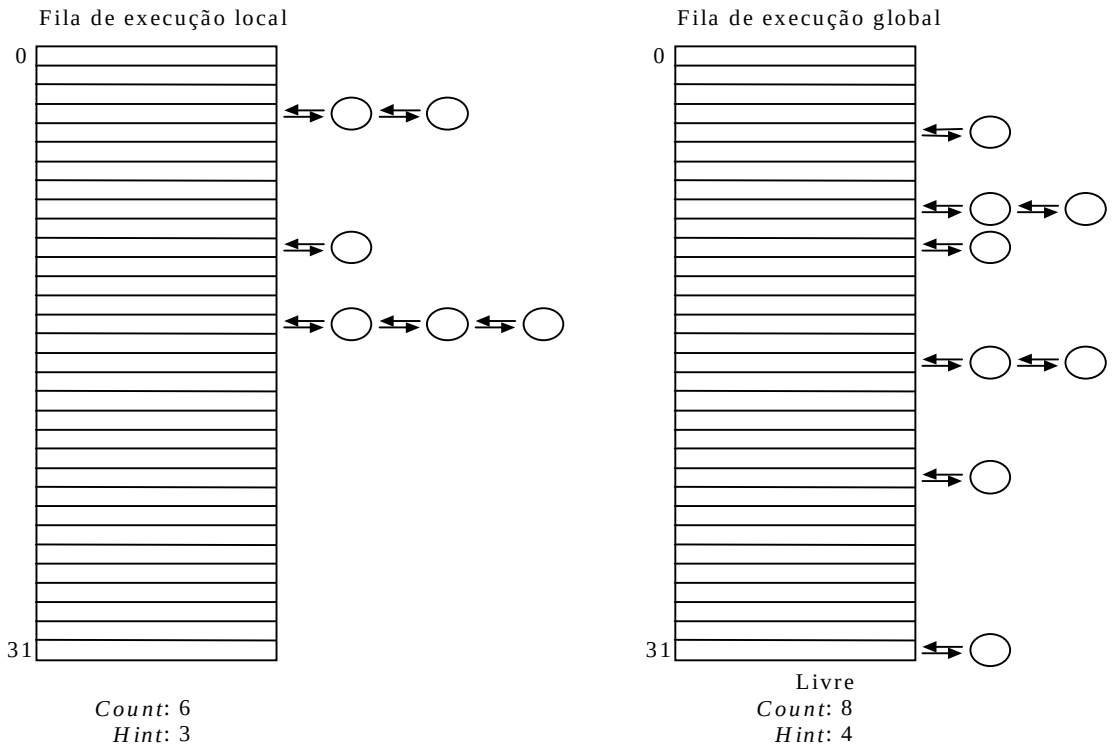
Uma segunda abordagem é usar uma thread Mach para cada thread C, neste caso as threads são escalonadas preemptivamente. Usando multiprocessador elas podem executar em paralelo. Outra abordagem utiliza uma thread Mach para cada processo. Por ultimo, é possível permitir um número de threads de usuário (arbitrário) utilizar um número de threads do kernel. Neste caso se obtém a maior flexibilidade e por isto é o mais usado.

Escalonamento. O escalonamento no Mach foi projetado em função da existência de um sistema multiprocessador. Neste caso, as CPUs de um multiprocessador podem ser atribuídas a conjuntos de processadores onde cada CPU pertence a somente 1 conjunto. As threads, po sua vez, são atribuídas a conjuntos de processador, sendo que cada conjunto de processadores tem uma coleção de CPUs a sua disposição e uma coleção de threads que precisam de processamento. O algoritmo de escalonamento deve atribuir threads a CPUs de uma maneira justa e eficiente. Os conjuntos de processadores são independentes uns dos outros em termos de recursos e usuários. Com este mecanismo é possível para um processo atribuir uma das suas threads (importante) para um conjunto com apenas uma CPU, assegurando que a mesma vai executar sempre. O mapeamento de threads em processadores pode ser feito dinamicamente.

O escalonamento no Mach é baseado em prioridades (0 a 31,127), sendo que cada thread tem 3 prioridades associadas (base, máxima, corrente). Todas as threads no Mach competem entre si pela CPU, as decisões de escalonamento não levam em conta que threads pertencem a que processos. Cada processador (conjunto) tem associado um array de com 32 filas de prontos (global) que correspondem as threads com prioridades correntes de 0 a 31. Quando uma thread com prioridade *n* torna-se executável, ela é colocada no final da fila *n*. Cada fila tem 3 variáveis associadas: um mutex usado para bloquear o sua da estrutura, um contador do numero de threads em todas as filas (valor 0 não tem nada para fazer), uma indicação de onde encontrar a thread mais prioritária.

Além disto, para cada CPU existe uma estrutura similar, que é a fila local. A fila local contém aquelas threads que estão permanentemente associadas aquela CPU (só podem rodar naquela CPU).

A figura abaixo mostra as filas de pronto global e local usadas para escalonamento de threads no Mach.



O algoritmo de escalonamento é executado quando uma thread bloqueia, termina ou não tem mais tempo para executar. Inicialmente, verifica se existem threads ativas na fila local, se a fila esta vazia verifica a fila global (exclusão mútua), se a fila esta vazia executa thread especial (idle). Quando existe uma thread na fila, ela é escalonada e executada por 1 quantum. O quantum é variável, quanto maior o número de threads ativas e menor o numero de CPUs menor o quantum. Em cada ciclo do relógio a CPU incrementa o contador de prioridade da thread corrente fazendo com que sua prioridade fique menor. Além disto uma thread pode diminuir sua prioridade e apontar seu sucessor (handoff scheduling) através de chamada de sistema.

Nomeação e Proteção

O Mach identifica recursos individuais com portas (ports). Para acessar um recurso, é necessário enviar uma mensagem para a porta correspondente. Os servidores em geral gerenciam várias portas, uma para cada recurso. Por exemplo, um servidor UNIX utiliza cerca de 2000 portas.
Proteger um recurso de acessos ilegais é comparado a proteger uma porta de envios ilegais. Para conseguir tratar com estes problemas no Mach, o kernel controla a aquisição das capacidades para as portas e também o servidor de rede controla as mensagens que chegam na rede. A capacidade para uma porta tem um campo que especifica os direitos de acesso da porta pertencentes ao processo que de posse da mesma. Os direitos de acesso das portas

podem ser : 1)direito de envio, permite enviar mensagens para a porta correspondente; 2) direito de 1 envio, permite apenas 1 envio depois anula os direitos; 3) direito de recepção, permite a recepção de mensagens da porta correspondente (pelo menos 1 processo). As comunicações no Mach são do tipo N-para-1. Os direitos de acesso a portas são conseguidos pelos (threads) processos a partir da criação de portas ou recebendo direitos enviados por mensagens (exeção bootstrap port).
Direitos de acesso a portas no Mach são armazenados e protegidos pelo kernel. Os processos referem-se a direitos de portas através de identificadores locais que são válidos apenas no espaço de nomes de porta local . Isto permite (implementadores do kernel) a escolha de representações eficientes para estas capacidades (apontador para filas de mensagem) e a escolha de nomes locais convenientes para o kernel procurar a capacidade pelo nome. Na verdade, assim como descritores de arquivos do UNIX, identificadores locais são inteiros usados para indexar uma tabela do kernel contendo as capacidades do processo.
O esquema de nomeação e proteção do Mach permite rápido acesso a filas locais de mensagens a partir de um identificador de nível de usuário. Para isto, é preciso que o kernel processe qualquer direitos transmitidos em mensagens entre processos. No mínimo, associar nomes locais e espaço nas tabelas do kernel a direitos de envio. Além disto, em um ambiente seguro, a transmissão de direitos de acesso a portas requer alguma forma de criptografia para se proteger contra formas de ataque a segurança.

Comunicação no Mach

A meta da comunicação no Mach é suportar uma variedade de estilos de comunicação de uma forma confiável e flexível. O mecanismo pode tratar passagem assíncrona de mensagens, RPC, streams, e outras formas.

Portas. A comunicação no Mach tem como elemento básico a porta, uma estrutura de dados do kernel. Uma porta é essencialmente uma caixa postal protegida. Quando uma thread em um processo deseja se comunicar com uma thread em outro processo, a thread enviante escreve a mensagem na porta e a recebedora retira a mesma. Cada porta é protegida para assegurar que somente processos autorizados podem enviar ou/e receber. Portas suportam comunicação unidirecional (UNIX pipe), uma porta usada para enviar uma mensagem de um cliente requisitando serviço para um servidor não pode ser usada para o servidor enviar de volta resposta. A comunicação é confiável e sequenciada (message streams), se uma thread envia mensagens para uma porta o sistema garante a entrega da mesma e em ordem. Entretanto, se duas threads escrevem na mesma porta, o sistema não fornece nenhuma garantia quanto ao sequenciamento das mensagens.
Quando uma porta é criada, um espaço de 64 bytes é alocado no kernel para gerenciamento da mesma. As mensagens endereçadas para a porta não são armazenadas nesta estrutura mas em outra estrutura do kernel, fila de mensagens. Informações mantidas pela estrutura de dados porta incluem: contador do numero de mensagens na fila, numero máximo de mensagens permitido, apontador para estrutura conjunto de portas (no caso da porta pertencer a um conjunto de portas), capacidades que podem usar a porta, capacidade usada

em caso de erro, lista de threads bloqueadas na porta, quem recebe da porta, a quem pertence (porta de processo, aponta para o processo; porta de thread, aponta para thread).
A criação de uma porta retorna um inteiro que identifica a porta (fd UNIX) e é usado para as chamadas subsequentes. Na comunicação entre dois processos A e B, cada um tem acesso a mesma porta. A envia a mensagem para a porta enquanto B retira a mensagem da porta. Neste caso, a mensagem é copiada de A para a porta e da porta para B.
As portas podem ser agrupadas formando conjunto de portas, sendo que é possível ler de um conjunto de portas (servidor lendo de várias portas), o kernel retorna uma mensagem de uma das portas do conjunto.

Capacidades. Para cada processo, o kernel mantém (no seu espaço de endereço) uma tabela das portas que o processo pode acessar. As entradas desta tabela são capacidades (lista de capacidades). Cada vez que uma thread cria uma porta o kernel associa uma capacidade na tabela e retorna o seu índice como nome da capacidade. Cada capacidade, além do apontador para a porta, contém direitos de acesso (receive, send e send-once) do dono da mesma. Nomes de capacidades tem significado somente no contexto de um processo (dois processos com acesso a mesma porta usam capacidades com nomes diferentes). Quando um processo termina ou é morto, a lista de capacidades dele é removida, as portas com capacidade do tipo receive são destruídas. O kernel associa um contador a cada porta de forma que quando uma capacidade é removida o contador é decrementado e somente quando zero a capacidade é realmente removida. Cada entrada da tabela de capacidade pode conter: 1) uma capacidade para uma porta, 2)uma capacidade para um conjunto de portas, 3) uma entrada nula, 4) um código indicando que a porta associada esta desativada (porta receive morta pelo término do processo).

Primitivas de gerência de portas. A gerência de portas dispõe de cerca de 20 primitivas que são invocadas através do envio de mensagens para a porta do processo. Algumas delas são listadas na tabela abaixo.

Chamada	Descrição
Allocate	Cria uma porta e insere sua capacidade na lista de capacidades
Destroy	Destroi uma porta e remove sua capacidade da lista
Deallocate	Remove uma capacidade da lista
Extract_right	Extrai a n-ésima capacidade de outro processo
Insert_right	Insere uma capacidade na lista de outro processo
Move_member	Move uma capacidade para/de um conjunto de capacidades
Set_qlimit	Numero de mensagens uma porta pode manter

No caso das primitivas que permitem extração e inserção de capacidades é necessário que o processo chamador possa acessar a porta do processo do processo destino. A primitiva Move_member permite a gerência dos conjuntos de portas a partir da inclusão/remoção de portas dos conjuntos.

Mensagens. No Mach existe uma única chamada responsável pelo envio e recepção de mensagens, *mach_msg*, com vários parâmetros e um grande numero de opções. A chamada é bastante complexa, existem 35 diferentes códigos de erro que podem ser retornados pela chamada. É possível, através da chamada, enviar mensagem e retornar imediatamente para o enviante, receber mensagem e bloquear se a mesma não esta disponível ou desistir, ou combinar estas duas alternativas enviando mensagem e bloqueando a espera de resposta (RPC). Uma chamada típica no Mach é :

Mach_msg(&hdr, options, send_size, rcv_size, rcv_port, timeout, notify_port);

hdr = ponteiro para msg (in ou out), msg tem um cabeçalho fixo e um corpo
options = bit para envio, bit para recepção (2 on RPC), bit timeout
send_size = tamanho da mensagem sendo enviada
rcv_size = tamanho da mensagem a receber
rcv_port = para recebimento, nome da capacidade da porta a receber
timeout = em milisegundos
notify_port = porta onde relatar status

Uma mensagem Mach pode ser simples ou complexa, a diferença é que mensagens simples não podem levar capacidades ou ponteiros protegidos. Outra informação do cabeçalho é o tamanho da mensagem que combina o cabeçalho e o corpo. Também no cabeçalho informação sobre nomes de capacidades são incluídas: especificando a porta destino e uma porta de recebimento de resposta (capacidades são índices da tabela do enviante). Numa mensagem RPC, o destino designa o servidor e a resposta indica ao servidor para qual porta enviar o resultado. O cabeçalho também inclui informação sobre o tipo da mensagem e o código da função, que não são examinados pelo kernel.
Mensagens enviadas e recebidas com sucesso são copiadas para o espaço de endereço do destino. Entretanto, se a porta destino está cheia o procedimento adotado vai depender das opções e direitos de acesso da porta: o enviante pode ser bloqueado, o enviante pode receber timeout.

Mensagens complexas no Mach são uma seqüência de pares <descritor, item de dados>, onde cada descritor < tipo, tamanho, quantidade (quantos itens do tipo)> indica o que esta no item de dados a seguir. Um campo de dados pode conter: bits, bytes, inteiros, ponto flutuante, booleanos, strings e capacidades. É possível através de mensagens complexas transferir capacidades de um processo para outro. O descritor deve especificar quando a capacidade é copiada ou movida. Valores no campo tipo de dados indicam para o kernel modificar os direitos de acesso da capacidade transmitida. Uma capacidade RECEIVE pode mudar para SEND ou SEND-ONCE de forma que o recebedor poderá enviar resposta para ela. O Mach prove uma forma de transferir dados entre processos sem fazer cópia. Um bit out-of-line é usado para fazer com que a palavra seguinte ao descritor contenha um endereço e os campos tamanho e quantidade contém um contador de 20-bits. Juntos eles especificam uma região do espaço de endereçamento virtual do enviante. No recebedor o kernel aloca espaço igual e mapeia as páginas do enviante no espaço do recebedor tornando-as copy-on-write. O endereço enviado é então modificado para o novo. Isto permite a transferência de dados de uma forma rápida. Entretanto, este método não é útil em comunicação envolvendo máquinas diferentes.

O Servidor de Rede. Quando a comunicação no Mach é realizada entre máquinas de uma rede, servidores no nível de usuário são responsáveis pelo seu gerenciamento. Toda máquina em um sistema Mach distribuído deve executar um servidor de rede.
O servidor é um processo multithread que realiza uma variedade de funções, incluindo: interface com threads locais, passagem de mensagens (forwarding), tradução de tipos de dados de máquina para máquina, gerência de capacidades, serviço de nome e tratamento de autenticação. Numa comunicação cliente/servidor, cliente na máquina A e servidor na máquina B, o método básico é o seguinte. Antes de o cliente comunicar com o servidor é necessário a criação de uma porta em A para funcionar como proxy para o servidor. O servidor de rede tem capacidade RECEIVE para esta porta. Os passos necessários na comunicação seguem:

1. cliente envia uma mensagem para a porta do servidor (proxy)
2. o servidor de rede pega a mensagem
3. servidor de rede verifica mapeamento da porta local em porta de rede, transforma em endereço de rede e constrói uma mensagem de rede contendo a mensagem a ser enviada
4. o servidor de rede remoto pega a mensagem verifica a porta e mapeia para porta local e escreve a mensagem para esta porta
5. o servidor lê a mensagem da porta local e executa a requisição.

Esta solução para tratamento de comunicação em rede oferece uma grande flexibilidade. Entretanto, o preço disto é uma redução significativa em termos de performance.

Gerência de Memória no Mach

O sistema de gerência de memória do Mach é bastante flexível, elaborado e baseado em paginação. Particularmente, ele o sistema de memória em parte dependente de máquina e independente de máquina de uma forma bastante clara. Esta separação torna o sistema mais portável. Um dos aspectos que torna o sistema de memória do Mach bastante particular é o fato de ele dividir o código em 3 partes: 1) o módulo *pmap*, executa no kernel e é responsável pelo gerenciamento da MMU (Memory management unit), inicializa registradores, tabela de páginas e captura page faults (código dependente da arquitetura da MMU). 2) a parte independente de máquina, executa no kernel, é responsável pelo processamento de page faults, gerência de mapas de endereço e troca de páginas. 3) o gerenciador de memória ou paginador externo, executa no nível de usuário, trata da parte lógica do sistema de memória, principalmente gerência de disco (memória secundária). Informações como: paginas virtuais em uso, paginas virtuais na memória, onde as páginas são mantidas em disco (quando não em memória), são mantidas pelo paginador externo. O kernel e o gerente de memória comunicam-se através de um protocolo bem definido possibilitando aos usuários a escrita de seus próprios gerenciadores.

Memória virtual. No modelo conceitual de memória no Mach o usuário visualiza um espaço de endereçamento virtual grande e linear. Máquinas com 32 bits o espaço de endereço do usuário vai de 0 a $2^{31} - 1$, suportado com paginação. Para suportar endereços

esparsos, o Mach prove uma forma de alocação e liberação de seções do espaço de endereçamento virtual chamadas *regiões*. A chamada para alocação especifica um endereço base e um tamanho para alocação da região ou simplesmente especifica um tamanho. Um conceito fundamental relacionado ao uso do espaço de endereçamento virtual é *objeto de memória*. Um objeto de memória pode ser uma página ou um conjunto de páginas assim como um arquivo ou outras estruturas especializadas e pode ser mapeado em uma parte não utilizada do espaço de endereço virtual, formando uma nova região. A manipulação de espaço de endereçamento virtual no Mach é suportada por um conjunto de chamadas de sistema (mensagens para a porta do processo), como mostra a tabela abaixo.

Chamada	Descrição
Allocate	Torna uma região do EEV usável
Deallocate	Invalida uma região do EEV
Map	Mapeia um OM em um EEV
Copy	Faz cópia de uma região em outro endereço virtual
Inherit	Indica o atributo de herança para uma região
Read	Ler dados de outro processo (EEV)
Write	Escrever dados em outro processo (EEV)

Compartilhamento de memória. Compartilhamento tem um papel importante no Mach, no caso das threads de um processo nenhum mecanismo especial é necessário, ou seja, elas visualizam o mesmo EE automaticamente. No caso do compartilhamento entre processos, importante em casos como: produtor/consumidor, multiprocessadores, criação de processos (UNIX), o Mach permite que processos indiquem um atributo de herança para cada região no seu EE. Este atributo pode assumir diferentes significados: 1) a região não é usada pelo processo filho, 2) a região é compartilhada entre o processo pai (protótipo) e o filho e, 3) a região no processo filho é uma cópia do pai. Esta ultima opção é usada na chamada *fork* para criação de processos. Entretanto, o método utilizado é copy-on-write.

Gerentes externos. Cada objeto de memória que é mapeado no espaço de endereçamento de um processo deve ser controlado por um gerenciador externo. Podem existir diferentes classes de objetos de memória sendo controlados por diferentes gerenciadores, cada um implementando sua própria semântica poderá determinar onde armazenar paginas que não estão na memória.
No mapeamento de um objeto em um espaço de endereçamento de um processo, o processo envia uma mensagem para o gerenciador requisitando o serviço. Este procedimento envolve 3 portas: 1) a porta de objeto, criada pelo gerenciador e mais tarde usada pelo kernel para informar ao gerenciador sobre page faults e outros eventos relacionados ao objeto, 2) a porta de controle, criada pelo kernel para que o gerenciador possa responder aos eventos informados (portas unidirecionais) e, 3) a porta de nomes, usada para identificar o objeto (uma thread requisitando a porta associada a um endereço virtual recebera como resposta um apontador para a porta de nome da região).
Quando o gerente externo (GE) faz um mapeamento de um objeto, ele fornece a capacidade para o objeto como um de seus parâmetros. Em seguida, o kernel cria as outras 2 portas e envia uma mensagem inicial para a porta do objeto informando-o sobre o porta de controle

e de nome. O GE então envia uma resposta ao kernel informando os atributos do objeto e quando manter ou não o objeto na cache depois do desmapeamento. Todas as páginas de um objeto são inicialmente marcadas como sendo não-leitura/não-escrita, de forma a forçar um trap quando usada pela primeira vez. A partir daí, o GE executa leitura na porta do objeto e bloqueia, ficando assim até o kernel requisitar através da escrita de uma mensagem na porta do objeto, liberando para execução a thread que mapeou o objeto. A thread, durante sua execução, tenta uma operação de escrita/leitura em uma pagina pertencente ao objeto mapeado, causando um page fault e um trap para o kernel. O kernel, envia uma mensagem (assíncrona) para o GE (porta do objeto) informando a página referenciada e requisitando que a mesma seja providenciada. A thread que gerou a page fault é bloqueada. O procedimento do GE é, verificar se a referência é legal e no caso positivo, buscar a mesma para o seu espaço de endereçamento e enviar para o kernel o apontador para a página. O kernel (K) então mapeia a página no espaço de endereçamento da thread e libera a mesma. Se a referência é ilegal ele manda uma mensagem de erro para o kernel. Para manter frames livres, o kernel possui uma thread que de tempos em tempos libera frames usando o algoritmo segunda chance. É responsabilidade GE escrever a página retirada de volta para o disco. É possível marcar páginas como sendo permanentes, possibilitando a permanência das mesmas na memória.
Para se ter uma idéia de como um GE funciona, um protocolo deve ser usado para comunicação entre o K e o GE. A tabela a seguir mostra as mensagens que são trocadas entre os dois.

Mensagens do kernel para o GE

Chamada	Descrição
Init	Inicializa mapeamento de um novo objeto de memória
Data_request	Requisita uma página específica para tratar um page fault
Data_write	Kernel informa a escrita de uma página para disco
Data_unlock	Retirar bloqueio de pagina
Lock_completed	Lock_request anterior completado
Terminate	Objeto não esta mais em uso

Mensagens do GE para o kernel

Chamada	Descrição
Set_attributes	Resposta ao Init
Data_provided	Pagina requisitada
Data_unavailable	Pagina não esta disponível
Lock_request	Requisita lock, flush ou clean paginas
Destroy	Retira um objeto não mais necessário

Os seguintes eventos são causadores da troca de mensagens entre o kernel e o GE:

1. Quando um objeto é mapeado através da chamada Map o kernel envia a mensagem Init para o GE e pode receber os atributos do objeto como resposta (set_attributes) ou uma mensagem de erro.
2. Quando o processo causa um page fault, o kernel requisita a pagina para o GE através da mensagem Data_request e o GE responde fornecendo a mesma (Data_provided) ou informando que a mesma não está disponível (Data_unavailable).
3. O kernel informa a escrita de uma página no disco (Data_write).
4. O GE pede ao kernel escrever uma página ou mudar permissões de acesso com a mensagem Lock_request e o kernel responde com Lock_completed.
5. O processo não consegue acessar uma página porque ela esta lock, o kernel informa isto para o GE (Data_unlock) que retira o lock (Lock_request).
6. O kernel informa o GE que o objeto de memória da mensagem (Terminate) não mais existe.
7. Um objeto de memória não é mais necessário, GE envia mensagem Destroy e recebe como resposta a mensagem Terminate.

Emulação do UNIX no Mach

Provavelmente o mais importante dos servidores que executam no Mach é um programa que contém uma grande quantidade de código do UNIX Berkeley. Este servidor é o emulador do UNIX. A emulação do UNIX no Mach consiste de duas partes: o servidor UNIX e a emulação de uma biblioteca de chamadas de sistema. Quando o sistema é inicializado, o servidor UNIX instrui o kernel para capturar todas as chamadas de sistema (traps) e relacionar as mesmas com endereços da biblioteca de emulação do processo UNIX realizando as chamadas. A partir daí, qualquer chamada de sistema feita por um processo UNIX resultará na passagem de controle temporária para o kernel e em seguida para a biblioteca de emulação do processo. Enquanto sob controle da biblioteca, todos os registradores da máquina tem os valores de quando o trap aconteceu (mecanismo de trampolim), os registradores são examinados para determinar qual chamada foi invocada e é feito um RPC para outro processo, o servidor UNIX, que executará a chamada e depois passará o controle para o programa de usuário novamente (sem interferência do kernel).
Quando o processo *init* cria filhos, estes herdam a biblioteca de emulação e o mecanismo de trampolim. O servidor é uma coleção de threads C responsáveis pelo tratamento de timers, comunicação em rede e outros dispositivos, e principalmente tratar as chamadas de sistema que são emitidas pelos processos UNIX. A comunicação utiliza os mecanismos do Mach. De uma forma geral, quando uma mensagem chega no servidor uma thread recebe a mesma, determina de qual processo ela vem, identifica a chamada de sistema e parâmetros, executa a mesma e envia resposta.
No caso de chamadas de E/S, o procedimento é diferente. Quando um arquivo é aberto, ele é mapeado no espaço de endereçamento do chamador (facilitar o acesso da biblioteca sem precisar RPC). Uma chamada READ é satisfeita pela biblioteca de emulação da seguinte forma: localiza os bytes a serem lidos no arquivo mapeado, localiza o buffer do usuário e copia de um para outro. Se acontecer page faults durante a copia, o gerente externo vai ser notificado e neste caso ele é uma thread dentro do servidor UNIX (i-node pager). O trabalho desta thread é buscar as páginas no disco, permitir o mapeamento das mesmas no

espaço de endereçamento do programa de usuário e também sincronizar operações em arquivos que estão abertos por vários processos simultaneamente.