Universidade do Contestado – UNC
Unidade Universitária de Mafra
Otávio Rodolfo Piske

Sistema Operacional Mach

MAFRA
Otávio Rodolfo Piske

Introdução

O sistema operacional Mach surgiu, como um projeto da Universidade de Rochester em 1975 e se chamava RIG (Rochester Intelligent Gateway). O RIG foi escrito para uma máquina de 16 bits da Data General, o Eclipse. O objetivo principal do projeto era demonstrar que sistemas operacionais poderiam ser construídos de uma forma modular, como uma coleção de processos se comunicando com troca de mensagens.

Quando um dos engenheiros que participaram do projeto, Richard Rashid, mudou-se para a universidade de Carnegie-Mellon, em 1979, ele continuou com o projeto de desenvolver tais sistemas operacionais, porém em hardware mais avançado. A máquina escolhida foi o PERQ, uma estação de trabalho com tela gráfica, mouse e conexão de rede. Esse novo sistema operacional foi chamado de Accent e adicionava proteção, a capacidade de operar transparentemente sobre a rede, memória virtual de 32 bits e outros recursos.

Uma versão inicial do sistema ficou pronta em 1981, mas já em 1984, uma média de 150 PERQs rodavam Accent. Entretanto o sistema UNIX estava ganhando espaço gradativamente. Esse fato levou Rashid a iniciar uma nova versão do seu sistema operacional chamado de Mach.

O sistema Mach era compatível como o UNIX e, portanto, a maioria do software poderia ser aproveitado. Além disso, o sistema Mach possuía alguns recursos a mais que o UNIX, tais como threads e suporte a multiprocessadores.

O projeto do Mach foi bastante impulsionado pela agência de defesa americana, DARPA. A primeira versão do Mach ficou pronta em 1986 e rodava em um computador VAX 11/784 com 4 CPUs. Pouco tempo depois, ele foi portado para IBM PC/RT e Sun 3. Por volta de 1987, o Mach rodava em máquinas Encore e Sequent (ambas multiprocessadas).

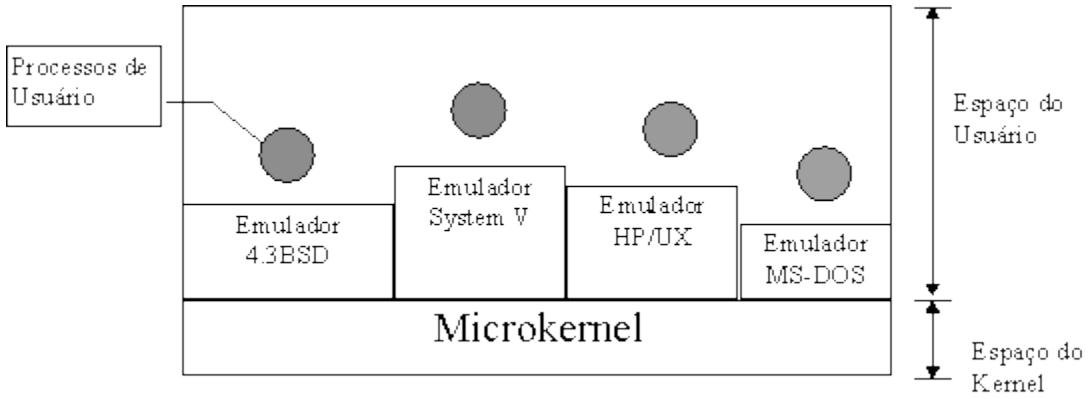
Logo em seguida, IBM, DEC e HP formaram a Open Software Fundation (OSF), que tinha como onjetivo tomar o controle do UNIX. A AT&T, criadora desse sistema, havia aliado-se com a Sun Microsystems para desenvolver o System V Release 4, e isso despertou o medo nos demais fabricantes. Finalmente, a OSF escolheu o Mach 2.5 como base para o seu sistema operacional, o OSF/1. Apesar de ambos sistemas conterem muitas linhas de código de Berkeley e até mesmo da AT&T, o objetivo da OSF era, ao menos, controlar a direção para a qual o UNIX estava indo.

Em 1988, o kernel do Mach 2.5 era grande e monolítico. Assim, nessa mesma época, a CMU retirou todo código de Berkeley e da AT&T de dentro do kernel e colocou-o no espaço do usuário. O que sobrou foi um sistema microkernel.

O sistema Mach

O sistema Mach foi construído de forma que outros sistema operacionais como UNIX ou DOS pudessem ser emulados em cima. Um dos objetivos também era fazer com que ele suportasse multiprocessadores e ainda assim fosse portável. A emulação de outros sistemas operacional é realizada através de uma arquitetura de camadas. Assim, o emulador funciona como uma camada fora do kernel e roda independentemente das outras aplicações. Outro fato que deve ser notado é que vários servidores podem estar

rodando simultaneamente na mesma máquina, o que permite que o usuário rode programas 4.3BSD, System V e MS-DOS ao mesmo tempo.



Assim como outros sistemas microkernel, o sistema Mach apenas provê serviços básicos como gerenciamento de processos e memória, comunicação e serviços de I/O. Outros serviços que antes residiam no espaço do kernel, foram transformados em processos de usuário, tais como o sistema de arquivos e diretórios.

São5 os recursos gerenciados pelo microkernel:

- processos
- threads
- objetos de memória
- portas
- mensagens

A estrutura básica no sistema Mach é o processo. Cada processo é um ambiente onde a execução acontece e possui um espaço de memória a ele associado para guardar dados e código (text) e uma ou mais pilhas. Da mesma forma, um canal de comunicação (ou porta) é alocada a um processo. Uma thread é uma entidade de execução e a ela estão associadas um contador de programa e um conjunto de registradores. Além disso, cada thread é associada e um processo, mas um processo pode conter várias threads. Um processo com uma única thread no Mach pode ser comparado a um processo comum do UNIX.

Assim, um processo no Mach consiste, basicamente, de um espaço de endereçamento e uma coleção de threads que executam nesse espaço. A idéia de processos como uma entidade ativa não existe no Mach. A execução (ou atividade) é exclusiva das threads. Assim, processos são usados apenas para alocação de recursos relacionados a um conjunto de threads.

Além do conjunto de threads e do espaço de endereçamento, aos processos também são associadas algumas portas de comunicação. Algumas portas são especiais e todos os processos as possuem:

- **process port**: é usada para fazer a comunicação do kernel com o processo. Os serviços do kernel são requisitados por essa porta apenas mandando uma mensagem a ela. Isso reduz o número de "system calls" a uma quantidade bastante pequena.
- **bootstrap port**: realiza a inicialização do processo quando esse é carregado. Através dessa porta o processo fica sabendo o nome da porta do sistema, e também se comunicar com o seu servidor (no caso de processos UNIX emulados).
- **exception port**: é usada para reportar exceções causadas pelo processo. Exceções típicas são divisão por zero e execução ilegal de instruções. Programas de depuração também usam essa porta para realizar seu trabalho.
- **registered ports**: são portas normalmente associadas a processos padrão do sistema, tal como o servidor de nomes, por exemplo.

O processo ainda possui a propriedade de estar habilitado a executar ou bloqueado. Essa última opção faz com que suas threads fiquem bloqueadas independentemente de seus estados individuais. Pode-se ainda escolher em qual processador uma thread irá executar e ainda qual sua prioridade.

No caso de processos emulados, especifica-se em que região do espaço de endereçamento encontram-se os tratadores de chamadas de função. Assim, os processos filhos dos emuladores herdam essas características e todas suas chamadas de sistema são direcionadas ao emulador e não ao kernel do Mach.

Além disso, cada processo possui registradores estatísticos que contabilizam, por exemplo, a quantidade de memória consumida e o tempo de execução das threads. Um processo que esteja interessado nessas informações pode obtê-las enviando uma mensagem à porta de processo.

Threads

Como mencionado anteriormente, uma thread pertence a um único processo, entretanto, um processo pode conter várias threads. Dessa forma, um processo é uma entidade passiva até que contenha, ao menos, uma thread.

Todas as threads de um processo compartilham seu espaço de endereçamento e os recursos a esse alocados. Mas elas podem também ter recursos privados. Um desses recursos é uma porta de comunicação chamada "thread port", que é análoga à porta de processo, pois serve como ponto de acesso da thread às funções do sistema.

Quem gerência as threads é o kernel do Mach, portanto estruturas especiais de controle devem estar contidas dentro do mesmo. Em um sistema com uma única CPU, a execução das threads é "timeshared", entretanto, em sistemas multiprocessados, várias threads podem estar executando ao mesmo tempo. Assim, o sistema deve prover mecanismos de sincronização e exclusão mutua necessários em sistemas com essas características.

Escalonamento

O sistema de escalonamento do Mach foi bastante influenciado pela sua característica de rodar em sistemas multiprocessados. Assim, pode-se assumir um sistema uniprocessado como um caso especial desse último, onde o numero de CPUs é igual a 1.

As CPUs em um multiprocessador podem ser associadas a um "conjunto de processadores" por software. Assim, cada CPU pertence a um único conjunto. Da mesma forma, threads também podem ser associadas a grupos de processadores por software.

Assim, cada conjunto de processadores possui um conjunto de CPUs e outro de threads necessitando processamento. Logo, o trabalho do escalonador é atribuir threads a CPUs de uma maneira eficiente. Essa estrutura também dá grande poder aos processos, que podem atribuir threads às CPUs de forma a distribuir o trabalho entre todos os processadores.

O escalonamento de threads no Mach é baseado em prioridades que são valores inteiros e variam de 0 a 31 ou 127, com 0 sendo a mais alta prioridade. Existem três prioridades associadas a cada thread:

- **base priority**: que é a prioridade que a thread pode ajustar por si só dentro de certos limites.
- **lowest priority**: que é a menor prioridade que a thread pode assumir.
- **current priority**: é usada (e ajustada pelo escalonador) em função do tempo de execução da thread e é usada para definir o escalonamento da thread.

Gerenciamento de memória

O sistema de gerenciamento de memória do Mach é baseado em páginas e também é dividido em 3 partes:

- **pmap**: esse módulo realiza as tarefas de mais baixo nível do gerenciamento de páginas e se comunica diretamente com o hardware (o MMU). Esse módulo deve ser rescrito a cada vez que o sistema á portado para uma outra máquina.
- **código independente de máquina**: é a parte do kernel que trata os sinais de page-fault, gerência o espaço de endereçamento e troca as páginas.
- **memory manager**: é um processo que roda no espaço do usuário e gerência a memória de forma a manter as informações de páginas que estão carregadas e o lugar no disco onde encontram-se as páginas fora da memória.

Essa arquitetura permite que o kernel fique ainda menor e mais simples, visto que muitas funções foram transformadas em processos do usuário. Outra vantagem nesse tipo de sistema é que o usuário pode escrever seu próprio gerenciador de memória, uma vez que o protocolo que funciona entre o kernel e o memory manager é conhecido. Por outro lado, o kernel teve que adquirir funções para se proteger de gerenciadores de memória maliciosos ou até mesmo mal feitos.

Memória Virtual

Conceitualmente, o sistema Mach permite ao usuário ver uma grande memória linear virtual, gerenciando-a a partir de páginas. Entretanto, os programas podem usar essa memória de forma muito esparsa, aumentando assim a quantidade de informações que deveriam ser guardadas no kernel para gerenciar todas essas páginas.

Para isso, o sistema Mach utiliza o conceito de regiões que determinam um endereço base e um tamanho do espaço de memória usado.

Um conceito bastante interessante nesse tópico é o de objeto de memória, que pode ser uma página ou um conjunto de páginas, ou ainda um arquivo mapeado. Um objeto de memória deve ser mapeado para uma região de memória virtual não utilizada. Arquivos mapeados dessa forma podem ser acessados com as instruções normais de leitura e escrita na memória. Quando um processo termina, seus arquivos mapeados aparecem no disco.

Compartilhamento de memória

O compartilhamento de recursos entre threads no sistema Mach é automático e não precisa ser programado. Dessa forma, se uma thread quer acessar o espaço de endereçamento de outra, basta fazê-lo, pois é o mesmo que o seu. Entretanto, esse compartilhamento não ocorre entre processos, principalmente quando esses estão localizados em processadores diferentes.

Assim, o Mach fornece três tipos de compartilhamento de memória entre processos pais e filhos:

- a região não pode ser usada pelo processo filho: indica que o processo pai não quer compartilhar aquela região e qualquer acesso a ela será tratado como um acesso a uma área não alocada
- a região é compartilhada: o compartilhamento acontece normalmente, e qualquer modificação em uma cópia acarreta na atualização da outra.
- a região no filho é uma cópia da do pai: a região é copiada e qualquer modificação será feita somente na cópia local. Essa opção é implementada utilizando-se um artifício de copiar-na-escrita, ou seja, a cópia física somente é realizada se houver alguma modificação na cópia. Isso economiza preciosos ciclos de CPU no caso de regiões que nunca são acessadas.

Gerenciadores de Memória Externos

No Mach, é possível criar gerenciadores de memória customizados e que funcionam no lugar do gerenciador de memória padrão do sistema. Esse sistema dá maior flexibilidade ao Mach.

Cada objeto de memória possui um gerenciador próprio. Assim, é possível implementar políticas diferentes para cada tipo de objeto diferenciando, por exemplo, o lugar onde as páginas de memória serão armazenadas e as regras de o que será feito com o objeto após ter sido usado.

Para isso, o gerenciador de memória necessita de 3 portas:

- **object port**: é criada pelo gerenciador de memória e é usada pelo kernel para informar ao gerenciador eventos como page fault e outros.
- **control port**: é criada pelo kernel para que o processo possa responder ao kernel em função desses eventos.
- **name port**: é usada como um tipo de nome que identifica o objeto.

Quando o gerenciador de memória mapeia um objeto, ele envia uma mensagem ao kernel contendo algumas as informações do objeto. Em seguida, o kernel responde com a criação das outras duas portas (control e name ports) e notifica essa ação ao processo. Esse último responde com uma mensagem de resposta contendo os demais atributos do objeto. Inicialmente, todas as páginas de um objeto mapeado estão marcadas como unreadable/unwritable para forçar um evento no primeiro acesso.

Memória Compartilhada Distribuída

O sistema de gerenciamento de memória do Mach permite perfeitamente que se implemente um esquema de memória compartilhada distribuída sem que o kernel precise ser alterado. A idéia é ter um espaço de endereçamento global e linear que é compartilhado por várias máquinas diferentes. Quando uma thread faz um acesso a uma página que não está carregada, é gerado um page fault. Quem trata esse page fault é o gerenciador de memória, que deve prover a página ao kernel. Entretanto, não há restrições quanto à maneira como essa tarefa é realizada.

Como o Mach possui diferentes gerenciadores de memória para diferentes tipos de objetos, é natural que se crie um novo tipo para atender a essa classe de aplicações. Pode-se ainda criar gerenciadores de memória com características diferentes. Por exemplo, um garante sempre a consistência dos dados, e outro garante somente que os dados são os mais recentes dentro de um intervalo de alguns segundos. Claramente o primeiro é bastante mais complicado de se implementar, mas alguns algoritmos podem não necessitar de tanta qualidade de serviço. Dessa forma, pode-se economizar processamento utilizando-se uma política mais flexível.

Pode-se ainda ter um processo gerenciador único de memória em todo cluster, pois o sistema de mensagens do Mach é transparente em relação à rede. Isso facilita bastante o trabalho no caso de se ter um gerenciador de memória central.

Comunicação no Mach

O objetivo do sistema de comunicação do Mach é suportar um variedade bastante grande de estilos de comunicação de uma maneira precisa e confiável. O Mach suporta mensagens assíncronas, RPC, "byte streams" e outras formas de comunicação. O sistema de comunicação entre processos é ainda baseado no RIG e Accent. Apesar disso, ele foi otimizado para o caso local e não o remoto.

A base do sistema de comunicação do Mach é uma estrutura de dados do kernel chamada porta. Quando uma thread de um processo quer se comunicar com outra de outro processo, ela envia uma mensagem para sua porta e, para receber a mensagem, a outra

retira-a da porta. Esse mecanismo suporta apenas portas unidirecionais, como pipes do UNIX.

O sistema de portas é seguro e confiável, pois, se uma thread envia uma mensagem por uma porta, o kernel garante que essa chegará ao seu destino. Entretanto, o sistema não garante nada em relação ao sequenciamento dessas mensagens.

Quando uma thread cria uma porta, ela recebe um inteiro que identifica aquela porta. Esse número é usado em acessos subsequentes àquela porta e é atribuído ao processo (pelo kernel) e não à thread.

Portas podem ser agrupadas em conjuntos de portas (até mesmo a mais de um). Isso facilita o trabalho de enviar uma mensagem a várias portas ou para ler das mesmas.

Enviando e Recebendo Mensagens

As mensagens no Mach possuem uma estrutura bem definida (cabeçalho e corpo) e existe uma única chamada de sistema para enviar e receber mensagens. Essa chamada de sistema é mascarada pela função mach_msg.

Essa função recebe sete parâmetros e é usada tanto para o recebimento quanto para o envio de mensagens. Se for realizada uma leitura em uma porta que não possui mensagens, a thread é bloqueada até que alguma outra mande uma mensagem para aquela porta.

Basicamente, o que o cabeçalho da mensagem contém é a porta de destino e resposta, o tipo da mensagem, tamanho e indicações de permissões (de resposta e do destinatário).

No corpo da mensagem, pode-se especificar uma estrutura de dados bastante simples, onde um campo (especificando tipo de dado) é seguido pelo seu valor. Da mesma forma, pode-se enviar uma grande quantidade de memória sem que seja feita uma cópia de uma thread para outra; existe uma estrutura especial que permite mapear uma região do espaço de endereçamento no espaço de outra thread (somente em threads que habitam a mesma máquina). Essa região fica marcada para ser copiada somente se houver uma escrita.

Para fazer esse tipo de comunicação através da rede, existe um processo chamado "network message server". Esse processo roda em cada nodo da rede e faz o mapeamento das portas locais de um nodo nas de outro. Além disso, o NMS realiza também as seguintes tarefas: transformar tipos de dados de uma representação para outra, gerenciar portas de uma maneira segura, fazer notificações remotas, prover um serviço de procura remota e gerenciar autenticação de outras máquinas.

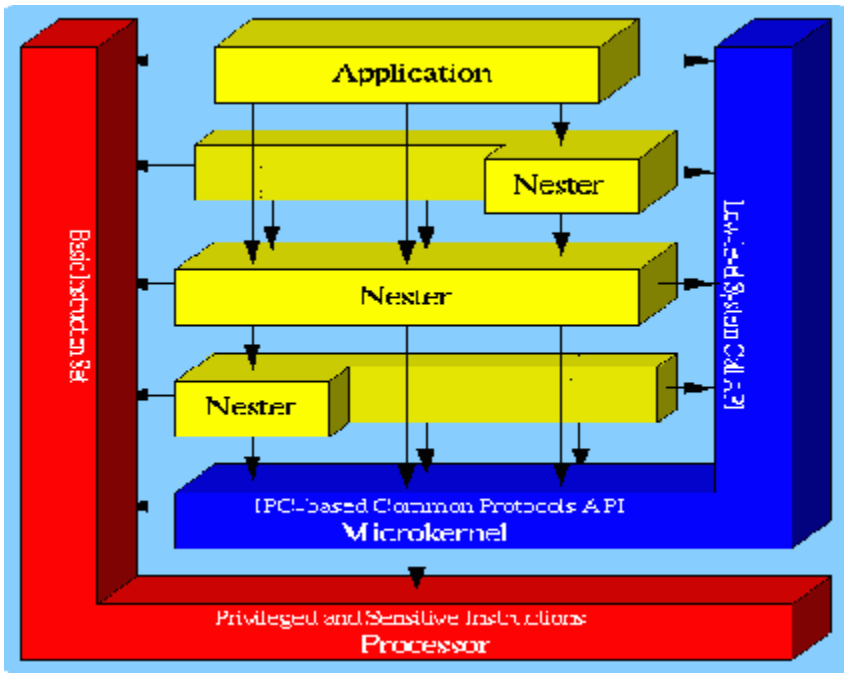
Depois

O projeto Mach 3 foi encerrado, mas a OSF continuou seu trabalho em cima de duas versões do mesmo: Mach 3 e MK++.

Por volta de 1995, a universidade de Utah começou um projeto chamado Mach 4 que tinha como objetivo investigar algumas idéias novas em cima do sistema, consertar alguns problemas e prover a base necessária para um projeto maior chamado Flux. Entretanto,

pretendia-se não fazer tantas modificações a ponto de torná-lo incompatível com as versões anteriores. Isso não foi possível, pois muitas modificações foram realizadas a fim de atender os objetivos do projeto Flux. Algumas versões do Mach 4 foram lançadas e distribuídas na rede.

O processo de modificação do kernel subdividiu-se em duas frentes chamadas de "head



branch" e "UK02 branch". A primeira era responsável pela pesquisa em si e a maior parte do desenvolvimento do Mach 4 aconteceu ali. Algumas modificações importantes foram realizadas por esse grupo:

- Migrating threads: um modelo para execução de RPC mais rapidamente.
- Kernel activations: algumas funções do kernel foram movidas para a área de usuários.

O caminho chamado de "UK02 branch" era responsável pelas modificações relativas a consertos de problemas existentes na versão anterior ou aumento de suas capacidades, como por exemplo suporte a novos sistemas de arquivos (MINIX e ext2), criação de um novo ambiente de desenvolvimento, novo sistema de boot, criação de arquivos "include" e modificação no estilo de codificação utilizada.

Por um tempo, o esforço de modificação do kernel do Mach foi a principal atividade dentro do projeto Flux, gerando uma grande quantidade de artigos que podem ser encontrados na rede (http://www.cs.utah.edu/projects/flexmach/papers.html#MACH).

Outros projetos estavam bastante relacionados com o Mach 4, como por exemplo o Lites, que objetivava criar uma versão pública do servidor BSD4.4Lite, mas ainda assim utilizável. Muitas importações foram realizadas com esse código que rodava em i386, PA-RISC, pc532, R3000 e Alpha. Na sua versão i386, o Lites suportava compatibilidade binária com o Linux, NetBSD e FreeBSD. Esse projeto também foi encerrado.

Com o fim do projeto Mach 4, gerando a base de conhecimento para o projeto Flux, foi criado em 1996 um sistema operacional chamado **Fluke [2]**, que mantém algumas características importantes do Mach, como o microkernel e o sistema de IPC, mas não aproveita grande parte de seu código, devido a grande complexidade do mesmo. O sistema Fluke propõe um novo paradigma de sistema operacional onde os processos são criados a partir de outros processos, mas esses filhos não executam seu código independentemente do seu pai. Na verdade, o processo pai cria um ambiente virtual onde o filho executará e, dessa forma, algumas funções que deveriam ser atendidas pelo sistema operacional agora podem ser realizadas pelo processo pai. Esse modelo é melhor compreendido num esquema de camadas onde os processos filhos ficam no topo e o sistema operacional embaixo da pilha. Esse esquema é apresentado na figura 2.

Outro projeto importante resultante do fim do Mach 4 é o **Flux OSKit [3]**, um kit de desenvolvimento e prototipação rápida de sistemas operacionais onde já estão implementadas várias funções básicas de um sistema e que possuem pouco interesse em termos de pesquisa. Módulos básicos como o carregador do sistema, o escalonador, o gerenciador de memória e uma infinidade de drivers para os mais diversos periféricos já estão disponíveis ao projetista do sistema operacional. Com esse kit, é possível desenvolver um novo sistema operacional em poucas horas, segundo o artigo que apresenta o projeto.

O sistema operacional Fluke utiliza toda a capacidade do OSKit, mas além do Fluke, muitas outras aplicações foram desenvolvidas utilizando esse pacote e muitas até mesmo na indústria. Isso possibilitou que o Kit não ficasse restrito apenas ao projeto do Fluke, mas atendesse a outras implementações com objetivos diferentes como um sistema baseado na linguagem ML chamado de ML/OS, outro baseado na linguagem SR chamado de SR/OS e um projeto que cria uma máquina virtual Java e um network computer baseado nesse sistema.

A linguagem Java despertou muito interesse no grupo de pesquisa Flux, provavelmente devido às suas similaridades com o trabalho desenvolvido lá (máquina virtual, linguagem não convencional e sistemas operacionais). Com isso, muitos trabalhos interessantes foram gerados com essa linguagem, onde destacam-se um sistema operacional Java (similar ao JavaOS, mas utilizando resultados e idéias das pesquisas realizadas pelo grupo) e um sistema de rede ativa, onde os pacotes de rede carregam código Java que é executado pelo roteador.

O Grupo de Pesquisa FLUX

O grupo de pesquisa Flux (http://www.cs.utah.edu/projects/flexmach/) trabalha com sistemas de software e seus interesses (e trabalhos) incluem muitas áreas como sistemas operacionais locais e distribuídos, segurança de informações e recursos, programação, linguagens não tradicionais, compiladores, redes, sistemas baseados em componentes, engenharia de software e métodos formais.

Atualmente, os principais projetos desenvolvidos pelo grupo são:

- Sistema operacional de alta segurança Fluke/Flask

- Linguagem de definição de interface Flick (e compilador)
- OSKit
- Sistemas baseados em Java: sistemas operacionais, gerenciamento de recursos, redes ativas. Esse tópico inclui um projeto para prover noção de processo dentro da máquina virtual Java

Existe muito software produzido por esse grupo e disponível em sua página. Entre eles, destacam-se o sistema de redes ativas (ANTS), o compilador IDL Flick e o Quarks, que é um sistema portável e eficiente de memória compartilhada distribuída. Além disso, encontram-se disponíveis nessa página o OSKit e o kernel do Fluke. Como mencionado anteriormente, alguns projetos que foram encerrados pelo grupo também disponibilizam seus resultados, que são o Mach 4 e o Lites.

As pesquisas realizadas por esse grupo na área de sistemas operacionais e segurança não existem por acaso. Entre os órgão financiadores dos projetos encontram-se DARPA, NSF, Hewlett-Packard, Novell e IBM.

A maioria das publicações do grupo encontram-se na rede no formato PostScript.

O Sistema Operacional Fluke

O Fluke [2] é um sistema de arquitetura virtualizável baseado em software que permite a criação de máquinas virtuais recursivas, ou seja, máquinas virtuais rodando sobre máquinas virtuais, implementado eficientemente sobre um sistema microkernel e rodando sobre hardware genérico.

Sistemas do tipo microkernel tentam distribuir as diversas funções do sistema operacional horizontalmente, colocando-as no espaço do usuário na forma de servidores (como o Lites, por exemplo). Já sistemas baseados em máquians virtuais recursivas tradicionais agem de outra forma, decompondo o sistema operacional de forma vertical, já que implementam as funções do sistema operacional na forma de pilha de "virtual machine monitors", cada qual exporta uma interface de máquina virtual compatível com a interface sobre a qual está executando.

Assim, o sistema operacional Fluke combina os dois paradigmas, o microkernel e o de máquinas virtuais, implementando uma arquitetura virtualizável que não pretende emular um hardware perfeitamente, mas sim é desenvolvido em software e sobre um sistema operacional microkernel.

O microkernel executa diretamente sobre a máquina e a arquitetura virtualizável é implementada em software na forma de "nested process". Os monitores de máquinas virtuais, chamados de nesters, podem eficientemente criar máquinas virtuais adicionais (outros nesters), sobre as quais outros nesters ou aplicações podem ser executadas. Essa arquitetura forma um espécie de pilha de nesters, onde no topo é executada a aplicação.

Em máquinas virtuais tradicionais (baseadas em hardware), cada máquina virtual exporta uma interface idêntica à máquina sobre a qual está rodando, provendo assim proteção e independência entre as várias máquinas virtuais. Já em uma arquitetura como a do

sistema Fluke (que é implementada sobre um microkernel), a interface exportada pelas máquinas virtuais (ou nesters) pode ser completamente diferente uma da outra.

Dessa forma, pode-se usar recursos tradicionais do sistema operacional, como o gerenciador de páginas, apenas quando necessário. Ao contrário de um sistema normal, onde, mesmo desabilitando seu uso, o recurso continua na memória, consumindo recursos sem ser utilizado. Essa independência é alcançada com a implementação de um nester responsável pelo gerenciamento de página. Assim, ele somente é usado quando necessário, ou seja, somente será colocado na pilha se for necessário.

Dessa forma, o sistema pode ser composto de acordo com a necessidade da aplicação que executará no topo. Isso garante uma grande flexibilidade sem que haja perda considerável de desempenho. Arquiteturas tradicionais de máquinas virtuais adicionam um overhead de 20% a 100% por camada acrescentada, enquanto a arquitetura implementada no Fluke apresentou índices de 0% a 35% por camada.

Conclusões

O sistema operacional Mach, apesar de ainda despertar bastante interesse, foi encerrado na versão 4. Entretanto, seus conceitos e idéias inovadoras deram origem a novos projetos e possibilitaram a criação de outros sistemas.

A combinação das idéias do Mach com outros trabalhos na área de sistemas operacionais impulsiona a pesquisa no sentido de encontrar formas alternativas de implementação de sistemas, visando flexibilidade sem perda de performance. Ainda assim mantendo a transparência dessas mudanças ao usuário.

O grupo de pesquisa Flux, de Utah, conta com apoio de importante órgão do governo americano e de empresas, o que garante a continuidade dos projetos por eles desenvolvidos.

Mais sobre o Sistema

[1] Tanembaum; Distributed Operating Systems, ...

[2] Bryan Ford, Mike Hibler, Jay Lepreau, Patrick Tullmann, Godmar Back, Stephen; Clawson **Microkernels Meet Recursive Virtual Machines**. Appears in Proc. OSDI'96, October,1996.

[3] Bryan Ford, Godmar Back, Greg Benson (U.C. Davis), Jay Lepreau, Albert Lin (MIT), Olin Shivers (MIT). **The Flux OSKit: A Substrate for OS and Language Research**. Appears in Proceedings of the 16th ACM Symposium on Operating Systems principles, Saint-Malo, France, October 1997