Tópicos em Sistemas Operacionais

Sistemas Operacionais Distribuídos e Multiprocessados

Por André Macedo, Marcelo Moraes, Thaís Fernandes

Sumário

- Teoria: Sistemas Multiprocessados
- Estudo de caso:K42
- Teoria: Sistemas Distribuídos
- Estudo de caso: Amoeba

Sistemas Multiprocessados

Sistemas Multiprocessados: Hardware

- Multiprocessadores UMA
  - Tempo de acesso à memoria uniforme
    - Memória privada
    - Cache
    - Memória compartilhada
  - Modelos:
    - Barramento único
    - Chaves crossbar
    - Redes de comutação
- Conectam no máximo 100 CPUs

Sistemas Multiprocessados: Hardware

- Multiprocessadores NUMA
  - Endereço único de memória compartilhada
  - Acesso remoto feito através de LOAD/STORE
  - Acesso local mais rápido que o remoto
- NC-NUMA: sem cache
- CC-NUMA: com cache (cache coerente)
  - Baseado em diretório: acesso por hardware especial

Sistemas Multiprocessados: Software

- Compartilhamento do sistema operacional
  - Cada CPU com um SO
    - Memória dividida estaticamente
    - Código do SO compartilhado
  - Mestre-escravo
    - Centralização das chamadas ao sistema
    - Melhor aproveitamento das CPUs ociosas
  - Multiprocessadores Simétricos
    - Divisão do SO em partes independentes
    - Uso de Mutex!

Sistemas Multiprocessados: Software

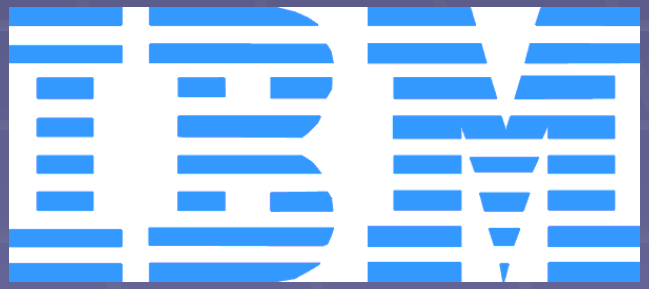
- Sincronização
  - Necessidade de um equivalente ao *test set and lock* (TSL)
    - Trava no barramento: hardware especial
      - Gasto de CPU com teste da trava (spin lock)
  - Uso de TSL com cache
  - Mutex individual em cada cache

Sistemas Multiprocessados: Software

- Escalonamento
  - Compartilhamento de tempo
  - Compartilhamento de espaço
  - Escalonamento em bando

K42

IBM Research



INTRODUÇÃO

O que é o K42?

- Novo kernel de sistema operacional para sistemas de multiprocessadores de 64-bits com cache coerente
- Open source
- Em constante evolução, incorporando mecanismos inovadores e técnicas de programação modernas

Motivação

- SOs tradicionais multiprocessados
  - Multiprocessadores pequenos
  - Latência de memória pequena em comparação com a velocidade dos processadores
  - Custo de compartilhamento de memória era baixo
  - Custo de acesso à memória uniforme
  - Cache e linhas de cache pequenas

Motivação

- K42
  - Terceira geração da pesquisa de sistemas operacionais multiprocessados.
  - Primeira geração: Hurricane OS
  - Segunda geração: Tornado OS
  - Desenvolvidos do zero especialmente para os multiprocessadores de memória compartilhada de hoje (NUMA)

Objetivos

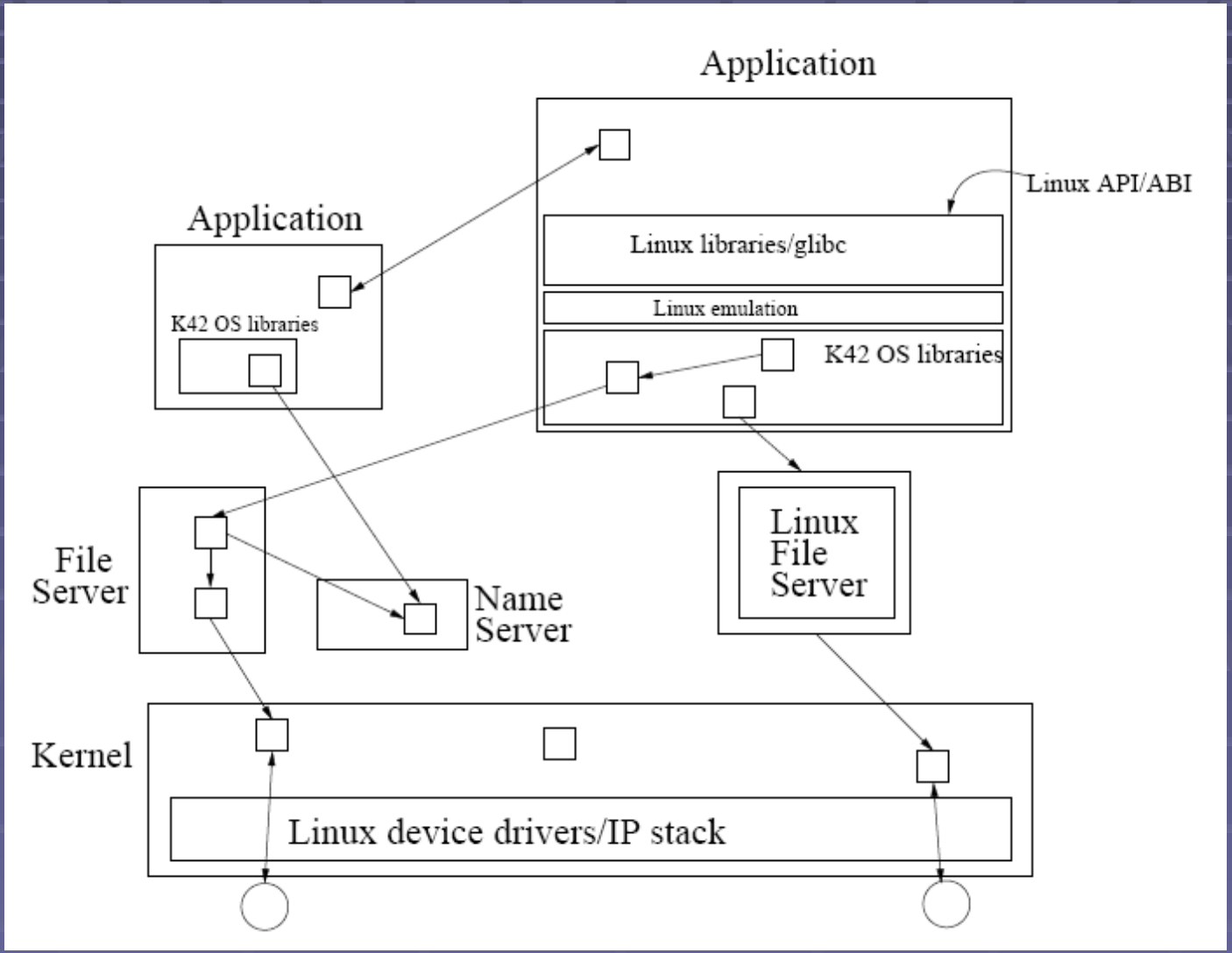
- Performance
- Escalabilidade
- Locabilidade
- Customização
- Manutenção
- Acessibilidade
- Aplicabilidade

ESTRUTURA DO K42

- Kernel
  - Gerenciamento de memória
  - Gerenciamento de processos
  - Infraestrutura de IPC
  - Escalonamento básico
- System Servers
- Bibliotecas em nível de aplicação
  - Evitar overhead de IPC
  - Flexibilidade

- Orientada a objetos
  - Toda comunicação entre processos é feita entre objetos
- Suporte à API do Linux
  - Através de uma camada de emulação que implementa chamadas ao sistema Linux chamando métodos em objetos K42.

Modelo Cliente-Servidor



TECNOLOGIA DO K42

- Suporta as interfaces externas e internas e modelos de execução do Linux, para suportar suas aplicações e kernel

- Design feito para que o K42 seja facilmente portado para novos hardwares e depois melhorado para explorar features específicas do hardware utilizado

- K42 mantém boa performance através de um mecanismo IPC eficiente (com performance comparável a chamadas de sistema) memória compartilhada entre clientes, servers e kernel, diminuindo custos de comunicação

- K42 é totalmente preemptivo e a maior parte dos dados do kernel pode ser paginada – libera mais memória para aplicações

- K42 usa clocks sincronizados em diferentes processadores, permitindo que processos sejam executados simultaneamente por pequenos períodos de tempo em múltiplos processadores

- Garbage collection modificado
  - Deleção de objetos K42 acontece somente após todas as threads que estejam rodando no momento terminaram

IMPLEMENTAÇÃO DE SERVIÇOS DE SISTEMA EM NÍVEL DE USUÁRIO

Bibliotecas no espaço de endereçamento da própria aplicação

- Customização sem sacrificar segurança e performance de outras aplicações
- Redução de overhead
- Libera o kernel e os system servers
- Implementação tão eficiente quanto os serviços de outros sistemas operacionais

Escalonamento de Threads

- Totalmente no nível de usuário
- Kernel – dispatcher representa o processo do usuário
- Consome poucos recursos do Kernel, torna o escalonamento mais eficiente e é mais flexível, possibilitando otimizações

Interrupções de timer

- Dispatcher mantém apenas o próximo timeout
  - O resto fica armazenado no espaço de endereçamento da aplicação
- Melhora performance
  - Maioria dos timeouts são cancelados antes de ocorrer
- Evento de timer
  - Notificação assíncrona para o dispatcher

Faltas de página

- O estado da thread é mantida no kernel apenas o tempo suficiente para determinar se a falta foi in-core
- Faltas in-core - kernel resolve. Caso contrário, dispatcher muda para outra thread ou desiste
- Quando a página é recuperada, um evento assíncrono é passado para o dispatcher, que decide o que fazer
- Custo pouco mais alto que salvar o estado do kernel por evitar salvar ou copiar registradores desnecessariamente

Serviços IPC

- Serviços do kernel são bem básicos:
  - Muda o processador do espaço de endereçamento do sender para o receiver
  - Mantém a maioria dos registradores intactos
  - Manda um identificador do sender para o receiver
- Bibliotecas no nível de usuário:
  - Organiza argumentos nos registradores
  - Define regiões compartilhadas para transferência de dados
  - Lida com pedidos de autenticação
- Tão eficiente quanto os melhores serviços IPC de Kernel na literatura

Serviços de I/O

- Tradicionalmente:
  - servers mantem um estado para cada pedido de uma thread cliente
- K42:
  - se o server não pode responder no momento, envia um erro e a aplicação bloqueia a thread no seu próprio espaço de endereçamento.
  - Server mantém informação de todas as aplicações ligadas a uma porta de comunicação e as notifica quando dados estão disponíveis.

MODELO ORIENTADO A OBJETOS

Customização

- Instâncias de objetos por recurso permite regras e implementações diferentes para cada caso
  - Exemplo: permite que cada região de memória tenha um tamanho de página diferente ou cada processo lidar com exceções diferentemente
- Componentes de uma aplicação podem ser trocados por novas implementações sem tirar o sistema do ar
  - Exemplo: a implementação de um arquivo pode ser modificada a medida que ele cresce, de uma otimizada para arquivos pequenos para uma otimizada para arquivos grandes

Performance de multiprocessadores

- Boa escalabilidade:
  - Objeto encapsula todos os dados necessários para gerenciar um recurso e todos os locks necessários para acessar os dados.
  - Não existe locks ou estrutura de dados globais
  - Acessos a diferentes recursos são gerenciados pelo sistema totalmente em paralelo.

Performance de multiprocessadores

- Clustered objects:
  - Tecnologia utilizada para objetos muito compartilhados
  - Permite que a implementação desses objetos seja particionada ou replicada nos processadores que a utilizam
  - Transparente para os clientes desse objeto

Interação cliente/servidor

- Stub-compiler: ligação do mecanismo IPC à interfaces C++
  - Objeto “Stub”: apresenta a interface para a aplicação cliente e organiza chamadas à mesma através do mecanismo IPC
  - Objeto “X”: recebe mensagens IPC e reorganiza em chamadas ao objeto real que exportou a interface
  - A ligação entre um objeto “X” e o seu target é feito em tempo de compilação baseado nas assinaturas dos métodos (polimorfismo)

Interação cliente/servidor

- Stub-compiler inclui um modelo para executar autenticação
  - Chamadas IPC recebem um id pelo kernel
  - Cada chamada possui diferente direitos de acesso que são verificados antes que o mecanismo IPC realize a chamada
  - Clientes com o acesso correto podem dar acesso ao mesmo objeto a outros clientes, independente do tipo de objeto

Outros aspectos

- Alta modularidade
  - Facilita contribuições ao código
  - Objetos com implementações específicas podem ser introduzidos sem penalidades para aplicações que não os utilizam
  - Facilita comparação entre duas implementações diferentes
  - Hot-swap permite que patches de segurança, correção ou melhoria de performance sejam instalados sem tirar o sistema do ar.

Desafios

- Cuidado extra ao usar C++
  - Uso de objetos grandes para evitar overhead
  - Examinar o código assembly gerado para descobrir bugs de performance no compilador
- Dificuldade de chegar a um estado global
  - Muitas instâncias – por exemplo, difícil de rodar a thread com a maior prioridade global
  - Área de pesquisa

Sistemas Distribuídos

Sistemas Distribuídos: Hardware

- Utiliza sistemas completamente separados
- Não tem limitação em espaço físico
- Funcionam sobre uma rede de dados comum

Sistemas Distribuídos: Software

- Cada CPU tem seu próprio SO
- Camada de abstração: Middleware
- Tentativa de uniformizar a interface entre usuário e sistemas

Sistemas Distribuídos: Software

- Middleware baseado em documento
  - Uso de repositórios de arquivos
  - Protocolo definido para acesso
- Exemplos:
  - Internet

Sistemas Distribuídos: Software

- Middleware baseado no sistema de arquivos
  - Modelo upload/download
    - Transferência de arquivos inteiros
  - Modelo acesso remoto
    - Transferência de trechos modificados
- Exemplo:
  - AFS

Sistemas Distribuídos: Software

- Middleware baseado em objetos compartilhados
  - Protocolo para utilização de qualquer recurso
- Exemplos:
  - CORBA
  - Globo

Sistemas Distribuídos: Software

- Middleware com base em coordenação
  - Baseado no uso de tuplas na rede
  - Conjunto mínimo de operações
- Exemplos:
  - Linda
  - Publica/Escreve
  - Jini

Amoeba

Filosofia

- Computadores estão se tornanado cada vez mais baratos e rápidos
- Aumentar a performance em redes de computadores
- Necessidade de lidar com hardware fisicamente distribuído e usando software logicamente centralizado
- Prover a sensação de um único e simples computador com sistema de timesharing.

Objetivos

- Deve ser pequeno
- Simples de usar
- Escalabilidade para um grande número de processadores
- Bom grau de tolerância a faltas
- Alta Performance
- Possibilidade de paralelismo
- Transparente para os usuários

Diferença entre Amoeba e S.O. em rede

Amoeba

- Usuário loga em um sistema como um todo
- O sistema decide o melhor lugar para rodar o programa
- Serviço de nomes de objetos, único e amplo

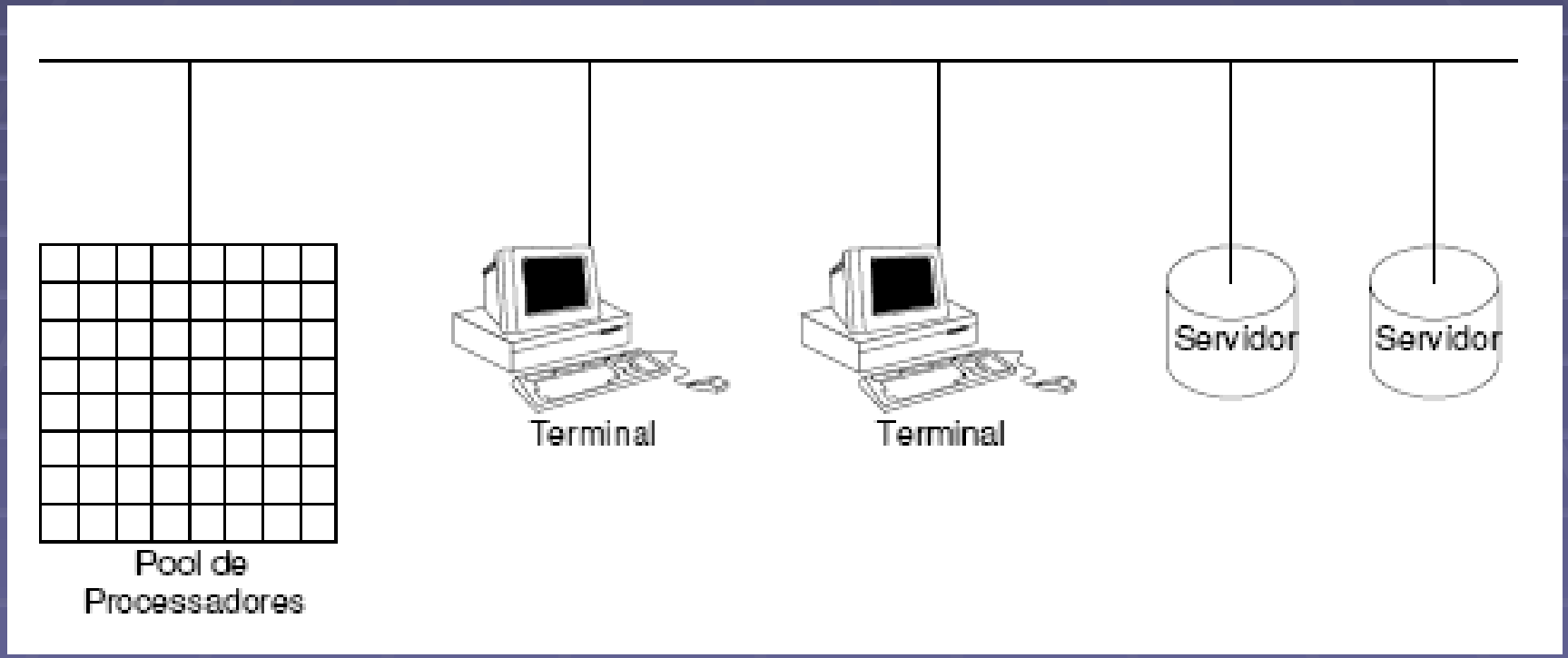
S.O em rede

- Usuário loga em uma máquina específica
- Prograna executa na máquina local, a menos que seja especificada outra.
- Arquivos são locais, a menos que um sistema remoto seja montado.

Arquitetura

- Workstations
- Pool de processadores
- Servidores especializados
  - Arquivo
  - Base de dados
  - Diretórios
  - Etc
- Gateway

Arquitetura



Workstations

- Uma por usuário
- Onde os usuários realizam suas tarefas
- Estas podem ser Suns, IBM, PC/AT e terminais X

Pool de Processadores

- Grupo de CPU's
- CPU pode ser dinamicamente alocada e depois retorna ao pool
- Oferece a possibilidade de processamento em paralelo.

Servidores especialiados

- Servidor de diretórios
- Servidor de arquivos
- Servidor de boot
- Servidor de banco de dados
- Entre outros…

Gateways

- Liga sistemas Amoeba em diferentes lugares em um sistema único e uniforme.
- Isola o sistema Amoeba das peculiaridades dos protocolos de rede que devem ser usados sobre as WAN's.

Amoeba Kernel

- Todas as máquinas Amoeba rodam o mesmo kernel
- O kernel é responsável pela gerenciamento de memória, I/O, comunicação entre processos, primitivas de objetos, processamento básico e multithreading.
- O objetivo é manter o kernel o menor possível para aumentar confiança e permitir ao S.O. rodar como se fosse um processo de usuário.

Objetos

- Amoeba é **baseado** em objetos e não **orientado** a objetos.
- São acessados através de capacidades.
- O sistema pode ser visto como uma coleção de objetos que podem ser tanto hardware como software. Objetos software são mais comuns.
- Cada objeto tem um lista de operações que podem ser realizadas e uma capacidade ou ticket.

Remote Procedure Call

- É utilizado para a comunicação entre cliente e servidor.
- Os processos de usuário enviam uma mensagem de pedido para o servidor que gerencia um objeto.
- O pedido contém a capacidade do objeto, a operação a ser realizada e alguns parâmetros

Remote Procedure Call

- O processo do usuário é bloqueado enquanto espera a chamada se completar.
- Depois de completa, o servidor responde com uma mensagem que desbloqueia o processo usuário.
- Enviando pedidos, sendo bloqueados e aceitando respostas resulta no processo de RPC. Este por sua vez é encapsulado em rotinas.

Remote Procedure Call

- Interface RPC é construída sobre o protocolo FLIP
- RPC é baseado em 4 primitivas
  - Getreq – Get requisition
  - Putreq – Put requisition
  - Trans - Transaction
  - Timeout – Time out

Capacidade ou ticket

- Para criar um objeto, o cliente manda um RPC para o servidor e recebe de volta uma capacidade para acessar esse objeto.
  - Uma capacidade é formada por 128 bits:
    - 48 bits com o *server port* (endereço do servidor e sua porta)
    - 24 bits com a identificação do objeto
    - 8 bits com os direitos de acesso
    - 48 bits de verificação
- Provê um mecanismo unificado para nomes, acesso e proteção de objetos
- É criptografada para evitar que outros clientes alterem os privilégios da capacidade.

Protocolo FLIP

- FLIP (Fast Local Internet Protocol) foi desenvolvido por Andrew Tanenbaum.
- Seu objetivo é otimizar a velocidade das RPC's do sistema.
- FLIP também provê automaticamente o menor roteamento de mensagens e provê automaticamente a passagem entre gateways de redes conectadas.

Getreq

- Usada por servidores
- O servidor inicializado, realiza uma operação de getreq com a porta que deseja ouvir.
- getreq bloqueia o servidor até que um cliente envie um pedido utilizando a operação trans.
- Quando um pedido chega, o servidor checa se a capacidade é válida e se tem privilégios suficientes.

Putreq

- Quando um servidor completa uma operação, ele manda uma resposta ao cliente usado putreq.
- Logo depois disso ele executa a operação getreq.

Trans

- Usada por clientes para enviar pedidos aos servidores.
- A porta do servidor é o primeiro parâmetro da operação trans.
- A operação trans bloqueia o cliente até que o servidor mande uma resposta.
- O kernel no qual a operação trans é executada tenta localizar o servidor transmitindo por broadcasting o pedido juntamente com a porta.

Trans

- Se um outro kernel tem um servidor esperando por um pedido naquela porta, ele responde e o RPC é enviado ao servidor, o qual processa o pedido e retorna uma resposta.
- O kernel mantêm um cachê de portas conhecidas para melhorar a performance para próximos RPC's
- Trans não é chamada diretamente pelos clientes, mas embutida em uma chamada de processo que organiza os dados e o envio de mensagens.

Timeout

- Usado para informar a quantidade de tempo gasta procurando um servidor para atender o pedido de uma operação trans.
- timeout padrão é 5 segundos.

Programação Cliente/Servidor

- Não é necessário que a programação do cliente ou do servidor se preocupe com a preparação de dados ou rotinas de transporte na rede.
- A própria AIL(Amoeba Interface Language) gera esse código automaticamente
- Servidorer podem executar processos multithreaded mas isso não é obrigatório

Threads

- Cada processo tem seu próprio espaço de endereço e contém multiplas threads.
- Essas threads tem suas próprias pilhas e descritor de processos, mas compartilha os dados globais e o código.

Diretórios

- Nomes de arquivos não implementados pelo servidor de arquivos
- Muito mais tipos de objetos que arquivos, então a necessidade de um serviço de nomes geral.
- Servidor de diretórios implementa o serviço de nomes, implementando um grafo direcionado arbitrário de diretórios

Diretórios

- Manipula os nomes de arquivos
- Objeto diretório é uma lista de pares.
  - Os pares são formados de uma string para o nome e um conjunto de capacidades.
- Um grau de tolerância extra é alcançado pela duplicação do servidor de diretórios que se comunicam entre si para manter a integridade dos diretórios.
- É o único que conhece a localização física de cada arquivo.

Arquivos

- O servidor de arquivos é conhecido como Bullet Server.
- Armazena os arquivos contiguamente no disco, para dar ao sistema uma alta performance.
- Servidores de arquivos são imutáveis. Uma vez criados não podem ser mudados.

Amoeba e emulação POSIX

- Biblioteca de Emulação POSIX provê uma compatibilidade de código fonte razoável.
- Isso provê um Amoeba com um ambiente de programação e simplifica a migração de softwares de sistemas UNIX-like.

Pontos importantes

- O sistema é gratuito
- Ele não teve uma atualização oficial em 10 anos.
- Pode-se utilizar CPUs mais velhas e lentas para crias um sistema poderoso.
- Micro-kernel permite que outros sistemas de arquivos sejam criados.
- Tem 4 objetivos principais
  - Distribuição
  - Paralelismo
  - Transparência
  - Performance
- Tem muitos comandos e programas parecidos com o UNIX
- Pode somente manipular programas tão grandes quanto a sua memória física

Muito Obrigado!

- Referências
  - Tanenbaum A., Sistemas Operacionais Modernos, www.prenhall.com/tanenbaum_br
  - http://www.eecg.toronto.edu/~tornado/
  - http://domino.research.ibm.com/comm/researc
  - http://www.cs.vu.nl/pub/amoeba/