PCS3446 – Sistemas Operacionais

Prof. João José Neto

Estudos de Caso

Aula 20 – Sistema Amoeba

Amoeba
/əˈmibə/

- Histórico -

A. Tanenbaum

- Origem na **Vrije Universiteit**, Amsterdam, **Holanda**
- **1981** - projeto de pesquisa em **sistemas distribuídos e paralelos** (Tanenbaum, mais três alunos de doutorado)
- **1983** - protótipo inicial em operação (**Amoeba 1.0**)
- **1984** - o grupo original se separou
- Nos anos seguintes, o projeto estendeu-se à **Noruega** e à **Inglaterra** (com o patrocínio da **Comunidade Européia**)
- **Amoeba 3.0** usava **RPC (Remote Procedure Calls)**, diferentemente das versões iniciais.
- Evoluiu por vários anos: emulação parcial do **UNIX**, comunicação de **grupos**, protocolo de baixo nível inédito.

- Objetivos e características -

- A maioria dos sistemas distribuídos partiu de um sistema existente, tal como o **UNIX**
- **Amoeba** começou do **zero**, com a finalidade de criar e experimentar **novas ideias**.
- Posteriormente, foi adicionada a **emulação** de **UNIX** para evitar a reescrita (recodificação) de muitos **aplicativos** já disponíveis.

- Transparência e Multiprocessamento -

- Objetivo principal - construir um **sistema operacional distribuído genuinamente transparente**.
- O usuário vê um sistema de **time-sharing** análogo aos usuais, sem se dar conta do **multiprocessamento** nele disponível.
- A diferença é que em todas as tarefas executadas é possível que se envolvam **diversas máquinas** (**servidores**).

- Ausência de máquina hospedeira -

- Amoeba é um sistema operacional que **não implementa** o conceito de **máquina hospedeira**, mas trata a todas uniformemente.
- Por essa mesma razão, do ponto de vista hierárquico, do sistema operacional, **nenhuma máquina é associada a qualquer particular proprietário**.
- Ao ingressar no Amoeba, o usuário se conecta sempre **ao sistema**, nunca a um particular hardware.

- Distribuição de carga -

- Depois que Amoeba estiver instalado em um dado processador, os comandos que forem acionados no sistema são em geral executados em alguma outra máquina (a que na ocasião estiver sendo **menos utilizada**)
- Isso tende a **distribuir a carga do sistema** mais ou menos **uniformemente** pelas máquinas disponíveis em cada momento.

- Distribuição de recursos -

- Assim como os processadores, os demais recursos disponíveis no computador são sempre considerados propriedades **do sistema** como um todo, sendo, por essa razão, **alocados pelo sistema**.
- No sistema operacional Amoeba, **alocações dedicadas** de recursos, embora necessárias, são em geral consideradas inconvenientes, por isso, quando utilizadas, são mantidas na posse do usuário pelo **mínimo tempo possível**.

- Transparência -

- No sistema Amoeba, a **transparência** é uma meta que é promovida até o seu limite: por exemplo, o comando ***amake*** (equivalente ao ***make*** do UNIX) dispara todas as tarefas necessárias, e o sistema decide, **sem a interferência do usuário**, como e quando cada uma delas deve ser executada: paralela ou sequencialmente .

- Experimental, Paralelo e Distribuído -

- Uma meta secundária do sistema Amoeba foi a de servir como **ambiente experimental** para a prática e desenvolvimento de atividades de **programação paralela e distribuída**.
- Assim, enquanto alguns usuários se apoiam no sistema Amoeba como se fosse um ambiente para uso particular, outros podem tirar proveito dos recursos que oferece em função da sua disponibilidade de **múltiplos processadores**.

- Paralelismo transparente -

- Dessa forma, embora o recurso do **paralelismo** esteja disponibilizado a todos, apenas o utilizam aqueles que realmente desejam fazer uso do mesmo, permanecendo **transparente** para os demais.
- Todavia, em sua maior parte, os **usuários** do sistema Amoeba são pessoas que se interessam por **sistemas distribuídos** e pelo uso do **paralelismo** em suas aplicações.

- Linguagem Orca -

- **Orca,** uma linguagem de programação de sistemas, de alto nível, foi especificamente desenvolvida para essa finalidade.
- Contudo, o próprio sistema operacional Amoeba, paradoxalmente, **não foi desenvolvido na linguagem Orca,** mas em **C**.

- Trecho de um programa, em linguagem **Orca** -

```
process worker(minimum: shared IntObject;  q: shared  taskqueue) ;
        r : route;
begin
        do # forever
                r:= q$remove_head( ) ;
                tsp(r,minimum);
        od;
end ;
function tsp(r: route;  minimum: shared IntObject) ;
begin
        # cut-off (partial) routes longer than the current best one
        if length(r) < minimum$value( ) then
                if ``r'' is a full solution(covering all cities) then
                        # r is a full route shorter than the current best
                        # route, so update the current best solution.
                        minimum$min(length(r));
                else
                        for all cities ``c'' not on route ``r'' do
                                # search route r extended with c
                                tsp(r||c , minimum) ;
                        od;
                fi;
        fi;
end ;
```

- Arquitetura -

- O número de processadores disponíveis pode ser bastante **elevado (centenas, até milhares)**, e cada processador tem **dezenas de Megabytes** de memória principal
(quantidade considerada bastante **grande para a época**)
- A grande força desse sistema é que ele pode **integrar,** de forma **trivial,** um conjunto **muito grande** de **processadores**.

- Recursos -

- Amoeba vê conjuntos de recursos tais como: **X-terminals**, **servidores** e um ***pool heterogêneo de processadores***
- O ***pool* de processadore**s é formado de computadores diversificados, com **variadas arquiteturas**: 68030, 386, VAX, SPARC.

- Multiprocessamento heterogêneo -

- Amoeba foi projetado para uso em **sistemas heterogêneos**, mesmo para processos de um mesmo programa.
- Para executar novas threads, prefere-se **alocar novos processadores**. Isso foi desde o início uma opção de projeto.
- No entanto, caso haja escassez de processadores, recorre-se à **multiprogramação**, porém o sistema não foi sintonizado para esse tipo de utilização.

- Memória -

- A hipótese da presença de **muita memória** (pelo padrão da época) nos processadores **teve forte impacto** no projeto do Amoeba.
- Um **terminal** típico deste sistema possui um X-terminal, tela grande *bit-mapped*, *mouse*, ou então, um computador com **X-Windows**

- Processadores -

- **Processadores** do *pool* são **baratos** porque só possuem uma placa de hardware, com uma conexão com a rede.
- Os processadores do *pool* podem ser computadores **pessoais** ou então **estações** de trabalho.

- *Pool* e servidores -

- O *pool* pode não estar fisicamente confinado em um único local físico.
- **Servidores especializados** são essenciais no sistema Amoeba, e têm processamento contínuo, por opção do administrador.

- Servidores e serviços -

- **Exemplo**: pode haver um ou mais servidores de diretórios, para otimizar o manuseio dos diretórios dos processadores.
- **Servidores** produzem e disponibilizam **serviços**. Para isso, sua implementação pode implicar o emprego de **diversos processadores**.

O **Microkernel** do sistema Amoeba

- Amoeba tem duas partes: o ***microkernel,*** residente em todos os processadores, e um conjunto de **servidores.**
- Segue o modelo de **cliente-servidor**, e os servidores realizam as funcionalidades que são usualmente implementadas pelos sistemas operacionais usuais.

- *Microkernel* -

- O mesmo *microkernel* é utilizado tanto no *pool* de **processadores**, como nos **terminais** e nos **servidores**.
- Responde por **quatro funções gerenciais**:
  - **processos e *threads***
  - **memória** (gerenciamento básico)
  - **comunicação**
  - operações de **Entrada/Saída de baixo nível**

- *Threads* -

- ***Threads*** (cada *thread* tem seus próprios registradores, contador de instruções e pilha).
- Conjunto de *threads* de um mesmo processo: similar aos processos independentes do UNIX, porém compartilhando o espaço de endereçamento.
- Uso típico de ***multithreads***:
  servidor de arquivos – solicitações diferentes abrem *threads* sequenciais independentes.

- **Memória segmentada**.
  Operações de leitura e escrita na memória (*Read, Write)*, com Mapeamento no espaço de endereçamento do *thread* chamador
- Um processo pode ter *diversos segmentos*, totalmente a critério do usuário:
  - de texto,
  - de dados,
  - de pilha, etc.

- Comunicação e E/S -

- **Comunicação** de processos:
  - **ponto a ponto** (cliente-servidor, com espera) e
  - de **grupo** (por mensagens).
- **Entrada/Saída:**
  drivers (não dinâmicos) para os dispositivos. Comunicam-se por meio de troca de *mensagens*.
- Protocolo **FLIP** foi desenvolvido com a meta de ser eficiente em computação distribuída.

- Amoeba Servers -

- Para minimizar o *kernel*, todas as operações que este executava devem transformar-se em serviços realizados externamente em modo usuário.
- Os servidores realizam todas essas tarefas que o *microkernel* não executa mais, minimizando-o, e tornando-o ágil e flexível.

- Modelo cliente-servidor -

- Amoeba se baseia no modelo **cliente-servidor**.
  - **Clientes** são programas escritos, em geral, pelo usuário.
  - **Servidores** são criados por programadores de sistema, ou também pelo usuário, para a prestação de serviços.

- Objetos, *capabilities*, *stubs* -

- Os **objetos** no sistema Amoeba funcionam como um tipo de dados abstrato
(dados encapsulados, associados a um conjunto exclusivo de operações específicas)
- Os **usuários** de cada objeto recebem, do servidor de objetos, um código de acesso (***capability***) para os seus usuários.
- Cada servidor tem seus ***stubs*** disponíveis na biblioteca. O usuário ativa esses *stubs* para requisitar **serviços**.

- Sistema de arquivos -

- A divisão do sistema de arquivos em dois servidores simplifica e flexibiliza sua operação:
  - ***Bullet server***
    É o servidor de arquivos propriamente dito.
    Cria arquivos que **não podem ser modificados**, apenas **removidos**.
  - ***Directory server*** ou ***soap server***
    Gerencia nomes de arquivos, caminhos na árvore e *capabilities*.

- Outros servidores -

- **Outros servidores** copiam objetos, iniciam processos, monitoram falhas no servidor, comunicam com o exterior.
- **Servidores do usuário** – executam diversas tarefas, específicas de cada aplicativo.

Objetos

- **Objeto** (tipo abstrato de dados, **passivo**) é o **conceito unificador** usado nos servidores e nos serviços do sistema Amoeba.
- Os objetos são criados e manipulados via **RPC** (***remote procedure call***) por um **servidor de objetos**.
- Este executa o pedido e, **sincronamente,** retorna o resultado ao solicitante.

- O uso dos objetos -

- Ao criar um objeto, o programa criador recebe do sistema a ***capability*** correspondente, espécie de autorização (passaporte) para acesso e manipulação desse objeto.
- Ao cliente é **irrelevante**:
  - a **localização física** dos seus objetos
  - a informação de qual seja exatamente **o particular servidor** que atende à sua solicitação.

Capabilities

- São códigos **criptografados** únicos, **transparentes** quanto ao local, e próprios de **cada objeto.**
- **Nomeiam e protegem** objetos de maneira **uniforme**, e são associados aos objetos na ocasião em que estes são criados.
- Contêm as seguintes informações:
  - ***server port*** (máquina do servidor),
  - ***object*** (identificação do objeto),
  - ***rights*** (direitos),
  - ***check*** (redundância)

- Proteção -

- **Proteção** – permite restringir, por meio de uma função **unidirecional,** o direito de acesso do usuário ao objeto, de acordo com a *capability* por ele apresentada.
- A **criptografia** unidirecional evita que os direitos de acesso sejam burlados.
- No sistema Amoeba **não são utilizadas listas** de controle de acesso.

- Operações disponíveis -

- ***Age*** (*garbage collection*)
- ***Copy / Destroy***
  (duplica/apaga objeto, fornece *capability* para a cópia)
- ***Setparams / Getparams***
  (estabelece/obtém parâmetros associados ao servidor)
- ***Info*** (obtém informação breve sobre o objeto)
- ***Status*** (obtém o estado corrente do servidor)
- ***Touch*** (finge que acabou de usar o objeto)
- ***Restrict*** (solicita novo *capability,* mais restrito)

- Processos e *Threads* -

- **Processo**
  (espaço de endereçamento, acrescido de um conjunto de *threads* associados ao programa)
- Processos em Amoeba são **objetos**, hierarquizados em **árvores**.
- ***Thread***
  (linha de processamento sequencial, disparada pelos processos)

- Criação e gerência de processos -

- Ao criar processos, o processo-pai ganha uma *capability* que lhe dá direito de **suspender, reiniciar** e **remover** seus processos-filhos
- **Gerência efetuada em três níveis**:
  - nível **baixo** (servidores, *threads* do *kernel*)
  - **médio** (bibliotecas, interfaces para usuários)
  - **alto** (*run server*)

- Processos -

- **Descritores de processos** – com informações:
  - Tipo de processador
  - Capability
  - Descritores de segmentos
  - Conjunto de *threads*
- **Segmentos** – com as seguintes informações:
  - Pilhas, com *stack pointers*
  - Texto, com *program counters*
  - Dados, compartilhados ou privados

- *Threads* -

- ***Threads***
  - Modelo simples.
  - Usam variáveis "**glocais**"
    - Invisíveis a outros *threads*, mas
    - Globais às rotinas do *thread*
- **Sincronização de *threads:***
  - ***Signal*** (interrupção assíncorna)
  - ***Mutex*** (semáforo binário)
  - Semáforo **de Dijkstra** (contador)
- ***Scheduling*** de ***threads***
  emprega um esquema de prioridades, e associa prioridades mais altas aos ***threads*** do ***kernel***.

- Memória -

- **Memória**
  - **Segmentada**, não paginada, com **alocação contígua**
  - **Não utiliza a paginação** oferecida pelas MMU's
- **Número arbitrário de segmentos** por processo.
- Operações: ***create, destroy, read, write***
- Permite criar **sistema de arquivos em memória** porque dispõe de ***read / write.***
- **Segmentos** podem, no espaço de endereçamento desejado:
  - **Ser mapeados** (associados a endereços físicos)
  - **Ser desmapeados** (eliminados do espaço físico)

Sumário

- **Amoeba** faz um conjunto de computadores independentes **parecer um único sistema** multiprogramado.
- Implementa **transparência em vários níveis**:
  - do **processador** em que é executado
  - da **localização** física dos arquivos
  - das **cópias** de segurança

Sumário (2)

- Dá acesso a todos os recursos de **multiprocessamento** ao usuário que deseje usufruir do **paralelismo** disponível.
- ***Microkernel*** com
  - gerenciamento em **dois níveis**
    - **de processos** e
    - **de memória**
  - **comunicação**
  - **entrada / saída**.

Sumário (3)

- Sistema de arquivos e demais componentes do SO são sempre executados como **processos do usuário**.
- ***Capabilities*** fornecem mecanismo simples para **nomear/proteger** objetos (implementam esquema de direitos de acesso aos recursos disponíveis no sistema)

Sumário (4)

- A proteção é feita por meio de **criptografia unidirecional**.
- As *capabilities* contêm uma redundância do tipo ***checksum*** para maior segurança
- Dois mecanismos de **comunicação**:
  - comunicação confiável de **grupo** (por mensagem) ou
  - RPC **ponto a ponto**.
- Utiliza o protocolo **FLIP, muito eficiente** e específico para ambiente distribuído.

Sumário (5)

- **Arquivos**: Utiliza três servidores para isso:
  - ***bullet server***
    (para os arquivos, todos **não modificáveis**),
  - ***directory server***
    (tolerante a falhas,
    converte *strings* ASCII em *capabilities*), e
  - **replication server**
    (manuseia **replicação preguiçosa**)

Esta apresentação foi baseada no material de Tanenbaum – *Modern Operating Systems,* cap.14 Prentice Hall, 1992