Sistemas Operacionais I

Modelo de Processo

Multiprogramação de quatro programas

4 processos sequenciais independentes

Somente um programa está ativo

**Tempo de Descompactador**

Baixa a resolução, menos pontos, tráfego menor

Principais elementos que levam à criação de processos:

1- Início de sistema

2- Execução de chamada ao sistema de criação de processos

3- Solicitação do usuário para criar um novo processo

4- início de um job (processo) em lote

**Término de Processos**

1. saída normal (voluntária)

2. saída por erro (voluntário) – de programação

3. erro fatal – tenta escrever na memoria e recebe erro. Não está no nosso controle

4. cancelamento por outro processo (involuntária) – gerenciador de tarefas, por exemplo.

Processo que chama processo 🡪 processo pai

Processo chamado 🡪 processo filho

* Windows não trata hierarquia de processos!!!! Todos os processos são processos pai.

Pronto 🡪 Rodando 🡪

Bloqueado

**27/08/19 – THREADS**

Escalonamento 🡪 cada fluxo de execução é uma thread.

Monothread 🡪 1 processo (fluxo de execução) por vez

Processador Multithread, S.O multithread,

Sem estes itens, não é possível rodar multithread.

Cada thread do usuário é gerenciada por uma thread do núcleo. Isso demanda muito recurso, mas é muito mais fácil em termos de programação. Se uma thread morre, a linha inteira morre também.

Se uma thread morre, procura outra para executar (multithread) (implementar é mais complicado, mas é bem mais eficiente).

Modelo 1-N o Kernel executa todos os threads  
não tem um sistema operacional específico.

Sem o multithreading, o sw não acompanha o hardware do jeito que gostaria

Startar uma thread é MUITO mais leve do que startar um processo inteiro.

Diagrama de Escalonamento de Processos

Pronto – esperando – executando

Dentro do contexto do processo, posso priorizar algumas threads

Starvation

Paralelismo x Concorrência

Threads Daemon

Encosto –

Daemon tools

**PROVA ATÉ AQUI**

**01/10/2019 - KERNEL**

Núcleo

Modo Kernel = supervisor (tipo um modo administrador – sudo, root, admin etc.)

Modulos principais, camada de comunicação básica (terminal), camada de comunicação gráfica, talvez intercomunicada...

OpenShift

Sistema e aplicações (codigos de alto nivel)🡪 kernel (cria ambiente para aplicações) 🡪 assembler (baixo nível) 🡪 firmware (BIOS mais limitada, faz o reconhecimento de todo o hardware) 🡪 hardware

IPC – Comunicação interprocessos

Formas de como os processos se comunicam entre si (FIFO, pipes, filas de mensagens, memória compartilhada, RPC – remoto – socket – “chats”, semáforo)

Pipe – mandar dados de um processo a outro em uma estrutura de dados. Fila.

Half duplex, unidimensional, um sentido só.

FIFO – é um pipe que fica residente na memória enquanto o S.O estiver ativo. Não preciso criar pipes o tempo todo, mas ocupa muita memória

Mensagens –

Memória compartilhada – posições estáticas de memória que apontam para um espaço da memória.

Semáforo – tipo a msm coisa

RPC (?)

Tipos de Kernel

microKernel – se comunica com serviços do Kernel. Poucas funções são acessadas diretamente. Se comunica com os serviços de alguma coisa, e não com a coisa diretamente. Tem q fazer um IPC. Ninguém tenha acesso às coisas do S.O. Resolve o problema do Kernel monolítico. Manutenção mais simples. (exemplo: android). Permissões (GPS, câmera etc.)

(nanoKernel) – kernel minimizado (Abstrai o kernel). HAL – Hardware Abstraction Layer. Promíscuo, uma placa de rede serve pra vários. Priorização de sistemas de camadas mais altas. Oferece suporte hard em tempo real.

Kernel monolítico – todas as funções concentradas em um lugar só (arquivo tipo DOS). Qualquer um pode acessar tudo. Não existe nas máquinas modernas (são obsoletos). Não estruturado, mas possui algumas estruturas. DLL dinamic ... libraries. FCFS - FIFO

Kernel Híbrido – tenho serviços e/ou posso acessar diretamente, desde que saiba mexer direito (monolítico e modular). Modernos são híbridos. Mac tem híbrido e monolítico para rodar tudo os programas legados. Não tao rigido quanto o monolítico. Arquivos podem ser acessados se não tiver em modo administrador por exemplo. Para instalar drive vc precisa do administrador. Camada reduzida (só em modo kernel), modo executivo (usuário).

ExoKernel – troca/atualiza as bibliotecas caso dê erros. Não é utilizado amplamente porque não tem bom processamento, mas é o melhor exemplo de todos.

Sistemas em camadas   
UNIX veio depois

~intervalo~

**Virtualização**

Java Virtual Machine

Memória Virtual

Virtualização de Storages

Virtualização de Desktops

Virtualização de Celulares

Computation of loading

Mainframes – alto custo

Solução: segurança, isolamento, ensino/aprendizagem, teste de aplicações e aplicações legadas.

VMM ou hypervisor

- fornece à VM uma cópia virtual (abstração) dos recursos físicos do sistema hospedeiro

Garante que várias VMs possam ser executadas simultaneamente sobre um mesmo hardware.

Cada núcleo do processador é dividdiodo em quatro aneis: 0, 1 2 e 3.

0: todo o processamento

3: tem algum processamento

1 e 2: a maior parte do tempo ociosos

Vt-x destina quase todo o processamento aos anéis 1 e 2 para desafogar a máquina.

Máquina virtual usa os anéis 1 e 2. Não entra em conflito de desempenho. Não da pra criar uma máquina virtual com mais de 50% da capacidade de processamento do sistema origem pois ele bloqueia.

Normalmente as GPUs estão em modo mínimo de processamento, então não da pra fazer a VM.

Virtualização total: hypervisor é o nanokernel. Só tem nanokernel na máquina. Vc vai criando as máquinas virtuais

Para-virtualização:

Virtualização assistida por hardware (hospedada): feita com a VM que a gente faz mesmo

Microsoft  
Virtual PC: virtualiza windows, só roda no windows (?)

Virtual Server:

Hyper-V: ótimo, supimpa, show

Xen

XenServer é free, da Citrix.

Docker

Container: um processo