Aula 2 - Introdução a Sistemas Operacionais - Parte 2

Sistemas Operacionais
Ciência da Computação
IFB - Campus Taguatinga

Professor João Victor de A. Oliveira

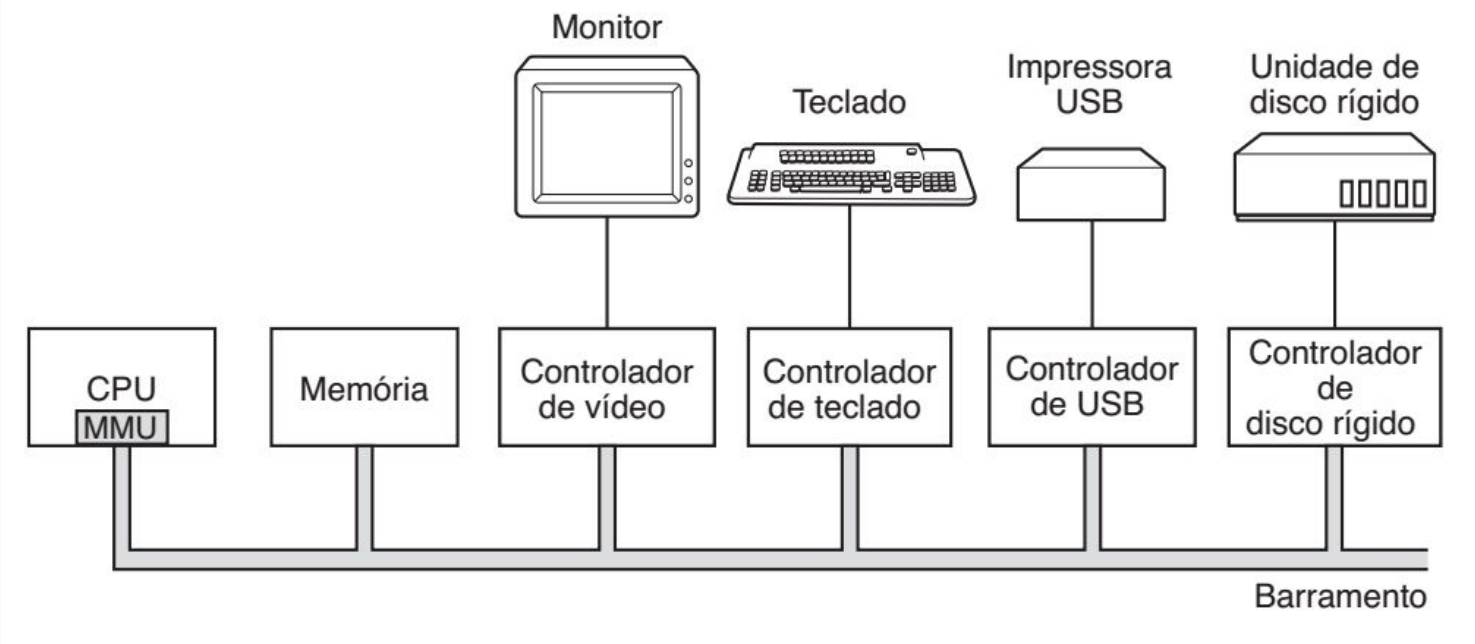
Na aula passada

- Funções de um Sistema operacional
- História de um Sistema Operacional

Hoje (Parte 1)

- Breve revisão de Arquitetura de Computadores
- Zoológico dos Sistemas operacionais

Revisão rápida Hardware de Computadores



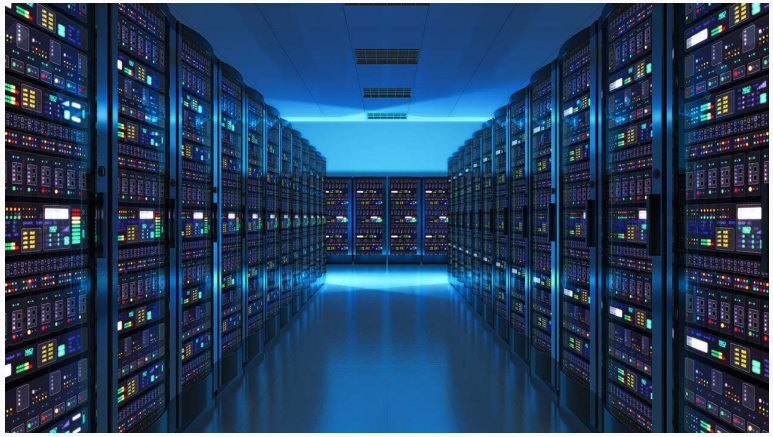
Modelo de computador pessoal simples

Zoológico dos Sistemas Operacionais

- Grande variedade de diferentes SOs ao longo da metade deste último século
  - SOs de grande porte
  - SOs de servidores
  - SOs de multiprocessadores
  - SOs de computadores pessoais
  - SOs de computadores portáteis
  - SOs embarcados
  - SOs de nós sensores (*sensor node)*
  - SOs de tempo real
  - SOs de cartões inteligentes (smart cards)

SOs de grande porte

- **Mainframes**
  - Máquinas do tamanho de uma sala, possuindo muitas vezes mais de 1000 discos e milhões de gigabytes de dados
    - Tem retornado às empresas como servidores sofisticados da web, para sites de comércio eletrônico em larga escala e para transações entre empresas (*business to business)*
  - Diferem dos computadores pessoais em termos de sua capacidade de E/S
    - Deve processar várias e várias tarefas ao mesmo tempo, a maioria delas exigindo grande quantidades de E/S



SOs de grande porte

- **SOs de grande porte geralmente oferecem 3 tipos de serviços:**
  - **Em lote (batch)**
    - Processa tarefas rotineiras sem qualquer usuário interativo presente
    - Ex.: processamento de apólices de seguros ou relatório de vendas
  - **Processamento de transações**
    - Lidam com grande números de requisições pequenas,
    - Ex.: reservas de companhias aéreas de centenas/milhares de usuários por segundo
  - **Tempo compartilhado (*timesharing*)**
    - Múltiplos usuários remotos executem tarefas no computador ao mesmo tempo
    - Ex.: consultas em bancos de dados
- Ex. de SO de grande porte: OS/390
  - Pouco a pouco estão sendo substituídos por variantes UNIX, como o linux

SOs de servidores

- Servem múltiplos usuários ao mesmo tempo por meio de uma rede e permitem que os usuários compartilhem recursos de hardware e software
  - Servidores podem fornecer serviços de impressão, de arquivo ou de web
- SOs típicos
  - Solaris, FreeBSD, Linux e Windows Server 201X

SOs de Multiprocessadores

- Maneira comum de se obter potência computacional: **Conectar múltiplas CPUs a um único sistema**
  - Computadores Paralelos, multicomputadores, multiprocessadores
  - Usam variações de SOs de servidores, com aspectos especiais para **comunicação, conectividade e consistência**
- "**Novidade**": Chips multinúcleo para computadores pessoais
  - Cada vez mais núcleos por chip
  - **Desafio:** Fazer que os aplicativos desenvolvidos usem toda essa potência computacional
- **SOs Típicos: Windows e Linux**

SOs de Computadores Pessoais

- **Oferecem suporte à multiprogramação**
  - Dezenas de programas inicializados no momento da inicialização do sistema
- **Tarefa: proporcionar um bom apoio a um único usuário**
  - Processamento de texto, planilhas e acesso à internet.
- **SO's Típicos:**
  - Windows, Linux, OS X da Apple

SOs de Computadores Portáteis

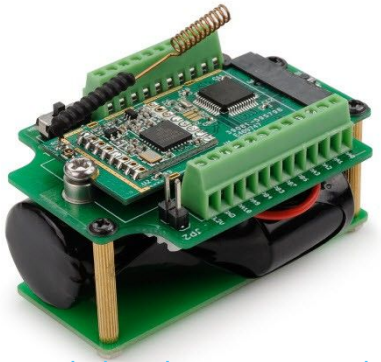
- Usados em Tablets, smartphones e outros computadores portáteis
  - **PDAs** - Personal Digital Assistant
  - **PMDs** - Personal Mobile Devices
- Contam com CPUs multinúcleos, GPS, câmeras e outros sensores, uma boa quantidade de memória e SOs sofisticados
- Exs. de SO:
  - Android
  - iOS

SOs Embarcados

- **Embarcados**
  - Dispositivos que não costumam ser vistos como computadores e que não aceitam softwares instalados pelo usuário
    - Microondas, Aparelhos de DVD, Telefones tradicionais, MP3 player, dentre outros
- **Principal Propriedade dos SOs embarcados:**
  - Nenhum software não confiável será executado nele um dia
  - Software está armazenado em memória ROM
  - **Não há necessidade para proteção entre aplicativos** (ninguém vai raquear seu Microondas)
- **Exs. de SOs embarcados:** *Embedded Linux, QNX e VxWorks*

SOs de nós sensores (*Sensor-node*)

- **Redes de nós de sensores estão sendo usados para diversas finalidades**
  - Nós são computadores minúsculos que se comunicam entre si e com uma estação-base usando comunicação sem fio
  - Movidos a bateria com rádios integrados
    - Precisam funcionar por longos períodos desacompanhados ao ar livre e frequentemente enfrentando condições severas
  - A rede deve ser **robusta** o suficiente para tolerar falhas de nós individuais, o que acontece com mais frequência à medida que as baterias começam a se esgotar



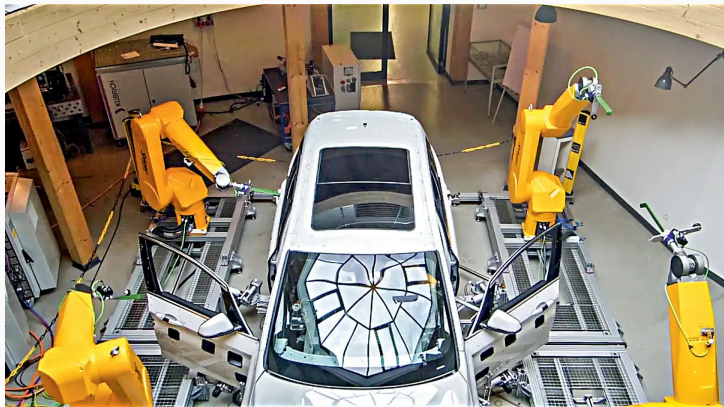
https://www.antratek.com/waterproof-long-range-wireless-lora-sensor-node

SOs de nós sensores (*Sensor-node*)

- **Cada nó sensor é um verdadeiro computador**
  - Possui CPU, RAM, ROM e um ou mais sensores ambientais
  - Executa um SO pequeno e simples, mas verdadeiro, geralmente **orientado a eventos**
    - Responde a eventos externos ou tomando medidas periodicamente com base em um relógio
  - Programas devem ser carregados antecipadamente
- SO mais conhecido: TinyOS

SOs em tempo real

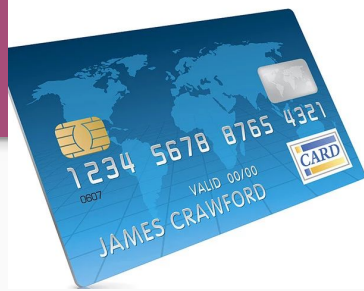
- **Caracterizados por ter o tempo como um parâmetro chave**
  - Precisam coletar dados a respeito do processo de produção e usá-los para controlar máquinas em uma fábrica, por exemplo.
  - Muitas vezes seguem prazos rígidos a serem cumpridos **(Sistemas de tempo real crítico)**
    - Se um carro está seguindo pela linha de montagem, determinadas ações têm de ocorrer em dados instantes.
    - Se, por exemplo, um robô soldador fizer as soldas cedo demais ou tarde demais, o carro será arruinado.



SOs em tempo real

- **SOs de tempo real não críticos**
  - Caso perca o prazo ocasionalmente, não causará danos permanentes
  - Ex.: Sistemas multimídia ou áudio digital, Smartphones, dentre outros
- **SOs em tempo real críticos normalmente são uma biblioteca conectada com programas aplicativos**
  - Todas as partes do SO são estreitamente acopladas e sem nenhuma proteção entre si
  - Ex. de SO: **eCos**
- Normalmente, SOs em tempo real e embarcados são bastante relacionados

SOs de cartões inteligentes (*smartcards*)



- Operam em dispositivos do tamanho de cartões de crédito contendo um **chip de CPU**
  - Possuem **severas restrições de memória e processamento de energia**
    - Alguns obtêm energia por contatos no leitor no qual são inseridos ou por indução
  - Alguns deles realizam apenas uma função (ex.: pagamentos eletrônicos), mas outros realizam diversas funções
  - Normalmente usam **SOs proprietários**
  - Gerenciamento de recursos e proteção são problemáticos quando dois ou mais applets (java) estão presentes na ROM
    - SO deve tratar dessas questões (normalmente os SOs são bastante primitivos)

Parte 2

- Conceitos de Sistemas Operacionais
  - Processos
  - Espaços de Endereçamento
  - Arquivos
  - Entrada/Saída
  - Proteção
  - Interpretador de Comandos (Shell)
- Estrutura de Sistemas Operacionais

Processos

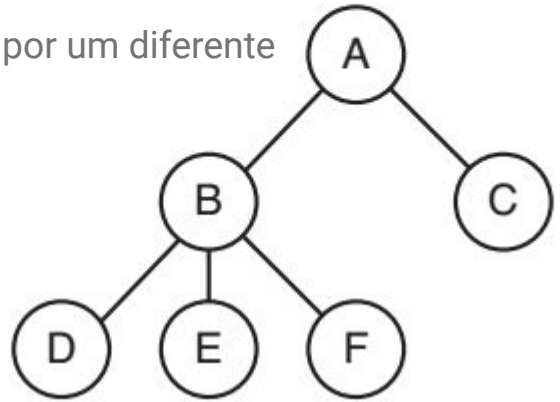
- **Um processo é um programa em execução**
  - Associado a um **espaço de endereçamento**
    - Posições de memória que vai de 0 a algum máximo onde o processo pode **ler ou escrever**
    - Contém o programa executável, os dados do programa e sua pilha
  - Também associado a um **conjunto de recursos**
    - Registradores, lista de arquivos abertos, alarmes pendentes, listas de processos relacionados, dentre outros
  - Um processo é, em essência, um contêiner que armazena todas as informações necessárias para executar um programa

Processos

- Em sistemas de multiprogramação, as informações de cada processo estão na **tabela de processos**
  - Um processo pode estar em execução ou suspenso em um dado momento (ignoremos os *"zumbis"* por enquanto :])
  - Um processo (**suspenso**), de modo geral, consiste em seu espaço de endereçamento (**imagem do núcleo)** e de sua **entrada na tabela de processos**
    - A entrada na tabela de processos armazena os conteúdos dos registradores e muitos itens necessários para **reiniciar** o processo mais tarde

Processos

- **Um processo (pai) pode criar um ou mais processos (filhos)**
  - Processos relacionados que estão cooperando entre si para finalizar alguma tarefa muitas vezes precisam comunicar entre si e sincronizar as atividades
    - Essa comunicação é chamada **Comunicação de Processos**
    - Veremos como fazer isso nas próximas aulas
- Outras chamadas de sistema de processos
  - Requisitar mais memória ou liberá-la (malloc( ) e free( ))
  - Esperar que um processo filho termine e sobrepor seu programa por um diferente
  - Dentre outros que veremos mais adiante



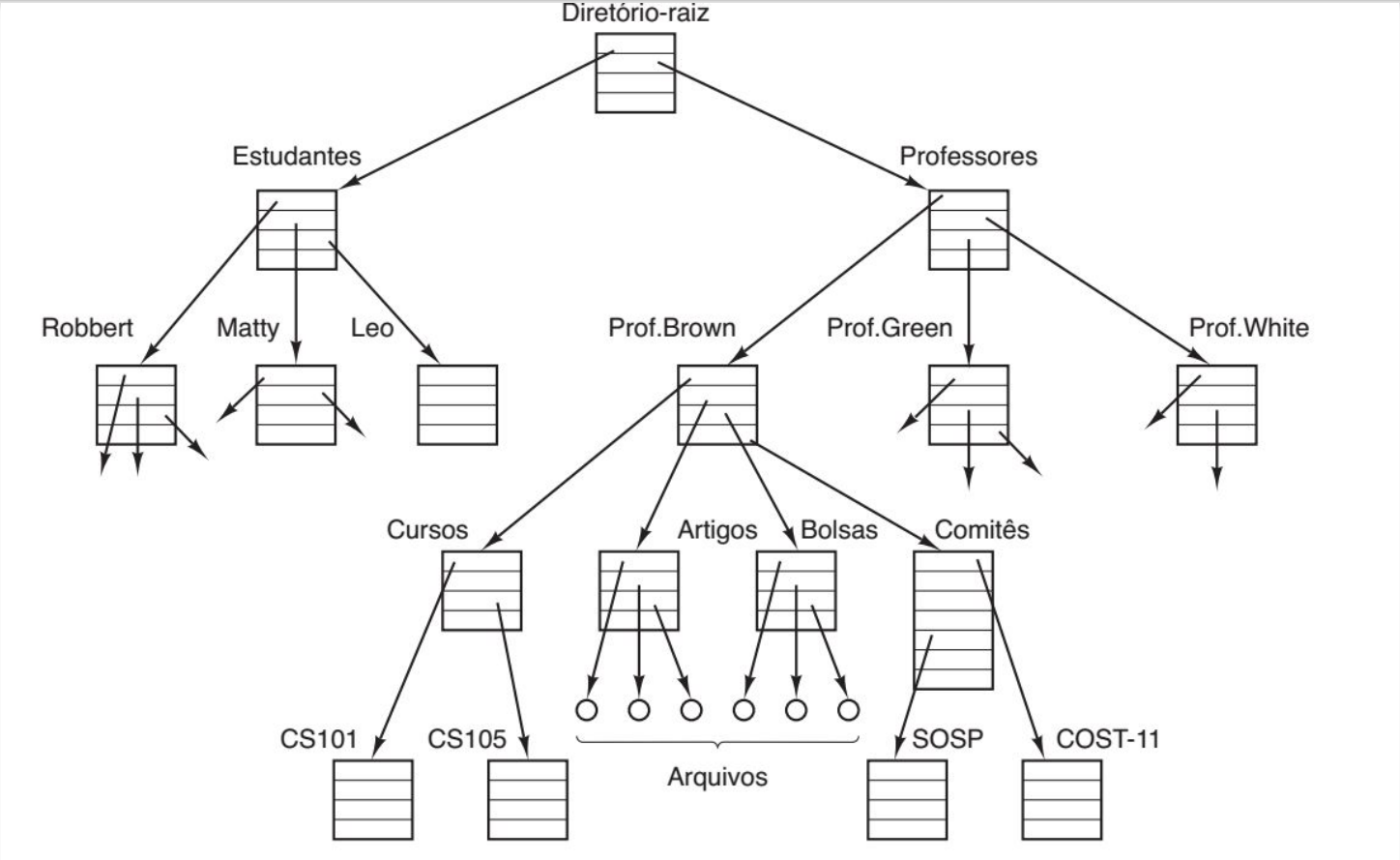
Espaços de Endereçamento

- SOs modernos permitem que múltiplos programas estejam na memória ao mesmo tempo
  - Para evitar que interfiram entre si, algum tipo de mecanismo de proteção é necessário
    - Normalmente via hardware controlado pelo SO
  - Cada processo tem um conjunto de endereços que pode usar
    - Geralmente é menor que o tamanho da memória principal
    - **O que acontece se um processo tem mais espaço de endereçamento do que o computador tem de memória principal e o processo quer usá-lo inteiramente?**
      - **Memória Virtual**
        - O SO mantém parte do espaço de endereçamento na memória principal e parte no disco, enviando trechos entre eles para lá e para cá conforme a necessidade (**swap in, swap out**)

Arquivos

- Conceito fundamental: **Sistemas de Arquivos (SA)**
  - **Funções importante do SA:**
    - Esconder as peculiaridades do disco e dos dispositivos de E/S
    - Apresentar ao programador um modelo agradável e claro de arquivos que sejam independentes dos dispositivos
  - Chamadas de sistemas usadas para **criar, remover, abrir, fechar, ler e escrever arquivos**
  - Local de um arquivo: **Diretório**
    - Chamadas de sistema também atuam na criação e remoção de diretórios
    - Entradas de diretórios podem ser arquivos ou outros diretórios, criando uma hierarquia de diretórios

Sistemas de arquivos para um departamento universitário



Arquivos

- Arquivos dentro de uma hierarquia de diretório podem ser especificados fornecendo seu **nome de caminho**
  - iniciado a partir do topo da hierarquia do diretório, chamado de **diretório raiz** (normalmente indicado por uma barra **/** ou barra invertida ****, no caso do windows)
  - Ex.:
    - /Professores/JoaoVictor/Cursos/SO.
- Cada processo possui, a cada instante, um **diretório de trabalho**
  - Nele são procurados nomes de caminhos que não começam com uma barra
  - Processos podem mudar seu diretório de trabalho emitindo uma chamada de sistema especificando o novo diretório de trabalho

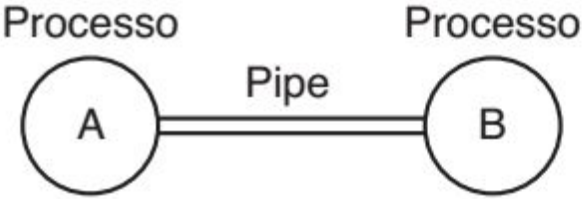
Arquivos

- **Arquivo Especial**
  - Permitem que dispositivos de E/S se pareçam com arquivos
  - Podem ser lidos e escritos com as mesmas chamadas de sistema que são usadas para ler e escrever arquivos
- **Dois tipos de arquivos especiais:**
  - **Arquivos especiais de bloco**
    - Modelam dispositivos que consistem numa coleção de blocos aleatoriamente endereçáveis, como os discos
  - **Arquivos especiais de caracteres**
    - Usados para modelar impressoras, modems e outros dispositivos que aceitam e enviam um fluxo de caracteres

Arquivos

- **PIPE**
  - Espécie de ***pseudoarquivo*** que pode ser usado para conectar dois processos
  - Quando A quer enviar dados para B, ele deve escrever no pipe como se fosse um arquivo
  - O processo B pode ler os dados a partir do pipe como se fosse um arquivo de entrada

FIGURA 1.16 Dois processos conectados por um pipe.



Entrada/Saída

- Todos os computadores têm dispositivos físicos para obter entradas e produzir saídas.
  - Todo SO tem um subsistema de E/S para gerenciar esses dispositivos
  - Alguns softwares de E/S são independentes do dispositivo
    - Aplicam-se igualmente bem a muitos ou a todos dispositivos de E/S
  - Outras partes dele, como drivers de dispositivo, são específicos a dispositivos de E/S particulares

- Veremos ao longo do curso mais detalhes sobre softwares de E/S

Proteção

- Cabe ao SO gerenciar a segurança do sistema de maneira que os arquivos sejam acessíveis somente por usuários autorizados
  - Arquivos em UNIX, por exemplo, usam um **código de proteção binário de 9 bits**
  - O código de proteção consiste de 3 campos de 3 bits
    - Um para o proprietário
    - Um para os outros membros do grupo do proprietário
    - Um para os demais usuários
  - Cada campo tem um bit de permissão de **leitura**, um bit de permissão de **escrita** e outro bit de permissão de **execução**
    - Esses bits são conhecidos como **bits rwx**

Proteção

- **Ex.:** rwxr-x--x (ou 751)
  - O proprietário pode ler, escrever e executar o arquivo,
  - Os membros do grupo podem ler ou executar o arquivo
  - Todos os demais apenas podem executar o arquivo
- Para um diretório, o **x** indica permissão de busca
  - Um traço significa ausência de permissão
- **Exercício:** O que significa o código de proteção 743 para um determinado diretório?

Interpretador de Comandos (shell)

- **SO é o responsável por executar todas as chamadas do sistema**
  - Embora o interpretador de comandos (shell, no UNIX) não faça parte do SO, ele faz uso intensivo de muitos aspectos do SO e serve assim como um **bom exemplo de como as chamadas do sistema são usadas**
  - **Exemplos de Shell:** sh, csh, ksh e bash

Interpretador de Comandos (shell)

- Quando qualquer usuário se conecta, o shell é iniciado
  - **O shell tem o terminal como entrada-padrão (STDIN) e saída-padrão (STDOUT)**
  - Ele inicia emitindo um caractere de prompt, um caractere como o cifrão do dólar, que diz ao usuário que o shell está esperando para aceitar um comando
  - Exemplo: Se o usuário digitar **date:**
    - O shell irá criar um **processo filho** e executa o processo **date** como um filho
    - Enquanto o processo filho estiver em execução, o shell espera que ele termine
    - Quando o filho termina, o shell emite o sinal de prompt de novo e tenta ler a próxima linha de entrada

Interpretador de Comandos (shell)

- Se o usuário quiser especificar que a **saída-padrão** seja redirecionada para um arquivo, ele pode usar o símbolo **>** para isso
  - **$ date > file**
- De modo similar a **entrada-padrão pode ser redirecionada**
  - **$ sort < file1 > file2**
    - Invoca o programa sort com a entrada vindo de file1 e a saída enviada para file2
- Saída de um programa pode ser a entrada de outro programa, através de um pipe
  - **$ cat file1 file2 file3 | sort > file4**

Interpretador de Comandos (shell)

- Se o usuário colocar um **&** após um comando, o shell não espera que ele termine
  - Em vez disso, ele dá um prompt imediatamente
    - **Ex.: sleep 30 &**
- Existem diversas outras operações em shell...
  - Veremos outras ao longo do curso :)

Chamadas de Sistemas

Chamadas de Sistemas

- **Principais Funções do SO:**
  - **Prover abstrações aos programas de usuário e**
    - ex.: criação, escrita, leitura e exclusão de arquivos
  - **Gerenciamento dos recursos computacionais**
    - Transparente ao usuário é feita automaticamente
- Interface entre SO e Programas de usuário feita a partir de abstrações
  - **Chamadas de sistemas**
  - Iremos estudar as chamadas de sistema do **Padrão POSIX (*international standard)***
    - UNIX, System V, BSD, LINUX, MINIX, ...

Chamadas de Sistemas

- **Implementação altamente dependentes da linguagem de máquina**
  - Normalmente são expressas em linguagem *assembly*
    - utilizadas a partir de bibliotecas para programas de linguagem C, por exemplo.
- Vale lembrar: só é possível executar uma instrução por vez (CPU de um único núcleo)
  - Quando um programa em modo usuário necessita de uma chamada de sistema, é necessário executar uma instrução de ***trap***

Exemplo de Chamada de Sistema

- **Chamada de Sistema: int read (fd, buffer, nbytes)**
  - Três parâmetros:
    - **fd:** especifica o arquivo
    - **buffer:** ponteiro indicando onde os dados serão armazenados
    - **nbytes:** indica quantos bytes serão lidos
  - Retorno:
    - **Inteiro indicando quantos bytes foram lidos**
      - Normalmente o mesmo que **nbytes**
      - Pode ser menor que **nbytes**, caso encontre o sinal de ***end-of-file (EOF)***
    - Caso o sistema não consiga executar a requisição é retornado o valor **-1**
      - Programas de usuário devem sempre checar se uma chamada de sistema apresentou erro!!!

Exemplos de chamadas de Sistema POSIX

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Exemplos de chamadas de Sistema POSIX

Directory- and file-system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

Correspondência dos chamados de sistema entre UNIX e Win32

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount, so no umount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

Estrutura dos Sistemas Operacionais

Estrutura dos sistemas operacionais

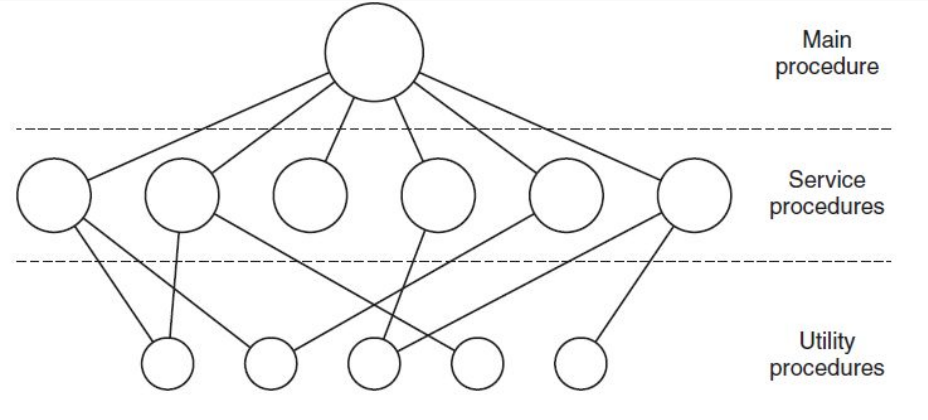
- **Examinaremos 6 estruturas diferentes**
  1. Sistemas monolíticos
  2. Sistemas em camadas
  3. Microkernels
  4. Sistemas cliente-servidor
  5. Máquinas virtuais
  6. Exokernels

Sistemas monolíticos

- Todo o SO é executado como um único programa em modo núcleo
  - Escrito como uma coleção de rotinas (*procedures*) interligadas em um gigantesco programa binário executável
  - Cada rotina é livre para para chamar outra rotina (não há ocultação de informações)
    - Torna o SO muito eficiente
    - Torna o SO difícil de entender
      - Quebra de qualquer uma dessas rotinas derrubará todo o sistema operacional!
- De certa forma, o SO não possui uma estrutura bem definida
  - Onde encontrar o programa executável de um SO do linux (ubuntu, por exemplo)?
    - cat /proc/cmdline

Sistemas monolíticos minimamente estruturados

- **SOs monolíticos minimamente estruturados**
  - **Estrutura básica**
    1. Um **programa principal** que invoca a rotina de serviço requisitada
    2. Um **conjunto de rotinas de serviço** que executam as chamadas de sistema
    3. Um **conjunto de rotinas utilitárias** que ajudam as rotinas de serviço



Sistema em Camadas

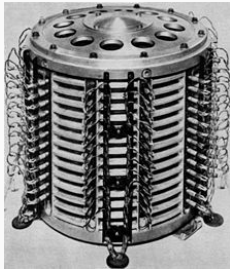
- **Organizar o SO como uma hierarquia de camadas**
  - Primeiro sistema desenvolvido dessa maneira: THE [Dijkstra, 1968], possuindo 6 camadas
    0. **Alocação do processador e multiprogramação**
       - Chaveamento de processos no CPU, quando ocorriam interrupções ou quando os temporizadores expiravam
       - Camadas superiores do SO não precisam se preocupar com o fato de que múltiplos processos estavam sendo executados em um único processador

Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

Sistema em Camadas

Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

1. **Gerenciamento de memória (Memória e Gerenciamento de tambor)**
   - Alocação de espaço para processos na memória principal em um tambor magnético de 512 k palavras usado para armazenar partes de processos **(páginas)** para as quais não havia espaço na memória principal
   - Processos das camadas de cima não precisavam se preocupar se eles estavam na memória ou no tambor magnético
2. **Comunicação operador-processo**
   - Acima desta camada cada processo tinha seu próprio console de operação
3. **Gerenciamento de entrada/saída**
   - Encarregava do gerenciamento dos dispositivos de E/S
   - Processos da Camadas acima podiam lidar com dispositivos de E/S abstratos mais acessíveis, ao invés de dispositivos reais com muitas peculiaridades



Sistema em Camadas

4. **Programas de usuário**
    - Não precisam se preocupar com o gerenciamento de processo, memória, console ou E/S
5. **Operador do sistema**
    - Só precisa se preocupar em usar os programas disponíveis 😄

Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

Sistema em Camadas

- Outra generalização do conceito de camadas no sistema MULTICS
  - **Série de anéis concêntricos**
    - Anéis internos tem mais privilégios do que os externos
    - Quando um procedimento em um anel exterior quer chamar um procedimento em um anel interior, ele tinha de fazer o equivalente de uma chamada de sistema ***(trap)***
  - **Vantagem**
    - **Pode ser estendido para estruturar subsistemas de usuário**
      - Um professor poderia escrever um programa para testar e atribuir notas a programas de estudantes executando-o no anel n
      - Já os programas dos estudantes seriam executados no anel n+1, de maneira que eles não pudessem mudar suas notas

Micronúcleos

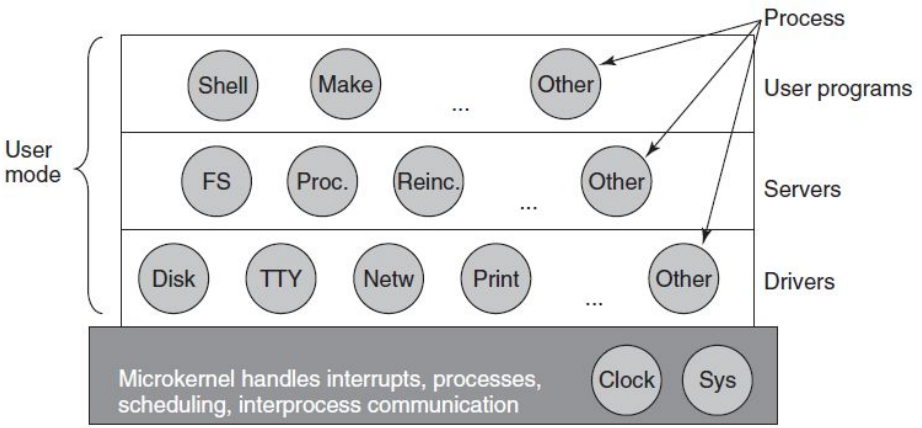
- **Ideia:** reduzir o tamanho das rotinas que executam em modo núcleo
  - Erros no código do núcleo em um sistema monolítico podem derrubar o sistema instantaneamente
    - Processos de usuários podem ser configurados para terem menos poder, de maneira que um erro possa não ser fatal
  - Sistemas industriais → 2 a 10 erros por mil linhas de código
    - Em um sistema monolítico de 5 milhões de linhas de código é provável que contenha entre 10.000 a 50.000 erros no núcleo

Micronúcleos

- **Objetivo**: atingir alta confiabilidade através da divisão do SO em **módulos pequenos** e **bem definidos**
  - Apenas um pequeno módulo - o micronúcleo - é executado em modo núcleo
  - Demais módulos executam em modo usuário
    - Um erro em um desses módulos não consegue derrubar o sistema inteiro
      - ex.: Erro no driver de áudio
- Ex. de SO baseado em micronúcleo: **OS X, baseado no micronúcleo Mach**

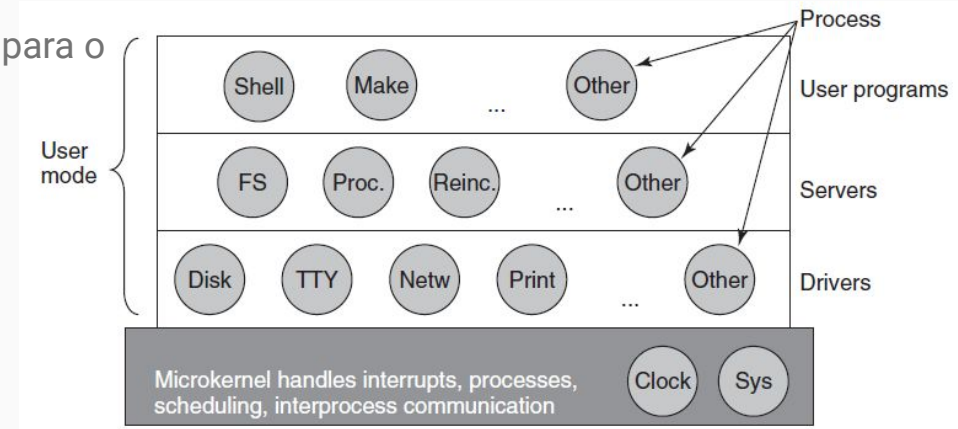
Micronúcleos

- Micronúcleo MINIX 3
  - 12.000 linhas de código C e 1400 linhas de Assembler
    - Código Assembler responsável por funções de baixo nível como capturar interrupções e chavear processos
    - Código C gerencia e escalona processos, lida com a comunicação entre eles e oferece um conjunto de +- 40 chamadas de núcleo
      - Isso permite que o resto do SO faça o seu trabalho
  - **Estrutura de processos**
    - **Núcleo**
    - **Drivers**
    - **Servidores**
    - **Programas de usuários**



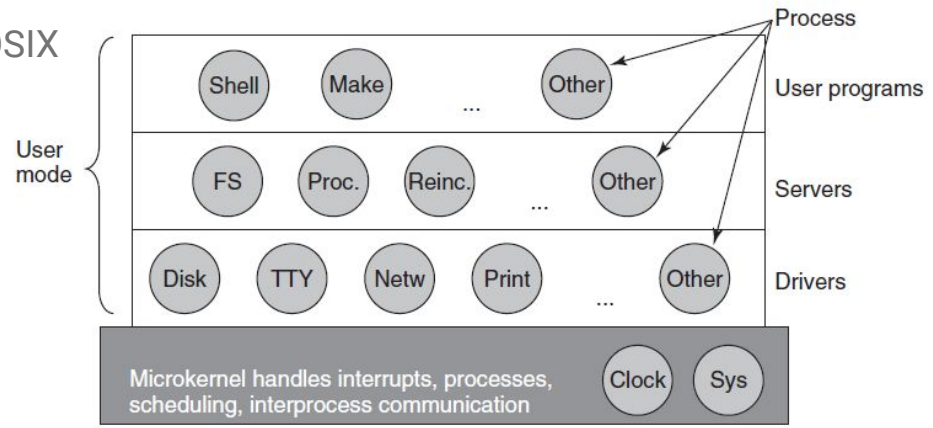
Micronúcleos

- **Estrutura de processos**
  - **Núcleo**
    - **Sys**: tratador de chamada de núcleo
    - **Clock**: driver de dispositivo de relógio
  - **Drivers**
    - Drivers constroem uma estrutura dizendo quais valores escrever (ou ler) para quais portas de E/S
    - Faz uma chamada de núcleo dizendo para o núcleo fazer a escrita (ou leitura) nessas portas



Micronúcleos

- **Estrutura de processos**
  - **Servidores**
    - Maior parte do SO
    - Servidores de arquivos (FS) gerenciam os sistemas de arquivos
    - Servidores de processos (proc) criam, destroem e gerenciam os processos
  - **Programas de usuários**
    - Obtém serviços de SO enviando mensagens curtas para os servidores
      - Solicita chamadas de sistema POSIX



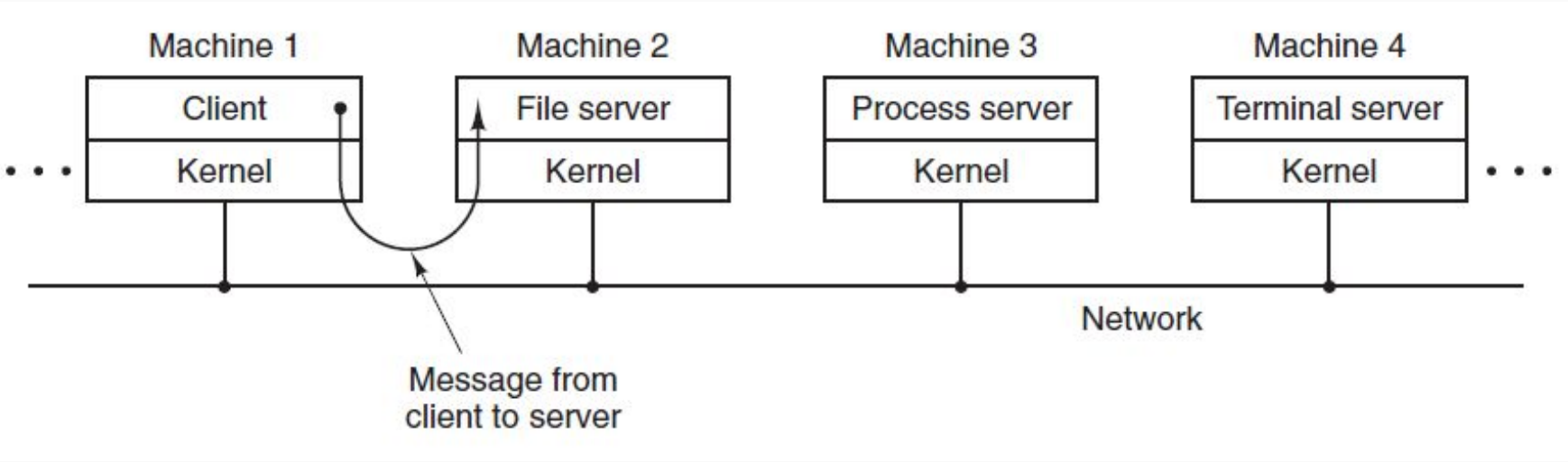
Micronúcleos

- **Servidor de reencarnação**
  - Conferir se outros servidores ou drivers estão funcionando corretamente
  - No caso de detecção de um servidor ou driver defeituoso, ele é automaticamente substituído sem qualquer intervenção do usuário
    - Sistema se regenera, podendo atingir alta confiabilidade
- Núcleo mínimo infere em colocar o **mecanismo** para fazer algo no núcleo, mas não a **política**
  - ex.: escalonamento de processos (prioridade de execução)
    - **Mecanismo** (núcleo): procurar pelo processo mais prioritário e executá-lo
    - **Política**: designar prioridades para os processos
      - Pode ser implementada por processos de modo usuário

Modelo Cliente-Servidor

- Variação da ideia do micronúcleo: **Cliente-servidor**
  - Distinguir duas classes de processos:
    - **Servidores:** prestam algum serviço
    - **Clientes:** usam esses serviços
  - Comunicação entre clientes e servidores via **troca de mensagens**
  - Processos clientes podem estar, ou não, na mesma máquina de processos servidores
    - Processos servidores podem estar recebendo mensagens de seus clientes via rede
    - Clientes não precisam saber se as mensagens são enviadas via rede ou entregues localmente em suas próprias máquinas

Modelo Cliente-Servidor



Máquinas Virtuais

- **VM/370 - 1979**
  - Na origem chamado de **CP/CMS**
  - Baseado na seguinte observação:
    - **Um sistema de compartilhamento de tempo fornece**
      1. Multiprogramação
      2. Uma máquina estendida com uma interface mais conveniente do que apenas o hardware
  - **A essência do VM/370 é separar completamente essas duas funções**

Máquinas Virtuais

- **VM/370 - 1979**
  - **Monitor de Máquina virtual - VMM**
    - Opera direto no hardware e realiza a multiprogramação
    - Fornece não uma, mas várias máquinas virtuais (**VMs)** para a camada seguinte
    - **Essas VMs não são máquinas estendidas, mas cópias exatas do hardware exposto, incluindo modo núcleo/usuário, E/S, interrupções e etc.**
      - Cada VM pode executar qualquer SO capaz de ser executado diretamente sobre o hardware

Exonúcleos

- Em vez de clonar a máquina real, outra estratégia é **dividi-la**
  - Ou seja,**dar a cada usuário um subconjunto dos recursos**
    - Desse modo uma VM pode obter blocos de disco de 0 a 1.023, a próxima pode ficar com os blocos 1.024 a 2.047 e assim por diante
  - Na camada de baixo, executando em modo núcleo, há um programa chamado **exonúcleo**
    - **Tarefa:** Alocar recursos às máquinas virtuais e então conferir tentativas de usá-las para assegurar que nenhuma máquina esteja tentando usar os recursos de outra pessoa

Exonúcleos

- **Cada VM no nível de usuário pode executar seu próprio SO**
  - Exceto que cada uma está restrita a usar apenas os recursos que ela pediu e foram alocados
  - **Vantagem:**
    - **Não precisa de uma camada de mapeamento (endereços de memória, por exemplo)**
      - Exonúcleo precisa apenas manter o registro de para qual máquina virtual foi atribuído qual recursos

Na próxima aula

- Processos
  - Introdução
  - Hierarquia de processos
  - Estados de processos
  - Implementação de processos
  - Modelando a multiprogramação
- Threads

Sugestão de leitura: seções 2.1 e 2.2

Referências

TANENBAUM, Andrew S e Bos, Herbert. Sistemas Operacionais Modernos. 4.ed. Pearson/Prentice-Hall. 2016.