INFORMÁTICA DE GESTÃO

Professor Doutor Ricardo Vardasca, PhD, ASIS, FRPS
ricardo.vardasca@islasantarem.pt

03 – Estruturas e Arquiteturas dos Sistemas Operativos

ISLA Santarém

Última aula...

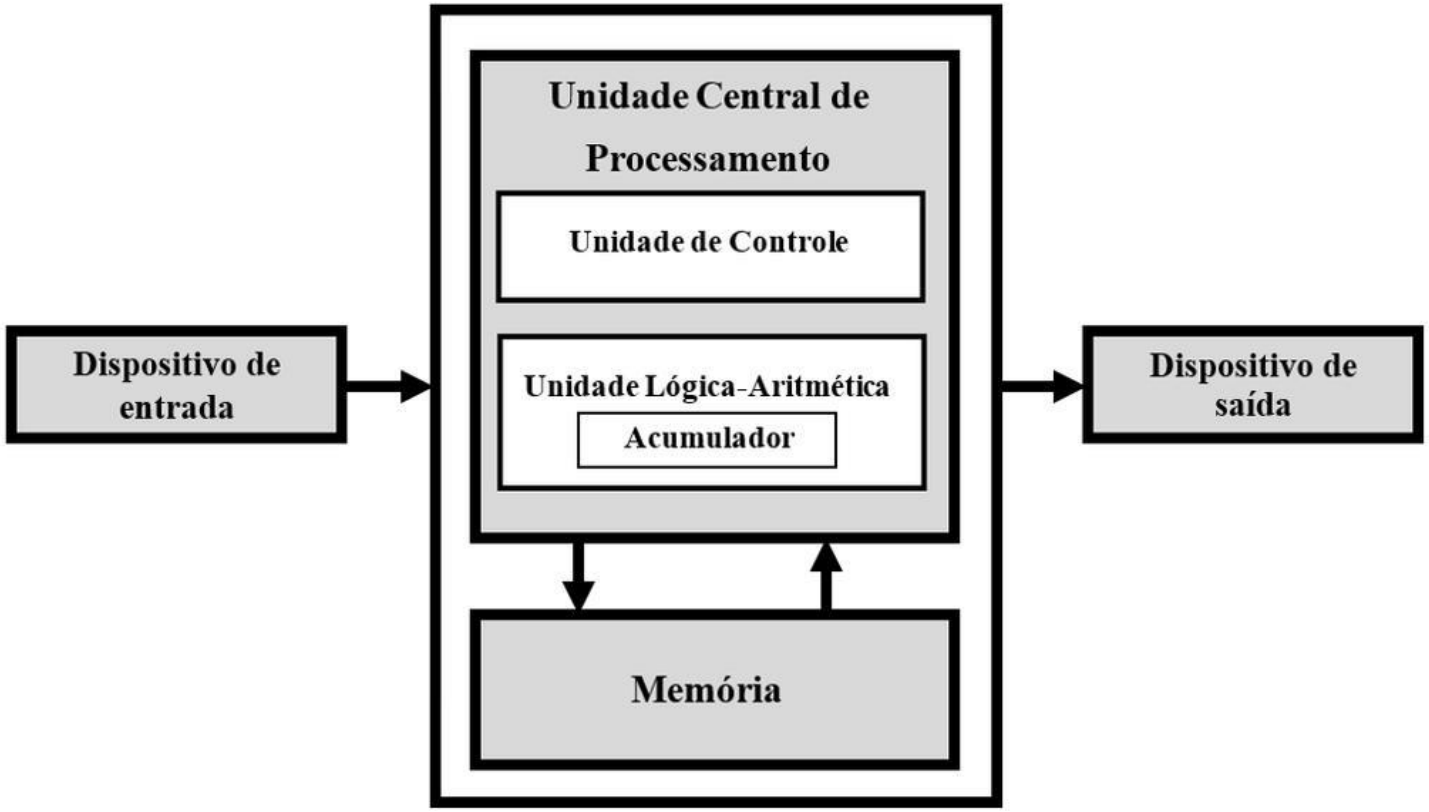
- Arquitectura do SO:
  - Elementos chave
  - Estrutura do SO
  - Decomposição funcional
  - Micro-núcleo
  - Estrutura monolítica
- Conceitos presentes em SO modernos
- Abstrações suportadas por um SO
- Necessidades de suporte e de proteção de hardware

Indice

- Estruturas de um SO
- Arquitecturas de SO:
  - Sistemas monolíticos, Sistemas em camadas, Sistemas micro-núcleo, Modelo Cliente-Servidor, Máquinas virtuais, *Containers,* Exonúcleo e Uninúcleo.
- Introdução a Processos
  - Threads vs Processos
  - Características dos Processos
  - Estados dos Processos

Estruturas de um SO

- Arquitetura de Von Newmann



Estruturas de um SO

- Serviços
- Interface com o utilizador
- Programas do sistema
- Desenho e implementação do SO
- Geração do SO
- Arranque do SO
- Chamadas do sistema

Estruturas de um SO (Serviços)

- Um conjunto de serviços do SO oferece funções que são úteis ao utilizador:
  - Interface com o utilizador – Quase todos os sistemas operacionais têm uma IU
    - Varia entre linha de comando (LC), interface gráfica com o utilizador (GUI), batch
  - Execução do programa – O sistema deve ser capaz de carregar um programa na memória e executar esse programa, terminar a execução, normal ou anormalmente (indicando erro)
  - Operações de E/S – Um programa em execução pode exigir E/S, que pode envolver um arquivo ou um dispositivo de E/S.
  - Manipulação do sistema de arquivos – O sistema de arquivos é de interesse particular. Obviamente, os programas precisam ler e gravar ficheiros e diretórios, criá-los e excluí-los, pesquisá-los, listar informação do ficheiro, gestão de permissões.
  - Comunicações – Processos podem trocar informações, no mesmo computador ou entre computadores de uma rede
    - As comunicações podem ser via memória partilhada ou por passagem de mensagens.

Estruturas de um SO (Serviços)

- Um conjunto de serviços do SO oferece funções que são úteis ao utilizador:
  - Deteção de erro – O SO precisa estar continuamente ciente dos erros possíveis
    - Pode ocorrer no CPU e no hardware de memória, em dispositivos de E/S, no programa do utilizador
    - Para cada tipo de erro, o SO deve tomar a ação apropriada para garantir a computação correta e coerente
    - Facilidades de depuração podem melhorar bastante as capacidades do utilizador e do programador para utilizar o sistema de modo eficiente
  - Outro conjunto de funções do SO existe para garantir a operação eficiente do próprio sistema por meio da partilha de recursos
    - Alocação de recursos – Quando vários utilizadores ou várias tarefas estão a ser executados simultaneamente, os recursos devem ser alocados a cada um deles
      - Muitos tipos de recursos – Alguns (como ciclos de CPU, memória principal e armazenamento de ficheiro) podem ter código de alocação especial, outros (como dispositivos de E/S) podem ter código geral de solicitação e libertação.

Estruturas de um SO (Serviços)

- Um conjunto de serviços do SO oferece funções que são úteis ao utilizador:
  - Outro conjunto de funções do SO existe para garantir a operação eficiente do próprio sistema por meio da partilha de recursos
    - Proteção e segurança – Os proprietários da informação armazenada num sistema de computador multiutilizador ou em rede podem querer controlar o uso dessa informação; processos concorrentes não devem interferir uns com os outros
    - A proteção envolve garantir que todo o acesso aos recursos do sistema é controlado
    - Segurança do sistema contra estranhos requer autenticação do utilizador, estende-se para defender dispositivos de E/S externos contra tentativas de acesso inválidas
    - Se um sistema tiver que ser protegido e seguro, deve ter instituídas nele precauções. Uma cadeia é tão forte quanto seu elo mais fraco.

Estruturas de um SO (IU)

- Interface amigável
  - Normalmente rato, teclado e monitor
  - Ícones representam ficheiros, programas, ações, etc.
  - Diversos botões do rato sobre objetos da interface causam diversas ações: fornecer informações, opções, executar função, abrir diretório (conhecido como pasta)
  - Inventado na Xerox PARC
- Muitos sistemas agora incluem LC e GUI
  - Microsoft Windows é GUI com shell de “comando” LC
  - Apple Mac OS X como interface GUI “Aqua” com kernel do UNIX (BSD) por baixo e shells disponíveis
  - Solaris é LC com interfaces GUI opcionais (Java Desktop, KDE)

Estruturas de um SO (Programas de sistema)

- Programas do sistema que oferecem um ambiente conveniente para desenvolvimento e execução do programa. Eles podem ser divididos em:
  - Manipulação de um ficheiro
  - Informação de estado
  - Modificação de um ficheiro
  - Suporte a linguagens de programação
  - Carregar e execução do programa
  - Comunicações (Mensagens)
  - Programas de aplicação
- A visão do SO pela maioria dos utilizadores é definida por programas do sistema, e não pelas chamadas do sistema reais

Estruturas de um SO (Programas de sistema)

- Fornecem um ambiente conveniente para desenvolvimento e execução de programas
  - Alguns deles são simplesmente IU para chamadas ao sistema; outros são muito mais complexos
- Gestão de ficheiros – Criam, eliminam, copiam, renomeiam, imprimem, listam e geralmente manipulam ficheiros e diretórios
- Informação do estado
  - Alguns pedem informações ao sistema – data, hora, quantidade de memória disponível, espaço em disco, número de utilizadores
  - Outros oferecem informações detalhadas de desempenho, logs e depuração
  - Normalmente, esses programas formatam e imprimem a saída no terminal ou em outros dispositivos de saída
  - Alguns sistemas implementam um registro – usado para armazenar e recuperar informações de configuração

Estruturas de um SO (Programas de sistema)

- Modificação de ficheiros
  - Editores de texto para criar e modificar ficheiros
  - Comandos especiais para pesquisar conteúdo de ficheiros ou realizar transformações do texto
- Suporte a linguagens de programação – Compiladores, assembladores, depuradores e interpretadores às vezes fornecidos
- Carregamento e execução de programa – Carregadores absolutos, carregadores relocáveis, editores de vínculo e carregadores de sobreposição, sistemas de depuração para linguagem de alto nível e de máquina
- Comunicações – Oferecem o mecanismo para criar ligações virtuais entre processos, utilizadores e sistemas computacionais
  - Permite que os utilizadores enviem mensagens para os ecrãs uns dos outros, naveguem na Web, enviem mensagens de e-mail, efetuem inicio de sessão remotamente, transfiram ficheiros de uma máquina para outra

Estruturas de um SO (Desenho e Implementação)

- Desenho e Implementação do SO não “solucionável”, mas algumas técnicas provaram ser bem sucedidas
- Estrutura interna de diferentes SOs pode variar bastante
- Começa-se definindo objetivos e especificações
- São afetados pela escolha do hardware, tipo do sistema
- Objetivos:
  - Do utilizador – o SO deve ser conveniente de usar, fácil de aprender, confiável, seguro e rápido
  - Do sistema – SO deve ser fácil de desenhar, implementar e manter, além de ser flexível, confiável, livre de erros e eficiente

Estruturas de um SO (Desenho e Implementação)

- Princípio importante para separar
  - Política: O que será feito?
  - Mecanismo: Como fazer isso?
- Mecanismos determinam como fazer algo, políticas decidem o que será feito
  - A separação entre política e mecanismo é um princípio muito importante, permite o máximo de flexibilidade se decisões políticas tiverem que ser alteradas mais tarde

Estruturas de um SO (Geração)

- Os SO são projetados para executar em qualquer uma de uma classe estabelecida de máquinas; o sistema precisa ser configurado para cada ponto de computador específico
- O programa **SYSGEN** obtém informações referentes à configuração específica do sistema de hardware
- **Booting** – iniciar um computador carregando o kernel (núcleo)
- Programa de **bootstrap** – código armazenado na ROM que é capaz de localizar o kernel, carrega-lo na memória e iniciar sua execução

Estruturas de um SO (Arranque)

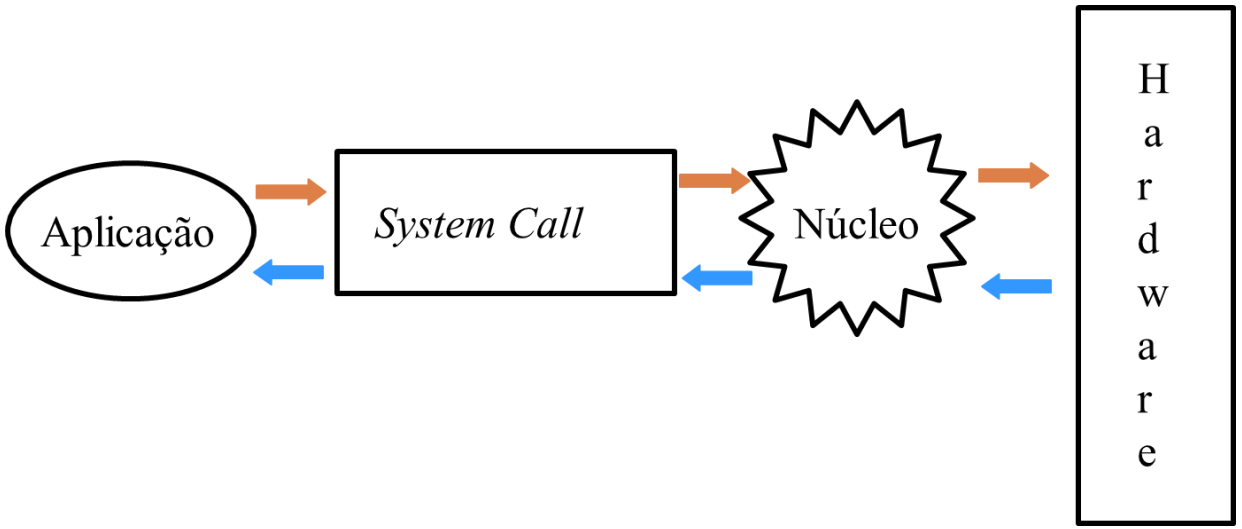
- O SO precisa estar disponível ao hardware, para que esse mesmo hardware possa iniciá-lo
  - Pequeno trecho de código – carregador de *bootstrap*, localiza o *kernel* (núcleo), carrega-o na memória e o inicia
  - Às vezes, trata-se de um processo em duas etapas, onde o bloco de boot no local fixo carrega o carregador de *bootstrap*
  - Quando o sistema é inicializado, a execução começa num local fixo da memória
    - *Firmware* usado para manter o código inicial de *boot* (arranque)

Chamadas ao Sistema – System Calls

- Interface de programação para os serviços fornecidos pelo SO
- Normalmente, escritas em uma linguagem de alto nível (C ou C++)
- Acedidas principalmente pelos programas por meio de uma Application Program Interface (API) de alto nível, ao invés do uso da chamada direta do sistema
- As três APIs mais comuns são: **Win32 API** para Windows, **POSIX API** para sistemas baseados em POSIX (incluindo praticamente todas as versões do UNIX, Linux e Mac OS X) e **Java API** para a Java Virtual Machine (JVM)

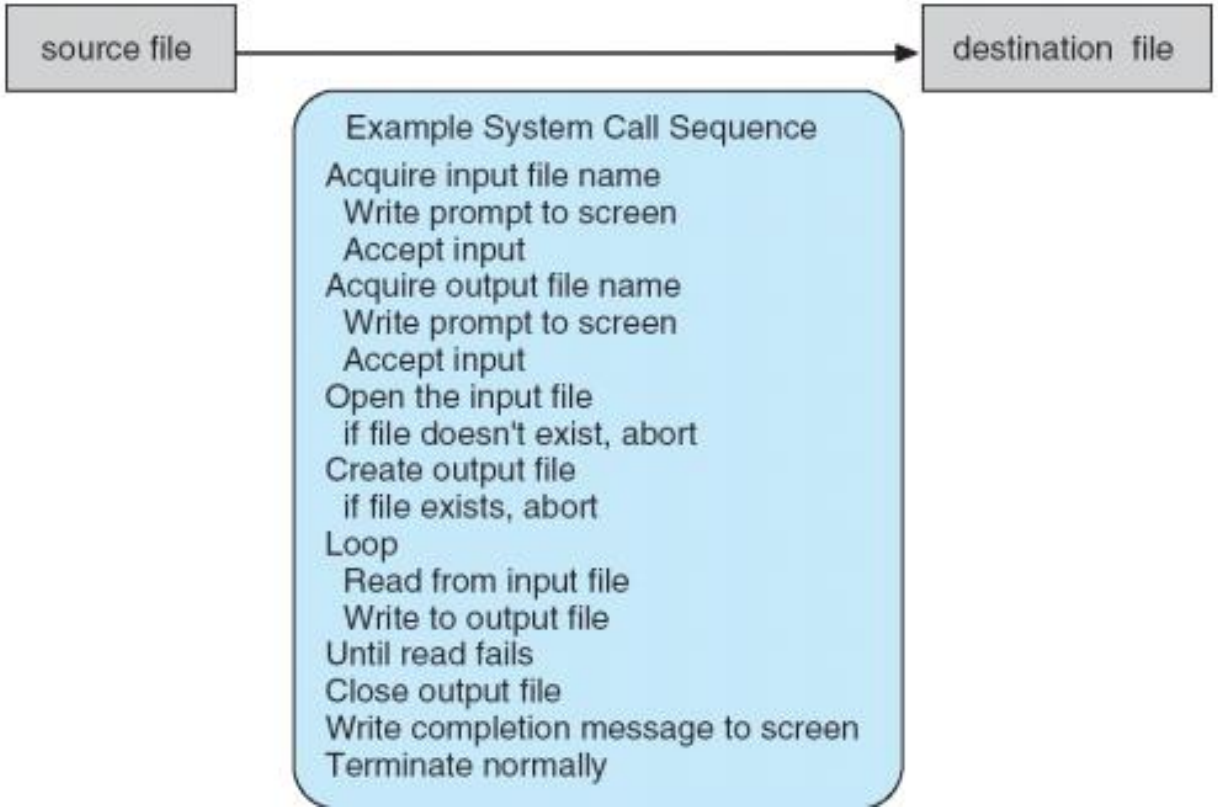
Chamadas ao Sistema – System Calls

- Mecanismo de proteção do núcleo do sistema e de acesso aos seus serviços.
- O utilizador (ou aplicação), quando deseja solicitar algum serviço do sistema, realiza uma chamada a uma de suas rotinas (ou serviços) através das ***system calls*** (chamadas do sistema).



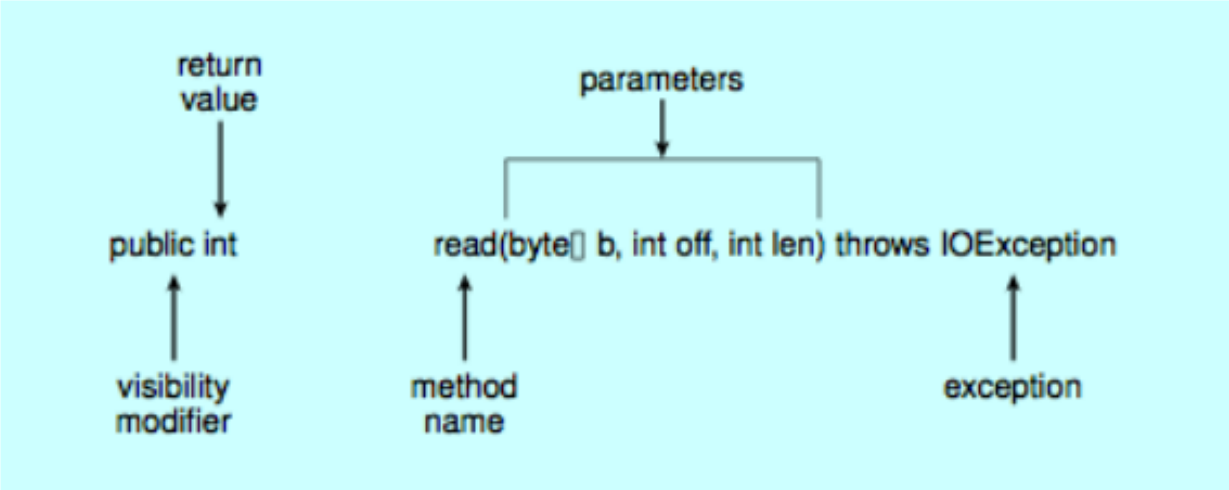
Chamadas ao Sistema – System Calls

- Sequência de chamada ao sistema para copiar o conteúdo de um ficheiro para outro



Chamadas ao Sistema – System Calls

Considere o comando Java read()



byte[] b – o buffer no qual os dados são lidos

int off – o offset inicial em b onde os dados são lidos

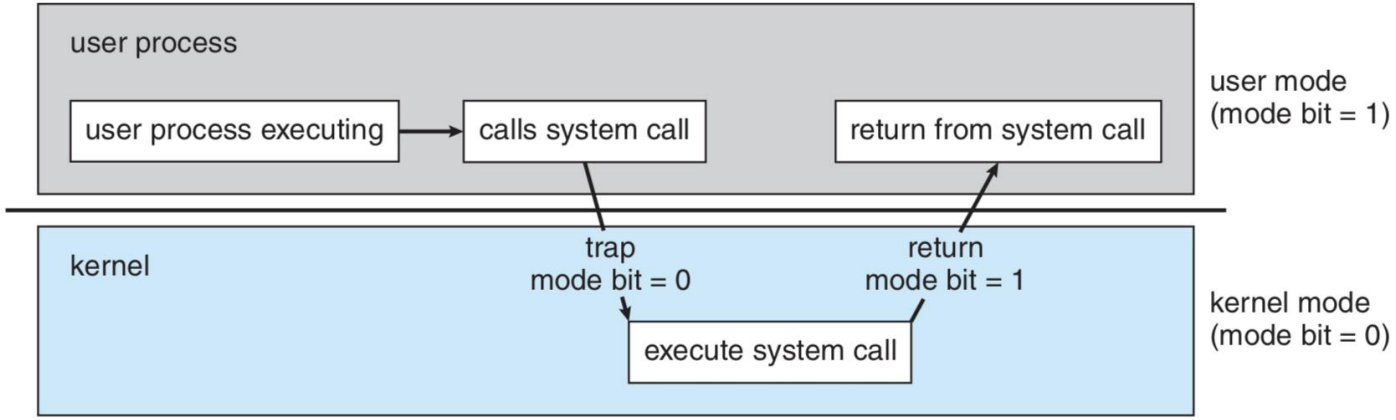
int len – o número máximo de bytes a serem lidos

Chamadas ao Sistema – System Calls

- Tipos de chamadas do sistema:
  - Controle de processo
  - Gestão de ficheiros
  - Gestão de dispositivos
  - Manutenção de informações
  - Comunicações

Modo Kernel e Utilizador

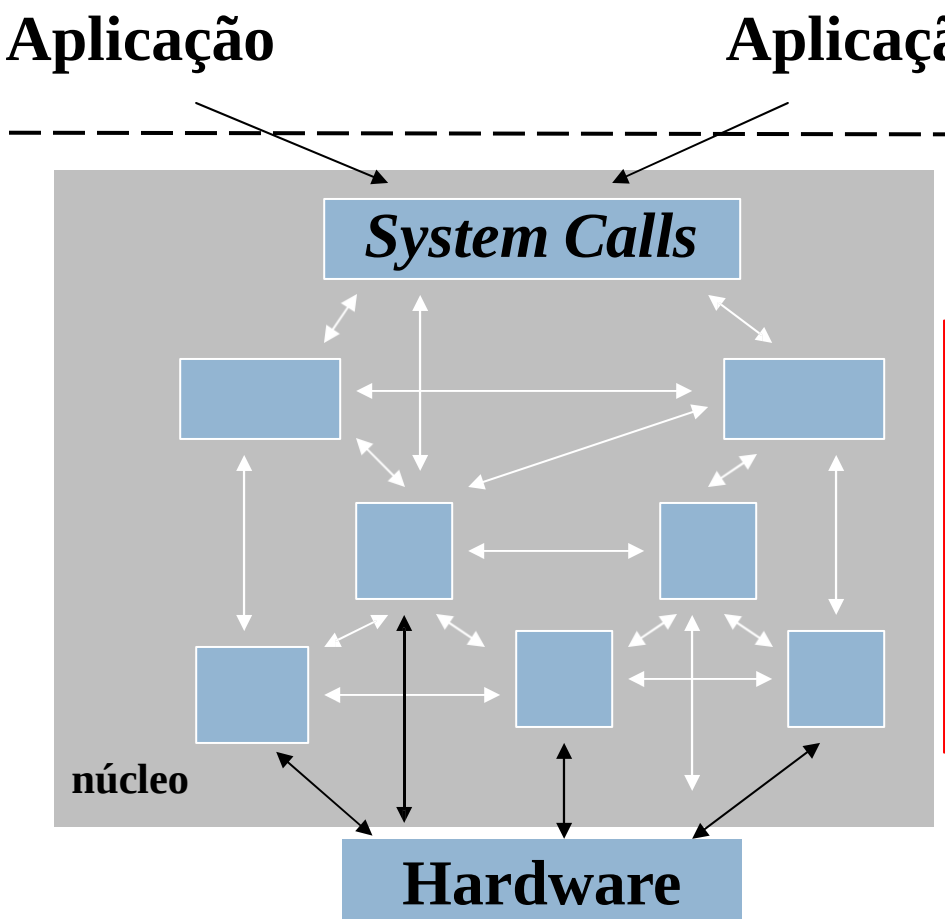
- SO executa em Modo Kernel, supervisor ou núcleo → protege o hardware da ação direta do utilizador.
- Os demais programas executam em modo utilizador e fazem chamadas ao kernel para terem acesso aos dispositivos.



Sistema Monolítico

- Nesta abordagem o SO inteiro é executado como um único programa no modo núcleo.
- A organização mais comum é aquela que estrutura o sistema como um conjunto de rotinas que podem interagir livremente umas com as outras.
- Pode ser comparada com uma aplicação formada por vários procedimentos que são compilados separadamente e depois ligados, formando um grande e único programa executável.
  - Grande desempenho
  - Uma falha pode paralisar o núcleo. O sistema pode parar por causa de um erro.
  - As interfaces e níveis de funcionamento não estão bem separados nem unificados. O excesso de liberdade torna o sistema vulnerável
  - Ex: Linux e FreeBSD

Sistema Monolítico



1. Um programa principal que invoca a rotina do serviço requerido.
2. Um conjunto de rotinas de serviço que executam as chamadas ao sistema.
3. Um conjunto de rotinas de utilidade que auxiliam as rotinas de serviço.

Sistema em Camadas

- Divide o SO em sistemas sobrepostos. Cada módulo oferece um conjunto de funções que pode ser usado por outros módulos.
- A vantagem da estruturação em camadas é isolar o SO, facilitando sua alteração e depuração, além de criar uma hierarquia de níveis de modos, protegendo as camadas mais internas.
- A sobreposição de várias camadas de software faz com que cada pedido de uma aplicação demore mais tempo a chegar ao dispositivo periférico ou recurso a ser acedido, prejudicando o desempenho do sistema.
- Não faz sentido dividir as funcionalidades de um núcleo do SO em camadas horizontais de abstração crescente, uma vez que essas funcionalidades são interdependentes, embora acedam muitas vezes a recursos distintos.

Sistema em Camadas

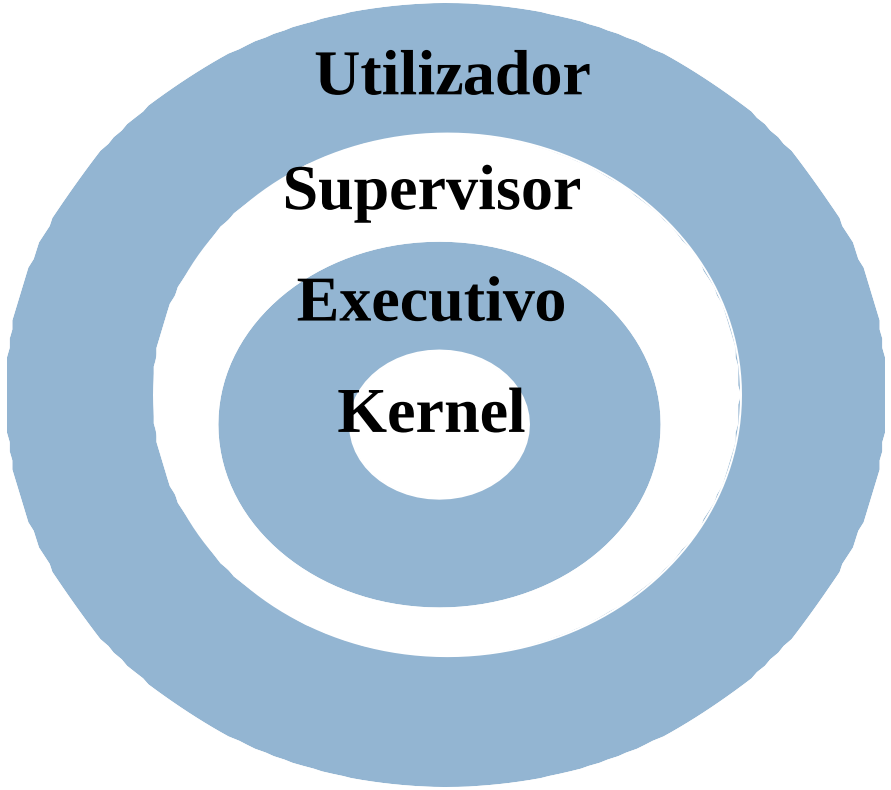
Camada	Função
5	Operador
4	Programa de utilizador
3	Gestão de E/S
2	Comunicação operador-processo
1	Gestão de memória
0	Alocação de processador e multiprogramação

Estrutura de um SO por camadas

Sistema em Camadas (anéis)

- Anéis mais internos são mais privilegiados que os externos;
- Procedimentos de anéis externos executavam chamadas ao sistema para utilizar os serviços dos anéis internos;
- Proteção dos segmentos de memória.

Sistema em Camadas (anéis)



Sistema Multics
VMS

+No sistema MULTICS VMS as camadas inferiores são as mais privilegiadas.

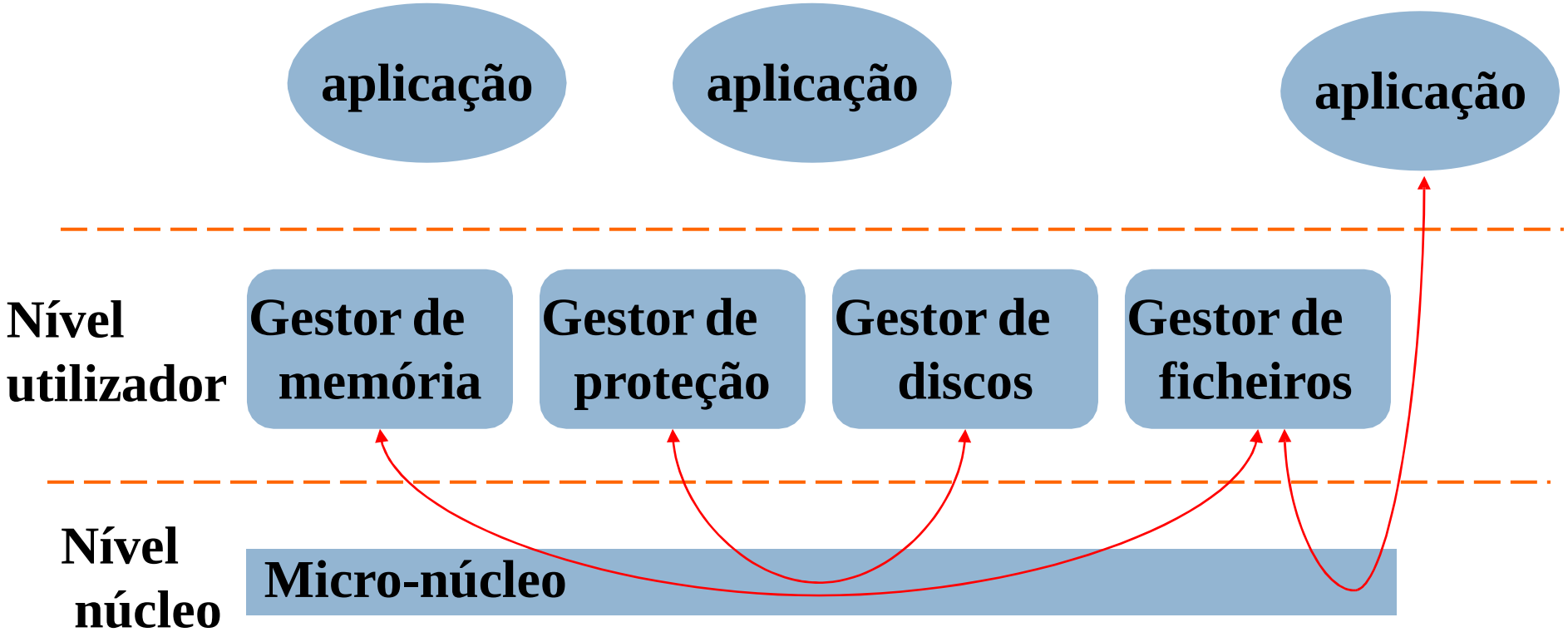
Sistemas micro-núcleo

- Uma tendência dos SO é tornar o núcleo o menor e mais simples possível e para implementar esta ideia o sistema é dividido em processos.
- Sempre que uma aplicação deseja algum serviço, esta solicita-o ao processo responsável, assim, a aplicação que solicita um serviço é chamada de cliente e o processo que responde a solicitação é chamado de servidor.
- Permite que os servidores executem em modo utilizador.
- Apenas o núcleo do sistema, responsável pela comunicação entre clientes e servidores, executa no modo kernel.
- O SO passa a ser de mais fácil manutenção.
- Não importa se o serviço está a ser processado num único processador, em múltiplos processadores ou num sistema distribuído.

Sistemas micro-núcleo

- Um ambiente distribuído permite que um cliente solicite um serviço e a resposta seja processada remotamente.
- A implementação é difícil e normalmente é implementado uma combinação do modelo de camadas com o cliente-servidor.
- O núcleo do sistema passa a incorporar o escalonamento e gestão de memória além das funções dos *device drivers*.

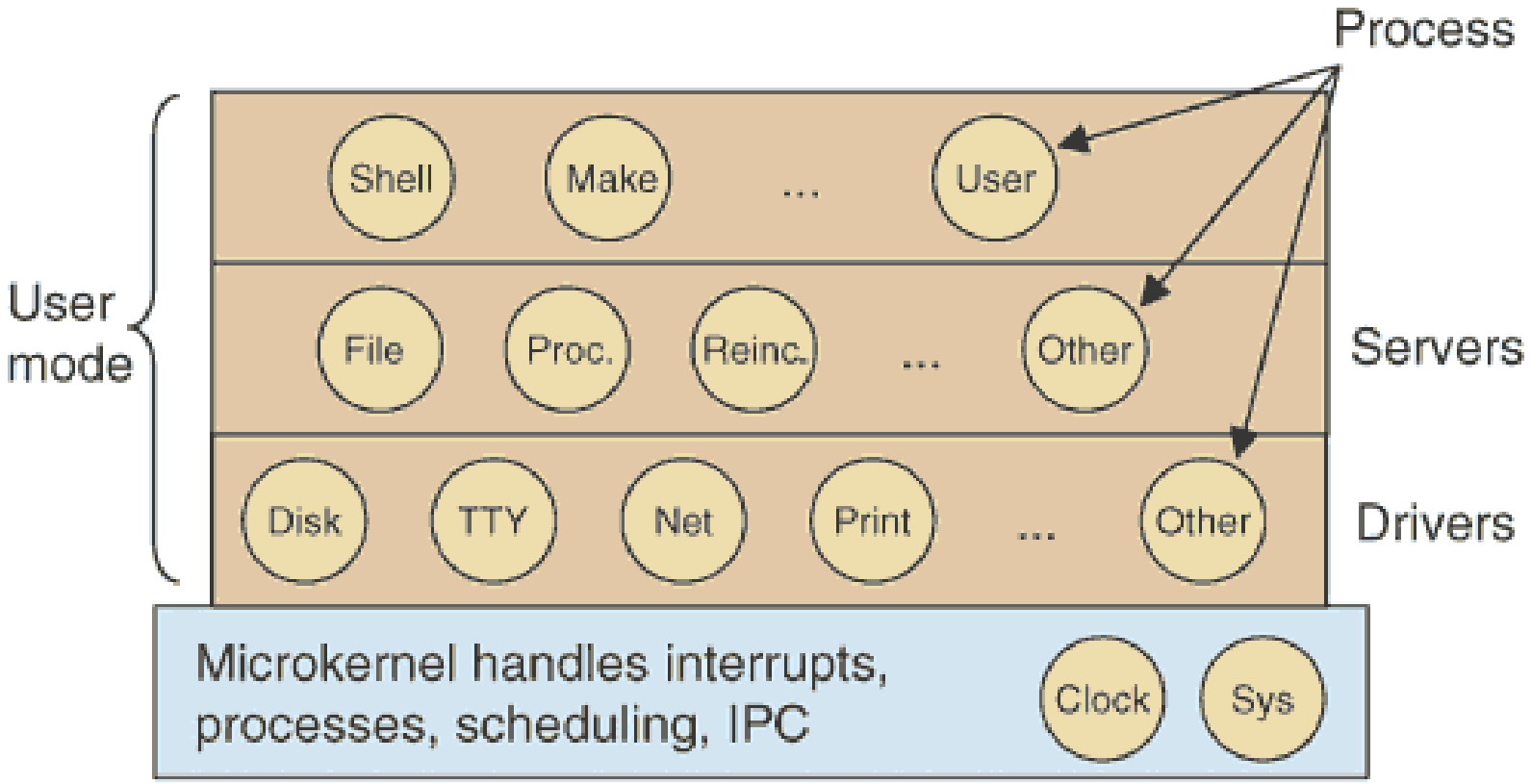
Sistemas micro-núcleo



Sistemas micro-núcleo

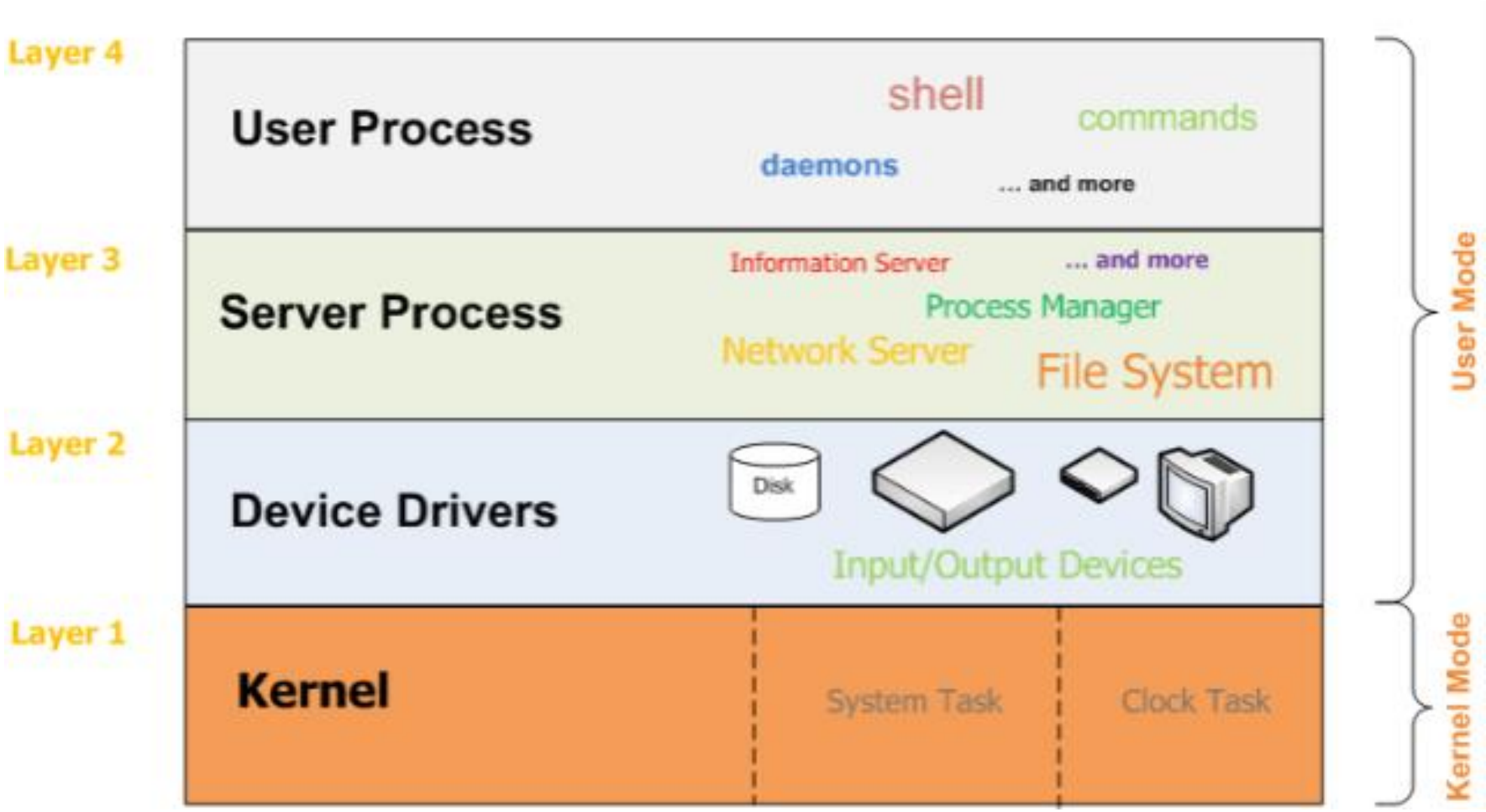
- A ideia básica por trás do micro-núcleo é alcançar alta confiabilidade por meio da divisão do SO em módulos pequenos, bem definidos, e apenas um desses módulos (micro-núcleo) ser executado no modo núcleo e o restante ser executado como processos de utilizador.
- Quando há execução de cada *driver* de dispositivo e cada sistema de ficheiros como processo separado, um erro em qualquer um deles pode bloquear aquele componente, mas não bloqueia o sistema inteiro.
- Os SO mícro-núcleos são comuns em aplicações de tempo real, industriais, aviação e militares, que são cruciais e tem requisitos de confiabilidade muito altos.
- Ex: Integrity, K42, PikeOS, QNX, Symbian e MINIX3.

Sistemas micro-núcleo



Estrutura do sistema MINIX3

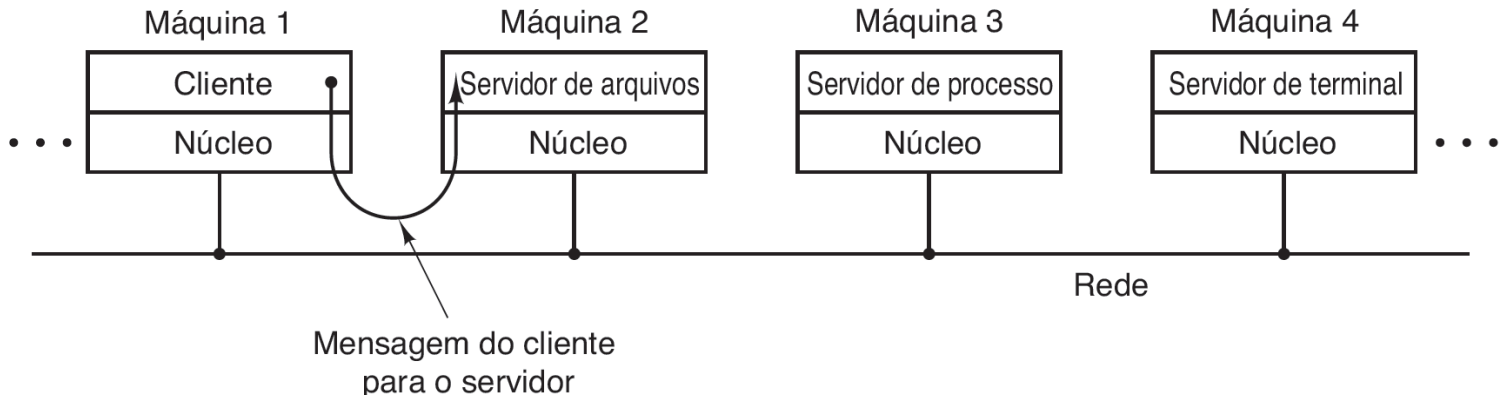
Sistemas micro-núcleo



Estrutura do sistema MINIX3

Modelo Cliente-Servidor

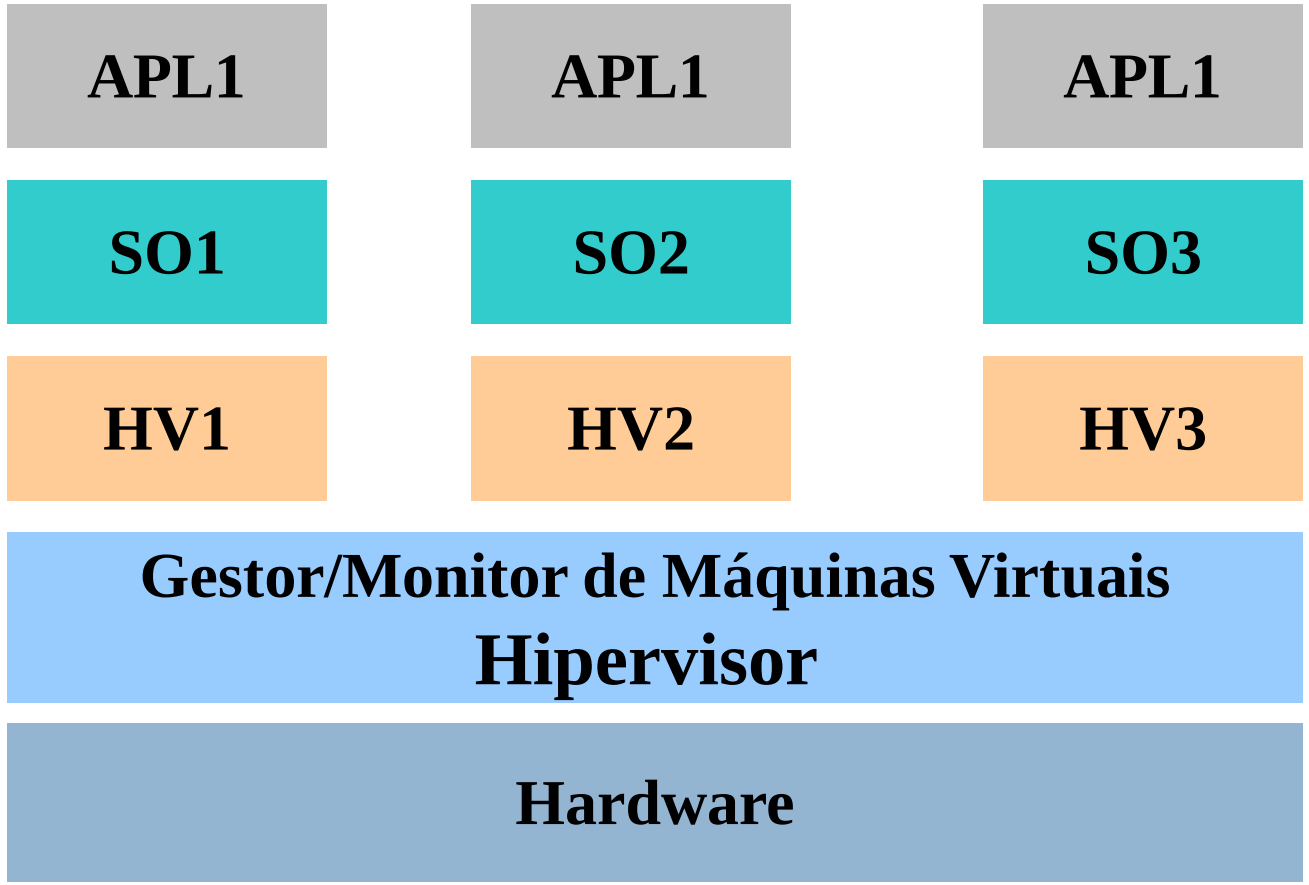
- Uma variação da ideia do micro-núcleo é distinguir entre duas classes de processos, os servidores, que prestam serviço, e os clientes, que usam os serviços.
- Frequentemente a camada inferior é o micro-núcleo, mas não de forma obrigatória. A essência é a presença de processos clientes e servidores.



Máquina Virtual

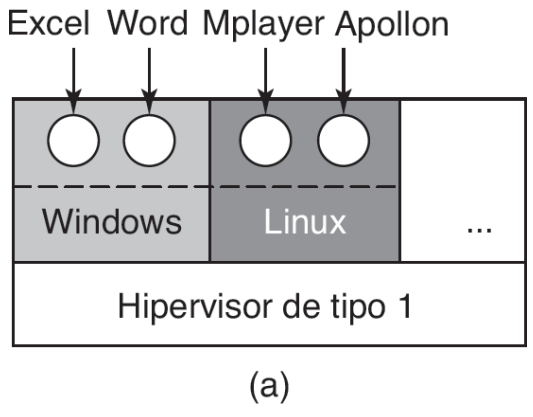
- Máquinas virtuais não são máquinas estendidas com ficheiros e outras características convenientes.
- São cópias exatas do hardware, inclusive com modos núcleo/utilizador, E/S, interrupções e tudo o que uma máquina real tem.
- Cada MV pode executar qualquer SO capaz de ser executado diretamente sobre o hardware.
- Diferentes MVs podem executar diferentes SOs.

Máquina Virtual

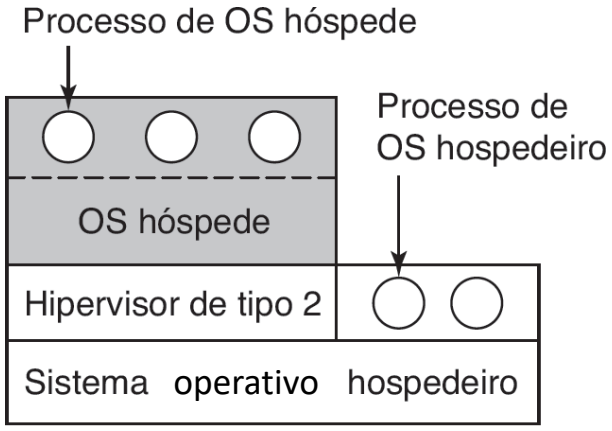


Máquina Virtual

- Hipervisor do tipo 1(a) são executados diretamente no hardware.
  - Ex: Eucalyptus
- Hipervisor do tipo 2(b) são executados como aplicações na camada superior do SO
  - Ex: VMware, VirtualBox



(a)

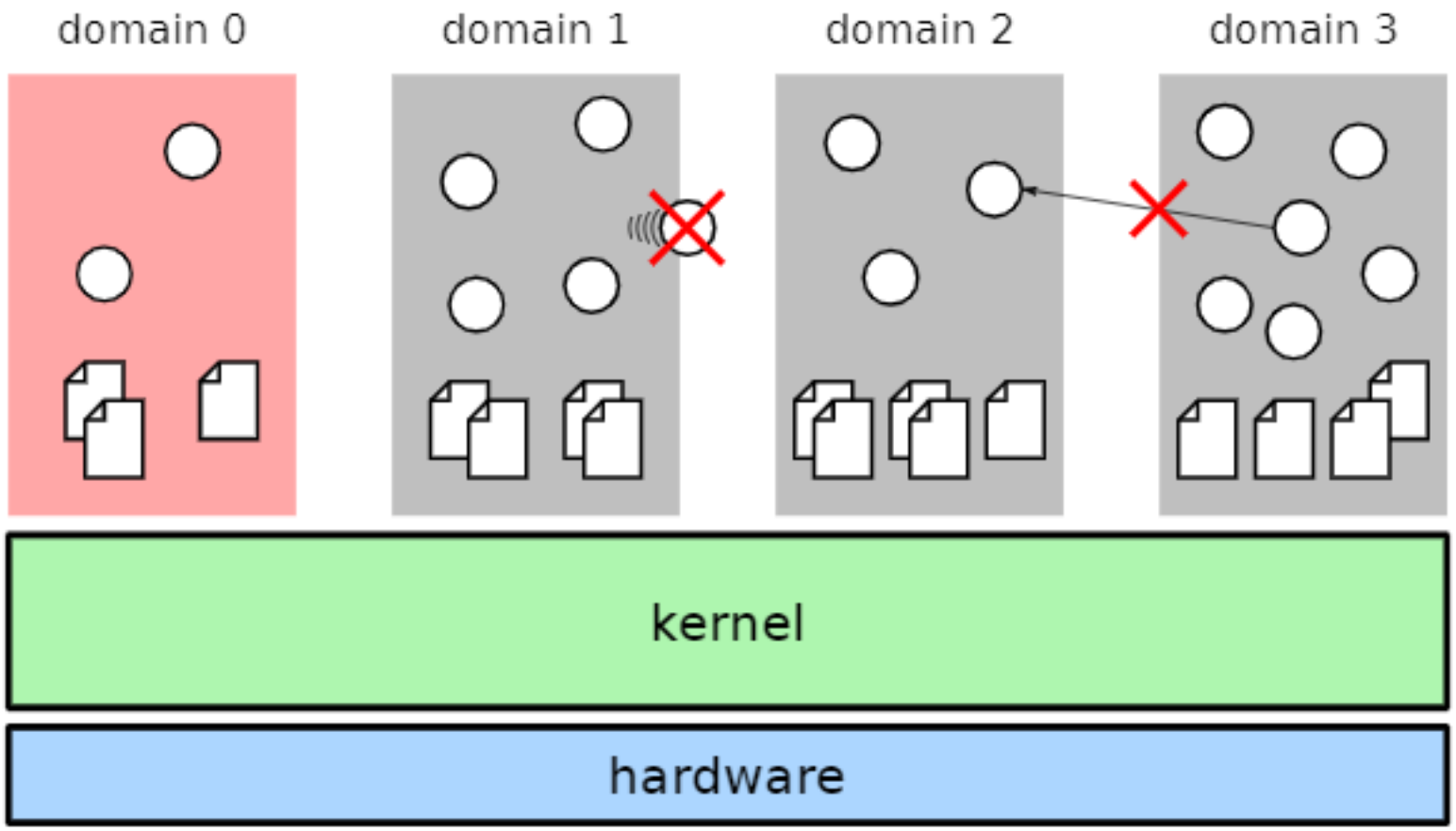


(b)

Containers

- Outra forma de isolar aplicações ou subsistemas num SO consiste na virtualização do espaço de utilizador (userspace).
- O espaço do utilizador é dividido em áreas isoladas denominadas domínios ou *containers*. A cada domínio é alocada uma parcela dos recursos do SO, como memória, tempo de processador e espaço em disco.
- Para garantir o isolamento entre os domínios, cada domínio tem sua própria interface de rede virtual e, portanto, seu próprio endereço de rede. Além disso, na maioria das implementações cada domínio tem seu próprio espaço de nomes para os identificadores de utilizadores, processos e primitivas de comunicação.
- Processos num mesmo domínio podem interagir entre si normalmente, mas processos em domínios distintos não podem ver ou interagir entre si, estão isolados.

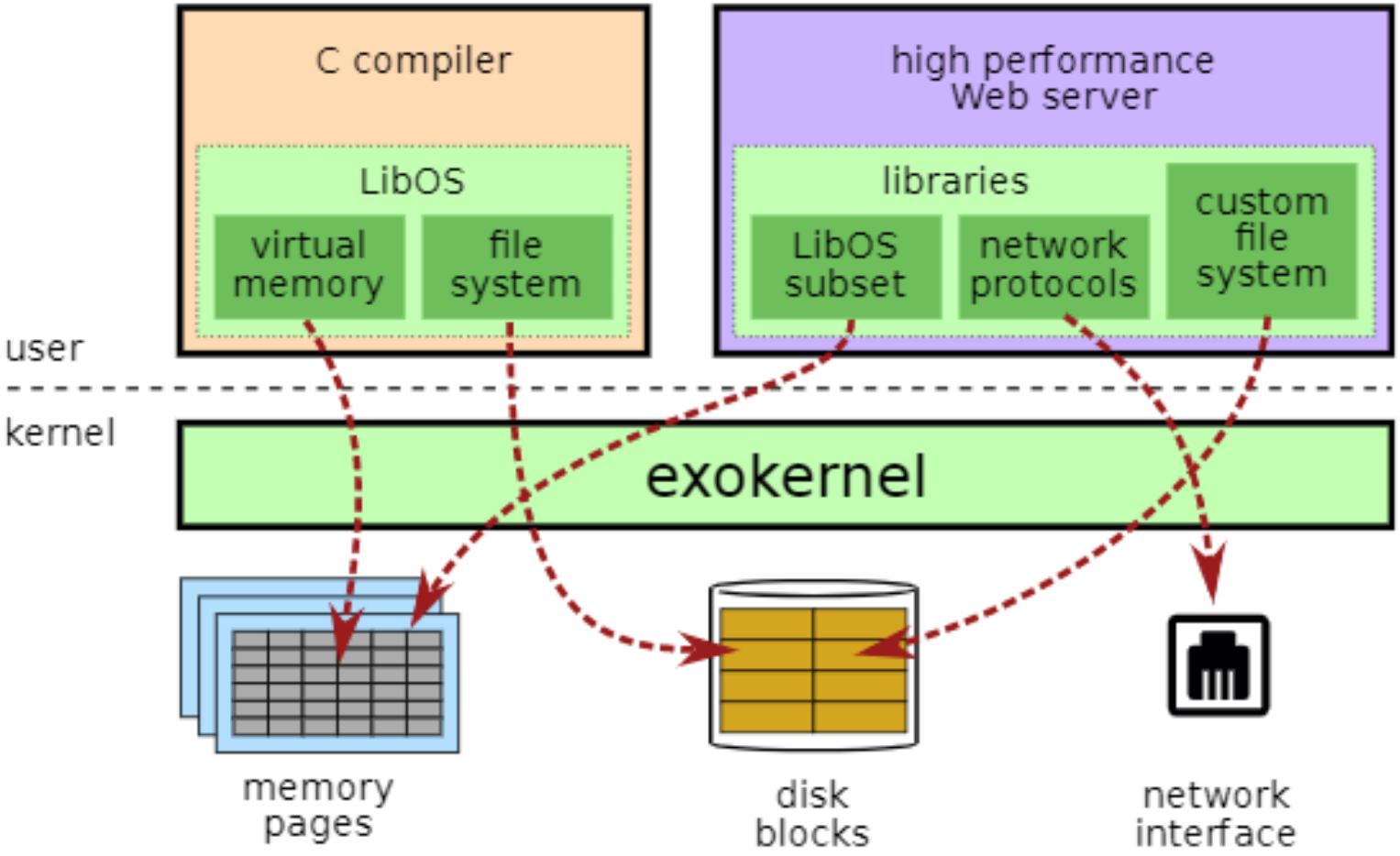
Containers



Exonúcleo

- Em vez de clonar a máquina real, como nas MVs, outra estratégia é dividi-la ou dar a cada utilizador um subconjunto de recursos.
  - Assim uma MV pode ter os blocos de 0 a 1023 do disco e outra de 1024 a 2047 e assim por diante.
- Na camada inferior há um programa (exonúcleo) responsável por alocar recursos às MVs e verificar as tentativas de utilização para assegurar que uma MV não esteja a utilizar os recursos de outra.
- Cada MV pode utilizar seu próprio SO mas somente com os recursos que pediu e que lhe foram alocados.
- Tem vantagem de separar, com menor custo, a multiprogramação (no exonúcleo) do código do SO do utilizador, visto que o exonúcleo mantém as MVs umas fora do alcance das outras.

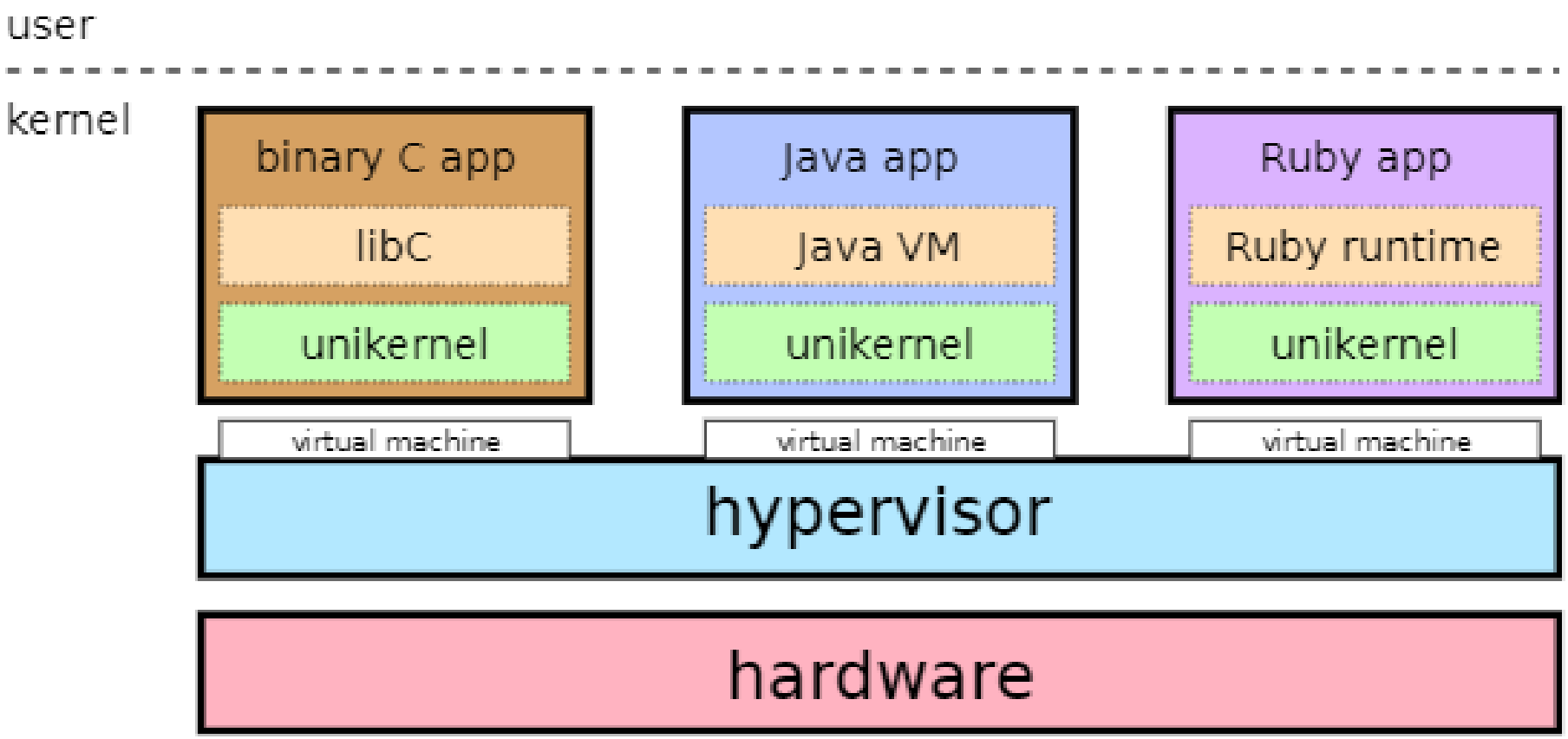
Exonúcleo


C compiler
LibOS
virtual memory
file system
high performance Web server
libraries
LibOS subset
network protocols
custom file system
user
kernel
exokernel
memory pages
disk blocks
network interface

Uninúcleo

- Um núcleo de SO, as bibliotecas e uma aplicação são compilados e ligados entre si, formando um bloco monolítico de código, que executa num único espaço de endereçamento, em modo privilegiado. Dessa forma, o custo da transição aplicação/núcleo nas chamadas de sistema diminui muito, gerando um forte ganho de desempenho.
- Incluí no código final somente os componentes necessários para suportar a aplicação-alvo e os *drivers* necessários para aceder ao *hardware* alvo, levando a um sistema compacto, que pode ser lançado rapidamente.
- São muito usados na Internet das Coisas (IoT)
- Exs: TinyOS, Osv e MirageOS

Uninúcleo


user
kernel
binary C app
libC
unikernel
Java app
Java VM
unikernel
Ruby app
Ruby runtime
unikernel
virtual machine
virtual machine
virtual machine
hypervisor
hardware

Processo

- Um processo pode ser entendido como um programa em execução. Para que a execução simultânea de diversos programas ocorra sem problemas é necessário que todas as informações do processo interrompido sejam guardadas para que quando este voltar a ser executado não lhe falte nenhuma informação e possa continuar a execução exatamente no ponto onde estava quando foi interrompido.
- O conceito de processo pode ser definido como sendo o conjunto de informações necessárias para que o SO implemente a concorrência de programas.

Processo vs. Programa

- **Programa** - entidade estática e permanente
  - Composto por uma sequência de instruções: passivo sob o ponto de vista do SO
- **Processo** - entidade dinâmica
  - Altera seu estado à medida que a sua execução avança;
  - Trata-se de uma abstração que representa um programa em execução;
  - É composto por:
    - Contexto de hardware
    - Contexto de software
    - Espaço de endereçamento

Processo

- Características/Propriedades de um processo:
  - Um programa pode gerar (criar) vários processos
  - Um processo tem duas partes:
    - Ativa - fluxo de controle
    - Passiva - espaço de endereçamento (memória, registos, ficheiros)
  - O mesmo programa executado por dois utilizadores gera dois processos
  - Um processo tem execução sequencial
  - O resultado da execução de um processo é independente da velocidade do processador em que for executado

Ambiente do processo

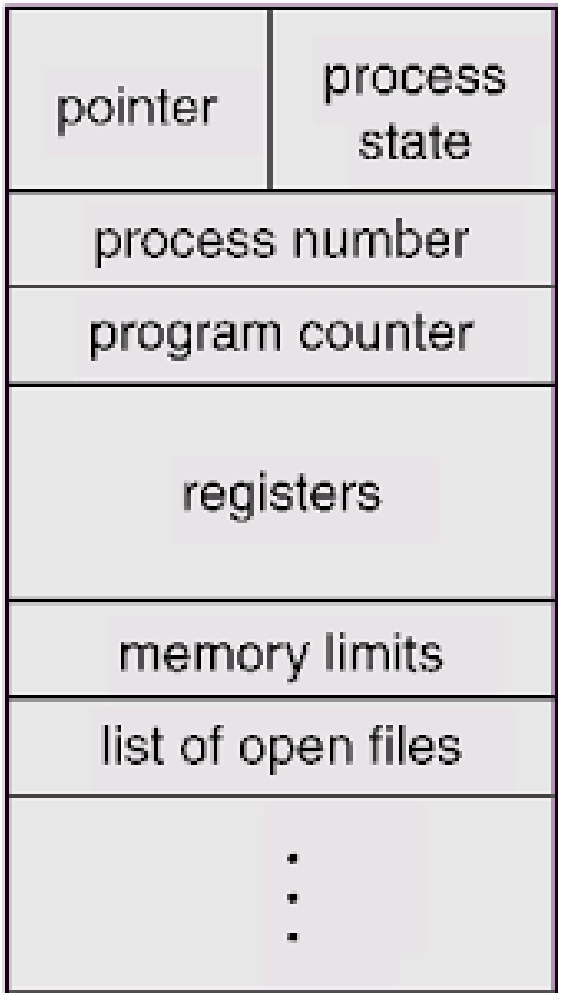
- Um processo também pode ser definido em termos do ambiente onde um programa é executado o que inclui a quantidade de recursos que o processo pode utilizar tais como espaço de endereçamento da memória principal, tempo de processador e área em disco.
- O resultado da execução de um mesmo programa pode variar em função do ambiente em que ele é executado devido aos recursos disponíveis no momento de sua execução.
- Caso um processo necessite de recursos não disponíveis o SO irá interromper a sua execução por falta de recursos disponíveis.

Contexto do processo

- **Identificação** - Cada processo possui um identificador (PID) que é representado por um número. O processo é identificado pelo SO através do PID. O processo também possui a identificação do utilizador ou processo que o criou (owner). Cada utilizador possui uma identificação no sistema (UID), atribuída ao processo no momento de sua criação.
- **Quotas** - As quotas são o limite de cada recurso do sistema que um processo pode alocar. Caso uma quota seja o insuficiente o processo pode ser executado lentamente, ser interrompido ou até não ser executado. São exemplos de quotas:
  - Numero máximo de arquivos abertos
  - Tamanho máximo da memória principal e secundária que pode ser alocada
  - Número máximo de operações de E/S pendentes
  - Tamanho máximo do buffer para operações de E/S
  - Número máximo de processos
- **Privilégios** - Os privilégios ou direitos definem as ações que um processo pode fazer em relação a ele mesmo, aos demais processos e ao SO.

Bloco de contolo do processo

- O processo é implementado pelo SO através de uma estrutura de dados chamada bloco de controle do processo (PCB). O SO, através do PCB, mantém todas as informações sobre o contexto de hardware, contexto de software e espaço de endereçamento. Cada processo possui seu PCB que mantêm todas as suas informações.
- OS PCBs de todos os processos são mantidos na memória principal em uma área exclusiva do SO. O tamanho desta área de memória é controlado por parâmetro no SO de acordo com o número máximo de processos que podem ser suportados pelo SO.



Estados de um processo

- **Criação (novo)** - Um processo é criado quando o SO cria um novo PCB, porem ainda não pode coloca-lo na lista de processos do estado pronto. Alguns SOs limitam o número de processos ativos em função de recursos disponíveis ou de desempenho. Por isso, os processos criados permanecem no estado de criação até que possam ser colocados no estado pronto quando ficam ativos e aguardando para serem executados
- **Terminado (encerrado)** - Um processo no estado de terminado não poderá mais ter nenhum programa executado no seu contexto nem ter nenhum recurso alocado. Porém o SO ainda mantém informações do processo em memória. O processo no estado terminado não é mais considerado ativo
  - Um processo pode passar para o estado terminado por razões como:
    - Término normal de execução
    - Eliminação por um outro processo
    - Eliminação forçada por ausência de recursos disponíveis no sistema

Estados de um processo

- Um processo ativo pode encontrar-se em um de três diferentes estados:
  - **Execução** - Quando o processo está em execução pela CPU
  - **Pronto** - Quando o processo aguarda para ser executado. O SO determina a ordem e o critério para que um processo em estado de pronto possa ter acesso ao processador.
  - **Espera** - Quando o processo aguarda um evento externo ou por algum recurso para prosseguir seu processamento. Por exemplo, o termino da gravação de um ficheiro ou a espera de determinada hora para iniciar a execução de processo

Estados de um processo

Mudança de estados de um processo

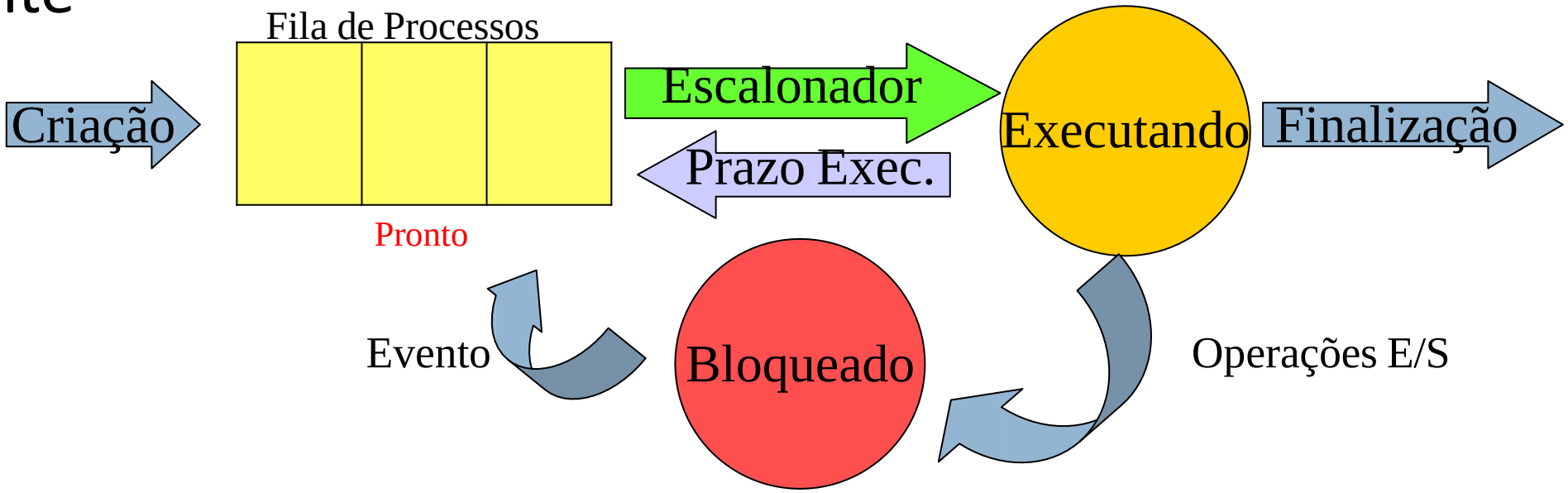
- **Pronto -> Execução** - Após ser criado o processo é colocado em uma lista de execução em estado de pronto onde fica a aguardar a sua vez para ser executado
- **Execução -> Espera** - Um estado em execução passa para o estado de espera por eventos externos ou por eventos gerados pelo próprio processo. Por exemplo, uma operação de entrada/saída
- **Espera -> Pronto** - Um processo em espera passa para o estado de pronto quando o recurso solicitado é concedido ou quando a operação solicitada é concluída. Um estado em espera sempre terá que voltar ao estado de pronto antes de prosseguir sua execução. Nenhum processo em espera passa diretamente para execução
- **Execução -> Pronto** - Um processo em execução passa para o estado de pronto por eventos gerados pelo sistema, como o término da fatia de tempo que o processo possui para sua execução

Processos em Background e Foreground

- Os processos também podem ser classificados pela forma de comunicação com o utilizador ou com outros processos. A comunicação dos processos é feita por canais de entrada e saída que podem estar associados a terminais, impressoras, ficheiros ou a outros processos
- Um processo é dito em **Foreground** quando permite a comunicação direta do utilizador com o processo durante o seu processamento. Os canais de entrada e saída são geralmente o teclado e o terminal, respetivamente. Desta forma, existe interação entre o utilizador e o processo durante a execução do processo
- Um processo é dito em **Background** quando não permite a comunicação direta do utilizador com o processo durante a sua execução. O processamento batch é geralmente associado a processos background

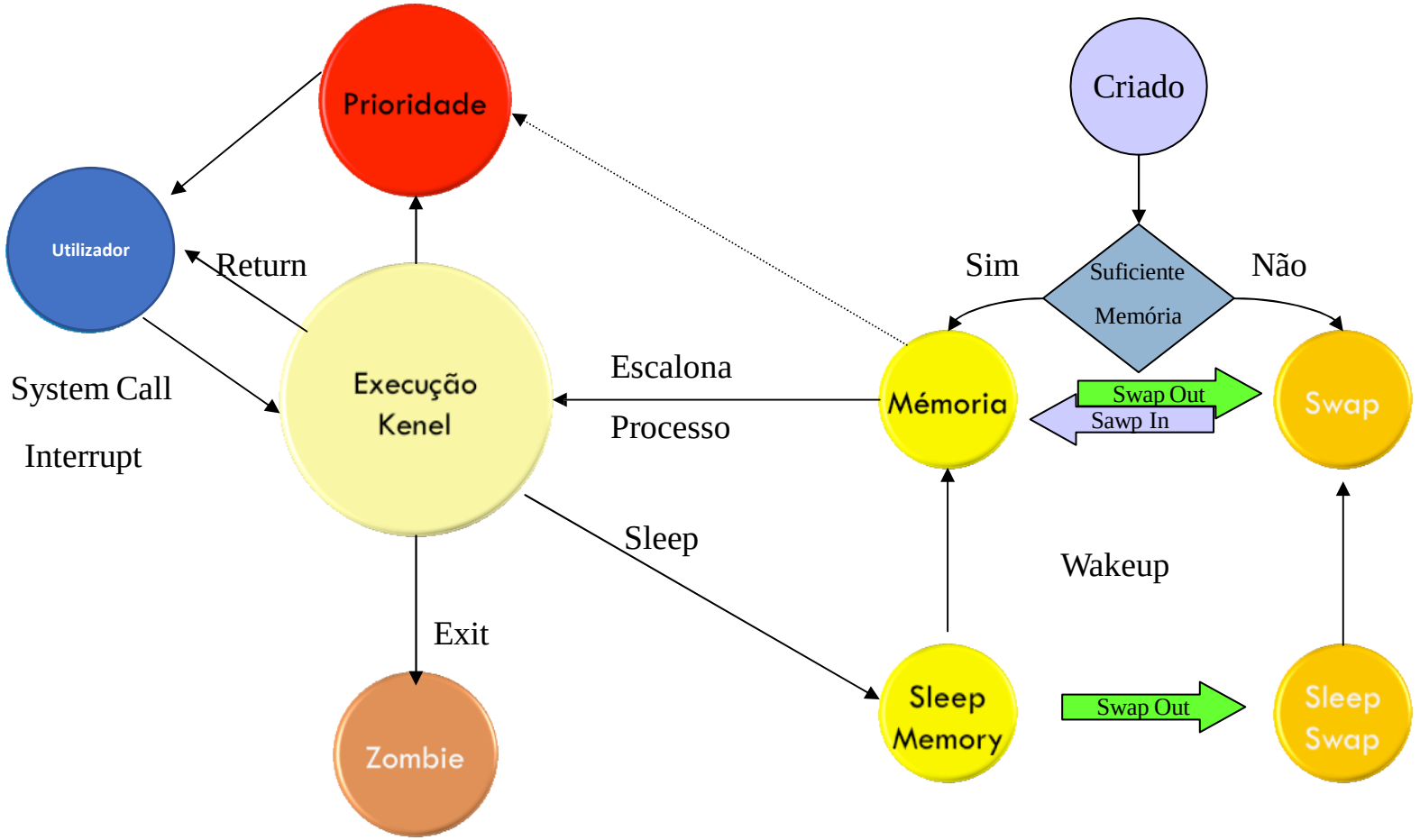
Escalonador de processos

- Programa que controla/decide que processo deve ser executado a cada instante



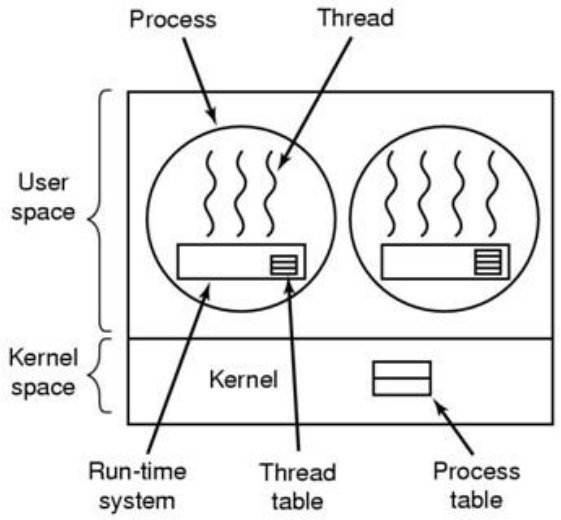
É função do Escalonador de Processos:
Dividir tempo de CPU para diferentes processos de forma a maximizar a utilização da CPU, fornecendo um tempo de resposta razoável!

Escalonador de processos



Thread

- Uma **thread** é a tarefa que um determinado programa realiza. Fio de execução, também conhecido como linha ou encadeamento de execução, é uma forma de um processo se dividir a si mesmo em duas ou mais tarefas que podem ser executadas concorrencialmente



Thread (Benefício)

- Programas concorrentes com múltiplas threads são mais rápidos do que implementados com múltiplos processos pois operações como criação, troca de contexto e eliminação das threads geram menor sobrecarga no SO
- Alem disso, como as threads compartilham o mesmo espaço de endereçamento, a comunicação entre as threads é feita de forma mais simples e mais eficiente e mais rápida
- Finalmente, as threads de um mesmo processo podem compartilhar outros recursos como ficheiros, temporizadores, sinais, atributos de segurança, etc

Threads (desempenho)

- O uso de threads pode ajudar a melhorar o desempenho do SO
- Em sistemas cliente-servidor uma solicitação de serviço remoto pode ser executada numa thead de forma assíncrona libertando o sistema para outras tarefas
- Num sistema monothread, se uma aplicação solicita um serviço remoto esta deve esperar o resultado antes de poder prosseguir sua execução

Threads (Arquitetura e implementação)

- Threads podem ser criadas através de uma biblioteca de funções fora do núcleo do SO (**modo utilizador**). Neste caso as threads são implementadas pela aplicação do utilizador e não pelo SO. É a aplicação que deve gerir as threads (sincronização e comunicação)

- Threads podem ser criadas pelo próprio núcleo do SO (**modo kernel**) através de chamadas às rotinas do sistema que oferecem todas as funções para gerir threads (sincronização e comunicação). Neste caso é o SO quem cuida das threads e decide que thread vai ser executada pelo processador num determinado momento (escalonamento de threads)

Threads (Arquitetura e implementação)

- Threads podem ser criadas por uma combinação de ambos (**modo híbrido**). Nesta implementação um processo pode ter várias threads em modo kernel e cada thread em modo kernel pode ter várias threads em modo utilizador. O núcleo do SO reconhece as threads em modo kernel e pode escalona-las individualmente. O problema desta implementação é que não é possível a comunicação entre threads em modo utilizador e threads em modo kernel
- Threads em **modo Scheduler Activations** usam o melhor do modo kernel e do modo utilizador. Usam as facilidades do modo kernel com o desempenho e a flexibilidade do modo utilizador. Nesta implementação o núcleo do sistema troca informações com a biblioteca de threads utilizando uma estrutura chamada scheduler activations. Esa implementação evita a troca de modo de acesso (utilizador-kernel-utilizador) uma vez que o kernel e a biblioteca de threads comunicam e trabalham entre si de forma cooperativa

Em suma…

- Qual a estrutura de um SO?
- Que arquiteturas de SO conhece e como estas se distinguem entre si?
- Como define um Processo?
- Que estados pode tomar um processo?
- Distinga processo em background de processo em foreground?
- Qual a distinção entre processo e thread?
- Quais as vantagens em usar múltiplas threads?
- Que tipos de threads conhece?
- Qual a função de um escalonador de processos/threads?