Principais eventos que levam à criação de processos:

1. Ao iniciar o sistema operacional (o init)
2. Usuário executa comando/ inicia programa através da shell
3. Atendimento de uma requisição específica (p.ex.processo **inet** cria processo para tratar requisição de rede: ftp, rsh, etc.)
4. Início de um programa em momento pré-determinado (através do cron daemon)
5. Processamento de um job de uma fila de jobs

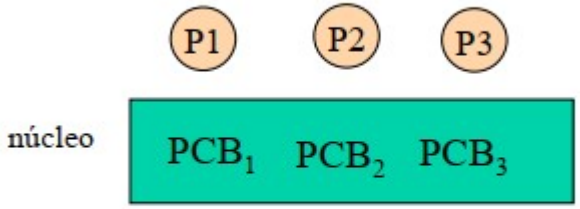
Em todos os casos, um processo pai cria um novo

Implementação de Processos

A cada processo estão associadas informações sobre o seu estado de execução (o seu *contexto de execução*), Estas ficam armazenadas em uma entrada da Tabela deProcessos (e em Process Control Blocks)

Gerenciamento de processos	Gerenciamento de memória	Gerenciamento de arquivos
Registradores	Ponteiro para o segmento de código	Diretório-raiz
Contador de programa	Ponteiro para o segmento de dados	Diretório de trabalho
Palavra de estado do programa	Ponteiro para o segmento de pilha	Descritores de arquivos
Ponteiro de pilha		Identificador (ID) do usuário
Estado do processo		Identificador (ID) do grupo
Prioridade		
Parâmetros de escalonamento		
Identificador (ID) do processo		
Processo pai		
Grupo do processo		
Sinais		
Momento em que o processo iniciou		
Tempo usado da CPU		
Tempo de CPU do filho		
Momento do próximo alarme		

Process Control Block (PCB)

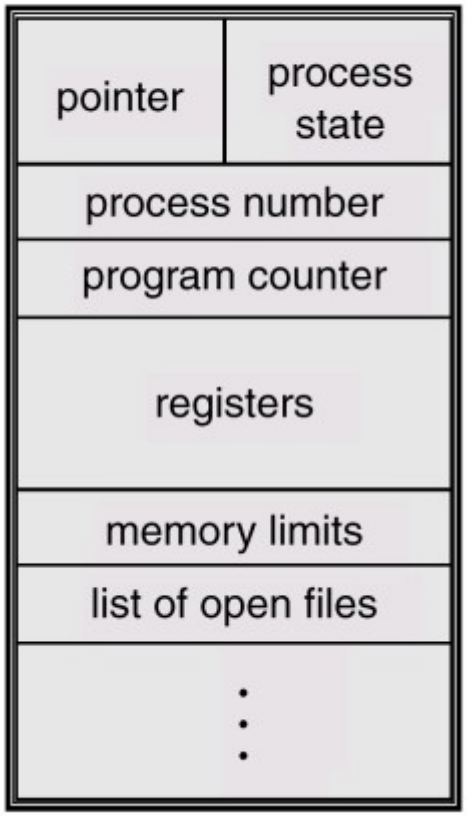


PCB contém informações que são necessárias para colocar o processo em execução.

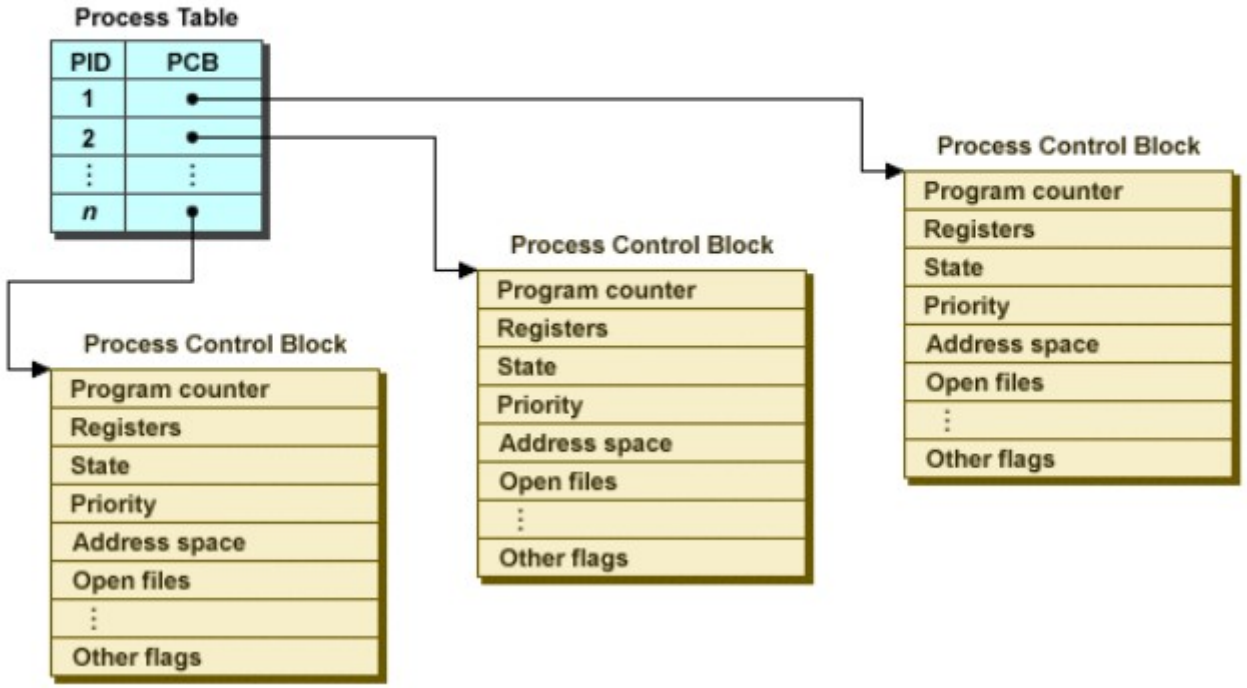
Para ser capaz de reiniciar um processo interrompido (ou esperando) o estado anterior em que deixou a CPU precisa ser restaurado;

Carrega-se a CPU (e e MMU) com os dados de contexto armazenados no PCB

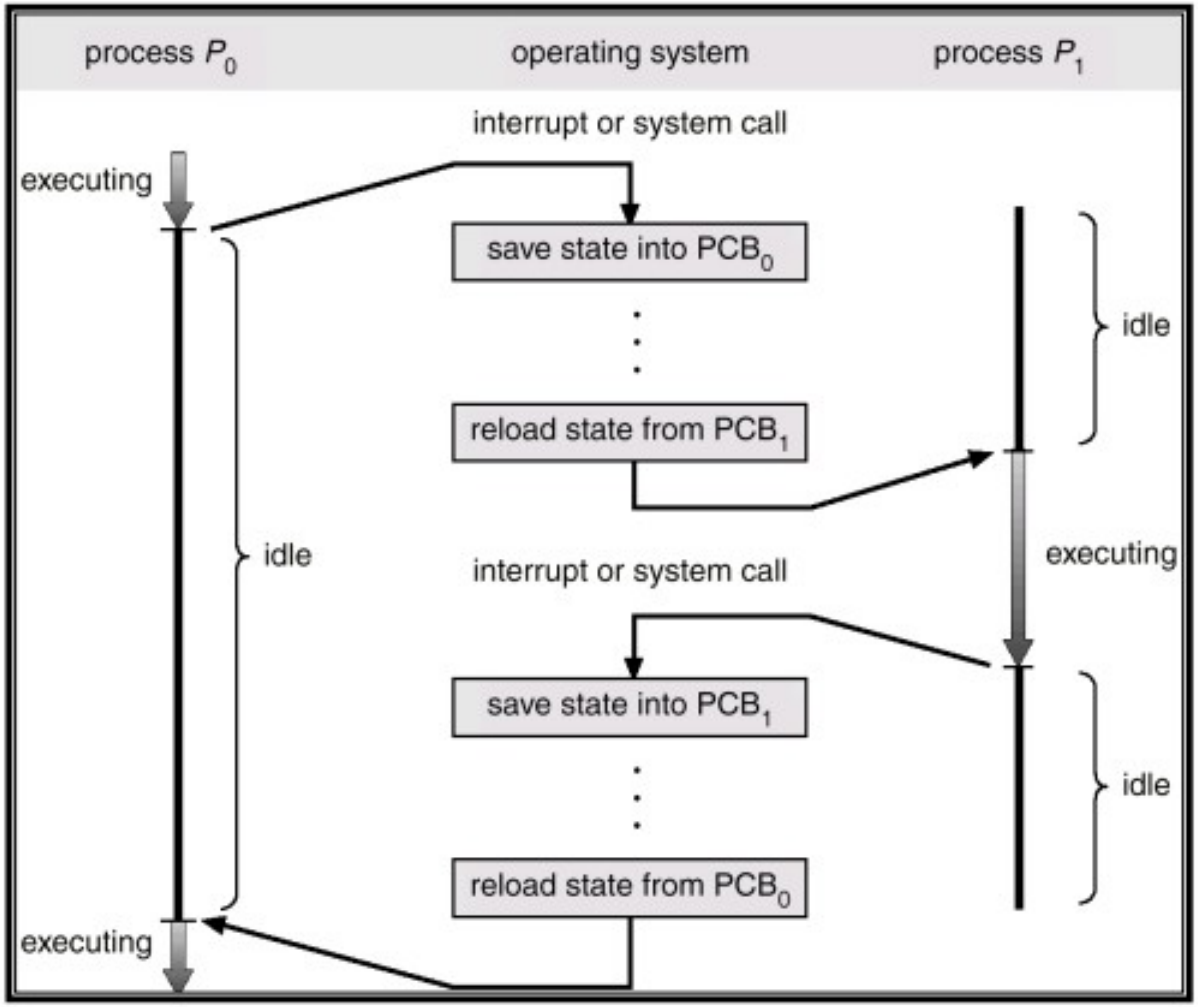
Em sistemas Unix, o PCB é uma estrutura no espaço do usuário que é acessada pelo núcleo (área u)

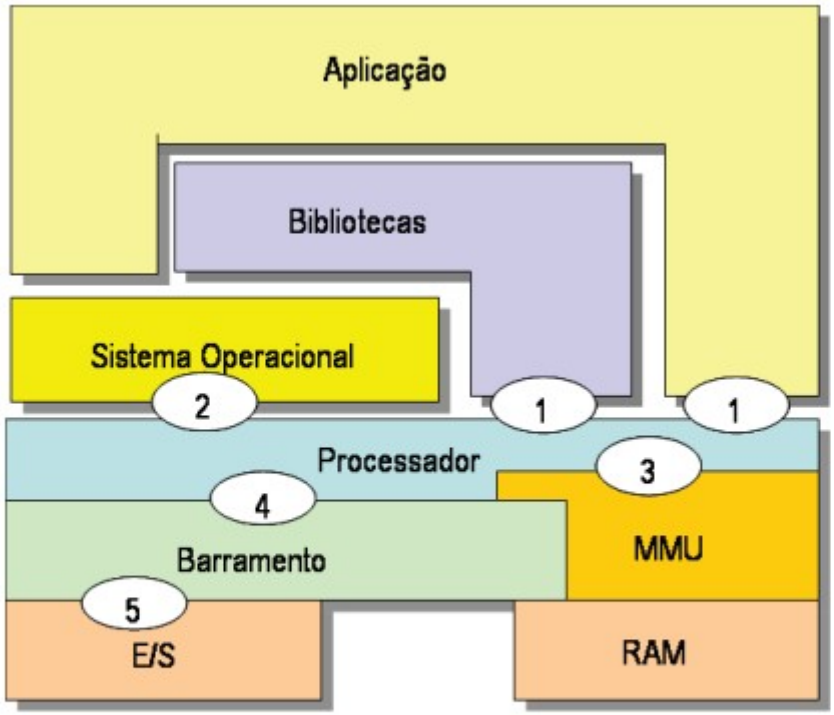


Process Table e PCB



TROCA DE CONTEXTO





Arquitetura de computadores como um conjunto de camadas de abstração (adaptada de [Smith e Nair 2005])

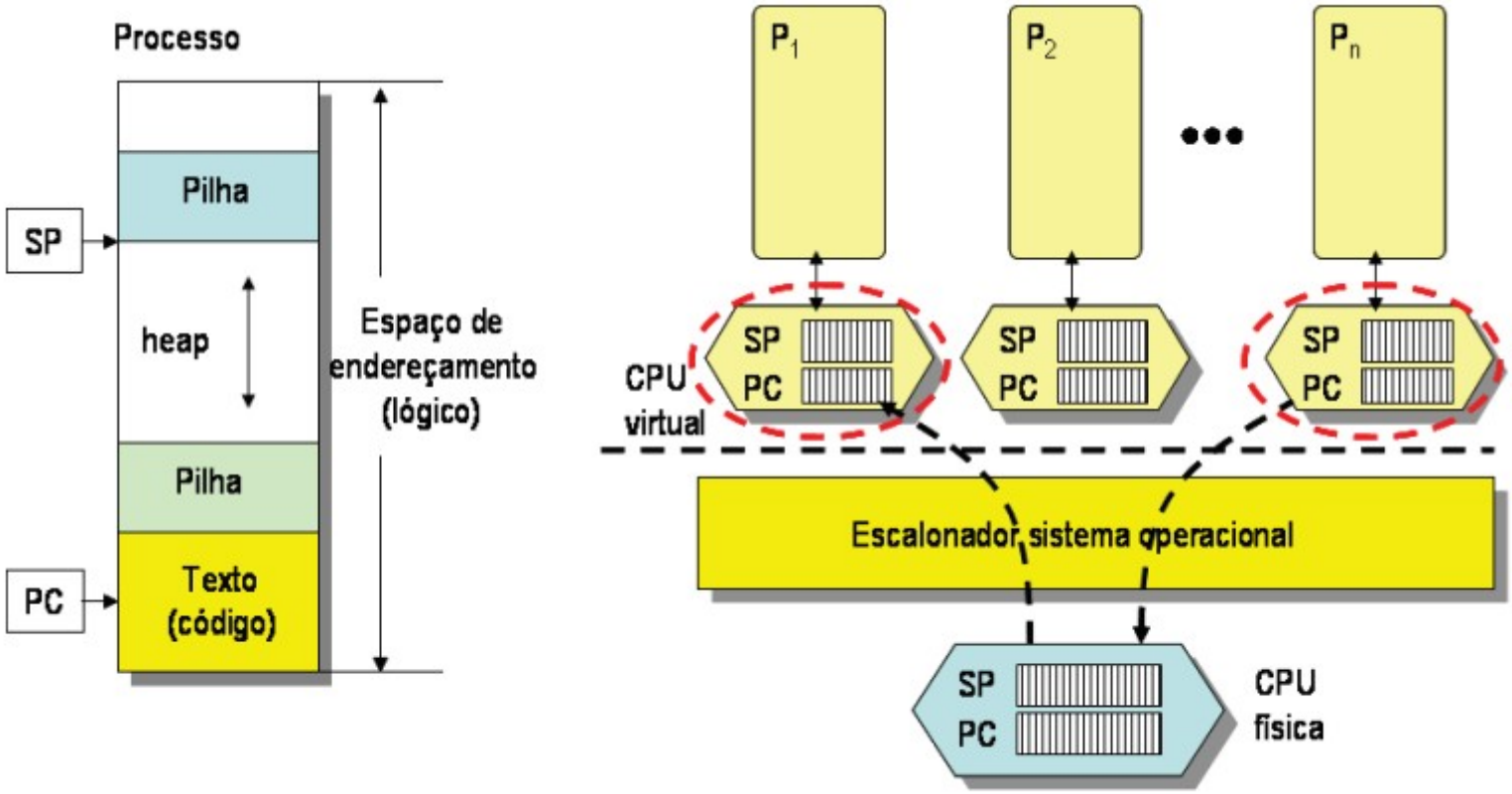


Figura 4.2 – A abstração de processo em um sistema operacional

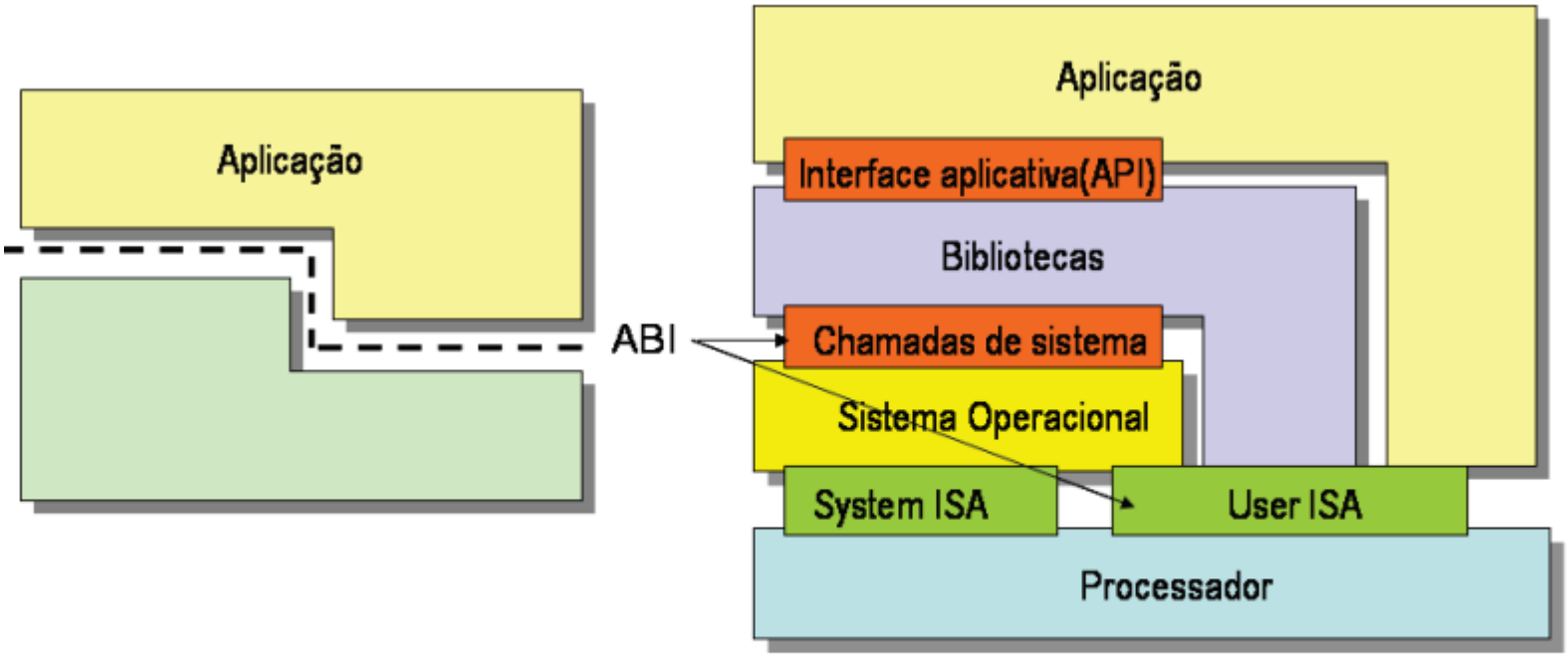


Figura 4.3 – Interfaces genéricas de um sistema de computação

Teorema de Popek & Goldberg III

Instruções sensıveis

Podem consultar ou alterar o status do processador, ou seja, os registradores que armazenam o status atual da execução no hardware

Intruções privilegiadas

Execuçao em modo usuario gera uma exceçao, ou seja, é interrompida e o sistema operacional é notificado.
Sao acessíveis somente por meio de codigos executando em nível privilegiado (codigo de n´ucleo).

Teorema de Popek & Goldberg IV

Segundo o teorema:

Toda instrução sensível deve ser também privilegiada Quando uma instrução sensível for executada por uma aplicação nao-privilegiada (n´ucleo do SO convidado, anel 1 ou 2) irá provocar uma interrupçãao que deverá ser interceptada e tratada pelo *hypervisor*
Hypervisor irá emular o efeito desejado da instruçãao sensível
Garantindo o **isolamento** entre VMs
Instruçõesnão-privilegiadas podem ser executadas diretamente no hardware real *!não*-prejudiciais `a virtualização
Arquitetura *x*86, a princıpio, nao obedecia este teorema
Instruçõoes sensıveis que executem sem gerar interrupções
Hypervisor nao consegue interceptá-las e interpretá-las

Virtualizaçao completa ou total

Hypervisor situado no anel 0
Disponibiliza uma réplica do hardware subjacente .Alguns autores chamam de *virtualização do hardware*
Toda interface (ISA) é virtualizada
Vantagem:
SO visitante nao precisa sofrer modificações para rodar na VM
Tradução binária: binário SO *! hypervisor !* binário hardware Em tempo de execuçãoo
Detectar e tratar instruções sens´ıveisnão-privilegiadas que não geram interrupções ao serem invocadas pelo SO *guest !* problema na *x*86
Desvantagem:
Queda do desempenho em até 30%
Exemplos: *VMware*, *VirtualBox* e *Qemu*

Paravirtualizaçãoo

SO convidado tem consciência" que está sendo virtualizado
SO convidado precisa ser modificado
Adaptação é feita no *kernel*
Tradução binária não é realizada
Aumenta o desempenho da virtualizaçãoo
Precisa verificar apenas interface de sistema (instruções sensíveis)

Interface de usuário é mantida (apps do usuário sem modificação)
Vantagem:
Acesso ao hardware diretamente (não-privilegiadas)
Monitorado pelo *hypervisor*, que informa os limites (*e.g.* disco e mem´oria)
Desvantagem:
Alteração do *kernel*
Linux: modificar 2995 linhas de c´odigo (≈ 1, 36% do c´odigo fonte)
Exemplo: *Xen*

Virtualização em *x*86

Arquitetura *x*86 não foi projetada, a princípio, para ser virtualizada 17 instruções sensíveis que não são privilegiadas cuja execuçãoo em modo usuário não gera uma exceçãoo, mas é tratada pela propria *x*86 e assim Viola o teorema de Popek & Goldberg
Problema:
Hypervisor executa no anel 0 (modo privilegiado) *!* gerencia recursos
SO ´unico software que deve executar no anel 0 *!* gerencia recursos
*x*86 for virtualizada *!* anel 1 ou 2
Virtualizaçãoo é transparente *!* a *x*86 \pensa" que está no anel 0
Executar uma das 17 instruções \cr´ıticas" *!* abortadas
Ideal: Gerar uma exceçãoo e ser tratada pelo *hypervisor*
Desafio: lidar com essas 17 instruções cr´ıticas
Virtualizaçãoo completa *!* traduçãoo binária (queda no desempenho)
Solução: *virtualizaçãoo assistida pelo hardware*
Intel-VT (*Vanderpool*) em 2005 e AMD-V (*pac´ıfica*) em 2006

Importância da Virtualização

Virtualização pode ser vista como a capacidade de executar múltiplos sistemas operacionas (SOs) em uma única plataforma física, dividindo os recursos de hardware. Em outras palavras, a virtualização permite ter duas ou mais máquinas virtuais (MVs), executando SOs diferentes simultaneamente em um mesmo computador de forma totalmente isolada.
O isolamento dos softwares em suas próprias MVs oferecido pela virtualização pode aumentar a segurança e confiabilidade de diversos sistemas. A segurança pode ser aumentada com o confinamento de invasões nas MVs em que ocorrem, impedindo que ela atinja outras aplicações que estejam executando em outras MVs. Enquanto a confiabilidade pode ser melhorada porque falhas de software em uma VM não afetam as outras MVs. Além disso, para assegurar maior tolerância a falhas é possível executar cópias idênticas da mesma carga de trabalho em duas MVs distintas e, assim, em caso de falha de uma das MVs, a outra pode fazer a cobertura instantaneamente

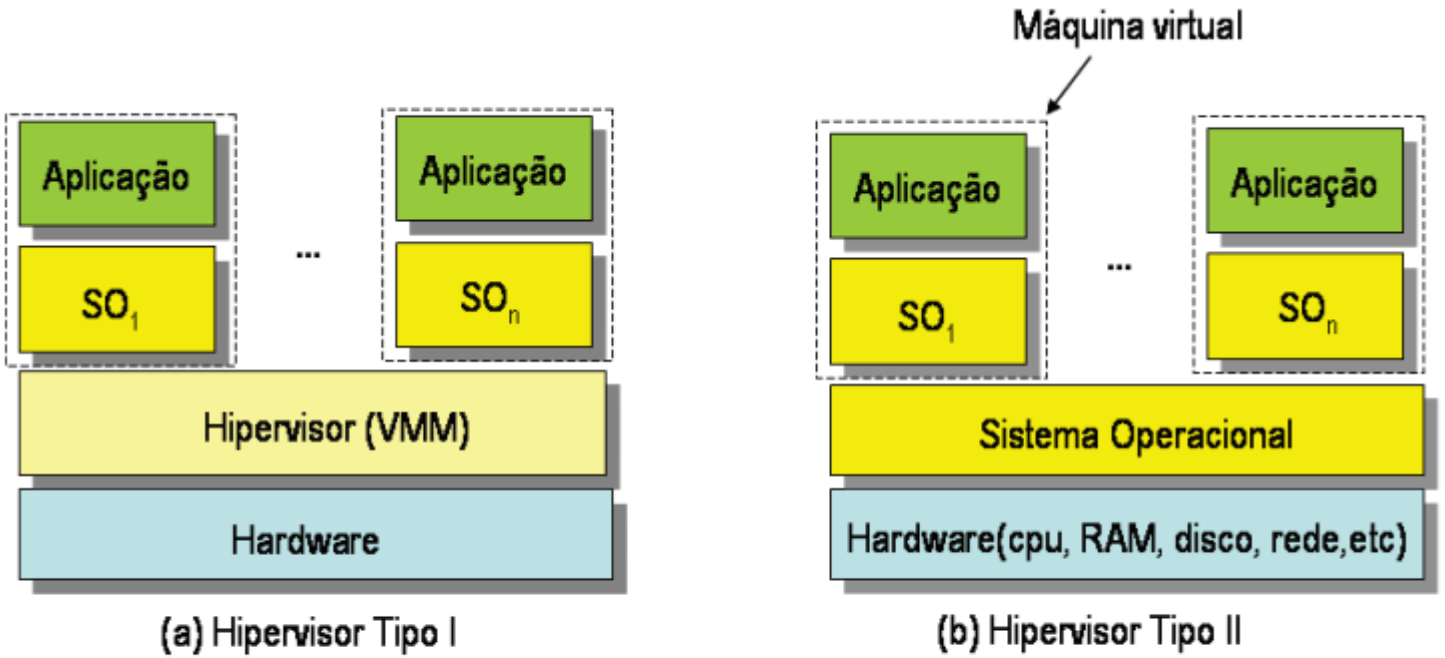


Figura 4.5 – Máquina virtual de processo: hipervisores tipos I e II

Os hipervisores tipo I, ou nativos, são aqueles que executam diretamente sobre o hardware de uma máquina real e as máquinasvirtuais são postas sobre ele (figura 4.5a). A função básica de um hipervisor nativo é compartilhar os recursos de hardware (processador, memória, meios de armazenamento e dispositivos de E/S) entre as diferentes máquinas virtuais de forma que cada uma delas tenha a ilusão de que esses recursos são privativos a ela

Os hipervisores tipo II, ou hóspedes, são caracterizados por executar sobre um sistema operacional nativo como se fossem um processo deste hardware virtual criado sobre os recursos de hardware oferecidos pelo sistema operacional nativo

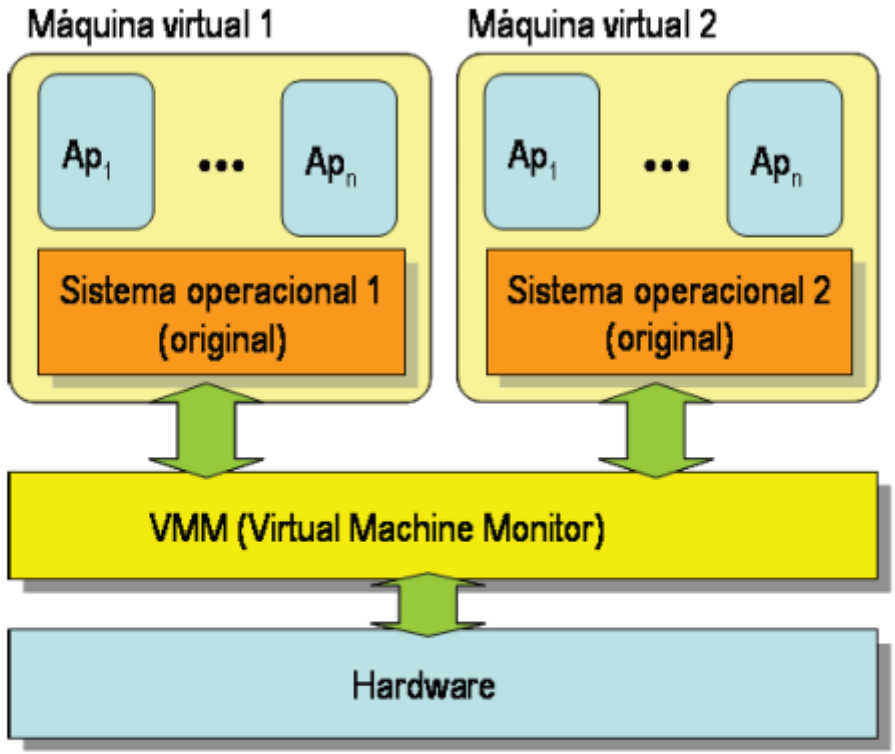


Figura 4.7 – Virtualização total

A Intel apresenta suas extensões para as arquiteturas x86 de 32 e 64 bits sob o nome IVT (*Intel Virtualization Technology*), ao passo que a AMDoferece esse suporte apenas para suas arquiteturas de 64 bits. A extensão da AMD é denominada de AMD-V, AMD-*Virtualization*.

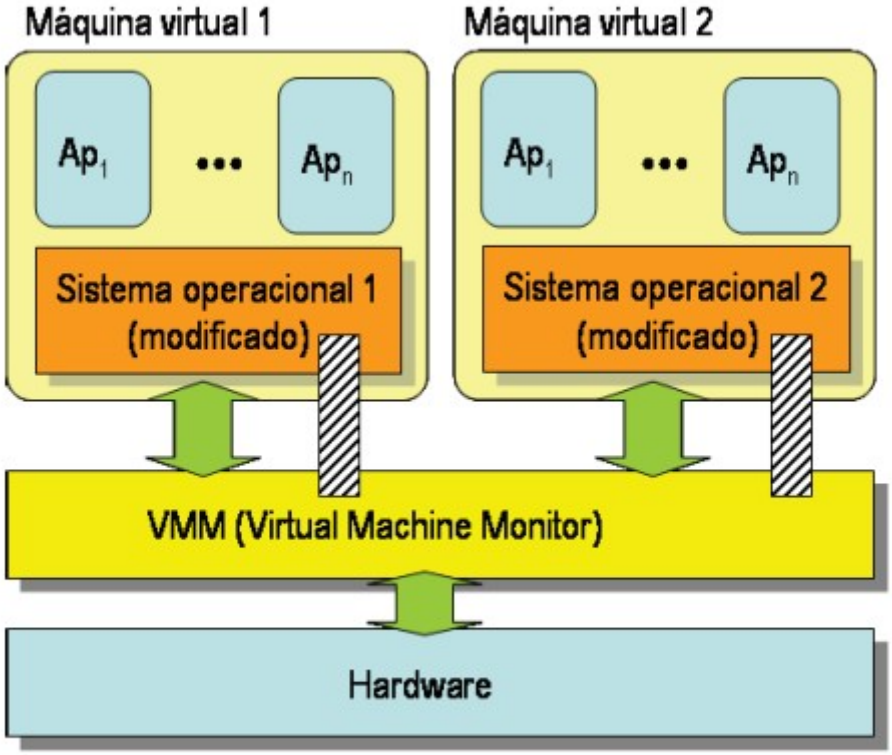
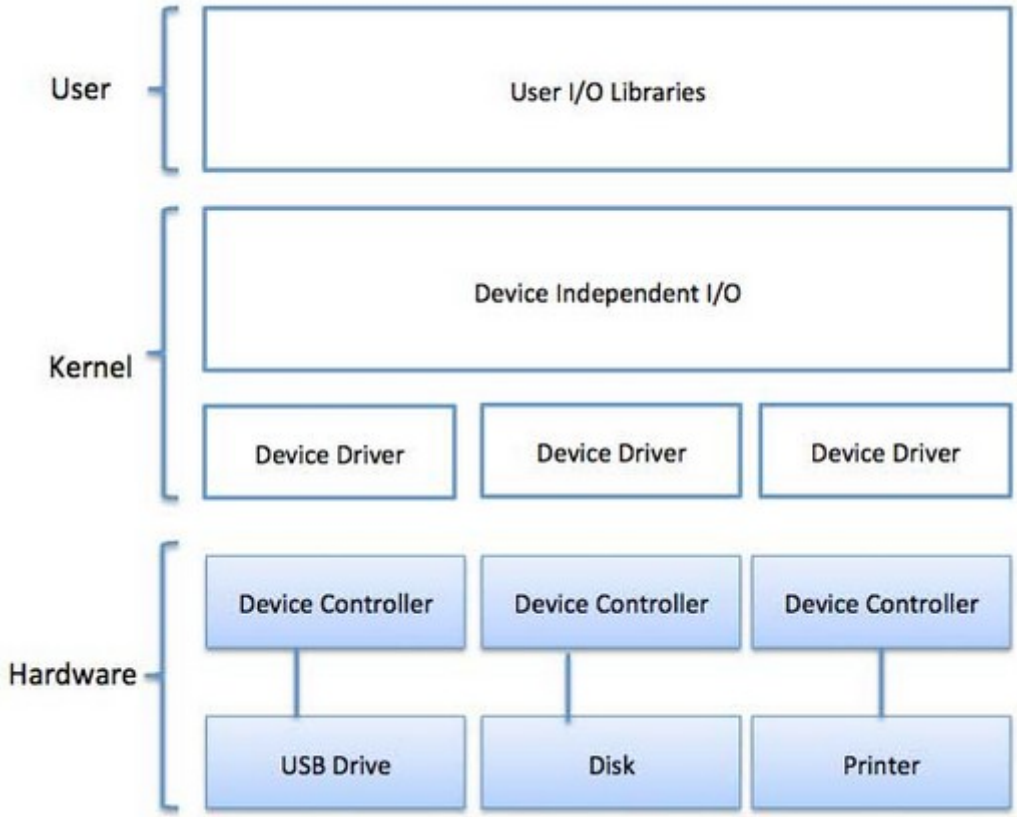
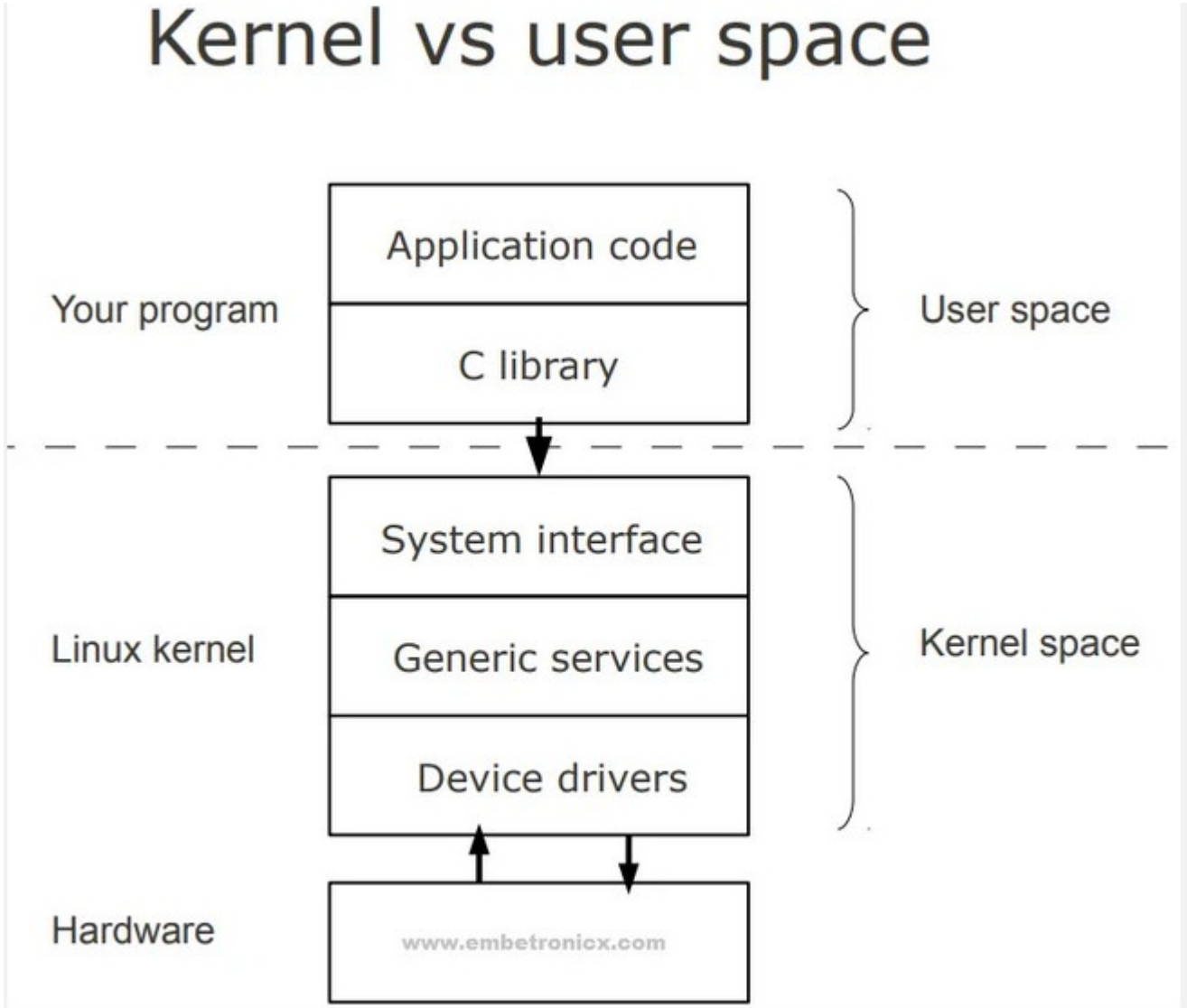
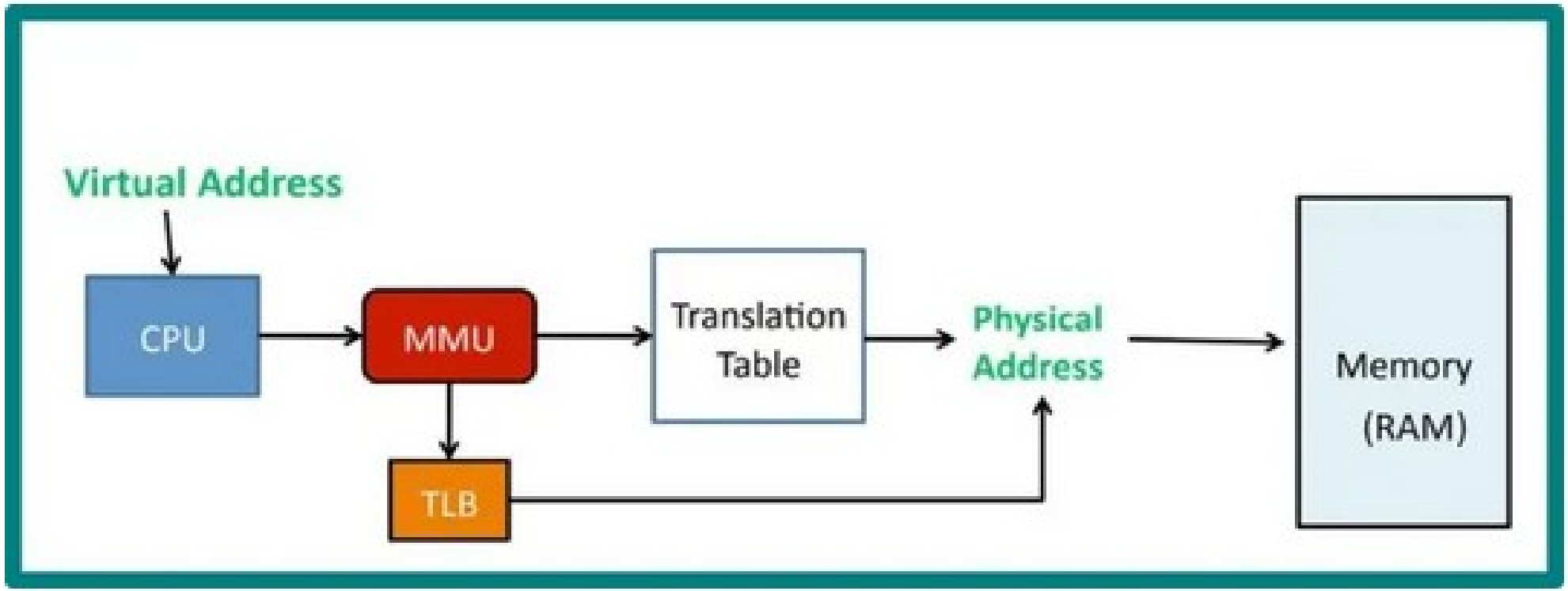


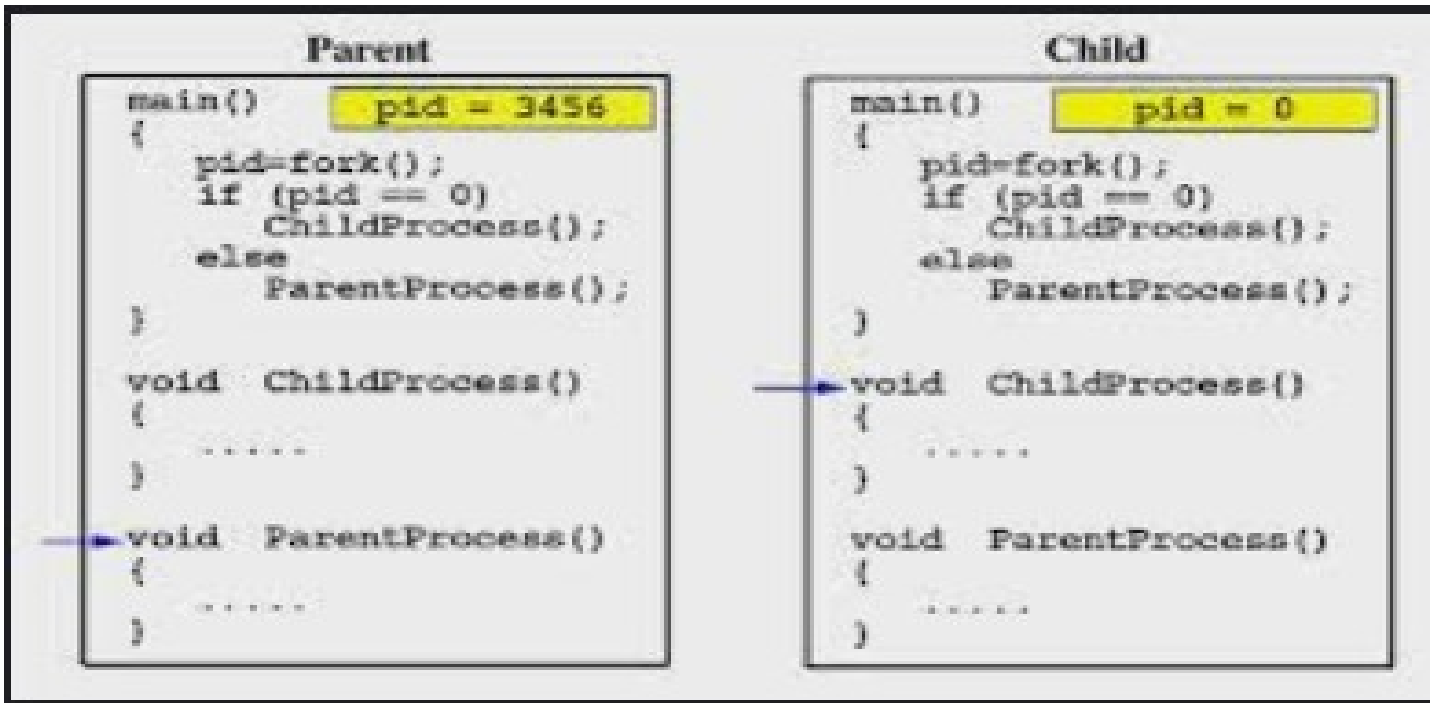
Figura 4.8 – Paravirtualização

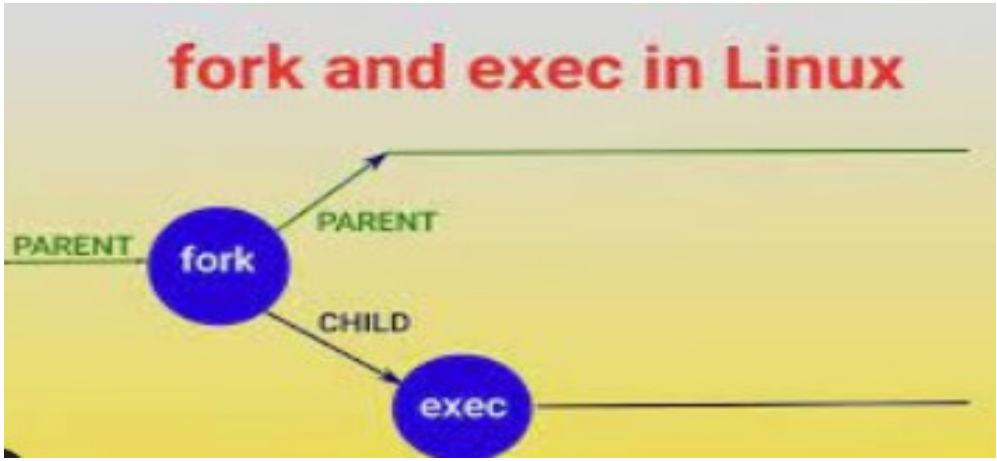

User
User I/O Libraries
Kernel
Device Independent I/O
Device Driver
Device Driver
Device Driver
Hardware
Device Controller
Device Controller
Device Controller
USB Drive
Disk
Printer



Kernel Space


Virtual Address
CPU
MMU
Translation Table
Physical Address
Memory (RAM)
TLB


Parent
main()
pid = 3456
{
pid=fork();
if (pid == 0)
ChildProcess();
else
ParentProcess();
}
void ChildProcess()
{
.....
}
void ParentProcess()
{
.....
}
Child
main()
pid = 0
{
pid=fork();
if (pid == 0)
ChildProcess();
else
ParentProcess();
}
void ChildProcess()
{
.....
}
void ParentProcess()
{
.....
}


fork and exec in Linux
PARENT
fork
PARENT
CHILD
exec