RAID
(Redundant Array Independent Disks)
Sistema de aumento de performance e confiabilidade dos discos.

- Mais de um disco;
- Controladora RAID (On-Board ou Off-Board);

RAID POR SOFTWARE

- Controladora RAID ON BOARD (Embutida na placa mãe);
- O controle do RAID é executado pelo Chipset da placa mãe;
- Menor performance;

RAID POR HARDWARE

- Necessita de uma controladora RAID off-board;
- Placa controladora conectada aos slots;
- Melhor performance;
- Mais caro;.

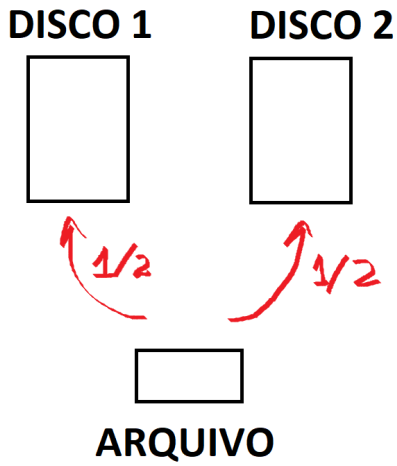
A controladora Raid possui um Setup próprio.

O Raid é baseado em duas tecnologias:

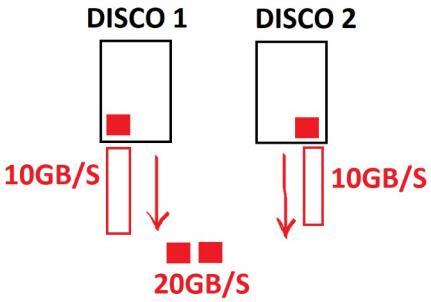
Data Stripping

- Une os Discos, somando suas capacidades e suas velocidades;
- Cada arquivo é dividido e suas partes ficam armazenadas nos diferentes discos;
- No processo de leituras as partes são lidas em paralelo, simultaneamente, multiplicando a velocidade pela quantidade de discos;
- Sistema sem redundância de dados;
- Aumenta a performance,porém não aumenta a segurança;

ARMAZENAMENTO



LEITURA



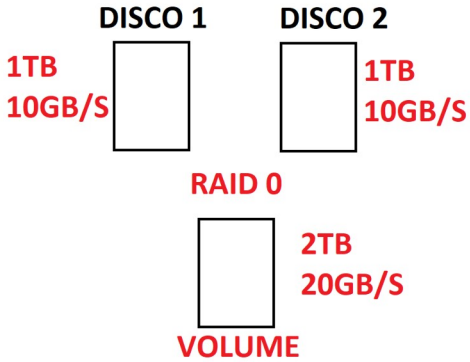
Mirroring

- Espelhamento
- Os dados armazenados em um disco são copiados para outro.
- Com sistema Hot Swap caso um disco falhe o outro disco assume o funcionamento do sistema;
- Não tem aumento de performance;
- Tem aumento de segurança;

RAID nível 0 (Data Stripping)

Trata-se do caso já exemplificado antes, do armazenamento de um único arquivo por dois ou mais discos, assim obtendo-se mais rapidez na transferência. Em inglês, chama-se essa técnica de stripping, que pode significar fracionamento, ou seja, fragmenta-se ou fraciona-se o arquivo em várias partes e cada uma é armazenada em um disco diferente. sendo todos eles acionados simultaneamente em uma transferência. Deve ser observado que este nível de RAID não aplica o conceito de redundância, pois não se está colocando a mesma parte do arquivo em mais de um disco. mas partes diferentes. Havendo algum problema em um dos discos, a parte nele armazenada pode ser perdida. Em face de suas características de velocidade, este nível é frequentemente usado em aplicações com grandes volumes de dados que requeiram rapidez de acesso, como CAD ou tratamento de imagens, vídeo e áudio, embora de custo elevado.

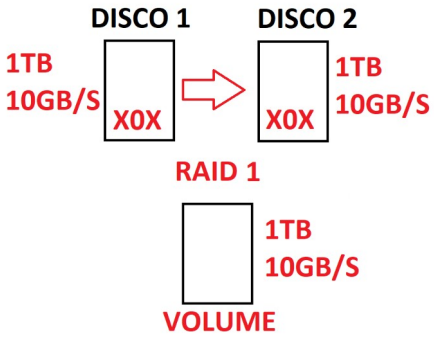
- Exclusivamente Data Stripping;
- Somar as capacidades dos discos;
- Somar as velocidades dos discos;
- Não tem nenhum recurso de segurança;
- MÍNIMO 2 DISCOS, SEMPRE EM PARES;
- Se um disco ou mais falhar o sistema inteiro será comprometido;



RAID nível 1 (Mirroring)

Consiste na implementação do outro objetivo da tecnologia RAID, de redundância, a qual é utilizada através de duplicação, triplicação ou mais de um determinado volume de dados por vários discos (espelhamento). Desse modo, cada transação para gravação de dados em um disco é realizada também no outro ou outros, definidos no espelhamento. Naturalmente, o espelhamento é realizado sem intervenção do usuário, por meio de lógica apropriada, que deve estar presente no sistema ou não funcionará. É possível combinar os dois níveis, O e 1, de modo a garantir rapidez e confiabilidade maiores. Por exemplo um sistema que possua dois discos pode usá-los para o nível O e acrescentar mais dois discos para espelhamento de maneira que os quatro discos sejam usados para os RAIO O e I de modo único. Seu emprego é apropriado, por exemplo, para servidores de arquivos ou sistemas gerenciadores de grandes bancos de dados.

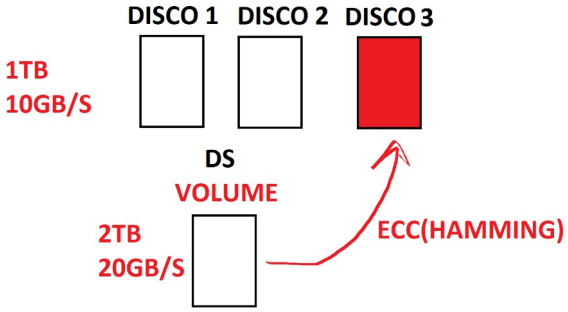
- RAID do Mirror
- Espelhamento
- O total de espaço disponível vai ser a metade do total de Hd´s;
- Não tem aumento de velocidade;
- Só tem aumento de segurança;
- As gravações são duplicadas;
- Redundância de discos;
- MÍNIMO DOIS DISCOS, SEMPRE EM PARES;



RAID nível 2

Este tipo de RAID adapta o mecanismo de detecção de falhas, para funcionar com a memória principal e através do emprego de acesso paralelo. Trata-se de uma especificação nunca implementada em face do seu elevado custo para um beneficio já implantado nos discos (tolerância a falhas) e por visar um tipo de problema (muitos erros em acessos a discos) raramente encontrado.

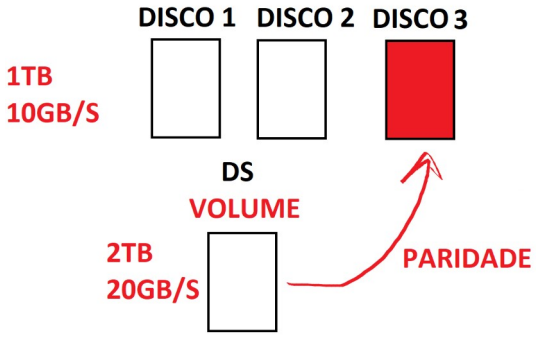
- Não é uma implementação aplicada na prática;
- Necessita de um disco extra;
- No disco extra fica gravado um código de ECC com algoritmo Hamming;
- Caso um disco venha a falhar o disco extra recupera os dados do mesmo;
- Une velocidade do data stripping e segurança do **ECC (Hamming)**.
- SEMPRE COM 3 DISCOS;



RAID nível 3

Neste nível, semelhante ao nível 2, os dados são divididos pelos vários discos e se usa um disco adicional para armazenar os dados de paridade (detecção de erros). Através da verificação dessa informação (paridade) pode-se garantir maior integridade dos dados, em caso de recuperação. Para usar o RAID 3 são necessários pelo menos três discos, oferecendo altas taxas de transferência e confiabilidade das informações.

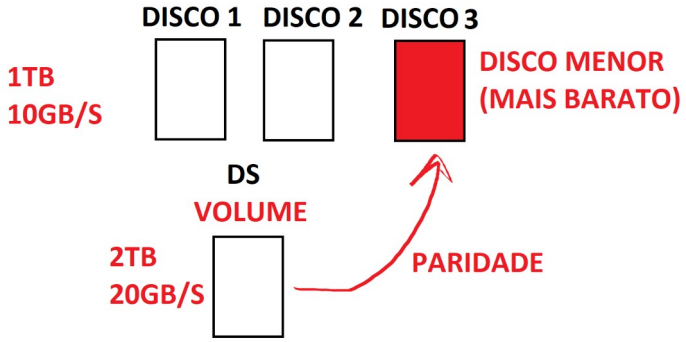
- Semelhante ao Raid 2;
- Aplicado na prática;
- Três discos, um extra para recuperação de dados e dois para dados;
- Utiliza a paridade (não recupera dados, somente encontra erros) no disco extra;
- 1001011
- Com um algoritmo de recuperação de dados proprietário;
- SEMPRE COM 3 DISCOS;



RAID nível 4

Neste tipo, basicamente semelhante ao nível 3, os dados são igualmente divididos entre todos os discos menos um, que servirá exclusivamente para inserir os elementos de paridade: A diferença é que no nível 4 o tamanho dos blocos a serem armazenados é grande, maior que no nível 3. Por isso, o rendimento é maior em um leitura, Ele é indicado para o caso de arquivos grandes, onde se requer maior integridade das informações, visto que em cada operação de leitura se efetua novamente o cálculo da paridade, obtendo-se, assim, maior confiabilidade (apesar do aumento do tempo de cada operação).

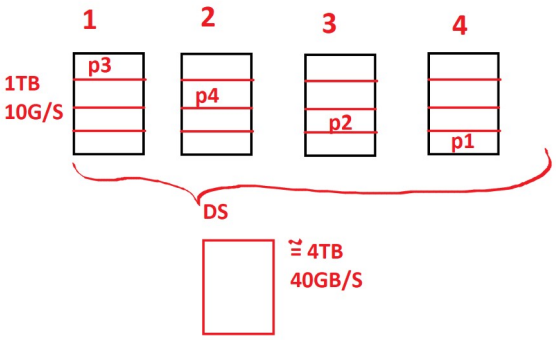
- Semelhante ao Raid 2,3;
- Disco Extra;
- Faz data stripping com dois discos;
- O terceiro disco é reservado para paridade;
- A paridade é medida de uma maneira diferente;
- Onde o disco extra pode ser menor que outros discos;
- Custo é menor;
- SEMPRE COM 3 DISCOS;



RAID nível 5

Semelhante ao nível 4, exceto pelo fato de que a paridade não se destina a um só disco, mas a toda a matriz. Nesse caso, o tempo de gravação é menor, pois não é necessário acessar o disco de paridade em cada operação de escrita. Não obstante, o nível 4 ainda possui melhor desempenho, pois no nível 5 a paridade é distribuída e ntre os discos. Ele precisa de pelo menos 3 discos para funcionar, mas é um dos mais utilizados em aplicações não muito pesadas.

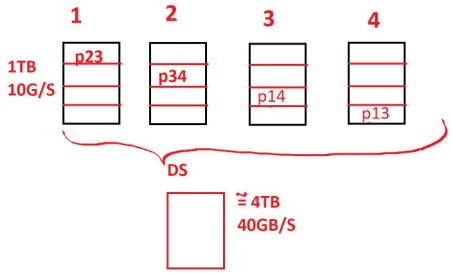
- Mais utilizado;
- Raid com **maior performance**;
- Necessita de pelo menos três discos;
- Paridade é armazenada em faixas;
- Não temos disco extra;
- A paridade fica armazenada no próprio disco;
- MÍNIMO 3;



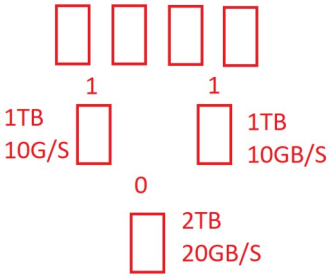
RAID nível 6

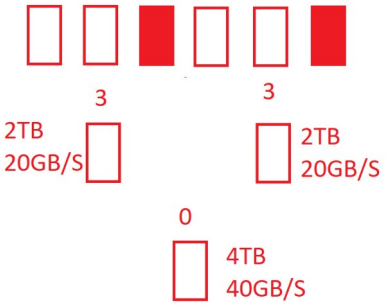
Este nível. que foi acrescentado ao mercado posteriormente, é baseado no nível 5, porém com a diferença de que nele há uma segunda gravação de paridade em todos os discos utilizados no sistema, aumentando, desse modo, a confiabilidade das informações.

- Idêntico ao RAID 5;
- Armazena duas paridades em cada disco aumentando a possibilidade de recuperação;
- Mais discos podem falhar por vez, aumentando a segurança;
- MÍNIMO 3;



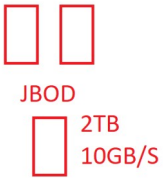
RAID 0 – DATA STRIPPING
RAID 1 – MIRROR
RAID 2 – DATA STRIPPING COM DISCO EXTRA COM ECC
RAID 3 – DATA STRIPPING COM DISCO EXTRA DE PARIDADE
RAID 4 – DATA STRIPPING COM DISCO EXTRA COM PARIDADE MENOR;
RAID 5 – DATA STRIPPING COM PARIDADE EM FAIXAS NO PRÓPRIO DISCO;
RAID 6 – DATA STRIPPING COM PARIDADE EM FAIXAS NO PRÓPRIO DISCO DUPLA;



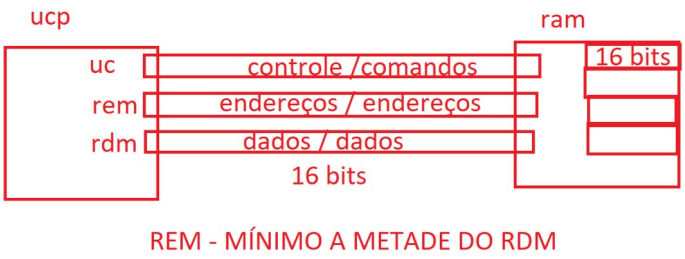


JBOD

- JUST A BUNCH OF DISKS;
- Une os discos sem fazer Data Stripping;
- Une os discos mas não soma as velocidades;
- Quando precisamos armazenar um arquivo grande em que o espaço necessário está em dois discos...
- Nem é Raid;



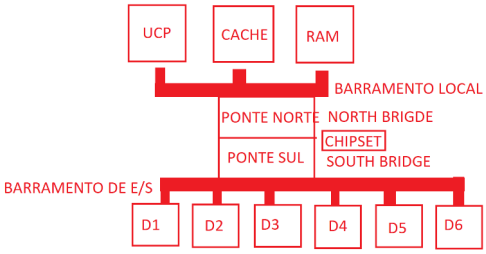
COMUNICAÇÃO ENTRE A RAM E A CPU



REM - MÍNIMO A METADE DO RDM

Barramento local
Barramento que interliga o processador / Memória / Cache

Divido em :
Barramento de dados : Transporta os dados (foto) : RDM = REGISTRADOR DE DADOS DA MEMÓRIA
Barramento de endereços: Transporta os endereços (376) : REM = REGISTRADOR DE ENDEREÇOS DA MEMÓRIA
Barramento de controle : Transporta os comandos (escrever) : UC = UNIDADE DE CONTROLE



Modos de endereçamento:
Forma de organização dos dados que se comunicam da RAM para o processador e vice versa;

Código da operação	Campo operando

Imediato:

- No campo operando temos o dado;
- O campo operando limita o tamanho da informação;
- Campo operando = 16KB, informação máxima transmitida será de 16KB;
- Nenhum acesso a memória para enviar os dados ; Ou 1 acesso para ler ou gravar;

Código da operação	**Dado**

Direto:

- No campo operando temos um endereço de memória que aponta para o dado;
- O campo operando não limita o tamanho da informação;
- O endereço faz referência a uma informação que está na memória;
- O campo operando limita o tamanho da memória;
- 1 acesso a memória;

Código da operação	**Endereço de memória**

Indireto:

- No campo operando temos um endereço de memória que aponta para outro endereço de memória que aponta para o dado.
- Um endereço que aponta para um endereço que aponta para a informação;
- O campo operando não limita a capacidade de memória acessada;
- 2 acessos a memória;

Código da operação	Endereço de memória

Por registrador

- A memória não é acessada;
- Somente os registradores;

Código da operação	Endereço de registrador

Por registrador direto

- No campo operando temos um endereço de registrador que aponta para o dado;

Código da operação	Endereço de registrador

Por registrador indireto

- No campo operando temos um endereço de registrador que aponta para outro registrador que aponta para o dado;
- O endereço de um registrador que aponta para outro registrador que aponta para o dado;

Indexado

- O campo operando será somado com um valor de um registrador;

Código da operação	Endereço de registrador

Base mais deslocamento

- O campo operando será somado a um deslocamento constante.
- Leitura de um vetor, matriz;

Código da operação	Endereço de registrador

Via estrutura de pilha

- Não há nenhum endereço referenciado, somente uma pilha;

Código da operação	Pilha

Processos e Threads

Estamos prestes a embarcar agora em um estudo detalhado de como os sistemas operacionais são projetados e construídos. O conceito mais central em qualquer sistema operacional é o **processo**.

- Processo é um programa em execução;
- Um programa não é um processo;
- Um arquivo não é um processo;

Processos : **É uma abstração de um programa em execução**. Tudo o mais depende desse conceito, e o projetista (e estudante) do sistema operacional deve ter uma compreensão profunda do que é um processo o mais cedo possível. Processos são uma das mais antigas e importantes abstrações que os sistemas operacionais proporcionam. Eles dão suporte à possibilidade de haver operações (pseudo) concorrentes mesmo quando há apenas uma CPU disponível, transformando uma única CPU em múltiplas CPUs virtuais. Sem a abstração de processo, a computação moderna não poderia existir.

Processos

Todos os computadores modernos frequentemente realizam várias tarefas ao mesmo tempo. As pessoas acostumadas a trabalhar com computadores talvez não estejam totalmente cientes desse fato, então alguns exemplos podem esclarecer este ponto. Primeiro, considere um servidor da web, em que solicitações de páginas da web chegam de toda parte. Quando uma solicitação chega, o servidor confere para ver se a página requisitada está em cache. Se estiver, ela é enviada de volta; se não, uma solicitação de acesso ao disco é iniciada para buscá-la. No entanto, do ponto de vista da CPU, as solicitações de acesso ao disco levam uma eternidade. Enquanto espera que uma solicitação de acesso ao disco seja concluída, muitas outras solicitações podem chegar. Se há múltiplos discos presentes, algumas ou todas as solicitações mais recentes podem ser enviadas para os outros discos muito antes de a primeira solicitação ter sido concluída. Está claro que algum método é necessário para modelar e controlar essa concorrência. Processos (e especialmente threads) podem ajudar nisso.

Agora considere um PC de usuário. Quando o sistema é inicializado, muitos processos são secretamente iniciados, quase sempre desconhecidos para o usuário. Por exemplo, um processo pode ser inicializado para esperar pela chegada de e-mails. Outro pode ser executado em prol do programa antivírus para conferir periodicamente se há novas definições de vírus disponíveis. Além disso, processos explícitos de usuários podem ser executados, imprimindo arquivos e salvando as fotos do usuário em um pen-drive, tudo isso enquanto o usuário está navegando na Web. Toda essa atividade tem de ser gerenciada, e um sistema de multiprogramação que dê suporte a múltiplos processos é muito útil nesse caso.

Em qualquer sistema de multiprogramação, a CPU muda de um processo para outro rapidamente, executando cada um por dezenas ou centenas de milissegundos. Enquanto, estritamente falando, em qualquer dado instante a CPU está executando apenas um processo, no curso de 1s ela pode trabalhar em vários deles, dando a ilusão do paralelismo. Às vezes, as pessoas falam em pseudoparalelismo neste contexto, para diferenciar do verdadeiro paralelismo de hardware dos sistemas multiprocessadores (que têm duas ou mais CPUs compartilhando a mesma memória física). Ter controle sobre múltiplas atividades em paralelo é algo difícil para as pessoas realizarem. Portanto, projetistas de sistemas operacionais através dos anos desenvolveram um modelo conceitual (processos sequenciais) que torna o paralelismo algo mais fácil de lidar.

Pseudoparalelismo = Simular a Execução de várias tarefas ao mesmo tempo. O processador fica somente uma fatia de tempo em cada processo (time slice);

- Revezamento dos processos no processador;
- Trocas rápidas entre os processos;
- Escalonamento;
- Multiprogramação

Paralelismo = Só existe quando temos mais de um processador ou mais de um núcleo.

- Vários processadores trabalhando ao mesmo tempo;
- Vários núcleos de um mesmo processador;

O modelo de processo

Nesse modelo, todos os softwares executáveis no computador, às vezes incluindo o sistema operacional, são organizados em uma série de processos sequenciais, ou, simplesmente, processos. Um processo é apenas uma instância de um programa em execução, incluindo os valores atuais do contador do programa, registradores e variáveis. Conceitualmente, cada processo tem sua própria CPU virtual. Na verdade, a CPU real troca a todo momento de processo em processo, mas, para compreender o sistema, é muito mais fácil pensar a respeito de uma coleção de processos sendo executados em (pseudo) paralelo do que tentar acompanhar como a CPU troca de um programa para o outro.

- Um processo recebe várias informações adicionais;

Multiprogramação

Procedimento de trocas rápidas entre os processos que acessam a CPU.

Com o chaveamento rápido da CPU (time slice) entre os processos, a taxa pela qual um processo realiza a sua computação não será uniforme e provavelmente nem reproduzível se os mesmos processos forem executados outra vez. Desse modo, processos não devem ser programados com suposições predefinidas sobre a temporização.

- As trocas rápidas acontecem pelo compartilhamento de tempo.
- Cada processo leva um tempo específico para ser executado.

A ideia fundamental aqui é que um processo é uma atividade de algum tipo.

Um processo é uma atividade

Partes de um processor:

* Um **programa** (Processo em execução)
* Uma **entrada** (Informações de pedidos)
* Uma **saída** (Informações de resposta)
* Um **estado**. (Em execução, Pronto , Bloqueado)

Um único processador pode ser compartilhado entre vários processos, com algum algoritmo de escalonamento sendo usado para determinar quando parar o trabalho em um processo e servir outro. Em comparação, um programa é algo que pode ser armazenado em disco sem fazer nada.

Vale a pena observar que se um programa está sendo executado duas vezes, é contado como dois processos. Por exemplo, muitas vezes é possível iniciar um processador de texto duas vezes ou imprimir dois arquivos ao mesmo tempo, se duas impressoras estiverem disponíveis.

Criação de processos

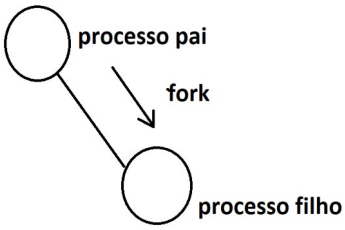
Sistemas operacionais precisam de alguma maneira para criar processos. Em sistemas muito simples, ou em sistemas projetados para executar apenas uma única aplicação (por exemplo, o controlador em um forno micro-ondas), pode ser possível ter todos os processos que serão em algum momento necessários quando o sistema for ligado. Em sistemas para fins gerais, no entanto, alguma maneira é necessária para criar e terminar processos, na medida do necessário, durante a operação. Vamos examinar agora algumas das questões. Quatro eventos principais fazem com que os processos sejam criados:

Um processo pode ser criado em 4 situações:

1. Inicialização do sistema.
2. Execução de uma chamada de sistema de criação de processo por um processo em execução.
3. Solicitação de um usuário para criar um novo processo.
4. Início de uma tarefa em lote.

- BOOT
- Chamada de Sistema
- Solicitação do usuário
- Tarefa em lote

Obs.: No **UNIX**, há apenas uma **chamada de sistema** para criar um novo processo: **fork.** Essa chamada cria um clone exato do processo que a chamou. Após a fork, os dois processos, o pai e o filho, têm a mesma imagem de memória, as mesmas variáveis de ambiente e os mesmos arquivos abertos.



Daemons

Processos que ficam em **segundo plano** para lidar com algumas atividades, como e-mail, páginas da web, notícias, impressão e assim por diante.

- Processos que são executados porém não conseguimos visualizar;
- Anti-vírus;

Término dos processos

Após um processo ter sido criado, ele começa a ser executado e realiza qualquer que seja o seu trabalho. No entanto, nada dura para sempre, nem mesmo os processos. Cedo ou tarde, o novo processo terminará, normalmente devido a uma das condições a seguir:

Quatro situações que um processo pode ser terminado:

1. Saída normal (voluntária).
2. Erro fatal (involuntário).
3. Saída por erro (voluntária).
4. Morto por outro processo (involuntário).

A maioria dos processos termina por terem realizado o seu trabalho. Quando um compilador termina de traduzir o programa dado a ele, o compilador executa uma chamada para dizer ao sistema operacional que ele terminou. Essa chamada é

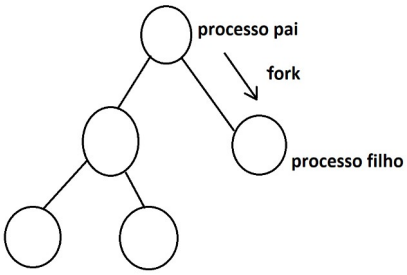
Chamadas de sistemas são os comando do sistema operacional para os programas:

exit = Chamada de sistema de **término** de processo em **UNIX**.
ExitProcess = Chamada de sistema de **término** de processo no **Windows**.

Hierarquias de processos

Em alguns sistemas, quando um processo cria outro, o processo pai e o processo filho continuam a ser associados de certas maneiras. O processo filho pode em si criar mais processos, formando uma hierarquia de processos. Observe que, diferentemente das plantas e dos animais que usam a reprodução sexual, um processo tem apenas um pai (mas zero, um, dois ou mais filhos). Então um processo lembra mais uma hidra do que, digamos, uma vaca.

- Um processo pai só termina quando os processos filho terminam;



Grupo de processos

Todos os processos e seus descendentes

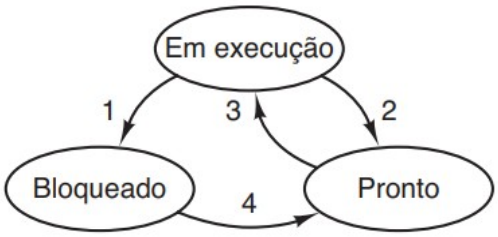
O Windows não tem conceito de uma hierarquia de processos. Todos os processos são iguais. O único indício de uma hierarquia ocorre quando um processo é criado e o pai recebe um identificador especial (chamado de handle) que ele pode usar para controlar o filho. No entanto, ele é livre para passar esse identificador para algum outro processo, desse modo invalidando a hierarquia. Processos em UNIX não podem deserdar seus filhos.

Estados de processos

Embora cada processo seja uma entidade independente, com seu próprio contador de programa e estado interno, processos muitas vezes precisam interagir entre si. Um processo pode gerar alguma saída que outro processo usa como entrada.

Os três estados nos quais um processo pode se encontrar são:

1. **Em execução** (realmente usando a CPU naquele instante).
2. **Pronto** (executável, temporariamente parado para deixar outro processo ser executado).
3. **Bloqueado** (incapaz de ser executado até que algum evento externo aconteça).



Transição entre processos:

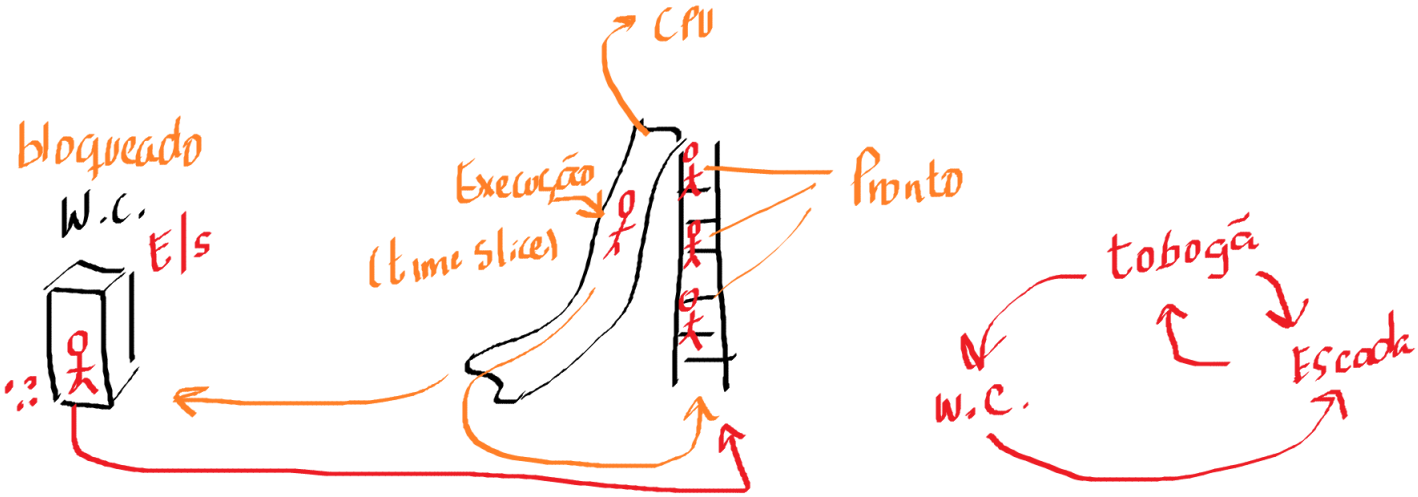
Execução → Bloqueado
Execução → Pronto
Pronto → Execução
Bloqueado → Pronto

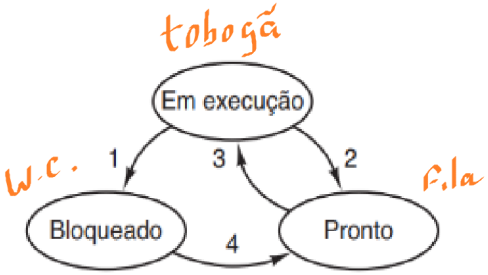
Execução → Bloqueado = Quando ele vai executar uma entrada e saída

Execução → Pronto = Acabou o time slice;

Pronto → Execução = O time slice do processo anterior acabou.

Bloqueado → Pronto = Quando o processo termina a entrada e saída;





Cpu-Bound = Quando o processo passa a maior parte do tempo em processamento (**Em execução**)
(Mais prejudicial para o desempenho)

Io-Bound = Quando o processo passa a maior parte do tempo executando entrada e saída (**Bloqueado**)
(Menos prejudicial para o desempenho)

Como apresentado na Figura, quatro transições são possíveis entre esses três estados.

A transição 1 ocorre quando o sistema operacional descobre que um processo não pode continuar agora. Em alguns sistemas o processo pode executar uma chamada de sistema, como em pause, para entrar em um estado bloqueado. Em outros, incluindo UNIX, quando um processo lê de um pipe ou de um arquivo especial (por exemplo, um terminal) e não há uma entrada disponível, o processo é automaticamente bloqueado.

As transições 2 e 3 são causadas pelo escalonador de processos, uma parte do sistema operacional, sem o processo nem saber a respeito delas. A transição 2 ocorre quando o escalonador decide que o processo em andamento foi executado por tempo suficiente, e é o momento de deixar outro processo ter algum tempo de CPU. A transição 3 ocorre quando todos os outros processos tiveram sua parcela justa e está na hora de o primeiro processo chegar à CPU para ser executado novamente. O escalonamento, isto é, decidir qual processo deve ser executado, quando e por quanto tempo, é um assunto importante; nós o examinaremos mais adiante neste capítulo. Muitos algoritmos foram desenvolvidos para tentar equilibrar as demandas concorrentes de eficiência para o sistema como um todo e justiça para os processos individuais. Estudaremos algumas delas ainda neste capítulo.

A transição 4 se verifica quando o evento externo pelo qual um processo estava esperando (como a chegada de alguma entrada) acontece. Se nenhum outro processo estiver sendo executado naquele instante, a transição 3 será desencadeada e o processo começará a ser executado. Caso contrário, ele talvez tenha de esperar no estado de pronto por um intervalo curto até que a CPU esteja disponível e chegue sua vez.

Portanto :

Um processo pode ir de:

Execução → Bloqueado
Execução → Pronto
Pronto → Execução
Bloqueado → Pronto

Implementação de processos

Para implementar o modelo de processos, o sistema operacional mantém uma tabela (um arranjo de estruturas) chamada de tabela de processos, com uma entrada para cada um deles. (Alguns autores chamam essas entradas de blocos de controle de processo.) Essas entradas contêm informações importantes sobre o estado do processo, incluindo o seu contador de programa, ponteiro de pilha, alocação de memória, estado dos arquivos abertos, informação sobre sua contabilidade e escalonamento e tudo o mais que deva ser salvo quando o processo é trocado do estado em execução para pronto ou bloqueado, de maneira que ele possa ser reiniciado mais tarde como se nunca tivesse sido parado.

Tabela de processos

Arranjo de estruturas que contém informações sobre todos os processos do computador.
Estado do processos
Pronto, Em execução, Bloqueado
Contador de programa
Program counter = Registrador que armazena o próximo processo a ser executado.
Ponteiro de pilha
Registrador que armazena o topo da pilha.

PILHA
O primeiro a chegar é o último a sair
O último a chegar é o primeiro a sair
LIFO = Last in First Out
UEPS = Último a entra primeiro a sair

FILA
O primeiro a chegar é o primeira a ser atendido
FIFO = First in First out
PEPS = Primeiro a chegar, primeiro a sair.

- Alocação de memória
Informações sobre onde o programa está **armazenado em memória**.

- Estado dos arquivos abertos
Informações sobre os arquivos que o processo está utilizando

- Informações de escalonamento
Multiplexação = Escalonamento
Quantas vezes o processo passou pelo time-slice
Quantos time slice faltam para o processo terminar

Alguns campos de uma entrada típica de uma tabela de processos:

Gerenciamento de Processo	Gerenciamento de Memória	Gerenciamento de arquivo
• Registros • Contador de programa • Palavra de estado do programa • Ponteiro da pilha • Estado do processo • Prioridade • Parâmetros de escalonamento • ID do processo • Processo pai • Grupo de processo • Sinais • Momento em que um processo foi iniciado • Tempo de CPU usado • Tempo de CPU do processo filho • Tempo do alarme seguinte	• Ponteiro para informações sobre o segmento de texto. • Ponteiro para informações sobre o segmento de dados. • Ponteiro para informações sobre o segmento de pilha.	Diretório-raiz Diretório de trabalho Descritores de arquivo ID do usuário ID do grupo

Threads

Em sistemas operacionais tradicionais, cada processo tem um espaço de endereçamento e um único thread de controle. Na realidade, essa é quase a definição de um processo. Não obstante isso, em muitas situações, é desejável ter múltiplos threads de controle no mesmo espaço de endereçamento executando em quase paralelo, como se eles fossem (quase) processos separados (exceto pelo espaço de endereçamento compartilhado).

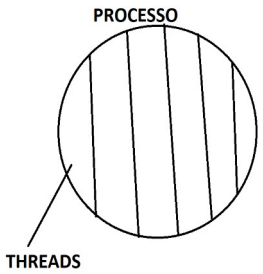
Ou seja Threads são divisões dos processos que podem ser executadas em paralelo.

Processador
8 Núcleos (Físicos)
16 Threads (Simula 16 núcleos)

THREADS

Divisões do processo que possibilitam escalonar partes do processo, ou ainda executar ao mesmo tempo.

- Execução de partes dos processos em paralelo;
- Execução precisa das partes úteis do processo;
- Aumenta a performance;



Sistema operacional Multithread

Sistema operacional capaz de dividir seus processos em threads.

Utilização de threads

Por que alguém iria querer ter um tipo de processo dentro de um processo? Na realidade, há várias razões para a existência desses miniprocessos, chamados threads. Vamos examinar agora algumas delas:

*** A principal razão para se ter threads é que em muitas aplicações múltiplas atividades estão ocorrendo simultaneamente e algumas delas podem bloquear de tempos em tempos. Ao decompormos uma aplicação dessas em múltiplos threads sequenciais que são executados em quase paralelo, o modelo de programação torna-se mais simples.**

*** Um segundo argumento para a existência dos threads é que como eles são mais leves do que os processos, eles são mais fáceis (isto é, mais rápidos) para criar e destruir do que os processos. Em muitos sistemas, criar um thread é algo de 10 a 100 vezes mais rápido do que criar um processo. Quando o número necessário de threads muda dinâmica e rapidamente, é útil se contar com essa propriedade.**

*** Uma terceira razão para a existência de threads também é o argumento do desempenho. O uso de threads não resulta em um ganho de desempenho quando todos eles são limitados pela CPU, mas quando há uma computação substancial e também E/S substancial, contar com threads permite que essas atividades se sobreponham, acelerando desse modo a aplicação.**

*** Por fim, threads são úteis em sistemas com múltiplas CPUs, onde o paralelismo real é possível.**