

PROCESSOS

Prof. Msc. Rodrigo D. Malara

PROCESSOS

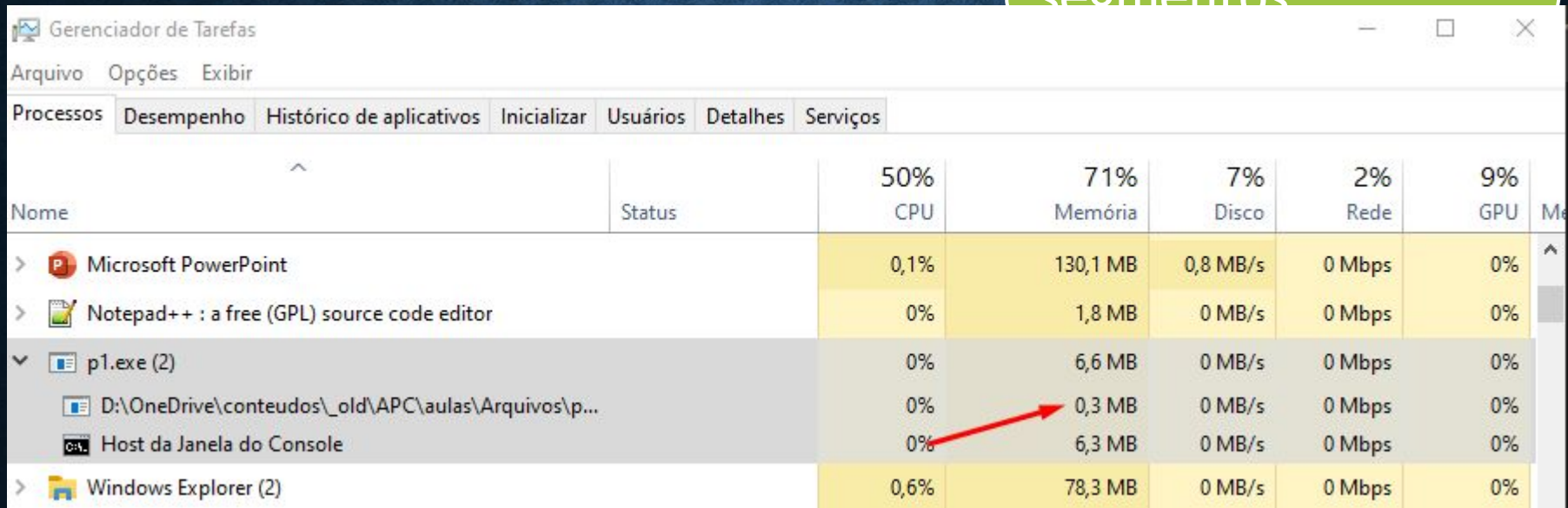
Segundo Tanenbaum, processo é “uma abstração de um programa em execução”.

Formalmente:

- Um ***processo computacional*** ou simplesmente ***processo*** pode ser entendido como uma atividade que ocorre em meio computacional, usualmente possuindo um objetivo definido, tendo duração finita e utilizando uma quantidade limitada de recursos computacionais.

ORGANIZAÇÃO DE UM PROCESSO NA MEMÓRIA

Os processos tem 4 segmentos:



Gerenciador de Tarefas

Arquivo Opções Exibir

Processos Desempenho Histórico de aplicativos Inicializar Usuários Detalhes Serviços

Nome	Status	50% CPU	71% Memória	7% Disco	2% Rede	9% GPU	Me
> Microsoft PowerPoint		0,1%	130,1 MB	0,8 MB/s	0 Mbps	0%	
> Notepad++ : a free (GPL) source code editor		0%	1,8 MB	0 MB/s	0 Mbps	0%	
▼ p1.exe (2)		0%	6,6 MB	0 MB/s	0 Mbps	0%	
D:\OneDrive\conteudos_old\APC\aulas\Arquivos\p...		0%	0,3 MB	0 MB/s	0 Mbps	0%	
Host da Janela do Console		0%	6,3 MB	0 MB/s	0 Mbps	0%	
> Windows Explorer (2)		0,6%	78,3 MB	0 MB/s	0 Mbps	0%	

ORGANIZAÇÃO DE UM PROCESSO NA MEMÓRIA

Gerenciador de Tarefas

Arquivo Opções Exibir

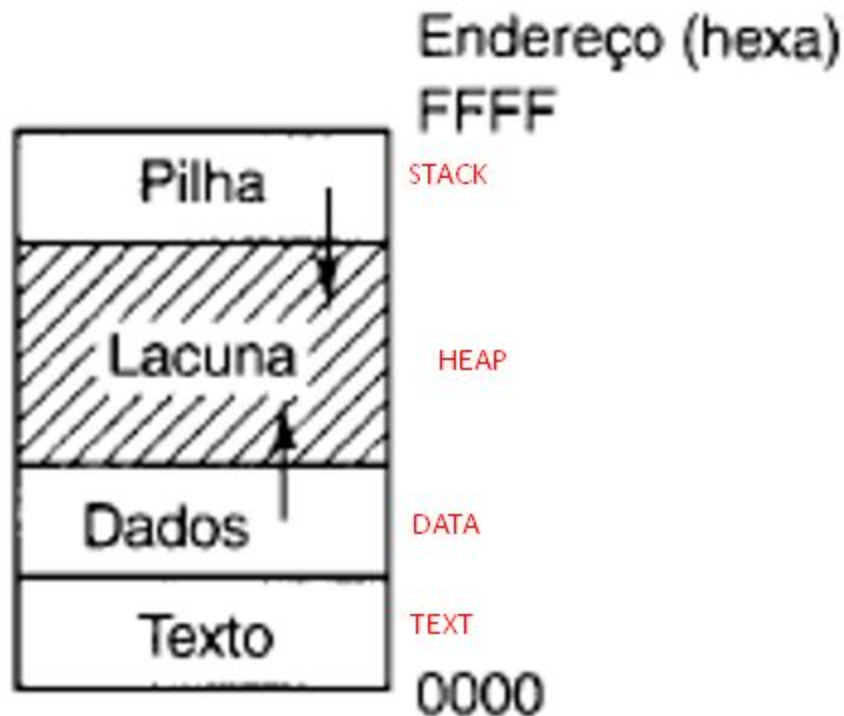
Processos Desempenho Histórico de aplicativos Inicializar Usuários Detalhes Serviços

Nome	Status	21% CPU	71% Memória	1% Disco	1% Rede	4% GPU	Me
> Microsoft PowerPoint		0%	102,9 MB	0 MB/s	0 Mbps	0%	
> Notepad++ : a free (GPL) source code editor		0%	1,8 MB	0 MB/s	0 Mbps	0%	
✓ p1.exe (2)		0%	6,6 MB	0 MB/s	0 Mbps	0%	
D:\OneDrive\conteudos_old\APC\aulas\Arquivos\p...		0%	0,2 MB	0 MB/s	0 Mbps	0%	
Host da Janela do Console		0%	6,4 MB	0 MB/s	0 Mbps	0%	
✓ p1.exe (2)		0%	6,6 MB	0 MB/s	0 Mbps	0%	
D:\OneDrive\conteudos_old\APC\aulas\Arquivos\p...		0%	0,2 MB	0 MB/s	0 Mbps	0%	
Host da Janela do Console		0%	6,4 MB	0 MB/s	0 Mbps	0%	
✓ p1.exe (2)		0%	6,5 MB	0 MB/s	0 Mbps	0%	
D:\OneDrive\conteudos_old\APC\aulas\Arquivos\p...		0%	0,2 MB	0 MB/s	0 Mbps	0%	
Host da Janela do Console		0%	6,3 MB	0 MB/s	0 Mbps	0%	
> Windows Explorer (2)		0,4%	80,3 MB	0 MB/s	0 Mbps	0%	
Processos em segundo plano (116)							
> Antimalware Service Executable		0,1%	265,6 MB	0 MB/s	0 Mbps	0%	
> Aplicativo de subsistema de spooler		0%	1,3 MB	0 MB/s	0 Mbps	0%	

Menos detalhes

Finalizar tarefa

ORGANIZAÇÃO DE UM PROCESSO NA MEMÓRIA



Os processos tem 4 segmentos:

- **Texto, dados e pilha**
- No exemplo ao lado, os três estão em um espaço de endereço, mas o espaço separado para instruções e para dados também é suportado.

ORGANIZAÇÃO DE UM PROCESSO NA MEMÓRIA

Pilha (cresce para baixo a cada parâmetro de função)

Dados (cresce para cima a cada nova variável)

Código (tamanho fixo, somente de leitura)



Texto ou Text

- É a primeira região de memória dentro do processo
 - O tamanho dessa região não se altera durante a execução
- Usada para armazenar o código binário do programa em execução
- Pode ser executado por um ou mais threads
- Cada thread necessita de um núcleo para executar

Dados ou Data

- É a segunda região de memória dentro do processo
- Usada para armazenar variáveis alocadas estaticamente
 - O tamanho dessa região não se altera durante a execução
- Exemplos de alocação estática de memória em C

```
int main() {  
    int soma;  
    float preco;  
}
```


Lacuna ou Heap

- É a terceira região de memória dentro do processo
- Usada para armazenar variáveis alocadas dinamicamente
 - O tamanho dessa região geralmente muda durante a execução
 - A medida que mais memória seja alocada dinamicamente
- Exemplo de alocação dinâmica de um vetor de 1000 elementos em C

```
#define TAMANHO 1000
```

```
int main() {
```

```
    int *vet = malloc(TAMANHO * sizeof(int));
```

Pilha

- Presente em todos os processos criados pelo SO
- É a 4ª região de memória dentro do processo, para auxiliar na chamada de funções e armazenar seus parâmetros ou argumentos;
 - Ao se invocar uma função ou procedimento, a função e seus argumentos são colocados na pilha do processo (PUSH);
 - A função chamada consulta os argumentos na pilha;
 - Ao terminar, os argumentos e a função são desempilhadas (POP).

Pilha - Exemplo

```
#include <stdio.h>
```

```
int main() {  
    int a = 3;  
    float b = 2f;  
    char c = 'x';  
    printf("Ola %d, %f, %c\n", a, b, c);  
}
```

compilação

main

empilhar "string"

empilhar a

empilhar b

empilhar c

invocar o printf

desempilhar (c)

desempilhar (b)

desempilhar (a)

desempilhar ("")

executa o código

do printf

> printf retorna para

main

Estouro de Pilha ou Stack Overflow

- Ocorre se o ponteiro de pilha de chamadas exceder o limite de pilha.
- A pilha de chamadas usa uma quantidade limitada de espaço de endereçamento, geralmente determinado no início do programa.
- Quando um programa tenta usar mais espaço do que o disponível na pilha de chamadas (ou seja, quando tenta acessar a memória além dos limites da pilha de chamadas, que é essencialmente um estouro de buffer), a pilha é chamada de "estouro"
- A execução do programa é sempre terminada pelo S.O. quando o estouro de pilha ocorre

Estouro de Heap ou Heap Overflow

- Ocorre quando o processo tenta solicitar espaço de memória para o Sistema Operacional (ex: malloc ou new) mas a solicitação não pode ser atendida
- Possíveis razões:
 - O sistema computacional não possui mais memória disponível - ex: programa feito em linguagem C
 - O processo atingiu o limite máximo de memória que ele pode ocupar definido no momento do início da sua execução - ex: programa feito em Java
- O programa pode ser terminado ou não quando ocorre um estouro de heap.

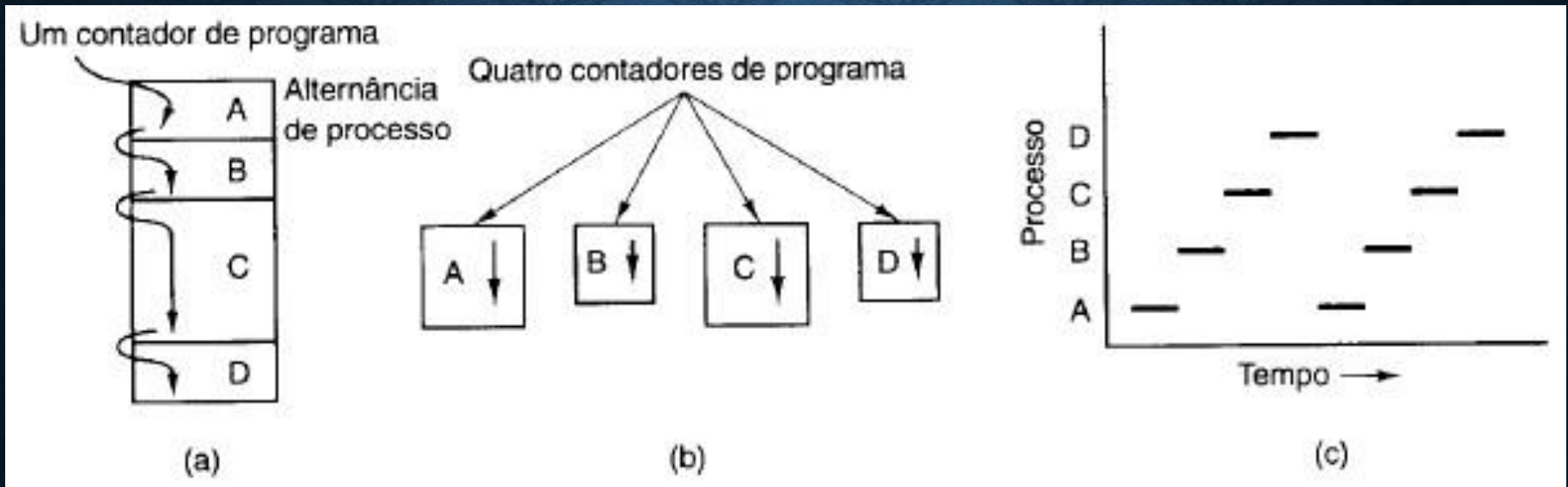
PROCESSOS

Simplificando, podemos entender um processo como um programa em execução o que envolve o **código do programa**, os **dados em uso**, os **registradores do processador**, sua **pilha (stack)** e o **contador de programa** além de outras informações relacionadas a sua execução.

Exemplos de processos computacionais

- Impressão de um documento
- Cópia de um arquivo
- A compilação de um programa
- Execução de um programa qualquer

MODELO DE PROCESSO



- a) Multiprogramação com 4 programas na memória;
- b) Modelo conceitual de 4 processos independentes;
- c) Apenas um programa está ativo num determinado instante.

MODELO DE PROCESSO

O **processo** é uma **unidade de contexto**

Contêiner de recursos utilizados por uma ou mais tarefas para sua execução.

Os processos são isolados entre si pelos mecanismos de proteção providos pelo hardware (isolamento de áreas de memória, níveis de operação e chamadas de sistema) e pela própria gerência de tarefas, que atribui os recursos aos processos (e não às tarefas), impedindo que uma tarefa em execução no processo **pa** acesse um recurso atribuído ao processo **pb**.

IMPLEMENTAÇÃO DE PROCESSOS

Para implementar o modelo de processo o SO mantém a “Tabela de Processos” em execução.

- Uma entrada para cada processo
- Armazena todos os dados sobre os processos
- Exemplo: entrada na tabela de processos

Gerenc. de processos

ID processo
Registradores
Contador de programa
Palavra de estado do programa
Ponteiro de pilha

Prioridade
Param. Escalonamento
Processo pai
Grupo do processo

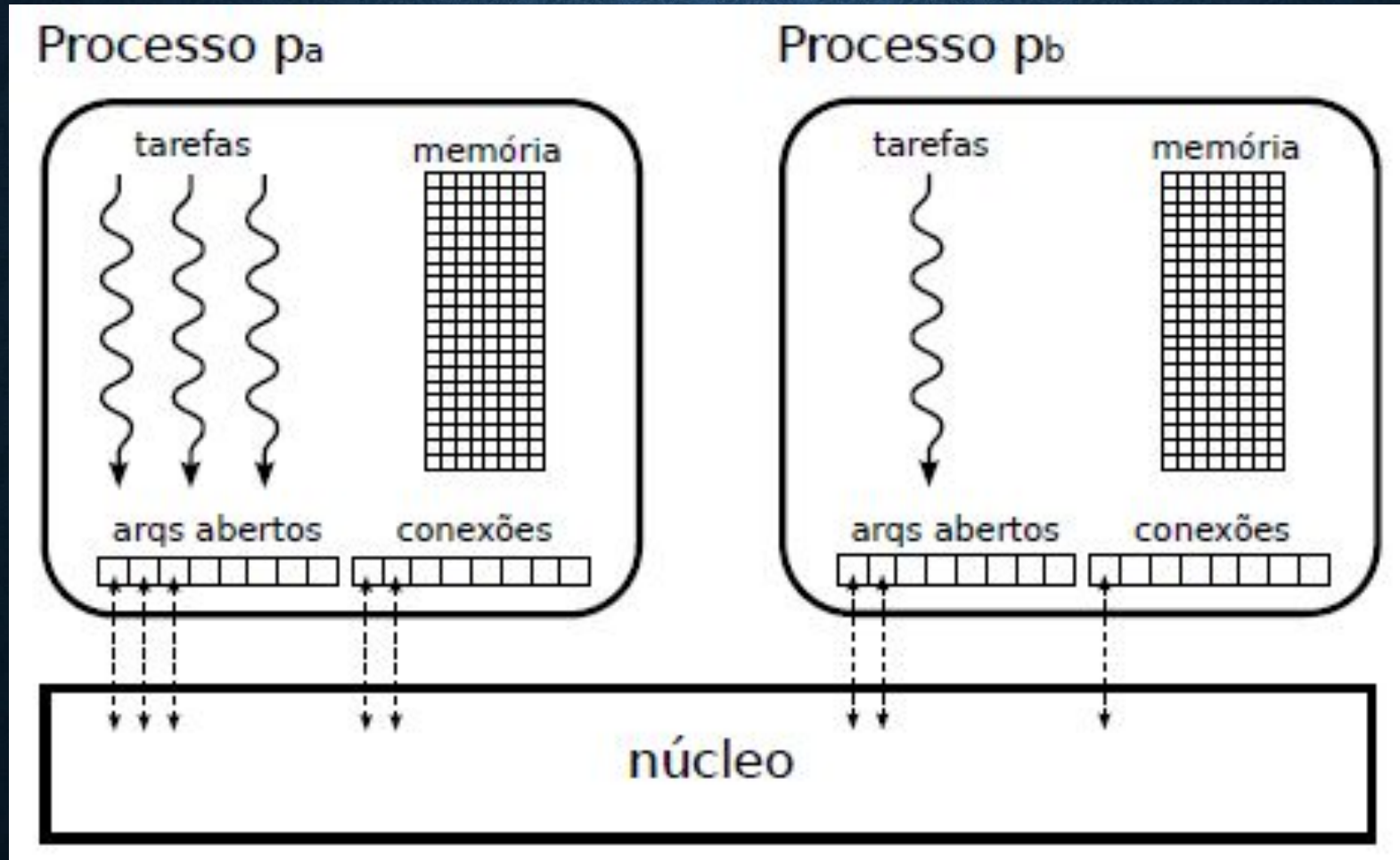
Gerenc. Memória

Ponteiro para segmento de Código
Ponteiro para segmento de dados
Ponteiro para segmento de lacuna (heap)
Ponteiro para segmento de pilha

Gereciamento de Arquivos

Diretório-raiz
Diretório de trabalho
Descritores de arquivos ID
Usuario (dono)
ID Grupo (dono)

MODELO DE PROCESSO



O processo visto como um contêiner de recursos.

CRIAÇÃO DE PROCESSOS

Eventos que levam a criação de processos

- Inicialização do Sistema (no Unix: init);
- Processo envia chamada ao sistema requisitando a criação de um novo processo filho (ex: Apache);
- Usuário inicializa algum programa;
- Sistema de jobs inicializa um trabalho.
- Etc...

FINALIZAÇÃO DE UM PROCESSO

Condições que levam à finalização do processo

- Saída normal: voluntária;
- Saída por erro: voluntária;
- Erro Fatal: involuntária;
- Envia-se um sinal para o processo para finalizá-lo

ESTADOS DOS PROCESSOS

- Possíveis estados para os processos

• Em Execução	• Bloqueado
• Pronto	• Terminado



- Escalonador na camada mais baixa de um SO:
 - Gerencia interrupções e agendamento de CPU.
- Todos os processos são sequenciais.

ESTADOS DOS PROCESSOS

- Diagrama de estado:

- **Executando** (realmente utilizando a CPU nesse instante).
- **Pronto** (executável; temporariamente parado para permitir que outro processo execute);
- **Bloqueado** (incapaz de executar até que algum evento externo aconteça).



1. O processo bloqueia para entrada
2. O agendador seleciona outro processo
3. O agendador seleciona esse processo
4. A entrada torna-se disponível

Troca de Contexto

- Ocorre quando o S.O. precisa retirar um processo de execução A e colocar outro processo B no processador para executar
- O estado atual da execução do processo A que vai ser retirado precisa ser salvo na sua respectiva entrada na tabela de processos
 - Os valores de todos os registradores do processador precisam ser salvos em memória
- Então os valores dos registradores do processo B que será iniciado no processador, devem ser copiados da tabela de processos (na RAM) para os registradores
- Assim que essa troca dos registradores for feita, a execução do programa B pode ser iniciada

Executando para Bloqueado

- Ocorre quando o processo precisa de apoio do S.O. como:
 - Interação com o usuário através de teclado e monitor
 - Mais área de memória para variáveis alocadas dinamicamente (malloc ou new)
 - Manipulação de arquivos
 - Interação com demais dispositivos dentre outros
 - Ou seja, quando o processo faz uma chamada ao Sistema Operacional ou System Call
- Essas tarefas podem demorar uma quantidade de tempo variável, dependendo do que foi solicitado ao S.O.

Executando para Bloqueado

- Então, o S.O. coloca o processo no estado Bloqueado até que a atividade que o processo solicitou seja terminada.
 - Caso a atividade solicitada envolva o uso de periféricos como discos rígidos, por exemplo, o desbloqueio só ocorrerá quando uma interrupção do dispositivo ocorrer, indicando que a atividade foi finalizada pelo dispositivo.
- Quando a atividade é terminada, o S.O. coloca o processo no estado Pronto (seta nro. 4 do diagrama de estados)

Executando para Pronto

- Quando um processo realiza apenas processamento usando processador e memória RAM, não há como outro programa entrar em execução em um determinado núcleo

Exemplo - o programa abaixo irá ficar executando até que ele seja cancelado pelo usuário

```
int main() {  
    int soma = 0;  
    while (1) { // loop infinito  
        soma = soma + 1;  
    }  
}
```


Executando para Pronto

- Para esse tipo de programa dar lugar para outro programa executar, o S.O. recorre a interrupções de relógio (clock)
- Ao iniciar o computador, o S.O. configura o clock para emitir uma interrupção de relógio a cada x milisegundos (ex: 70 ms)
- A essa quantidade de tempo, é dado o nome de *Quantum*
 - *Tempo máximo que um processo pode permanecer em execução*
- Dessa forma, o S.O. consegue remover o programa que estava monopolizando o núcleo do processador.
- O programa interrompido é colocado no estado pronto, para que ele seja escalonado novamente, conforme as políticas de escalonamento do escalonador de processos,

Pronto para Executando

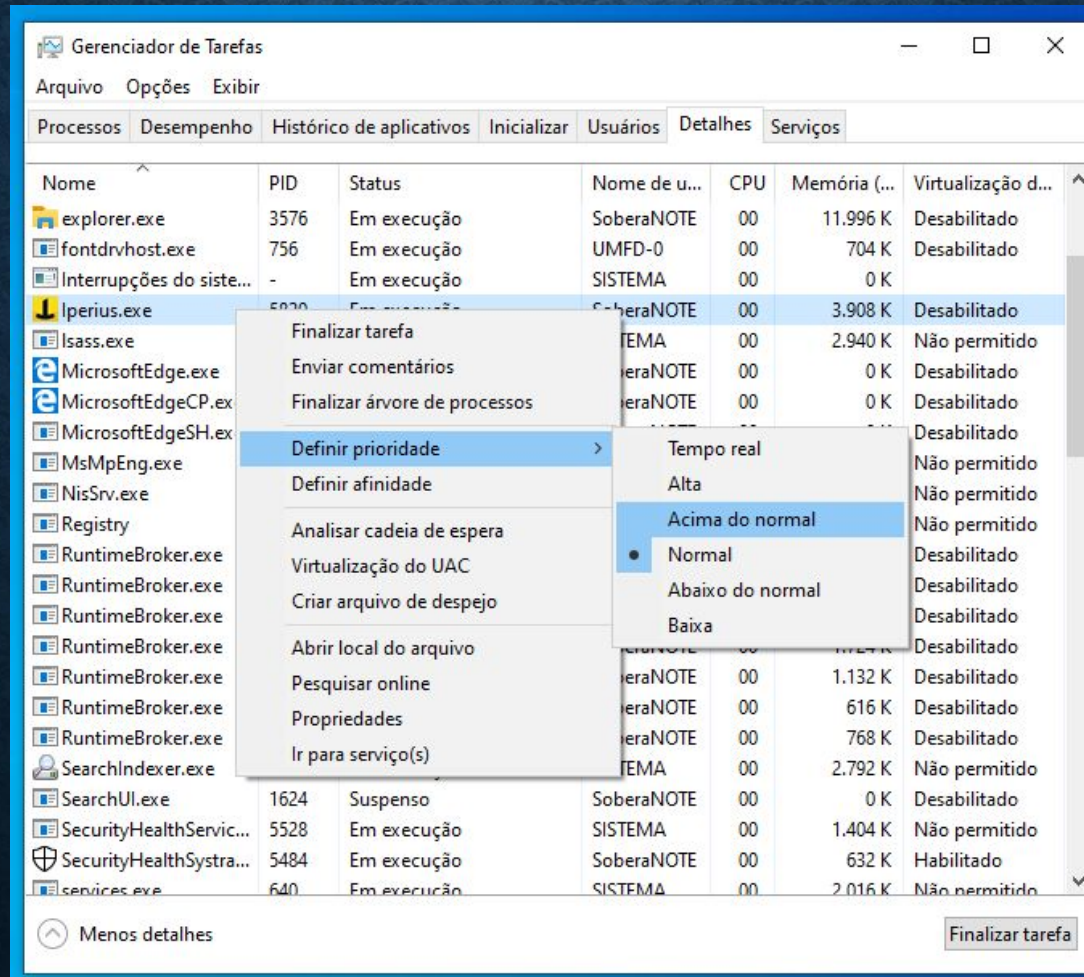
- Nos últimos 50 anos, o processador tem sido o componente de maior performance em um sistema computacional.
- Caso haja apenas um processo a ser executado, o processador ficará muito ocioso pois os outros componentes são muito lentos perto do processador (memória, disco rígido/SSD, etc)
 - E quando o processo precisar dos demais componentes, o processador ficaria ocioso, o que não é algo interessante
- Dessa forma, desde meados dos anos 60, os S.O.s mantêm diversos programas carregados na memória RAM prontos para executar

Pronto para Executando

- Os processos que estão aguardando a sua vez de utilizar o processador, ficam no estado Pronto
- O escalonador de processos é o componente do S.O. que decide qual processo deve executar e quando
- Quando o usuário precisa interferir na lógica de escalonamento, ele pode usar a definição de prioridades para os processos.
- No Windows existem 6 prioridades diferentes
- No Unix são 21 prioridades diferentes, de -20 a +20, sendo que as maiores prioridades são a de menor valor, ex: -20

Pronto para Executando

- Prioridades de processos no Windows



Pronto para Executando

- Prioridades de processos no Unix

```
top - 00:29:43 up 94 days, 24 min, 1 user, load average: 0.00, 0.01, 0.00
Tasks: 44 total, 1 running, 43 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.5%us, 0.2%sy, 0.0%ni, 99.3%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 4194304k total, 858904k used, 3335400k free, 0k buffers
Swap: 0k total, 0k used, 0k free, 0k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
19547	root	18	0	2119m	446m	25m	S	1.0	10.9	3:56.91	java
1	root	15	0	2152	672	580	S	0.0	0.0	0:35.70	init
1435	root	18	0	38236	31m	2560	S	0.0	0.8	0:03.55	spamd
9428	root	15	0	7200	1060	664	S	0.0	0.0	0:39.08	sshd
18211	root	15	0	10056	2856	2296	S	0.0	0.1	0:01.70	sshd
18293	usuarios	15	0	10056	1752	1180	S	0.0	0.0	0:02.62	sshd
18306	usuarios	18	0	6652	1536	1180	S	0.0	0.0	0:00.00	sftp-server
26458	root	15	0	38688	31m	1908	S	0.0	0.8	0:17.55	spamd
27974	root	15	-4	2256	556	344	S	0.0	0.0	0:00.00	udev
28125	root	20	0	3660	1244	1072	S	0.0	0.0	0:00.00	mysqld safe
28199	mysql	15	0	183m	69m	5776	S	0.0	1.7	89:42.16	mysqld
28218	root	18	0	32692	576	472	S	0.0	0.0	0:31.07	brcm_iscsiuio
28519	root	15	0	1808	568	480	S	0.0	0.0	1:59.85	syslogd
28643	root	18	0	1213m	215m	8776	S	0.0	5.3	257:49.33	java
29711	root	18	0	2828	860	692	S	0.0	0.0	0:00.00	xinetd
30030	mailnull	15	0	10356	2652	2160	S	0.0	0.1	0:17.45	exim
30035	root	15	0	10056	2956	2396	S	0.0	0.1	0:00.01	sshd

HIERARQUIA DE PROCESSOS

Processo-pai cria processo-filho

- Processos filho podem criar seus próprios processos.

Criação de uma hierarquia

- No UNIX isso se chama "grupo de processos".

O MS Windows não possui o conceito de hierarquia

- Todos os processos são iguais.