

SISTEMAS OPERACIONAIS

ADS

SANDRO ROBERTO ARMELIN

PROCESSOS



Arthur de Azevedo

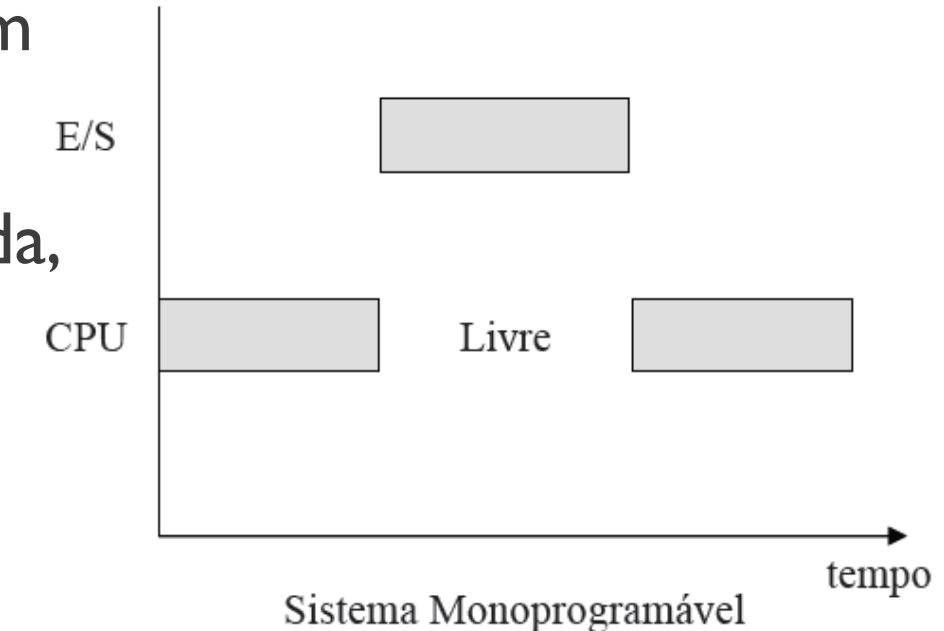
CONCORRÊNCIA

- Sistema Operacional – Visto como um conjunto de rotinas que executam concurrentemente de forma ordenada.
- Possibilidade de o processador executar instruções em paralelo com as operações e E/S permite que diversas tarefas sejam executadas concurrentemente.
- Sistemas **Multiprogramáveis** – Surgiram a partir da limitação dos sistemas **monoprogramáveis** (**Processador dedicado**).

TIPOS DE SISTEMAS OPERACIONAIS

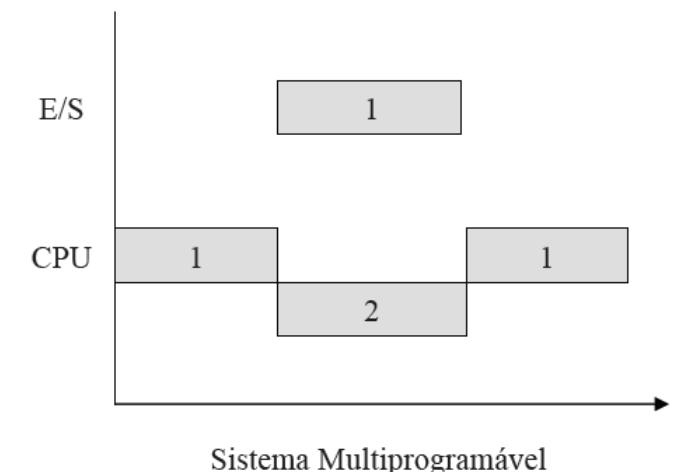
■ Sistemas Monoprogramáveis/Monotarefa

- Voltados tipicamente para a execução de um único programa.
- Qualquer outra aplicação, para ser executada, deveria aguardar o término do programa corrente.
- Processador, a memória e os periféricos permanecem exclusivamente dedicados à execução de um único programa.



TIPO DE SISTEMAS OPERACIONAIS

- **Sistemas Multiprogramáveis/Multitarefa**
 - Recursos compartilhados entre os diversas aplicações: enquanto um programa espera por um evento, outros programas podem estar processando neste mesmo intervalo de tempo.
 - O sistema operacional se incumbe de gerenciar o acesso concorrente aos seus diversos recursos.



PROCESSOS.

- Conceito Fundamental e mais importante para os Sistemas Operacionais.
- **Uma abstração de um programa em execução.**
- Permite a capacidade de operações “pseudo”concorrentes, mesmo quando há apenas uma CPU.
- Base para implementação de sistemas multiprogramaveis.
- A gerencia é função do SO, que para executar os programas é associado um programa a um processo.
- **Ao executar um programa o usuário tem impressão de possuir todo processador para seu uso. Não é verdade, visto que os recursos estão sendo compartilhados e a unidade de processamento também.**

PROCESSOS.

- Considere um servidor WEB – Solicitações de páginas Web chegam de toda parte.
- Quando uma solicitação chega, o servidor verifica se a pagina necessária esta em cache. Se estiver, é enviada de volta; Se não,
- Uma solicitação de acesso ao disco é iniciada para buscá-la. Entretanto, do ponto de vista da CPU, as solicitações de acesso ao disco duram uma eternidade. Enquanto espera que a solicitação ao disco seja concluída, muitas outras solicitações podem chegar.
- Se houver múltiplos discos, algumas delas podem ser enviadas rapidamente para outro discos antes de a primeira solicitação ser atendida.
- **EVIDENTE QUE É NECESSÁRIO UMA FORMA DE CONTROLAR E MODELAR ESSA SIMULTANEIDADE.AÍ QUE ENTRA OS PROCESSOS E AS THREADS.**

PROCESSOS.

- Impressão ao usuário de PARALELISMO.
- PSEUDOPARALELISMO para diferenciar do verdadeiro paralelismo de hardware nos sistemas multiprocessadores.
- Chaveamento entre os processos em execução com taxa não uniforme.
- Única CPU? E processadores multinúcleo? Fato que um processador executa um processo de cada vez, com mais núcleos, **cada um** deles pode executar apenas **um processo por vez**.

PROCESSOS – MESMO PROGRAMA SENDO EXECUTADO 2X

- Mesma aplicação sendo executada **2 vezes: 2 processos.**
- Exemplo: Abrir Processador de texto duas vezes ou imprimir dois arquivos ao mesmo tempo.
- S.O. PODE COMPARTILHAR O **MESMO** CÓDIGO ENTRE OS PROCESSOS, ESTANDO **APENAS UMA COPIA NA MEMÓRIA**. MAS EXISTEM 2 PROCESSOS EM EXECUÇÃO.

CRIAÇÃO DOS PROCESSOS:

- **Durante inicialização do S.O.** – vários processos são criados, alguns **background** e outros **foreground**.
- **Por um outro processo em execução** – Um processo em execução faz chamadas de sistema (System calls) para criar novos processos.
- **Requisição do usuário** – Abrindo um aplicação.
- **Tarefa em lote** – Batch job.

PROCESSOS FOREGROUND E BACKGROUND

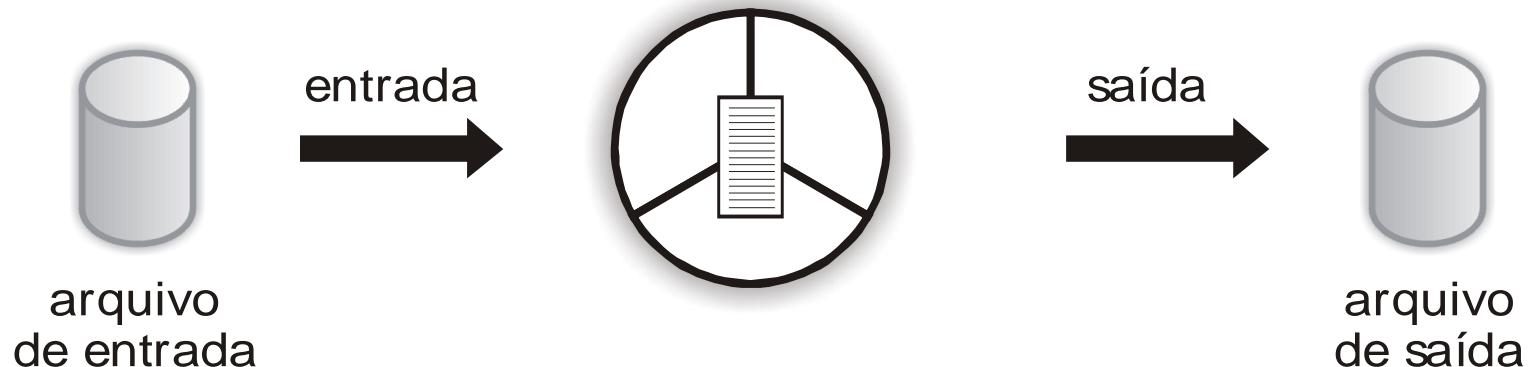
- **Foreground (Primeiro plano)** – Interagem com usuários.. .
- **Background (Segundo plano)** – Não estão associados a usuários em particular.
 - Processo designado a aceitar mensagens eletrônicas sendo recebidas. Fica a maior parte do tempo inativo mas surgindo quando mensagem chega.

PROCESSOS FOREGROUND E BACKGROUND

(a) Processo Foreground



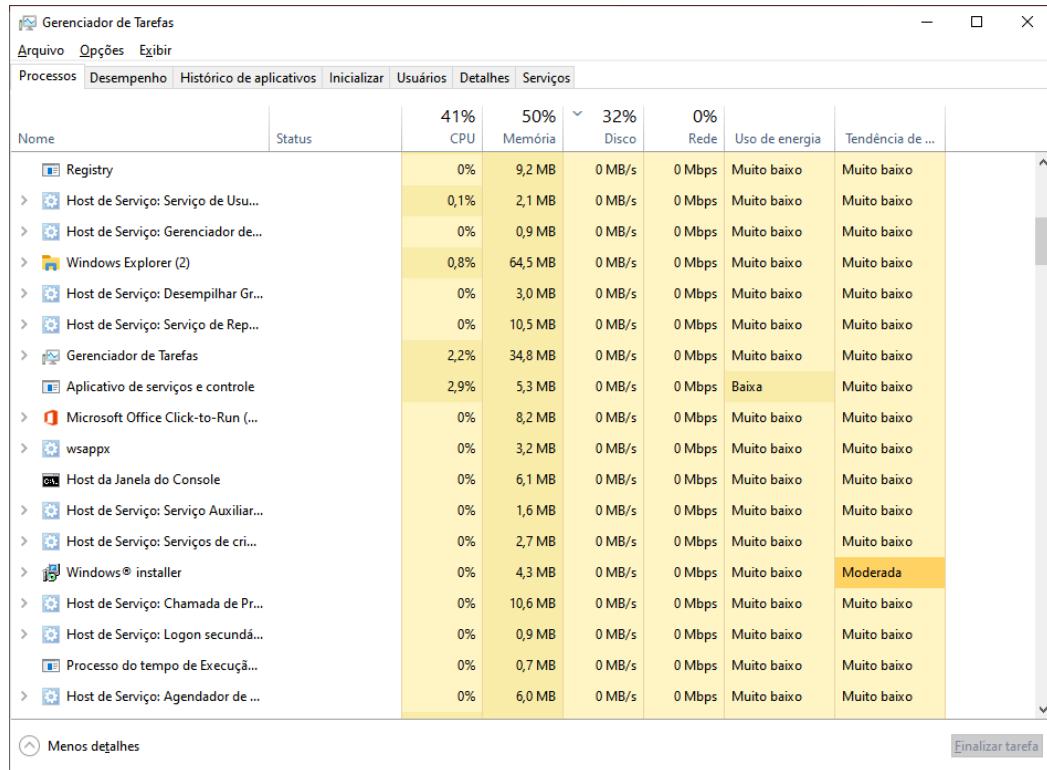
(b) Processo Background



TERMINO DE PROCESSOS, APÓS EXECUÇÃO DO SEU TRABALHO:

- **Saída normal** – Voluntária – Terminam o seu trabalho
- **Saída por erro** – Voluntária – Processo descobre um erro. Ex: digitando um comando inexistente por exemplo.
- **Erro fatal** – Erro no programa. Ex: referencia à memória inexistente ou divisão por zero.
- **Cancelamento por outro processo** – Cancelamento do processo. Kill no Linux.

GERENCIAMENTO DE PROCESSOS WINDOWS X LINUX



```
Tarefas: 110 total, 1 executando, 109 dormindo, 0 parado, 0 zumbi
%Cpu(s): 0,3 us, 1,3 sy, 0,0 ni, 98,4 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
KiB Mem: 507568 total, 308968 used, 198600 free, 43932 buffers
KiB Swap: 522236 total, 0 used, 522236 free, 151400 cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1597 root 20 0 5288 1372 1028 R 1,6 0,3 0:00.23 top
  6 root 20 0 0 0 0 S 0,3 0,0 0:00.26 kworker/u2:0
  1 root 20 0 4048 2344 1340 S 0,0 0,5 0:04.46 init
  2 root 20 0 0 0 0 S 0,0 0,0 0:00.00 kthreadd
  3 root 20 0 0 0 0 S 0,0 0,0 0:01.18 ksoftirqd/0
  4 root 20 0 0 0 0 S 0,0 0,0 0:00.00 kworker/0:0
  5 root 0 -20 0 0 0 S 0,0 0,0 0:00.00 kworker/0:0H
  7 root rt 0 0 0 0 S 0,0 0,0 0:00.00 migration/0
  8 root 20 0 0 0 0 S 0,0 0,0 0:00.00 rcu_bh
  9 root 20 0 0 0 0 S 0,0 0,0 0:00.65 rcu_sched
10 root rt 0 0 0 0 S 0,0 0,0 0:00.00 watchdog/0
11 root 0 -20 0 0 0 S 0,0 0,0 0:00.00 khelper
12 root 20 0 0 0 0 S 0,0 0,0 0:00.00 kdevtmpfs
13 root 0 -20 0 0 0 S 0,0 0,0 0:00.00 netns
14 root 0 -20 0 0 0 S 0,0 0,0 0:00.00 writeback
15 root 0 -20 0 0 0 S 0,0 0,0 0:00.00 kintegrityd
16 root 0 -20 0 0 0 S 0,0 0,0 0:00.00 bioset
17 root 0 -20 0 0 0 S 0,0 0,0 0:00.00 kworker/u3:0
18 root 0 -20 0 0 0 S 0,0 0,0 0:00.00 kblockd
19 root 0 -20 0 0 0 S 0,0 0,0 0:00.00 ata_sff
20 root 20 0 0 0 0 S 0,0 0,0 0:00.00 khubd
21 root 0 -20 0 0 0 S 0,0 0,0 0:00.00 md
22 root 0 -20 0 0 0 S 0,0 0,0 0:00.00 devfreq_wq
```

root@Linux-VirtualBox:~#

WINDOWS – GERENCIADOR DE TAREFAS.

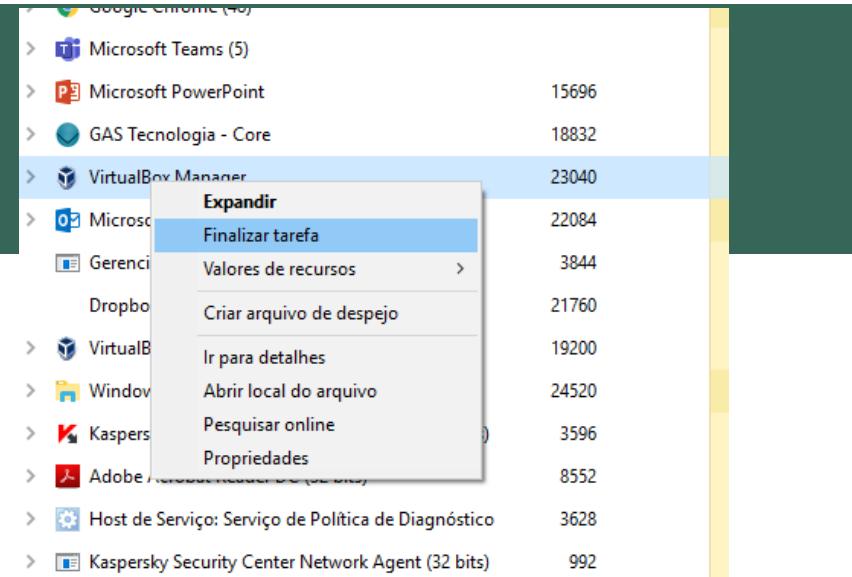
- Processos
- Desempenho
- Histórico aplicativos
- Inicializar
- Usuários
- Detalhes
- Serviços
- Gpu – Processamento gráfico
 - Junho 2017.

The screenshot shows the Windows Task Manager interface. A context menu is open over the process list, with 'Tipo' selected. The menu includes options like Status, Fornecedor, PID, Nome do processo, Linha de comando, CPU, Memória, Disco, Rede, GPU, and Mecanismo de GPU. The main table displays processes such as Google Chrome, Microsoft Teams, GAS Tecnologia - Core, Kaspersky Endpoint Security, Microsoft Outlook, VirtualBox Manager, Gerenciador de Janelas da Área de Trabalho, Microsoft PowerPoint, Dropbox, Adobe Acrobat Reader DC, Host de Serviço: Serviço de Política de Diagnóstico, Indexador do Microsoft Windows Search, Kaspersky Security Center Network Agent, Gerenciador de Tarefas, Host de Serviço: Sistema Local, and WMI Provider Host. The columns show CPU usage (5%), Memória (74%), Disco (0%), Rede (0%), GPU (0%), and Mecanismo de GPU (0%).

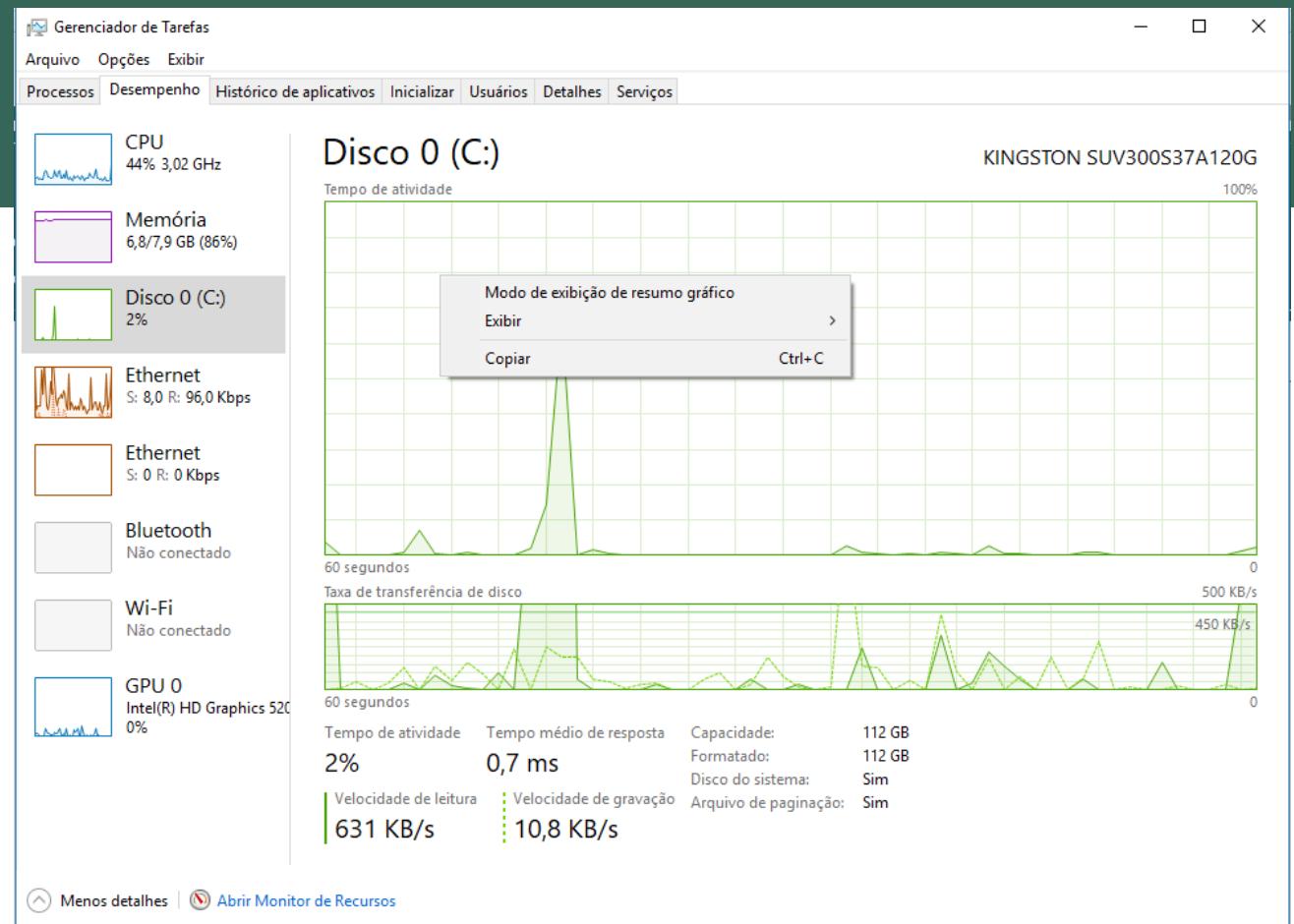
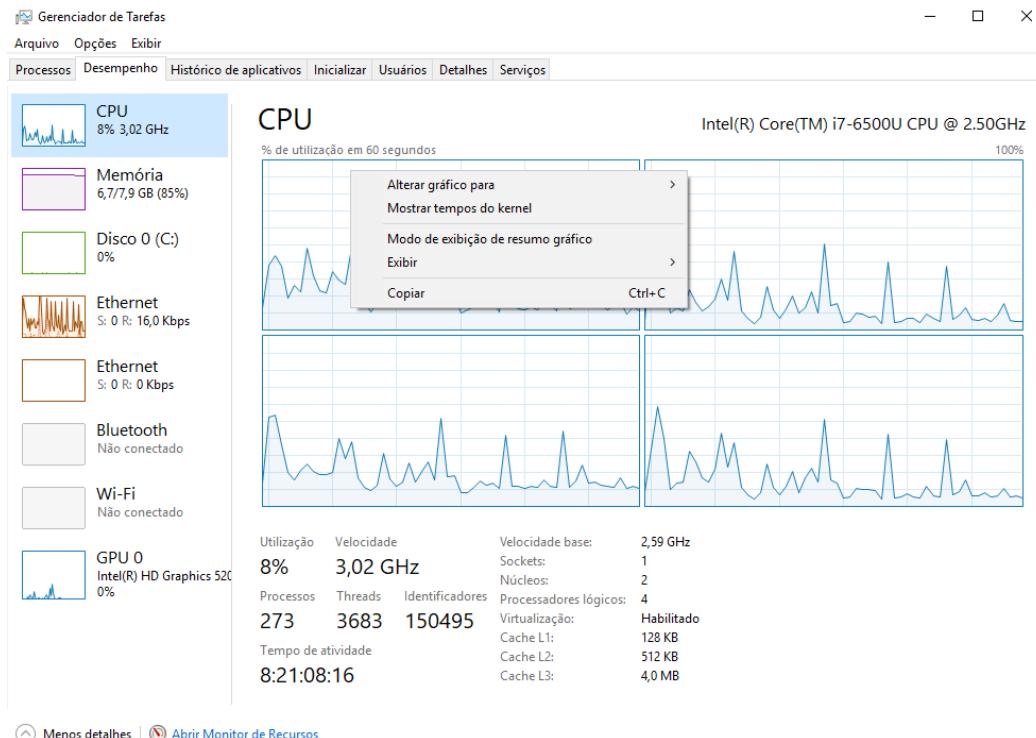
Nome	Tipo	5%	74%	0%	0%	0%	Mecanismo de GPU
	Status	1,4%	840,9 MB	0,1 MB/s	0 Mbps	0%	GPU 0 - 3D
	Fornecedor	0%	238,5 MB	0 MB/s	0 Mbps	0%	
	PID	0%	90,7 MB	0 MB/s	0 Mbps	0%	
	Nome do processo	0%	89,3 MB	0 MB/s	0 Mbps	0%	
	Linha de comando	0%	75,7 MB	0 MB/s	0 Mbps	0%	
	CPU	0,5%	71,1 MB	0 MB/s	0 Mbps	0%	
	Memória	0%	65,2 MB	0 MB/s	0 Mbps	0,1%	GPU 0 - 3D
	Disco	0%	49,6 MB	0 MB/s	0 Mbps	0%	
	Rede	0%	48,3 MB	0 MB/s	0 Mbps	0%	
	GPU	0%	47,5 MB	0 MB/s	0 Mbps	0%	
	Mecanismo de GPU	1,2%	45,4 MB	0 MB/s	0 Mbps	0%	
	Valores de recursos >	0,3%	39,5 MB	0 MB/s	0 Mbps	0%	
		0%	38,7 MB	0 MB/s	0 Mbps	0%	
		0%	33,4 MB	0 MB/s	0 Mbps	0%	
		0%	33,0 MB	0 MB/s	0 Mbps	0%	
		1,2%	22,0 MB	0 MB/s	0 Mbps	0%	
		0%	20,9 MB	0 MB/s	0 Mbps	0%	
		0%	19,5 MB	0 MB/s	0 Mbps	0%	
		0%	17,6 MB	0 MB/s	0 Mbps	0%	

PROCESSOS

- Gerenciar processos
- Expandir processos vinculados
- Arquivo de despejo – cria arquivo dmp que pode ser analisado pelo o Microsoft Debugging Tools for Windows (ferramenta de depuração da Microsoft).
- Ir para detalhes.
- Exibir local do arquivo.



DESEMPENHO



OUTROS SERVIÇOS...

Nome	Tempo de CPU	Rede	Rede limitada	Atualizações de ...
Adobe Photoshop Express	0:00:00	0 MB	0 MB	0 MB
Alarmes e Relógio	0:00:00	0 MB	0 MB	0 MB
Área de Trabalho Remota	0:00:00	0 MB	0 MB	0 MB

- Histórico de aplicativos por usuário.
- Inicializar – Aplicativos programados para inicializar com o S.O.
- Usuários conectados.
- Detalhes do processos / threads.
- Serviços.

Processos	Desempenho	Histórico de aplicativos	Iniciar	Usuários	Detalhes	Serviços
Nome	PID	Descrição				Status
AdobeARMservice	24396	Adobe Acrobat Update Service				Em execução
AJRouter		Serviço de Roteador AllJoyn				Parado
ALG		Serviço Gateway de Camada de Aplicativo				Parado
AnnIDSvc		Identidade do Aplicativo				Parado

PROCESSOS X SERVIÇOS.

- Aplicativo – Interação do usuários na área de trabalho.
- Processo – Instancia de um determinado executável. Uma aplicação = vários processos.
- Serviço – Processos que são executados em segundo plano (Background) sem interação com a área de trabalho.

LINUX.

- Comandos:
- Top
- PS – informações reduzidas dos processos daquela interface tty.
- PS aux | more | PS aux | grep –i vim
- Kill ID do processo
- Renice prioridade –p ID
 - Prioridade variando de 19 (menos significativa) a -20 (mais significativa).
 - Por padrão todos processos usuários ganham prioridade 0
 - Ex. renice -10 –p 1516

TOP

Tarefas: 110 total, 1 executando, 109 dormindo, 0 parado, 0 zumbi											
%Cpu(s): 0,3 us, 1,3 sy, 0,0 ni, 98,4 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st											
KiB Mem: 507568 total, 308968 used, 198600 free, 43932 buffers											
KiB Swap: 522236 total, 0 used, 522236 free, 151400 cacheu											
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1597	root	20	0	5288	1372	1028	R	1,6	0,3	0:00.23	top
6	root	20	0	0	0	0	S	0,3	0,0	0:00.26	kworker/u2:0
1	root	20	0	4048	2344	1240	S	0,0	0,5	0:04.48	init
2	root	20	0	0	0	0	S	0,0	0,0	0:00.00	kthreadad
3	root	20	0	0	0	0	S	0,0	0,0	0:01.18	ksoftirqd/0
4	root	20	0	0	0	0	S	0,0	0,0	0:00.00	kworker/0:0
5	root	0	-20	0	0	0	S	0,0	0,0	0:00.00	kworker/0:0H
7	root	rt	0	0	0	0	S	0,0	0,0	0:00.00	migration/0
8	root	20	0	0	0	0	S	0,0	0,0	0:00.00	rcu_bh
9	root	20	0	0	0	0	S	0,0	0,0	0:00.85	rcu_sched
10	root	rt	0	0	0	0	S	0,0	0,0	0:00.00	watchdog/0
11	root	0	-20	0	0	0	S	0,0	0,0	0:00.00	khelper
12	root	20	0	0	0	0	S	0,0	0,0	0:00.00	kdevtmpfs
13	root	0	-20	0	0	0	S	0,0	0,0	0:00.00	netns
14	root	0	-20	0	0	0	S	0,0	0,0	0:00.00	writelock
15	root	0	-20	0	0	0	S	0,0	0,0	0:00.00	kintegrityd
16	root	0	-20	0	0	0	S	0,0	0,0	0:00.00	bisect
17	root	0	-20	0	0	0	S	0,0	0,0	0:00.00	kworker/u3:0
18	root	0	-20	0	0	0	S	0,0	0,0	0:00.00	kblockd
19	root	0	-20	0	0	0	S	0,0	0,0	0:00.00	ata_sff
20	root	20	0	0	0	0	S	0,0	0,0	0:00.00	khubd
21	root	0	-20	0	0	0	S	0,0	0,0	0:00.00	md
22	root	0	-20	0	0	0	S	0,0	0,0	0:00.00	devfreq_wq

root@Linux-VirtualBox:~#

Processo “morto” porém continua existindo.

Processo em Stop.

Quanto do tamanho VIRT é realmente compartilhável (memória ou bibliotecas).

Representa o tamanho residente.

Quantidade de memória física real consumida de um processo, equivalente a %MEM

Tamanho virtual de um processo, que é a soma de memória que está utilizando.

ID do processo.

LINUX, OS PROCESSOS POSSUEM UM DOS SEGUINTE STATUS:

- Running: Processo ativo, executando.
- Waiting: "Esperando" por algum evento do sistema. I/o em disco por exemplo.
 - Em situação de falha, pode ser que um processo em estado de waiting nunca saia, configurando deadlock.
- Suspended: "Suspenso", parado pelo usuário.
- Zombie: Finalizou a execução mas ainda esta na tabela de processos porque algum processo "pai" ainda não sabe que ele terminou. Processo normal é para ser comunicado.

ALGUMAS OPÇÕES COMANDO TOP.

- Top –u root
- Outra opção htop
 - sudo apt-get install htop

The screenshot shows the htop process running in a terminal window. The window title is "Linux [Executando] - Oracle VM VirtualBox". The htop interface displays system statistics at the top, including CPU usage (Tasks: 40, 45 thr; 1 running), load average (Load average: 0.21 0.23 0.10), and uptime (Uptime: 00:02:55). Below this is a detailed list of processes, each showing its PID, user, priority (PRI), nice value (NI), virtual memory (VIRT), resident memory (RES), shared memory (SHR), CPU usage (CPU%), memory usage (MEM%), and the time it has been running (TIME+). The command being run is htop. The terminal window also includes a menu bar with options like Arquivo, Máquina, Visualizar, Entrada, Dispositivos, and Ajuda, and a status bar at the bottom with keyboard shortcuts for various functions.

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
2051	root	20	0	19224	4008	3340	R	0.7	0.1	0:00.03	htop
1	root	20	0	161M	10472	7800	S	0.0	0.3	0:01.96	/sbin/init splash
291	root	19	-1	33284	11928	10648	S	0.0	0.4	0:00.29	/lib/systemd/systemd-journald
311	root	20	0	18240	4644	3196	S	0.0	0.2	0:00.53	/lib/systemd/systemd-udevd
398	systemd-r	20	0	22620	8372	7296	S	0.0	0.3	0:00.11	/lib/systemd/systemd-resolved
410	systemd-t	20	0	90260	6212	5440	S	0.0	0.2	0:00.00	/lib/systemd/systemd-timesyncd
400	systemd-t	20	0	90260	6212	5440	S	0.0	0.2	0:00.05	/lib/systemd/systemd-timesyncd
438	messagebu	20	0	8552	5368	3780	S	0.0	0.2	0:00.51	/usr/bin/dbus-daemon --system --address=
441	root	20	0	52228	19192	11136	S	0.0	0.6	0:00.21	/usr/bin/python3 /usr/bin/networkd-d
443	root	20	0	14664	7148	6216	S	0.0	0.2	0:00.17	/lib/systemd/systemd-logind
445	root	20	0	38668	8640	7080	S	0.0	0.3	0:00.04	/usr/sbin/cupsd -1
471	root	20	0	81732	3788	3484	S	0.0	0.1	0:00.00	/usr/sbin/irqbalance --foreground
453	root	20	0	81732	3788	3484	S	0.0	0.1	0:00.01	/usr/sbin/irqbalance --foreground
477	root	20	0	387M	13768	11588	S	0.0	0.5	0:00.00	/usr/lib/udisks2/udisksd
501	root	20	0	387M	13768	11588	S	0.0	0.5	0:00.00	/usr/lib/udisks2/udisksd
583	root	20	0	387M	13768	11588	S	0.0	0.5	0:00.00	/usr/lib/udisks2/udisksd
657	root	20	0	387M	13768	11588	S	0.0	0.5	0:00.00	/usr/lib/udisks2/udisksd
456	root	20	0	387M	13768	11588	S	0.0	0.5	0:00.15	/usr/lib/udisks2/udisksd
466	avahi	20	0	8532	3620	3304	S	0.0	0.1	0:00.06	avahi-daemon: running [usuario-Virtua
483	root	20	0	244M	9568	8536	S	0.0	0.3	0:00.01	/usr/lib/accountsservice/accounts-dae
502	root	20	0	244M	9568	8536	S	0.0	0.3	0:00.01	/usr/lib/accountsservice/accounts-dae
472	root	20	0	244M	9568	8536	S	0.0	0.3	0:00.08	/usr/lib/accountsservice/accounts-dae
478	root	20	0	2500	776	708	S	0.0	0.0	0:00.01	/usr/sbin/acpid
481	root	20	0	17860	2848	2640	S	0.0	0.1	0:00.00	/usr/sbin/cron -f
537	root	20	0	307M	10300	8796	S	0.0	0.3	0:00.00	/usr/sbin/ModemManager --filter-polici
540	root	20	0	307M	10300	8796	S	0.0	0.3	0:00.00	/usr/sbin/ModemManager --filter-polici
484	root	20	0	307M	10300	8796	S	0.0	0.3	0:00.10	/usr/sbin/ModemManager --filter-polici
569	root	20	0	489M	21644	18536	S	0.0	0.7	0:00.00	/usr/sbin/NetworkManager --no-daemon
574	root	20	0	489M	21644	18536	S	0.0	0.7	0:00.04	/usr/sbin/NetworkManager --no-daemon

PS -AUX

- **USER** Usuário que iniciou o processo (dono).

PID Número único do processo.

%CPU Utilização da CPU em porcentagem.

START A hora em que o processo foi iniciado. Caso a hora seja do dia anterior, é representado pelo dia e mês.

COMMAND O comando executado e todos seus argumentos.

- Propriedade do comando OS
- A:lista todos processos.
- U:exibe nome usuário owner
- X:processos sem relação com terminal.

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START
root	1	0.0	0.0	23844	2060	?	Ss	15:28
root	2	0.0	0.0	0	0	?	S	15:28
root	3	0.0	0.0	0	0	?	S	15:28
root	4	0.0	0.0	0	0	?	S	15:28
root	5	0.0	0.0	0	0	?	S	15:28
root	6	0.0	0.0	0	0	?	S	15:28
	7	0.0	0.0	0	0	?	S	15:28

“MATANDO” UM PROCESSO.

- Criar em tty1 um arquivo de texto.
- Abrir interface tty2 e executar comando tail –f arquivotexto.
- Voltar para interface 1 e utilizar o comando para descobrir qual o ID do processo tail e utilizar o comando Kill.

Ubuntu 1Sem2015 [Executando] - Oracle VM VirtualBox

```
root@Linux-VirtualBox:~# tail -f texto  
arquivo texto
```

```
root@Linux-VirtualBox:/# kill 1807  
root@Linux-VirtualBox:/# _
```

```
root@Linux-VirtualBox:~# tail -f texto  
arquivo texto  
Terminado  
root@Linux-VirtualBox:~#
```

Tarefa	PID	Uso CPU (%)	Uso Memória (kB)	Processo	Estado	Data	Tempo Exec.	Comando
root	1792	0.0	0.0		S	23:41	0:00	[kworker/u2:0]
root	1793	0.0	0.0		S	23:46	0:00	[kworker/u2:2]
root	1807	0.0	0.1	4284 548 tty2	S+	23:49	0:00	tail -f texto
root	1809	0.0	0.2	5280 1204 tty1	R+	23:49	0:00	ps -aux

“PARANDO” UM PROCESSO

- Coloca novamente o comando tail para rodar...
- Utiliza comando ps –aux para conferir o ID e o top para conferir os processos com status parado.
- Utiliza o comando Kill –STOP ID
- Como é um comando shell ele aborda a execução.
- Kill –cont id para reativar o processo.

```
root@Linux-VirtualBox:~# tail -f texto  
arquivo texto
```

```
top - 23:53:33 up 36 min,  3 users,  load average: 0,00, 0,01, 0,05  
Tarefas: 107 total,   1 executando, 106 dormindo,   0 parado,   0 zumbi  
%Cpu(s):  0,7 us,  1,3 sy,  0,0 ni, 98,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st  
KiB Mem:  507568 total,  311552 used,  196016 free,   43940 buffers  
KiB Swap: 522236 total,        0 used, 522236 free, 152488 cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1820	root	20	0	5288	1368	1028	R	2,3	0,3	0:00.12	top
23	root	20	0	0	0	0	S	0,3	0,0	0:03.57	kworker/0:1
1	root	20	0	4048	2344	1340	S	0,0	0,5	0:04.47	init
2	root	20	0	0	0	0	S	0,0	0,0	0:00.00	kthreadd

```
root@Linux-VirtualBox:/# kill -stop 1818
```

```
root@Linux-VirtualBox:~# tail -f texto  
arquivo texto
```

```
[2]+  Parado                  tail -f texto  
root@Linux-VirtualBox:~#
```

PRIORIDADE – COMANDO RENICE

- Prioridade que vão de -20 a +20.
- Menor o número, maior sua prioridade.
 - -20 = prioridade mais alta possível
 - 19 = prioridade mais baixa possível
 - 0 = neutro
- Renice -18 idprocess

top - 13:08:22 up 46 min, 2 users, load average: 0,02, 0,04, 0,01										
Tarefas: 1 total, 0 em exec., 1 dormindo, 0 parado, 0 zumbi										
%CPU(s): 0,0 us, 0,0 sis, 0,0 ni, 100,0 oc, 0,0 ag, 0,0 ih, 0,0 is 0,0 tr										
MB mem : 2935,1 total, 2338,2 livre, 182,6 usados, 414,3 buff/cache										
MB swap: 472,5 total, 472,5 livre, 0,0 usados, 2601,7 mem dispon.										
PID USUARIO PR NI VIRT RES SHR S %CPU %MEM TEMPO+ COMANDO										
1295 root	1	-19	17080	764	700	S	0,0	0,0	0:00.00	tail

Priority Nível de prioridade definida pelo Kernel.

Valor definido pelo comando RENICE.

RENICE - PRIORIDADE

- Aumentar prioridade – fazer com que SO execute + vezes o processo.
- PS –a na outra interface (tty) para visualizar o id do processo.
- Ou, ps –a | grep tail
- No top.
- Renice -19 1295
- Top –p idprocess

```
 34 root      0 -20      0      0      0 I  0,0  0,0  0:00.00 tpm_dev_wq
1380 root     20  0 20532  3744  3156 R  0,0  0,1  0:00.20 top
1295 root     20  0 16692   764   700 S  0,0  0,0  0:00.00 tail
 312 root     20  0 18240  4568  3140 S  0,0  0,2  0:00.55 systemd-udevd
 394 systemd+ 20  0  90260  6120  5340 S  0,0  0,2  0:00.29 systemd-timesyncd
```

```
top - 13:08:22 up 46 min,  2 users,  load average: 0,02, 0,04, 0,01
Tarefas:  1 total,  0 em exec.,  1 dormindo,  0 parado,  0 zumbi
%CPU(s): 0,0 us, 0,0 sis, 0,0 ni,100,0 oc, 0,0 ag, 0,0 ih, 0,0 is 0,0 tr
MB mem : 2935,1 total, 2338,2 livre, 182,6 usados, 414,3 buff/cache
MB swap: 472,5 total, 472,5 livre, 0,0 usados, 2601,7 mem dispon.
```

PID	USUARIO	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TEMPO+ COMANDO
1295	root	1	-19	17080	764	700	S	0,0	0,0	0:00.00 tail

```
root@usuario-VirtualBox:~# renice -19 1295
1295 (process ID) com prioridade antiga 0, prioridade nova -19
root@usuario-VirtualBox:~#
```

ESTADOS DE PROCESSOS.



ESTADOS DE UM PROCESSO.

- **EM EXECUÇÃO:** Utilizando a CPU naquele momento.
- **PRONTO:** Parado para dar lugar a outro processo.
Aguardando o escalonamento.
- **BLOQUEADO:** Não pode executar pois depende de um evento externo. ***Mesmo com CPU disponível não pode executar.**

TRANSIÇÕES ENTRE OS ESTADOS.



- **T1 EXECUÇÃO PARA BLOQUEADO:** Processo bloqueia aguardando uma entrada. Evento dependente de Entrada e saída.
- **T2 EXECUÇÃO PARA PRONTO:** Escalonador seleciona outro processo. Processo já teve seu tempo necessário de CPU, dando oportunidade para outro.
- **T3 PRONTO PARA EXECUÇÃO:** Escalonador de processos seleciona um dos processos da fila. Algoritmos para decidir quando e por quanto tempo os processos deve executar. **Competição / Equilíbrio.**
- **T4 BLOQUEADO PARA PRONTO:** A entrada se torna disponível processo volta para a fila de pronto.

DE FORMA GERAL, O QUE OS PROCESSO FAZEM:

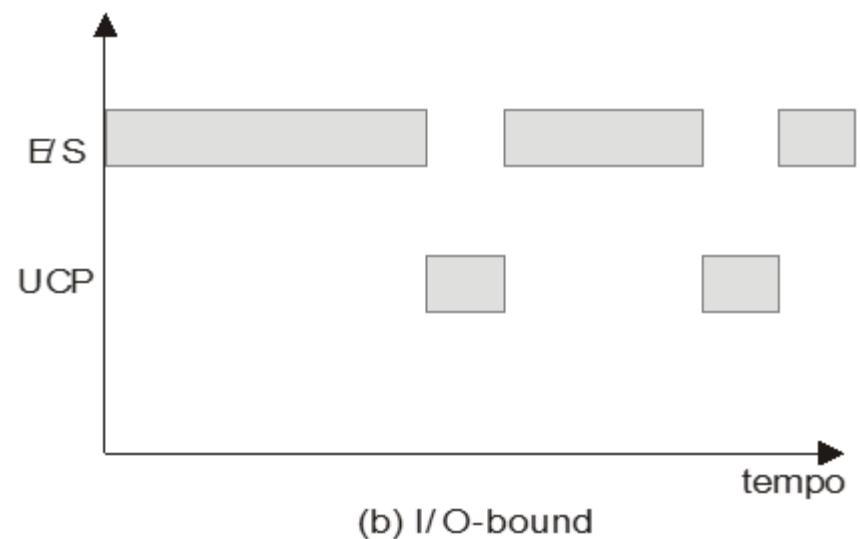
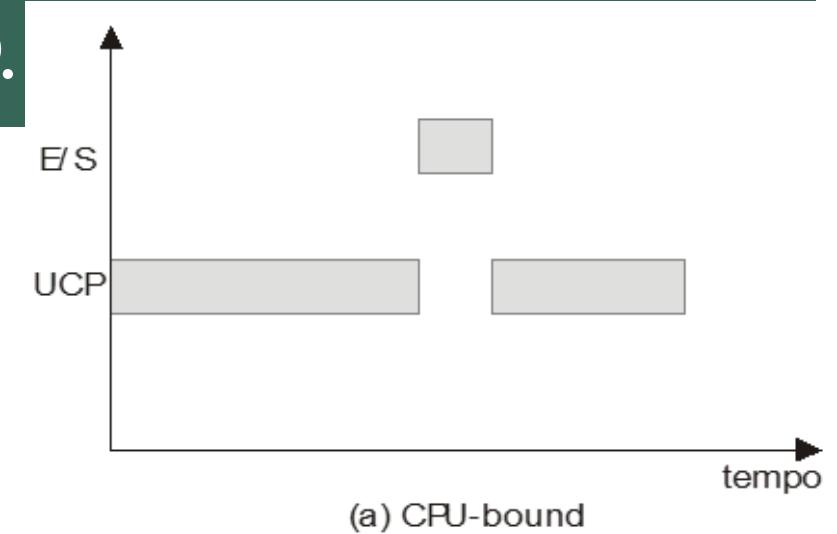
- Alguns são responsáveis por executar comandos dos usuários.
- Outros são parte do S.O. - responsabilidade de fazer requisição por serviço de arquivos ou gerenciar funcionamento de um acionador de disco.
- Em geral: PROCESSOS DE USUÁRIOS, DE DISCO, DE TERMINAIS, ETC.
- **Eles Bloqueiam quando estão a espera de que algo aconteça.**

TABELA DE PROCESSOS.

- Principal responsável pela possibilidade de implementação de processos.
- Também chamada de Process Control Blocks.
- Informações de cada processo:
 - Estado
 - ID
 - Contador de programa – Qtd de vezes que foi executado pelo processador.
 - Ponteiro da pilha
 - Alocação de memória,
 - Estado dos arquivos abertos.

PROCESSOS CPU-BOUND E I/O BOUND.

- Processo **CPU-BOUND** quando o processo passa maior parte do tempo no estado de **execução**, ou seja, utilizando mais o **processador**, realizando assim, **poucas** operações de leitura e gravação.
- Processo **I/O-BOUND** processo passa maior parte do tempo no estado de **espera**, pois realiza mais operações de **E/S**.



FUNÇÃO GETPID() E GETPPID()

- Função do padrão **POSIX** (Portable Operation System Interface) – Família de normas definidas pelo IEE para compatibilidade entre sistemas.
- **POSIX X LINUX** – Unix serviu de base para o padrão.
- `getpid ()`: retorna o ID do processo de chamada.
- `getppid ()`: retorna o ID do processo do pai do processo de chamada. Se o processo de chamada foi criado pela função `fork ()` e o processo pai ainda existir no momento da chamada. Caso contrário, essa função retornará um valor de 1, que é o ID do processo para o processo init.
- `Getpgrp()`: retorna o ID do grupo do processo caso tenha filhos (threads)

LINGUAGEM C – AMBIENTE LINUX UBUNTU.

- gcc -v
- Compilar gcc codigo.c -o codigo
- Executar: ./codigo
- **Instalando o gcc...**
- rm /var/lib/dpkg/lock
- apt-get update ou apt update
- apt-get install gcc ou apt install gcc

```
root@usuario-VirtualBox:/home# gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/8/lto-wrapper
OFFLOAD_TARGET_NAMES=nvptx-none
OFFLOAD_TARGET_DEFAULT=1
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 8.3.0-6ubuntu1' --with-bugurl=file:///usr/share/doc/gcc-8/README.Bugs --enable-languages=c,ada,c++,go,brig,d,fortran,objc,obj-c++ --prefix=/usr --with-gcc-major-version-only --program-suffix=-8 --program-prefix=x86_64-linux-gnu- --enable-shared --enable-linker-build-id --libexecdir=/usr/lib --without-included-gettext --enable-threads=posix --libdir=/usr/lib --enable-nls --enable-bootstrap --enable-clocale=gnu --enable-libstdcxx-debug --enable-libstdcxx-time=yes --with-default-libstdcxx-abi=new --enable-gnu-unique-object --disable-vtable-verify --enable-libmpx --enable-plugin --enable-default-pie --with-system-zlib --with-target-system-zlib --enable-objc-gc=auto --enable-multiarch --disable-werror --with-arch-32=i686 --with-abi=m64 --with-multilib-list=m32,m64,mx32 --enable-multilib --with-tune=generic --enable-offload-targets=nvptx-none --without-cuda-driver --enable-checking=release --build=x86_64-linux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu
Thread model: posix
gcc version 8.3.0 (Ubuntu 8.3.0-6ubuntu1)
root@usuario-VirtualBox:/home#
```

GNU nano 3.2

id.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main()
{
    system("echo 'teste \x033[01;32m mensagem\x033[01;37m.'");
    printf ("Eu sou processo %d. Meu pai é o processo %d. Id do grupo é o processo %d,",
            getpid(), getppid(), getpgrp());
    printf ("\n");
    system("ps -a | grep tty3");
    int id;
    scanf ("%d", &id);
    exit(0);
}
```

```
root@usuario-VirtualBox:/home# cat id.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main()
{
    system("echo 'teste \x033[01;32m mensagem\x033[01;37m.'");
    printf ("Eu sou processo %d. Meu pai é o processo %d. Id do grupo é o processo %d," ,getpid(), getppid(), getpgrp());
    printf ("\n");
    system("ps -a | grep tty3");
    exit(0);
}

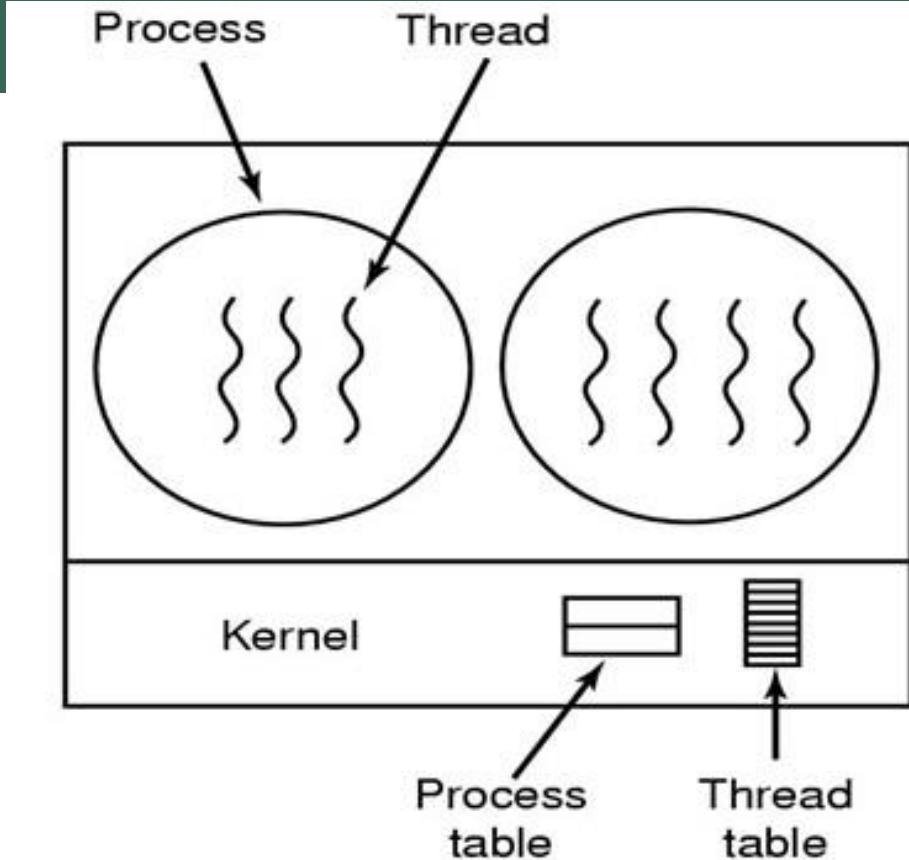
root@usuario-VirtualBox:/home#
```

AVALIANDO PROCESSOS...

```
root@usuario-VirtualBox:/home#
root@usuario-VirtualBox:/home#
root@usuario-VirtualBox:/home#
root@usuario-VirtualBox:/home# ./id
teste mensagem.
teste mensagem.
Eu sou processo 3468. Meu pai é o processo 1881. Id do grupo é o processo 3468,
1881  tty3    00:00:00 bash
3468  tty3    00:00:00 id
3471  tty3    00:00:00 sh
3472  tty3    00:00:00 ps
3473  tty3    00:00:00 grep
root@usuario-VirtualBox:/home#
```

THREADS

- **Mini processos** – Processos dentro de um processo.
- **Inicialmente CADA PROCESSO** tem um espaço de endereçamento e uma única thread.
- Processos paralelos, sub-processos, processos filho.



THREADS

- **Porque threads?**
- Há situações desejável múltiplas threads de controle para **mesmo espaço endereçamento.**
- **Processos paralelos que são executados de forma independente.**

ARGUMENTOS PARA AS THREADS

- MESMO ARGUMENTO DA EXISTENCIA DOS PROCESSOS,
Mas a divisão de tarefas para os próprios processos em mesmo
endereçamento.
- Criação de Threads são mais fáceis, pois não tem recursos
associados a elas. 100x mais rápido do que criar um processo.

THREADS – UM EXEMPLO.

- Considere um processador de texto com documento com muitas páginas.
- Alteração em uma página e na sequencia outra alteração em outra página requer uma reorganização do editor de texto.
- Thread, enquanto uma thread interage com usuário (teclado e mouse) a outra faz reformatação em segundo plano.
- Outra thread para salvamento automático do editor. Não sendo necessário bloquear o teclado e mouse.
- 3 Processos não funcionaria pois todos estariam trabalhando sobre o documento. 3 Threads que compartilham a memória comum.



PROCESSADOR DE TEXTO COM TRÊS THREADS.



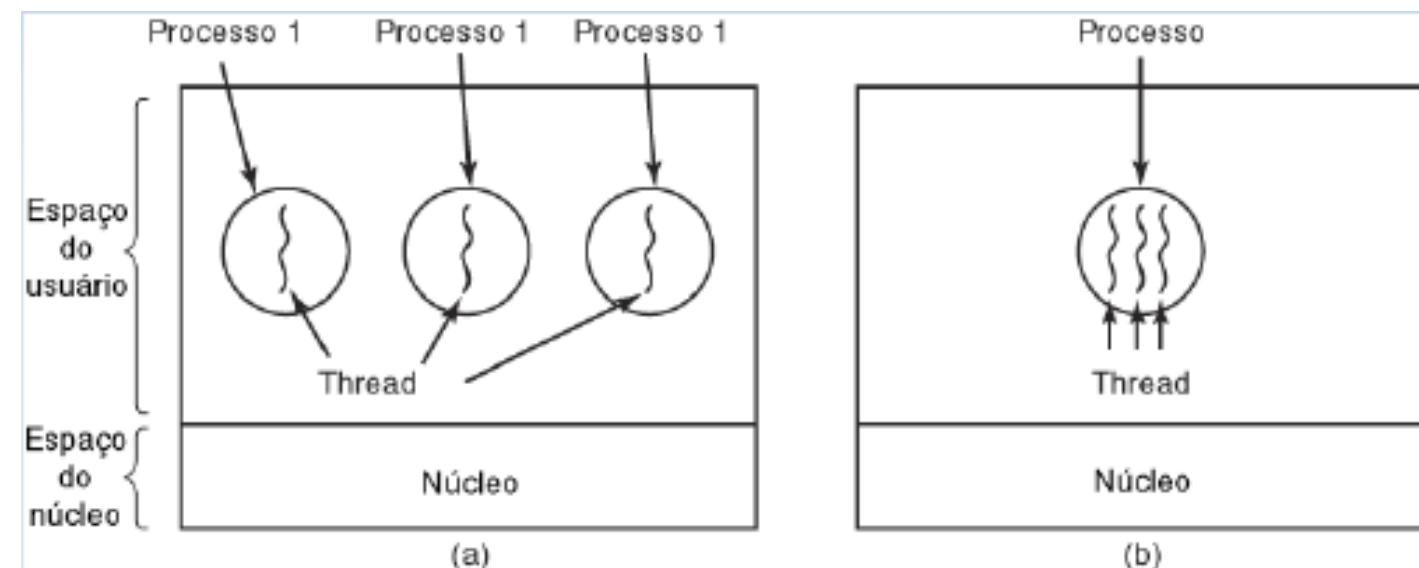
OUTRO EXEMPLO...

- Matriz de uma planilha eletrônica.
- Interação do usuário com dados de entrada em células interferindo no resultado de outras células devido a utilização de formulas.
- Uma thread em segundo plano realiza o recalculo das células.



THREADS.

- Figura (a) três processos tradicionais, cada um com sua thread de controle.
 - Cada um deles opera em espaço de endereçamento diferentes.
- Figura (b) único processo com 3 threads.
 - Todos os threads compartilham o mesmo espaço de endereçamento.



IMPORTANTE SOBRE OS DOIS TIPOS DE THREADS.

- Todos os threads tendo o mesmo espaço de endereçamento significa que compartilham variáveis globais.
- Cada thread pode ter acesso a qualquer endereço de memória dentro do espaço de endereçamento do processo.
- Não há proteção entre threads.
- **Não tem necessidade pois são processos gerenciados por processo pai.**
- Já em processos independentes, podem ser usuários diferentes.

ESTADOS DE UMA THREAD.

- **Mesmo conceito de processos.**

- Estados:

- **EM EXECUÇÃO:** Utilizando a CPU naquele momento.
- **PRONTO:** Parado para dar lugar a outro processo. Aguardando o escalonamento.
- **BLOQUEADO:** Não pode executar pois depende de um evento externo



CRIANDO UM PROCESSO FILHO – FORK.

- **FORK** – Função de chamada de sistema padrão POSIX.
- Cria um processo filho. O processo pai é aquele que usou o fork.
- Por exemplo, suponha que você programou um software em C, e nele usou a chamada **fork()**. Esse programa em C, executando, é o processo pai. Fork cria o processo filho idêntico ao pai, inclusive tendo as mesmas variáveis e registros.
- É uma copia exatamente igual ao pai.
- Apesar de processo filho, vai ser executado independente. O que acontece em um processo não ocorre nos outros, São processos distintos.

FUNÇÃO WAIT.

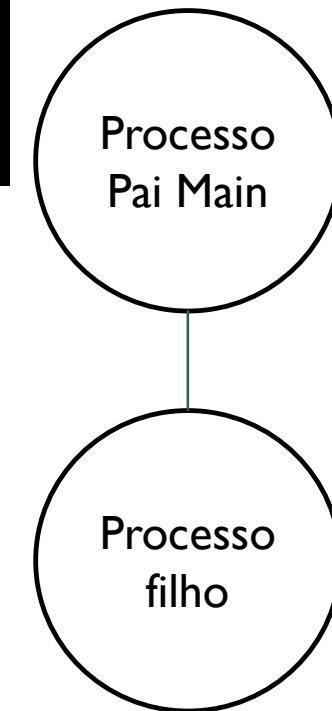
- Permite que o processo pai espere o término de um processo filho.
- A função devolve o status de retorno de qualquer processo filho que termine.

FORK I

```
GNU nano 3.2

#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <wait.h>
int main()
{
    int id;
    id = fork();
    if (id != 0)
    {
        printf ("eu sou o pai e espero pelo filho %d\n",id);
        wait(0);
        printf ("meu filho acabou de terminar vou terminar tambem\n");
    }
    else{
        printf ("sou filho %d e espero 10 segundo\n",getpid());
        sleep(10);
        printf ("ja esperou e vou embora\n");
    }
    exit(0);
}
```

```
root@usuario-VirtualBox:/home# gcc fork.c -o fork
root@usuario-VirtualBox:/home# ./fork
eu sou o pai e espero pelo filho 1453
sou filho 1453 e espero 10 segundo
ja esperou e vou embora
meu filho acabou de terminar vou terminar tambem
root@usuario-VirtualBox:/home#
```



A partir do fork o bloco de código é compartilhado para os 2 processos, desta forma o if é utilizado para que cada processo execute somente sua parte. O Wait aguarda o processo dependente terminar.

tty4

OBSERVAÇÕES:

- **Processos antes da execução do programa fork**
- **Processos durante a execução dos 10 segundos do processo filho.**
- **Processos após termino do fork.**

tty3

```
root@usuario-VirtualBox:/home# ./fork
eu sou o pai e espero pelo filho 1668
sou filho 1668 e espero 10 segundo
ja esperou e vou embora
meu filho acabou de terminar vou terminar tambem
root@usuario-VirtualBox:/home# ps -a
  PID TTY      TIME CMD
 1629 tty3    00:00:00 bash
 1669 tty3    00:00:00 ps
root@usuario-VirtualBox:/home# ./fork
eu sou o pai e espero pelo filho 1759
sou filho 1759 e espero 10 segundo
ja esperou e vou embora
meu filho acabou de terminar vou terminar tambem
root@usuario-VirtualBox:/home# _
```

```
root@usuario-VirtualBox:~# ps -a
  PID TTY      TIME CMD
 1629 tty3    00:00:00 bash
 1735 tty4    00:00:00 bash
 1745 tty1    00:00:00 ps
root@usuario-VirtualBox:~# ps -a
  PID TTY      TIME CMD
 1629 tty3    00:00:00 bash
 1735 tty4    00:00:00 bash
 1758 tty3    00:00:00 fork
 1759 tty3    00:00:00 fork
 1760 tty4    00:00:00 ps
root@usuario-VirtualBox:~# ps -a
  PID TTY      TIME CMD
 1629 tty3    00:00:00 bash
 1735 tty4    00:00:00 bash
 1774 tty4    00:00:00 ps
root@usuario-VirtualBox:~# _
```

- **Observando os processos antes de executar.**
- **Durante os 10 segundos que o processo filho aguarda abrir outra interface (tty4) e aplicar o comando ps.**

FORK 2

- Processo filho conta ate 5
- E depois volta o controle para o pai.
- Após fork, os dois processos estão rodando independentemente.
- Função wait (processo esperado, status, parametro zero para aguardar)

```
GNU nano 2.2.6                               Arquivo: fork2.c

#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int main()
{
    pid_t filho;
    int i, status;
    filho = fork();
    if (filho == 0) //codigo do filho
    {
        printf("Sou processo filho.\n");
        for (i=0; i<5; i++)
        {
            printf("%d\n", i);
            sleep(2);
        }
        exit(0);
    }else{ //codigo do pai
        wait(filho,&status,0);
        printf("Sou pai. Agora posso executar o meu codigo");
    }
    return(0);
}
```

REFERÊNCIA BIBLIOGRÁFICA.

- TANEMBAUM, Andrew S. Sistemas Operacionais Modernos. 2º Ed. Pearson, 2005.
- MACHADO, Francis Berenger; MAIA, Luiz Paulo. Arquitetura de Sistemas Operacionais. 4º Edição, LTC, 1996.