Monitoramento em Ambientes Linux

Avaliação de Desempenho de Sistemas - ADS

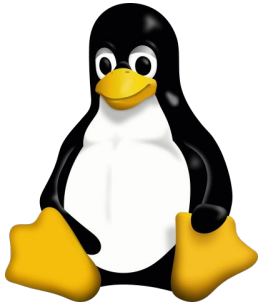
Orientador: Prof. Paulo Maciel

Instrutores: Erick Barros, Luan Lins
{ebn, lcsl2}@cin.ufpe.br

MoDCS Research Group

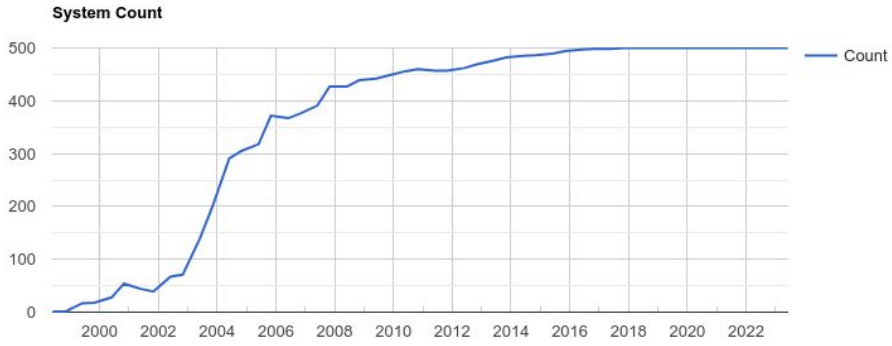
Centro de Informática UFPE

Avaliação de Desempenho de Sistemas



- Por que monitorar o ambiente?
- O que Monitorar?
- Como Monitorar?
- Onde começar?

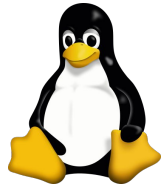
Operating system Family / Linux



List	Count	System Share (%)	Rmax (GFlops)	Rpeak (GFlops)	Cores
Jun 2023	500	100	5,239,024,666	7,825,920,476	95,461,168
Nov 2022	500	100	4,864,384,416	7,540,692,518	94,792,868
Jun 2022	500	100	4,403,083,214	6,848,441,668	91,432,184
Nov 2021	500	100	3,036,861,784	4,788,894,945	81,260,216
Jun 2021	500	100	2,786,058,800	4,404,026,213	76,925,752
Nov 2020	500	100	2,428,761,852	3,846,035,876	72,466,126
Jun 2020	500	100	2,206,134,394	3,481,213,144	71,159,810
Nov 2019	500	100	1,646,887,143	2,746,327,220	63,153,986
Jun 2019	500	100	1,559,575,380	2,463,872,555	59,106,366
Nov 2018	500	100	1,414,955,582	2,213,309,357	59,086,318
Jun 2018	500	100	1,210,914,864	1,921,664,802	58,055,634

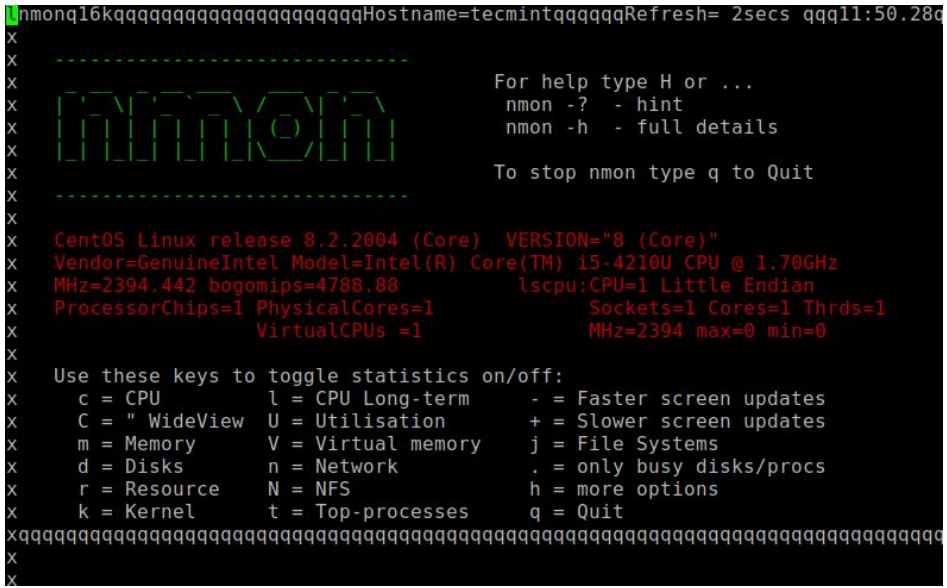
https://www.top500.org/statistics/details/osfam/1/

Monitoramento Linux



Ferramentas essenciais

- Shell (sh, csh, tcsh, zsh, bash)
- /proc
- ps
- top
- uptime
- vmstat
- free
- stress
- sysstat (iostat, mpstat, pidstat)
- tcpdump (redes)



Interpretador de comandos - SHELL

É o programa responsável em interpretar as instruções enviadas pelo usuário e seus programas ao sistema operacional (o kernel). Ele executa comandos lidos do dispositivo de entrada padrão (teclado) ou de um arquivo executável. É a principal ligação entre o usuário, os programas e o kernel.

Os comandos podem ser enviados de duas maneiras para o interpretador:

- Interativa: comando digitados e passados ao shell;
- Não-interativa: uso de arquivos com a sequência dos comandos (scripts).

Interpretador de comandos - SHELL

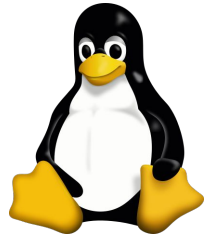
Linux

- *bash* (o mais conhecido)

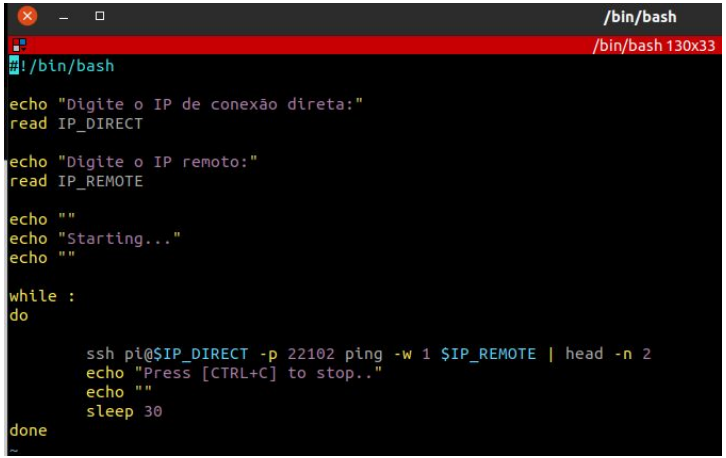
Windows

- *bash* (WSL) ou PowerShell
- Comandos executados em interfaces gráficas (GUI - graphical user interface), necessitam de outro programa para interagir com o shell, conhecidos como **Emuladores de Terminal**:
  - konsole e gnome-terminal são exemplos.

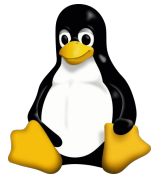
Shell script



- É um arquivo de texto (.sh, ps1, ….);
- Automatiza tarefas específicas (tediosas e repetitivas) que permitem a ausência do operador do sistema;
- Executam um conjunto de comandos;
- Podem se valer de lógicas simples (*for, while, if e else*) ou sistemas complexos.



echo “Hello world”

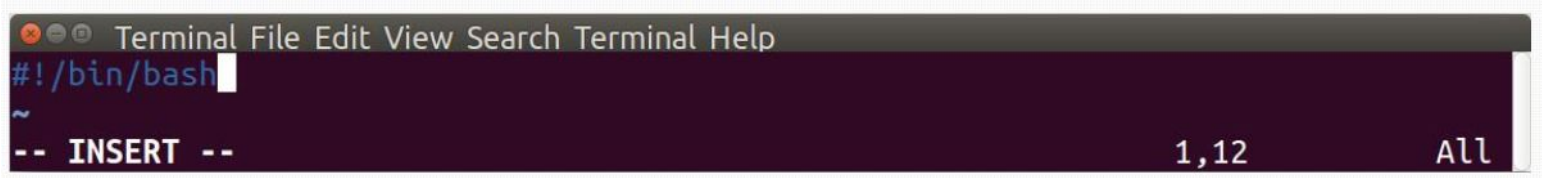


- Variáveis;
- Primeiro Script;
- Permissão de execução (chmod +x, ou chmod 755);
- sh [nome do script] ou ./script
- #!/bin/bash
- Por que shell e não Java, Python, Ruby, etc…;
- É uma linguagem interpretada.

r -> leitura (4)
w -> escrita (2)
x -> execução(1)

```
echo "$(exiftool *.doc | grep Pages | cut -d ":" -f2 | tr "\n" "+")" | bc
```

Sintaxe Scripts Shell



A primeira linha de um `shell script` define qual o interpretador de comandos será utilizado.

Os interpretadores de comandos **bash**, **sh**, **ksh** e **csh** são diferentes implementações de shells, cada uma com suas próprias características e sintaxe específicas.

- **Bash (Bourne Again SHell)**: O Bash é uma das implementações mais populares e amplamente usadas de um interpretador de comando Unix. O Bash se tornou o shell padrão em muitas distribuições de Linux.
- **sh (Bourne Shell)**: O sh, também conhecido como Bourne Shell, é o shell original do Unix. Ele foi desenvolvido por Stephen Bourne e é o precursor de muitos outros shells.
- **ksh (Korn Shell)**: O ksh, ou Korn Shell, foi desenvolvido por David Korn como uma extensão do sh original. Ele preserva a compatibilidade com o sh, mas adiciona recursos mais avançados e uma sintaxe mais rica.
- **csh (C Shell)**: O csh, ou C Shell, foi desenvolvido por Bill Joy como uma alternativa ao sh original. Ele possui uma sintaxe semelhante ao estilo da linguagem de programação C e foi projetado para ser mais interativo e fácil de usar, especialmente para usuários acostumados com a linguagem C.

Sintaxe Scripts Shell

A linguagem do `shell` não é tipada, ou seja, pode-se armazenar qualquer tipo de valor em uma variável, desde strings a números.

Variáveis

Para declará-las basta seguir a sintaxe `nome_da_variavel=valor`:

- `nome_da_variavel`: sequência de caracteres que deve começar por qualquer letra maiúscula ou minúscula, ou `underscore (_)`;
- `valor`: qualquer dado que deva ser valorado à variável

Observe que não deve haver espaços entre o sinal de igual e o nome e o valor da variável.

```
1 #!/bin/bash
2 site=www.devmedia.com.br
3 meu_numero_favorito=13
4 _cidade="Porto Alegre"
5 echo "Um ótimo site para você aprender a programar e se manter atualizado é: $site"
6 echo "Meu número favorito é: $meu_numero_favorito"
7 echo "Minha cidade natal é: $_cidade"
```

Para utilizarmos o valor da variável coloca-se o `$` (cifrão) na frente de seu nome

Sintaxe Scripts Shell

É possível armazenar o resultado de um comando em uma variável. Isso é muito útil em situações em que se usará este resultado em mais de um lugar ao longo do script.

Há duas sintaxes para isso:

- `nome_da_variavel=$(comando)`
- `nome_da_variavel=`comando``

Você pode escolher a que melhor lhe agrada ou empregar as duas nos seus scripts.

Informações relativas a todos os discos e partições do sistema:

```
1 #!/bin/bash
2 system_info=`df -h` # Também poderia ser system_info=$(df -h)
3 echo “$system_info”
```

Sintaxe Scripts Shell

Comandos de seleção ou de tomada de decisão

```
1 if [ CONDICAO ];
2 then
3   AÇÕES
4 Fi
```

Lembre-se que, se utilizar o `[`, você deve fechá-lo com o `]`, e deixando sempre espaços ao redor. Isso é muito importante, pois são um "atalho" para o comando '**test**'. Isso significa que, alternativamente, você poderia querer não utilizar os colchetes:

Onde:

- `CONDICAO`: teste que, se verdadeiro, passará o controle para o bloco dentro do `then`;
- `AÇÕES`: comandos a serem executados se o resultado de `CONDICAO` for verdadeiro.

É muito comum o desenvolvedor esquecer de fechar o `if`. Lembre-se sempre que, para cada `if` aberto, você precisa fechá-lo com o `fi`.

Sintaxe Scripts Shell

Vamos a um exemplo em que o usuário deverá digitar um número e verificaremos se ele está em um determinado intervalo

```
#!/bin/bash
echo “Digite um número qualquer:”
read numero;
if [ “$numero” -gt 20 ];
then
  echo “Este número é maior que 20!”
fi
```

Parâmetros mais comuns utilizados com o comando if:

- **n** string1: o comprimento de string1 é diferente de 0;
- **z** string1: o comprimento de string1 é zero;
  string1 = string2: string1 e string2 são idênticas;
  string1 **!=** string2: string1 e string2 são diferentes;
  inteiro1 **-eq** inteiro2: inteiro1 possui o mesmo valor que inteiro2;
  inteiro1 **-ne** inteiro2: inteiro1 não possui o mesmo valor que inteiro2;
  inteiro1 **-gt** inteiro2: inteiro1 é maior que inteiro2;
  inteiro1 **-ge** inteiro2: inteiro1 é maior ou igual a inteiro2;
  inteiro1 **-lt** inteiro2: inteiro1 é menor que inteiro2;
  inteiro1 **-le** inteiro2: inteiro1 é menor ou igual a inteiro2;
- **e** nome_do_arquivo: verifica se nome_do_arquivo existe;
- **d** nome_do_arquivo: verifica se nome_do_arquivo é um diretório;
- **f** nome_do_arquivo: verifica se nome_do_arquivo é um arquivo regular (texto, imagem, programa, docs, planilhas).

Sintaxe Scripts Shell

O comando Else

Existe a possibilidade de também tratar o caso em que o nosso teste falha. Para isso temos o comando `else`.

```
if [ CONDICAO ];
   then
       AÇÕES_1
  else
       AÇÕES_2
fi
```

```
#!/bin/bash
  echo “Digite um número qualquer:”
  read numero;
  if [ “$numero” -ge 0 ];
   then
    echo “O número $numero é positivo!”
  else
   echo “O número $numero é negativo!”
fi
```

Onde:

- `CONDICAO`: teste que, se verdadeiro, passará o controle para o bloco dentro do `then`;
- `AÇÕES_1`: comandos a serem executados se o resultado de `CONDICAO` for verdadeiro;
- `AÇÕES_2`: comandos a serem executados se o resultado de `CONDICAO` for falso.

Sintaxe Scripts Shell

O comando Elif

Há casos em que temos mais de uma condição a ser testada, todas correlacionadas. Para isso temos o comando `elif`, cuja sintaxe é:

```
if [ CONDICAO_1 ];
 then
  AÇÕES_1
 elif [ CONDICAO_2 ];
 then
   AÇÕES_2
 elif [ CONDICAO_3 ];
 then
   AÇÕES_3
                .
                .
                .
                .
 elif [ CONDICAO_N ];
 then
  AÇÕES_N
Fi
```

```
#!/bin/bash
echo "Selecione uma opção:"
echo "1 - Exibir data e hora do sistema"
echo "2 - Exibir o resultado da divisão 10/2"
echo "3 - Exibir uma mensagem”
read opcao;
if [ $opcao == "1" ];
then
  data=$(date +"%T, %d/%m/%y, %A")
  echo "$data"
elif [ $opcao == "2" ];
then
   result=$((10/2))
   echo "divisao de 10/2 = $result"
elif [ $opcao == "3" ];
 then
 echo "Informe o seu nome:"
 read nome;
 echo "Bem-vindo ao mundo do shell script, $nome!"
fi
```

O `bash` não tem suporte nativo a divisões em ponto flutuante, apenas divisões inteiras. Caso queira efetuar este tipo de operação, precisará de um comando externo, como `dc` ou `bc`

Observe a linha: `result=$((10/2))`

Veja que utilizamos dois conjuntos de **parênteses** para encapsular a operação de divisão. Em shell script precisamos realizar operações matemáticas entre parênteses.

Sintaxe Scripts Shell

LOOPS Condicionais

Loops são muito úteis para ficar iterando sobre determinadas ações até que uma condição seja satisfeita e interrompa o laço.

O primeiro deles é o `for`

```
for VARIAVEL in VALOR_1, VALOR_2 … VALOR_N;
  do
      AÇÕES
  done
```

A sequência `VALOR_1, VALOR_2 … VALOR_N;` na sintaxe pode ser substituída por: `{VALOR_1..VALOR_N};`

Observe que são apenas duas reticências.

Onde:

- `VARIAVEL`: variável cujo valor será inicializado e incrementado, respeitando os limites dos valores do conjunto fornecido;
- `VALOR_1, VALOR_2 … VALOR_N`: valores que `VARIAVEL` poderá assumir durante o `loop`;
- `AÇÕES`: ações a serem tomadas repetidamente até que o valor de `VARIAVEL` ultrapasse o último valor informado no conjunto de valores fornecido.

Sintaxe Scripts Shell

LOOPS Condicionais

Quando o loop for começa, a variável é inicializada com o primeiro valor do conjunto, e ocorre a primeira iteração (entrada no laço e execução dos comandos).

Para as iterações seguintes, os valores do conjunto serão atribuídos à variável, sucessivamente, até que se alcance o último e o loop termine a execução.

```
#!/bin/bash
 echo “Testando o loop for”
 for i in {10..0};
 do
   echo “$i”
 done
```

conta decrescendo de 10 a 0.

```
#!/bin/bash
 echo “Testando o comando seq”
 for i in $(seq 1 5 100);
 do
  echo “$i”
done
```

Outra forma de criarmos sequências de valores é com o comando `seq`

Observe que foi criada uma sequência de 1 até 100, com intervalo de 5.

Sintaxe Scripts Shell

Loop while

Enquanto o loop for é mais ideal para quando sabemos até quanto contar, o loop while é bom para quando não temos essa noção, mas sabemos de uma condição que deverá ser atendida para o laço terminar.

```
while [ CONDICAO ];
      do
          AÇÕES
   done
```

Onde:

- `CONDICAO`: condição cuja veracidade determina a permanência no laço;
- `AÇÕES`: ações a serem tomadas enquanto `CONDICAO` for verdadeira.

```
#!/bin/bash
  echo “Informe o que você quiser, -1 para sair”
  read dado;
  while [ $dado != “-1” ];
  do
    echo “Você digitou $dado”
  read dado;
done
```

Um exemplo que exibe ao usuário o que ele digitou, enquanto ele não informar `-1`.

Sintaxe Scripts Shell

Loop while

```
#!/bin/bash
echo “Informe até que valor positivo e maior que zero contar:”
read valor;
i=1
while [ $i -le $valor ];
 do
  echo “$i”
  ((i=$i+1))
done
```

Exemplo com contador

AWK command

O **AWK** é uma ferramenta para processar linhas e colunas mais fácil de usar do que muitas linguagens de programação convencionais. Ele pode ser considerado um pseudo-C interpretador, por entender os mesmos operadores aritméticos de C. O **AWK** também possui funções de manipulação de strings, portanto, pode pesquisar strings específicas e modificar a saída.

Existindo em três variações:

- AWK - the (very old) original from AT&T
- NAWK - A newer, improved version from AT&T
- GAWK - The Free Software foundation's version

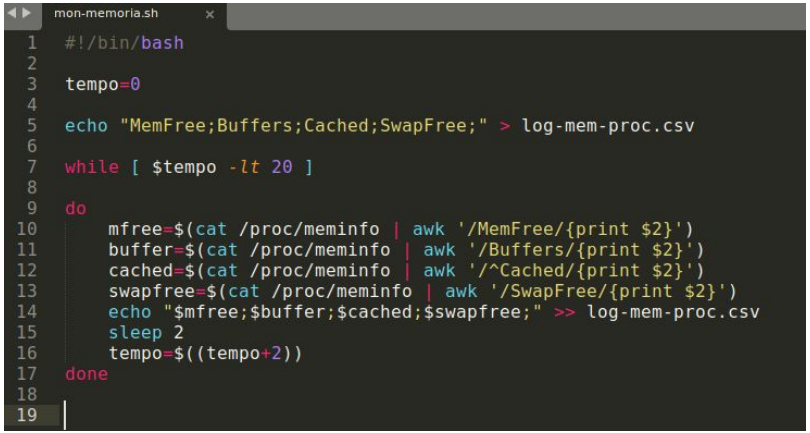
seu nome significa a abreviatura de seus criadores- Aho, Kernighan e, Weinberger

sintaxe: $ awk ‘/manager/{print $1}’ example.txt

AWK command

$ awk -F “,” ‘/dk9102/ {print $4,$2,$6;}’ compras.csv

dk9102 marcio 2013-10-01
dk9102 maria 2013-10-02
dk9102 alex 2013-10-01
dk9102 jason 2013-10-03



$ awk -F “,” ‘/dk9102/ {print $4,$2,$6;}’ compras.csv

-F “,” –> diz ao awk para considerar a vírgula como separador de campos

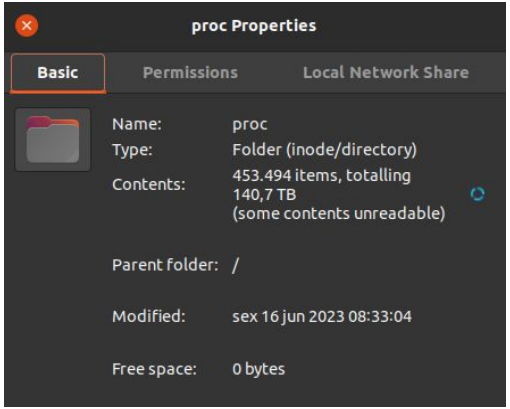
/dk9102/ –> padrão a ser procurado

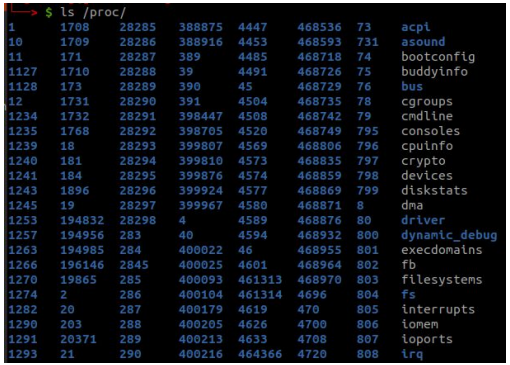
{print $4,$2$,6} –> imprimir o quarto, segundo e sexto campo (produto,nome,data)

sintaxe: $ awk ‘/manager/{print $1}’ example.txt

Utilizando o /proc

- /proc
  - É o sistema de arquivos do Kernel do GNU/Linux. Ele oferece um método de ler, gravar e modificar dinamicamente os parâmetros do kernel;
  - A modificação dos arquivos do diretório **/proc** é o método mais usado para modificar a configuração do sistema e muitos programas também dependem deste diretório para funcionar.
  - É estruturado como uma **hierarquia de diretórios e arquivos**.





Utilizando o /proc

- /proc
  - Vários contadores de desempenho disponíveis:
    - /proc/stat
    - /proc/meminfo
    - /proc/vmstat
    - /proc/diskstats
    - /proc/net
    - /proc/<PID process>...

```
$ cat /proc/cpuinfo
processor       : 0
vendor_id       : AuthenticAMD
cpu family      : 23
model           : 8
model name      : AMD Ryzen 5 1600 Six-Core Processor
stepping        : 2
microcode       : 0x800820d
cpu MHz         : 1550.000
cache size      : 512 KB
physical id     : 0
siblings        : 12
core id         : 0
cpu cores       : 6
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
```

cat - concatenar e exibir o conteúdo de arquivos.

cat [file name]

/proc/stat

Quando você lê o arquivo **/proc/stat**, geralmente encontrará uma única linha que começa com a palavra-chave "**cpu**", seguida de vários números.

```
cpu  123456 789  2345678 987654321 54321 0  0 0 0
```

Cada número representa uma estatística diferente relacionada à CPU

- `123456`: Jiffies em modo de usuário - Tempo gasto executando processos de usuário.
- `789`: Jiffies em modo "nice" - Tempo gasto executando processos de usuário com prioridade reduzida ("nice").
- `2345678`: Jiffies em modo de sistema - Tempo gasto executando processos do sistema/kernel.
- `987654321`: Jiffies de ociosidade - Tempo ocioso, sem processos para executar.
- `54321`: Jiffies de espera de E/S - Tempo gasto aguardando a conclusão de operações de entrada/saída (E/S).
- `0`: Jiffies de interrupção (IRQ) - Tempo gasto tratando interrupções de hardware.
- `0`: Jiffies de interrupção de software (SoftIRQ) - Tempo gasto tratando interrupções de software.
- `0`: Jiffies de "steal" - Tempo gasto em outros sistemas operacionais em execução em um ambiente virtualizado.
- `0`: Jiffies de convidado (guest) - Tempo gasto executando uma CPU virtual para sistemas operacionais convidados sob o controle do kernel Linux.
- `0`: Jiffies de convidado "nice" - Tempo gasto executando uma CPU convidada com prioridade reduzida ("nice").

/proc/stat

`intr`: Esta linha fornece estatísticas sobre as interrupções do sistema. Cada número representa o número de interrupções de um determinado tipo que ocorreram desde o início do sistema.

`ctxt`: Essa linha mostra o número de mudanças de contexto, ou seja, o número de vezes que o sistema trocou de execução entre diferentes tarefas. Geralmente, inclui mudanças de contexto voluntárias e mudanças de contexto causadas por interrupções.

`btime`: Essa linha indica o tempo de inicialização do sistema, representado como o número de segundos desde a data de referência do Unix (1º de janeiro de 1970).

`processes`: Essa linha mostra o número total de processos criados desde o início do sistema.

`procs_running`: Essa linha indica o número atual de processos em execução.

`procs_blocked`: Essa linha mostra o número de processos atualmente bloqueados, aguardando algum evento (como E/S) para continuar sua execução.

/proc/meminfo

O arquivo `/proc/meminfo` é um arquivo virtual no sistema operacional Linux que fornece informações sobre o uso e estatísticas de memória do sistema

```
MemTotal:       16339048 kB
MemFree:         4599852 kB
MemAvailable:    9478244 kB
Buffers:           74548 kB
Cached:          7013152 kB
SwapCached:            0 kB
Active:          8654188 kB
Inactive:        5623832 kB
...
```

Campos comuns

- `MemTotal`: Total de memória física disponível em kilobytes (kB).
- `MemFree`: A quantidade de memória física que não está em uso por nenhum processo.
- `MemAvailable`: Uma estimativa da quantidade de memória disponível para iniciar novos aplicativos sem fazer swap.
- `Buffers` e `Cached`: Memória usada para buffers do sistema de arquivos e cache de disco.
- `SwapCached`: Memória trocada para o arquivo de swap, mas continua presente nele.
- `Active` e `Inactive`: Memória usada por processos ativos e inativos, respectivamente.

/proc/vmstat

O arquivo `/proc/vmstat` é um arquivo virtual localizado no diretório `/proc` em sistemas Linux. Ele fornece informações sobre vários aspectos do uso da memória virtual do sistema.

```
cat /proc/vmstat
```

Campos comuns

- `nr_free_pages`: Número de páginas livres no sistema.
- `nr_alloc_batch`: Número de páginas por alocação em lote.
- `pgpgin`: Número total de páginas lidas do disco.
- `pgpgout`: Número total de páginas gravadas no disco.
- `pswpin`: Número total de páginas de swap trazidas para dentro.
- `pswpout`: Número total de páginas de swap trazidas para fora.

/proc/diskstats

O arquivo `/proc/diskstats` é um arquivo virtual no sistema operacional Linux que fornece informações sobre as estatísticas de E/S de disco para todos os dispositivos de bloco no sistema.

```
cat /proc/diskstats
```

Campos comuns

- **Major**: O número major do dispositivo.
- **Minor**: O número minor do dispositivo.
- **Device**: O nome do dispositivo.
- **Reads Completed**: O número total de leituras concluídas com sucesso.
- **Reads Merged**: O número de leituras mescladas com leituras adjacentes.
- **Sectors Read**: O número total de setores lidos com sucesso.
- **Time Spent Reading (ms)**: O tempo total em milissegundos gasto na leitura.
- **Writes Completed**: O número total de gravações concluídas com sucesso.
- **Writes Merged**: O número de gravações mescladas com gravações adjacentes.
- **Sectors Written**: O número total de setores gravados com sucesso.
- **Time Spent Writing (ms)**: O tempo total em milissegundos gasto na gravação.
- **I/Os Currently in Progress**: O número de operações de E/S em andamento no momento.
- **Time Spent Doing I/Os (ms)**: O tempo total em milissegundos gasto em operações de E/S.
- **Weighted Time Spent Doing I/Os (ms)**: O tempo total ponderado em milissegundos gasto em operações de E/S (inclui o tempo de E/S de outros dispositivos de bloco compartilhando o mesmo hardware).

/proc/net

O diretório `/proc/net` em sistemas Linux fornece informações relacionadas a vários componentes e estatísticas de rede. Ele contém vários arquivos que oferecem detalhes sobre interfaces de rede, protocolos, conexões e outras informações relacionadas à rede.

Arquivos comuns

- `/proc/net/dev`: Este arquivo fornece estatísticas para interfaces de rede, incluindo pacotes recebidos e transmitidos, bytes, erros e outras informações relevantes.
- `/proc/net/arp`: Ele exibe o cache do Protocolo de Resolução de Endereço (ARP), que contém mapeamentos de endereços IP para endereços MAC na rede local.
- `/proc/net/route`: Este arquivo apresenta a tabela de roteamento IP do kernel, que mostra as informações de roteamento para diferentes redes e interfaces.
- `/proc/net/tcp`: Ele fornece informações sobre conexões TCP ativas, incluindo endereços IP e números de porta local e remoto, estado e várias métricas relacionadas ao TCP.
- `/proc/net/udp`: Semelhante a `/proc/net/tcp`, este arquivo exibe informações sobre conexões UDP (User Datagram Protocol) ativas, incluindo endereços IP e números de porta local e remoto.
- `/proc/net/raw`: Ele contém informações sobre soquetes brutos, incluindo o protocolo associado a cada soquete.
- `/proc/net/snmp`: Este arquivo apresenta dados estatísticos relacionados ao Protocolo Simples de Gerenciamento de Rede (SNMP) para monitorar o desempenho da rede.

/proc/<PID process>

No sistema operacional baseado em Linux, o diretório `/proc` é um sistema de arquivos virtual que fornece uma interface para as estruturas de dados do kernel e informações sobre os processos em execução.

Arquivos comuns

- `/proc/<PID>/cmdline`: Este arquivo contém os argumentos da linha de comando passados para o processo quando ele foi iniciado. Os argumentos são geralmente separados por bytes nulos.
- `/proc/<PID>/cwd`: Este é um link simbólico que aponta para o diretório de trabalho atual do processo.
- `/proc/<PID>/environ`: Este arquivo contém as variáveis de ambiente associadas ao processo.
- `/proc/<PID>/exe`: Este é um link simbólico que aponta para o arquivo executável do processo.
- `/proc/<PID>/fd`: Este diretório contém links simbólicos para os descritores de arquivo abertos pelo processo.
- `/proc/<PID>/status`: Este arquivo fornece várias informações de status sobre o processo, como seu estado, uso de memória, uso de CPU e muito mais.
- `/proc/<PID>/maps`: Este arquivo lista os mapeamentos de memória do processo, incluindo informações sobre as bibliotecas mapeadas e seus endereços.

/proc/<PID>/status

O arquivo `/proc/<PID>/status` é um arquivo virtual no sistema operacional Linux que fornece várias informações de status sobre um processo específico identificado pelo seu ID de processo (PID). O espaço `<PID>` no caminho do arquivo deve ser substituído pelo ID de processo real do processo que você está interessado.

Campos comuns

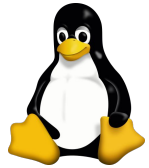
- `Name`: O nome do processo.
- `State`: O estado atual do processo (por exemplo, em execução, dormindo, parado).
- `Pid`: O ID do processo.
- `PPid`: O ID do processo pai.
- `Uid`: Os IDs de usuário associados ao processo.
- `Gid`: Os IDs de grupo associados ao processo.
- `VmPeak`: O tamanho máximo de memória virtual usado pelo processo.
- `VmSize`: O tamanho atual da memória virtual do processo.

/proc/<PID>/status

- Em muitas situações, é essencial medir o uso de recursos para um processo específico.
  - **VmSize:** memória alocada para o processo.
  - **VmHWM**: valor máximo atingido pelo RSS.
    - **Residente set size**: memória física usada pelo processo.

```
──> $ cat /proc/385946/status | grep Vm
VmPeak: 19897740 kB
VmSize:  6071448 kB
VmLck:         0 kB
VmPin:         0 kB
VmHWM:   1531248 kB
VmRSS:    977996 kB
VmData:  2186784 kB
VmStk:       144 kB
VmExe:       632 kB
VmLib:    230656 kB
VmPTE:      5100 kB
VmSwap:        0 kB
```

Exemplo: /proc/meminfo



- Possibilita o monitoramento em tempo real ou,
- Geração de arquivo de *log* para coleta e posterior visualização.

```
└─> $ watch -n 5 cat /proc/meminfo
```

mon-memoria.sh

```
#!/bin/bash

tempo=0

echo "MemFree;Buffers;Cached;SwapFree;" > log-mem-proc.csv

while [ $tempo -lt 20 ]

do
    mfree=$(cat /proc/meminfo | awk '/MemFree/{print $2}')
    buffer=$(cat /proc/meminfo | awk '/Buffers/{print $2}')
    cached=$(cat /proc/meminfo | awk '/^Cached/{print $2}')
    swapfree=$(cat /proc/meminfo | awk '/SwapFree/{print $2}')
    echo "$mfree;$buffer;$cached;$swapfree;" >> log-mem-proc.csv
    sleep 2
    tempo=$((tempo+2))
done

```

watch - execute a program periodically, showing output fullscreen

Let’s start coding!

ps command process (Process Status)

- O comando *ps* (status do processo), mostra informações sobre uma selação de processos ativos.
  - obtém informações sobre processos específicos;
  - em execução ou não;
  - qual usuário do sistema está executando o processo;
  - consumo de memória e cpu pelo processo;
  - tempo do processo em execução;
- Normalmente utilizado com os argumentos:
  - a = show processes for all users
  - u = display the process's user/owner
  - x = also show processes not attached to a terminal

ps command process (Process Status)

- Se executado sem nenhum argumento, ***ps*** retorna apenas o processo em execução na conta do usuário conectado ao terminal.
- Exemplo:

Processo do próprio terminal bash

Processo do comando ***ps*** executado

```
$ ps
    PID TTY          TIME CMD
 475636 pts/1    00:00:00 bash
 475639 pts/1    00:00:00 ps
```

ps command process (Process Status)

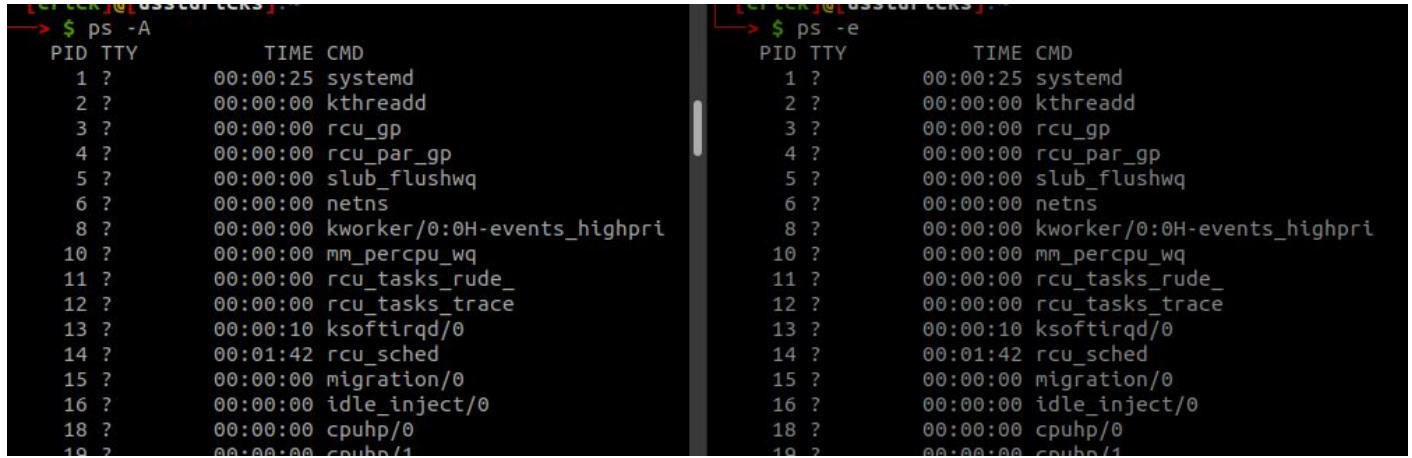
O comando **ps** aceita execuções em três estilos:

- Estilo BSD Unix: instrução definida sem hifenização (“$ ps aux”);
- Estilo AT&T Unix: instrução definida com um hífen hifenização (“$ ps -aux”);
- Estilo GNU/Linux: instrução definida com 2 hífens (“$ ps --aux”).

Em distros atuais, os 3 estilos são aceitos.

ps command process (Process Status)

- Para imprimir todos os processos em execução no sistema, use qualquer um dos seguintes comandos:
  - $ ps -A
  - $ ps -e



Comandos equivalentes

ps -aux

Ao usar o comando "**ps**" com as opções "**aux**", ele exibe diversos campos de informações sobre os processos.

```
$ ps aux | grep chrome
```

```
ps aux | grep chrome | awk '{print $2}' | head -n 1
```

pega a primeira linha e coluna 2 do processo do chrome

Campos comuns

- USER: O nome de usuário do proprietário do processo.
- PID: O ID do processo, um identificador único para cada processo em execução.
- %CPU: A porcentagem de uso de CPU pelo processo.
- %MEM: A porcentagem de uso de memória pelo processo.
- VSZ: O tamanho da memória virtual do processo em kilobytes.
- RSS: O tamanho do conjunto de residentes, que é a memória física não trocada usada pelo processo.
- TTY: O terminal associado ao processo.
- STAT: O estado do processo, como "R" para executando, "S" para suspenso, "Z" para zumbi, etc.
- START: O horário em que o processo foi iniciado.
- TIME: O tempo total de CPU usado pelo processo.
- COMMAND: O comando ou nome do programa que iniciou o processo.

ps command process (Process Status)

- O output do comando tem os seguintes significados:

Column	Description
USER	The user account under which this process is running
PID	Process ID of this process
%CPU	CPU time used by this process (in percentage).
%MEM	Physical memory used by this process (in percentage).
VSZ	Virtual memory used by this process (in bytes).
RSS	Resident Set Size, the non-swappable physical memory used by this process (in KiB)
TTY	Terminal from which this process is started. Question mark (?) sign represents that this process is not started from a terminal.
STAT	Process state. *Explained in next table.*
START	Starting time and date of this process
TIME	Total CPU time used by this process
COMMAND	The command with all its arguments which started this process

ps command process (Process Status)

- O output do comando tem os seguintes significados:

D	uninterruptible sleep (usually IO)
R	running or runnable (on run queue)
S	interruptible sleep (waiting for an event to complete)
T	stopped by job control signal
t	stopped by debugger during the tracing
w	paging (not valid since the 2.6.xx kernel)
x	dead (should never be seen)
Z	defunct ("zombie") process, terminated but not reaped by its parent
<	high-priority (not nice to other users)
N	low-priority (nice to other users)
L	has pages locked into memory (for real-time and custom IO)
s	is a session leader
l	is multi-threaded (using CLONE_THREAD, like NPTL pthreads do)
+	is in the foreground process group

ps command process (Process Status)

- CPU é exibido em percentual de tempo gasto em execução durante todo o ciclo de vida do processo.
- O RSS não conta algumas partes de um processo, incluindo as tabelas de páginas, pilha do kernel, structure thread_info e struct task_struct.
- VSZ é o tamanho virtual do processo (código + dados + pilha).
- Processos marcados como <defunct> são processos “zumbis”.

Se o comprimento do nome de usuário for maior que o comprimento da coluna de exibição, o nome de usuário será truncado.

Let’s coding!

```
#!/bin/bash

tempo=0

echo "State;VmSize;VmHwM;VmRSS;" > log-ps-processos.csv
pid=$(ps -aux | grep firefox | awk '{print $2}' | head -n 1)

while [ $tempo -lt 60 ]

do
        state=$(cat /proc/$pid/status | awk '/State/{print $2}')
        vmsize=$(cat /proc/$pid/status | awk '/VmSize/{print $2}')
        vmhwm=$(cat /proc/$pid/status | awk '/VmHWM/{print $2}')
        vmrss=$(cat /proc/$pid/status | awk '/VmRSS/{print $2}')
        echo "$state;$vmsize;$vmhwm;$vmrss;" >> log-ps-processos.csv
        sleep 2
        tempo=$((tempo+2))
done

```

Monitoramento no Linux

Comando **TOP**:

- Fornece uma visão em tempo real da execução de processo no sistema
- Sintaxe: top [opções]
  - -**d** atraso: Especifica o atraso em segundos entre as atualizações de tela. Default 3s;
  - **-i** ignora processos ociosos;
  - **-n** num: Exibe num interações e depois finaliza.
  - **-b r**oda em modo de batch - útil para mandar a saída do top para outros programas ou arquivos de log.

Monitoramento no Linux

Comando **TOP** - opções interativas:

- **h** gera uma tela de ajuda;
- **k** termina um processo (será pedido o PID);
- **q** sai do programa.

Monitoramento no Linux

Comando **TOP** - **significado do output**:

- **PID:** número identificador do processo;
- **USER**: usuário criador do processo;
- **PR**: prioridade da tarefa;
- **NI**: valor NICE da tarefa;
- **VIRT**: memória virtual usada;
- **RES**: memória física usada;
- **SHR**: memória compartilhada usada;
- **S**: estado do processo (S - sleeping, R=running, T=stopped, Z=zombie)
- **%CPU**: percentual de tempo de CPU;
- **%MEM**: percentual de memória física;
- **TIME+**: tempo total de atividade da tarefa que ela foi iniciada;
- **COMMAND**: nome do processo.

Let's coding!

monitoramento-top.sh

```
#!/bin/bash

tempo=0
echo "State;" > log-mem-processo.csv
pid=$(ps aux | grep chrome | awk '{print $2}' | head -n 1)
while [ $tempo -lt 60 ]
do
    virt=$(top -b -n 1 | awk '/chrome/{print $5}')
    echo "$virt;" >> log-mem-processo.csv
    sleep 2
    tempo=$((tempo+2))
done
```

Monitoramento Linux

Comando **vmstat**

- Este comando reporta informações sobre processos, memória, paginação, blocos de I/O, traps e atividades de CPU.
  - **Vmstat [opções]**
    - -**S** M usa a unidade de MB em vez do padrão KB;
    - -**a** Mostra a memória ativa e inativa;
    - -**d** Mostra estatísticas de discos;
    - -**p** Partição mostra informações de R/W na partição especificada;
    - -**s** Mostra estatísticas em formato de tabela.

Monitoramento Linux

Comando **vmstat** - campos

- Procs
  - **r:** Nº de processos esperando para rodar
  - **b:** Nº de processos em dormência ininterrupta
- Memory
  - **Swpd**: memória virtual usada
  - **Free**: memória livre
  - **Buff**: memória usada como buffer
  - **Cache**: memória usada como cache
- Swap
  - **si:** memória trocada a partir do disco
  - **so**: memória trocada para o disco

Monitoramento Linux

Comando **vmstat** - campos

- io
  - **bi:** blocos recebidos de um dispositivos de bloco (bloco/s)
  - **bo:** blocos enviados a um dispositivo de bloco (bloco/s)
- System
  - **in**: nº de interrupções por segundo, incluindo *clock*
  - **cs**: nº de mudanças de contexto por segundo
- Cpu
  - **us:** Tempo gasto rodando código que não é kernel
  - **sy**: Tempo gasto rodando código do kernel
  - **id**: Tempo gasto em ociosidade
  - **wa**: Tempo gasto esperando por I/O

Monitoramento Linux

Comando **uptime**

- Mostra o tempo atual; há quanto tempo o sistema está rodando, quantos usuários estão logados no sistema e as médias de carga do sistema nos últimos 1,5 e 15 minutos.

```
└──> $ uptime
01:40:03 up 6 days, 17:07,  1 user,  load average: 0,72, 0,90, 0,96
```

Monitoramento Linux

Comando **free**

- Exibe a quantidade de memória livre e usada no sistema
- Sintaxe: free [opções]
  - -b Mostra o uso da memória em bytes
  - -k Mostra o uso da memória em KB
  - -m Mostra o uso da memória em MB
  - -t Exibe em linha as horas totais
  - -s n Operação contínua em intervalos de n segundos

Utilizando o Sysstat

O **sysstat** é um pacote de utilitários para coleta de dados de desempenho, variando em:

- **iostat**: Disco e I/O em geral
- **mpstat**: Processador e memória
- **pidstat**: Monitoramento de processos

Utilizando o Sysstat

- Comando **iostat**
  - Mostra informações sobre o uso de **CPU** e várias estatísticas sobre I/O do sistema.
  - Sintaxe: iostat [opções]
  - **-c** Mostra apenas estatísticas da CPU
  - **-d** Mostra apenas estatísticas de I/O de disco
  - **-p** [unidade de disco nomeada, ex: sda]: Mostra apenas estatísticas para o disco primário, para distros Linux, sda; Se secundário, sdb…

iostat

Campos Comuns

- **%user**: A porcentagem de tempo da CPU gasto em processos de nível de usuário.
- **%nice**: A porcentagem de tempo da CPU gasto em processos de nível de usuário com um valor "nice" positivo (prioridade mais baixa).
- **%system**: A porcentagem de tempo da CPU gasto em processos de nível do sistema.
- **%iowait**: A porcentagem de tempo da CPU gasto aguardando a conclusão de operações de I/O.
- **%steal**: A porcentagem de tempo da CPU gasto em espera involuntária devido à virtualização.
- **%idle**: A porcentagem de tempo da CPU ociosa.

- **Device**: Esta coluna lista o nome dos dispositivos de armazenamento que estão sendo monitorados.
- **tps**: O número de transferências por segundo, indicando a atividade geral de I/O no dispositivo.
- **kB_read/s**: O número médio de kilobytes lidos do dispositivo por segundo.
- **kB_wrtn/s**: O número médio de kilobytes escritos no dispositivo por segundo.
- **kB_read**: O número total de kilobytes lidos do dispositivo desde que o sistema foi iniciado ou desde o último reset.
- **kB_wrtn**: O número total de kilobytes escritos no dispositivo desde que o sistema foi iniciado ou desde o último reset.

Utilizando o Sysstat

- Comando **mpstat**
  - Exibe estatísticas sobre todos os processadores existentes na máquina.

- Sintaxe: mpstat [opções]
  - -P ALL -exibir estatísticas para todas as CPUs
  - [Num][num] - tempo de coleta dos dados e loop

mpstat

Campos Comuns

- **Timestamp**: A hora em que as estatísticas foram coletadas.
- **CPU**: O número do processador específico ou "all" para exibir as estatísticas para todas as CPUs.
- **%usr**: A porcentagem de tempo da CPU gasto em processos de usuário.
- **%nice**: A porcentagem de tempo da CPU gasto em processos de usuário com um valor "nice" positivo (prioridade mais baixa).
- **%sys**: A porcentagem de tempo da CPU gasto em processos do sistema.
- **%iowait**: A porcentagem de tempo da CPU gasto aguardando operações de I/O.
- **%irq**: A porcentagem de tempo da CPU gasto em tratamento de interrupções.
- **%soft**: A porcentagem de tempo da CPU gasto em tratamento de interrupções de software.
- **%steal**: A porcentagem de tempo da CPU gasto em espera involuntária devido à virtualização.
- **%guest**: A porcentagem de tempo da CPU gasto em processos de máquina virtual hospedada.
- **%gnice**: A porcentagem de tempo da CPU gasto em processos de usuário com um valor "nice" positivo em máquinas virtuais hospedadas.
- **%idle**: A porcentagem de tempo da CPU ociosa.

Utilizando o Sysstat

- Comando **pidstat**
  - Possibilita o monitoramento de informações que estão localizadas no diretório /proc/<pid>/…,
- Sintaxe: pidstat [opções]
  - -d estatística de I/O
  - -u utilização de CPU
  - -p <PID> número do processo
  - -r page faults e utilização da memória
    - [num] [num] intervalo em segundos e número de relatórios.

pidstat

Campos Comuns

- **Timestamp**: A hora em que as estatísticas foram coletadas.
- **UID**: O ID do usuário associado ao processo.
- **PID**: O ID do processo.
- **%usr**: A porcentagem de tempo da CPU gasto pelo processo em modo de usuário.
- **%system**: A porcentagem de tempo da CPU gasto pelo processo em modo de sistema.
- **%guest**: A porcentagem de tempo da CPU gasto pelo processo em modo de máquina virtual hospedada.
- **%CPU**: A porcentagem total de tempo da CPU gasto pelo processo (incluindo todos os modos).
- **CPU:** O número do processador no qual o processo está sendo executado.
- **Command**: O nome do comando ou processo em execução.

Utilizando o Sysstat

- Comando **pidstat**

```
stress --vm 1 --vm-bytes 10M -t 300s
```

```
root@cliente-IPMH61R2:/home/cliente# pidstat -p 21671 -r 2 3
Linux 3.13.0-37-generic (cliente-IPMH61R2)      21-11-2014      _x86_64_        (4 CPU)

12:09:12      UID       PID  minflt/s  majflt/s     VSZ    RSS   %MEM  Command
12:09:14     1000     21671      0,00      0,00    7308    432   0,01  stress
12:09:16     1000     21671      0,00      0,00    7308    432   0,01  stress
12:09:18     1000     21671      0,00      0,00    7308    432   0,01  stress
Average:     1000     21671      0,00      0,00    7308    432   0,01  stress
root@cliente-IPMH61R2:/home/cliente#
```

stress

Campos Comuns

O comando "**stress**" é usado para impor uma quantidade controlada de estresse em vários componentes do sistema, como **CPU**, **memória**, **E/S** ou até mesmo processos específicos.

- `--cpu <número_de_threads>`: Especifica a quantidade de threads da CPU a serem utilizadas para o teste de estresse na CPU.
- `--vm <número_de_processos>`: Define o número de processos para estressar a memória.
- `--vm-bytes <quantidade_de_memória>`: Especifica a quantidade de memória RAM a ser utilizada pelos processos estressantes.
- `--io <número_de_processos>`: Define o número de processos para estressar a E/S (entrada/saída).
- `--hdd <número_de_operações_em_discos>`: Especifica o número de operações de leitura/escrita de disco a serem realizadas durante o teste de estresse na E/S.
- `--timeout <tempo_em_segundos>`: Define o tempo limite (em segundos) para a execução do teste de estresse.
- `--help` ou `-h`: Exibe informações de ajuda sobre o comando `stress`, mostrando a lista completa de parâmetros e opções disponíveis.

Let's coding!

```
#!/bin/bash

tempo=0

echo "time;cpu0;cpu1;cpu2;cpu3;cput;used;free;shared;total;rawait;wawait;" > log_monitoring_process.csv
while [ $tempo -lt 60 ]

do
        time=$(uptime | awk '{print $1}')
        cpu0=$(mpstat -P 0 | tail -n 1 | awk '{print $4}')
        cpu1=$(mpstat -P 1 | tail -n 1 | awk '{print $4}')
        cpu2=$(mpstat -P 2 | tail -n 1 | awk '{print $4}')
        cpu3=$(mpstat -P 3 | tail -n 1 | awk '{print $4}')
        cput=$(mpstat | tail -n 1 | awk '{print $4}')
        used=$(free -m | awk '/Mem/{print $3}')
        free=$(free -m | awk '/Mem/{print $4}')
        shared=$(free -m | awk '/Mem/{print $5}')
        total=$(free -m | awk '/Mem/{print $2}')
        rawait=$(iostat -x | awk '/sda/{print $6}')
        wawait=$(iostat -x | awk '/sda/{print $12}')

        echo "$time;$cpu0;$cpu1;$cpu2;$cpu3;$cput;$used;$free;$shared;$total;$rawait;$wawait;" >>
log_monitoring_process.csv
        sleep 2
        tempo=$((tempo+2))
done
```

Utilizando o dstat

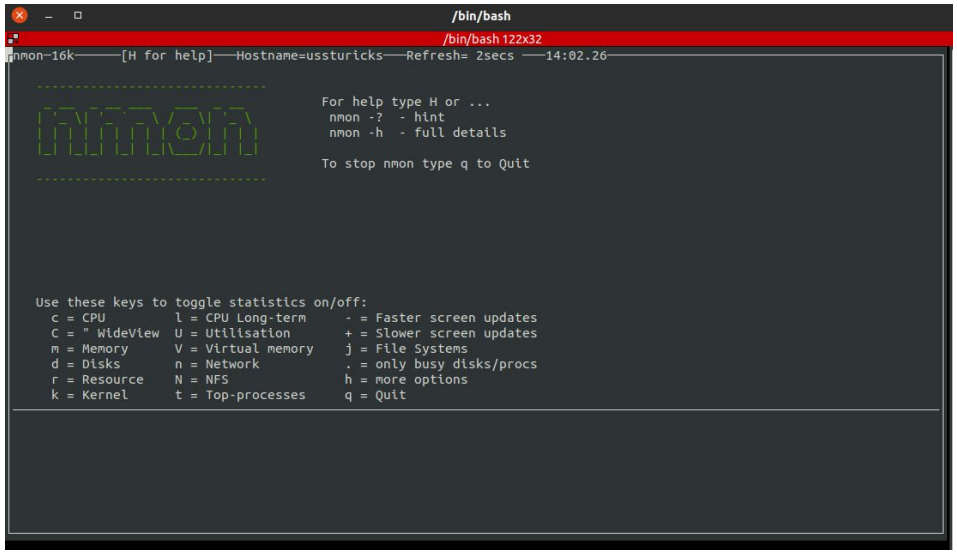
Comando **dstat**

- Permite efetuar monitoramento e verificar performance do sistema Linux, possuindo características dos comandos **top**, **vmstat**, **free**, **iostat** **combinadas**.
  - Sintaxe: dstat [opções]
    - dstat não permite ajustar o intervalo de atualização para n segundos
  - -m uso da memória
  - -c estatística de CPU
  - -d estatística de disco
  - -i interrupções
  - -n estatísticas de uso de rede
  - --fs estatísticas do sistema de arquivos
  - --ntp mostra a hora a partir de um servidor de NTP

Utilizando o NMON

Nigel’s performance **Mon**itor;

- Por default, o nmon inicia em modo live com monitoramento via terminal.



Utilizando o NMON

- Os comandos são organizados como segue:
  - $ nmon -f s “seconds” -c “count”

- Onde -f é salvar em arquivo (file), -s o intervalo entre as capturas e -c o número de capturas.
- O arquivo de saída é um arquivo de texto com extensão *.nmon e no formato CSV.
- A saída vai para o diretório atual, no seguinte formato:
  - <hostname>_<date>_<time>.nmon

Utilizando o NMON

- Possibilita especificar o local onde será armazenado o arquivo de saída, por meio da flag -m
  - $ nmon -f s “seconds” -c “count” -m /home/…

- E agendar o início do experimento
  - $ 0 0 * * * /usr/local/bin/nmon -f -s 120 -c 720 -m /home, onde
  - *0 hora, o minutos, * dias do mês, * mês, * dia da semana* /usr/local…..

Utilizando o NMON

Exemplo: monitorar o sistema por um minuto

- 1 min = 60 segundos;
- 60 s com intervalos entre amostra de 5 s;
- 60/5 = 12 coletas.

Carga de trabalho:

- $stress -m 2 –vm-bytes 1G -t 50s

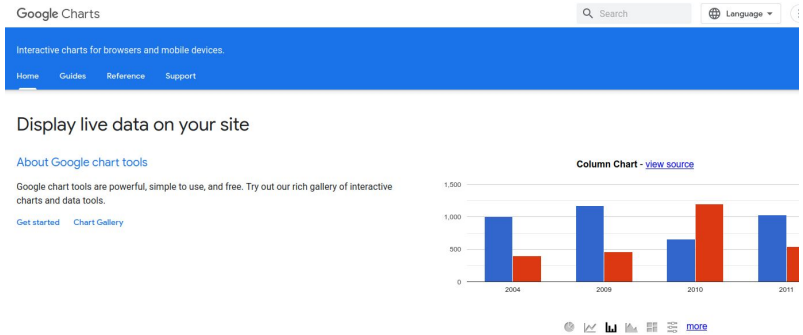
Monitoramento:

- $nmon -f -s 5 -c 12

Utilizando o NMON

Dados obtidos via nmon podem ser analisados através de outras ferramentas (Excel, R, Mathematica). Muitos dados tornam a análise mais precisa, porém implicam um grande massa de dados que em sua análise tende a consumir uma grande quantidade de memória;

- https://developers.google.com/chart é uma biblioteca que pode facilitar, se utilizada em conjunto com o nmonchart.



Utilizando o NMON

1. Instalação do nmonchar https://github.com/aguther/nmonchart
2. Extrair pasta nmonchart-master, e instala o *ksh*;
3. Atribui permissões de leitura, escrita e execução ao *build.sh*;
4. Executando temos:
    a. **$./nmonchart nmonfile.nmon output.html**

Monitoramento de Rede - TCPDUMP

É uma ferramenta de análise de tráfego de rede que, como o próprio nome evidencia, está ligada ao protocolo TCP/IP. Utiliza a libpcap, uma biblioteca C/C++ portátil para captura de tráfego de rede.

- **Instalação**:
  - $ sudo apt install tcpdump
- **Verificação de versões:**
  - tcpdump –version
- **Sitnaxe**:
  - tcpdump [opções]

Monitoramento de Rede - TCPDUMP

Opções principais:

- Lista de interfaces
  - tcpdump -D
- Interface específica
  - tcpdump -i wlp3s0
- Captura N pacotes
  - tcpdump -i wlp3s0 -c 10

Monitoramento de Rede - TCPDUMP

Campos

1. Timestamp;
2. Tipo do pacote;
3. [Endereço de origem].[porta de origem]
4. Direção do fluxo do pacote: **>**
5. [Endereço de destino].[porta de destino]
6. Bit de controle.

Monitoramento de Rede - TCPDUMP

Opções principais

- Exibe somente IP (não resolve nomes).
  - **tcpdump -n -i wlp3s0**
- Grava os pacotes capturados em um arquivo
  - **tcpdump -w file.pcap**
- Faz a leitura dos pacotes a partir de um arquivo de captura.
  - **tcpdump -r file.pcap**
- Faz a leitura dos pacotes a partir de um arquivo e exibe a data e hora da captura
  - **tcpdump -tttt -r file.pcap**

Monitoramento de Rede - TCPDUMP

Opções principais

- Mostra o conteúdo do pacote em HEX e ASCII.
  - **tcpdump -XX -i [nome da placa de rede conectada]**
- Captura os pacotes originados apenas do host com o IP ou hostname.
  - **tcpdump -n src host 192.168.0.1**

Monitoramento de Rede - TCPDUMP

Opções principais

- Captura pacotes destinados apenas ao host com IP ou hostname
  - **tcpdump -i wlp3s0 dst host 192.168.0.1**
- Captura apenas pacotes que sejam maiores que X bytes:
  - **tcpdump -i wlp3s0 greater 100**
    - captura pacotes maiores que 100 bytes.

Monitoramento de Rede - TCPDUMP

Opções principais

- Captura apenas pacotes que sejam menores que X bytes:
  - **tcpdump -i wlp3s0 less 100**
    - captura pacotes menores que 100 bytes.
- Trabalho com pacotes destinados a uma porta específica
  - tcpdump -i wlp3s0 port 22

Monitoramento de Rede - TCPDUMP

Opções principais

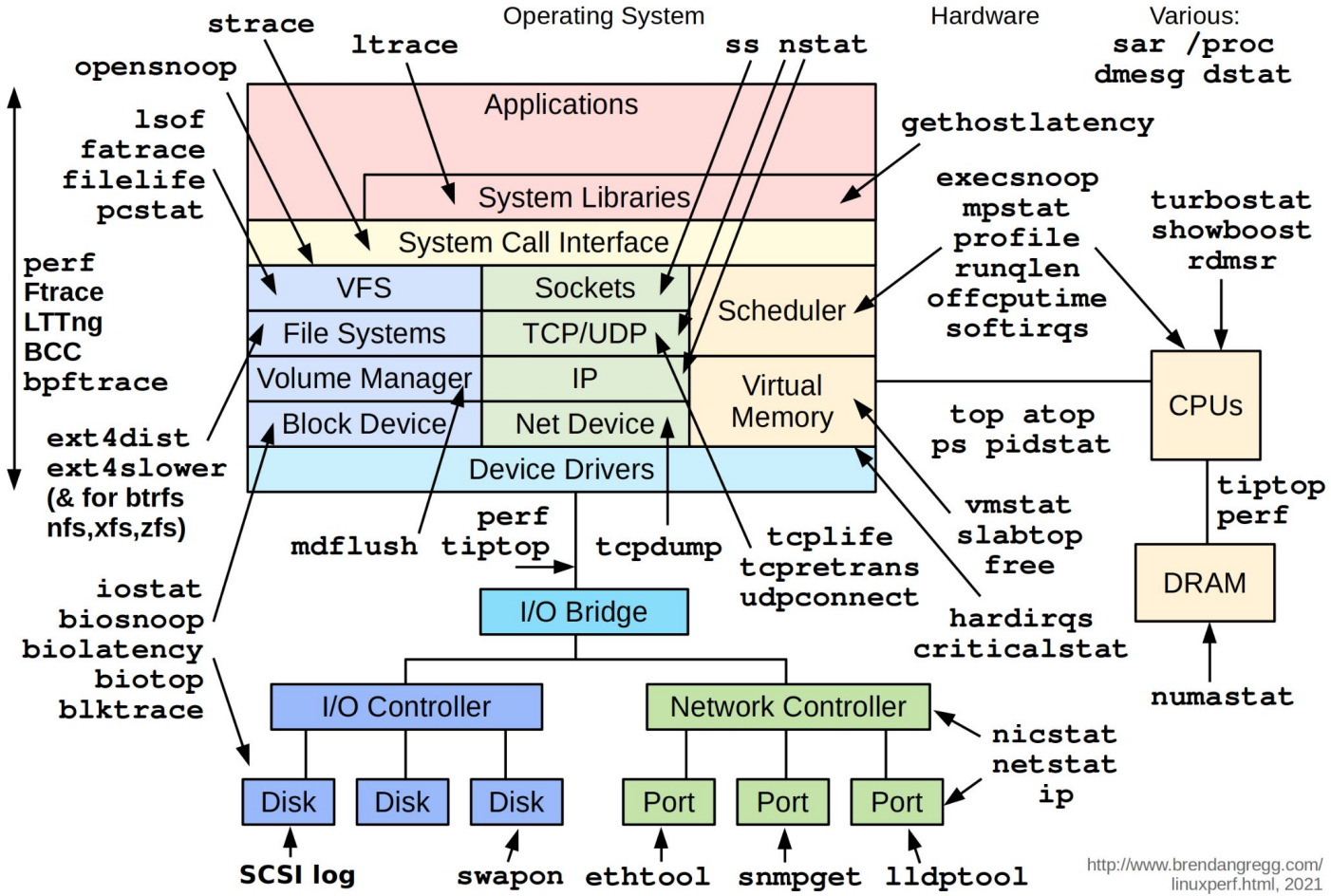
- Captura pacotes em um intervalo de portas determinado
  - **tcpdump -i wlp3s0 portrange 22-80**
- Captura somente tráfego associado ao protocolo ICMP na interface wlp3s0
  - **tcpdump -i wlp3s0 icmp**

Monitoramento de Rede - TCPDUMP

Opções principais

- Captura pacotes somente se satisfazerem às condições A e B simultaneamente.
  - **tcpdump -i wlp3s0 dst A and B**
    - *por exemplo: 192.168.0.1 and icmp*
- Captura os pacotes que não satisfaçam uma condição determinada para A
  - **tcpdump -i wlp3s0 -s 64 not A**
    - *por exemplo: not port 22*

Linux Performance Observability Tools



http://www.brendangregg.com/linuxperf.html, 2021

Utilizando o Crontab

Crontab

- Permite que usuários de sistemas Unix executem comandos ou scripts em uma determinada data e hora. Podendo ser agendados para execução periódica.
  - Sintaxe: crontab [opções]
    - crontab -e
      - executa o ***nano*** e cria um script para o crontab
    - crontab -r
      - deleta o script usado anteriormente
    - crontab -l
      - Lista tudo.

Utilizando o Crontab

```
* * * * * comando a ser executado
- - - - -
| | | | |
| | | | ----- Dia da semana (0 - 7) (domingo = 0 ou 7)
| | | ------- Mês (1-12)
| | --------- Dia do mês (1-31)
| ----------- Hora (0 - 23)
------------- Minuto (0 - 59)
```

Exemplo:

- **10 00 * * * /usr/sbin/tcpdump -n -c 30000 -w ~/monitoring/debug.txt**

Para uma interação a cada tempo utilizar */num:

```
*/2 * * * * comando > /caminho/do/arquivo.log
```

Utilizando o Crontab

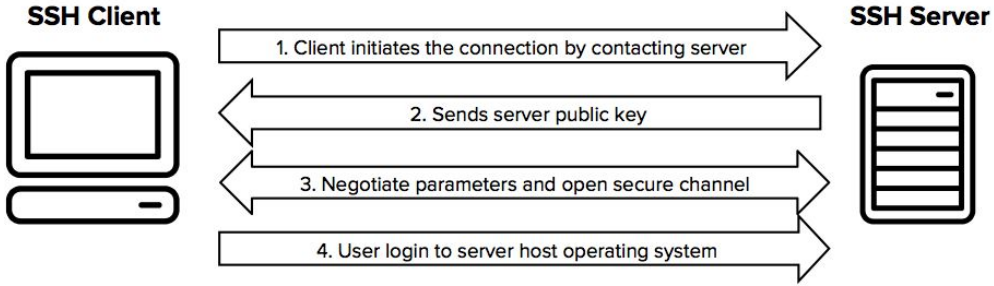
Strings	Significado
@reboot	Execute uma vez, na inicialização.
@yearly	Execute uma vez por ano, "0 0 1 1 *".
@annually	(o mesmo que @yearly)
@monthly	Execute uma vez por mês, "0 0 1 * *".
@weekly	Execute uma vez por semana, "0 0 * * 0".
@daily	Execute uma vez por dia, "0 0 * * *".
@midnight	(o mesmo que @daily)
@hourly	Execute uma vez por hora, "0 * * * *".

Refinando temos: @daily /usr/sbin/tcpdump -n -c 30000 -w ~/monitoring/debug.txt

Utilizando o SSH

SSH (Secure SHELL): O Secure Shell Protocol (SSH) é um protocolo criptográfico de rede para operar serviços de rede com segurança em uma rede não segura. Os aplicativos SSH são baseados em uma arquitetura cliente-servidor, conectando uma instância do cliente SSH com um servidor SSH. O SSH opera como um conjunto de protocolos em camadas compreendendo três componentes hierárquicos principais: a **camada de transporte** fornece autenticação, confidencialidade e integridade do servidor; o **protocolo de autenticação do usuário** valida o usuário para o servidor; e o **protocolo de conexão** multiplexa o túnel criptografado em vários canais lógicos de comunicação.

https://en.wikipedia.org/wiki/Secure_Shell



Utilizando o SSH

SSH (Secure SHELL):

- Instalação
  - sudo apt install openssh-server
    - para habilitar o serviço SSH em sua máquina local
    - porta 22 ficará em modo: *listening* aguardando por conexões remotas
    - acesso: ***usuario_remoto@ip_da_maquina_remota***

- Em distribuições ***unix-like***, o cliente ssh já vem instalado *default*.

Referências

- Man-pages do Linux
- MoDCS http://www.modcs.org/
- Performance Tools
- Advanced TCPDUMP
- Site do iostat:
  - http://sebastien.godard.pagesperso-orange.fr
- Jain, Raj. "The art of computer system performance analysis: techniques for experimental design, measurement, simulation and modeling." *New York: John Willey* (1991).
- Lilja, David J. *Measuring computer performance: a practitioner's guide*. Cambridge University Press, 2005.