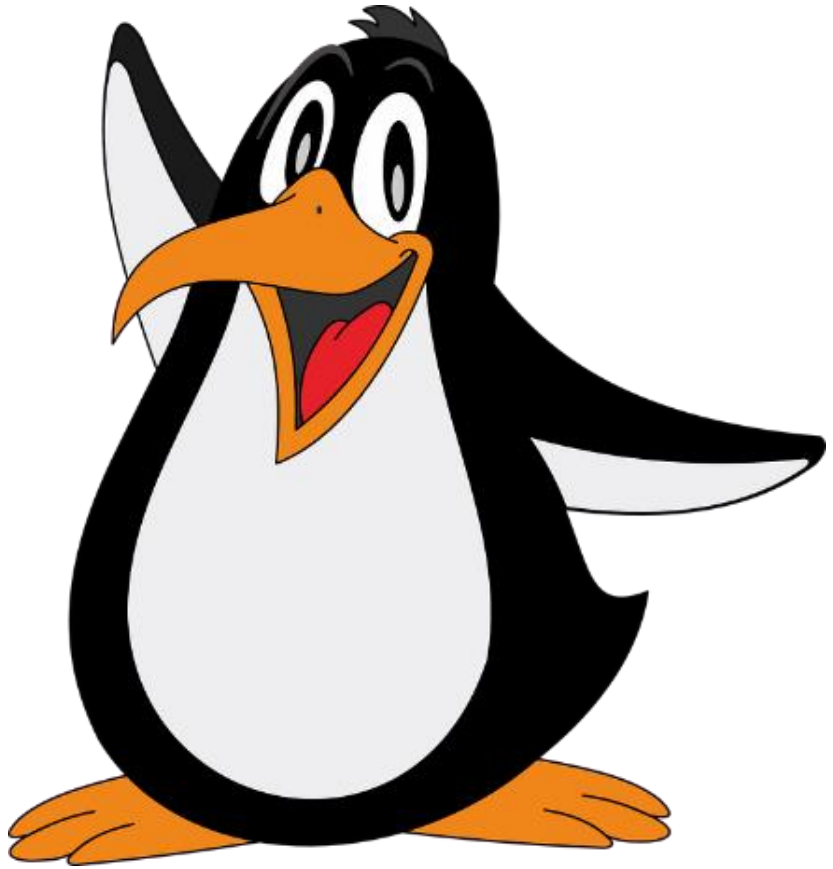
# Sistemas Operacionais II

## O Sistema de Arquivos `/proc`

Fabricio Breve
fabricio.breve@unesp.br
https://www.fabriciobreve.com

# Sumário

- Extraindo Informação de **`/proc`**
- Entradas de Processo
- Informações de *Hardware*
- Informações de Núcleo
- Drives, Montagens, e Sistemas de Arquivos
- Estatísticas do Sistema

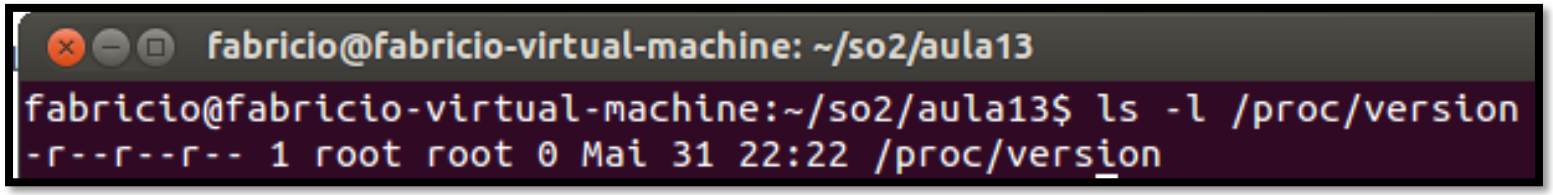


# O Sistema de Arquivos **`/proc`**

- **`/proc`** é um sistema de arquivos especial.
  - Arquivos em **`/proc`** não correspondem a arquivos reais em um dispositivo físico**.**
  - São objetos “mágicos” que se comportam como arquivos, mas fornecem acesso a parâmetros, estruturas de dados, e estatísticas no núcleo.
  - O “conteúdo” de tais arquivos não são blocos fixos de dados, são gerados dinamicamente pelo núcleo do Linux quando você lê do arquivo.
  - É possível mudar a configuração do núcleo em execução escrevendo em alguns arquivos em **`/proc`**.

# O Sistema de Arquivos **`/proc`**

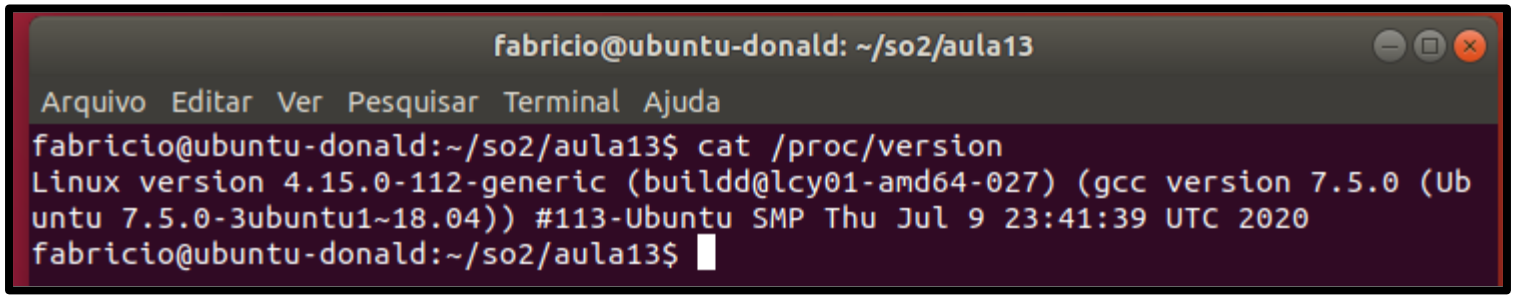
- Exemplo:
  - **`ls -l /proc/version`**
    - O tamanho do arquivo é zero, pois o conteúdo será gerado pelo núcleo, de forma que o conceito de tamanho não se aplica.
    - A hora de modificação do arquivo é a hora do sistema.
    - O “conteúdo” é uma *string* descrevendo a versão do núcleo do Linux.
      - Essas informações são usadas por **`uname`**.



# O Sistema de Arquivos **`/proc`**

- Você pode ler **`/proc/version`** como leria qualquer outro arquivo.
  - Exemplo:
    - **`cat /proc/version`**
  - As várias entradas no sistema de arquivos **`/proc`** são descritas na página de manual de **`proc`**, seção 5.
    - **`man 5 proc`**

Veja em execução:
https://youtu.be/dEem2_SrXfI



# Extraindo Informação de **`/proc`**

- A maioria das entradas em **`/proc`** fornecem informações formatadas para serem lidas por humanos.
  - Porém o formato é simples o suficiente para ser processado por um *parser*.
  - Exemplo:
    - **`cat /proc/cpuinfo`**

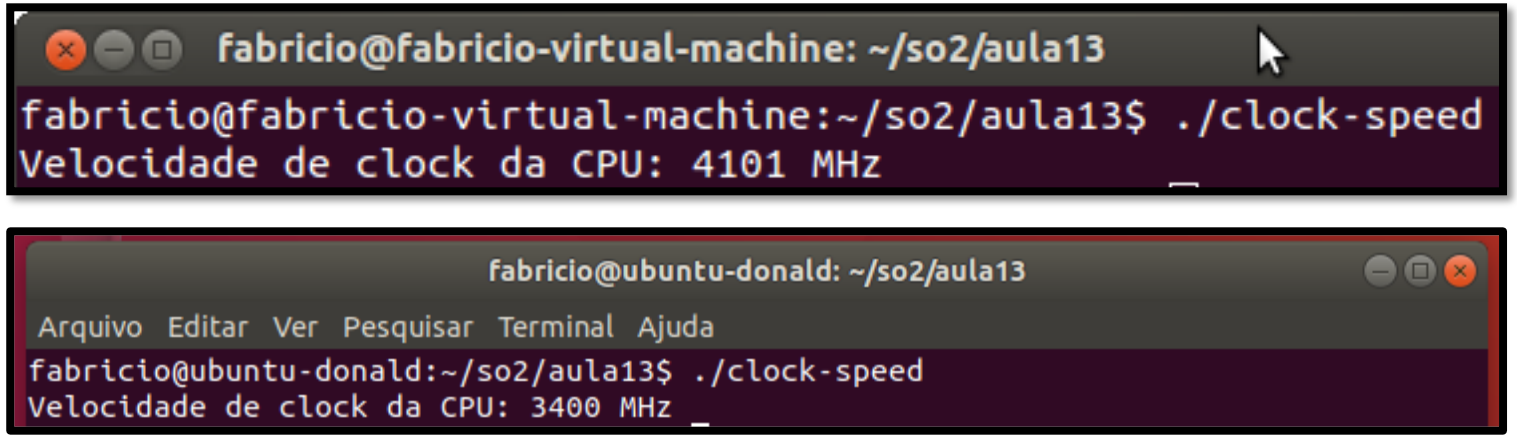
Veja em execução:
https://youtu.be/3tm-aSpdeKg

# Extraindo Informação de **/proc**

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 60
model name      : Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz
stepping        : 3
cpu MHz         : 3400.002
cache size      : 8192 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 4
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx rdtscp lm constant_tsc
rep_good nopl xtopology nonstop_tsc cpuid tsc_known_freq pni pclmulqdq vmx ssse3 cx16 pcid sse4_1 sse4_2 movbe popcnt aes xsave avx rdrand hypervisor lahf_lm
abm invpcid_single pti tpr_shadow flexpriority fsgsbase avx2 invpcid md_clear flush_l1d
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs itlb_multihit srbds
bogomips        : 6800.00
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:
```

# Extraindo Informação de `/proc`

- O programa do exemplo a seguir lê o arquivo em um buffer e faz a análise gramatical utilizando **`sscanf`**, extraindo a velocidade do clock da CPU.



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

/* Retorna a velocidade do clock da CPU do sistema em MHz, conforme reportado por
   /proc/cpuinfo.  Em uma máquina com múltiplos processadores, retorna a velocidade
   da primeira CPU.  Em caso de erro retorna zero.  */

float get_cpu_clock_speed ()
{
  FILE* fp;
  char buffer[65536];
  size_t bytes_read;
  char* match;
  float clock_speed;

  /* Lê o conteúdo completo de /proc/cpuinfo para o buffer.  */
  fp = fopen ("/proc/cpuinfo", "r");
  bytes_read = fread (buffer, 1, sizeof (buffer), fp);
  fclose (fp);
  /* Sai fora se a leitura falhar ou se o buffer não for grande o suficiente.  */
  if (bytes_read == 0 || bytes_read == sizeof (buffer))
    return 0;
  /* termina o texto com NUL.  */
  buffer[bytes_read] = '\0';
  /* Localiza a linha que começa com "cpu MHz".  */
  match = strstr (buffer, "cpu MHz");
  if (match == NULL)
    return 0;
  /* Faz a análise da linha e extrai a velocidade do clock.  */
  sscanf (match, "cpu MHz : %f", &clock_speed);
  return clock_speed;
}


int main ()
{
  printf ("Velocidade de clock da CPU: %4.0f MHz\n", get_cpu_clock_speed ());
  return 0;
}
```

Coloque um tamanho maior no buffer, do contrário ele não será grande o suficiente para armazenar informações de processadores com múltiplos núcleos.

***clock-speed.c***

Extrai a Velocidade de Clock de ***/proc/cpuinfo***

Veja em execução:
https://youtu.be/gl2pDd0V27g

# Entradas de Processos

- Para cada processo rodando no Linux existe um diretório no sistema de arquivos **`/proc`**.
  - O nome de cada diretório é o ID do processo correspondente.
  - Estes diretórios aparecem e desaparecem dinamicamente conforme processos iniciam e terminam no sistema.
  - Cada diretório contém várias entradas fornecendo acesso a informações sobre os processos em execução.

# Entradas de Processos

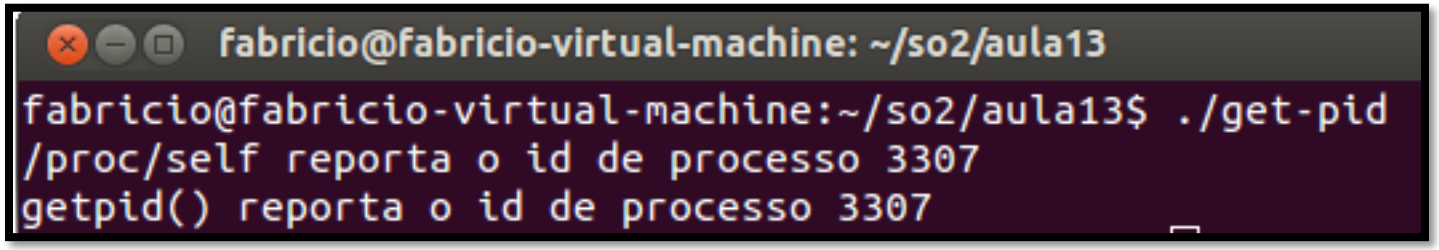
- Cada diretório de processo contém estas entradas:
  - **`cmdline`** contém uma lista de argumentos para o processo.
  - **`cwd`** contém um *link* simbólico que aponta para o diretório de trabalho atual do processo.
  - **`environ`** contém o ambiente do processo.
  - **`exe`** é um *link* simbólico para a imagem do executável do processo.
  - **`fd`** é um subdiretório que contém entradas para descritores de arquivos abertos pelo processo.
  - **`maps`** mostra informações sobre os arquivos mapeados no endereços do processo.
    - Inclui o executável do processo, bibliotecas compartilhadas, etc.
  - **`root`** é um *link* simbólico para o diretório raiz deste processo.
  - **`stat`** contém informações de status e estatísticas do processo.
    - Mesma informação de **status**, mas em formato numérico cru, em uma única linha. Difícil de ler, mas mais adequado para *parsers*.
  - **`statm`** contém informações sobre a memória usada pelo processo.
  - **`status`** contém informações de status e estatísticas do processo formatadas para serem compreendidas por humanos.
  - Etc...

# `/proc/self`

- **`/proc/self`** é um *link* simbólico para o diretório em **`/proc`** que corresponde ao processo atual.
  - Útil para que um programa acesse informações sobre seu próprio processo.
  - O destino do *link* depende do programa que faz a chamada.

# /proc/self

- Exemplo:
  - O programa a seguir lê **/proc/self** para determinar o ID do processo (PID).
    - Este exemplo é apenas ilustrativo, na prática é bem mais fácil usar a função **getpid** para obter o mesmo resultado.
    - Este programa usa a chamada de sistema **readlink** para extrair o alvo do *link* simbólico.



```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

/* Retorna o id de processo para os processos que o chamam, como determinado
   pelo link simbólico /proc/self.  */

pid_t get_pid_from_proc_self ()
{
  char target[32];
  int pid;
  /* Lê o alvo do link simbólico.  */
  readlink ("/proc/self", target, sizeof (target));
  /* O alvo é um diretório nomeado para o id de processo.  */
  sscanf (target, "%d", &pid);
  return (pid_t) pid;
}

int main ()
{
  printf ("/proc/self reporta o id de processo %d\n",
          (int) get_pid_from_proc_self ());
  printf ("getpid() reporta o id de processo %d\n", (int) getpid
());
  return 0;
}
```

***get-pid.c***

Obtém o ID de Processo de **`/proc/self`**

Veja em execução:
https://youtu.be/IdXSkWyktXQ

# Lista de Argumentos de Processos

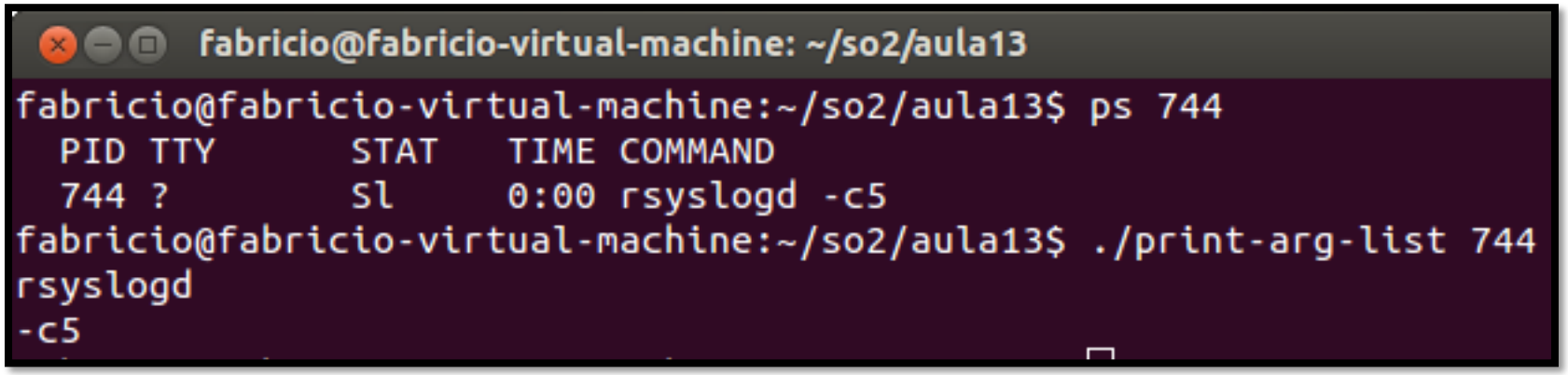
- A entrada **`cmdline`** contém a lista de argumentos do processo.
  - É apresentada como uma única *string* de caracteres, com argumentos separados por **`NUL`**.
  - A maioria das funções de *string* esperam que a *string* toda seja terminada com um único **`NUL`** e não lidam com **`NUL`** embutido dentro da *string*, portanto você deve lidar com esse conteúdo de maneira especial.

# **NUL** versus **NULL**

- **NUL** é o caractere com valor inteiro **0**, diferente de **NULL** que é um ponteiro com valor **0**.
- Em C, as *strings* normalmente são terminadas com **NUL**.
  - Exemplo: “Olá, mundo!” é uma *string* que ocupa 12 bytes, pois há um **NUL** implícito após o ponto de exclamação, indicando o fim da *string*.
- **NULL** é um valor de ponteiro que nunca corresponderá a um endereço real de memória em seus programas.
- Em C, **NUL** é representado como a constante de caractere **‘\0’**, ou **(char) 0**.
- NULL depende do sistema operacional
  - No Linux é definido como **((void *) 0)** em C e simplesmente **0** em C++

# Lista de Argumentos de Processos

- Exemplo:
  - O programa a seguir usa as entradas **`cmdline`** em **`/proc`** para imprimir a lista de argumentos do processo cujo ID for especificado.
    - Como podem existir vários **`NUL`** no conteúdo de **`cmdline`**, não podemos usar **`strlen`** para verificar o tamanho da *string* (ela conta até encontrar **`NUL`**).
      - Em vez disso, pegamos a quantidade de *bytes* lidos retornada por **`read`**.



```c
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <string.h>

/* Imprime a lista de argumentos, um argumento por linha, do processo
   dado por PID.  */

void print_process_arg_list (pid_t pid)
{
  int fd;
  char filename[24];
  char arg_list[1024];
  size_t length;
  char* next_arg;

  /* Gera o nome do arquivo cmdline para o processo. */
  snprintf (filename, sizeof (filename), "/proc/%d/cmdline", (int) pid);
  /* Lê o conteúdo do arquivo. */
  fd = open (filename, O_RDONLY);
  length = read (fd, arg_list, sizeof (arg_list));
  close (fd);
  /* read não termina o buffer com NUL, portanto vamos fazê-lo aqui. */
  arg_list[length] = '\0';

  /* Loop nos argumentos. Argumentos são separados por NULs. */
  next_arg = arg_list;
  while (next_arg < arg_list + length) {
    /* Imprime os argumentos. Cada um é terminado em NUL, portanto podemos
       tratá-los como uma lista comum de strings.  */
    printf ("%s\n", next_arg);
    /* Avança para o próximo argumento. Como cada argumento é
       terminado em NUL, strlen conta o tamanho do próximo argumento,
       não a lista toda de argumentos.  */
    next_arg += strlen (next_arg) + 1;
  }
}
```

Inclua **<string.h>** para evitar o aviso na compilação

```c
int main (int argc, char* argv[])
{
  pid_t pid = (pid_t) atoi (argv[1]);
  print_process_arg_list (pid);
  return 0;
}
```

## *print-arg-list.c*

Imprime a Lista de Argumentos de um Processo em Execução

Veja em execução:
https://youtu.be/Z95cdW2vxfQ

# Ambiente de Processos

- A entrada **`environ`** contém o ambiente do processo.
  - As variáveis de ambiente são separadas por **`NUL`**s.
  - Exemplo:
    - O programa a seguir recebe um ID de processo na linha de comando e imprime o ambiente daquele processo lido de **`/proc`**.

```c
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <string.h>

/* Imprime o ambiente, uma variável por linha, do
   processo dado por PID.  */

void print_process_environment (pid_t pid)
{
  int fd;
  char filename[24];
  char environment[65536];
  size_t length;
  char* next_var;

  /* Gera o nome do arquivo de ambiente para o processo. */
  snprintf (filename, sizeof (filename), "/proc/%d/environ", (int) pid);
  /* Read the contents of the file. */
  fd = open (filename, O_RDONLY);
  length = read (fd, environment, sizeof (environment));
  close (fd);
  /* read não temrina o buffer com NUL, portanto vamos fazê-lo aqui. */
  environment[length] = '\0';

  /* Loop nas variáveis. Variáveis são separadas por NULs. */
  next_var = environment;
  while (next_var < environment + length) {
    /* Imprime a variável. Cada variável é terminada em NUL, então trate-as
       como strings comuns.  */
    printf ("%s\n", next_var);
    /* Avance para a próxima variável. Como cada variável é
       terminada em NUL, strlen conta o tamanho da próxima variável,
       não a lista inteira de variáveis.  */
    next_var += strlen (next_var) + 1;
  }
}
```

Inclua **<string.h>** para evitar o aviso na compilação

Coloque um tamanho maior no buffer, do contrário ele não será grande o suficiente para armazenar ambientes com muitas variáveis

```c
int main (int argc, char* argv[])
{
  pid_t pid = (pid_t) atoi (argv[1]);
  print_process_environment (pid);
  return 0;
}
```

***print-environment.c***

Acessando as Variáveis de Ambiente de um Processo

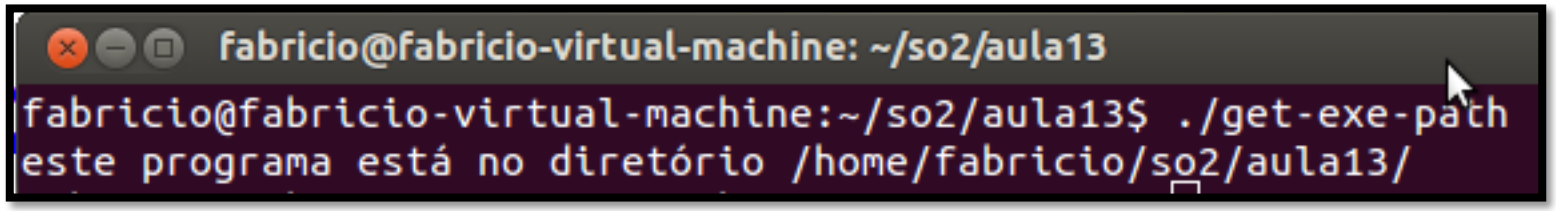
Veja em execução:
https://youtu.be/WoGX1D2c01g

# Executáveis de Processos

- A entrada **exe** aponta para o arquivo executável que está executando no processo.
  - Já vimos que o nome do executável do programa é tipicamente passado como primeiro elemento da lista de argumentos.
    - Mas isto é apenas convenção, um programa pode ser chamado com qualquer lista de argumentos.
    - A entrada **exe** é uma maneira mais confiável de determinar qual executável está sendo executado.

# Executáveis de Processos

- As vezes é útil extrair o caminho que contém o executável do sistema de arquivos **`/proc`**.
  - Para muitos programas, arquivos auxiliares estão instalados em diretórios com caminhos conhecidos com relação ao executável do programa principal, portanto é necessário determinar onde o executável realmente está.
  - No exemplo a seguir, a função **`get_executable_path`** determina o caminho do executável que está executando o processo chamador através do link simbólico **`/proc/self/exe`**



```c
#include <limits.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

/* Encontre o caminho contendo o programa executável atualmente sendo executado.
   O caminho é colocado em BUFFER, cujo tamnho é LEN.  Retorna
   o número de caracteres no caminho, ou -1 em caso de erro.  */

size_t get_executable_path (char* buffer, size_t len)
{
  char* path_end;
  /* Lê o alvo de /proc/self/exe.  */
  if (readlink ("/proc/self/exe", buffer, len) <= 0)
    return -1;
  /* Encontra a última ocorrência de uma barra para frente, o separador de caminho.  */
  path_end = strrchr (buffer, '/');
  if (path_end == NULL)
    return -1;
  /* Avança para o caractere após a última barra.  */
  ++path_end;
  /* Obtém o diretório contendo o programa truncando o
     caminho após a última barra.  */
  *path_end = '\0';
  /* O tamanho do caminho é o número de caracteres até a
     última barra.  */
  return (size_t) (path_end - buffer);
}

int main ()
{
  char path[PATH_MAX];
  get_executable_path (path, sizeof (path));
  printf ("este programa está no diretório %s\n", path);
  return 0;
}
```

***get-exe-path.c***

Pega o Caminho do Executável do Programa sendo Executado Atualmente

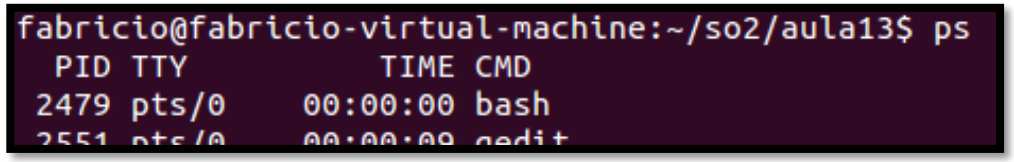
Veja em execução:
https://youtu.be/WuC9QKCdves

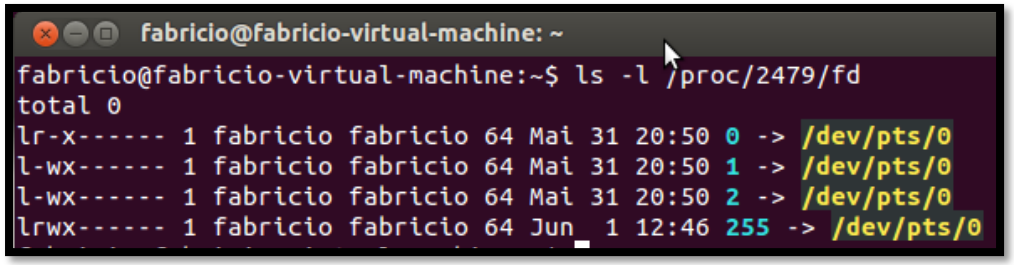
# Descritores de Arquivos de Processos

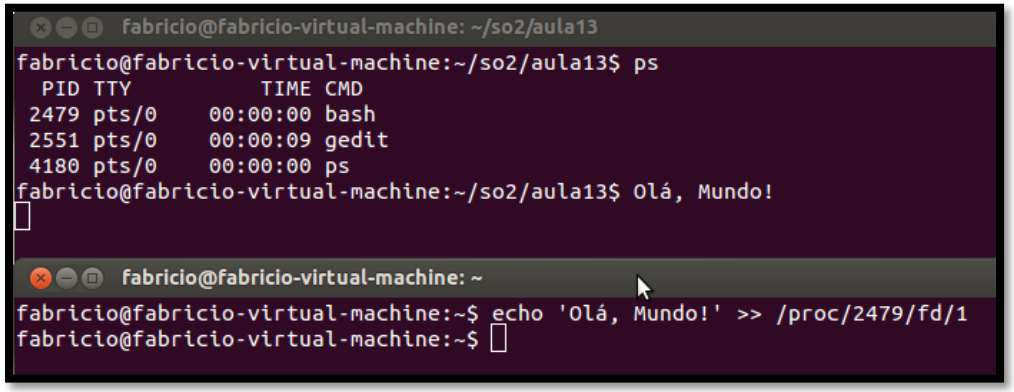
- A entrada **`fd`** é um subdiretório que contém entradas para os descritores de arquivos abertos por um processo.
  - Cada entrada é um *link* simbólico para o arquivo ou dispositivo aberto naquele descritor de arquivo.
  - Você pode escrever ou ler destes *links* simbólicos.
    - Isto escreve ou lê dos arquivos ou dispositivos correspondentes que estão abertos no processo alvo.
  - As entradas no subdiretório **`fd`** são nomeadas pelos números dos descritores de arquivos.

# Descritores de Arquivos de Processos

- Exemplo:
  - Abras dois terminais.
  - No primeiro terminal, use **ps** para saber o ID de processo da *shell*.
  - No segundo terminal, veja o conteúdo do subdiretório **`fd`** para aquele processo.
  - Lembre-se que os descritores de arquivos **`0`**, **`1`** e **`2`** estão associados com entrada, saída e erro padrão, respectivamente. Portanto, escreva para o dispositivo ligado ao **`stdout`** no processo da *shell*, que nesse caso é o pseudo TTY da primeira janela.




Veja em execução:
https://youtu.be/8_uiHXLipNg

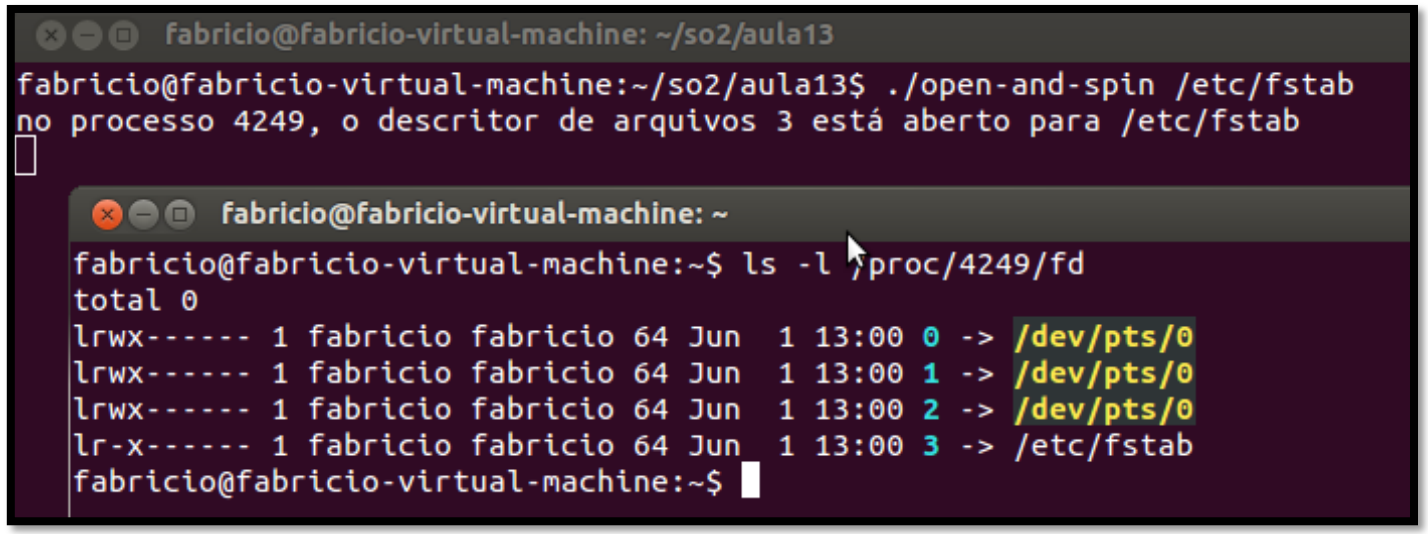
# Descritores de Arquivos de Processos

- Descritores de arquivos além de entrada, saída e erro padrão também aparecem no subdiretório **`fd`**.
- O programa a seguir, abre um descritor de arquivos para um arquivo especificado na linha de comando e entra em um *loop* infinito.

***open-and-spin.c***

Abre um Arquivo para Leitura

```c
#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main (int argc, char* argv[])
{
  const char* const filename = argv[1];
  int fd = open (filename, O_RDONLY);
  printf ("no processo %d, o descritor de arquivos %d está aberto para %s\n",
          (int) getpid (), (int) fd, filename);
  while (1);
  return 0;
}
```



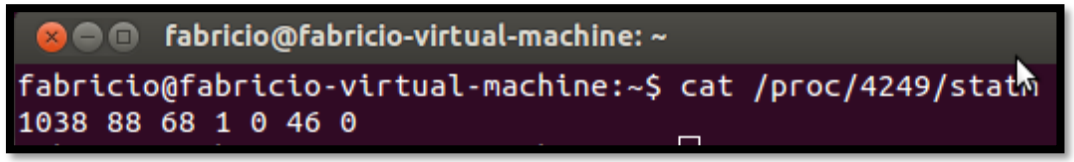

Veja em execução:
https://youtu.be/eUMyemVqEQ0

# Descritores de Arquivos de Processos

- Descritores de arquivos para *pipes* ou *sockets* também aparecem em **`fd`**
  - Neste caso, o *link* simbólico correspondente ao descritor de arquivo mostrará “socket” ou “pipe” em vez de apontar para um arquivo ou dispositivo comum.

# Estatísticas de Memória de Processos

- A entrada **`statm`** contém uma lista de 7 números, separados por espaços. Cada número é um contador do número de páginas de memória usada pelo processo em uma categoria particular. As categorias são, por ordem de aparição:
  - O tamanho total do processo.
  - O tamanho do processo residente em memória física.
  - A memória compartilhada com outros processos – isto é, memória mapeada pelo processo e pelo menos um outro (tais como bibliotecas ou páginas de cópia-na-escrita não tocadas).
  - Tamanho de texto do processo – isto é, o tamanho do código executável carregado.
  - O tamanho das bibliotecas compartilhadas mapeadas neste processo.
  - A memória usada por este processo em sua pilha.
  - O número de páginas sujas – isto é, páginas de memória que foram modificadas pelo programa.




Veja em execução:
https://youtu.be/Q4ld55vZcYk

# Estatísticas do Processo

- A entrada **`status`** contém uma variedade de informações sobre o processo, formatadas para serem compreendidas por humanos.
- Dentre as informações estão os IDs do processo e do processo pai, os IDs de usuário e grupo reais e efetivos, uso de memória, e máscaras de bits especificando quais sinais são capturados, ignorados, e bloqueados.

Veja em execução:
https://youtu.be/KqyEZqEsKeI

```
fabricio@fabricio-virtual-machine:~$ cat /proc/4249/status
Name:                   open-and-spin
State:                  R (running)
Tgid:                   4249
Pid:                    4249
PPid:                   2479
TracerPid:              0
Uid:                    1000    1000    1000    1000
Gid:                    1000    1000    1000    1000
FDSize:                 256
Groups:                 4 24 27 30 46 109 124 1000
VmPeak:                  4220 kB
VmSize:                  4152 kB
VmLck:                      0 kB
VmPin:                      0 kB
VmHWM:                   352 kB
VmRSS:                   352 kB
VmData:                   48 kB
VmStk:                   136 kB
VmExe:                     4 kB
VmLib:                  1876 kB
VmPTE:                    28 kB
VmSwap:                     0 kB
Threads:                1
SigQ:                   0/7799
SigPnd:                 0000000000000000
ShdPnd:                 0000000000000000
SigBlk:                 0000000000000000
SigIgn:                 0000000000000000
SigCgt:                 0000000000000000
CapInh:                 0000000000000000
CapPrm:                 0000000000000000
CapEff:                 0000000000000000
CapBnd:                 ffffffffffffffff
Cpus_allowed:           ffffffff
Cpus_allowed_list:      0-31
Mems_allowed:           00000000,00000001
Mems_allowed_list:      0
voluntary_ctxt_switches:0
nonvoluntary_ctxt_switches:             63118
```

# Informações de Hardware

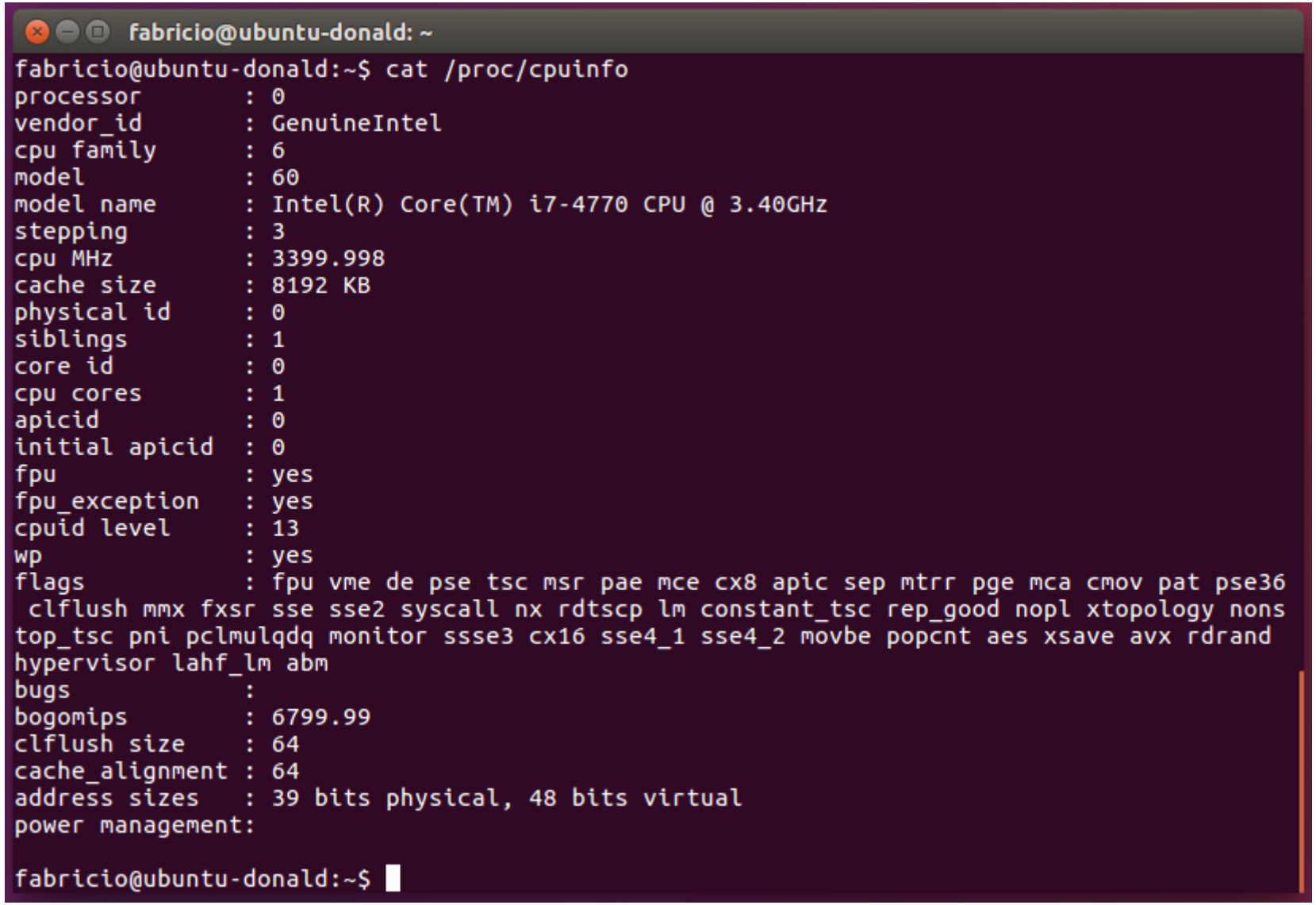
- Muitas das outras entradas no sistema de arquivos **`/proc`** fornecem informações sobre o hardware do sistema.
- A seguir veremos algumas das mais úteis.

# Informações da CPU

- **`/proc/cpuinfo`** contém informações sobre a CPU (ou CPUs) executando o sistema Linux.
  - O campo **`Processor`** indica o número do processador: **`0`**, **`1`**, **`2`**, **`3`**, etc.
  - Os campos **`Vendor`**, **`CPU Family`**, **`Model`** e **`Stepping`** permitem determinar o modelo exato e revisão da CPU.
  - Os campos de *flag* permitem verificar quais *flags* estão ajustadas, o que indica características disponíveis no processador.
    - Exemplos: instruções MMX, SSE, SSE2, etc.
- A maioria das informações são derivadas da instrução **`cpuid`** de assembly x86.
- O último elemento, **`bogomips`** é uma medida de desempenho do processador calculada pelo Linux.
  - Obtida com um pequeno *loop*, não é um bom indicador do desempenho geral do processador.

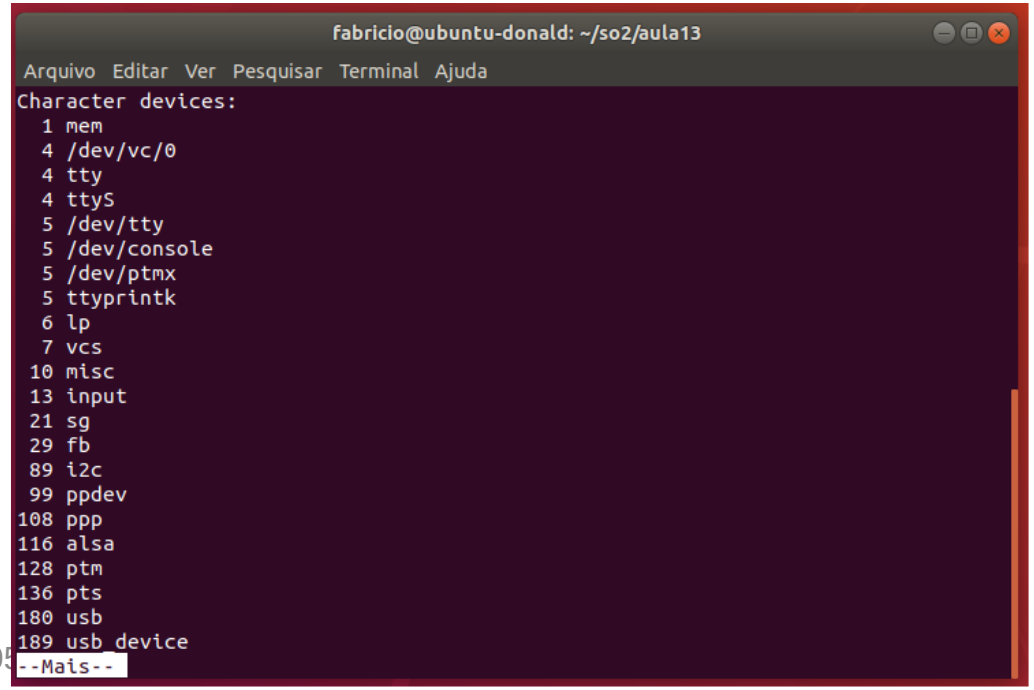
Veja em execução:
https://youtu.be/N2ERsWQmJG8

# Informações da CPU



# Informações de Dispositivos

- O arquivo **`/proc/devices`** lista os números de dispositivo principais para dispositivos de caractere e de bloco disponíveis no sistema.



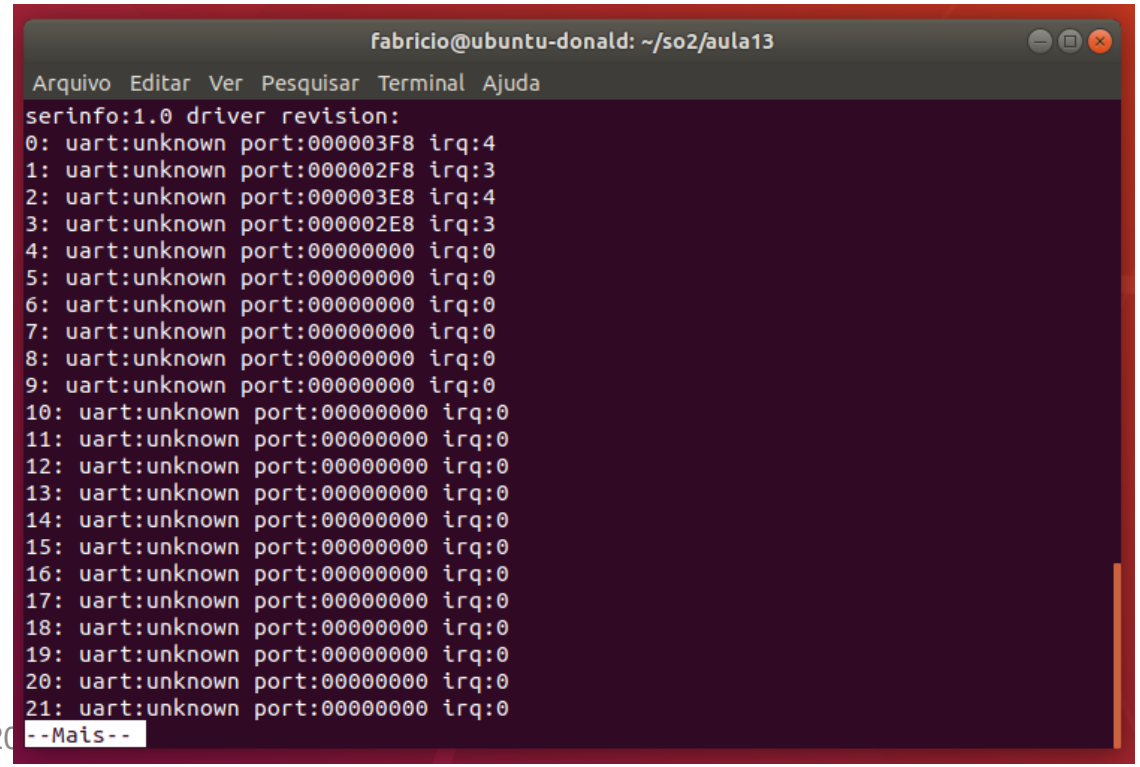

```
fabricio@fabricio-virtual-machine:~$ cat /proc/devices
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  5 ttyprintk
  6 lp
  7 vcs
 10 misc
 13 input
 14 sound
 21 sg
 29 fb
 99 ppdev
108 ppp
116 alsa
128 ptm
136 pts
180 usb
189 usb_device
216 rfcomm
226 drm
251 hidraw
252 usbmon
253 bsg
254 rtc
```

```
Block devices:
  1 ramdisk
  2 fd
259 blkext
  7 loop
  8 sd
  9 md
 11 sr
 65 sd
 66 sd
 67 sd
 68 sd
 69 sd
 70 sd
 71 sd
128 sd
129 sd
130 sd
131 sd
132 sd
133 sd
134 sd
135 sd
252 device-mapper
253 virtblk
254 mdp
```

Veja em execução:
https://youtu.be/VoQJz7dg7Ks

# Informações sobre Portais Seriais

- **`/proc/tty/driver/serial`** lista informações de configuração e estatísticas sobre portas seriais.



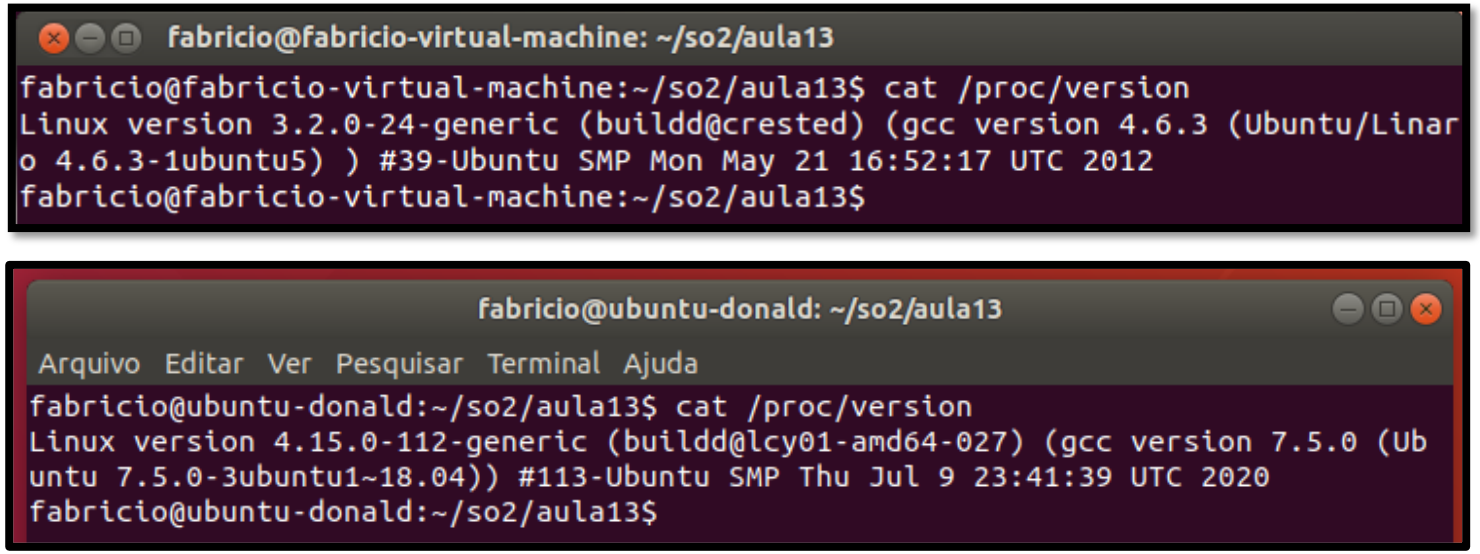

```
fabricio@fabricio-virtual-machine:~/so2/aula13$ sudo cat /proc/tty/driver/serial
serinfo:1.0 driver revision:
0: uart:16550A port:000003F8 irq:4 tx:84 rx:7 CTS|DSR|CD
1: uart:16550A port:000002F8 irq:3 tx:0 rx:0 CTS|DSR|CD
2: uart:unknown port:000003E8 irq:4
3: uart:unknown port:000002E8 irq:3
4: uart:unknown port:00000000 irq:0
5: uart:unknown port:00000000 irq:0
6: uart:unknown port:00000000 irq:0
7: uart:unknown port:00000000 irq:0
8: uart:unknown port:00000000 irq:0
9: uart:unknown port:00000000 irq:0
10: uart:unknown port:00000000 irq:0
11: uart:unknown port:00000000 irq:0
12: uart:unknown port:00000000 irq:0
13: uart:unknown port:00000000 irq:0
14: uart:unknown port:00000000 irq:0
15: uart:unknown port:00000000 irq:0
16: uart:unknown port:00000000 irq:0
17: uart:unknown port:00000000 irq:0
18: uart:unknown port:00000000 irq:0
19: uart:unknown port:00000000 irq:0
20: uart:unknown port:00000000 irq:0
21: uart:unknown port:00000000 irq:0
22: uart:unknown port:00000000 irq:0
23: uart:unknown port:00000000 irq:0
24: uart:unknown port:00000000 irq:0
25: uart:unknown port:00000000 irq:0
26: uart:unknown port:00000000 irq:0
27: uart:unknown port:00000000 irq:0
28: uart:unknown port:00000000 irq:0
29: uart:unknown port:00000000 irq:0
30: uart:unknown port:00000000 irq:0
31: uart:unknown port:00000000 irq:0
```

# Informações de Núcleo

- Muitas das entradas em **`/proc`** fornecem acesso a informações sobre configurações e estado do núcleo.
- Veremos algumas delas a seguir.

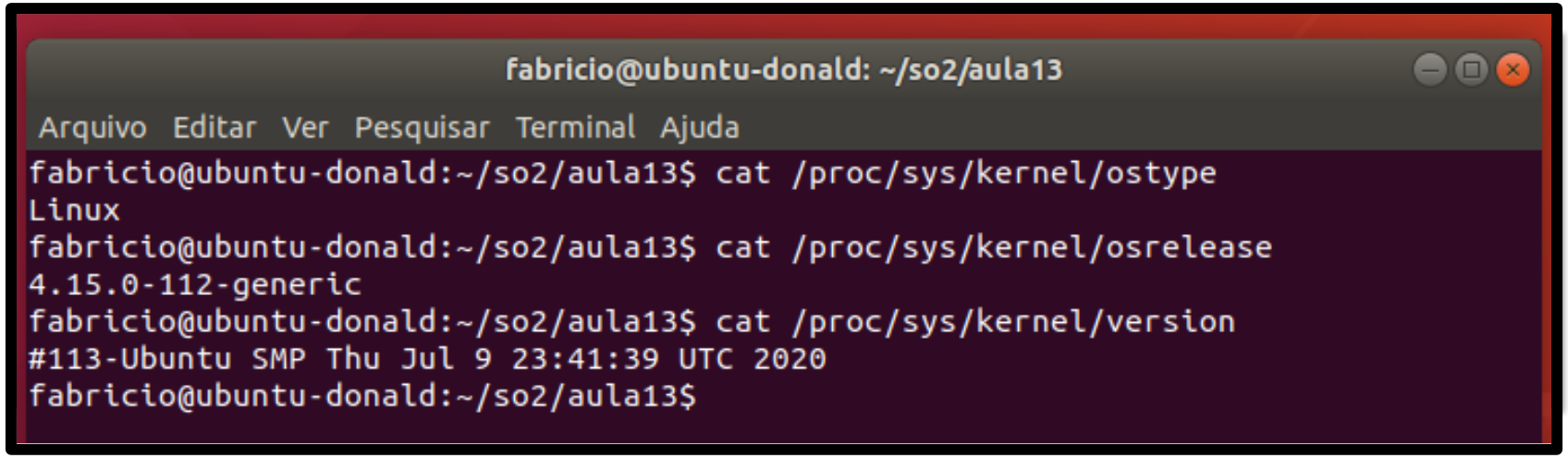
# Informações de Versão

- **`/proc/version`** contém uma longa *string* descrevendo o número de versão do núcleo e de como foi construído: usuário que compilou, máquina em que foi compilado, data que foi compilado, e versão do compilador utilizada.



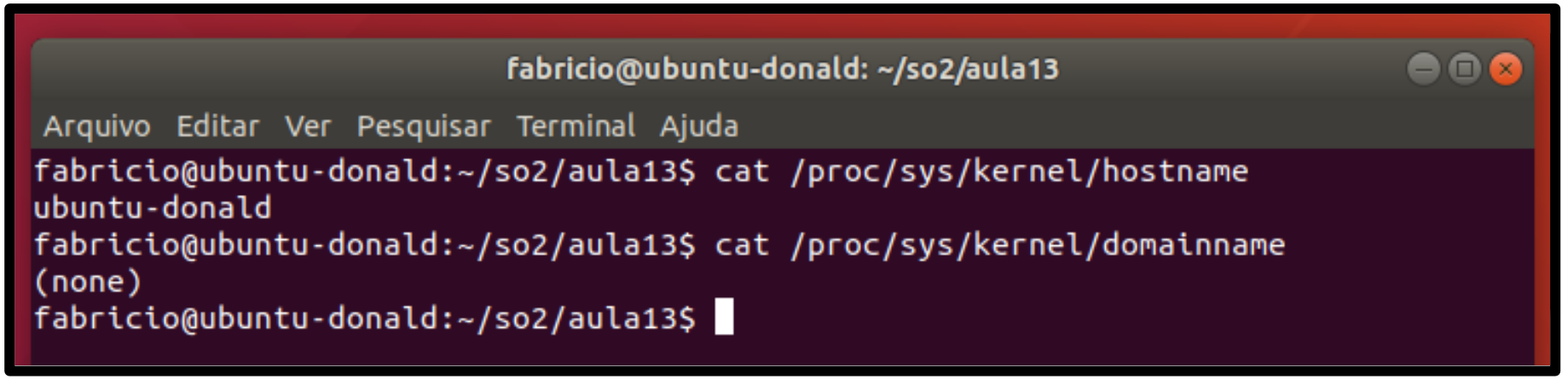
# Informações de Versão

- Alguns dos itens mais importantes também estão disponíveis em entradas separadas:



# Nome de Hospedeiro e de Domínio

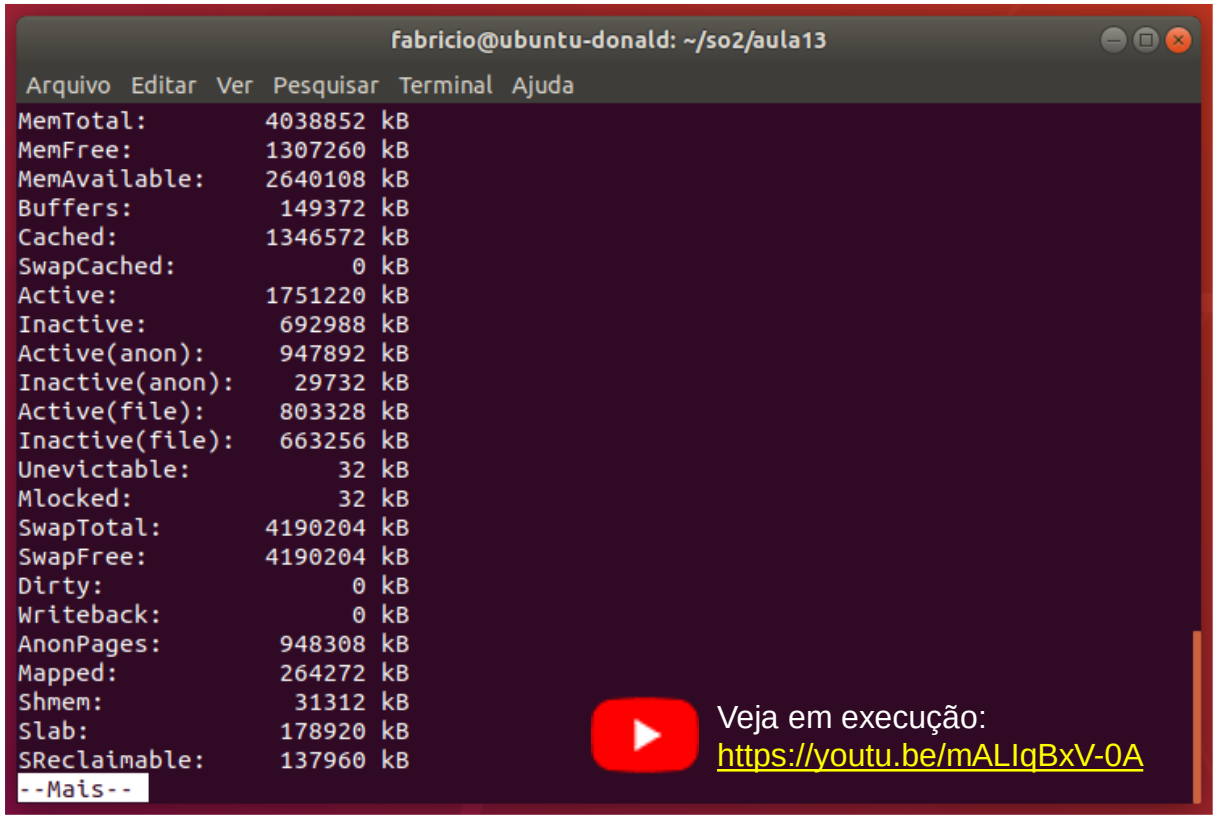
- **/proc/sys/kernel/hostname**
- **/proc/sys/kernel/domainname**




Veja em execução:
https://youtu.be/FMl1LxWC4Yo

# Uso de Memória

- A entrada **`/proc/meminfo`** contém informações sobre o uso de memória do sistema.




```
fabricio@ubuntu-donald:~/so2/aula13$ cat /proc/meminfo
MemTotal:       4038852 kB
MemFree:        1207892 kB
MemAvailable:   2578792 kB
Buffers:         150572 kB
Cached:         1375360 kB
SwapCached:           0 kB
Active:         1808396 kB
Inactive:        732584 kB
Active(anon):   1003612 kB
Inactive(anon):   32956 kB
Active(file):    804784 kB
Inactive(file):  699628 kB
Unevictable:         32 kB
Mlocked:             32 kB
SwapTotal:      4190204 kB
SwapFree:       4190204 kB
Dirty:               24 kB
Writeback:            0 kB
AnonPages:      1015188 kB
Mapped:          288576 kB
Shmem:            34536 kB
Slab:            179600 kB
SReclaimable:    138396 kB
SUnreclaim:       41204 kB
KernelStack:      10704 kB
PageTables:       41452 kB
NFS_Unstable:         0 kB
Bounce:               0 kB
WritebackTmp:         0 kB
CommitLimit:    6209628 kB
Committed_AS:   5744260 kB
VmallocTotal:   34359738367 kB
VmallocUsed:          0 kB
VmallocChunk:         0 kB
HardwareCorrupted:    0 kB
AnonHugePages:    10240 kB
ShmemHugePages:       0 kB
ShmemPmdMapped:       0 kB
CmaTotal:             0 kB
CmaFree:              0 kB
HugePages_Total:      0
HugePages_Free:       0
HugePages_Rsvd:       0
HugePages_Surp:       0
Hugepagesize:      2048 kB
DirectMap4k:     192448 kB
DirectMap2M:    4001792 kB
```

# Drives, Montagens e Arquivos de Sistema

- O sistema de arquivos **`/proc`** também contém informações sobre drives de disco presentes no sistema e sistemas de arquivos montados a partir deles.

# Sistemas de Arquivos

- A entrada **`/proc/filesystems`** mostra os tipos de sistemas de arquivos conhecidos pelo núcleo.
  - Não é muito útil porque não é completo. Sistemas de arquivos podem ser carregados e descarregados dinamicamente como módulos de núcleo.
  - Mostra apenas sistemas de arquivos *linkados* estaticamente ou atualmente carregados como módulos.
  - Outros sistemas de arquivos podem estar disponíveis, mas não carregados.

```
fabricio@ubuntu-donald:~/so2/aula13$ cat /proc/filesystems
nodev	sysfs
nodev	rootfs
nodev	ramfs
nodev	bdev
nodev	proc
nodev	cpuset
nodev	cgroup
nodev	cgroup2
nodev	tmpfs
nodev	devtmpfs
nodev	configfs
nodev	debugfs
nodev	tracefs
nodev	securityfs
nodev	sockfs
nodev	dax
nodev	bpf
nodev	pipefs
nodev	hugetlbfs
nodev	devpts
	ext3
	ext2
	ext4
	squashfs
	vfat
nodev	ecryptfs
	fuseblk
nodev	fuse
nodev	fusectl
nodev	pstore
nodev	mqueue
nodev	autofs
	iso9660
```

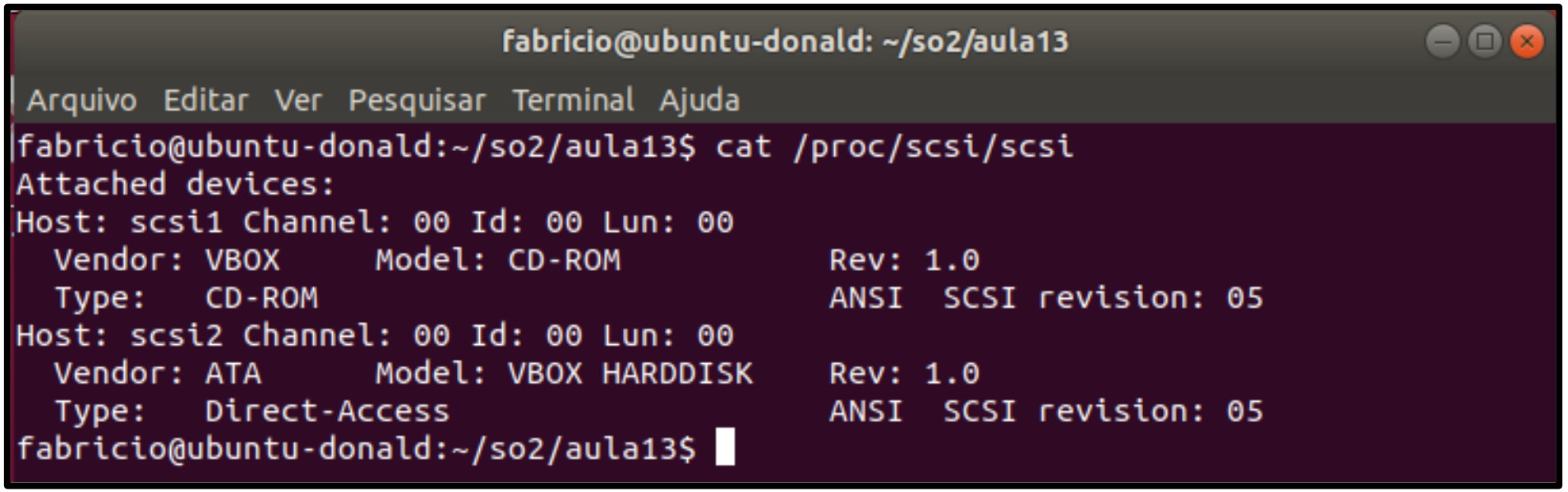
Veja em execução:
https://youtu.be/qNwbO0U66uo

# Drives e Partições

- Se o sistema tiver drives IDE, eles estarão listados em **`/proc/ide`**
  - Para cada controladora haverá um subdiretório.
    - Exemplo: **`/proc/ide/ide0/`**
  - Para cada dispositivo físico, haverá um subdiretório no diretório da controladora.
    - Exemplo: **`/proc/ide/ide0/hda/`**
  - Dentro do diretório de cada dispositivo há informações como tipo de mídia, modelo do dispositivo, capacidade, etc.

# Drives e Partições

- Se drives SATA ou SCSI estiverem presentes, o arquivo **`/proc/scsi/scsi`** conterá um resumo de seus valores de identificação:




Veja em execução:
https://youtu.be/4cWQ6hjv_Nc

# Drives e Partições

- A entrada **`/proc/partitions`** mostra as partições de dispositivos de disco reconhecidas.
  - Para cada partição, a saída inclui os números de dispositivo, a quantidade de blocos de 1024 *bytes*, e o nome de dispositivo que corresponde à partição.

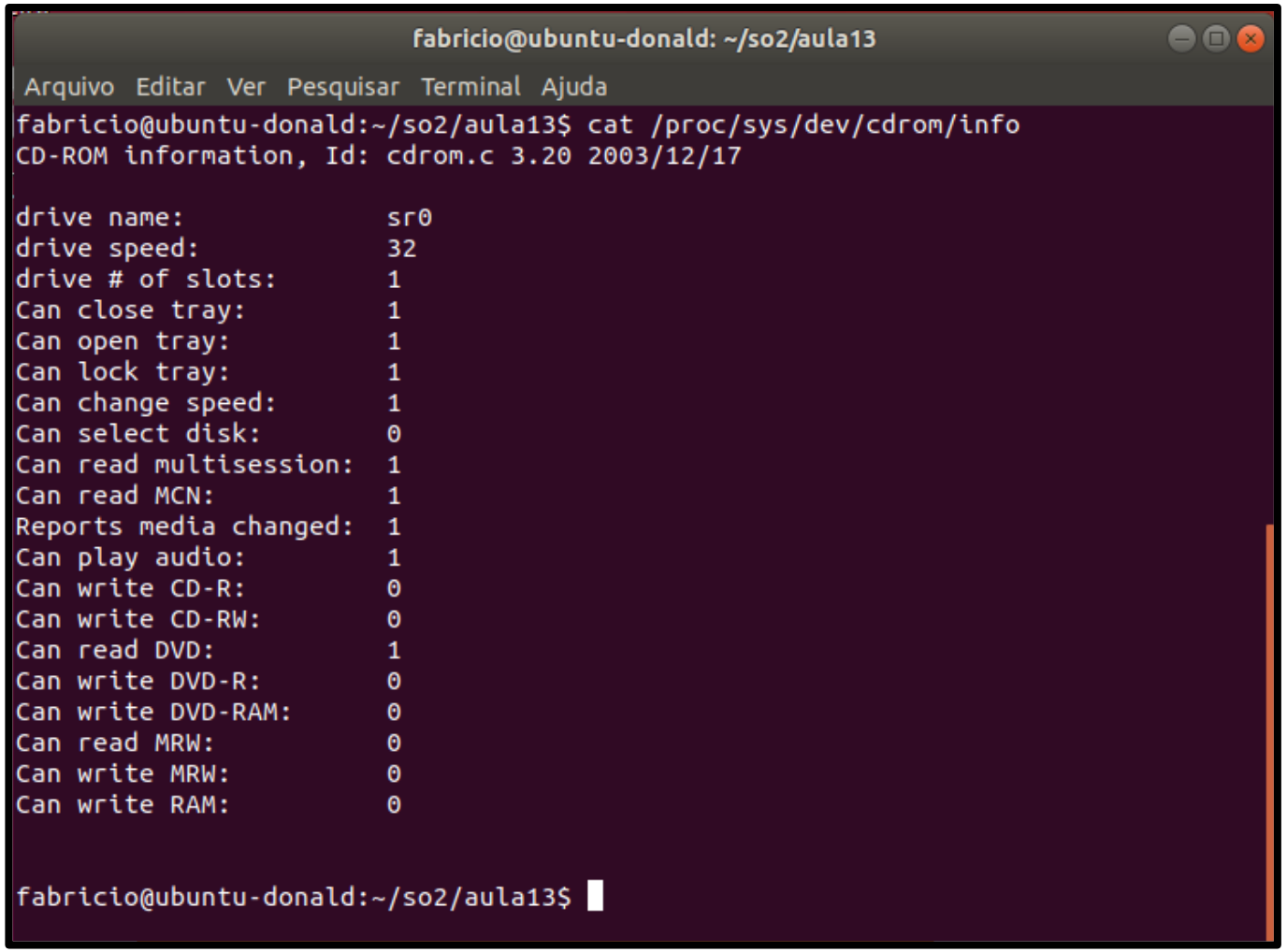
Veja em execução:
https://youtu.be/z5q7LVBq8Yg

```
fabricio@ubuntu-donald: ~/so2/aula13
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
fabricio@ubuntu-donald:~/so2/aula13$ cat /proc/partitions
major minor  #blocks  name

   7        0      98820 loop0
   7        1      99312 loop1
   7        2     144032 loop2
   7        3      56648 loop3
   7        4      56276 loop4
   7        5      63580 loop5
   7        6       9288 loop6
   7        7     261700 loop7
  11        0      59206 sr0
   8        0   20971520 sda
   8        1     248832 sda1
   8        2          1 sda2
   8        5   20719616 sda5
 253        0   16506880 dm-0
 253        1    4190208 dm-1
   7        8       2220 loop8
   7        9     261700 loop9
   7       10     144044 loop10
   7       11        276 loop11
   7       12       2496 loop12
   7       13       2220 loop13
   7       14        956 loop14
   7       15        276 loop15
fabricio@ubuntu-donald:~/so2/aula13$
```

# Drives e Partições

- A entrada **`/proc/sys/dev/cdrom/info`** mostra informações diversas sobre a capacidade de um drive de CD-ROM.

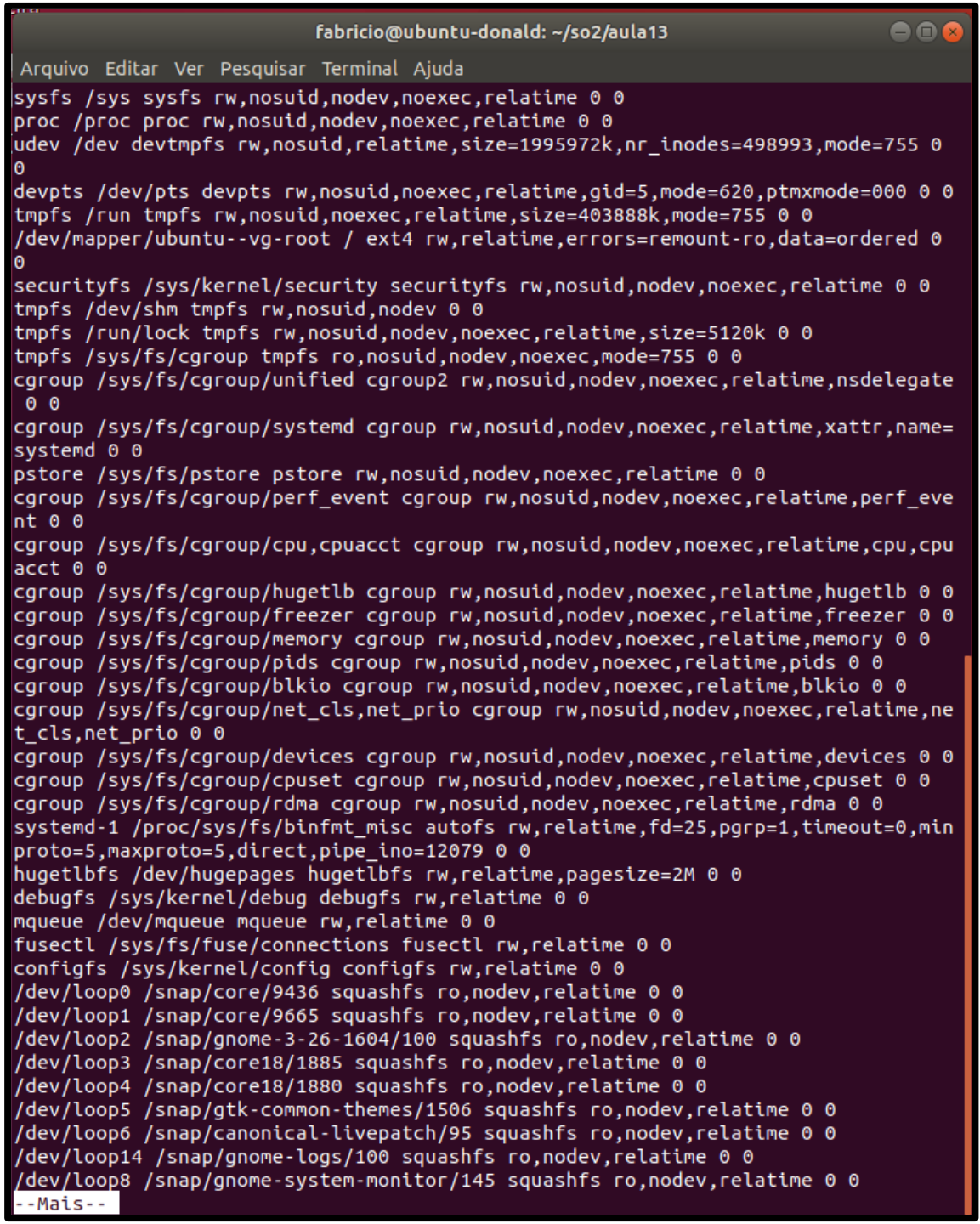
Veja em execução:
https://youtu.be/U-Lii1_R4Hg



# Montagens

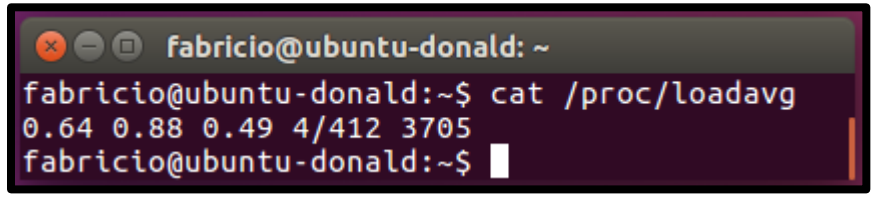
- O arquivo **`/proc/mounts`** fornece um resumo de sistemas de arquivos montados.
  - Cada linha corresponde a um único descritor de montagem e lista o dispositivo montado, o ponto de montagem, e outras informações.

Veja em execução:
https://youtu.be/WIvBr-Muxug



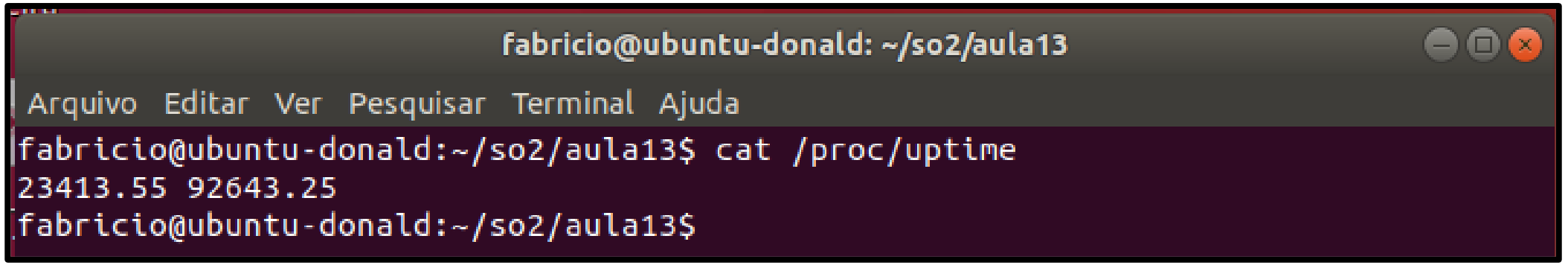
# Estatísticas do Sistema

- **`/proc/loadavg`** oferece informações sobre a carga do sistema.
  - Os primeiros 3 números representam a média da quantidade de tarefas ativas no sistema – processos executando – nos últimos 1, 5 e 15 minutos, respectivamente.
  - O quarto número é a quantidade atual de tarefas executáveis (agendadas para executar em vez de bloqueadas em uma chamada do sistema) e o número total de processos no sistema.
  - O quinto número é o ID de processo do último processo que executou.



# Estatísticas do Sistema

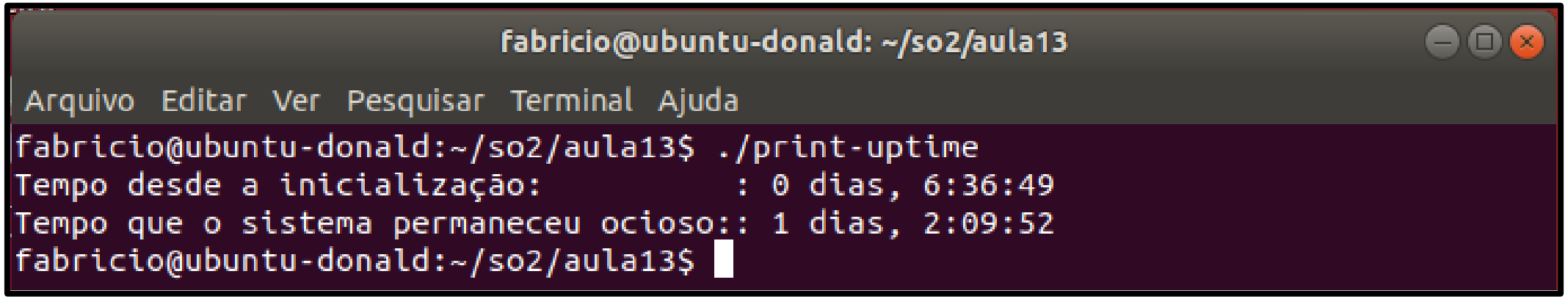
- **`/proc/uptime`** contém a quantidade de tempo desde que o sistema foi inicializado e a quantidade de tempo que o sistema permaneceu ocioso desde então.
  - Ambos são dados em segundos




Veja em execução:
https://youtu.be/soe7AepBytI

# Estatísticas do Sistema

- O programa a seguir extrai o tempo em que o sistema está ativo e o tempo que o sistema permaneceu ocioso e os mostra com unidades amigáveis.



## *print-uptime.c*

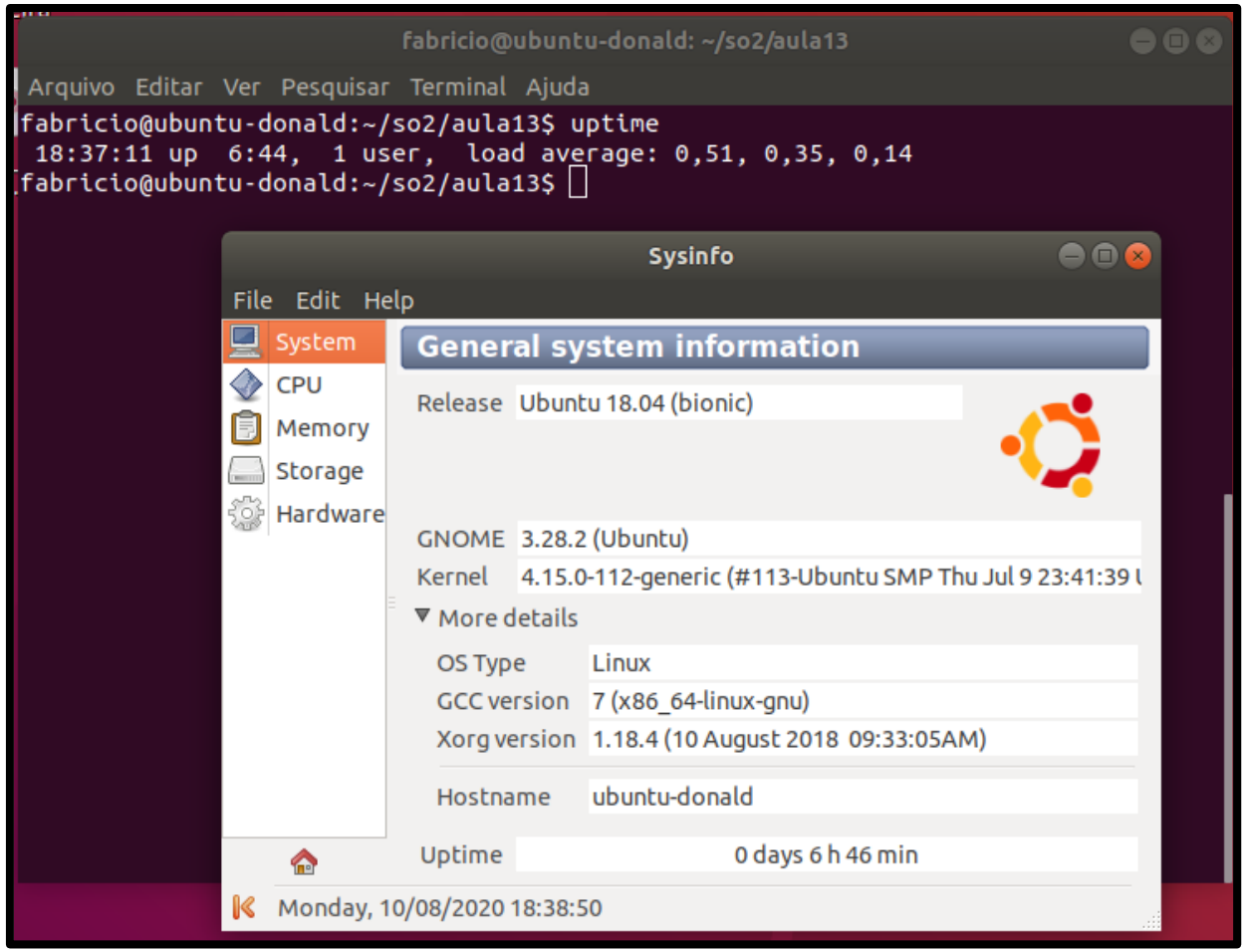
Imprime o Tempo desde a Inicialização do Sistema e o Tempo Ocioso

```c
#include <stdio.h>

/* Sumariza uma duração de tempo para a saída padrão.  TIME é a
   quantidade de tempo, em segundos, e LABEL é um rótulo descritivo.  */

void print_time (char* label, long time)
{
  /* Constantes de conversão.  */
  const long minute = 60;
  const long hour = minute * 60;
  const long day = hour * 24;
  /* Produz saída.  */
  printf ("%s: %ld dias, %ld:%02ld:%02ld\n", label, time / day,
          (time % day) / hour, (time % hour) / minute, time % minute);
}

int main ()
{
  FILE* fp;
  double uptime, idle_time;
  /* Lê o tempo que o sistema está ativo e o tempo ocioso acumulado de /proc/uptime.  */
  fp = fopen ("/proc/uptime", "r");
  fscanf (fp, "%lf %lf\n", &uptime, &idle_time);
  fclose (fp);
  /* Resume-o.  */
  print_time ("Tempo desde a inicialização:          ", (long) uptime);
  print_time ("Tempo que o sistema permaneceu ocioso:", (long) idle_time);
  return 0;
}
```

Veja em execução:
https://youtu.be/02K-2ioVtKw

# Estatísticas do Sistema

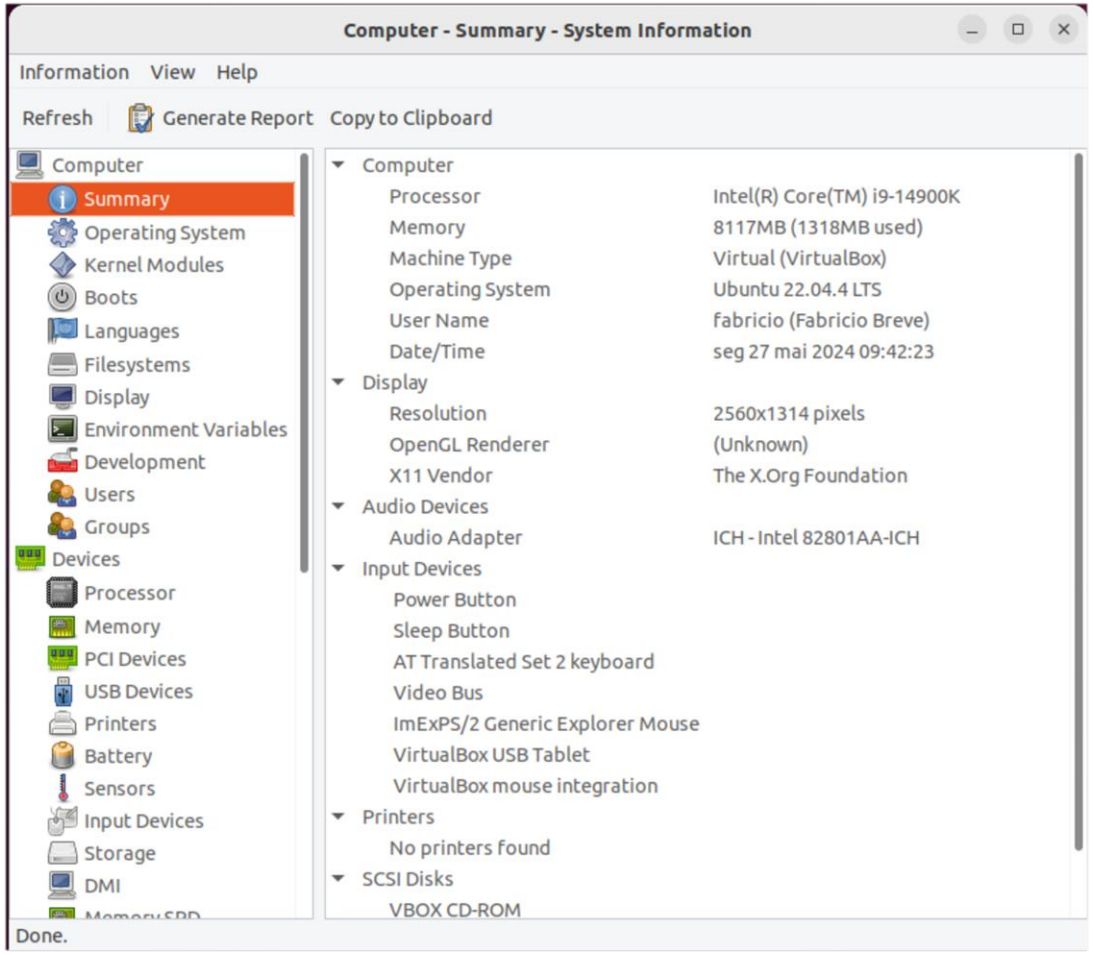
- O comando **`uptime`** e a chamada de sistema **`sysinfo`** também podem obter o tempo em que o sistema está ativo.
- O comando **`uptime`** também pode mostrar a carga média do sistema encontrada em **`/proc/loadavg`**.

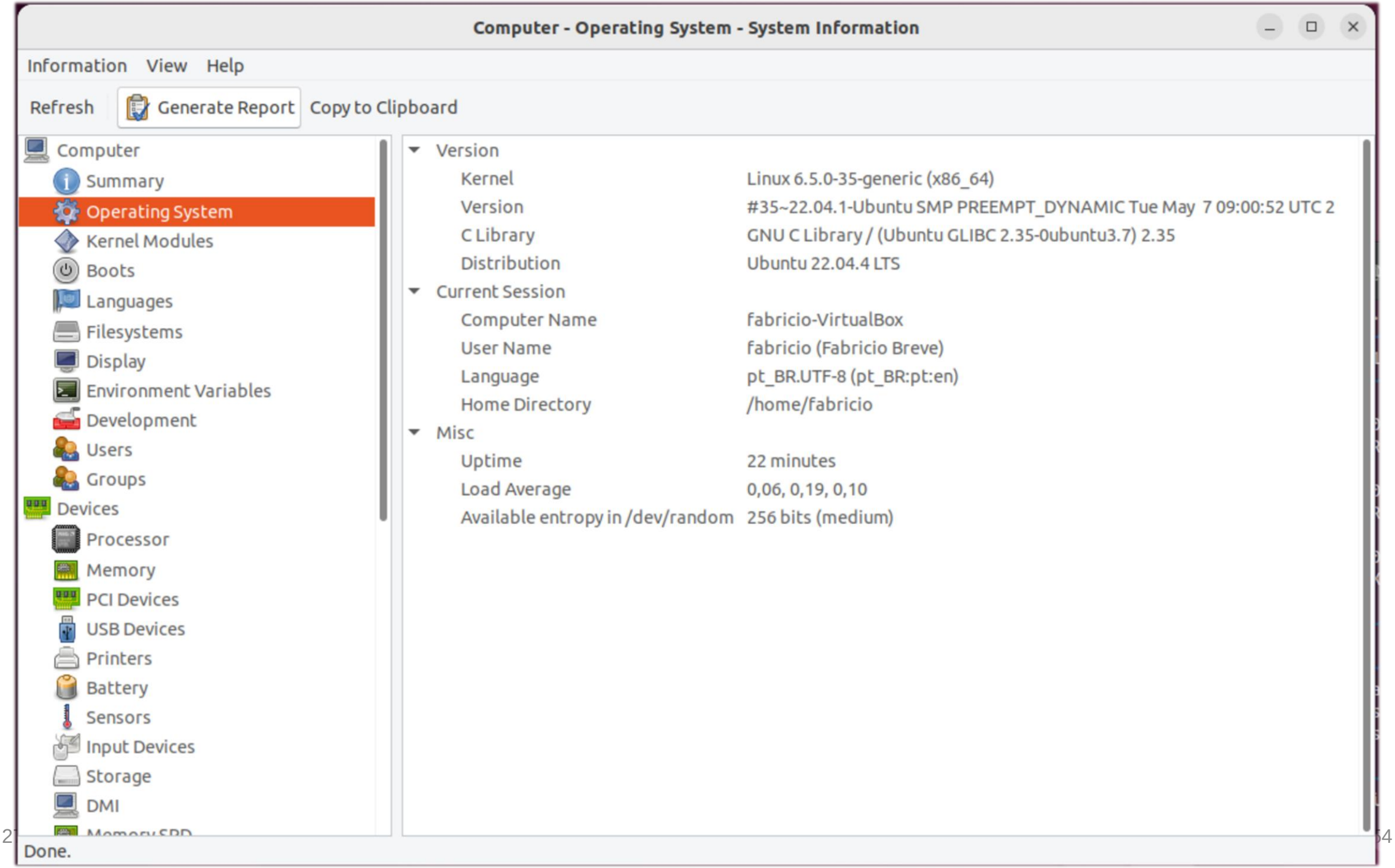



Veja em execução:
https://youtu.be/mHdC6P4LyV0

# Estatísticas do Sistema

- **`sysinfo`** não está mais presente nas versões atuais do Ubuntu.
- A partir do Ubuntu 20 é possível instalar o **`hardinfo`**.
  - `sudo apt install hardinfo`




Computer - Operating System - System Information
Information
View
Help
Refresh
Generate Report
Copy to Clipboard
Computer
Summary
Operating System
Kernel Modules
Boots
Languages
Filesystems
Display
Environment Variables
Development
Users
Groups
Devices
Processor
Memory
PCI Devices
USB Devices
Printers
Battery
Sensors
Input Devices
Storage
DMI
Version
Kernel
Linux 6.5.0-35-generic (x86_64)
Version
#35~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Tue May 7 09:00:52 UTC 2
C Library
GNU C Library / (Ubuntu GLIBC 2.35-0ubuntu3.7) 2.35
Distribution
Ubuntu 22.04.4 LTS
Current Session
Computer Name
fabricio-VirtualBox
User Name
fabricio (Fabricio Breve)
Language
pt_BR.UTF-8 (pt_BR:pt:en)
Home Directory
/home/fabricio
Misc
Uptime
22 minutes
Load Average
0,06, 0,19, 0,10
Available entropy in /dev/random
256 bits (medium)
Done.

# Referências Bibliográficas


1. NEMETH, Evi.; SNYDER, Garth; HEIN, Trent R.;. *Manual Completo do Linux: Guia do Administrador*. São Paulo: Pearson Prentice Hall, 2007.
2. DEITEL, H. M.; DEITEL, P. J.; CHOFFNES, D. R.; *Sistemas Operacionais: terceira edição*. São Paulo: Pearson Prentice Hall, 2005. Cap. 20.
3. MITCHELL, Mark; OLDHAM, Jeffrey; SAMUEL, Alex; *Advanced Linux Programming*. New Riders Publishing: 2001. Cap. 7.
4. TANENBAUM, Andrew S.; BOS, Herbert. *Sistemas Operacionais Modernos.* 4ed. São Paulo: Pearson Education do Brasil, 2016. Cap. 10.