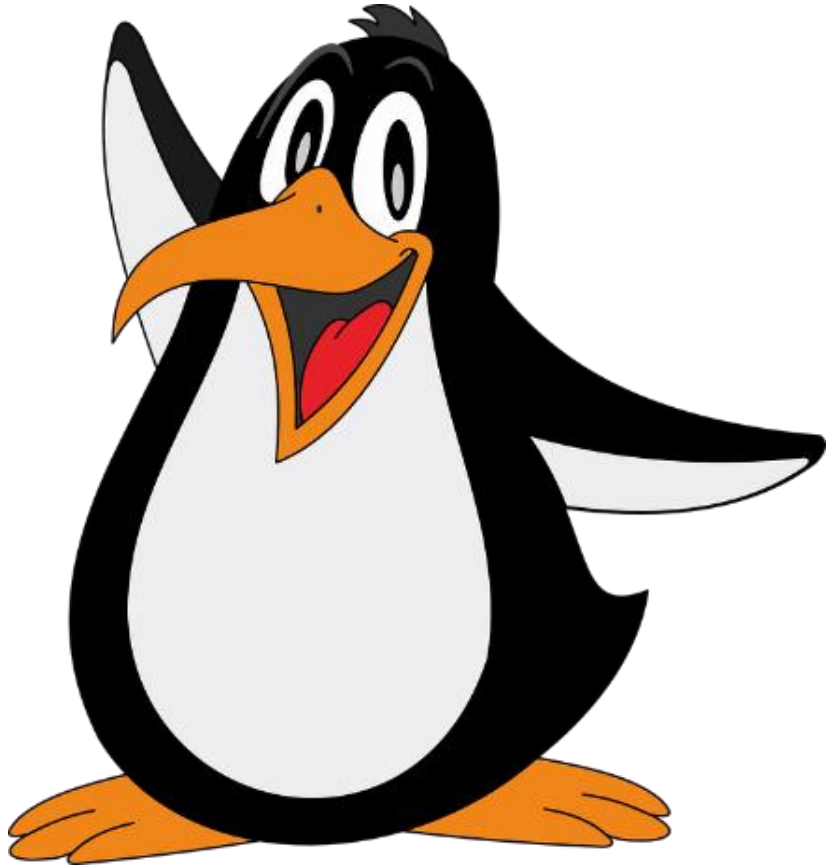
Sistemas Operacionais II

Dispositivos

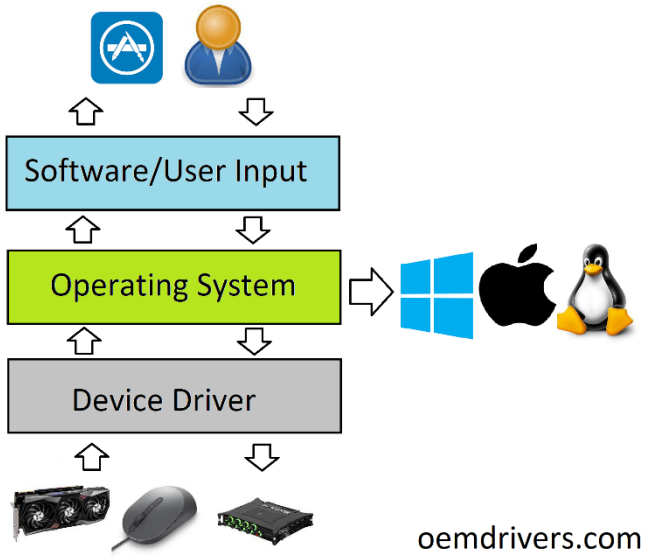
Fabricio Breve
fabricio.breve@unesp.br
https://www.fabriciobreve.com

Sumário

- Tipos de Dispositivos
- Números de Dispositivos
- Entradas de Dispositivos
- Dispositivos de *Hardware*
- Dispositivos Especiais
- PTYs
- ioctl

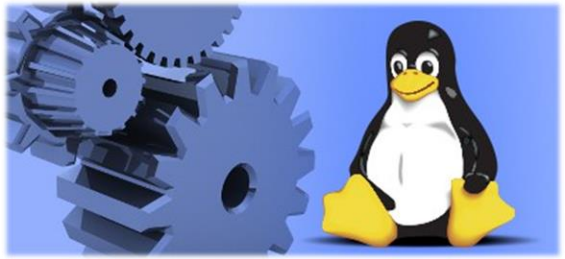


Dispositivos



oemdrivers.com

- O Linux e outros sistemas operacionais se comunicam com o *hardware* através de componentes de *softwares* modularizados chamados de *driver de dispositivo* (*device driver*).
- O *driver* de dispositivo esconde as peculiaridades dos protocolos de comunicação do *hardware* e permite ao sistema operacional interagir com o dispositivo através de uma interface padrão.

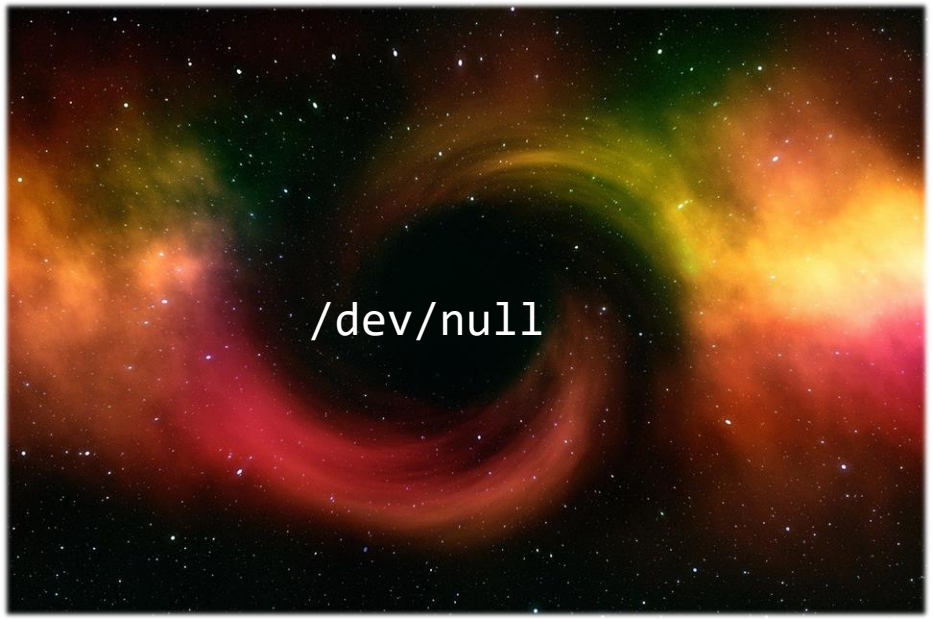


Dispositivos

- No Linux, *drivers* de dispositivos fazem parte do núcleo.
  - Podem ser *linkados* estaticamente no núcleo.
  - Podem ser carregados como módulos de núcleo.
- Os *drivers* não são acessíveis por processos de usuário.
  - Porém é possível interagir com eles através de objetos que aparecem como arquivos no sistema de arquivos.
    - Utiliza-se as mesmas funções de entrada/saída de baixo nível ou da biblioteca padrão do C.

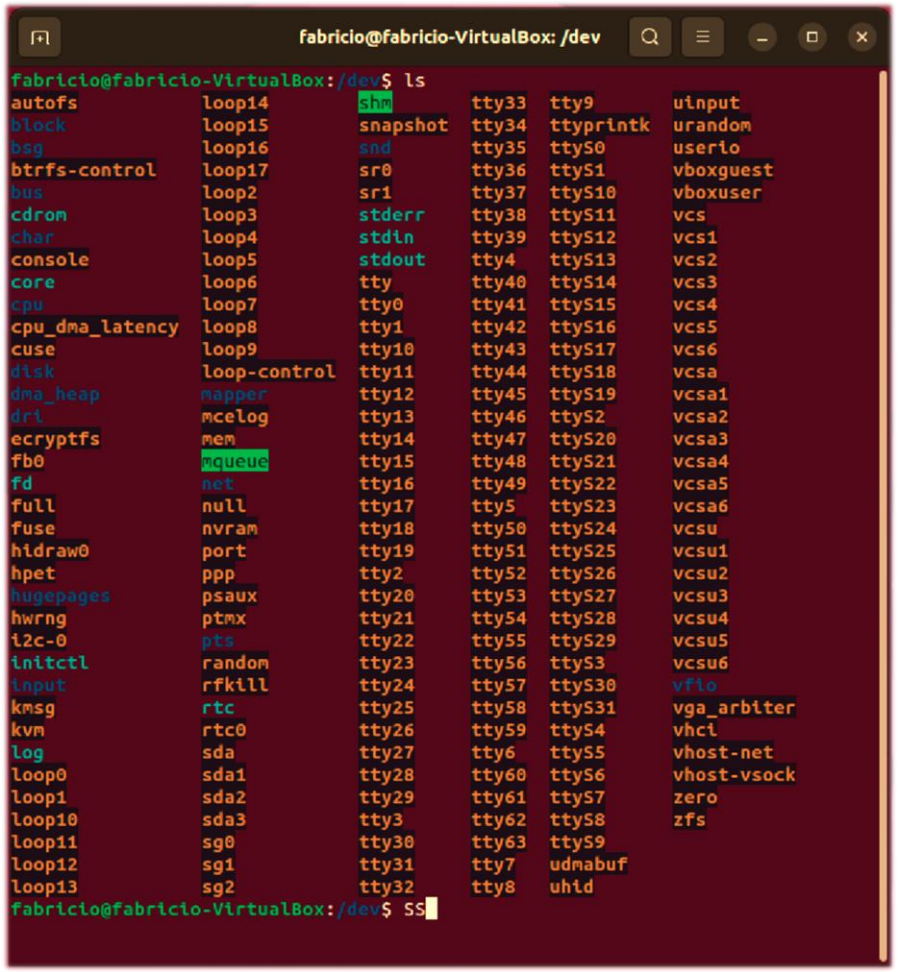
Dispositivos

- O Linux também fornece objetos (no sistema de arquivos) que se comunicam diretamente com o núcleo.
  - Não estão ligados a dispositivos de hardware.
  - Oferecem comportamento especializado que pode ser utilizado por aplicativos e programas do sistema.



Tipos de Dispositivos

- Arquivos de dispositivos não são arquivos comuns.
  - Não representam regiões de dados no sistema de arquivos baseado em disco.
  - Leituras e escritas são passadas ao *driver* de dispositivo correspondente.



Tipos de Dispositivos

- Existem dois tipos de arquivos de dispositivos:
  - *Dispositivos de caractere*
    - Representa um dispositivo de *hardware* que lê ou escreve um fluxo serial de *bytes* de dados.
    - Exemplo: portas seriais e paralelas, drives de fita, dispositivos de terminal, placas de som.
  - *Dispositivos de bloco*
    - Representa um dispositivo de *hardware* que lê ou escreve dados em blocos de tamanho fixo, oferecendo acesso aleatório aos dados armazenados nele.
    - Exemplo: discos rígidos.

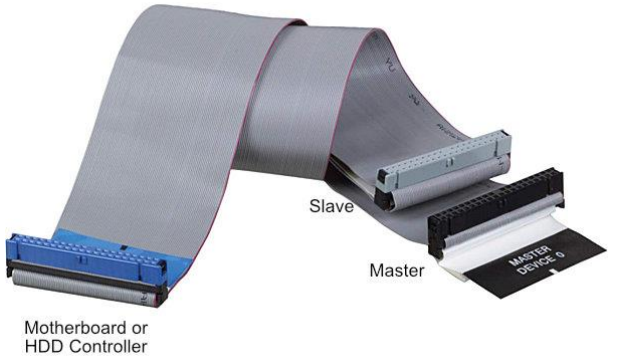
Tipos de Dispositivos

- Programas típicos nunca usam dispositivos de bloco.
  - Discos são representados como dispositivos de blocos, mas cada partição tipicamente contém um sistema de arquivos, que são montados no Linux.
  - Apenas o código de núcleo que implementa o sistema de arquivos precisa acessar o dispositivo de bloco diretamente.
  - Programas típicos acessam o dispositivo de bloco através de arquivos normais e diretórios.

Ext4

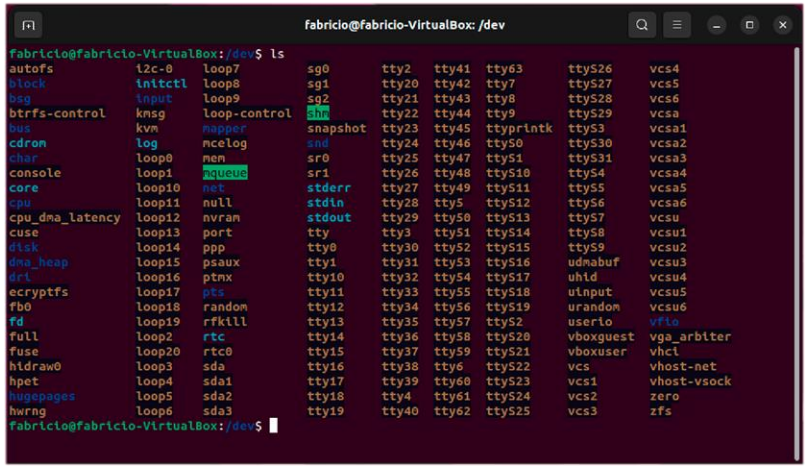
Números de Dispositivos

- O Linux identifica dispositivos usando dois números:
  - Número principal.
    - Especifica a qual *driver* o dispositivo corresponde.
  - Número secundário.
    - Distingue diferentes dispositivos controlados por um mesmo *driver*.
- Exemplo:
  - O número principal 3 corresponde à controladora IDE primária.
  - O dispositivo “mestre” tem número secundário 0 e o “escravo” tem número secundário 64.
  - Partições no dispositivo mestre são indicadas por 1, 2, 3, ...
  - Partições no dispositivo escravo são indicadas por 65, 66, 67, ...



Entradas de Dispositivos

- São semelhantes a arquivos normais:
  - Você pode movê-las com **`mv`**.
  - Você pode excluí-las com **`rm`**.
- Mas há algumas diferenças:
  - Se você tentar copiá-la com **`cp`**, estará pegando *bytes* do dispositivo (se este suportar leitura) e escrevendo-os no arquivo de destino.
  - Se você tentar sobrescreve-lo, estará escrevendo *bytes* no dispositivo correspondente.



Entradas de Dispositivos

- Para criar entradas de dispositivos no sistema de arquivos:
  - Utilize o comando **`mknod`**.
    - `man 1 mknod`
  - Utilize a chamada de função **`mknod`** em seus programas.
    - `man 2 mknod`
  - Ambos podem ser usados somente pelo *root*.
  - Criar uma entrada de dispositivo no sistema de arquivos não implica que o *driver* de dispositivo ou o dispositivo de *hardware* estão presentes ou disponíveis
    - A entrada de dispositivo é apenas um portal para a comunicação com o *driver*, se este existir.

Entradas de Dispositivos

- Para criar uma entrada de dispositivo com **mknod**:
  - Argumentos:
    1. Caminho para a entrada a ser criada no sistema de arquivos
    2. Use **b** para dispositivo de bloco ou **c** para dispositivo de caractere
    3. Número de dispositivo principal
    4. Número de dispositivo secundário
  - Exemplo:
    - **`mknod ./lp0 c 6 0`**

Veja em execução:
https://youtu.be/hkUxi3hXX6M

Entradas de Dispositivos

- Ao listar com o comando **`ls -l`** ou **`ls -o`**:
  - o primeiro caractere indica se o dispositivo é de bloco (**`b`**) ou de caractere (**`c`**).
  - No lugar onde seria o tamanho do arquivo, aparecem os números de dispositivo.

```
fabricio@fabricio-virtual-machine:~$ ls -l lp0
crw-r--r-- 1 root root 6, 0 Mai 25 14:25 lp0
```

  - Dentro de um programa, estas informações podem ser obtidas com **`stat`**
  - Para remover a entrada, use **`rm`**
    - Isto não remove o *driver* de dispositivo ou o dispositivo, apenas remove a entrada no sistema de arquivos.

O diretório **/dev**

- Por convenção, sistemas Linux incluem um diretório **/dev** contendo entradas de dispositivos para dispositivos que ele conhece.
  - Os nomes são padronizados e correspondem à números de dispositivos.
  - Exemplo:
    - Disco SATA e suas respectivas partições:

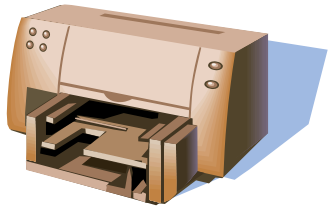
```
fabricio@fabricio-virtual-machine:~$ ls -l /dev/sd*
brw-rw---- 1 root disk 8, 0 Mai 25 14:17 /dev/sda
brw-rw---- 1 root disk 8, 1 Mai 25 14:17 /dev/sda1
brw-rw---- 1 root disk 8, 2 Mai 25 14:17 /dev/sda2
brw-rw---- 1 root disk 8, 5 Mai 25 14:17 /dev/sda5
```

O diretório **/dev**

- Na maioria dos casos, não se deve usar **mknod** para criar suas próprias entradas de dispositivos, mas sim usar as entradas existentes em **/dev**.
- Apenas administradores de sistemas ou desenvolvedores trabalhando com *hardware* especializado precisam criar entradas.
  - A maioria das distribuições Linux incluem facilidades para ajudar administradores de sistemas a criar entradas de dispositivo padrão com nomes corretos.

Acessando dispositivos através das entradas

- No caso de dispositivos de caracteres, ler ou escrever dos mesmos pode ser tão simples quanto ler ou escrever um arquivo.
  - Exemplo:
    - Se existir uma impressora paralela no sistema, podemos imprimir um documento usando:
      - **`cat documento.txt > /dev/lp0`**
    - É preciso permissão de escrita para isso, que em geral é dada somente ao *root* e ao *daemon* de impressão (lpd).
    - A impressora precisa ter capacidade de imprimir texto simples não formatado.



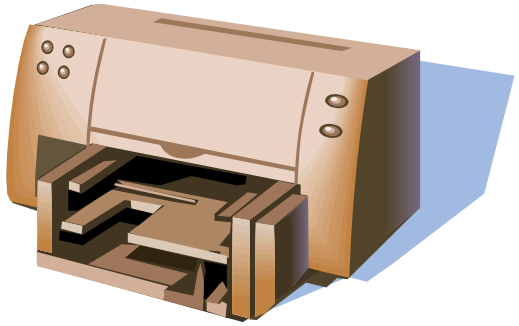
Acessando dispositivos através das entradas

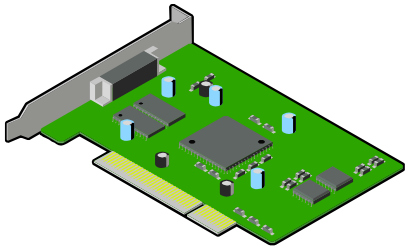
- Em um programa, enviar dados ao dispositivo também é simples.
  - Exemplo:
    - Usando funções de Entrada/Saída de baixo nível para enviar o conteúdo do *buffer* para **`/dev/lp0`**:

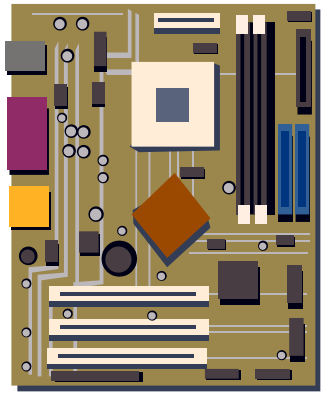
```
int fd = open("/dev/lp0", O_WRONLY);
write (fd, buffer, buffer_length);
close (fd);
```

Dispositivos de Hardware

- A seguir, são listados alguns dispositivos de bloco e dispositivos de caractere comuns.







Dispositivos de Bloco

Dispositivo	Nome	Principal	Secundário
Primeiro drive de disco flexível	/dev/fd0	2	0
Segundo drive de disco flexível	/dev/fd1	2	1
Primeira controladora IDE, dispositivo mestre	/dev/hda	3	0
Primeira controladora IDE, dispositivo mestre, primeira partição	/dev/hda1	3	1
Primeira controladora IDE, dispositivo escravo	/dev/hdb	3	64
Primeira controladora IDE, dispositivo escravo, segunda partição	/dev/hdb1	3	65
Segunda controladora IDE, dispositivo mestre	/dev/hdc	22	0
Segunda controladora IDE, dispositivo escravo	/dev/hdd	22	64
Primeiro drive SCSI ou SATA	/dev/sda	8	0
Primeiro drive SCSI ou SATA, primeira partição	/dev/sda1	8	1
Segundo drive SCSI ou SATA	/dev/sdb	8	16
Segundo drive SCSI ou SATA, primeira partição	/dev/sdb1	8	17
Primeiro drive de CD-ROM SCSI ou SATA	/dev/sdc0	11	0
Segundo drive de CD-ROM SCSI ou SATA	/dev/sdc1	11	1

Dispositivos de Caractere

Dispositivo	Nome	Principal	Secundário
Primeira porta paralela	`/dev/lp0` ou `/dev/par0`	6	0
Segunda porta paralela	`/dev/lp1` ou `/dev/par1`	6	1
Primeira porta serial	`/dev/ttyS0`	4	64
Segunda porta serial	`/dev/ttyS1`	4	65
Drive de fita IDE	`/dev/ht0`	37	0
Primeiro drive de fita SCSI	`/dev/st0`	9	0
Segundo drive de fita SCSI	`/dev/st1`	9	1
Console do sistema	`/dev/console`	5	1
Primeiro terminal virtual	`/dev/tty1`	4	1
Segundo terminal virtual	`/dev/tty2`	4	2
Dispositivo de terminal do processo atual	`/dev/tty`	5	0
Placa de som	`/dev/audio`	14	4

Dispositivos de *Hardware*

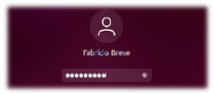
- É comum haver mais de uma entrada para o mesmo *hardware*.
  - Em geral oferecem diferentes semânticas
    - Exemplos:
      - Para fitas IDE:
        - » **`/dev/ht0`** rebobina a fita quando o descritor é fechado.
        - » **`/dev/nht0`** não rebobina a fita quando o descritor é fechado.

Dispositivos de *Hardware*

- Em algumas ocasiões, dados podem ser escritos diretamente para dispositivos de caracteres.
  - Exemplos:

- Programa terminal acessando *modem* diretamente através de porta serial, o que for digitado no terminal é passado diretamente ao computador remoto.

- Programa de *backup* em fita escrevendo diretamente para o dispositivo de fita, por implementar seu próprio esquema de compressão e checagem de erro.

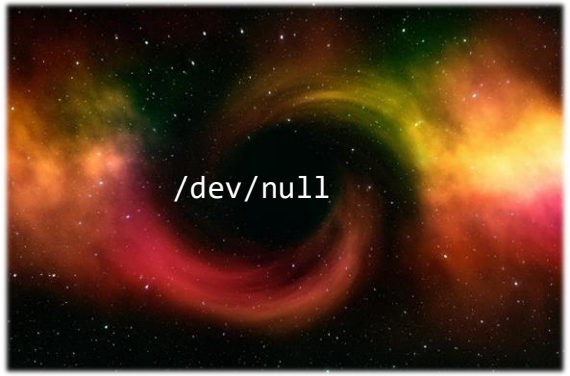


- Um programa que precisa ler uma senha pode acessar diretamente o terminal para evitar *scripts* que usem redirecionamento de entrada e saída padrão por questões de segurança.

Dispositivos Especiais

- O Linux também fornece dispositivos de caractere que não correspondem a dispositivos de *hardware*.
  - Todos usam o número de dispositivo principal 1, que está associado com o dispositivo de memória do núcleo, em vez de um *driver* de dispositivo.

/dev/null



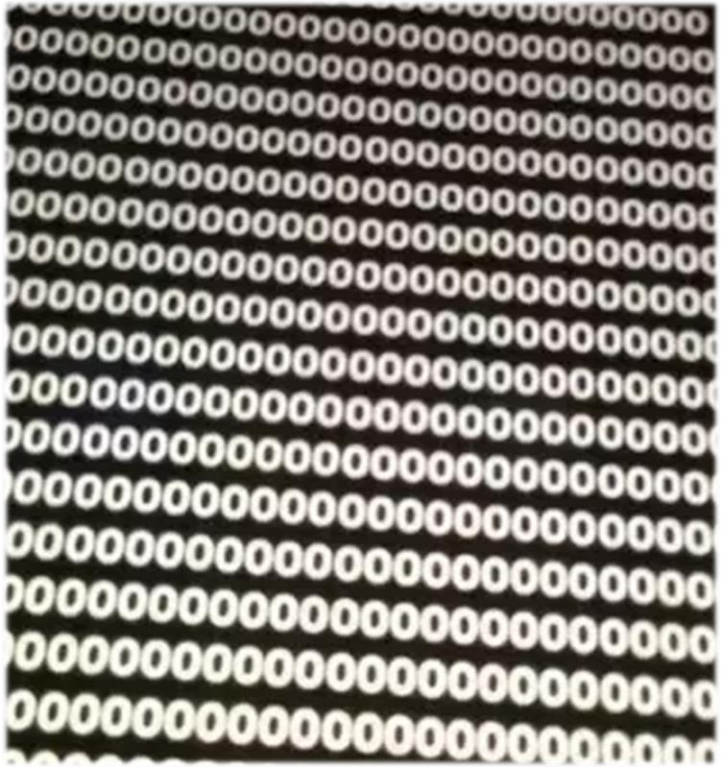
- Serve para dois propósitos:
  - O Linux descarta toda saída escrita em **/dev/null**
    - Útil para descartar uma saída quando ela é indesejada.
    - Exemplo:
      - **comando-com-saida > /dev/null**
  - Ler de **/dev/null** sempre resulta em um fim-de-arquivo.
    - Útil para testar o comportamento de um programa ao encontrar o fim de arquivo
    - Exemplo:
      - **cp /dev/null arquivo-vazio**
      - **ls -l arquivo-vazio**

```
fabricio@ubuntu-donald:~/so2$ cp /dev/null arquivo-vazio
fabricio@ubuntu-donald:~/so2$ ls -la arquivo-vazio
-rw-r--r-- 1 fabricio fabricio 0 ago  3 16:02 arquivo-vazio
```

Veja em execução:
https://youtu.be/X-E3sTFGTB8

/dev/zero

- Se comporta como um arquivo infinitamente longo composto apenas de *bytes* **0**.
- Bastante usado para alocar memória.



/dev/full

- Se comporta como se fosse um arquivo no sistema de arquivos que não tem mais espaço.
- Uma escrita a **/dev/full** falha e ajusta **errno** para **ENOSPC**, que indica que o dispositivo está cheio.
- Exemplo:
  - **cp /etc/fstab /dev/full**

```
fabricio@ubuntu-donald:~/so2$ cp /etc/fstab /dev/full
cp: erro de escrita de '/dev/full': Não há espaço disponível no dispositivo
```

- Útil para testar como um programa se comportará se ficar sem espaço em disco enquanto escreve para um arquivo.

Veja em execução:
https://youtu.be/esti9rvDMzo

Dispositivos de Números Aleatórios

- Os dispositivos especiais **`/dev/random`** e **`/dev/urandom`** fornecem acesso ao gerador de números aleatórios embutido no núcleo do Linux.
  - Funções da biblioteca do C, como **`rand`**, geram números pseudoaleatórios, que podem ser reproduzidos se a mesma semente for usada.
    - Comportamento inevitável, pois computadores são intrinsicamente determinísticos e previsíveis.
    - Podem implicar em problemas de segurança se forem usados em algoritmos de criptografia.
  - **`/dev/random`** e **`/dev/urandom`** usam **você** como fonte externa de aleatoriedade
    - Medem e usam o tempo entre suas ações de entrada, como pressionamento de teclas e movimento do mouse, para gerar números aleatórios de alta qualidade.

Dispositivos de Números Aleatórios

- Você pode acessar os números aleatórios gerados lendo **`/dev/random`** ou **`/dev/urandom`**
- **`/dev/random`**
  - **Originalmente**: se você tentar ler uma grande quantidade de números, mas não gerar entradas, o Linux bloqueia a operação de leitura quando o estoque de "aleatoriedade" acaba, e só desbloqueia quando mais "aleatoriedade" for gerada.
  - **A partir do *kernel* 5.6 (2020)**: só bloqueia até que o gerador de números aleatórios fortes esteja pronto, e depois se comporta como o `/dev/urandom`.
- **`/dev/urandom`**
  - Não bloqueia mesmo que não haja "aleatoriedade".

Dispositivos de Números Aleatórios

- Teste **`/dev/random`** e **`/dev/urandom`** usando:
  - **`od -t x1 /dev/urandom`**
  - **`od -t x1 /dev/random`**

Veja em execução:
https://youtu.be/oytK_Hw0d2o

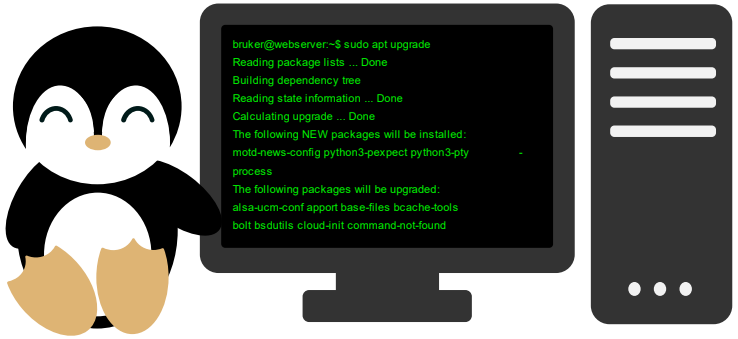
```
fabricio@ubuntu-donald:~/so2$ od -t x1 /dev/random
0000000 82 8e 13 65 8a bf ad aa 62 12 2d b1 e9 2b e6 8c
0000020 6d f0 2d d3 cd 4d 82 52 db a7 ac 6d 83 45 82 1d
0000040 d5 48 49 bf 67 c1 c9 97 c1 01 fa 56 d5 3e 9c 8b
0000060 8e cf 85 0d bd 7c c5 68 50 82 b4 61 9e 3f 2a 1b
0000100 22 9c 14 cd e1 a4 3a 7a 51 01 8b 05 74 f7 5e bd
0000120 1d 08 19 db 83 e9 3b f0 a9 5c e0 67 b2 34 67 6e
0000140 67 af 41 81 5a 60 86 9e f4 ec 0c 71 a9 36 cc 47
0000160 d4 3e 2f 85 44 71 9e 76 98 64 93 37 23 13 c6 08
0000200 a3 8f c8 96 98 e1 df 44 ff 3f 92 07 13 e0 4e d6
0000220 2e b7 9c d4 48 13 e7 59 ee 98 88 93 fb cc 98 b2
0000240 77 3a d4 fa 0f 6b fd ba bd 73 ac e0 e9 af 4c 6c
0000260 88 98 fe 0c 90 f0 9f 5d 15 f1 e1 ca f1 98 f4 97
0000300 90 ca 2c 9e ea 05 cb 57 1b e1 76 f6 c0 23 a2 32
```

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

/* Retorna um inteiro aleatório entre MIN e MAX, inclusive.  Obtenha aleatoriedade de /dev/random.  */

int random_number (int min, int max)
{
  /* Armazene o descritor de arquivo aberto para /dev/random em uma variável estática.  Desta forma, não
precisamos abrir o arquivo cada vez que esta função é chamada.  */
  static int dev_random_fd = -1;

  char* next_random_byte;
  int bytes_to_read;
  unsigned random_value;

  /* Certifique-se que MAX é maior que MIN.  */
  assert (max > min);

  /* Se esta é a primeira vez que esta função é chamada, abra um descritor de arquivo para /dev/random.  */
  if (dev_random_fd == -1) {
    dev_random_fd = open ("/dev/random", O_RDONLY);
    assert (dev_random_fd != -1);
  }

  /* Leia o número de bytes aleatórios suficientes para encher uma variável inteiro.  */
  next_random_byte = (char*) &random_value;
  bytes_to_read = sizeof (random_value);
  /* Loop até até ler o número de bytes suficiente.  Como /dev/random é preenchido a partir de ações do
usuário, a leitura pode bloquear e somente retornar um único byte de cada vez.  */
  do {
    int bytes_read;
    bytes_read = read (dev_random_fd, next_random_byte, bytes_to_read);
    bytes_to_read -= bytes_read;
    next_random_byte += bytes_read;
  } while (bytes_to_read > 0);

  /* Computar um número aleatório na faixa correta.  */
  return min + (random_value % (max - min + 1));
}
```

random_number.c

Função para Gerar um Número Aleatório usando /dev/random

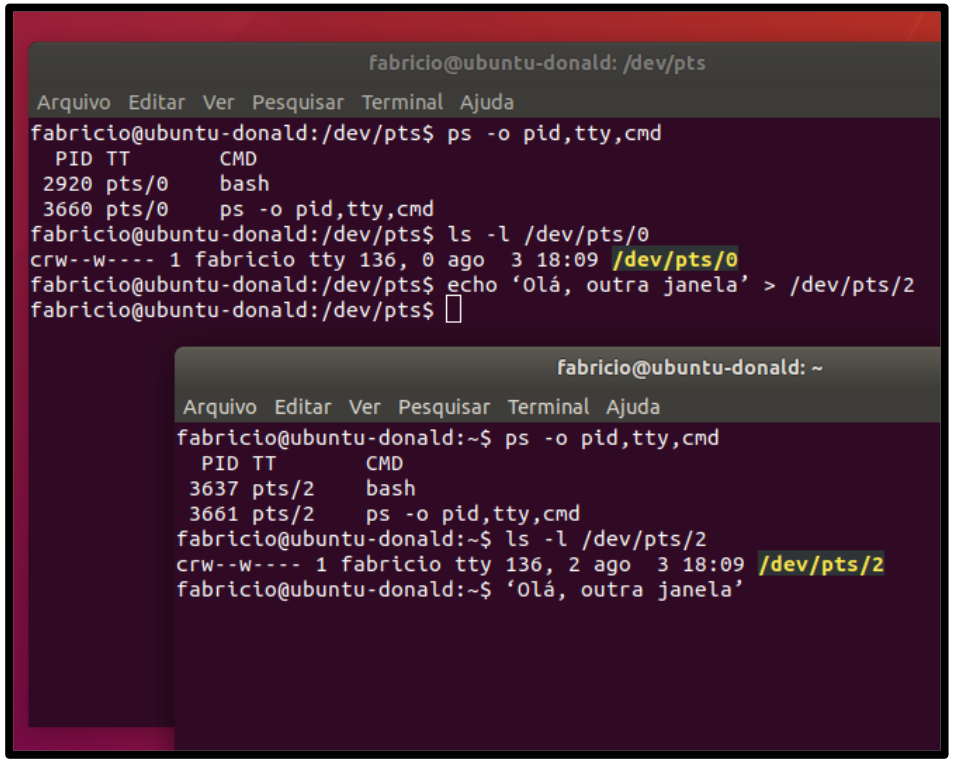
PTYs



- PTYs são pseudo-terminais
  - Para cada janela de terminal que você abre, o Linux cria uma entrada em **`/dev/pts`**
    - Este é um diretório especial criado pelo núcleo dinamicamente.
    - Seu conteúdo varia e reflete o estado atual do sistema sendo executado.
    - PTYs são numerados, e cada PTY corresponde a uma entrada em **`/dev/pts`**
    - Verifique o terminal associado com um processo usando **`ps -o pid,tty,cmd`**

Demonstração de PTY

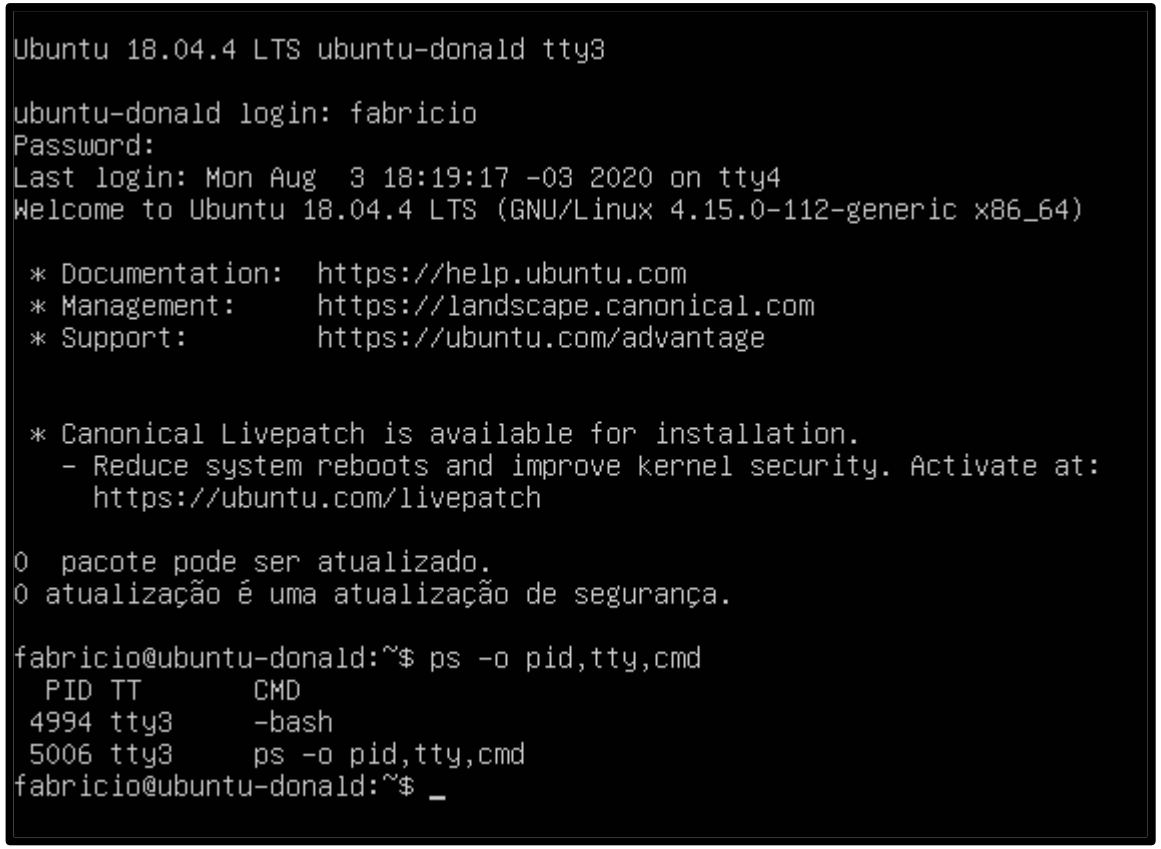
- Abra dois terminais.
  - Em um terminal, determine o PTY associado com:
    - **`ps -o pid,tty,cmd`**
  - Verifique a entrada correspondente com:
    - **`ls -l /dev/pts/2`**
      (troque **2** pelo número de seu PTY)
  - No outro terminal digite:
    - **`echo 'Olá, outra janela' > /dev/pts/2`**



Veja em execução:
https://youtu.be/CbCc0aMgSTU

TTY

- Pressione **Ctrl+Alt+F3** para ir a um terminal virtual em modo de texto.
  - Em vez de **F3** pode ser **F1**, **F2**, **F4**, **F5**, etc. dependendo da distribuição e versão do Linux.
- Digite **`ps -o pid,tty,cmd`** para verificar que ele está executando em um dispositivo de terminal comum em vez de um PTY.



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

ioctl

- Chamada de sistema para controlar dispositivos de *hardware*.
  - O primeiro argumento é um descritor de arquivos, que deve estar aberto com o dispositivo a ser controlado.
  - O segundo argumento é um código de requisição que indica a operação a ser realizada.
  - Podem haver argumentos adicionais dependendo da requisição.
- Muitos dos códigos de requisição disponíveis estão listados na página de manual **`ioctl_list`**
  - Usar **`ioctl`** requer um conhecimento detalhado do *driver* de dispositivo correspondente ao *hardware* a ser controlado

ioctl

- O exemplo a seguir mostra um pequeno programa que ejeta o disco em um *drive* de CD-ROM.
- Ele recebe como parâmetro o nome de dispositivo correspondente ao *drive* de CD-ROM.
  - **`/dev/cdrom`**

```c
#include <fcntl.h>
#include <linux/cdrom.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main (int argc, char* argv[])
{
  /* Abre um descritor de arquivo com o dispositivo especificado na linha de comando.  */
  int fd = open (argv[1], O_RDONLY | O_NONBLOCK);
  /* Ejeta o CD-ROM.  */
  ioctl (fd, CDROMEJECT);
  /* Fecha o descritor de arquivo.  */
  close (fd);

  return 0;
}
```

Acrescente a flag O_NONBLOCK

cdrom-eject.c

Ejeta um CD-ROM

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. 6.
4. TANENBAUM, Andrew S.; BOS, Herbert. *Sistemas Operacionais Modernos.* 4ed. São Paulo: Pearson Education do Brasil, 2016. Cap. 10.