C2110 *UNIX and programming*

Lesson 3 / Module 1

PS / 2020 Distance form of teaching: Rev1

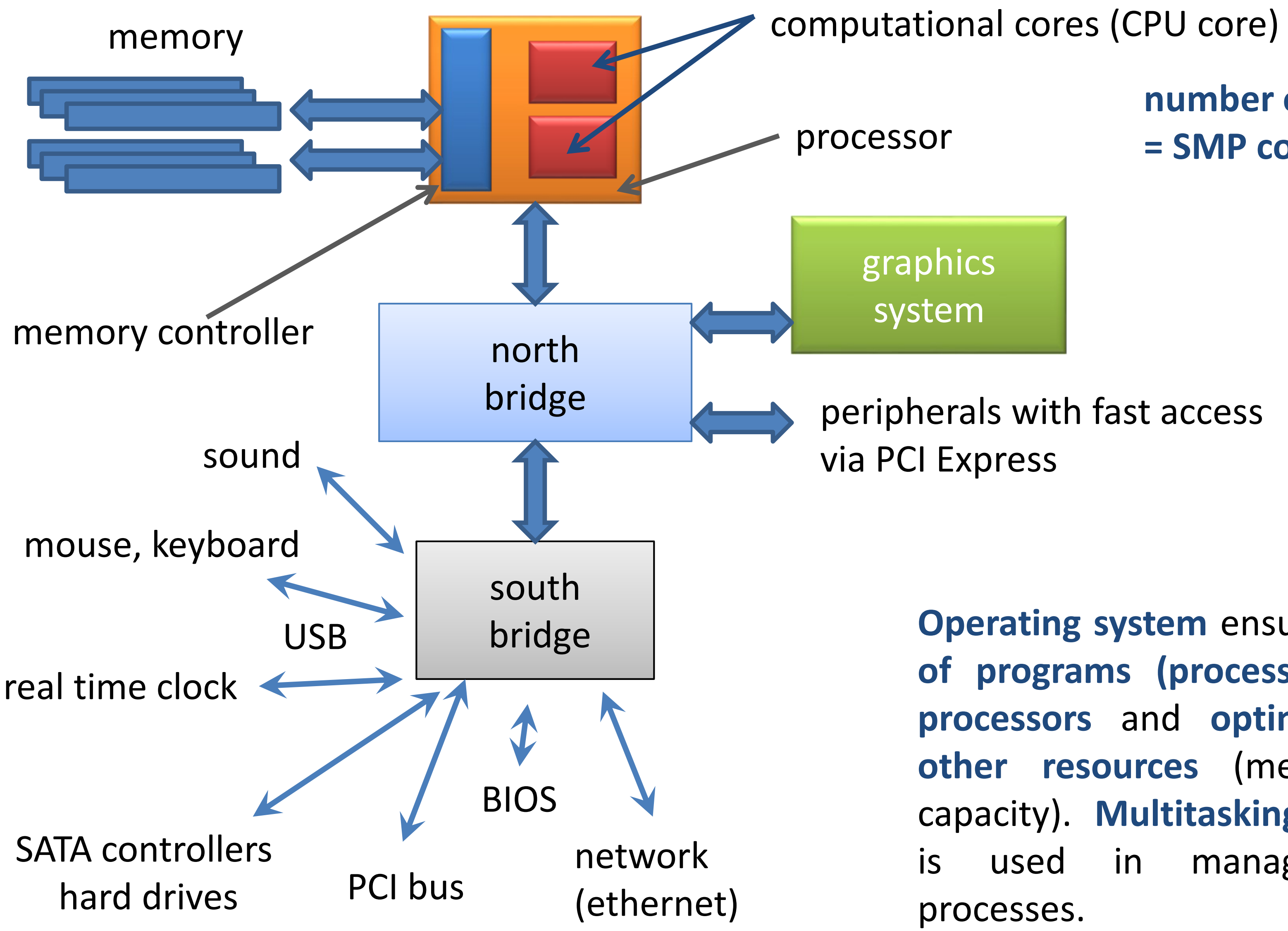
Petr Kulhanek

kulhanek@chemi.muni.cz

National Center for Biomolecular Research, Faculty of Science
Masaryk University, Kamenice 5, CZ-62500 Brno

Processes

Internal Scheme of a Computer



number of CPU cores > 1 = SMP computer

Operating system ensures **running of programs (processes) on the processors** and **optimal use of other resources** (memory, disk capacity). **Multitasking** technique is used in management of processes.

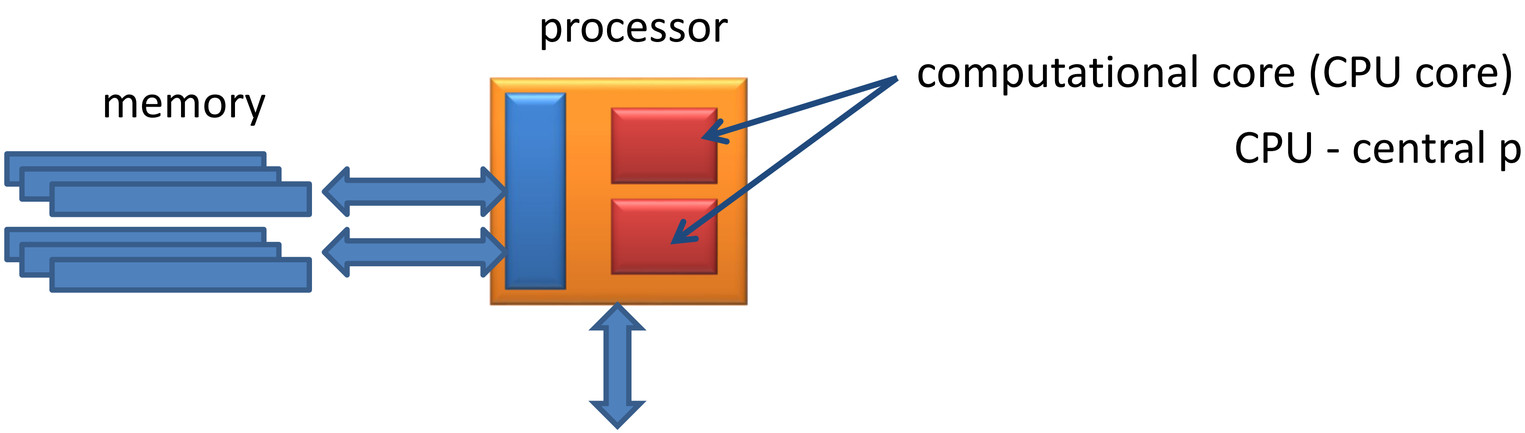
Process and Multitasking

Process in informatics is the name for a **running computer program**. The process is **located in the computer operating memory** as **the sequence of machine instructions executed by the processor**. It contains not only the code of the executed program, but also dynamically changing data that the process processes. One program can run on the computer as multiple processes with different data (for example, a web browser running multiple times that displays different pages). **Management of processes is performed by the operating system**, which ensures their separate running, allocates them computer system resources and allows the user to manage processes (start, stop, etc.).

Multitasking in computer science indicates **the ability of the operating system to perform several processes simultaneously** (at least seemingly). The core of the operating system very quickly switches processes running on the processor or processors (so-called change of context), so that the computer user has the impression that they are running at the same time.

edited from wikipedia.org

SMP - Symmetric Multiprocessing



CPU - central processing unit

In the past, the performance of processors increased, in addition to better architecture, also by speed of instruction processing (processor frequency), which nowadays encounters physical limitations of the technology used (reliability, heat loss, ...). Another direction was the introduction of more computing cores (approximately since 2005 for x86 architecture) on one physical chip. **Today's computers now commonly have more than one processor.**

Symmetric multiprocessing (SMP) in infomatics is a type of **multiprocessor systems** in which all processors of the computer are equivalent. Increasing the number of processors that share the same operating memory on the computer leads **to increase of computer performance**, although not in a linear way, because part of the power is consumed for overhead (locking data structures, controlling processors and their communication with each other).

edited from wikipedia.org

Overview of Running Processes

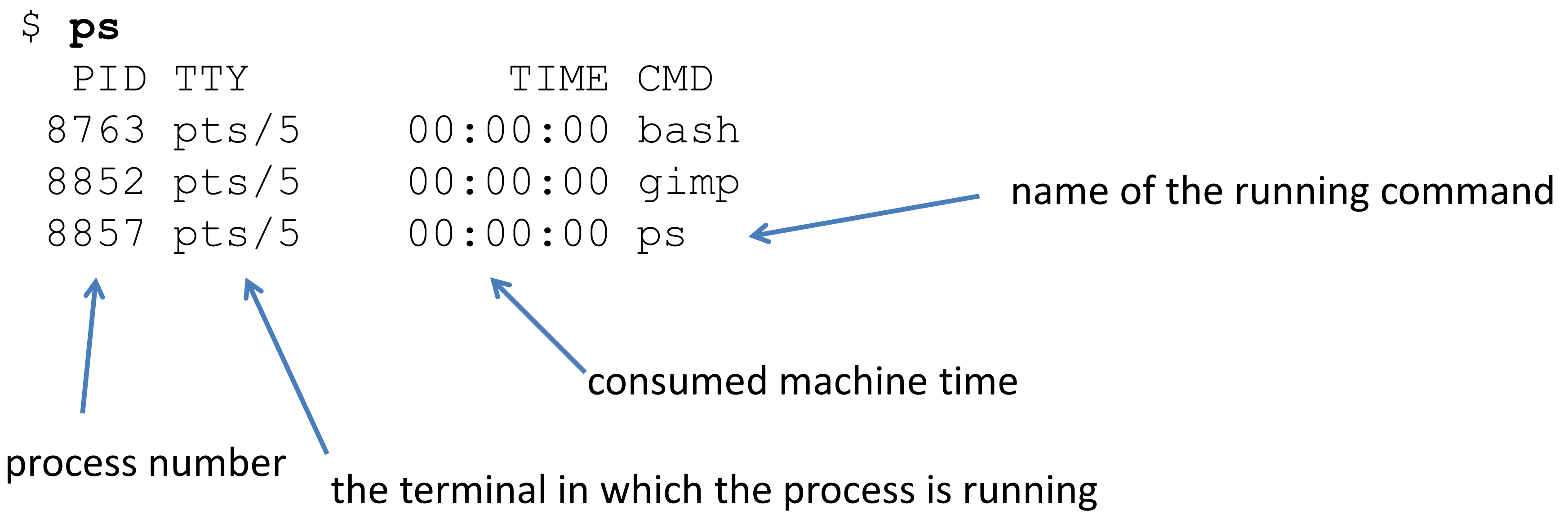
Processes can be listed with the following commands:

ps lists the processes running in the given terminal or according to the specified specifications

`(ps -u user_name)`

top continuously displays processes sorted by their CPU load (end with q key)

pstree a list showing the hierarchy of processes



```
$ ps
  PID TTY          TIME CMD
 8763 pts/5    00:00:00 bash
 8852 pts/5    00:00:00 gimp
 8857 pts/5    00:00:00 ps
```

name of the running command

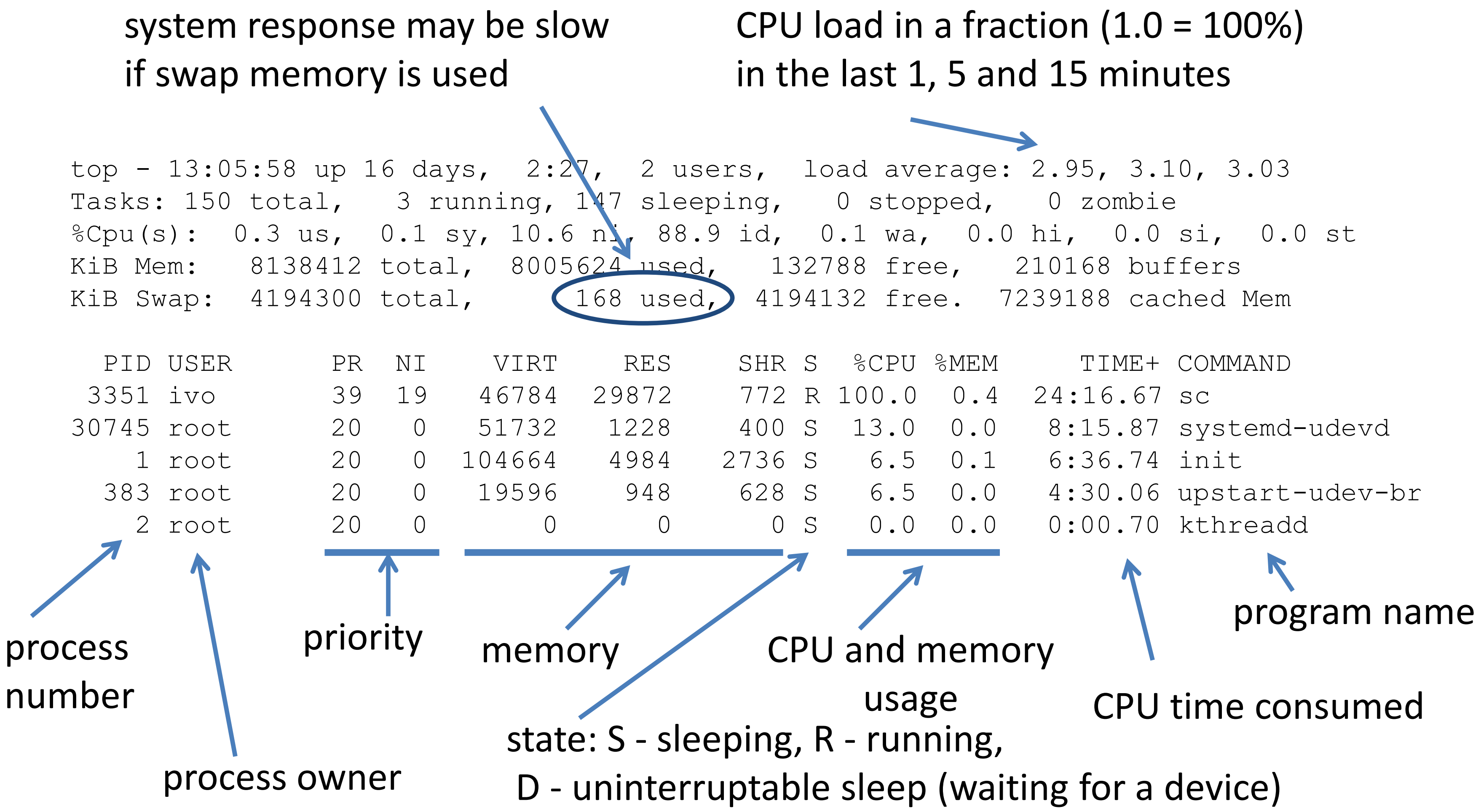
consumed machine time

process number

the terminal in which the process is running

Overview of Running Processes - top

By command **top**, it is possible to monitor running processes at regular intervals. The command is terminated by the key **q** (quit).



system response may be slow
if swap memory is used

CPU load in a fraction (1.0 = 100%)
in the last 1, 5 and 15 minutes

```
top - 13:05:58 up 16 days,  2:27,  2 users,  load average: 2.95, 3.10, 3.03
Tasks: 150 total,   3 running, 147 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.3 us,  0.1 sy, 10.6 ni, 88.9 id,  0.1 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   8138412 total,  8005624 used,   132788 free,   210168 buffers
KiB Swap:  4194300 total,      168 used,  4194132 free.  7239188 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 3351 ivo       39  19   46784  29872    772 R 100.0  0.4  24:16.67 sc
30745 root      20   0   51732   1228    400 S  13.0  0.0   8:15.87 systemd-udevd
    1 root      20   0  104664   4984   2736 S   6.5  0.1   6:36.74 init
  383 root      20   0   19596    948    628 S   6.5  0.0   4:30.06 upstart-udev-br
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.70 kthreadd
```

process number

process owner

priority

memory

state: S - sleeping, R - running,
D - uninterruptable sleep (waiting for a device)

CPU and memory usage

CPU time consumed

program name

Run Commands and Applications

In order to run a command, shell needs to **know the way** to the file that contains a binary program or script.

1. The path to the command is first searched in the table with already used commands:

```
$ hash
     hits     command
     1      /bin/rm
     3      /bin/ls
```

The table can be deleted with the command:

```
$ hash -r
```

2. If the command is not found, it is searched in the directories specified in the system variable **PATH,** which are separated by a colon.

```
$ echo $PATH
```

directory search order

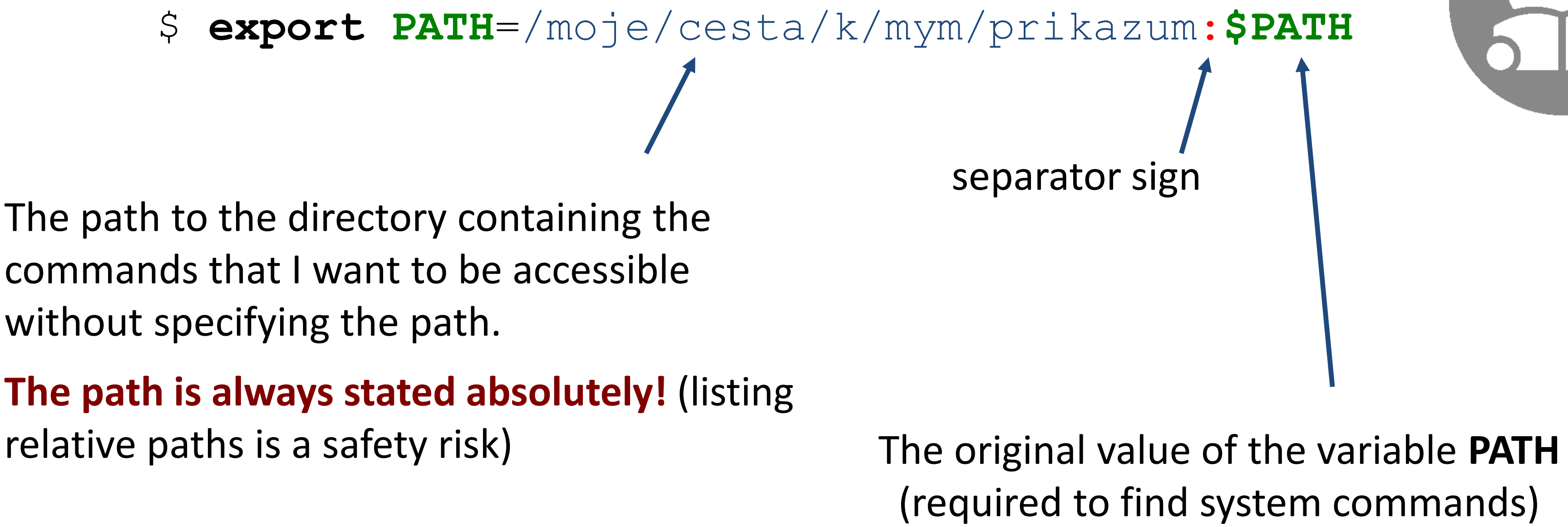
```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:
/bin:/usr/games:/usr/local/games:/snap/bin:/usr/bin
```

3. If the command is not found, an error is indicated. Otherwise, the command is run and the path is stored in a table.

```
$ prt
bash: prt: command not found
```

Modification of PATH variable

Manual change of variable PATH



```
$ export PATH=/moje/cesta/k/mym/prikazum:$PATH
```

separator sign

The path to the directory containing the commands that I want to be accessible without specifying the path.

The path is always stated absolutely! (listing relative paths is a safety risk)

The original value of the variable **PATH** (required to find system commands)

Automated change of variable PATH

The automated change of the PATH variable (and possibly other system variables) is performed by the command **module.**

```
$ module add vmd
```

This will be disucessed in Lesson 4

Path to Commands, Documentation

Path to a command or application, if it exists, can be found by the command **type** or **whereis**

ls is in shell alias of command **ls** with color option

```
$ type ls
ls is aliased to 'ls --color=auto'
```

```
$ whereis ls
ls: /bin/ls /usr/share/man/man1/ls.1.gz
```

command **ls is** program stored in the file /bin/ls (man ls)

```
$ type pwd
pwd is a shell builtin
```

command **pwd** is implemented as an internal command of shell (documentation of pwd command is in man bash)

```
$ whereis pwd
pwd: /bin/pwd /usr/include/pwd.h /usr/share/man/man1/pwd.1.gz
```

some commands may have multiple implementations (man pwd), internal commands of shell are used first

Foreground and Background

Running applications in the foreground

```
$ gimp
```

processes running **in the foreground block the terminal** because they use its standard input and output

Running applications in the background

```
$ gimp &
```

processes running **in the background do not block the terminal**

at the end (after arguments and redirects) of the command, we type an ampersand

Terminal (useful keyboard shortcuts):

Ctrl + Z pauses the process, further fate of the process can be controled with the use of commands:

jobs lists the processes that shell manages

bg moves the process to the background

fg moves the process to the foreground

disown unbinds the process from the shell (process is not terminated with termination of the shell)

Commands and Applications ...

User programs and scripts

```
$ ./muj_script

$ ~/bin/my_application
```

the name of the program or script is given including the path (absolute or relative)

Cancellation of the output into the terminal

```
$ kwrite &> /dev/null
```

output redirection is specified at the end of the command (after arguments)

Running applications in the background

```
$ gimp &> /dev/null &
```

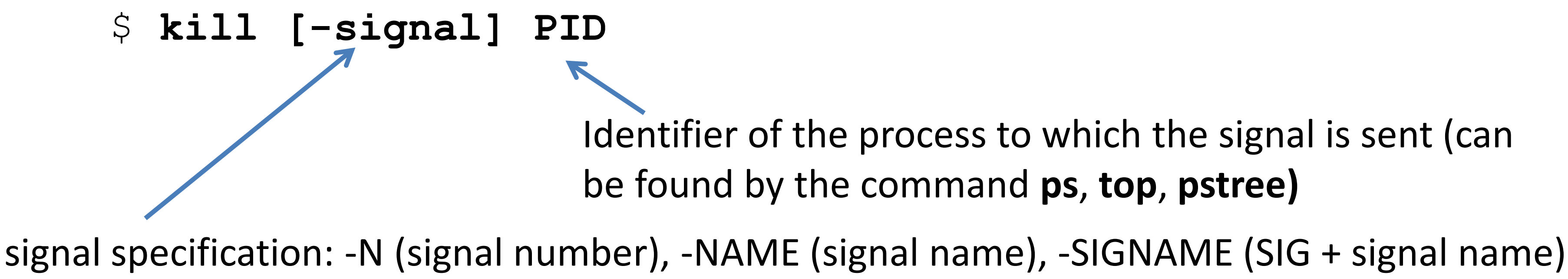
at the end (after arguments and redirects) of the statement we type an ampersand

Signals and Processes

Terminal (useful keyboard shortcuts):

! **Ctrl + C** sends a SIGINT signal to the running process (Interrupt), the process is forcibly terminated in most cases

Command kill:



```
$ kill [-signal] PID
```

Identifier of the process to which the signal is sent (can be found by the command **ps**, **top**, **pstree)**

signal specification: -N (signal number), -NAME (signal name), -SIGNAME (SIG + signal name)

Useful signals:

TERM	**15**	termination request (process can respond to signal)
INT	**2**	request for interruption (**Ctrl + C** equivalent), process may respond
KILL	**9**	end (**the process cannot ignore the signal, it is forcibly terminated**)
STOP		pauses process (process cannot ignore the signal), equivalent to **Ctrl + Z**
CONT		resumes run of paused process (process cannot ignore signal)

Overview of Commands

top continuously displays processes sorted by CPU load (end with q key)

ps lists processes running in given terminal or according to the specification (`ps -u user_name`)

pstree lists processes (tree listing)

type lists the path to the standard application/command (including internal commands of shell)

whereis lists the path to the standard application/command

time lists the length of the process run

sleep waits for specified time

kill sends a signal to process, can be used to terminate problematic programs

ssh runs the command on the remote computer

jobs lists background processes

fg brings the process to the foreground

bg movess the process to the background

nohup starts the process without interacting with the terminal (C2115)

wait waits for background processes to complete (C2115)

Exercise 1

1. Open a new terminal on the workstation wolf02.ncbr.muni.cz
2. List a table with already used commands (List should be empty).
3. Run the command **ls** and print the table with the commands already used.
4. Where is the file containing the program for the command **ls**. Use the command **type** and **whereis**. What is the difference between the two commands?
5. What is the size and access rights of the file that contains the program **ls**.
6. List the contents of the PATH variable (echo $PATH).
7. Does it contain the path to the directory in which the command **ls** is located?
8. Make a copy of the file with **ls** program to your home directory under the name **my_ls**.
9. Run the program **my_ls** and compare its output with the command **ls**. How do the outputs differ?
10. Delete **my_ls** file.

Exercise 2

1. Open a new terminal on the workstation wolf03.ncbr.muni.cz
2. Run the command sleep 60. What does the number 60 indicate?
3. Run the command sleep 300.
4. End it with Ctrl + C
5. Run the command sleep 300 and let it run.
6. Open a new terminal on the workstation wolf03.ncbr.muni.cz
7. List your running processes (ps -u username)
8. Terminate process sleep with the command kill
9. Repeat steps 5, 7, 8 for different signals (SIGTERM, SIGINT and SIGKILL)