C2110 *UNIX and programming*

4th lesson

Processes

Petr Kulhánek

kulhanek@chemi.muni.cz

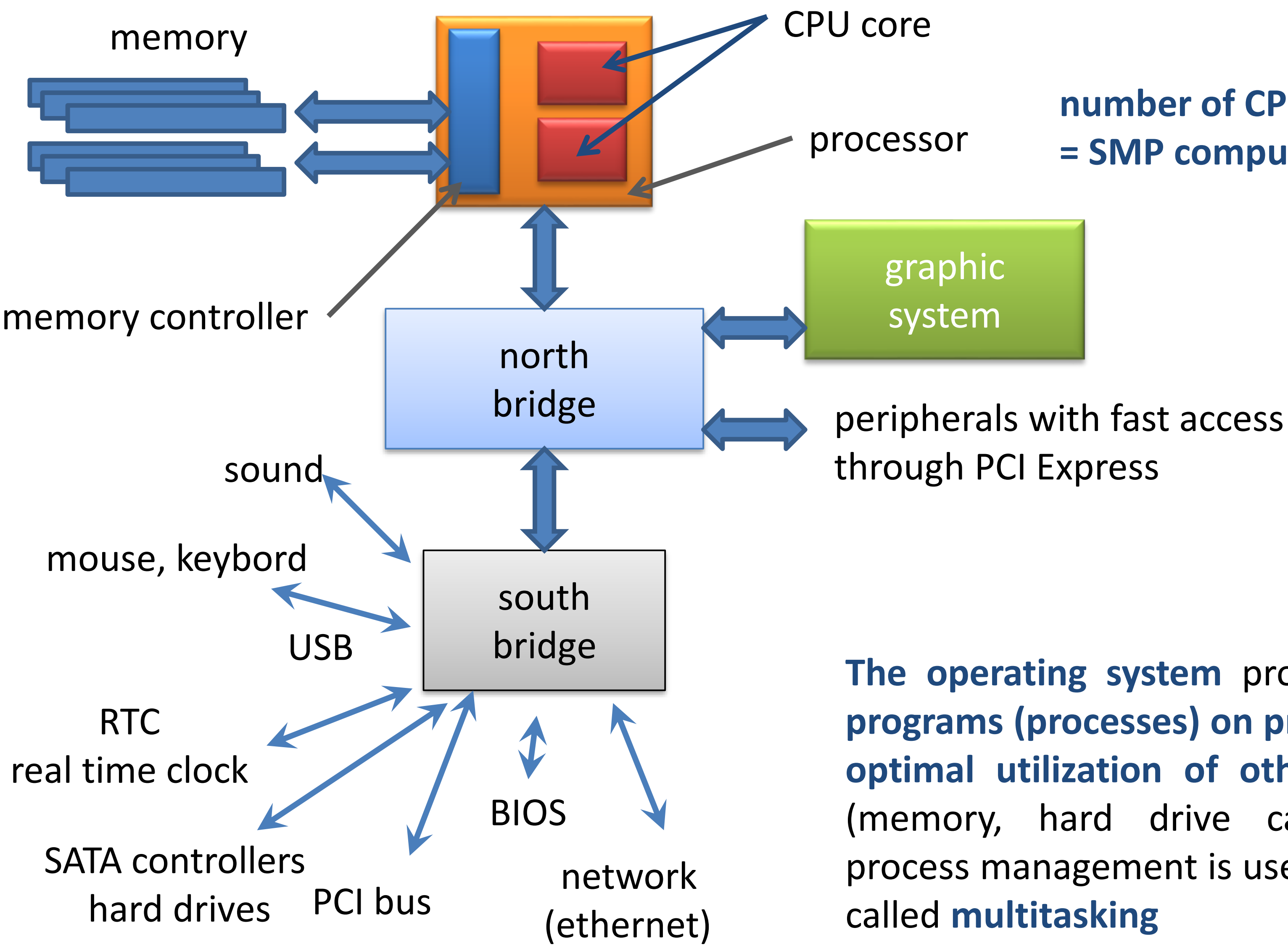
National Centre for Biomolecular Research, Faculty of Science,
Masaryk University, Kamenice 5, CZ-62500 Brno

Contents

- **Processes**
  - **process, processor, multitasking, monitoring**
  - **process execution, PATH variable**
- **Process communication with the environment**
  - **standard input and output, error output, redirection, pipes, commands**

Processes

Computer scheme



number of CPU cores > 1
= SMP computer

The operating system provides **run of programs (processes) on processors** and **optimal utilization of other resources** (memory, hard drive capacity). For process management is used technology called **multitasking**

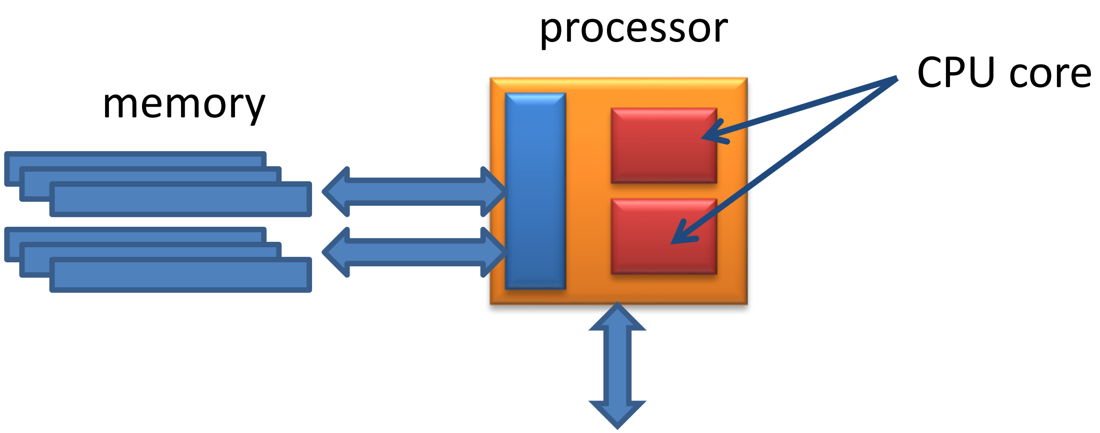
Processes and multitasking

Process is in computer science a term for **running instance of a computer program**. Process is **located in RAM** of computer in a form of a **sequence of machine instructions,** which are **executed by a processor**. It contains not only the code of a running program, but also dynamically changing data, that are handled by a process. One program can run multiple times as processeses with different data (for example, a web browser displaying different web pages). **Management of processes is handled by operating system,** that ensures isolated run of the processes, OS alocates system resources and allows users to manage processes (execution, termination, etc.).

Multitasking (used in a multiprocessing system) in computer science indicates **ability of operating system to perform several processes at the same time** (at least seemingly). Kernel of the operating system very quickly switches between running processes on a processor or processors, so an user has impression that more processes run at the same time.

adapted from wikipedia.org

SMP - Symmetric multiprocessing



CPU – Central Processing Unit

In the past, the processor speed had been improved by design changes which allowed to increase CPU clock rate and thus to speed up execution of the program. This strategy reach its limits (reliability, heat losses, ...) a few years ago. Thus the computational power was improved by putting more CPU (cores) on the same chip (since 2005 for x86).
Therefore, today's computers are multiprocessors systems.

Symmetric multiprocessing (SMP) is in computer science term for type of **multi-processor systems**, where all processors are equivalent. Increasing number of processors, which share the same memory, leads to **improvement of computer performance**, although not in a linear manner, since some part of power is consumed on overhead (locking data structures, processors control and their mutual communication).

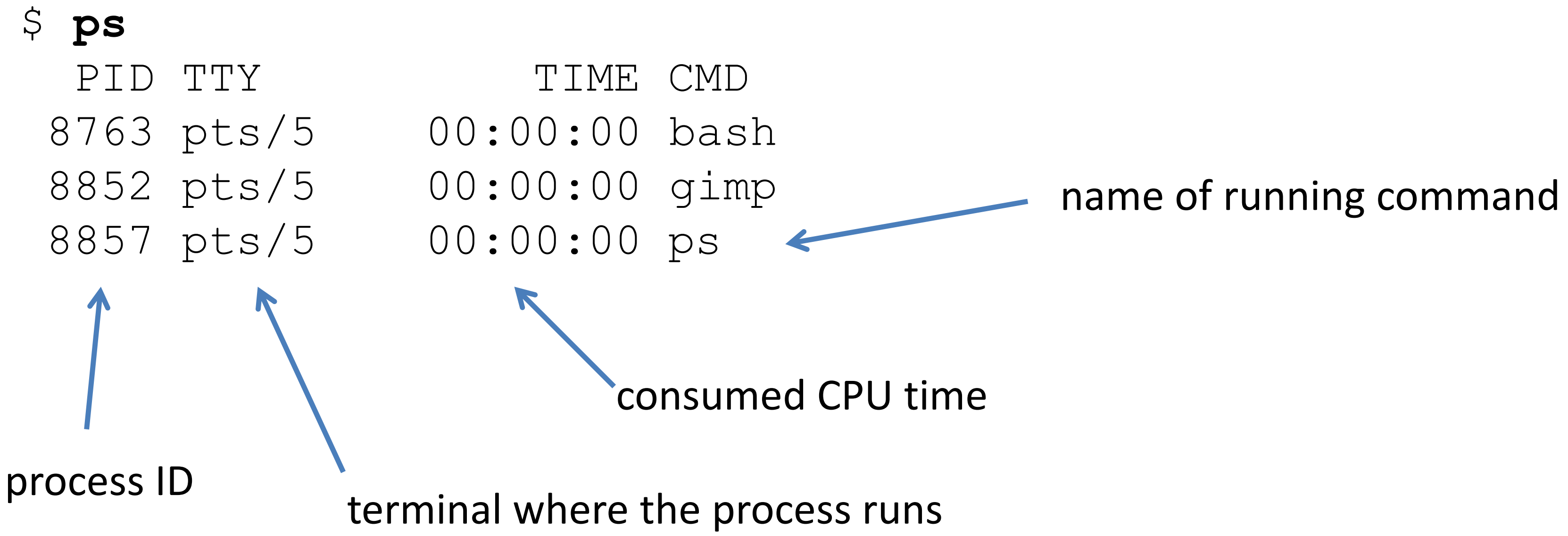
adapted from wikipedia.org

List of running processes

Commands for processes listing:

ps lists processes running in a given terminal or filtered by user specifications (`ps -u user_name`)

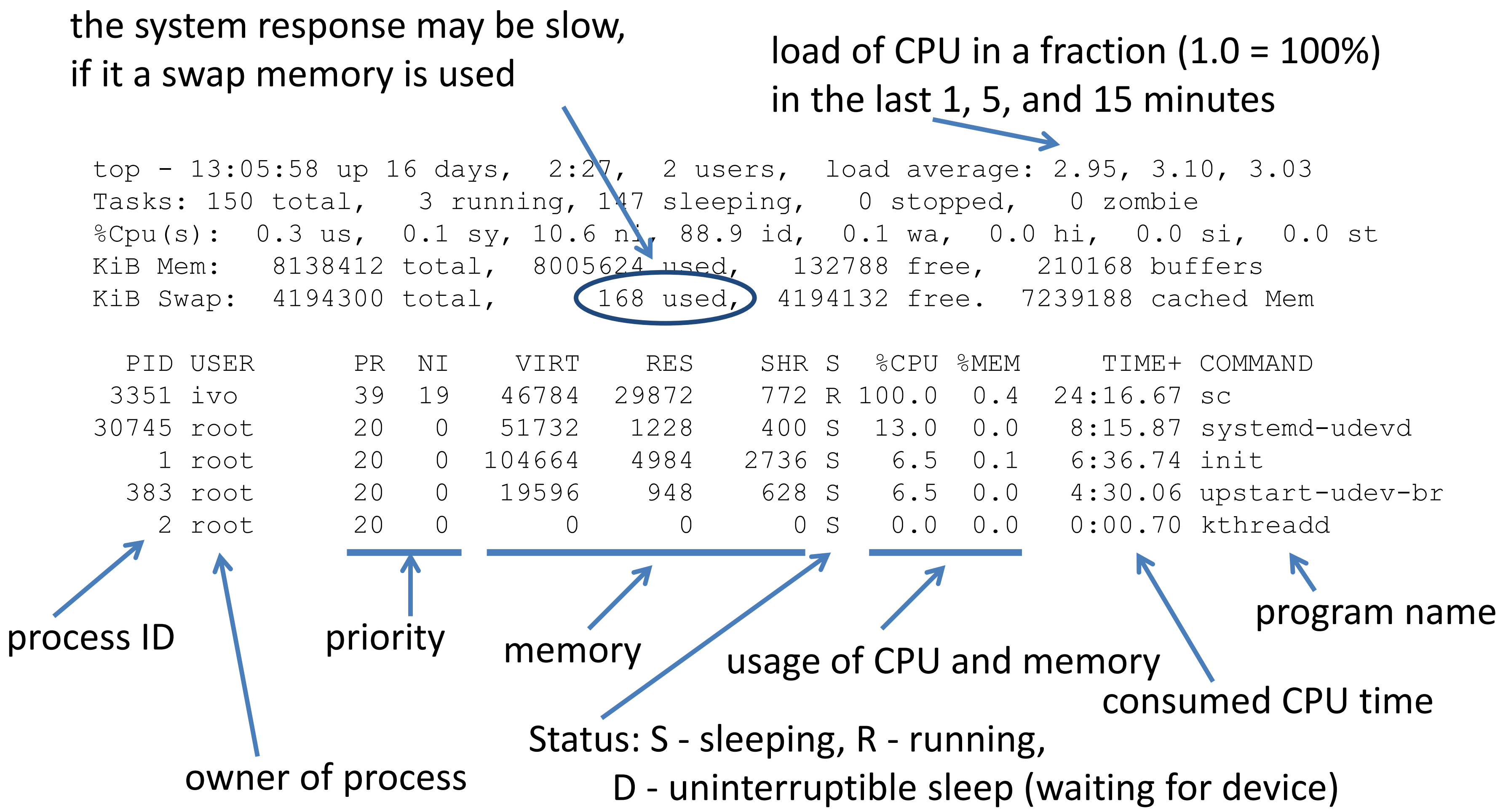
Top continuously displays processes sorted by CPU load (termination by the q key)



```
$ 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
```

List of running processes - top

The **top** command monitors running processes in regular intervals. top is terminated by the **q** (quit) key.



the system response may be slow, if it a swap memory is used

load of CPU 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 ID

priority

memory

program name

usage of CPU and memory

consumed CPU time

Status: S - sleeping, R - running,
D - uninterruptible sleep (waiting for device)

owner of process

Command and application run

To run a command by a shell, the shell **has to know a path** to the file that contains binary program or script.

1. First, path to a command is searched in a table with already used commands:

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

Table can be deleted by:

```
$ hash -r
```

2. If a command is not found, then a shell searchs a command file in directories listed in the system variable **PATH**, where individual paths are separated by a colon.

```
$ echo $PATH
.../usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

Order of directory search

Path to command or application, if it exists, can be accessed by the **type** or **whereis** commands:

```
$ type ls
/bin/ls
$ type pwd
pwd is a shell builtin
```

Command **ls** is saved in the file /bin/ls

pwd is implemented as a shell builtin

Modification of PATH variable

Manual modification of PATH variable

```
$ export PATH=/my/path/to/my/commands:$PATH
```

The path to the directory with your commands, which will be possible to execute without typing a path.
Paths have to be always absolute!
(usage of relative paths is serious security risk)

Separator

The default value of the **PATH** variable (needed to find a system commands)

Automatic modification of the PATH variable

Automatic modification of the variable PATH (and possibly other system variables) is performed by the **module** command:

```
$ module add vmd
```

Commands and applications run...

User programs and scripts

```
$ ./muj_script

$ ~/bin/my_application
```

Name of program or script must be given **with the path** (relative or absolute).

Redirection of program output

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

Redirection is specified at the end of the command (after arguments)

Running of applications in background

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

ampersand is given at the end (after arguments and redirection) of command

Terminal (useful shortcuts):

Ctrl + C **it** sends the **SIGINT** (interrupt) signal to running process
the process is usually terminated

Ctrl + D it closes an input stream of running process

Ctrl + Z suspends run of process, the fate of the process can be controlled using commands **bg, fg, disown**

Exercise I

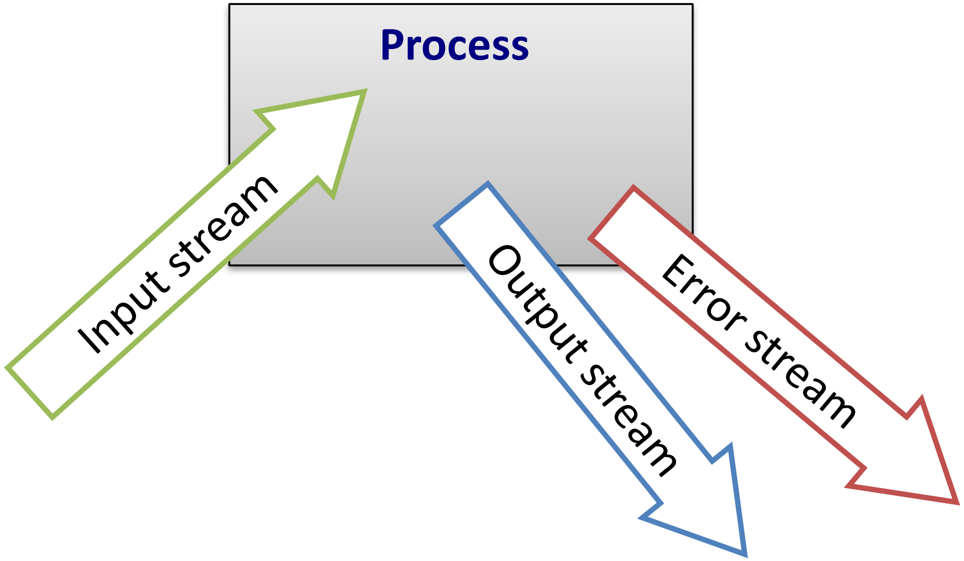
1. Print a table of already used commands (a list should be empty)
2. Run the ls command and again print the table with already used commands.
3. Where is a file that contains the program for the ls command? Use the type and whereis commands. What is difference between these two commands?
4. What is the size and permissions of the file that contains the program ls.
5. Print the content of the PATH variable (echo $ PATH)
6. Is the nemesis program listed in the PATH directories?
7. Add the nemesis module.
8. List contents of the variable PATH again.
9. What is the path with the nemesis program?
10. What is the size and permissions of the file containing program nemesis?
11. Create a copy of the program ls into your home directory with name ‘my_ls’.
12. Run the program my_ls.
13. Remove permissions for execution from the my_ls file.
14. Try to run the my_ls program again. What will happen?

Process in environment

Process is able to communicate with environment in various ways:

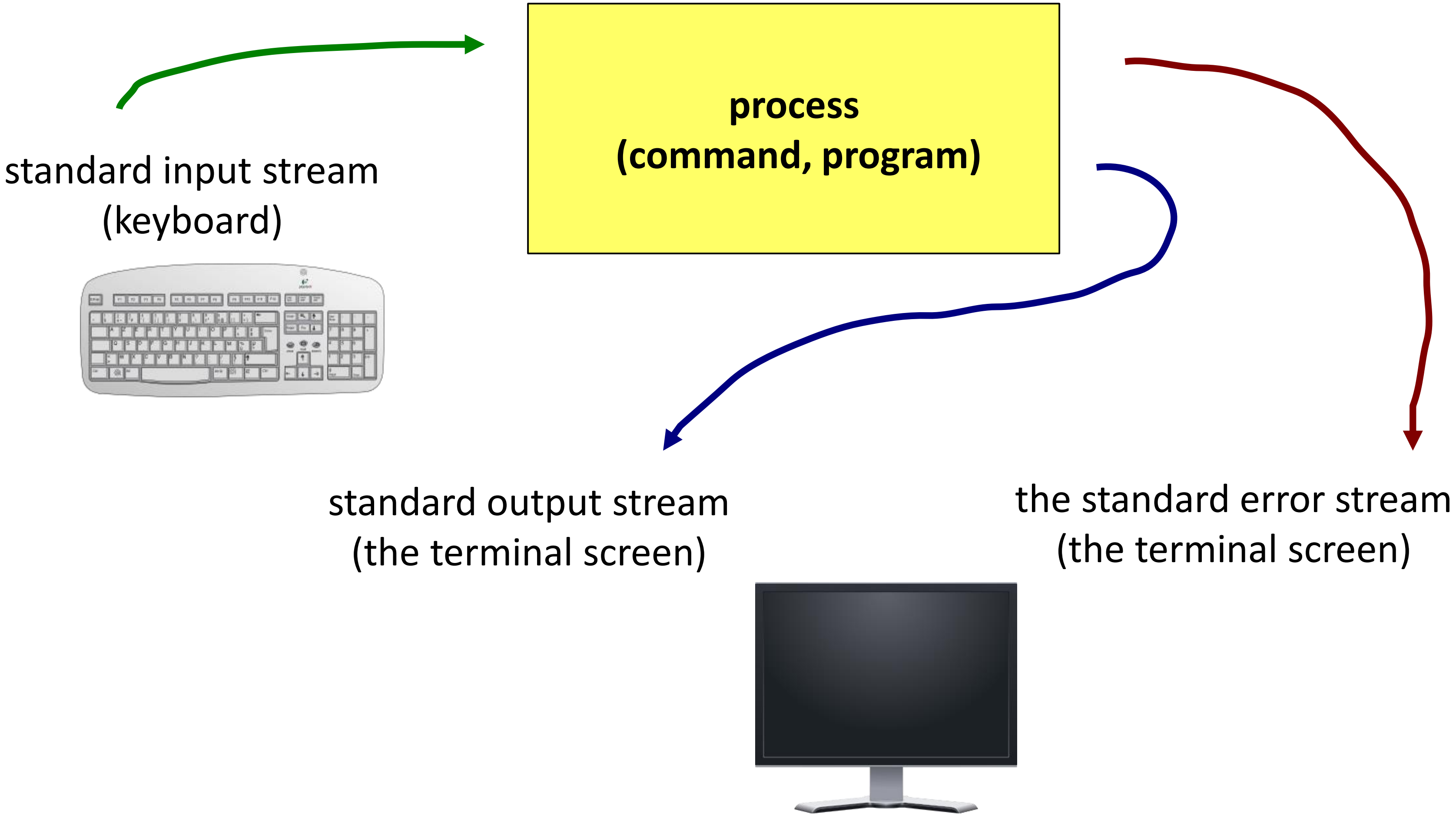
- GUI (Graphical User Interface = using the appropriate API)
- signals, shared memory, MPI (Message Passing Interface), etc.
- Standard streams

One of possibilities is to read input data from the **standard input stream**, print output data to the **standard output or error stream**.



Standard streams

Input-output streams of the process are used to **communicate** with the environment. Each process opens **three standard streams:**



process
(command, program)

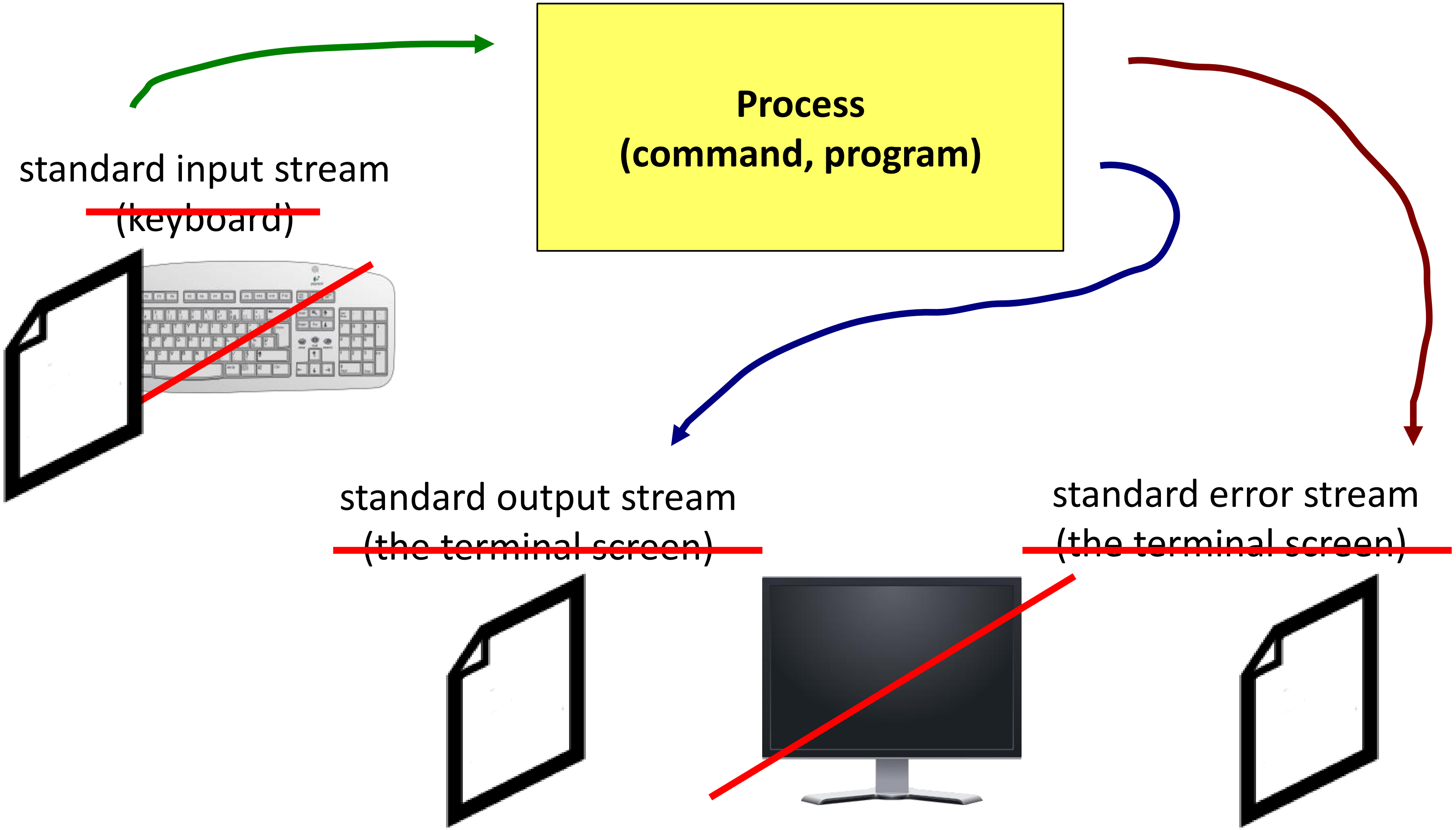
standard input stream
(keyboard)

standard output stream
(the terminal screen)

the standard error stream
(the terminal screen)

Redirection

Input-output streams is possible to redirect to use files instead of keyboard or monitor.



Process
(command, program)

standard input stream
~~(keyboard)~~

standard output stream
~~(the terminal screen)~~

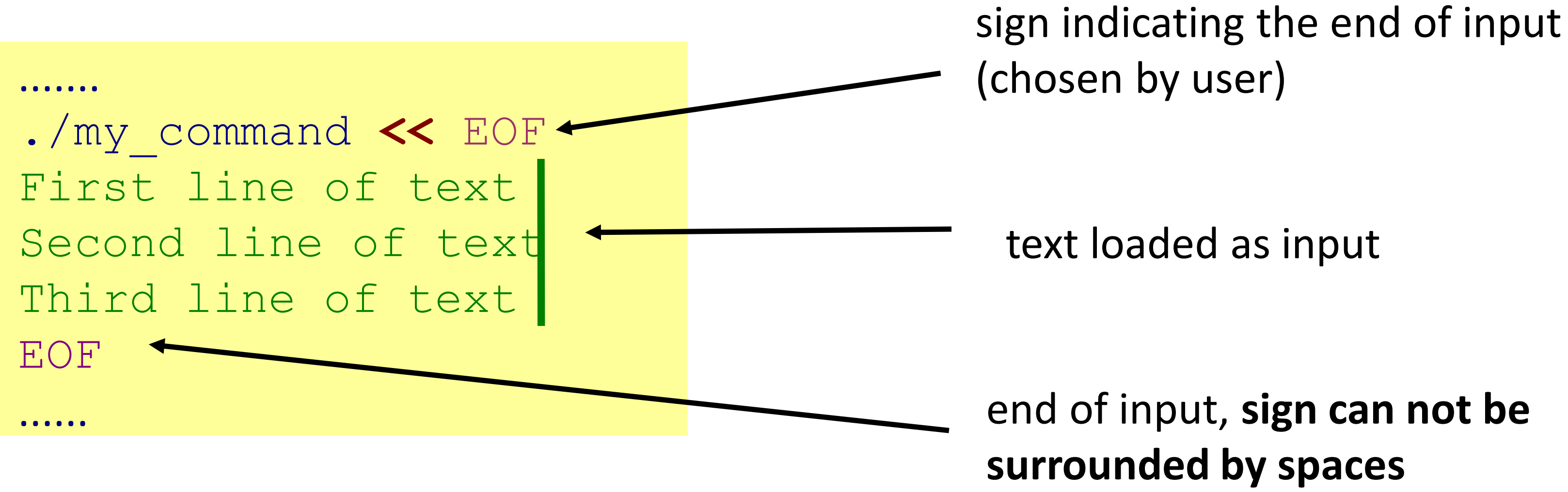
standard error stream
~~(the terminal screen)~~

Přesměrování vstupu

Redirection of standard input of program my_command from file **input.txt.**

```
$ my_command < input.txt
```

Redirection of standard input of my_command program from a script file



```
.......
./my_command << EOF
First line of text
Second line of text
Third line of text
EOF
......
```

sign indicating the end of input (chosen by user)

text loaded as input

end of input, **sign can not be surrounded by spaces**

This type of redirection is particularly advantageous to use in scripts, but it also works in command line. The advantage is expansion of variables in the loaded text.

Redirection of output

Redirection of standard output of the program my_command to file **output.txt**. (Output.txt file is created. If it exists, the original content is **overwrited**.)

```
$ my_command > output.txt
```

Redirection of standard output of the program my_command to file **output.txt**. (Output.txt file is created. If it exists, the output of my_command is **added in the end of file**.)

```
$ my_command >> output.txt
```

Similar rules apply for **standard error output**, in this case are used following operators

```
$ my_command 2> errors.txt
$ my_command 2>> errors.txt
```

Connection of standard streams

Standard output and standard error output of the program my_command is possible to redirect at the same time to file **output.txt**.

```
$ my_command &> output.txt
```

```
$ my_command &>> output.txt
```

works in new versions of bash

Alternative solutions for &>>: first it is necessary to **redirect** the standard output and then **connect** standard error output with standard output.

```
$ my_command >> output.txt 2>&1
```

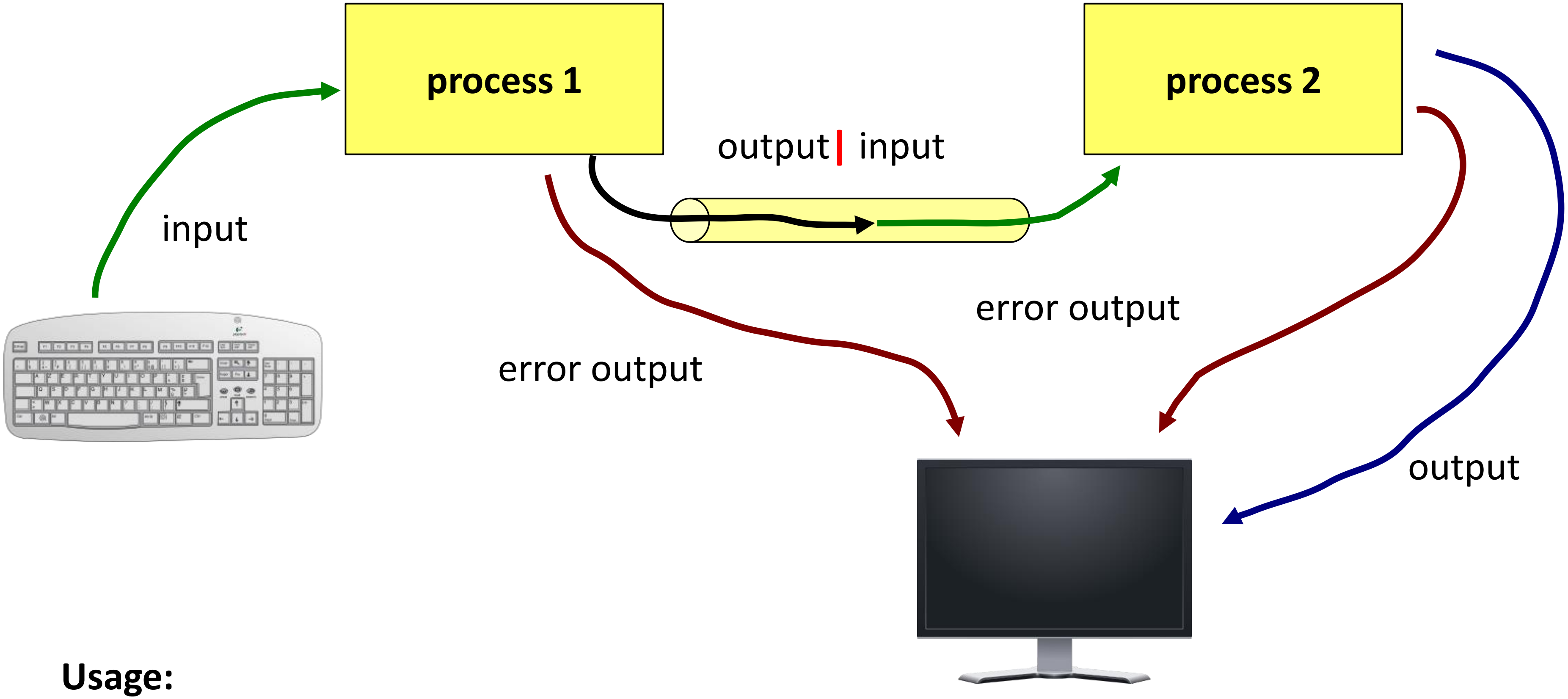
order is important!

~~$ my_command 2>&1 >> output.txt~~

do not work

Pipes

Pipes are used to connect the standard output of one process to the standard input of different process.

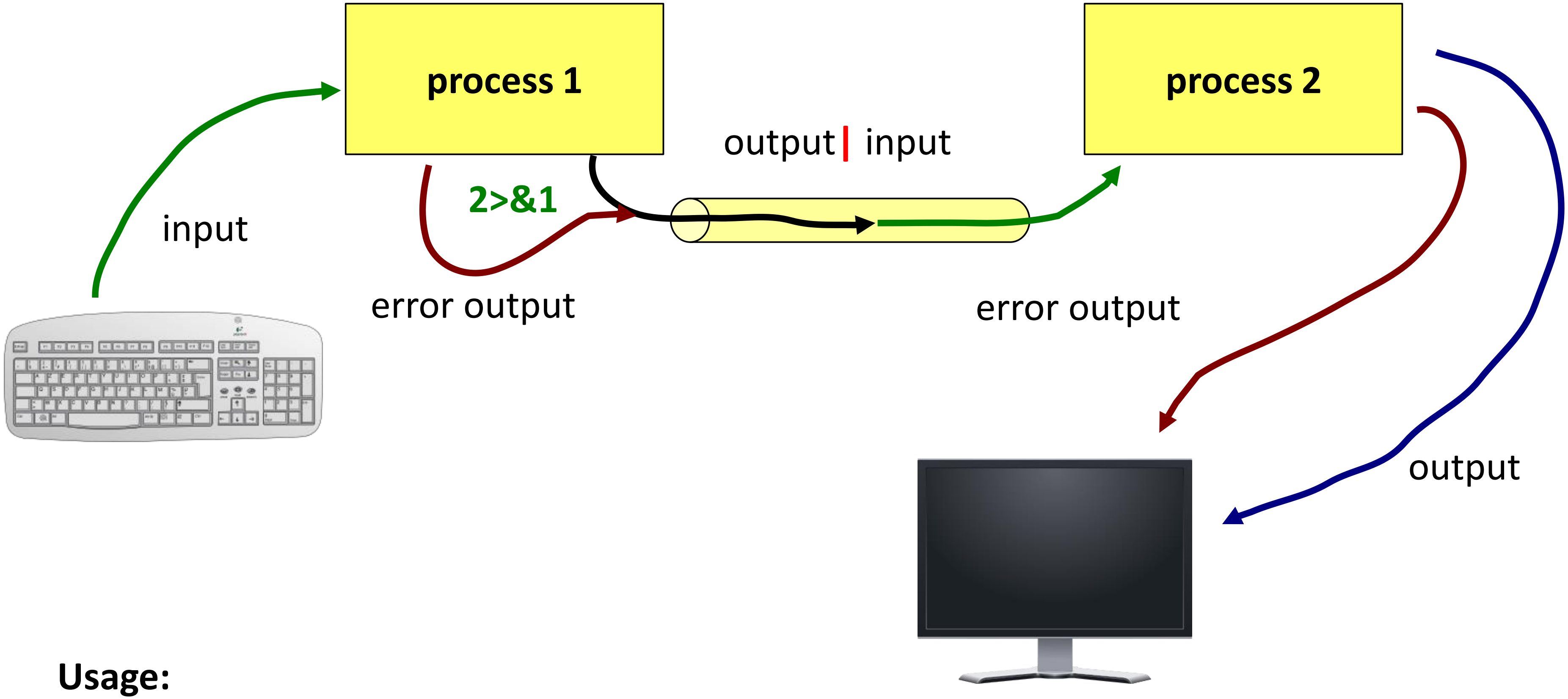


Usage:

```
$ command_1 | command_2
```

Pipes and error stream

Transfer of standard error output by pipe is possible to do after its connection with standard input.



Usage:

```
$ command_1 2>&1 | command_2
```

Commands to exercise

cat merges the contents of multiple files into one (sequentially), or prints the content of one file

paste merges the contents of multiple files into one (in parallel)

wc prints file information (number of lines, words and characters)

head prints first line(s) of file

tail prints last line(s) of file

Showcase:

```
$ cat file1.txt file2.txt
```

connects content of file1.txt and file2.txt sequentially, print the output on the screen

```
$ paste file1.txt file2.txt
```

connects content of file1.txt and file2.txt in parallel, print the output on the screen

```
$ wc file.txt
```

prints the number of lines, words and characters of the file file.txt

```
$ head -15 file.txt
```

prints the first 15 lines of the file file.txt

```
$ tail -6 file.txt
```

prints the last 6 lines of the file file file.txt

Command for exercise ...

tr command is used to transform or delete characters from standard input. Result is given to standard output.

Examples:

```
$ cat file.txt | tr --delete "qwe"
```

from the content of **file.txt** removes characters "q", "w" and "e"

```
$ cat file.txt | tr --delete "[:space:]"
```

from the content of **file.txt** removes all whitespaces

```
$ echo $PATH | tr ":" "\n"
```

In the text send by echo command will be replaced all characters ":" by character for newline “\n"

Exercise II

1. Find all files with suffix .f90 in the directory /home/kulhanek/Documents/C2110/Lesson03/, save the list of files into ~/Processes/list.txt
2. How many lines does the file list.txt contain?
3. Print first two lines from list.txt on the screen and then into the file two_lines.txt
4. Print only the third line of the file list.txt
5. In /proc directory, find all files that begin with the letters cpu. Remove information about unauthorized access by redirection of error stream to /dev/null
6. Print directories contained in the PATH variable, each on a separate line.
7. Activate the vmd module. How will it change the content of the PATH variable?

Conclusion

Conclusion

- Process is instance of running program. Operating system uses multitasking to run multiple processes on multiple cores.
- Program is a binary file directly executed by the processor.
- If program exists in any directory listed in the PATH variable, you can specify program name without a path. Otherwise it is necessary to include the path.
- Each process can communicate using three streams. It is possible to manipulate with these streams. It is possible to redirect streams or connect them with each other.

Homework

- Exercise lessons 1 to 4
- Text editors

Text editors

- vi, vim, nano
- graphical text editors (kwrite, gedit, kate)

Text editors – installation

Try to use individual text editors on your installated Ubuntu 14.04 LTS. If they will not be available, it is possible to install them as follows:

```
$ sudo apt-get install vim
$ sudo apt-get install kwrite
$ sudo apt-get install kate
$ sudo apt-get install gedit
$ sudo apt-get install nano
```

When you will be asked, enter the password for your account.

In default installation, it is installed vi editor in compatibility mode, which is useful to replace by improved version (vim). Installation see above.

vi/vim, nano

vi/vim editor is a standard text editor for UNIX-like operating systems. It works only in text mode, and its use is trivial.

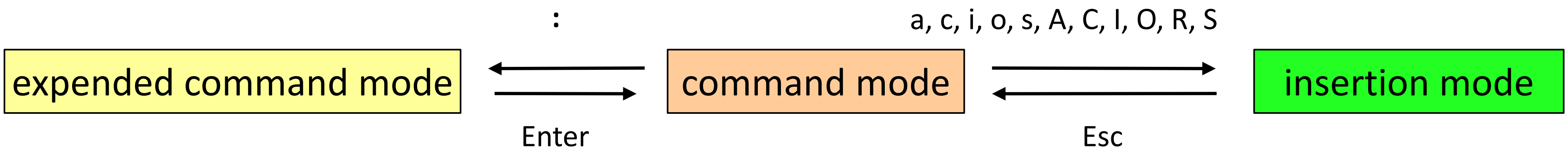
- It is advisable to learn how to open a file, go into edit mode, edit text, save changes and close the editor.
- Allows scripting (using variables, loops, arrays, associative arrays), eg . for creating automatic texts of the loaded data.
- When in this room, you will run the vi command, it automatically starts program vim (Vi IMporoved)
- Between the original vi and vim, it is a difference in handling.

Editor nano is the default text editor on some distributions (Ubuntu).

- less versatile than vim
- more direct control

vi – základy

Pracovní módy editoru



:

a, c, i, o, s, A, C, I, O, R, S

expended command mode

command mode

insertion mode

Enter

Esc

Launch of editor

$ vi	**launchs** of editor
$ vi *filename*	**launchs** of editor and **opens file** filename

Turn off editoru

:q	**turns off** editor
:q!	**turns off editor without saving of changes**
:w	**saves** file
:w *filename*	**saves** file to *filename*
:wq	**turns off and saves** file

File modification

i	insertion of text **to** position of cursor
a	insertion of text **behind** position of cursor

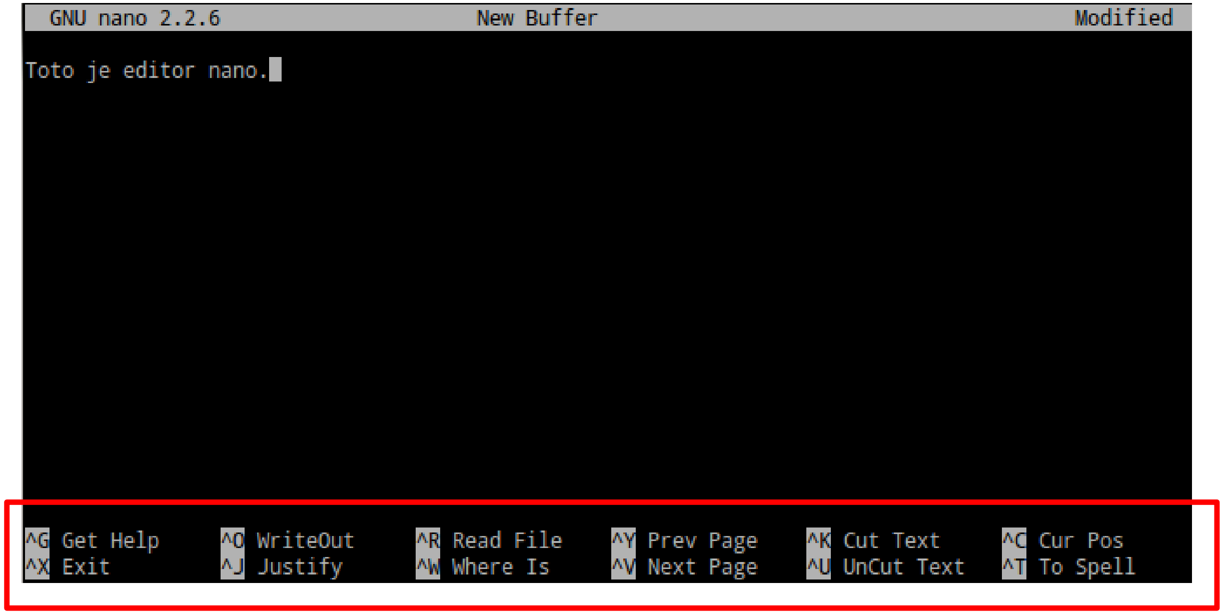
More functionality: in file vi appendix

nano

Launch of editor

$ nano **launchs** of editor

$ nano ***filename*** **launchs** of editor and **opens file** filename



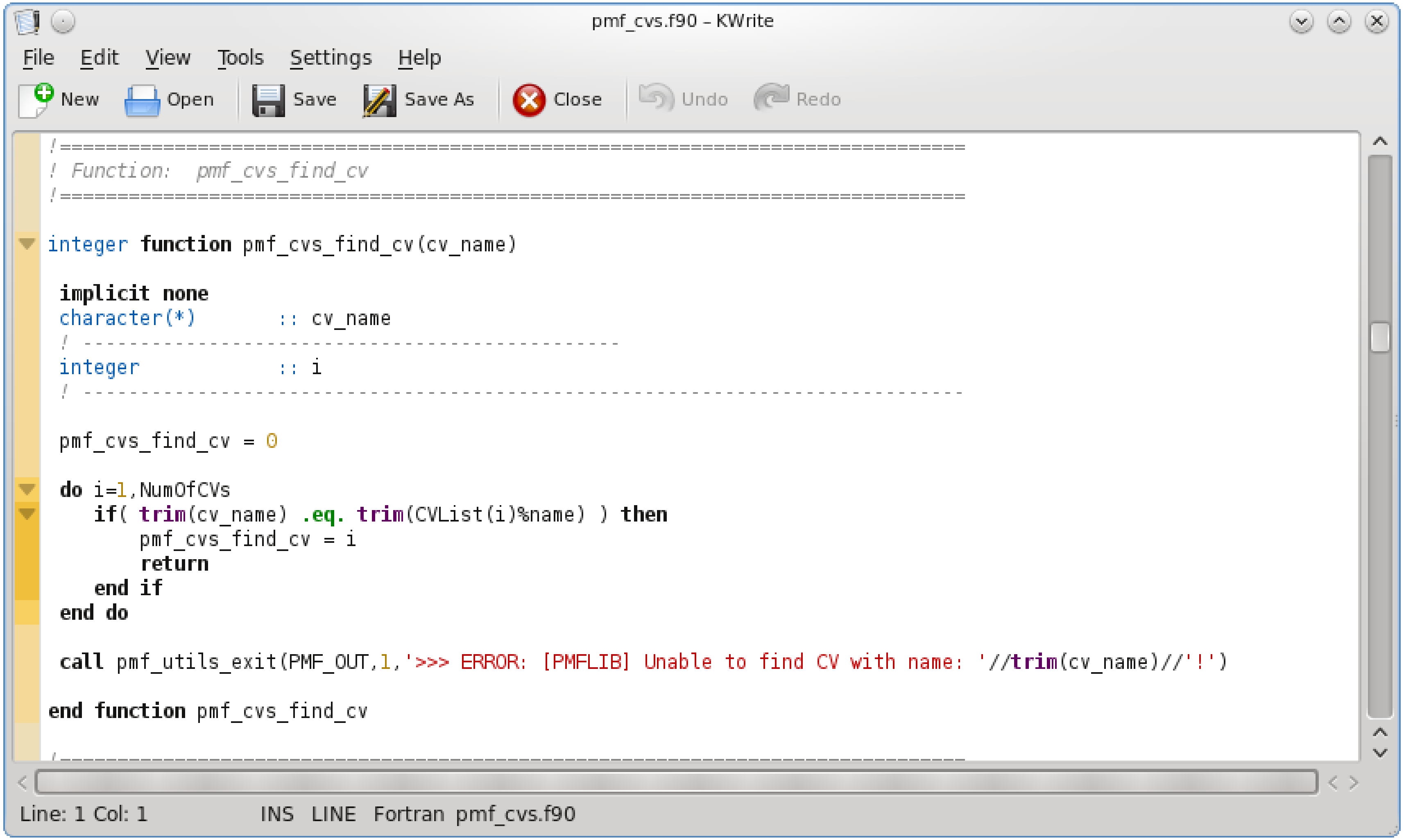
```
 GNU nano 2.2.6              New Buffer                     Modified

Toto je editor nano.

^G Get Help   ^O WriteOut   ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit       ^J Justify    ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
```

More straight handling - menu at the bottom suggests possible action. To select events use combination or individual characters

^letter - eg. ^X is same as Ctrl + X

M-letter - eg. M-M is same as the Alt + M

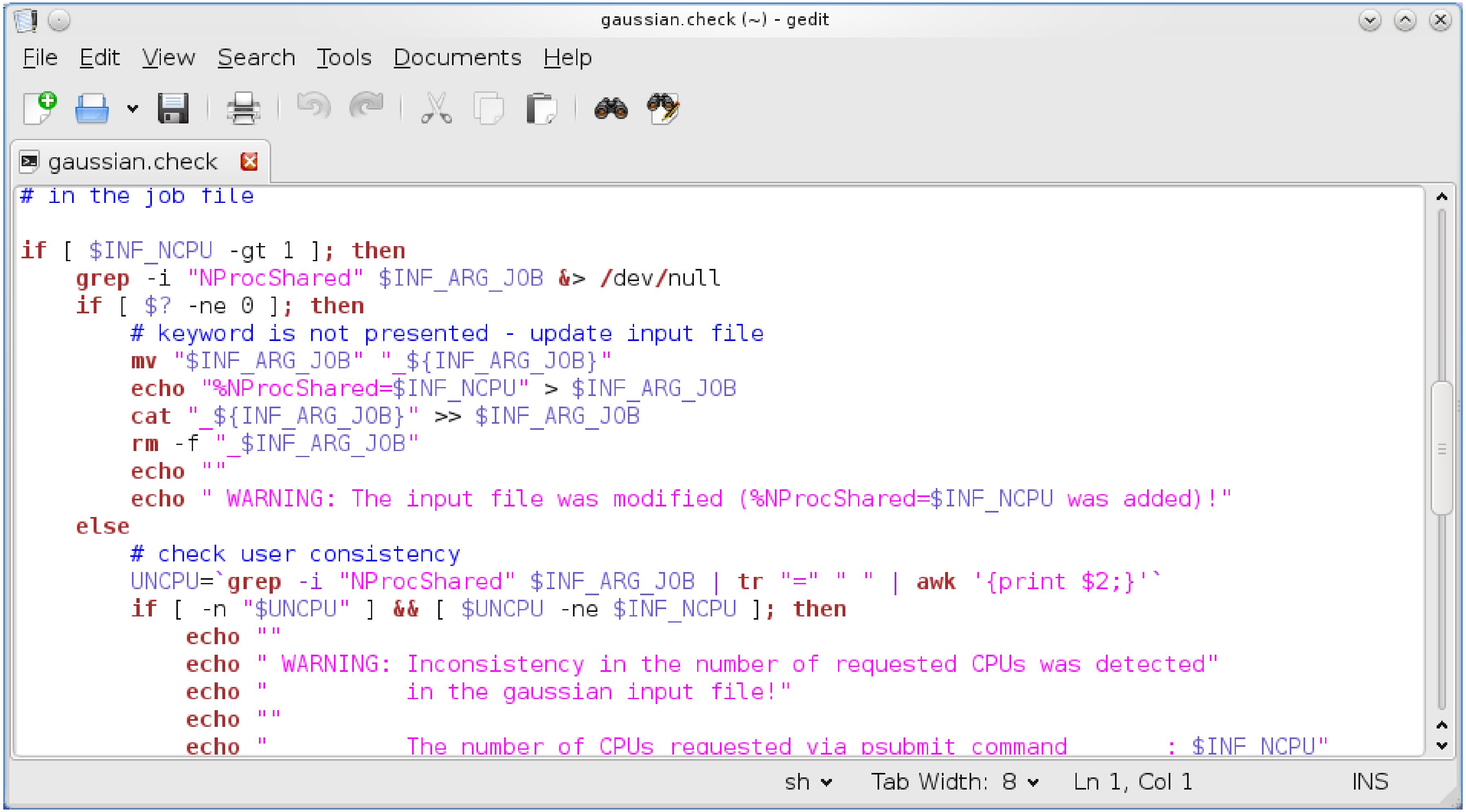
kwrite



```
!===============================================================================
! Function:  pmf_cvs_find_cv
!===============================================================================

integer function pmf_cvs_find_cv(cv_name)

 implicit none
 character(*)       :: cv_name
 ! -----------------------------------------------
 integer            :: i
 ! -----------------------------------------------------------------------------

 pmf_cvs_find_cv = 0

 do i=1,NumOfCVs
    if( trim(cv_name) .eq. trim(CVList(i)%name) ) then
        pmf_cvs_find_cv = i
        return
    end if
 end do

 call pmf_utils_exit(PMF_OUT,1,'>>> ERROR: [PMFLIB] Unable to find CV with name: '//trim(cv_name)//'!')

end function pmf_cvs_find_cv

!===============================================================================
```

Expanded functionality: **kate**

gedit


gaussian.check (~) - gedit
File Edit View Search Tools Documents Help
gaussian.check
sh
Tab Width: 8
Ln 1, Col 1
INS

```
# in the job file

if [ $INF_NCPU -gt 1 ]; then
    grep -i "NProcShared" $INF_ARG_JOB &> /dev/null
    if [ $? -ne 0 ]; then
        # keyword is not presented - update input file
        mv "$INF_ARG_JOB" "_${INF_ARG_JOB}"
        echo "%NProcShared=$INF_NCPU" > $INF_ARG_JOB
        cat "_${INF_ARG_JOB}" >> $INF_ARG_JOB
        rm -f "_$INF_ARG_JOB"
        echo ""
        echo " WARNING: The input file was modified (%NProcShared=$INF_NCPU was added)!"
    else
        # check user consistency
        UNCPU=`grep -i "NProcShared" $INF_ARG_JOB | tr "=" " " | awk '{print $2;}'`
        if [ -n "$UNCPU" ] && [ $UNCPU -ne $INF_NCPU ]; then
            echo ""
            echo " WARNING: Inconsistency in the number of requested CPUs was detected"
            echo "          in the gaussian input file!"
            echo ""
            echo "          The number of CPUs requested via psubmit command    : $INF_NCPU"
```

Homework

1. Write ten lines long text in vi editor, with two or more words on each line. Save text into file mydata.txt
2. Use command wc to verify, that the file is actually ten lines long.
3. Using the pipe(s), write a sequence of commands that prints on the screen only number of words in the file mydata.txt
4. In a graphical editor of your choice, create a file containing ten words, each word on a new line. Save text into file second_data.txt
5. Use paste command to create a file all_data.txt that will contain content of files mydata.txt and second_data.txt in parallel.
6. Use wc command to verify that the file contains just ten lines.
7. Open all_data.txt file in a graphical text editor and visually verify the content of the file
8. Try to work in different text editors and choose the one which fits you best.