

# ***C2110 Operační systém UNIX a základy programování***

**4. lekce**

**Procesy**

**Petr Kulhánek, Jakub Štěpán**

[kulhanek@chemi.muni.cz](mailto:kulhanek@chemi.muni.cz)

Národní centrum pro výzkum biomolekul, Přírodovědecká fakulta  
Masarykova univerzita, Kamenice 5, CZ-62500 Brno

## ➤ **Procesy**

- **proces, multitasking, monitoring**
- **spouštění procesů, proměnná PATH**

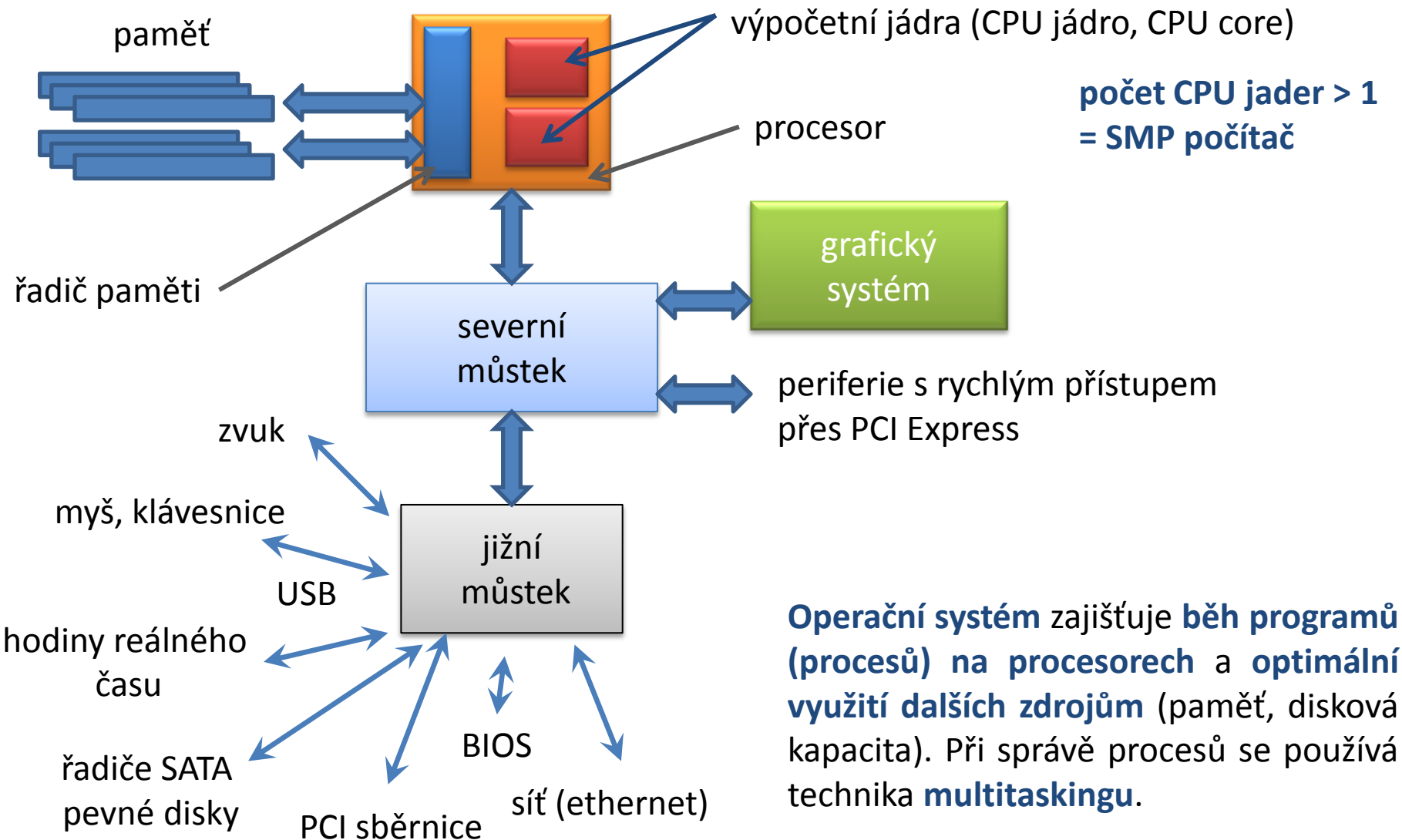
## ➤ **Komunikace procesu s okolím**

- **standardní vstup a výstup, chybový výstup, přesměrování, roury, příkazy**

# Procesy

---

# Vnitřní schéma počítače



**Operační systém** zajišťuje **běh programů (procesů)** na procesorech a **optimální využití dalších zdrojů** (paměť, disková kapacita). Při správě procesů se používá technika **multitaskingu**.

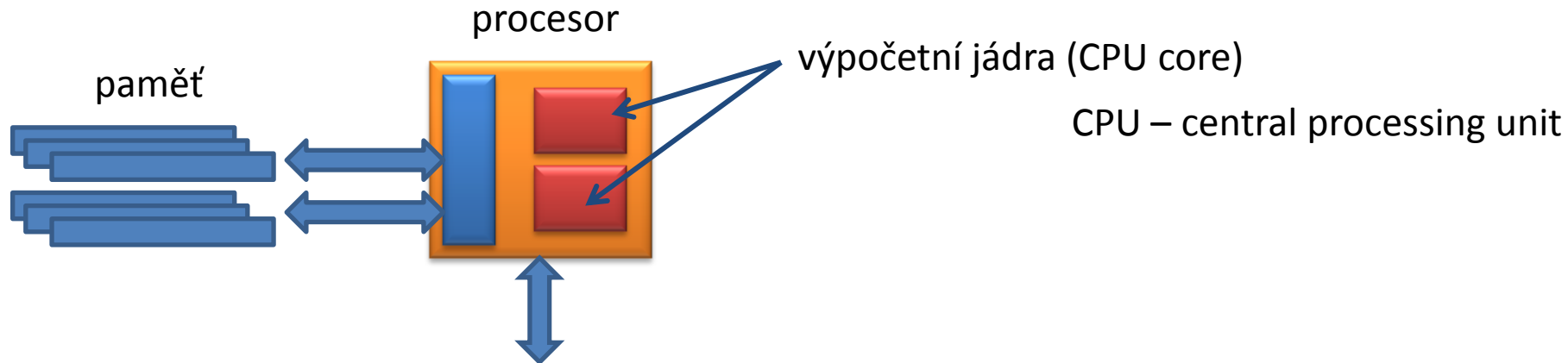
# Proces a multitasking

**Proces** (anglicky process) je v informatice název pro **spuštěný počítačový program**. Proces je **umístěn v operační paměti počítače** v podobě **sledu strojových instrukcí vykonávaných procesorem**. Obsahuje nejen kód vykonávaného programu, ale i dynamicky měnící se data, která proces zpracovává. Jeden program může v počítači běžet jako více procesů s různými daty (například vícekrát spuštěný webový prohlížeč zobrazující různé stránky). **Správu procesů vykonává operační systém**, který zajišťuje jejich oddělený běh, přiděluje jim systémové prostředky počítače a umožňuje uživateli procesy spravovat (spouštět, ukončovat atp.).

**Multitasking** (z angličtiny, multi = mnoho, task = úloha, používán ve víceúlohovém systému) označuje v informatice **schopnost operačního systému provádět několik procesů současně** (přinejmenším zdánlivě). Jádro operačního systému velmi rychle střídá na procesoru či procesorech běžící procesy (tzv. změna kontextu), takže uživatel počítače má dojem, že běží současně.

upraveno z wikipedia.org

# SMP – Symetrický multiprocessing



V minulosti se rychlost výkonu procesorů zvyšovala kromě lepší architektury i rychlostí zpracovávání instrukcí (frekvence procesoru), což v dnešní době naráží na fyzikální omezení používané technologie (spolehlivost, tepelné ztráty, ...). Dalším směrem bylo tedy uvedení více výpočetních jader (cca od roku 2005 pro x86 architekturu) na jednom fyzickém čipu. **Dnešní počítače jsou tak již běžně víceprocesorové.**

**Symetrický multiprocessing** (SMP, anglicky Symmetric multiprocessing) je v informatice označení pro druh **víceprocesorových systémů**, u kterých jsou všechny procesory v počítači rovnocenné. Zvýšení počtu procesorů, které v počítači sdílí stejnou operační paměť, vede **ke zvýšení výkonu počítače**, i když ne lineárním způsobem, protože část výkonu je spotřebována na režii (zamykání datových struktur, řízení procesorů a jejich vzájemná komunikace).

upraveno z wikipedia.org

# Přehled běžících procesů

## Procesy lze vypsat příkazy:

- ps** vypíše procesy běžící v daném terminálu nebo podle zadaných specifikací (`ps -u user_name`)
- top** průběžně zobrazuje procesy setříděné podle zátěže procesoru (ukončení klávesou `q`)

```
$ 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
```

jméno spuštěného příkazu

číslo procesu

terminál, ve kterém proces běží

spotřebovaný strojový čas

# Přehled běžících procesů - top

Příkazem **top** je možné v pravidelných intervalech monitorovat běžící procesy. Běh příkazu se ukončuje klávesou **q** (quit).

odezva systému může být pomalá,  
pokud je používána swapovací paměť

zatížené CPU ve zlomku (1.0 = 100%)  
v poslední 1, 5 a 15 minutách

```
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

číslo procesu

vlastník procesu

priorita

paměť

využití CPU a paměti

jméno programu

spotřebovaný CPU čas

stav: S – sleeping, R – running,

D – nepřerušitelný spánek (čeká na zařízení)



# Spouštění příkazů a aplikací

Aby mohl shell zadaný příkaz spustit, potřebuje **znát cestu** k souboru, který obsahuje binární program nebo skript.

1. Cesta k příkazu se nejdříve hledá v tabulce s již použitými příkazy:

```
$ hash
```

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

Tabulku lze smazat příkazem:

```
$ hash -r
```

2. Pokud není příkaz nalezen, hledá se v adresářích uvedených v systémové proměnné **PATH**, které jsou odděleny dvojtečkou.

```
$ echo $PATH
```

pořadí prohledávání adresářů

```
.../usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```



**Cestu** k příkazu či aplikaci, pokud existuje, lze zjistit příkazem **type** nebo **whereis**

```
$ type ls
```

```
/bin/ls
```

příkaz **ls** je program uložen v souboru /bin/ls

```
$ type pwd
```

```
pwd is a shell builtin
```

příkaz **pwd** je implementován jako vnitřní příkaz shellu

# Úprava proměnné PATH

## Manuální změna proměnné PATH

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



oddělující znak

Cesta k adresáři obsahující příkazy, u kterých chci, aby byly přístupné bez uvádění cesty.

**Cesta se vždy uvádí absolutně!** (uvádění relativních cest je bezpečnostním rizikem)

Původní hodnota proměnné **PATH**  
(nutné pro nalezení systémových příkazů)

## Automatizovaná změna proměnné PATH

Automatizovanou změnu proměnné PATH (a případně jiných systémových proměnných) provádí příkaz **module**.

```
$ module add vmd
```

# Spouštění příkazů a aplikací ...

## Uživatelské programy a skripty

```
$ ./muj_script
```

```
$ ~/bin/my_application
```

jméno programu nebo skriptu  
udáváme **včetně cesty** (absolutní  
nebo relativní)

## Zrušení výpisu do terminálu

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

↙ přesměrování výstupu uvádíme na konec příkazu  
(za argumenty)

## Spouštění aplikací na pozadí

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

↙ na konec (za argumenty a přesměrování) příkazu  
vedeme ampersand

## Terminál (užitečné klávesové zkratky):

**Ctrl+C** běžícímu procesu zašle signál **SIGINT** (Interrupt), proces je ve většině případů násilně ukončen

**Ctrl+D** zavře vstupní proud spuštěného procesu

**Ctrl+Z** pozastaví běh procesu, další osud procesu lze kontrolovat pomocí příkazů **bg**, **fg**, **disown**



# Cvičení

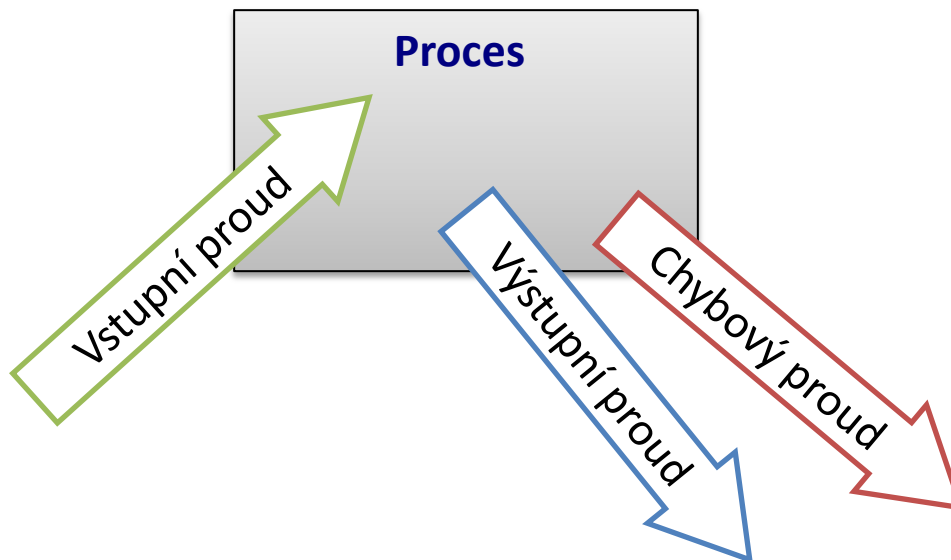
1. Vypište tabulku s již použitými příkazy (Výpis by měl být prázdný).
2. Spusťte příkaz **ls** a opět vypište tabulku s již použitými příkazy.
3. Kde se nalézá soubor obsahující program k příkazu **ls**. Použijte příkaz **type** a **whereis**. Jaký je mezi oběma příkazy rozdíl?
4. Jakou velikost a přístupová práva má soubor, který obsahuje program **ls**.
5. Vypište obsah proměnné **PATH** (echo \$PATH).
6. Je v adresářích uvedených v proměnné **PATH** dostupný program **nemesis**?
7. Přidejte modul **nemesis**.
8. Vypište obsah proměnné **PATH**.
9. V kterém adresáři je dostupný program **nemesis**?
10. Jakou velikost a přístupová práva má soubor, který obsahuje program **nemesis**.
11. Vytvořte kopii souboru s programem **ls** do vašeho domovského adresáře pod názvem **my\_ls**.
12. Spusťte program **my\_ls**.
13. Odstraňte souboru **my\_ls** přístupová práva pro spuštění.
14. Pokuste se znovu program **my\_ls** spustit. Co se stane?

# Komunikace procesu s okolím

**Proces** může komunikovat s okolím celou řadou způsobů:

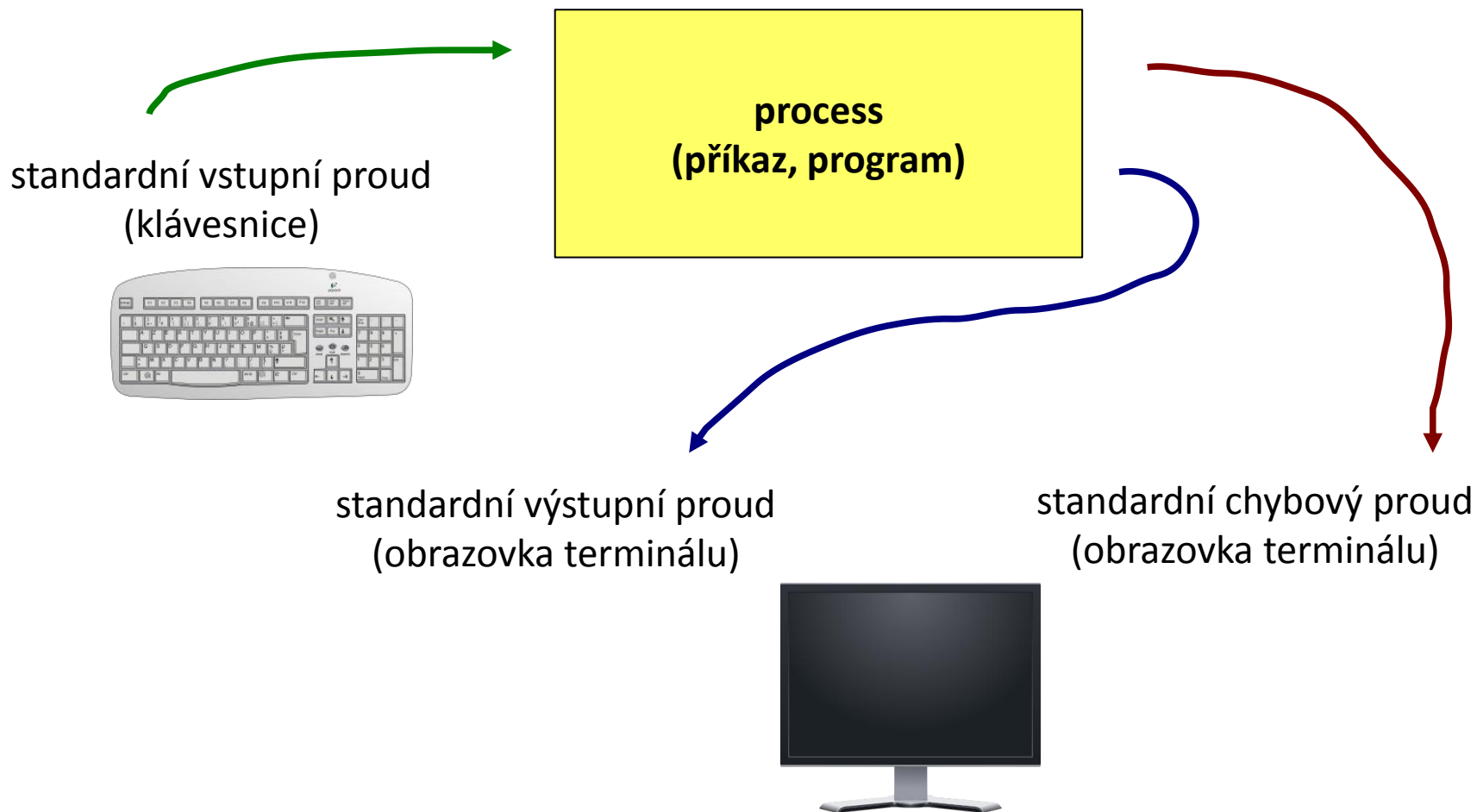
- GUI (Graphical User Interface = použitím příslušného API)
- signály, sdílená paměť, MPI (Message Passing Interface), atd.
- standardní proudy

Jednou z možností je načítání vstupních dat ze **standardního vstupního proudu**, výpis výstupních dat do **standardního výstupního** či **chybového proudu**.



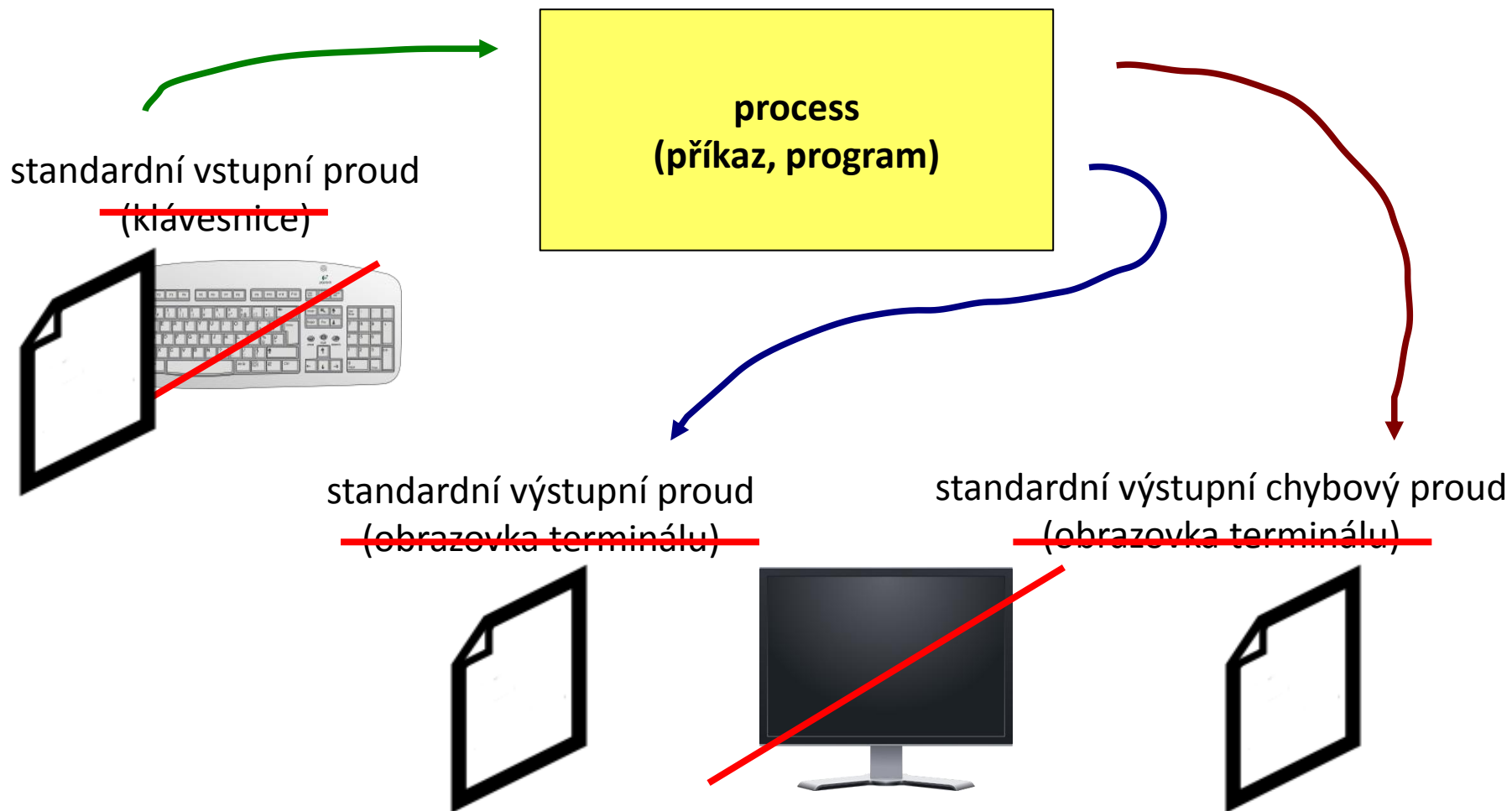
# Standardní proudy

**Vstupně-výstupní proudy** slouží procesu ke **komunikaci** se svým okolím. Každý proces otevírá **tři standardní proudy**:



# Přesměrování

**Vstupně-výstupní proudy** lze přesměrovat tak, aby používaly **soubory** místo klávesnice či obrazovky.



# Přesměrování vstupu

Přesměrování standardního vstupu programu `my_command` ze souboru `input.txt`.

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

Přesměrování standardního vstupu programu `my_command` ze souboru skriptu.

```
.....  
./my_command << EOF  
první radka textu  
druhá radka textu  
třetí radka textu  
EOF  
.....
```

značka určující konec vstupu  
(volí uživatel)

text, který tvoří načítaný vstup

konec vstupu, značku *nesmí*  
obklopotvat mezery

Tento způsob přesměrování je obzvláště výhodné používat ve skriptech, nicméně funguje i v příkazové řádce. Výhodou je expanze proměnných v načítaném textu.



# Přesměrování výstupu

**Přesměrování standardního výstupu** programu `my_command` do souboru `output.txt`.  
(Soubor `output.txt` je vytvořen. Pokud již existuje, je jeho původní obsah **smazán**.)

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

**Přesměrování standardního výstupu** programu `my_command` do souboru `output.txt`.  
(Soubor `output.txt` je vytvořen. Pokud již existuje, je výstup programu `my_command` **připojen** na jeho konec.)

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

Podobná pravidla platí pro standardní **chybový** výstup, v tomto případě se používají následující operátory:

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

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

# Spojování výstupních proudů

Standardní výstup **a** standardní chybový výstup programu `my_command` lze současně přesměrovat do souboru **output.txt**.

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

Výše uvedený postup nelze použít pro operátor `>>`.

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

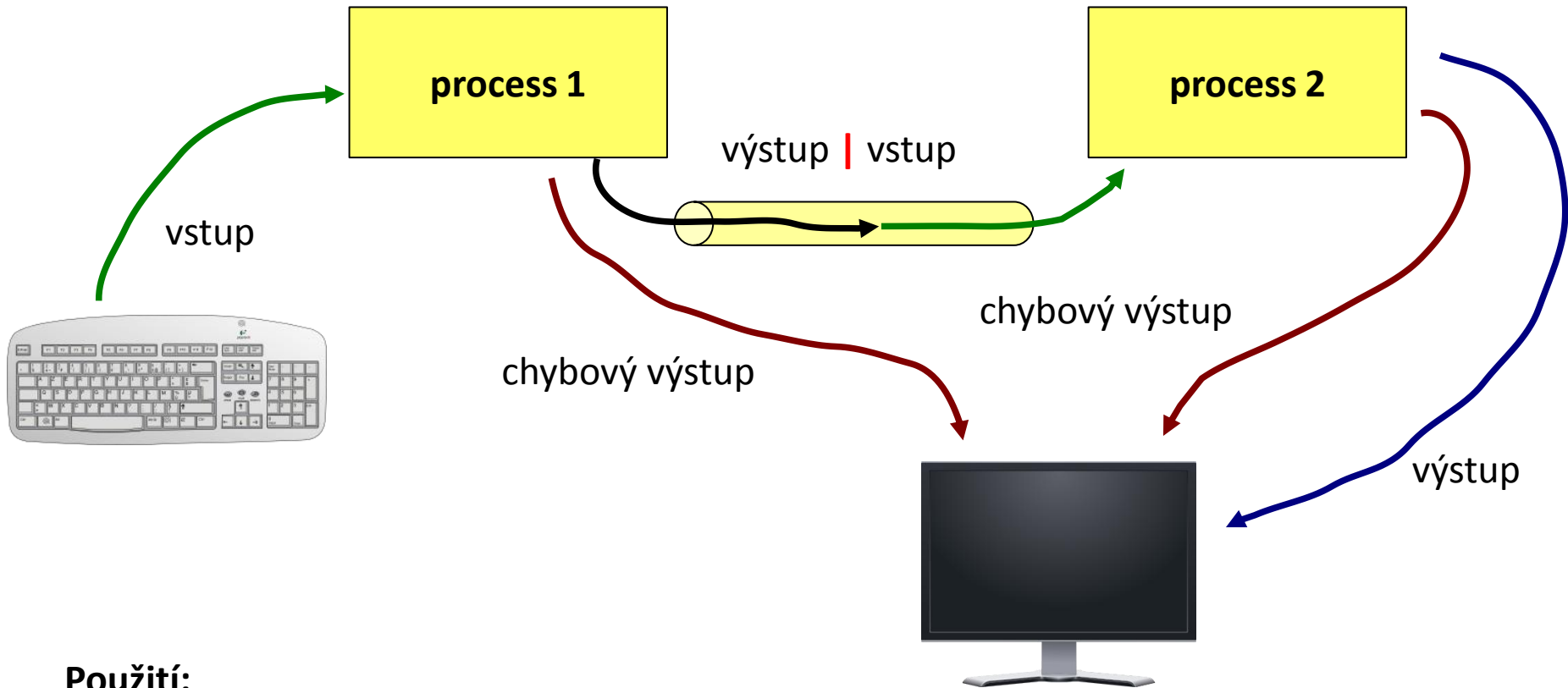
**Řešení:** Nejdříve je nutné **přesměrovat** standardní výstup a poté **spojit** standardní chybový výstup s výstupem standardním.

```
$ my_command >> output.txt 2>&1      pořadí je důležité!
```

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

# Roury (pípy)

**Roury** slouží ke spojování standardního výstupu jednoho procesu se standardním vstupem jiného procesu.

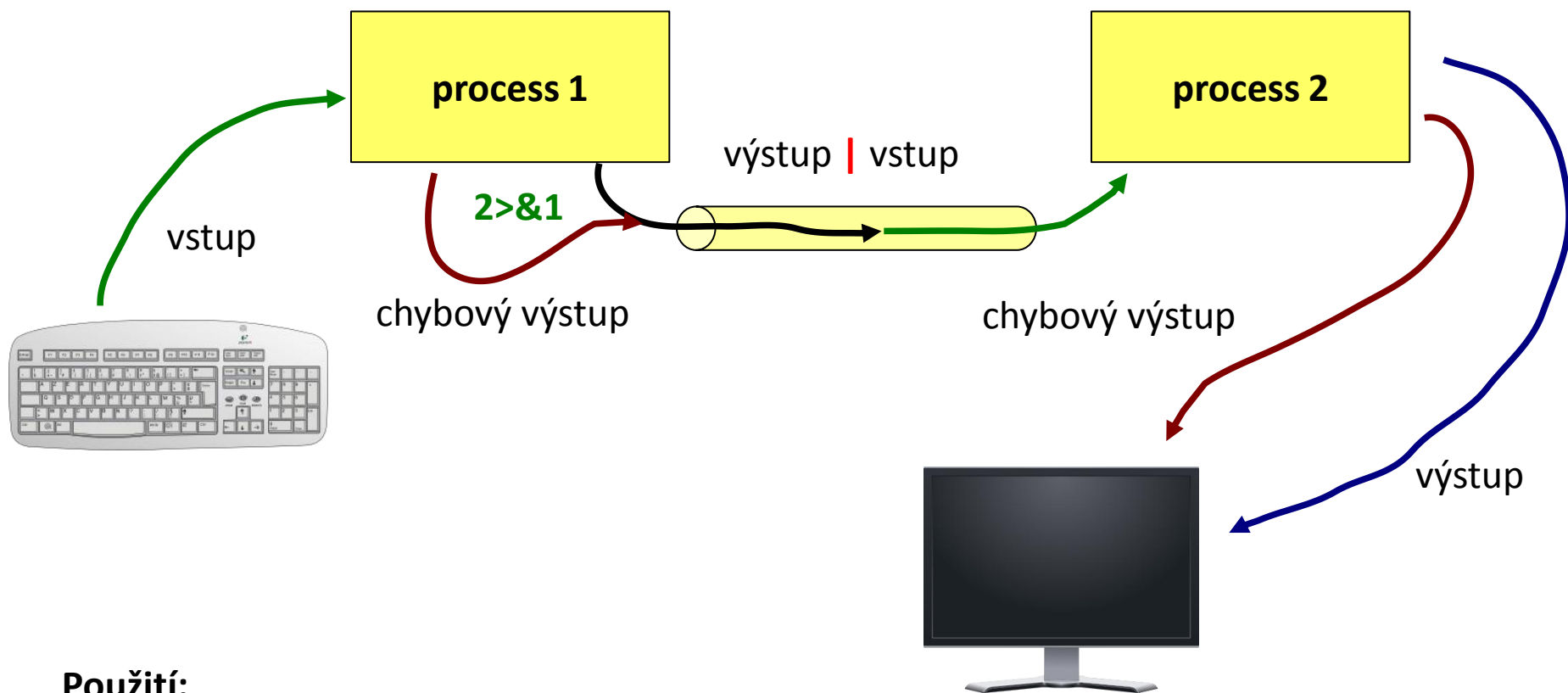


**Použití:**

```
$ command_1 | command_2
```

# Roury a chybový proud

Přenos standardního chybového výstupu přes rouru je možné provést po jeho spojení se standardním výstupem.



**Použití:**

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

# Příkazy pro cvičení

- cat** spojí obsah více souborů do jednoho (za sebe), případně vypíše obsah jednoho souboru
- paste** spojí obsah více souborů do jednoho (vedle sebe)
- wc** informace o souboru (počet řádků, slov a znaků)
- head** vypíše úvodní část souboru
- tail** vypíše koncovou část souboru

## Ukázky použití:

- \$ `cat soubor1.txt soubor2.txt`  
spojí obsah souborů soubor1.txt a soubor2.txt za sebe a výsledek vypíše na obrazovku
- \$ `paste soubor1.txt soubor2.txt`  
spojí obsah souborů soubor1.txt a soubor2.txt vedle sebe a výsledek vypíše na obrazovku
- \$ `wc soubor.txt`  
vypíše počet řádků, slov a znaků, které obsahuje soubor soubor.txt
- \$ `head -15 soubor.txt`  
vypíše prvních 15 řádků ze souboru soubor.txt
- \$ `tail -6 soubor.txt`  
vypíše posledních 6 řádků ze souboru soubor.txt

# Příkazy pro cvičení ...

Příkaz **tr** slouží k transformaci nebo mazání znaků ze standardního vstupu. Výsledek je zasílán do standardního výstupu.

## Příklady:

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

z obsahu souboru **soubor.txt** odstraní znaky "q", "w" a "e"

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

z obsahu souboru **soubor.txt** odstraní všechny bílé znaky

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

v textu zasláného příkazem echo budou nahrazeny znaky ":" znakem nového řádku "\n"

# Cvičení

1. Nalezněte všechny soubory s koncovkou **.f90** , které obsahuje adresář **/home/kulhanek/Documents/C2110/Lesson03/**, seznam souborů uložte do souboru **~/Procesy/seznam.txt**
2. Kolik řádků obsahuje soubor **seznam.txt** ?
3. Vypište první dva řádky ze souboru **seznam.txt** nejdříve na obrazovku a poté do souboru **dva\_radky.txt**
4. Vypište pouze třetí řádek ze souboru **seznam.txt**
5. V adresáři **/proc** nalezněte všechny soubory, které začínají písmeny **cpu** . Z výpisu odstraňte informace o nepovoleném přístupu přesměrováním chybového proudu do **/dev/null**
6. Vypište adresáře obsažené v proměnné **PATH**, každý na jeden řádek.
7. Aktivujte modul **vmd**. Jakým způsobem se změní obsah proměnné **PATH**?

# Závěr

---



# Závěr

- Proces je instance běžícího programu. Operační systém zajišťuje pomocí multitaskingu souběžný běh několika procesů na několika procesorech.
- Ke spuštění programu stačí uvést jméno, pokud program existuje v adresáři uvedeném v proměnné PATH. V opačném případě je nutné program během spouštění uvést včetně cesty.
- Každý proces může využít pro komunikaci s okolím tři proudy. Uživatel s těmito proudy může manipulovat. Proudů je možné přesměrovat či spojovat.
- Program je binární soubor vykonávaný přímo procesorem. Skript je textový soubor, který je vykonáván interpretem.

# Domácí úkoly

---

- Samostatně procvičujte látku z Lekce 1 až 4
- Textové editory



První písemný test v následující hodině.

# Textové editory

---

- **vi, vim, nano**
- **grafické textové editory (kwrite, gedit, kate)**



# Textové editory - instalace

Jednotlivé textové editory si vyzkoušejte ve vaší instalaci Ubuntu 14.04 LTS. Pokud nebudou dostupné, tak si je nainstalujte následovně:

```
$ 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
```

Pokud budete dotázáni, zadávejte heslo k vašemu účtu.

Ve výchozí instalaci je instalován vi editor v kompatibilním módu, který je vhodné nahradit rozšířenou verzí (vim). Instalace viz výše.

# vi/vim, nano

**Editor vi / vim** je standardním textovým editorem v operačních systémech UNIXového typu. Pracuje pouze v textovém módu a jeho používání je **netriviální**.

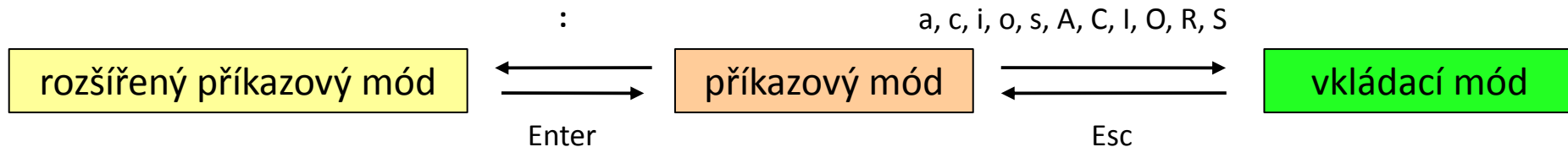
- Je vhodné se naučit, jak otevřít soubor, přejít do editačního módu, upravovat text, uložit provedené změny a editor ukončit.
- Umožňuje skriptování (použití proměnných, cyklů, polí, asociativních polí) např. pro vytvoření automatických textů z načtených dat.
- Přestože v učebně budete spouštět příkaz `vi`, automaticky se spustí program `vim` (Vi IMporoved)
- Mezi původním `vi` a `vim` je rozdíl v ovládní.

**Editor nano** je výchozím textovým editor v některých distribucích (UBUNTU).

- Méně univerzální než `vim`
- Přímočařejší ovládní

# vi – základy

## Pracovní módy editoru



## Spuštění editoru

**\$ vi** start editoru  
**\$ vi filename** start editoru a **otevření souboru** filename

## Ukončení editoru

**:q** ukončení editoru  
**:q!** ukončení editoru bez uložení změn  
**:w** uložení souboru  
**:w filename** uložení souboru po jménem *filename*  
**:wq** ukončení s uložením souboru

## Změny souboru

**i** text bude vkládán **od** pozice kurzoru  
**a** text bude vkládán **za** pozici kurzoru

**Další funkcionality – doprovodný dokument!**

# nano

## Spuštění editoru

`$ nano` start editoru

`$ nano filename` start editoru a **otevření souboru** 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
```

**Přímočařejší ovládání – menu v dolní části napovídá možné akce.** Pro volbu akce slouží kombinace nebo samostatná písmena

**^písmeno** – např. ^X je kombinace Ctrl + X

**M-písmeno** – např. M-M je kombinace Alt+M

# kwrite

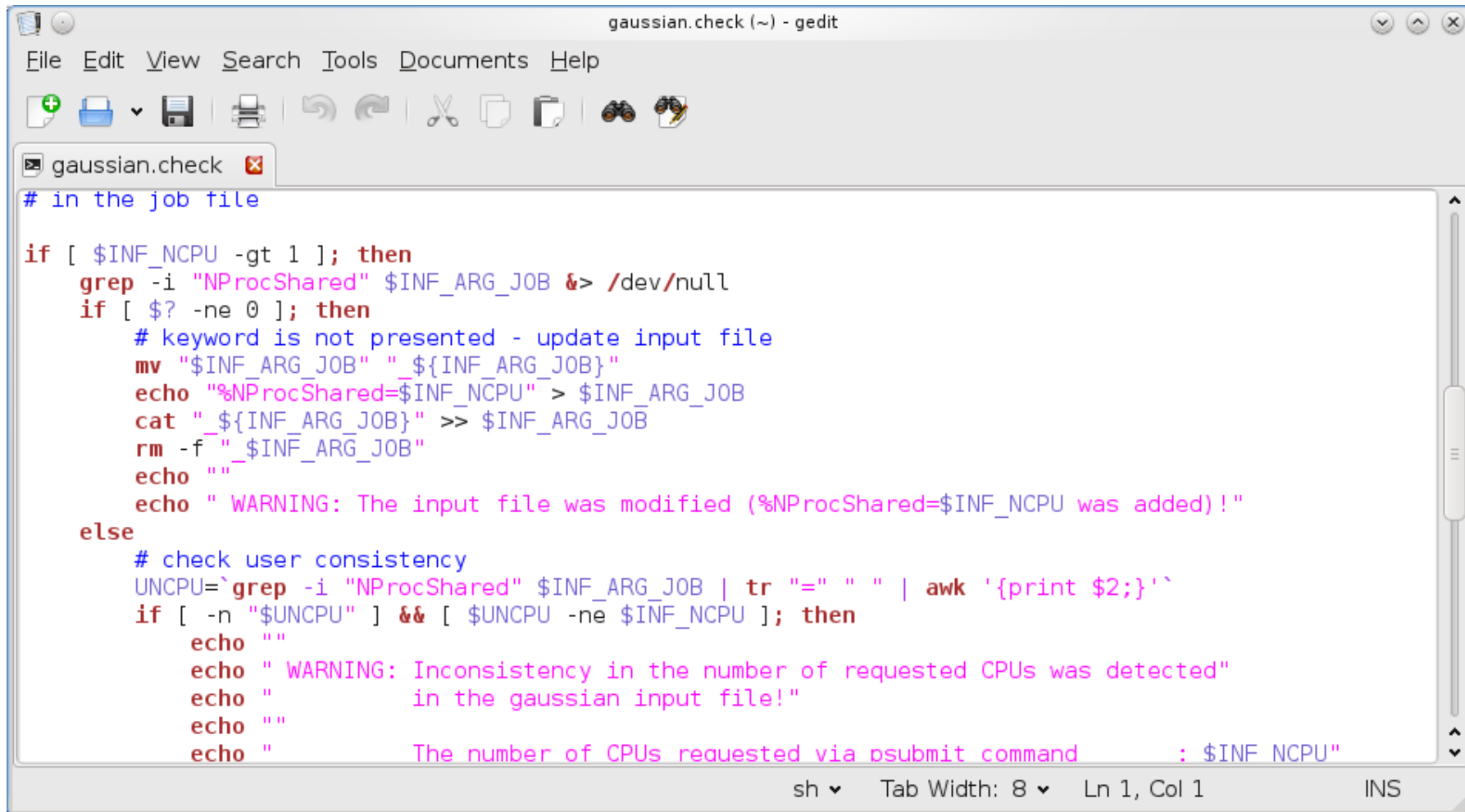
```
=====  
! Function: pmf_cvs_find_cv  
!=====  
integer function pmf_cvs_find_cv(cv_name)  
  
implicit none  
character(*) :: cv_name  
!-----  
integer :: i  
!-----  
  
pmf_cvs_find_cv = 0  
  
do i=1, NumOfCVs  
if( trim(cv_name) .eq. trim(CVList(i)%name) ) then  
pmf_cvs_find_cv = i  
return  
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  
  
!=====
```

Line: 1 Col: 1      INS LINE Fortran pmf\_cvs.f90

Rozšířená funkcionálníta: **kate**



# gedit



The image shows a window titled "gaussian.check (~) - gedit" with a menu bar (File, Edit, View, Search, Tools, Documents, Help) and a toolbar. The main text area contains a shell script for validating Gaussian input files. The script checks for the presence of "NProcShared" in the input file and updates it if necessary. It also checks for consistency between the number of CPUs requested in the input file and the number of CPUs requested via the psubmit command.

```
gaussian.check
# 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"
    fi
  fi
fi
```

sh Tab Width: 8 Ln 1, Col 1 INS

# Domácí úkoly

1. V editoru vi napište text, který bude obsahovat deset řádků. Na každém řádku budou dvě a více slov. Text uložte do souboru `mojedata.txt`
2. Příkazem `wc` ověřte, že soubor `mojedata.txt` má skutečně deset řádků.
3. Za použití `rour(y)` napište sekvenci příkazů, které na obrazovku vypíší pouze počet slov v souboru `mojedata.txt`
4. V grafickém textovém editoru (dle vašeho výběru) vytvořte soubor, který bude obsahovat deset slov, každé slovo na novém řádku. Text uložte do souboru `druha_data.txt`
5. Pomocí příkazu `paste` vytvořte soubor `vsechna_data.txt`, který bude obsahovat obsah souborů `mojedata.txt` a `druha_data.txt` vedle sebe.
6. Příkazem `wc` ověřte, že soubor `vsechna_data.txt` obsahuje právě deset řádků.
7. Soubor `vsechna_data.txt` otevřete v grafickém textovém editoru a jeho obsah ověřte vizuálně.
8. Vyzkoušejte si práci v jednotlivých textových editorech a vyberte si ten, se kterým se vám nejlépe pracuje.