B33POS: Počítačové systémy

Lekce 01. Úvod

Petr Štěpán
stepan@fel.cvut.cz

18. září, 2025

Obsah

Motivace

Trojice na sebe navazujících předmětů:

- PSY - Počítačové systémy (nový předmět)
  - Cílem je odlehčit předmětům APO, OSY a umožnit jít více do hloubky
- APO - Achitektura počítačů
- OSY - Operační systémy

Cíle těchto tří předmětů:

- naučit Vás jak funguje počítač
- pokud nevíte jak funguje počítač, pak nevíte co bude dělat Váš program
- Vaše odbornost bude vytváření programů
- efektivně používat počítač jako nástroj vývoje programů

Motivace

Naučit se dobře programovat v jazyce C.

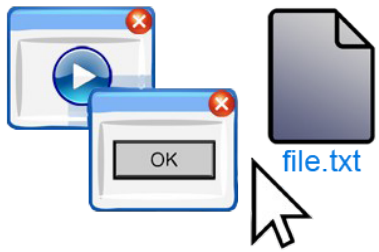
- K tomu primárně slouží Procedurální programování - PRP
- Tento předmět Vám pomůže hlouběji pochopit používají jazyka C
- Naučíte se:
  - co se děje, když se program pustí,
  - jak se data v počítači ukládají,
  - jakým způsobem se program přeloží pro procesor,
  - jak si organizovat práci na programech,
  - jak programy otestovat.

Další motivace

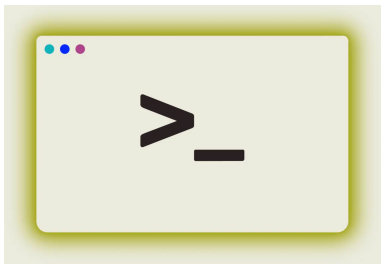
- Většina z vás používá počítač s okny a tlačítky a ovládá počítač myší a klávesnicí
- Pro vzdálenou správu počítačů jsou okna a myš neefektivní
- Naučit se ovládat efektivně počítač
- Využívat počítač jako pracovní nástroj k vývoji programů
- To co zde budeme probírat Vám pomůže programovat roboty, např. drony skupiny MRS

Ovládání počítače

Operační systém



- GUI - Graphical User Interface



- CLI - Command line interface

- Historie CLI - terminály

Řádkové shelly - CLI

Kdy je CLI daleko lepší než GUI?

- Představte si, že potřebujete převést obrázek z formátu bmp na png.
  - V GUI otevřete obrázek a exportujete ho jako png.
  - V CLI napíšete příkaz `convert obr.bmp obr.png`
- Představte si, že potřebujete převést 100 obrázků z formátu bmp na png.
  - V GUI otevřete 100× obrázek a 100× exportujete ho jako png.
  - V CLI napíšete skript s konstrukcí for a stisknete 1× enter

Řádkové shelly - CLI

Řádkový shell je prostředí pro ovládání počítač textem

- Windows – cmd, powershell
- Posixové systémy – systémy splňující standard POSIX – Linux, MacOS, Free BSD a mnoho dalších
- Shell funguje buď interaktivně - uživatel zadává příkazy, které se okamžitě vykonávají
  - Po stisknutí ENTER se příkaz provede, pokud není zadáno, že pokračuje na další řádce
- Druhá možnost je vytvářet skripty - textové programy shellu o více řádkách, které mohou definovat funkce a složité vlastnosti

Obsah

BASH

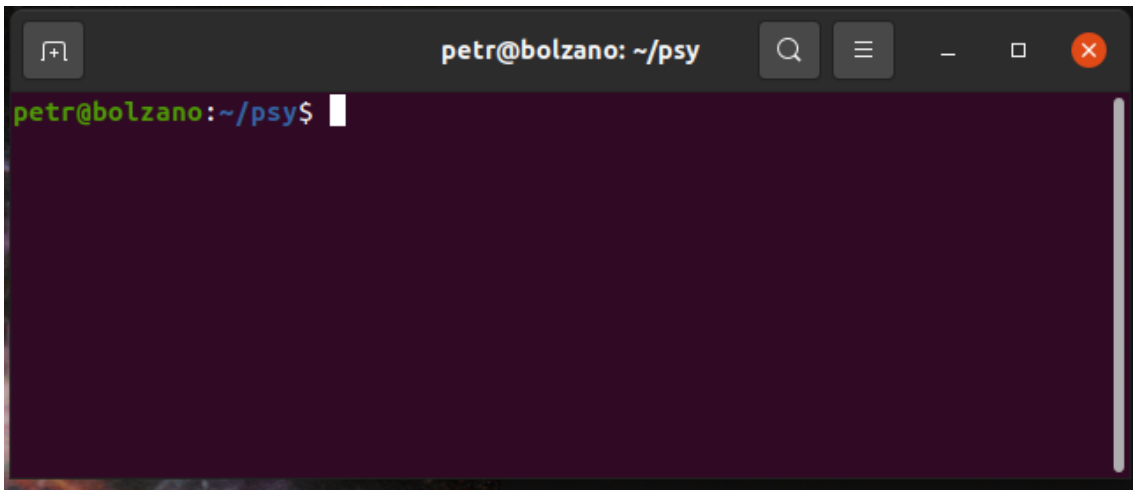
BASH

- jeden z prvních shellů byl Bourne shell (/bin/sh) z roku 1976 od Stephena Bourne
- Bash je zkratka Bourne again shell
- vytvořen v roce 1989 Brianem Foxem jako 100% svobodný program (oproti firemním Unixům a i Bourne shellu)
- paralelně vznikly další shelly, např. csh - ovlivněný jazykem C, nebo zsh - odvozeno od jména profesora Zhong Shao, který vedl Paula Falstada, autora zsh
- ostatní shelly mají malé odchylky

Prompt – Výzva

Po spuštění terminálu uvidíte něco podobného:

- Blikající kurzor označuje tzv. prompt česky také výzvu
- Text před kurzorem zobrazuje konfigurovatelné informace o pracovním adresáři, názvu počítače, jména uživatele, apod.
- Text výzvy lze konfigurovat v proměnné PS1, například
  `PS1="\\u@\\h: \\W\\$"`



Shell – základní funkce

- Umožňuje spouštět programy
- Umožňuje přesměrovat vstup a výstup programu
- Umožňuje zjistit, zda byl program úspěšný
- Definuje příkazy, které ovlivňují vykonávání programů a příkazů
- Umožňuje používat proměnné, do kterých se ukládají hodnoty - výstupy programů, konstanty, operace s řetězci, výpočty

Ukázka příkazů

Vyzkoušejte si tyto příkazy napsat v terminálu:

- date
- echo hello
- echo "Hello world"
- echo $PATH
- which echo

Příkaz `echo`

Příkaz `echo` vytiskne zadané argumenty.
Proč se při příkazu `echo Hello  world` vytiskne pouze s jednou mezerou `Hello world`?

- mezera, nebo více mezer oddělují parametry
- uvozovky `echo "Hello  world"` vytiskne zadaný počet mezer, protože uvozovky vytvoří z řetězce jeden argument
- apostrofy `echo 'Hello  world'` vytiskne přesně zadaný řetězec
- výraz `$jméno_proměnné` je nahrazen hodnotou uloženou v proměnné `jméno_proměnné`
- `echo '$A='"$A"` uvozovky nezabrání nahrazení jména proměnné její hodnotou, ale apostrofy ano

Proměnné

Můžete vytvořit proměnnou s libovolným jménem, začínající písmenem, může obsahovat i číslice a znak _.
Malá a velká písmena se rozlišují a jedná se o různé proměnné.

- proměnná obsahuje jen řetězec
- pokud obsahuje číslo, pak se na něj nahlíží jako na řetěz, pokud se nepoužije speciální konstrukce
  - `A=10` # hodnota 10
  - `B=a${A}b` # hodnota a10b
  - `C=$A+1` # hodnota 10+1
  - `D=$(( A+1 ))` # hodnota 11

POZOR na mezery

- `A = 10` # chyba spuštění programu A
- `A= date` # chyba nenastaví proměnnou A a zavolá program `date`
- `let A=5 + 6` # chyba špatný operand
- `let "A=5 + 6"` # správně A=11

Obsah

Soubory

Základní funkce shellu je usnadnit manipulaci se soubory.

- Potřebujeme jednoznačný identifikátor určující soubor
- Abychom mohli mít soubory se stejným jménem, je nutné aby byly v jiném adresáři
- Jednoznačný identifikátor souboru je cesta k adresáři a jméno souboru - namespace
- cesta k adresáři je určena:
  - absolutně od kořenového adresáře
  - relativně od aktuálního pracovního adresáře

Jména Soubory

- Původně byly názvy souborů velmi omezené, např. MS-DOS umožňoval mít maximálně 8 znaků písmena, číslice nebo _ pak následovala . a po ní max 3 znaky přípony (označení těchto jmen 8.3)
- V současné době všechny operační systémy umožňují použít v názvu souborů téměř všechny znaky a délka je omezena buď na 255/260 bajtů, nebo je i neomezena.
- Pro zadání jména souborů s problematickými znaky (mezera, hvězdička, otazník, lomítko, apod.) je nutné před citlivý znak zadat znak
  - například: `mezera\ a\ hvezdicka\*.txt` je název souboru s mezerou a hvězdičkou

Jména Soubory

- Soubory, které začínají znakem `.` jsou takzvané skryté, tedy pro jejich výpis je potřeba použít přepínač.
  - smyslem skrytých souborů je zpřehlednit výpis uživatelských souborů a oddělit soubory, které konfigurují systém, nebo jiné programy
  - např. soubor `.bashrc` obsahuje příkazy, které se provedou vždy při spuštění programu bash
  - pokud chcete vynutit spuštění změněného inicializačního souboru, můžete použít příkaz `source ~/.bashrc`

Absolutní cesta

- Kořenový adresář
- Plně přesně určuje
  - POSIXových systémech se značí pomocí lomítka
  - V systému MS Windows je více kořenových adresářů rozlišených písmenem, znakem dvojtečka a zpětným lomítkem
- Absolutní cesta začíná označením kořenového adresáře, dále může obsahovat posloupnost jmen adresářů oddělených lomítkem (POSIX) nebo zpětným lomítkem (MS Windows) zakončených jménem souboru
- Příklady absolutní cesty v POSIX:
  - `/home/petr/pocitacove_systemy/read.me`
  - `/read.me`
  - `/lib/../home/petr/pocitacove_systemy/read.me`

Relativní cesta

- Pracovní adresář
  - jeden adresář je vybraný jako pracovní
  - pracovní adresář lze změnit příkazem cd (change the working directory)
- Relativní cesta může obsahovat posloupnost jmen adresářů oddělených lomítkem (POSIX) nebo zpětným lomítkem (MS Windows) zakončených jménem souboru
- Příklady relativní cesty v POSIX:
  - `pocitacove_systemy/read.me`
  - `read.me`
  - `../../home/petr/pocitacove_systemy/read.me`

Pracovní adresář

V každém adresáři existují dva skryté speciální adresáře:

- Adresář . – odkazuje na současný pracovní adresář
- Adresář .. – odkazuje na nadřazený adresář (jedině pro kořenový adresář odkazuje stejně jako . na současný pracovní adresář)

Příkazy týkající se pracovního adresáře:

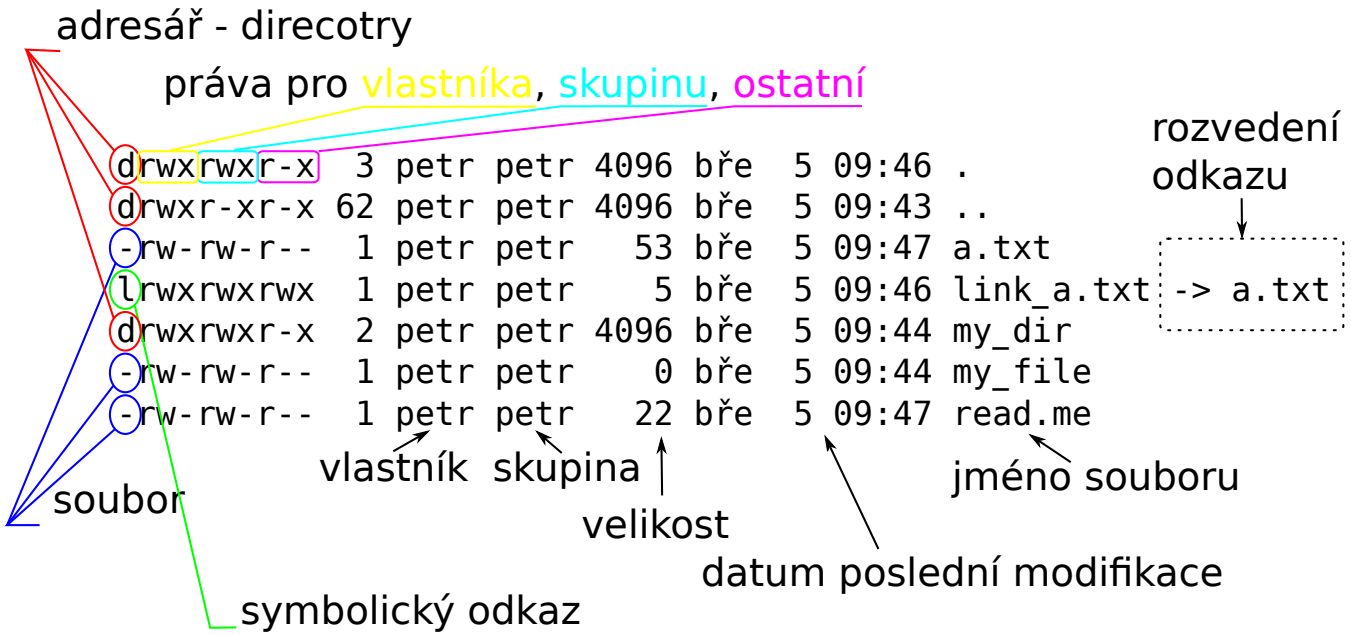
- pwd – print working directory
- cd – change directory
  - `cd` – přepne se na domácí adresář
  - `cd ~/..` – přepne se do adresáře zadaného cestou z domácího adresáře
  - `cd -` – přepne se do předchozího adresáře

Práce s adresáři

- mkdir
  - vytvoří adresář buď podle zadané absolutní cesty, nebo relativně k pracovnímu adresáři
  - `mkdir psy` - vytvoří adresář psy pod pracovním adresářem
  - `mkdir -p ../psy` - pokud adresář `../psy` existuje, tak nezpůsobí chybu
  - `mkdir /tmp/psy` - vytvoří adresář psy v adresáři `/tmp`, pokud by adresář `/tmp` neexistoval pak se nevytvoří ani adresář `psy`
- rmdir - remove directory
  - odstraní prázdný adresář
  - jinak se musí nejdřív smazat soubory a podadresáře
  - alternativně `rm -r adr`, smaže rekurzivně všechny soubory i podadresáře (i s jejich soubory) v adresáři `adr` a nakonec smaže i adresář `adr`.

Obsah adresáře

- ls
- ls cesta
  - obsah adresáře
  - přepínače -l, -s
  - příklad:



Práva k souborům

Práva pro vlastníka souboru, skupinu vlastníků, ostatní uživatele:

- Význam práv k souborům
  - r čtení obsahu souboru
  - w zápis změna souboru
  - x soubor lze spustit (i pro textové soubory)
- Význam práv k adresářům
  - r čtení obsahu adresáře
  - w zápis vytváření, mazání, přejmenovávání souborů nebo podadresářů
  - x vstup do adresáře, důležité i pro čtení a zápis souborů z adresáře

Speciální práva k souborům

Práva s pro uživatele, skupinu a ostatní:

- suid – spuštění programu s právy vlastníka souboru místo s právy aktuálního uživatele (využití např. passwd získá root práva a umožní změnit uživateli heslo, na což má práva jen root)
- sgid – spuštěné soubory získají skupinová oprávnění souboru, pro adresáře platí, že všechny nově vytvořené soubory a podadresáře získají skupinová práva daného adresáře
- sticky – v současnosti má význam pouze pro adresáře, zbraňuje smazat soubory v adresáři, které nepatří vlastníkovi (w bit dává plná práva k adresáři, vytvářet i mazat; používá se pro /tmp adresář).

Nastavení práv k souborům

- chmod
  - octalové číslo r - 4, w - 2, x - 1
    - číslo obsahuje tři cifry UGO – U práva pro vlastníka, G práva pro uživatele, který není vlastník, ale patří do stejné skupiny, O práva pro ostatní
    - pro speciální práva přidáme 4 číslo SXXX – S - 1 sticky, S - 2 sgid, S - 4 suid
  - nebo pomocí a+x, nebo o-w
- chown – změna vlastníka a skupiny
  - změna vlastníka a skupiny změní vlastně i práva k souborům
  - můžete změnit vlastníka souborů

Manipulace se soubory

- mv - move files, cp - copy file
  - cp i mv stejný formát zadání souborů, jediný rozdíl, že při mv se původní soubory smažou
  - `mv lecture_01.pdf ../psy/lectures/psy_lecture_01.pdf` - přesune soubor `lecture_01.pdf` do adresáře `../psy/lecturesa` přejmenuje ho na soubor `psy_lecture_01.pdf`
  - `cp lecture_01.pdf lecture_02.pdf ../psy/lectures` - překopíruje oba soubory `lecture_01.pdf` a `lecture_02.pdf` do adresáře `../psy/lectures`

Manipulace se soubory

- rm - remove file
  - nenávratně smaže soubor nebo více souborů, nebo adresáře
  - příkaz selže, pokud k tomu uživatel nemá práva
  - přepínač -r rekurzivně maže i adresáře a jejich soubory
- touch - touch file
  - nastaví čas modifikace na teď
  - pokud soubor neexistuje, tak ho vytvoří jako prázdný soubor

Manuál k příkazům

- man ls
  - podrobnější než ls –help
  - přehled všech přepínačů programu
  - přehled všech možných výsledků programu, v podstatě přehled všech možných chybových hlášení
  - odkazy na související příkazy
  - někdy příklady použití zadaného příkazu

Obsah

Programovací jazyk C

Nejjednodušší program "Hello FEL"

```c
#include <stdio.h>
int main() {
    printf("Hello FEL!\n");
    return 0;
}
```

- Program vytiskne na výstup větu `Hello FEL!`
- Program končí příkazem return 0
  - Proč 0, proč ne třeba 1?
  - Co se s touto hodnotou stane?

Výsledek programu

Každý program má jako výsledek jedno celé číslo.

- výsledek posledního programu se uloží do proměnné $?
  - hodnota 0 - True (Pravda) - program skončil úspěšně
  - hodnota rozdílná od 0 - False (Nepravda) - Číslo chyby, která nastala
  - Program v jazyce C/C++, návratová hodnota funkce main
  - příklad: `mv nemam_prava.txt nove_jmeno.txt; echo $?`
  - příklad: `mv nemam_prava.txt nove_jmeno.txt; if [ $? != 0 ]; then echo fail; fi`

Podmíněné vykonání příkazů

- konstrukce
  `if podminka; then prikazy; fi`
  - povinná klíčová slova `if`, `then`, `fi`
  - volitelná klíčová slova `else`, `elif`
  - znak `;` může být nahrazen novou řádkou
- konstrukce
  `if podminka; then prikazy else prikazy; fi`
- konstrukce
  `if podminka; then prikazy elif podminka; then prikazy; else prikazy; fi`

Podmínky

Podmínka je libovolný program, jehož návratová hodnota určuje zda je podmínka splněna

- například program `[`
  - i když to tak nevypadá, tak existuje program `[`, který vyhodnotí zadané parametry a podle toho nastaví výslednou hodnotu
  - protože jde o program musí být výrazy uvnitř odděleny mezerou
  - lze testovat:
    - zda existuje soubor, zda se jedná o adresář, symbolický odkaz
    - porovnávat hodnotu proměnné, či testovat, zda je proměnná definována
    - porovnávat numerickou hodnotu proměnné

Podmínky

Podmínky souboru

- -e soubor existuje
- -f jedná se o regulární soubor
- -s soubor nemá nulovou velikost
- -d jedná se o adresář
- -p jedná se o rouru
- -L jedná se o symbolický odkaz
- -r má práva ke čtení
- -w má práva k zápisu
- -x má práva ke spuštění
- -nt první soubor je novější než druhý
- -ot první soubor je starší než druhý

Porovnání celých čísel

- -eq čísla jsou shodná
- -ne čísla nejsou shodná
- -gt první číslo je větší
- -ge první číslo je větší nebo rovno
- -lt první číslo je menší
- -le první číslo je menší nebo rovno

Porovnání řetězců

- $=$ řetězce jsou shodné
- $==$ řetězce jsou shodné
- $!=$ řetězce nejsou shodné
- $<$ abecední porovnání
- $>$ abecední porovnání
- -z řetězec má nulovou délku
- -n řetězec má nenulovou délku

Podmínky

Spojení více podmínek

- -a logický součin – and
- -o logický součet – or
- ! negace výrazu

Existují další alternativy

- operátor `[[ výraz ]]`
- operátor `(( výraz ))` výrazy podobné jazyku C

Výsledek programu

Logické kombinace příkazů bez použití `if` se zkráceným vyhodnocením

- logické and &&, logické or ||
- příklad: `mv nemam_prava.txt nove_jmeno.txt || echo fail`
- příklad: `mv nemam_prava.txt nove_jmeno.txt && echo soubor je prejmenovan`

Proměnné

Hodnoty proměnné lze měnit pomocí následujících operací

- `${#jmeno}` délka řetězce proměnné jmeno
- `${jmeno:poc}` podřetězec proměnné jmeno začínající znakem pos do konce řetězce
- `${jmeno:poc:delka}` podřetězec proměnné jmeno začínající znakem pos o délce delka
- `${jmeno#retez}` odstraní nejkratší výskyt řetězce retez ze začátku řetězce proměnné jmeno
- `${jmeno##retez}` odstraní nejdelší výskyt řetězce retez ze začátku řetězce proměnné jmeno
- `${jmeno%retez}` odstraní nejkratší výskyt řetězce retez z konce řetězce proměnné jmeno
- `${jmeno%%retez}` odstraní nejdelší výskyt řetězce retez z konce řetězce proměnné jmeno

Jak může být nejkratší a nejdelší výskyt podřetězce v řetězci?

Neurčitý řetězec

Takzvaný globbing umožňuje využívat speciální znaky

- * reprezentuje libovolný řetězec
- ? reprezentuje jeden libovolný znak
- $[a - z]$ reprezentuje jeden znak v zadaném rozsahu

Výsledkem globbingu je nalezení všech souborů, které odpovídají zadaným podmínkám.

- `*.txt` všechny soubory, které končí na řetězec `.txt`
- `*[0-9]*` všechny soubory, které ve svém názvu obsahují alespoň jednu číslici

Neurčitý řetězec

Neurčitý řetězec může být nahrazen i podřetězcem v proměnné:

- `${jmeno##*/}` odstraní jméno adresáře z proměnné jmeno
- `${jmeno%.txt}.bak` změní koncovku u jména souboru z proměnné jmeno z `.txt` na `.bak`

Cykly

For cyklus má podobnou strukturu jako v jazyce Python

- for i in seznam; do prikazy; done
  - klíčová slova jsou `for`, `do`, `done`
  - seznam může být libovolný řetězec, kde jsou položky odděleny mezerou, tabulátorem, nebo znakem nová řádka
  - pro seznam můžeme použít i globbing

Příklad:

- `for i in *.txt; do cp $i zaloha\$i; done`
- zkopíruje všechny soubory s koncovkou `.txt` do adresáře zaloha

Cykly

- `for i in *.txt; do cp $i zaloha\$i; done`
  - středník ; nahrazuje znak nový řádek
  - středník umožňuje zapsat více řádkovou konstrukci na jeden řádek

For cyklus bez středníků

```
for i in *.txt
do
  cp $i zaloha\$i
done
```

Cykly

Příští týden se podíváme na další konstrukce bash.

Testovací kvíz

Kolik jste toho zatím pochopili?

A Vše bez problému.

B Téměř vše.

C Téměř nic.

D Vůbec nic.