# Vývoj

V roce 1983 byl založen projekt GNU, jehož cílem bylo vytvořit nový operační systém unixového typu, který by byl složen jen ze svobodného software. Otcem a zakladatelem projektu je Richard Matthew Stallman. Za tímto účelem sepsal Stallman novou licenci GNU GPL, pod kterou jsou šířeny všechny části systému GNU.

Během necelých deseti let se GNU stal zcela použitelným systémem kompatibilním s komerčními unixy. K dispozici byly všechny důležité aplikace, systémové knihovny, překladač GCC, textový editor a další. Chybělo jen jádro, které by zajistilo samotný běh systému a komunikaci s hardware. Proto byl v roce 1990 zahájen vývoj jádra Hurd.

Na druhé straně světa v Helsinkách ale v roce 1991 začal finský student Linus Torvalds pracovat na vývoji vlastního unixového jádra. Ve škole se totiž seznámil s unixovým operačním systémem Minix. Linusovi se unixový koncept velmi líbil a chtěl mít na svém PC také jeden.

Minix byl ovšem až příliš jednoduchý a navíc k němu nebylo možno získat zdrojové kódy. „Velké unixy” byly zase velmi drahé a student si je nemohl dovolit. Proto se Linus rozhodl jít cestou nejmenšího odporu a napsat si vlastní operační systém, který by se podobal Minixu a byl provozovatelný na běžném PC. Do Usenetu Linus poslal legendární zprávu:

Hello everybody out there using minix -  
I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones.

První verze linuxového jádra (0.01) byla vydána na Internetu 17. září 1991, další následovala v říjnu téhož roku. Od té doby se na tomto projektu podílely tisíce vývojářů z celého světa. Model vývoje linuxového jádra a podobného softwaru je krásně popsán v eseji Erica S. Raymonda Katedrála a bazar.

Před vydáním verze 0.01 Linus naimplementoval dostatek POSIXových systémových volání, aby bylo možné spustit shell GNU Bash. Díky tomuto základnímu prostředí se vývoj mohl rozběhnout mnohem rychleji. Původně bylo pro nastavení, kompilaci a instalaci Linuxu potřeba mít funkční systém Minix. První verze Linuxu též vyžadovaly pro spuštění z disku, aby běžel jiný operační systém, ale brzy vznikly nezávislé spouštěče, z nichž nejznámější bylo LILO. Linuxový systém zanedlouho předběhl Minix co do funkčnosti. Torvalds a další vývojáři uzpůsobili jádro, aby lépe spolupracovalo s komponentami z projektu GNU a s dalšími uživatelskými programy, aby tak vznikl kompletní, plně funkční, svobodný operační systém.

Jelikož byl vývoj Linuxu mnohem úspěšnější a rychlejší jak vývoj Hurdu, poměrně brzy došlo k logickému kroku a operační systém GNU se začal používat společně s jádrem Linux.

Tím vznikl výsledný produkt se správným názvem GNU/Linux. Velmi často se používá jen krátké označení Linux, ale podstatná část systému pochází právě z projektu GNU, který kromě Linuxu může běžet s jádry Hurd, Minix, FreeBSD a dalšími.

Velmi často je také chybně uváděno, že Linux je součástí GNU nebo naopak. Jedná se stále o dva zcela nezávislé projekty, které ovšem dohromady tvoří nejpoužívanější kombinaci, kterou můžeme ve světě svobodného software nalézt.

Logem a maskotem Linuxu je tučnák Tux vycházející z obrázku Larryho Ewinga z roku 1996. Kromě toho existují i jiná, méně známá zpodobnění. Jméno „Linux“ nevytvořil sám Torvalds, ale Ari Lemmke, který pracoval na helsinské univerzitě jako správce FTP serveru ftp.funet.fi, kde byla uveřejněna první verze Linuxu. Torvalds navrhoval jméno „Freax“ jako free (svobodný) + freak (blázen) + x (unixový systém), ale to se Lemmkemu nelíbilo a na FTP serveru vytvořil adresář „Linux“ jako „Linusův Minix“. Toto jméno se později stalo ochrannou známkou

## Současnost

Spojením GNU, Linuxu a dalších projektů vznikají takzvané distribuce, jež jsou kompilací jednotlivých částí, a tvoří tak komplexní operační systém. Nikdy proto doopravdy nepracujeme jen s Linuxem nebo GNU, ale s konkrétní distribucí.

Hurd není stále dokončen a jeho moderní návrh velmi komplikuje vývoj a testování. Práci na Hurdu se také věnuje řádově méně programátorů než práci na Linuxu. Přesto lidé z GNU stále plánují jeho dokončení.

Linus dnes žije v Portlandu v Oregonu a je stále správcem vývoje a ponechává si výhradní právo rozhodování ve všech otázkách kolem projektu. Aktuální jádra obsahují přibližně 2 % jeho kódů.

## Distribuce

Linux se převážně používá jako jádro GNU/Linuxových distribucí (zkráceně též distro). Ty jsou sestavovány jednotlivci, týmy dobrovolníků, ale i komerčními firmami za cílem finančního zisku. Typická distribuce zahrnuje jádro, další systémový a aplikační software spolu s prostředky, jak celý systém nainstalovat na počítač. Různé distribuce byly vyvinuty k různým účelům, mezi něž patří možnost mít hotový systém připravený k použití, lokalizace, podpora určité počítačové architektury, použití v real-time a v embedded systémech. Některé záměrně obsahují výhradně svobodný software. V současné době existuje kolem 450 různých distribucí.

Nejpoužívanější distribuce:

* Debian - Servery, desktopy
* Ubuntu - Dekstopy, servery minoritne
* Gentoo - Desktopy, server, … Kompilace jadra (je v ustavu)
* Suse - Desktopy i servery. Existujekomerční suse linux Enterprise desktop (server) SLED (SLES)
* Fedora - Testovaci verze Redhatu
* RedHat - komerční, jak servery, tak desktopy

# Správa uživatelů

## Uživatelé

Unix je víceuživatelský systém. Každý uživatel musí mít uživatelský účet.

Při přihlašování do systému *se heslo nezobrazuje* a *jsou rozlišována malá a velká písmena*. Každý program běží pod určitým uživatelem, každý soubor je vlastněn některým uživatelem. Pro snadnou identifikaci v programech má každý uživatel přiřazen identifikátor – číslo – tzv. **UID**

Pro snazší přiřazování práv uživatelům jsou zavedeny skupiny. Je-li určitá činnost pro skupinu povolena pro skupinu, je povolena pro každého uživatele, který je jejím členem. Uživatel může být členem více skupin, ale jednu má vždy nastavenu jako základní. Podobně jako uživatel má i skupina svůj číselný identifikátor, označuje se jako **GID.** Seznam skupin je v souboru **/etc/group**

Uživatel, může své GID změnit příkazem ***newgr*** na kteroukoli jinou skupinu, jíž je členem. Aktuální UID a GID lez zjistit příkazem ***id***. Toto UID a GID budou mít všechny soubory, které uživatel vytvoří a procesy, které spustí

Specifickým postavení mezi uživateli má uživatel s **UID 0**,bývá označován jako **superuživatel – root**

* je standardně oprávněn ke všem činnostem
* některé privilegované úlohy smí provádět pouze uživatel root

Tato absolutní moc je určitým bezpečnostním rizikem, doporučuje se proto pod tímto uživatelem provádět pouze ty úlohy, které nikdo jiný provádět nesmí (administrace systému) a pro běžnou práci používat standardní uživatelské jméno.

Databáze uživatelů je uložena v souboru **/etc/passwd**

Každý uživatel je na zvláštním řádku, každý řádek obsahuje několik položek oddělených dvojtečkou.

Část souboru může vypadat např. takto:

root:x:0:0::/root:/bin/bash

bin:x:l:l:bin:/bin:

interbase:x:45:45:Firebird server:/home/wwws:

mike:x:1000:100:Michal Kubecek:/home/mike:/bin/bash

honza:x:1001:100:Jan Novak:/home/honza:/bin/tcsh

honza:x:1001:100:Jan Novak:/home/honza:/bin/tcsh

Uziv\_Jmeno:x:UID:GID:Plne Jmeno:Home\_DIR:Bash

* ***usermod*** (oprava údajů uživatele)
* ***chfn*** (změna plného jména)
* ***chsh*** (změna shellu).

Na starších systémech obsahovala druhá položka heslo zakódované pomocí algoritmu *DES*. Kódování je jednosměrné, tj. není možné ze zakódovaného hesla odvodit heslo nezakódované. Bohužel však tento algoritmus umožňoval hesla pouze do délky osmi znaků a při rostoucí výpočetní síle počítačů už je dnes možné taková hesla najít metodou pokusu a omylu (hrubou sílou / prostým vyzkoušením všech kombinací). Proti tomuto nebezpečí se používají dvě metody. Za prvé se přešlo na používání algoritmu *MD5*, který umožňuje delší hesla. Za druhé se používají *shadow passwords*: ve druhé položce **/etc/passwd** je pouze znak „**x**“ a hesla (zašifrovaná) jsou v souboru **/etc/shadow**. Na rozdíl od souboru **/etc/passwd**, který musí být čitelný pro všechny uživatele (zobrazování jména usera místo UID při výpisu souborů, …), je **/etc/shadow** čitelný pouze pro uživatele root. Typický řádek souboru /etc/shadow pak vypadá například takto:

**Petr:$l$/gbfw9my\$UlQTCONLgm9mGy2XnjPlU/:11686:0:99999:7:::**

* první položka obsahuje **login name**
* druhá zašifrované **heslo**
* další položky pak informace o **stáří hesla** a parametrech účtu (lze je měnit příkazem ***chage****)*

Mezi ně patří údaje:

* po jaké době smí/musí být heslo změněno
* jak dlouho před vypršením hesla bude uživatel varován
* jak dlouho po vypršení hesla bude zablokován účet

Druhou metodou, jak zablokovat účet, je přepsat heslo v **/etc/shadow** např. na „**\***“. Tento řetězec totiž nemůže být v žádném případě zašifrovaným heslem, takže se uživatel nemůže nalogovat. Naopak, pokud změníte druhou položku na prázdnou (mezi dvojtečkami nebude nic, ani mezery), může se uživatel nalogovat bez hesla. Nezapomeňte po editaci zkontrolovat, že soubor /**etc/shadow** není čitelný pro nikoho jiného než pro uživatele **root**. Ještě vhodnější je editovat jej pouze příkazem „**vipw -s**".

Heslo uživatele lze změnit příkazem ***passwd***. Je-li spuštěn obyčejným uživatelem, změní jeho heslo (nejdříve se zeptá na stávající heslo, potom dvakrát na nové). Je-li spuštěn uživatelem root, lze jako argument zadat jméno uživatele a příkaz změní jeho heslo. Je-li příkaz spuštěn uživatelem root, na staré heslo se neptá, a to ani v případě, že mění heslo uživatele root.

## Skupiny

Databáze skupin je uložena v souboru **/etc/group**.   
Formát souboru je podobný:

root::0:root

bin::1:root,bin,daemon

daemon::2:root,bin,daemon

sys::3:root,bin,adm

floppy::11:root,mike

users::100:

vedeni::101:jirka,honza

* první položka obsahuje **jméno skupiny**
* třetí **GID skupiny**
* čtvrtá **jména členů** oddělená čárkami (členové, kteří mají skupinu jako výchozí, nemusejí být uvedeni)

Ve druhé položce může být heslo, které je potřeba zadat pro přihlášení ke skupině. Hesla opět bývají uložena ve stínovém souboru, jeho jméno je tentokrát /etc/gshadow. K editaci souboru **/etc/group** se používá např. příkaz ***vigr***, k editaci souboru **/etc/gshadow** příkaz „***vigr -s***".

Přidání uživatele je komplexní úkol: kromě samotného zápisu do zmíněných souborů je třeba vytvořit domácí adresář uživatele, mailbox, nastavit vlastníky a přístupová práva atd. Všechny tyto činnosti zajistí příkaz ***useradd***, jemuž lze zadat parametry nového uživatele pomocí argumentů. Analogický příkaz ***groupadd*** slouží k vytvoření skupiny. Příkaz ***usermod*** umožňuje měnit parametry existujícího uživatele. Jednotlivé distribuce obvykle obsahují i komfortnější (grafické) nástroje pro správu uživatelů, případně tyto funkce obsahuje příslušný univerzální konfigurační nástroj.

Často používaným „programem“ (utilitou) je příkaz ***su***, který standardně spustí nový shell s UID uživatele, jehož jméno je zadáno jako argument. Je-li příkaz spuštěn bez parametru, spustí shell s UID superuživatele (odtud i název příkazu). Je-li, su spouštěn jiným uživatelem než root, je třeba zadat heslo příslušného uživatele. Parametrem ***„-c***“ lze zadat příkaz, který se má pod zvolenou identitou spustit (místo shellu).

# Správa procesů

Každý program v Linuxu/Unixu se po spuštění stává procesem, který je plně v jádra (kernelu). Kernel danému programu přidělí na základě výše jeho priority část paměťového prostoru a případně další systémové prostředky, je-li to vyžadováno. Protože Linux je operační systém, který plně využívá multitasking, bude se každý proces snažit o získání co největšího procesorového času. Zde bude potom hrát roli výše priority procesu. Systém vnitřně využívá řadu front a semaforů, kterými řídí procesy a stará se o přidělování procesoru a dalších systémových prostředků.

Díky multitaskingu se uživatelům zdá, že v systému běží spousta procesů zároveň. Ano, běží, ale procesor zpracovává v daném čase vždy jen jeden proces (bavím se o jednoprocesorovém PC).

**Příkaz pro** výpis procesů v systému se nazývá **ps** (process status).

libor@mail:/root$ ps  
 PID TTY TIME CMD  
17346 pts/0 00:00:00 su  
17347 pts/0 00:00:00 bash  
17364 pts/0 00:00:00 ps

Příkaz ps bez parametrů vypíše pouze procesy uživatele, který tento příkaz spustil a pouze ty procesy, které byly spuštěny ze stejného terminálu jako samotný příkaz ps.

Chceme-li vypsat všechny procesy běžící v systému, tak použijeme následující parametry:

libor@mail:~$ ps ax  
 PID TTY STAT TIME COMMAND  
 1 ? Ss 0:01 init [2]  
 2 ? S 0:00 [migration/0]  
 3 ? SN 0:01 [ksoftirqd/0]  
 4 ? S< 0:00 [events/0]  
 5 ? S< 0:00 [khelper]  
 141 ? S 0:00 [pdflush]  
 1788 ? S< 0:00 [kmirrord]  
 1824 ? S< 0:00 [kjournald]  
 2016 ? Ss 0:00 /sbin/portmap  
 2241 ? Ss 0:02 /sbin/syslogd

Vysvětlivky

**PID,** udává identifikační číslo procesu **(které je v systému jedinečné).** Proces s názvem init, dalo by se říci rodičovský proces všech procesů v systému, bude mít vždy číslo 1. Init se spouští během startu.

**TTY** podává informaci o tom, ze kterého terminálu byl proces spuštěn. Pokud zde bude znak ‚?‘, znamená to, že daný proces není svázán s žádným terminálem (což jsou obvykle procesy spouštěné při inicializaci systému, nebo deamoni).

**STAT** podává informaci o tom, v **jakém stavu se daný proces nachází**.

**TIME**, podává informace o procesorovém času, který byl danému procesu již přidělen.

**COMMAN,** podává informace o příkazu a jeho parametrech, což je pro nás hlavní informace o daném procesu.

STATUSY PROCESU

|  |  |
| --- | --- |
| **S** | proces usnul - čeká až na něj přijde řada a bude mu přidělen procesor |
| **W** | paměťový prostor vyhrazený danému procesu byl kompletně uložen na disk (odswapován) |
| **R** | proces je právě zpracováván procesorem |
| **T** | proces byl pozastaven |
| **D** | proces je v nepřerušitelném spánku (v tomto stavu jsou většinou procesy svázané s I/O operacemi) |
| **Z** | proces, jehož rodičovský proces již ukončil svoji činnost (třeba díky nějaké závažné chybě a nechal po sobě sirotka. Rodičem Zombie procesu se stává Init). Zombie procesy v systému jsou obvykle důsledkem špatně napsaných programů. |
| **L** | proces má uzamknuté stránky v paměti - platí obvykle pro procesy pracující v reálném čase |
| **<** | proces s vysokou prioritou |
| **N** | proces s nízkou prioritou |

Příkaz **pstree**, nám vypisuje stromovou strukturu procesů ze které je dobře patrné rodičovství a posloupnost procesů.

libor@mail:~$ pstree  
init─┬─acpid  
 ├─apache2───11\*[apache2]  
 ├─asterisk───16\*[{asterisk}]  
 ├─atd  
 ├─cron  
 ├─cupsd  
 ├─6\*[getty]  
 ├─kthread─┬─aio/0  
 │ ├─2\*[pdflush]  
 │ └─scsi\_eh\_1  
 ├─master─┬─pickup  
 │ ├─qmgr  
 │ └─showq  
 ├─mysqld\_safe─┬─logger  
 │ └─mysqld───9\*[{mysqld}]  
 ├─10\*[perl]  
 ├─portmap  
 ├─pptpd  
 ├─smbd───2\*[smbd]  
 ├─sshd───sshd───bash───su───bash───pstree  
 ├─syslogd

**Příkaz top**. Tento příkaz na rozdíl od předchozího příkazu ps nevypisuje pouze výpis aktuálního stavu procesů v systému, ale dokáže tento výpis dynamicky po určitých časových intervalech měnit. Díky tomu můžeme v reálném čase sledovat změny stavu procesů, jejich "boj" o procesor, aktuální velikost paměti, které procesy alokují a spoustu dalších užitečných informací. Defaultní časový interval změny výpisu je nastaven na 5 sekund, lze jej však změnit pomocí parametru –‚d‘.

Výpis

top –b  
top-10:02:37 up 69 days, 34 min, 7 users, load average: 0.00, 0.10, 0.15  
Tasks: 127 total, 1 running, 125 sleeping, 1 stopped, 0 zombie  
Cpu(s): 11.2% us, 1.7% sy, 0.8% ni, 84.3% id, 1.7% wa, 0.1% hi, 0.2% si  
Mem: 385836k total, 380732k used, 5104k free, 42856k buffers  
Swap: 497972k total, 232456k used, 265516k free, 73372k cached  
  
 PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND  
 1 root 16 0 1580 164 1424 S 0.0 0.0 0:05.97 init  
 2 root 34 19 0 0 0 S 0.0 0.0 0:01.03 ksoftirqd/0  
 3 root 5 -10 0 0 0 S 0.0 0.0 0:17.46 events/0  
 4 root 5 -10 0 0 0 S 0.0 0.0 0:33.54 kblockd/0  
 5 root 15 0 0 0 0 S 0.0 0.0 0:01.78 kapmd  
 8 root 15 0 0 0 0 S 0.0 0.0 2:23.28 kswapd0  
 9 root 10 -10 0 0 0 S 0.0 0.0 0:00.00 aio/0  
 11 root 19 0 0 0 0 S 0.0 0.0 0:00.00 kseriod  
 15 root 15 0 0 0 0 S 0.0 0.0 0:31.93 kjournald  
 166 root 15 0 0 0 0 S 0.0 0.0 0:03.37 khubd  
 363 root 15 0 0 0 0 S 0.0 0.0 0:55.04 kjournald

(Výpis je moc složitý nevejde se na stránku, jen pro představu, jeho sloupce uvedeny níže)

**PID** – identifikace  
**USER**- identifikace uživatele   
**PRI** - aktuální priorita procesu  
**NICE**- priorita příkazem nice, záporná znamená vyšší  
**SIZE** - celková velikost procesu v paměti (velikost kódu + velikost zásobníku + velikost dat) kB  
**RSS**-kolik fyzické paměti je procesu vyhrazeno v kB  
**SHARE**-obsahuje velikost sdílené paměti  
**STAT** udává stav procesu (indikace stavu je stejná jako u příkazu ps výše)  
**CPU-** obsahuje procentuální informace o využití procesoru daným procesem v okamžiku výpisu  
**%MEM -** informuje o procentu využití fyzické paměti daným procesem.   
**TIME -** udává celkový procesorový čas, po který byl daný proces od spuštění až do okamžiku výpisu zpracováván *(z tohoto času je patrné, že spousta procesů má procesorový čas velmi malý, protože mnoho z nich často čeká na nějaké I/O operace)*.   
**COMMAND-** popisuje daný proces/příkaz.

Zrušení procesu (či poslání signálu danému procesu, chcete-li) se provádí příkazem **kill** nebo případně příkazem **killall**. Rozdíl mezi těmito dvěma příkazy je pouze v tom, že příkaz **kill** využívá k **identifikaci daného procesu jeho identifikační číslo (PID)** - *tedy číslo v prvním sloupci výpisu příkazu ps,* naproti tomu příkaz **killall** využívá **k identifikaci procesu jeho jméno** a dokáže tak **najednou obsloužit více procesů se stejným jménem**, což je vhodné zejména při **ukončování démonů** a jejich případných potomků.

Pro výpis signálů, jež můžeme procesům poslat, použijeme

man 7 signal  
Případně:  
kill -l

Pokud by se proces bránil a nechtěl by se ukončit, můžeme použít násilnější variantu:  
kill -9 cislo

Signály lze psát místo číselního označení, i celým názvem.  
kill -SIGKILL cislo = kill -9 cislo

Dalším signálem, který se také velice často využívá je signál úrovně 1 **(SIGHUP),** který se využívá zejména u démonů a slouží ke znovunačtení konfiguračních souborů, aniž bychom museli ukončit a znovu spustit daného démona. Pozor, není to však daným pravidlem, protože některé druhy démonů pro znovunačtení svých konfiguračních souborů používají signály úrovně **SIGUSR1** či **SIGUSR2.**

Dalším příkazem, který úzce souvisí se správou procesů, je příkaz sloužící ke změně priority daného procesu. Jedná se o příkaz **nice.** Pomocí příkazu nice můžeme jako běžný uživatel měnit prioritu námi spouštěných procesů a to v rozsahu **od 0 do +19.** Jako běžný uživatel můžeme tedy prioritu **pouze snižovat** od původní hodnoty (obvykle hodnota 0), kterou mimo jiné zjistíme příkazem nice bez parametru. Jako uživatel root můžeme prioritu jednotlivých procesů jak snižovat, tak i zvyšovat a to v rozmezí hodnot **-20** (nejvyšší priorita) až **+19** (nejnižší priorita).

Spuštení programu se zvýšenou prioritou:  
nice -8 ./preved\_porno.pes  
Spustení programu se sníženou prioritou:  
nice +5 /usr/local/bin/zmensifotky.pes /home/ftp/vasek/porno/

Pro změnu priority procesu běžícího v systému slouží příkaz renice

Běžný uživatel může měnit prioritu pouze svých procesů a navíc může prioritu pouze snižovat stejně, jako tomu bylo u příkazu nice.  
renice +10 cislo

Příkaz nohup je obvykle vhodný v případě, kdy se chceme odhlásit z dané konsole a přesto chceme, aby námi spuštěný proces v dané konsoli pracoval i nadále po našem odhlášení. Jediným parametrem příkazu nohup je název spouštěného programu včetně jeho parametrů.   
nohup program &

Takto bude program stále pracovat i po mém odhlášení z dané konsole a výstup z tohoto programu + případný chybový výstup bude přesměrován do souboru $HOME/nohup.out. Program nohup nepřesouvá automaticky daný proces na pozadí, přesun na pozadí je třeba provést pomocí &, či proces pozastavit a přesunout jej na pozadí pomocí příkazu bg.

# Správa paměti

**Memory Management Unit** (MMU) = hardware umístěný na CPU, který např. překládá logické adresy na fyzické

**Memory Manager** = software, který je součástí OS, udržuje informaci o volné a přidělené paměti, přiděluje a uvolnuje paměť, zajišťuje swapping procesů

## Základní techniky správy paměti

### Jednoúlohový systém:

* každý proces musí obsahovat ovladače V/V (I/O) zařízení
* v paměti je pouze jeden proces, kus paměti je vyčleněn pro jádro OS a kus pro tento program

### Víceúlohový systém:

#### Fixní oblasti

* rozdělení paměti (většinou při startu systému) na různě velké, avšak fixní, části
* program je nahrán do stejně velké nebo větší oblasti
* velikost paměťových oblastí se během běhu OS nemění
* **výhody:** jednoduchá implementace, malé režijní náklady
* **nevýhody**: interní fragmentace (místo uvnitř oblasti není využito na 100 %);
* počet aktivních procesů je fixní
* implementace pomocí:
  + oddělené vstupní fronty pro každou oblast - nevýhodou je, že dochází k nerovnoměrnému obsazení front
  + společné vstupní fronty

strategie výběru úlohy pro volnou oblast:

##### Best Fit

* nalezení největší úlohy, která se vejde do oblasti
* znevýhodňuje malé (interaktivní) úlohy
* lze zlepšit počítadlem, které počítá, kolikrát byl daný proces předběhnut - žádný proces nesmí být předběhnut více než k-krát

##### First Fit

* nalezení první úlohy, která se vejde do oblasti
* plýtvá místem větších oblastí

existují i další strategie: **Worst Fit, Last Fith**

#### Dynamické oblasti

počet, umístění a velikost oblastí se mění dynamicky, tak jak jednotlivé procesy vznikají, zanikají a přesouvají se mezi hlavní pamětí a diskem

výhody:

* "žádná" interní fragmentace
* efektivnější využití paměti

nevýhody

* externí fragmentace (možno setřásání paměti, ale je to časově náročné)

**Zvětšující se procesy:**

* datový segment procesu může měnit svojí velikost během výpočtu.
* proto musíme alokovat více paměti než je na počátku potřeba.

#### Požadavky na správu paměti

##### přemístění (Relocation)

* při kompilaci není většinou známo umístění procesu ve fyzické paměti
* každý odkaz do paměti v programu musí být přepočítán podle aktuálního umístění procesu ve fyzické paměti
* v programu se můžeme odkazovat na další programy
* různé přístupy k zavádění a spojování programu

##### ochrana

* každý proces musí být chráněn před nechtěnými přístupy z ostatních procesů
* paměť je rozdělena na bloky - každý je svázán s n-bitovým ochranným klíčem, který určuje, zda úloha smí přistupovat k datům v tomto bloku
* každá oblast paměti má dva registry (bázový a limitní), které obsahují nejmenší a největší možnou adresu této oblasti - porovnává se při přístupu do paměti
* nevýhodou je potřeba sčítání a porovnávání při každém přístupu do paměti (proto se tento problém řeší pomocí speciálního HW)

##### sdílení

* umožnění přístupu ke společné paměti (např. několik procesů provádí stejný program)
* logická organizace (logický adresový prostor)
* lineární (jednorozměrný) adresový prostor
* SW se obvykle skládá z modulů ⇒ více lineárních prostorů (např. s různými právy, kompilovány v různém čase, …)
* fyzická organizace (fyzický adresový prostor)
* fyzická paměť se skládá z různých úrovní (hierarchie pamětí)

#### Zavádění programu

##### absolutní zavedení

* každý odkaz v programu obsahuje absolutní fyzickou adresu
* program musí být zaveden od dané fyzické adresy
* při sestavování programu musím určit, kam bude program zaveden

##### přemístitelné zavedení

* každý odkaz do paměti v programu obsahuje relativní adresu
* **informace o paměťových odkazech je uložena v "Relocation Dictionary"**
* přepočítání relativní adresy na fyzickou se provede při zavedení programu

##### Dynamic Run-time Loading

* každý odkaz do paměti v programu obsahuje relativní adresu
* program zaveden do paměti s relativními adresami, která se přepočítává na fyzickou až při provádění instrukce

#### Spojování programu

**statické spojování** - vytvoří se jeden "load module" s relativními adresami vztaženými k začátku modulu

**dynamické spojování** - "load module" obsahuje odkazy na další programy

**load-time dynamic linking** - odkazy na další programy se nahradí při zavedení do paměti

**run-time dynamic linking** - odkazy na další programy se nahradí při provádění instrukce

#### Správa použité a volné paměti

Správa použité a volné paměti se stará o to jak udržovat informaci o volné a přidělené paměti

##### Bitové mapy

* paměť je rozdělena na alokační jednotky (AU, veliké několik KB)
* každá AU má korespondující bit ve speciálním "řetězci" – bitové mapě (0 - volná, 1 - přidělená)
* nalezení volných AU = nalezení souvislého řetězce nulových bitů

**problémy:**

* velké AU – máme sice malou bitovou mapu, ale plýtváme hlavní pamětí
* malé AU – sice lépe využíváme paměť, ale máme velkou bitovou mapu
* hledání v bitové mapě je pomalé!

##### Zřetězené seznamy

zřetězený seznam volných (H - Hole) a přidělených (P - Process) paměťových segmentů

# Souborové systémy

## ext2: Second Extended Filesystem

Tento souborový systém je téměř tak starý jako Linux sám. Ještě nedávno jste jej nalezli přednastavený ve většině linuxových distribucí. Mezi jeho hlavní přednosti patří léty prověřená spolehlivost a slušná rychlost všech běžných operací. Nevýhodou je pomalost těch méně běžných (práce s milionem souborů v jednom adresáři apod.), pevně daný počet inodů a nutnost provádět kontrolu integrity po každém nekorektním restartu systému.

Při zakládání filesystému, ať už pomocí mkfs, nebo pomocí instalátoru distribuce, si lze zvolit velikost bloku. Použitelné velikosti se pohybují od 1024 do 4096 bajtů, menší bloky trochu šetří místo na disku, větší umožňují rychlejší práci se systémem a zejména rychlejší běh fsck. Na velikosti bloku rovněž závisí maximální možná velikost souboru. I pro bloky o velikosti 1 kB ovšem tento limit činí více než 16 GB.

Počet inodů na filesystému si mkfs zvolí podle kapacity oddílu, ve výchozím nastavení je to jeden inode na každých 8 kB kapacity, u malých oddílů na 4 kB. Tento poměr je při zakládání filesystému možné změnit, což se vyplatí zejména, pokud plánujeme skladovat na disku velké množství malých souborů nebo naopak několik málo velikánských.

Taktéž je možné určit, jaká část kapacity oddílu bude rezervována pro procesy spuštěné s rootovskými právy. Obvykle to bývá pět procent, u velkých oddílů nebo oddílů používaných jedním uživatelem se může vyplatit tuto hodnotu zmenšit.

## ext3: Third Extended Filesystem

Tento filesystém má zcela shodnou strukturu s filesystémem Ext2 až na to, že provádí **žurnálování**. Existující filesystém Ext2 je možno přetvořit na Ext3 pouhým přidáním speciálního souboru obsahujícího žurnál. Ext3 je možno zpětně namountovat jako Ext2 (v takovém případě se neprovede obnovení žurnálu — proto před tím nesmělo dojít k pádu). Ext3 může pracovat ve třech režimech:

* **Unordered data** nebude udržováno pořadí mezi daty a metadaty, takže zde existuje bezpečnostní problém
* **Ordered data** bude zajištěno, že data budou zapsána dříve než commit transakce, takže je filesystém bezpečný
* **Journal data** data budou zapisována do žurnálu. To dvakrát zpomalí rychlost zápisu dat, ale způsobí, že bude udržováno i pořadí jednotlivých volání funkce write. Doporučená metoda je ordered data, což zajišťuje bezpečnost a nezpomaluje tolik jako žurnálovaná data.

Ext3 se nachází v jádrech 2.4 i 2.6 (je možno stáhnout patch Ext3 i pro jádra 2.2), ale z komentářů v kódu je vidět, že ještě úplně odladěný není — může se vyskytovat špatné předpovídání velikosti transakce, což může vést k přeplnění žurnálu, synchronizace metadat a dat také není absolutně v pořádku. Tyto chyby se v běžných případech moc nevyskytují.

## JFS: IBM Journaling Filesystem

Tento zajímavý kousek vznikl začátkem 90. let v laboratořích firmy IBM. Původně byl navržen pro operační systém AIX a časem přenesen i pod OS/2 a následně pod Linux. Byl to pravděpodobně první filesystém, ve kterém se objevilo žurnálování a celá řada dalších, na svou dobu velmi pokrokových, triků jako třeba dynamické alokování inodů a komprimované uložení alokačních tabulek.

JFS velice efektivně zvládne všechny druhy zátěže a snad jedinou jeho potenciální nevýhodou je to, že nepodporuje média menší než 16 MB. Při těchto vlastnostech je až ku podivu, že není v linuxovém světě moc známý.

## XFS

Systém XFS byl vyvinut firmou Silicon Graphics pro operační systém IRIX a posléze zveřejněn a přenesen i pod Linux. Hlavním cílem autorů tohoto filesystému byl velký výkon při práci s objemnými daty. Zda se to zdařilo, posuďte sami: při jednom z testů se povedlo na 32procesorovém serveru dosáhnout přenosové rychlosti 7 GB/s. Co se týče pokročilosti návrhu a výkonu v obvyklejších případech, je dosti podobný JFS.

Jistou daní za tyto výhody je ovšem značná složitost kódu. Zdrojové texty XFS v jádru Linuxu měří více než sto tisíc řádků řádků (JFS má okolo 30000, ext3 cca 15000) a je těžko k uvěření, že by se v nich neskrývaly žádné další chyby.

## ReiserFS a Reiser4

Tyto dva souborové systémy pochází z dílny Hanse Reisera a jeho firmy Namesys. Jsou založeny na dosti neobvyklých a zajímavých nápadech, jako je třeba uložení kompletní struktury filesystému do jednoho vyváženého stromu. Snaží se nabídnout efektivní práci s velkým množstvím maličkých souborů a spojit tak výhody souborového systému s výhodami databází. Aby toho dosáhly, ukládají zvlášť konce ("ocásky" neboli tails) souborů, které nezaplní celý blok; do jednoho bloku obvykle uloží ocásky více souborů. To na druhou stranu zpomaluje práci se středně velkými soubory, takže se někdy vyplatí filesystém namountovat s volbou notail, čímž se tato vlastnost vypne.

Znatelnou vadou na kráse těchto filesystémů je absence spolehlivého nástroje na opravu rozpadlých oddílů. To by sice díky žurnálování nemělo teoreticky nikdy nastat, ale praxe nám tuto teorii obvykle brzy rozmluví.

V současných linuxových jádrech je integrován ReiserFS verze 3. Autoři od té doby dokončili verzi 4, která by měla odstraňovat zmíněné nevýhody a také fungovat o poznání rychleji. O její integraci do jádra se prozatím vedou mezi autory a kernelovými vývojáři vleklé spory, protkané spoustou technických problémů i osobních antipatií.

## Žurnálování

Žurnálování je metoda, která umožňuje udržet po výpadku proudu zcela konzistentní stav filesystému. Není třeba provádět kontrolu filesystému a nevznikají ztracené bloky nebo inody jako u soft updates. Na filesystému je vyčleněna oblast (veliká obvykle pár megabytů) nazývaná žurnál. Ovladač filesystému musí změny provádět v tzv. transakcích. Transakce je sada několika modifikací dat, která převádí filesystém z jednoho konzistentního stavu do druhého konzistentního stavu. Například při vytváření souboru se vyrobí transakce, do které náležejí následující operace: zapsání položky adresáře, zapsání inody, zapsání bitu v bitmapě alokovaných inod, zmenšení počítadla volných inod v superbloku. Operace náležející transakci se nezapisují na disk, ale nejdříve se zapisují do žurnálu. Až je celá transakce v žurnálu zapsaná, zapíše se do něj speciální značka — commit transakce. Poté se buffery obsahující jednotlivé modifikace mohou zapisovat na disk jako při asynchronním zápisu. Do žurnálu se zpravidla zapisuje pořád dopředu, a když dojde k jeho zaplnění, začne se zapisovat opět od začátku (při této operaci je třeba zapsat na příslušná místa na disku všechny buffery obsahující modifikovaná data). Při mountu filesystému se kontroluje žurnál. Pokud je v něm nalezena transakce včetně commit značky, tak se všechny operace náležející této transakci zapíší na příslušná místa na disku. Pokud je v žurnálu nalezena neúplná transakce bez commitu, ignoruje se.

Žurnálování zajišťuje absolutní konzistenci filesystému po výpadku. Pokud počítač spadl během zápisu transakce do žurnálu, je transakce v žurnálu neúplná a bude při příštím startu ignorována. V takovém případě máme garantováno, že ještě nedošlo k zápisu žádných dat na disk mimo žurnál. Pokud celý zápis transakce do žurnálu proběhl (včetně commit značky) a počítač spadl během zápisu dat na disk, bude při příštím startu celá transakce z žurnálu rekonstruována, čímž budou nekompletní data na disku přepsána. Ať spadne počítač v jakémkoli okamžiku, data budou konzistentní. Aby žurnálování mohlo správně fungovat, předpokládá se, že disk je schopen provést tzv. atomický zápis sektoru — tj. v sektoru se po výpadku budou nacházet buď úplná stará, nebo úplná nová data. Pokud dojde k výpadku proudu během zápisu sektoru, disk musí být schopen tento zápis dokončit. Současné disky to umějí. Schopnost atomického zápisu sektoru nemají například diskety (pokud disketovou mechaniku vypnete během zápisu dat, výsledkem bude neúplně zapsaný sektor, který nejspíše bude produkovat CRC chybu), proto používání žurnálového filesystému na disketách nemá smysl.

Do žurnálu se zpravidla zapisují pouze metadata (tj. řídící informace filesystému) a nikoli samotná data obsažená v souborech. Důvodem je rychlost — při žurnálování je potřeba na disk zapsat dvakrát více dat než při normálním zápisu a zapisování celých souborů by proces zápisu dvakrát zpomalilo. Pokud se data do žurnálu nezapisují, je třeba zajistit jejich konzistenci, aby nevznikl výše popsaný bezpečnostní problém: metadata jsou zapsaná, data nejsou zapsaná, uživatel po pádu bude moci číst cizí smazané soubory. Obecně platí, že před zapsáním commit značky je potřeba zapsat všechna nově vzniklá data. Musí se také udržovat konzistence mezi datovými a metadatovými buffery, což logiku bufferové cache komplikuje (například není možné zapsat data do bloku, ve kterém dříve byla metadata, je-li v žurnálu ještě nějaká transakce pracující s těmito metadaty).

Neexistuje žurnálový filesystém pro FreeBSD. Pro Linux existuje několik žurnálových filesystémů.

Porovnání výkonu např: <http://linuxgazette.net/102/piszcz.html>

# Shell

**Unixový shell** (též **příkazový procesor**, v doslovném překladu „unixová skořápka“) je název textového uživatelského rozhraní, které je předchůdcem grafického uživatelského rozhraní. Shell je spuštěn po přihlášení uživatele do systému, vytvoří příkazový řádek, pomocí kterého uživatel může počítač ovládat a jeho ukončením je uživatel ze systému odhlášen.

Shell je tradičním rozhraním pro operační systém UNIX a Unixu podobné systémy, ve kterých si uživatel může vybrat z široké nabídky různých shellů. V operačním systému DOS existuje podobný, ale mnohem jednodušší program COMMAND.COM. V Microsoft Windows ho nahrazuje program CMD.EXE, který se v poslední verzi systému Windows Vista unixovým shellům přibližuje.

## Příkazový řádek

Shell vytváří prostředí příkazového řádku, do kterého uživatel zadává názvy příkazů, které chce spustit. Shell tyto příkazy interpretuje, spouští odpovídající programy a umožňuje sledovat jejich výstup. Umožňuje příkazům předávat parametry, seskupovat je, slučovat příkazy do skriptů a podobně. Shell představuje „skořápku“, která skrývá uživateli detaily ovládání kernelu (jádra) operačního systému. Mnoho uživatelů unixových systémů dosud považuje moderní příkazový řádek shellu za mnohem pohodlnější způsob ovládání počítače než grafické uživatelské rozhraní.

## Různé shelly

Shell je z hlediska systému program, který je zpravidla spuštěn v okamžiku přihlášení uživatele do systému. Ve většině unixových systémů si uživatelé mohou vybrat shell, který chtějí použít.

Nejdůležitějšími shelly jsou:

* Bourne shell (sh)
* Bourne-again shell (bash)
* C shell (csh)
* Korn shell (ksh)
* TENEX C shell (tcsh)

V raných verzích Unixu byl používán Bourne shell, který se stal de facto standardem; každý unixový systém má přinejmenším jeden shell kompatibilní s Bourne shellem. V souborové hierarchii je takový shell umístěn v /bin/sh. Na některých systémech, jako BSD, je /bin/sh Bourne shell nebo jeho ekvivalent, ale v ostatních systémech jako Linux je /bin/sh obvykle odkaz na kompatibilní, ale funkčně bohatší shell. Norma POSIX specifikuje standardní shell jako striktní podmnožinu Korn shellu.

Shelly v Unixu mohou být rozděleny zhruba do čtyř kategorií: shelly podobné Bourne shellu, podobné C Shellu, netradiční a historické. Na většině unixových systémů můžete použít pro zjištění, jaký shell používáte, příkaz echo $SHELL.

## Bourne shell

**Bourne shell**, nebo **sh**, byl standardní Unixový shell od Unix verze 7 a nahradil Thompson shell, jehož spustitelný soubor měl stejné jméno sh. Tento shell byl vyvinutý Stephem Bournem z AT&T Bell Laboratories a byl uvolněný ve verzi Unixu 7 v roce 1977. Toto vydaní se distribuovalo vysokým školám a univerzitám. Tento shell zůstává populární jako standard pro Unixové účty. Program Bourne shell nebo kompatibilní program je umístěný většinou v */bin/sh*.

## Bash

**Bash** je unixový (Linux/Unix/BSD) příkazový shell interpreter naprogramovaný v rámci projektu GNU. Název je akronym k názvu **B**ourne **a**gain **sh**ell - je založen na **B**ourne **Sh**ellu (bsh), což byl nejpoužívanější unixový shell. Originální **BSH** byl napsaný panem Stephenem Bournem v Bellových laboratořích a **Bash** byl napsán Brianem Foxem v roce 1987. Bash byl také portován na operační systém Microsoft Windows projektem Cygwin.

Bash se snaží o širokou kompatibilitu, takže převzal možnosti ovládání a vlastnosti i z dalších Shellů jako jsou např. Korn shell a C shell (ksh a csh). Zajímavou vlastností je automatické rozpoznání pod kterým jménem byl spuštěn a přizpůsobení syntaxe danému typu Shellu.

Cílem vývojářů je dosáhnout 100% kompatibility s implementací IEEE POSIX Shellu a specifikace nástrojů (tools specification) (IEEE Working Group 1003.2).

Bash však není jen velmi výkonným shellem, ale také mocným scriptovacím jazykem. Podporuje práci s proměnými, cykly while, for, do, funkce a mnoho dalšího.

## C shell

**C shell** (**csh**) je jedna z verzí Unix shellu vyvinutý Bill Joyem pro BSD Unix system.

Byl původně odvozen z šestého vydání Unixu /bin/sh (kterým byl Thompson shell), předchůdce Bourne shell. Jeho skladba je formovaná podle programovacího jazyka C. C shell přidal mnoho vylepšení oproti Bourne shell, jako například aliases a historii příkazů.

V současné době, původní C shell není v Unixu široce využíván. Byl nahrazen jinými shelly, jako je například Tenex C shell (tcsh) založený na původním C shell kódu.

### Shelly kompatibilní s Bourne shellem

Bourne shell (sh) – Napsal Steve Bourne v Bell Labs. Poprvé distribuován s Unixem 7, přibližně v roce 1978 a postupně vylepšen.

* Almquist shell (ash) – Napsán jako náhrada Bourne Shellu pod BSD licencí; je často používán na slabém hardwaru.
* Bourne-Again shell (bash) – Napsán jako část projektu GNU, aby rozšiřoval funkčnost Bourne shellu.
* Korn shell (ksh) – Napsal David Korn v Bell Labs.
* Z shell (zsh)

### Shelly kompatibilní s C shellem

C shell (csh) – Napsal Bill Joy na University of California v Berkeley. Prvně distribuováno s BSD, kolem roku 1979.

* TENEX C shell (tcsh)

### Netradiční

* fish, „přátelský interaktivní shell“, prvně vydaný v roce 2005.
* mudsh, inteligentní hře podobný shell, který funguje jako MUD.
* zoidberg, napsaný, konfigurovaný a operující výhradně v Perlu
* rc, standardní shell z Plan 9 from Bell Labs a Unixu 10, napsal Tom Duff.
* es, shell podporující funkcionální programování, napsán v polovině 90. let.
* scsh (Scheme Shell)

### Historické

* Thompson shell – První Unixový shell, napsal Ken Thompson v Bell Labs. Distribuován s verzemi Unixu od 1. do 6., od roku 1971 do 1975. Moderními standardy považován za základní a na současných systémech se již nepoužívá, ačkoli je dostupný jako část starých Unixových systémů.
* PWB shell nebo také Mashey shell – Verze Thompsonova shellu, rozšířil John Mashey a ostatní, když byli v Bell Labs. Distribuována s *Programmer’s Workbench UNIX* okolo roku 1976.

### Grafické shelly

Grafická uživatelská rozhraní pro Unix, například GNOME a KDE, jsou občas nazývána *vizuální shell* nebo *grafický shell*.

# Obsah

Obsah

[1 Vývoj 1](#_Toc197065904)

[2 Správa uživatelů 1](#_Toc197065905)

[2.1 Uživatelé 1](#_Toc197065906)

[2.2 Skupiny 2](#_Toc197065907)

[3 Správa procesů 3](#_Toc197065908)

[4 Správa paměti 7](#_Toc197065909)

[4.1 Základní techniky správy paměti 7](#_Toc197065910)

[4.1.1 Jednoúlohový systém: 7](#_Toc197065911)

[4.1.2 Víceúlohový systém: 7](#_Toc197065912)

[5 Souborové systémy 10](#_Toc197065913)

[5.1 ext2: Second Extended Filesystem 10](#_Toc197065914)

[5.2 ext3: Third Extended Filesystem 10](#_Toc197065915)

[5.3 JFS: IBM Journaling Filesystem 10](#_Toc197065916)

[5.4 XFS 11](#_Toc197065917)

[5.5 ReiserFS a Reiser4 11](#_Toc197065918)

[5.6 Žurnálování 11](#_Toc197065919)

[6 Shell 12](#_Toc197065920)

[6.1 Příkazový řádek 13](#_Toc197065921)

[6.2 Různé shelly 13](#_Toc197065922)

[6.3 Bourne shell 13](#_Toc197065923)

[6.4 Bash 13](#_Toc197065924)

[6.5 C shell 14](#_Toc197065925)

[6.5.1 Shelly kompatibilní s Bourne shellem 14](#_Toc197065926)

[6.5.2 Shelly kompatibilní s C shellem 14](#_Toc197065927)

[6.5.3 Netradiční 14](#_Toc197065928)

[6.5.4 Historické 15](#_Toc197065929)

[6.5.5 Grafické shelly 15](#_Toc197065930)

[7 Obsah 15](#_Toc197065931)