

Vypracováno převážně z přednášek a wiki.

BEZ ZÁRUKY!!!

1) Okruh

Základní znaky open-source software, licence (GPL, LGPL), vývojový model, spolupráce s komunitou, koexistence komerčních firem a open-source komunit, možnosti generování zisku.

Základní znaky open-source software

Open source popisuje způsoby výroby a vývoje, které upřednostňují přístup ke kompletnímu zdrojovému kódu pro všechny v procesu výroby, distribuce a užití zúčastněné strany.

- Vše lze **zkoumat, sledovat a modifikovat** funkci.
- Přitom získané znalosti smíte použít v jiných aplikacích a to i uzavřených a nehrozí vám **žaloby za vyzrazení tajemství**, mnohaleté **závazky mlčenlivosti** a mnohaleté **zákazy práce v určitém oboru**.
- Pro velké firmy je to jedna z mála možností jak spolupracovat bez úzkostného strachu z konkurence, patentů, právníků.
- Lze **vydělávat na podpoře, správě, distribuci, tvorbě rozšíření**.

Licence

GPL: (General Public License) - původně pro projekt GNU. GPL je nejpopulárnějším a dobře známým příkladem silně copyleftové licence, která vyžaduje, aby byla odvozená díla dostupná pod toutéž licencí.

LGPL: (Lesser General Public License) - je upravená, permissivnější verze GPL, původně zamýšlená pro některé knihovny. LGPL aplikuje copyleftové restrikce na program samotný, neuplatňuje se však na software, který tento program linkuje. => knihovna s licencí LGPL může být součástí softwaru pod jinou licencí.

BSD: (Berkeley Software Distribution) – jedna z nejsvobodnějších licencí. Licence dovoluje komerční využití, včetně využití v proprietárním softwaru bez zveřejněného zdrojového kódu. Licence vyžaduje v programu uvádět informaci o autorech a zřeknutí se odpovědnosti.

MIT a Apache: licence nejsou copyleft. Odvozenina může být pod jinou licencí. Je ale nutné uvést disclaimer(zřeknutí se odpovědnosti) a z jakého projektu byla odvozenina vytvořena. V proprietárních systémech musí být dodávána i tato licence.

Creative commons: je licence pro jakýkoli obsah, nejenom pro software. Do licence si člověk může dát 1-4 z těchto atributů:

Attribution - nutno uvést autora originálního projektu

Noncommercial - možno upravovat jenom pro nekomerční účely

NoDerivatives - možno pouze kopírovat dané dílo, ale nikoliv ho upravovat

Share-alike - znamená virální copyleft

Vývojový model:

Otevřený model vývoje SW – **FOSS** (Free and Open Source Software)
Svobodně licencován pro použití, studování, úpravy a vylepšení. => Dostupné
zdrojové kódy.

Spolupráce s komunitou:

- Svobodná licence
- Stránky na Internetu a možnost stahování zdarma
- Verzovací systém
- Veřejný mailinglist
- Wiki stránky projektu
- Fórum, IRC

Koexistence komerčních firem a open-source komunit:

- Nasazení open source sw v rámci komerční firmy, lze snížit licenční poplatky. (často až na nulu)
- Nemůže nastat naprostá závislost na jednom dodavateli.
- Není zde hrozba ztráty podpory - pokud původní dodavatel již není ochoten (cena služby), najde se jiný.
- Je zde možnost úpravy sw podle vlastních požadavků.

Možnosti generování zisku.

Zisk z prodeje HW – svobodný SW zvyšuje poptávku po HW. (HP, IBM, SUN)

Placená podpora a distribuce - Samotné SW je zdarma. Prodej zkušeností a zákaznický placený vývoj úprav/rozšíření. (RedHat)

Placená rozšíření – stabilní složitý základ sw je dostupný zdarma a vývoj specializovaného rozšíření pro zákazníka je zpoplatněn.

Dvojitá licencování - Nabídka alternativní licence za peníze. Neochota přijmout podmínky FOSS (GPL).

Reklama – šíření povědomí o společnosti. Ekonomika založená na webu (zobrazení reklamy).

2) Okruh

Správa rozsáhlých softwarových projektů, systémy pro verzování, sledování chyb, koordinace vývojářů a vedení projektů.

Správa rozsáhlých softwarových projektů:

Požadavky pro úspěch projektu:

- Svobodná licence
- Stránky na Internetu a možnost stahování zdarma
- Uživatelská komunita
- Aktivní a nepřetržitý vývoj
- Flexibilita a možnost úprav a rozšiřování
- Otevřený a jednoduchý vývoj
- Dodržování standardů a formátů
- Jasné vedení, jinak vznikne pouze balast

CHYBY: pozdní uvolňování verzí, registrace pro stažení, nutnost povolení pro začlenění každého příspěvku, Dohody o utajení (NDAs), ...

Systémy pro verzování:

správa zdrojových kódů: **diff** – nástroj pro porovnání dvou souborů, nástroj **patch** aplikuje změny na starý soubor (**patch** také - výstup z programu diff – seznam změn v souboru) a **tar**

posíláno přímo v těle e-mailu (žádné html), každý si udržuje vlastní zdrojový strom
distribuováno přes FTP a usenet

Změnové soubory Patch – unidiff dnešní standard (+++ ----)

První SCM (source code management system) **RCS** a **SCCS**

- pouze jeden uživatel může editovat soubor v jednom čase
- pracuje pouze přímo ze soubory na disku
- bez možnosti slučování nezávislých změn (Merge)
- dokumentují historii vývoje

Concurrent Versions System CVS – paralelní správa verzí

- Založený na základech a formátu RCS
- Dovoluje paralelní vývoj (na jednom počítači i distribuovaně)
- Obsahuje základní nástroje na řešení slučování (merge) a řešení konfliktů – jsou založené na nástrojích diff a patch
- slabá podpora slučování větví
- téměř všechny operace vyžadují komunikaci se serverem
- žádná podpora pro přejmenování souborů a jen velmi malá pro práci s adresáři

Centralizovaná:

- CVS server je centrální prvek
- Vývojář má pouze svůj (aktuální) checkout/sandbox/pískoviště
- Většina/všechna metadata (historie atd.) jsou uloženy pouze na centrálním serveru

Distribuovaná správa verzí

- Každý vývojář má vlastní kopii celé historie projektu
- Většina takových systémů nabízí podporu pro snadné zakládání
- větví a jejich slučování

Subversion SVN

- implementace CVS
- Snaha o znovuvytvoření systému s centrální správou verzí
- Řeší mnoho omezení CVS
- Revize jsou zaznamenávány přes celý projekt
- Stále často užívané

Distribuované SCM

- **Birkeeper** : použit při vývoji jádra Linuxu, složitý licenční model
- **Mercurial, Monotone**
- **Git**

GIT:

- Distribuovaný systém správy verzí
- Linus Torvalds
- vychází z projektu BitKeeper
- svobodný software, licence GPL
- Mocná podpora pro nelineární vývoj. Git podporuje rychlé vytváření větví a rychlé slučování (merge)
- Distribuovaný vývoj. Git poskytuje každému vývojáři lokální kopii celé historie vývoje. Změny se kopírují z jednoho takového úložiště do jiného.

- Git je velmi rychlý (až 10x rychlejší než ostatní než jiné systémy pro správu verzí – testy združení Mozilla) a škálovatelný.
- Git zpracovává data odlišně od všech ostatních SCM, uchovává data jako sadu snímků svého malého souborového systému. Po každém uložení stavu, git vyfotí jak vypadají všechny soubory a uloží odkaz na tento snímek.
- Téměř každá operace je lokální
- tři stavy souboru :

Zapsáno (committed) - data bezpečně uložena ve vaší lokální databázi

Změněno (modified) - Změněno znamená, že v souboru byly provedeny změny, avšak soubor ještě nebyl zapsán do databáze

Připraveno k zapsání (staged) - změněný soubor v jeho aktuální verzi určili k tomu, aby byl zapsán v další revizi (tzv. commit).

Z toho vyplývá, že projekt je v systému Git rozdělen do tří hlavních částí: **adresář systému Git (Git directory)**, **pracovní adresář (working directory)** a **oblast připravených změn (staging area)**. V adresáři Git ukládá systém databázi metadat a objektů k projektu.

Pracovní adresář obsahuje lokální kopii jedné verze projektu. Tyto soubory jsou staženy ze zkomprimované databáze v adresáři Git a umístěny na disk, abyste je mohli upravovat.

Oblast připravených změn je jednoduchý soubor, většinou uložený v adresáři Git, který obsahuje informace o tom, co bude obsahovat příští revize.

Získání repozitáře: git clone (klonování existujícího repozitáře), git init (importování existujícího projektu)

Kompilační farma:

- **Rozsáhlá škála strojů (často i virtualizace), různé HW platformy a operační systémy**
- Automaticky spouští regresní testy po každém commitu (změně)
- Chyby při kompilaci nebo testu mohou být poslané na email a jsou k dispozici v logu (přes web)

Regresní testy - Cílem regresního testování je zajistit, aby změny software, jako je přidávání nových funkcí, nebo úpravy stávajících funkcí, neovlivnily nepříznivě funkcionality ostatních částí aplikace, na které změny nemají mít vliv.

Příklady: Tinderbox, Samba build farm, Buildbot

Sledování chyb:

Koordinace vývojářů a vedení projektů:

Jaké jsou obecně možné formy řízení open-source projektů a jaký je stav v případě Linuxového jádra a projektu Debian?

- Právo na rozhodování v podstatě vždy založené na míře osobních zásluh
- Neexistuje jednotný systém, vždy nějaký vývoj během života projektu
- Řídicí výbor, Diktátor (Linus), rotování funkcí
- Důležitá je komunikace, spolupráce ochota poslouchat a nalézt a podřídít se společné vizi

3) Okruh

Binární distribuce, jejich vznik, správa aplikací a koncepce balíčků (Debian DEB, RedHat RPM) a přehled důležitých open-source projektů.

Binární distribuce, jejich vznik

- samotné jádro nestačí, jsou nutné nějaké nástroje v userspace
- pokud jsou k dispozici pouze zdrojové kódy, tak bez již běžícího systému a nástrojů, jsou nepoužitelné ==> vznik binárních distribucí

Pro zavedení systému je nutná binární distribuce jádra, základních nástrojů a nějaký instalátor / zavaděč.

Plnohodnotný systém musí řešit instalaci a boot

boot – je řešen jednodušší aplikací postavené nad BIOSEM nebo přímo HW.

Instalace – řešeno pomocí aplikace napsané pro minimální konfiguraci přímo cílovému systému.

Správa aplikací v Linuxu probíhá přes **package manager** na jednom místě. Package managers: Aptitude, Yum

Debian DEB

Od začátku předpokládal kompletně nekomerční projekt vyvíjený stovkami dobrovolníků.

- + Velmi stabilní
- + Výjimečný systém řízení/kontroly kvality
- + mnoho balíčků a nejvíce podporovaných architektur ve světě Linuxu
- Konzervativní, ne vždy poslední technologie (i kvůli množství architektur)
- pomalý vývojový cyklus (1- 3 roky)

DEB balíček se skládá ze tří souborů zabalených programem **ar**

debianbinary – pouze identifikátor formátu. Současná verze "2.0"

control.tar.gz – veškeré metainformace o balíčku

data.tar, **data.tar.gz**, **data.tar.bz2** – vlastní data/soubory v adresářové struktuře systému

Deriváty: Ubuntu

RedHat RPM

Hat je profesionálním hráčem na poli opensource SW
vývoj probíhá komunitně, firma však zajišťuje kvalitu, záruky a placenou podporu
od vývoje, přes školení až po 24/7 služby

- + Distribuce je podporovaná velkou firmou
 - + Placení vývojáři přímo přispívají do klíčových projektů (kernel, GCC, binutils),
 - + zásada RHEL je že vše musí být OSS
 - + Dlouhá doba podpory
 - Podpora/záruky/originál je drahý
 - pomalý distribuční cyklus (2 – 3 roky)
- => spíše na servery než desktop

Správa balíčků: formát RPM

Soubor `<name><version><release>.<architecture>.rpm`

Hlavička identifikující RPM

Digitální signatura pro ověření integrity a původu

Sekce obsahující metadata (jméno balíčku, verzi, architekturu, závislosti, seznam souborů atd.)

Tvorba balíčků podle **.spec** souborů. Zdrojové balíčky **.src.rpm**

4) Okruh

Základní přehled o skladbě OS POSIXového typu, jádro, [model ovladačů,] správa paměti, uživatelské prostředí (procesy, knihovny, souborové systémy).

GNU

Projekt GNU je projekt zaměřený na svobodný software, inspirovaný operačním systémem unixového typu. **Původní cíl byl vyvinout operační systém se svobodnou licencí, který však neobsahuje žádný kód původního UNIXu.** Jeho jméno je rekurzivní zkratka pro *GNU's Not Unix*.

Projekt GNU založil v roce 1983 programátor Richard Stallman.

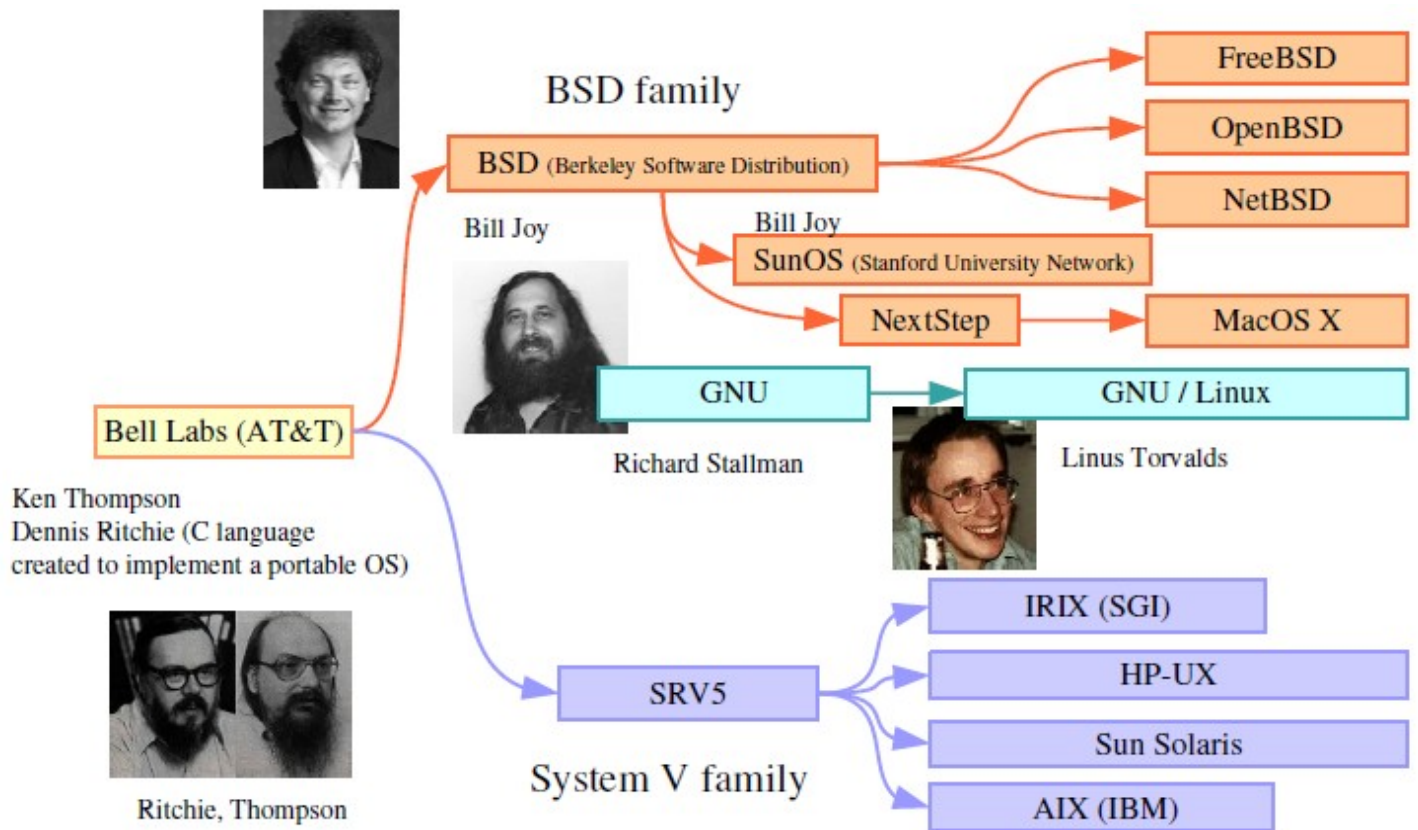
Cílem projektu bylo vytvořit kompletní svobodný operační systém. To se podařilo naplnit v roce **1992, kdy byla poslední chybějící součást, jádro (kernel), doplněna nezávisle vytvořeným svobodným jádrem Linux.**

GNU/Linux – Linus Torvald

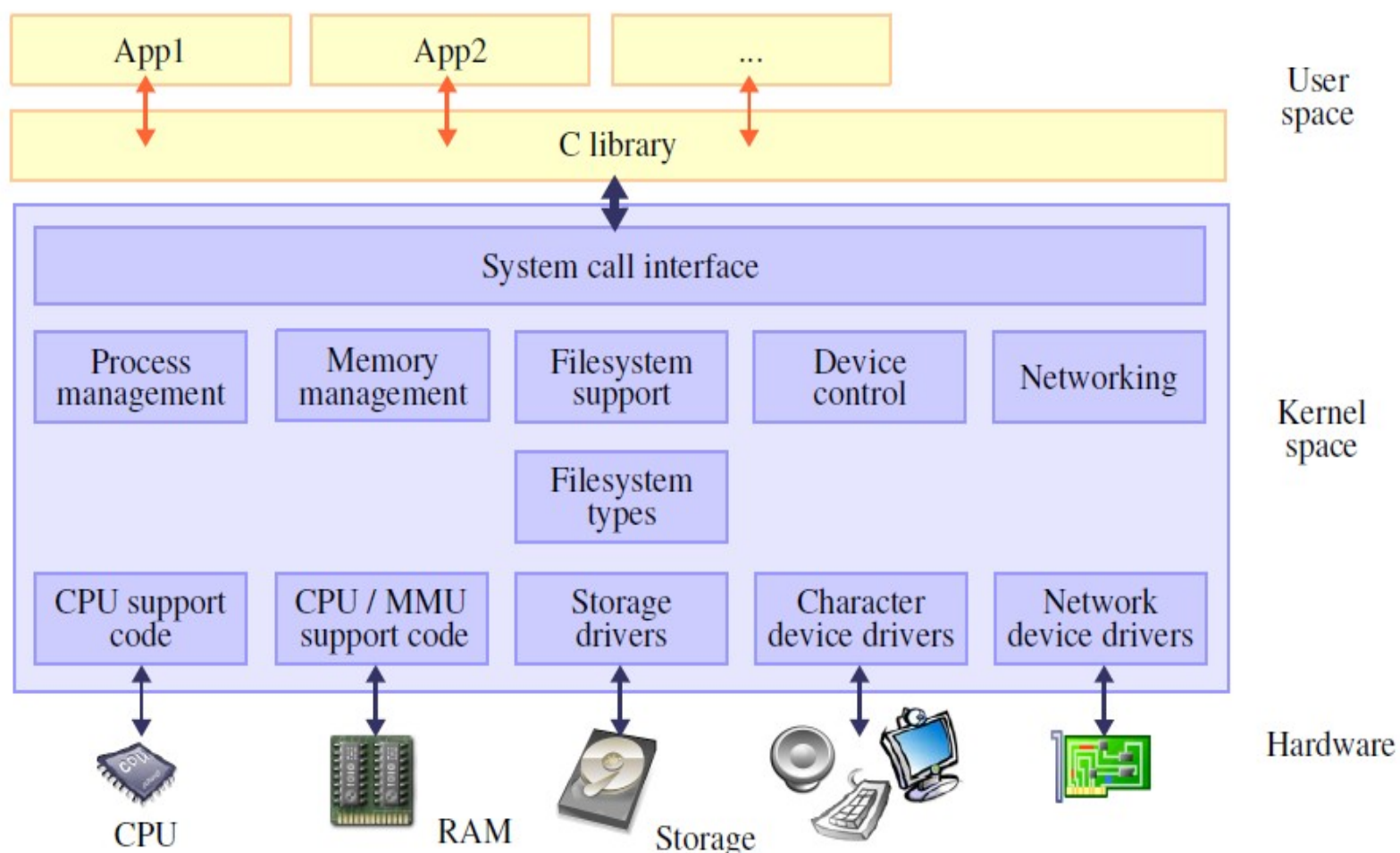
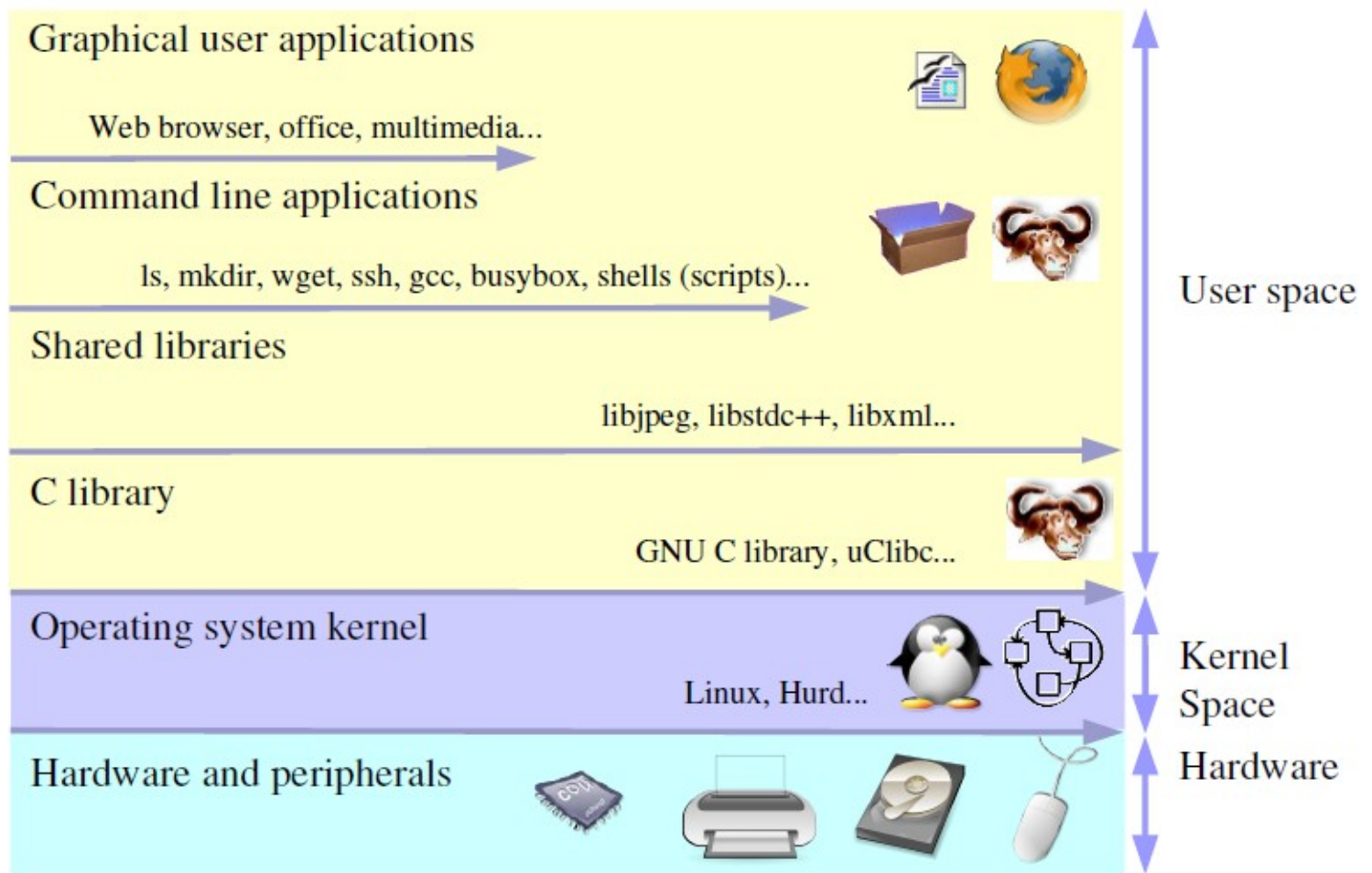
UNIX, GNU, GNU/Linux

UNIX – ochranná známka operačního systému vytvořeného v Bellových laboratořích firmy [AT&T](#)
Většina současných operačních systémů je unixovými systémy různou měrou inspirována.

- 1970 Unix vznikl ze zaniklého projektu Multics od Bell Labs
- 1983 Richard Stallman z MIT založil GNU project pro unix-like system. Založil Free Software Foundation, napsal spoustu toolů pod free licencí
- 1991 Linus Torvalds napsal vlastní jádro a složil ho s GNU prostředím, vzniká GNU/Linux



Architektura (Unixových) systémů



Posix

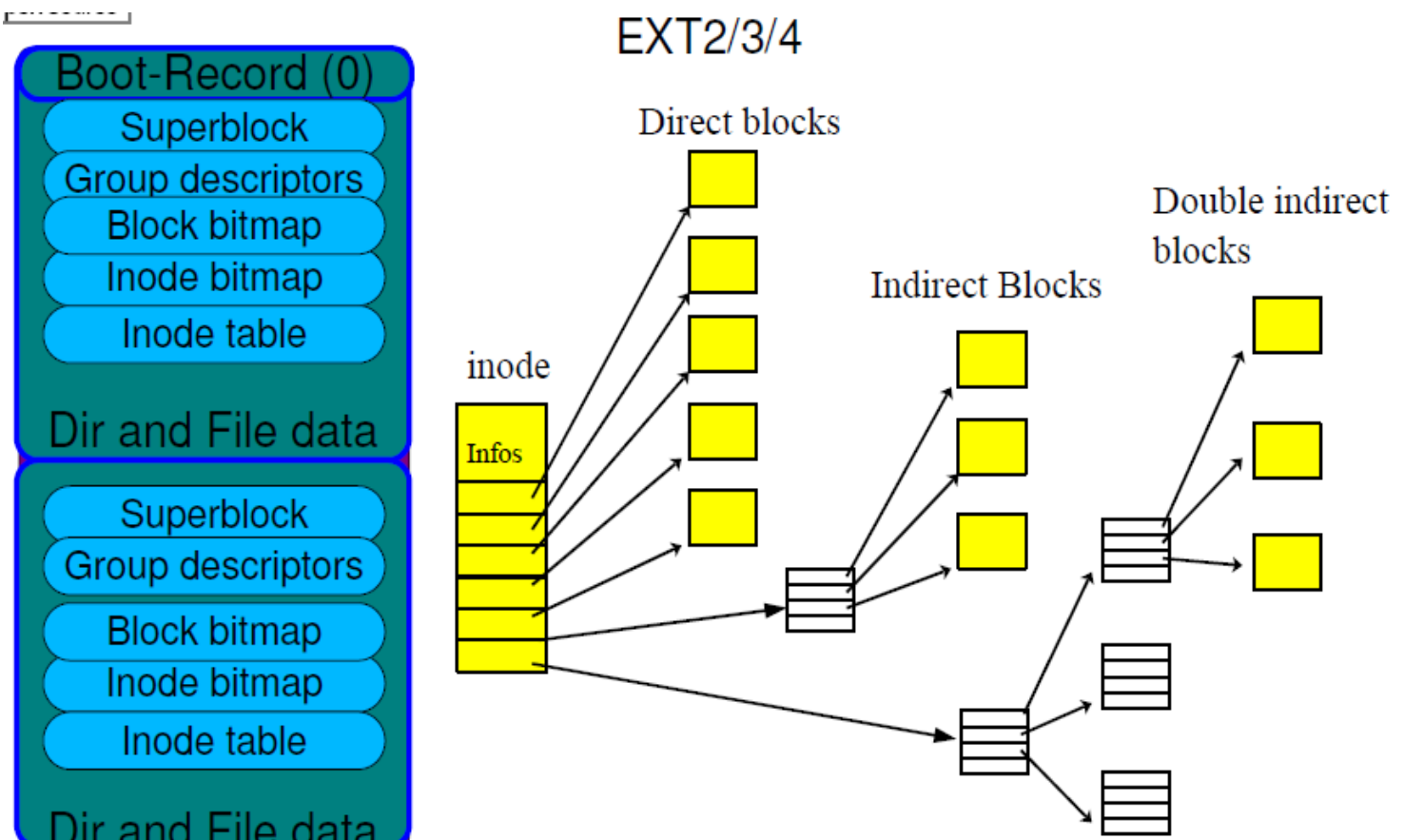
POSIX (*Portable Operating System Interface*) – standard používaný hlavně unixovými operačními systémy.

Jeho úkolem bylo vytvořit jednotné rozhraní, které mělo zajistit přenositelnost programů mezi různými hardwarovými platformami. Definuje rozhraní nejen pro programátory (API), ale i pro uživatele (v podobě utilit pro příkazovou řádku).

Vytvořením mnoha různých klonů původního **AT&T Unixu** došlo k vytvoření mnoha různých doplňkových systémových volání, funkcí, programů a demonů. Většina z nich byla uzavřená a často se i standardní programy lišily ovládáním. **Vznikla proto řada standardů POSIX (Portable Operating System Interface), které definují systémová volání, knihovní funkce a chování programů v POSIX kompatibilním operačním systému.** I přes tuto standardizační snahu je ve světě Unixu mnoho různých odchylek a vylepšení.

S jistotou je však možno prohlásit, že POSIX spolu s dalšími standardy přinesl řád.

File System



Původně **Linux** používal především diskové buffery, ale **vzhledem k požadavku na možnost mapování souborů do paměťových prostorů (mmap)** se data mohla objevit **ve dvou kopiích** (pro moderní VMM nepřijatelné) nebo se složitě po mmapu upravovaly buffer heads tak, aby směřovaly do stránek. Postupem času se proto zcela přešlo na **pagecache**.

mmap

alternativa k systémovému volání read, write (pro nesequenční volání a pokud jsou data

malá)

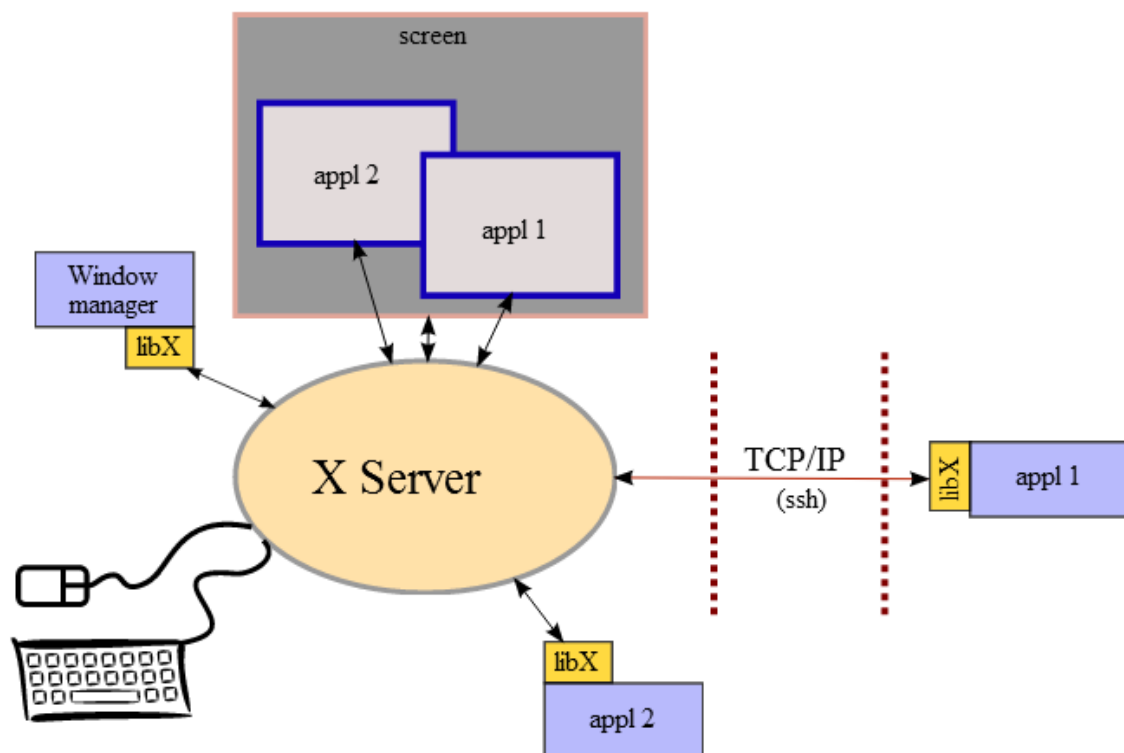
5) Okruh

Grafická uživatelské rozhraní (Qt, koncepce signal-slot, Gtk, koncepce X - display server).

Grafický subsystém v UNIXu

X Window System

- Současná podoba systému oken X se principiálně nemění a nová vydání jsou rozšiřující především z pohledu způsobů implementace grafických režimů.
- **Klient-server** architektura, klient (GUI aplikace) nemusí nutně běžet na stejném stroji, jako server.
- Server se stará o klávesnici, myš a zobrazení grafického výstupu aplikací.
- Klient komunikuje se serverem pomocí socketu. Používá [X-protokol](#).



Samotný **X Server** má na starost pouze zobrazení na obrazovku, obsluhu hardware (grafická karta, myš, klávesnice, touchpad) a sám o sobě je těžko použitelný.

Proto se používá spolu se **správce oken** (anglicky window manager), který se stará o ovládání uživatelem (přesun a změna velikosti okna) a podobně (horní lišta oken, ohraničení oken, změna velikosti, překrývání atp.).

Z hlediska X Serveru je ovšem správce oken jen další klient, a proto je snadno zaměnitelný. Pro Linux existuje několik desítek správců oken. (Metacity, Window Maker)

X protokol

- Specifikuje způsob komunikace mezi X-Serverem a X-Klienty.
- Nejstřednější implementací je Xlib.

Používat přímo xlib by bylo poněkud těžkopádné, proto vzniklo mnoho knihoven, které se snaží tvorbu GUI aplikací zjednodušit. Např. GTK+, Qt, HTML + DOM + JavaScript, ...

X Window System byl vyvinut na MIT v roce 1984 jako snaha o sjednocení GUI prostředí a přenositelnost grafických aplikací. Současná verze X11 pochází z roku 1987.

X Window System je důsledně zpětně kompatibilní (staré aplikace je možné zobrazovat i v novém prostředí).

Využívá model **klient-server**, kde vstupy a výstupy zajišťuje X Server ovládající příslušný hardware (grafická karta, myš, klávesnice) a klientem jsou jednotlivé aplikace. Pro komunikaci mezi aplikacemi a X Serverem je používán X protokol, který je přenášen pomocí IPC nebo TCP/IP

X Server však nezajišťuje logickou obsluhu zobrazovaných informací, a proto je pro správu zobrazených oken jednotlivých aplikací používán samostatný správce oken (anglicky window manager). Správce oken umožňuje uživateli organizovat zobrazené aplikace, resp. jejich okna (přesun a změna velikosti okna, horní lišta oken, ohraničení oken, překrývání atp.). Z hlediska X Serveru je správce oken jen další aplikace (klient), a proto je snadno zaměnitelný.

X Server je specifický program, který ovládá hardware a umožňuje tak vytvořit grafické uživatelské prostředí (GUI)

GTK+ (GIMP Toolkit) - grafická knihovna

Napsán v jazyce C

GPL licence.

Současná verze již multiplatformní.

Součást GNU projektu.

Použita pro prostředí Gnome

Využívá jej mnoho vyšších programovacích jazyků pro snadný wrapping.

QT (GIMP Toolkit) - grafická knihovna

Napsána v jazyce C++ => obtížnější wrapping.

Od samého počátku multiplatformní (dokonce vlastní make systém).

Signal/Slot koncept (implementace observer pattern).

Komerční a GPL licence

Implementují velké množství objektů, které se využívají při tvorbě aplikací jako kontajnery, grafické formáty, sockety, SQL konektivita, SSL, JavaScript interpreter, QML, ...

Model/View architektura.

Implicitně sdílené objekty.

Cenou, kterou za to platíme je MOC (Meta Object Compiler)

Signal-Slot vs. Callback

Důležitou vlastností Qt toolkitu je přítomnost **signálů a slotů pro komunikaci mezi objekty**; např. pokud se ve widgetu uskutečnila akce, která změnila jeho stav, tak o tom může být informován widget umístěný v jiném okně aplikace.

Signály a sloty tvoří velmi silný programátorský nástroj.

Místo signálů a slotů se dříve pro tento typ komunikace používal tzv. **callback**, což je **ukazatel na metodu objektu, kterou chceme vyvolat po nějaké události jiného objektu.**

Tento přístup měl dvě nevýhody. **Nebyla zde při volání typová kontrola a metody volané pomocí callback jsou silně vázané tzn.** Volaná metoda musí znát ukazatel na metodu, z které byla vyvolána.

Při používání signálů a slotů se tyto dvě nevýhody nevyskytují. **Pro jejich použití se definuje spojení signálu se slotem pomocí metody connect. V případě potřeby se pak spojené signály pouze "vyvolávají".**

Při propojování signálů a slotů může být s jedním slotem spojeno několik různých signálů a stejně tak na jeden signál napojeno několik slotů. Sloty mohou být použity pro přijímání signálů a zároveň mohou být použity jako standardní metoda objektu.

Model-view-controller (MVC)

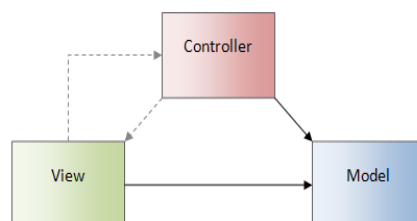
architektonický návrhový vzor

rozděluje datový model aplikace UI a logiku aplikace do tří nezávislých komponent tak, že modifikace některé z nich má jen minimální vliv na ostatní.

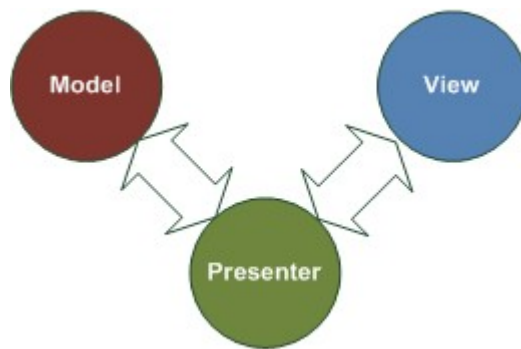
Model (model), což je doménově specifická reprezentace informací, s nimiž aplikace pracuje.

View (pohled), který převádí data reprezentovaná modelem do podoby vhodné k interaktivní prezentaci uživateli.

Controller (řadič), který reaguje na události (typicky pocházející od uživatele) a zajišťuje změny v modelu nebo v pohledu.



Model-View-Presenter



Model/View architektura(v QT)

Roli **Controlleru** přebírá View, resp. objekty **Delegate**.

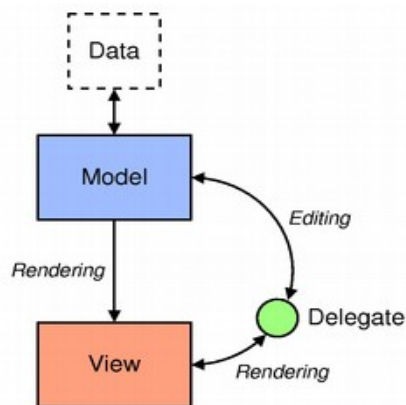
Úloha **Modelu** zůstává stejná jako ve standardním schématu MVC. Model musí implementovat alespoň minimální API, aby mohl spolupracovat s Qt View.

View data zobrazuje a nabízí základní editační nástroje (implicitní Delegate), pokud je podporuje Model, tj. pokud Model umí nebo má nastaven zápis dat.

Složitější interaktivní vstup dat spravují specializované instance **Delegate**.

Komunikace mezi jednotlivými komponentami probíhá přes signály a sloty.

Zobrazení dat je plně v režii view widgetu, proto není problém data jednoho modelu jednou zobrazovat jako tabulku, zároveň jako graf anebo třeba stromovou strukturu.



Delegát - zpracovává uživatelské vstupy, přičemž různé indexy modelu (sloupce, řady) mohou mít nastavené různé delegáty.

Umožňuje implementovat specifické zobrazování dat.

6) Okruh

Linux pro vestavé systémy, architektury procesorů, křížový překlad, vývojové nástroje, knihovny, aplikace Linuxu v průmyslu, multimediálních zařízeních a v bezpečnostně kritických aplikacích (nanokernely, virtualizace).

architektury procesorů

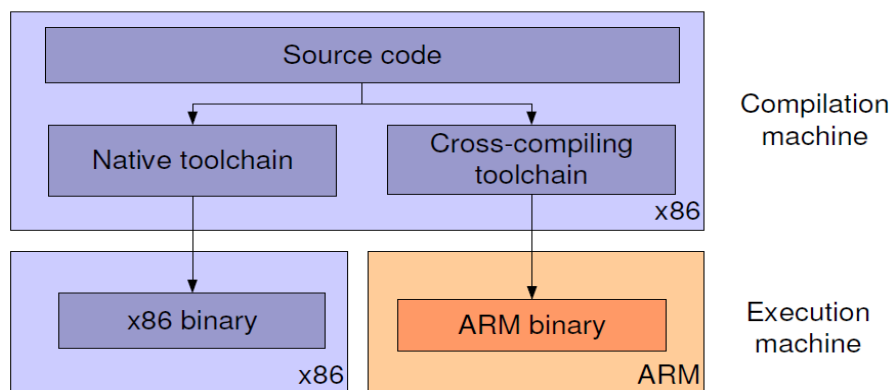
- x86
- x86_64
- ARM
- PowerPC
- Sparc

Křížový překlad

Křížový překladač - upravený kompilátor, který generuje kód spustitelný na jiné platformě, než na které je samotný překlad zdrojových kódů spuštěn.

Používá v případech, kdy jsou zdrojové kódy společné pro více cílových platform, na kterých může být program provozován (např.: Linux, Windows) nebo různé varianty téhož systému (16, 32, 64 bitový)

též využívána pro generování spustitelných souborů pro vestavěné systémy a při překladu pro platformy, které nejsou samy schopny kompilace (např.: jednočipové počítače bez OS nebo mobilní telefony).



Canadian Cross - technika vytvoření křížového kompilátoru pro jiné počítače. Na prvním počítači (A) je vytvořen kompilátor, který běží na druhém počítači (B). Na počítači (B) je potom finálně zkompileován program pro třetí počítač (C).

Přenositelnost kódu:

Je nutné **psát čistě** a používat jen to, co je jazykem deklarováno

Používat pokud to je možné **standardizovaná API** (např. POSIX)

Nepředpokládat pořadí byte/charů ve slově (little/bigendian)
endian.h: `__BYTE_ORDER`, `__LITTLE_ENDIAN`, `__BIG_ENDIAN`
byteswap.h: `bswap_16`, `bswap_32`

Nepředpokládat počet bitů v adresační jednotce (`CHAR_BIT`)

Nikdy nepřetypovat ukazatel na int a zpět, dokonce ani na long
(především, kvůli Win32_64), `intptr_t`, `uintptr_t`

Vnější síťové formáty vždy oddělovat od vnitřních

- Zarovnání struktur se může lišit
IDL (Interface description language)/External Data Representation (XDR)
- textové formáty
XML, XMLRPC, SOAP, HTML, JSON atd.
Pozor: zjednoduší řešení ale parsování je pomalé

External Data Representation (XDR)

Zajišťuje kódování/zabalení dat/hodnot způsobem, který je nezávislý na architektuře použitého počítačového systému.

Zakódovaná data mohou být přenášena heterogenním prostředím.

Encoding - kódování lokální reprezentace do přenosového XDR reprezentace

Decoding – převod XDR reprezentace do lokálního formátu použitému v systému příjemce

Remote procedure call (RPC, vzdálené volání procedur)

technologie dovolující programu vykonat proceduru, která může být uložena na jiném místě než je umístěn sám volající program. Příkladem budiž výpočet funkce na jiném počítači v síti.

Základní mechanismus pro budování klient/server řešení

Na straně klienta volání náhradní funkce – **klient stub**

Ta zabalí (**marshalling**) data do **XDR** a pošle přes přenosový kanál (TCP/IP).

Na straně serveru data přijata – **server stub** => dekódování(**unmarshalling**) do lokální podoby => volání uživatelské funkce serveru

Její návratová data zakódovaná do XDR => předaná klientovi

Corba:

Tvoří ucelenou architekturu pro podporu tvorby distribuovaných objektově orientovaných aplikací.

Základní funkcí architektury CORBA je **podpora jazykově neutrálního transparentního použití distribuovaných objektů**. Klient může používat dostupné objekty bez ohledu na to, v jakém jazyce jsou implementovány, kde a na jakém počítači běží a jakým komunikačním protokolem jsou dostupné. Objektem se přitom rozumí identifikovatelná, zapouzdřená entita, která poskytuje nějaké *služby*. Klient přistupuje k těmto službám zasíláním *požadavků*. Forma generování požadavků závisí na jazyce, ve kterém je klient napsán.

7) Okruh

Minimální sada komponent pro start a běh systému a interakci s ním (shell, core-utils, BusyBox).

BusyBox

Obsahuje minimalizované alternativy základních nástrojů jako ls, init, sh, getty až po minimalizovaný web server

Minimalizovaná C knihovna

GPL licence

Binárka < 500kB

Použití: ve vestavných zařízeních a během startu v distribucích využívajících initial ramdisk/initramfs

Podporované architektury: všechny – x86, x86_64, ARM, Sparc, PowerPC,...

Instalátory a spouštěcí ramdisky téměř všech distribucí Debian, Red Hat, Mandriva, ...

Vestavné distribuce: Amazon Kindle, ZyXEL Routers,...

Shell

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.

Bourn SHell (sh), **Bourne-Again Shell**(bash), **C shell** (csh), **Korn shell** (ksh)

Core-utils

GNU Core Utilities nebo též **coreutils** je balík softwaru z projektu GNU, obsahující mnoho základních nástrojů, jako jsou cat, ls a rm pro unixové systémy. Jde o kombinaci řady dřívějších balíčků, například **textutils**, **shellutils**, a **fileutils**, společně s různými dalšími utilitami.