Jihočeská univerzita v Českých Budějovicích

Bakalářská práce

České Budějovice 2007 **Adam Jalůvka**

Jihočeská univerzita v Českých Budějovicích

Pedagogická fakulta – Katedra fyziky

Linuxový terminálový server pro tenkého klienta + výpočetní klastr

Bakalářská práce

Vedoucí práce: Ing. Šerý Michal

Autor: Jalůvka Adam

Anotace

Cílem této práce je seznámit čtenáře s projekty LTSP a openMosix, na jejichž základech je vystavěno komplexní řešení terminálového serveru a výpočetního klastru s automatickým vyrovnáváním zátěže. Práce není zaměřena jen na oba projekty schopné nezávislé existence vedle sebe, ale i na jejich vzájemné propojení, přinášející další výhody, které by oddělené projekty nepřinesly. Jako první výhodu můžeme chápat snadné připojení nových uzlů do výpočetního klastru bez nutnosti instalace, jako výhodu druhou pak přispívání uzlů svými systémovými prostředky terminálovému serveru a ostatním klientům.

Annotation

The goal of this work is to aquaint the reader with projects LTSP and openMosix. The complex sollution of the terminal server and the computing cluster with an automatic load balancing is created on the basis of these projects. This work is not only aimed at projects which are already capable to stay independently next to each other, but it is also aimed at the linkage of projects which produce benefits. The first advantage is the ability of an easy connection of new nodes to the computing cluster without the need of installation. The second advantage is that nodes can contribute with its systems resources to the terminal server and other clients.

Prohlašuji že předloženou práci jsem vypracoval samostatně, pouze s použitím uvedené (citované) literatury.

V Českých Budějovicích dne 27. 11. 2007 ……

Obsah

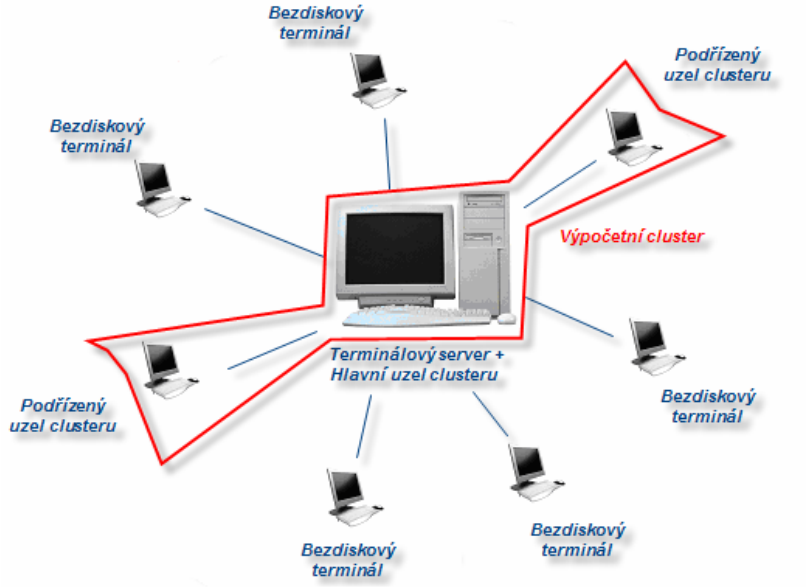
1 Úvod

Ve světě počítačového průmyslu se zdá, že čas běží rychleji než ve světě lidském. Během jednoho dvou roků se může uživatel seznámit s několika novými generacemi nejvýkonnějších počítačů, a v tom samém roce několik generací upadne do zapomnění. Pokrok se neustále zrychluje, a čím dál častěji se stává, že náš stolní počítač již nezvládne nejnovější software, který jsme si vyhlédli. Takové počítače většinou s pocitem špatně investovaných peněz dáváme do bazarů, nebo nevyužitých skříní, a to i přesto že hardware se nejen nestačil fyzicky opotřebovat, ale ani zaprášit. Naštěstí existuje způsob, jak sice zastaralé, ale fyzicky zdravé počítače využít, a to i pro náročné výpočty, se kterými se zapotí i ten nejmodernější počítač současné doby.

Hlavním cílem této absolventské práce je seznámit čtenáře s dvěmi technologiemi různě zaměřenými, které se však vzájemným spojením stanou jedna pro druhou nespornou výhodou. První z technologií je označena zkratkou **LTSP** z anglického spojení slov **Linux Terminal Server Project**. Technologie druhá nese název **openMosix**, kde anglický výraz „open**“** napovídá, že technologie pochází ze světa „otevřeného software“ a výraz „Mosix“ spíše skrývá informaci, že se jedná o výpočetní klastr (z anglického výrazu cluster = shluk, chumel, hvězdokupa) s automatickým vyrovnáváním zátěže.

Jednoduchým sečtením výhod LSTP+openMosix, získáme terminálové stanice (často známé pod názvem tenký klient), se schopností přispívat svými systémovými prostředky (výkonem procesoru a pamětí) jednomu velkému celku, klastru. Z podnikové, školní nebo laboratorní sítě, se tak snadnou cestou, bez instalace na koncové stanice, stane celek, který svými možnostmi může uspokojit studenta s potřebou rychle vyhledat vlakové spojení na internetu, sekretářku píšící dopis v kancelářské aplikaci i vědce požadujícího vysoký výkon pro složité výpočty.

Tato práce by měla dále ukázat, že dříve zmíněná tvrzení je možné nejen realizovat, ale i užitečným způsobem využít, v praxi, a že pořizovací cena veškerého potřebného softwarového vybavení je rovna nule.



Obr. č. 1 – Schéma výpočetního klastru v prostředí LTSP

2 Projekt LTSP

Linux Terminal Server Project (LTSP) [1] je instalační sada balíčků pro Linux, která nám umožňuje připojit se nevýkonným „tenkým klientem“ (terminálem) k serveru s operačním systémem Linux. Aplikace typicky běžící na serveru přijímají vstupní informace z klávesnice nebo myši tenkého klienta a po zpracování zobrazují svůj výstup na obrazovce tenkého klienta. Instalační sada LTSP je také dostupná jako součást kompletní distribuce např. K12LTSP, SkoleLinux a EduLinux. LTSP představuje základnu pro snadné vybudování sítě terminálových pracovních stanic, které bude později možné rozšířit o schopnost stát se plnohodnotným členem výpočetního klastru.

Klientská stanice může být nakonfigurována tak, aby pracovala v jednom ze tří režimů:

- **Grafické rozhraní X Windows systému**

  Použitím X Windows může být stanice používána pro přístup k jakékoliv aplikaci na serveru, nebo na jiných serverech v síti, a to v přehledném uživatelsky příjemném grafickém prostředí.

- **Znakový konferenční režim přes Telnet**

  Stanice může zahájit jedno či více *telnet* spojení se serverem, který tímto způsobem můžeme vzdáleně řídit. Každé sezení bude na vlastní virtuální obrazovce. Stisknutí *Alt-F1* až do *Alt-F9* bude přepínat mezi jednotlivými konferencemi sezení.

- **Příkazový řádek**

  Stanice může být nakonfigurována tak, aby poskytla příkazový řádek (*shell, bash*) přímo na terminálu. To je velmi užitečné při odstraňování problémů s X Windows nebo NFS, ale dá se využít i pro zadávání jednoduchých úloh nevyžadujících grafické ovládání.

3 Projekt openMosix

> Citát: *openMosix a „svět otevřeného softwaru postupuje obřími kroky. Je to svět kde slunce nikdy nezapadá a kde národní hranice, rasa a náboženství nemají žádný význam. Co se počítá je kód. A ten přichází často a ve vysoké kvalitě.“* ***Moshe Bar*** *(zakladatel a vedoucí projektu)*

Představení

OpenMosix [2] je rozšíření linuxového jádra pro paralelizaci samostatných systémů. Toto rozšíření přemění počítačovou síť složenou z běžných počítačů v „superpočítač“ pro linuxové aplikace.

Jakmile jednou nainstalujeme openMosix, nody (uzly) klastru začínají komunikovat mezi sebou a klastr sám sebe přizpůsobuje zátěži. Procesy pocházející z jakéhokoliv nodu, který je vůči ostatním přetížený, mohou migrovat na ostatní, zatímco openMosix soustavně optimalizuje přidělování systémových zdrojů. Toho je dosaženo tzv. záplatou (*patch*) pro linuxové jádro, která vytvoří spolehlivý základ pro rychlý, snadno dostupný, adaptivní a na náklady výhodný klastr s automatickým vyrovnáváním zátěže. Díky plně automatickému démonu (z angl. daemon) vestavěnému v openMosix mohou být průběžně za běhu klastru přidávány a odebírány nody, čímž dochází k rozšiřování systémových prostředků a zvyšování výkonu.

Není potřeba programovat aplikace speciálně pro openMosix, protože všechny openMosix rozšíření jsou uvnitř jádra. Každá linuxová aplikace automaticky získává výhody distribuovaných výpočtů koncepce openMosix. Klastr se chová podobně jako Symetrický Multi-Procesor, ale jeho řešení snadno přesáhne tisíce nodů, které sami o sobě mohou být založené na SMP.

OpenMosix komunita je velmi aktivní, přispívá přídavnými aplikacemi a sdílí užitečné informace se všemi uživateli. Stránka komunity a přídavných openMosix aplikací obsahuje seznam těchto sdílených aplikací. Všechny jsou pod licencí GPL.

Historie

OpenMosix je „Open Source“ projekt pod licencí GPLv2 navazující na projekt MOSIX. Nová verze projektu MOSIX se stala licencovaným softwarem koncem roku 2001. OpenMosix vydal pan Moshe Bar 10. února 2002, aby udržel vysoce považované linuxové klastrové řešení jako svobodný software. OpenMosix si rychle získal mezinárodní tým dobrovolníků, který ovládl předchozí verze kódu postavené na GPL a okamžitě vylepšil a rozšířil toto řešení. Uživatelé celosvětově přijali openMosix za svůj, aby ochraňoval jejich investice a rozšířil klastrové operace.

4 Server

4.1 Linuxová distribuce

V dnešním světě otevřených operačních systémů používajících jádro Linux, vyrůstá ročně mnoho „příbuzných“, více či méně kompatibilních, systémů s rozmanitým aplikačním vybavením podle cílové skupiny uživatelů nebo použití. Těmto „příbuzným“ se v profesionální terminologii říká „distribuce operačního systému Linux“. Jak již bylo naznačeno, distribucí s různým zaměřením je nemalá skupina, a proto je nutné před výběrem konkrétní distribuce zvážit zda se vůbec pro náš projekt hodí či nikoliv.

Doporučení pro výběr distribuce

- První spolu související kritéria, která ovlivňovala můj výběr distribuce jsou otevřenost distribuce a cena. Z toho vyplývá že distribuce by měla být z řad „otevřeného software“ (open source) a zdarma. Kromě ceny jsou další výhody domyslitelné. Open source distribuce bývají velmi dobře zdokumentovány, na jejich vývoji se zpravidla podílí více vývojářů i z různých zemí, takže zdrojové kódy jsou přehledné, okomentované a neustále k dispozici na internetu. Taková distribuce je většinou obklopena větší uživatelskou základnou a z toho vyplývající komunitou nadšenců, odborníků a přispěvatelů různých internetových diskuzí, tématicky zaměřených na „tu svojí“ distribuci. Překážky které by mohly nastat při instalaci nebo konfiguraci neběžných řešení, se pak snadněji překonávají a podpora ze strany nadšenců je dle mých vlastních zkušeností k nezaplacení.

- Distribuce by měla být vyzrálá. Tímto výrokem není myšleno, že by naše distribuce měla obsahovat nejnovější technologie, výstřelky techniky, či podporu posledního typu hardware. Pod pojmem vyzrálost si představuji odladěné jádro systému, stabilitu v distribuci obsažených aplikací, přítomnost minimálního množství experimentálních aplikací, aplikované bezpečnostní záplaty a ostatní aktualizace. Podle jmenovaných vlastností je zřejmé, že by se nemělo jednat o distribuci která měla premiéru minulý týden a ohlédneme se raději po již déle známé vyzkoušené distribuci.

- Z hlediska zátěže na systémové prostředky jsem dále postupoval ve výběru spíše směrem k praktickému využití aplikací než k animovanému vykreslování oken. Nejmodernější správci oken z rodin Gnome a KDE by z hlediska nároku na výkon měli zůstat zapomenuti.

- Jedním z nejvíce limitujících kritérií, kterého jsem se musel držet, bylo přihlédnutí k požadavkům rozšíření openMosix pro jádro Linuxu. Poslední verze openMosix označená jako stabilní, byla vydána pro jádro řady 2.4.x, čímž se posouváme ve výběru o několik let zpět. Ne proto, že by starší verze jádra nebyla schopná běžet nad moderní distribucí z poslední doby, ale proto, že by se muselo přistoupit k instalaci dalšího podpůrného softwaru (např: překladače

GCC) takových verzí, které nová distribuce pravděpodobně již nebude obsahovat. V případě že by bylo z nějakého důvodu potřeba realizovat výpočetní klastr na jádru řady 2.6.x, běžně obsaženém v nových distribucích, bylo by vhodné počkat na vývoj openMosix pro tuto řadu jádra, nebo se obrátit směrem ke komerčním řešením bratrského projektu MOSIX.

- Poslední ne nevýznamnou podmínkou výběru distribuce byla jazyková lokalizace.

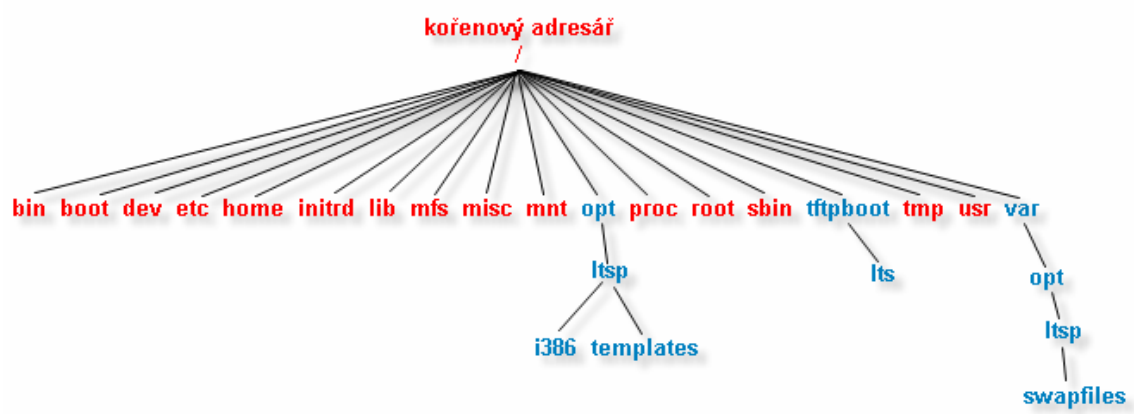
Pro testování terminálového serveru a výpočetního klastru openMosix jsem zvolil distribuci Red Hat 9 s těmito vlastnostmi:

- otevřená dobře dokumentovaná distribuce
- volně ke stažení na internetu
- velmi rozšířená a pro uživatele známá
- velmi dobrá podpora ze strany projektů LTSP, openMosix
- do češtiny lokalizované aplikace (Mozilla, Open Office)
- obsahuje všechny nástroje vhodné pro kompilaci jádra řady 2.4.x
- podpora velkého množství hardwaru
- minimální hardwarové nároky a velikost instalace (vhodná pro testování ve virtuálních prostředí VMWare, Virtual PC)

4.2 Adresářová struktura

Adresářová struktura unixových operačních systémů se výrazně liší od toho, na co jsou uživatelé zvyklí z operačních systémů typu Windows. Struktura na první pohled působí nelogicky a bez znalostí, k jakému účelu adresář slouží, není možné rozlišit zda se jedná o adresář, který je nezbytný pro samotný systém, nebo byl dodatečně vytvořen některým z uživatelů. Přesto je hierarchie i struktura jasně dána, a i když můžeme u různých linuxových distribucí od různých výrobců čekat dílčí rozdíly, ty důležité adresáře najdeme na svém místě. Je tomu tak proto, že je udržován tzv. „standard hierarchie souborového systému“ (Filesystem Hierarchy Standard), ze kterého všichni vývojáři vycházejí. Tento standard je neustále udržován a patentován týmem složeným z pánů Daniel Quinlan, Paul „Rusty“ Russell, Christopher Yeoh. Jeho aktuální verzi je možné nalézt na internetové adrese http://www.pathname.com/.

V této kapitole si uvedeme zkrácený přehled a popis adresářové struktury LTSP serveru s důrazem na adresáře, které mají pro jeho funkčnost speciální význam.



Obr. č. 2 – Adresářová struktura s adresáři významnými pro LTSP

/ – kořenový adresář systému (root)

/bin – binární soubory a příkazy systému

/boot – statické soubory zavaděče systému, jádro systému a ostatní soubory nezbytné pro start systému. Velmi často je do tohoto adresáře připojován (viz. konfigurační soubor */etc/fstab*) samostatný diskový oddíl obsahující všechny výše zmíněné soubory. Zvyšuje se tím bezpečnost a oddělí se jádro operačního systému od zbylé aplikační části a knihoven systému.

/dev – soubory zařízení (speciální soubory zprostředkovávající přístup k hardwaru počítače a systémovým prostředkům)

/etc – specifické konfigurace pro operační systém, služby systému a některé aplikace

/home – adresář s domácími složkami jednotlivých uživatelů

/initrd – při startu systému zde bývá dočasně připojen ramdisk

/lib – nezbytné sdílené knihovny a moduly jádra

/mfs – souborový systém mosix – výpočetní klastr openMosix používá pro zvýšení výkonu při výměně informací mezi jednotlivými uzly klastru vlastní typ souborového systému. Souborový systém MFS je připojen do tohoto adresáře.

/misc – adresář s různorodou architekturou, nezávislá data

/mnt – složka pro připojování souborových systémů připojovaných přes NFS, výměných médií cdrom, disketových mechanik, flashdisků atd.

/opt – určená pro instalace volitelných aplikací

../ltsp/ – kořenový adresář instalace LTSP

../i386 – zde je obsažen celý adresářový strom tenkého klienta, který je později připojen pomocí služby NFS.

../templates – inicializační soubory a soubory nutné pro počáteční konfiguraci

/tftpboot – kořenový adresář pro tftp server

../lts – složka obsahuje překompilovaná jádra, která se distribuují jednotlivým tenkým klientům a upravené obrazy rom síťových karet.

/var – složka souborů s proměnným obsahem jako jsou *.log* soubory a dočasné soubory

../opt/ltsp/swapfiles – pokud je terminál nakonfigurován s podporou odkládacích souborů vytvářejí se zde dočasné soubory (vhodné v případě kdy terminál nedisponuje dostatečným množstvím paměti RAM).

4.3 Služby a protokoly

4.3.1 DHCP

Dynamic Host Configuration Protocol je aplikační protokol patřící do rodiny TCP/IP. Může být použit pro automatické přidělování různých parametrů koncovým stanicím v síti. Nejčastěji je používán pro přidělování síťových parametrů jako je IP adresa, síťová maska, adresa standardního směrovače (default gateway) a adresy DNS serverů. Ve velkých sítích je vhodné posílat třeba i adresy doporučených NTP, WINS, SMTP serverů. Využití protokolu rozšiřuje i možnost posílat uživatelsky definované parametry, a to bez toho, aby pro klienta neznámé parametry způsobovaly otravná chybová hlášení. Parametry, kterým daný klient nerozumí, jsou jednoduše ignorovány, což nám umožňuje vytvoření různorodé sítě.

Výhody DHCP protokolu

- nastavení parametrů je automatické a nezatěžuje uživatele
- zaručuje, že se na síti nevyskytnou dvě stanice se stejnou IP adresou (konflikt IP adres)
- přečíslování sítě lze iniciovat vzdáleně bez zásahu do práce uživatelů
- DHCP je schopné předat všechny parametry nezbytné pro naše linuxové terminály

DHCP vychází ze staršího BOOTP protokolu, který však přiděloval adresy na neomezenou dobu. DHCP je s BOOTP zpětně kompatibilní, je však třeba brát na zřetel zmíněné neomezení doby přidělení adresy, které by v některých sítích nemuselo být žádoucí.

Jak DHCP funguje

Klient komunikuje na UDP portu 68, server naslouchá požadavkům na UDP portu 67. Po připojení do sítě klient vyšle pomocí všesměrového vysílání (*broadcast*) paket DHCPDISCOVER. Pokud má DHCP server statický záznam pro stanici s touto MAC adresou, nebo je server nakonfigurován tak, aby přiděloval adresy libovolně z určitého rozsahu, nabídne IP adresu klientovi paketem DHCPOFFER. Klient si potvrzení nabízené adresy vyžádá paketem DHCPREQUEST a server mu ji vzápětí schválí

odpovědí DHCPACK. Obdrží-li klient toto potvrzení, může IP adresu a ostatní zaslané parametry začít používat. Po uplynutí doby zapůjčení musí klient dostat nové DHCPACK potvrzení, že IP adresu smí používat.

DHCP je první z protokolů, který terminály využívají během startovacího procesu. Je tedy vhodné, vzhledem k případnému řešení vzniklých problémů, nakonfigurovat tento protokol na serveru jako první. Ve skutečnosti se budou terminály během startovacího procesu dotazovat serveru na informace dvakrát. Jednou, aby získali informaci, které jádro stáhnout a spustit. Další požadavek nastane tehdy, když jádro znovu požaduje přidělení IP adresy a informace o kořenovém souborovém systému připojovaným přes NFS.

Důležitá nastavení dhcpd.conf

- *option subnet-mask 255.255.255.0;* – posílá klientovi síťovou masku.
- *option broadcast-address 192.168.1.255;* – posílá klientovi adresu pro všesměrová vysílání.
- *option routers 192.168.1.254;* – posílá klientovi adresu výchozí brány.
- *option domain-name-servers 192.168.1.254;* – posílá klientovi adresu DNS serverů.
- *option domain-name „ltsp.org“* – posílá klientovi jméno příslušné domény.
- *get-lease-hostnames true;* – oznamuje *dhcpd* démonovi, aby nahlédl do souboru */etc/hosts* nebo do DNS, a aby přiřadil stanici vyhrazené doménové jméno.
- *next-server 192.168.1.15;* – tato volba říká klientovi, který server obsluhuje službu TFTP. Pokud používáme *dhcpd* verze 3.0.2 nebo starší, nemusíme zahrnout tuto řádku v případě, že TFTP server je na stejném stroji jako DHCP. Pokud je verze *dhcpd* novější, tak tato volba musí být bezpodmínečně zahrnuta.
- *option root-path „192.168.0.254:/opt/ltsp/i386“;* – sdělí klientovi dvě informace:
    1) IP adresu NFS serveru
    2) adresář který má být připojen jako kořenový souborový systém klienta
- *subnet 192.168.1.0 netmask 255.255.255.0 {...}* – deklarace podsítě je vyžadována pro každé síťové rozhraní, na kterém poslouchá *dhcpd*. Tento příklad ukazuje samostatnou síť, což ukazuje že *dhcpd* poslouchá pouze na jednom rozhraní.
- *range 192.168.1.100 192.168.1.199;* – toto nastavení říká *dhcpd* aby dynamicky přiděloval adresy z rozsahu adres končících na .100-.199

- *if substring (option vendor-class-identifier, 0, 9)=„PXEClient“ {...}* – podmínka kontroluje hodnoty prvních 9 bajtů identifikátoru výrobce (vendor-class-identifier). Tímto způsobem lze zjistit, jestli požadavek DHCP-REQUEST pochází od PXE bootrom, nebo od Etherboot bootrom. Tuto otázku potřebujeme mít zodpovězenou, protože každý typ bootrom potřebuje stáhnout z TFTP serveru rozdílný soubor.

- *filename "/lts/eb-5.3.2-amdhomepna.zpxe";* – zde je uveden název souboru Etherboot bootrom, která má nahradit PXE bootrom, neschopnou stáhnout celé linuxové jádro (bootrom omezení na 64k)

- *filename "/lts/omosix1";* – zde je uveden název obrazu souboru (image), který obsahuje linuxové jádro spolu s inicializačním ramdiskem. Tento soubor byl vytvořen z původně odděleného souboru linuxového jádra a obrazu ramdisku pomocí nástroje *mknbi-linux*.

```
# soubor /etc/dhcpd.conf
# dhcpd.conf
ddns-update-style             ad-hoc;
option subnet-mask            255.255.255.0;
option broadcast-address      192.168.1.255;
option routers                192.168.1.15;
option domain-name-servers    192.168.1.15;
option domain-name            "ltsp.org";
option option-128 code 128 = string;
option option-129 code 129 = text;
use-host-decl-names on;
get-lease-hostnames           true;
next-server                   192.168.1.15;
option root-path              "192.168.1.15:/opt/ltsp/i386";
subnet 192.168.1.0 netmask 255.255.255.0 {
#    range   192.168.0.100   192.168.0.199;
#-----------------------------------------------------------
host Term1 {
    hardware ethernet 00:0C:29:2D:89:6F; # Term1 VMWare
    fixed-address 192.168.1.17;
    option option-128     e4:45:74:68:00:00; # toto neni MAC adresa
    option option-129     "init=/linuxrc";
    if substring (option vendor-class-identifier, 0, 9) = "PXEClient"
{
        filename "/lts/eb-5.3.2-amdhomepna.zpxe"; # Etherboot bootrom
sitove karty AMD
    }
    else{
        filename "/lts/omosix1"; # jadro terminalu s openMosix
    }
}
#-----------------------------------------------------------
host Term2 {
    hardware ethernet 00:0C:29:32:4D:2B;
    fixed-address 192.168.1.106;
    option option-128     e4:45:74:68:00:00; # toto neni MAC adresa
    option option-129     "init=/linuxrc";
    if substring (option vendor-class-identifier, 0, 9) = "PXEClient"
{
        filename "/lts/eb-5.3.2-amdhomepna.zpxe"; # Etherboot bootrom
sitove karty AMD
    }
```

```
    else{
        filename "/lts/omosix1"; # jadro terminalu s openMosix
    }
}
#-------------------------------------------------------------
}
```

Kontrola DHCP serveru

Zkontrolovat zda DHCP server běží můžeme následujícím příkazem:

```
[root@Hive /]# netstat -apn | grep ":67"
udp        0      0 0.0.0.0:67              0.0.0.0:*
1174/dhcpd
[root@Hive /]# _
```

Obr. č. 3 – Výstup programu netstat

Jestliže DHCP server neběží, můžeme jej spustit příkazem:

```
[root@Hive /]# /etc/init.d/dhcpd start
Startuji dhcpd:
[root@Hive /]# _
```

Obr. č. 4 – Restart služby dhcpd

Potom podruhé zkontrolujeme, zda je vše v pořádku příkazem *netstat*. Jakmile DHCP server běží, můžeme se přesunout k dalšímu kroku, kterým je konfigurace TFTP serveru.

4.3.2 TFTP

Trivial File Transfer Protocol (TFTP) definovaný roku 1980 je velmi jednoduchý přenosový protokol s velmi zúženou funkcionalitou podobnou klasickému FTP. Díky své jednoduchosti je snadný na implementaci a zabírá velmi malé množství paměti, takže je i dnes stále používaným protokolem pro přenos malých souborů po síti.

Základní vlastnosti

- pro svou funkci používá UDP port 69 (klasické FTP používá TCP port 21)
- neumí listovat v adresářové struktuře
- neobsahuje žádný systém ověřování uživatelů
- nemá implementován šifrovací algoritmus
- lze jej použít pouze pro komunikaci typu čtení/zápis souboru ze/na server(u)

Popis přenosu

1. Počítač A (dále jen A), který iniciuje komunikaci vyšle síťový paket RRQ (požadavek na čtení) nebo WRQ (požadavek pro zápis) počítači B (dále jen B) na portu číslo 69 spolu s informací o názvu souboru, kterého se požadavek týká, a informací o módu přenosu

2. B odpoví paketem ACK (potvrzení přijetí) na WRQ nebo přímo datovým paketem na RRQ. Paket je odeslán na čerstvě alokovaném dočasném portu, stejně tak jako všechny následující pakety pro B.

3. Počítač, který vystupuje v komunikaci jako hostitel (ten kde je požadovaný soubor uložen), zasílá očíslované datové pakety cílovému počítači až na poslední paket, který již obsahuje data bez očíslování. Cílový počítač potvrzuje přijetí každého z datových paketů ACK paketem.

4. Poslední datový paket musí obsahovat méně dat než je maximální velikost datového bloku v paketu. Jestliže velikost dat je přesný násobek velikosti datového bloku, je vyslán poslední paket uzavírající komunikaci s datovým blokem obsahujícím 0 bytů dat.

4.3.3 NFS

Network File System je síťový protokol, který terminály používají pro připojení souborového systému LTSP ze serveru. Během procesu startování terminál nejdříve připojí exportovaný adresář a následně provede přesun kořenového adresáře. Pak pokračuje v procesu *sysinit*, který načítá specifické konfigurace pro terminál ze souboru *lts.conf*.

Nastavení exportu adresářů

Náš LTSP server musí být nakonfigurován tak, aby exportoval několik adresářů přes NFS. Potřebujeme se ujistit že soubor */etc/exports* obsahuje následující záznamy. To provedeme výpisem souboru */etc/exports*, jak ukazuje obrázek č. 5.

```
[root@Hive /]# cat /etc/exports
## LTS-begin ##

#
# The lines between the 'LTS-begin' and the 'LTS-end' were added
# on: Út říj 17 17:29:16 CEST 2000 by the ltsp installation script.
# For more information, visit the ltsp homepage
# at http://www.ltsp.org
#

/opt/ltsp/i386                  192.168.1.0/255.255.255.0(rw,no_root_squash,sync
)
/var/opt/ltsp/swapfiles         192.168.1.0/255.255.255.0(rw,no_root_squash,asyn
c)

#
# The following entries need to be uncommented if you want
# Local App support in ltsp
#
/home                   192.168.1.0/255.255.255.0(rw,no_root_squash,sync)

## LTS-end ##
[root@Hive /]# _
```

Obr. č. 5 – Výpis souboru /etc/exports

Adresář */opt/ltsp/i386* je nezbytný pro start terminálu a musí se shodovat s cestou v souboru *dhcpd.conf* u parametru *option root-path* (např. *option root-path = „/opt/ltsp/i386“;*)

Kontrola funkčnosti serveru

Pro otestování, zda NFS na našem serveru běží, je vhodné použít několik příkazů.

Pro otestování zda v systému běží všechny NFS služby, spustíme skript */etc/init.d/nfs* s následujícím výstupem:

```
[root@Hive tmp]# /etc/init.d/nfs status
rpc.mountd (pid 1165) běží...
nfsd (pid 1159 1158 1157 1156 1155 1154 1153 1150) běží...
rpc.rquotad (pid 1146) běží...
[root@Hive tmp]# _
```

Obr. č. 6 – Zjištění stavu služby nfs

Pokud chceme zjistit které adresáře server exportuje, použijeme příkaz *showmount*:

```
[root@Hive tmp]# showmount 192.168.1.15 -e
Export list for 192.168.1.15:
/home                   192.168.1.0/255.255.255.0
/opt/ltsp/i386          192.168.1.0/255.255.255.0
/var/opt/ltsp/swapfiles 192.168.1.0/255.255.255.0
[root@Hive tmp]# _
```

Obr. č. 7 – Výpis složek exportovaných na serveru

4.3.4 XDMCP

X Display Manager Control Protocol je poslední z protokolů, který terminál použije při startovacím procesu, takže je vhodné konfigurovat tento protokol jako poslední. Jestliže nastane situace kdy náš terminál je schopen nastartovat, ale zastaví se na šedé obrazovce s velkým „X“ kurzorem myši, můžeme si být jistí, že jediné co mezi námi a úspěšným startem stojí je právě XDMCP protokol. Konfigurace XDMCP není stejná na všech serverech. Záleží na tom jakého správce displeje (display manager - DM) na serveru používáme.

Jestliže jsme schopni se přihlásit k serveru z lokální konzole v grafickém prostředí, máme jistotu že DM běží korektně a můžeme se věnovat konkrétním příkladům konfigurace pro tři nejznámější správce displeje XDM, GDM a KDM. V moderních distribucích jsou většinou obsaženi všichni tři jmenovaní, takže se nemusíme obávat že bychom si nevybrali. Nejstarší z nich XDM je dnes již mnohaletým standardem a je obsažen již v samotné distribuci X Windows systému. GDM je manažer displeje dodávaný s prostředím Gnome a KDM s prostředím KDE. Pokud již máme Linux nainstalován a nejsme si jisti, který ze jmenovaných používáme, stačí nám pro zodpovězení této otázky spustit následující příkaz.

```
[root@Hive /]# ps -e | grep ".dm"
 1338 ?        00:00:00 gdm-binary
 1387 ?        00:00:00 gdm-binary
 1398 ?        00:00:00 gdmgreeter
[root@Hive /]# _
```

Obr. č. 8 – Výstup programů ps a grep

Měli bychom vidět výpis, který ukazuje, že jako manažer displeje používáme GDM.

Jak zjistit že je XDMCP dostupný

Abychom zjistili že displej manažer naslouchá XDMCP požadavkům, použijme následující příkaz:

```
[root@Hive /]# netstat -apn | grep ":177"
udp        0      0 0.0.0.0:177             0.0.0.0:*
1338/
[root@Hive /]# _
```

Obr. č. 9 – Zjištění stavu XDMCP protokolu

Jestliže není XDMCP dostupný, měli bychom se ujistit, že jsme ve správné úrovni běhu (runlevel 5). Pokud nejsme, snadno se do ní přepneme příkazem */sbin/init 5*. Po zprovoznění XDMCP, by měli všechny terminály, původně končící na šedé obrazovce, zobrazovat grafické přihlašovací okno.

4.4 Lokální aplikace z pohledu serveru

4.4.1 Parametry souboru lts.conf

V souboru *lts.conf* je nutné nastavit několik voleb:

LOCAL_APPS - Volba musí být nastavena na hodnotu „Y“. To způsobí následující změny ve spouštění pracovní stanice:

- přes NFS se připojí adresář */home*
- na stanici se vytvoří soubor */var/yp/nicknames*
- na stanici se spustí program *portmapper*
- spustí se démon *xinetd*
- na stanici se vytvoří konfigurační soubor */etc/yp.conf*
- „domainname“ se nastaví na hodnotu parametru *NIS_DOMAIN* ze souboru *lts.conf*
- na stanici se spustí démon *ypbaind*

NIS_DOMAIN - Všechny pracovní stanice na síti, které chtějí být spojeny s konkrétním NIS serverem musí patřit do stejné NIS domény. Toto nijak nesouvisí s doménou DNS, proto musí být nastaven parametr *NIS_DOMAIN* nastavující doménu, do které bude pracovní stanice patřit.

NIS_SERVER - Pokud tento parametr není definován v *lts.conf*, pak se NIS na stanice pokouší vyhledat NIS server všesměrovým vysíláním, ke kterému se po nalezení připojí. Toto chování můžeme ovlivnit definováním parametru *NIS_SERVER* v souboru lts.conf, kterým můžeme klientovi situaci ulehčit a vybrat konkrétní NIS server pro připojení.

4.4.2 Síťová informační služba NIS

NIS je služba typu klient/server. Na serveru standardně běží démon, přijímající žádosti od klientů, v našem případě terminálových stanic. Na pracovní stanici běží proces nazvaný *ypbind*. Vždy, když na pracovní stanici vznikne potřeba získání informací o uživateli, ověření hesla nebo vyhledání domovského adresáře, použije stanice *ypbind* pro spojení s procesem *ypserv* na serveru. Jestliže je v naší síti již NIS server provozován, není třeba tuto konfiguraci na LTSP serveru provést a stačí pouze doplnit parametry *NIS_DOMAIN* a *NIS_SERVER* tak, aby odpovídali našemu aktuálně nastavenému schématu. V opačném případě je nutné provést konfiguraci *ypserv* démona.

4.4.3 Konfigurace lokální aplikace

Pro nastavení lokálně běžící aplikace je třeba znát všechny součásti aplikace a zpřístupnit je pracovní stanici. Není možné jen jednoduše exportovat adresáře, které aplikace využívá pro svůj běh na serveru. Hlavní skutečností, kterou si musíme uvědomit, je, že pravděpodobně nebudeme mít v naší síti stanice stejné architektury

jako je architektura serveru. Například by nastala situace, kdy by se aplikace běžící na stanici s CPU Pentium (architektura i586), dotazovala na knihovny serveru s CPU Pentium II (architektura i686), a to by nebylo dobré pro kompatibilitu aplikací. Nejčistší způsob jak toto vyřešit, je mít připravený kompletní strom se všemi binárními soubory a knihovnami, které pracovní stanice potřebuje, nezávislý na binárních souborech a knihovnách serveru. Konfigurace pro lokální aplikace tedy vyžaduje, aby všechny zmíněné soubory byly vloženy do stromu adresářové struktury stanice */opt/ltsp/i386/*.

Jako příklad si uvedeme aplikaci *gaim*, která je AOL komunikačním klientem umožňujícím komunikovat s dalšími lidmi na AOL diskuzních fórech.

- Prvním krokem je nalezení binárního souboru *gaim*, což je spustitelný soubor aplikace

- V dalším kroku nad ním spustíme program *ldd* (viz obrázek č.10), který nám vypíše seznam všech knihoven, včetně cest, které program *gaim* používá.

- Poté je nutné všechny zjištěné soubory nakopírovat do stromu stanice tj. */opt/ltsp/i386/...*

```
[root@Hive etc]# ldd /usr/bin/gaim
        libnsl.so.1 => /lib/libnsl.so.1 (0x40034000)
        libgdk_pixbuf.so.2 => /usr/lib/libgdk_pixbuf.so.2 (0x40049000)
        libgtk-1.2.so.0 => /usr/lib/libgtk-1.2.so.0 (0x4005f000)
        libgdk-1.2.so.0 => /usr/lib/libgdk-1.2.so.0 (0x401a7000)
        libgmodule-1.2.so.0 => /usr/lib/libgmodule-1.2.so.0 (0x401df000)
        libglib-1.2.so.0 => /usr/lib/libglib-1.2.so.0 (0x401e2000)
        libdl.so.2 => /lib/libdl.so.2 (0x40208000)
        libXi.so.6 => /usr/X11R6/lib/libXi.so.6 (0x4020b000)
        libXext.so.6 => /usr/X11R6/lib/libXext.so.6 (0x40213000)
        libX11.so.6 => /usr/X11R6/lib/libX11.so.6 (0x40221000)
        libesd.so.0 => /usr/lib/libesd.so.0 (0x40300000)
        libaudiofile.so.0 => /usr/lib/libaudiofile.so.0 (0x40308000)
        libm.so.6 => /lib/libm.so.6 (0x4032e000)
        libc.so.6 => /lib/libc.so.6 (0x40350000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
[root@Hive etc]# _
```

Obr. č. 10 – Nalezení knihoven používaných programem gaim pomocí ldd

4.4.4 Spuštění lokálních aplikací

Program pro X Windows se typicky spouští tam kde běží okenní manažer. Jestliže okenní manažer běží na serveru, pak i aplikace bude spuštěna zde, a pouze vstupy/výstupy budou přesměrovány na stanici. Pokud chceme lokálně běžící aplikaci, toto chování musíme pozměnit. Toho docílíme programem *rsh*.

Příklad spuštění *gaim* na pracovní stanici:

```
HOST`echo $DISPLAY | awk –F: ‘{ print $1 }’`
rsh ${HOST} /usr/bin/gaim –display ${DISPLAY}
```

Výše uvedený příklad může být zadán v okně programu *xterm*, nebo může být spuštěn pomocí skriptu a ikony z pracovní plochy.

4.5 Instalace serveru

4.5.1 Instalace a konfigurace LTSP

Základní prostředí pro LTSP je tvořeno čtyřmi skupinami instalačních balíčků, které jsou ke stažení dostupné na stránkách projektu *www.ltsp.org*.

- Core – jádro LTSP
- Linux kernel – upravená linuxová jádra pro klienty
- X Windows – systém pro grafické terminály na založené na X Windows
- Local apps – podpora pro lokální aplikace

Balíčky jsou do skupin rozděleny podle základní funkčnosti a zvyšují variabilitu instalace. Ne každý chce provozovat ve svém prostředí lokální aplikace nebo grafické terminály, stejně tak může někdo sáhnout k vlastní kompilaci jádra pro klienty. První balíček *ltsp_core-3.0.9* je nezbytný pro všechny typy instalací LTSP, zajišťuje na serveru základní funkcionalitu a základní konfigurace linuxové distribuce. Mimo čtyři základní instalační balíčky budeme, v souvislosti s grafickými kartami, kterými jsou osazeny naše terminálové stanice, potřebovat instalovat odpovídající balíček s ovladači příslušející k použitému hardwaru. Dalšími volitelnými, ale neméně užitečnými balíčky mohou být podpora zvukových karet na terminálu, podpora web kamer, snímacích zařízení, bezdrátových síťových karet a dalších vymožeností počítačového průmyslu. Většina LTSP balíčků je dostupná ve formátech RPM, TGZ nebo DEB, což nám dává svobodu v rozhodování jakou instalační metodu použít. Balíčky ve formátech RPM a DEB se instalují jednoduchým spuštěním, instalace komprimovaného formátu TGZ je o trochu složitější a je třeba provést několik kroků navíc. Součástí každého instalačního balíčku je textový soubor *INSTALL* nebo *README* popisující jak instalovat.

Pro instalaci z archívů TGZ, kterou jsem zvolil pro jejich univerzálnost, jednoduše provedeme několik příkazů dle následujícího postupu:

```
cd /install
tar xvzf ltsp_core-3.0.9-i386.tgz   // dekomprimace tgz archívu
cd ltsp_core
./install.sh                        // spuštění instalačního skriptu
```

Základní balíček jádra (ltsp_core)

- ***ltsp_core-3.0.9-i386.tgz***

Balíček jádra LTSP je nutno instalovat vždy jako první. Dalším krokem je provedení inicializace spuštěním:

```
cd /opt/ltsp/templates
./ltsp_initialize            // inicializační skript LTSP
```

Inicializační skript vytváří nebo modifikuje řadu standardních konfiguračních souborů distribuce. Proto je vhodné provést zálohy těchto souborů, pokud v systému existují:

- */etc/X11/xdm/Access* – konfigurační soubor který povolí vzdálené přihlášení
- */etc/X11/xdm/Xsetup_workstation* – nastavuje logo přihlašovací obrazovky
- */etc/dhcpd.conf.example* – příklady konfigurace *dhcp*
- */etc/exports* – konfigurační soubor služby *nfs*
- */etc/X11/gdm/gdm.conf* – konfigurační soubor pro *gdm*
- */etc/X11/gdm/Init/Default* – startovací skript pro *gdm*
- */etc/hosts.allow* – konfigurační soubor pro *tcp wrappers*
- */etc/inittab* – konfigurační soubor pro *init*
- */etc/X11/xdm/ltsp.gif* – logo na pozadí v přihlašovacím dialogu
- */etc/rc.d/rc5.d/S60nfs* – startovací odkaz pro *nfs*
- */etc/rc.d/rc5.d/S13portmap* – startovací odkaz služby *portmapper*
- */etc/sysconfig/syslog* – obsahuje informace pro spuštění *syslogd*
- */etc/xinet.d/tftp* – povoluje FTP démona
- */etc/X11/xdm/xdm-config* – hlavní konfigurační soubor *xdm/kdm*

Balíček s linuxovým jádrem klienta (ltsp_kernel)

- ***ltsp_kernel-3.0.15-i386.tgz***

Minimální instalace LTSP pro textový režim se skládá z **ltsp_core** a **ltsp_kernel**. Druhý balíček ***ltsp_kernel-3.0.15-i386.tgz*** nainstalujeme obdobným způsobem jako předešlý. Balíček v sobě obsahuje obraz inicializačního ramdisku *initrd*. Uvnitř *initrd* jsou obsaženy veškeré moduly ovladačů síťových karet a malý dhcp klient pro pracovní stanice pojmenovaný *dhclient*. Voláním dhcp z prostředí uživatele, namísto z prostředí jádra, umožníme to, že se NFS server může lišit od TFTP serveru. Dále je možné kompilovat ovladače jako moduly a automaticky určit jaký modul nahrát (platí pouze pro karty PCI).

Balíček grafického prostředí X Windows (ltsp_x_core)

Jestliže chceme provozovat na terminálech grafické prostředí postavené na systému X Windows, je nutné stáhnout a nainstalovat tyto balíčky:

- ***ltsp_x_core-3.0.4-i386.tgz*** – základní komponenty X Windows (včetně ovladačů XFree86 4.1.0)
- ***ltsp_x_fonts-3.0.0-i386.tgz*** – základní fonty písem systému X Windows

Některé čipové sady videokaret bohužel nejsou podporovány v XFree86 4.1.0. Pro takovéto případy je nutné stáhnout a instalovat starší verzi XFree86 3.3.6 specifickou vždy pro určité typy čipových sad:

- ***ltsp_x336_3dlabs-3.0.0-i386.tgz*** – XFree86 3.36 X server pro čipové sady 3dlabs
- ***ltsp_x336_8514-3.0.0-i386.tgz*** – XFree86 3.36 X server pro čipové sady 8514
- ***ltsp_x336_agx-3.0.0-i386.tgz*** – XFree86 3.36 X server pro čipové sady agx
- ***ltsp_x336_fbdev-3.0.0-i386.tgz*** – XFree86 3.36 X server pro čipové sady FrameBuffer
- ***ltsp_x336_i128-3.0.0-i386.tgz*** – XFree86 3.36 X server pro čipové sady i128
- ***ltsp_x336_mach32-3.0.0-i386.tgz*** – XFree86 3.36 X server pro čipové sady Mach32
- ***ltsp_x336_mach64-3.0.0-i386.tgz*** – XFree86 3.36 X server pro čipové sady Mach64
- ***ltsp_x336_mach8-3.0.0-i386.tgz*** – XFree86 3.36 X server pro čipové sady Mach8
- ***ltsp_x336_mono-3.0.0-i386.tgz*** – XFree86 3.36 X server pro monochromatické čipové sady
- ***ltsp_x336_p9000-3.0.0-i386.tgz*** – XFree86 3.36 X server pro čipové sady P9000
- ***ltsp_x336_s3-3.0.0-i386.tgz*** – XFree86 3.36 X server pro čipové sady S3
- ***ltsp_x336_s3v-3.0.0-i386.tgz*** – XFree86 3.36 X server pro čipové sady S3Virge
- ***ltsp_x336_svga-3.0.0-i386.tgz*** – XFree86 3.36 X server pro čipové sady kompatibilní s SVGA
- ***ltsp_x336_vga16-3.0.0-i386.tgz*** – XFree86 3.36 X server pro čipové sady kompatibilní s VGA16
- ***ltsp_x336_w32-3.0.0-i386.tgz*** – XFree86 3.36 X server pro čipové sady w32

Balíček pro podporu lokálních aplikací (ltsp_local_apps)

- ***ltsp_local_apps-3.0.0-i386.tgz***

Jestliže máme některé klienty s dostatečným výkonem a množstvím paměti, můžeme běh některých aplikací přesunout přímo na klientskou stanici. Tento balíček připraví prostředí pro běh takových aplikací. Pro lokální aplikace je dále nutné zprovoznit službu NIS a provést další konfigurace, podrobněji rozepsané v kapitolách 4.4 a 5.5.

Balíček pro vytvoření vlastního jádra (ltsp_initrd_kit)

- ***ltsp_initrd_kit-3.0.13-i386.tgz***

Balíček je nutný pouze v případě, že se rozhodneme sestavit vlastní jádro klientské stanice. Obsahuje skripty a základní souborový systém ramdisku, který je stažen spolu s jádrem, když pracovní stanice startuje.

Po instalaci všech potřebných balíčků, se můžeme soustředit na konfiguraci prostředí a zprovoznění všech potřebných serverových služeb.

4.5.2 Instalace a konfigurace openMosix

OpenMosix pracuje na úrovni jádra operačního systému, z čehož plynou výhody, kterými aplikace spouštěná z uživatelského prostoru nedisponuje. Protikladem těmto získaným výhodám je komplikovanost instalace a značně zvýšené nároky na znalosti konfigurace na úrovni jádra systému Linux.

Překlad jádra serveru s podporou openMosix

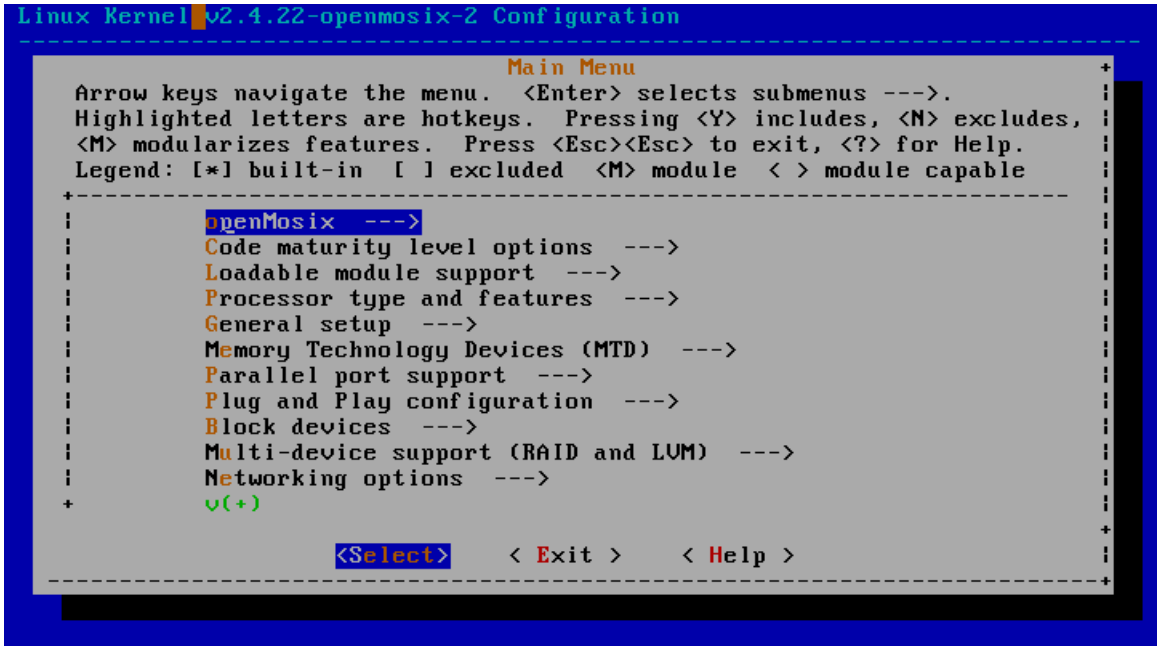
Prvním krokem je stažení komprimovaných archívů zdrojových souborů jádra příslušné verze ***linux-2.4.22.tar.bz2*** z internetového serveru *http://www.kernel.org/* a odpovídajícího archívu záplaty (patch) ***openMosix-2.4.22-2.bz2*** z adresy *http://openmosix.sourceforge.net/*. Archív jádra dekomprimujeme a nakopírujeme do */usr/src*. Dekomprimovaný soubor s patchem openMosix *openMosix-2.4.22-2* nakopírujeme do předchozím kopírováním vzniklého */usr/src/linux-2.4.22/* a do složky se přepneme. V dalším kroku aplikujeme záplatu openMosix do zdrojových souborů jádra příkazem:

```
patch –p1 < openMosix-2.4.22-2
```

Zdrojové soubory jádra jsou připraveny, můžeme tedy pokračovat spuštěním konfigurace příkazem:

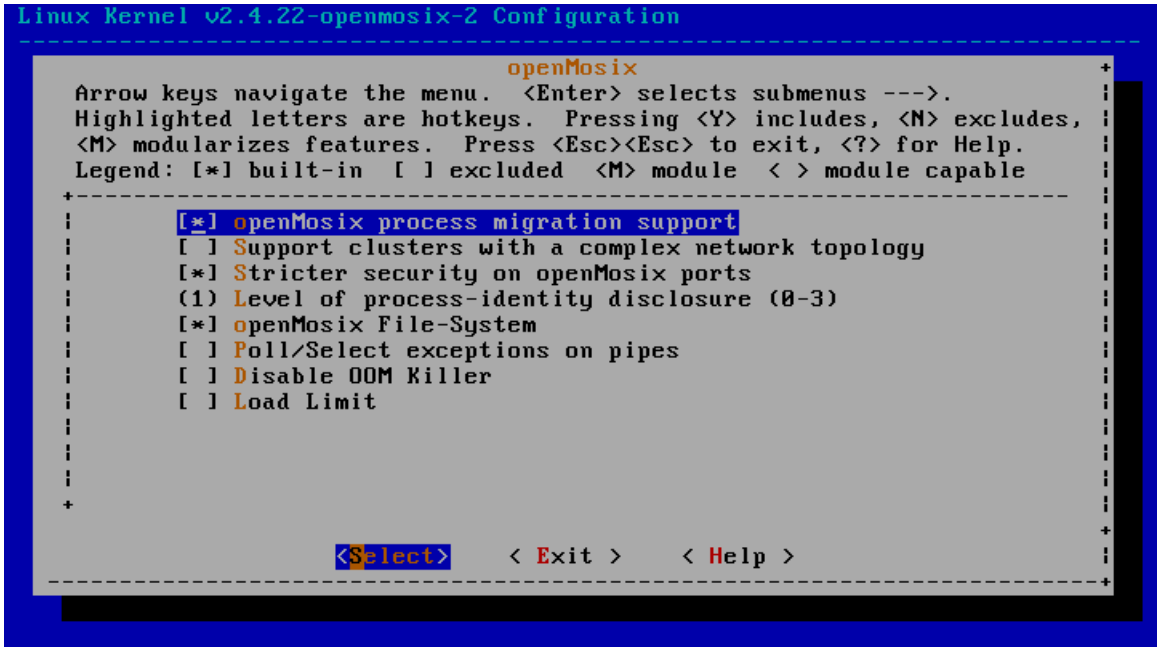
```
make menuconfig.
```

Po spuštění příkazu se objeví konfigurační obrazovka jádra s novou podskupinou konfiguračních voleb pro openMosix (viz. obr. č. 11).



Obr. č. 11 – Konfigurace jádra, základní obrazovka

Pokud se přepneme do skupiny openMosix, zobrazí se řada voleb, které ovlivňují chování klastru. Ve většině případů si vystačíme s výchozím nastavením, které zobrazuje obrázek č. 12.



Obr. č. 12 – Konfigurace jádra, nastavení openMosix

Abychom dobře porozuměli, která volba co ovlivňuje, uvedeme si stručný popis jednotlivých voleb konfigurace openMosix:

- **openMosix process migration support** – volba představuje základní vlastnost openMosix a musí být aktivována. Slouží k aktivaci migrace procesů.

- **Support clusters with a complex network topology** – aktivací volby oznamujeme že, se jedná o komplexní síť spojenou více směrovači (router). Pokud používáme pouze jeden přepínač (switch) nebo rozbočovač (HUB), je vhodné tuto volbu deaktivovat.

- **Maximum network-topology complexity to support (2-10)** – tato možnost je k dispozici pouze pokud je aktivována možnost předešlá. Pokud pracujeme s komplexní sítí, musíme zadat stupeň komplexnosti nejvzdálenějších počítačů, tj. počet směrovačů mezi počítači plus jedna.

- **Stricter security on openMosix ports** – určuje, zda bude openMosix kontrolovat příchozí a odchozí pakety všech uzlů klastru. Pokud chceme zvýšit výkon klastru na úkor bezpečnosti, doporučuje se tuto volbu vypnout.

- **Level of process-identity disclosure (0-3)** – slouží pro získávání informací o procesech spouštěných na vzdálených uzlech. Pro získání „PID“ stačí nastavit hodnotu na „1“.

- **openMosix File-System** – volba zapíná podporu speciálního souborového systému MFS, který slouží jednotlivým členům klastru k výměně informací. Používání MFS zvyšuje výkon a je doporučeno mít volbu povolenu.

- **Poll/Select exceptions on pipes** – podpora pro procesy standardizované v POSIX.

- **Disable OOM Killer** – volba vypíná Out Of Memory Killer

- **Load Limit** – umožňuje snadno nastavovat a kontrolovat limity zátěže pro každý uzel klastru

Poznámka: Popis ostatních skupin a voleb pro konfiguraci jádra daleko převyšuje rámec této práce a nebudu se jim dále věnovat. Kompletní konfigurační soubor je v elektronické podobě obsažen na přiloženém DVD.

Jakmile máme jádro nakonfigurováno v souladu s požadovanými vlastnostmi a hardwarovou konfigurací našeho serveru, můžeme pokračovat v přípravách kompilace jádra. Nejprve je nutné vyřešit závislosti a to příkazem:

```
make dep
```

Dále je vhodné před samotnou kompilací provést odstranění starých nepotřebných souborů, zejména tehdy, pokud již provádíme kompilaci poněkolikáté. To provedeme příkazem:

```
make clean
```

Samotná kompilace jádra do komprimované binární formy se provede příkazem:

```
make bzImage
```

Pokračujeme překompilováním modulů pro jádro:

```
make modules
```

Moduly nainstalujeme:

```
make modules_install
```

A pokud vše předešlé proběhlo bez chyb, nainstalujeme jádro:

```
make install
```

Poslední příkaz přidá odkaz na nové jádro do zavaděče LILO nebo GRUB, aniž bychom museli konfigurační soubor editovat ručně. Vizuální překontrolování konfigurace přesto doporučuji. Je také velmi žádoucí ponechat původní funkční jádro jako druhou volbu zavaděče, pro případy, že by nové jádro nepracovalo správně. Jádro je nainstalované, můžeme tedy provést restart serveru a nové jádro vyzkoušet.

Instalace uživatelských nástrojů openMosix

Pokud chceme získat větší kontrolu a přehled nad klastrem, musíme přistoupit k instalaci uživatelských nástrojů „openMosix Tools“ (***openmosix-tools-0.3.6-2.tar.gz***). Tyto nástroje spravuje jejich autor David Santo Orcero a internetový odkaz na ně je uveden na stránkách projektu openMosix.

Instalace společných balíčků LTSP a openMosix

Existují dva softwarové balíčky, které jsou nezbytné pro vzájemné budoucí propojení tenkých klientů mezi sebou a serverem, a tím k vytvoření klastru. Balíčky obsahují soubory upravených systémových knihoven, prostředky pro vytvoření speciálního souborového systému MFS a spouštěcí skripty, které po nainstalování do struktury souborového systému tenkého klienta, zabezpečí spuštění všech potřebných programů pro zajištění funkcionality openMosix. Z internetové adresy http://www.lpmo.edu/~daniau/ltsp-mosix/ je potřeba stáhnout a nainstalovat níže uvedené balíčky, jejichž autorem je William Daniau [3]:

- ***ltsp_mosix_core-1.0beta4.tar.gz***
- ***ltsp_openmosix_userland_0.3.4.tar.gz***

Instalaci zajistí skript *install.sh* spuštěný přímo z jednotlivých adresářů vzniklých po rozbalení komprimovaných archívů.

Konfigurace openMosix

Uživatelské nástroje openMosix obsahují program (démon *omdiscd*) schopný detekovat do sítě nově připojené uzly klastru. Pokud je takový uzel nalezen, dojde k automatickému začlenění do klastru, bez nutnosti další manuální konfigurace. Démon nachází uplatnění v sítích s malým počtem uzlů v klastru. Pro větší počet uzlů jej není vhodné používat, protože démon nepřetržitě posílá všesměrové zprávy (broadcast messages) do celé sítě a tím snižuje její celkový výkon.

Standardní konfigurační soubor openMosixu obsahující mapu klastru se může jmenovat *openmosix.map* nebo krátce *hpc.map*. Pro zakázání démona jednoduše zeditujeme soubor */etc/hpc.map* dle našich potřeb:

```
# Static openMosix configuration
# ==============================
#
# Each line in this file should contain 3 fields, statically mapping
# IP addresses to openMosix node-numbers:
#
# 1) The first openMosix node-number in this range.
# 2) The IP address of the above node (or node-name from /etc/hosts).
# 3) The number of nodes in this range.
#
# Note: If you don't create a valid IP <-> node-number mapping, the
#     autodiscovery daemon will be started, automatically assigning
#     node-numbers to all visible openMosix machines.
```

```
#
# Example: 10 machines with IP addresses 192.168.1.50 - 192.168.1.59
#        which will have openMosix node-numbers 1-10:
#
# 1      192.168.1.50       10
#
# MOSIX-#  IP  number-of-nodes
# ============================
1 192.168.1.15    1
2 192.168.1.17    1
3 192.168.1.106   1
```

Syntaxe souboru sestává ze tří základních údajů:

- první číslo uzlu z rozsahu
- IP adresa tohoto prvního uzlu z rozsahu
- počet uzlů (IP adres) v rozsahu

Soubor *hpc.map* je kritickým bodem celé konfigurace našeho klastru. Není to kvůli složité syntaxi, ale proto, že je nutné udržovat tento soubor aktualizovaný na všech uzlech klastru. Tato nevýhoda se téměř ztrácí při použití klastrového řešení spolu s tenkými klienty. Soubor jednoduše nakopírujeme do adresáře */etc* souborového systému klientů, jenž exportujeme přes NFS. Udržování konfiguračního souboru se nám tím zúží na dva exempláře. Po dokončení konfigurace můžeme vypsat všechny dostupné uzly příkazem *showmap*. Pokud je vše v pořádku, měli bychom na obrazovce vidět následující výpis:

```
[root@Hive root]# showmap
My Node-Id: 0x0001

Base Node-Id Address          Count
------------ ---------------- -----
0x0001       192.168.1.15     1
0x0002       192.168.1.17     1
0x0003       192.168.1.106    1
[root@Hive root]# _
```

Obr. č. 13 – Zobrazení mapy uzlů klastru

4.5.3 Instalace monitorovacích nástrojů

Mosmon

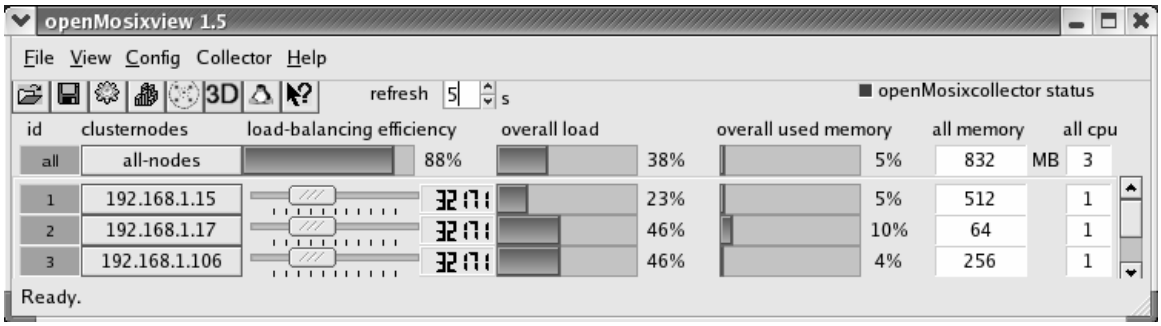
Instalací balíčků ***openmosix-tools-0.3.6-2.tar.gz*** pro server, a ***ltsp_openmosix_userland_0.3.4.tar.gz*** pro klienty, jsme získali základní monitorovací nástroj *mosmon* (viz. obrázek č. 14). Nástroj je schopný přepínat rychlými zkratkami mezi mnoha pohledy, které zobrazují souhrnné informace o všech uzlech klastru, jako je stav vytížení, obsazení paměti, výkon jednotlivých uzlů a další. Jednoduchý souhrn možností získáme vypsáním nápovědy k programu zadáním příkazu *mosmon –help.*

Obr. č. 14 – Monitorovací program mosmon

openMosixview 1.5

Dalším vhodným pomocníkem se může stát program *openMosixview 1.5* [4] (viz. obr. č. 15), který je možné získat stažením stejnojmenného balíčku ***openmosixview1.5.tar.gz*** z internetové adresy *http://www.openmosixview.com/*. Nástroj v grafickém uživatelském prostředí poskytuje mnoho zajímavých informací o stavu klastru, jednotlivých uzlech i migrujících procesech, prostřednictvím několika přepínatelných programových modulů:

- **openMosixview** – základní monitoring a administrace aplikací
- **openMosixprocs** – modul zobrazující procesy s možností jejich administrace
- **openMosixcollector** – démon, který sbírá a ukládá informace o klastru a jednotlivých uzlech
- **openMosixanalyzer** – nástroj pro analýzu informací posbíraných programem openMosixcollector
- **openMosixhistory** – historie procesů klastru
- **openMosixmigmon** – monitoruje migrující procesy
- **openMosixpidlog** – sledování jednotlivých procesů
- **3dmosmon** – monitoruje klastr, a na základě získaných informací vytváří 3d pohled



Obr. č. 15 – Monitorovací program openMosixview 1.5

4.6 Odhadování výkonu

Výpočet, jak výkonný a vybavený server bude vhodný pro naše řešení je velmi subjektivní otázka. Zahrnuje mnoho proměnných a vstupních podmínek, které je nutné při výpočtu zohlednit. Jsou však některá zásadní pravidla a nejlepší zkušenosti, jak se ideálnímu vzorci pro výpočet částečně přiblížit.

4.6.1 Operační paměť RAM

Když odhadujeme jaký vytvořit server, jedním z nejdůležitějších parametrů je bezpochyby množství operační paměti, kterou vložíme do serveru. Standardně všechny aplikace běžící na serveru používají CPU a paměť serveru. Je nutné, aby bylo dostatek paměti pro všechny aplikace, které všichni uživatelé mohou mít spuštěny v ten samý čas. Jestliže server nebude mít dostatek paměti, začíná používat odkládací oddíl nebo soubor. Odkládání je pro výkon celého systému velmi špatná věc a není rozdíl, kolik procesorů je v serveru nebo jak rychlé jsou, protože jakmile jednou začne server data odkládat, ani nejrychlejší CPU současnosti nepomůže. Jestliže není dostatek operační paměti, programy budou zbytečně pomalu nahrávány, i když ostatní systémové prostředky nejsou využity naplno. Linux přednačítá soubory do operační paměti RAM, čímž může být nahrávací čas aplikace významně zkrácen, což bude jistě znát zejména u velkých aplikací jako jsou OpenOffice nebo Mozilla. Provozujeme-li web server (apache, mysql a PHP) nebo jiné služby, budeme potřebovat operační paměti mnohem více.

Poznámka: Některé základní desky mohou být osazeny od dvou do osmi DIMM sockety na CPU, ale nemusí být schopny bežet na maximální rychlosti ve více než čtyřech. Proto je vždy třeba důkladně přečíst kvalitní manuál k základní desce.

Kolik paměti potřebujeme?

Vzorec který používám je následující:

Celkové množství paměti = 256MB + 50MB x počet terminálů

Prvních 256MB paměti je pro server bez terminálů. Potom je třeba plánovat okolo 50MB na každý terminál. Pro extra služby jako web a databáze použijme raději 512MB nebo 1024MB. Pro takovéto služby je další vhodnou možností použít oddělený server.

Takže pokud chceme poskytovat služby 10 LTSP terminálům na našem serveru, můžeme použít tento recept:

Celková paměť = 256 + (50 x 10) = 256 + 500 = 756MB

Toto je rozumný odhad množství paměti, kterou bychom v serveru měli mít. Ale předtím než můžeme definitivně říct že 756MB bude pro naší práci dostačující, musíme zvážit aplikace, které budou uživatelé spouštět. Jestliže všichni budou používat Firefox, OpenOffice, Evolution a plnohodnotné desktopové prostředí jako Gnome nebo KDE a všechny tyto aplikace poběží současně ve stejný čas, tak pravděpodobně 70MB na terminál je přiměřený odhad. Pokud je prostředí přísně kontrolováno a uživateli je pouze povoleno spouštět jen konkrétní aplikace, které nevyžadují tolik RAM a je použit nějaký méně robustní okenní manažer jako ICEWM XFCE nebo WindowMaker, potom možná můžeme vyjít s menším množstvím RAM. 30MB na terminál by mohl být dobrý odhad. Dále musíme zvážit uživatelské prostředí. Jestliže plánujeme systém do školního prostředí, kde 30-ti uživatelům běží stejné aplikace ve stejný čas, budeme potřebovat o trošku více operační paměti. Další možnou variantou je kancelářské nastavení, kde ve stejný čas někteří pracovníci jsou na obchodních schůzkách, jiní telefonují a jenom jedna třetina terminálů je aktuálně používána. Potom nároky na operační paměť nebudou tak velké.

4.6.2 CPU

Jaký typ CPU použít a kolik?

SMP

Více procesorů je velmi dobrý nápad v LTSP prostředí. Se všemi uživateli sdílejícími serverové aplikace jeden uživatel používající intenzivní program, který kriticky přetíží CPU, může mít velmi negativní vliv na instance ostatních uživatelů. Linuxové jádro odvádí dobrou práci při plánování úloh, ale jednotlivý proces stále může zahltit systém. Mít více než jedno CPU může zaručeně pomoci.

Více CPU vs. Dvoujádro vs. Hyper-Threading

Jestliže můžeme jít do více reálných procesorů, je to určitě nejlepší varianta. Dvoujádrové procesory jsou stále novinka a nemám s nimi zatím mnoho zkušeností, ale výkon by měl být hodně blízký jako při použití více oddělených procesorů. Jestliže nemůžeme jít cestou více procesorů nebo dvoujádrových procesorů potom Hyper-Threading je další nejlepší volbou v pořadí. Není to úplně to samé jako více procesorů, ale jsou tu určité části CPU, které jsou duplikované. Uvádí se že výkon se znásobí okolo 1,3x porovnáme-li klasické CPU a CPU s Hyper-Threading na stejné frekvenci.

Jak výkonné CPU je potřeba, můžeme vydedukovat podle zkušeností s typickým systémem s CPU na frekvenci 3GHz. Jeden uživatel využije zhruba od jednoho do tří procent výkonu, takže bez využívání odkládacího prostoru by měl systém zvládnout 30 uživatelů. Jestliže máme uživatelů 60, měli bychom použít 2 CPU nebo dvoujádrové CPU s linuxovým jádrem umožňujícím SMP. Některé základní desky pro Opteron mají možnost osazení čtyřmi procesory s možností připojení druhé vrstvy s dalšími čtyřmi paticemi pro CPU, takže kdokoliv odhadne že horní limit uživatelů by měl být 480. Po zvážení potřeby redundance a cen komponent může být levnější a tedy lepší použít více serverů.

32-bitové vs. 64-bitové CPU

Je dostupných několik nových 64-bitových CPU a ty mohou značně zvýšit počet terminálů, které můžete přiřadit serveru. Je důležité se ujistit, že provozujete 64-bitovou verzi Linuxu na takovýchto serverech, abyste získali všechny výhody těchto CPU. Dalším faktem ke zvážení je, že ještě mnoho aplikací není připraveno pro 64-bity a to může způsobit problémy s výkonem. Nakonec, každý software, který přichází v úvahu, bude přepracován pro 64-bity, takže by to nemusel být zas tak velký problém.

Intel vs. AMD

Všechno co tady mohu doporučit je, aby si každý vybral dodavatele, se kterým je nejvíce spokojen. Válka mezi těmito prodejci se přenesla do soudních síní, kde AMD obvinilo Intel z nefér praktik. AMD v poslední době převzalo vedení s Opterony a AMD64, ale Intel se nesklonil. Opteron má na čipu integrovaný řadič paměti, takže je zde teoretický důvod proč je pro víceprocesorové prostředí nejlepší. Ve 32-bitech AMD mělo významnou výhodu v ceně, ale ta zmizela když vyšlo AMD64. Ceny klesají.

4.6.3 Pevný disk

Je vhodné použít nejrychlejší pevný disk, který si můžeme dovolit a určitě je vhodné zvážit použití RAID, abychom zredukovali šance že poškození disku způsobí zhroucení celého systému.

SCSI

SCSI disky a řadiče jsou osvědčeným standardem pro servery, a jestliže plánujeme víc než 20 terminálů, potom bychom měli určitě zvážit rychlé disky s 15-ti tisíci otáčkami za minutu připojené do SCSI řadiče.

SATA

SATA se ukazují jako velmi pěkná technologie pro disky nižší cenové kategorie. Výkon je velmi dobrý. Mé osobní nejlepší nastavení je pár SATA disků spojených do 3Ware RAID řadiče spolu zrcadlených (RAID-1). Tato konfigurace by měla zaručit bezproblémové soužití 10-20 šťastných uživatelů. Pro 20 uživatelů a více bychom pouze přidali víc disků v konfiguraci stripe/mirror (RAID-10).

ATA/IDE

Staré ATA disky jsou vhodné pro malé LTSP servery s několika uživateli. Použijme jich pár spolu s ATA RAID řadičem nebo použijme softwarový RAID pro získání lepšího výkonu.

4.6.4 Síť

Jak rychlá by měla být?

10Mb/100Mb/1Gb za sekundu.

V prostředí LTSP je každý terminál připojen do rozbočovač (HUB) nebo přepínače a ten je pak připojen k serveru. To znamená, že veškerý síťový provoz běží mezi přepínačem a serverem. Důraz bychom měli dát na co možná nejrychlejší linku, kterou můžeme mezi přepínačem a serverem zajistit. Dále bychom měli zvážit použití gigabitového spojení mezi serverem a přepínačem a potom 100Mb mezi přepínačem a klienty. A určitě by bylo vhodné použít přepínač namísto rozbočovače. Výkon přepínače je výrazně lepší než u rozbočovačů.

4.6.5 Zdroje el. energie

Přestože volba zdroje nebude mít vliv na rychlost serveru, určitě by se mělo zvážit výhod redundance zdrojů. Náhlý výpadek sítě s několika desítkami uživatelů může vést k nezvratné ztrátě velkého objemu dat. Při výběru zdrojů bychom měli dát přednost kvalitním ověřeným výrobkům s dlouhou životností.

4.6.6 Shrnutí

Informace výše jsou tu proto, aby nám dali zevrubný nápad, co potřebujeme pro vybudování LTSP prostředí. Cíle se mohou lišit. Také budeme muset zvážit, co termín „přijatelný výkon“ znamená pro naše uživatele. Ve většině případů, kdy si nemůžeme dovolit nejlepší hardware, je stále lepší něco vydat pro uživatele. I pomalý systém je lepší než žádný systém.

5 Klient

5.1 Metody startu klienta po síti

5.1.1 PXE

Preboot eXecution Environment je prostředí umožňující start počítače za použití síťové karty nezávisle na dostupných zařízeních pro ukládání dat (pevné disky, disky typu flash, diskety) a instalovaných operačních systémech. Jestliže síťová karta má integrované PXE, potom jej můžeme použít pro natažení linuxového jádra. PXE je technologie bootrom obdobná Etherboot nebo Netboot.

Jedinou podmínkou aby PXE boot pracoval je mít povolenou PXE bootrom na naší síťové kartě a nastavenou prioritu bootování na bootování ze sítě. PXE má omezení, které působí to, že je schopné nahrát pouze soubory menší než 32KB. Linuxové jádro je zpravidla o dost větší, takže jej nemůžeme přes PXE nahrávat přímo. Je potřeba nejdříve nahrát kód tzv. „Network Bootstrap program“ neboli NBP, který po svém natažení umožní stažení linuxového jádra bez velikostního omezení. Vhodný NBP pro nahrání linuxových jader nazvaný *pxelinux.0* je součástí balíčku *syslinux* od jádrového vývojáře jménem H. Peter Anvin.

Start přes PXE sestává z těchto kroků:

- PXE bootrom inicializuje síťovou kartu s vysílá DHCP požadavek
- DHCP server odpovídá IP adresou a jménem NBP pro nahrání
- PXE bootrom stáhne NBP umístí jej do paměti a začne spouštět
- NBP využívá TFTP pro stažení konfiguračního souboru ze serveru
- Konfigurační soubor obsahuje jméno jádra, jméno obrazu inicializačního ramdisku a volby pro jádro jakmile je nahráno

  Zde je příklad konfiguračního souboru *pxelinux*:

```
prompt=0
label linux
kernel bzImage-2.4.24-ltsp-4
append init=/linuxrc rw root=/dev/ram0 initrd=initrd-2.4.24-
ltsp-4.gz
```

- Potom NBP používá TFTP pro natažení linuxového jádra a inicializaci ramdisku (*initrd*)
- Řízení je potom předáno linuxovému jádru, to nastartuje připojí *initrd* a pokračuje se spouštěním tenkého klienta

5.1.2 Etherboot

Etherboot je balíček Open Source software pro vytvoření ROM obrazů schopný stáhnout kód přes síť typu ethernet, který může být spuštěn na počítačích x86. Mnoho síťových adaptérů má patici, kam může být instalován čip ROM. Etherboot je kód, který může být umístěn v takovéto ROM. Pro použití Etherboot, jestliže již máme

připravenou síťovou kartu s Etherboot bootrom, budeme možná muset změnit konfiguraci BIOS tak, aby boot nejprve probíhal ze sítě před disketovou mechanikou nebo harddiskem.

Etherboot podporuje velmi rozsáhlý počet síťových karet. Přes 200 modelů a další jsou neustále přidávány. Etherboot dále umožňuje vytvoření několika typů obrazů lišících se účelem použití. Významnou pomoc s vytvářením ROM obrazů představují internetové stránky www.rom-o-matic.net, kde lze snadno vygenerovat potřebný soubor v těchto možných formátech:

- *.zdsk* – obraz pro disketu
- *.zrom* – obraz pro nahrání přímo do ROM síťové karty
- *.iso* – spustitelný obraz (bez funkce „legacy floppy emulation“)
- *.iso* – spustitelný obraz (s funkcí „legacy floppy emulation“)
- *.zlilo* – formát jádra s možností zavést z LILO, GRUB, SYSLINUX
- *.zpxe* – NBP pro nahrání přes PXE síťové karty (použito v této práci)
- *.com* – obraz spustitelný z prostředí DOS
- *.zhd* – obraz diskové partition
- *.elf* – (Generic) ROM obraz
- *.elf* – (LinuxBIOS) ROM obraz

Ať už jsme vybrali vytvoření diskety nebo vypálení kódu do EPROM, musíme v první řadě určit, který model síťové karty máme.

Výběr Etherboot ovladače pro ISA síťové karty

Pro starší na ISA založené síťové karty není zas až tak důležité určit přesný typ. Většina z nich je buď karta typu ne2000 nebo 3Com 3c509. Potřebujeme tedy použít takový ovladač, který je schopen vybrat správný typ média na kartě, oba 10-base2 (Coax) a 10 base-T (Twisted Pair).

Výběr Etherboot ovladače pro PCI síťové karty

Pro PCI karty je důležité abychom vybrali ovladač, který souhlasí s údajem PCI Vendor (výrobcem) a Device ID (identifikačním číslem) síťové karty. Někdy máme štěstí a přesně víme, který model karty máme, protože číslo modelu je vytištěné na kartě a odpovídá popisu jednoho Etherboot modelu. V mnoha případech však bude situace složitější a budeme muset nalézt kartu pomocí PCI ID. Nejjednodušším způsobem zjištění potřebných hodnot, je vložit síťovou kartu do stroje, kde je nainstalován Linux a použít níže uvedené příkazy. Jakmile máme spuštěný Linux můžeme použít příkaz *lspci –n* s výstupem viz. obrázek č.16.

```
[root@Hive root]# lspci -n
00:00.0 Class 0600: 8086:7190 (rev 01)
00:01.0 Class 0604: 8086:7191 (rev 01)
00:07.0 Class 0601: 8086:7110 (rev 08)
00:07.1 Class 0101: 8086:7111 (rev 01)
00:07.2 Class 0c03: 8086:7112
00:07.3 Class 0680: 8086:7113 (rev 08)
00:0f.0 Class 0300: 15ad:0405
00:10.0 Class 0100: 104b:1040 (rev 01)
00:11.0 Class 0604: 15ad:0790 (rev 02)
02:00.0 Class 0200: 1022:2000 (rev 10)
02:01.0 Class 0401: 1274:1371 (rev 02)
02:02.0 Class 0c03: 15ad:0770
[root@Hive root]# _
```

Obr. č. 16 – Výstup příkazu lspci –n

V příkladu výše můžeme vidět položky pro každou PCI kartu v sytému. Potřebujeme pouze pohlédnout na zařízení *Class 0200***.** Takže spusťme příkaz znovu a podívejme se pouze po rozhraních *ethernet* a to příkazem s mnohem přehlednějším výstupem (viz. následující obrázek).

```
[root@Hive root]# lspci -n | grep "Class 0200"
02:00.0 Class 0200: 1022:2000 (rev 10)
[root@Hive root]# _
```

Obr. č. 17 – Výstup příkazu lspci –n | grep "Class 0200"

Čísla PCI ID jsou 1022:2000. První pole, 1022 je PCI Vendor ID. V tomto příkladu je výrobce společnost „Advanced Micro Devices". Druhé pole 2000 je PCI Device ID. To nám říká o který model síťové karty se jedná. V tomto případě to je karta „PCnet32".

Pokud naše stanice má disketovou nebo CD-ROM mechaniku, je další snadnou cestou použít některou z dnešních minimalistických distribucí nebo „Live distribucí", nastartovat z ní systém a informace získat zadáním stejných příkazů jako v případě předchozím. Další jednoduchou možnost jak získat informace o PCI kartách typu ethernet, zobrazuje obrázek číslo 18.

```
[root@Hive net]# cat /proc/pci  | grep "Ethernet"
    Ethernet controller: Advanced Micro Devices [AMD] 79c970 [PCnet32 LANCE] (re
v 16).
[root@Hive net]# _
```

Obr. č. 18 – Výpis souboru /proc/pci

5.2 Jádro klienta

Na internetových stránkách projektu LTSP je k dispozici ke stažení balíček **ltsp_kernel-3.0.15** obsahující jádro verze 2.4.24 upravené pro síťový start pomocí PXE nebo Etherboot. V balíčku jsou dále obsaženy všechny pro jádro potřebné moduly, zejména moduly všech možných, linuxem podporovaných, síťových karet, kterými klient může být osazen. Originální LTSP jádro s ovladači zaváděnými pomocí modulů nemá aplikovaný openMosix patch, takže pokud chceme klienta využít i jako plnohodnotného člena klastru, jádro použít nemůžeme. Přesto doporučuji tento balíček nainstalovat. Oceníme jej zejména při testování, zda je pro všechny požadované služby správně konfigurován server i při odstraňování síťových problémů. Bylo řečeno, že

pokud chceme zapojit klienta jako další nod výpočetního klastru, musíme provést několik vlastních konfigurací a úprav, které si popíšeme v následujících kapitolách.

5.2.1 Konfigurace a překlad jádra klienta

OpenMosix vyžaduje, aby jádra jednotlivých členů klastru bylo stejné verze a stejné konfigurace voleb souvisejících s openMosix. Je tím zajištěna kompatibilita a stejné chování nodů. Kompletní sestavení jádra provádíme na serveru, a je tedy vhodné vyjít ze zdrojových kódů, které jsme použili pro server, a kde je již aplikován openMosix patch. Při konfiguraci a překladu procházíme stejnými kroky jako při sestavování jádra serveru, ale neprovedeme závěrečnou instalaci jádra abychom si nepřepsali jádro serveru. Pro krátké shrnutí si vypíšeme seznam použitých příkazů, tak jak půjdou za sebou:

```
1. make menuconfig
2. make dep
3. make clean
4. make bzImage
5. make modules
6. make modules_install
```

Samotná konfigurace klientského jádra zahrnuje několik typických nastavení. Jádro pro terminálové stanice lze konfigurovat dvojím způsobem a to pro použití s inicializačním ramdiskem (s podporou initrd) a pro použití bez ramdisku (bez podpory initrd). Obvyklejší a podle mého názoru vhodnější způsob je použití s podporou initrd. Následující volby je nutné povolit při konfiguraci:

- **File systems –> /dev filesystem support** – souborový systém */dev* musí být povolen. Není nutné aktivovat volbu pro automatické připojení při startu (Automatically mount at boot). Připojení místo jádra vykoná skript */linuxrc*.

- **Block devices –> RAM disk support** – podpora pro souborový systém vytvářený v RAM

- **Block devices –> Initial RAM disk (initrd) support**

- **Processor type and features –> Processor family** – nesmíme zapomenout na výběr správného modelu procesoru. Vhodné je vypnout i podporu SMP, pokud není třeba.

- **Network device support – Ethernet (10 or 100Mbit) – EISA,VLB, PCI and on board controllers** - výběr konkrétní síťové karty a nastavení pro zavádění jejího ovladače jako modul <M>

- **openMosix** – nastavení všech voleb v této skupině se musí shodovat s konfigurací jádra serveru a s jádry ostatních členů klastru

Pokud se nepoužije příkaz make install zůstane po konfiguraci a sestavení jádro ponecháno ve složce */usr/src/linux-2.4.22/arch/i386/boot/bzImage*, odkud si jej později načtou skripty, které zajistí úpravy pro síťový start.

5.2.2 Úprava jádra pro síťový start

Vytvoření obrazu jádra spustitelného přes síť vyžaduje instalaci dalších nástrojů obsažených v instalačních balíčcích **ltsp_initrd_kit** a **mknbi** [5].

mknbi (make network boot image)

Aby Etherboot mohl pracovat s linuxovým jádrem je nutné jej tzv. označit. Nástroj pro označení (tagging) jádra se nazývá *mknbi-linux*. Nástroj je volán skriptem *buildk*, který instaluje výše zmíněný softwarový balíček. Součástí balíčku je také program *mkelf-linux* používaný pro spojení obrazu ramdisku s obrazem jádra v jeden.

ltsp initrd kit

Je softwarový balíček obsahující skript *buildk* a všechny potřebné příkazy pro přípravu obrazu jádra (*bzImage-2.4.22-openmosix-2*). Dále obsahuje šablonu kompletní struktury souborového systému ramdisku, včetně knihoven a aplikací. Šablona bude použita pro vytvoření obrazu ramdisku (*initrd-2.4.22-openmosix-2.gz*).

Cestu k výslednému obrazu jádra po všech úpravách je nakonec nutno zadat pomocí parametru *filename* příslušnému terminálu v *dhcpd.conf*. Vzhledem k tomu, že budeme pravděpodobně vytvářet více jader pro různé typy terminálů, je vhodné si spouštění nástrojů zjednodušit skriptem:

```
#!/bin/bash
cd /jadro/ltsp_initrd_kit/
./buildk
mkelf-linux -ip=dhcp -rootdir=192.168.1.15:/opt/ltsp/i386/
/jadro/ltsp_initrd_kit/bzImage-2.4.22-openmosix-2
/jadro/ltsp_initrd_kit/initrd-2.4.22-openmosix-2.gz >
/tftpboot/lts/omosix1
```

5.3 Konfigurace klienta, lts.conf

Návrh LTSP předem počítá s možnou odlišností terminálových stanic, ať už po stránce hardwarové, tak po stránce možného využití aplikací. Liší-li se stanice, je logické že se bude muset upravit i konfigurace. Tato, pro koncové stanice specifická, nastavení se čtou z konfiguračního souboru */opt/ltsp/i386/etc/lts.conf*. Soubor je rozčleněn na dvě základní sekce. První sekce obsahuje standardní (default) nastavení, zatímco druhá se týká konkrétního nastavení jednotlivých stanic.

```
# příklad souboru /etc/dhcpd.conf
# dhcpd.conf
# Config file for the Linux Terminal Server Project (www.ltsp.org)
[Default]
SERVER             = 192.168.1.15
XSERVER            = auto
X_MOUSE_PROTOCOL   = "PS/2"
X_MOUSE_DEVICE     = "/dev/psaux"
X_MOUSE_RESOLUTION = 400
X_MOUSE_BUTTONS    = 3
USE_XFS            = N
LOCAL_APPS         = N
RUNLEVEL           = 5
#-----------------------------------------------------------------------
# specifikace nastavení terminálu
#-----------------------------------------------------------------------
[Term1] # stanice v textovém prostředí
MOSIX             = Y
LOCAL_APPS        = N
USE_NFS_SWAP      = N
SWAPFILE_SIZE     = 48m
RUNLEVEL          = 3
#-----------------------------------------------------------------------
[Term2] # stanice s grafickým prostředím
XSERVER           = XF86_SVGA
MOSIX             = Y
X_COLOR_DEPTH     = 8
X_MODE_0          = 640x480
LOCAL_APPS        = N
USE_NFS_SWAP      = N
SWAPFILE_SIZE     = 48m
RUNLEVEL          = 5
#-----------------------------------------------------------------------
...
```

5.3.1 Všeobecné parametry

Poznámka: řádky začínající znakem „#” jsou ignorovány a lze je použít pro komentáře.

LTSP_BASEDIR
Určuje, kde je umístěn kořenový adresář souborového systému pro LTSP. Pokud není nastaveno, použije se implicitní hodnota */opt/ltsp*

SERVER
Pokud každý ze serverů není specifikován vlastním parametrem *XDM_SERVER*, *TELNET_HOST*, *XFS_SERVER*, *SYSLOG_HOST* je IP adresa uvedená touto volbou použita pro všechny jmenované.

SYSLOG_HOST
Pokud chceme zaznamenávat systémové informace jinde, musíme v této volbě uvést adresu tohoto stroje. Nenastavení této volby znamená použití implicitní hodnoty, kterou nastaví podle parametru *SERVER*.

NFS_SERVER
Nastavení IP adresy NFS serveru odkud se připojuje souborový systém */home*. Implicitní volba podle hodnoty SERVER.

USE_NFS_SWAP
Nastavením parametru na hodnotu *Y* povolujeme používání odkládacích souborů přes NFS. Implicitní hodnota je *N*.

SWAPFILE_SIZE
Parametr kterým určujeme velikost odkládacího souboru. Standardní velikost 64MB.

SWAP_SERVER
Odkládací soubor může existovat na jiném serveru v síti. Standardní hodnota je nastavena podle parametru *NFS_SERVER*.

NFS_SWAPDIR
Určuje, který adresář serveru obsahující odkládací soubory bude exportován přes NFS. Standardní hodnota je */var/opt/ltsp/swapfiles*. Tuto implicitní cestu musí také obsahovat soubor */etc/exports*.

TELNET_HOST
Určuje server pro připojení přes znakový terminál *telnet*. Implicitní hodnotu bere z parametru *SERVER*.

DNS_SERVER
Parametr se použije při vytváření *resolv.conf* souboru.

SEARCH_DOMAIN
Parametr se použije při vytváření *resolv.conf* souboru.

MODULE_01 až MODULE_10

Pomocí těchto parametrů můžeme zavést až 10 jaderných modulů.

Příklad:
MODULE_01 = uart401.o
MODULE_02 = sb.o io=0x220 irq=5 dma=1
MODULE_03 = opl3.o

Jestliže je jako hodnota parametru absolutní cesta, potom se pro zavedení použije příkaz *insmod*. Jinak se použije příkaz *modprobe*.

RAMDISK_SIZE
Při startu stanice se vytvořený ramdisk připojí do adresáře */tmp*. Tímto parametrem můžeme specifikovat velikost souborového systému. Údaj se uvádí v jednotkách kilobytů (1024 bajtů). Hodnota 1024 tedy způsobí vytvoření ramdisku o velikosti 1 megabyte. Pokud změníme hodnotu tohoto parametru je také nutné změnit velikost ramdisku uvnitř jádra. Jádro je pak nutné znovu zkompilovat. Pokud používáme Etherboot nebo Netboot, je nutné jádro po zkompilování označit (tagging) pomocí programu *mknbi-linux* s příslušnou informací o velikosti ramdisku. Standardní velikost ramdisku je 1024 (1MB)

RCFILE_01 až RCFILE_10
Pomocí těchto parametrů můžeme ukázat skriptu *rc.local* na dalších 1 až 10 skriptů pro spuštění. Skripty musí být uloženy v */etc/rc.d* adresáři adresářové sktruktury terminálu, odkud je *rc.local* zavolá.

SOUND
Pokud je nainstalovaný LTSP balíček pro zvuk, nastavením tohoto parametru na hodnotu *Y* způsobíme vykonání skriptu *rc.sound*, který nastaví zvukovou kartu a spustí příslušného démona. Standardní hodnota parametru je *N*.

5.3.2 Parametry X-Windows

XDM_SERVER
Tímto parametrem můžeme nasměrovat XDM k jinému stroji než k standardnímu serveru. Pokud není tento parametr specifikován, použije se implicitní hodnota z parametru *SERVER*.

XSERVER
Parametr definuje jaký X server bude na pracovní stanici běžet. Pro PCI a AGP grafické adaptéry často není třeba. Skript *rc.local* je schopen většinu grafických adaptéru detekovat automaticky. Pro grafické karty, které skript *rc.local* neúspěšně detekuje, a pro všechny karty typu ISA, můžeme v tomto vstupu specifikovat jméno konkrétního ovladače nebo X serveru. Pokud hodnota začíná „*XF86_*“ pak se použije Xfree86 verze 3.3.6, jinak XFRee86 4.1.x. Standardní hodnota je *auto*.

X_MODE_0 až X_MODE_2
Terminálu můžeme nadefinovat až tři různá rozlišení. Parametr může obsahovat přímo hodnotu rozlišení nebo kompletní „modeline“.

Například:
```
X_MODE_0 = 800x600
```
nebo
```
X_MODE_0 = 800x600 60.75 800 864 928 1088 600 616 621 657 –Hsync Vsync
```

Pokud nespecifikujeme žádný parametr *X_MODE_x* použije se implicitní modelování a tři nejběžnější rozlišení 1024x768, 800x600, 640x480.

X_MOUSE_PROTOCOL
Může být použita jakákoliv hodnota, která koresponduje s klíčovými slovy „XFRee86 Pointer Control“. Typickými hodnotami jsou *Microsoft* a *PS/2*. Implicitní je *PS/2*.

X_MOUSE_DEVICE
Představuje zařízení, ke kterému je myš připojena. Jedná-li se o sériovou myš, pak to bude sériové rozhraní řadiče, typicky */dev/ttyS0* nebo */dev/ttyS1*. Pokud se jedná o myš PS/2 pak to bude hodnota */dev/psaux*, která je přednastavena implicitně a není ji nutné uvádět.

X_MOUSE_RESOLUTION
Toto ovlivňuje hodnotu rozlišení v souboru *XF86Config*. Typická hodnota pro sériovou myš je *50*, pro PS/2 pak může být *400*. Standardní přednastavená hodnota je 400.

X_BUTTONS
Parametrem sdělíme systému kolik má myš tlačítek. Obvykle se nastavuje *2* nebo *3*. Standardně je systém nastaven pro tří-tlačítkovou myš.

X_MOUSE_EMULATE3BTN
Hodnotou *Y* parametru můžeme zapnout emulování tří tlačítek na dvou-tlačítkové myši. Funkci třetího tlačítka pak simuluje současný stisk obou tlačítek. Implicitní hodnota je *N*.

X_MOUSE_BAUD
Parametr definuje přenosovou rychlost pro sériovou myš. Standardní hodnota je *1200*.

X_COLOR_DEPTH
Nastaví kolik bitů má být použito pro barevnou hloubku. Možné hodnoty jsou *8, 15, 16, 24, 32*. 8 bitů definuje 256 barev, 16 bitů 65536 barev, 24 bitů 16 miliónů barev a 32 bitů dá 4.2 miliard barev. Ne všechny X servery však podporují všechny hodnoty a je třeba si zjistit informace, abychom předešli chybě. Implicitní hodnota je 16.

USE_XFS
Volba povoluje nebo zakazuje čtení fontů z XFS (X Font Serveru). Font server poskytuje jednoduchý způsob distribuce fontů z jednoho místa. Hodnoty volby mohou být *Y* a *N*. Implicitní hodnota je *N*.

XFS_SERVER
Chceme-li specifikovat jiný X Font Server než který je uveden volbou SERVER, můžeme zde zadat jeho IP adresu.

X_HORZSYNC
Volbou se konfiguruje parametr *HorizSync* z XFree86. Implicitní hodnota je *31-62*.

X_VERTREFRESH
Nastavuje vertikální obnovovací frekvenci, parametr *VertRefresh* XFree386. Implicitně *55-90*.

XF86CONFIG_FILE
Volbou definujeme název našeho vlastního souboru XF86Config, který musíme uložit na server do adresáře */opt/ltsp/i386/etc* .
Například:
```
XF86CONFIG_FILE = XF86Config.Terminal_1
```

5.3.3 Parametry dotykové obrazovky

USE_TOUCH
Pokud používáme dotykovou obrazovku, je třeba nastavit tento parametr na hodnotu *Y*. Nastavením parametru se zpřístupní další konfigurační volby nezbytné pro tento typ obrazovek.

X_TOUCH_DEVICE
Dotyková obrazovka pracuje podobně jako myš přes sériové rozhraní počítače. Volbou specifikujeme, které sériové rozhraní je použito. Například je možné nastavit */dev/ttyS0*. tato volba nemá žádnou implicitní hodnotu.

X_TOUCH_MINX
Kalibrační parametr *EloTouch* dotykové obrazovky. Standardní hodnota je *433*.

X_TOUCH_MAXX
Kalibrační parametr *EloTouch* dotykové obrazovky. Standardní hodnota je *3588*.

X_TOUCH_MINY
Kalibrační parametr *EloTouch* dotykové obrazovky. Standardní hodnota je *569*.

X_TOUCH_MAXY
Kalibrační parametr *EloTouch* dotykové obrazovky. Standardní hodnota je *3526*.

X_TOUCH_UNDELAY
Kalibrační parametr *EloTouch* dotykové obrazovky. Standardní hodnota je *10*.

X_TOUCH_RPTDELAY
Kalibrační parametr *EloTouch* dotykové obrazovky. Standardní hodnota je *10*.

5.3.4 Parametry lokálních aplikací

LOCAL_APPS
Nastavením hodnoty na *Y* povolíme používání aplikací běžících lokálně na stanici. Aby aplikace mohly lokálně běžet je třeba na serveru provést několik dalších nastavení. Více v kapitole týkající se lokálních aplikací. Implicitní hodnota je *N*.

NIS_DOMAIN
Aby bylo možné používat lokální aplikace, je potřeba mít nakonfigurovaný NIS server. Tento parametr specifikuje NIS doménové jméno, které musí být shodné s doménovým jménem definované NIS serverem. To není stejné jako internetová doména. Implicitní hodnota je *ltsp*.

NIS_SERVER
Nastavuje IP adresu našeho NIS serveru, pokud nechceme aby byl NIS server vyhledáván pomocí všesměrového vysílání.

5.3.5 Parametry klávesnice

Všechny podpůrné soubory klávesnice jsou nyní umístěny v */opt/ltsp/i386*, takže konfigurace mezinárodní klávesnice je nyní záležitostí konfigurace XFree86. Pro tuto konfiguraci máme k dispozici několik parametrů. Hodnoty těchto parametrů můžeme vyčíst z dokumentace k XFree86. Vše co je platné v XFree86 je platné pro tyto parametry.

XkbTypes
Standardní hodnota je *default*.

XkbCompat
Standardní hodnota je *default*.

XkbSymbols
Standardní hodnota je *us(pc101)*.

XkbModel
Standardní hodnota je *pc101*.

XkbLayout
Standardní hodnota je *us*.

5.3.6 Parametry pro tiskárnu

K pracovní stanici mohou být připojeny až tři tiskárny. Kombinace sériové a paralelní tiskárny mohou být konfigurovány pomocí následujících voleb v *lts.conf*.

PRINTER_0_DEVICE
Název zařízení první tiskárny. Jsou povoleny názvy jako */dev/lp0*, */dev/ttyS0* nebo */dev/ttyS1*.

PRINTER_0_TYPE
Parametr typu tiskárny. Platná volba pro paralelní je *P*, pro sériovou *S*.

PRINTER_0_PORT
Specifikuje číslo použitého TCP/IP portu. Standardně se použije *9100*.

PRINTER_0_SPEED
Nastavuje přenosovou rychlost. Standardně se použije *9600*.

PRINTER_0_FLOWCTRL
Pro sériové tiskárny může být definována regulace toku dat. Volba *S* pro softwarovou (XON/XOFF) regulaci toku, *H* pro hardwarovou (CTS/RTS) regulaci toku dat. Pokud není specifikováno použije se *S*.

PRINTER_0_PARITY
Specifikuje paritu. Volby jsou *E, O, N*. standardní hodnota je *N*.

PRINTER_0_DATABITS
Definuje počet datových bitů. Volby jsou *5, 6, 7, 8*. Jestliže není specifikováno, použije se *8*.

5.4 Start klienta

Start bezdiskové stanice prochází řadou kroků. Porozumění co se během tohoto procesu děje, zjednoduší řešení problémů, které by mohly nastat.

Pro start stanice jsou důležité čtyři základní služby:

- **DHCP**
- **TFTP**
- **NFS**
- **XDMCP**

LTSP je velmi přizpůsobitelný. Všechny z výše uvedených služeb můžou být obstarány jedním osamoceným serverem nebo několika různými servery. V našem případě vycházíme z předpokladu, že všechny služby zajišťuje server jediný.

Kroky kterými stanice prochází

1. Po zapnutí stanice a provedení všech základních testů, BIOS hledá aktivní přídavné paměti ROM. Pokud je v BIOS nastavena priorita startu systému na LAN a nalezne ROM v síťové kartě, provede její inicializaci a spuštění v ní obsaženého kódu.

2. Pokud karta obsahuje PXE, vyšle do lokální sítě žádost obsahující svou MAC adresu o přidělení IP adresy.

3. Program *dhcpd* běžící na serveru zachytí tuto žádost, přečte MAC adresu a porovná ji se svým konfiguračním souborem.

4. Pokud existuje záznam odpovídající MAC adrese vytvoří podle něj paket s informacemi, který jako odpověď stanici odešle. Odpověď obsahuje tyto informace:

    - IP adresu přidělenou stanici
    - síťovou masku lokální sítě
    - novou Etherboot ROM síťové karty

5. Klient přijme odpověď, načte novou Etherboot ROM, provede reinicializaci síťové karty, nastaví IP adresu a masku. Poté odešle další požadavek na DHCP server.

6. Server z nového dotazu pozná, že karta obsahuje novou Etherboot ROM, která je již schopná načítat větší soubory než s PXE a pošle odpověď s těmito doplňujícími informacemi:
    - cestu kde je uložené linuxové jádro
    - cestu ke kořenovému adresáři souborového systému pro připojení
    - volitelné parametry pro linuxové jádro

7. Etherboot přijme odpověď s rozšířenými parametry a přes TFTP začne stahovat komprimované linuxové jádro.

8. Po úspěšném stažení je jádro dekomprimováno, umístěno v paměti klienta a začne jeho spouštění.

9. Jádro zinicializuje celý systém včetně všech periferních zařízení.

10. Za obrazem jádra je umístěn obraz ramdisku, který je dočasně připojen jako kořenový souborový systém.

11. Když jádro dokončí inicializaci ramdisku, obvykle spouští program *init*. V tomto případě ale má jádro informaci, že má spouštět skript *linuxrc*. Toho docílíme volitelným parametrem *init=/linuxrc*.

12. *Linuxrc* skript začíná skenovat PCI sběrnice a vyhledává síťovou kartu. Pro každé PCI zařízení které najde nahlédne do */etc/niclist* kde hledá shodný záznam.

13. Pokud je nalezena shoda, je vráceno jméno modulu NIC adaptéru a dojde k jeho zavedení. Pro karty ISA musí být modul ovladače specifikován v příkazové řádce

jádra s příslušným IRQ, adresou a dalšími parametry, které by mohly být vyžadovány.

14. Pak je spuštěn malý DHCP klient (*dhclient*) a je vznesen další dotaz na DHCP server. Toto je třeba udělat odděleně, abychom získali více informací, než které jsme obdrželi předchozími dotazy.

15. Až *dhclient* obdrží odpověď ze serveru, spustí se */etc/dhclient-script*, který vezme informace které obdržel a nastaví rozhraní *eth0*.

16. Do této doby byl kořenovým souborovým systémem *ramdisk*. Nyní skript */linuxrc* připojí nový kořenový souborový systém přes NFS. Exportovaný adresář ze serveru je typicky */opt/ltsp/i386*. Nejprve nemůže dojít k připojení nového souborového systému jako */ (root)*, ale připojí jej jako */mnt*. Pak je skriptem *pivot_root* přemístěn aktuální kořen souborového systému do standardního umístění. Po dokončení je NFS souborový systém připojen jako */ (root)* a starý kořenový souborový systém ve složce */oldroot*.

17. Jakmile je připojování a záměna kořenových souborových systémů kompletní, jsme hotovi s */linuxrc* skriptem a spustí se skutečný program */sbin/init*.

18. *Init* načte soubor */etc/inittab* a začne nastavovat prostředí pracovní stanice

19. Jedna z prvních položek souboru *inittab* je příkaz *rc.local***,** který poběží zatímco pracovní stanice bude ve stavu inicializace (*sysinit*).

20. Script *rc.sysinit* založí 1 MB ramdisk pro shromažďování všech věcí, které bude třeba zapsat nebo jiným způsobem modifikovat.

21. Ramdisk bude připojen jako adresář */tmp*. Každý soubor do kterého bude třeba zapisovat bude aktuálně existovat v adresáři */tmp* a budou na něj ukazovat symbolické odkazy.

22. Připojí se souborový systém */proc*.

23. Provede se rozbor souboru *lts.conf* a všechny parametry které náleží pracovní stanici budou nastaveny jako proměnné prostředí pro použití skriptem *rc.local*.

24. Jestliže je stanice nakonfigurována na odkládání přes NFS (swap over NFS), bude exportovaná složka */var/opt/ltsp/swapfiles* na klientovi připojena do */tmp/swapfiles*. Poté, pokud ještě není pro tuto stanici vytvořen, bude automaticky vytvořen odkládací soubor dle parametru získaného z *lts.conf*. Odkládací soubor bude zpřístupněn příkazem *swapon*.

25. Je nakonfigurováno síťové rozhraní *loopback*, s IP adresou 127.0.0.1

26. Jestliže je nastaveno používání lokálních aplikací, pak je připojen adresář */home* a dojde k zpřístupnění domácích adresářů.

27. Je vytvořeno několik adresářů v souborovém systému */tmp* pro držení dočasných souborů potřebných dokud systém běží. Budou založeny tyto adresáře:

    - */tmp/compiled*
    - */tmp/var*
    - */tmp/var/run*
    - */tmp/var/log*
    - */tmp/var/lock*
    - */tmp/var/lock/subsys*

28. Nyní dojde k nakonfigurování systému X Windows. V souboru *lts.conf* se nachází parametr *XSERVER*. Pokud je tento parametr vynechán nebo je nastaven na hodnotu *„auto“*, dojde k automatické detekci grafických karet. Následně podle hodnoty *PCI Vendor* (výrobce) a *id* (identifikátoru) zařízení proběhne prohledání souboru */etc/vidlist*. Pokud je karta podporovaná serverem *XFRee86 4.X* vrátí program *pci_scan* jméno potřebného modulu. Je-li podporovaná pouze serverem verze *XFree86 3.3.6 pci_scan* vrátí jméno X serveru, který se má použít.

29. Pokud se použije *XFree86 4.X*, zavolá se skript */etc/rc.setupx* pro vytvoření souboru *XF86Config*, pokud *XFree86 3.3.6*, potom je konfigurační soubor vytvořen skriptem */etc/rc.setup3*.

30. Po ukončení skriptu *rc.setupx* je řízení vráceno zpět do *rc.local*. Následně je vytvořen skript */tmp/start_ws*, který je zodpovědný za start X serveru.

31. Zakládá se */tmp/syslog.conf*. Tento soubor bude obsahovat informace, které říkají programu *syslogd*, na kterému počítači v síti se mají vytvářet *log* soubory a posílat patřičné informace. Host *syslog* je specifikován v souboru *lts.conf*. Je zde symbolický odkaz nazvaný */etc/syslog.conf*, který ukazuje na soubor */tmp/syslog.conf*.

32. Je nastartován program *syslogd* za použití konfiguračního souboru vzniklého v předešlém kroku.

33. Řízení je opět vráceno zpět do *init*. *Init* vyhledá nastavenou hodnotu *initdefault* určující implicitní úroveň běhu (*runlevel*), do které má být systém nastartován. Od *lts_core-2.08* je hodnota *initdefault* rovna *2*.

34. Úroveň 2 způsobí spuštění skriptu ***set_runlevel*,** který pomocí souboru *lts.conf* určí v jaké úrovni bude pracovní stanice běžet.

    Standardními úrovněmi jsou pro LTSP 3, 4 a 5:

    - 3 – nastartuje *shell*
    - 4 – *Telnet*, jedno nebo více sezení
    - 5 – Nastartuje X Windows server a vyšle se *XDMCP* dotaz na server, který zobrazí dialogové okno pro přihlášení do systému některého z přihlašovacích manažerů XDM, GDM nebo KDM.

5.5 Lokální aplikace z pohledu klienta

LTSP nám umožňuje spouštět aplikace různými způsoby buď vzdáleně na serveru nebo lokálně na pracovní stanici. Nejsnadnější cestou je spouštět aplikace na serveru. Klientská aplikace běžící na serveru používá CPU a paměť serveru, výstup se zobrazuje na monitoru stanice a pro vstupy je využívána klávesnice s myší pracovní stanice. Toto je základní schopnost X Windows a říkáme že pracovní stanice pracuje jako standardní X Windows terminál.

Aby mohl uživatel spouštět aplikace lokálně na pracovní stanici, potřebujeme znát některé systémové informace o uživateli:

- ID uživatele
- Skupinu do které uživatel patří
- Domovský adresář uživatele

LTSP v tomto ohledu spoléhá na síťovou službu NIS (Network Information Services) dříve nazývanou „Yellow Pages“, která předává pracovním stanicím informace o uživateli.

5.5.1 Výhody lokálních aplikací

Výhody lokálně běžících aplikací jsou následující:

- Redukují náklady na server. V rozsáhlých sítích s paměťově náročnými aplikacemi běh aplikace na pracovní stanici může poskytnout lepší výkon, v případě že pracovní stanice disponuje dostatečným výkonem a množstvím paměti aby to zvládla.
- Aplikace neovlivňuje ostatní uživatele.
- Snadněji se konfiguruje podpora zvuku, když zvuková aplikace běží přímo na pracovní stanici.

5.5.2 Nastavení podpory pro lokální aplikace

Schopnost lokálního běhu aplikací klade na pracovní stanici i síť mnohem více nároků:

- Stanice potřebuje více RAM a výkonnější CPU.

- NIS – pro řízení lokální aplikace na stanici byla vybrána služba NIS, která slouží k ověřování uživatelů.

- Musíme na NFS serveru exportovat další adresáře potřebné pro tyto aplikace. To může způsobit nemalé zvýšení síťové aktivity.

- Pomalejší start aplikací. Protože každá aplikace běží na vlastní CPU, ztrácíme výhodu sdílet kód mezi více instancemi stejného programu. Sdílení redukuje čas, který potřebuje každá další instance programu ke svému spuštění.

6 Testování výpočetního klastru

V kapitole č. 3 jsme si řekli, že openMosix je rozšíření linuxového jádra pro paralelizaci samostatných systémů. Když se nad tím zamyslíme, můžeme již z této věty obecně odvodit, jaké výpočty je vhodné řešit výpočetním klastrem a jaké nikoliv. Pro pomoc s naším zamýšlením si uvedeme zjednodušenou úlohu s výpočtem prvočísel v intervalu 1-1000.

Pokud budeme hledat prvočísla od 1 do 1000 jedním počítačem, můžeme počítat například vzestupně od 1, a nebo si interval 1-1000 rozdělit na několik, třeba 10, intervalů. Pokud zvolíme rozdělení na více intervalů, můžeme, ale nemusíme, si jednotlivé intervaly počítat na přeskáčku, výsledný čas nalezení všech prvočísel však bude ve všech případech stejný. Jiná situace nastává, když budeme mít počítačů 10, všechny budou stejně rychlé a ve stejný čas jim zadáme tu samou úlohu, hledání prvočísel v intervalu 1-1000. I přesto, že všechny počítače pilně počítaly svou úlohu, vyřešili ji za stejný čas jako jeden počítač. Úloha se vyřešila, ale všichni víme, že to šlo lépe. V tomto případě je vhodné použít ono rozdělení do intervalů, úlohu tak rozmělnit na 10 dílčích úloh s částečnými výsledky a každému z 10-ti počítačů poslat interval jen jeden. Výsledek obdržíme 10x rychleji. Když pak výsledky výpočtů necháme jednoduchým programem posílat přes počítačovou síť do souboru, který si zobrazujeme na monitoru u kterého sedíme, tak můžeme našemu řešení s trochou nadsázky říkat výpočetní klastr.

Pro úplné pochopení a tím získání znalostí pro otestování našeho klastru, si musíme uvědomit, jak rozdělit úlohu pod systémem Linux a jak pracuje klastr openMosix. Na Linuxu je každá úloha reprezentována v systému systémovým procesem. Pokud se jedná o úlohu volanou nějakým nadřazeným procesem, pak říkáme, že se jedná o podproces rodičovského procesu. Vlastní proces vzniká po spuštění jakéhokoliv programu. Mám-li navázat na předchozí úlohu s prvočísly, není nic jednoduššího, než v libovolném skriptovacím nebo jiném jazyce napsat skript, který jako parametr bude mít interval, ve kterém se mají hledat prvočísla, ten spustit tolikrát a se všemi intervaly, ve kterých chceme hledat a je vše vykonáno. V systému se objeví tolik procesů kolikrát byl skript spuštěn, klastr openMosix si zjistí jejich počet, vytíženost a dostupnost uzlů a bez našeho přičinění je rozdistribuuje na ty části klastru, kde jsou volné systémové prostředky. To je vše, co je nutné, znát pro testování.

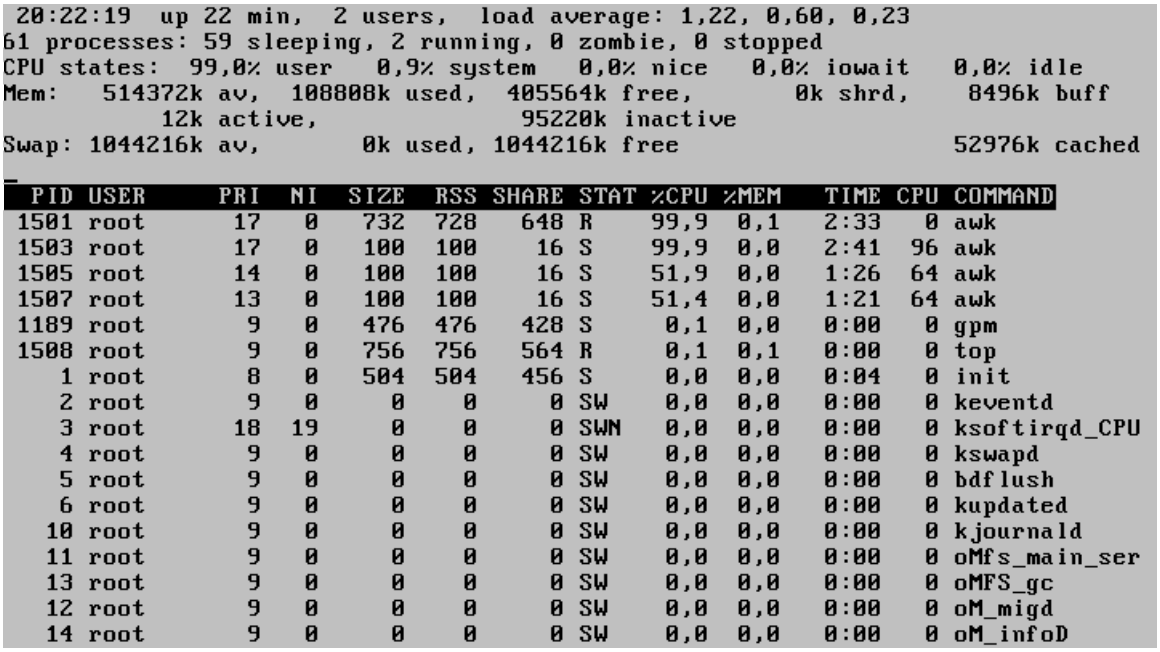
6.1 Testovací skript

Jakmile máme nainstalovaný server a nakonfigurované terminálové klienty přichází řada na testování našeho řešení. Pro první zatížení klastru není třeba komplikovaně instalovat nějaké reálné aplikace, ale na maximální vytížení klastru bude dostačovat jednoduchý skript s nekonečným cyklem. Jednoduchý dvouřádkový skript lze realizovat pomocí shellu a univerzálního počítačového jazyka AWK. Výsledný skript pak vypadá takto:

```
#!/bin/sh
awk 'BEGIN {while(1);}' &
```

Pro samotné otestování stačí skript spustit tolikrát, kolik máme v klastru aktivních uzlů. Mezi spuštěním jednotlivých instancí je vhodné programem *mosmon* nebo

jakýmkoliv z monitorovacích nástrojů, jejichž instalaci jsme si podrobněji popsali v kapitole 4.5.3, sledovat postupné přerozdělování systémových prostředků. To je způsobené migrací jednotlivých instancí testovacího skriptu. Spustíme-li ze serveru nebo z libovolného uzlu čtyři instance testovacího skriptu, můžeme v programu *top* (viz. obrázek č.19) pozorovat následující výpis. Otisk obrazovky z programu *mosmon* (viz. obrázek č.20) nám pak zobrazuje tomu odpovídající rozložení zátěže na jednotlivých uzlech. Otestováno máme a skripty můžeme postupně ukončovat z programu top stisknutím klávesy „k“ a zadáním příslušného PID procesu. Pokud jsme skript spustili mnohokrát máme možnost hromadně ukončit všechny příkazem *killall awk*. Video z testování a skript *loopit.sh*, obsahující nekonečnou smyčku, jsou přiloženy na DVD, které je součástí této práce.



```
 20:22:19  up 22 min,  2 users,  load average: 1,22, 0,60, 0,23
61 processes: 59 sleeping, 2 running, 0 zombie, 0 stopped
CPU states:  99,0% user   0,9% system   0,0% nice   0,0% iowait   0,0% idle
Mem:   514372k av,  108808k used,  405564k free,       0k shrd,    8496k buff
           12k active,              95220k inactive
Swap: 1044216k av,       0k used, 1044216k free                   52976k cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME CPU COMMAND
 1501 root      17   0   732  728   648 R    99,9  0,1   2:33   0 awk
 1503 root      17   0   100  100    16 S    99,9  0,0   2:41  96 awk
 1505 root      14   0   100  100    16 S    51,9  0,0   1:26  64 awk
 1507 root      13   0   100  100    16 S    51,4  0,0   1:21  64 awk
 1189 root       9   0   476  476   428 S     0,1  0,0   0:00   0 gpm
 1508 root       9   0   756  756   564 R     0,1  0,1   0:00   0 top
    1 root       8   0   504  504   456 S     0,0  0,0   0:04   0 init
    2 root       9   0     0    0     0 SW    0,0  0,0   0:00   0 keventd
    3 root      18  19     0    0     0 SWN   0,0  0,0   0:00   0 ksoftirqd_CPU
    4 root       9   0     0    0     0 SW    0,0  0,0   0:00   0 kswapd
    5 root       9   0     0    0     0 SW    0,0  0,0   0:00   0 bdflush
    6 root       9   0     0    0     0 SW    0,0  0,0   0:00   0 kupdated
   10 root       9   0     0    0     0 SW    0,0  0,0   0:00   0 kjournald
   11 root       9   0     0    0     0 SW    0,0  0,0   0:00   0 oMfs_main_ser
   13 root       9   0     0    0     0 SW    0,0  0,0   0:00   0 oMFS_gc
   12 root       9   0     0    0     0 SW    0,0  0,0   0:00   0 oM_migd
   14 root       9   0     0    0     0 SW    0,0  0,0   0:00   0 oM_infoD
```

Obr. č. 19 – Program top zobrazující čtyři instance testovacího skriptu (awk)

Obr. č. 20 – Program mosmon zobrazující vytížení uzlů pro čtyři instance testovacího skriptu (awk)

6.2 Cisilia

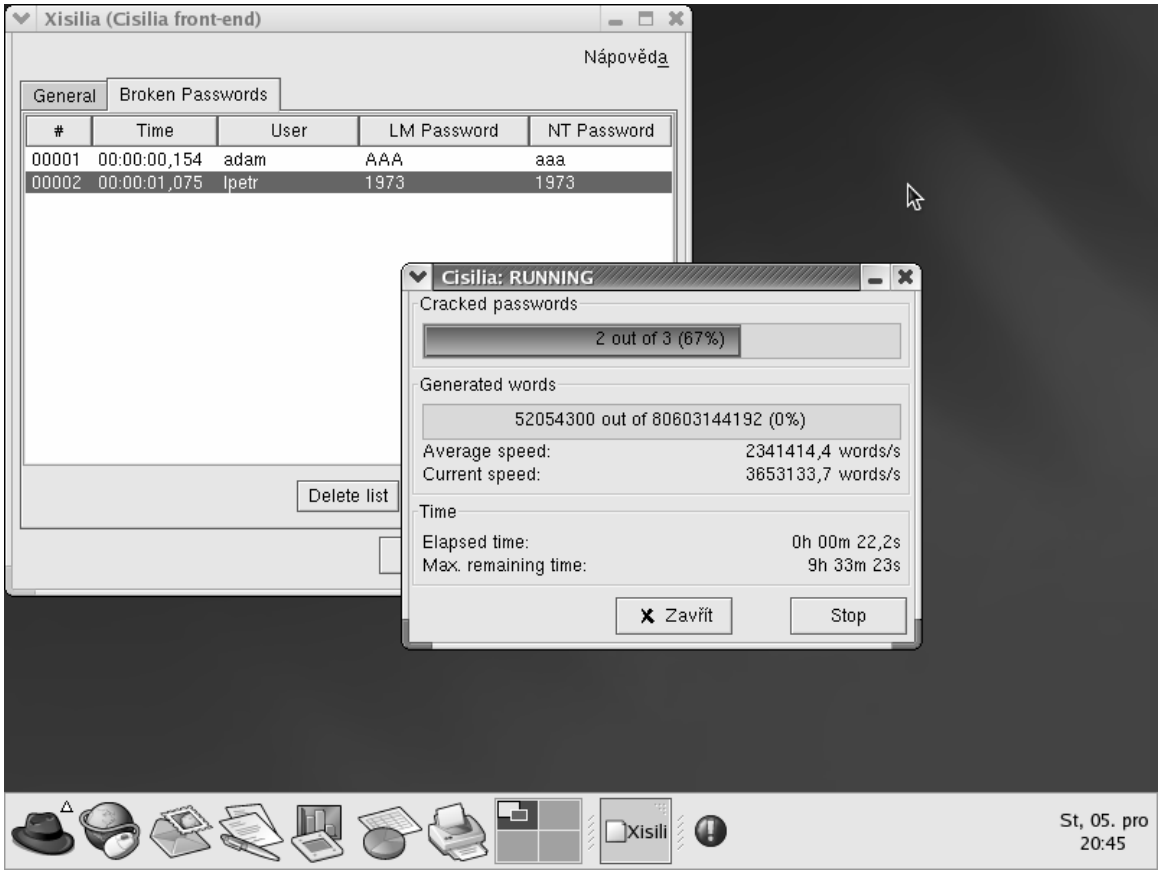
Cisilia [7] je multiprocesní systém pro lámání hesel. Současná verze (0.7.3) je schopná zrekonstruovat hesla uživatelských účtů systémů Windows NT/2000/XP a Samby z DES/MD4 hash. Ačkoliv Cisilia obsahuje schéma pro obnovu běžných hesel za pomoci slovníku, primárně je systém zaměřen na útok pomocí hrubé síly. To je důvod proč je navržen jako multiprocesní paralelní systém s možností běhu na multiprocesorech (SMP) nebo systémech typu klastr s vyrovnáváním zátěže (openMosix, MOSIX). Cisilia rozdělí rozsahy možných hesel mezi podprocesy, které sami vytvoří několik svých procesů „potomků“. Jestliže je Cisilia spuštěna na klastru s vyrovnáváním zátěže, jsou potomci migrováni na ostatní nody, čímž dochází ke zvýšení výpočetní rychlosti. Cisilia také disponuje flexibilním a inteligentním mechanismem pro testování různých znakových sad (abeced) v takovém pořadí, aby došlo k urychlení úlohy rekonstrukce hesla.

Xisilia je grafickou nadstavbou programu Cisilia s uživatelsky přívětivějším rozhraním.

Pro konečné otestování a demonstraci síly výpočetního klastru jsem zvolil program *xisilia*. Jako vstup pro tento program jsem použil reálný soubor s uživatelskými jmény a hesly uloženými pomocí MD4 hash. Vypsaný soubor uvádím níže. Na obrázku č.21 vidíme aktuální stav po několika vteřinách běhu programu, po prolomení triviálních hesel uživatelských účtů *adam* a *lpetr*. Uživatelský účet *eva* má vytvořeno komplikovanější heslo, sestávající z několika znaků a číslic, jehož prolomení bude vyžadovat více času. Kompletní video zachycující start výpočetního klastru a proces lámání hesel je součástí přiloženého DVD.

Výpis souboru *smbpasswd*:

```
lpetr:1017:3115B8996C8503EAAAD3B435B51404EE:BE07529D7D2D42D85C5E04C6E3
73C8E0:[U           ]:LCT-46F76F70:
adam:1018:1C3A2B6D939A1021AAD3B435B51404EE:E24106942BF38BCF57A6A4B2901
6EFF6:[U           ]:LCT-47259DBB:
eva:1019:9C1CAD563FCDB7F59C5014AE4718A7EE:DCE2074EB5DF09A8F1C017540044
E7A7:[U           ]:LCT-47259DC8:
```



Obr. č. 21 – Lámání hesel programem Xisilia

7 Závěr

Hlavním cílem absolventské práce bylo seznámit čtenáře s technologiemi LTSP a openMosix. Propojením těchto dvou technologií vzniklo komplexní řešení, násobící přínosy na úroveň, které by vedle sebe samostatně stojící řešení nikdy nedosáhly. Projekty LTSP a openMosix byly použity k vytvoření různorodé, na architektuře nezávislé, počítačové sítě, složené ze serveru a tenkých klientů, navzájem spojených do výpočetního klastru. Vzniklá síť, zároveň tvořící výpočetní klastr, je schopná uspokojit nejen uživatele, který využívá běžné aplikace, ale i uživatele požadujícího, pro řešení složitých a časově náročných výpočtů, maximální dostupný výkon. Tuto práci jsem se snažil napsat formou natolik podrobnou, aby byly zřejmé všechny podstatné oblasti, na kterých závisí funkční propojení zmiňovaných technologií, a aby mohla posloužit jako návod pro bezproblémové vytvoření kompletního řešení. Některé kapitoly, zejména kapitola zabývající se odhadováním výkonu, by pak měly sloužit jako doporučení, ze kterého je možné vycházet při vlastním návrhu sítě s odlišnými požadavky, než ze kterých jsem vycházel v mém případě. Kapitola týkající se testování výpočetního klastru a aplikace pro lámání hesel Cisilia představila využití řešení na praktické paralelizovatelné úloze v praxi a snad dala rozhled v tom, jakým způsobem lze řešení využít.

Přílohy

1. Seznam literatury a internetových zdrojů

[1] http://www.ltsp.org/
[2] http://openmosix.sourceforge.net/
[3] http://www.lpmo.edu/~daniau/ltsp-mosix/
[4] http://www.openmosixview.com/
[5] http://etherboot.sourceforge.net/doc/html/mknbi.html
[7] http://www.citefa.gov.ar/SitioSI6_EN/si6.htm

2. Seznam zkratek a výrazů obsažených v textu

AMD – anglická zkratka z Advanced Micro Devices. Jde o jednoho z největších výrobců integrovaných obvodů současnosti. AMD v roce 2006 převzalo společnost ATI, která produkovala grafické čipy, čipové sady a komponenty pro spotřební elektroniku.

AWK - je univerzální počítačový jazyk, navržený pro zpracovávání textových dat, ať už v podobě textových souborů nebo proudů. Název AWK je odvozen z příjmení svých tvůrců, kterými jsou Alfred V. **A**ho, Peter J. **W**einberger a Brian W. **K**ernighan.

BootROM – paměť typu ROM kterou bývají osazené síťové karty. Může obsahovat Etherboot nebo běžnější PXE.

démon – (z anglického daemon) – jde o proces systému Linux zpravidla zajišťující nějakou službu nebo činnost. Démon běží skryt na pozadí a o jeho činnosti běžný uživatel nemusí vědět.

GPL – GNU General Public Licence – jde o velmi často používanou licencí volně šiřitelného software. Jejím autorem je Richard Stallman.

HUB – jde o síťový rozbočovač, který je jako aktivní prvek schopen větvení sítě ethernet a tvoří základ hvězdicové topologie sítě. V dnešní době je nahrazován jeho nástupce přepínačem (switch).

Hyper-Threading – jedná se o technologii firmy Intel, která se poprvé objevila u procesorů Pentium 4. Tato technologie zvyšuje výkon procesoru v prostředí víceúlohových systémů.

KLASTR – z anglického cluster = shluk, hvězdokupa, seskupení. Počítačový klastr je dvojice či skupina autonomních počítačů, které velmi úzce spolupracují na konkrétní úloze, nebo společně poskytují určitou službu. Nejběžnější druhy klastrů na které narazíme ve světě informačních technologií:

- vysoce dostupný klastr (High-availability cluster)
- výpočetní klastr (High-performance computing cluster)
- úložný cluster (Storage cluster)
- serverový cluster (Server cluster)

LILO, GRUB – zavaděče operačního systému běžně dodávané s distribucemi OS Linux.

Live distribuce – tímto pojmem jsou běžně označovány distribuce operačního systému Linux, které jsou schopné běžet bez přímé instalace na pevný disk. Ke konfiguraci hardware dojde automaticky při startu systému ze spustitelného CD, diskety nebo paměti typu flash.

NIS – Network Information Services

NOD – odborný termín pro uzel klastru.

OOM – Out Of Memory – je, pro operační systém, velmi nezdravý stav, kdy dojde k vyčerpání veškeré operační paměti i odkládacího prostoru.

Opteron – typ procesoru od společnosti AMD. Jde o první procesor osmé generace založený na architektuře x86-64 (AMD64). Byl vydán 22. dubna 2003 a je primárně určen pro serverový trh.

Out Of Memory Killer - je záchranný program, který, pokud nastane OOM, doslova obětuje některý běžící program, pokud selžou všechny ostatní pokusy o uvolnění operační paměti.

POSIX – (Portable Operating System Interface) je přenositelné rozhraní pro operační systémy, standardizované jako IEEE 1003 a ISO/IEC 9945, vycházející ze systémů UNIX. Určuje, jak mají POSIX-konformní systémy vypadat, co mají umět a co se jak dělá.

RAID – Redundant Array of Independent Disks (redundantní pole z nezávislých disků) – jde o typ diskových řadičů, zajišťující pomocí speciálních funkcí koordinovanou práci dvou a více fyzických pevných disků.

SMP – Symmetric MultiProcessing – technologie umožňující použití více procesorů na jedné základní desce. Procesory sdílejí paměť a mohou se podílet na zpracování stejné úlohy.

3. Seznam elektronických příloh na DVD

Popis přílohy	**Umístění na DVD**
Bakalářská práce - Adam Jalůvka (PDF)	/BP_Adam_Jaluvka_LTSP+openMosix.pdf
Instalační balíčky LTSP3.0 + manuál	/LTSP3/
openMosix patch pro jádro verze 2.4.22	/openMosix-2.4.22-2
Nástroje openMosix	/openmosix-tools-0.3.6-2.i386.rpm
Zdroje a konfigurace jádra 2.4.22	/Kernel/
Instalační balíčky LTSP+openMosix	/LTSP_openMosix
Instalační balíček programu Csilia	/Cisilia/
Instalační balíček programu Xisilia	/Xisilia/
Skripty použité v práci	/Sripts/
Videa z testování	/Videos/
ROM obrazy použitých NIC	/NIC_ROM/
Instalační balíček openMosixView 1.5	/openmosixview-1.5-redhat90.i386.rpm
Nástroje MKNBI	/mknbi-1.4.4-3.src.rpm
Testovací soubor s kódovanými hesly	/smbpasswd
Operační systém Red Hat 9 (.img)	/Red_Hat_9/
VMWare workstation 6 (Shareware)	/VMWare/
Komprimovaný obraz serveru (VMWare)	/Hive – Red Hat Linux 9.zip
Komprimovaný obraz terminálu 1	/Term1.zip
Komprimovaný obraz terminálu 2	/Term2.zip