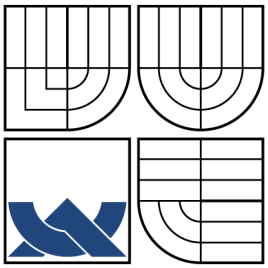
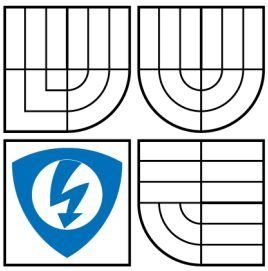


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

VYTVOŘENÍ MODULU JÁDRA A KONFIGURACE SÍŤOVÉHO SERVERU PRO OPERAČNÍ SYSTÉM LINUX

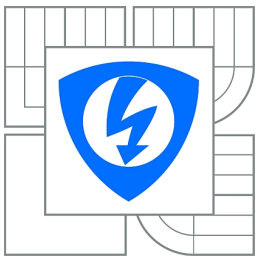
KERNEL MODULE DEFINITION AND NETWORKING SERVER CONFIGURATION FOR LINUX OPERATING SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. LADISLAV PECHO
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. DAN KOMOSNÝ, Ph.D.
SUPERVISOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student: Bc. Ladislav Pecho ***ID:*** 83374
Ročník: 2 ***Akademický rok:*** 2009/2010

NÁZEV TÉMATU:

Vytvoření modulu jádra a konfigurace síťového serveru pro operační systém Linux

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s problematikou vytvoření, kompilace a zavedení modulu jádra pro operační systém GNU/Linux. Popište nezbytné části zdrojového kódu jaderného modulu a vysvětlete k čemu slouží. Dále se seznamte s konfigurací serverů DHCP a TFTP. Servery nakonfigurujte tak, aby umožnily startování klientských stanic přes síť. Uvedené postupy zpracujte ve formě podkladů pro realizaci počítačových cvičení.

DOPORUČENÁ LITERATURA:

[1] PUŽMANOVÁ, R. TCP/IP v kostce. 1. vyd. České Budějovice : Kopp, 2004. 607 s. ISBN 80-7232-236-2.
[2] NEMETH, E., SNYDER, G., HEIN T. Linux - Kompletní příručka administrátora. Computer Press, 2004. 880 s. ISBN: 80-722-6919-4.

Termín zadání: 29.1.2010 ***Termín odevzdání:*** 26.5.2010

Vedoucí práce: doc. Ing. Dan Komosný, Ph.D.

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Účelem mé práce je vytvořit dvě počítačová cvičení. První z nich ukazuje, jak napsat jednoduchý modul pro jádro Linuxu, následně předvádí jako ukázku vytvoření souboru v systému procfs, jehož obsah může být rozhodnut až v okamžiku jeho čtení. Druhé cvičení popisuje konfiguraci serveru pro spuštění počítače přes síť. Využívá se serveru DHCP a přenosu souborů pomocí serveru TFTP, které obstará program dnsmasq. Ukázková konfigurace zobrazí na klientské stanici nabídku, ze které je možné volit mezi jádrem Linuxu, obrazem diskety, síťovým instalátorem Linuxu, diagnostickým nástrojem Memtest, nebo hrou Grub Invaders, běžící v textovém režimu. Použitý zavaděč je pxelinux z projektu syslinux.

KLÍČOVÁ SLOVA

Linux, jádro, modul, procfs, DHCP, TFTP, dnsmasq, pxelinux

ABSTRACT

The goal of my work is to create two computer labs lessons. The first one is a guide in writing a simple Linux kernel module, and in making a file inside the procfs filesystem. The content of the file can be decided as late as at the moment of reading. The second lesson describes the configuration of a server which allows a computer to boot over network. DHCP server is used with a TFTP server for file transfers which are both handled by the dnsmasq program. The example configuration will display a menu on the client station. It is possible to select to boot either a Linux kernel, floppy disk image, Linux network installer, diagnostic tool Memtest, or a text mode game Grub Invaders. The chosen bootloader is pxelinux from the syslinux project.

KEYWORDS

Linux, kernel, module, procfs, DHCP, TFTP, dnsmasq, pxelinux

PECHO, Ladislav *Vytvoření modulu jádra a konfigurace síťového serveru pro operační systém Linux*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2010. 60 s. Vedoucí práce byl doc. Ing. Dan Komosný, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Vytvoření modulu jádra a konfigurace síťového serveru pro operační systém Linux“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu práce doc. Ing. Danu Komosnému, Ph.D. za velmi užitečnou metodickou pomoc a cenné rady při zpracování diplomové práce.

V Brně dne ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

Úkolem diplomové práce je seznámit se s problematikou programování modulu jádra pro operační systému Linux a dále možnosti spuštění počítače bez operačního systému přes síť. Stěžejním úkolem bylo vytvořit návod, který tuto problematiku umožní studentům pochopit v rámci počítačových cvičení a především si vše vyzkoušet.

Stručně zde uvedu odkazy na literaturu a myšlenkový proces, který vedl k řešení. Samotné odkazy na literaturu nedávám do těla návodu pro studenty, jelikož by působily rušivě. Dle mého názoru student nemusí vědět o problémech, na které jsem narážel, ale potřebuje mít přehledně konečné řešení.

Pro modul jádra jsem jako výchozí bod použil Průvodce programování jádra Linuxu [1]. První odstavec teoretického úvodu pro studenty jsem vypracoval podle tohoto pramene. Tento zdroj ovšem počítá se starší verzí jádra, tudíž některé postupy se mi už dnes nepodařilo zopakovat (kompilace modulu, definice parametrů měnitelných z příkazové řádky). S prvním mi pomohl článek [2], druhý problém jsem vyřešil tím, že jsem se podíval, jak jsou proměnné definovány u existujících modulů v současném jádře. V samotném zdrojovém kódu [3] mi orientaci velmi usnadňoval vyhledávač [4]. Pro lepší pochopení detailů jsem použil podrobnou knihu o ovladačích v Linuxu [5, kap. 2]. Aby praktická ukázka pro studenty nebyla závislá na konkrétním hardware (anebo jeho implementaci ve virtuálním systému, se kterým studenti pracují), bylo nejvhodnější využít nějakou funkčnost jádra, která není spojená s hardwarem, ale zároveň není dostupná programátorům v userspace (uživatelský prostor, běžné programy). K tomu jsem využil příklad pro práci se systémem souborů procfs, který je ve zdrojovém kódu jádra [3, 4] umístěn v adresáři `Documentation/DocBook/procfs_example.c`. Tento příklad jsem upravil a rozšíril a je použit v samotném cvičení.

Pro spouštění počítače pomocí DHCP serveru jsem našel několik programů. Syslinux, gPXE a GRUB [6, 7, 8]. Syslinux není jen síťový zavaděč, ale jedna jeho komponenta, isolinux, umí zavádět i jádra z disků CD, lze v něm pomocí konfiguračních skriptů definovat i poměrně komplexní nabídku, ze které uživatel může vybírat ještě před startem systému. To zaručilo jeho poměrně velkou rozšířenost na instalačních CD velkého počtu dnešních distribucí. I z toho důvodu jsem ho nakonec vybral pro realizaci ukázky. Projekt gPXE (dříve etherboot) je primárně hlavně náhradou pro firmware, který je uložený v síťových kartách. Obvykle síťová karta po získání nastavení konfigurace z DHCP serveru zbytek přenosů realizuje pomocí protokolu TFTP[1]. Naopak gPXE přináší podporu pro protokol HTTP, umožňuje

[1] Trivial File Transfer Protocol

start systému z úložiště SAN[2] pomocí AoE[3] a iSCSI[4]. Přitom samotný gPXE nemusí být spuštěn výhradně z paměti na síťové kartě - může být spuštěn i později, jako klasický zavaděč přes síť, nebo může být na disketě, USB disku apod. Jako poslední jsem objevil upravený GRUB s podporou PXE. Jde ale o starší verzi a zdá se, že tato úprava není příliš využívána. Osobně jej chápu jako experiment, který se moc neujal.

Za zmínku stojí projekt BKO [9], který byl spuštěn teprve 21. září 2009. Jde o upravený obraz gPXE, který dokáže pomocí protokolu HTTP spustit celou řadu nástrojů a distribucí po internetu přímo ze serveru projektu. Ve skutečnosti tento upravený gPXE také využívá syslinux. gPXE přináší podporu HTTP přenosů, syslinux je použit pro zobrazení nabídky a spuštění jádra s ramdiskem. Ramdisk obsahuje upravený skript a ovladač, který vyhledá kořenový systém souborů na internetu opět pomocí HTTP protokolu.

Pro praktickou ukázku spouštění po síti jsem vybral programy Memtest, FreeDOS, a hru Invaders [11, 10, 12]. Hra Invaders používá jiný formát binárního souboru než klasické jádro a v syslinuxu jsem měl problém ho spustit (GRUB si s ním poradil bez problému). Zřejmě tento problém časem zmizí, prozatím jsem zvolil řešení použít balíček [13] z distribuce Debian a rozbalit ho ručně [14].

Podle požadavku mělo být cvičení realizovatelné na školním systému s distribucí CentOS, ale úkony jsou na distribuci nezávislé. Drobnější komplikací v CentOS byla starší verze syslinuxu, která nepodporovala některé příkazy, proto je v návodu uvedený postup, jak získat aktuální verzi. Doporučuji nejnovější stabilní verzi, protože s testovací verzí 4.00-pre6 jsem narazil na několik chyb. V testovací verzi byla zapomenutá hláška, která se objevila při každém spuštění, nebylo možné spustit memtest a síťový instalátor Debianu nefungoval, protože vznikl konflikt se starším souborem vesamenu.c32, který by jinak bylo nutné nahradit novější verzí.

[2]Storage Area Network

[3]ATA over Ethernet, funguje na 2. vrstvě referenčního modelu ISO/OSI

[4]Internet Small Computer System Interface, SCSI over IP, pracuje na 3. vrstvě referenčního modelu ISO/OSI

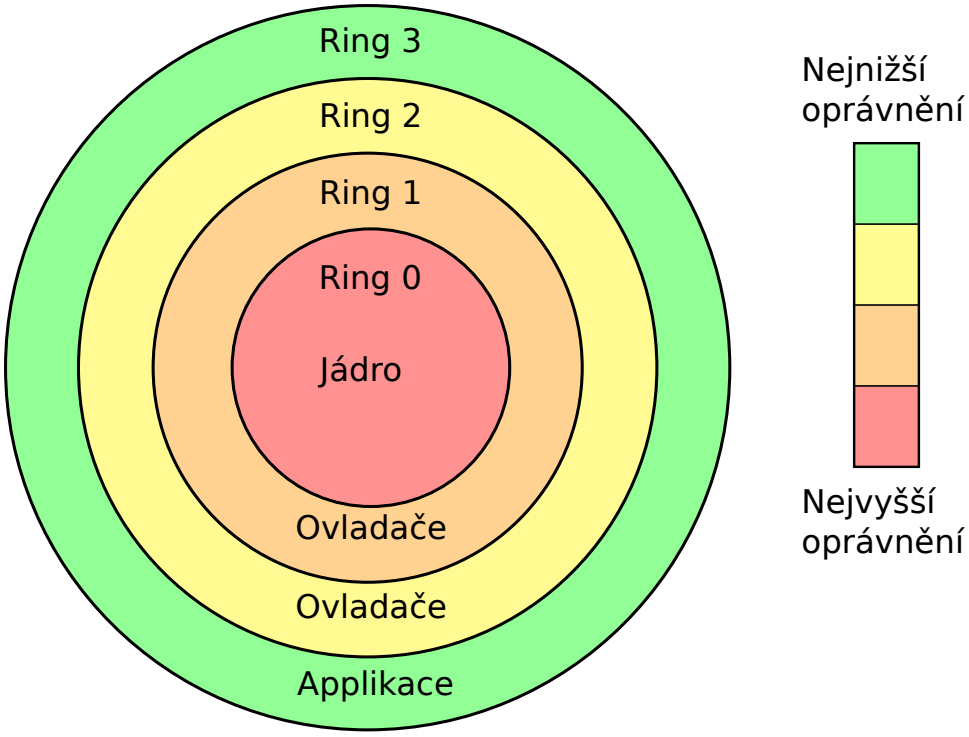
1 TEORETICKÁ ČÁST

1.1 Typy jader operačních systémů

Jádro je základní část všech operačních systémů. Je to nezbytná a tudíž nejdůležitější součást.

Jeho hlavní úlohou je ovládat hardwarové prostředky počítače a zpřístupnit je aplikacím pomocí rozhraní. Aplikace tyto prostředky mohou využívat souběžně. Jádro je tedy vrstvou mezi hardwarem a aplikacemi, které používá uživatel.

V architektuře operačních systémů můžeme zjednodušeně pozorovat dva základní režimy. Privilegovaný mód, který běží v režimu jádra (kernel space), a uživatelský mód (user space), ve kterém jsou spuštěny běžné aplikace. Některé operační systémy rozlišují více úrovní privilegií (např. ring 0-3, viz. obr. 1.1), nicméně obecně platí, že aplikace (procesy) běží v uživatelském režimu s nejnižšími privilegii, zatímco důležitější součásti používají vyšší privilegia, někdy stejná jako samotné jádro. Podle uspořádání těchto „důležitějších součástí“ vůči jádru můžeme rozlišit několik druhů jader.



Obr. 1.1: Úrovně privilegií: Některé operační systémy rozlišují několik kruhů (rings) oprávnění pro režim jádra, ovladačů zařízení a aplikací.

Nejčastější architekturou jader operačních systémů jsou: monolitické jádro (ma-

ximum funkčnosti je v prostoru jádra), mikrojádro (naopak se snaží co nejvíce funkcí implementovat v uživatelském prostoru), další typy jsou hybridní jádro (kompromis mezi těmito dvěma), exokernel (implementuje přístup k prostředkům s minimální abstrakcí). Ještě se někdy rozlišuje nanokernel (velmi malé mikrojádro) a modulární jádro. [15]

1.1.1 Monolitické jádro

Název monolitický pochází z řeckého monolithos, kde mono znamená jediný, a lithos znamená kámen.[16] Tedy jediný, celistvý; pevný kus.

Operační systém s monolitickým jádrem obsahuje veškeré systémové služby, správu paměti, obsluhu přerušení a vstupně výstupní komunikace, systém souborů, atd. přímo v prostoru jádra (obr. 1.2). Je postaveno po vrstvách, základy tvoří základní řízení procesů, nad rozhraním se zbytkem operačního systému jsou knihovny a aplikace.[17]

Vložení všech základních systémových funkcí do prostoru jádra přináší tři zásadní komplikace: velikost jádra, nemožnost rozšířitelnosti a obtížnou údržbu. Ladění, nebo přidání nových vlastností znamená překompilování celého jádra. To je nákladné na čas a prostředky, protože kompilace nového jádra může trvat několik hodin a vyžaduje poměrně hodně paměti.

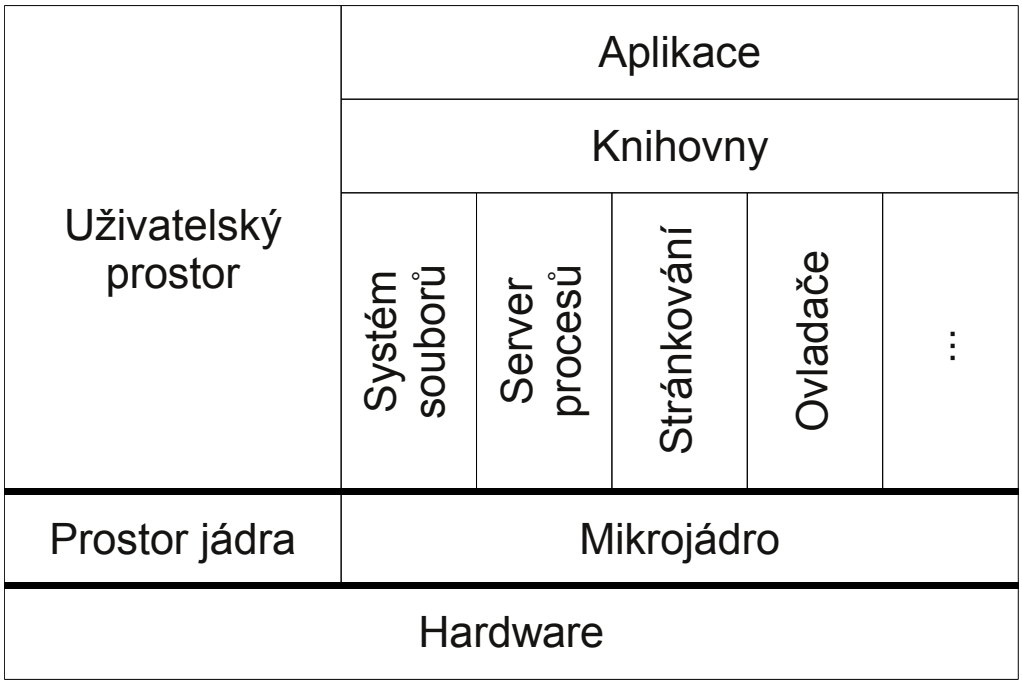
Obr. 1.2: Schéma OS s monolitickým jádrem

1.1.2 Mikrojádro

Aby se vyřešilo omezení rozširitelnosti a údržby monolitických jader, vznikl nápad mikrojádra. Koncept (obr. 1.3) spočíval ve zredukování jádra na základní komunikaci s procesy a řízení vstupů a výstupů a umožnění ostatním systémovým službám být v uživatelském režimu v podobě obyčejných procesů (takzvaných serverů). Existuje server pro řízení událostí s pamětí, jiný server řeší management procesů, jiný má na starosti ovladače, atd.[17]

Protože servery již neběží v režimu jádra, je potřeba častější přepínání kontextu mezi procesy, aby uživatelské procesy mohly přejít do privilegovaného režimu a opět jej opustit. Tímto způsobem mikrojádro už není blok systémových služeb, ale reprezentuje pouze několik základních abstrakcí a objektů, které ovládají komunikaci mezi procesy, dále procesem a podřízeným hardwarem. Jelikož komunikace už neprobíhá přímo, vznikl systém pro řízení zpráv, který umožňuje nezávislou komunikaci a napomáhá rozšířitelnosti.

Výhodou mikrojader je jejich malá velikost (samotný kód v režimu jádra je skutečně malý), snadná rozšířitelnost o nové funkce, dále teoretická větší robustnost v případě chyb. Pokud zhavaruje nějaký server, je to obyčejný proces, který neohrozí běh jádra.



Obr. 1.3: Schéma OS s mikrojádrem

1.1.3 Hybridní jádro

Koncept hybridního jádra je pomyslným kompromisem mezi monolitickým jádrem a mikrojádrem.

Myšlenka spočívá v návrhu jeho struktury podobné mikrojádru, ale implementovanou ve smyslu monolitického jádra. Narozdíl od mikrojádra, všechny (nebo téměř všechny) služby operačního systému jsou v prostoru jádra.[17]

Hlavní motivací tohoto návrhu je mít jasnou vnitřní strukturu jádra, kde jednotlivé služby poskytují servery uvnitř jádra, zároveň nedochází k nutnosti častého přepínání kontextu mezi uživatelským režimem a režimem jádra, což je jinak časově poměrně náročná operace.

1.1.4 Exokernel

Původně se programátoři jader snažili zpřístupnit prostředky počítace aplikacím tím, že je nutili komunikovat s hardwarem pomocí nějakého koncepčního modelu. Tyto modely jsou třeba systém souborů pro úložný prostor na disku, virtuální adresní prostor pro práci s pamětí, plánovače pro správu úloh, sokety pro síťovou komunikaci.

Tyto abstrakce hardwaru obecně usnadňují tvorbu aplikací, ale omezují výkon a brání experimentům s novými abstrakcemi.

Aplikace zaměřená na bezpečnost může vyžadovat systém souborů, který nezanechává stará data na disku, zatímco aplikace zaměřená na spolehlivost může potřebovat systém souborů, který si tato data ponechává pro zotavení z chyb.

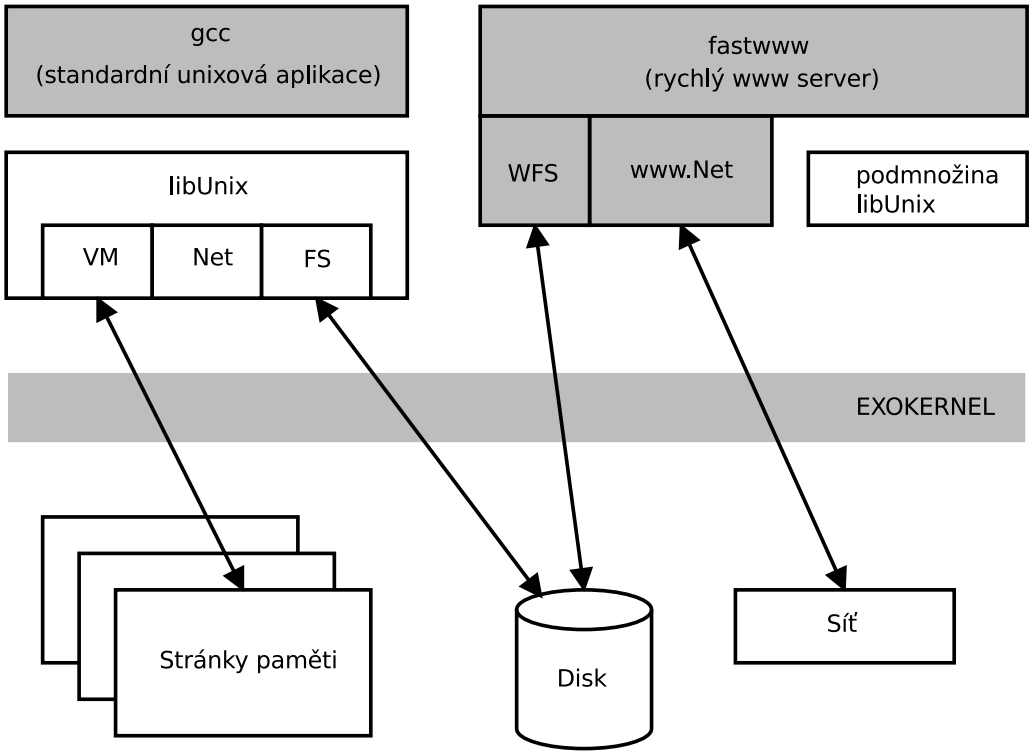
Jedna možnost je kompletně odstranit jádro a pracovat v aplikaci přímo s hardwarem, ale pak je celý stroj vyhrazený pouze této aplikaci (a zároveň je tato aplikace vytvořená na míru pouze tomuto stroji).

Koncept exojádra je kompromis: jádro zpřístupní základní fyzické prostředky stroje (např. bloky na disku, paměťové stránky a čas procesoru) více aplikacím a nechá každý program rozhodnout, co s těmito prostředky bude dělat.[18]

Program pak může volat podpůrnou knihovnu, která implementuje abstrakci, kterou potřebuje, nebo může implementovat svoji vlastní.

Takto může například nad jedním exojádrem běžet „obyčejný“ operační systém, a vedle něj speciálně uzpůsobená aplikace, která přistupuje přímo k disku, paměti, procesoru a síťové kartě (např. velmi rychlý webový server). Tento princip zobrazuje obr. 1.4.

Název exokernel tedy znamená podle předpony řeckého původu exo- (vnější) možnost obejít některé tradiční vrstvy abstrakce.



Obr. 1.4: Architektura exokernelu

1.1.5 Nanokernel

Jde o velmi malé mikrojádro. Neexistuje jasné kritérium, od kdy už jádro můžeme považovat za nanojádro, tudíž častokrát se o stejném jádře může někdy hovořit jako o mikrojádru, jindy o nanojádru.

Většina nanojader používá systém pro předávání zpráv pro komunikaci s ostatními komponentami, aby byla architektura systému co nejvíc nezávislá. Je to víceméně odlehčené jádro bez jakéhokoliv druhu abstrakce hardwaru, takže vyžaduje ovladač zařízení od jakéhokoliv základního zdroje v architektuře.

1.1.6 Modulární jádro

Modulární jádro dokáže přímo do prostoru jádra za běhu systému přidat nový programový kód, může jít o ovladač od zařízení, případně podpora jiné funkce. Modul je většinou možné také za běhu systému odebrat.[19, kap. 1.2]

Pokud o nějakém jádře tvrdíme, že je modulární, máme na mysli, že má tuto schopnost, ovšem to neznamená, že jde o pevně danou samostatnou kategorii. Můžeme se tedy setkat s monolitickým jádrem, které dokáže pracovat s moduly, ale třeba i s hybridním jádrem, které tuto funkčnost má taktéž.

Výhodou operačních systému s podporou modulů je, že k dispozici může být velké množství modulů, které podporují rozličný hardware, ale do paměti se načtou pouze ty, které jsou zrovna potřeba. Moduly usnadňují vývoj, jelikož pro jejich vyzkoušení není třeba kompilovat celé jádro a restartovat systém, stačí přeložit samotný modul.

Nevýhodou je snížená stabilita. Potenciálně nekvalitní kód, který se do jádra dostane při vývoji a testování nového modulu může nepříznivě ovlivnit chod celého jádra.

Jistou nevýhodou je čas, který je potřeba pro vložení vybraných modulů do jádra při startu systému, ovšem tento problém v praxi není tolik významný, protože po zavedení modulu už další zpomalení nenastává.[20]

Nejefektivnější přístup je tedy takový, že jádro obsahuje pouze nezbytně nutné součásti, které umožní načíst zbytek modulů, které se vyberou podle toho, který hardware je zrovna k dispozici. K tomu jsou potřeba například ovladače od řadiče pevných disků, práce se souborovým systémem, na kterém jsou moduly uloženy. Spuštění jádra může také pomoci zavaděč, který zvládne ze systému souborů alespoň číst, spustí jádro a moduly překopíruje do paměti RAM, ze kterých se vyberou pouze ty vhodné, které budou zavedeny do prostoru jádra.

1.2 Jádro operačního systému Linux

Jádro operačního systému Linux v dělění jader na monolitické jádro, a mikrojádro patří do skupiny monolitických jader. Pokud se podíváme na popis ostatních podruhů, nalzeneme prvky i z ostatních konceptů. Například existuje ovladač FUSE (Filesystem in User Space), který umožňuje snadno implementovat podporu souborových systémů. Samotný ovladač FUSE je přímo v jádře, komponenty dalších autorů, které implementují samotný souborový systém, jsou v uživatelském prostoru, viz obr. 1.5. Vidíme tedy podobnost s mikrojádry.

Linux také obsahuje systém modulů jádra, kterými se budeme podrobně zabývat v prvním počítačovém cvičení v kap. 2. Tudíž z tohoto pohledu jde o modulární jádro. Před kompilací jádra lze podporu modulů zakázat (pak vznikne čisté monolitické jádro, v jednom souboru), nebo povolit. U většiny dalších součástí lze následně vybrat, jestli budou pevnou součástí obrazu jádra, nebo jestli budou v samostatném souboru jako modul, pokud byla podpora modulů povolena. [19, kap 1.1]

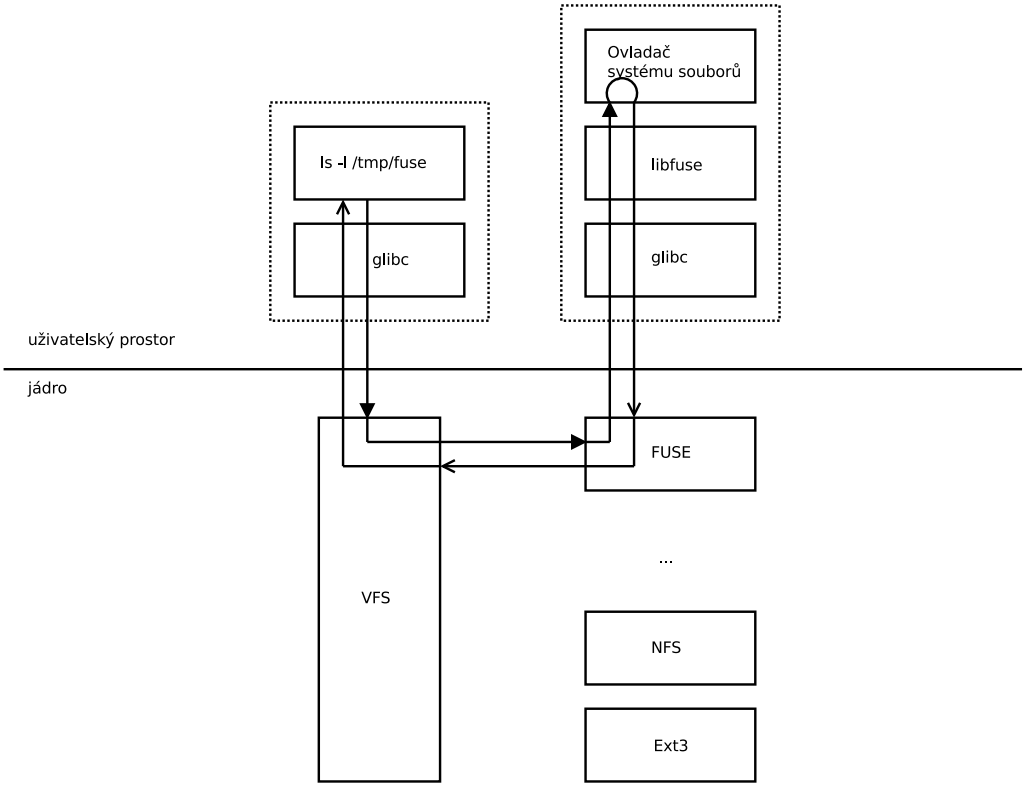
Moduly Linuxového jádra mají příponu `.ko` a nejsou přenosné do jiné verze jádra, jelikož vývojaři neudržují mezi verzemi jader zaručené API[1], nebo ABI[2]. Proto každý

[1] Application Programming Interface
[2] Application Binary Interface

modul s sebou nese informaci, pro které jádro byl přeložen. Pokud se pokusíme modul vložit do jiného jádra, bude odmítnut.

Pro dané jádro je možné přeložit modul, který nebyl původně součástí stromu zdrojového kódu jádra. Cizí modul se musí kompilovat proti zdrojovému kódu jádra, nebo alespoň jeho hlavičkám. Vzniklý binární modul je opět použitelný pouze s tou verzí jádra, proti které byl kompilován.

Programy, které slouží pro práci s moduly, jsou `insmod` (vložení souboru s modulem do jádra), `rmmod` (odebrání modulu) a `modinfo`. Poslední program je `modprobe`, který v případě nutnosti načte i moduly, na kterých aktuálně přidávaný modul závisí, jejichž závislosti se udržují v souboru `modules.dep` a seznam lze ručně zaktualizovat příkazem `depmod`.



Obr. 1.5: Přístup k souboru přes ovladač FUSE: Program `ls` chce získat obsah adresáře /tmp/fuse, který je připojen ve virtuálním systému souborů (VFS) ovladačem FUSE. Ovladač pak komunikuje s procesem v uživatelském prostoru a výsledek vrací při návratu ze systémového volání

1.3 Zavádění operačního systému

1.3.1 Tradiční způsob z pevného disku

Abychom mohli vysvětlit, jak funguje zavádění po síti, řekneme si zjednodušeně, jak se spouští operační systém, který je nainstalován přímo v počítači.

Nejprve se aktivuje BIOS, který je uložený přímo na základní desce počítače a aktivuje nejdůležitější části. Zjistí, jaký typ procesru se v počítači nachází, kolik je dostupné paměti RAM, vyhledá pevné disky, další zavaděč zpravidla vyhledá na primárním hlavním (primary master) pevném disku. Načte jeho první sektor, takzvaný MBR[3], a jeho obsah spustí. Další kroky se mohou poměrně lišit v závislosti na tom, jaký operační systém je použitý, obvykle zavaděč v MBR rozkóduje tabulku oddílu na disku (partiton table), najde oddíl, který je označený příznakem boot a v něm se pokusí najít své další případné součásti, nebo přímo jádro operačního systému. Odkaz na „další část“ je obvykle uložený v prvním bloku systému souborů, přímo číslem sektoru, takže zavaděč nemusí nutně umět pracovat se souborovým systémem.

V této fázi je zavaděč plně závislý na BIOSu na základní desce, zejména pokud jde o práci s pevným diskem a čtení jeho sektorů, protože do MBR se vejde pouze 512 bajtů, není možné, aby první zavaděč měl vlastní implementaci ovladačů pevného disku a systému souborů. BIOS práci se systémy souborů neposkytuje, proto zavaděč hledá další část odkazem přímo na sektor disku.

Po nalezení své součásti, která je uložena už na oddílu disku, vstupuje zavaděč do další fáze. Již rozpozná jednotlivé systémy souborů, vyhledá soubor s jádrem operačního systému, případně další součást (např. ramdisk), překopíruje je do operační paměti a jádro spustí.

Jádro se aktivuje a spustí se samotný operační systém, který zjistí, jaký hardware je k dispozici. V případě, že jádro bylo spuštěno i s ramdiskem, přečte si seznam modulů, které jsou v něm uloženy a dle potřeby je aktivuje.

V této části má jádro k dispozici ovladače od veškerých úložných zařízení a BIOS pro práci s diskem již nepoužívá.

Následně se aktivují další ovladače přímo v systému souborů, pokud jsou potřeba, spustí se první proces, uživatelské rozhraní atd.

1.3.2 Start systému přes síť

Při startu po síti nastává první rozdíl už v chování BIOSu. Ten kromě pevných disků prohledá síťové karty a místo vyhledání programu v MBR pevného disku aktivuje

[3]Master Boot Record - hlavní zavaděcí záznam

program, který je uložený v paměti síťové karty. Výrobci síťových karet tento program tvoří podle specifikace PXE, která definuje, jakým způsobem BIOS a síťová karta spolu komunikují, také klade požadavky na síťové protokoly, které má PXE klient podporovat. Mezi ty patří protokol DHCP (dynamic host configuration protocol) a TFTP (trivial file transfer protocol) eventuelně MTFTP (multicast trivial file transfer protocol). Za povšimnutí stojí, že PXE klient nepodporuje protokol DNS, ani TCP.[4][21]

PXE klient v síťové kartě přes protokol DHCP zjistí nastavení sítě a pomocí protokolu TFTP stáhne ze severu program podle pokynu DHCP serveru a spustí jej. Může to již být přímo jádro nějakého operačního systému, nebo pomocný program, který zobrazí nabídku a umožní uživateli zvolit nějakou variantu, jak se bude ve spouštění pokračovat.

V případě, že jde o jednoduchý pomocný program, pak platí, že ten se pro přenos dalších potřebných souborů pro start operačního systému spoléhá výhradně na implementaci přenosu dat v PXE klientovi v síťové kartě. Stejně tak, jako BIOS při startu počítače z pevného disku je zodpovědný za načítání bloků pevného disku na požadavek zavaděče v MBR, tak síťový zavaděč může vyžádat stažení dalších souborů pomocí PXE klienta v síťové kartě.

Po volbě uživatelem, nebo časové prodlevě následuje stažení jádra (případně i ramdisku) a spouštění pak pokračuje. Pokud se celý operační systém nevejde do ramdisku, musí být pomocí nějakého vyššího protokolu (HTTP, NFS, SAMBA...) schopen najít zbytek souborů na síťovém serveru. Tento krok už ale nemá s PXE specifikací nic společného.

Jak nakonfigurovat DHCP a TFTP servery v operačním systému Linux pro obsluhu PXE klientů demonstruje počítačové cvičení v kapitole 3. To zahrnuje soubory, které TFTP server poskytuje - zavaděč, nabídka pro uživatele i nasazení několika operačních systémů bootovatelných po síti.

1.3.3 Možnosti instalace operačního systému

Mezi určitě tradiční způsob, jak na počítač nainstalovat požadovaný operační systém, patří instalace z CD, případně DVD. (Ať už jde o zakoupený nosič, nebo vyrobený svépomocí z volně stažitelného obrazu na internetu). Tradiční instalační médium má nevýhodu v tom, že nainstalovaný systém je nanejvýš tak aktuální, jako použité médium.

[4]Existuje projekt gPXE, dříve etherboot, který si klade za cíl vyrobit PXE klient, který má podporu vyšších protokolů včetně HTTP. Tento zavaděč lze uložit do paměti na síťové kartě a tím původní zavaděč od výrobce přepsat. Pro méně dobrodružné uživatele lze gPXE spustit po síti a využít jako mezičlánek pro spuštění operačního systému.

Například distribuce operačního systému Debian dále nabízí obraz speciálního CD, které je velmi malé, obsahuje pouze jádro a pár nezbytně nutných balíčků, během instalace se veškeré další součásti stahují z internetu. Nepřekonatelnou výhodou je, že nainstalovaný operační systém je vždy aktuální, bez ohledu na to, jak starý obraz CD jsme použili.[5] To na druhou stranu přináší jinou komplikaci - pokud budeme instalovat systém podruhé na počítač ve stejné místnosti, veškeré balíčky, které byly už jednou z internetu staženy, budou staženy podruhé.

CD s instalací, která stahuje data po internetu, inspirovala distributory k tomu, aby nabídli stejnou možnost i na klíčenky USB. Například pokud si správce serverů vytvoří takovou USB klíčenku, může jí i třeba několik měsícu používat k instalaci serverů (předpokládejme, že čistá instalace serveru nepotřebuje tolik balíčků a servery není třeba instalovat nárazově, ale průběžně). Určitě je pohodlnější s sebou nosit USB flash disk než CD, které je navíc náchylnější na mechanické poškození.

V případě počítačových učeben se jako nejideálnější řešení nabízí spouštění počítače přes síť. Jeden z „operačních systémů“, které po síti můžeme spustit, může být vlastně instalátor, který zajistí instalaci na lokální disk. Přitom se nejaktuálnější balíčky mohou stahovat z lokálního zrcadla na místní síti. Tímto způsobem jsou data stažena z internetu jen jednou, ale můžeme instalaci spustit i na více počítačích bez nutnosti mít více fyzických instalačních médií. Tuto myšlenku se snaží nepřímo studentům naznačit úkol ve cvičení v kap. 2, kde studenti mimo jiné zprovozní síťový instalátor linuxové distribuce Debian.

1.4 Komponenty pro start počítače po síti

Většina součástí už byla vyjmenovaná, v této části je upřesníme a uvedeme konkrétní příklady.

1.4.1 DHCP server

Jak již bylo řečeno, jde o zkratku Dynamic Host Configuration Protocol. Jde o protokol, který slouží pro definování síťové konfigurace klientských počítačů, které jsou nastaveny na automatickou konfiguraci (mají DHCP klient). Z pohledu spouštění počítačů po síti je DHCP server nepostradatelný prvek, protože klientský počítač potřebuje znát správnou síťovou konfiguraci, aby mohl vůbec zahájit jakýkoliv přenos.

Bavíme se výhradně o implementaci DHCP pro protokol IP verze 4. V současné době protokol verze 6 neobsahuje žádný standard ekvivalentní PXE[22]. Nicméně

[5] Přesto, pokud máme opravdu staré CD, mohou nastat potíže, pokud jádro na CD nebude mít ovladače od novějšího řadiče pevného disku apod.

jelikož protokoly verze 4 i 6 mohou fungovat vedle sebe, jakmile je operační systém spuštěný, může IPv6 použít, pokud jej podporuje a je podporován v síti.

DHCP protokol využívá UDP pakety na portu 67 na straně serveru, a na portu 68 na straně klienta. Jde o stavový protokol, klienti musejí o konfiguraci žádat znovu v intervalu stanoveném DHCP serverem (tzv. lease time).

Mezi konkrétní implementace DHCP serverů v operačním systému Linux patří zejména `dhcpd`, což je implementace od Internet Systems Consortium, dále mnohem jednodušší verzí je `dnsmasq`.

1.4.2 TFTP server

Trivial File Transfer Protocol je „Jednoduchý protokol pro přenos souborů“. Slouží pro přenos souborů, jeho jednoduchost spočívá v tom, že nezná uživatelská jména ani hesla, neumí zjistit seznam dostupných souborů, ani přecházet mezi adresáři.

Pro přenos se používá UDP protokol na portu 69. Soubory se přenáší po fragmentech velkých maximálně 512 bajtů. Protokol TFTP má vlastní mechanismus opakovaného přenosu ztracených úseků, jelikož UDP protokol je nespolehlivý.

Mezi konkrétní TFTP servery patří například `tftpd`, `atftpd`, ale také již zmíněný `dnsmasq`.

1.4.3 Zavaděč

Zde jde o obsah, který je poskytovaný samotným TFTP serverem a o kterém informuje DHCP server. Je to první spuštěný program, který není přítomný v počítači. Zavaděč může požádat o stažení dalších souborů. V posledním kroku to určitě bude jádro nějakého operačního systému. V mezikroku může získat nějaký konfigurační soubor s informacemi o nabídce, která se zobrazí uživateli apod.

Konkrétní implementací tohoto zavaděče je `pxelinux`, který umí interpretovat konfiguraci pro zobrazení uživatelské nabídky stejným způsobem jako ostatní komponeny projektu `syslinux`, mezi které sám patří.

2 CVIČENÍ MODUL JÁDRA OS LINUX

2.1 Teoretický úvod

Co je přesně modul jádra? Moduly jsou části kódu, které mohou být vloženy a odebrány z jádra na požádání. Rozšiřují funkčnost jádra bez nutnosti restartovat systém. Například jedním druhem modulů je ovladač zařízení, který umožňuje jádru použít hadrware připojený k systému. Bez modulů bychom museli kompilovat monolitická jádra a novou funkčnost přidávat přímo do binárního souboru s jádrem. Kromě většího jádra toto přináší tu nevýhodu, že bychom museli znovu přeložit celé jádro a restartovat systém vždy, kdybychom chtěli novou funkci.

V tomto cvičení se budeme zabývat vytvořením a kompilací modulu do jádra. Ukážeme si:

- povinné součásti zdrojového kódu,
- jak se uvádí použitá licence,
- ukážeme si makra, která uvolňují nepoužitou paměť,
- jak napsat ovladač, který vytvoří virtuální soubor v adresáři `/proc`,
- dále si ukážeme, jak modulu předat parametry z příkazové řádky.

Před samotnou kompilací je nutné mít nainstalované hlavičky k aktuálnímu jádru, program `make` a kompilátor `gcc`. V systému CentOS vše zařídí balíky `make`, `kernel-devel` a `kernel-headers`. V distribuci Debian a Ubuntu bude místo balíčku `kernel-devel` potřeba `build-essential` a hlavičky používaného jádra jsou zahrnuty v balíčku `linux-headers-$(uname -r)`[1].

Potřebné balíčky jsou ve školním systému již nainstalované, uvádíme je pro úplnost, pokud byste si postup chtěli vyzkoušet doma.

2.2 Vytvoření prvního jednoduchého modulu

Nejjednodušší modul musí obsahovat tyto části:

- odkaz na hlavičkový soubor `module.h`,
- inicializační funkci `int init_module()`, která vrací 0, pokud nedošlo k chybě,
- ukončovací funkci `void cleanup_module()`.

Aby modul vůbec něco dělal, použijeme systémové volání `printk()`, které funguje obdobně jako standardní funkce `printf()` z jazyka C. Rozdíl spočívá v tom,

[1] text `$(uname -r)` nahrazuje číslo verze jádra, například `linux-headers-2.6.30`

že text nevypisuje na obrazovku, ale do systémového logu, který zobrazíme pomocí `dmesg`. Pro zobrazenou hlášku můžeme uvést i prioritu, která je definována v hlavičkovém souboru `/lib/modules/*/source/include/linux/kernel.h` takto:

```
#define KERN_EMERG      "<0>"   /* system is unusable                   */
#define KERN_ALERT      "<1>"   /* action must be taken immediately     */
#define KERN_CRIT       "<2>"   /* critical conditions                  */
#define KERN_ERR        "<3>"   /* error conditions                     */
#define KERN_WARNING    "<4>"   /* warning conditions                   */
#define KERN_NOTICE     "<5>"   /* normal but significant condition     */
#define KERN_INFO       "<6>"   /* informational                        */
#define KERN_DEBUG      "<7>"   /* debug-level messages                 */
```

Buď prioritu neuvedeme vůbec, nebo ji můžeme výslovně změnit.
Funkce `printk()` není určena hlavně pro komunikaci s uživatelem, ale pro předávání informací o pochodech jádra. Pokud například zapíšeme zprávu s prioritou `KERN_DEBUG`, uživatel se ji nedozví, ale může se k ní dostat vývojař, který bude hledat v našem kódu chyby a nastaví si podrobnější výpisy. Je vhodnější použít konstanty `KERN_*`, než psát prioritu konkrétními čísly, ale ukážeme si oba způsoby.

Dále do našeho příkladu také uvedeme licenci GPL[2]. Údaj není povinný. Jádro rozpozná následující licence (jsou definované v souboru `/lib/modules/*/source/include/linux/module.h`): „GPL“, „GPL v2“, „GPL and additional rights“, „Dual BSD/GPL“, „Dual MIT/GPL“, „Dual MPL/GPL“, „Proprietary“.

Nemusíme se příliš zabývat rozdíly mezi licencemi. Důležité je, jestli je licence svobodná, nebo proprietární. Proprietární licence „poskrvní“ jádro a v logu uvidíte hlášku „kernel tainted“ a v souboru `/proc/sys/kernel/tainted` bude nenulová hodnota. Tento mechanizmus umožňuje uživateli zkontrolovat, jestli má výhradně svobodný software v jádře a komunitě umožňuje nezabývat se řešením problémů jádra se součástmi, které nejsou svobodné. *Pokud licence není uvedena v modulu vůbec, je považován za nesvobodný!* Pokud dojde k poskvrnění jádra, nestačí modul odebrat. Je nutné jádro spustit znovu (to znamená restartovat počítač).

Otevřete terminál a vytvořte si svůj adresář, ve kterém budeme pracovat, a přepněte se do něj.

```
mkdir modul
cd modul
```

Vytvoříme soubor `hello-1.c`

```
#include <linux/module.h>   /* musi obsahovat kazdy modul */
#include <linux/kernel.h>   /* je v nem definovan KERN_ALERT */

MODULE_LICENSE("GPL");

int init_module(void)
{
```

[2]General Public License - „Obecná veřejná licence“

```
    printk("<1>Hello world.\n"); // priorita uvedena rucne

    // nenulova hodnota znamena chybu
    return 0;
}


void cleanup_module(void)
{
  printk(KERN_ALERT "Goodbye world.\n");
}
```

2.3 Kompilace, vložení modulu do jádra, odstranění

2.3.1 Vytvoření souboru Makefile

Nyní musíme modul zkompilovat. Vytvoříme vedle souboru `hello-1.c` také soubor `Makefile` (pozor na velké písmeno M). A vepíšeme do něj následující:

```
obj-m := hello-1.o

KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)

default:
	$(MAKE) -C $(KDIR) M=$(PWD) modules

clean:
	$(MAKE) -C $(KDIR) M=$(PWD) clean
```

Pozor – na místě vyznačených mezer musí být znak TAB, jinak překlad selže.

Trochu blíže vysvětlíme, k čemu jednotlivé příkazy slouží. Nejprve jsme definovali, které objekty chceme vytvořit jako modul příkazem `obj-m`. Další přípustné varianty jsou `obj-y` a `obj-n`. Možnosti y-n-m by vám mohly připomenout volby konfigurace jednotlivých součástí jádra - jestli budou pevnou součástí jádra (y), nebudou v jádře vůbec (n), anebo jako modul (m). Objekty `obj-n` bude `make` ignorovat, `obj-y` bude kompilovat jako objekty, které by se ve finálním kroku překladu jádra staly součástí binárního souboru `vmlinuz`, `obj-m` kompiluje jako moduly s příponou `.ko`.

V našem `Makefile` dále definujeme dvě proměnné – `KDIR`, která odkazuje na adresář, kde obvykle bývají nainstalované hlavičky od aktuálně spuštěného jádra (výraz v závorce bude nahrazen číslem verze spuštého jádra), dále proměnnou `PWD`, což je aktuální adresář. Následuje definice výchozího (*default*) cíle (příkaz, který se má vykonat, pokud voláme `make` bez parametrů). Make, který spustíme my, spustí

tedy další instanci příkazu `make` s parametrem `-C` a cestou k hlavičkám od jádra, s parametrem `-M` a aktuálním adresářem a cílem `modules`.

Další cíl, který jsme definovali, je `make clean`, který funguje obdobně. Opět zavolá další instanci programu `make` a přidá parametry s cestou k hlavičkám a aktuálnímu adresáři a přidá akci `clean` (vymazat soubory, které vznikly při překladu).

2.3.2 Kompilace zdrojového kódu

Nyní spustíme příkaz `make` a zobrazí se něco podobného

```
make -C /lib/modules/2.6.18-92.1.22.el5/build M=/home/student_bsos/modul modules
make[1]: Entering directory '/usr/src/kernels/2.6.18-92.1.22.el5-i686'
  CC [M]  /home/student_bsos/modul/hello-1.o
  Building modules, stage 2.
  MODPOST
  CC      /home/student_bsos/modul/hello-1.mod.o
  LD [M]  /home/student_bsos/modul/hello-1.ko
make[1]: Leaving directory '/usr/src/kernels/2.6.18-92.1.22.el5-i686'
```

Na prvním řádku vidíme, že byl spuštěn příkaz `make` s doplněnými parametry, řádky začínající textem `CC` znamenají spuštění překladače (v našem případě `gcc`), `LD` spouští linker, zde `ld`. Hodnoty v proměnných `CC` a `LD` lze změnit v `Makefile`, nebo na příkazové řádce. Pro zajímavost konkrétní příkaz, který byl použit pro překlad, nebo linkování lze najít v souborech `.hello-1.*.cmd` na začátku.

„Uklidit“ můžeme příkazem `make clean`, který smaže všechny soubory vytvořené při kompilaci.[3]

Po spuštění příkazu `make` se vytvořilo několik souborů. Nás zejména zajímá `hello-1.ko`, což je náš modul. Prozkoumáme ho příkazem:

```
/sbin/modinfo hello-1.ko
```

A dostaneme výstup:

```
filename:       hello-1.ko
license:        GPL
srcversion:     DF8DE965DEFED469EB1E13B
depends:
vermagic:       2.6.18-92.1.22.el5 SMP mod_unload 686 REGPARM 4KSTACKS gcc-4.1
```

Vidíme jeho licenci a verzi jádra, pro který byl modul zkompilován.

Nyní ho do jádra vložíme.

```
sudo /sbin/insmod hello-1.ko
```

poté se podíváme do logu

```
dmesg | tail
```

[3] ve školním systému po sobě z nějakého důvodu `make` zanechá prázdný soubor `Module.symvers`.

a na konci uvidíme

```
Hello world.
```

dále se podíváme, že je modul stále v jádře

```
/sbin/lsmod | head
```

na začátku uvidíme

```
Module                  Size  Used by
hello_1                  908  0
```

Všimněte si, že `lsmod` vypisuje v názvu modulu podtržítko a ne pomlčku. Pomlčka a podtržítko jsou považovány za stejný znak. Jde údajně o lepší pohodlí uživatele, protože výpis `lsmod` by tak měl být přehlednější. Příkazy `rmmod` a modprobe, které hned vysvětlíme, také považují pomlčku a podtržítko za stejný znak. Pozor, u `insmod` to tak není, tomu dáváme jako parametr přesný název souboru.

Nyní modul z jádra vyndáme

```
sudo /sbin/rmmod hello_1
```

a přesvědčíme se příkazem `lsmod`, že byl modul odstraněn, podíváme se do logu a uvidíme, že se objevila hláška

```
Goodbye world.
```

Poznámka: Proč jednou píšeme příponu `.ko` a podruhé ne?
Příkaz `insmod` vyžaduje jako parametr vždy název souboru, a ne název modulu.

Příkaz `rmmod` vždy prohledává seznam modulů, které jsou aktuálně nahrané přímo v jádře, akceptuje jak název bez přípony, tak s příponou `.ko`. My ji nepíšeme jednoduše z toho důvodu, že je to kratší.

Alternativou k `insmod` je příkaz `modprobe`, který příponu `.ko` neakceptuje, dokáže automaticky načíst i moduly, na kterých je zaváděný modul závislý a nemohl by jinak fungovat. Příkaz `modprobe` hledá pouze mezi moduly, které se nachází v cestě `/lib/modules/2.6.*/`.

Pokud bychom vlastní modul chtěli přidat, nakopírovali bychom ho kamkoliv do této cesty, nejlépe do adresáře `misc`, a následně spustili příkaz `depmod -a`, který projde celý adresář `/lib/modules/2.6.*/` a zapíše si seznam všech modulů, které jsou k dispozici, a jejich vzájemné závislosti. Od tohoto okamžiku bude `modprobe` fungovat s naším vlastním modulem. My to ale dělat nebudeme.

2.4 Makefile - modul z více souborů

V `Makefile` je možné definovat, že některý modul se skládá z více zdrojových souborů. Například:

```
obj-m += startstop.o
startstop-objs += start.o stop.o
#anebo startstop-y += start.o stop.o
```

Místo `:=` (přiřazení) použijeme `+=` (přidání), aby se předchozí hodnota v proměnné `obj-m` nepřepsala.

Můžeme to vyzkoušet.

```
cp hello-1.c start.c
```

ze souboru `start.c` smažeme deklaraci i tělo funkce `cleanup_module()`

```
cp hello-1.c stop.c
```

ze souboru `stop.c` smažeme funkci `init_module()`

Do našeho `Makefile` přidáme novou definici `obj-m` (původní definice pro hello-1 může zůstat) a zkompilujeme příkazem `make`. (nezapomeneme místo `:=` použít `+=`)

```
obj-m += hello-1.o
obj-m += startstop.o
startstop-objs += start.o stop.o

KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)

default:
	$(MAKE) -C $(KDIR) M=$(PWD) modules

clean:
	$(MAKE) -C $(KDIR) M=$(PWD) clean
```

Vznikne `startstop.ko` složené ze souborů `start.c` a `stop.c`

Abychom ukázali, k čemu je to konkrétně dobré, podíváme se na výňatek souboru `/fs/ext3/Makefile` ze zdrojového kódu jádra:

```
#
# Makefile for the linux ext3-filesystem routines.
#

obj-$(CONFIG_EXT3_FS) += ext3.o

ext3-y  := balloc.o bitmap.o dir.o file.o fsync.o ialloc.o inode.o \
           ioctl.o namei.o super.o symlink.o hash.o resize.o ext3_jbd.o

ext3-$(CONFIG_EXT3_FS_XATTR)     += xattr.o xattr_user.o xattr_trusted.o
ext3-$(CONFIG_EXT3_FS_POSIX_ACL) += acl.o
ext3-$(CONFIG_EXT3_FS_SECURITY)  += xattr_security.o
```

`CONFIG_EXT3_FS` je proměnná, která se nastavuje během konfigurace jádra např. příkazem `make menuconfig` a která může mít hodnoty y, m, nebo n. Všechny soubory v druhém řádku definují tvorbu objektu `ext3.o`, pokud bude potřeba. Ten se buď použije k vytvoření modulu, nebo se zakomponuje do jádra (m, respektive y).

Následně se do objektu `ext3.o` mohou přidat volitelně další 3 objekty, které jsou uvedeny dále. Hodnota příslušných proměnných (`XATTR`, `POSIX_ACL`, `SECURITY`) může nabývat hodnot y, nebo n. Tímto mechanizmem je tedy docílená volba funkcí, které bude jádro podporovat.

Otázka: V našem příkladě máme v souboru `start.c` i `stop.c` uvedenu licenci GPL, proto je výsledný modul také licencován jako GPL. Co se stane, pokud licence GPL bude pouze v jednom z nich a v druhém nebude uvedena? Co se stane pokud v jednom ze souborů bude licence Proprietary? (vyzkoušejte, zkontrolujte výsledek příkazem `/sbin/modinfo startstop.ko`)

Odpověď: Pokud je v jednom souboru licence GPL a ve druhém neuvedena, výsledný modul je licencován jako GPL. Pokud je v alespoň jednom souboru licence Proprietary, je celý modul licencován nesvobodně, bez ohledu na to, jakou licenci mají další součásti. Jakmile nesvobodný modul vložíme do jádra, celé jádro se stává nesvobodné (je „poskvrněno“ nesvobodnou licencí).

2.5 Vlastní inicializační a ukončovací funkce

Namísto povinné funkce `init_module()` a `cleanup_module()` můžeme použít svůj vlastní název. Stačí úvodní a ukončovací funkci definovat na konci souboru `.c` deklarací `module_init(nazev);` a `module_exit(nazev);`. Názvy původních funkcí `init_module()` a `cleanup_module()` pak můžeme změnit. Zkopírujeme `hello-1.c` do souboru `hello-2.c` a dále ho upravíme.

```
cp hello-1.c hello-2.c
```

přejmenujeme deklarace našich funkcí

```
int zacatek(void)
void konec(void)
```

A úplně na konci deklarujeme, že funkce `zacatek()` a `konec()` jsou inicializační a ukončovací funkcí našeho modulu.

```
module_init(zacatek);
module_exit(konec);
```

Do souboru Makefile přidáme další řádek

```
obj-m += hello-2.o
```

Modul přeložte a vyzkoušejte ho vložit a odebrat z jádra. Opět byste měli vidět v `dmesg` stejný výstup, jako u prvního modulu.

2.6 Optimalizační makra

V hlavičkovém souboru `/lib/modules/*/source/include/linux/init.h` jsou definována makra, která umožní ušetřit pamět, a je velmi dobrou zvyklostí tato makra používat. Jde o deklaraci `__init` a `__exit`, které se píší těsně před název inicializační a ukončovací funkce, dále o `__initdata` a `__exitdata`, které se píší u deklarací proměnných před znak „rovná se“.

Makro `__init`, resp. `__initdata` označuje takové funkce, respektive proměnné, které jsou použity pouze v inicializační funkci a ne jinde. Po ukončení inicializační funkce jsou části označené těmito makry odstraněny z paměti.

Makro `__exit` a `__exitdata` označuje funkce a proměnné, které jsou potřeba pouze při volání ukončovací funkce (ne dřív). U modulu jádra tato makra nezpůsobí nic, ale pokud zdrojový kód zakompilujeme do jádra jako pevnou součást, takto označené funkce (a proměnné) překladač přeskočí a do jádra je nezačlení, protože by pouze zbytečně zabíraly místo (ukončovací funkce by se stejně nikdy nepoužila). Obvykle je použití proměnných s označením `__exitdata` méně časté, protože proměnných, které jsou třeba pouze při vypínání modulu a ne dříve, mnoho nebude.

Použití ukážeme na konkrétním příkladě. Zkopírujeme `hello-2.c` do souboru `hello-3.c` a přidáme příslušný parametr do `obj-m` v `Makefile`, jak už známe, a upravíme soubor `hello-3.c` takto:
vložíme do hlavičky soubor `linux/init.h`

```
#include <linux/init.h>     /* __init a __exit makra */
```

změníme deklaraci funkcí:

```
int __init zacatek(void)
void __exit konec(void)
```

přidáme definici dvou proměnných (obě umístíme před deklaraci funkce `zacatek()`):

```
static int zacatek_data __initdata = 3;
static int konec_data   __exitdata = 5;
```

proměnné použijeme ve funkcích:

```
printk("<1>Hello world. %d\n", zacatek_data);
...
printk(KERN_ALERT "Goodbye world. %d\n", konec_data);
```

Jak jsme řekli, funkce `printk()` je velmi podobná standardní funkci `printf()` jazyka `C`. Značka `%d` bude nahrazena dekadickým číslem. Hodnoty, které se dosazují místo značek, funkce očekává jako další parametry.

Pokud ve volání funkcí `printk()` přehodíme „omylem“ proměnné `zacatek_data` a `konec_data`, `make` nás upozorní, že jsme udělali chybu a pokračuje v kompilaci. Pokud přesto takový modul zavedeme a odebereme z jádra, v logu se objeví

```
Hello world. 5
Goodbye world. 0
```

Vysvětlete proč.

Odpověď: Kompilujeme modul, tudíž v něm je nutně zahrnuta i ukončovací sekce – při startu je hodnota 5 dostupná. Ihned po zavedení je startovací sekce z paměti uvolněna, tudíž hodnota 3 už v paměti později není a vypíše se 0. Kdybychom kompilovali pevnou součást jádra (obj-y), vypíše se 0 a 0, protože ukončovací sekce bude taktéž úplně chybět.

2.7 Ukázka práce se soubory v procfs systému

Podle pokynů vyučujícího si stáhněte soubor `procfs.c`, zkompilujte ho (přidejte obj-m do `Makefile`), prozkoumejte soubor `.ko` příkazem `modinfo`, vložte modul do jádra a podívejte se na zdrojový kód.

`modinfo` nám oproti předchozím příkladům zobrazuje navíc jméno autora a popis modulu. Toho se dosáhlo následující deklarací na začátku souboru (ve zdrojovém kódu může být kdekoliv, ale vhodný je začátek, nebo konec)

```
MODULE_AUTHOR("Cervena Karkulka");
MODULE_DESCRIPTION("Modul ukazuje tvorbu souboru v /proc");
```

V adresáři `/proc/bsos` vznikly dva soubory - `pocitadlo` a `vstup_vystup`. Pokud budeme prohlížet obsah souboru `pocitadlo` (např. příkazem `cat`), bude se v něm zvyšovat číslo. Druhý soubor umožňuje do něj zapsat nějaká data pomocí přesměrování. Vyzkoušejte:

```
$ cat /proc/bsos/vstup_vystup
Zkus do me neco zapsat pomoci echo "..." > soubor ...
$ echo "text" > /proc/bsos/vstup_vystup
$ cat /proc/bsos/vstup_vystup
text
$
```

Stručně vysvětlíme, jak je toho docíleno. Nejprve bylo nutné přidat hlavičkový soubor `proc_fs.h`, ve kterém jsou definovány všechny funkce, které budeme používat. Naše inicializační funkce `procfs_init()` vytváří nejprve adresář `bsos` voláním funkce `proc_mkdir()`, dále soubory voláním funkce `create_proc_entry()`, jejíž parametry jsou *název* nově vytvářeného objektu, *oprávnění* a rodičovský *adresář*, ve kterém má být objekt vytvořen.

Název souboru je textová hodnota, oprávnění je číslo a rodičovský adresář je struktura typu `proc_dir_entry`. Funkce `proc_mkdir()` a `create_proc_entry()` rovněž vrací objekt typu `proc_dir_entry`, který si musíme uložit, pokud s ním chceme dále pracovat.

Pokud bychom chtěli objekt vytvořit v kořenovém adresáři (`/proc`), předáme hodnotu NULL.

Při odebrání modulu soubory a adresář mažeme funkcí `remove_proc_entry()`. Pokud bychom to neudělali, soubory by zůstaly vytvořené i po odstranění modulu z jádra.

Pro úplnost uvedeme definici těchto funkcí v souboru
`/lib/modules/*/source/include/linux/proc_fs.h`

```
extern struct proc_dir_entry *proc_mkdir(const char *,struct proc_dir_entry *);
extern struct proc_dir_entry *create_proc_entry(const char *name, mode_t mode,
                                                struct proc_dir_entry *parent);
extern void remove_proc_entry(const char *name, struct proc_dir_entry *parent);
```

Navrácené hodnoty typu `proc_dir_entry` si uložíme a můžeme k nim přiřadit obslužné funkce při čtení souboru (`read_proc`) a při zápisu do něj (`write_proc`). Pokud v hlavičkovém souboru budeme hledat definici struktury `proc_dir_entry`, dopracujeme se k tomu, jak mají být obslužné funkce definovány:

```
struct proc_dir_entry {
...
        read_proc_t *read_proc;
        write_proc_t *write_proc;
...
};
```

dále pak

```
typedef int (read_proc_t)(char *page, char **start, off_t off,
                          int count, int *eof, void *data);
typedef int (write_proc_t)(struct file *file, const char __user *buffer,
                           unsigned long count, void *data);
```

Naše obslužné funkce pro čtení a zápis musí přijímat stejné parametry a vracet hodnotu stejného typu, jako definice v `proc_fs.h`. Jedná se konkrétně o naše funkce `pocitadlo_read()`, která slouží k výpisu hodnoty počítadla, dále `vv_read()` a `vv_write()`, které načítají a ukládají data do paměti.

Čtecí funkce má za úkol nastavit ukazatel *start* na místo v paměti, kde je začátek dat, které se mají předat uživateli a návratová hodnota volané funkce je délka takto předaných dat. Zapisovací funkce má k dispozici data od uživatele v poli *buffer*, jejich délku v proměnné *count* a je na ní, jak s nimi naloží. Návratová hodnota zapisovací funkce je množství dat, které jsme úspěšně načetli.

Ukazatel *data* ukazuje na místo v paměti, které jsme si sami určili v inicializační funkci.

2.8 Parametry z příkazové řádky

Najděte řádek

```
static int counter1 = 1;
```

a přidejte za něj

```
module_param(counter1, int, 0);
MODULE_PARM_DESC(counter1, "startovaci hodnota pro pocitadlo");
```

Modul znovu přeložte a použijte `/sbin/modinfo`, všimněte si rozdílu. Pokud je starý modul stále zavedený v jádře, odstraňte ho a zaveďte takto

```
sudo /sbin/insmod procfs.ko counter1=100
```

a podívejte na hodnotu počítadla v souboru `/proc/bsos/pocitadlo`

Je tedy možné změnit počáteční hodnotu počítadla při zavedení modulu z příkazové řádky. To zařídila definice `module_param()`, jejíž povinné parametry jsou *název* proměnné, její *typ*, dále pak *oprávnění* pro soubor vytvořený v `/sys/module/procfs/parameters/`. 0 zde znamená, že se soubor nevytvoří. Vytvoření souboru pouze pro čtení můžeme hodnotou `S_IRUGO`, pak by bylo možné aktuální hodnotu parametru najít v souboru `.../parameters/counter1`. To se může hodit při hledání chyb. Deklarace `MODULE_PARM_DESC` je nepovinná a určuje popis, který zobrazí program `modinfo`.

2.9 Samostatný úkol

Pokuste se zorientovat v souboru `procfs.c` a proveďte následující úpravy: Vytvořte soubor `/proc/bsos/pocitadlo2`, který bude obsahovat další počítadlo, jehož parametr lze nastavit z příkazové řádky proměnnou `counter2`. Nezapomeňte ve funkci `procfs_exit()`, která se volá při odebrání modulu, soubor `pocitadlo2` vymazat.

2.10 Úklid pracoviště

Jakmile vyučující překontroluje vaši práci, dejte pracoviště do původního stavu – vymažte adresář `modul`, který jste vytvořili na začátku cvičení a z jádra odstraňte všechny vlastní moduly, které tam mohly zůstat, případně počítač restartujte.

2.11 Kontrolní otázky

Otázka 1. Lze zkompilované moduly .ko přenést na jiný počítač?

Ano i ne. Záleží především na tom, jaké jádro na druhém počítači je a s jakou konfigurací bylo přeloženo. Modul obsahuje podrobné informace o tom, pro které jádro byl přeložen, a nepůjde do jiného jádra vložit.

Otázka 2. Pokud programátor udělá v modulu chybu, jaké mohou být následky? (ohrozí to pouze funkci samotného modulu?)

Neošetřený zápis za hranice nějakého pole může změnit obsah paměti jádra, nebo i samotného kódu v jádře, který našemu modulu nepatří. Teoreticky se může stát cokoliv - od obyčejného pádu systému, až po zapsání nesprávných dat na nesprávné místo pevného disku.

Otázka 3. Jaká je výhoda modulů jádra oproti kompilaci kódu přímo do jádra?

Moduly umožnují aktivovat pouze ty části kódu, které jsou aktuálně potřeba (například není třeba mít k dispozici kód, který podporuje nějaké zařízení, které není v počítači přítomno). Dále bez modulů bychom museli kompilovat kompletní monolitická jádra a novou funkčnost přidávat přímo do binárního souboru s jádrem. Kromě většího jádra toto přináší tu nevýhodu, že bychom museli znovu přeložit celé jádro a restartovat systém vždy, kdybychom chtěli novou funkci.

Otázka 4. K čemu slouží parametry modulů jádra? (můžeme je zjistit příkazem modinfo)

Jsou to proměnné, jejichž hodnotu můžeme měnit při zavedení modulu z příkazové řádky a tím ovlivnit výchozí chování zaváděného modulu. Konkrétní možnosti záleží na tom, jaké proměnné nám autor modulu umožnil měnit. Nejčastěji používané typy jsou integer (číslo), boolean (pravdivostní hodnota) a string (textový řetězec).

3 CVIČENÍ SÍŤOVÝ SERVER

3.1 Teoretický úvod

V tomto cvičení se budeme věnovat konfiguraci serveru, který umožní spustit počítač po síti, přitom stanice nemusí mít ani pevný disk. Počítač musí bootování po síti podporovat, respektive BIOS[1] na základní desce počítače a mít vhodnou síťovou kartu. V našem pokusu využijeme jako bezdiskovou stanici virtuální počítač ve VirtualBoxu.

Na straně serveru je nutné zprovoznit DHCP[2] server, který klientské stanici sdělí informaci o zavaděči, který si stanice stáhne přes protokol TFTP[3] a spustí. (DHCP server sdělí stanici IP adresu TFTP serveru a název souboru, který má být načten).

Zavaděčů (přesněji PXE[4] boot loaderů) existuje určitě velké množství napříč operačními systémy. Zavaděč obvykle stáhne další soubory s konfigurací (ze stejného TFTP serveru), případně zobrazí nabídku uživateli a spustí samotný operační systém (jeho jádro). Operační systém pak běží buď pouze v paměti RAM, anebo aktivuje ovladače od síťové karty a zbytek systému získá přes síť, zpravidla jiným, robustnějším, protokolem než TFTP. Z toho plyne, že spouštěný operační systém by měl být na tento proces uzpůsoben.

Pro konfiguraci serveru použijeme program `dnsmasq`, který má funkčnost jak serveru DHCP, tak TFTP. Jeho konfigurace je poměrně jednoduchá a lze se s ním setkat mimo jiné v distribuci OpenWRT, kterou lze použít například v domácích WiFi přístupových bodech.

Jako PXE zavaděč použijeme `pxelinux`, který je součástí projektu `syslinux`, který umožňuje spouštět soubory různých formátů. Lze v něm i vytvářet nabídku pro uživatele.

Ukážeme si, že spouštěným „operačním systémem“ může být poměrně pestrá škála programů.

3.2 Konfigurace programu VirtualBox

Než začneme vůbec konfigurovat náš server, musíme mít propojenou bezdiskovou stanici a server, aby se navzájem viděly po síti.

Protože by bylo nepraktické mít v učebně dalších 26 počítačů, použijeme VirtualBox. V něm najdete přednastavenou stanici bez disku a systém CentOS. Tyto

[1] Basic Input Output System
[2] Dynamic Host Configuration Protocol
[3] Trivial File Transfer Protocol
[4] Preboot eXecution Environment

dva virtuální počítače jsou vzájemně propojené vnitřní sítí intnet. Virtualizovaný systém CentOS je k ní připojen rozhraním `eth1`, bezdisková stanice svým jediným rozhraním. Z hostitelského systému do této sítě není vidět.

Ověříme, že bezdisková stanice je „fyzicky“ připojena k síťovému rozhraní `eth1` následujícím způsobem. Ve virtuálním CentOSu spustíme

```
sudo /usr/sbin/tcpdump -i eth1
```

A spustíme bezdiskovou stanici. Měli bychom zachytit DHCP dotaz na rozhraní `eth1`.

```
IP 0.0.0.0.bootpc > 255.255.255.255.bootps: BOOTP/DHCP, Request from 08:00:27:dc:fa
:9b (oui Unknown), length: 548
```

3.3 Instalace zavaděče syslinux, struktura TFTP serveru

V domovském adresáři vytvoříme adresář `work`, veškeré nastavení budeme provádět v něm. Nejprve si musíme vytvořit základ pro náš TFTP server, ze kterého si spouštěný počítač bude stahovat potřebné soubory. V CentOS, který je v učebnách, se tyto soubory nachází v cestě `/usr/lib/syslinux` a dokumentace v `/usr/share/doc/syslinux-*`. Bohužel verze, která je v CentOS, je poměrně stará a obsahuje chyby. Použijeme novější verzi. V době psaní tohoto textu byla nejnovější stabilní verze `syslinux-3.83.zip`.

V archívu jsou kromě přeložených programů také zdrojové kódy, následujícím postupem z archívu získáme pouze binární soubory, přehledně uspořádané.

```
mkdir ~/work
cd ~/work
wget http://<pocitac>/<cesta_k_souboru>/syslinux-3.83.zip
mkdir tmp
cd tmp
unzip ../syslinux-3.83.zip
make INSTALLROOT=~/work/syslinux local-install
```

Poslední příkaz spouští program `make` a makro `local-install`, které je definované v `Makefile` uvnitř archívu. Toto makro není úplně obvyklé, vytvořili ho autoři syslinuxu, slouží k linkování již předkompilovaných objektů a instalaci hotových binárních souborů do systému. Je to obdoba příkazu `make install`, který pouze instaluje binární soubory. Proměnná `INSTALLROOT` je naopak bežný způsob, jak program `make` donutit, aby neinstaloval do kořenového adresáře systému, ale jinam.

Nyní máme v pracovním adresáři `work` stažený archív, adresář `tmp` a adresář `syslinux`, do kterého se nakopírovaly všechny součásti syslinuxu bez zdrojových

kódů. Binární soubory jsou v adresáři `work/syslinux/usr/share/syslinux`. Dokumentace zůstala společně se zdrojovým kódem v adresáři `work/tmp/doc`.

Přesto adresář `syslinux` obsahuje stále mnohem více věcí, než my budeme potřebovat.

Vytvoříme adresář `tftp`, který později bude kořenový adresář pro náš TFTP server a nakopírujeme do něj pouze soubory, které později budeme potřebovat.

```
cd ~/work
mkdir tftp
cd syslinux/usr/share/syslinux
cp pxelinux.0 menu.c32 vesamenu.c32 memdisk mboot.c32 ~/work/tftp
cp poweroff.c32 reboot.com ~/work/tftp
```

- `pxelinux.0` je hlavní zavaděč, který počítač spouští po získání konfigurace z DHCP serveru,
- `menu.c32` a `vesamenu.c32` jsou pomocné programy, které načtou konfigurační soubor a zobrazí nabídku,
- `memdisk` je pomocný zavaděč, který načte obraz disku do paměti a zahájí spouštění systému. Nejčastěji se používá pro zavádění disket,
- `mboot.c32` je zavaděč, který spouští jádro vyžadující standard MultiBoot.[5] Využívá jej například jádro hypervisoru XEN[6],
- `poweroff.c32` a `reboot.com` jsou programy, které vypnou, respektive restartují počítač.

Nyní máme v prostoru pro TFTP server nahraný zavaděč pxelinux z projektu syslinux. Než začneme tvořit složitější konfiguraci, vytvoříme soubor `default` v *adresáři* `pxelinux.cfg` a vložíme do něj text „prompt 1“.

```
cd ~/work/tftp
mkdir pxelinux.cfg
cd pxelinux.cfg
echo "prompt 1" > default
```

Tímto jsme pxelinux nastavili tak, aby ihned po spuštění zobrazil příkazovou řádku a čekal na reakci uživatele.

[5] MultiBoot je standard, který definuje komunikaci mezi poslední fází zavaděče a jádrem. Běžná jádra v linuxových distribucích tento standard nepoužívají

[6] XEN je virtualizační nástroj. XEN hypervisor je jádro, které je spuštěno jako první (`xen.gz`), to následně zavádí jádro upraveného hostitelského operačního systému (tzv. domain0)

3.4 Konfigurace a spuštění DHCP a TFTP serveru

V pracovním adresáři `work` vytvoříme soubor `dnsmasq.conf` s následujícím obsahem

```
interface=eth1
bind-interfaces
dhcp-leasefile=/tmp/dnsmasq.leases
dhcp-range=192.168.57.50,192.168.57.250,600
dhcp-boot=pxelinux.0
enable-tftp
tftp-root=/home/student_bsos/work/tftp
```

První dva řádky určují, na kterém rozhraní server bude odpovídat, dále volba `bind-interfaces` zakazuje serveru použít IP adresu 0.0.0.0 a vyžaduje, aby rozhraní `eth1` opravdu existovalo, `dhcp-leasefile` určuje soubor, ve kterém jsou vedeny záznamy o přidělených adresách, následuje rozsah adres, které server bude přidělovat a dobu, po které je stanice povinna požádat o prodloužení v sekundách. Konečně `dhcp-boot` je jméno zavaděče, který si má stanice stáhnout z TFTP serveru, pokud chce startovat po síti. Příkaz `enable-tftp` přikazuje programu `dnsmasq`, aby poskytoval službu TFTP a poslední je kořenový adresář pro TFTP server. Další ukázky nastavení `dnsmasq` s komentáři jsou v souboru `/etc/dnsmasq.conf`

Další parametry, které jsme neuvedli, si `dnsmasq` zjistí automaticky - masku sítě zjistí z nastavení síťového rozhraní `eth1`, dále klientovi oznámí, že výchozí brána, DNS server a TFTP odpovídá IP adrese na rozhraní `eth1`.

Server můžeme spustit. Otevřete si nový terminál a v něm spusťte

```
cd ~/work
sudo /usr/sbin/dnsmasq -C dnsmasq.conf -d
```

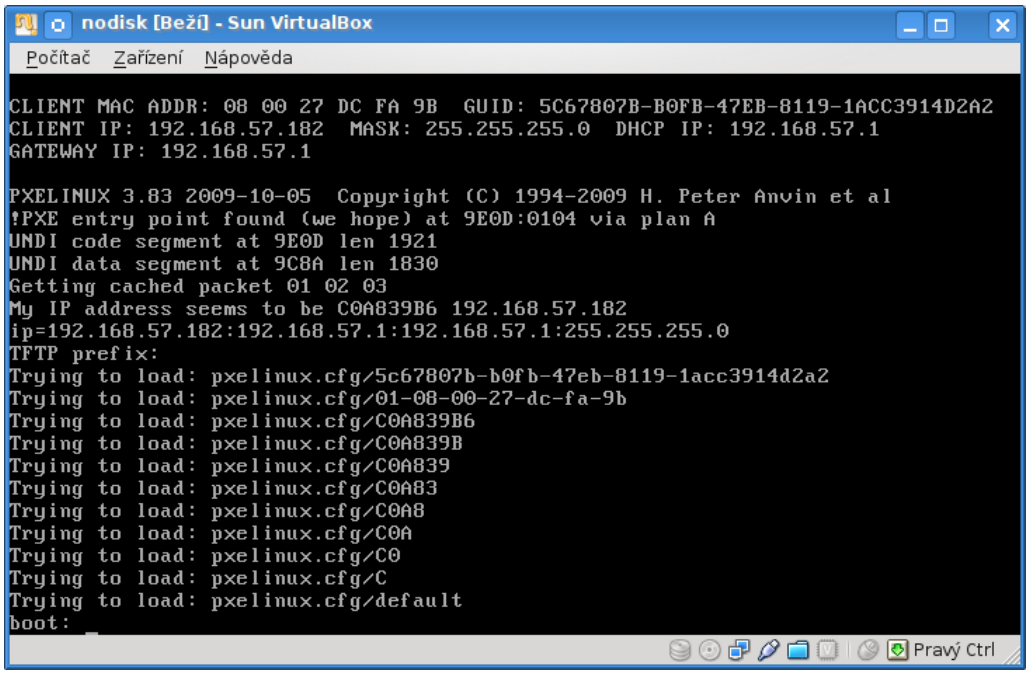
Parametr `-C` mění výchozí konfigurační soubor z `/etc/dnsmasq.conf` na ten náš, parametr `-d` zabrání programu se přepnout do režimu démon, takže nebude spuštěn na pozadí, ale činnost DHCP a TFTP serveru uvidíme přímo v terminálu. Při ostrém provozu se tento parametr samozřejmě nepoužívá.

Pokud se nebojevila žádná chybová hláška, můžeme spustit virtuální počítač bez disku se zaplým spouštením po síti. Po úspěšném spuštění počítač čeká na reakci uživatele s hláškou `boot:` (obr. 3.1)

3.5 Spuštění počítače

DHCP server po úspěšném spuštění počítače vypsal podobné hlášky:

```
dnsmasq: DHCPDISCOVER(eth1) 08:00:27:dc:fa:9b
dnsmasq: DHCPOFFER(eth1) 192.168.57.182 08:00:27:dc:fa:9b
dnsmasq: DHCPREQUEST(eth1) 192.168.57.182 08:00:27:dc:fa:9b
```



Obr. 3.1: První spuštění počítače po síti ve VirtualBoxu

```
dnsmasq: DHCPACK(eth1) 192.168.57.182 08:00:27:dc:fa:9b
dnsmasq: TFTP sent /home/student_bsos/work/tftp/pxelinux.0 to 192.168.57.182
dnsmasq: TFTP sent /home/student_bsos/work/tftp/pxelinux.cfg/default to 192.168.5...
```

Výpis z počítače je na obr. 3.1. Nejprve počítač (repsektive BIOS, případně firmware na síťové kartě) získá nastavení IP adresy z DHCP serveru. (Klient hledá vhodný server - DISCOVER, server odpovídá a nabízí volnou adresu - OFFER, klient o ni žádá - REQUEST, server potvrzuje - ACK). Klient se kromě nastavení IP adresy, masky, výchozí brány a DNS serveru také dozví adresu TFTP serveru a soubor, který si má stáhnout. Po stažení souboru `pxelinux.0` ho klient spustí.

Tento program se pak snaží najít vhodnou konfiguraci a kontaktuje stejný TFTP server. Konfiguraci hledá v adresáři `pxelinux.cfg`. Nejdříve se pokusí otevřít soubor, který názvem odpovídá identifikátoru, který vypsal BIOS virtuálního počítače jako GUID. To je identifikátor, který je jedinečný pro každý počítač, respektive základní desku. Pokud soubor neexistuje, zkusí se otevřít soubor s názvem MAC adresy síťové karty počítače. Jestliže se to nepovede, následuje vyhledávání konfigurace podle IP adresy. Všimněte si, že IP adresa je zapsaná v hexadecimální soustavě, IP adresa je postupně zobecňována odebráním posledního znaku (to znamená ignorování 4 bitů na konci adresy), až zbyde poslední znak. Pokud se ani tohle nepovedlo, hledá se soubor `default`, který jsme vytvořili my, v něm je zatím pouze uloženo, aby se zobrazila příkazová řádka a nic dalšího.

Uvedené pořadí hledání konfiguračních souborů umožňuje individualizovat chování pxelinuxu různým stanicím. Pro snadnější přepočet IP adresy z formátu IP adresy do hexadecimálního čísla umožnuje příkaz `gethostip`, který je součástí balíčku `syslinux`.

3.6 Konfigurace zavaděče syslinux

Další nastavení budeme provádět úpravou souboru `default`, jde o konfigurační soubor projektu `syslinux`. Projekt obsahuje několik druhů zavaděčů. Jednak pro instalaci do MBR oblasti disku, nebo do oddílu disku (`syslinux`), pro zaváděcí sektor k umístění na spouštěcí CD (`isolinux`), a pro spuštění přes síť (`pxelinux`). Ať už budeme systém zavádět jakýmkoliv z uvedených způsobů, `syslinux` nakonec načte soubor s konfigurací, ve kterém najde operační systémy, které lze spustit, případně jeden systém a k němu několik různých režimů. Obvyklá je také výchozí volba, případně prodleva, po které se výchozí volba provede. Všechny volby v konfiguračním souboru *nerozlišují* velká a malá písmena.
Globální parametry:

- `default` *popisek* - výchozí popisek, který bude spuštěn po stisku klávesy Enter (není-li uvedeno, platí hodnota *linux*),
- `timeout` *prodleva* - za jak dlouho se provede akce, pokud uživatel nereaguje (desetiny sekund),
- `totaltimeout` *prodleva* - za jak dlouho se provede akce, přestože uživatel reaguje (stiskl nějakou klávesu, pohybuje se v menu...),
- `ontimeout` *popisek* - pokud se po vypršení času má spustit jiná akce, než `default` (`default` pak půjde spustit jen klávesou Enter),
- `allowoptions 0` - zakáže měnit uživateli parametry, které se předají jádru (klávesou TAB), viz. obr. 3.2,
- `prompt 1` - zobrazí nabídku; jinak je nutno při startu přidržet klávesu Shift, Alt anebo mít zaplý Caps-Lock, či Scroll-Lock pro vyvolání nabídky, anebo se automaticky spustí `default`,
- `ui` *menu.c32* nebo *vesamenu.c32* - spustí další program (vesa)menu, který projde konfiguraci a zobrazí menu v textovém, nebo grafickém režimu,
- `say` *text* - vypíše text na obrazovku; nemá význam pokud je použitý příkaz `ui`.

Spustitelnou položku přidají následující příkazy:

`label` *popisek*
`kernel` *jádro*
`append` *další parametry pro start jádra (nepovinné)*

Slovo *popisek* chápeme jako klíčové slovo, které jednoznačně identifikuje spouštěný systém. Mělo by být jednoslovné, rozumně dlouhé a bez speciálních znaků, např. linux, win98, winxp apod.

3.6.1 Příkaz kernel a přípona

Klíčové slovo `kernel` rozlišuje formát souboru podle přípony (nikoliv podle skutečného obsahu, což někdy může dělat problémy). Formáty jsou rozpoznávány podle přípon následovně:

Tab. 3.1: Odhad obsahu souboru na základě přípony

bez přípony, nebo jiná	Obraz jádra Linuxu
`.0`	PXE zaváděcí program (pouze pxelinux)
`.bin`	CD boot sektor (pouze ISOLINUX)
`.bs`	Boot sektor (pouze SYSLINUX)
`.bss`	Boot sektor, superblok DOSu bude vložen (SYSLINUX)
`.c32`	COM32 obraz (32-bitový COMBOOT)
`.cbt`	COMBOOT obraz (nespustitelný z DOSu)
`.com`	COMBOOT obraz (spustitelný z DOSu)
`.img`	Obraz diskety (pouze ISOLINUX)

Pro rozpoznávání přípon dále platí ještě jedno pravidlo. Příponu nemusíme uvádět a `syslinux` se ke každému souboru, který uvedeme, pokouší nejdříve připojovat tyto přípony, až nakonec vyzkouší název tak, jak jsme jej uvedli, tj. spustí se jako obraz jádra Linuxu. To by za normálních okolností nemělo dělat problém.

Co ovšem může přivést zbytečné komplikace při neznalosti pravidla odhadování formátu souboru podle přípony, je například situace, kdy startujeme jádro `vmlinuz-redhat-9.0` - to by `syslinux` vyhodnotil jako PXE zaváděcí program a start by zhavaroval. Obdobný problém nastane, pokud chceme startovat soubor s příponou `.bin`, ale nejde o obraz CD, ale opět např. o jádro. Nastává stejný problém.

Takovou situaci můžeme řešit dvěma způsoby. Buď soubor přejmenujeme tak, aby měl odpovídající příponu, anebo namísto příkazu `kernel` napíšeme klíčové slovo,

o jaký jde formát. Starší verze `syslinuxu` prozměnu nemusí toto nové klíčové slovo znát, proto je dobré to vždy otestovat. Namísto příkazu `kernel` můžeme použít následující slova:

Tab. 3.2: Seznam klíčových slov pro určení obsahu souboru

`linux` *soubor*	Linux jádro (výchozí)
`boot` *soubor*	zaváděcí program (.bs, .bin)
`bss` *soubor*	BSS obraz (.bss)
`pxe` *soubor*	PXE zaváděcí program (.0)
`fdimage` *soubor*	Obraz diskety (.img)
`comboot` *soubor*	COMBOOT program (.com, .cbt)
`com32` *soubor*	COM32 program (.c32)
`config` *soubor*	načíst jiný konfigurační soubor

3.6.2 Spuštění jádra

Jako první vytvoříme záznam pro spuštění jádra, využijeme jádro, které je nainstalované v CentOS. Nakopírujeme ho do adresáře TFTP serveru, nejlépe pod názvem bez čísla verze, aby se nám konfigurace psala snadněji.

```
cp /boot/vmlinuz-* ~/work/tftp/vmlinuz
```

Do souboru `default` přidáme následující konfiguraci.

```
label linux
kernel vmlinuz
#append initrd=initrd.img
```

Zakomentovaný řádek ukazuje, jak bychom přidali ramdisk. Zkuste znovu spustit virtuální počítač a po spuštění stiskněte Enter, nebo napište `linux` a potvrďte Enterem. Stáhne se jádro a spustí, po nějaké chvíli se spouštění zastaví na chybové hlášce, protože nebyl nalezen kořenový adresář se systémem. Pokud bychom chtěli opravdu spustit funkční systém, je nutné dodat ramdisk, který bude obsahovat buď minimalistický systém, který bude v RAM, nebo skript, který připojí systém souborů po síti z nějakého dalšího serveru po jiném protokolu.

3.6.3 Spuštění obrazu diskety

Nejprve získáme nějakou spouštěcí disketu. Stáhneme instalační disketu pro FreeDOS. Umístění souboru `fdboot.img` vám řekne vyučující.

Jestliže se disketu pokusíte spustit příkazem `kernel`, nedočkáte se očekávaného výsledku. Dokonce selže i příkaz `fdimage`, který by podle dokumentace měl být naprosto v pořádku. Zde si musíme poradit jinak a použít program `memdisk`. Ten funguje tak, že disketu načte do horní paměti a sám sebe do dolní paměti DOSu. Nakonec přesměruje instrukci přerušení INT 13h (přístup k disku) a INT15h (přístup k paměti). Pokud nerozumíte, o co jde, tak nezoufejte. Vlastně to znamená, že `memdisk` na sebe převezme požadavky pro přístup k paměti a disku a bude je místo standardní implementace v BIOSu obsluhovat sám, a to takovým způsobem, aby startující systém z diskety nepoznal, že vlastně běží v paměti. Pevný disk je jinak dostupný normálně, pokud v počítači je.

Do souboru `default` přidáme tedy tyto řádky

```
label disketa
kernel memdisk
append initrd=fdboot.img
```

Opět konfiguraci vyzkoušejte. Na příkazové řádce syslinuxu s dotazem `boot:` napište příkaz `disketa`. Po spuštění FreeDOSu zvolte volbu „FreeDOS Safe Mode“[7], po spuštění operačního systému DOS v něm můžete vyzkoušet příkazy `dir` a `cd`.

3.6.4 Uživatelská nabídka

Už máme v nabídce více položek, můžeme si ukázat, jak vytvořit nabídku. Připište na začátek souboru default

```
ui menu.c32
# anebo
ui vesamenu32
```

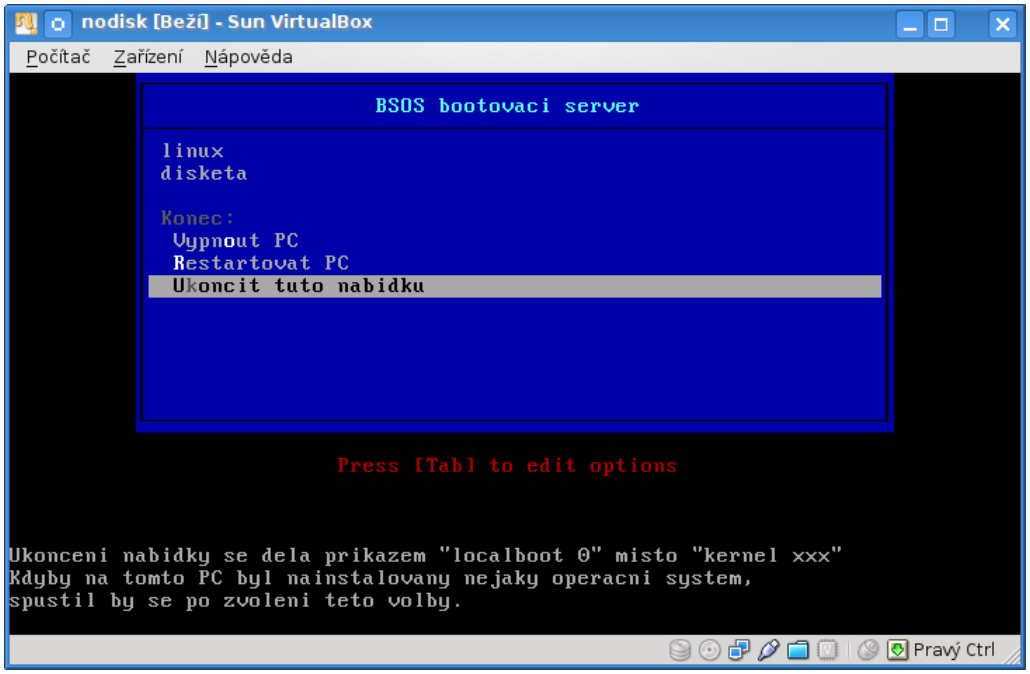
Vyberte si nabídku, která se vám víc líbí. Nyní můžeme přidat další popisky.

- Popisek v záhlaví celého menu uděláme příkazem `menu title` *text*,
- u každé položky můžeme upřesnit popisek pomocí `menu label` *text*, znak `^` vyznačí následující písmeno jako horkou klávesu,
- upřesňující text přidáme mezi řádky `text help` a `endtext`,
- prázdnou řádku přidáme pomocí `menu separator`,
- neaktivní řádek, který lze použít jako popisek, udělá příkaz `menu disable`,
- odsazení položky od okraje se dělá příkazem `menu indent` *číslo*.

Následuje ukázka pro další položky aby si uživatel mohl vybrat, jak ukončit bootovací menu (restart, vypnutí počítače, nebo pokračovat), výsledek je na obr. 3.2.

[7]Pokud vybereme instalaci FreeDOSu, zavedou se ovladače pro CD-ROM, zobrazí se zpráva, že nemáme v počítači hardisk a zobrazí se příkazová řádka. Jakmile teď zadáme příkaz `dir`, příkazová řádka se zacyklí. Lze se pouze domnívat, že jde o konflikt s ovladači FreeDOSu a memdiskem

```
menu title BSOS bootovaci server

# puvodni definice pro linux, disketa

menu separator

label
menu label Konec:
menu disable

label poweroff
menu label Vypn^out PC
menu indent 1
kernel poweroff.com

label reboot
menu label ^Restartovat PC
menu indent 1
kernel reboot.c32

label konec
menu label U^koncit tuto nabidku
menu indent 1
localboot 0
text help
Ukonceni nabidky se dela prikazem "localboot␣0" misto "kernel␣xxx"
Kdyby na tomto PC byl nainstalovany nejaky operacni system,
spustil by se po zvoleni teto volby.
endtext
```



Obr. 3.2: Uživatelská nabídka

Anglické hlášky o stisku klávesy TAB se můžeme zbavit dvěma způsoby. Buď funkci zakážeme příkazem `allowoptions 0`, nebo změníme vysvětlující text jednoduchým příkazem `menu tabmsg` *text*.

3.7 Spuštění instalátoru

Do naší nabídky můžeme přidat další nabídku, klidně i nějakého hotového projektu. Zkusme na našem serveru přidat možnost spustit přes síť instalátor jiné distribuce, např. Debian. Adresu, ze které soubor `netboot.tar.gz` stáhnete, upřesní vyučující. Instalátor stáhneme do adresáře `work` a rozbalíme do adresáře `debian`.

```
cd ~/work
wget http://<pocitac>/<cesta_k_souboru>/netboot.tar.gz
mkdir debian
cd debian
tar xzvf ../netboot.tar.gz
```

Pro lepší orientaci doporučuju spustit program `mc` a podívat se na uspořádání vzniklých souborů. Všechny důležité soubory jsou v adresáři `debian-installer`, ostatní soubory mimo tento adresář (`pxelinux.0`, `pxelinux.cfg`) jsou symbolické odkazy, které odkazují dovnitř adresáře `debian-installer`. Soubor `default` se nachází v `debian-installer/i386/pxelinux.cfg/default`.

Do adresáře `tftp` bude stačit nakopírovat adresář `debian-installer` a z našeho `default` odkázat na soubor s konfigurací instalátoru.

```
cd ~/work/debian
cp -R debian-installer ../tftp/
```

a do *našeho* souboru `default` přidáme následující (za definici linux a disketa)

```
label debian
config /debian-installer/pxelinux.cfg/default
```

a konfiguraci vyzkoušíme. Klidně vyberte v nabídce Debianu možnost Install, teprve pak se začne instalátor zavádět po síti. Než se objeví grafická nabídka s dotazem pro volbu jazyka, bude to chvíli trvat. Instalace samozřejmě nepůjde dokončit - virtuální počítač nemá pevný disk.

3.8 Samostatný úkol

Přidejte do nabídky položku memtest a invaders, u všech položek (včetně linux, disketa a debian) umožněte na volbu přejít horkou klávesou. Dále nastavte prodlevu při nečinnosti uživatele 5 sekund, po které se spustí memtest. Pokud uživatel stiskne nějakou klávesu, odpočet se zruší. Zakažte uživateli měnit parametry spouštění klávesou TAB.

3.8.1 Nápověda ke spuštění programu Memtest

Memtest stáhněte z adresy, kterou zadá vyučující, soubor `memtest86+-4.00.bin.gz`, zakomprimovaný soubor `.gz` rozbalí příkaz `gunzip`. Pozor na kombinaci přípony `.bin` a příkazu `kernel`, jak jsme uvedli v sekci 3.6.1

3.8.2 Nápověda ke spuštění hry Grub invaders

Jde o jednoduchou hru. Původní verze [12] bohužel s aktuální verzí pxelinuxu nejde spustit, proto použijte upravenou verzi z distribuce debian [13].

```
cd ~/work
wget http://<pocitac>/<cesta_k_souboru>/grub-invaders_1.0.0-10_i386.deb
mkdir invaders
cd invaders
ar xv ../grub-invaders_1.0.0-10_i386.deb
tar xzvf data.tar.gz
```

Klientské stanici je třeba předat soubor `invaders.exec`, je po rozbalení obou archívů v podadresáři `boot`. Soubor je ve formátu MultiBoot, je nutné ho zavést programem `mboot.c32` (příkaz `kernel mboot.c32`), zaváděné jádro tento program hledá jako parametr za příkazem `append`.

3.9 Další možnosti

Syslinux nabízí další volby, které jsme neukazovali, pokud máte čas, můžete je vyhledat v dokumentaci. Užitečná může být ochrana heslem (lze například zamezit přístup ke změně parametrů, nebo kompletně zablokovat celou nabídku). Je možné do konfigurace uvést přímo hesla v čistém textu (to není příliš bezpečné), anebo lze použít hash. Dále lze měnit barvy nabídek, zobrazit informativní obrazovky po stisku kláves F1-F9, případně vložit na pozadí nabídky obrázek. Inspirovat se můžete rovněž z konfigurace instalátoru debianu.

3.10 Úklid pracoviště

Jakmile vyučující zkontroluje, že se vám úkol podařil, ukončete běžící `dnsmasq` klávesou Ctrl-C, nebo použijte příkaz `kill` a smažte adresář `work`.

3.11 Kontrolní otázky

Otázka 1. Které dva protokoly je nutné použít, aby bylo možné spustit počítač po síti?

V první fázi je nutné stanici předat konfiguraci sítě, to zařídí DHCP server, na základě této konfigurace si pak stanice stáhne zavaděč, který spustí. Stahování zavaděče probíhá pomocí protkolu TFTP.

Otázka 2. Jaký je rozdíl v protokolu TFTP oproti protokolu FTP? Je TFTP spolehlivý protokol?

Protokol TFTP využívá datagramy UDP, nikoliv TCP. Nezná uživatelské jméno a heslo, neumí vypisovat seznam souborů v adresáři, ani měnit aktuální adresář. Pro přenos se používá pouze port 69 (FTP používá TCP porty 21 a 20). Co se týče spolehlivosti, samotný protokol UDP je nespolehlivý, nezaručuje, že odeslaný datagram bude doručený, neřeší jeho opakované odeslání v případě nedoručení. Tuto funkci má v sobě přímo protokol TFTP, každý datový paket musí být potvrzený, pokud není, žádá se o jeho opakované odeslání. UDP tedy spolehlivý není, TFTP ano.

Otázka 3. Může být server TFTP umístěný fyzicky jinde, než server DHCP?

Ano, protože DHCP server lze nastavit tak, aby klientské stanici předal IP adresu TFTP serveru rozdílnou od serveru DHCP.

Otázka 4. Které údaje získá z DHCP serveru stanice startující po síti kromě informace o IP adrese?

DHCP server musí vždy přidělit masku sítě, výchozí bránu a DNS server. Tyto údaje jsou nutné, aby fungovalo správně připojení do sítě (bez startu přes síť). Mezi další nepovinné údaje patří například doména, WINS server (okolní počítače), NTP server (informace o aktuálním čase) atd. Stanice, která má spouštět operační systém po síti dále potřebuje znát adresu serveru, ze kterého má stáhnout zavaděč a také jeho název.

4 ZÁVĚR

Práce obsahuje návody pro dvě počítačová cvičení. První vysvětluje tvorbu modulu pro jádro operačního systému Linux, druhé cvičení ukazuje konfiguraci síťového serveru, který umožní klientským počítačům startovat přes síť, a to bez nutnosti mít pevný disk.

Modul demonstruje použití systému souborů procfs. Soubor vytvořený v tomto systému fyzicky neexistuje, jakmile program v uživatelském prostoru se k tomuto souboru pokouší přistoupit, je volána funkce v modulu jádra, která určí jeho obsah.

Síťový server vytváří nabídku pro uživatele, který si může vybrat několik programů, které chce spustit. Může to být například jádro systému Linux, obraz spouštěcí diskety, diagnostický program Memtest, ale i jednoduchá hra Invaders.

Zde prezentovaná cvičení byla v letním semestru roku 2010 zařazena do výuky v předmětu Síťové operační systémy, osobně jsem pomáhal v příslušné počítačové učebně provést potřebné změny ve studentských počítačích, aby cvičení mohla být realizována.

Kromě větší časové náročnosti zadání samotná cvičení proběhla bez vážnějších připomínek a byla funkční.

LITERATURA

[1] SALZMAN, Peter Jay, POMERANTZ, Ori. *The Linux Kernel Module Programming Guide* [online]. Poslední úpravy 4. 4. 2003 [cit. 2009-10-20]. <http://www.faqs.org/docs/kernel/index.html>.

[2] *Howto: Build Linux Kernel Module Against Installed Kernel w/o Full Kernel Source Tree* [online]. Poslední úpravy 4. 11. 2008 [cit. 2009-10-21]. <http://www.cyberciti.biz/tips/build-linux-kernel-module-against-installed-kernel-source-tree.html>.

[3] *Linux Kernel Source Tree 2.6.30-2* [online]. Poslední aktualizace 20. 7. 2009 [cit. 2009-11-20]. <ftp://ftp.kernel.org/pub/linux/kernel/v2.6/linux-2.6.30.2.tar.bz2>.

[4] *The Linux Cross Reference* [online]. Poslední aktualizace 9. 9. 2009, verze 2.6.31 [cit. 2009-11-26]. <http://lxr.linux.no/linux>.

[5] CORBET, Jonathan, RUBINI, Alessandro, KROAH-HARTMAN Greg. *Linux Device Drivers* [online]. 3. vyd. Sebastopol: O'Reilly Media, 27. 1. 2005. 616 s. ISBN 0-596-00590-3. Dostupné z URL <http://lwn.net/Kernel/LDD3/>.

[6] *Common Problems - Syslinux wiki* [online]. Poslední úpravy 7. 10. 2009 [cit. 2009-11-19]. <http://syslinux.zytor.com/wiki/index.php/Common_Problems>.

[7] *Etherboot/gPXE Wiki* [online]. Poslední změna 22. 9. 2009 [cit. 2009-11-27]. <http://etherboot.org>.

[8] LACKORZYNSKI, Adam. *PXE enabled GRUB* [online]. Poslední aktualizace 15.9.2006 [cit. 2009-11-19]. <http://os.inf.tu-dresden.de/pipermail/l4-hackers/2006/002452.html>.

[9] *boot.kernel.org(BKO) : Booting your machine over HTTP* [online]. [cit. 2009-11-27]. <boot.kernel.org>.

[10] *The FreeDOS Project* [online]. Poslední aktualizace 23. 10. 2009 [cit. 2009-11-20]. <http://www.freedos.org/>.

[11] *Memtest86+ - Advanced Memory Diagnostic Tool* [online]. Poslední aktualizace 22. 9. 2009 [cit. 2009-11-27]. <http://www.memtest.org/>.

[12] THIELE, Erik. *Invaders* [online]. [cit. 2009-11-19]. <http://www.erikyyy.de/invaders/>.

[13] *Package grub-invaders*. [cit. 2009-11-19]. <http://packages.debian.org/grub-invaders>.

[14] NOTARAS, George. *How to extract RPM or DEB packages* [online]. Poslední úpravy 28. 1. 2008 [cit. 2009-11-20]. <http://www.g-loaded.eu/2008/01/28/how-to-extract-rpm-or-deb-packages/>.

[15] *Kernel - OSDev wiki* [online]. Poslední úpravy 26. 8. 2009 [cit. 2010-04-25]. <http://wiki.osdev.org/Kernels>.

[16] *Etymology of the Latin word monolithus* [online]. [cit. 2010-04-25]. <http://www.myetymology.com/latin/monolithus.html>.

[17] ROCH, Benjamin. *Monolithic kernel vs. Microkernel* [online]. Publikováno 28. 1. 2008 [cit. 2010-04-25]. <http://www.vmars.tuwien.ac.at/courses/akti12/journal/04ss/article_04ss_Roch.pdf>.

[18] *MIT Exokernel Operating System* [online]. Poslední aktualizace 5. 3. 1998 [cit. 2010-04-28]. <http://pdos.csail.mit.edu/exo/>.

[19] *Linux System Administration* [online]. 2005 [cit. 2009-05-02]. <http://www.nongnu.org/lpi-manuals/lpi-102/html/index.html>.

[20] *Current Trends in Operating System kernels* [online]. Poslední aktualizace 21. 7. 2003 [cit. 2009-05-02]. <http://db.glug-bom.org/lug-authors/philip/docs/os-tech.html>.

[21] *Preboot Execution Environment (PXE) Specification* [online]. Verze 2.1, vyd. 20.9.1999 [cit. 2009-05-04]. 103s. <http://download.intel.com/design/archives/wfm/downloads/pxespec.pdf>.

[22] *Does Intel offer an IPv6 version of PXE?* [online]. Poslední aktualizace 16. 11. 2009 [cit. 2009-05-05]. <http://www.intel.com/support/network/sb/cs-028553.htm>.

SEZNAM PŘÍLOH

A MODUL JÁDRA - ZDROJOVÝ KÓD

A.1 Soubor procfs.c - zadání samostatného úkolu

```
#include <linux/kernel.h>
#include <linux/module.h>

/* pouzivame proc filesystem */
#include <linux/proc_fs.h>

// zde je definovana funkce copy_from_user,
// kterou pouzivame ve funkci vv_write
#include <asm/uaccess.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Cervena_Karkulka");
MODULE_DESCRIPTION("Modul_ukazuje_tvorbu_souboru_v_/proc");

// pamet pro obsah souboru vv
#define VV_LEN 1024
static char data_vv1[VV_LEN+1]="Zkus_do_me_neco_zapsat_pomoci_echo_\"...\"_>_soubor_...\n";

// pocitadlo pristupu
static int counter1 = 1;

// funkce, ktera se spusti pri otevreni souboru s pocitadlem
int pocitadlo_read(char *page,
                   char **start,
                   off_t off,
                   int count,
                   int *eof, void *data)
{
  int len;                        // pocet znaku, ktere maji byt precteny
  static char my_buffer[80];      // static zustane v pameti i pri ukonceni teto funkce
  int *counter=(int*)data;        // datovou strukturu spojenou se souborem budeme povzaovat za int

  // pokud se jadro dotazuje na pokracovani, vracime, ze zadna dalsi data nejsou
  if (off > 0)
    return 0;

  // naplnime buffer a zjistime delku
  len = sprintf(my_buffer,
    "Rikam_ti_po_%d._jdi_pryc!\n", *counter);
  (*counter)++; // zvysime pocitadlo

  // funkci, ktera nas volala rekneme, kde jsou data k precteni
  *start = my_buffer;
  // a vratime hodnotu delky
  return len;
}

// zobrazi data pri otevreni souboru
int vv_read(char *page,
            char **start,
            off_t off,
            int count,
            int *eof, void *data)
{
  int len=0;
  char *ch=(char*)data;           // data jsou retezec znaku

  if (off > 0)
    return 0;

  while (ch[len]!=0) len++;       // vypocitame delku

  *start = ch; // data jsou na ukazateli ch

  return len;  // a jejich delka

}

// pri zapisu do souboru
int vv_write(struct file *file,
             const char *buffer,
```

```
                unsigned long count,
                void *data)
{
        int len;
        char *ch=(char*)data;

        if(count > VV_LEN)     // pokud prislo vic znaku, nez pojme pamet
                len = VV_LEN; // nacteme jen to, co se vejde
        else
                len = count;  // jinak precteme tolik, kolik prislo

        if(copy_from_user(ch, buffer, len)) // data prekopirujeme z buffer do ch
                return -EFAULT;

        ch[len] = '\0';  // a pridame ukoncovaci znak

        return len;
}

// definice souboru
struct proc_dir_entry *adresar, *soubor_poc, *soubor_vv;

// inicializace modulu
int __init procfs_init(void)
{
    adresar = proc_mkdir("bsos", NULL); // vytvorit adresar

    soubor_poc = create_proc_entry("pocitadlo",0666,adresar);
    soubor_poc->read_proc = pocitadlo_read;
    soubor_poc->data=&counter1;

    soubor_vv = create_proc_entry("vstup_vystup",0666,adresar);
    soubor_vv->read_proc = vv_read;
    soubor_vv->write_proc = vv_write;
    soubor_vv->data=&data_vv1;

    return 0;
}


// ukonceni modulu - uklid souboru
void __exit procfs_exit(void)
{
    remove_proc_entry("vstup_vystup",adresar);
    remove_proc_entry("pocitadlo",adresar);
    remove_proc_entry("bsos",NULL);
}

module_init(procfs_init);
module_exit(procfs_exit);
```

A.2 Soubor procfs.c - ukázkové řešení úkolu

```
#include <linux/kernel.h>
#include <linux/module.h>

/* pouzivame proc filesystem */
#include <linux/proc_fs.h>

// zde je definovana funkce copy_from_user,
// kterou pouzivame ve funkci vv_write
#include <asm/uaccess.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Cervena_Karkulka");
MODULE_DESCRIPTION("Modul_ukazuje_tvorbu_souboru_v_/proc");

// pamet pro obsah souboru vv
#define VV_LEN 1024
static char data_vv1[VV_LEN+1]="Zkus_do_me_neco_zapsat_pomoci_echo_\"...\"_->_soubor_...\n";

// pocitadlo pristupu
static int counter1 = 1;
static int counter2 = 1;

//promenna, typ, opravneni v /sys/module/*/parameters/*
// definovano v linux/include/linux/stat.h
// 0 pokud nebudeme exportovat
// S_IRUGO pro cteni vsem
module_param(counter1, int, 0);
MODULE_PARM_DESC(counter1, "startovaci_hodnota_pro_pocitadlo");

module_param(counter2, int, 0);
MODULE_PARM_DESC(counter2, "startovaci_hodnota_pro_pocitadlo2");

// funkce, ktera se spusti pri otevreni souboru s pocitadlem
int pocitadlo_read(char *page,
                   char **start,
                   off_t off,
                   int count,
                   int *eof, void *data)
{
  int len;                        // pocet znaku, ktere maji byt precteny
  static char my_buffer[80];      // static zustane v pameti i pri ukonceni teto funkce
  int *counter=(int*)data;        // datovou strukturu spojenou se souborem budeme povzaovat za int

  // pokud se jadro dotazuje na pokracovani, vracime, ze zadna dalsi data nejsou
  if (off > 0)
    return 0;

  // naplnime buffer a zjistime delku
  len = sprintf(my_buffer,
    "Rikam_ti_po_%d._jdi_pryc!\n", *counter);
  (*counter)++; // zvysime pocitadlo

  // funkci, ktera nas volala rekneme, kde jsou data k precteni
  *start = my_buffer;
  // a vratime hodnotu delky
  return len;
}

// zobrazi data pri otevreni souboru
int vv_read(char *page,
            char **start,
            off_t off,
            int count,
            int *eof, void *data)
{
  int len=0;
  char *ch=(char*)data;           // data jsou retezec znaku

  if (off > 0)
    return 0;

  while (ch[len]!=0) len++;       // vypocitame delku

  *start = ch; // data jsou na ukazateli ch

  return len;  // a jejich delka
```

```
}

// pri zapisu do souboru
int vv_write(struct file *file,
             const char *buffer,
             unsigned long count,
             void *data)
{
        int len;
        char *ch=(char*)data;

        if(count > VV_LEN)     // pokud prislo vic znaku, nez pojme pamet
                len = VV_LEN; // nacteme jen to, co se vejde
        else
                len = count;  // jinak precteme tolik, kolik prislo

        if(copy_from_user(ch, buffer, len)) // data prekopirujeme z buffer do ch
                return -EFAULT;

        ch[len] = '\0';  // a pridame ukoncovaci znak

        return len;
}

// definice souboru
struct proc_dir_entry *adresar, *soubor_poc, *soubor_poc2, *soubor_vv;

// inicializace modulu
int __init procfs_init(void)
{
    adresar = proc_mkdir("bsos", NULL); // vytvorit adresar

    soubor_poc = create_proc_entry("pocitadlo",0666,adresar);
    soubor_poc->read_proc = pocitadlo_read;
    soubor_poc->data=&counter1;

    soubor_poc2 = create_proc_entry("pocitadlo2",0666,adresar);
    soubor_poc2->read_proc = pocitadlo_read;
    soubor_poc2->data=&counter2;

    soubor_vv = create_proc_entry("vstup_vystup",0666,adresar);
    soubor_vv->read_proc = vv_read;
    soubor_vv->write_proc = vv_write;
    soubor_vv->data=&data_vv1;

    return 0;
}


// ukonceni modulu - uklid souboru
void __exit procfs_exit(void)
{
    remove_proc_entry("vstup_vystup",adresar);
    remove_proc_entry("pocitadlo",adresar);
    remove_proc_entry("pocitadlo2",adresar);
    remove_proc_entry("bsos",NULL);
}

module_init(procfs_init);
module_exit(procfs_exit);
```

B SÍŤOVÝ SERVER - KONFIGURACE

B.1 Soubor default - stav těsně před zadáním úkolu

```
prompt 1
ui vesamenu.c32

menu title BSOS bootovaci server

label linux
kernel vmlinuz
#append initrd=initrd.img

label disketa
kernel memdisk
append initrd=fdboot.img

label debian
config debian-installer/i386/pxelinux.cfg/default

menu separator

label
menu label Konec:
menu disable

label poweroff
menu label Vypn^out PC
menu indent 1
kernel poweroff.com

label reboot
menu label ^Restartovat PC
menu indent 1
kernel reboot.c32

label konec
menu label U^koncit tuto nabidku
menu indent 1
localboot 0
text help
Ukonceni nabidky se dela prikazem "localboot 0" misto "kernel xxx"
Kdyby na tomto PC byl nainstalovany nejaky operacni system,
spustil by se po zvoleni teto volby.
endtext
```

B.2 Soubor default - ukázkové řešení úkolu

```
prompt 1
ui vesamenu.c32
timeout 50
allowoptions 0
default memtest

menu title BSOS bootovaci server

label linux
menu label ^Linux jadro
kernel vmlinuz
#append initrd=initrd.img

label disketa
menu label ^Disketa
kernel memdisk
append initrd=fdboot.img

label debian
menu label ^Instalace Debianu
config debian-installer/i386/pxelinux.cfg/default

label memtest
menu label ^Memtest
linux memtest86+-4.00.bin
#nebo kernel memtest (bez pripony .bin)

label invaders
menu label ^Grub invaders
kernel mboot.c32
append invaders.exec

menu separator

label
menu label Konec:
menu disable

label poweroff
menu label Vypn^out PC
menu indent 1
kernel poweroff.com

label reboot
menu label ^Restartovat PC
menu indent 1
kernel reboot.c32

label konec
menu label U^koncit tuto nabidku
menu indent 1
localboot 0
text help
Ukonceni nabidky se dela prikazem "localboot 0" misto "kernel xxx"
Kdyby na tomto PC byl nainstalovany nejaky operacni system,
spustil by se po zvoleni teto volby.
endtext
```

C TESTOVÉ OTÁZKY

Vždy je správná právě jedna odpověď, je vyznačená znakem *.

C.1 Modul jádra

```
Najdi chybu: Jestliže je v souboru /proc/sys/kernel/tainted hodnota 0,
všechny komponenty jádra i zavedené moduly jsou licencovány svobodnou licencí
jádro není poskvrněné
alespoň jedna komponenta je licencovaná pod licencí GPL
*alespoň jedna komponenta používá licenci Proprietary

Najdi chybu: Jestliže je v souboru /proc/sys/kernel/tainted hodnota 0,
všechny komponenty jádra i zavedené moduly jsou licencovány svobodnou licencí
jádro není poskvrněné
alespoň jedna komponenta je licencovaná pod licencí GPL
*alespoň jedna komponenta nemá uvedenou licenci vůbec

Jádro je poskvrněno pokud:
všechny komponenty používají svobodnou licencni
všechny komponenty používají licenci GPL
všechny komponenty používají licenci GPL v2
*některá součást nemá uvedenou licenci vůbec

Jádro je poskvrněno pokud:
všechny komponenty používají svobodnou licencni
všechny komponenty používají licenci GPL
všechny komponenty používají licenci GPL v2
*některá součást používá licenci Proprietary

Dojde-li k poskvrnění jádra při vložení modulu, lze to vrátit zpět:
odebráním problematického modulu příkazem rmmod
*restartem počítače
zapsáním hodnoty 0 do /proc/sys/kernel/tainted
je nutné zkompilovat nové neposkvrněné jádro

Dojde-li k poskvrnění jádra při vložení modulu, lze to vrátit zpět:
změna je nevratná
*restartem počítače
zapsáním hodnoty 0 do /proc/sys/kernel/tainted
je nutné zkompilovat nové neposkvrněné jádro

Dojde-li k poskvrnění jádra při vložení modulu, lze to vrátit zpět:
odebráním problematického modulu příkazem rmmod
*restartem počítače
zapsáním hodnoty 0 do /proc/sys/kernel/tainted
změna je nevratná

Dojde-li k poskvrnění jádra při vložení modulu, lze to vrátit zpět:
odebráním problematického modulu příkazem rmmod
*restartem počítače
změna je nevratná
je nutné zkompilovat nové neposkvrněné jádro

Modul vznikne kompilací:
právě jednoho zdrojového souboru .c
```

*jednoho, nebo více souborů .c
kompilací jednoho souboru .mod
kompilací jednoho, nebo více souborů .mod

Moduly mají příponu:
*.ko
.c
.mod
.o

Moduly mají příponu:
*.ko
.c
.module
.o

Moduly mají příponu:
*.ko
.c
.kernel
.o

Co NEplatí o filesystému typu proc?
kernelspace kód (včetně modulů) v něm může vytvořit soubor/y
jsou v něm mimo jiné informace o stavu spuštěných procesů
se obvykle nachází v adresáři /proc
*uživatel root do něj může zapisovat

Co NEplatí o filesystému typu proc?
*kernelspace kód (včetně modulů) v něm nemůže vytvořit žádný soubor
jsou v něm mimo jiné informace o stavu spuštěných procesů
se obvykle nachází v adresáři /proc
uživatel root do něj nemůže zapisovat

Co je to modul jádra?
*Objekt, který lze vložit přímo do jádra a přidává mu nové funkce.
Je to proces, který běží s právy uživatele root.
Je to proces, který může spustit i neprivilegovaný uživatel.
Je to patch pro zdrojový kód jádra, který přidá novou funkci. Po aplikaci patche je nutné jádro znovu přeložit.

C.2 Síťový server

Zavaděč se přenáší protokolem
*TFTP
HTTP
FTP
RTP

Zavaděč se přenáší protokolem
*TFTP
SFTP
FTP
HTTP

Zavaděč se přenáší protokolem
*TFTP
HTTPS

FTP
HTTP

Zavaděč se přenáší protokolem
*TFTP
HTTPS
FTP
HTTP

Zavaděč se přenáší protokolem
*TFTP
RTP
FTP
SFTP

Najdi chybu:
*Protokol UDP je spolehlivý.
Protokol TFTP je spolehlivý.
Protokol TCP je spolehlivý.
Zavaděč se přenáší protokolem TFTP.

Najdi chybu:
Protokol UDP je nespolehlivý.
*Protokol TFTP je nespolehlivý.
Protokol TCP je spolehlivý.
Zavaděč se přenáší protokolem TFTP.

Najdi chybu:
Protokol UDP je nespolehlivý.
Protokol TFTP je spolehlivý.
*Protokol TCP je nespolehlivý.
Zavaděč se přenáší protokolem TFTP.

Najdi chybu:
Protokol UDP je nespolehlivý.
Protokol FTP je spolehlivý.
Protokol TCP je spolehlivý.
*Zavaděč se přenáší protokolem FTP.

DHCP server klientské stanici startující po síti mimo jiné přiděluje:
*IP adresu
MAC adresu
GUID
binární obraz zavaděče

DHCP server klientské stanici startující po síti mimo jiné přiděluje:
*masku sítě
MAC adresu
GUID
jádro operačního systému

DHCP server klientské stanici startující po síti mimo jiné přiděluje:
*IP adresu TFTP serveru
MAC adresu
GUID
binární obraz zavaděče

DHCP server klientské stanici startující po síti mimo jiné přiděluje:
*jméno souboru se zavaděčem

MAC adresu
GUID
jádro operačního systému

DHCP server klientské stanici startující po síti mimo jiné přiděluje:
*výchozí bránu
MAC adresu
jádro operačního systému
binární obraz zavaděče

DHCP server klientské stanici startující po síti mimo jiné přiděluje:
*výchozí bránu
MAC adresu
GUID
binární obraz zavaděče

Pro vztah mezi IP adresou DHCP serveru a TFTP serveru ve fungujícím PXE prostředí platí:
jsou vždy shodné
jsou vždy různé
*záleží na nastavení chování DHCP serveru
záleží na nastavení chování TFTP serveru