VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

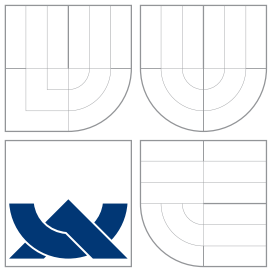
IMPLEMENTACE OPENVPN NA PLATFORMĚ WINDOWS CE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

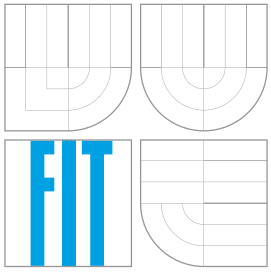
AUTOR PRÁCE
AUTHOR

Bc. OLDŘICH EŠNER

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

IMPLEMENTACE OPENVPN NA PLATFORMĚ WINDOWS CE

PORTING OPENVPN TO WINDOWS CE PLATFORM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. OLDŘICH EŠNER
AUTHOR

VEDOUCÍ PRÁCE — Ing. ONDŘEJ RYŠAVÝ, Ph.D.
SUPERVISOR

BRNO 2008

Abstrakt

Motivací pro vznik této diplomové práce, která navazuje na stejnojmenný semestrální projekt, byl převod aplikace pro tvorbu virtuálních privátních sítí OpenVPN z operačního systému Windows XP na platformu Windows CE Embedded 6.0. Práce pojednává obecně o virtuálních privátních sítích, a podrobněji o jedné z jejich implementací - OpenVPN. Uvádí základní vlastnosti operačního systému Windows CE, dále popisuje princip ovladačů zařízení v operačních systémech na bázi Windows NT, používaný Windows Driver Model, síťový model NDIS a také model ovladačů na Windows CE - Stream Interface Model. Práce pokračuje popisem komunikace v programu OpenVPN, zejména rolí virtuálních síťových adaptérů TUN/TAP. Následuje návrh převodu ovladačů adaptéru TUN/TAP s podrobným popisem omezení a nutných změn mezi oběma platformami. Výsledkem je implementovaný síťový ovladač TAP, jehož funkčnost je ověřena testovací aplikací emulující chování TUN adaptéru. Práce končí hodnocením dosažených výsledků, možnostmi pokračování v tomto tématu a vlastním přínosem celého projektu.

Klíčová slova

Virtuální privátní sítě, VPN, OpenVPN, Windows CE 6.0, Windows Embedded, Ovladače, TUN/TAP, NDIS, WDM, IRP, Asynchronní I/O, Stream interface model, NetDCU10

Abstract

The motivation for inception of this MSc. thesis which follows on from a term project of the same name was the transfer of the application for building private virtual OpenVPN networks from Windows XP operating system to Windows CE Embedded 6.0 platform. The project deals with virtual private networks in general and looks more closely at its implementation - OpenVPN. It also introduces the basic features of the Windows CE operating system. The project goes on to describe device drivers in NT-based Windows operating systems, the Windows Driver Model used, the NDIS network interface model and also the model of Windows CE drivers - the Stream Interface Model. The project continues with a description of communication in OpenVPN application and primarily the role of TUN/TAP virtual network interfaces. This is followed by a proposal for transfer of TUN/TAP adapter drivers together with a description of limitations and necessary modifications between both platforms. As a result a TAP network device driver is implemented whose function is verified by test application that emulates the behaviour of a TUN adapter. The project concludes with an evaluation of the achieved results, the possibilities for further work on this theme and with the overall contribution of this project.

Keywords

Virtual private networks, VPN, OpenVPN, Windows CE, Windows Embedded, Drivers, TUN/TAP, NDIS, WDM, IRP, Overlapped I/O, Stream interface model, NetDCU10

Citace

Oldřich Ešner: Implementace OpenVPN na platformě
Windows CE, diplomová práce, Brno, FIT VUT v Brně, 2008

Implementace OpenVPN na platformě Windows CE

Prohlášení

Prohlašuji, že jsem tento diplomový projekt vypracoval samostatně pod vedením pana Ing. Ondřeje Ryšavého, Ph.D.

.....
Oldřich Ešner
14. května 2008

Poděkování

Na tomto místě bych rád poděkoval Ing. Ondřeji Ryšavému, Ph.D. za odborné vedení diplomové práce a poskytnuté konzultace. Dále bych rád poděkoval své rodině a přátelům za jejich podporu po celou dobu studia.

Obsah

Kapitola 1

Úvod

1.1 Motivace

Firmy v dnešní době čelí rozhodování, zda podporovat nemalé množství různých prostředků pro komunikaci mezi velkým počtem podnikových sítí nacházejících se na více místech, stejně tak jako se snaží zredukovat náklady jejich komunikační infrastruktury. Zaměstnanci očekávají přístup k prostředkům jejich firemního intranetu z jakéhokoli místa na světě. Také obchodní partneři se připojují do společného extranetu pro sdílení obchodních informací, ať už pro krátkodobé projekty nebo pro dlouhodobou strategickou podporu.

Zároveň podniky zjišťují, že starší řešení k budování rozsáhlých sítí mezi hlavní podnikovou sítí a pobočkami, jakými jsou vyhrazené pronajaté linky nebo frame-relay okruhy neposkytují flexibilitu potřebnou pro rychlé vytvoření nových spojení mezi obchodními partnery nebo podporu projektových týmů. Mezitím, růst počtu vzdálených pracovníků a stále více obchodních zástupců pohybujících se mimo dosah firemní sítě spotřebovává jak prostředky, tak více peněz za servery pro vzdálený přístup, skupiny modemů včetně výdajů za telefonní poplatky.

Virtuální privátní sítě (VPN) využívající internet mají potenciál vyřešit velkou část těchto obchodních problémů týkající se síťové komunikace. VPN dovolují manažerům se připojit do vzdálené pobočky, a projektovým týmům do jejich firemní sítě, vše se zachováním nízkých nákladů. Též poskytují vzdálený přístup zaměstnancům, spolu s redukováním interních požadavků na vybavení a podporu.

Lepším řešením, než být závislý na vyhrazených pronajatých linkách nebo pevných frame relay virtuálních okruzích, jsou VPN využívající internet. Využívají otevřenou, distribuovanou infrastrukturu internetu pro zasílání dat mezi podnikovými sítěmi. Firmy používající VPN nastaví spojení k lokálním přístupovým bodům (points of presence - POP) jejich poskytovatelů připojení k internetu (internet service provider, ISP) a nechají na příslušném poskytovateli, jakým způsobem jsou data zaslána k cíli. Protože je internet veřejná síť s otevřeným přenosem většiny dat, VPN zahrnují opatření pro šifrování dat mezi VPN sítěmi, které chrání data proti odposlouchávání (důvěrnost), a před manipulací dat třetími stranami (integrita).

Navíc, VPN není omezeno podnikovou sítí a sítěmi na pobočkách. Výhodou je, že VPN může poskytnout zabezpečené připojení pro mobilní pracovníky. Ti se mohou připojit ke své podnikové VPN přes POP aktuálního ISP, což snižuje výdaje za instalaci a správu řady přístupových modemů v podnikové síti. Informace čerpány z Internatinal Engineering Consortium [1].

Druhým aspektem je stále se rozšiřující využívaní kapesních počítačů (PDA) ve firmách.

Slouží jako efektivní pracovní nástroj pro organizaci času, kontaktů, dokumentů a připojení k internetu spolu se zachováním kompaktních rozměru. Je pochopitelné, že i tací uživatelé vyžadují připojení ke své podnikové síti z kapesního počítače, zejména přes firemní VPN.

Při tvorbě VPN máme v současné době několik možností. Lze použít nějaké proprietární VPN, VPN založené na IPSec nebo některou otevřenou VPN. Proprietární řešení jsou většinou dražší a hodí se spíše pro větší nasazení, oproti tomu je IPSec levnější varianta, ovšem naráží na složitější konfiguraci. Existují také open source VPN používající SSL/TLS, jako například OpenVPN. Tento program je implementován na více platformách (Windows, Linux, Solaris, BSD, Mac OS X), ovšem chybí mezi nimi operační systém Windows CE pro vestavěná zařízení a kapesní počítače.

1.2 Cíl projektu

Jak již úvod napověděl, cílem této práce je zajistit funkční implementaci open source projektu OpenVPN na platformě Windows CE, konkrétně verzi Embedded 6.0. V implementaci budeme vycházet z funkční verze OpenVPN pro operačním systém Windows XP SP2. Pro testování programu máme k dispozici vývojový kit NetDCU10 od firmy F&S Elektronik Systeme GmbH.

Ve druhé kapitole se seznámíme s používanými technologiemi. Nejprve, strukturou a principem virtuálních privátních sítí, dále projektem OpenVPN, jeho architekturou a základním použitím. Získáme také informace o platformě Windows CE Embedded 6.0, vlastnostech a struktuře operačního systému. Na konci této kapitoly uvedeme vlastnosti vývojového kitu NetDCU10.

Převod OpenVPN bude z velké části založen na přizpůsobení síťového ovladače prostředí Windows CE, proto třetí kapitola s názvem „Analýza“ uvede obecné vlastnosti ovladačů, model WDM pro operační systémy na bázi Windows NT, Stream interface model operačních systémů Windows CE a rozhraní společné oběma typům operačních systému pro síťovou komunikaci Network driver interface specification (NDIS). Poslední sekce se bude zabývat komunikačním modelem OpenVPN, propojením a vazbami na virtuální síťový ovladač TUN/TAP.

Čtvrtá kapitola popíše specifika převodu, konkrétní kroky nutné k tomu, abychom mohli aplikaci úspěšně portovat na jinou platformu. Detailně rozebere klíčové části převodu, nejprve obecně, poté pro ovladače TAP, TUN a nakonec pro samotné OpenVPN.

Implementace a testování je předmětem kapitoly páté, ve které dáme navrženému řešení praktickou podobu. Zmíníme vývojová prostředí, postup a problémy při implementaci, registraci a spuštění ovladače v systému. Nakonec navrhneme a realizujeme testovací aplikaci, která ověří správnou funkčnost vytvořeného TAP ovladače. Samotná aplikace OpenVPN nebude předmětem implementace.

V závěrečné kapitole zhodnotíme dosažené výsledky, možnosti pokračování v projektu a také vlastní přínos této diplomové práce.

Poznámka: Téměř všechny informace v tomto textu jsou čerpány z literatury psané v anglickém jazyce, kde existuje velké množství ustálených pojmů, které se nepřekládají. Ačkoli jsem se v maximální míře snažil používat český jazyk, v určitých případech by bylo použití násilně přeloženého výrazu spíše na škodu, což by znesnadňovalo by pochopení popisovaného problému.

Kapitola 2

Technologie

Tato kapitola popisuje technologie používané při řešení projektu. Nejprve se zmíníme obecně o sítích VPN, jejich rozdělení a vlastnostech. Následně přejdeme k jedné z jejich konkrétních implementací - OpenVPN. V další sekci popíšeme platformu Windows CE Embedded 6.0, strukturu a vlastnosti systému. Nakonec zmíníme vývojový kit NetDCU10.

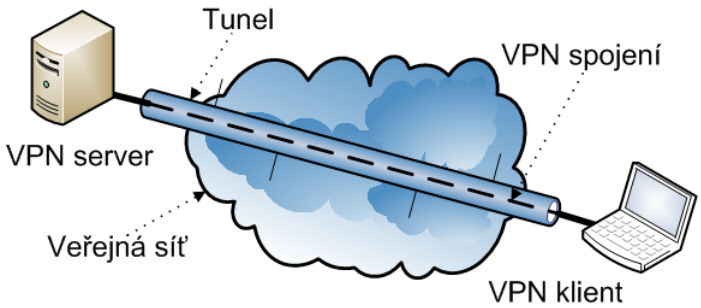
2.1 VPN

Tato část s obecným označením VPN popisuje technologii virtuálních privátních sítí, principy funkce, používané protokoly, a v neposlední řadě se zabývá zajištěním bezpečnosti při přenosu dat veřejnou síťovou infrastrukturou.

2.1.1 Úvod do VPN

Virtuální privátní síť (VPN) je rozšíření privátní sítě, které zahrnuje spojení mezi sdílenými nebo veřejnými sítěmi, jako např. internet. VPN umožňuje zasílat data mezi dvěma počítači přes veřejnou nebo sdílenou síť, čímž napodobuje vlastnosti dvoubodového (point-to-point) privátního spojení.

K emulaci point-to-point spojení jsou data zapouzdřena a je k nim přidána hlavička, která poskytuje směrovací informace umožňující datům dosáhnout cíle přes veřejné nebo sdílené sítě. Pro emulaci privátní linky, tedy zajištění důvěrnosti, jsou zasílaná data šifrována. Část spojení, ve kterém jsou privátní data zapouzdřena, se nazývá tunel. Druhá část spojení, ve které jsou soukromá data šifrována se nazývá VPN spojení (VPN connection).



Obrázek 2.1: Propojení virtuální privátní sítě

VPN dovoluje uživatelům pracovat z domova, nebo se na cestách bezpečně připojit ke vzdálenému podnikovému serveru za použití veřejné sítě (např. internetu). Z pohledu uživatele se VPN spojení jeví jako point-to-point propojení mezi jeho počítačem a podnikovým serverem. VPN technologie také umožňuje firmám propojit své pobočky nebo jiné společnosti přes veřejnou síť. V obou případech, zabezpečená komunikace mezi sítěmi se provádí jako v privátní síti, ačkoli komunikace probíhá v sítí veřejné - odtud název virtuální privátní síť. Další informace o VPN lze nalézt v [6].

Topologie VPN:

- **Site-to-site VPN** - propojení geograficky distribuovaných pobočkových intranetů. Vznikají vytvořením tunelu mezi hraničními uzly (směrovač, firewall) a používají se řešení založené na IPSec, MPLS.
- **Remote access VPN** - sítě pro připojení vzdálených uživatelů k síti. Většinou jako dvoubodové (point-to-point) spojení, používají se protokoly druhé vrstvy (L2TP, L2F, PPTP) nebo řešení založené na vyšších vrstvách (IPSec, SSL VPN, SSH VPN).
- **Extranet** - vytvoření sítě vně podnikového intranetu, přístupný pouze partnerským organizacím.

2.1.2 VPN tunely

VPN využívá mechanismu tunelování mezi koncovými klientskými sítěmi, kdy tunel představuje logický point-to-point spoj, který ve skutečnosti může vést přes komplexní propojené sítě. Definují jej dva koncové body: vstup a výstup z tunelu a mechanismus přenosu paketu tunelem. Hraniční uzly tvořící konce tunelu zodpovídají za zapouzdření, resp. rozbalení původních paketů/rámců přenášených přes transportní síť tunelem. Po opuštění tunelu a jejich rozbalení se rámce/pakety směrují k cíli běžným způsobem. VPN tunely podrobněji popsány v [13].

Nový paket/rámec (přenášený veřejnou sítí) obsahuje následující položky:

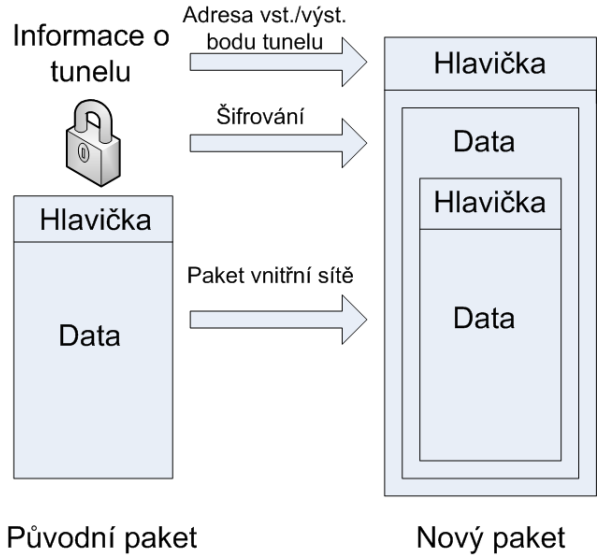
- Hlavičku, obsahující informace o tunelu (adresa cílového bodu)
- V datové částí původní IP paket (síťový rámec), ovšem v zašifrované podobě

2.1.3 Principy VPN

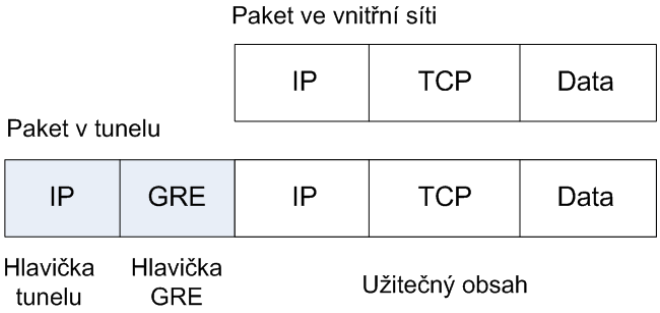
V posledních několika letech bylo představeno mnoho různých konceptů pro VPN sítě. V principu může být tunelování prováděno na téměř všech vrstvách referenčního modelu ISO OSI (viz. [10]).

GRE

General routing encapsulation (GRE) je standardem pro tunelování dat, definován v roce 1994 v RFC 1701 a 1702. Byl navržen pro zapouzdření různých paketů síťové vrstvy do nových IP (tunelovacích) paketů. Originální paket je užitečným obsahem výsledného paketu. Byl vyvinutý firmou Cisco a navržen jako bezstavový. Nezajišťuje šifrování přenášených dat.



Obrázek 2.2: Přenášení dat tunelem



Obrázek 2.3: Princip GRE

Protokoly 2. vrstvy OSI modelu

Zapouzdření dat na druhé (linkové) vrstvě OSI modelu má velkou výhodu - tunel je schopný přenášet protokoly nezaložené na IP. IP je rozšířen v internetu a ethernetových sítích, nicméně existují i jiné standardy, jako např. Internetwork packet exchange (IPX). VPN technologie operující na druhé vrstvě mohou teoreticky tunelovat libovolný druh paketu. Ve většině případů je nainstalováno virtuální point-to-point (PPP) zařízení, které je používáno ke spojení s druhou stranou tunelu.
Následující 4 VPN technologie druhé vrstvy jsou definovány v RFC, používají šifrování a poskytují uživatelskou autentizaci:

- **Point to point tunneling protocol (PPTP)** - vyvíjen za spolupráce firmy Microsoft, rozšířil původní protokol PPP, a je integrován ve všech novějších operačních systémech Microsoftu. PPTP používá pro zapouzdření mechanismus GRE. Nevýhodou je, že umožňuje pouze jeden tunel současně mezi komunikujícími stranami.
- **Layer 2 forwarding (L2F)** - vyvíjen firmou CISCO téměř ve stejné době jako

PPTP a nabízí více možností než PPTP, zejména co se týče tunelování rámců a více otevřených tunelů v jeden okamžik.

- **Layer 2 tunneling protocol (L2TP)** - přijímaný jako průmyslový standard, hodně využíván firmou Cisco, a také ostatními výrobci síťových zařízení. Kombinuje výhody protokolů PPTP a L2F. Přestože nepodporuje vlastní bezpečnostní mechanismy, může být kombinován s jinými protokoly zajišťujícími bezpečnost, jako např. IPSec.
- **Layer 2 security protocol (L2Sec)** - vyvinut pro řešení bezpečnostních nedostatků protokolu IPSec, používá většinou SSL/TLS.

Protokoly 3. vrstvy OSI modelu

IPSec je nejrozšířenějším představitelem tunelovacích protokolů. Přesněji řečeno souboru protokolů pro zabezpečení IP komunikace autentizací a/nebo šifrováním každého IP paketu. IPSec také zahrnuje protokoly pro správu kryptografických klíčů. Pracuje ve dvou režimech:

- **Transportní režim** - šifrována pouze užitečná data, hlavička IP protokolu není modifikována ani šifrována, určen pro komunikaci veřejných koncových uzlů
- **Tunelový režim** - celý paket (data a hlavička) je zašifrován a/nebo autentizován. Poté je zapouzdřen do nového IP paketu pro směrování sítí. Tunelový režim je určen pro komunikaci mezi dvěmi sítěmi: koncovým uzlem a sítí, nebo dvěma koncovými uzly.

Protokoly 4. vrstvy OSI modelu

Máme také možnost vytvořit VPN je na aplikační vrstvě. Tento přístup zajišťují protokoly Secure socket layers (SSL) a Transport layer security (TLS). Uživatel může přistoupit k VPN síti podniku přes internetový prohlížeč. Spojení je navázáno přihlášením přes zabezpečenou HTTPS stránku. SSL je praktickým řešením bezpečného vzdáleného přístupu tam, kde IPSec není vhodné. Jedná se především o prostředí s překladem adres (NAT, Network Address Translation), což může IPSec činit potíže.

2.1.4 Bezpečnost

Je třeba zajistit, aby spolu komunikovaly oprávněné strany, jejichž totožnost je ověřená, a aby měly přístup k prostředkům, které mají k dispozici. Při komunikaci přes veřejnou síť také hrozí, že by data mohla být odposlechnuta, případně změněna. Popíšeme zajištění bezpečnosti pouze u protokolů IPSec a SSL/TLS.

IPSec

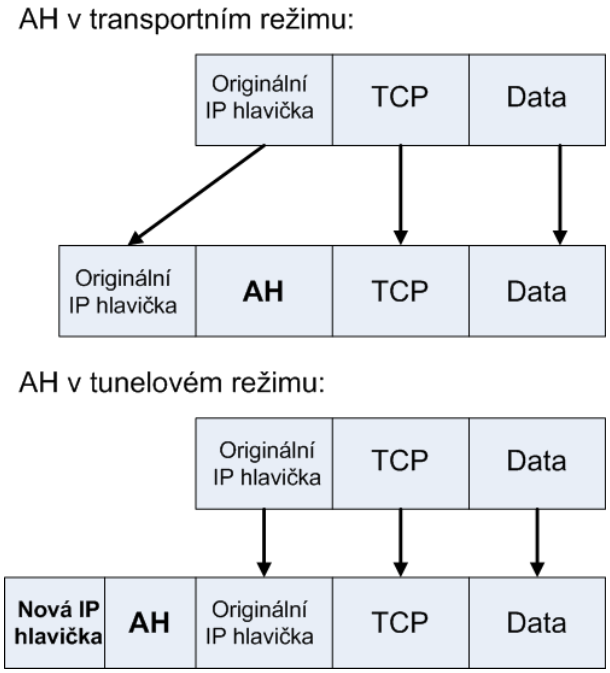
IPSec zajišťuje následující bezpečnostní služby (viz. [3]):

- Důvěrnost dat (šifrováním přenášených dat)
- Integritu dat (autentizační hlavičkou)
- Autentizaci zdroje dat
- Ochranu proti opakovanému posílání dat

- Kontrolu přístupu

K tomu využívá dva bezpečnostní protokoly:

1. **AH** - authentication header. Zajišťuje autentizaci a integritu dat odesílatele, ochranu proti opětovnému poslání datagramu. Používají se hashovací algoritmy HMAC-MD5 a HMAC-SHA1.



Obrázek 2.4: AH v transportním a tunelovém režimu

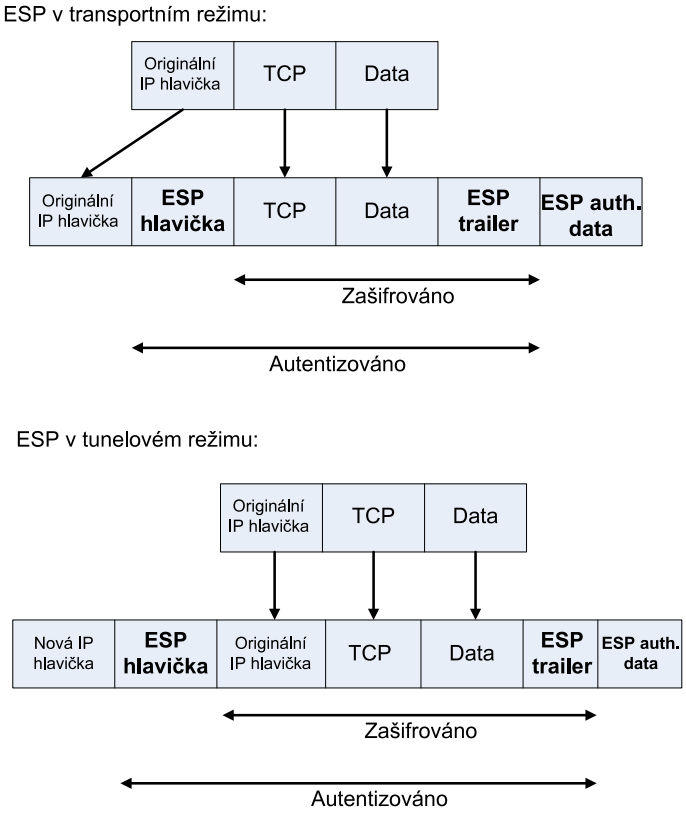
2. **ESP** - encapsulating security payload. Poskytuje důvěrnost zapouzdřením dat IP datagramu, volitelně zajišťuje integritu a autentizaci, pro šifrování využívá symetrické šifrovací algoritmy DES, 3DES a asymetrickou RSA. Pro autentizaci a integritu hashovací funkce HMAC-MD5, HMAC-SHA1

Pro šifrování využívá IPsec především symetrické šifry, které jsou rychlejší (DES, 3DES). Problém s výměnou klíčů řeší protokol IKE (Internet key exchange) využívající asymetrický algoritmus Diffie-Hellman.

SSL

Z hlediska bezpečnosti slabší ochrana než u IPSec, ale méně náročná na implementaci (nezabezpečuje veškerou komunikaci, ale pouze některé aplikace typu klient-server). Vhodné pro zabezpečení webové nebo emailové komunikace, případně sdílení souborů. Nehodí se pro relačně orientované aplikace.

SSL zajišťuje autentizaci uživatele, integritu a šifrování aplikačních dat. Vyžaduje spolehlivý transportní protokol, jako např. TCP. Používá asymetrické šifrování veřejným a soukromým klíčem. Podporuje obousměrnou autentizaci, lze využít jména a hesla, jména a tokenu, nebo digitálních certifikátů X.509.



Obrázek 2.5: ESP v transportním a tunelovém režimu

2.2 OpenVPN

V této sekci popíšeme jednu z implementací virtuálních privátních sítí založenou na protokolu SSL - OpenVPN. Začneme stručným úvodem, historií vývoje, zajišťovanou bezpečností a nakonec příkladem jednoduché konfigurace.

2.2.1 Úvod do OpenVPN

OpenVPN je open source aplikace pro vytváření šifrovaných VPN tunelů mezi dvěma počítači. Program byl napsán Jamesem Yonanem a je distribuován pod licencí GNU GPL.

OpenVPN je plnohodnotné VPN založené na protokolu SSL. Umožňuje vytvářet topologie sítí remote access a site-to-site (viz. sekce 2.1.1). Pracuje na druhé nebo třetí vrstvě referenčního modelu OSI zajištěné protokoly SSL/TLS, podporuje autentizační metody založené na certifikátech, smart kartách a autentizaci jménem/heslem. VPN spojení lze tunelovat přes téměř každý firewall, pro příchozí spojení stačí mít otevřený jeden port. Podporuje mnoho platforem - Linux, Solaris, OpenBSD, FreeBSD, NetBSD, Mac OS X a Windows 2000/XP.

Historie

OpenVPN vzniklo v květnu roku 2001 úvodním releasem označeným jako 0.9. Funkčnost byla omezena pouze na tunelování IP přes UDP, symetrickou šifrou Blowfish a podepisováním hashovací funkcí SHA HMAC. Verze 1.0 z března 2002 již obsahovala podporu SSL/TLS pro autentizaci a výměnu klíčů. Vývoj verzí 1.x pokračoval až do května 2004, kdy poslední měla označení 1.6.0. Mezitím byly opravovány chyby, přidána podpora konfigurace přes externí skripty, instalace přes RPM balíčky, portování na další platformy,

podpora TCP a DHCP.

Paralelně k vylepšování a vývoji OpenVPN 1.x začal v lednu 2003 vývoj verze 2. V únoru 2004 byla na světě první testovací verze s podporou vícenásobného připojování klientů k serveru (multiclient - server). Tato vlastnost je jedna z nejvýznamnějších - několik klientů se může připojit k serveru přes stejný port. V tuto dobu byly dvě vývojové větve 1.6-beta7 a 2.0-test3 spojeny a další vývoj pokračoval pouze verzí 2.x. Než vyšel finální release OpenVPN 2.0 v dubnu 2005, bylo do té doby vydáno 20 betaverzí a 21 release kandidátů. K březnu 2008 existuje poslední stabilní release 2.0.9 a release kandidát č. 7 verze 2.1, která podporuje nejnovější operační systém firmy Microsoft - Windows Vista. Nejvýraznější novinky přidány do verze 2.0 byly následující:

- Podpora více klientů - OpenVPN nabízí speciální režim, kde klientům autentizovaným přes TLS je přes IP poskytnuta adresa jako DHCP serverem. Tímto způsobem může až 128 klientů přes tunel komunikovat ve stejný čas přes TCP nebo UDP port.
- Push/pull volba - nastavení klientů může být řízeno serverem. Po úspěšném sestavení tunelu může server klientovi sdělit, aby použil jiné síťové nastavení.
- Rozhraní pro vzdálenou správu (přes telnet)
- Rozšíření virtuálního síťového ovladače pro operační systém Windows.

Informace čerpány z [4] a [10].

2.2.2 Bezpečnost

OpenVPN nabízí dva autentizační režimy:

1. **Statické klíče** - použití statických sdílených klíčů
2. **TLS** - použití SSL/TLS a certifikátů pro autentizaci a výměnu klíčů

V režimu statických klíčů je vygenerován klíč a sdílen mezi dvěma OpenVPN partnery předtím, než je tunel vytvořen. Statický klíč obsahuje 4 nezávislé klíče: HMAC pro odesílání, HMAC pro příjem, šifrování a dešifrování. První dva klíče zajišťují integritu (hashovací funkce), druhé dva důvěrnost. V implicitním nastavení sdílejí obě strany stejný HMAC pro příjem a odesílání a stejný klíč pro šifrování/dešifrování.

V režimu SSL/TLS je sestaveno SSL spojení s oboustrannou autentizací (každá strana musí vlastnit svůj certifikát, podporovány certifikáty X.509). Jestliže je autentizace přes SSL/TSL úspěšná, klíče pro HMAC a šifrování/dešifrování jsou náhodně vygenerované funkcí knihovny OpenSSL `RAND_bytes` a vyměněny mezi SSL/TSL spojením. Obě strany mají odlišný HMAC pro odesílání a přijímání, šifrování a dešifrování. Podrobněji o bezpečnosti popisuje [14].

2.2.3 Konfigurace

OpenVPN nabízí široké možnosti konfigurace, ukážeme si nejjednodušší nastavení pro vytvoření tunelu mezi dvěma uzly s použitím sdíleného klíče. Detailnější popis konfiguračních příkazů OpenVPN lze nalézt v [10].

Mějme dva koncové uzly, první s adresou 147.100.10.1 a druhý 147.100.10.2. Mezi nimi chceme vytvořit šifrovaný tunel se sdíleným statickým klíčem. Vnitřní síť bude privátní, s adresou prvního uzlu 10.10.10.1 a druhého uzlu 10.10.10.2.

Nejprve je nutné pomocí příkazu `openvpn --genkey --secret secret.key` vygenerovat sdílený klíč, který bezpečnou cestou dopravíme na druhý uzel. Konfigurační soubor pro první uzel bude mít následující obsah:

```
remote 147.100.10.2 #druhá strana tunelu
ifconfig 10.0.0.1 255.255.255.0 #adresa a maska vnitřní sítě
port 5001 #port, ke kterému je služba OpenVPN připojena
proto udp #protokol komunikace
dev tap0 #typ a číslo virtuálního rozhraní
secret secret.key #soubor se sdíleným klíčem
ping 10 #intervaly mezi kontrolou spojení
```

Konfigurační soubor druhého uzlu se liší pouze ve dvou řádcích, jinak obsah stejný jako u prvního:

```
remote 147.100.10.1
ifconfig 10.0.0.2 255.255.255.0
```

Poté již na obou uzlech spustíme program `openvpn.exe` s parametrem `--config config_file.ovpn`, kde v souboru `config_file.ovpn` se nachází výše popsané nastavení. Nyní je již veškerá komunikace mezi uzly šifrována. Do takto vytvořeného tunelu lze přesměrovat libovolný síťový provoz. Při výpadku spojení se OpenVPN automaticky pokusí o obnovu.

2.3 Windows CE Embedded 6.0

Detailní popis operačního systému Windows CE by zabralo více prostoru, než dovoluje rozsah této práce, proto si uvedeme jen základní koncepty a odlišnosti od desktopových verzí Windows. Informace v této sekci čerpány z [9].

2.3.1 Úvod a historie

Windows CE je nejmenší systém z rodiny operačních systému Microsoft Windows. Byl navrhován jako kompaktní, multivláknový operační systém s nízkou spotřebou, volitelným grafickým uživatelským rozhraním a podmnožinou Win32 API.

K popisu historie Windows CE je nutné pochopit rozdíly mezi operačním systémem a produkty, které jej používají. Windows CE je vyvíjen skupinou programátorů ve firmě Microsoft. Jiné firmy, které vyvíjí zařízení, jako např. řada Windows Mobile používají nejvhodnější verzi Windows CE, která je dostupná v době vydání zařízení na trh.

Historie zařízení

První výrobky navržené pro Windows CE byly kapesní organizéry s obrazovkami o velikosti 480 na 240 pixelů nebo 640 na 240 pixelů s klávesnicí. Byly uvedeny koncem roku 1996, využívaly operační systém Windows CE 2.0.

V lednu roku 1998 představil Microsoft dvě nové platformy - Palm-size PC a Auto PC. Palm-size PC bylo reakcí na zařízení ovládaná perem, kterým tehdy dominovaly systémy založené na Palm OS. Nicméně, Palm-size PC se na trhu příliš neujaly.

V dubnu 2000 se dostalo do prodeje nové zařízení od Microsoftu - Pocket PC, což byla výrazně vylepšená verze staršího Palm-size PC. První Pocket PC používalo doposud

nevydanou verzi operačního systému Windows CE 3.0. Uživatelské rozhraní se změnilo do dnešní podoby s tzv. „today“ obrazovkou. Největší změnou byl ovšem rapidně vylepšený výkon samotného Windows CE.

Pocket PC byl aktualizovaný v roce 2001 a pojmenován Pocket PC 2002. Toto vydání bylo postaveno na finální verzi Windows CE 3.0 a obsahovalo vylepšení především v oblasti uživatelského rozhraní. Začal také vývoj zařízení Pocket PC Phone Edition, který integroval podporu mobilních telefonů do Pocket PC. Tato zařízení kombinovala funkcionalitu Pocket PC spolu s vlastnostmi mobilních telefonů.

Jiná vývojová skupina uvnitř Microsoftu vydala tzv. Smart display, což byl systém založený na OS Windows CE .NET 4.1, který integroval tablet s bezdrátovým modulem, který se připojoval k základně (počítači). Pokud byl Smart displej připojen k základně, choval se jako druhý monitor, pokud byl odpojen, sloužil jako mobilní displej pro počítač. Byl základem technologie remote desktop.

Na jaře roku 2003 uveřejnil tým vyvíjející Pocket PC aktualizaci nazvanou Pocket PC 2003. Změna uživatelského rozhraní zde nebyla tak patrná, jako u předchozí verze, ovšem zásadně se vylepšila stabilita a výkon, protože tento systém byl postaven na Windows CE .NET 4.2. Byl to také první systém, který integroval podporu pro Bluetooth.

Další aktualizace Pocket PC a platformy Smartphone nazývaná Pocket PC/Smartphone 2003 byla vydána v březnu 2004. Zařízení podporovala různá rozlišení displeje, rotace a vylepšenou podporu komunikace. Tyto systémy byly postaveny na lehce modifikovaném jádře Windows CE .NET 4.2.

V květnu 2005 přešly platformy Pocket PC a Smartphone pod jeden název, a to Windows Mobile. Vycházely z operačního systému Windows CE 5, což zahrnovalo přechod ze souborového systému založeného na RAM na souborový systém založený na flash pamětech. Tato změna chránila před ztrátou dat způsobenou výpadkem napájení.

Tým vyvíjející Windows Mobile pokračoval v únoru v 2007 s vydáním verze Windows Mobile 6. Je stavěn stále na jádře operačního systému Windows CE 5, ačkoli verze Windows CE 6 vyšla několik měsíců předtím. Názvosloví se opět změnilo, Pocket PC na Windows Mobile Classic, Pocket PC Phone Edition na Windows Mobile Professional a Smartphone na Windows Mobile Standard.

V současné době stále pokračuje vývoj nových zařízení, mezi nejnovější patří multimediální přehrávač Zune od Microsoftu postavený na Windows CE.

Historie operačního systému

Vývoj Windows CE začal verzí 1.0, kdy se jednalo o jednoduchý organizér. Verze 2.0 byla vydána s uvedením Hanheld PC 2.0. Windows CE 2.0 přidal podporu pro sítě, zahrnující standardní funkce Windows pro práci se sítí, NDIS miniport model, obecný síťový ovladač NE2000, podporu komponentové technologie COM, zobrazování větší hloubky pixelů než 2 bity jako u verze CE 1.0.

V srpnu 1998 bylo představeno zařízení H/PC Professional spolu s novou verzí operačního systému 2.11. Windows CE 2.11 byl service pack k oficiálně nevydané verzi 2.1. Zároveň vyšel SDK Microsoft Platform Builder pro vývoj aplikací na této platformě. CE 2.11 přidává podporu pro soubory větší než 4MB, příkazový řádek a IrDA stack.

Dlouho očekávaný Windows CE 3.0 vyšel v polovině roku 2000. Následoval dubnový příchod Pocket PC, který obsahoval interní build OS Windows CE 3.0. Největší novinkou bylo přepracované jádro, optimalizované pro lepší podporu real-time aplikací. Nově definuje 256 priorit pro vlákna oproti předchozím 8, nastavovatelné časové kvantum pro

vlákna, vnořená přerušení a zredukované prodlevy jádra. Dále byla vylepšena podpora COM a DCOM, možnost využití 256MB RAM pro datové úložiště, limit pro jeden soubor zvětšen na 32MB. Vychází též Platform Builder 3.0

Další verze Windows CE nezahrnovala jen nové vlastnosti, ale i změnu jména. Windows CE .NET 4.0 vyšlo v roce 2001, změnilo organizaci virtuální paměti, nový model nahrávání ovladačů, podporu služeb operačního systému, Bluetooth a 1394 (FireWire). I když bylo do jména OS přidáno .NET, tato verze ještě neobsahovala .NET Compact Framework.

Později v roce 2001 vyšla verze Windows CE 4.1, přidává podporu IPv6, Winsock 2 a .NET Compact Framework.

V polovině roku 2003 byla uvedena na trh verze Windows CE .NET 4.2. Poskytuje nové možnosti pro výrobce, kteří chtějí podporovat aplikace pro Pocket PC na vestavěných zařízeních. Výkon zvýšen podporou stránkování hardwarem.

V červenci 2004 vyšla verze Windows CE 5.0. Opět zvýšen výkon, především u síťového zásobníku a souborového systému. Výrazně vylepšen Platform Builder usnadňující převod operačního systému na nový hardware.

Největší změna a aktualizace v historii CE přišla s verzí Windows Embedded 6.0 v lednu 2006. Jádro Windows CE 6 bylo kompletně přepsáno, podporuje až 32 tisíc souběžně běžících procesů oproti 32 v předchozí verzi a 2GB paměti na proces oproti 32MB v předchozí verzi.

2.3.2 Správa paměti

Systémy s OS Windows CE mají paměť typu RAM i ROM, ovšem obě tyto paměti se používají odlišně než na běžných PC.

RAM

Paměť RAM je na Windows CE používána za stejným účelem, jako na jiných operačních systémech, to znamená pro haldu (heap), zásobník, a někdy kód aplikací. Narozdíl od jiných OS, část RAM může být využita pro ukládání objektů. Další rozdíl je ten, že objekty v RAM zůstávají i po vypnutí a resetu systému, protože systémy s Windows CE obsahují kromě hlavní baterie i záložní, která stále napájí paměť RAM. Hranici mezi využitím RAM pro systém a úložiště lze nastavit. Jestliže objektové úložiště na RAM není využito, musí být nahrazeno jiným souborovým systémem, třeba jako flash pamětí nebo jednoduše souborovým systémem ROM (pouze pro čtení).

ROM

Na osobních počítačích se paměť ROM používá k uložení BIOSu a má typicky 64-128 kB. Ve Windows CE se její velikost pohybuje od 4 do 32 MB a obsahuje celý operační systém, stejně jako aplikace s ním dodávané. ROM paměť se tedy chová jako pevný disk pouze pro čtení.

Programy uložené v ROM paměti mohou být navrženy jako Execute in place (XIP). To znamená, že je lze spustit přímo z ROM bez toho, aby se nejprve musely nahrát do RAM. Výhodou je rychlejší start aplikace a nezabírá místo v RAM. Programy, které nejsou v ROM, ale v objektovém úložišti, flash paměti nebo jiném pevném disku musí být před spuštěním načteny do RAM.

Virtuální paměť

Windows CE také podporují mechanismus virtuální paměti. Její výhodou je fakt, že aplikace nejsou závislé na fyzické implementaci paměti zařízení (Windows CE podporují širokou škálu HW implementací), a také že mohou využívat mnohem větší adresový prostor, než je fyzicky přítomný v systému.

Virtuální paměť je stránkována, nejmenší část paměti, se kterou procesor pracuje se nazývá stránka. Pokud aplikace přistupuje ke stránce, procesor přeloží virtuální adresu na fyzickou v ROM nebo RAM. Operační systém určuje, zda je stránka platná. Pokud ano, namapuje fyzickou stránku paměti do virtuální. Velikost stránky na Windows CE je 4kB, stejně jako na 32 bitových implementacích Windows XP a Vista. Virtuální stránky mohou být ve třech stavech: volná, rezervovaná nebo potvrzená. Volná znamená, že může být využitá pro alokaci. Rezervovaná značí, že už nemůže být využita operačním systémem nebo jiným vláknem v procesu. Stránka ve stavu rezervovaná nemůže být použita ani aplikací, dokud není namapována do fyzické paměti. Pro namapování musí přejít do stavu potvrzená. Potvrzená stránka byla rezervována aplikací a namapována na fyzickou adresu.

Aplikační adresový prostor

Virtuální adresový prostor přístupný aplikacím na Windows CE 6 se výrazně zvětšil oproti předchozím verzím Windows CE, kde měl velikost 32 MB virtuálního prostoru na aplikaci. Nyní lze použít až 2 GB prostoru, z toho je 1 GB dostupný aplikacím pro alokaci paměti, druhý 1 GB pro speciální účely.

2.3.3 Moduly, procesy, vlákna

Stejně jako Windows Vista, Windows CE je multiprogramový a multivláknový operační systém. Soubory se spustitelným kódem se nazývají moduly. Windows CE podporuje dva typu modulů: s příponou EXE a dynamicky linkované knihovny - DLL. Spuštěním modulu vzniká proces, který má svůj oddělený paměťový prostor. Procesů může v jeden okamžik běžet v CE 6.0 až 32 tisíc, předchozí verze byly omezeny na 32 spuštěných procesů v jeden okamžik. Každý proces má minimálně jedno vlákno, vlákna jsou založená na podobném principu jako na ostatních verzích Windows.

Moduly

Formát souborů pro moduly je stejný jako na desktopových verzích Windows (PE formát). Narozdíl od Windows XP a Vista zde není podpora pro souborový formát SYS používaný pro ovladače zařízení. Místo toho jsou ve Windows CE implementovány ovladače jako DLL soubory.

Procesy

Procesy na Windows CE obsahují oproti Windows XP a Vista méně stavových informací a to především z důvodu, že systém nepodporuje koncept aktuálních adresářů, všechny cesty se musí uvádět jako absolutní. Dále nepodporuje proměnné prostředí. Proces tedy nemusí tyto informace ukládat.

2.3.4 Souborový systém

Windows CE poskytuje plnohodnotný souborový zásobník (stack), který podporuje širokou škálu souborových systémů a úložných médií (flash, pevné disky). Unikátní je souborový systém založený na RAM paměti zvaný úložiště objektů (object store). Úložiště objektů je implementováno jako databáze, ovšem navenek skryto přes standardní Win32 API funkce pro práci se soubory.

Aplikační rozhraní pro práci se soubory je přímo zděděno z Win32 API. Mezi hlavní rozdíly patří, že Windows CE nepoužívá pro jména disků znaky (C:, D:, ...). Místo toho je cesta k souboru definovaná z kořenového adresáře, kde se do složek připojují kořenové adresáře jednotlivých souborových systémů. Spolu s chybějícími jmény jednotlivých disků není podporován koncept aktuálních adresářů, tzn. soubory jsou vždy reprezentovány plnou cestou.

Formát jmen souborů je stejný jako na ostatních verzích Windows. Přípona je tříznaková a definuje typ souboru. Povolené znaky ve jméně jsou stejné jako na desktopových verzích Windows, podporuje dlouhá jména souborů.

2.3.5 NetDCU10

NetDCU jsou vestavěná počítačová zařízení s operačním systémem Windows CE nebo Linux. Specializovaným výrobcem této rodiny jednodeskových řídících počítačů je firma F&S Elektronik Systeme GmbH.

NetDCU10 vychází ze systému NetDCU8. Je postaven na procesoru Samsung ARM9 s frekvencí 400 MHz. Obsahuje 64MB flash/SDRAM, rozhraní pro připojení externích SecureDigital (SD) flash karet pro rozšíření paměti, podporuje rozhraní LCD displeje, dotykového panelu, 3xRS232, 2xUSB (host, zařízení), klávesnice a ethernetové rozhraní.

NetDCU10 běží na nejnovějším operačním systému firmy Microsoft Windows Embedded CE 6.0. Vývoj aplikací je možný v prostředí Visual Studio 2005 v jazycích C++, C# nebo VB.NET.

Obrázek 2.6: NetDCU10 bez vývojového kitu s rozhraními

Kapitola 3

Analýza

Převod programu OpenVPN na platformu Windows CE sestává ze dvou kroků:

1. Převod virtuálního síťového ovladače TUN/TAP
2. Převod samotného programu OpenVPN

V této kapitole nejprve podrobněji popíšeme obecnou strukturu a funkci ovladačů v operačním systému Windows na bázi NT, poté přejdeme k popisu ovladačů na platformě Windows CE. Cílem první části implementace je síťový ovladač, pro který operační systémy Microsoftu používají subsystém NDIS (Network driver interface specification), jež bude taktéž popsán. Nakonec zmíníme strukturu samotného programu OpenVPN, jeho návaznost na systém a komunikaci se síťovými ovladači. Převod OpenVPN bude cílem druhé části implementace.

3.1 Ovladače zařízení (Device drivers)

V této sekci nejprve uvedeme obecné vlastnosti ovladačů, a poté popíšeme rámcově architekturu Windows driver model (WDM), která se používá pro psaní ovladačů zařízení v operačním systému Windows. Všechny následující informace získány z [7].

3.1.1 Ovladače - úvod

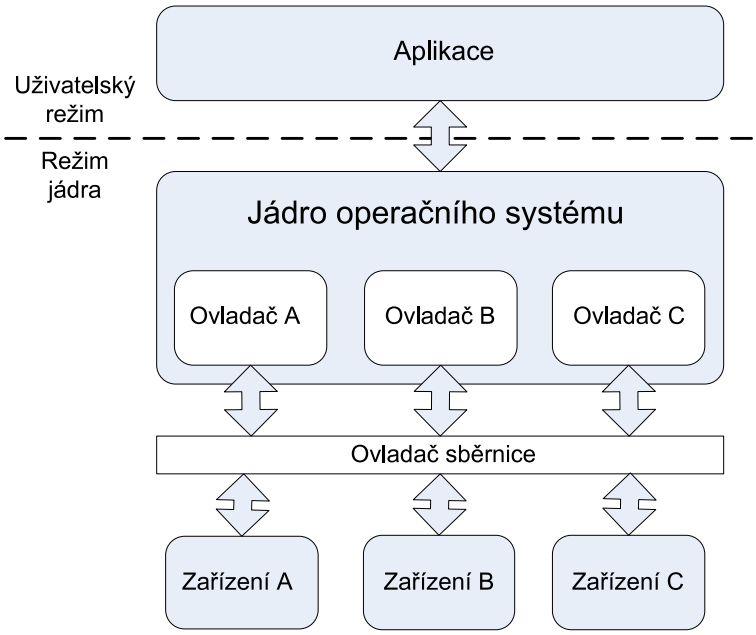
Ovladač zařízení je software, který umožňuje operačnímu systému (OS) pracovat s hardwarem. Některé ovladače jsou součástí OS, jiné distribuovány výrobcem HW. Jsou závislé na konkrétním hardwaru a specifické pro každý operační systém.

Ovladač zajišťuje řízení hardware a zároveň komunikuje se zbytkem operačního systému pomocí obecnějších rozhraní, která zajišťují abstrakci zařízení. Abstrakcí je myšleno použití stejného nebo podobného rozhraní pro podobné typy zařízení.

Ovladač typicky komunikuje se zařízením prostřednictvím sběrnice nebo komunikuje se subsystémem, ke kterému je hardware připojen. Jestliže program vyvolá přes rozhraní operačního systému rutinu ovladače, vyšle se příkaz k zařízení. V okamžiku, kdy je příkaz zpracován, jsou data zaslána zpět k ovladači a ten spustí obslužnou rutinu volajícího programu. Obrázek 3.1 znázorňuje umístění ovladačů v architektuře operačního systému.

Ovladače se nejčastěji využívají pro:

- grafické karty



Obrázek 3.1: Umístění ovladačů v operačních systémech

- síťové adaptéry
- zvukové karty
- počítačové tiskárny, skenery
- zařízení na sběrnici (bus)
- úložné zařízení, souborové systémy
- ...

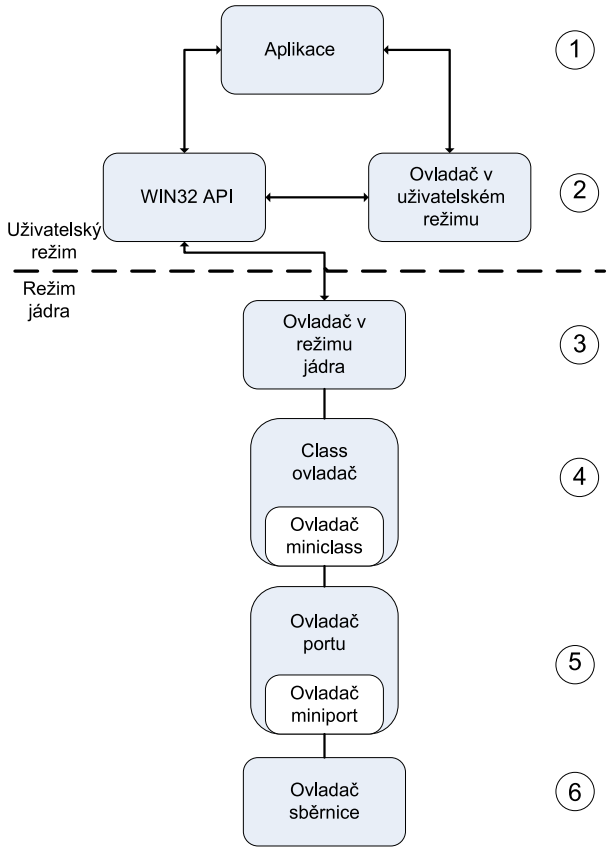
3.1.2 Architektura ovladačů ve Windows

Operační systémy Windows podporují vrstvovou architekturu ovladačů. Každé zařízení je obsluhováno řetězcem ovladačů, obvykle nazývaným zásobník ovladačů (driver stack). Obrázek 3.2 znázorňuje typy ovladačů, které se mohou vyskytovat ve frontě ovladačů pro hypotetické zařízení. Popis vrstev:

1. Nad zásobníkem ovladačů je aplikace. Aplikace zpracovává požadavky od uživatelů a jiných aplikací, a volá buď metodu rozhraní Win32 API nebo rutinu nabízenou ovladačem v uživatelském režimu (user-mode driver)

2. Ovladač v uživatelském režimu zpracovává požadavky z aplikací nebo z rozhraní Win32 API. Pro požadavky, které vyžadují služby jádra, user-mode ovladač volá Win32 API, které zavolá příslušnou rutinu jádra za účelem provedení příslušné akce. Ovladače v uživatelském režimu jsou obvykle implementovány jako DLL knihovny.

3. Ovladač v režimu jádra (kernel-mode driver) zpracovává požadavky podobně jako ovladač v uživatelském režimu, s výjimkou toho, že tyto požadavky jsou prováděny v režimu jádra



Obrázek 3.2: Existující typy ovladačů v OS Windows

4. Dvojice ovladačů class a miniclass poskytují převážnou část podpory specifické pro dané zařízení. Class ovladače dodávají hardwarově nezávislou podporu pro jednotlivou třídu zařízení, jsou typicky dodávány výrobcem OS.

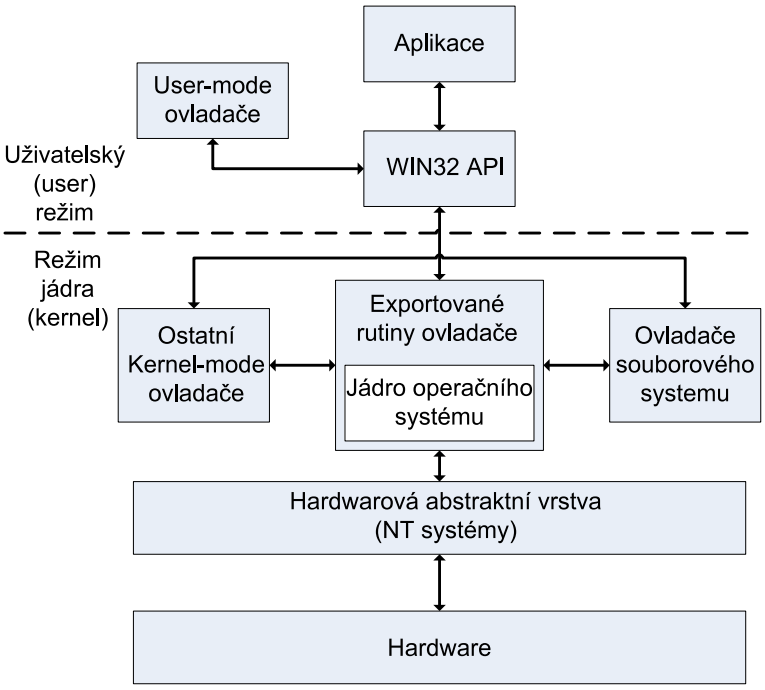
   Miniclass ovladače zpracovávají operace pro specifické typy zařízení jednotlivé třídy. Tyto ovladače jsou obvykle dodávány výrobcem hardwaru.

5. Odpovídající ovladač portu (pro některá zařízení nazývána host controller, případně host adapter driver) podporuje požadované I/O operace pro port, hub nebo fyzické zařízení, které má pod sebou, a přes které je zařízení připojené. Přítomnost těchto ovladačů závisí na typu zařízení a na sběrnici, ke které je připojené (všechny fronty ovladačů pro úložná zařízení mají port driver, např. SCSI port driver poskytuje podporu pro I/O přes SCSI sběrnici).

   Odpovídající miniport ovladač zpracovává operace specifické pro ovladač portu. Pro většinu typů zařízení je ovladač portu dodáván výrobcem OS a miniport výrobcem zařízení.

6. Ovladač sběrnice je pro velkou většinu sběrnic distribuován výrobcem OS

Například grafické karty vyžadují tzv. display driver, video port driver a video miniport driver. Display driver je analogický k ovladači v režimu jádra na předchozím obrázku. Poskytuje obecné vykreslovací schopnosti a často pracuje více jak s jednou grafickou kartou. Video port driver podporuje na zařízení nezávislé grafické operace. Pracuje ve spojení s video miniport ovladačem, který poskytuje funkcionalitu specifickou pro každý typ grafické karty. V tomto příkladu není vyžadován class/miniclass ovladač.



Obrázek 3.3: Architektura komponent operačního systému Windows

3.1.3 Rozdělení ovladačů ve Windows

Existují dva základní typy ovladačů ve Windows:

1. **ovladače v uživatelském režimu (user-mode drivers)** - typicky poskytují rozhraní mezi aplikacemi a ovladači v režimu jádra, případně jinými komponentami OS.

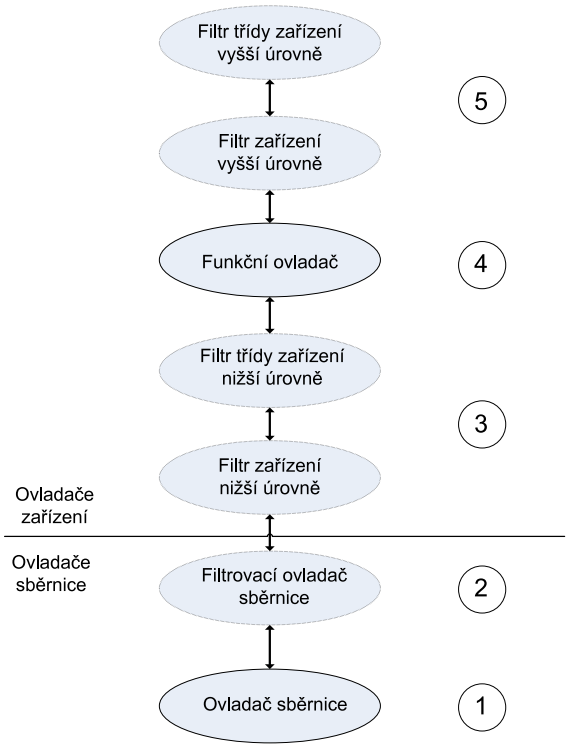
2. **ovladače v režimu jádra (kernel-mode drivers)** - patří mezi ně většina ovladačů OS, spouští se v režimu jádra, jsou ve většině případů vrstvené. Bývají implementovány jako samostatné modulární komponenty, které mají přesně definovaný soubor operací. Základní rozdělení vrstev je následující:

   - **Ovladače nejvyšší úrovně (highest-level drivers)** - patří mezi ně ovladače souborového subsystému (NTFS, FAT, CDFS)
   - **Ovladače střední úrovně (intermediate drivers)** - lze je dále rozdělit na funkční a filtrovací ovladače, jsou závislé na podpoře ovladačů nejnižší úrovně, zajišťují obvykle specifickou funkcionalitu pro dané hardwarové zařízení
   - **Ovladače nejnižší úrovně (lowest-level drivers)** - řídí sběrnici, na kterou je zařízení připojeno (bus drivers)

Obecně, ovladače nejvyšší úrovně získají data od aplikace, která mohou nějakým způsobem filtrovat (intermediate) a pošlou je k ovladači nejnižší úrovně k zajištění požadované funkcionality. Obrázek 3.3 znázorňuje celkový pohled na komponenty operačního systému Windows a vztah ovladačů k těmto komponentám.

3.1.4 WDM - Windows driver model

K zajištění přenositelnosti ovladačů mezi jednotlivými verzemi OS Windows byl představen tzv. Windows driver model (WDM). Je podporován v operačních systémech Windows 98/NT/2000/XP/Vista. WDM se stále vyvíjí, zpětná kompatibilita je zachována, avšak



Obrázek 3.4: Vztahy mezi jednotlivými typy ovladačů

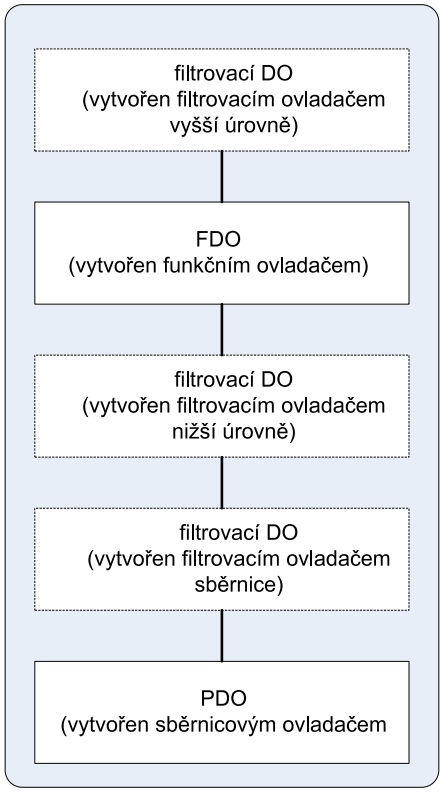
na starších verzích OS nemusí být zajištěna funkcionalita ovladače psaného v novější verzi WDM. WDM definuje tři typy ovladačů:

- **Ovladače sběrnice (bus drivers)** - řídí jednotlivé vstupně/výstupní sběrnice a poskytují funkcionalitu, která je nezávislá na jednotlivých typech zařízení.
- **Funkční ovladače (function drivers)** - řídí jednotlivá zařízení
- **Filtrovací ovladače (filter drivers)** - filtrují vstupně/výstupní požadavky na zařízení, třídu zařízení nebo sběrnici.

Obrázek 3.4 znázorňuje vztah mezi sběrnicovým, funkčním a filtrovacím ovladačem.

Každé zařízení má typicky ovladač pro rodičovskou vstupně/výstupní sběrnici, funkční ovladač pro samotné zařízení a žádné nebo několik filtrovacích ovladačů. Bližší popis obrázku 3.4:

1. **Sběrnicový ovladač** - obsluhuje sběrnici či adaptér. Pro každý typ sběrnice na počítači existuje jeden ovladač. OS většinou poskytuje ovladače pro běžné typy sběrnic (PCI, SCSI, USB)
2. **Filtr ovladače sběrnice** - typicky přidává dodatečnou funkcionalitu sběrnici, v systému jich může existovat libovolný počet. Příkladem může být ACPI filtr, spravující napájení a zapínání/vypínání zařízení na sběrnici. ACPI filtr je transparentní pro ostatní ovladače.
3. **Filtrovací ovladače nižší úrovně** - modifikují chování hardwarového zařízení a také I/O požadavků, jsou volitelné. Např. pro třídu zařízení ovladačů myši zajišťuje akceleraci prováděním nelineární konverze dat o pohybu myši.



Obrázek 3.5: Zásobník zařízení (device stack)

4. **Funkční ovladače** - hlavní ovladač zařízení, nutný k jeho funkci, obvykle dodáván výrobcem hardwaru

5. **Filtrovací ovladače vyšší úrovně** - poskytují dodatečnou funkcionalitu zařízení, volitelné. Pro klávesnici může například zajistit dodatečné bezpečnostní kontroly dat o stisku kláves.

3.1.5 Objekty zařízení (Device objects)

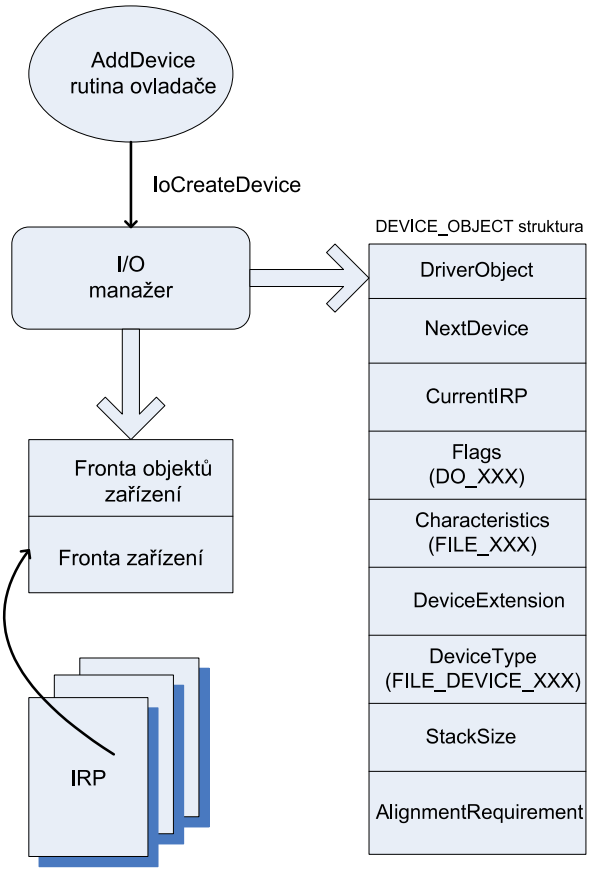
V operačním systému jsou zařízení reprezentovány tzv. objekty zařízení (device objects). Ke každému fyzickému (virtuálnímu) zařízení je přiřazen jeden či více objektů. Pokud aplikace vyžaduje provést určitou operaci na zařízení, cílem volání je právě objekt zařízení.

Ovladače jádra musí vytvořit alespoň jednu instanci objektu pro každé zařízení s následujícími výjimkami:

- ovladače mají přiřazenou třídu ovladačů, a proto nemusí vytvářet vlastní objekt zařízení

- ovladače jsou součástí specifických subsystémů, například ovladače NDIS (bude zmíněn dále), které mají vlastní objekty zařízení vytvářené daným subsystémem

Zařízení jsou obvykle reprezentovaná několika objekty zařízení, jeden pro každý ovladač. Objekty zařízení jsou uchovávány v tzv. zásobníku zařízení (na obrázku 3.5). Kdykoli se vyskytne žádost o provedení operace na zařízení, systém zašle datovou strukturu IRP (I/O request packet) ovladači zařízení, které se nachází na začátku fronty zařízení. Každý ovladač buď IRP zpracuje nebo jej pošle k ovladači, který je přiřazen nižšímu objektu zařízení v zásobníku zařízení.



Obrázek 3.6: Architektura WDM

Objekty zařízení jsou reprezentovány strukturami `DEVICE_OBJECT`, které jsou řízeny správcem objektů (object manager). Systém též poskytuje vyhrazený prostor pro specifické vlastnosti jednotlivých zařízení, nazývaný „device extension“. Na obrázku 3.6 vidíme vztah mezi objekty zařízení a I/O manažerem.

Ve WDM existují 3 typy objektů zařízení:

- **Fyzické objekty zařízení (Physical device object PDO)** - reprezentují zařízení na sběrnici pro ovladač sběrnice (bus driver)
- **Funkční objekty zařízení (Function device object FDO)** - reprezentují zařízení pro funkční ovladač (function driver)
- **Filtrovací objekty zařízení (Filter device object DO)** - reprezentují zařízení pro filtrovací ovladač (filter driver)

3.1.6 Komponenty ovladačů jádra

Každý ovladač jádra je vytvářen na množině systémem definovaných standardních rutinách ovladačů (standard driver routines). Ovladače jádra zpracovávají vstupně/výstupní pakety požadavků (I/O request packets, dále jen IRPs) právě pomocí těchto standardních rutin.

Všechny ovladače, bez ohledu na jejich úroveň v řetězci ovladačů, musí mít implementovánu základní množinu standardních rutin určených pro zpracování IRP. Zda musí implementovat i jiné rutiny závisí na tom, co daný ovladač řídí. Obecně, ovladače nižší úrovně vyžadují více obslužných rutin než ovladače vyšší úrovně, které typicky zašlou IRP k nižšímu ovladači pro zpracování.

Standardní rutiny mohou být rozděleny do dvou skupin:

1. povinné, které každý ovladač musí implementovat:
   - **DriverEntry** - inicializuje ovladač a jeho objekt
   - **AddDevice** - inicializuje zařízení a vytváří objekty zařízení
   - **Dispatch routines** - přijímají a zpracovávají IRP
   - **Unload** - uvolňuje systémové prostředky, vyžadované ovladačem
2. volitelné, které závisí na typu ovladače a na pozici, kde se nachází ve frontě zařízení: (výpis jen některých, kompletní seznam v [7])
   - **StartIo** - začíná vstupně/výstupní operaci na fyzickém zařízení
   - **Interrupt service routine** - ukládá stav zařízení při přerušení
   - **Deferred procedure calls** - zpracovává přerušení po uložení stavu zařízení
   - **AdapterControl** - iniciuje DMA operace
   - **IoCompletion** - ukončuje zpracování IRP ovladačem
   - **Cancel** - ruší zpracovávání IRP ovladačem

Vstupními argumenty standardních rutin je cílový objekt zařízení a zpracovávané (aktuální) IRP.

3.1.7 I/O request packets - IRPs

Protože problematika zpracování IRP je poměrně složitá a kompletní popis by překročil rozsah této práce, uvedeme pouze základní koncepty a vlastnosti IRPs. Případné zájemce lze odkázat na [2].

Úvod do IRPs

V operačních systémech Windows na bázi NT komunikuje operační systém s ovladači zasíláním vstupně/výstupních paketů s požadavky - I/O request packets - IRPs. Datová struktura zapouzdřující IRP nepopisuje pouze I/O požadavek, ale také zpracovává informace o stavech požadavku, jak je zasílán postupně přes jednotlivé ovladače, které jej zpracovávají. Datová struktura IRP slouží dvěma účelům, a může být definováno jako:

- kontejner pro I/O požadavky
- zásobník nezávislých vláken s požadavky (thread-independent call stack)

IRP jako kontejner pro I/O požadavky

Operační systém zasílá nejvíce I/O požadavků k ovladači právě přes IRPs. IRPs jsou vhodné pro tento účel protože:

- mohou být zpracovávány asynchronně
- mohou být zrušeny
- lze je zpracovat více jak jedním ovladačem

Obrázek 3.7: Struktura IRP

Datová struktura IRP zabaluje informace, které ovladač potřebuje k odpovědi na I/O požadavek. Požadavek může být zaslán jak z uživatelského režimu, tak z režimu jádra. Každé IRP má dvě části:

1. Hlavičku, která popisuje primární I/O požadavek
2. Pole parametrů, které popisují vedlejší požadavky (někdy nazývané pod-požadavky)

Velikost hlavičky je stejná pro každé IRP, velikost pole parametrů závisí na počtu ovladačů, které budou IRP zpracovávat.

IRP hlavička: IRP je obvykle zpracováváno zásobníkem ovladačů. Hlavička IRP obsahuje data, která jsou používána každým ovladačem, který zpracovává IRP. Ovladač, který právě zpracovává IRP je nazýván „aktuálním vlastníkem IRP“ (current owner). Hlavička každého IRP obsahuje stav zpracování a odkazy na následující položky:

- **Buffery** pro čtení vstupu a zápis výstupu IRP
- **Oblast paměti** pro ovladač, který právě zpracovává IRP
- **Rutina** získaná aktuálním vlastníkem IRP, kterou operační systém volá, jestliže je IRP rušeno
- **Parametry** pro aktuální vedlejší požadavek

Pole parametrů: Za hlavičkou se nachází pole vedlejších požadavků. IRP může mít více jak jeden vedlejší požadavek, protože IRP je většinou zpracováno více ovladači. Každé IRP je alokováno (I/O manažerem) s pevným počtem vedlejších požadavků, obvykle jedním pro každý ovladač ze zásobníku. Toto číslo koresponduje s položkou `StackSize` (viz. obrázek 3.6) prvního objektu zařízení z fronty, tedy ovladač uprostřed fronty může alokovat méně. Jestliže ovladače musí přeposlat požadavek do jiné fronty zařízení, je nutné vytvořit nové IRP.

Každý vedlejší požadavek je reprezentován strukturou `IO_STACK_LOCATION` a IRP obvykle obsahuje jednu takovou strukturu pro každý ovladač v zásobníku zařízení, ke které je IRP zasláno. Pole v hlavičce IRP identifikuje strukturu `IO_STACK_LOCATION`, která se aktuálně používá. Hodnota tohoto pole se nazývá ukazatel zásobníku (IRP stack pointer) nebo aktuální pozice v zásobníku (current stack location). Struktura `IO_STACK_LOCATION` obsahuje následující položky:

- Primární (major) a vedlejší (minor) kódy funkcí pro IRP

- Argumenty specifické těmto kódům
- Ukazatel na objekt zařízení pro odpovídající ovladač
- Ukazatel na rutinu IoCompletion, jestliže ovladač definuje
- Ukazatel na souborový objekt asociovaný s požadavkem
- Volitelné příznaky

Vedlejší požadavky pracují se stejnými buffery, které definuje IRP ve své hlavičce. Množina primárních a vedlejších kódů, které jednotlivé zařízení zpracovává, je specifické pro každé zařízení. Nicméně, ovladače střední a nejnižší úrovně obvykle zpracovávají následující soubor primárních kódů:

- **IRP_MJ_CREATE** - otevírá cílový objekt zařízení
- **IRP_MJ_READ** - přenáší data ze zařízení
- **IRP_MJ_WRITE** - přenáší data do zařízení
- **IRP_MJ_DEVICE_CONTROL** - nastavuje nebo resetuje zařízení, podle systémem definovaných, specifických I/O kontrolních kódů (IOCTL)
- **IRP_MJ_CLOSE** - zavře cílový objekt zařízení
- **IRP_MJ_PNP** - provede Plug and Play operaci na zařízení, IRP_MJ_PNP je zasílané PnP manažerem přes I/O manažer
- **IRP_MJ_POWER** - provede power operaci na zařízení, IRP_MJ_POWER zasláno power manažerem přes I/O manažer

IRP jako zásobník nezávislých vláken s požadavky

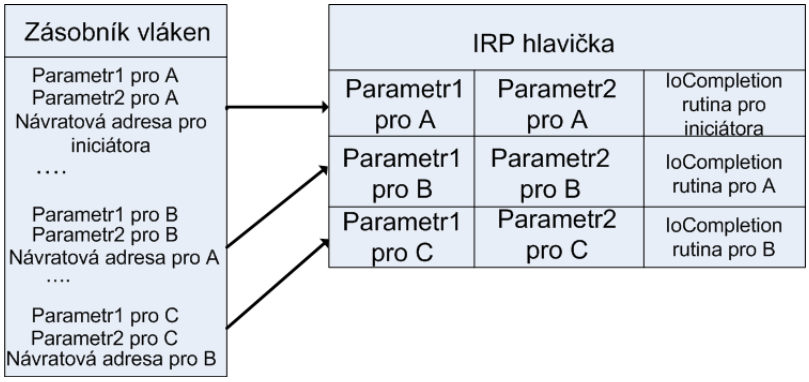
Provedení I/O operací obvykle vyžaduje více než jeden ovladač. Každý ovladač vytváří pro zařízení již zmiňovaný objekt zařízení a tyto objekty jsou organizovány hierarchicky do tzv. zásobníku zařízení. IRP je zasíláno postupně od jednoho ovladače k dalšímu. Pro každý ovladač ze zásobníku obsahuje IRP ukazatel na strukturu `IO_STACK_LOCATION`. Protože ovladače zpracovávají požadavky asynchronně, IRP se dá přirovnat k zásobníku vláken, jak můžeme vidět na obrázku 3.8.

Na levé straně obrázku 3.8 zásobník vláken zobrazuje, jak mohou být parametry a návratové hodnoty pro ovladače A, B, C organizovány do zásobníku. Na pravé straně obrázku vidíme, jak tyto parametry a návratové adresy odpovídají struktuře `IO_STACK_LOCATION` v IRP.

Přeposlání IRP dalšímu ovladači fronty

Jakmile zpracovávající rutina ovladače obdrží IRP, musí zavolat funkci `IoGetCurrentIrpStackLocation`, čímž získá strukturu `IO_STACK_LOCATION`, a může zkontrolovat, zda jsou všechny parametry správné. Pokud ovladač není schopen požadavek obsloužit a dokončit sám, může udělat jednu z následujících možností:

- Poslat IRP na zpracování dalším (nižším) ovladačům ke zpracování
- Vytvořit jedno nebo více nových IRP a poslat je nižším ovladačům



Obrázek 3.8: IRP jako zásobník vláken

Dokončení IRP

Ovladač docílí voláním funkce `IoCompleteRequest` dokončení IRP, jestliže je splněna jedna z následujících podmínek:

- Ovladač zjistí, že kvůli chybným parametrům nemůže pokračovat zpracovávání IRP
- Ovladač je schopný zpracovat požadovanou I/O operaci bez toho, aby přeposílal IRP dalším ovladačům ve frontě (stav IRP je `STATUS_SUCCESS`)
- IRP bylo zrušeno (stav IRP je `STATUS_CANCELLED`)

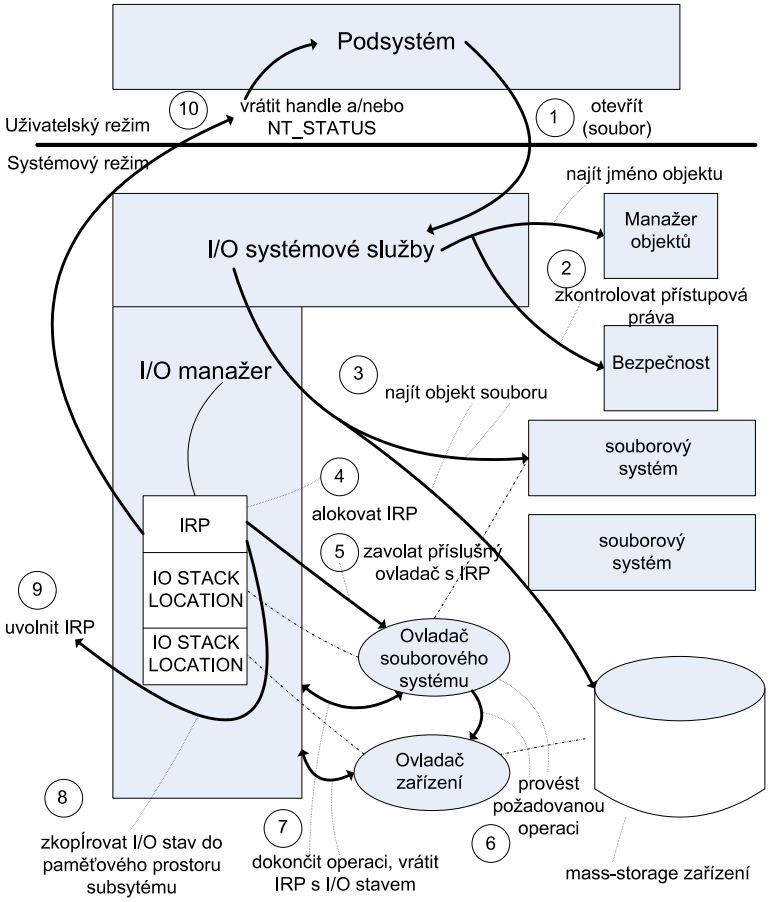
Rušení IRP

Ovladače, ve kterých mohou IRPs zůstat ve frontě libovolně dlouhou dobu (např. uživatel může zrušit zaslaný I/O požadavek) musí mít jednu nebo více tzv. **Cancel** rutin ke korektnímu dokončení uživatelem zrušených I/O požadavků. Zavolání Cancel rutiny je uskutečněno systémovou funkcí `IoCancelIrp`

Příklad zpracování IRP

Obrázek 3.9 ukazuje co se stane, jestliže podsystém OS otevírá objekt reprezentující datový soubor pro aplikaci. Podrobnější popis - otevírání souboru:

1. Podsystém zavolá systémovou I/O službu k otevření pojmenovaného souboru.
2. I/O manažer zavolá manažera objektů k vyhledání pojmenovaného objektu a také zkontroluje, zda má subsystém příslušná přístupová práva k otevření souboru.
3. Kontrola, zda je připojen souborový systém, v opačném případě dočasné přerušení požadavku na otevření souboru, postupné volání jednotlivých souborových systému ke zjištění, na kterém disku je objekt uložen, poté připojení souborového systému a obnovení požadavku.
4. I/O manažer alokuje paměť pro inicializaci IRP, otevření souboru je ekvivalentní primárnímu kódu IRP_MJ_CREATE.



Obrázek 3.9: Příklad na I/O požadavek

5. I/O manažer zavolá ovladač souborového systému s IRP parametrem. Ovladač přistoupí ke struktuře `IO_STACK_LOCATION` v IRP ke zjištění, jakou operaci má provést, zkontroluje parametry. Poté zjistí, jestli je požadovaný soubor v cache paměti, pokud ne, vyvolá nižší ovladač pro načtení souboru z disku do cache.

6. Oba ovladače zpracují IRP a ukončí požadovanou I/O operaci.

7. Ovladač vrátí IRP I/O manažeru se vstupně/výstupním stavem uloženém uvnitř IRP k indikaci, zda operace proběhla v pořádku či ne.

8. I/O manažer získá vstupně/výstupní stav z IRP a vrátí stavovou informaci přes podsystém volajícímu.

9. I/O manažer uvolní dokončený IRP.

10. I/O manažer vrátí ukazatel na soubor podsystému, jestliže operace byla úspěšná. Pokud se vyskytla chyba, vrátí příslušný chybový status.

3.2 Ovladače ve Windows CE

Stručně popíšeme vlastnosti ovladačů na operačním systému Windows CE, přístup k nim a nejpoužívanější model pro tvorbu ovladačů - stream interface model.

3.2.1 Úvod do ovladačů ve Windows CE

Nahrávání ovladačů ve Windows CE řídí tři hlavní procesy. Prvním procesem, který spouští jádro je `FileSys.exe`, proces souborového systému, který nahrává ovladače souborového systému. Po načtení souborového systému a registrů operačního systému je spuštěn proces s názvem „správce zařízení“ (device manager) reprezentovaný souborem `device.exe`. Tento proces načítá většinu ovladačů zařízení v systému (USB, NDIS, baterie, sériové ovladače, atd.). Nakonec je spuštěn proces pro grafiku, okna a události `GWES.exe`, který nahraje specializované ovladače pro displej a klávesnici.

Ve Windows CE 5 a dřívějších běžely všechny ovladače v rámci uživatelského procesu `device.exe`. Tento proces se choval jako kterákoli jiná aplikace se stejným paměťovým omezením a nižším výkonem vycházejícím z meziprocesových volání mezi aplikacemi, ovladačem, souborovým systémem a jádrem. Navíc, protože byly všechny ovladače zavedeny v rámci jednoho procesu, chybná funkce jednoho z nich mohla způsobit pád správce zařízení `device.exe`, což mělo za následek pád všech ostatních ovladačů.

Nová architektura jádra Windows CE 6 poskytuje větší flexibilitu poskytnutím dvou různých ovladačových modelů. Původní funkcionalita správce zařízení `device.exe` byla přesunuta do jádra operačního systému, a byl vytvořen nový uživatelský správce zařízení `udevice.exe`. Tím vznikla možnost nahrát ovladače v režimu jádra nebo v uživatelském režimu. Ovladače spuštěné v uživatelském režimu běží izolovaně od jádra a zbytku systému, pád ovladače způsobí pouze pád jedné z instancí procesu `udevice.exe`, a celý systém by měl zůstat neporušený. O způsobu, zda nahrát ovladač v uživatelském režimu nebo v režimu jádra rozhoduje pouze bitový příznak v registrech operačního systému.

Ovladače v režimu jádra

Ovladače v režimu jádra jsou nyní nahrávány modulem `device.dll`, který je výchozím způsobem pro nahrávání ovladačů, a podobá se nejvíce chování na Windows CE 5. Největší výhodou je výkon - tímto eliminujeme náročné meziprocesové volání. Pro uživatelské procesy není nutné před přístupem k ovladači přepínat kontext, jelikož nyní tyto procesy spolupracují s jádrem. Jádro také zahrnuje souborový systém a standardní ovladače zařízení. Je nutné ovšem klást vetší důraz na robustnost ovladačů zaváděných v režimu jádra, případný pád jednoho z nich způsobí pád celého jádra, a tím i operačního systému.

Ovladače v uživatelském režimu

Windows CE 6 poskytuje nový mechanismus pro nahrávání ovladačů do procesu běžícího v uživatelském režimu - `udevice.exe`. Nabízí větší systémovou stabilitu a bezpečnost (případný pád ovladače nemusí způsobit pád celého systému), ovšem za cenu nižšího výkonu. Ovladače v uživatelském režimu jsou až na velice malé rozdíly kompatibilní s ovladači v režimu jádra, a ovladač pro uživatelský režim může být beze změn nahrán do jádra systému.

3.2.2 Přístup k ovladačům zařízení

Většina ovladačů vystavuje operačnímu systému tzv. stream rozhraní (stream interface). Tyto ovladače, nazývané jako streamové ovladače (stream drivers), poskytují stejné vstupní funkce k ovladači bez ohledu na hardware, který řídí. V systému existuje několik nestreamových ovladačů, jako např. ovladač displeje, klávesnice, dotykové obrazovky mající jiné

rozhraní vzhledem k OS a jsou často nazývané jako nativní. Další informace o těchto ovladačích na [11].

Aplikace přistupují k ovladači zařízení ve Windows CE přes vstupně/výstupní funkce `CreateFile`, `ReadFile`, `WriteFile` a `CloseHandle`. Aplikace otevírá zařízení použitím `CreateFile` se jménem zařízení, které má pět znaků (tři znaky, číslo a dvojtečka, např. COM3:). Jakmile je zařízení otevřeno, mohou být data zasílaná funkcí `WriteFile` a čtena použitím `ReadFile`.

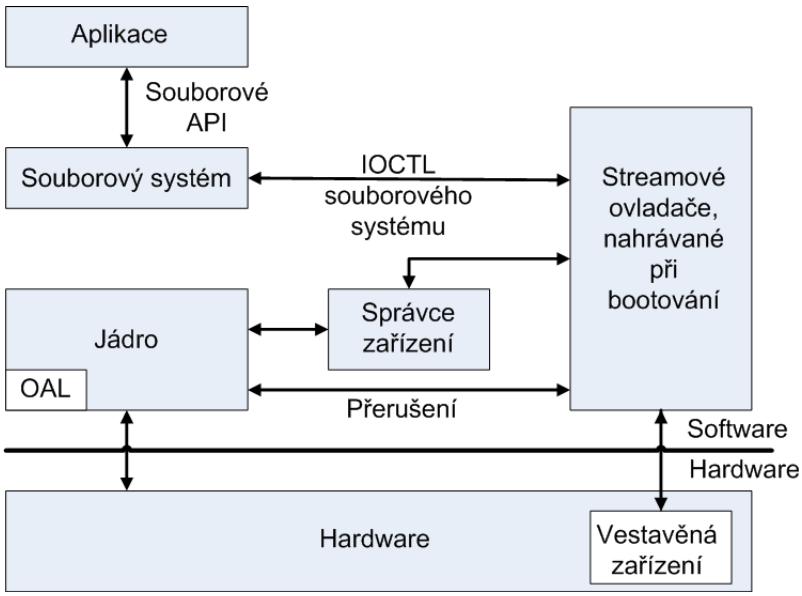
Zařízení je nastavováno a řízeno použitím funkce `DeviceIoControl`. Jedním z parametrů této funkce je tzv. **IOCTL kód**. Jeho hodnota definuje operaci, kterou chceme na zařízení provést. Každý ovladač má vlastní množinu IOCTL kódů.

3.2.3 Stream device drivers model

Nejpoužívanější ovladačový model pro Windows CE je streamový model (stream interface model). Tento model pochází z nejstarších implementací operačního systému Unix. Streamový ovladač exportuje funkce jako OPEN, CLOSE, READ, WRITE nebo CONTROL k řízení hardwaru, který je přiřazen k ovladači.

Streamový model je vhodný pro taková vstupně/výstupní zařízení, která se mohou chovat jako producent/konzument dat. Příkladem může být sériový port, síťová karta, USB port. Opačným příkladem je např. displej.

Streamový ovladač přijímá příkazy ze správce zařízení a z aplikací prostřednictvím systémových souborových volání. Všechny streamové ovladače, ať už vestavěné nebo nainstalované, nahrané při bootování systému nebo dynamicky, mají podobné chování ve vztahu k ostatním komponentám systému. Obrázek 3.10 znázorňuje vztah mezi jednotlivými částmi systému pro obecný streamový ovladač.



Obrázek 3.10: Architektura streamových ovladačů

Streamový ovladač vystavuje 12 vstupních funkcí, které správce zařízení volá pro komunikaci s ovladačem:

- **xxx_Init** - nahrávání instance ovladače
- **xxx_PreDeinit** - volána předtím, než je ovladač odstraněn z paměti
- **xxx_DeInit** - odstraňování ovladače z paměti
- **xxx_Open** - ovladač je otevírán z aplikace funkcí `CreateFile`
- **xxx_PreClose** - volána předtím, než se zavolá funkce xxx_CLOSE
- **xxx_Close** - aplikace volá funkci `CloseHandle`
- **xxx_Read** - aplikace volá funkci `ReadFile`
- **xxx_Write** - aplikace volá funkci `WriteFile`
- **xxx_Seek** - aplikace volá funkci `SetFilePointer`
- **xxx_IOControl** - aplikace volá funkci `DeviceIoControl`
- **xxx_PowerDown** - volána předtím, než je systém pozastaven
- **xxx_PowerUp** - volána předtím, než je systém znovu obnoven

Znaky „xxx“, které předchází každé jméno funkce jsou tříznakové jméno ovladače, pokud ovladač má jméno. Například, pro COM port je název funkce např. `COM_Init`, `COM_Deinit`, atd. Pro nepojmenované ovladače (bez prefixové hodnoty definované v registrech) se jména vstupních funkcí udávají bez prvních tří znaků, např. `Init`, `Deinit`, atd.

3.3 Síťové rozhraní Windows pro ovladače jádra

Jelikož je jednou z hlavních částí této práce návrh síťového ovladače, je nutné se seznámit s architekturou síťového rozhraní na operačních systémech Windows. Popíšeme vztah k referenčnímu modelu ISO OSI a význam rozhraní NDIS pro tvorbu ovladačů. Všechny následující informace čerpány ze zdroje [7].

3.3.1 Síťová architektura Windows

Síťová architektura operačních systémů Microsoft Windows je založená na sedmivrstvém referenčním modelu ISO OSI. Model popisuje síť jako soubor vrstev se specifickou množinou funkcí přidělených k jednotlivým vrstvám. Každá vrstva nabízí služby vyšším vrstvám a zároveň skrývá detaily, jak jsou služby implementovány.

Síťové ovladače systému Microsoft Windows implementují dolní 4 vrstvy OSI modelu:

- **Fyzická vrstva** - nejnižší vrstva OSI modelu, spravuje odesílání a přijímání toku nestrukturovaných bitů na fyzickém médiu. Popisuje elektrické, optické, mechanické a provozní rozhraní na fyzickém médiu. Ve Windows je fyzická vrstva implementována kartou síťového rozhraní (network interface card, NIC), jejím vysílačem/přijímačem a médiem, ke kterému je NIC připojená.
- **Linková vrstva** - rozdělena na dvě podvrstvy:

Obrázek 3.11: Referenční model ISO OSI

  - **LLC** - poskytuje bezchybový přenos rámců z jednoho uzlu do druhého, sestavuje a ukončuje logická spojení, řídí tok, pořadí, potvrzování a znovuposílání nepotvrzených rámců
  - **MAC** - spravuje přístup k fyzické vrstvě, kontroluje chyby rámců, a zařizuje rozpoznávání adres příchozích rámců

  Ve Windows je podvrstva LLC implementována transportními ovladači a podvrstva MAC kartou síťového rozhraní. NIC je řízen softwarovým ovladačem nazývaným ovladač miniportu (miniport driver). Windows podporuje několik variací miniport ovladačů zahrnující WDM miniport ovladače, miniport call managers (MCMs) a miniport intermediate ovladače.

- **Síťová vrstva** - síťová vrstva zabezpečuje adresování a směrování paketů v síti od zdroje k cíli přes několik mezilehlých prvků. Směrování může být vykonáváno dynamicky (datagramová služba) v závislosti na aktuálním stavu komunikačního systému nebo staticky, kdy se na začátku spojení vytvoří virtuální cesta přes mezilehlé prvky (spojově orientovaná služba).

- **Transportní vrstva** - přijímá data z relační vrstvy, rozkládá je na menší části - pakety a odevzdává je síťové vrstvě. Zabezpečuje, aby se všechny části zprávy dostaly správně k příjemci a byly ve správném pořadí. Vrstva vytváří síťová spojení, multiplexuje a demultiplexuje data mezi transportními spoji koncových procesů, sestavuje nebo ruší několik spojení současně. Alespoň minimální transportní vrstva je vyžadována zásobníkem protokolů, které zahrnují spolehlivou síťovou nebo LLC podvrstvu schopnou vytvářet virtuální okruhy. Jestliže zásobník protokolů nezahrnuje podvrstvu LLC a pokud síťová vrstva je nespolehlivá a/nebo podporuje datagramy, transportní vrstva by měla zajišťovat sekvencování a potvrzování rámců, stejně jako znovuposílání nepotvrzených rámců. V síťové architektuře Windows je LLC, síťová a transportní vrstva implementována softwarovými ovladači nazývanými protokolové ovladače (protocol drivers), nazývané také jako transportní ovladače (transport drivers).

3.3.2 Architektura NDIS ovladačů

Specifikace ovladačů síťového rozhraní (Network driver interface specification, NDIS) je knihovna, která odděluje síťový hardware od síťových ovladačů. NDIS také specifikuje standardní rozhraní mezi vrstvenými síťovými ovladači, čímž abstrahuje ovladače nižší úrovně (lower level), které řídí hardware, od ovladačů vyšší úrovně, jako transportní ovladače. NDIS také udržuje stavové informace a parametry síťových ovladačů, což zahrnuje ukazatele na funkce, handle na objekty zařízení, parametry a ostatní systémové hodnoty. NDIS podporuje následující typy síťových ovladačů:

- Miniport ovladače (miniport drivers)
- Intermediate ovladače (intermediate drivers)
- Protokolové ovladače (protocol drivers)

V nejnovější verzi NDIS 6.0 přibývají filtrovací ovladače (filter drivers), ovšem tuto verzi podporuje pouze operační systém Windows Vista. S ohledem na implementační prostředí Windows XP a Windows CE Embedded 6.0 budeme popisovat verzi NDIS 5.1, která je podporována v obou těchto systémech.

Miniport ovladače

Miniport ovladače mají dvě základní funkce:

1. Spravují síťové adaptéry (NIC), tzn. zasílání a příjem dat přes NIC
2. Propojují NIC s ovladači vyšších vrstev (intermediate a protokolové ovladače)

Miniport ovladač komunikuje s NIC a ovladači vyšší úrovně přes knihovnu NDIS. Ta exportuje množinu funkcí (`NdisMXxx` a jiné `NdisXxx` funkce), které zapouzdřují všechny funkce operačního systému, které musí miniport ovladač volat. Naopak, miniport ovladač musí exportovat množinu vstupních bodů (`MiniportXxx` funkce), které NDIS volá pro vlastní účely nebo v zastoupení ovladačů vyšší úrovně, které chtějí přistoupit k miniport ovladači.

Komunikace miniport ovladače s ovladači vyšší úrovně při zasílání a přijímání dat vypadá následovně:

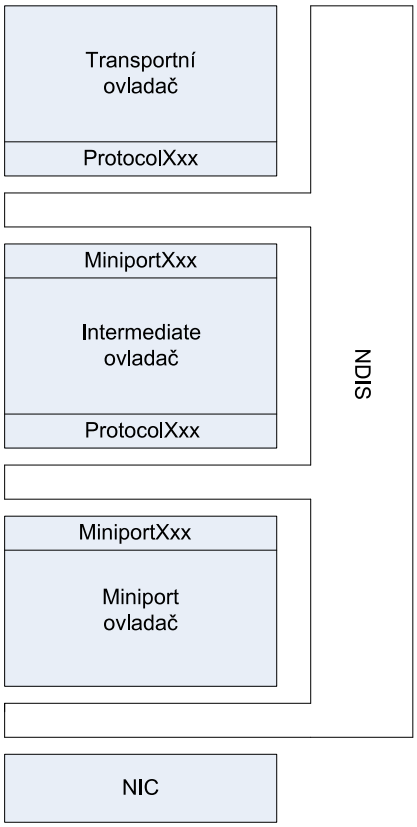
Jestliže transportní ovladač vlastní paket k zaslání, zavolá `NdisXxx` funkci exportovanou knihovnou NDIS. NDIS poté pošle paket k ovladači miniportu voláním příslušné `MiniportXxx` funkce exportované miniport ovladačem. Ten potom paket přepošle k NIC voláním `NdisXxx` funkcí.

Když NIC přijme paket, vyvolá přerušení, které je zpracováno NDIS rozhraním nebo miniport ovladačem asociovaném k NIC. NDIS upozorní miniport ovladač voláním příslušné `MiniportXxx` funkce. Miniport ovladač vyvolá přenos dat z NIC a indikuje přítomnost příchozího paketu pro ovladače vyšších vrstev voláním `NdisXxx` funkce.

NDIS podporuje ovladače miniportu pro spojovaná i nespojovaná prostředí. Nespojované ovladače řídí NIC pro nespojovaná síťová média, jako např. Ethernet, spojované ovladače řídí NIC pro spojovaná síťová média, jako např. ISDN.

Intermediate ovladače

Jak můžeme vidět na obrázku 3.12, intermediate ovladače jsou typicky umístěné mezi miniport a protokolovými ovladači.



Obrázek 3.12: Architektura NDIS ovladačů

Intermediate ovladač musí komunikovat s oběma typy ovladačů - horními protokolovými a dolními miniportovými. K tomu musí exportovat dva typy funkcí:

- **Protokolové vstupní funkce** - na spodní hraně, NDIS volá `ProtocolXxx` funkce pro požadavky komunikaci s dolními miniport ovladači. Intermediate ovladač se tváří pro miniport ovladač jako protokolový ovladač.
- **Miniport vstupní funkce** - na horní hraně, NDIS volá `MiniportXxx` funkce pro požadavky na komunikaci od jednoho nebo více protokolových ovladačů. Intermediate ovladač se tváří pro protokolový ovladač jako miniport ovladač.

Intermediate ovladač exportuje podmnožinu `MiniportXxx` funkcí na horní hranu a také exportuje jeden nebo více virtuálních síťových adaptérů, ke kterým se mohou protokolové ovladače navázat. Protokolovému ovladači se takový virtuální adaptér tváří jako fyzický NIC. Když protokolový ovladač zašle pakety nebo požadavky virtuálnímu adaptéru, intermediate ovladač přepošle tyto pakety nebo požadavky k miniport ovladači. Stejně tak to platí pro opačný směr, od miniport ovladače, přes intermediate až k protokolovému ovladači.

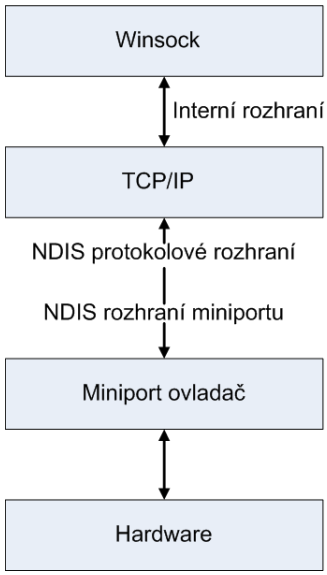
Intermediate ovladače mohou být použity k převodu různých síťových médií, vyvažování přenosu paketů mezi více než jedním NIC, případně k provádění speciálních funkcí, jako např. šifrování a dešifrování dat.

Protokolové ovladače

Protokolové ovladače jsou nejvyššími ovladači v hierarchii NDIS. Často se používají jako nejnižší ovladače v transportním ovladači, který implementuje transportní protokolový zásobník, jako např. TCP/IP nebo IPX/SPX. Transportní protokolový ovladač alokuje pakety, kopíruje data ze zasílající aplikace do paketu a posílá pakety k nižším ovladačům voláním NDIS funkcí. Protokolový ovladač také poskytuje protokolové rozhraní pro přijímání příchozích paketů z nejbližšího nižšího ovladače. Transportní protokolový ovladač přenáší data k příslušné klientské aplikaci.

Na spodní hraně, protokolový ovladač sousedí s intermediate a miniport ovladači. Protokolový ovladač volá `NdisXxx` funkce k zasílání paketů, čtení a nastavování informací, které jsou prováděny nižšími ovladači a používány službami operačního systému. Protokolový ovladač také exportuje množinu vstupních funkcí (`ProtocolXxx` funkce), které NDIS volá pro vlastní účely nebo v zastoupení nižších ovladačů k indikaci příchozích paketů, stavu nižších ovladačů a s ostatní komunikací s protokolovým ovladačem.

Na horní hraně, transportní ovladač má soukromé rozhraní k vyšším ovladačům v protokolovém zásobníku.



Obrázek 3.13: Architektura NDIS protokolového ovladače

3.4 Struktura OpenVPN

Strukturou OpenVPN je myšlen popis z hlediska komunikace této aplikace s okolím, tedy síťovými ovladači, a to jak fyzickými, tak virtuálními. Začneme obecným popisem TUN/-TAP rozhraní - funkcionalitou a využitím, zmíníme rozdíly mezi implementací TUN/TAP na Unixových a Windows systémech. Potom již budeme mít dostatek informací k popisu obecné komunikace OpenVPN se síťovými adaptéry, obzvláště TUN a TAP, detailně zmíníme vnitřní chování pří zápisu a čtení na tato, a z těchto zařízení.

3.4.1 Síťové rozhraní TUN/TAP

Po obecném popisu, co je TUN a TAP, bude vysvětlen rozdíl v implementacích těchto zařízení na systémech Windows a Unix.

TUN/TAP obecně

TUN a TAP jsou virtuální síťové ovladače. Implementují síťová zařízení, která jsou postavena pouze na softwarovém základu, což se liší od běžných síťových zařízení, která potřebují pro svou činnost fyzické síťové adaptéry (hardware), viz. [5]. Ovladače TUN a TAP jsou zahrnuty ve všech moderních Linux/Unixových distribucích, včetně dodatečných implementací na operačních systémech Windows a Mac OS X.

TAP simuluje ethernetové zařízení a pracuje na 2. vrstvě OSI modelu s ethernetovými rámci. Tento režim je nazýván mostem (bridge), protože jednotlivé sítě jsou propojeny jako přes hardwarový most. Ovladač TAP je implementován v režimu jádra.

TUN (ze slova TUNel) simuluje zařízení 3. (síťové) vrstvy a pracuje s IP pakety. Je použit jako virtuální point-to-point rozhraní, stejně jako modem nebo DSL linka. Režim činnosti je nazýván směrovací (routed) nebo též tunelový (abstrakce tunelu). Ovladač TUN je implementován v uživatelském režimu (v případě operačního systému Windows a využívá funkcionalitu zařízení TAP) nebo v režimu jádra (na Unixu). Dále budeme ovladače TUN a TAP nazývat společným jménem TUN/TAP, které značí obecně používané pojmenování pro virtuální síťové zařízení.

Pakety zasílané operačním systémem z privátní sítě přes TUN/TAP jsou doručovány do uživatelského prostoru aplikace, která je k zařízení připojena. Program běžící v uživatelském režimu může také posílat pakety do privátní sítě přes TUN/TAP. V takovém případě zařízení TUN/TAP předává tyto pakety síťovému zásobníku operačního systému (network stack), a tím simuluje jejich příjem z externího zdroje (tedy simulace příjmu z fyzického síťového rozhraní).

TUN/TAP na Unixu

Na operačních systémech Linux, FreeBSD a Solaris je implementace TUN a TAP zařízení odlišná oproti OS Windows. Operační systémy na bázi Unix mají podporu TUN a TAP zařízení implementovanou v jádře (`\dev\tun`, `\dev\tap`). V OpenVPN existuje struktura zapouzdřující obě tato zařízení, která se chovají jako běžný soubor reprezentovaný popisovačem (file descriptor). Po spuštění OpenVPN se nastaví, v jakém režimu má virtuální síťový ovladač pracovat, dle toho se vytvoří příslušná instance zařízení (TUN nebo TAP), a může probíhat komunikace.

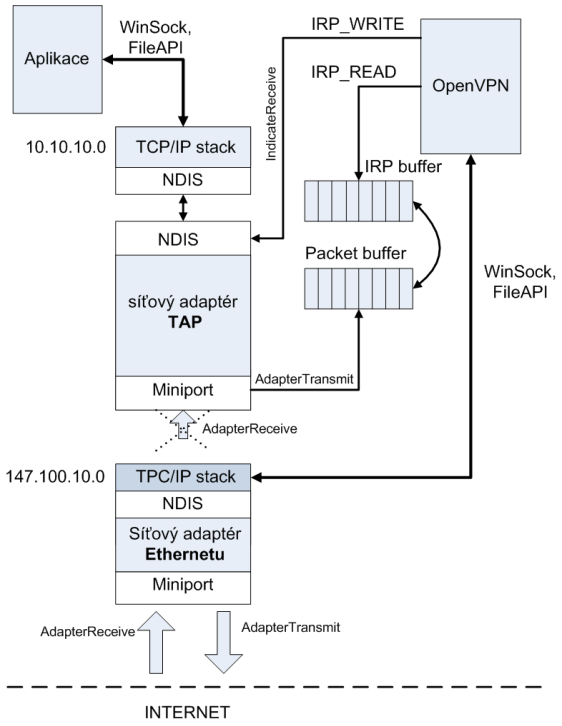
TUN/TAP na Windows

Hlavním rozdílem oproti Unixu je fakt, že operační systémy Windows nemají vnitřně podporu TUN/TAP, a proto je nutné tato zařízení implementovat. TAP se chová jako běžný síťový ovladač běžící v režimu jádra a využívá standardní rozhraní OS Windows pro tvorbu ovladačů, jako WDM a NDIS. Implementace je v případě TUN výrazně odlišná. Zjednodušeně se dá říct, že TUN je pouze jakási obálka (wrapper) pro OpenVPN, která vnitřně používá TAP zařízení. Důvodem je, že implementace ovladače TAP zařízení je na OS Windows koncipována tak, aby mohla pracovat jak v režimu most (na 2. vrstvě OSI modelu), tak v režimu tunelovém (na 3. vrstvě OSI modelu). TUN tedy není klasický síťový ovladač,

tedy zařízení, které se objeví v systému samostatně, ale pouze prostředek OpenVPN pro ovládání TAP. Nutno poznamenat, že pro aplikaci OpenVPN se tento přístup jeví jako zcela transparentní; až vnitřní implementace TUN určí, zda čteme na otevřeném popisovači `\dev\tun` (v případě Unixu) nebo na otevřeném handlu TAP zařízení (v případě Windows). Ačkoli se tedy TUN nevyskytuje v systému jako samostatné zařízení, budeme dále v textu z principu funkce TUN takto nazývat. S TAP zařízením lze také pracovat samostatně, a využívat jej v jiných aplikacích.

3.4.2 Obecné schéma komunikace v OpenVPN

OpenVPN naslouchá na TUN/TAP zařízení, zachytává přenos, šifruje data a posílá je přes fyzické síťové rozhraní k druhé VPN stanici. Tam jiný OpenVPN proces data přijímá, dešifruje a předává je virtuálnímu síťovému zařízení. Zjednodušené principiální znázornění komunikace OpenVPN a síťových zařízení je na obrázku 3.14.



Obrázek 3.14: Obecné schéma komunikace OpenVPN se síťovými zařízeními

Komunikace s ovladačem TAP

TAP jsou dvě zařízení v jednom. Prvním je NDIS miniport ovladač a druhé běžné streamové zařízení, ze kterého lze číst funkcí `ReadFile` a zapisovat do něj funkcí `WriteFile`. Síťový zásobník (network stack) využívá první rozhraní, aplikace, tedy i OpenVPN, druhé.

TAP lze z uživatelského režimu otevřít, číst a zapisovat. Pokud chce aplikace zaslat data na síťové rozhraní TAP, zavolá funkci `WriteFile` (vytvoření IRP s požadavkem zápisu), která vytvoří paket, a zašle jej pomocí příslušné funkce knihovny NDIS do vnitřní (privátní) sítě. Jestliže chce aplikace získat paket ze síťového zařízení (požadavek na čtení), je situace

složitější. Voláním funkce `ReadFile` se vytvoří IRP požadavek se žádostí o čtení. Pokud se nachází v tzv. packet bufferu (fronta nezpracovaných paketů) dříve zaslaný paket z vnitřní sítě, je spojen s požadavkem na čtení, z fronty vyjmut, a celá procedura skončí. Za situace, kdy je packet buffer prázdný, požadavek na čtení je uložen do tzv. IRP bufferu (fronta požadavků na čtení), a vyřízen, jakmile přijde z vnitřní sítě paket na TAP zařízení.

Komunikace s ovladačem TUN

Jak již bylo řečeno, TUN je pouze prostředník, přes který OpenVPN pracuje s TAP zařízením. TUN po spuštění OpenVPN zavolá funkci `open_tun`, která otevře pomocí `CreateFile` TAP zařízení, nastaví režim činnosti (most, tunel) a přidělí IP adresu (přes DHCP nebo manuálně). Poté OpenVPN volá funkce TUN `read_tun` (`write_tun`), které vnitřně vytvoří požadavek na čtení (zápis) na TAP zařízení (funkcemi `ReadFile` nebo `WriteFile`). Po ukončení aplikace funkce `close_tun` odebere IP adresu TAP adaptéru, zruší všechny nezpracované I/O požadavky, a zařízení uzavře funkcí `CloseHandle`.

Čtení a zápis využívá tzv. overlapped I/O, což jsou asynchronní požadavky na čtení a zápis. Po zavolání funkce `ReadFile` nebo `WriteFile` nemusí aplikace čekat na dokončení vstupně/výstupního požadavku, ale pokračuje dále v běhu, pozdější zpracování I/O požadavku je indikováno asynchronně. Bližší popis lze nalézt v sekci 3.4.4.

3.4.3 TAP - komunikace s OpenVPN

Komunikaci rozdělíme na dvě části, nejprve zápis na TAP, poté čtení z TAP.

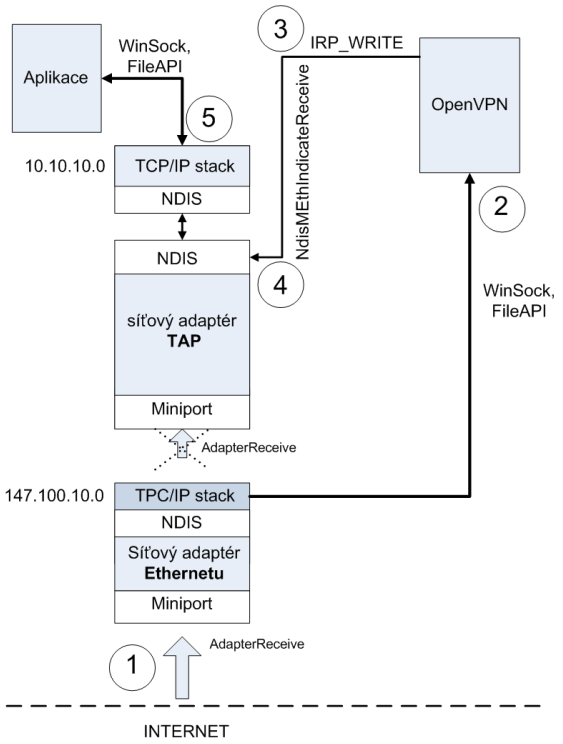
Zápis na TAP zařízení

Nyní si detailně popíšeme, jak probíhá zápis dat na síťový adaptér TAP. OpenVPN funkcí `CreateFile` vytvoří instanci TAP zařízení, která se pro něj jeví jako běžný soubor. Z vnějšku (1) (druhá strana privátní sítě) přijde do aplikace požadavek na zaslání paketu do privátní sítě (2). OpenVPN zavolá funkci `WriteFile` s parametrem instance otevřeného TAP adaptéru, a jako vstupní data vloží buffer obsahující ethernetový rámec (režim mostu) nebo IP paket (režim tunelu). Voláním funkce `WriteFile` se v systému vytvoří IRP požadavek s primárním kódem `IRP_MJ_WRITE` (3). V obslužné rutině tohoto IRP je zavolána funkce rozhraní NDIS `NdisMEthIndicateReceive` (4), která zašle paket do privátní sítě (přes network stack) (5). Tím simulujeme neexistující fyzické rozhraní virtuálního TAP adaptéru pro **příjem**, tedy funkci `AdapterReceive`, která by v běžném případě zpracovávala příchozí pakety na adaptér, a zasílala je do privátní sítě.

Požadavek na odeslání paketu z privátní sítě obsluhuje standardně funkce `AdapterTransmit` (volaná network stackem), ovšem místo přeposílání na fyzické rozhraní (**vysílání**) ukládá paket do tzv. packet bufferu, pokud zatím nikdo z druhého konce privátní sítě nezaslal požadavek na čtení. V opačném případě se ve funkci `AdapterTransmit` tento požadavek zpracuje a dokončí. Celý postup je znázorněn na obrázku 3.15

Čtení z TAP zařízení

Jak bylo nastíněno v sekci 3.4.2, čtení - tedy získání paketů z virtuálního TAP adaptéru - je komplikovanější. Na úvod platí vše, co pro zápis. Voláním standardní funkce systému `CreateFile` vytvoříme instanci TAP adaptéru, na které voláme funkci `ReadFile` pro čtení



Obrázek 3.15: Komunikace OpenVPN s TAP - zápis

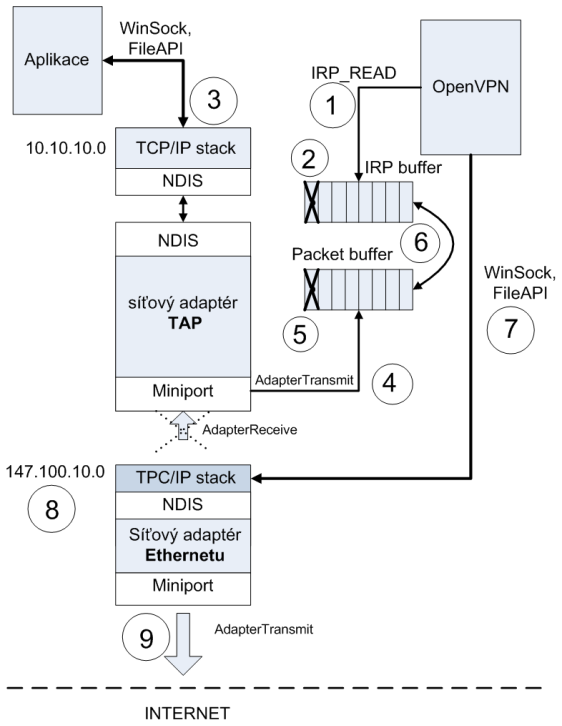
dat (získání datagramu nebo paketu, dle režimu funkce TAP adaptéru) (1). Tím se interně vytvoří IRP požadavek s primárním kódem `IRP_MJ_READ`.

Rutina ovladače `TapDeviceHook` zpracovávající IRP dělá následující: požadavek na čtení může být uspokojen jen tehdy, pokud existuje ve frontě paketů (packet buffer) nějaký paket, dříve zaslaný z privátní sítě (funkcí `AdapterTransmit`). Pokud ano, je požadavek na čtení spojen s paketem čekajícím na odeslání. Celá procedura tímto končí. Funkce `ReadFile` v aplikaci OpenVPN vrátí výsledek - paket, který si OpenVPN zpracuje, zašifruje a zašle přes fyzické síťové rozhraní na druhý konec privátní sítě.

V případě, že packet buffer je prázdný, musíme IRP požadavek na čtení uložit do tzv. IRP bufferu pro pozdější zpracování (2). Pozdějším zpracováním je myšlena pouze jediná situace, a to interní volání funkce NDIS - `AdapterTransmit` (voláno network stackem). Jak již bylo řečeno u popisu zápisu na TAP zařízení, funkce `AdapterTrasmit` (4) je volána v případě, kdy vnitřní síť chce odeslat paket (3). Tedy, pokud existuje v IRP bufferu nějaký dříve uložený požadavek na čtení, je spojen s aktuálně odesílaným paketem (6). Pokud v IRP bufferu existuje alespoň jeden další požadavek na čtení, a zároveň v packet bufferu dosud neodeslaný paket, jsou postupně tyto položky front spojovány. Celá procedura probíhá do okamžiku, kdy je alespoň jeden z bufferů prázdný. Vše výše zmíněné popisuje obrázek 3.16

3.4.4 TUN - komunikace s OpenVPN

Komunikaci zařízení TUN popíšeme stejným stylem jako TAP (čtení, zápis), navíc přidáme nezbytné informace o asynchronních požadavcích, na kterých je interakce TUN a systému postavena.



Obrázek 3.16: Komunikace OpenVPN s TAP - čtení

Asynchronní komunikace v TUN

Pro účely rychlejší komunikace TUN s operačním systémem byla vytvořena struktura s názvem `overlapped_io`, která zapouzdřuje asynchronní požadavek na čtení nebo zápis. Požadavek se může nacházet ve třech různých stavech:

1. `IOSTATE_INITIAL` - počáteční stav, bez požadavku na I/O operaci
2. `IOSTATE_QUEUED` - byl vystaven požadavek na čtení nebo zápis, ale nemohl být uskutečněn okamžitě, zpracování bude probíhat asynchronně
3. `IOSTATE_IMMEDIATE_RETURN` - byl vystaven požadavek na I/O operaci, který se okamžitě zpracoval a vrátil výsledek, nebo se vyskytla chyba

Pro pochopení činnosti TUN je nutné zmínit standardní objekt operačního systému Windows s názvem `OVERLAPPED`, nacházející se jako jedna z vlastností struktury `overlapped_io`, který obsahuje handle na tzv. objekt události (event object). Objekt události je synchronizační objekt, který aplikace používají pro signalizaci výskytu události čekajícímu vláknu. Objekt se může nacházet ve dvou stavech - signalizovaném a resetovaném. Objekt události je vytvoření funkcí `CreateEvent`, kde jeden z parametrů určuje počáteční stav (signalizován, resetován). Poté lze manuálně nastavit resetovaný stav funkcí `ResetEvent` a signalizovaný stav funkcí `SetEvent`. Více informací o objektech události lze najít v [8].

Zápis na TUN zařízení

Pokud chce OpenVPN poslat TUN adaptéru data, využívá k tomu funkci `write_tun_buffered`. Ta zkontroluje, zda je aktivní nějaký dříve zadaný vstupně/výstupní požadavek na

zápis. Jestliže ano, snaží se jej dokončit (`tun_finalize`). Následuje volání funkce `tun_write_queue`.

Zápis na TUN proběhne ve `tun_write_queue` pouze v případě, když se asynchronní požadavek na zápis nachází ve stavu `IOSTATE_INITIAL`. Poté proběhne inicializace bufferu pro zápis, nastavení objektu události do resetovaného stavu (`ResetEvent`), a samotné volání `WriteFile` s parametrem `overlapped` na TAP zařízení. Objekt `overlapped` se nachází ve výše popisované struktuře `overlapped_io` a obsahuje resetovaný objekt události.

Nyní testujeme vrácenou hodnotu po návratu z funkce `WriteFile`. Jestliže skončila úspěšně, znamená to změnu stavu asynchronního požadavku na `IOSTATE_IMMEDIATE_RETURN`, a ruční nastavení objektu události do signalizovaného stavu (`SetEvent`). Pokud vrácená hodnota signalizuje neúspěch, znamená to zařazení asynchronního požadavku do fronty (`IOSTATE_QUEUED`), nebo chybu při zápisu na TAP zařízení (změníme stav na `IOSTATE_IMMEDIATE_RETURN`, objekt události do signalizovaného stavu a uložíme chybový kód).

Asynchronní požadavek je kompletně zpracován až funkcí `tun_finalize`. Ta testuje jednotlivé stavy:

- `IOSTATE_QUEUED` - zavolá funkci `GetOverlappedResult` pro vyzvednutí výsledku asynchronní operace s parametrem okamžitého návratu (teoreticky může též čekat na dokončení), v případě úspěšného návratu je asynchronní požadavek převeden do stavu `IOSTATE_INITIAL`, objekt události resetován a příslušné buffery naplněny daty. V případě neúspěšného návratu a výskytu chyby je provedena stejná činnost (změna stavu na `IOSTATE_INITIAL`, resetování objektu události). Neúspěšný návrat a kód chyby `ERROR_IO_INCOMPLETE` znamená, že požadavek stále není zpracován, a čeká se asynchronně dále.

- `IOSTATE_IMMEDIATE_RETURN` - objekt požadavku na I/O je převeden do stavu `IOSTATE_INITIAL`, objekt události resetován. Pokud bylo stavu `IOSTATE_IMMEDIATE_RETURN` docíleno chybou, je tato chyba indikována (`SetLastError`), v opačném případě (úspěšném) přenos načtených/zapsaných dat do bufferů.

- `IOSTATE_INITIAL` - v okamžiku, kdy je objekt asynchronního požadavku v tomto stavu by se funkce `tun_finalize` neměla nikdy vyvolat, a je indikována chyba (`SetLastError`).

Čtení z TUN zařízení

V případě iniciování požadavku na čtení z TUN je situace složitější, ovšem pro pochopení bude dostačovat nepatrně zjednodušený popis. Pokud chce OpenVPN číst z adaptéru data, využívá k tomu funkci `read_tun_buffered`, která volá pouze funkci `tun_finalize` (zpracování všech dříve vystavených asynchronních požadavků na čtení). Aby došlo k samotnému čtení pomocí funkce `tun_read_queue`, musí OpenVPN uskutečnit volání `tun_set` (předchází nastavení vnitřních událostí, souvisí s interní funkcionalitou OpenVPN).

Principiálně je ovšem samotné čtení z `tun_read_queue` naprosto analogické funkci `tun_write_queue` popsané v předchozí části s tím rozdílem, že místo volání `WriteFile` je volána funkce `ReadFile` na TAP adaptéru. Také zpracování asynchronního požadavku je uskutečněno ve funkci `tun_finalize`, která má stejné chování.

3.4.5 OpenVPN - interní komunikace

Vzhledem k obsáhlosti samotného ovladače TUN/TAP uvedeme pouze stručný, a velice zjednodušený popis interní komunikace OpenVPN s okolím.

Celé jádro OpenVPN je založeno na zpracování událostí (events). Po spuštění aplikace se vytvoří hlavní výkonná smyčka, která provádí následující činnosti:

```
while(true) {
  - nastavení vnitřních časovačů smyčky událostí;
  - nastavení událostí a čekání na události;
  - zpracování dokončené události;
  if (příchod signálu)
    break;
}
```

Zpracovávaných událostí programem OpenVPN je velké množství, zaměříme se pouze na ty, které se týkají I/O komunikace se sokety a TUN/TAP. I/O události mohou být dvojího druhu:

- **Rychlé** - pro veškerý zápis, a to buď na soket nebo TUN/TAP (bez nutnosti čekat)
- **Pomalé** - pro čtení ze soketu nebo TUN/TAP, kde nemáme jistotu, že bude paket okamžitě k dispozici

Ve smyčce čekání na události mohou nastat následující případy:

1. **TCP/UDP port získal data pro čtení** - vytvoří se rychlá událost pro zápis na TUN/TAP
2. **TCP/UDP port chce zapsat data** - vytvoří se pomalá událost se žádostí o čtení na TUN/TAP
3. **TUN získal data pro čtení** - vytvoří se rychlá událost pro zápis na soket
4. **TUN chce zapsat data** - vytvoří se pomalá událost se žádostí o čtení na soket
5. **Přišel signál** - zpracování signálu dle typu
6. **Vypršení události** - smazání události z fronty nezpracovaných událostí

Podrobněji popíšeme případ **TCP/UDP port chce zapsat data**, který je ve schématu komunikace OpenVPN - TUN/TAP nejsložitější, a pro pozdější návrh řešení nejpodstatnější. Princip je velice podobný případu **TUN chce zapsat data**.

1. Vnější síť vytvoří požadavek na čtení z vnitřní (privátní) sítě, což znamená zápis dat z TUN/TAP na TCP/UDP port.
2. Vytvoření pomalé události se žádostí o čtení na TUN/TAP - funkce `tun_set`, viz. sekce 3.4.4
3. Převod asynchronního požadavku do stavu `IOSTATE_IMMEDIATE_RETURN` nebo čekajícího `IOSTATE_QUEUED`

4. Nyní se čeká na dokončení operace čtení na TUN, což v praxi znamená čekání na nastavení objektu události do stavu signalizován (ovladačem TAP, handle `hEvent` ve struktuře `OVERLAPPED`). Čekání je neblokující, a současně může probíhat další zpracování událostí v OpenVPN.

5. Jakmile TAP získá data z vnitřní sítě, automaticky nastaví objekt události do stavu signalizován, a smyčka zpracovávání událostí v OpenVPN zaregistruje tuto změnu.

6. Zpracuje se dokončená událost, tedy dokončení asynchronní operace funkcí `tun_finalize`, a proběhne zápis paketu na soket (TCP/UDP port). Poté TCP/IP stack provede odeslání do vnější sítě (protějšku se spuštěnou instancí OpenVPN).

3.5 Popis implementace ovladače TAP

V této sekci si stručně popíšeme implementaci TUN/TAP zařízení na operačním systému Windows. Nejprve softwarové struktury (objekty) zapouzdřující TAP zařízení, dále funkce využívané rozhraním NDIS, WDM a doplňkové funkce.

3.5.1 Softwarové struktury ovladače

Zařízení TAP reprezentují zejména dvě struktury:

- **TapAdapter** - zapouzdřuje vlastnosti obecného síťového ovladače
- **TapExtension** - zapouzdřuje vlastnosti specifické pro TAP ovladač

TapAdapter

Tato struktura obsahuje informace potřebné pro obecnou síťovou kartu, tedy jméno zařízení, MAC adresu, lokální IP adresu, vzdálenou IP adresu, síťovou masku, ethernetové hlavičky pro protokol ARP, broadcastovou adresu, handle obsahující vytvořený NDIS miniport, čítače přijatých a odeslaných paketů, komunikační médium, MTU, síťové adresy pro DHCP, multicastové seznamy a strukturu s názvem `TapExtension` obsahující všechny ostatní informace o adaptéru.

TapExtension

TapExtension zapouzdřuje vlastnosti specifické pro TAP zařízení. Jsou jimi fronta pro nezpracované požadavky na zaslání paketu (packet buffer), fronta pro nezpracované požadavky na čtení paketu (IRP buffer), struktura DEVICE_OBJECT (viz. sekce 3.1.5), čítač otevřených TAP zařízení, zámky pro čtení a zápis do fronty (packet, IRP), jméno TAP zařízení a informace o stavu zařízení.

3.5.2 Funkce pro NDIS

Nyní bude následovat výčet většiny funkcí ovladače TAP volaných a používaných rozhraním NDIS, a současně vytvářejících ovladač miniportu.

- **DriverEntry** - vstupní bod pro všechny ovladače (více v sekci 3.1.6), registrování funkcí miniportu, vytvoření miniportu

- **AdapterCreate** - vytvoří instanci struktury `TapAdapter`, naplní ji daty, volá funkci `CreateTapDevice`, která registruje instanci zařízení pro přístup z uživatelského režimu
- **AdapterHalt** - odstraní instanci spuštěného zařízení ze systému a uvolní alokované systémové prostředky
- **AdapterReset** - resetuje zařízení, v implementaci ovladače TAP nevyužita
- **AdapterReceive** - v případě běžné síťové karty zpracovává přijaté pakety na fyzickém rozhraní, zde nevyužita a její funkcionalita nahrazena voláním `NdisMEthIndicateReceive`
- **AdapterTransmit** - volána síťovým zásobníkem při požadavku na odeslání paketu z privátní sítě, kontroluje obsah paketu, rozlišuje typ, pokud ve frontě čekajících požadavků na čtení (IRP buffer) existuje alespoň jedna položka, je spojena s tímto paketem, čímž je vlastně paket odeslán. Podrobnější popis v sekci 3.4.3
- **AdapterQuery** - vrací informace o adaptéru dle typu požadavku reprezentovaného tzv. OID. Identifikátory objektů rozhraní NDIS (OIDs) jsou množina systémem definovaných konstant ve tvaru `OID_XXX`, např. `OID_GEN_LINK_SPEED`
- **AdapterModify** - nastavuje vlastnosti a chování zařízení dle typu OID zaslaného v parametrech funkce
- **TapDriverUnload** - volána při požadavku na deregistraci ovladače miniportu, nikoli pouze jedné instance (viz. funkce `AdapterHalt`), uvolňuje alokované prostředky operačního systému

3.5.3 Funkce pro WDM

V této části si popíšeme důležité funkce pro pochopení činnosti TAP adaptéru používající obecné techniky tvorby ovladačů zařízení modelu WDM operačního systému Windows XP.

- **CreateTapDevice** - vytváří instanci zařízení pro přístup z uživatelského režimu (otevření funkcí `CreateFile`, atd.), registruje funkce pro zachytávání IRP požadavků (IRP_MJ_OPEN, IRP_MJ_READ, a další), inicializuje fronty (IRP, packet) a jejich zámky pro výlučný přístup
- **DestroyTapDevice** - deregistruje instanci TAP zařízení, vyprazdňuje fronty, dealokuje používané systémové prostředky
- **TapDeviceHook** - rutina pro zachytávání a zpracovávání IRP paketů. Přijme požadavek, zkontroluje správnost IRP, a dle typu provádí příslušnou činnost. Primární kódy IRP rozepíšeme podrobněji:
  - **IRP_MJ_DEVICE_CONTROL** - vyvolána z aplikace funkcí `DeviceIoControl`, podle vedlejšího parametru IRP nastavuje a řídí instanci TAP zařízení (zjištění MAC adresy, MTU, nastavení režimu činnosti adaptéru, stavu média, konfigurace DHCP)
  - **IRP_MJ_CREATE** - zavoláním funkce `CreateFile` vznikne IRP s tímto primárním kódem, proběhne resetování adaptéru a změna stavu na „otevřený“

  - **IRP_MJ_CLOSE** - funkcí `CloseFile` volané z uživatelského režimu uzavřeme zařízení, resetujeme adaptér, vyprázdníme fronty a změníme stav na „uzavřený“
  - **IRP_MJ_READ** - IRP s požadavkem na čtení vznikne spuštěním funkce `ReadFile`, jestliže v packet bufferu existuje nějaký paket, je spojen s aktuálním požadavkem na čtení, pokud neexistuje, je IRP vloženo do fronty požadavků na čtení, IRP označeno stavem „nevyřízený“ (funkce `IoMarkIrpPending`)
  - **IRP_MJ_WRITE** - poslední primární kód vzniká zavoláním `WriteFile`, dle režimu činnosti adaptéru (most, tunel) paket zpracován, a funkcí `NdisMEthIndicateReceive` odeslán do privátní sítě.

- **CompleteIRP** - nastaví příslušný kód stavu indikující úspěšné zpracování či chybu, a dokončí korektně IRP požadavek systémovým voláním `IoCompleteRequest`, více v sekci 3.1.7

- **CancelIRP** - vyjmutí IRP paketu z fronty nezpracovaných požadavků na čtení, nastavení kódu stavu IRP na `STATUS_CANCELLED` a zavolání funkce `IoCompleteRequest`, viz. sekce 3.1.7

3.5.4 Ostatní funkce

Zbývá popsat poslední poměrně důležité funkce adaptéru. Význam ostatních funkcí lze vyčíst a poměrně dobře pochopit ze zdrojových kódů ovladače.

- **SetMediaStatus** - nastavuje stav média adaptéru na připojen (`NDIS_STATUS_MEDIA_CONNECT`) nebo odpojen (`NDIS_STATUS_MEDIA_DISCONNECT`)

- **InjectPacket** - použita v případě, kdy interně generujeme pakety jako odpovědi na ARP či DHCP (funkce `ProcessARP`, `ProcessDHCP`), zaslání zpracovaného paketu do sítě funkcí `NdisMEthIndicateReceive`

3.6 Popis implementace ovladače TUN

Stejně jako v případě TAP adaptéru, popíšeme struktury vytvářející TUN zařízení, poté nejvýznamnější funkce pro komunikaci TUN se systémem.

3.6.1 Softwarové struktury ovladače

Ovladač TUN definují v zásadě pouze dvě struktury, a to `tuntap_options` a `tuntap`. První jmenovaná obsahuje vlastnosti specifické pro operační systém, na kterém je TUN implementován. Pro Unixové platformy obsahuje minimum informací, protože samotná podpora je již zahrnuta v jádře. Druhá jmenovaná zapouzdřuje v případě Unixu TUN a TAP zařízení, dle popisovače souboru se určí, v jakém režimu se bude s adaptérem pracovat. V případě Windows poskytuje pouze rozhraní pro ovládání TAP adaptéru z uživatelského režimu. Struktury popíšeme z hlediska významu pro Windows.

- **tuntap_options** - obsahuje typ IP adresy (IPv4, IPv6), parametry pro DHCP, jméno domény, DHCP adresy pro WINS, NTP, DNS a NBDD (NetBIOS datagram distribution)

- **tuntap** - obsahuje typ zařízení (TUN, TAP), instanci struktury `tuntap_options`, aktuální jméno TUN/TAP zařízení, lokální IP adresa, maska sítě, broadcastová adresa, handle na vytvořenou instanci zařízení z `CreateFile`, struktury `overlapped_io` pro čtení a zápis a doplňkové údaje pro IP Helper API

3.6.2 Základní funkce

TUN obsahuje desítky funkcí, vypíšeme ovšem jen ty nejdůležitější, podstatné z hlediska komunikace TUN zařízení.

- **open_tun** - otevře instanci TAP zařízení, nastaví režim činnosti (most, tunel), přidělí IP adresu (manuálně, DHCP)
- **close_tun** - uzavře instanci TAP zařízení, odebere IP adresu, zruší nezpracované I/O požadavky na TAP
- **tun_write_queue** - inicializace zápisu pomocí `WriteFile`, samotný zápis, uvedení asynchronního požadavku do příslušného stavu, požadavek připraven na dokončení funkcí `tun_finalize`
- **tun_read_queue** - inicializace čtení pomocí `ReadFile` a samotné čtení, uvedení asynchronního požadavku do příslušného stavu, požadavek připraven na dokončení funkcí `tun_finalize`
- **tun_finalize** - ukončí asynchronní požadavek, v případě stále nedokončené I/O operace čeká dále na dokončení, jinak naplní buffery daty, případně nastaví chybové kódy
- **write_tun_buffered** - obálka pro volání `tun_write_queue`, kontrola dosud nezpracovaného asyn. požadavku, případné dokončení, poté samotný zápis
- **read_tun_buffered** - pouze prostředník pro volání `tun_finalize`
- **tun_set** - po vnitřním nastavení událostí pro čtení (v OpenVPN) zavolání `tun_read_queue`

Kapitola 4

Návrh

Se znalostmi získanými z předchozí kapitoly můžeme přejít k návrhu převodu. Cílem celého projektu bylo převést celou aplikaci OpenVPN na operační systém Windows CE 6.0. Z podrobné analýzy vyplynulo, že první uvažovaná část převodu, tedy síťový ovladač TUN/TAP bude natolik náročná, že není možné ve standardním rozsahu diplomové práce celé zadání splnit. Proto v této kapitole bude pouze popis převodu síťového ovladače.

Na úvod obecně shrneme odlišnosti operačních systémů Windows XP a Windows CE z hlediska tvorby ovladačů, a z toho vyplývající důsledky pro TUN/TAP ovladač. Při převodu bude kladen důraz na ovladač TAP, kde je nutné udělat nejvíce změn. Převod se budeme snažit rozdělit na dvě části, nejprve převod TAP, poté TUN. Nicméně, obě tato zařízení jsou na operačním systému Windows XP na sobě natolik závislá (přesněji řečeno, TUN závislý na TAP), že se určité části budou prolínat.

4.1 Princip převodu

Při převodu narážíme na dva zásadní problémy. První se týká čistě ovladačové části, druhý platí obecně i pro tvorbu běžných aplikací.

1. **Komunikační mechanismus IRP** - Windows CE nepodporují IRP
2. **Asynchronní I/O požadavky** - Windows CE nepodporují overlapped I/O

V implementaci TAP se používají IRP pro komunikaci ovladače s aplikací spuštěnou v uživatelském režimu. Otevření nebo uzavření, zápis nebo čtení z TAP znamená interní vygenerování IRP paketu s daným kódem požadavku.

Overlapped I/O, neboli asynchronní vstupně/výstupní požadavky používá ovladač TUN při komunikaci s TAP, a to při čtení nebo zápisu. Pokud bychom tento mechanismus neměli, bylo by nutné po každém zavolání funkce pro čtení/zápis paketu z TAP zařízení čekat, až se I/O událost dokončí, čímž by se stala aplikace nepoužitelná.

Než přejdeme ke konkrétnímu řešení pro jednotlivé ovladače, ještě popíšeme rozdíly ve tvorbě ovladačů z hlediska načítání systémem, možnosti nahrazení mechanismu IRP na Windows CE, stejně tak emulaci asynchronních vstupně/výstupních požadavků.

4.1.1 Rozdíly ve tvorbě ovladačů

Pokud chceme vytvořit ovladač na operačním systému Windows XP, je situace poněkud složitější, než v případě Windows CE. Musíme mít speciální překladač, jehož výsledkem je

binární soubor s příponou `.sys` obsahující samotný ovladač a soubor typu `.inf`, který je důležitý pro instalaci ovladače do systému.

Ve Windows CE jsou ovladače reprezentovány souborem s příponou `.dll`, což je standardní dynamicky linkovaná knihovna. Není třeba speciální překladač, ovladač se vytvoří jako kterákoli jiná dynamická knihovna, s dodržením jistých pravidel. Ta spočívají v exportování funkcí nutných pro zavedení ovladače do systému. Jednou z povinných funkcí je `DllMain`, která je první volanou při nahrávání ovladače (souvislost s obecným konceptem tvorby dynamických knihoven). Pokud se budeme bavit právě o síťovém ovladači používajícím rozhraní NDIS, potom musíme též exportovat funkci `DriverEntry`. Další exportované hlavičky funkcí poté závisí na tom, jaký typ ovladače tvoříme.

Automatické načítání ovladačů zajistíme jednoduše zapsáním několika údajů do registrů operačního systému, konkrétní příklad uvedeme v další kapitole s názvem implementace a testování.

Posledním, již dříve zmiňovaným rozdílem je vlastnost vzniklá příchodem Windows CE 6.0, a to možnost načítání ovladačů v uživatelském režimu.

4.1.2 Alternativa IRP

Neexistující podpora komplexního mechanismu IRP znamená, že operační systém musí nabízet nějakou alternativu, jak zpracovávat příchozí požadavky z uživatelského režimu. Tou je v podsekci 3.2.3 popsaný obecný model pro tvorbu ovladačů na Windows CE - stream interface model. S pomocí něj exportujeme kromě povinných funkcí ovladače ještě ty, které budou zachytávat a zpracovávat požadavky na TAP zařízení.

Ve Windows XP stačila pro tento účel jedna funkce - v případě TAP nazvaná `TapDeviceHook`, která uvnitř dle příchozího primárního IRP kódu určila, kterou událost aplikace z uživatelského režimu vyvolala (otevření, čtení, zápis..). Nyní pro každý zpracovávaný požadavek vytvoříme samostatnou funkci, jejíž rozhraní exportujeme do `dll` souboru s ovladačem.

4.1.3 Asynchronní I/O

Tato část převodu bude nejsložitější, zahrnuje jak změny v ovladači TAP, tak v TUN i samotné aplikaci OpenVPN. Navrhneme řešení především pro TAP a TUN, integrace s OpenVPN bude předmětem dalšího případného projektu na toto téma.

Zopakujme, že při požadavku na vstupně/výstupní událost bez asynchronního zpracování musí aplikace čekat na dokončení dané operace. Čekání znamená pozastavení vlákna v systému, které I/O operaci iniciovalo.

V případě asynchronního zpracování je pro každý I/O požadavek vytvořeno vnitřně vlákno, které samostatně čeká na dokončení I/O operace, a po dokončení uvědomí hlavní aplikaci (pomocí objektů událostí), že je operace hotova. Z popsaného je patrné, že při žádosti o čtení nebo zápis, které trvá jistou dobu, není hlavní aplikace blokována a může provádět jinou užitečnou činnost.

Emulací asynchronních I/O je myšleno nahrazení funkcionality struktury `OVERLAPPED` vkládané jako poslední parametr funkce `ReadFile` nebo `WriteFile` jiným mechanismem. Takovou alternativou bude použití objektů událostí (vnitřně používané strukturou `OVERLAPPED`), které budeme nastavovat manuálně, nikoli automaticky jako za použití overlapped I/O na operačním systému Windows XP.

Z důvodů jednoduchosti lze uvažovat operace zápisu na TAP zařízení jako synchronní, protože odeslání paketu do vnitřní sítě proběhne téměř okamžitě, operace `NdisMEthIndi-`

cateReceive není blokující. Dále se budeme soustředit pouze na operaci čtení z TAP, jejíž neblokující variantu bude možné bez velkých změn případně převést i na operaci zápisu.

4.2 Převod ovladače TAP

Dostáváme se k nejdůležitější části, a to převodu ovladače TAP. Bude se skládat ze tří částí, nejprve nahrazení mechanismu IRP pro zachytávání událostí na zařízení, poté emulace IRP požadavků na čtení včetně popisu mapovaní paměti mezi procesy (pro přenos načtených dat ze zařízení do uživatelského prostoru aplikace), a nakonec emulace vlastního zpracovávání asynchronních I/O požadavků.

4.2.1 Převod na streamové rozhraní

O principech streamového rozhraní bylo řečeno vše, nyní přistoupíme ke konkrétnímu případu - zařízení TAP. Každý primární kód IRP, který zpracovává rutina `TapDeviceHook` nahradíme samostatnou funkcí ve formátu `TAP_xxx`, kde `xxx` je jméno funkce. Navíc je třeba přidat dvě speciální funkce, první se volá při registraci zařízení v systému - `ActivateDeviceEx`, druhá při deregistraci zařízení - `DeactivateDevice`. Činnosti prováděné jednotlivými funkcemi plně korespondují s činnostmi prováděnými jednotlivými částmi rutiny `TapDeviceHook`, až na detaily týkající se zpracování IRP. Tabulka 4.1 znázorňuje primární kódy IRP, a k nim odpovídající jména funkcí, včetně funkcí pro registraci a deregistraci zařízení.

Primární kód IRP	**Nahrazující funkce**
IRP_MJ_CREATE	TAP_Open
IRP_MJ_CLOSE	TAP_Close
IRP_MJ_READ	TAP_Read
IRP_MJ_WRITE	TAP_Write
IRP_MJ_DEVICE_CONTROL	TAP_IOControl
Systémová funkce	**Volaná funkce**
ActivateDeviceEx	TAP_Init
DeactivateDevice	TAP_Deinit

Tabulka 4.1: Funkce nahrazující primární kódy IRP

Pro úplnost doplníme funkci `TAP_Seek` (volaná z uživatelského režimu funkcí `SetFilePointer`), která mění nastavení pozice ukazatele dat ve čteném zařízení, v případě TAP ovladače není využita. Pokud bychom implementovali ovladače fyzického zařízení, bylo by nutné přidat ještě funkce `TAP_PowerUp` a `TAP_PowerDown`, které obnoví či přeruší napájení připojeného adaptéru v systému v případě, že power management je řízen softwarově. TAP je virtuální ovladač, a proto tyto funkce nepoužívá.

Všechny výše popsané funkce s prefixem `TAP` jsou exportovány v `.dll` knihovně implementující ovladač, systém je poté interně na žádost aplikací z uživatelského režimu volá.

4.2.2 Mapování ukazatelů mezi procesy

V této části by se dalo poměrně obsáhle napsat o způsobu mapovaní ukazatelů mezi procesy (tzv. marshalování), z důvodu omezeného rozsahu práce uvedeme pouze základní infor-

mace. Nutno říci, že v meziprocesové komunikaci došlo k velkým rozdílům mezi operačním systémem Windows CE 5.0 a Windows CE 6.0 (z důvodů změn v paměťové architektuře), na což budeme klást důraz. Následující výklad je důležitý zejména pro pozdější pochopení principu emulace asynchronních I/O.

Úvod

Začněme základními pojmy, které je nutné znát pro pochopení meziprocesové komunikace:

- **Ukazatel na parametr (pointer parameter)** - ukazatel, který je předán v rozhraní funkce a ukazuje na daný parametr (buffer). V následujícím příkladu je proměnná `pBuffer` ukazatelem na parametr:

```
WriteFile(hFile, pBuffer, dwBufferSize, ...);
```

- **Zapouzdřený ukazatel (embedded pointer)** - ukazatel, který je předán funkci, a je uložený uvnitř nějaké struktury. V další ukázce kódu je proměnná `pEmbedded` zapouzdřeným ukazatelem:

```
struct MyStruct  {
   BYTE *pEmbedded;
   DWORD dwSize;
};
DeviceIoControl(hFile, pMyStruct, sizeof(MyStruct), ...);
```

- **Kontrola přístupu (access checking)** - ověření, zda volající proces má oprávnění přistupovat k parametru (bufferu). Zde se liší přístup v jednotlivých verzích operačního systému Windows CE:
  - CE 5.0 - ovladače používaly funkci `MapCallerPtr()` ke kontrole přístupu ukazatelů na parametry a zapouzdřených ukazatelům. Jádro také kontrolovalo přístup k parametrům, ovšem neznalo velikost bufferů, na které se ukazatel vázal. Kontrola probíhala pouze pomocí jednoho bytu bufferu.
  - CE 6.0 - API pro kontrolu přístupu bylo změněno, a zahrnuje informaci o velikosti parametrů. Jádro nyní provádí plnou kontrolu přístupu k parametrům. Ovladače musí pouze ověřit přístup k parametrům v zapouzdřených ukazatelích pomocí funkce `CeOpenCallerBuffer`, která také zodpovídá za marshalování dat, jak bude vysvětleno dále.
- **Bezpečná kopie (secure copy)** - vytvoření kopie bufferu pro vyloučení souběžné modifikace dat ovladačem a aplikací.
- **Synchronní přístup (synchronous)** - přístup k parametrům uvnitř volané funkce (funkce ovladače), ve vláknu volajícího (vlákno aplikace).

Princip funkce

Mapování ukazatelů neboli marshalování je příprava ukazatele pro přístup ovladače do paměti volajícího (aplikace). Ovladač běží v rámci jiného procesu, než aplikace, která jej

volá. Virtuální paměť každého procesu je implicitně chráněna proti přístupu z jiného procesu, z pochopitelných bezpečnostních důvodů. Ovladač tedy musí provést několik činností, aby mohl přistupovat k bufferu umístěném v jiném adresovém prostoru.

Synchronní přístup do paměti je prováděn během volání funkce ovladače v rámci spuštěného vlákna volajícího (aplikace). Jestliže vlákno ovladače přistupuje do paměti jiného procesu po skončení volání vlákna volajícího, potom se tento režim nazývá asynchronním.

Ve Windows CE 5.0 sdílely všechny procesy stejný adresový prostor. Pro získání ukazatele na volající paměť stačilo, aby ovladač namapoval ukazatel adresového prostoru aplikace. Mapování byla jednoduchá transformace hodnoty ukazatele, který ukazoval na tzv. „slot" jiného procesu ve společném adresovém prostoru. V CE 5 byla paměť rozdělena na 32 slotů, pro aplikace přístupných 31, poslední zabíralo jádro OS. S tím souvisel maximální počet spuštěných procesů na 32 a maximální velikost operační paměti 32MB.

Ve Windows CE 6.0 má každý proces svůj unikátní adresový prostor. Marshalování paměti proto nemůže být prostá transformace ukazatele. Každá paměť musí být zkopírována z jednoho procesu do druhého (duplikace) nebo musí být alokována nová virtuální adresa v procesu ovladače, a ukazovat na stejnou fyzickou paměť volajícího (alias). Systémové prostředky jsou alokovány uvnitř procesu ovladače, a musí být uvolněny, jakmile ovladače ukončí práci s pamětí.

Marshalování ve Windows CE 6.0 také určuje, zda je parametr pouze pro čtení, zápis, nebo současně pro čtení a zápis. Pro vysvětlení, co musí ovladač udělat, aby marshaloval paměť, popíšeme odděleně synchronní a asynchronní přístup:

- **Synchronní přístup** - jádro automaticky mapuje ukazatele na parametry. Ovladač se stará pouze o zapouzdřené ukazatele. V CE 5.0, ovladače používaly funkci `MapCallerPtr`, v CE 6.0 používají funkci `CeOpenCallerBuffer` k marshalování zapouzdřených ukazatelů, a funkci `CeFreeCallerBuffer`, jestliže práci s buffery končí. Jak funkce `MapCallerPtr`, tak `CeOpenCallerBuffer` kontrolují automaticky oprávněnost přístupu k paměti.

- **Asynchronní přístup** - složitější, v CE 5.0 byl každý procesový „slot" chráněn proti přístupu z jiného procesu. Každé vlákno mělo vlastní skupinu oprávnění, k přístupu k jednotlivým procesovým slotům. Protože volající vlákno spouští příslušnou rutinu ovladače, nese s sebou práva pro přístup do procesu vlastníka této paměti (aplikace). Přístupy do paměti volajícího mohou být úspěšné právě v tomto vlákně, ostatní vlákna (tedy asynchronní) musí nejprve získat oprávnění k přístupu do jiných procesových slotů.

  V CE 6, jako v předchozí verzi systému, musí být uskutečněna určitá příprava pro přístup ovladače do paměti volajícího v jiném vláknu. Způsob marshalování paměti se liší mezi režimem jádra a uživatelským režimem, a mezi ukazateli na parametry a zapouzdřenými ukazateli. Jediný způsob pro zajištění, že kód ovladače bude pracovat správně ve všech těchto případech, je připravit parametry pro asynchronní přístup předtím, než k nim přistoupí jiné vlákno.

Pro asynchronní přístup jsou ukazatele na parametry a zapouzdřené ukazatele spravovány stejným způsobem. Předpokládejme, že začínáme s bufferem, který již byl mapován nebo marshalován pro synchronní přístup (provedeno buď automaticky jádrem nebo funkcí `CeOpenCallerBuffer`), pro zajištění asynchronního přístupu musí být provedeny následující kroky:

- Pro CE 5.0, ovladač musí zavolat funkci `SetProcPermissions` v asynchronním vláknu, aby mohl přistupovat k bufferu v jiném procesu (v synchronním režimu tato práva získáme funkcí `GetCurrentPermissions`)
- V CE 6.0 musí ovladač zavolat `CeAllocAsynchronousBuffer` pro připravení asynchronní verze parametru, který je připraven pro synchronní použití. Volání této funkce musí být prováděno synchronně, před předáním parametru do asynchronního vlákna. Jestliže vlákno skončí práci s bufferem, zavolá `CeFreeAsynchronousBuffer`, a uvolní alokované prostředky.

Pozn: ne všechny asynchronní případy jsou podporovány pro ovladače v uživatelském režimu. Ty nemohou asynchronně zapisovat do ukazatelů na parametry. Další informace týkající se marshalování, včetně bezpečných kopií parametrů a zpracování výjimek lze najít v [12].

Shrnutí

Použití funkcí na Windows CE 6:

- **Parametr - použitý synchronně** - jestliže je nutná bezpečná kopie, použije se funkce `CeAllocDuplicateBuffer` (`CeFreeDuplicateBuffer`), jinak samotný ukazatel (jádrem automaticky mapovaný)
- **Parametr - použitý asynchronně** - jestliže je nutná bezpečná kopie, použije se funkce `CeAllocDuplicateBuffer` (`CeFreeDuplicateBuffer`), jinak funkce `CeAllocAsynchronousBuffer` (`CeFreeAsynchronousBuffer`)
- **Zapouzdřený parametr - použitý synchronně** - použití funkce `CeOpenCallerBuffer` (`CeCloseCallerBuffer`) pro marshalování a bezpečnou kopii (v případě nutnosti)
- **Zapouzdřený parametr - použitý asynchronně** - zavolání `CeOpenCallerBuffer` a poté `CeAllocAsynchronousBuffer`, na konci zavolána funkce `CeFreeAsynchronousBuffer` před `CeCloseCallerBuffer`

4.2.3 Emulace IRP pro čtení

Při požadavku na čtení z instance TAP adaptéru vzniklo IRP indikující čtení. V případě nemožnosti okamžitě IRP zpracovat (žádná data v packet bufferu), došlo k uložení do fronty nezpracovaných požadavků na čtení (IRP buffer). Na Windows CE ovšem žádné IRP nevznikají, proto je nutné vytvořit vlastní strukturu, která IRP emuluje, abychom mohli požadavky na čtení později identifikovat a zpracovat.

Funkce `ReadFile` (stejně tak `WriteFile`) má v hlavičce jako jeden z parametrů buffer, do kterého se data ovladačem zapisují (případně čtou). Za situace, že požadavek na čtení či zápis zpracujeme synchronně, tedy ve funkci `TAP_Read` (`TAP_Write`), stačí pouze zkopírovat určitou část paměti obsahující data do výstupního bufferu (nebo naopak).

Jiná situace nastává v případě, že nejsme schopni okamžitě I/O uskutečnit a používáme asynchronní zpracování. Tehdy funkce `TAP_Read` (`TAP_Write`) vrátí status indikující nemožnost získat (zapsat) data okamžitě, přičemž si interně tento požadavek na čtení či zápis uložíme pro pozdější zpracování.

Dosud se zdá vše v pořádku. Jenže tím, že se vrátíme z funkce `TAP_Read` nebo `TAP_Write` ztratíme ukazatel na vstupní/výstupní buffer, ze kterého máme data později číst, případně do něj zapisovat. Nabízí se celkem intuitivní řešení, a to si ukazatel jednoduše uložit do struktury obsahující nezpracované I/O požadavky. Kdybychom pracovali v rámci jednoho spuštěného procesu, je možné takový postup zvolit. Nicméně, zde komunikují dva spuštěné procesy, proces běžící v uživatelském režimu (OpenVPN), a samotný ovladač běžící buď v režimu jádra nebo také uživatelském režimu. V každém případě, ovladač využívá jiný proces než aplikace.

V operačních systémech obecně platí, že z důvodů bezpečnosti není možné přistupovat do adresového prostoru jiného procesu. Ovšem v mnoha případech vzniká potřeba vzájemné komunikace dvou oddělených procesů a programátoři vyžadují v tomto směru určitou podporu ze strany operačního systému. Takový mechanismus se nazývá **mapování ukazatelů mezi procesy**, případně **marshalování ukazatelů**, a byl popsán v sekci 4.2.2.

4.2.4 Emulace asynchronních I/O

Emulaci asynchronních I/O popíšeme nejprve z pohledu nově vytvořených struktur a následně vytvořených funkcí.

Struktury

Nejprve bylo třeba vytvořit prostředníka, přes kterého bude aplikace komunikovat s ovladačem TUN. Tím je struktura `OverlappedReadRequest`, která kombinuje funkcionalitu standardní struktury `OVERLAPPED` a IRP. Obsahuje handle na objekt události, přes který bude indikována informace o dokončení čtení (ovladačem TAP). Dále zapouzdřuje ukazatele na buffer s načtenými daty, velikost načtených dat a status zpracovávání požadavku na čtení.

V implementaci TAP pro Windows XP ukládal ovladač nezpracované požadavky na čtení (IRP) do vnitřní fronty (IRP queue). Nyní IRP nemáme k dispozici, proto musíme navrhnout další strukturu, která se bude vkládat do IRP fronty. Její název zní `TapReadRequest`, a obsahuje tři položky - ukazatel na strukturu `OverlappedReadRequest`, ukazatel na buffer, do kterého se budou zapisovat data a ukazatel na otevřenou instanci TAP zařízení (pokud TAP uzavřeme dříve, než se zpracují všechny I/O požadavky, musíme je identifikovat, a korektně odstranit z fronty). V `TapReadRequest` existují ještě další dva ukazatele na `OverlappedReadRequest` a buffer, které slouží pro uložení marshalovaných ukazatelů (jeden pro otevření zapouzdřených ukazatelů, druhý pro vytvoření asynchronní varianty).

Funkce

Jestliže nemůžeme ve funkci `ReadFile` okamžitě získat paket z vnitřní sítě, musíme jej uložit. Proto vytvoříme funkcí `NewReadRequest` novou instanci pseudo-IRP požadavku na čtení (struktury `TapReadRequest`). Při vytváření též dojde k marshalování parametrů a vytvoření asynchronních variant.

V případě, že získáme paket, zavoláme funkci, která čekající asynchronní požadavek naplní daty a korektně ukončí. Tou byla na Windows XP funkce `CompleteIRP`. Náhražkou je funkce `CompleteReadRequest`, která provádí téměř stejnou činnost, a na konci nastaví objekt události struktury `OverlappedReadRequest` do signalizovaného stavu. Tím nepřímo upozorní vnitřní smyčku událostí OpenVPN, že jsou data k dispozici, a je možné s nimi provádět další činnosti.

Po dokončení asynchronního požadavku na čtení ve funkci `DeleteReadRequest` uvolníme alokované zdroje a uzavřeme marshalované ukazatele.

4.3 Převod virtuálního ovladače TUN

Zatímco zařízení TAP může existovat samostatně, TUN nikoliv (bavíme se o operačním systému Windows). Je závislý na rozhraní TAP a vnitřním komunikačním mechanismu OpenVPN. Jelikož v rámci této diplomové práce nebude řešen převod OpenVPN, není možné celý ovladač TUN převést na platformu Windows CE. Uvedeme pouze změny, které je nutné vykonat v komunikační části TUN s ovladačem TAP, aby navržený způsob emulace asynchronních I/O fungoval správně. V implementaci narazíme na další problém, a to nemožnost otestovat nový mechanismus I/O přímo v TUN. Ovladač importuje několik modulů z aplikace OpenVPN, které mohou potřebovat také úpravu.

4.3.1 Emulace asynchronních I/O

Při převodu na nový mechanismus se musíme především vyrovnat s nemožností použít parametr `OVERLAPPED` ve funkcích `ReadFile` a `WriteFile`. Nyní může každé volání jedné z těchto funkcí teoreticky blokovat další komunikaci až do získání (zapsání) dat. Jak jsme v úvodních informacích této kapitoly naznačili, volání `WriteFile` budeme považovat za synchronní, protože vnitřní implementace funkce `TAP_WRITE` ovladače TAP není blokující.

Pro čtení použijeme stejnou funkci, jako v předchozí implementaci (`ReadFile`), ovšem vnitřně ji implementujeme též jako neblokující. Rozdíl oproti `WriteFile` je ten, že musíme později nějakým způsobem informovat o připravenosti dat. Tím je společná struktura aplikaci i ovladači `OverlappedReadRequest`, kterou předáme jako parametr s daty ve funkci `ReadFile`. TAP ji buď okamžitě naplní daty, pokud jsou k dispozici, nebo si ji uloží vnitřně do fronty, a procedura okamžitě skončí.

Smyčka událostí OpenVPN později zaregistruje signalizovaný objekt požadavku na čtení a zavolá funkci `tun_finalize` pro dokončení asynchronního čtení, tedy přenesení dat z bufferů.

Struktury

Jedinou změněnou strukturou bude `tun_tap`. A to tím způsobem, že přidáme pouze instanci naší společné struktury `OverlappedReadRequest`. Instanci struktury `overlapped_io` s názvem `writes`, tedy instanci asynchronního požadavku pro zápis nebudeme využívat, pouze nastavíme její vnitřní proměnnou objektu události `hEvent` do stále signalizovaného stavu. Tím se oklame OpenVPN, zápis se bude chovat jako synchronní.

Funkce

Z komunikační části pozměníme především tři následující funkce:

- **tun_write_win32** - veškerý kód zajišťující asynchronní zápis nahradit pouhým voláním `WriteFile`.
- **tun_read_queue** - před samotným vyvoláním funkce pro čtení musíme správně nastavit instanci `OverlappedReadRequest` struktury `tun_tap` a poté ji předat jako parametr volání funkce `ReadFile`. Následující zpracování stavu asynchronního požadavku je stejné jako u původní implementace.

- **tun_finalize_read** - zpracování taktéž téměř stejné, jediným rozdílem je získání výsledku asynchronního volání - náhrada funkce `GetOverlappedResult` funkcí `WaitForSingleObject` s parametrem okamžitého návratu. Tím zjistíme, zda byl objekt události signalizován nebo je stále v resetovaném (čekajícím) stavu.

4.4 Převod OpenVPN

Jednou z podmínek k převedení a otestování aplikace OpenVPN na operačním systému Windows CE je funkční implementace síťového ovladače TUN/TAP na této platformě. Samotná analýza, převod a implementace TUN/TAP je natolik obsáhlá, že zabere celý rozsah tohoto projektu. Z toho vyplývá, že splnění úkolu stanoveného zadáním bude nutné přesunout do případného navazujícího projektu. Ten se může odprostit od specifik programování v režimu jádra a věnovat se pouze aplikační části.

Dosavadní převod ovladačové části je koncipován tak, aby mohlo být v aplikaci OpenVPN provedeno co nejméně změn. Nicméně, je stále dosti pravděpodobné, že při převodu narazíme na chybějící knihovny, neexistující konfigurační moduly (např. netsh, pro které TUN nabízí ve výchozí verzi rozhraní), nebo také způsob zavádění ovladače do systému.

Vše výše popsané není až tak výrazně složitým problémem, jako spíše časovým. OpenVPN je v současně době již poměrně rozsáhlým projektem, a pochopení vnitřní komunikace samotného programu zabere také určitou dobu, především z důvodu neexistující programové dokumentace. Každou část musíme analyzovat postupným a pracným procházením zdrojových kódů. Z tohoto důvodu by bylo zajímavé interní strukturu OpenVPN kvalitně a přehledně zdokumentovat.

Kapitola 5

Implementace a testování

Implementace je poměrně zásadní část tohoto projektu. Převádí vše dosud uvedené do praktické podoby. Nedílnou součástí implementace je testování, kterým ověříme, zda navržený převod a implementace byla správná.

Začneme popisem implementačního prostředí, dále převodem jednotlivých součástí ovladačů, protože jak návrh napověděl, samotná aplikace OpenVPN nebude prakticky realizována. Po krátké zmínce o problémech, které provázely implementaci přejdeme k registraci a spuštění ovladače v systému. Kapitolu zakončí popis testovacího programu, který ověří správnou funkčnost síťového ovladače.

5.1 Implementační prostředí

Implementace probíhá v prostředí operačního systému Microsoft Windows XP SP2 na vestavěném zařízení NetDCU10 běžícím na operačním systému Microsoft Windows CE Embedded 6.0. Zařízení nemá displej, proto veškerá komunikace probíhá přes ethernetové rozhraní a příkazovou řádku (telnet), správa souborového systému a registru přes USB a program ActiveSync ve verzi 4.5.0. Ladící informace jsou vypisovány do textového terminálu připojeného přes sériový port.

5.1.1 Vývojové prostředí

Vývojovým prostředím bylo zvoleno Microsoft Visual Studio 2005 SP1 Professional Edition s přidanou podporou vývoje na zařízení NetDCU10. Jiná volba prakticky neexistuje, jelikož výrobce vestavěného zařízení oficiálně nepodporuje alternativní vývojová prostředí.

Užitečné byly též nástroje pro vzdálenou správu dodávané s aplikací Microsoft Visual Studio 2005 využívající ActiveSync pro připojení k zařízení. Jsou jimi:

- **Remote Registry editor** - pohodlná vzdálená úprava registrů operačního systému Windows CE 6.0 ve stylu aplikace `regedit` na OS Windows XP.
- **Remote File viewer** - prohlížeč obsahu souborového systému.
- **Remote Zoom in** - vytvoří bitmapový obraz aktuálního obsahu obrazovky na vestavěném zařízení. Protože nemáme k dispozici displej, je tato volba jediným způsobem, jak získat obsah obrazovky.
- **Process Viewer** - správa běžících procesů na Windows CE. Ve výchozí konfiguraci nenabízí tento systém žádnou podporu kontroly procesů přes terminál, a protože přes

telnet nefunguje známá klávesová zkratka Ctrl+C pro násilné ukončení procesu, je tato aplikace jedinou jednoduše dostupnou možností, jak procesy z jakéhokoli důvodu ukončit.

5.1.2 Windows Driver development kit (DDK)

Komplexní balík pro vývoj ovladačů na operačním systému Windows XP. Není zdarma přístupný, pro účely tohoto projektu sloužil pouze pro pokusné sestavení TAP ovladače na Windows XP, a poté jako cenný zdroj informací týkající se tvorby ovladačů obecně.

5.1.3 Platform builder pro Windows CE

Komplexní sada nástrojů a příkladů pro tvorbu vlastní platformy založené na OS Windows CE. Na vytvořené platformě můžeme poté provádět veškerý vývoj a testy jako na reálném zařízení. Balík je licencovaný a není volně k dispozici. V tomto projektu sloužil především jako zdroj informací a sbírka zdrojových kódů různých ovladačů, z jejichž struktury se dalo vycházet při převodu ovladače TAP.

5.2 Součásti implementace

Ačkoli implementace obsáhla velkou část prostoru věnovaného řešenému projektu, všechny praktické informace byly popsány v sekci návrh. Shrneme jen to, co se podařilo implementovat.

5.2.1 Ovladač TAP

Implementace probíhala dle návrhu. Došlo nejprve k vytvoření vstupního bodu ovladače `DllMain` a vstupního bodu pro NDIS `DriverEntry`. Celý vývoj probíhal postupně s přidáváním jednotlivých funkcí a testovaní správné funkcionality. Nakonec došlo k exportování funkcí streamového rozhraní `TAP_XXX`.

5.2.2 Ovladač TUN

Jak již víme, ovladač TUN je závislý na realizaci samotné aplikace OpenVPN. Přestože je teoreticky možné provést implementaci, výsledek nelze jednoduše otestovat. Spokojíme se tedy s návrhem převodu, jehož správnost ověří navazující projekty na toto téma, a pro účely otestovaní ovladače TAP navrhneme vlastní zjednodušenou aplikaci, principiálně podobnou ovladači TUN.

5.2.3 Aplikace OpenVPN

Analýza ovladačové části OpenVPN zabrala značný rozsah této práce, a proto aplikace nebude implementována. Více informací uvedeno v sekci 4.4.

5.3 Implementační problémy

Při každé implementaci se řeší desítky a stovky různých, větších či menších problémů. Zmíníme jen ty, které mohou být zajímavé pro další projekty na podobné téma.

5.3.1 Prostředí implementace

Implementace probíhala inkrementálně, vždy s otestováním dodané funkce. Na začátku se vyskytnul zásadní problém - jak ovladač ladit. Zároveň robustní a nejjednodušší formou byly zvoleny textové výpisy. Více o ladění se lze dočíst v sekci 5.4.3.

Velký problém občas nastával s absencí displeje vestavěného zařízení. Ačkoli není nutný pro vývoj ovladače, celou proceduru by v jistých případech urychlil.

Další nepříjemnost nastala, pokud byla v ovladači nějaká chyba, která způsobila jeho uváznutí. V takovém případě nešlo ovladač jednoduše z paměti odstranit, protože se nejedná o běžně spuštěný proces, ale načtenou dynamickou knihovnu. Jediným řešením v dané situaci bylo vestavěný systém restartovat, což práci poměrně zdržovalo.

5.3.2 Řetězce ANSI vs. UNICODE

Poměrně častým problémem, na které autor tohoto textu narážel, byla rozdílná práce s řetězci. Windows CE vnitřně podporuje řetězce ve formátu UNICODE (velikost 2 byty), ovšem často používaná standardní knihovna jazyka C je navržena pro práci s ANSI řetězci (velikost 1 byte).

V případě vývoje ovladače je nutné pracovat s oběma reprezentacemi (jádro systému používá řetězce UNICODE, uživatelský režim řetězce ANSI), a často využívat funkce pro konverzi mezi těmito dvěma typy.

5.4 Spuštění ovladače

V této fázi již máme ovladač implementován. Nyní je třeba jej v systému zaregistrovat a spustit. Jak vše provést se dozvíme z následujících řádků, a konkrétní případy nastavení lze zjistit z příloh tohoto textu.

5.4.1 Registrace ovladače v systému

Registrace je poměrně jednoduchá a spočívá v zapsání určitých hodnot do registrů operačního systému Windows CE. Snad je užitečné znovu zopakovat, že TAP jsou dvě zařízení v jednom - NDIS miniport a streamové zařízení. Proto budou v registrech dva zápisy, jeden pro potřeby knihovny NDIS, druhý pro registraci zařízení za účelem pozdějšího otevření z uživatelského režimu.

Registrace miniportu

V registrech pod klíčem `HKEY_LOCAL_MACHINE\Comm` se nachází mj. všechny ovladače, které zavádí rozhraní NDIS po startu operačního systému. Pro každý síťový ovladač musíme vytvořit minimálně dva klíče:

- **Ovladač**, který chceme zavést obsahující kromě nezbytných údajů také odkazy na své instance (např. `TAP Adapter`)
- **Instanci ovladače**, kde musí existovat minimálně jedna, pro každou se vytvoří nový klíč (např. `TAP Adapter1`, `TAP Adapter2`..)

Každý klíč se síťovým ovladačem obsahuje minimálně tři řetězce:

- **Group**, označující třídu ovladačů, do kterých patří (v případě síťových ovladačů hodnota `NDIS`)
- **ImagePath**, což je řetězec definující cestu k dynamické knihovně s ovladačem (např. `\FFSDISK\TAP_CE.dll`)
- **DisplayName**, libovolný popis názvu ovladače

Síťový adaptér musí být svázán s určitým protokolem, který jej bude shora zastřešovat. V běžných případech se jedná o protokol TCP/IP. Do jeho klíče s názvem `Linkage` zapíšeme do multiřetězce `Bind` identifikátory připojených síťových ovladačů. Konkrétní příklad nastavení je uveden v příloze A.1.

Registrace streamového zařízení

Streamovou část TAP ovladače lze sice teoreticky použít samostatně, ovšem v naší implementaci nemá kromě testování smysl. Na Windows CE se spouští po startu systému ty ovladače, které mají zápis v klíči registru `HKEY_LOCAL_MACHINE\Drivers\BuiltIn`. Po startu nechceme spouštět část TAP, proto si vytvoříme zápis samostatně, v jiném klíči registru. Cestu k tomuto klíči poté napevno definujeme v ovladači, kde si implementace miniportu streamový ovladač v případě potřeby sama zavede a uvolní.

Registrace je opět velice jednoduchá, stačí tři zápisy v klíči vztahujícímu se k ovladači:

- **Dll** - řetězec definující cestu k dynamické knihovně s ovladačem.
- **Order** - pořadí v načítacím procesu po startu, v našem případě nemá smysl.
- **Prefix** - třípísmenný řetězec, který předáváme jako první argument funkci `CreateFile` při otevírání instance zařízení (např. `COM`, `TAP`..).

Opět, konkrétní příklad je uveden v příloze A.1.

5.4.2 Načtení ovladače do systému

Nyní je ovladač korektně registrován a lze přistoupit k nahrání do paměti. Máme dvě možnosti - automatické nebo manuální načtení. První se týká spuštění při startu operačního systému, druhé spuštění na žádost uživatele.

Automatické nahrání ovladače

Většina ovladačů v systému je automaticky nahrávána po startu systému, stačí pouze platný zápis hodnot v příslušném klíči registru operačního systému. Ovladač je buď načtený samostatně nebo jako součást ovladače jiného (v případě členství ve třídě ovladačů).

Ruční nahrání ovladače

Pokud se jedná o streamový ovladač, načtení do paměti lze uskutečnit funkcí `ActivateDevice(Ex)`, poté lze zařízení klasicky otevřít. Odstranění ovladače z paměti provádí funkce `DeactivateDevice`.

Speciálním případem ve Windows CE jsou síťové ovladače, které spravuje knihovna (v podstatě také ovladač) NDIS. Načítá ty ovladače, které mají v registrech vlastnost `Group` s hodnotou `NDIS`. Nahrání do paměti proběhne po zavolání funkce `DeviceIoControl`

na otevřenou instanci NDIS zařízení s parametrem IOCTL_NDIS_REGISTER_ADAPTER a názvem klíče adaptéru společně s jednou z jeho instancí. Odstranění z paměti proběhne podobným způsobem, s kódem požadavku IOCTL_NDIS_DEREGISTER_ADAPTER a názvem klíče adaptéru.

Ve Windows CE také existuje speciální konzolová aplikace `ndisconfig`, která umožňuje síťové ovladače načítat a odstraňovat z paměti jednodušším způsobem, než psaním vlastních programů pro manuální načtení. Stačí na příkazovém řádku předat speciální parametry. Příklad v příloze A.2.1.

5.4.3 Ladění

Ladění ovladačů je obecně komplikované. Existuje několik možností, jak ovladač ladit, k čemuž jsou k dispozici specializované nástroje. Naše omezení spočívá v tom, že k vestavěnému systému přistupujeme vzdáleně a možnosti nejsou tak široké. Nástroje dodávané firmou Microsoft pro vzdálené ladění s názvem `Remote tools` nejsou pro naše účely vhodné, protože nemáme k dispozici displej pro zobrazení výsledků.

Dalším nástrojem je `Windows CE Test Kit` dodávaný s Platform builderem pro Windows CE 6.0. Jedná se o komplexní testovací prostředí pro všechny možné druhy ovladačů, ovšem testuje ovladač jako celek, nikoli pouze částečně funkční části.

Posledním možným řešením jsou ladící výpisy. Původní implementace TAP na Windows XP implementuje vlastní debugovací mechanismus, který je na Windows CE nepoužitelný. Proto bylo nutné všechny výpisy nahradit standardním makrem Windows pro výpis informací `DEBUGMSG`, které má téměř stejné možnosti jako funkce `printf`.

Nastal problém, kam ladící informace vypisovat. Ve výchozím nastavení zařízení NetDCU10 je ladící výstup zakázaný. Proto jej bylo nutné v boot loaderu zařízení povolit a přesměrovat na sériový port. Poté se spustila konzole připojená ke `COM1` portu, kde se výstup ladění zobrazoval. Schéma zapojení v příloze B.

5.4.4 Výsledek implementace TAP

Po spuštění ovladače se v systému objevil nový síťový adaptér s názvem `TAP Adapter`, což se dá ověřit aplikací `ipconfig`. Po krátké době je adaptéru přidělena IP adresa dle automatických hodnot v registrech, které ke klíči adaptéru doplnil `TCP/IP` stack.

Dostupnost adaptéru se ověří např. programem `ping` na danou IP adresu TAP zařízení. Další standardní aplikací pro sledování sítě je `netstat`, přes který lze sledovat statistiky toku dat přes síťový adaptér.

5.5 Testovací aplikace

Spuštění TAP adaptéru v systému ještě nezaručuje jeho správnou funkčnost. Ta se ověří až navázáním aplikace v uživatelském režimu na adaptér, přijímáním paketů z adaptéru, zpracováním a zasláním paketů zpět na adaptér. Stručně uvedeme několik programů, které bylo nutné v rámci vývoje vytvořit, poté návrhem finální testovací aplikace, a nakonec popisem nedostatků implementace TAP ovladače na operačním systému Windows CE 6.0.

5.5.1 Pomocné aplikace

V průběhu implementace bylo vytvořeno několik aplikací. Nejprve základní pro ruční načtení ovladače do systému, ověření schopnosti zapsat na adaptér a číst z něj. Dále dvě komu-

nikační jednotky na bázi klient-server, které přes knihovnu Winsock komunikovaly mezi sebou přes TAP adaptér. Nakonec, několik miniaplikací pro zjednodušení registrace a deregistrace ovladače v systému.

5.5.2 Návrh demonstrační aplikace

Návrh demonstrační aplikace by měl emulovat chování TUN ovladače, a tím vlastně běžné chování TAP adaptéru v kontextu celé aplikace OpenVPN. Je tedy třeba otestovat tyto funkce:

- Získání paketu z TAP adaptéru
- Zpracování paketu, tzn. generovat nějakým způsobem odpověď
- Zaslání paketu s odpovědí zpět na adaptér

Jako vhodné se pro tuto činnost jeví využití programu `ping`. Ten zasílá na adaptér ICMP paket typu ECHO, který musí odpovědět ICMP paketem typu REPLY. Princip funkce bude následující:

- Čekání ve smyčce na příchod ICMP paketu z vnitřní sítě TAP adaptéru (tzn. čekání na spuštění programu ping s IP adresou sítě, ve které se TAP nachází)
- Zpracování paketu, tzn. vytvoření odpovědi na ICMP paket
- Zaslání paketu s odpovědí zpět na adaptér, ve spuštěném programu ping by se měla indikovat správně odpověď s určitou časovou prodlevou

Po spuštění pingu na IP adresu virtuálního uzlu v testované síti se nejprve vygeneruje ARP paket s dotazem na MAC adresu cílového uzlu (velikost 42 bytů). Proto v testovací aplikaci musíme nejprve zachytit ARP paket typu REQUEST, vygenerovat virtuální MAC adresu (cílový uzel s danou IP fyzicky neexistuje) a poslat na adaptér ARP paket typu REPLY. Až poté jsou generovány standardní ICMP pakety na adaptér (velikost 74 bytů).

5.5.3 Implementace demonstrační aplikace

Při implementaci bylo nejprve nutné vytvořit struktury reprezentující ARP a ICMP pakety, abychom mohli získaný proud dat správně interpretovat a generovat odpovědi.

Pro ARP již existovala v původní implementaci TAP ovladače struktura, proto nebylo obtížné načtený buffer dat přetypovat na typ ARP, vygenerovat virtuální MAC adresu, změnit cílovou a zdrojovou MAC adresu uzlů a typ ARP paketu z REQUEST na REPLY.

Složitější bylo zpracování ICMP. Vlastní ICMP paket se skládá z ethernetové hlavičky (12 bytů), IP hlavičky (22 bytů), ICMP hlavičky (8 bytů) a ICMP dat (volitelná součást, na Windows velikost 32 bytů), celkem tedy 74 bytů. V původní implementaci TAP existují struktury pro ethernetovou hlavičku a IP hlavičku, bylo třeba vytvořit strukturu reprezentující hlavičku ICMP a datovou část ICMP.

Nejprve vytvoříme požadavek na čtení na TAP zařízení. Poté čekáme na paket z vnitřní sítě, tzn. blokující čekání na signalizování objektu události pomocí funkce `WaitForSingleObject`. Po přijetí ICMP paketu prohodíme zdrojovou a cílovou adresu MAC adresu z ethernetové hlavičky, zdrojovou a cílovou IP adresu z IP hlavičky, změníme typ ICMP na REPLY a spočítáme nový kontrolní součet pro IP hlavičku a ICMP hlavičku. Datová část ICMP zůstává nezměněna. Celá procedura se opakuje v nekonečné smyčce.

5.5.4 Výsledky testování

Výsledkem je aplikace, která je schopná odpovídat na ICMP dotazy typu ECHO, které jsou odesílány přes TAP adaptér, a zasílány zpět na TAP adaptér. Tím demonstrujeme správnou činnost ovladače, a tím tedy dílčí fáze celého převodu OpenVPN na Windows CE 6.0.

5.5.5 Nedostatky ovladače

Po spuštění ovladače v systému přijde v krátké době z neznámého důvodu od operačního systému (NDIS) zpráva na zařízení s požadavkem na vypnutí (power off). Tím se znemožní veškerá práce s adaptérem, až do manuálního zaslání požadavku na zapnutí (power on). Tuto proceduru je třeba udělat vždy jen jednou (po spuštění), podruhé požadavek na vypnutí již nepřijde. Z tohoto důvodu vznikl dávkový soubor `powerup`, který adaptér opět uvede do činnosti.

Kapitola 6

Závěr

V závěrečné kapitole zhodnotíme celý postup při realizaci projektu a dosažené výsledky. Nejprve shrneme původně uvažovaný cíl práce, v sekci „Výsledky projektu“ podrobněji popíšeme, co se opravdu podařilo udělat, s případnou možností navázat na výsledky mé práce. Na úplný závěr se čtenář dozví, jaký byl přínos diplomové práce.

6.1 Cíl projektu

Cílem této diplomové práce je převod implementace OpenVPN z operačního systému Windows XP na operační systém Windows CE 6.0. Převod obnáší seznámení se s implementací OpenVPN na operačním systému Windows XP a platformou Windows CE. Dále, obecná analýza tvorby ovladačů, zejména síťových, s upozorněním na specifika jednotlivých operačních systémů. Z toho vychází návrh, který již konkrétně popisuje převod aplikace na vestavěný systém NetDCU10 postaveném na OS Windows CE 6.0. Implementace a testování realizuje samotný převod, jejíž výsledkem je funkční spustitelná aplikace.

6.2 Výsledky projektu

Na počátku práce se zdálo reálné v rámci rozsahu diplomové práce celé zadání splnit. Z podrobné analýzy vyplynulo, že dílčí část převodu, tedy virtuální síťový ovladač TUN/TAP je natolik obsáhlý, aby dokázal zaplnit celou práci.

Ve snaze zachovat práci a výstupy v rozumné kvalitě bylo upuštěno od snahy kompletně realizovat úvodní požadavky, a důraz se kladl především na analýzu a převod síťového ovladače, který je jádrem celé komunikace v OpenVPN.

Začátek tohoto textu uvedl čtenáře obecně do podvědomí virtuálních privátních sítí, především jedné z jejich konkrétních implementací - OpenVPN. Dále popisoval nejnovější operační systém pro vestavěná zařízení od firmy Microsoft - Windows CE Embedded 6.0.

Z minimálních znalostí z oblasti tvorby ovladačů vycházela analýza. Výklad se snažil být stručný, avšak natolik informačně bohatý, aby zasáhnul většinu partií, se kterými se může tvůrce ovladačů, zejména síťových, potkat. Návrh se téměř výhradně soustředil na převod ovladačů TUN a TAP, řešení absence komunikačního prostředku ovladačů pod Windows XP - IRP, nemožnost přímo zpracovávat asynchronní požadavky na vstup/výstup, a také specifika tvorby ovladačů na Windows CE 6.0.

Implementace byla realizováním všech předchozích předpokladů a navržených řešení, jejíž výsledkem je plně funkční implementace virtuálního síťového ovladače běžícího v re-

žimu jádra - TAP. Ovladač TUN je pouze softwarová obálka běžící v uživatelském režimu, prostředník pro komunikaci OpenVPN s ovladačem TAP. Neexistující implementace OpenVPN znemožnila TUN otestovat, proto byl nahrazen principálně stejným řešením.

Tím je aplikace s názvem TestTAP, která je schopna získat data ze síťového adaptéru, určitým způsobem zpracovat, a vrátit zpět. Přesněji řečeno, TestTAP reaguje na ARP REQUEST a ICMP ECHO pakety, na které generuje odpověď. Původce těchto paketů je program ping, standardně dostupný na operačních systémech podporujících síťovou komunikaci. Ping směřuje dotazy na dostupnost uzlu ležícího ve stejné síti jako TAP adaptér právě na tento adaptér, kde se aplikace TestTAP tváří právě jako dotazovaný uzel a vrací ICMP REPLY (ARP REPLY) pakety zpět na TAP.

Program TestTAP tedy ověřil, že byl převod ovladače TAP na zařízení NetDCU10 s operačním systémem Windows CE 6.0 správný, a lze považovat tuto fázi převodu za kompletní.

6.3 Pokračování v projektu

Nabízí se poměrně velké možnosti, jak na projekt navázat. V první řadě by bylo vhodné se detailně seznámit s jádrem OpenVPN, vnitřními komunikačními mechanismy, navrhnout a převést aplikaci společně s ovladačem TUN na Windows CE 6.0. Mnohé již napověděla tato práce, ze které lze využít jak některé informace, tak praktický výsledek - síťový ovladač TAP.

Dále je důležité zmínit, že TAP zařízení, ačkoli zavádí specifický komunikační mechanismus, není závislé na žádné jiné součásti a aplikaci. Lze jej tedy využít pro jiné účely, příbuzné projekty, případně pouze pro experimentování. K dispozici jsou dostatečně okomentované zdrojové kódy, které lze v případě potřeby upravit a přizpůsobit vnější chování TAP ke konkrétním účelům.

6.4 Přínos projektu

Třebaže praktický výsledek - ovladač TAP - již nebude dále využíván, nebude projekt bezcenný. Jako zásadní přínos považuji informace seskupené v této práci. Hlavním důvodem je fakt, že nikde neexistuje programová dokumentace popisující schéma komunikace OpenVPN s okolím, stejně tak jako architektura ovladače TUN/TAP. Studování principů funkce metodami reverzního inženýrství není pohodlné, proto tedy doufám, že jsem dalším zájemcům tímto usnadnil práci.

Věřím, že tato práce bude užitečná i pro začínající tvůrce ovladačů, především na Windows CE 6.0, k čemuž jim tento text bude sloužit jako úvodní zdroj informací.

Dodatek A

Instalace ovladače na Windows CE 6.0

V této příloze popíšeme podrobněji postup při registraci a spuštění ovladače na Windows CE 6.0 včetně konkrétních příkladů.

A.1 Zápis od registru

Pomocným nástrojem `Remote registry editor` se připojíme k vestavěnému zařízení NetDCU10 a nastavíme příslušné hodnoty v registrech tak, abychom mohli ovladač nahrát do paměti.

A.1.1 Registrace miniportu

Nejprve zapíšeme informace pro rozhraní NDIS do klíče `HKEY_LOCAL_MACHINE\Comm`, aby mohl být ovladač spuštěn. Symbolem `//` je označen komentář.

1. TAP adaptér:

```
[HKEY_LOCAL_MACHINE\Comm\TAP Adapter] //klíč registru
"Group"="NDIS"                        //třída ovladačů NDIS
"ImagePath"="\\FFSDISK\\TAP_CE.dll"   //umístění ovladače
"DisplayName"="Tap virtual adapter"   //popis adaptéru

[HKEY_LOCAL_MACHINE\Comm\TAP Adapter\Linkage]
"Route"="TAP Adapter1"                //vazba na instance adaptéru
```

2. Instance TAP Adaptéru:

```
[HKEY_LOCAL_MACHINE\Comm\TAP Adapter1]  //klíč registru
"Group"="NDIS"                           //třída ovladačů NDIS
"ImagePath"="\\FFSDISK\\TAP_CE.dll"     //umístění ovladače
"DisplayName"="Tap virtual adapter instance"  //popis adaptéru

[HKEY_LOCAL_MACHINE\Comm\TAP Adapter1\Parms]  //klíč s parametry
"StreamName"="TAP"            //zkrácené jméno ovladače
```

```
"BusNumber"=dword:00000000    //číslo sběrnice, pro TAP hodnota 0
"BusType"=dword:00000000      //typ sběrnice, pro TAP hodnota 0
"StreamIndex"=dword:00000001  //číslo instance
```

3. Zápis do nastavení TCP/IP:

```
[HKEY_LOCAL_MACHINE\Comm\Tcpip\Linkage]
"Bind"="TAP Adapter1"               //připojené adaptéry k protokolu
```

A.1.2 Registrace streamového zařízení

Dále musíme vytvořit zápis do registru pro streamovou část ovladače. Adresu s klíčem obsahujícím informace o TAP zařízení se poté předají jako jeden z parametrů funkce `ActivateDevice(Ex)`. Pokud bychom chtěli streamovou část načíst vždy po startu systému, zapíší se informace do klíče `HKEY_LOCAL_MACHINE\Drivers\BuiltIn`. V našem případě je vhodné, aby se instance TAP zařízení načetla až po spuštění miniportu, proto informace zapíšeme do vlastního klíče.

```
[HKEY_LOCAL_MACHINE\MyDrivers\TAP] //klíč pro streamovou část
"Dll"="\\FFSDISK\\TAP_CE.dll"      //umístění ovladače
"Prefix"="TAP"                     //třípísmenná zkratka
```

A.2 Spuštění ovladače

Jestliže je ovladač v registrech systému korektně zaregistrovaný, můžeme přistoupit ke spuštění. V případě automatického spuštění není třeba udělat nic jiného, než zapsat hodnoty do registrů na příslušná místa. Možnosti manuálního načtení popíšeme detailněji.

A.2.1 Manuální spuštění

V případě manuálního spuštění máme dvě možnosti. První je spuštění vlastním programem, tzn. zavolání funkce `DeviceIoControl` pro zaregistrování TAP miniportu. Ten vnitřně automaticky registruje streamovou část.

Druhou a jednodušší možností je využít utility `ndisconfig` operačního systému, která pomocí příslušných parametrů příkazové řádky zaregistruje miniport.

- Programové spuštění:

```
//otevření NDIS zařízení
hNdis = CreateFile(DD_NDIS_DEVICE_NAME,
                   GENERIC_READ | GENERIC_WRITE,
             FILE_SHARE_READ | FILE_SHARE_WRITE,
             NULL, OPEN_ALWAYS, 0, NULL);

//zaregistrování TAP miniportu
bResult = DeviceIoControl(hNdis,
     IOCTL_NDIS_REGISTER_ADAPTER,
     L"TAP Adapter\0TAP Adapter1\0\0",
     sizeof(L"TAP Adapter\0TAP Adapter1\0\0"),
```

```
      NULL,
      0,
      NULL,
      NULL);


//otevření streamové části TAP zařízení
hTap = CreateFile(TAPNAME,
                  GENERIC_READ | GENERIC_WRITE,
          0, 0,
                  OPEN_EXISTING,
                  FILE_ATTRIBUTE_NORMAL,
                  NULL);

//práce s instancí TAP
//ReadFile(hTap,...), WriteFile(hTap,...)

//uzavření streamové části TAP zařízení
CloseHandle(hTap);

//odregistrování TAP miniportu ze systému
bResult = DeviceIoControl(hNdis,
    IOCTL_NDIS_DEREGISTER_ADAPTER,
    L"TAP Adapter1\0\0",
    sizeof(L"TAP Adapter1\0\0"),
    NULL,
    0,
    NULL,
      NULL);

//uzavření NDIS zařízení
CloseHandle(hNdis);
```

- Spuštění pomocí utility `ndisconfig`:

```
  //registrace TAP miniportu
  ndisconfig adapter add "tap adapter" "tap adapter1"

  //odregistrování TAP miniportu
  ndisconfig adapter del "tap adapter1"

  //popisovanou chybu se správou napájení vyřešíme tímto příkazem
  ndisconfig power set "tap adapter1" D0
```

Dodatek B

Ladění ovladače

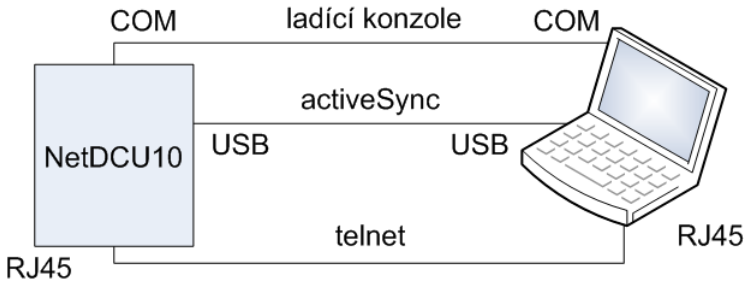
V kapitole s názvem ladění ovladače nejprve popíšeme schéma zapojení zařízení NetDCU10 k počítači a nastavení konzole pro výpis ladících informací. V druhé sekci bude uveden konkrétní příklad debugovacích informací generovaných ovladačem TAP za pomocí makra `DEBUGMSG`.

B.1 Zapojení a nastavení

Vývoj a ladění na NetDCU10 není příliš pohodlné, proto v této sekci uvedeme možnosti, které vestavěné zařízení nabízí.

B.1.1 Schéma zapojení

Obrázek B.1 zobrazuje zapojení NetDCU10 k počítači pro plnohodnotný vývoj.



Obrázek B.1: Schéma zapojení NetDCU10 k počítači

Skládá se ze tří částí:

1. **Komunikace přes ethernetový kabel** - z důvodu absence dotykového displeje na zařízení je komunikace přes telnet jedinou možností, jak spouštět vzdáleně aplikace. Postup je následující:
   - Spuštění programu `cmd` ve Windows XP
   - Zadání příkazu `telnet <IP adresa>`, kde IP adresa je adresou ethernetového rozhraní vestavěného zařízení, ve výchozím nastavení `169.254.4.165`.

- Na straně NetDCU10 je spuštěn démon `telnetd`, který naslouchá na portu 22 a funguje téměř jako každý jiný interpret příkazů.

2. **Komunikace přes USB kabel** - komunikace probíhá přes synchronizační program ActiveSync, který slouží především jako prostředek pro přenos souborů na vestavěné zařízení (obdoba průzkumníku na Windows XP), a také jako rozhraní pro následující aplikace:

   - Remote Registry Editor - obdoba programu `regedit` na Windows XP, ovšem po editaci klíče je nutné změny uložit manuálně přes utilitu pro správu NetDCU10 s názvem `ndcucfg`, a zadat příkaz `reg save`
   - Remote Process Viewer - prohlížení běžících procesů na NetDCU10 a násilné ukončení. Démon `telnetd` nepodporuje klávesovou zkratku Ctrl + C pro přerušení aktuálně běžícího procesu v konzoli.

3. **Komunikace přes sériový kabel** - pro výpis ladících informací a nastavení zavaděče systému. Bližší popis je uveden v následující podsekci.

B.1.2 Nastavení ladící konzole

Pro komunikaci s ladícím portem COM lze využít libovolnou softwarovou konzoli, která umí komunikovat přes sériový port. V případě NetDCU10 lze využít dodávanou aplikaci `DcuTerm`. Ve výchozím nastavení je debugovací port přesměrován na COM1. Další potřebná nastavení:

- Přenosová rychlost (baud rate) - 38400
- Datové bity (data bits) - 8
- Stop bity (stop bits) - 1
- Parita - žádná
- Řízení toku (flow control) - žádné

Poté lze port COM1 otevřít a sledovat ladící výpisy.

B.1.3 Použití makra DEBUGMSG

Standardní makro DEBUGMSG pro výpis ladících informací je použitelné na většině operačních systémů od Microsoftu. Syntaxe je následující: `DEBUGMSG(condition, (text))`, kde `condition` je podmínka nabývající hodnot true nebo false, v případě true je text vypsán. Tělo zprávy označené jako `text` má zápis stejný jako funkce `printf`, tedy formátovací řetězec a parametry. Formátovací řetězec je třeba ohraničit písmenem `L` označujícím, že výpis bude ve formátu unicode. Příklad z TAP ovladače:

```
DEBUGMSG(TRUE,(L"\t[%s] pBuffer is NULL for TAP_READ\n",
         NAME (l_Adapter)));
```

B.2 Ladění pomocí výpisů

Nyní bude následovat ladící výpis po spuštění ovladače a práci s ním, včetně komentování příslušných akcí.

```
DLL_PROCESS_ATTACH //načtení procesu s ovladačem do paměti
DRIVER ENTRY       //registrace miniportu
  [TAP] version [9.1] My  080:32 0:32 registered miniport successfully
  Registry Path: '\Comm\tap adapter' //klíč s údaji o ovladači

ADAPTER_CREATE function   //vytvoření miniportu
  [TAP] NdisReadConfiguration (MiniportName=TAP ADAPTER1) //jméno adaptéru
  Adapter name: [TAP ADAPTER1]
  [TAP ADAPTER1] Using MAC 0:ff:0:0:0:0

CREATE TAP DEVICE   //vytvoření streamové části
  [TAP] version [9.1] creating tap device: TAP ADAPTER1
  TAP REG STRING NAME: '\MyDrivers\TAP' //klíč s údaji o zařízení

TAP_Init  //aktivace TAP zařízení, po volání ActivateDeviceEx()
  TAP DEVICE SUCCESSFULLY ACTIVATED
  [TAP ADAPTER1] successfully created TAP device [TAP DEVICE:1]

//změna vlastností TAP adaptéru systémem
ADAPTER_MODIFY function
  [TAP ADAPTER1] Setting [OID_GEN_CURRENT_LOOKAHEAD] to [128]
ADAPTER_MODIFY function
  [TAP ADAPTER1] Setting [OID_GEN_CURRENT_PACKET_FILTER] to [0x0b]
ADAPTER_MODIFY function
  [TAP ADAPTER1] Setting [OID_802_3_MULTICAST_LIST]

//síť odesílá paket přes miniport, prozatím není otevřena instance TAP
ADAPTER_TRANSMIT function
  ADAPTER_TRANSMIT no opened TAP
  ADAPTER_TRANSMIT -exit success

TAP_Open //otevření TAP funkcí CreateFile z uživatelského režimu
  [TAP ADAPTER1] [TAP] release [9.1] open request (m_TapOpens=0)
  TAP_Open m_TapOpens == 1

TAP_Read //požadavek na čtení z uživatelského režimu
  NewReadRequest function  //nová interní žádost
  Sizeof packet buffer: 0 OK //žádný paket není k dispozici
  TAP_READ, l_PacketBuffer is NULL, saving request //uložení požadavku

//spuštění ping na IP adresu uzlu v síti TAP
ADAPTER_TRANSMIT function
  //ARP paket ke zjištění MAC cílového uzlu
  AdapterTransmit ARP src= 0:ff:0:0:0:0  dest= ff:ff:ff:ff:ff:ff
```

```
                        OP=0x0001  M=0x0001(6)  P=0x0800(4)
                        MacSrc= 0:ff:0:0:0:0  MacDest= 0:0:0:0:0:0
                        IPSrc= 10.0.0.1  IPDest= 10.0.0.3

  PACKET PUSHED OK  //uložení paketu
  CompleteReadRequest function //spojení požadavku na čtení s paketem
  SetEvent //signál uživatelskému režimu o získaném paketu
  DeleteReadRequest function //smazání požadavku na čtení

TAP_Write  //odpověď na ARP z uživatelského režimu
  TAP WRITE ARP src= 0:0:0:0:0:0  dest= 0:ff:0:0:0:0
                OP=0x0002  M=0x0001(6)  P=0x0800(4)
                MacSrc= 0:0:0:0:0:0  MacDest= 0:ff:0:0:0:0
                IPSrc= 10.0.0.3  IPDest= 10.0.0.1

ADAPTER_TRANSMIT function
  //ICMP ECHO paket, velikost 74 bytů
  AdapterTransmit IPv4 ICMP[74]  ipproto=1  10.0.0.1  ->  10.0.0.3
  PACKET PUSHED OK

TAP_Read  //požadavek na čtení
  NewReadRequest function
  Sizeof packet buffer: 1 OK //máme k dispozici paket
  TAP_READ, l_PacketBuffer OK
  CompleteReadRequest function
  SetEvent
  DeleteReadRequest function

TAP_Write //odpověd na ICMP - REPLY z uživatelského režimu
  TAP WRITE IPv4 ICMP[74]  ipproto=1  10.0.0.3  ->  10.0.0.1

TAP_Read
  NewReadRequest function
  Sizeof packet buffer: 0 OK //nemáme paket k dispozici
  //uložení a pozdější zpracování
  TAP_READ, l_PacketBuffer is NULL, saving request //čekání na další paket

//vše se opakuje ještě třikrát (standardně 4 ICMP pakety)
.
.
.


TAP_Close //CloseHandle na TAP z uživatelského režimu
  [TAP ADAPTER1] [TAP] release [9.1] close/cleanup request

FLUSH_QUEUES //vyprázdnění nezpracovaných paketů a požadavků na čtení
  DeleteReadRequest function
```

```
  [TAP DEVICE:1] [TAP] FlushQueues n_IRP=[1,1,16] n_Packet=[0,1,64]

ADAPTER HALT //zastavení adaptéru
  [TAP ADAPTER1] is being halted
  DESTROY TAP DEVICE[TAP DEVICE:1] Destroying tap device


TAP_Deinit //deregistrace TAP zařízení - DeactivateDevice()
  [TAP] Deregistering TAP device, status=1
  [TAP ADAPTER1] Freeing Resources

TAP DRIVER UNLOAD //odstranění miniportu ze systému
  [TAP] version [9.1] My  08 [TAP] version 0:32 unloaded

DLL_PROCESS_DETACH //odstranění ovladače z paměti
```

Dodatek C

Obsah přiloženého CD

Popis adresářové struktury na CD přiloženém k technické zprávě.

C.1 Adresářová struktura

- **Ovladac** - praktické výstupy projektu
  - **DLL** - dynamická knihovna ovladače TAP
  - **Zdrojove_kody** - projekt vývojového prostředí Visual Studio 2005
  - **Pomocne_aplikace** - pomocné aplikace včetně zdrojových kódů
    - **TestTAP** - aplikace testující funkčnost ovladače TAP
    - **AddIpAddress** - aplikace přidělující TAP zařízení IP adresu
    - **Davkove_soubory** - dávky pro spuštění TAP zařízení
- **Technicka_zprava** - technická zpráva ve formátu pdf

Literatura

[1] International Engineering Consortium, *On-Line Education: WPF: Virtual Private Networks (VPNs), Introduction*. [online], [cit. 2007-12-28].
URL `http://www.iec.org/online/tutorials/vpn/topic01.html`

[2] Microsoft Corporation, *Handling IRPs: What Every Driver Writer Needs to Know*. [online], poslední úprava dne 2.10.2007, [cit. 2007-12-20].
URL `http://www.microsoft.com/whdc/driver/kernel/IRPs.mspx`

[3] WIKIPEDIA, the free encyclopedia, *IPsec*. [online], poslední úprava dne 26.12.2007, [cit. 2007-12-28].
URL `http://en.wikipedia.org/wiki/IPsec`

[4] WIKIPEDIA, the free encyclopedia, *OpenVPN*. [online], poslední úprava dne 23.12.2007, [cit. 2007-12-28].
URL `http://en.wikipedia.org/wiki/OpenVPN`

[5] WIKIPEDIA, the free encyclopedia, *TUN/TAP*. [online], poslední úprava dne 26.4.2007, [cit. 2007-12-20].
URL `http://en.wikipedia.org/wiki/TUN/TAP`

[6] Microsoft TechNet, *Virtual Private Networking: An Overview*. [online], September 2001, [cit. 2007-12-28].
URL `http://technet.microsoft.com/en-us/library/bb742566.aspx`

[7] Microsoft Corporation, *Windows Driver Kit*. [online], 2006, [cit. 2007-12-20].
URL `http://msdn2.microsoft.com/en-us/library/aa972908.aspx`

[8] Microsoft Corporation, *MSDN - Event Objects*. [online], 4 2008, [cit. 2008-04-12].
URL `http://msdn2.microsoft.com/en-us/library/ms682655(VS.85).aspx`

[9] Boling, D.: *Programming Windows Embedded CE 6.0*. Redmond, Washington, USA: Microsoft Press, Čtvrté vydání, 2008, ISBN 98052-6399.

[10] Feilner, M.: *OpenVPN - Building and Integrating Virtual Private Networks*. Olton, Birmingham, UK: Packt Publishing, April 2006, ISBN 1-904811-85-X.

[11] Heil, D. G.: *Writing Windows CE Device Drivers: Principle to Practice*. [online], 9 2006, [cit. 2007-12-20].
URL `http://www.embedded.com/columns/technicalinsights/-193003990?_requestid=752707`

[12] Loh, S.: Microsoft Corporation, *Memory marshalling in Windows CE*. [online], 12 2006, [cit. 2008-04-12].
URL `http://www.windowsfordevices.com/articles/AT5831219184.html`

[13] Pužmanová, R.: *Virtuální privátní sítě pro vzdálený přístup*. [online], September 2006, [cit. 2007-12-28].
URL `http://www.dsl.cz/clanky-dsl/clanek-511/-Virtualni-privatni-site-pro-vzdaleny-pristup`

[14] Yonan, J.: *OpenVPN Security Overview*. [online], [cit. 2007-12-20].
URL `http://openvpn.net/security.html`