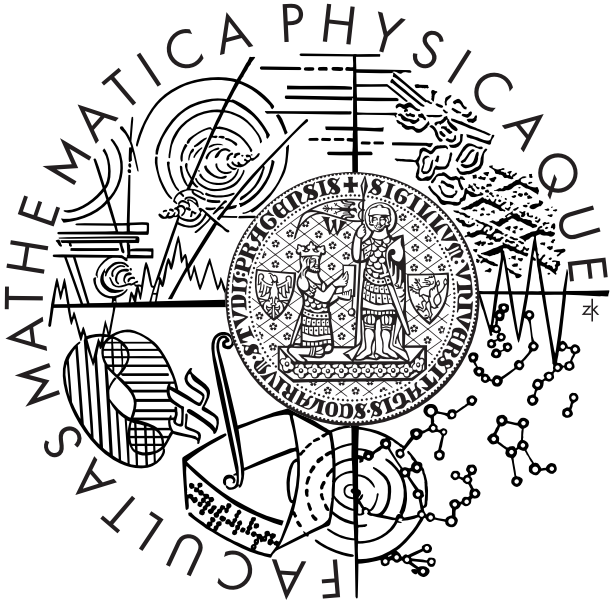
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Lenka Trochtová

Rozhraní pro ovladače zařízení v HelenOS

Katedra softwarového inženýrství

Vedoucí diplomové práce: Mgr. Martin Děcký

Studijní program: Informatika, Softwarové systémy

Děkuji svému vedoucímu Martinu Děckému za užitečné připomínky jak k návrhu, tak k textu práce. Děkuji i ostatním členům týmu, který vyvíjí operační systém HelenOS, za jejich ochotu zodpovídat mé dotazy, jmenovitě děkuji Jakubu Jermářovi, Jiřímu Svobodovi a Pavlu Římskému. Děkuji svému zaměstnavateli, firmě S.ICZ, za pochopení a podporu.

Prohlašuji, že jsem svou diplomovou práci napsala samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne Lenka Trochtová

Obsah

Seznam obrázků

Název práce: Rozhraní pro ovladače zařízení v HelenOS
Autor: Lenka Trochtová
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí diplomové práce: Mgr. Martin Děcký
e-mail vedoucího: Martin.Decky@mff.cuni.cz
Abstrakt: Uvedení do problematiky ovladačů zařízení, význam jednotného rozhraní pro ovladače zařízení v rámci operačního systému. Stručný popis základních principů hardware z pohledu programátora. Přehled vlastností operačního systému HelenOS – základní funkční bloky mikrojádra a stávající podpora ovladačů zařízení v uživatelském prostoru, např. podpora pro přístup k registrům zařízení a zpracování přerušení z uživatelského prostoru. Obecné funkce rozhraní pro ovladače zařízení a jejich vztah k potřebám operačního systému HelenOS. Problémy obvykle řešené současnými driver frameworky a známé přístupy k jejich řešení – device-centric vs. driver-centric přístup k přiřazování zařízení a ovladačů, identifikátory modelů zařízení, počet podporovaných tříd zařízení, vstupní body ovladače, reprezentace stromu zařízení. Návrh a implementace řešení – hierarchická správa zařízení, automatická detekce zařízení, předdefinovaná rozhraní pro přístup k zařízení, třídy zařízení, řízení životního cyklu ovladače a přiřazování ovladače k zařízení, instalace a konfigurace ovladače. Popis několika existujících řešení a jejich srovnání – ovladače zařízení v operačních systémech Windows, Solaris a Linux. Zhodnocení a možnosti dalšího vývoje.
Klíčová slova: ovladač zařízení, framework, HelenOS, operační systém

Title: Device drivers interface in HelenOS system
Author: Lenka Trochtová
Department: Department of Software Engineering
Supervisor: Mgr. Martin Děcký
Supervisor's e-mail address: Martin.Decky@mff.cuni.cz
Abstract: An introduction into device drivers, the significance of the unified interface for device drivers within the operating system. A brief description of the basic principles of hardware from the programmer's point of view. The HelenOS operating system overview – the description of the basic funcional blocks of microkernel, the current support of user-space device drivers, e.g. the support for accessing device's registers and interrupt processing from user-space. Common functions of the interface for device drivers and their relation to the needs of the HelenOS operating system. The problems usually solved by contemporary driver frameworks and the known ways of solving them – device-centric vs. driver-centric device-to-driver matching, device IDs, the number of supported device classes, device driver entry points, the representation of a device tree. Design and implementation – hierarchical device management, device auto-detection, predefined interfaces for access to devices, device classes, controlling the device driver's life cycle and device-to-driver matching, the installation and the configuration of a device driver. The description of device drivers in some contemporary operating system and their comparison – device drivers in Windows, Solaris and Linux. The evaluation and the possibilities of future development.
Keywords: device driver, framework, HelenOS, operating system

Kapitola 1

Úvod

Jednou ze základních funkcí operačního systému je poskytovat aplikacím vyšší úroveň abstrakce nad hardware. Operační systém umožňuje aplikacím přistupovat k zařízením počítače prostřednictvím jednotného, platformně nezávislého rozhraní. Aplikace tedy nemusejí znát specifické vlastnosti dané architektury ani konkrétního modelu zařízení. Důležitou součást operačního systému, která tuto abstrakci umožňuje, tvoří subsystém ovladačů zařízení.

Většina v současné době používaných operačních systémů je postavena na monolitickém jádře, kde součástí jádra jsou mimo jiné i ovladače zařízení. Zkušenosti s těmito systémy ukazují, že ovladače zařízení v monolitickém operačním systému jsou nejčastějším zdrojem chyb v jádře a mají tak značně nepříznivý dopad na stabilitu celého systému.

Ve snaze omezit tento problém začaly některé z těchto systémů podporovat vývoj ovladačů vybraných typů zařízení v uživatelském prostoru. Podstatná část ovladačů zařízení v těchto systémech však nadále zůstává součástí jádra.

Důslednější řešení tohoto problému nabízejí operační systémy postavené na mikrojádru, které se snaží nejen většinu ovladačů zařízení, ale i jiné součásti operačního systému, které u monolitických systémů bývají obvykle součástí jádra, umisťovat do uživatelského prostoru. Operační systémy tohoto typu se zatím nedočkaly masového rozšíření, ale jsou předmětem výzkumu. Jedním z takových výzkumných projektů je operační systém HelenOS.

HelenOS je výzkumný operační systém vyvíjený převážně studenty a absolventy Matematicko-fyzikální fakulty Univerzity Karlovy v Praze. Cílem projektu HelenOS je vytvořit kompletní a reálně použitelný operační systém a v průběhu jeho vývoje dát prostor pro studium a praktické ověření nových myšlenek v oblasti operačních systémů. V rámci projektu HelenOS vzniká celá řada studentských semestrálních projektů a diplomových prací.

Operační systém HelenOS je postavený na mikrojádru a již od samého začátku byl vyvíjen s důrazem na platformní neutralitu a snadnou přenositelnost. V průběhu svého vývoje byl portován na celou řadu architektur procesorů – konkrétně na AMD64 (x86-64), ARM, IA-32, IA-64 (Itanium), 32-bitový MIPS, 32-bitový PowerPC a SPARC V9.

Před vznikem této práce již v operačním systému HelenOS existovala podpora pro psaní jednoduchých ovladačů zařízení, ovladačům ale chybělo jednotné a ucelené rozhraní, podpora pro hierarchický model zařízení a autokonfiguraci, což se ukázalo pro další vývoj značně limitující. Kromě toho byly existující ovladače v uživatelském prostoru závislé na informacích předávaných z jádra.

Náplní této diplomové práce je vytvořit návrh obecného rozhraní pro ovladače

v systému HelenOS, který by zmíněné nedostatky odstranil a současně byl dostatečně obecný, aby zachoval dobré vlastnosti systému HelenOS, zejména jeho platformní neutrálnost. Práce zároveň demonstruje návrh driver frameworku v uživatelském prostoru v operačním systému postaveném na mikrojádru, popisuje jednotlivé fáze návrhu od analýzy problémů, které musí obecně driver framework řešit, přes diskusi alternativ při řešení těchto problémů až po detailnější návrh vybraného řešení a prototypovou implementaci přizpůsobenou operačnímu systému HelenOS.

1.1 Zdrojové soubory

Veškeré informace týkající se operačního systému HelenOS včetně zdrojových souborů lze najít na oficiálních stránkách projektu http://www.helenos.org.

Zdrojové soubory se nacházejí v úložišti systému pro správu verzí Bazaar, které je rozděleno na několik větví. Zdrojové soubory z hlavní vývojové větve lze z úložiště stáhnout příkazem:

bzr branch bzr://bzr.helenos.org/mainline HelenOS

Pro tuto diplomovou práci byla zřízena samostatná větev, jejíž zdrojové soubory lze z úložiště načíst příkazem:

bzr branch lp:~helenos-dd/helenos/dd HelenOS_dd

1.2 Organizace textu

Kapitola 2 vysvětluje základní pojmy z oblasti hardware.

Kapitola 3 popisuje ty části architektury systému HelenOS, které je třeba znát pro další pochopení práce. Tato kapitola taktéž popisuje podporu pro ovladače zařízení v HelenOS v době před vznikem této práce.

Kapitola 4 popisuje nejčastější problémy řešené rozhraním pro ovladače zařízení a vyjmenovává některé typické funkce driver frameworku.

Kapitola 5 z hlediska operačního systému HelenOS rozebírá jednotlivé problémy řešené rozhraním pro ovladače zařízení, které byly uvedené v předchozí kapitole, a na základě potřeb systému HelenOS vytyčuje cíle výsledného návrhu.

Kapitola 6 se vrací k jednotlivým cílům a problémům z předchozí kapitoly, vyjmenovává známé přístupy k jejich řešení a rozebírá výhody a nevýhody těchto přístupů jak z obecného hlediska, tak z hlediska jejich použitelnosti v operačním systému HelenOS.

Předmětem kapitoly 7 je návrh řešení dříve uvedených problémů pro operační systém HelenOS. Tato kapitola rozebírá varianty řešení různých problémů, které byly při návrhu uvažovány, a vysvětluje důvody, které vedly k výběru výsledného řešení.

Kapitola 8 popisuje prototypovou implementaci rozhraní pro ovladače zařízení v operačním systému HelenOS. Součástí textu kapitoly je popis rozdělení funkcí driver frameworku do několika komponent, popis jednotlivých součástí frameworku a jejich vzájemné spolupráce, ukázka jednoduchého ovladače a popis integrace s původním rozhraním pro ovladače zařízení.

Kapitola 10 obsahuje zhodnocení výsledného návrhu a možnosti jeho budoucího vylepšení.

Kapitola 2

Hardware – základní pojmy

V této kapitole si vysvětlíme některé základní pojmy z oblasti hardware, které jsou nutné pro pochopení následujících kapitol této práce.

Registry zařízení tvoří rozhraní mezi softwarem a hardwarem. Lze si je představit jako paměťové buňky, jejichž čtením či modifikací je možné ovládat zařízení. Význam jednotlivých bitů v rámci registrů zařízení je definován specifikací daného modelu hardware, registry zařízení lze zjednodušeně rozdělit do tří funkčních kategorií:

- *řídicí registry* – zápisem do nich lze měnit režim práce zařízení
- *stavové registry* – čtením z nich lze zjišťovat aktuální stav zařízení (včetně např. chyb)
- *datové registry* – zápisem do nich lze docílit výstupu na zařízení, čtením z nich vstupu ze zařízení

Podle způsobu přístupu lze registry rozdělit na:

- *Paměťově mapované registry* – z hlediska přístupu se neliší od operační paměti počítače – jsou adresovatelné prostřednictvím adres stejného typu a přistupuje se k nim pomocí stejných instrukcí procesoru. Tyto registry zařízení jsou tedy součástí fyzického adresového prostoru stejně jako fyzická paměť počítače.
- *Registry v I/O adresovém prostoru* – jsou součástí samostatného adresového prostoru a přistupuje se k nim pomocí speciálních instrukcí odlišných od instrukcí pro přístup k paměti. Samostatný I/O adresový prostor je podporován jen na některých architekturách procesorů – např. na procesorech vyráběných firmou Intel.

Přerušení (interrupt) je mechanismus, jímž si zařízení může vyžádat pozornost procesoru. Zjednodušeně si lze představit, že zařízení vyvolá přerušení tak, že přivede signál na vstupní pin procesoru, a tím donutí procesor, aby dočasně přerušil právě vykonávanou činnost a místo ní se věnoval obsluze události, na niž ho zařízení takto upozornilo. Rozlišujeme dva způsoby signalizace přerušení – úrovní a hranou.

- *Přerušení signalizovaná hranou* (*edge-triggered interrupts*) mají podobu dočasné změny napětí na vodiči, který vede od zařízení za účelem signalizace přerušení.
- *Přerušení signalizovaná úrovní* (*level-triggered interrupts*) mají podobu trvalé změny napětí na vodiči. Tato změna trvá, dokud není zařízení potvrzeno přijetí přerušení (to se děje obvykle čtením nebo modifikací některého z registrů zařízení).

Vodič určený pro signalizaci přerušení obvykle nevede ze zařízení přímo na vstupní pin procesoru, místo toho se na cestě mezi zařízením a procesorem vyskytuje tzv. řadič přerušení. *Řadič přerušení* je čip, jehož úkolem je serializovat přerušení z více zdrojů a přivádět je na jediný vstupní pin procesoru. Jednotlivým zdrojům přerušení přiřazuje prioritu, a pokud jsou současně signalizována přerušení z více zdrojů, informuje o nich procesor v pořadí určeném prioritou těchto zdrojů. Kromě toho řadič přerušení umožňuje přerušení z vybraných zdrojů dočasně zakázat, což znamená, že tato přerušení pak nejsou signalizována procesoru. Přerušení ze zakázaných zdrojů nejsou signalizována procesoru, dokud nejsou dané zdroje přerušení opětovně povoleny.

Plug and play (PnP) je mechanismus, který umožňuje automatickou detekci hardware a přidělování systémových prostředků zařízením bez zásahu uživatele a bez nutnosti fyzické konfigurace hardware (např. ručního nastavování jumperů). Rozlišujeme dva typy PnP:

- *Boot-time plug and play* je automatická detekce a konfigurace hardware v průběhu startu systému. Tímto způsobem se typicky detekují a konfigurují např. zařízení připojená na sběrnici PCI (Peripheral Component Interconnect).
- *Hotplug* je typ PnP, který podporuje přidávání a odebírání zařízení za běhu systému. Příkladem sběrnice, která podporuje hotplug funkcionalitu, je sběrnice USB (Universal Serial Bus).

U některých zařízení bez podpory PnP lze automatickou detekci simulovat tak, že se prozkoumají adresy, na nichž se zařízení v daném systému typicky vyskytuje. O prozkoumání zadaných adres je většinou požádán ovladač určený pro ovládání zařízení daného typu a prozkoumání obvykle provádí pokusným čtením a zápisem několika málo dat z/do registrů předpokládaného zařízení. Odpovídá-li výsledek těchto operací hardwarové specifikaci zařízení, ovladač systému oznámí, že zařízení je přítomno. V opačném případě se předpokládá, že zařízení na dané adrese přítomno není, nebo nefunguje správně. Podobným způsobem se v případě, že je na dané adrese zařízení nalezeno, může zkoumat, jaké přerušení zařízení používá. Tento způsob ověření přítomnosti a správné funkčnosti zařízení se někdy označuje jako *device probing*.

Kapitola 3

Operační systém HelenOS

Tato kapitola obsahuje stručný úvod do architektury operačního systému HelenOS. Účelem této kapitoly není poskytnout čtenáři úplný a vyčerpávající popis celého systému, zaměřuje se spíše na ty části architektury, které jsou nutné pro pochopení dalšího textu a pro orientaci ve zdrojových souborech prototypové implementace. Detailnější popis systému obsahuje dokumentace k architektuře systému HelenOS [helenos].

> POZNÁMKA
>
> Jelikož HelenOS je operační systém založený na mikrojádru, budeme se v následujícím textu držet zažité terminologie a místo termínu proces, který se používá spíše v případě monolitických systémů, budeme používat označení *task*.

3.1 Architektura systému

HelenOS je multiserverový operační systém postavený na vlastním mikrojádru. To znamená, že jen nezbytná část funkcionality operačního systému je umístěna v jádře a zbytek služeb operačního systému implementuje řada oddělených serverových tasků běžících v uživatelském prostoru, které s klientskými aplikacemi komunikují prostřednictvím meziprocesové komunikace. Toto rozdělení funkcí operačního systému umožňuje lepší modularitu a spolehlivost. Jelikož každý ze serverových tasků má vlastní oddělený adresní prostor, má pád takového tasku nepříznivý vliv jen na ty aplikace, které na něm – ať už přímo či nepřímo – závisejí.

3.1.1 Jádro

Jádro systému HelenOS umožňuje běh vláken, jejich plánování a základní synchronizaci, obsahuje správu paměti a podporu pro tasky, z nichž každý má vlastní adresní prostor a může sdružovat jedno nebo více vláken.

Jádro umožňuje jednotlivým taskům navzájem komunikovat prostřednictvím meziprocesové komunikace (*inter-process communication*, zkráceně *IPC*). Prostředky meziprocesové komunikace v systému HelenOS zahrnují posílání krátkých zpráv s omezeným počtem celočíselných parametrů, kopírování delších paměťových úseků nebo

sdílení paměti mezi komunikujícími tasky. Meziprocesová komunikace v operačním systému HelenOS je asynchronní.

Mezi další funkce jádra, které stojí za zmínění, patří `sysinfo`, které slouží jako obecný mechanismus předávání informací typu klíč-hodnota z jádra do uživatelského prostoru, a kconsole (kernel console), což je nástroj používaný k interaktivnímu ladění jádra. Pro podporu nástroje kconsole – tedy pro potřeby ladění – jádro obsahuje také několik ovladačů základních vstupně-výstupních zařízení.

Zdrojové soubory jádra jsou rozděleny na tři části – na část platformně závislou, část sdílenou více platformami a část platformně zcela neutrální. Platformně závislá část je rozčleněna na části odpovídající jednotlivým podporovaným architekturám procesorů.

3.1.2 Uživatelský prostor

Mezi služby operačního systému implementované v uživatelském prostoru patří souborové systémy (implementující virtuální souborový systém a konkrétní souborové systémy – tmpfs, FAT, devfs), virtuální konzole, loader, síťování, ovladače zařízení a device mapper. Všechny tyto služby jsou implementovány jako samostatné serverové tasky, z nichž některé mohou běžet současně i ve více instancích. Výsadní postavení mezi nimi má služba naming service, u níž se ostatní služby registrují. Každý task, který je v systému HelenOS spuštěn, má implicitně otevřené spojení na naming service a s její pomocí se může v případě potřeby připojit k libovolné službě, která je u naming service registrována. Jednotlivé tasky spolu mohou komunikovat prostřednictvím tzv. asynchronního frameworku, který tvoří přívětivější nadstavbu nad nízkoúrovňovou meziprocesovou komunikací implementovanou v jádře. Jak asynchronní framework funguje a jak se používá, je popsáno v tutoriálu [helenos_ipc] na stránkách projektu. Pro účely této práce snad postačí jen krátký úryvek z tohoto materiálu týkající se terminologie meziprocesové komunikace v operačním systému HelenOS (volně přeloženo):

„Terminologie použitá pro popis meziprocesové komunikace v operačním systému HelenOS je založená na přirozené abstrakci telefonního hovoru mezi člověkem na jedné straně a záznamníkem na druhé straně spojení. Přítomnost záznamníku určuje asynchronní povahu komunikace – hovor není okamžitě vyřízen, ale nejprve musí být ze záznamníku vyzvednut druhou stranou.“

Použijeme-li tuto terminologii, chce-li task komunikovat, musí mít telefon na protistranu.

> POZNÁMKA
>
> Většina kódu v uživatelském prostoru je platformně nezávislá.

3.2 Původní rozhraní pro ovladače zařízení

Tato část textu popisuje rozhraní pro ovladače zařízení, které existovalo v systému HelenOS před vznikem této práce.

Ovladače v uživatelském prostoru byly samostatné tasky – jeden pro každou instanci ovládaného zařízení. Stávající rozhraní pro ovladače zařízení umožňovalo těmto ovladačům z uživatelského prostoru:

- prostřednictvím IPC přijímat od jádra notifikace o přerušeních vyvolaných ovládaným zařízením,
- asociovat s přerušením akci, která se má provést v jádře bezprostředně po přijetí přerušení,
- přistupovat k paměťově mapovaným registrům zařízení,
- přistupovat k registrům zařízení v I/O adresovém prostoru,
- řídit preempci,
- zaregistrovat sebe a ovládaná zařízení u device mapperu[1].

Pro přístup k některým z těchto funkcí musel task vlastnit odpovídající oprávnění.

Všechny v této kapitole uvedené funkce frameworku byly nadále zachovány, jen některé z nich byly v souvislosti s touto diplomovou prací mírně rozšířeny.

3.2.1 Přístup k registrům zařízení

Původní rozhraní pro práci s registry zařízení mělo dvě samostatné části – jednu pro práci s registry v odděleném I/O adresovém prostoru a druhou pro práci s paměťově mapovanými registry.

Možnost přistupovat k registrům v odděleném I/O adresovém prostoru měl každý task, který obdržel oprávnění CAP_IOSPACE_MANAGER. Před samotným přístupem k registrům zařízení si task musel potřebný rozsah adres v I/O prostoru zarezervovat voláním funkce `iospace_enable`, jíž předal počáteční adresu a délku rozsahu v bytech. Pokud rezervace proběhla v pořádku, mohl task číst a modifikovat obsah registrů zařízení pomocí funkcí `inx` a `outx`, kde `x` udává velikost registru v bytech.

Obdobně probíhala práce s paměťově mapovanými registry. K paměťově mapovaným registrům směl přistupovat task, který vlastnil oprávnění CAP_MEM_MANAGER. Task s tímto oprávněním si mohl voláním funkce `physmem_map` namapovat zadaný rozsah fyzických adres do svého virtuálního adresového prostoru, a pokud namapování proběhlo úspěšně, mohl pak k datům na těchto adresách přistupovat obdobným způsobem jako k libovolným jiným datům ve své virtuální paměti.

U samotného procesu namapování zadaného rozsahu fyzických adres se ještě na chvilku zastavíme. V systému HelenOS existoval mechanismus pro ochranu fyzické paměti – tzv. fyzické oblasti. *Fyzická oblast (physical memory area – parea)* specifikovala interval fyzických adres a jeho atributy. V jádře systému HelenOS se nacházel seznam všech fyzických oblastí, jejichž namapování do virtuálního adresového prostoru bylo povoleno. V okamžiku, kdy task požádal o namapování rozsahu fyzických adres, byla v jádře v obsluze příslušného systémového volání vyhledána odpovídající fyzická oblast a byly zkontrolovány její atributy. Pokud nebyla fyzická oblast nalezena, nebo atributy nebyly v pořádku vzhledem k vyžádanému typu mapování, byla žádost o namapování daného úseku fyzické paměti zamítnuta. Tímto bylo zabráněno svévolnému namapování libovolného úseku fyzické paměti taskem s potřebným oprávněním.

[1] Device mapper je služba, která zprostředkovává komunikaci mezi klientskými aplikacemi a ovladači zaregistrovaných zařízení. Více viz 3.2.3.

Ze způsobu, jakým jádro ošetřovalo namapování rozsahu fyzických adres do virtuálního adresového prostoru tasku, vyplývá, že jádro muselo předem znát všechny intervaly fyzických adres, kde se nacházely registry zařízení ovládaných z uživatelského prostoru. Toto způsobovalo nežádoucí závislost ovladačů v uživatelském prostoru na jádře. Každý ovladač v uživatelském prostoru, který potřeboval přistupovat k paměťově mapovaným registrům, k sobě musel v jádře mít minimalistický ovladač, který ještě před startem ovladače v uživatelském prostoru musel vytvořit odpovídající fyzickou oblast a zaregistrovat ji v jádře jako vhodnou k namapování do virtuálního adresového prostoru tasku. Jelikož tento přístup významně omezoval možnosti autokonfigurace systému z uživatelského prostoru, která je hlavní náplní této práce, autorka tohoto textu autory původního rozhraní pro ovladače na tento problém upozornila a autoři původního rozhraní problém odstranili.

Odstranění tohoto problému se dosáhlo s pomocí tzv. zón. *Zóna* je datová struktura, která v jádře systému HelenOS reprezentuje souvislý úsek fyzické paměti. Zóny se v systému HelenOS používají pro popis nainstalované fyzické paměti a paměti firmware.

Algoritmus rozhodující o povolení či zamítnutí namapování rozsahu fyzických adres do virtuálního adresového prostoru tasku se změnil tak, že kontrolu fyzických oblastí nyní provádí pouze pro intervaly fyzických adres, které se překrývají s některou ze zón, které popisují nainstalovanou fyzickou paměť. Pokud je zadaný interval fyzických adres mimo všechny zóny, případně je-li součástí zóny popisující paměť firmware, je žádost o namapování automaticky schválena a vyřízena.

Kromě změny algoritmu pro rozhodování o namapování intervalu fyzických adres došlo k vnějšímu sjednocení API pro přístup k paměťově mapovaným registrům a k registrům v odděleném I/O adresovém prostoru.

3.2.2 Zpracování přerušení

Task, který vlastní oprávnění CAP_IRQ_REG, se může zaregistrovat jako příjemce zpráv o příchozích přerušeních. Registrace se provádí voláním funkce `ipc_register_irq`, jíž se předá číslo přerušení, jednoznačný identifikátor zařízení, číslo IPC metody použité při notifikaci a tzv. *pseudokód*. Pseudokód specifikuje akce, které se mají provést v jádře bezprostředně po příchodu přerušení. Pseudokód je implementován jako pole struktur, které reprezentují speciální příkazy a specifikují argumenty těchto příkazů. Příkazy pseudokódu jsou po příchodu přerušení interpretovány handlerem přerušení v jádře a umožňují například čtení či zápis obsahu registru zařízení, přijetí či odmítnutí přerušení (je potřeba kvůli sdílení přerušení více zařízeními), bitové operace s argumenty pseudokódu a podmíněné provedení několika následujících příkazů.

Hlavním důvodem pro zavedení pseudokódu byla existence zařízení, která signalizují přerušení úrovní (level triggered interrupts – viz kapitola 2). Tato zařízení signalizují přerušení tak dlouho, dokud není odpovídajícím způsobem obslouženo (např. přečtením dat z registru zařízení a podobně). Pokud by nebylo přerušení od takovéhoto zařízení obslouženo v handleru přerušení v jádře, došlo by k němu vzápětí znovu – ještě před tím, než by bylo naplánováno odpovídající vlákno ovladače zařízení, které by mohlo přerušení obsloužit z uživatelského prostoru – a výsledkem by bylo zacyklení.

Kromě registrace čísla přerušení a pseudokódu si musí task, který chce zpracovávat přerušení, zaregistrovat funkci, která bude zpětně volána v rámci přijetí IPC notifikace

o přerušení. Registrace této funkce se provádí voláním funkce `async_set_interrupt_received`.

Zjednodušený scénář zpracování přerušení je tedy následující: Po příchodu přerušení je v jádře zavolán interrupt handler, který vyhledá v tabulce zaregistrovaných přerušení obslužný pseudokód a ten interpretuje. Je-li přerušení sdíleno, je obslužných pseudokódů registrováno hned několik současně. V takovém případě jsou pseudokódy zaregistrované pro dané přerušení postupně interpretovány jeden po druhém, dokud některý z nich přerušení nepřijme. Následně je do uživatelského prostoru zaslána notifikace o přerušení tomu tasku, jehož pseudokód po vyhodnocení dané přerušení přijal. Součástí IPC notifikace je výstup zpracování odpovídajícího pseudokódu (např. hodnoty registrů zařízení, které v jádře interpretovaný pseudokód přečetl a podobně).

Aby mohl task úspěšně přijímat notifikace o přerušení, je potřeba provést ještě jeden krok – povolit dané přerušení. Původní rozhraní pro ovladače zařízení neumožňovalo ovladačům povolit konkrétní přerušení z uživatelského prostoru. Potřebná přerušení se proto povolovala provizorně v jádře v průběhu inicializace systému a to ještě dříve, než byly v uživatelském prostoru spuštěny ovladače zařízení, které tato přerušení používaly. Bylo tedy nutné znát předem čísla všech přerušení, která bude třeba v době běhu systému povolit. To značně omezovalo možnosti ovladačů v uživatelském prostoru, např. nebylo možné z uživatelského prostoru provést automatickou detekci a konfiguraci zařízení, jelikož v takovém případě není předem známo, která zařízení budou detektována a která přerušení budou používat.

Za zmínku stojí ještě jeden problém, který souvisí s přístupem k paměťově mapovaným registrům z pseudokódu. Ovladač zařízení si tyto registry mapuje do svého virtuálního adresového prostoru a pomocí virtuálních adres z tohoto prostoru k nim pak také přistupuje. Oproti tomu pseudokód je interpretován v jádře, kde tyto virtuální adresy nemusí být platné, tedy je nelze jednoduše z pseudokódu použít. Proto jsou v pseudokódu používány pro registry tohoto typu virtuální adresy jádra. Jak tyto adresy ale ovladač získá? Ovladače využívající původní driver framework toto řešily tak, že každý z nich k sobě v jádře měl minimalistický ovladač. Tento ovladač v jádře ještě před startem ovladače v uživatelském prostoru požádal jádro o namapování příslušného rozsahu fyzických adres, který odpovídal registrům ovládaného zařízení, do virtuálního adresového prostoru jádra a výslednou virtuální adresu zveřejnil prostřednictvím rozhraní `sysinfo`. Ovladač v uživatelském prostoru si pak tuto adresu skrze rozhraní `sysinfo` přečetl a použil ji v pseudokódu. Toto opět vytvářelo nežádoucí závislost ovladače v uživatelském prostoru na jádře. Jako řešení bylo navrženo rozšíření rozhraní jádra o nové systémové volání, jehož prostřednictvím by si jádro do svého virtuálního adresového prostoru namapovalo zadanou fyzickou adresu a virtuální adresu pro dané mapování by pak vrátilo volajícímu tasku. Tyto virtuální adresy by se pak mohly použít z pseudokódu, neboť by jejich platnost v jádře byla zaručena.

Toto řešení zatím nebylo implementováno a z časových důvodů a také z důvodu neaktuálnosti (zatím to žádný ovladač používající nové části driver frameworku nepotřebuje) nebude implementováno ani v rámci této práce a je ponecháno jako možnost budoucího rozšíření.

3.2.3 Device mapper a devfs

Ovladače v uživatelském prostoru nabízejí své služby klientským aplikacím prostřednictvím tzv. device mapperu. *Device mapper* je v systému HelenOS služba implementovaná jako serverový task v uživatelském prostoru. Hlavním účelem této služby je zprostředkovávat komunikaci mezi aplikacemi, které potřebují používat fyzická zařízení, a ovladači těchto zařízení v uživatelském prostoru.

Každý ovladač v uživatelském prostoru se po startu zaregistruje u device mapperu. Ve chvíli, kdy ovladač dokončí inicializaci ovládaných zařízení, může si u device mapperu zaregistrovat i tato zařízení. Zaregistrováním ovladač zařízení zveřejní, takže od té chvíle je viditelné pro klientské aplikace. Chce-li aplikace komunikovat se zařízením, připojí se pomocí IPC k device mapperu, vyhledá požadované zařízení v seznamu registrovaných zařízení a požádá device mapper o připojení k zařízení. Device mapper následně tuto žádost o připojení přepošle ovladači, který si zařízení zaregistroval. Od této chvíle klientská aplikace komunikuje s ovladačem zařízení a jeho prostřednictvím k zařízení přistupuje.

Zařízení jsou u device mapperu registrována pod symbolickými jmény a v rámci registrace jsou jim přidělovány jednoznačné identifikátory, pomocí nichž se dá k zařízením připojit. Součástí symbolického jména, pod kterým je zařízení registrováno, je název tzv. jmenného prostoru (namespace), jehož součástí se má zařízení stát. Vnoření jmenných prostorů není zatím device mapperem podporováno, takže jmenné prostory a jména samotných zařízení registrovaných u device mapperu tvoří dvouúrovňovou hierarchii.

Kromě standardního API device mapperu je tato dvouúrovňová hierarchie přístupná také prostřednictvím devfs (device file system). Jmenné prostory registrované u device mapperu odpovídají adresářům v podstromě složky `/dev` a zařízení jsou reprezentována jako soubory v těchto adresářích. Dovoluje-li to rozhraní zařízení implementované příslušným ovladačem, lze k zařízení přistupovat – číst z něj a zapisovat do něj data – prostřednictvím API pro práci se soubory.

Kapitola 4

Obecné rozhraní pro ovladače zařízení

V této kapitole si popíšeme typické funkce rozhraní pro ovladače zařízení v moderním operačním systému. Popis těchto funkcí a jejich rozdělení do vrstev nám umožní vytvořit si lepší představu, jaké by měly být kladeny požadavky na návrh rozhraní v operačním systému HelenOS.

Typické rozhraní pro ovladače zařízení (driver framework) má za úkol:

umožnit platformní neutralitu ovladačů
: Na nejnižší úrovni rozhraní pro ovladače zařízení odstiňuje ovladače zařízení od nízkoúrovňových detailů přístupu k hardware, zejména od těch závislých na dané platformně. Rozhraní pro ovladače zařízení může ovladačům nabízet platformně neutrální API pro přístup k registrům zařízení, sjednocovat způsob práce s přerušením, nabízet funkce pro převody dat s odlišnou endianitou a podobně.

definovat vnější rozhraní ovladačů zařízení
: Součástí vnějšího rozhraní ovladače zařízení je část umožňující komunikaci s ovladačem jako takovým a část pro komunikaci s ovládaným zařízením.

  Rozhraní ovladače jako takového může sloužit např. pro zjišťování a nastavování vlastností ovladače nebo jeho pomocí může framework říci ovladači, které zařízení má ovládat.

  Vnější rozhraní ovladače pro komunikaci se zařízením umožňuje pracovat s daným zařízením bez znalosti specifických vlastností daného hardware – jak toto rozhraní vypadá, je do značné míry nezávislé na konkrétním modelu zařízení a je určeno především účelem daného zařízení. Jiné rozhraní poskytuje ovladač k zařízení pro uživatelský vstup (např. myš, klávesnice) a jiné k zařízení pro persistentní uložení dat (např. pevný disk).

  V případě, že ovládané zařízení je sběrnice, může ovladač mít ještě další rozhraní, a to rozhraní pro jednotný přístup k zařízením připojeným k dané sběrnici.

  Podobu všech těchto rozhraní ovladače definuje driver framework.

vytvářet abstrakci zařízení
: Nad vnějším rozhraním ovladače vytváří rozhraní pro ovladače zařízení abstrakci ovládaného zařízení, s kterou pracují ostatní části operačního systému a aplikace. Tato abstrakce někdy tvoří pouze tenkou obálku nad vnějším rozhraním ovladače v podobě API pro přístup k jeho službám, jindy je rozdíl mezi vnějším rozhraním ovladače a API pro přístup k jím ovládanému zařízení znatelnější.

Např. se znakovými zařízeními v systému unixového typu lze pracovat stejným způsobem jako se soubory, úkolem odpovídající části rozhraní pro ovladače zařízení je tedy vytvářet iluzi práce se soubory nad rozhraním ovladače znakového zařízení.

Rozhraní pro ovladače zařízení dále může:

provádět přiřazení ovladače k zařízení
Rozhraní pro ovladače zařízení určuje, zda a jakým způsobem je zařízení přiřazen odpovídající ovladač. Z hlediska driver frameworku je nejjednodušší nechat ovladač, aby své zařízení nalezl sám. V případě, že toto není možné nebo žádoucí, musí rozhraní pro ovladače zařízení definovat způsob, jak najít pro zařízení vhodný ovladač a jak mu toto zařízení předat.

řídit životní cyklus ovladačů
V nejjednodušším případě jsou ovladače všech zařízení, která se mohou v systému objevit, přítomny po celou dobu běhu operačního systému. Jsou-li např. v daném operačním systému ovladače implementované v jádře, stávají se jeho trvalou součástí, v případě ovladačů, které jsou implementovány jako samostatné programy určené pro běh v uživatelském prostoru, jsou ovladače spuštěny ihned po startu systému a nadále trvale běží.

Alternativou je zavádět ovladače až v době, kdy jsou potřeba – např. ve chvíli, kdy je v systému nalezeno zařízení, které daný ovladač má ovládat, případně lze ovladače zavádět na žádost uživatele. Současné nejběžněji používané operační systémy toto umožňují a ovladače v nich jsou implementovány jako dynamicky zaváděné moduly jádra.

přidělovat prostředky zařízením
Některá zařízení podporují softwarovou konfiguraci adres, na kterých se budou nacházet jejich registry (ať už paměťově mapované nebo v samostatném I/O adresovém prostoru), a přerušení, která budou tato zařízení používat. Takto konfigurovatelná zařízení přítomná v systému je třeba nastavit tak, aby nedocházelo ke konfliktům v adresách a číslech přerušení jim přidělených. Při startu počítače toto nastavení většinou dostačujícím způsobem provede firmware, v některých případech ale může operační systém chtít toto nastavení změnit a tyto systémové prostředky přerozdělit – např. proto, že je dokáže rozdělit efektivněji nebo proto, že bylo za běhu připojeno nové hotplug zařízení a při využití rozdělení prostředků, které provedl firmware, na něj nezbývá dostatečný rozsah adres požadovaných vlastností. V takovém případě má přerozdělení prostředků na starost odpovídající součást driver frameworku.

kontrolovat přístup k zařízením
Definuje-li systém určitá bezpečnostní pravidla ve vztahu k zařízením, může být úkolem driver frameworku (alternativně konkrétních ovladačů) dodržování těchto pravidel vynucovat – tj. řídit přístup ostatních částí systému nebo aplikací k zařízením.

specifikovat rozhraní pro komunikaci mezi ovladači
V některých případech potřebují jednotlivé ovladače zařízení navzájem komunikovat. Typická je například komunikace mezi ovladačem koncového zařízení

a ovladačem sběrnice, na níž je zařízení připojeno. Driver framework může definovat podobu této komunikace, stanovit pro ni určitá pravidla a poskytovat ovladačům podpůrný mechanismus tuto komunikaci usnadňující.

umožňovat instalaci a konfiguraci ovladačů
Nejsou-li ovladače zařízení napevno zabudovanou součástí operačního systému, je třeba definovat způsob, jakým lze do systému nové ovladače instalovat a případně provádět jejich konfiguraci. V některých operačních systémech stačí zkopírovat do správného adresáře binární obraz ovladače společně s textovým konfiguračním souborem (např. v operačním systému Solaris). V jiných je třeba vytvořit instalační skript nebo program (např. soubor s příponou `inf` v operačním systému Windows) a konfigurace ovladačů se může provádět jinak než pomocí textového konfiguračního souboru (v operačním systému Windows se konfigurovatelné parametry ovladačů umisťují do obecné hierarchické databáze pro ukládání systémové konfigurace – tzv. Windows Registry).

podporovat specifické technologie
Moderní hardware mívá zabudovánu podporu pro určité technologie – např. PnP, hotplug, power management. Aby mohly být možnosti tohoto hardware plně využity, je potřeba, aby tyto technologie byly podporovány ze strany frameworku pro ovladače zařízení.

Samotný ovladač zařízení můžeme rozdělit na dvě pomyslné vrstvy:

- logickou, která implementuje vnější rozhraní ovladače, a
- fyzickou, která se stará o komunikaci s hardware.

Některé frameworky pro ovladače zařízení umožňují tyto dvě vrstvy skutečně oddělit – např. do dvou samostatných ovladačů (Windows port a miniport driver).

Kapitola 5

Cíle návrhu

V předchozích kapitolách jsme uvedli typické funkce obecného rozhraní pro ovladače zařízení a popsali jsme, které části rozhraní pro ovladače zařízení byly v systému HelenOS implementovány již před vznikem této práce. V této kapitole se na základě těchto znalostí pokusíme určit hlavní cíle dalšího návrhu rozhraní pro ovladače zařízení v operačním systému HelenOS tak, abychom vyhověli specifickým potřebám tohoto systému.

5.1 Funkce frameworku ve vztahu k systému HelenOS

Nejprve se vrátíme k jednotlivým bodům popisujícím typické funkce driver frameworku z předchozí kapitoly a rozebereme je z hlediska aktuálních potřeb operačního systému HelenOS:

platformní neutralita ovladačů
: Původní rozhraní pro ovladače zařízení v HelenOS ve značné míře umožňovalo psát ovladače zařízení platformně neutrálním způsobem – ovladače zařízení byly samostatné serverové tasky v uživatelském prostoru a pro přístup k registrům zařízení a pro práci s přerušeními používaly jednotné na platformě nezávislé API. Tato část návrhu je tedy již v zásadě vyřešena, zbývá jen odstranit některé nedostatky stávajícího rozhraní pro práci s registry zařízení a s přerušením – stávající rozhraní neřešilo endianitu registrů zařízení, neumožňovalo povolení přerušení z uživatelského prostoru atd. (viz 3.2).

vnější rozhraní ovladačů a abstrakce zařízení
: Vnější rozhraní ovladačů, které využívaly původní driver framework, bylo definováno protokolem, který tyto ovladače implementovaly nad IPC. Aplikace, která chtěla přistupovat k zařízení ovladače, musela tento protokol znát. Pokud více ovladačů implementovalo tentýž protokol – např. proto, že jimi ovládaná zařízení byla stejného typu z hlediska funkce a způsobu použití – musel každý z těchto ovladačů mít vlastní implementaci tohoto IPC protokolu, tj. každý ovladač musel umět rozpoznat jednotlivé IPC zprávy, musel umět správně interpretovat jejich argumenty a musel vědět, kdy má provést příjem, zaslání nebo sdílení většího množství dat. To vedlo k podstatnému množství stále se opakujícího kódu a k horší přehlednosti zdrojového kódu ovladačů. V neposlední řadě byla důsledkem tohoto přístupu i větší pracnost vývoje nového ovladače – měl-li nový ovladač být svým vnějším rozhraním kompatibilní s některým z již existujících

protokolů, musel vývojář daný protokol napřed detailně nastudovat, a jelikož takový protokol obvykle nebyl příliš dokumentován, musel jej vývojář nastudovat ze zdrojových kódů jiných ovladačů, což bylo nejen časově náročné, ale také při tom hrozila dezinterpretace některých částí protokolu a narušení požadované kompatibility. Po nastudování protokolu musel vývojář zpracování IPC ve svém ovladači implementovat místo toho, aby se mohl soustředit výhradně na specifické vlastnosti konkrétního modelu hardware.

Jedním z cílů návrhu tedy je zjednodušit vývoj ovladačů s jednotným vnějším rozhraním – a to konkrétně omezit duplicitní zpracování komunikačního protokolu a omezit riziko jeho nesprávné interpretace. Dalším cílem je v zájmu jednodušší spolupráce mezi ovladači a ostatními částmi systému zvýšit důraz na jednotné vnější rozhraní ovladačů – toho by mělo být dosaženo nikoliv definováním kompletního vnějšího rozhraní ovladačů jednou pro vždy, ale spíše vytvořením mechanismu, který bude umožňovat toto rozhraní definovat a v budoucnu rozšiřovat, jak bude postupně do operačního systému HelenOS přidávána podpora pro nové typy zařízení.

rozhraní mezi ovladači zařízení

Společně s nárůstem počtu podporovaných zařízení bude v systému HelenOS vzrůstat potřeba vzájemné komunikace mezi ovladači. Stejně, jako je výhodné mít do určité míry jednotné rozhraní pro komunikaci klientských aplikací s ovladači, tak je výhodné mít jednotné rozhraní pro komunikaci mezi ovladači navzájem. Jelikož operační systém HelenOS se bude v budoucnu nadále prudce vyvíjet, nastává obdobný problém jako u sjednocování vnějšího rozhraní ovladačů pro komunikaci s aplikacemi – spíše než definitivně specifikovat rozhraní pro vzájemnou komunikaci ovladačů bude lepší poskytnout vývojářům operačního systému HelenOS obecný mechanismus, jak toto rozhraní definovat a v budoucnu podle potřeby rozšiřovat. Pokud to bude možné, bude tento mechanismus v zájmu jednoduchosti stejný jak pro definování rozhraní pro komunikaci s aplikacemi, tak pro definování rozhraní pro komunikaci s ostatními ovladači.

podpora specifických technologií

- PnP – stávající framework pro ovladače zařízení v HelenOS neobsahoval jednotné rozhraní ani jinou podporu pro automatickou detekci a konfiguraci zařízení. Aby bylo možné jednoduše vytvářet ovladače zařízení, která vyžadují softwarovou detekci a konfiguraci, je třeba tuto část frameworku navrhnout a implementovat. Jelikož automatickou detekci a konfiguraci podporuje většina moderních sběrnic, je toto omezení stávajícího frameworku značně limitující, neboť významně omezuje množinu zařízení, která lze z operačního systému HelenOS zpřístupnit. Podpora automatické detekce je tedy jedním z cílů návrhu a má mezi ostatními cíli vysokou prioritu. Vysokou prioritu má zejména podpora boot-time plug and play, jelikož právě tu budou potřebovat zařízení, jejichž ovladače je v plánu v nejbližší době přidat, ale návrh by měl být přinejmenším snadno rozšiřitelný o hotplug funkcionalitu. Automatická konfigurace prostředků – adres a čísel přerušení – plug and play zařízení zatím prioritou není, neboť pro boot-time plug and play bude prozatím postačovat konfigurace převzatá od firmware.
- Power management – podpora správy napájení je značně pokročilá vlastnost rozhraní pro ovladače zařízení a není nezbytně nutná pro chod operač-

ního systému. V této fázi vývoje systému HelenOS nemá smysl o takto pokročilých vlastnostech uvažovat, vzhledem k velmi omezené množině podporovaných zařízení by případný návrh správy napájení nešlo v dohledné době plně realizovat a prakticky ověřit.

řízení životního cyklu ovladače, přiřazení ovladače k zařízení
Původní rozhraní pro ovladače zařízení v HelenOS tyto problémy neřešilo – po startu systému byly v uživatelském prostoru spuštěny všechny ovladače, které byly na dané platformně k dispozici, a tyto ovladače si musely sami v rámci své inicializace zjistit, zda jsou přítomna zařízení, která mají ovládat. V některých případech k tomu používaly informace o ovládaných zařízeních předané z jádra prostřednictvím rozhraní `sysinfo`, tyto informace pro ně často připravil minimalistický ovladač téhož zařízení implementovaný v jádře – tentýž, který jim případně předem povolil přerušení (viz 3.2).

V případě, že tento přístup – tj. spuštění všech ovladačů po startu systému a nalezení ovládaných zařízení jejich svépomocí – nebude dostačující, případně optimální vzhledem k dosažení ostatních cílů, bude třeba navrhnout jiné řešení. Vzhledem k tomu, že jedním z cílů návrhu je podpora automatické detekce zařízení, při níž nelze předem určit, která zařízení budou v systému nalezena a jaké ovladače budou tedy potřeba, bylo by výhodné navrhnout mechanismus, který umožní až po nalezení zařízení vyhledat vhodný ovladač, spustit jej a zařízení mu předat. Postup, kdy jsou při startu automaticky spuštěny všechny dostupné ovladače, je v porovnání s tímto řešením neefektivní.

instalace a konfigurace ovladačů
Jak poroste počet zařízení podporovaných operačním systémem HelenOS, bude stále potřebnější definovat jednotný způsob instalace a konfigurace ovladačů zařízení.

5.2 Shrnutí

Z předchozího rozboru vyplynulo, že cíle návrhu budou následující:

- odstranění drobných nedostatků stávajícího rozhraní ve vztahu k práci s registry zařízení a s přerušením
  - přidání podpory pro povolení přerušení z uživatelského prostoru
  - zlepšení možností práce s paměťově mapovanými registry z pseudokódu
  - doplnění funkcí pro práci s daty s různou endianitou
- vytvoření obecného mechanismu pro definování vnějšího rozhraní ovladačů
- podpora automatické detekce zařízení po startu systému (s možností rozšíření na hotplug)
- řízení životního cyklu ovladače a vzájemného přiřazení ovladače a zařízení
- definování způsobu instalace a konfigurace ovladačů

Návrh by měl mít také určité obecné vlastnosti vzhledem k povaze operačního systému HelenOS – tento operační systém se stále ještě intenzivně mění a je vyvíjen nekomerčně relativně malou skupinou vývojářů. Návrh by tedy měl být jednoduchý a snadno rozšiřitelný.

Kapitola 6

Analýza

V této kapitole se vrátíme k některým problémům řešeným driver frameworky (viz 4), popíšeme si známé přístupy k jejich řešení a zhodnotíme jejich výhody a nevýhody – obecně i z pohledu operačního systému HelenOS.

6.1 Instalace a konfigurace ovladače

Ovladač se obvykle skládá ze dvou částí – binárního spustitelného kódu a konfiguračních dat. Binární kód ovladače se často vyskytuje ve formě zásuvného modulu jádra (u ovladačů nacházejících se v jádře monolitických operačních systémů), případně programu či knihovny v uživatelském prostoru. Konfigurační data mohou mít podobu textového souboru (Solaris), ale i binárních dat ve speciálním formátu (ovladače v systému Windows jsou konfigurovatelné prostřednictvím záznamů ve Windows Registry). Některé vlastnosti ovladače mohou být zakomponované do téhož souboru, který obsahuje jeho binární kód (v Linuxu jsou informace jako verze jádra, pro kterou byl ovladač přeložen, licence a další parametry ovladače součástí zaveditelného modulu jádra s binárním kódem ovladače).

Způsob instalace ovladače do značné míry závisí na způsobu jeho konfigurace. Obvyklou součástí instalace ovladače je zkopírování jeho binárního kódu do některého ze standardních adresářů používaných v daném operačním systému pro ukládání ovladačů, případně obecně modulů jádra. Dalším krokem je vytvoření konfigurace ovladače.

Nejjednodušší způsob je použit v operačním systému Solaris, kde se konfigurace ovladače nachází v textovém souboru (název tohoto souboru se skládá z názvu ovladače a přípony `.conf`). Instalaci ovladače v tomto operačním systému lze provést nakopírováním binárního a konfiguračního souboru ovladače do jednoho ze standardních adresářů určených pro instalaci rozšiřujících modulů jádra – např. adresář `/usr/kernel/drv` je určen pro 32-bitové ovladače nezávislé na platformně.

Instalace ovladače v systémech, kde má konfigurace ovladačů binární formu, je obvykle již o něco složitější a vyžaduje použití speciálních instalačních nástrojů.

V rámci instalace ovladače v operačním systému Windows jsou vytvořeny záznamy ve Windows Registry, ve kterých je uložena konfigurace ovladače. Instalace ovladače se obvykle provádí jedním ze dvou způsobů – pomocí instalačního souboru s příponou `INF` nebo prostřednictvím instalačního programu. Soubor s příponou `INF` je textový soubor, který popisuje akce, které se mají provést při instalaci ovladače, včetně kopírování souborů a vytváření nových záznamů ve Windows Registry. Kromě toho tento

soubor specifikuje identifikátory modelů zařízení, které umí ovladač ovládat.

V Linuxu lze za běhu instalovat i odebírat moduly jádra (tedy i ovladače zařízení) pomocí utility modprobe spouštěné z příkazové řádky. Parametry příkazu modprobe specifikují název modulu jádra (moduly se vyhledávají ve standardním adresáři), akci prováděnou s modulem (přidání nebo odebrání modulu) a dodatečné parametry modulu, které ovlivňují jeho vlastnosti a chování.

Pro potřeby operačního systému HelenOS bude zatím stačit podobný způsob konfigurace a instalace, jako je v operačním systému Solaris – umístění binárního spustitelného souboru ovladače a konfiguračního souboru ovladače do adresáře vyhrazeného pro ovladače zařízení. Tento způsob je nejjednodušší, nevyžaduje definici žádných speciálních konfiguračních a instalačních mechanismů ani implementaci příslušných nástrojů, a zároveň dostačujícím způsobem plní svůj účel.

6.2 Životní cyklus ovladače a přiřazení k zařízení

Jsou známy dva základní přístupy k řízení životního cyklu ovladače a k jeho přiřazování k ovládaným zařízením:

driver-centric

Iniciátorem přiřazení ovladače k zařízení je ovladač. Ovladač je v první řadě načten (spuštěn) a poté se sám začne aktivně shánět po zařízeních, která by mohl ovládat. Načtení ovladače iniciuje operační systém (např. na základě konfigurace, která specifikuje ovladače, které se mají zavést při startu) nebo ručně uživatel.

Tento přístup je vhodný spíše pro ovládání hardware, který nepodporuje automatickou detekci, a často se používal ve starších operačních systémech, které vznikly v době, kdy technologie PnP ještě nebyla rozšířená.

Příkladem použití tohoto přístupu jsou starší operační systémy Windows NT (3.1 - 4.0). Ovladače, které se mají při startu zavést jsou v těchto operačních systémech specifikovány ve Windows Registry a jsou rozděleny do několika kategorií, které určují, v jaké fázi startu systému je daný ovladač zaveden. Po zavedení je provedena inicializace ovladače a v rámci ní ovladač vyhledá zařízení, která bude ovládat – možností je několik (viz [win2k]):

- Ovladač má seznam hardwarových prostředků (čísel přerušení, adres registrů a podobně) všech zařízení, na která by se mohla v systému vyskytnout a pro která by zároveň byl ovladač vhodný. Ovladač v rámci své inicializace provede device probing pro zařízení ze seznamu a poté systému oznámí přítomnost nalezených zařízení.
- Seznam zařízení a jejich hardwarových prostředků specifikuje instalační program ovladače.
- Device probing (případně automatickou detekci pro PnP zařízení) provede operační systém při startu ještě před zavedením ovladače a informace o nalezených zařízeních a jimi používaných prostředcích zapíše do Windows Registry, odkud si je při své inicializaci přečte ovladač.

Jak je zřejmé z poslední zmíněné možnosti, lze v omezené míře v driver-centric systémech podporovat boot-time PnP. Pro podporu hotplug tento přístup ale již nestačí, což je jeho výraznou nevýhodou.

Další nevýhodou tohoto přístupu je, že jsou zaváděny (spouštěny) i ty ovladače, které nejsou potřeba, protože se pro ně nakonec nenajde vhodné zařízení. Výhodou tohoto přístupu je v některých případech jednodušší podpora ze strany driver frameworku a dobré přizpůsobení potřebám starších zařízení, která nepodporují automatickou detekci.

device-centric

Přiřazení ovladače k zařízení je iniciováno nalezením zařízení. Ve chvíli, kdy je zařízení nalezeno, vyhledá se pro něj vhodný ovladač, je-li to potřeba, je ovladač zaveden, a následně je zařízení ovladači předáno.

Tento přístup je vhodný zejména pro zařízení, která podporují automatickou detekci (boot-time PnP i hotplug), a vyskytuje se v novějších operačních systémech.

Příkladem device-centric přístupu je WDM (Windows Driver Model – viz [win2k]), který se používá pro psaní ovladačů v operačním systému Windows od verze 2000. Ovladače zařízení následující tento model v rámci své inicializace nespecifikují ovládaná zařízení, ale pouze zaregistrují u systému funkce ovladače. Jednou z těchto funkcí je funkce `AddDevice`, která slouží pro předání zařízení ovladači. Ve chvíli, kdy je nalezeno zařízení, je pro něj vybrán vhodný ovladač. Pokud ovladač nebyl dosud zaveden, je načten a zinicializován. Následně je zavolána funkce `AddDevice` daného ovladače a zařízení je ovladači jejím prostřednictvím předáno.

Výhodou device-centric přístupu je větší pružnost, zjednodušení ovladačů některých zařízení (ovladače nemusejí samy provádět detekci PnP zařízení) a dobrá podpora PnP včetně hotplug.

Původní rozhraní pro ovladače zařízení v operačním systému HelenOS volilo driver-centric přístup. S požadavkem na podporu boot-time PnP s možnou budoucí rozšiřitelností o podporu hotplug je potřeba tento přístup změnit a přejít na device-centric model.

Při použití device-centric přístupu vyvstává další problém, který je třeba řešit – rozhraní pro ovladače zařízení musí definovat způsob, jakým se pro nalezené zařízení najde nejvhodnější ovladač.

Často se tento problém řeší tak, že se definuje jednotný způsob, jímž lze specifikovat, jaké modely zařízení daný ovladač podporuje. Obvykle lze určit nejen přesný model zařízení, ale i specifikovat celou rodinu navzájem kompatibilních zařízení. Jsou-li pro dané zařízení dostupné dva vhodné ovladače – jeden učený pro konkrétní model zařízení a jeden určený pro obecnější rodinu zařízení – přednost je dána ovladači konkrétního modelu zařízení před obecnějším ovladačem. Příkladem takového řešení jsou WDM ovladače ve Windows. Ve Windows lze v instalačním souboru WDM ovladače (soubor s příponou `INF`) specifikovat jedno tzv. hardware ID a libovolné množství tzv. compatible IDs. *Hardware ID* je textový řetězec identifikující přesný model zařízení, *compatible ID* identifikuje obecnější model zařízení, se kterým je ovladač kompatibilní. Výhodou tohoto řešení je pružnost – není-li k dispozici ovladač pro konkrétní model zařízení, použije se ovladač obecný, ale jsou-li k dispozici oba, použije se ten, který lépe zná specifické vlastnosti daného zařízení. Další výhodou tohoto řešení je jednoduchost

– ze strany ovladačů sběrnic i koncových zařízení vyžaduje jen minimální podporu, tj. ovladač koncového zařízení musí pouze specifikovat modely zařízení, která umí ovládat, a ovladač sběrnice musí specifikovat modely zařízení, která jsou na jeho sběrnici připojená.

O něco obecnější variantu tohoto řešení představuje číselné ohodnocení vhodnosti ovladače pro dané zařízení – ovladač specifikuje několik modelů zařízení, která umí ovládat, a ke každému z nich přiřadí číselné ohodnocení, které udává vhodnost ovladače pro daný typ zařízení. Zařízení je předáno ovladači, který přiřadil jeho modelu nejvyšší ohodnocení. Podobné řešení používá driver framework I/O Kit v operačním systému Mac OS X – proces přiřazení ovladače k zařízení se v tomto frameworku skládá ze tří fází, přičemž v druhé z nich jsou ze seznamu vybrány ovladače, které ve svém konfiguračním souboru (ve formátu XML) přiřadily odpovídajícímu modelu zařízení nejvyšší číselné ohodnocení (více viz [iokit]).

Alternativně může být rozhodnutí, který ovladač je vhodný pro dané zařízení, závislé na typu sběrnice, na které se zařízení nachází. Příkladem jsou ovladače v operačním systému Linux. Tyto ovladače specifikují seznam identifikátorů zařízení, která podporují (umějí je ovládat). Zároveň se tyto ovladače registrují u ovladače sběrnice, na které se může nacházet jejich zařízení, přičemž právě ovladač sběrnice jako jediný umí správně interpretovat identifikátory ovladačem podporovaných zařízení. Ve chvíli, kdy ovladač sběrnice nalezne nové zařízení na ovládané sběrnici, projde seznam registrovaných ovladačů a pro každý z nich na základě seznamu identifikátorů podporovaných zařízení rozhodne, zda je ovladač vhodný pro nalezené zařízení. Zařízení je předáno prvnímu vyhovujícímu ovladači. Výhodou tohoto řešení je možnost lepšího přizpůsobení se specifickým vlastnostem konkrétních sběrnic, nevýhodou větší pracnost implementace ovladače sběrnice – ovladač sběrnice má za úkol rozhodnout o vhodnosti registrovaného ovladače pro připojené zařízení.

6.3 Automatická detekce a strom zařízení

Základní scénář automatické detekce zařízení je v mnoha moderních operačních systémech přibližně stejný. Detekci zařízení provádí ovladač sběrnice, která podporuje technologii plug and play, a nález každého zařízení připojeného na ovládanou sběrnici hlásí frameworku pro ovladače zařízení. Pro každé nalezené zařízení je následně vyhledán vhodný ovladač (je-li takový ovladač v systému nainstalovaný) a zařízení je mu předáno. Pokud právě předané zařízení je další sběrnice (most na další sběrnici), postup se rekurzivně opakuje – ovladač provede detekci zařízení připojených na tuto sběrnici a přítomnost nalezených zařízení oznámí frameworku.

K předání zařízení ovladači obvykle slouží k tomuto účelu určený vstupní bod ovladače – funkce, kterou si za tímto účelem ovladač zaregistroval u driver frameworku. Např. v operačním systému Windows je to funkce `AddDevice` odkazovaná ze struktury reprezentující daný ovladač, v operačním systému Solaris je to funkce `attach` odkazovaná ze struktury `dev_ops`, která sdružuje funkce ovladače pro manipulaci s ovládanými zařízeními, a v operačním systému Linux je to funkce `probe` odkazovaná ze struktury ovladače. Obdobně je pro registraci zařízení nalezeného ovladačem sběrnice použitá příslušná funkce driver frameworku.

V monolitických operačních systémech podporujících automatickou detekci bývají zařízení obvykle reprezentována jednotným způsobem – v operačním systému Windows je zařízení reprezentováno strukturou `DEVICE_OBJECT`, v operačním systému

Linux strukturou `kobject` a v operačním systému Solaris strukturou `dev_info_t`. Ve všech těchto operačních systémech je součástí každého objektu zařízení ukazatel na objekt zařízení sběrnice, na kterou je jím reprezentované zařízení připojené. Ačkoliv objekty těchto zařízení náležejí různým ovladačům, nacházejí se všechny v témže adresovém prostoru – ve virtuálním adresovém prostoru jádra – a tvoří společně ucelenou stromovou strukturu odpovídající fyzickému zapojení reprezentovaných zařízení. Tato stromová struktura je postavena při startu systému během popsaného rekurzivního procesu detekce zařízení a jejich předávání ovladačům a nadále může být měněna, jsou-li zařízení přidávána a odebírána za běhu systému.

Existence této stromové struktury (stromu zařízení) je poměrně důležitá, neboť nad ní lze provádět řadu důležitých operací – např. odebírání zařízení za běhu nebo správa paměti vyžaduje znalost vzájemného fyzického zapojení zařízení. Při odebírání zařízení je třeba vědět, zda na něj nejsou připojena další zařízení a nemají být odebrána společně s ním. Akce související se správou napájení je třeba provádět ve správném pořadí ve stromě, např. odebírat napájení lze pouze směrem od listů ke kořeni, nikdy ne obráceně – nelze nejprve odebrat napájení sběrnici a teprve poté se snažit nastavit na ni připojená zařízení do klidového režimu, protože po odebrání napájení sběrnici nelze s připojenými zařízeními komunikovat.

Znalost vzájemného fyzického zapojení zařízení není důležitá jen pro akce, které je potřeba provádět se stromem zařízení jako celkem (přechod do úsporného režimu napájení) nebo s jeho částí (odebrání podstromu, který odpovídá odebíranému zařízení). Částečná znalost umístění zařízení ve stromě je obvykle důležitá i pro správnou funkci konkrétních ovladačů. Ovladač zařízení často potřebuje služby ovladače sběrnice, na kterou je jeho zařízení připojeno. Minimální informace, kterou ovladač zařízení tedy potřebuje, je znalost rodiče jím ovládaného zařízení ve stromě, tj. ovladač zařízení potřebuje mít odkaz na objekt zařízení sběrnice, na kterou je jím ovládané zařízení připojeno.

V případě ovladačů zařízení implementovaných jako samostatné procesy (tasky) v uživatelském prostoru je situace složitější – každý z ovladačů má svůj samostatný virtuální adresový prostor, v němž se nacházejí objekty reprezentující ovládaná zařízení. Propojení objektu zařízení s objektem rodičovského zařízení (sběrnice) je tedy třeba provést jiným způsobem než u ovladačů v jádře, stejně jako je potřeba jinak dosáhnout možnosti procházet celý strom zařízení. Dále je potřeba si uvědomit, že odlišné jsou i prostředky vzájemné komunikace ovladačů – zatímco u ovladačů sídlících v tomtéž virtuálním adresovém prostoru lze mezi ovladači komunikovat prostým voláním funkcí, v případě ovladačů v samostatných procesech je potřeba alespoň na nejnižší úrovni použít prostředků meziprocesové komunikace.

Základní možnosti reprezentace vzájemných vztahů zařízení z hlediska fyzického zapojení jsou dvě (a lze je částečně kombinovat):

centralizovaný strom zařízení

V systému existuje jedna centrální služba, která průběžně udržuje aktuální podobu celého stromu zařízení. Služba si pro každé zařízení pamatuje, který ovladač jej ovládá (pokud zařízení nějaký ovladač ovládá), případně uchovává další informace o vlastnostech zařízení. Tato služba umožňuje pohodlné procházení stromu zařízení a jejím prostřednictvím mohou navzájem komunikovat ovladače. Např. ovladač koncového zařízení se může pomocí této služby propojit s ovladačem nadřazené sběrnice – díky znalosti stromu zařízení a ovladačů přiřazených zařízením ve stromě může tato centrální služba zprostředkovat komu-

nikaci mezi ovladačem sběrnice a ovladačem na ni připojeného zařízení.

Aktuální podobu stromu zařízení tato služba udržuje ve spolupráci s ovladači sběrnic. Ovladače sběrnice zasílají této službě zprávy o přidání či odebrání zařízení na ovládané sběrnici a služba tyto změny aplikuje na svůj interní obraz stromu zařízení.

Součástí této služby může být implementace některých obvyklých operací prováděných se stromem zařízení jako celkem – např. správa napájení (podobné akce je ale pochopitelně třeba provádět ve spolupráci s ovladači konkrétních zařízení, úlohou této služby je spíše na základě znalosti stromu zařízení tyto akce inteligentně koordinovat).

Služba uchovávající strom zařízení musí být dostatečně spolehlivá, protože pokud by došlo k jejímu pádu, nemohly by spolu ovladače zařízení nadále komunikovat a nebylo by možné procházet strom zařízení a provádět nad ním potřebné operace.

distribuovaný strom zařízení

V systému neexistuje centrální služba, která by udržovala aktuální podobu stromu zařízení, místo toho podobu stromu zařízení udržují ovladače zařízení. Každý ovladač má o fyzickém zapojení zařízení pouze částečnou představu – pro každé ovládané zařízení zná jeho bezprostředního předka ve stromě (tj. zařízení sběrnice, na kterou je ovládané zařízení připojeno) a je-li ovládané zařízení samo sběrnice, zná i jeho bezprostřední potomky (zařízení připojená na ovládanou sběrnici). Jak předek zařízení, tak jeho potomci jsou pravděpodobně ovládáni jinými ovladači a ovladač zařízení s těmito ovladači musí být propojený. K propojení ovladače sběrnice s ovladačem připojeného zařízení musí dojít v rámci předání připojeného zařízení jeho ovladači a toto spojení je potřeba udržet po celou dobu, kdy je na sběrnici zařízení přítomno.

Absence centrální služby uchovávající celý strom zařízení je vykoupena větší složitostí ovladačů a zvýšenými nároky na jejich vzájemnou komunikaci – jelikož neexistuje centrální prvek, který by koordinoval operace prováděné nad stromem zařízení, musejí se ovladače při provádění těchto akcí na koordinaci vzájemně domlouvat. Např. zjistí-li za běhu ovladač sběrnice, že některé ze zařízení připojených na jím ovládanou sběrnici bylo neočekávaně odebráno, musí tuto skutečnost oznámit ovladači odebraného zařízení, a ten tuto skutečnost musí oznámit ovladačům všech zařízení, která byla na odebrané zařízení připojena atd. – zpráva o odebrání se musí rekurzivně rozšířit do celého podstromu, který ve stromě zařízení měl ve svém vrcholu odebrané zařízení.

V případě, že dojde k pádu některého z ovladačů sběrnic (zařízení reprezentovaná vnitřními uzly stromu zařízení), dojde k roztříštění stromu zařízení.

6.4 Abstrakce zařízení a rozhraní ovladače

Prostřednictvím vnějšího rozhraní ovladače lze komunikovat jak s ovladačem jako takovým (např. předávat mu zařízení), tak se zařízeními, která ovládá.

6.4.1 Vstupní body ovladače

Vnější rozhraní ovladačů, které se nacházejí v jádře, obvykle bývá tvořeno sadou standardizovaných funkcí, které ovladač implementuje a registruje u driver frameworku. Tyto standardizované funkce odpovídají jednotlivým operacím, které lze s ovladačem či s ovládanými zařízeními provádět, a registrací těchto funkcí ovladač tyto operace zpřístupní ostatním částem systému. Tyto standardizované funkce bývají označovány jako *vstupní body ovladače zařízení (device driver entry points)*.

Kromě vstupních bodů, které ovladač zpřístupní registrací u driver frameworku, obsahuje ovladač obvykle jeden předem známý vstupní bod, který je volán driver frameworkem za účelem inicializace ovladače. Ve funkci, která představuje tento vstupní bod, probíhá registrace dalších vstupních bodů ovladače u frameworku. Driver framework obvykle definuje mechanismus, pomocí nějž tuto funkci identifikuje, aniž by u něj byla ovladačem předem registrována – některé frameworky definují speciální název, který tato funkce musí mít (např. `DriverEntry` pro ovladače v jádře operačního systému Windows, `_init` pro ovladače v jádře operačního systému Solaris), jiné definují jiný způsob, jak tuto funkci označit (např. v operačním systému Linux je tato funkce v rámci ovladače implementovaného jako zaveditelný modul jádra označena voláním makra `module_init`, jemuž se předá název této funkce).

Jsou-li ovladače implementovány jako samostatné procesy (tasky) v uživatelském prostoru, musí být jejich rozhraní alespoň na nejnižší úrovni implementováno prostřednictvím meziprocesové komunikace, pomocí níž se zpřístupní rozhraní ovladače ostatním částem systému. Ostatní části systému zasílají ovladači prostřednictvím meziprocesové komunikace zprávy, které specifikují operace, které má ovladač provést.

Tyto zprávy může ovladač zpracovávat přímo sám, případně je za něj může částečně zpracovat framework – stejně jako vstupní body ovladače v jádře monolitického systému mají standardizovanou podobu, tak i protokol meziprocesové komunikace používaný pro zasílání žádostí o provedení operace ovladači musí být nějakým způsobem standardizován, a proto je možné, aby jej driver framework pro ovladač částečně předzpracoval. V případě, že meziprocesovou komunikaci za ovladač obstará driver framework, je možné pro ovladače zavést vstupní body podobné jako u ovladačů v jádře. Tyto vstupní body by se registrovaly u driver frameworku (např. z funkce `main` ovladače) a ten by je volal v rámci zpracování meziprocesové komunikace. Výhodou tohoto uspořádání je redukce kódu ovladače – vzhledem k tomu, že protokol meziprocesové komunikace je standardizovaný, vedlo by jeho zpracování v každém ovladači zvlášť k nárůstu zbytečného, neustále se opakujícího kódu.

6.4.2 Rozhraní pro přístup k zařízení

Část vstupních bodů ovladače slouží ke komunikaci se zařízením – prostřednictvím těchto vstupních bodů ovladač umožňuje ostatním částem systému přistupovat k zařízení a provádět s ním odpovídající operace. To, jaké operace prostřednictvím vstupních bodů ovladač pro dané zařízení implementuje, určuje, jakým způsobem mohou ostatní části systému se zařízením manipulovat – tyto operace z pohledu ostatních částí systému tvoří rozhraní daného zařízení. Na základě těchto rozhraní lze zařízení rozdělit do několika funkčních tříd – zařízení v rámci téže třídy jsou z hlediska způsobu manipulace a nabízených funkcí stejná; ačkoliv se vnitřním způsobem fungování mohou tato zařízení zásadně lišit, z pohledu těch částí systému, které k nim přistupují prostřednictvím ovladačů, se neliší.

Granularita funkčních tříd zařízení se v různých operačních systémech liší – v některých operačních systémech je těchto funkčních tříd jen několik málo, v jiných je jich relativně hodně. Tato granularita je výsledkem kompromisu mezi dvěma protichůdnými cíli – jednoduchostí a přesností.

Na jednu stranu je výhodné mít co nejmenší počet funkčních tříd zařízení, protože se zmenšuje počet a složitost různých subsystémů driver frameworku, které nad těmito třídami zařízení vytváří vyšší úrovně abstrakce, a tím se také sjednocuje API pro práci se zařízeními na těchto vyšších úrovních.

Na druhou stranu je žádoucí skrze ovladače a driver framework zpřístupnit co největší počet funkcí, které mohou v systému nainstalovaná zařízení nabídnout. Malým množstvím jednoduchých funkčních tříd nelze zpřístupnit všechny funkce a možné způsoby použití nepřeberného množství zařízení, která se vyrábějí. Zjednodušení a sjednocení rozhraní pro přístup k zařízení si vždy vybírá daň v podobě omezeného využití možností daného zařízení. Nejlepšího využití možností zařízení by se dosáhlo tak, že by se pro každý model zařízení definovalo rozhraní jemu přesně na míru.

Příkladem operačních systémů, ve kterých se nachází pouze malé množství funkčních tříd zařízení, jsou Linux a Solaris (a obecně řada dalších operačních systémů unixového typu). Tyto operační systémy si v zásadě vystačí s pouhými třemi třídami zařízení, mezi něž patří zařízení bloková, zařízení znaková a síťová rozhraní. Naopak v operační systému Windows je předdefinovaných tříd zařízení několik desítek a dá se předpokládat, že v budoucnu budou přibývat další. Kromě toho lze nové třídy dynamicky přidávat, což je užitečné zejména pro podporu některých nestandardních zařízení.

Rozhodnutí, jak rozdělit zařízení do funkčních tříd, patří při návrhu rozhraní pro ovladače zařízení k jednomu z nejtěžších a vyžaduje léta zkušeností, kterými autorka této práce nedisponuje. Kromě toho operační systém HelenOS se bude nadále intenzivně vyvíjet, takže není zatím ani žádoucí specifikovat definitivní seznam tříd zařízení a rozhraní pro přístup k zařízením. Z tohoto důvodu se návrh rozhraní pro ovladače zařízení v HelenOS zaměřil spíše na vytvoření obecného mechanismu, pomocí nějž je možné rozhraní pro přístup k zařízení definovat. Tento mechanismus bude popsán v následujících kapitolách.

Další zajímavý problém týkající se rozhraní pro přístup k zařízení je, s čím dané rozhraní asociovat. Ve většině operačních systémů je sada funkcí, které implementuje ovladač pro přístup k zařízení, asociována s ovladačem jako takovým, protože se předpokládá, že všechna zařízení, která ovladač ovládá, jsou stejného typu. V některých případech to ale nemusí platit, a pak je tento přístup poněkud nešikovný – v čem spočívá nevýhoda tohoto přístupu si vysvětlíme na příkladu operačního systému Windows.

Windows

V operačním systému Windows dostává ovladač povely, jaké operace má provést s ovládaným zařízením, prostřednictvím tzv. *IRPs (I/O requtest packets)*. IRP je univerzální datová struktura používaná pro popis typu a parametrů operace. IRP obvykle vytváří I/O Manager, což je součást jádra, která zprostředkovává komunikaci s ovladači ostatním částem systému – aplikacím v uživatelském prostoru, jiným ovladačům i některým dalším částem jádra. I/O Manager je také jediná část jádra, která může přímo volat funkce, které ovladač registroval u driver frameworku. IRP jsou vždy zasílány ovladači na konkrétní objekt zařízení, jiná možnost komunikace s ovladačem není – je-

li třeba nastavovat za běhu některé globální vlastnosti ovladače, které nejsou vázané na konkrétní ovládané zařízení, musí ovladač vytvořit v rámci své inicializace speciální objekt zařízení a na něj pak lze směrovat řídicí IRPs ovlivňující globální vlastnosti a chování ovladače.

Jelikož jakákoliv komunikace s ovladačem musí být směrována na objekt zařízení, které ovladač ovládá, musí i ovladač zařízení s ovladačem nadřazené sběrnice komunikovat tímto způsobem. Ve Windows proto ovladači sběrnice nenáleží pouze objekt zařízení reprezentující ovládanou sběrnici, ale kromě toho ovladač sběrnice vytváří objekt zařízení pro každé zařízení, které je na ovládané sběrnici připojené. Těmto objektům zařízení se říká *physical device objects (PDOs)* a zjednodušeně si lze představit, že reprezentují sloty na sběrnici, do nichž jsou připojena koncová zařízení. Ovladač sběrnice pro tyto PDOs implementuje obecné operace, které lze provádět se všemi zařízeními na daném typu sběrnice (např. každé zařízení na sběrnici PCI má konfigurační adresový prostor a ten lze s využitím ovladače sběrnice PCI číst). Ovladač koncového zařízení může při ovládání svého zařízení využívat služeb ovladače nadřazené sběrnice tak, že zašle IRP popisující požadovanou obecnou operaci na PDO, který odpovídá jemu ovládanému zařízení.

Zařízení je u svého ovladače (nikoliv u ovladače nadřazené sběrnice) reprezentováno objektem, kterému se říká *functional device object (FDO)*. Pro každé zařízení ve fyzické hierarchii, kterému byl přiřazen ovladač, tedy existují alespoň dva objekty zařízení – FDO náležející ovladači zařízení a PDO náležející ovladači nadřazené sběrnice.

Operace, kterou IRP reprezentuje, je v rámci IRP specifikována tzv. *hlavním a vedlejším kódem funkce* (*major function code, minor function code*). Hlavní funkční kód specifikuje základní typ operace (např. IRP_MJ_READ pro čtení a IRP_MJ_WRITE pro zápis), vedlejší funkční kód může typ operace dále upřesnit. IRPs jsou zpracovány speciálními funkcemi ovladače – tzv. *dispatch rutinami*. Každému hlavnímu funkčnímu kódu odpovídá jedna dispatch rutina, kterou ovladač určí v rámci své inicializace. Této dispatch rutině následně I/O Manager předává ke zpracování všechna IRP s odpovídajícím hlavním funkčním kódem.

Pokud ovladač ovládá zařízení několika různých typů – např. ovladač sběrnice může ovládat zařízení reprezentující sběrnici, dále zařízení na sběrnici připojená a speciální zařízení pro globální řízení a konfiguraci ovladače za běhu – pak musí v každé implementované dispatch rutině zkontrolovat typ zařízení[1] a teprve na jeho základě se rozhodnout, jakým způsobem danou operaci provede. To poněkud znepřehledňuje kód.

Alternativou k tomuto uspořádání je asociovat funkce implementující příslušné operace přímo s konkrétním objektem zařízení, což se více blíží objektovému modelu – objektu zařízení jsou přiřazena nejen data, ale i operace. Toto alternativní uspořádání nevyžaduje explicitní udržování typu zařízení ovladačem u objektu zařízení ani zkoumání typu zařízení v rámci implementace dané operace – funkce, která operaci implementuje, již automaticky předpokládá ten správný typ zařízení. Jediné, co je třeba, je při inicializaci asociovat s daným objektem zařízení tu správnou sadu funkcí.

[1] Pokud ovladač ovládá zařízení více typů, musí si při inicializaci každého objektu zařízení poznamenat, jaký typ zařízení daný objekt reprezentuje.

Kapitola 7

Návrh

V této kapitole popíšeme návrh rozhraní pro ovladače zařízení v operačním systému HelenOS. Vrátíme se k problémům nastíněným v předchozích kapitolách a pokusíme se při jejich řešení aplikovat některé postupy, jejichž výhody a nevýhody byly rozebrány v kapitole 6:

- strom zařízení bude centralizovaný
- ovladače se budou instalovat zkopírováním spustitelného binárního souboru a konfiguračních souborů do standardního adresáře
- konfigurace ovladače bude mít podobu jednoho nebo více textových souborů
- device-centric přístup k řízení životního cyklu ovladače a k jeho přiřazování k ovládaným zařízením
- přiřazování ovladačů k zařízením na základě identifikátorů modelů zařízení, optimální výběr ovladače z více vhodných variant
- obecný mechanismus pro definování standardních rozhraní pro přístup k zařízení
- asociace rozhraní pro přístup k zařízení s konkrétním zařízením, nikoliv s jeho ovladačem

7.1 Základní součásti

Při komunikaci se zařízením v uživatelském prostoru navzájem spolupracují součásti systému, které lze rozdělit do následujících kategorií:

- klientské aplikace – aplikace, které přistupují k zařízením prostřednictvím jejich ovladačů
- ovladače zařízení
- podpůrné části driver frameworku

Do poslední uvedené kategorie patří centrální správa stromu zařízení, služby pro zprostředkovávání komunikace mezi ovladači a klientskými aplikacemi, přiřazování ovladačů k zařízením a řízení životního cyklu ovladačů. Jelikož tyto funkce spolu úzce

souvisejí a jelikož jednotlivé součásti jsou samostatné tasky, které musejí navzájem komunikovat prostřednictvím meziprocesové komunikace, jejíž implementace je pracná, bylo v zájmu zjednodušení implementace rozhodnuto všechny tyto funkce implementovat v rámci jedné univerzální služby nazvané *správce zařízení*.

Návrh počítá s tím, že ovladače jsou samostatné serverové tasky. Původně byly uvažovány ještě další dvě alternativy, ale byly zavrženy.

První uvažovanou alternativou bylo mít všechny ovladače v rámci jednoho tasku. Výhodou takového uspořádání je, že ovladače pro vzájemnou komunikaci nepotřebují používat prostředků meziprocesové komunikace, čímž se omezí s IPC spojená režie. Nevýhodou by byla výrazně zhoršená stabilita ovladačů – chyba v jednom ovladači by způsobila pád všech ostatních ovladačů, což je v rozporu s filosofií multiserverových operačních systémů.

Druhou uvažovanou alternativou byla možnost prostřednictvím konfigurace rozdělovat jednotlivé ovladače do volitelného počtu tasků. Tato varianta by byla obecná a umožňovala by vyvažovat výhody obou předchozích uspořádání – stabilitu oddělených ovladačů a efektivitu komunikace ovladačů v rámci jednoho tasku. Tato varianta byla záhy vyhodnocena jako příliš implementačně náročná.

7.2 Správa ovladačů a zařízení

Centrální správu ovladačů a zařízení provádí správce zařízení, mezi jeho hlavní úkoly patří:

udržovat strom zařízení

Správce zařízení má za úkol udržovat reprezentaci aktuálního fyzického zapojení zařízení v interní stromové struktuře. Výhody existence centrálního stromu zařízení již byly popsány v kapitole 6.

Jelikož správce zařízení sám neumí zjišťovat přítomnost zařízení, musí při udržování aktuální podoby stromu zařízení úzce spolupracovat s ovladači – ovladače sběrnic prostřednictvím meziprocesové komunikace oznamují správci zařízení, jaká zařízení se nacházejí na ovládané sběrnici, a průběžně hlásí případné změny. Opačný přístup, kdy by se správce zařízení na seznam připojených zařízení aktivně dotazoval ovladačů sběrnic, je nevhodný pro budoucí rozšíření o podporu hotplug technologií (správce zařízení by se musel po celou dobu běhu systému pravidelně dotazovat ovladačů sběrnic na změny, systém by pak byl zahlcen velkým množstvím většinou zbytečné meziprocesové komunikace).

Podobu stromu zařízení mohou ovlivňovat i ovladače koncových zařízení – některé sběrnice nepodporují automatickou detekci zařízení, jejich ovladače tedy mohou pouze nahlásit, jaká zařízení se na sběrnici pravděpodobně nacházejí (např. na základě konfigurace), ale není zcela jisté, zda daná zařízení jsou doopravdy přítomna. O tom, zda je zařízení skutečně přítomno, v tomto případě rozhodne jeho ovladač – provede tzv. device probing (viz kapitola 2) – a výsledek oznámí správci zařízení. Device probing se provádí v rámci inicializace zařízení ovladačem, jenž byl zařízení přiřazen.

Správce zařízení ve stromě fyzického zapojení pro každé zařízení udržuje některé jeho vlastnosti. V rámci návrhu bylo uvažováno několik variant, zda a jaké vlastnosti se mají pro zařízení udržovat v centrálním stromě a jaké pouze u jeho ovladače. Mezi uvažovanými variantami zvítězil kompromis optimalizující využití

meziprocesové komunikace – část vlastností, která je společná pro všechna zařízení a která je vyžadována při provádění operací nad stromem zařízení, je uchována v centrálním stromě, a zbytek vlastností se nachází u ovladačů zařízení, kterých se lze na tyto vlastnosti zeptat. Takže např. jméno zařízení je uchováno v centrálním stromě, protože tuto vlastnost mají obecně všechna zařízení. Pokud by jméno zařízení ve stromě nebylo, musel by se na něj správce zařízení dotazovat ovladače, kdykoliv by prováděl prohledávání stromu zařízení, což by vedlo k nadměrné meziprocesové komunikaci. Naopak seznam přiřazených hardwarových prostředků (čísla přerušení a rozsahy adres) nemají všechna zařízení a proto je tato vlastnost uchována pouze u ovladače nadřazené sběrnice (právě tento ovladač umí tyto vlastnosti detekovat) a jemu lze na tyto vlastnosti poslat dotaz (to obvykle dělají ovladače připojených zařízení, ve chvíli, kdy tato zařízení inicializují). Pokud by tato vlastnost byla udržována v centrálním stromě, musely by ovladače sběrnic tyto informace zasílat správci zařízení pro všechna připojená zařízení a správce by je na žádost zasílal ovladačům připojených zařízení. To by vyžadovalo více meziprocesové komunikace než současné řešení, kdy je tato informace zasílána ovladači připojeného zařízení přímo a až ve chvíli, kdy si ji sám vyžádá.

řídit životní cyklus ovladačů a přiřazovat je k zařízením

Způsob přiřazování ovladačů k zařízením a řízení životního cyklu ovladačů správcem zařízení je přísně device-centric (viz oddíl 6.2). Proces přiřazení ovladače k zařízení iniciuje ovladač sběrnice tím, že správci zařízení oznámí přítomnost na ovládanou sběrnici připojeného zařízení. Součástí zprávy o nalezení zařízení je informace o modelu daného zařízení. Na základě této informace správce najde vhodný ovladač, je-li to třeba, spustí pro tento ovladač nový task a zařízení mu předá.

Při návrhu byly na mechanismus pro přiřazení vhodného ovladače k zařízení kladeny tyto požadavky:

- přiřazení na základě hardwarového modelu zařízení
- jednotný způsob specifikace hardwarového modelu (aby přiřazování mohl centrálně řídit správce zařízení a nemusely jej implementovat např. ovladače sběrnic)
- možnost specifikovat jak konkrétní model zařízení, tak obecnější rodinu navzájem kompatibilních zařízení
- výběr nejvhodnějšího ovladače z více přípustných variant (upřednostnění ovladače, který umí ovládat konkrétní model zařízení, před ovladačem, který umí ovládat obecnější rodinu navzájem kompatibilních zařízení)
- obecnost, pružnost, jednoduchost řešení

Na základě těchto požadavků byl navržen následující způsob přiřazování ovladače k zařízení:

- Ovladač sběrnice v rámci zprávy o nalezení nového zařízení zašle správci zařízení několik identifikátorů modelů zařízení, se kterými je nalezené zařízení kompatibilní.
- Identifikátory zařízení jsou textové řetězce (obecnost).

- Společně s identifikátorem zařízení je správci zasláno číselné ohodnocení specifičnosti modelu zařízení – čím vyšší hodnota, tím konkrétnější model zařízení.
- Ovladač zařízení specifikuje modely zařízení, které umí ovládat, ve svém konfiguračním souboru. (Není tedy třeba ovladač pouštět kvůli tomu, aby se zjistilo, zda umí ovládat daný typ zařízení. Toto řešení je v souladu s device-centric filosofií a umožňuje ovladač pustit až tehdy, když je nalezeno zařízení, které by měl ovládat, což je mnohem efektivnější než pouštět všechny dostupné ovladače předem při startu systému.)
- Modely zařízení jsou v konfiguračním souboru ovladače opět specifikovány textovými řetězci a jsou doplněny číselným ohodnocením, které určuje vhodnost ovladače pro ovládání daného zařízení (nižším ohodnocením lze naznačit, že ovladač je neúplný nebo umí zařízení ovládat jen částečně – např. jen pro čtení).
- Zařízení je přiřazen ovladač z množiny ovladačů, které dovedou ovládat kompatibilní model zařízení. Z této skupiny ovladačů je na základě ohodnocení vybrán nejvhodnější ovladač (jak přesně se vypočítá vhodnost ovladače z ohodnocení je popsáno v kapitole 8).

zprostředkovávat komunikaci s ovladači

Další z úkolů správce zařízení je zprostředkovávat komunikaci s ovladači zařízení jak klientským aplikacím, tak jiným ovladačům.

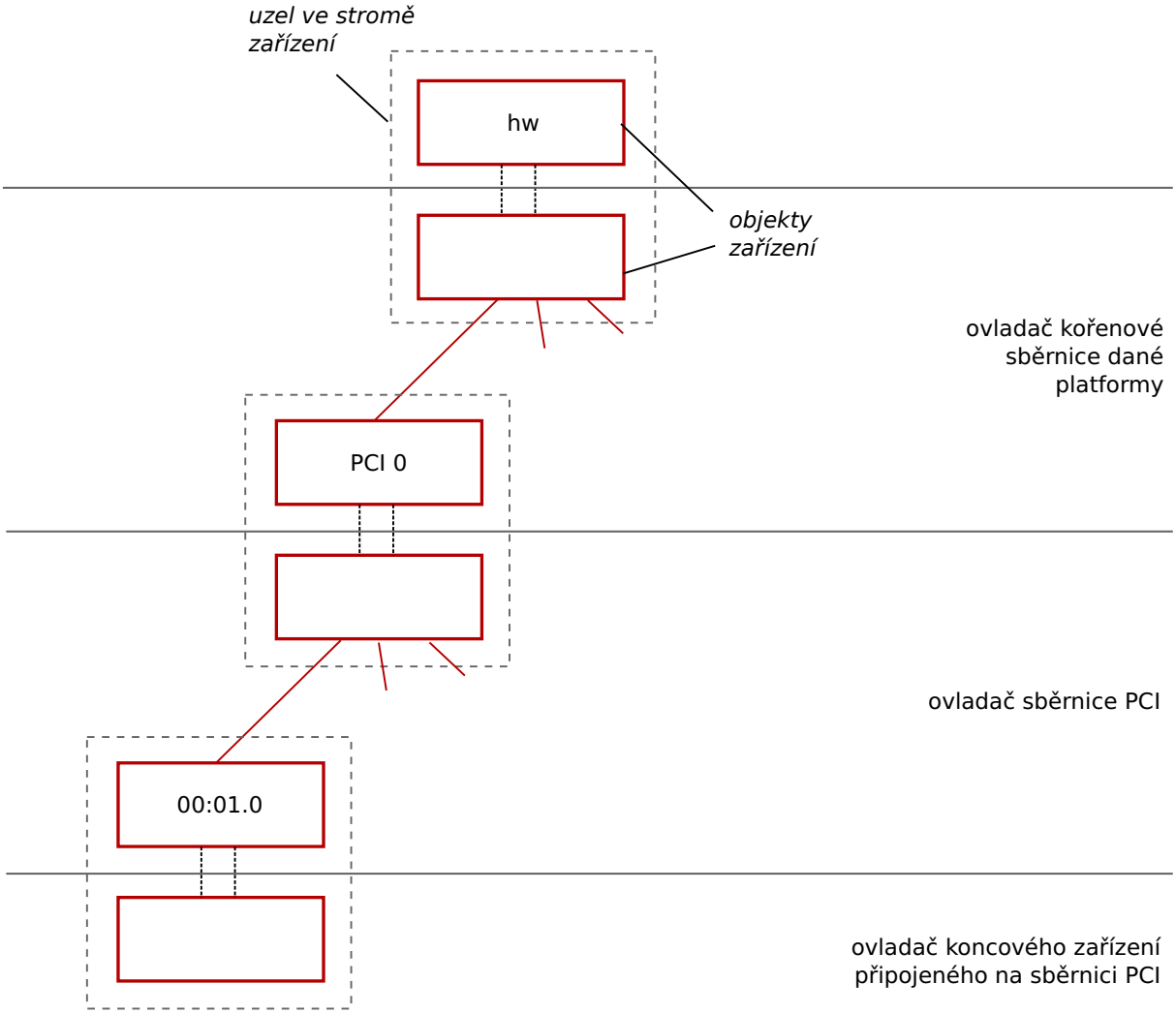
Navázání komunikace je vždy směrováno na konkrétní zařízení, ke kterému chce klient prostřednictvím ovladače přistupovat. Správce zařízení ví, které ovladače přiřadil zařízením ve stromě, a tak jim může žádost o navázání spojení od klienta přeposlat. Klientem může být jak běžná aplikace, tak jiný ovladač zařízení – např. ovladač koncového zařízení tímto způsobem komunikuje s ovladačem sběrnice, na kterou je jeho zařízení připojené. V tomto ohledu je návrh do značné míry inspirovaný WDM ovladači v operačním systému Windows (viz [win2k]).

Jakákoliv komunikace s WDM ovladačem zařízení probíhá na nejnižší úrovni zasíláním IRP, které reprezentují požadované operace, na objekty zařízení. Analogicky návrh rozhraní pro ovladače zařízení v operačním systému HelenOS počítá s tím, že klienti komunikují s ovladačem prostřednictvím zasílání IPC zpráv, které jsou směrovány na konkrétní objekt zařízení ovladače.

Ve Windows je tentýž způsob komunikace použit i při komunikaci mezi ovladačem zařízení s ovladačem sběrnice, na které je dané zařízení připojeno – ovladač sběrnice vytváří objekt zařízení pro každé zařízení připojené na ovládanou sběrnici, tomuto objektu zařízení se říká physical device object (PDO) a ovladač připojeného zařízení na něj posílá IRPs specifikující akce, které má ovladač sběrnice s daným zařízením provést. Samotný ovladač zařízení pro ovládané zařízení taktéž vytváří objekt zařízení – tzv. functional device object (FDO) – a na tento objekt zařízení jsou zasílány IRP žádosti specifikující akce, které má se zařízením provést jeho ovladač.

Podobný přístup byl použit v návrhu driver frameworku pro HelenOS. Stejně jako ve Windows i zde jsou dva objekty zařízení pro každé zařízení ve fyzické hierarchii. Jeden z těchto objektů náleží ovladači zařízení a druhý ovladači sběrnice, na niž je zařízení připojeno. Na první z těchto objektů směrují své žádosti

klientské aplikace. Tyto žádosti mají podobu IPC zpráv, které specifikují operace, které se mají provést s daným zařízením, a zpracování těchto žádostí provádí ovladač zařízení. Na druhý z objektů směruje IPC zprávy s žádostmi o provedení operací s daným zařízením ovladač tohoto zařízení a zpracování těchto žádostí je prováděno ovladačem sběrnice, na niž je zařízení připojeno. Výsledná hierarchie objektů zařízení vypadá podobně jako na obrázku 7.1. Obrázek znázorňuje hierarchii tří zařízení (dvě z nich jsou sběrnice). Každému zařízení odpovídají dva objekty zařízení, z nichž každý náleží jinému ovladači zařízení. Dvojici těchto objektů zařízení odpovídá jeden uzel ve stromě zařízení.



Obrázek 7.1: Objekty zařízení ve vztahu k hierarchii fyzického zapojení

7.3 Rozhraní ovladače pro přístup k zařízení

Návrh rozhraní pro přístup k zařízením si klade následující cíle (některé z nich na základě předchozího rozboru – viz 6.4):

- navrhnout obecný mechanismus pro definování standardních rozhraní pro přístup k zařízením

- asociovat funkce, kterými ovladač implementuje operace nad zařízením, přímo s konkrétním objektem zařízení, nikoliv s celým ovladačem (možnost ovládat více zařízení různého typu v jednom ovladači – obvykle v ovladači sběrnice)
- umožnit specifikovat funkční třídu zařízení a umožnit dynamické přidávání nových tříd pro zařízení nestandardního typu (klientské aplikace obvykle nezajímá, kde se zařízení nachází ve stromě fyzického zapojení, ale jakého funkčního typu zařízení je)

Jak již bylo uvedeno, komunikace se zařízením probíhá na nejnižší úrovni zasíláním IPC zpráv ovladači a tyto zprávy specifikují požadované operace a jejich parametry. Standardní rozhraní pro přístup k zařízení by tedy šlo definovat jako určitý protokol pro meziprocesovou komunikaci mezi klientem a ovladačem.

Každý ovladač, který by pro přístup k ovládanému zařízení poskytoval některé standardní rozhraní, by musel implementovat příslušný protokol a zpracování odpovídajících zpráv meziprocesové komunikace. Protože protokol meziprocesové komunikace pro standardní rozhraní se nemění, je zpracování příslušné meziprocesové komunikace stále stejné, a tak se nabízí možnost kód provádějící toto zpracování vyčlenit z ovladačů do samostatné knihovny.

K tomu je potřeba rozdělit kód, který implementuje danou operaci se zařízením, na část, která je společná všem ovladačům a týká se zpracování meziprocesové komunikace, a na část, která je specifická pro konkrétní ovladač a týká se komunikace se zařízením. Definujeme-li standardní rozhraní pro přístup k zařízení jako sadu operací, jež lze se zařízením provádět, pak můžeme toto standardní rozhraní rozdělit na obecnou část určenou pro zpracování meziprocesové komunikace a na část specifickou pro daný ovladač, která přímo implementuje odpovídající operace nad zařízením. První část rozhraní může být vyčleněna do samostatné knihovny, zatímco druhou část musí implementovat ovladač.

Řešení s předdefinovanými rozhraními, které mají obecnou část pro zpracování IPC vyčleněnou do knihovny a specifickou část implementovanou ovladači, bylo v novém driver frameworku pro HelenOS použito. Vybrané řešení umožňuje s konkrétním objektem zařízení asociovat sadu několika vybraných předdefinovaných rozhraní. Kromě použití předdefinovaných rozhraní je ovladači umožněno definovat a implementovat vlastní operace a asociovat je se zařízením, v tomto případě ale musí ovladač pro tyto operace odpovídající meziprocesovou komunikaci zpracovat sám. Předpokládá se, že ovladače tuto možnost využijí v případě, kdy budou ovládat nějaké nestandardní zařízení poskytující nestandardní operace a předdefinovaná standardní rozhraní jim nebudou stačit.

Popis konkrétních datových struktur a funkcí, jimiž je oddělení uvedených dvou částí rozhraní dosaženo, se nachází v implementační části této práce (viz 8).

Kromě rozhraní pro přístup k zařízení je také potřeba klientským aplikacím poskytnout informaci, do jaké funkční třídy dané zařízení patří. Ke zjištění této skutečnosti nestačí znát sadu standardních rozhraní asociovaných s daným zařízením – některá nestandardní zařízení nemusejí mít asociovaná žádná standardní rozhraní a jejich ovladače pro ně mohou definovat vlastní operace, tato nestandardní zařízení nemusejí mít z hlediska funkce nic společného a je třeba mezi nimi rozlišovat. Stejně tak některá zařízení, jejichž ovladače pro ně poskytují předdefinovaná rozhraní, mohou být určena k naprosto odlišnému účelu, ačkoliv jsou asociovaná se stejnou sadou předdefinovaných rozhraní (tato situace je obzvláště pravděpodobná, jsou-li asociovaná rozhraní dostatečně obecná).

Pro označení funkčního typu zařízení (zařazení zařízení do funkční třídy) je tedy třeba navrhnout jiný mechanismus. Nejjednodušší je přiřadit k zařízení jednoznačný identifikátor funkční třídy, do níž zařízení náleží.

Řešení, které bylo zvoleno, počítá s identifikátory v podobě textových řetězců, protože takové identifikátory tříd lze snadno za běhu přidávat a rozšiřovat tak množinu dostupných funkčních tříd. Kromě toho je možné zařízení přiřadit do více funkčních tříd najednou – hlavním účelem této volby je nahradit dědičnost tříd způsobem, který je implementačně jednoduchý. Některé třídy mohou být specifičtější a jiné obecnější. Tím, že ovladač zařadí zařízení nejen do specifičtější třídy, ale i do její obecnější nadtřídy, umožní k zařízení přístup i aplikacím, které chtějí pracovat se zařízeními z obecnější třídy.

Podobný způsob přiřazování do tříd používají WDM ovladače v operačním systému Windows, kde jsou tzv. *device interface classes*. Device interface class označuje třídu zařízení stejného funkčního typu, každá taková třída má globálně jednoznačný identifikátor (GUID). Zařízení je do dané třídy zařazeno voláním funkce `IoRegisterDeviceInterface`, jíž se předá GUID třídy a objekt zařízení. Dokumentace k této funkci uvádí následující příklad použití registrace zařízení jak u specifičtější, tak u obecnější třídy: „*For example, a fault-tolerant volume driver might register an instance of a fault-tolerant-volume interface and an instance of a volume interface for a particular volume.*“

Kapitola 8

Implementace

V této kapitole popíšeme implementaci prototypu frameworku pro ovladače zařízení v operačním systému HelenOS. Popsány budou pouze ty části, které byly v rámci této práce nově přidány. Části, které byly implementovány již v původním rozhraní pro ovladače zařízení (viz oddíl 3.2), byly zachovány a převzaty.

POZNÁMKA

Nebude-li řečeno jinak, bude v této kapitole termínem *ovladač*, případně *ovladač zařízení*, označován ovladač zařízení využívající rozhraní pro ovladače zařízení vytvořené v rámci této práce. Budou-li myšleny ovladače vytvořené čistě s pomocí původního driver frameworku, bude to explicitně uvedeno.

Prototypová implementace má následující součásti:

- Správce zařízení (Device manager) – serverový task, který provádí centrální správu zařízení a jejich ovladačů. Správce zařízení udržuje seznam dostupných ovladačů, aktuální podobu stromu zařízení a seznam zařízení podle funkčních tříd.
- Knihovna libdrv – tato knihovna je staticky linkována ke každému ovladači zařízení. Knihovna má za úkol sjednotit vnější rozhraní ovladačů jak při komunikaci s ostatními částmi systému, tak při komunikaci mezi sebou navzájem.
- Ovladače zařízení – prototypová implementace obsahuje několik ovladačů sběrnic a jeden ovladač koncového zařízení. Ovladače jsou implementovány jako samostatné serverové tasky. Každý ovladač běží v nejvýše jedné instanci, jedna běžící instance ovladače může ovládat několik zařízení téhož typu současně.

Správce zařízení a jednotlivé ovladače spolu navzájem komunikují prostřednictvím IPC. Komunikaci se správcem zařízení a klientskými aplikacemi za ovladače z větší míry obstarává knihovna libdrv, která implementuje příslušný komunikační protokol. Každý ovladač při své inicializaci u knihovny libdrv zaregistruje několik svých vstupních bodů, které později knihovna volá při zpracování IPC komunikace se správcem zařízení a s klientskými aplikacemi. To zajišťuje, že všechny ovladače s ostatními částmi systému komunikují stejným protokolem a tento protokol stejným způsobem

interpretují. Současně je tímto způsobem omezen výskyt neustále se opakujícího kódu a zdrojové kódy ovladačů jsou díky tomu kratší a přehlednější.

8.1 Správce zařízení

Zdrojové soubory správce zařízení se nacházejí v adresáři `uspace/srv/devman`. IPC zprávy používané pro komunikaci se správcem zařízení jsou definovány hlavičkovým souborem `uspace/lib/c/include/ipc/devman.h`.

Správce zařízení (Device manager) centralizuje informace o ovladačích a jimi ovládaných zařízeních. Mezi jeho funkce patří:

- správa ovladačů
- správa zařízení na základě fyzické hierarchie
- správa zařízení podle funkčních tříd
- přiřazování ovladačů k zařízením
- řízení životního cyklu ovladačů
- propojování klientských aplikací s ovladači

8.1.1 Správa ovladačů

Device manager si udržuje seznam všech v systému nainstalovaných ovladačů a jejich vlastností.

Tento seznam vytvoří po svém startu tak, že prohledá adresář, kde se nacházejí ovladače zařízení – v tuto chvíli je to adresář `/srv/drivers`, který se nachází na souborovém systému RAM disku používaném pro počáteční inicializaci systému – a přečte si potřebné informace z konfiguračních souborů ovladačů. Adresář `/srv/drivers` obsahuje jeden podadresář pro každý ovladač zařízení a každý takový podadresář je pojmenovaný stejně jako ovladač, kterému náleží. Adresář ovladače `A` (pojmenovaný `A`) obsahuje alespoň dva soubory – binární spustitelný soubor `A` a konfigurační soubor `A.ma`, který obsahuje seznam identifikátorů pro přiřazování ovladače k zařízení a číselné ohodnocení těchto identifikátorů. Na každém řádku tohoto souboru se nachází napřed číselné ohodnocení a za ním je alespoň jedním bílým znakem oddělený textový identifikátor modelu zařízení. Čím vhodnější je ovladač pro ovládání daného modelu zařízení, tím vyšší má ohodnocení identifikátor tohoto zařízení. Ve zdrojových souborech prototypové implementace se pro označení identifikátoru modelu zařízení používá termín *match ID* a pro jeho ohodnocení *match score*.

Seznam nainstalovaných ovladačů, který si Device manager uchovává po celou dobu svého běhu, obsahuje pro každý ovladač:

- jméno ovladače
- cestu k binárnímu spustitelnému souboru ovladače
- aktuální stav ovladače (např. informaci, zda již v systému běží instance daného ovladače)

- IPC spojení na běžící instanci ovladače (pokud nějaká v systému existuje)
- seznam identifikátorů pro přiřazení ovladače k zařízení (match ids) a jejich ohodnocení (match scores)
- seznam zařízení, která byla ovladači přiřazena

8.1.2 Strom zařízení

Správce zařízení si po celou dobu svého běhu udržuje aktuální podobu stromu zařízení. Tento strom zařízení reflektuje fyzické zapojení zařízení – rodičem zařízení ve stromě je sběrnice, na kterou je zařízení připojeno. Vnitřní uzly stromu reprezentují zařízení sběrnicového typu a listy reprezentují koncová zařízení. Kromě zařízení fyzicky přítomných v systému může strom obsahovat také virtuální zařízení.

Každé zařízení ve stromě má kromě seznamu synů a odkazu na rodičovské zařízení také jednoznačný číselný identifikátor (handle[1]), jméno, ovladač (pokud se ho povedlo zařízení přiřadit) a stav udávající míru použitelnosti. Možné jsou následující stavy – zařízení může být dosud neinicializované, připravené pro použití, porouchané nebo nepřítomné – poslední zmíněný stav se používá u zařízení připojených na sběrnicích, které nepodporují PnP. Ovladače těchto typů sběrnic hlásí na sběrnici přítomnost zařízení, která tam být mohou, ale nemusí, konečné slovo pak mají ovladače těchto zařízení, které poté, co jsou jim tato zařízení předána, ověří jejich skutečnou přítomnost (viz device probing v kapitole 2) a výsledek oznámí správci zařízení, který odpovídajícím způsobem upraví stav zařízení ve stromě.

Kromě virtuálního zařízení v kořeni stromu, které Device manager vytvoří při své inicializaci, jsou všechna zařízení do stromu přidávána na žádost ovladačů sběrnic, na kterých jsou tato zařízení připojena. Ve chvíli, kdy ovladač zařízení typu sběrnice najde na ovládané sběrnici připojené zařízení, zašle zprávu o nalezení synovského zařízení správci zařízení. Hlášení o nálezu synovského zařízení je doplněné identifikátorem rodičovského zařízení (tj. sběrnice, na které se zařízení nachází), jménem, které zařízení bylo na sběrnici přiřazeno, a seznamem dvojic: identifikátor modelu zařízení (match ID) – číselné ohodnocení (match score). Číselné ohodnocení udává, jak moc je specifikace modelu zařízení přesná – přesná identifikace konkrétního modelu zařízení (např. konkrétní model grafické karty od konkrétního výrobce) má vyšší ohodnocení než identifikace širší rodiny navzájem kompatibilních zařízení (např. grafická karta kompatibilní s VGA). Na základě znalosti identifikátoru (handle) rodičovského zařízení správce zařízení připojí synovské zařízení na správné místo ve stromě a zapamatuje si u něj informace předané ovladačem rodiče.

V současné implementaci prototypu mohou zařízení do stromu pouze přibývat, protože zatím nebyl vytvořen žádný ovladač sběrnice, která podporuje hotplug funkcionalitu, a tedy chyběla motivace pro přidání a prostor pro řádné otestování možnosti zařízení odebírat. Správce zařízení je nicméně navržen tak, aby tuto funkcionalitu bylo snadné v budoucnu přidat.

[1] Zařízení registrovaná u device mapperu mají také přiřazeny jednoznačné číselné identifikátory (handles), tyto identifikátory ale nemají nic společného s identifikátory ve stromě zařízení, které přiděluje správce zařízení, a v tomto textu se o nich nebudeme zmiňovat.

8.1.3 Přiřazení ovladače k zařízení a životní cyklus ovladače

Ve chvíli, kdy je zaregistrováno nové zařízení, se správce zařízení pokusí pro toto zařízení vyhledat nejlepší dostupný ovladač. Postup při hledání nejvhodnějšího ovladače je následující: Pro každý ovladač v seznamu nainstalovaných ovladačů se vypočítá ohodnocení přiřazení daného ovladače k zařízení. Zařízení je následně přiřazen ovladač s nejvyšším nenulovým ohodnocením. Vycházejí-li ohodnocení přiřazení všem ovladačům nulová, není zařízení přiřazen žádný ovladač.

Výpočet ohodnocení přiřazení ovladače k zařízení se skládá z následujících kroků:

- Nejprve je potřeba vyhledat shody v textových identifikátorech modelů zařízení v seznamech match IDs ovladače a zařízení. Shoda (match) znamená, že dané match ID se nachází jak v seznamu match IDs zařízení, tak v seznamu match IDs ovladače.
- Poté se vypočítá ohodnocení pro každou nalezenou shodu. Ohodnocení shody se vypočítá vynásobením match score, které je danému match ID přiřazeno v seznamu match IDs ovladače, s match score, které je témuž match ID přiřazeno v seznamu match IDs zařízení.
- Výsledné ohodnocení přiřazení ovladače k zařízení je maximum z ohodnocení nalezených shod, případně nula, pokud žádná shoda nebyla nalezena.

Poté, co správce zařízení přiřadí zařízení ovladač, pokusí se tuto skutečnost ovladači oznámit a zařízení mu předat. Zdali k předání dojde ihned po přiřazení, nebo je předání odloženo na později, závisí na aktuálním stavu ovladače. Ovladač se může nacházet ve třech různých stavech:

1. ovladač dosud nemá v systému běžící instanci,
2. ovladač byl již správcem zařízení spuštěn, ale dosud se u správce neohlásil,
3. ovladač již běží a jeho běžící instance se u správce zařízení ohlásila a správce na ni má otevřené spojení.

> POZNÁMKA
>
> Ovladač má v systému vždy nejvýše jednu běžící instanci, která ovládá všechna zařízení, kterým byl daný ovladač přiřazen. Ovladače napsané za pomoci původního rozhraní pro ovladače zařízení takto nefungovaly, každé zařízení bylo ovládáno samostatnou běžící instancí ovladače.

Nachází-li se ovladač bezprostředně po přiřazení k zařízení ve stavu 1, je správcem zařízení spuštěn (čímž přejde do stavu 2) a předání zařízení je odloženo na dobu, kdy se běžící instance ovladače správci ohlásí a ovladač tak přejde do stavu 3.

Nachází-li se ovladač ve stavu 2, nic se neděje, předání zařízení se odkládá na dobu, kdy přejde do stavu 3.

Nachází-li se ovladač ve stavu 3, zařízení je mu předáno.

Pokud byla předání některých zařízení odložena, dojde k nim při přechodu ovladače do stavu 3. Poté, co se ovladač připojí ke správci zařízení a oznámí mu, že je připraven zpracovávat příchozí žádosti, projde správce seznam zařízení, kterým byl ovladač přiřazen, a jedno po druhém je předá ovladači.

Předání zařízení probíhá tak, že správce zařízení naváže spojení s běžící instancí ovladače a zašle mu IPC zprávu DRIVER_ADD_DEVICE. Součástí této zprávy je handle a jméno zařízení (jméno je použito pro ladicí účely). Ovladač po přijetí zprávy zařízení převezme, a pokud to uzná za vhodné, zařízení zkontroluje a zinicializuje. Nakonec ovladač správci na zprávu odpoví – buď zařízení přijme a oznámí, že zařízení je připraveno k použití, nebo vrátí chybu. Návratem odpovídající chyby může ovladač správci oznámit např. i to, že zařízení není ve skutečnosti přítomno, případně, že přítomno je, ale nefunguje tak, jak by mělo.

Pokud ovladač potřebuje komunikovat s ovladačem sběrnice, na které je připojeno jím ovládané zařízení, zašle správci zařízení IPC zprávu DEVMAN_CONNECT_TO_PARENTS_DEVICE, jejíž součástí je handle ovládaného zařízení. Správce zařízení pak ovladač připojí k ovladači rodičovské sběrnice, jíž předá handle připojeného synovského zařízení. Poté, co takto ovladač naváže spojení s ovladačem rodičovského zařízení, může mu posílat zprávy specifikující akce s ovládaným zařízením (případně dotazy na vlastnosti zařízení), aniž by musel jako součást těchto zpráv uvádět handle zařízení – stačilo, že jej uvedl na začátku spojení. Ovladač zařízení tedy v podstatně není připojen na ovladač rodiče jako takový, ale je připojen na konkrétní zařízení – na to, které sám také ovládá.

8.1.4 Funkční třídy zařízení

Funkční třídy zařízení sdružují zařízení, jež plní stejnou funkci (mají stejný účel) a z pohledu klientských aplikací se používají stejným způsobem – jejich ovladače nabízejí stejné rozhraní, pomocí něhož lze k těmto zařízením přistupovat.

V driver frameworku jsou funkční třídy zařízení identifikovány textovými řetězci. Seznam tříd a zařízení, která do těchto tříd patří, udržuje správce zařízení. Jedno zařízení může být zařazeno do více tříd, např. lze simulovat jednoduchou dědičnost zařazením zařízení do dvou tříd, z nichž jedna je obecnější (zařízení pro ukládání dat po blocích) a druhá specifičtější (flash disk). Třídy je možné dynamicky přidávat.

Zařazení zařízení do funkční třídy iniciuje jeho ovladač tím, že zašle správci zařízení IPC zprávu DEVMAN_ADD_DEVICE_TO_CLASS doplněnou o handle zařízení a název třídy. Pokud správce zařízení dosud neměl v seznamu tříd třídu daného jména, je tato třída vytvořena a do seznamu přidána. Zařízení je do požadované třídy zařazeno a je mu přiděleno jednoznačné jméno v rámci této třídy.

8.1.5 Navázání spojení klientské aplikace s ovladačem

Chce-li aplikace přistupovat k danému zařízení, musí se připojit k jeho ovladači. Připojení k ovladači zprostředkuje správce zařízení poté, co mu aplikace zašle IPC zprávu DEVMAN_CONNECT_TO_DEVICE doplněnou číselným identifikátorem zařízení[2]. Správce zařízení vyhledá zařízení s daným identifikátorem, zjistí, který ovladač jej ovládá, a

[2]K zaslání této zprávy a k připojení k zařízení může klientská aplikace použít funkci `devmap_device_connect` (viz hlavičkový soubor `devman.h`).

přepošle mu žádost o spojení. Pokud nedojde k chybě, je spojení navázáno a aplikace může pomocí IPC zpráv instruovat ovladač, co má se zařízením provést. Identifikátor zařízení je součástí tohoto spojení, takže při zasílání IPC zpráv specifikujících akce, které se mají provést s daným zařízením, ovladači nemusí aplikace tento identifikátor znovu předávat.

8.2 Ovladače zařízení a knihovna libdrv

Ovladač zařízení je samostatný serverový task. V systému může být nejvýše jedna běžící instance daného ovladače a ta může ovládat několik zařízení současně. Ke každému ovladači je staticky linkována knihovna libdrv.

Tato knihovna utváří vnější rozhraní ovladačů – rozhraní pro komunikaci s ovladačem jako takovým i rozhraní pro přístup k ovládaným zařízením. Ostatním částem systému je toto vnější rozhraní ovladače přístupné prostřednictvím meziprocesové komunikace. Samotný ovladač se do značné míry o zpracování této meziprocesové komunikace starat nemusí, zpracování IPC za něj v mnoha případech provádí knihovna libdrv – místo toho, aby IPC zpracovával ovladač sám, registruje si u knihovny libdrv funkce, které jsou knihovnou v rámci zpracování příchozích zpráv zpětně volány – tyto funkce jsou jakousi obdobou vstupních bodů ovladačů v některých monolitických operačních systémech (např. v Linuxu). Dále knihovna libdrv definuje některé základní datové struktury používané ovladači a významným způsobem určuje strukturu ovladače.

8.2.1 Struktura ovladače zařízení

Strukturu ovladače si vysvětlíme na ukázce minimalistického ovladače – viz zdrojový kód `sample.c` (příklad 8.1) a konfigurační soubor `sample.ma` (příklad 8.2).

> POZNÁMKA
>
> Konfigurační soubor `sample.ma` a binární spustitelný soubor ukázkového ovladače `sample` by se po překladu a instalaci ovladače nacházely v adresáři `/srv/-drivers/sample/`.

Konfigurační soubor ukázkového ovladače obsahuje jediný identifikátor modelu zařízení (sample_match_id), pro jehož ovládání je ovladač určen. Identifikátoru předchází číselné ohodnocení vhodnosti ovladače pro ovládání daného zařízení – pokud se v systému nenajde ovladač s vyšším ohodnocení pro daný model zařízení, budou všechna zařízení tohoto typu nalezená v systému předána našemu ukázkovému ovladači.

Zařízení je v ovladači i v knihovně libdrv reprezentováno datovým typem device_t, což je struktura popisující společné vlastnosti všech zařízení – handle a jméno zařízení, operace asociované se zařízením a ukazatel na data, která si s daným zařízením asociával ovladač, volitelně IPC spojení na ovladač rodiče (je-li rodičovské zařízení ovládané jiným ovladačem), případně odkaz na strukturu rodičovského zařízení (v případě, že

je ovládáno tím samým ovladačem[3]).

```
#include <driver.h>

#define NAME "sample"

// Callback function for passing a new device to the sample driver
static int sample_add_device(device_t *dev) {
  // Initialize the device here...

  return EOK;
}

// The sample device driver's standard operations.
static driver_ops_t sample_ops = {
  .add_device = &sample_add_device
};

// The sample device driver structure.
static driver_t sample_driver = {
  .name = NAME,
  .driver_ops = &sample_ops
};

int main(int argc, char *argv[]) {
  // Initialize the driver here...

  return driver_main(&sample_driver);
}
```

Příklad 8.1: Zdrojový kód minimalistického ovladače – sample.c

```
10 sample_match_id
```

Příklad 8.2: Konfigurační soubor minimalistického ovladače – sample.ma

Ovladač je reprezentován strukturou typu driver_t, která kromě jména ovladače obsahuje odkaz na strukturu s operacemi ovladače (typu driver_ops_t). Jméno ovladače musí být shodné se jménem adresáře a binárního souboru ovladače (v případě našeho ukázkového ovladače je to tedy "sample").

Pro předání zařízení ukázkovému ovladači slouží funkce `sample_add_device` odkazovaná ze struktury operací ovladače sample_ops. Tato funkce bere jediný parametr – ukazatel na datovou strukturu reprezentující předávané zařízení. Funkce `sample_add_device` je volána knihovnou libdrv ve chvíli, kdy ovladači přijde od správce zařízení IPC zpráva DRIVER_ADD_DEVICE.

Klíčovou úlohu v propojení ovladače s ostatními částmi systému plní funkce `driver_main` (z knihovny libdrv). Poté, co ovladač úspěšně ukončí inicializaci vlastních

[3]To je možné v případě ovladačů sběrnic – ovladače sběrnic používají strukturu `device_t` jak k reprezentaci ovládané sběrnice, tak k reprezentaci zařízení, která jsou na ni připojena.

datových struktur, zavolá tuto funkci a předá jí odkaz na strukturu ovladače (`sample_driver`). Funkce `driver_main` použije jméno ovladače vyplněné ve struktuře pro registraci ovladače u správce zařízení – správce si na základě jména přiřadí běžící instanci ovladače ke správnému ovladači ve svém interním seznamu ovladačů nainstalovaných v systému. V rámci registrace ovladače u správce zařízení funkce `driver_main` také nastaví obslužnou funkci pro příjem a zpracování příchozích IPC zpráv (tato funkce je součástí libdrv). Dále tato funkce zaregistruje strukturu ovladače u libdrv a následně předá řízení asynchronnímu frameworku, který pak čeká na příchod IPC zpráv.

Kdykoliv přijde od správce zařízení IPC zpráva, asynchronní framework ji předá obslužné funkci z libdrv. Obslužná funkce zpracuje argumenty dané IPC zprávy, předá je odpovídající funkci ovladače a data navrácená touto funkcí zašle správci jako odpověď. Např. přijde-li zpráva DRIVER_ADD_DEVICE, obslužná funkce přijme informace o nově přidávaném zařízení, vytvoří a vyplní strukturu zařízení (typu device_t) a předá ji funkci `add_device` ze struktury operací ovladače (driver_ops_t). V případě našeho ukázkového ovladače je tedy volána funkce `sample_add_device`. Po návratu z funkce `add_device` zašle obslužná funkce z libdrv návratovou hodnotu funkce `add_device` správci zařízení jako odpověď na zprávu. Struktury všech zařízení, která ovladač úspěšně převzal (tj. funkce `add_device` pro ně vrátila návratovou hodnotu indikující úspěch), jsou zařazeny do seznamu ovládaných zařízení, který je spravován knihovnou libdrv.

Náš ukázkový ovladač v uvedené podobě tedy umí:

- zaregistrovat svou běžící instanci u správce zařízení a
- převzít zařízení od správce zařízení.

Co ukázkový ovladač zatím neumí, je provádět operace s ovládanými zařízeními na žádost klientských aplikací. K tomu, aby to dokázal, je třeba provést ještě jeden krok – asociovat nějaké operace s ovládaným zařízením. Asociovat operace se zařízením lze vyplněním struktury operací zařízení (typ device_ops_t) ukazateli na funkce, které budou implementovat požadované operace nad daným zařízením, a nastavením ukazatele `ops` ve struktuře device_t odpovídající danému zařízení na adresu této struktury operací.

Deklarace struktury operací je následující:

```
typedef struct device_ops {
  int (*open)(device_t *dev);
  void (*close)(device_t *dev);
  void *interfaces[DEV_IFACE_COUNT];
  remote_handler_t *default_handler;
} device_ops_t;
```

Položky struktury pojmenované `open` a `close` odkazují na implementaci operací, které se nad zařízením provedou při připojení klienta na zařízení (`open`) a při jeho pozdějším odpojení od zařízení (`close`). Tyto položky jsou stejně jako všechny ostatní položky struktury volitelné, a nejsou-li použité, musejí obsahovat nulový ukazatel.

Pole `interfaces` slouží pro definování implementace tzv. rozhraní zařízení. *Rozhraní zařízení (device interface)* je předdefinovaná sada standardních operací, které k sobě navzájem logicky náležejí a lze je provádět nad daným zařízením. Rozhraní zařízení jsou definována knihovnou libdrv, účelem rozhraní zařízení je redukovat množství

kódu ovladače pro zpracování meziprocesové komunikace a poskytnout jednotnou abstrakci pro práci se zařízeními téhož typu. Implementace rozhraní zařízení se skládá z několika částí:

identifikátoru rozhraní
Každému rozhraní je přiřazen jednoznačný číselný identifikátor, který slouží jako index do pole `interfaces` ve struktuře device_ops. Identifikace rozhraní zařízení je také třeba při komunikaci klientské aplikace, která chce se zařízením manipulovat, s ovladačem zařízení.

> POZNÁMKA
>
> Při meziprocesové komunikaci mezi klientskou aplikací a ovladačem se k číslu rozhraní přičítá konstanta IPC_FIRST_USER_METHOD. Důvodem je to, že identifikátor rozhraní je v rámci meziprocesové komunikace předáván jako první argument IPC zprávy – tj. jako tzv. IPC metoda – a IPC metody menší než IPC_FIRST_USER_METHOD jsou vyhrazeny pro použití systémem. Pokud by se tedy konstanta IPC_FIRST_USER_METHOD k číslu rozhraní nepřičetla, hrozil by konflikt s některou ze standardních IPC metod definovaných systémem.

identifikátorů jednotlivých operací
Každá operace, která je součástí rozhraní, má v rámci tohoto rozhraní jednoznačný číselný identifikátor. Tento identifikátor se používá při meziprocesové komunikaci.

Komunikuje-li tedy klientská aplikace s ovladačem zařízení za účelem manipulace se zařízením, předává ovladači v rámci IPC zpráv vždy číslo rozhraní jako první argument zprávy a identifikátor operace jako druhý argument. Následující argumenty určují parametry dané operace.

lokální implementace rozhraní
Lokální část rozhraní je definována strukturou, která odkazuje na funkce ovladače, které implementují jednotlivé operace rozhraní. Adresa této struktury musí být vyplněna v poli `interfaces` struktury device_ops na indexu shodném s identifikátorem rozhraní.

vzdálené implementace rozhraní
Pro každé předdefinované rozhraní je knihovnou libdrv implementována jeho tzv. vzdálená část (ve zdrojových kódech je označena jako remote interface). Vzdálená část rozhraní automatizuje zpracování meziprocesové komunikace mezi klientem, který přistupuje k zařízení prostřednictvím daného rozhraní, a ovladačem, který toto rozhraní implementuje (viz lokální implementace rozhraní). Díky existenci vzdálené části rozhraní se ovladač nemusí starat o zpracování meziprocesové komunikace a může se soustředit výhradně na odpovídající komunikaci se zařízením. Způsob, jakým funguje vzdálená část rozhraní, je podrobněji popsán v části věnované knihovně libdrv (viz oddíl 8.2.2).

Poslední položka struktury device_ops nazvaná `default_handler` definuje implicitní obsluhu meziprocesové komunikace mezi klientskou aplikací, která přistupuje k zařízení, a ovladačem, který ono zařízení ovládá. Tato implicitní obslužná funkce je použita v případě, že příchozí IPC zpráva není rezervována pro systém a zároveň není zpracována žádným z předdefinovaných rozhraní, která jsou asociována s cílovým zařízením.

POZNÁMKA

Použití implicitní obsluhy meziprocesové komunikace má pro ovladač oproti použití předdefinovaného rozhraní jednu nevýhodu – knihovna libdrv za ovladač nezpracuje argumenty dané IPC zprávy, pouze ovladači předá adresu struktury daného zařízení a zbytek si musí ovladač zpracovat sám. Důvodem je to, že knihovna libdrv zná pouze protokol meziprocesové komunikace pro předdefinovaná rozhraní, ale nemůže znát protokol, který si ovladač definuje sám (a pro který se z toho důvodu používá implicitní obsluha).

POZNÁMKA

Konstanta DEV_FIRST_CUSTOM_METHOD udává nejnižší číslo IPC metody, které může ovladač použít při implementaci vlastního protokolu pro komunikaci s ovládaným zařízením. Metody s čísly nižšími, než je DEV_FIRST_CUSTOM_METHOD, jsou buď rezervovány pro systém, nebo je používá některé z předdefinovaných rozhraní.

Ovladače sběrnic mohou asociovat operace jak se zařízením sběrnicového typu, které jim bylo správcem zařízení předáno, tak se zařízením, které je na danou sběrnici připojeno. Ovladač koncového zařízení díky tomu může komunikovat s ovladačem sběrnice, na kterou je jeho zařízení připojené, a zjišťovat od něj informace o ovládaném zařízení, případně provádět s ovládaným zařízením určité pro daný typ sběrnice specifické akce. Např. ovladač zařízení, které je připojené na sběrnici PCI, může prostřednictvím předdefinovaného rozhraní implementovaného ovladačem PCI sběrnice zjišťovat, jaké hardwarové prostředky (adresy registrů a čísla přerušení) jsou přiděleny jím ovládanému zařízení.

Chce-li ovladač zařízení komunikovat s ovladačem rodičovského zařízení, musí s ním nejprve navázat meziprocesovou komunikaci. Navázání komunikace zprostředkovává správce zařízení. Ovladač může správce zařízení o zprostředkování komunikace požádat voláním knihovní funkce `devman_parent_device_connect`, které předá handle ovládaného zařízení. Tato funkce zašle správci zařízení zprávu DEVMAN_CONNECT_TO_PARENTS_DEVICE doplněnou o handle ovládaného zařízení. Správce zařízení podle předaného handle vyhledá ve stromě zařízení, zjistí, které zařízení je jeho rodičem, a ovladači rodičovského zařízení žádost o navázání komunikace

přepošle.

Pro lepší názornost provedeme nakonec rozšíření ukázkového ovladače o předdefinované rozhraní zařízení – pro tento účel použijeme znakové rozhraní. Znakové rozhraní má přiřazen číselný identifikátor CHAR_DEV_IFACE a jeho lokální část, kterou musí implementovat ovladač zařízení, má-li zařízení znakové rozhraní podporovat, je definováno následující strukturou z knihovny libdrv:

```
typedef struct char_iface {
  int (*read)(device_t *dev, char *buf, size_t count);
  int (*write)(device_t *dev, char *buf, size_t count);
} char_iface_t;
```

Aby bylo možné komunikovat se zařízením ovládaným ukázkovým ovladačem prostřednictvím znakového rozhraní, musí ukázkový ovladač:

- implementovat funkce pro zápis a čtení ze zařízení vyhovující prototypům uvedeným v deklaraci typu char_iface_t
- vytvořit instanci struktury typu char_iface_t a adresy těchto funkcí do ní vyplnit
- vytvořit instanci struktury operací zařízení (typ device_ops_t) a do pole interfaces vyplnit adresu struktury znakového rozhraní (char_iface_t) na pozici danou indexem s hodnotou CHAR_DEV_IFACE
- v rámci inicializace zařízení (tj. při jeho předání ve funkci `sample_add_device`) nastavit adresu struktury operací do položky `ops` struktury daného zařízení (device_t)

Po aplikování těchto změn (viz příklad 8.3) je možno číst a zapisovat data na zařízení ovládaná ukázkovým ovladačem (čtená data budou obsahovat samé nuly, zapisovaná budou zahazována – podobně jako tomu je u zařízení /dev/null v systémech unixového typu).

```
#include <mem.h>
#include <driver.h>
#include <char.h>

#define NAME "sample"

// read zeros from the device
static int sample_read(device_t *dev, char *buf, size_t count) {
  memset(buf, 0, count);
  return count;
}

static int sample_write(device_t *dev, char *buf, size_t count) {
  return count;
}

// The character device interface implementation
static char_iface_t sample_char_interface = {
  .read = &sample_read,
  .write = &sample_write
};

// The operations associated with each device handled by the  ↩
   sample driver
static device_ops_t sample_device_ops = {
  .interfaces[CHAR_DEV_IFACE] = &sample_char_interface;
};

// Callback function for passing a new device to the sample driver
static int sample_add_device(device_t *dev) {
  // associate operations with the device
  dev->ops = &sample_device_ops;
  return EOK;
}

// The sample device driver's standard operations.
static driver_ops_t sample_ops = {
  .add_device = &sample_add_device
};

// The sample device driver structure.
static driver_t sample_driver = {
  .name = NAME,
  .driver_ops = &sample_ops
};

int main(int argc, char *argv[]) {
  return driver_main(&sample_driver);
}
```

Příklad 8.3: Zdrojový kód (sample.c) ukázkového ovladače po doplnění znakového rozhraní pro komunikaci s ovládanými zařízeními

8.2.2 Knihovna libdrv

Zdrojové soubory knihovny se nacházejí v adresáři `uspace/lib/drv`. Knihovna libdrv je staticky linkovaná ke každému ovladači. Knihovna libdrv:

- definuje základní datové struktury používané ovladači a pomocné funkce pro manipulaci s nimi
- zpracovává meziprocesovou komunikaci mezi ovladačem a správcem zařízení
  - po startu ovladače zaregistruje ovladač u správce zařízení
  - přijímá od správce zařízení zprávy a zpracovává je
- řídí tok dat mezi klientskou aplikací a ovladačem
- definuje standardní rozhraní pro přístup k zařízení – jak z klientské aplikace, tak z jiného ovladače
  - definuje a implementuje vnější podobu standardního rozhraní zařízení – definuje protokol meziprocesové komunikace a implementuje odpovídající zpracování IPC zpráv na straně ovladače zařízení
  - definuje část rozhraní zařízení, kterou musí implementovat ovladač
- usnadňuje ovladači práci s přerušením
- udržuje seznam zařízení, která byla ovladači úspěšně předána

Vnější rozhraní ovladače

Inicializace vnějšího rozhraní ovladače probíhá ve funkci `driver_main`, které ovladač po své inicializaci předá řízení.

Funkce `driver_main`:

1. si zapamatuje adresu struktury, která reprezentuje ovladač (zapamatuje si ji ve statické proměnné `driver`, při zpracování příchozích zpráv ji použije pro přístup ke vstupním bodům ovladače)
2. provede inicializaci pomocných datových struktur pro práci s přerušením (více viz 8.2.2 – Zpracování přerušení)
3. nastaví generickou obslužnou funkci pro příjem notifikací o přerušeních z jádra (viz 8.2.2 – Zpracování přerušení) – nastavení obslužné funkce se provádí její registrací u asynchronního manažera
4. zaregistruje běžící instanci ovladače u správce zařízení a zároveň nastaví obslužnou funkci pro zpracování meziprocesové komunikace – obslužná funkce se zaregistruje jako obsluha příchozích IPC spojení u asynchronního manažera
5. předá řízení asynchronnímu manažerovi – asynchronní manažer čeká na příchozí IPC spojení (případně notifikace o přerušení z jádra), provádí jejich nízkoúrovňové zpracování a volá příslušné obslužné funkce, které byly u něj předtím zaregistrovány

> POZNÁMKA
>
> Pro každé příchozí IPC spojení asynchronní manažer vytvoří nové vlákénko, v rámci nějž je následně zpracována komunikace mezi klientským a serverovým taskem. Ovladač tedy může obsluhovat několik klientů najednou, a proto je třeba při implementaci ovladače dbát na správnou synchronizaci.

```
int driver_main(driver_t *drv)
{
  // remember the driver structure - driver_ops will be called
  // by generic handler for incoming connections
  driver = drv;❶

  // initialize the list of interrupt contexts
  init_interrupt_context_list(&interrupt_contexts);❷

  // set generic interrupt handler
  async_set_interrupt_received(driver_irq_handler);❸

  // register driver by device manager
  // with generic handler for incoming connections
  devman_driver_register(driver->name, driver_connection);❹

  async_manager();❺

  // Never reached
  return 0;
}
```

Příchozí meziprocesová komunikace je zpracována funkcí `driver_connection` z knihovny libdrv. Task, který chce komunikovat s ovladačem, musí v rámci úvodní IPC zprávy, kterou je spojení navázáno, specifikovat rozhraní ovladače, které bude v rámci dané meziprocesové komunikace použito. Toto rozhraní je určeno prvním argumentem úvodní IPC zprávy – hodnotou argumentu je číselný identifikátor tohoto rozhraní.

Použité rozhraní může být jedním ze tří obecných rozhraní ovladače:

rozhraní pro komunikaci se správcem zařízení
: Správce zařízení prostřednictvím tohoto rozhraní komunikuje s ovladačem – např. předává ovladači k ovládání nově nalezená zařízení. Toto rozhraní je identifikováno číselnou konstantou DRIVER_DEVMAN z výčtového typu driver_interface_t.

rozhraní pro komunikaci s klientskou aplikací
: Prostřednictvím tohoto rozhraní ovladače může aplikace manipulovat s ovládaným zařízením. Identifikátor rozhraní má hodnotu určenou konstantou DRIVER_CLIENT z výčtového typu driver_interface_t.

rozhraní pro komunikaci s jiným ovladačem
Toto rozhraní obvykle používá ovladač synovského zařízení pro komunikaci s ovladačem rodičovského zařízení. Identifikátor rozhraní má hodnotu určenou konstantou DRIVER_DRIVER z výčtového typu driver_interface_t.

Funkce `driver_connection` na základě identifikátoru rozhraní vybere funkci, která bude komunikaci v rámci daného spojení obsluhovat – rozhraní pro komunikaci s klientskou aplikací je stejně jako rozhraní pro komunikaci s jiným ovladačem implementováno funkcí `driver_connection_gen`, rozhraní pro komunikaci se správcem zařízení implementuje funkce `driver_connection_devman` (obě funkce jsou součástí knihovny libdrv).

Důvod, proč je rozhraní pro komunikaci s klientskou aplikací implementováno podobně jako rozhraní pro komunikaci s jiným ovladačem, je, že jak klientská aplikace, tak jiný ovladač se připojují k ovladači za účelem manipulace s ovládaným zařízením a používají k tomu stejný mechanismus. Tím mechanismem jsou rozhraní zařízení – standardní předdefinovaná nebo definovaná konkrétním ovladačem (viz `default_handler` ve struktuře operací ovladače device_ops_t).

Komunikace se správcem zařízení

Komunikaci se správcem zařízení lze rozdělit na následující části:

počáteční registrace ovladače u správce zařízení
Registraci běžící instance ovladače provádí funkce devman_driver_register volaná z funkce driver_main. Poté, co získá od naming service spojení na správce zařízení, zašle funkce `devman_driver_register` správci zařízení zprávu DEVMAN_DRIVER_REGISTER a následně textový řetězec se jménem ovladače. Správce zařízení pak vyhledá ve svém seznamu nainstalovaných ovladačů podle jména příslušný ovladač, změní jeho stav (již běží jeho instance), a zapamatuje si IPC spojení na ovladač.

obsluha žádostí, které správce zařízení zasílá ovladači
Obsluhu žádostí zaslaných od správce zařízení provádí funkce `driver_connection_devman`, která přijímá příchozí zprávy od správce a volá odpovídající obslužné funkce. V současné době správce zařízení tímto způsobem pouze předává ovladači nově nalezená zařízení – IPC zprávou s číslem metody DRIVER_ADD_DEVICE. V budoucnu při rozšíření o možnost zařízení odebírat by správce takto mohl zjišťovat od ovladače, zda je možné zařízení odebrat, mohl by nařídit ovladači uvést zařízení do klidového stavu a mohl zařízení ovladači odebrat.

IPC zpráva s číslem metody DRIVER_ADD_DEVICE je zpracována tak, že knihovna libdrv přijme od správce zařízení handle a jméno předávaného zařízení, vyplní jimi nově alokovanou strukturu zařízení a předá tuto strukturu funkci `add_device`, která je součástí u knihovny zaregistrovaných operací ovladače (driver_ops_t ve struktuře driver_t daného ovladače). Pokud tato funkce vrátí návratovou hodnotu značící úspěch, je zařízení přidáno do seznamu zařízení ovládaných ovladačem. Tato návratová hodnota je nakonec předána správci zařízení jako výsledek operace předání zařízení.

registrace synovských zařízení (ovladače sběrnic)
Ovladače zařízení sběrnicového typu mají na starost detekci zařízení připojených

na ovládanou sběrnici. Nalezená synovská zařízení pak ovladače sběrnic musejí zaregistrovat u správce zařízení, aby těmto zařízením přiřadil vhodné ovladače a aby tato zařízení mohly používat ostatní části systému (aplikace, souborové systémy atd.).

Registraci synovského zařízení provádí ovladač sběrnice voláním funkce `child_device_register` z knihovny libdrv. Ovladač sběrnice této funkci předá strukturu zařízení sběrnice a strukturu synovského zařízení, kterou předtím ovladač vytvořil (funkcí `create_device` z libdrv). Funkce `child_device_register` přidá strukturu synovského zařízení do seznamu zařízení ovladače a zašle správci zařízení zprávu DEVMAN_ADD_CHILD_DEVICE, kterou doplní identifikátorem (handle) rodičovského zařízení, jménem synovského zařízení a seznamem match ids a odpovídajících match scores. Nedojde-li k chybě, správce zařízení přidá synovské zařízení na správné místo ve stromě zařízení, který si udržuje, a vrátí ovladači sběrnice handle, který přiřadil synovskému zařízení. (Tohoto handle v budoucnosti použije ovladač synovského zařízení, pokud se bude chtít připojit k ovladači rodiče.)

Rozhraní pro komunikaci se zařízením

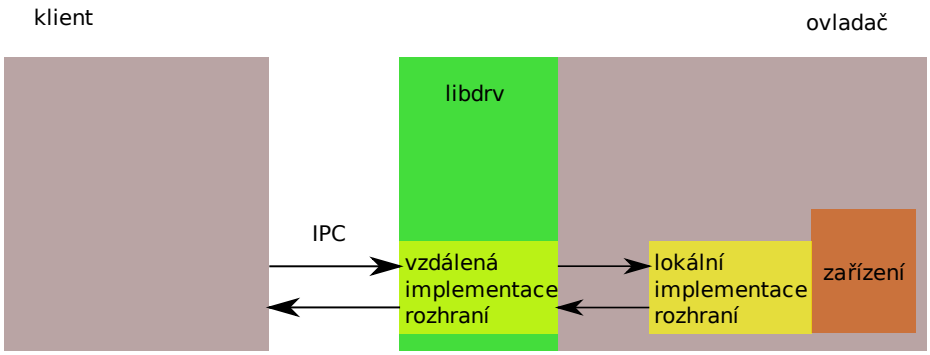
Klientské aplikace (stejně jako jiné ovladače zařízení) se k ovladači zařízení připojují vždy na konkrétní jím ovládané zařízení. Protože pro komunikaci se zařízením se používají v obou případech tytéž mechanismy – tj. operace asociované se zařízením a předdefinovaná rozhraní zařízení – provádí obsluhu obou těchto případů meziprocesové komunikace tatáž knihovní funkce `driver_connection_gen`.

Funkce `driver_connection_gen` je volána ve chvíli, kdy je navázáno nové spojení. Tato funkce na začátku přijme od druhé strany komunikace handle zařízení, se kterým chce druhá strana komunikovat. Následně funkce podle handle vyhledá v seznamu ovládaných zařízení příslušnou strukturu zařízení, která je pak platná v průběhu celé komunikace. Pokud není zařízení v seznamu nalezeno, je vrácena chyba a komunikace je okamžitě přerušena. V opačném případě funkce `driver_connection_gen` v cyklu přijímá žádosti od klientské aplikace (nebo ovladače) a volá funkce pro jejich zpracování, dokud klientská aplikace komunikaci neukončí.

Zpracování klientské žádosti je předáno buď některému z předdefinovaných rozhraní (pokud číslo metody dané IPC zprávy některému z nich odpovídá), nebo implicitnímu handleru. Pokud metoda dané zprávy neodpovídá žádnému z předdefinovaných rozhraní a ovladač neimplementuje implicitní handler, je klientovi vrácena chyba. Stejně tak je chyba vrácena i v případě, že metoda odpovídá standardnímu předdefinovanému rozhraní, které ovladač neimplementuje.

Každé předdefinované rozhraní má jednoznačný číselný identifikátor definovaný ve výčtovém typu dev_inferface_idx_t. Každé metodě rozhraní je přiřazen jednoznačný číselný identifikátor v rámci daného rozhraní.

Při zpracování klientské žádosti předdefinovaným rozhraním spolupracuje tzv. vzdálená a tzv. lokální část tohoto rozhraní (viz obrázek 8.1). *Vzdálená část rozhraní (remote interface)* definuje IPC protokol odpovídající danému rozhraní a je implementována knihovnou libdrv. *Lokální část rozhraní (local interface)* definuje dané rozhraní nezávisle na IPC protokolu a implementuje ji ovladač pro konkrétní model zařízení.



Obrázek 8.1: Komunikace klientské aplikace se zařízením prostřednictvím předdefinovaného rozhraní

Lokální část rozhraní je definována strukturou, která obsahuje ukazatele na funkce, které představují z objektového hlediska metody rozhraní. Každé lokální rozhraní má vlastní strukturu (vlastní datový typ), jednotlivé funkce rozhraní definované touto strukturou mohou mít navzájem odlišný prototyp (neberou stejné argumenty). Lokální část rozhraní implementuje ovladač tak, že:

- implementuje jednotlivé funkce rozhraní,
- vyplní jejich adresy do struktury rozhraní,
- odkaz na tuto strukturu přidá do pole `interfaces` struktury operací zařízení (typu device_ops_t) na index daný číselným identifikátorem rozhraní a
- strukturu operací zařízení přiřadí zařízení, pro které dané rozhraní implementuje.

Vzdálená část rozhraní se skládá z tabulky vzdálených metod, což jsou funkce implementující zpracování protokolu meziprocesové komunikace pro jednotlivé metody rozhraní. Tabulka vzdálených metod je indexována jednoznačnými číselnými identifikátory, které jsou metodám přiřazeny v rámci daného rozhraní. Všechny vzdálené metody mají tentýž funkční prototyp:

```
typedef void remote_iface_func_t(
    device_t*, void *, ipc_callid_t, ipc_call_t *);
```

Prvním argumentem vzdálené metody je struktura zařízení, s nímž je prováděna příslušná operace. Druhý argument obsahuje adresu struktury, která představuje lokální implementaci daného rozhraní. Následují argumenty specifikující zpracovávanou zprávu meziprocesové komunikace a její argumenty.

Vzdálená metoda rozumí dané IPC zprávě a jejím argumentům, případně dalším záležitostem souvisejícím s daným protokolem – někdy je kromě zpracování argumentů dané zprávy potřeba provést dodatečný příjem, zaslaní nebo sdílení většího množství dat.

Kromě toho vzdálená metoda zná datový typ struktury reprezentující lokální část rozhraní a ví, která funkce z této struktury implementuje tutéž operaci nad zařízením jako tato vzdálená metoda, a ví, jaké tato funkce bere parametry.

Díky znalosti lokální části rozhraní i IPC protokolu vzdálená metoda umí přijmout data od klientské aplikace, předzpracovat je a pak je ve správném pořadí předat odpovídající funkci z lokální části implementace rozhraní. Data navrácená touto funkcí umí správně interpretovat a předat zpět klientovi.

Pro lepší názornost následuje komentovaný výpis funkce `driver_connection_gen`, ve které probíhá hlavní zpracování žádostí od klientských aplikací. Z důvodu úspory místa výpis není úplný – neobsahuje zpracování některých chybových stavů (neexistující handler pro zprávu s danou metodou atd.), ladicí výpisy a komentáře v angličtině, které se ve zdrojových souborech nacházejí.

Funkce `driver_connection_gen`

1. V seznamu zařízení se podle handle, které ovladači zaslala klientská aplikace, vyhledá struktura zařízení, s kterým se bude komunikovat.
2. Pokud ovladač implementuje pro dané zařízení funkci `open` (ze struktury device_ops_t), je tato funkce zavolána. Tato funkce slouží pro inicializaci zařízení ve chvíli, kdy se k němu připojí nový klient. Prostřednictvím této funkce může ovladač také připojení klienta odmítnout (např. pokud chce zabránit, aby k zařízení přistupovalo více klientů současně).
3. Následuje cyklus zpracování jednotlivých žádostí o manipulaci se zařízením.
4. Příjem zprávy. Jedna zpráva obvykle odpovídá jedné operaci se zařízením.
5. Pro každou příchozí zprávu je třeba prozkoumat její metodu, aby bylo jasné, co se má se zařízením provést.
6. Přijde-li zpráva o ukončení komunikace ze strany klienta (tj. zpráva s číslem metody IPC_M_PHONE_HUNGUP) a implementuje-li ovladač pro zařízení funkci `close` (ze struktury device_ops_t), je tato funkce zavolána. Tato funkce slouží k oznámení skutečnosti, že klient dokončil práci se zařízením, ovladači zařízení.
7. Metoda příchozí zprávy je přepočítána na identifikátor předdefinovaného rozhraní zařízení.
8. Pokud se identifikátor rozhraní zařízení nenachází v rozsahu identifikátorů předdefinovaných rozhraní, je zpráva předána k obsluze implicitnímu handleru, který pro dané zařízení implementuje ovladač. Kromě zprávy je implicitnímu handleru předána i struktura zařízení. Po zpracování implicitním handlerem zpracování dané zprávy končí.
9. Pokud metoda zprávy odpovídá předdefinovanému rozhraní, je vyhledána lokální část tohoto rozhraní implementovaná ovladačem.
10. Dále je vyhledána vzdálená část téhož rozhraní implementovaná knihovnou libdrv.
11. Následně je nalezena metoda vzdálené části rozhraní, jejíž identifikátor se nachází v prvním argumentu zprávy.
12. Metoda vzdálené části rozhraní je zavolána a v parametrech jsou jí předány: struktura zařízení, lokální část implementace rozhraní a zpracovávaná zpráva. Tato metoda klientskou žádost obslouží s pomocí znalosti odpovídající části protokolu a s pomocí lokální části implementace rozhraní.

```
static void driver_connection_gen(
            ipc_callid_t iid, ipc_call_t *icall, bool drv)
{
  device_handle_t handle = IPC_GET_ARG2(*icall);
  device_t *dev = driver_get_device(&devices, handle);❶

  int ret = EOK;
  if (NULL != dev->ops->open)
    ret = (*dev->ops->open)(dev);❷

  ipc_answer_0(iid, ret);
  if (EOK != ret)
    return;

  while (1) {❸
    ipc_callid_t callid; ipc_call_t call; int iface_idx;
    callid = async_get_call(&call);❹
    ipcarg_t method = IPC_GET_METHOD(call);

    switch  (method) {❺
    case IPC_M_PHONE_HUNGUP:
      if (NULL != dev->ops->close)
        (*dev->ops->close)(dev);❻
      ipc_answer_0(callid, EOK);
      return;
    default:
      iface_idx = DEV_IFACE_IDX(method);❼

      if (!is_valid_iface_idx(iface_idx)) {
        remote_handler_t *default_handler =
          device_get_default_handler(dev);
        (*default_handler)(dev, callid, &call);❽
        break;
      }

      void *iface = device_get_iface(dev, iface_idx);❾

      remote_iface_t* rem_iface = get_remote_iface(iface_idx);❿

      ipcarg_t iface_method_idx = IPC_GET_ARG1(call);
      remote_iface_func_ptr_t iface_method_ptr =
        get_remote_method(rem_iface, iface_method_idx);⓫

      (*iface_method_ptr)(dev, iface, callid, &call);⓬
      break;
    }
  }
}
```

Zpracování přerušení

Knihovna libdrv poskytuje ovladačům vlastní funkce pro registraci a obsluhu přerušení, které lépe vyhovují datovému modelu použitému novými ovladači než podobné funkce, které jsou součástí původního rozhraní pro ovladače zařízení. Funkce z původního rozhraní jsou funkcemi z libdrv vnitřně volány, libdrv nad nimi tvoří pouze tenkou, uživatelsky přívětivější obálku.

Knihovna libdrv umožňuje pro každou dvojici tvořenou strukturou zařízení a číslem přerušení zaregistrovat funkci pro obsluhu přerušení. Tato funkce je volána knihovnou libdrv ve chvíli, kdy přijde pro dané zařízení z jádra notifikace o přerušení s daným číslem.

Registrace obslužné funkce se provádí voláním funkce:

```
int register_interrupt_handler(
    device_t *dev, int irq, interrupt_handler_t *handler,
    irq_code_t *pseudocode);
```

První argument funkce je zařízení, pro nějž obslužnou funkci registrujeme. Druhý argument obsahuje číslo přerušení, třetí adresu funkce, která se má zavolat při příchodu notifikace o daném přerušení z jádra. Poslední argument obsahuje pseudokód, který má handler přerušení v jádře vykonat bezprostředně po příchodu registrovaného přerušení. Definice i zpracování pseudokódu je převzato z původního rozhraní pro ovladače zařízení.

Registrace obsluhy daného přerušení pro dané zařízení se dá zrušit voláním funkce:

```
int unregister_interrupt_handler(device_t *dev, int irq);
```

Samotná obslužná funkce má následující prototyp:

```
typedef void interrupt_handler_t(
    device_t *dev, ipc_callid_t iid, ipc_call_t *icall);
```

Prvním argumentem obslužné funkce je zařízení, které vyvolalo přerušení. Další dva argumenty popisují IPC zprávu (notifikaci) zaslanou ovladači z jádra po příchodu přerušení a její argumenty (v nich se mohou nacházet výsledky interpretace pseudokódu v jádře – prostřednictvím těchto argumentů může pseudokód předávat ovladači data z jádra).

Výhody použití libdrv oproti použití čistě původního driver frameworku jsou:

- Původní rozhraní umožňovalo ovladači zaregistrovat si pouze jednu obslužnou funkci pro příjem všech notifikací o přerušení. Nové rozhraní umožňuje zaregistrovat si jednu obslužnou funkci pro každou dvojici zařízení a čísla přerušení.
- Jedním z parametrů předávaných obslužné funkci knihovnou libdrv je struktura zařízení, které přerušení vyvolalo. Původní rozhraní pro ovladače zařízení nepočítalo s možností ovládat více zařízení v rámci jedné běžící instance ovladače, takže informaci o tom, které zařízení přerušení vyvolalo, v rámci notifikace ovladači nepředávalo.

Knihovna libdrv při implementaci registrace a obsluhy přerušení interně používá původní rozhraní pro ovladače zařízení a tzv. kontexty přerušení. *Kontext přerušení (interrupt context)* asociuje konkrétní zařízení a číslo přerušení s obslužnou funkcí. Každý kontext přerušení má knihovnou libdrv v rámci ovladače přiřazen jednoznačný číselný identifikátor. Knihovna libdrv si pro ovladač udržuje seznam kontextů přerušení.

Ve chvíli, kdy ovladač zavolá funkci `register_interrupt_handler`, je vytvořen nový kontext přerušení, knihovna libdrv mu přiřadí jednoznačný identifikátor, zařadí jej do seznamu kontextů přerušení a zaregistruje si dané přerušení u původního driver frameworku. V rámci registrace knihovna libdrv požádá původní driver framework, aby při notifikaci o daném přerušení použil identifikátor kontextu přerušení jako číslo metody notifikační IPC zprávy.

Při inicializaci ovladače ve funkci `driver_main` knihovna libdrv nastaví u asynchronního manažera svou funkci pro příjem notifikací o přerušení z jádra:

```
// set generic interrupt handler
async_set_interrupt_received(driver_irq_handler);
```

Po příchodu notifikace o přerušení funkce `driver_irq_handler` vyhledá podle identifikátoru předaném v metodě notifikační IPC zprávy kontext přerušení, vezme z něj adresu struktury zařízení a adresu obslužné funkce, kterou si zaregistroval ovladač. Tuto funkci následně zavolá a předá jí adresu struktury zařízení a IPC zprávu, která byla použita pro notifikaci.

Povolení přerušení

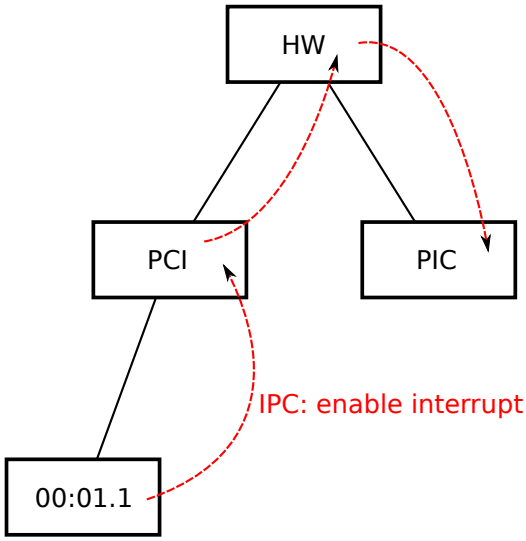
Povolení přerušení bylo z časových důvodů vyřešeno pouze provizorním způsobem – prostřednictvím dočasně zavedeného systémového volání lze o povolení, případně zakázání přerušení, požádat ovladač řadiče přerušení, který se nachází v jádře. Toto řešení nefunguje na všech platformách a v budoucnu bude nahrazeno použitím ovladače řadiče přerušení v uživatelském prostoru (ten zatím nebyl pro architekturu IA32, na které byla prototypová implementace vyvíjena, implementován).

S povolením přerušení z uživatelského prostoru souvisí zajímavý problém. Tento problém se týká uspořádání ve stromě zařízení. Ovladač obvykle požaduje služby od ovladače sběrnice, na kterou je jím ovládané zařízení připojeno. To znamená, že komunikace mezi ovladači obvykle ve stromě zařízení probíhá směrem nahoru od synovských zařízení k jejich rodičům. To může fungovat i rekurzivně, ovladač sběrnice může přeposlat žádost o další úroveň výš svému rodiči atd., dokud se žádost nedostane ke kořeni stromu. Řadiče přerušení se z tohoto schématu poněkud vymykají – řadič přerušení, jehož služby ovladač zařízení potřebuje použít pro povolení přerušení pro ovládané zařízení, se nemusí ve stromě nacházet na cestě od zařízení ke kořeni.

Tento problém lze řešit následujícím způsobem: Ve stromě zařízení se po cestě od zařízení, které potřebuje povolit přerušení, do kořene stromu nachází sběrnice, jejíž ovladač ví, kde ve stromě se nachází řadič přerušení, který by měl požadované přerušení povolit. Ovladač této sběrnice pak může žádosti o povolení přerušení zpracovávat tak, že se pomocí správce zařízení připojí k ovladači tohoto řadiče (to je možné, když je známa cesta k řadiči ve stromě zařízení) a žádost o povolení přerušení ovladači řadiče přepošle.

Toto řešení je demonstrováno na obrázku 8.2. Na obrázku vidíme část stromu zařízení (pro architekturu IA-32). V kořeni se nachází zařízení reprezentující kořenovou sběrnici na dané platformně (tedy kořen stromu všech fyzických zařízení), pod kořenem jsou zapojena dvě zařízení – jedno reprezentuje sběrnici PCI a druhé řadič přerušení Programmable Interrupt Controller (PIC). Řadič přerušení je reprezentován jako jedno zařízení, ve skutečnosti sice funkci řadiče přerušení zastávají dva samostatné čipy, ty ale navzájem spolupracují, jako by šlo o jediné zařízení. Na sběrnici PCI je připojeno jedno koncové zařízení. Ovladač tohoto koncového zařízení potřebuje pro

toto zařízení povolit přerušení, a tak prostřednictvím meziprocesové komunikace zašle odpovídající žádost ovladači rodičovské PCI sběrnice. Ovladač sběrnice PCI žádost přepošle ovladači ve stromě o úroveň výš, tedy ovladači kořenové sběrnice pro danou platformu. Tento ovladač ví, kde ve stromě zařízení se na dané platformně nachází řadič přerušení, a díky tomu správně přepošle žádost jeho ovladači, který žádost zpracuje a přerušení povolí.



Obrázek 8.2: Povolení přerušení z uživatelského prostoru (port pro IA-32)

8.2.3 Konkrétní ovladače

Součástí prototypové implementace jsou následující ovladače:

root

Minimalistický ovladač virtuální sběrnice, která se nachází v kořeni pomyslného stromu všech fyzických i virtuálních zařízení. Tento ovladač má jediný úkol: od správce zařízení převzít zařízení v kořeni stromu a u správce pro toto zařízení zaregistrovat synovské zařízení reprezentující kořen podstromu, ve kterém se nacházejí všechna fyzická zařízení. Synovské zařízení pojmenováno "hw" a jako jediné match id je mu nastaven textový řetězec s názvem platformy, na které ovladač běží. Např. pro počítače s procesorem architektury IA-32 se použije řetězec "ia32".

V budoucnu by mohl ovladač root sloužit také k vytváření virtuálních zařízení.

Tomuto ovladači jsou přiřazována zařízení s match id "root" (v systému je jen jedno takové zařízení a vytváří jej samotný správce zařízení v rámci své inicializace).

Zdrojové soubory tohoto ovladače se nacházejí v adresáři `uspace/srv/drivers/-root`.

rootia32

Ovladač pomyslného zařízení v kořeni stromu všech fyzických zařízení pro architekturu IA-32. Jeho úkolem je zatím pouze vytvořit a zaregistrovat synovské zařízení reprezentující kořenovou PCI sběrnici. Pro toto synovské zařízení implementuje rozhraní, jehož pomocí může jiný ovladač zjistit, jaké hardwarové

prostředky byly tomuto zařízení přiděleny (přidělen mu je rozsah adres v I/O prostoru, které na této platformně slouží k přístupu ke konfiguračnímu adresovému prostoru sběrnice PCI). Synovskému zařízení je tímto ovladačem přiřazeno match id "intel_pci".

Ovladači je ovládané zařízení přiřazeno na základě match id "ia32".

Zdrojové soubory ovladače na nacházejí v adresáři `uspace/srv/drivers/-rootia32`.

pciintel

Ovladač PCI sběrnice, který implementuje přístup ke konfiguračnímu adresovému prostoru známý pod názvem Configuration Mechanism #1 (viz. [pci]). Ke konfiguračnímu prostoru se přistupuje prostřednictvím dvou speciálních registrů v odděleném I/O adresovém prostoru – do prvního z nich se zadává adresa v konfiguračním adresovém prostoru, pomocí druhého se přistupuje k datům na této adrese. Tento způsob implementace konfiguračnímu adresovému prostoru PCI se používá na intelských platformách (s procesory z rodiny x86), na jiných platformách (např. UltraSPARC II – viz [psycho]) bývá konfigurační adresový prostor sběrnice PCI paměťově mapovaný.

Tento ovladač průchodem konfiguračního adresového prostoru detekuje zařízení připojená na ovládanou PCI sběrnici a zjišťuje, jaké hardwarové prostředky jim byly přidělené firmwarem. Nalezená synovská zařízení registruje u správce zařízení a implementuje pro ně rozhraní, jehož prostřednictvím mohou ostatní ovladače zjišťovat, jaké adresové rozsahy a čísla přerušení jsou těmto zařízením přiděleny. Nalezeným synovským zařízením přiděluje match ids ve tvaru "pci/ven=xxxx&dev=yyyy", kde `xxxx` je hexadecimální číselný identifikátor výrobce zařízení (tzv. vendor ID) a `yyyy` je identifikátor přidělený modelu zařízení výrobcem (tzv. device ID) – viz [pcidatabase].

Ovladači je přiřazeno zařízení s match id "intel_pci".

Zdrojové soubory ovladače na nacházejí v adresáři `uspace/srv/drivers/-pciintel`.

isa

Ovladač ISA sběrnice. Tento ovladač má za úkol provést u správce zařízení registraci zařízení, která se na dané sběrnici nacházejí. Jelikož ISA nepodporuje automatickou detekci zařízení, jsou u správce zařízení registrována zařízení specifikovaná konfiguračním souborem ovladače – `uspace/srv/drivers/isa/-isa.dev`. Tento konfigurační soubor specifikuje, jaká zařízení mají být registrována a jaké rozsahy adres a čísla přerušení jsou jim přiděleny – přidělené adresy a čísla přerušení lze pro každé zařízení zjišťovat prostřednictvím odpovídajícího rozhraní, které tento ovladač pro synovská zařízení implementuje.

Ovladači je přiřazeno zařízení s match id "pci/ven=8086&dev=7000", což je match id přiřazené ovladačem sběrnice PCI mostu z PCI sběrnice na ISA sběrnici na virtuálním stroji, na kterém byl prototyp testován – tj. na PC simulovaném simulátorem qemu (simulovaná architektura IA-32).

Zdrojové soubory ovladače na nacházejí v adresáři `uspace/srv/drivers/-isa`.

ns8250

Ovladač sériového portu (pro rodinu čipů kompatibilních s NS 8259).

Ovladači jsou přiřazena všechna zařízení s match id "isa/serial" (tato match id přiděluje ovladač sběrnice ISA sériovým portům připojeným na ovládanou sběrnici). Pro každé zařízení, které je tomuto ovladači předáno, ovladač kontroluje, zda je zařízení skutečně v systému přítomno – ovladač rodičovské sběrnice ISA to zjistit neumí, protože sběrnice ISA nepodporuje automatickou detekci. V rámci této kontroly od ovladače ISA sběrnice zjistí, jaké číslo přerušení a jaké adresy byly zařízení přiřazeny. Přítomnost zařízení následně kontroluje několika pokusnými zápisy a čtením dat z daných adres.

Pro ovládaná zařízení (sériové porty) implementuje znakové rozhraní, pomocí nějž mohou klientské aplikace zasílat a přijímat data. Kromě toho definuje vlastní protokol pro nastavování parametrů sériové komunikace. Nastavování parametrů sériové komunikace je implementováno implicitním handlerem definovaným v rámci operací asociovaných s ovládanými zařízeními.

Zdrojové soubory ovladače na nacházejí v adresáři `uspace/srv/drivers/ns8259`. Definice IPC protokolu pro zjišťování a nastavování parametrů sériové komunikace se nachází v souboru `uspace/lib/c/ipc/serial_ctl.h`.

8.3 Integrace s device mapperem a devfs

Správce zařízení registruje některá zařízení u device mapperu a umožňuje tak přístup k těmto zařízením prostřednictvím souborového systému devfs. Správce zařízení registruje u device mapperu zařízení jak podle fyzické hierarchie, tak podle zařazení do tříd. Současná implementace zveřejnění fyzické a funkční hierarchie zařízení prostřednictvím device mapperu a devfs odděluje zařízení, jejichž ovladače používající nový driver framework, od ostatních zařízení.

Současná implementace device mapperu i devfs má určitá omezení, jimž se musel správce zařízení přizpůsobit.

Device mapper umožňuje registrovat zařízení pouze v dvouúrovňové hierarchii – zařízení lze rozdělit do jmenných prostorů (namespaces), jmenné prostory ale nelze vnořovat. Fyzická hierarchie zařízení má pochopitelně více úrovní, tedy nelze strom zařízení zveřejnit prostřednictvím device mapperu (resp. devfs) přirozeným způsobem, kde by vnitřní uzly stromu zařízení odpovídaly jmenným prostorům (resp. adresářům) a koncové uzly zařízením (resp. souborům).

Souborový systém devfs, stejně jako ostatní existující implementace souborových systémů v operačním systému HelenOS nepodporuje symbolické linky – podpora pro symbolické linky zatím není zavedena ani ve virtuálním souborovém systému (vfs). Nelze tedy zatím propojit navzájem si odpovídající zařízení ve dvou ortogonálních hierarchiích – hierarchii fyzického zapojení a hierarchii podle funkčních tříd.

S ohledem na tato omezení jsou zařízení u device mapperu registrována následujícím (provizorním) způsobem:

strom zařízení

Strom zařízení se nachází v adresáři `/dev/devices` (odpovídá jmennému prostoru `devices` u device mapperu). Každé zařízení ve stromě je reprezentováno souborem, jehož název obsahuje celou cestu k zařízení ve stromě, přičemž jako

oddělovač jmen jednotlivých uzlů ve stromě na cestě k zařízení slouží zpětné lomítko (nekoliduje s lomítkem používaným pro oddělení jednotlivých elementů cesty v souborovém systému). Jméno zařízení v kořeni stromu je prázdný řetězec.

Např. Mějme zařízení `A` zapojené ve stromě bezprostředně pod kořenem, zařízení `A` je rodičem zařízení `B` a to je rodičem zařízení `C`. Zařízení `C` tedy bude u device mapperu registrováno pod názvem `\A\B\C` ve jmenném prostoru `devices`. Celá cesta k souboru zařízení `C` tedy bude `/dev/devices/\A\B\C`.

funkční třídy zařízení

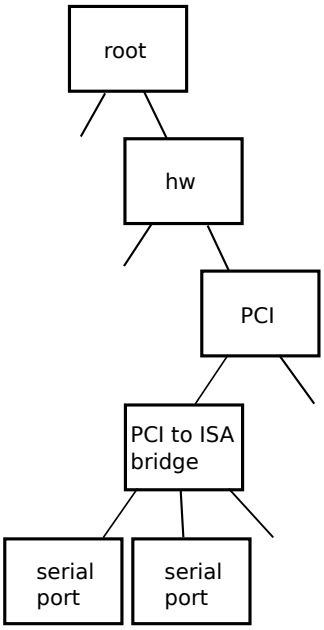
Každé zařízení může být zařazeno do libovolného počtu tříd, v rámci třídy je zařízení přidělen jednoznačný identifikátor.

Hierarchie zařízení rozdělených podle tříd se nachází v adresáři `/dev/class` (odpovídá jmennému prostoru `class` u device mapperu). Zařízení, které bylo zařazeno do třídy `X` a byl mu v rámci této třídy přiřazen identifikátor `Y`, bude reprezentováno souborem `X\Y`. Celá cesta k souboru zařízení v rámci třídy bude `/dev/class/X\Y`.

Soubor zařízení v rámci třídy není nijak propojen se souborem téhož zařízení v rámci fyzické hierarchie. Pokud bude v budoucnu do souborových systémů v HelenOS přidána podpora pro symbolické linky, mohl by soubor reprezentující zařízení v rámci třídy být symbolickým linkem na soubor téhož zařízení v rámci fyzické hierarchie.

8.4 Inicializace

V tato část textu pro úplnost uvádí typický scénář inicializace jednotlivých součástí driver frameworku a jejich vzájemné spolupráce po startu systému. Scénář je demonstrován na architektuře IA-32 simulované pomocí simulátoru qemu. Použitý virtuální stroj má mimo jiné sběrnici PCI, most ze sběrnice PCI na sběrnici ISA a dva sériové porty (kompatibilní s NS 8250). Tomuto fyzickému zapojení odpovídá strom zařízení na obrázku 8.3.



Obrázek 8.3: Strom zařízení na virtuálním stroji v qemu

Inicializaci uživatelského prostoru obstarává task init (viz zdrojové soubory v adresáři `uspace/app/init`), který po startu systému spustí řadu služeb. Ze služeb souvisejících s driver frameworkem jsou nejprve spuštěny serverové tasky pro Device file system (zkráceně devfs, viz `uspace/srv/fs/devfs`) a Virtual file system (zkráceně vfs, viz `uspace/srv/vfs`), u kterého se devfs registruje jako jedna z implementací souborového systému. Vzápětí je připojen kořenový adresář a pod něj adresář /dev pro Device file system. Jako další je spuštěn Device mapper a po něm ovladače napsané s pomocí původního rozhraní pro ovladače zařízení, které u něj registrují sebe i ovládaná zařízení. Jako poslední je spuštěn Device manager.

Device mapper a Device manager se po svém startu registrují u Naming service, aby jejich služby byly přístupné ostatním aplikacím v uživatelském prostoru.

Device manager v rámci své inicializace nejprve vytvoří seznam všech dostupných ovladačů – prohledá adresář `/srv/drivers` a pro každý v něm nalezený ovladač načte z konfiguračního souboru ovladače seznam match IDs pro přiřazování ovladačů k zařízením a zapamatuje si cestu ke spustitelnému binárnímu souboru ovladače.

Dalším a posledním krokem v inicializaci správce zařízení je vytvoření a registrace virtuálního zařízení, které má funkci pomyslného kořene stromu všech virtuálních i fyzických zařízení, kořenové zařízení se chová jako instance speciální virtuální sběrnice. Kořenovému zařízení (sběrnici) je přiřazeno jediné speciální match ID "root". Registrace kořenového zařízení probíhá obdobně jako registrace ostatních zařízení u správce – tj. v rámci této registrace je podle match IDs vyhledán vhodný ovladač – v tomto případě ovladač root – následně je přiřazen kořenovému zařízení, a jelikož ještě neběží, je spuštěn. Od této chvíle je správce zařízení plně inicializován a čeká na příchod IPC žádostí od ovladačů a klientských aplikací.

Po startu a inicializaci ovladač root u správce zařízení zaregistruje svou běžící instanci a tím mu oznámí, že je připraven zpracovávat příchozí žádosti. Správce zařízení se k běžící instanci ovladače připojí a předá jí všechna zařízení, která byla ovladači dosud přiřazena – tj. v tomto případě kořen stromu zařízení. Ovladač správci zařízení oznámí, že předávané zařízení je v pořádku a že s jeho převzetím souhlasí, a následně u správce zařízení zaregistruje nové zařízení, které reprezentuje kořen všech fyzických

zařízení na dané platformně. Toto zařízení je registrováno jako syn kořene stromu zařízení – tj. jako zařízení připojené na kořenové sběrnici. Match ID registrované společně s tímto synovským zařízením je textový identifikátor dané platformy – v tomto případě "ia32".

Kořenu fyzických zařízení Device manager přiřadí ovladač rootia32 a tento ovladač spustí. Ovladač se po startu ohlásí a zařízení je mu úspěšně předáno. Ovladač pro toto zařízení vytvoří synovské zařízení, které bude reprezentovat instanci sběrnice PCI. S pomocí libdrv přiřadí ovladač tomuto zařízení device interface s identifikátorem HW_RES_DEV_IFACE, pomocí nějž se může ovladač zařízení zeptat ovladače rodičovského zařízení, jaké hardwarové prostředky byly zařízení přiděleny. Poté ovladač toto zařízení zaregistruje u Device manageru.

Device manager zařízení, které reprezentuje instanci PCI sběrnice, přiřadí ovladač a spustí jej. Ovladač PCI sběrnice od správce zařízení po startu sběrnici převezme a vyžádá si od správce připojení na ovladač rodičovského zařízení (tj. na běžící instanci ovladače rootia32). Žádosti je vyhověno a ovladač PCI sběrnice se připojí na své zařízení u rodičovského ovladače a pomocí odpovídající funkce z device interface s identifikátorem HW_RES_DEV_IFACE zjistí, na jakých adresách se v odděleném I/O adresovém prostoru nacházejí registry, které ovladač potřebuje pro přístup ke konfiguračnímu adresovému prostoru PCI. Průchodem konfiguračního adresového prostoru ovladač PCI sběrnice provede detekci všech zařízení na tuto sběrnici připojených a každé z nich zaregistruje jako syna PCI sběrnice u správce zařízení, přičemž při registraci je jako match ID použit textový řetězec udávající PCI vendor a device ID.

Správci zařízení se povede najít ovladač jen pro jedno ze zařízení připojených na PCI sběrnici – pro most propojující PCI a ISA sběrnici. Podle match ID se na ovládání tohoto zařízení nejlépe hodí ovladač ISA sběrnice, a tak je spuštěn a zařízení je mu předáno. Jelikož sběrnice ISA obecně nepodporuje PnP, vyhledá její ovladač připojená zařízení a jim přidělené hardwarové prostředky (adresy, čísla přerušení) ne na sběrnici, ale v konfiguračním souboru a takto nalezená zařízení následně zaregistruje u Device manageru. Ještě před registrací u správce zařízení zaregistruje ovladač ISA každému synovskému zařízení device interface HW_RES_DEV_IFACE u libdrv.

Ze zařízení na ISA sběrnici se povede přiřadit ovladač pouze dvěma sériovým portům. Jejich ovladač (ns8250) v rámci předání zařízení zjistí od ovladače sběrnice ISA jejich adresy a zkusmým čtením a zápisem z nich ověří, zda jsou zařízení na sběrnici skutečně přítomna. Pokud daná instance sériového portu je přítomna, ovladač si pro toto zařízení zaregistruje znakový device interface u libdrv a oznámí správci zařízení, že předání bylo úspěšné a zařízení je v pořádku, kromě toho ještě přidá zařízení do funkční třídy "serial". Pokud zařízení není přítomno, vrátí ovladač správci odpovídající chybu.

Každé zařízení, které je úspěšně předáno ovladači, je vzápětí správcem zařízení zaregistrováno u Device mapperu ve jmenném prostoru `devices`. Díky tomu lze prohlížet strom zařízení z devfs. Zařízení jsou taktéž u device mapperu registrována při zařazení do funkční třídy (a to ve jmenném prostoru `class`), prostřednictvím devfs tedy lze do určité míry prozkoumat jak vzájemné zapojení zařízení do fyzické hierarchie, tak rozdělení zařízení do funkčních tříd. Po dokončení inicializace popsané touto částí textu lze tedy na příkazové řádce operačního systému HelenOS spuštěného z qemu po zadání správných příkazů spatřit následující výstup:

```
/ # ls /dev/devices
\hw
```

```
\hw\pci0
\hw\pci0\00:01.0
\hw\pci0\00:01.0\com1
\hw\pci0\00:01.0\com2
/ # ls /dev/class
serial\1
serial\2
```

Kapitola 9

Porovnání s existujícími řešeními

V této kapitole srovnáme části rozhraní pro ovladače zařízení v operačním systému HelenOS vytvořené v rámci této práce s odpovídajícími částmi rozhraní pro ovladače zařízení v jiných operačních systémech. Zaměříme se na jednotlivé problémy popsané v kapitolách 5 a 6.

9.1 Instalace a konfigurace

Instalace a konfigurace ovladačů v operačním systému HelenOS je podobná instalaci a konfiguraci ovladačů v operačním systému Solaris. Konfigurace má podobu jednoduchého textového souboru a instalace ovladače spočívá v nakopírování souborů ovladače do standardního adresáře, který je v daném operačním systému vyhrazen k uchovávání nainstalovaných ovladačů. Toto řešení je implementačně nenáročné a plní svůj účel.

O něco složitější je konfigurace ovladačů v operačním systému Linux. Pokud není ovladač zakompilován přímo do jádra, má podobu dynamicky zaveditelného modulu jádra a konfigurovatelné parametry ovladače jsou součástí tohoto zaveditelného modulu. Pro nastavení hodnot těchto parametrů je potřeba použít specializovanou utilitu – modprobe. Program modprobe je spouštěný z příkazové řádky a slouží k dynamickému zavádění modulů do jádra. Při zavádění modulu umožňuje nastavit hodnoty konfigurovatelných parametrů zaváděného modulu. Utilita modprobe je také používána při startu systému pro automatické zavádění některých modulů. Seznam těchto modulů a hodnot jejich konfigurovatelných parametrů se nachází v konfiguračním souboru `/etc/modules.conf`.

V operačním systému Windows se instalace ovladačů provádí pomocí instalačních souborů s příponou `INF` (případně instalačním programem). Tyto instalační soubory specifikují binární součásti ovladače (soubory s příponou `SYS`) a adresáře, kam se mají v rámci instalace ovladače nakopírovat. Dále instalační soubory specifikují identifikátory modelů zařízení, pro něž se má ovladač použít, a záznamy, které se pro ovladač mají nově přidat do Windows Registry a ve kterých budou ve speciálním binárním formátu uloženy konfigurovatelné parametry ovladače. Toto řešení je asi ze všech zmíněných implementačně nejsložitější.

9.2 Řízení životního cyklu ovladače

V operačním systému Windows a v operačním systému Solaris je ovladač zařízení dynamicky zaváděn do jádra až ve chvíli, kdy je nalezeno zařízení, které by měl ovládat. Podobně je tomu nyní i v operačním systému HelenOS – ovladač je v uživatelském prostoru spuštěn až ve chvíli, kdy je potřeba mu k ovládání předat zařízení. Výhodou tohoto přístupu je úspora systémových prostředků – zaváděny, případně spouštěny, jsou jen ty ovladače, které jsou skutečně potřeba.

V operačním systému Linux je situace složitější. V tomto systému neexistuje žádný univerzální mechanismus pro dynamické zavedení modulu ovladače do jádra až ve chvíli, kdy je nalezeno zařízení, které by tento ovladač měl ovládat. Celá řada modulů různých ovladačů je proto zaváděna automaticky již při startu systému bez ohledu na to, zda jsou či nejsou přítomna zařízení, k jejichž ovládání jsou tyto ovladače určeny. Každý z těchto ovladačů se po zavedení svého modulu registruje u ovladače sběrnice, na které by se mělo nacházet zařízení, pro něž je ovladač určen. Ovladače sběrnic pak registrovaným ovladačům předávají zařízení, která nalezly na ovládaných sběrnicích.

Postupem času se do Linuxu pro některé typy zařízení přidává podpora pro zavádění jejich ovladačů až za běhu v reakci na událost nalezení příslušného zařízení. Tato podpora je specifická pro určitý typ sběrnice, na kterou jsou zařízení připojena, a obvykle vyžaduje spolupráci s uživatelským prostorem – např. z jádra je v souvislosti s nalezením nového zařízení na daném typu sběrnice spuštěn proces v uživatelském prostoru, který tuto událost obslouží zavedením modulu vhodného ovladače do jádra (při nalezení zařízení na sběrnici USB se pouští program `/sbin/hotplug`).

Přístup, který zvolil operační systém Linux, je poněkud nesystematický a nepřehledný.

9.3 Přiřazování ovladače k zařízení

V operačních systémech HelenOS, Solaris a Windows (pro WDM ovladače) existuje jednotný způsob přiřazování ovladačů k zařízením. Výhodou všech těchto přístupů je relativní jednoduchost, obecnost a možnost vybrat z více ovladačů ten nejvhodnější.

V operačním systému Windows jsou ovladače k zařízením přiřazovány na základě identifikátorů modelů zařízení. Ovladač ve svém instalačním souboru (v souboru s příponou `INF`) specifikuje seznam identifikátorů modelů zařízení, která umí ovládat. Některé identifikátory identifikují specifičtější model zařízení a jiné obecnější. Podobné identifikátory jsou přiřazeny jednotlivým zařízením v době jejich detekce. Zařízení je přiřazen nejvhodnější dostupný ovladač z množiny ovladačů, jejichž seznam identifikátorů má neprázdný průnik se seznamem identifikátorů daného zařízení. Vybrán je ovladač se shodou v co nejspecifičtějším identifikátoru modelu zařízení, zohledněna je také verze ovladače (novější má přednost) a bezpečnost (digitálně podepsaný ovladač má přednost před nepodepsaným).

V operačním systému Solaris má každý uzel ve stromě zařízení asociováno několik vlastností (properties). Jedna z nich se nazývá `name` a je povinná. Volitelně může mít zařízení také vlastnost označenou `compatible`. Tyto vlastnosti jsou použity pro přiřazení ovladače k zařízení. Vlastnost `compatible` obsahuje setříděný seznam jmen ovladačů od nejvhodnějšího ovladače po nejméně vhodný. Nejprve se driver framework pokusí přiřadit ovladač na základě vlastnosti `compatible` (v pořadí od nejvhodnějšího ovladače po ten nejméně vhodný), a pokud neuspěje, použije vlastnost `name`.

Strom zařízení a vlastnosti zařízení v něm získá operační systém částečně od firmware a částečně od ovladačů sběrnic pro zařízení, která nedetekoval firmware. Podrobnější popis viz [solaris].

Způsob přiřazování ovladačů k zařízením v operačním systému HelenOS používá ohodnocené identifikátory modelů zařízení a byl popsán v předchozích částech textu (viz 8.1.3).

V operačním systému Linux je způsob přiřazování ovladačů k zařízením specifický pro daný typ sběrnice a toto přiřazování implementuje příslušný ovladač sběrnice. U tohoto ovladače se registrují potenciální ovladače zařízení na něj připojených. Při nálezu nově připojeného zařízení ovladač sběrnice projde seznam zaregistrovaných ovladačů a na základě jejich vlastností hledá vhodný ovladač pro dané zařízení (jak se pozná vhodný ovladač, určí konkrétní ovladač sběrnice). Jakmile je nalezen první vhodný ovladač, zařízení je mu předáno. Např. ovladače zařízení připojených na sběrnici PCI musejí vyplnit strukturu `pci_driver`, součástí této struktury je tabulka identifikátorů modelů zařízení, která ovladač umí ovládat. Pokud jeden z identifikátorů odpovídá nalezenému zařízení, je zařízení předáno funkci `probe` ovladače. Ovladač v této funkci ověří, zda opravdu umí ovládat dané zařízení. Pokud zařízení ovladač ovládat neumí, vrátí z funkce `probe` chybovou návratovou hodnotu a hledání vhodného ovladače v seznamu pokračuje.

9.4 Rozhraní pro přístup k zařízení

Rozhraní pro přístup k zařízení v operačních systémech Solaris a Linux je podobné. Oba tyto operační systémy podporují tři základní třídy zařízení – znaková zařízení, bloková zařízení a síťová rozhraní – a pro každou z těchto tříd zařízení definují strukturu, do níž může ovladač zařízení vyplnit funkce, které implementují operace dané třídy pro ovládaná zařízení. Kromě těchto předdefinovaných operací si může ovladač definovat operace vlastní – konkrétně si může definovat speciální číselné kódy identifikující tyto operace, zpřístupnit je aplikacím v uživatelském prostoru prostřednictvím hlavičkového souboru a implementovat funkci (`ioctl`), které budou tyto kódy zasílány a která bude implementovat jim odpovídající operace. Jelikož třemi třídami zařízení nelze popsat všechny možné typy zařízení a jejich operace, je tento způsob často používán pro zpřístupnění dodatečných operací zařízení, s kterými předem definované třídy nepočítaly. To v některých případech může vést k nekonzistenci – co ovladač, to vlastní svépomocí definované rozhraní – a k nepřehlednosti.

Jiný přístup byl zvolen v operačním systému Windows pro WDM ovladače, předdefinovaných tříd zařízení je zde několik desítek a tvůrci ovladačů mohou dynamicky přidávat třídy další. Vstupní body ovladače pro přístup k zařízení jsou nicméně vždy implementovány stejnou sadou funkcí (tzv. dispatch rutinami), těmto dispatch rutinám jsou zasílány tzv. IRPs (I/O Request Packets), které popisují operace, které se mají s ovládaným zařízením provést, a specifikují parametry těchto operací. Typ operace je identifikován hlavním a vedlejším funkčním kódem IRP. Každému hlavnímu funkčnímu kódu odpovídá jedna dispatch rutina, která slouží ke zpracování všech příchozích IRP s tímto hlavním funkčním kódem.

Stejně jako v systémech Solaris a Linux si ovladač může i v systému Windows definovat vlastní řídicí kódy (I/O contol code), jimiž pak lze ovladač instruovat k provádění speciálních operací s ovládaným zařízením. Tyto řídicí kódy jsou ovladači zasílány ve vedlejších funkčních kódech IRPs s hlavním funkčním kódem IRP_MJ_DEVICE-

_CONTROL. Kromě řídicích kódů, které si definují ovladače samy, je v systému předem definována řada standardních řídicích kódů pro každou předdefinovanou třídu zařízení.

Množství tříd zařízení v operačním systému Windows má za důsledek pestrost nabízených funkcí, ale také složitost navazujících subsystémů. Pro začínajícího vývojáře ovladačů může být toto množství zdrcující. Obecně jsou ovladače ve Windows poměrně komplikované a proniknutí do této oblasti vyžaduje delší dobu studia.

V operačním systému HelenOS zatím nejsou pevně definovány třídy zařízení ani rozhraní pro přístup k zařízením. Součástí návrhu bylo spíše definování obecných mechanismů, jejichž pomocí lze třídy i rozhraní postupně při dalším vývoji přidávat. Jak bylo vidět na příkladu operačních systémů Windows, Solaris a Linux, oba možné extrémní přístupy mají své nevýhody. Proto bude potřeba budoucí návrh sady předdefinovaných rozhraní v operačním systému HelenOS pečlivě zvážit a pokusit se nalézt rovnováhu mezi uvedenými dvěma extrémy.

Kapitola 10

Závěr

V této práci byl vytvořen jednoduchý návrh rozhraní pro ovladače zařízení v operačním systému HelenOS. Hlavním zaměřením práce bylo rozšíření původního driver frameworku o podporu automatické detekce zařízení a jejich reprezentaci v hierarchické struktuře odpovídající jejich fyzickému zapojení. Součástí návrhu bylo definování jednotného rozhraní ovladačů pro přístup k zařízením, toto rozhraní bylo navrženo velice obecně a je možné jej v budoucnu snadno rozšiřovat o nové typy zařízení. Navržené rozhraní pro přístup k zařízením efektivně využívá meziprocesovou komunikaci, odstiňuje od detailů jejího zpracování jednotlivé ovladače, čímž zjednodušuje a zpřehledňuje jejich kód a šetří čas jejich vývojářům.

Jelikož návrh rozhraní pro ovladače zařízení je velice široké téma, byl součástí práce podrobný rozbor cílů a priorit návrhu. Práce systematicky popisuje jednotlivé fáze vývoje od zmíněného stanovení cílů, přes rozbor známých přístupů k jejich řešení, až po návrh řešení a prototypovou implementaci. Na problémy je nahlíženo obecně i z pohledu operačního systému postaveného na mikrojádru.

Součástí práce je také popis původního rozhraní pro ovladače zařízení a vysvětlení, jakým způsobem jsou s ním integrovány nově navržené a implementované části driver frameworku.

10.1 Splnění cílů

V této práci se podařilo splnit většinu cílů uvedených v oddílu 5.2.

Byly navrženy a implementovány části driver frameworku potřebné pro podporu automatické detekce zařízení po startu systému (boot-time plug and play), při návrhu a implementaci byla zohledněna možnost budoucího rozšíření o podporu přidávání a odebírání zařízení za běhu (hotplug).

Bylo navrženo vnější rozhraní ovladačů jak pro komunikaci s řídicími částmi frameworku při automatické konfiguraci systému, tak pro zpřístupnění ovládaných zařízení klientským aplikacím. Byla vytvořena knihovna, která umožňuje ovladačům implementovat toto rozhraní za požadavku pouze minimálního úsilí z jejich strany. Stejný mechanismus, jaký byl použit pro komunikaci mezi ovladači a klientskými aplikacemi, byl použit i pro komunikaci mezi jednotlivými ovladači při jejich vzájemné spolupráci v rámci hierarchického systému pro správu zařízení.

Byl navržen jednoduchý způsob instalace a konfigurace zařízení. Byla navržena a implementována pravidla pro řízení životního cyklu ovladačů a jejich přiřazování k ovladačům. Řízení životního cyklu ovladačů následuje device-centric přístup a díky

tomu je velice efektivní – ovladače jsou spouštěny až ve chvíli, kdy jsou nalezena zařízení, pro jejichž ovládání jsou tyto ovladače potřeba. Mechanismus použitý pro přiřazení ovladače k zařízení není příliš implementačně náročný, ale přitom je velice pružný a obecný. Umožňuje vybrat z více vhodných ovladačů ten nejlepší – umožňuje dát přednost ovladači konkrétního modelu zařízení před obecnějším ovladačem určeným pro obecnější rodinu navzájem kompatibilních zařízení, stejně tak upřednostňuje použití ovladače, který plně podporuje všechny funkce zařízení, před ovladačem, který podporuje jen jejich část.

Navrhovaná vylepšení původních částí driver frameworku sice nebyla z časových důvodů implementována, ale v textu bylo navrženo, jakými způsoby by je šlo implementovat v budoucnu.

10.2 Přínos práce

Vytvořením jednotného rozhraní pro ovladače zařízením a zavedením podpory pro automatickou detekci zařízení se operační systém HelenOS posunul o krůček dál směrem k moderním operačním systémům, které nabízejí svým uživatelům širokou podporu různých typů zařízení bez nutnosti větších zásahů do konfigurace systému ze strany uživatele. Podpora automatické detekce a hierarchického systému zařízení otevřela nové pole působnosti při vývoji systému HelenOS. Nyní je nově možné implementovat ovladače pro celou řadu zařízení, která tuto podporu vyžadují a která by bez této podpory nebylo možné jednoduše z operačního systému HelenOS zpřístupnit. Podpůrná knihovna, jež byla v rámci této práce implementována, v budoucnu usnadní vývoj nových ovladačů.

10.3 Možnosti budoucího rozšíření

Vzhledem k povaze tématu práce jsou možnosti budoucího rozšíření opravdu široké.

V nejbližších fázích vývoje je potřeba v uživatelském prostoru implementovat ovladače pro řadiče přerušení a pro další důležitá zařízení. S tím souvisí i podpora dalších architektur, na které byl operační systém HelenOS portován. Samotný driver framework je sice napsán dostatečně obecně, ale dosud nebyly implementovány žádné ovladače, které by jej využívaly na jiných architekturách než na IA-32.

Dalším krokem by mohlo být vylepšení integrace nových částí driver frameworku s device mapperem a devfs. K tomu bude mimo jiné potřeba rozšířit device mapper a devfs o podporu více než dvou úrovní hierarchie zařízení.

Návrh také počítá s tím, že se v budoucnu bude společně s přidáváním podpory pro nové typy zařízení intenzivně rozšiřovat množina rozhraní pro přístup k zařízením.

Jakmile se vyskytne příležitost napsat ovladač některé ze sběrnic s podporou hotplug funkcionality, bude potřeba odpovídajícím způsobem rozšířit driver framework. Návrh s tímto rozšířením počítá.

Ve vzdálenější budoucnosti lze očekávat rozšíření i o další funkce – např. správu napájení.

Kapitola 11

Literatura

[helenos] *HelenOS design documentation*

[helenos_ipc] *IPC for Dummies*

[iokit] *I/O Kit Fundamentals*, Apple Inc., 2007.

[isa] Mark Sokos, *The ISA and PC/104 Bus*

[lin] Lukáš Jelínek, *Jádro systému Linux*, Computer Press, a.s., 2008.

[lindd] Jonathan Corbet, Alessandro Rubini a Greg Kroah-Hartman, *Linux Device Drivers*, O'Reilly Media, Inc., 2005, Third Edition.

[osdev] *OS Development Forum and Wiki*

[pci] Don Anderson a Tom Shanley, *PCI System Architecture*, Mindshare, Inc., 1999, Fourth Edition.

[pcidatabase] *PCI Vendor and Device Lists*

[piix3] *82371FB (PIIX) and 82371SB (PIIX3) PCI ISA IDE Xcelerator*, Intel Corporation, 1997.

[psycho] *UPA to PCI Interface User's Manual*, A Sun Microsystems, Inc., 1997.

[root] Pavel Tišnovský, *Seriál Co se děje v počítači*, www.root.cz, 2008-2009.

[serial] Christian Blum, *The Serial Port*, 1995, Release 19.

[solaris] *Writing Device Drivers*, Sun Microsystems, Inc., 2008.

[win2k] Art Baker a Jerry Lozano, *The Windows 2000 Device Driver Book*, Prentice Hall PTR, 2001, Second Edition.