VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

OVLADAČ PROTOKOLU MIWI PRO LINUX

MIWI PROTOCOL DRIVER FOR LINUX

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — Richard Wolfert

AUTHOR

VEDOUCÍ PRÁCE — Ing. Tomáš Novotný

SUPERVISOR

BRNO 2015

Abstrakt

Tato bakalářská práce se zabývá tvorbou ovladače zařízení pro Linux. Výsledný ovladač dokáže obsluhovat PAN koordinátor protokolu MiWi. Komunikace se zařízením je uživateli zpřístupněná formou socketového rozhraní. Teoretická část popisuje princip činnosti a tvorbu ovladačů zařízení pro jádro systému Linux. Mezi hlavní částí práce patří návrh, implementace, výsledné testování a zhodnocení výsledků.

Abstract

This bachelor's thesis describes creating of device driver for Linux. The result driver is able to handle PAN coordinator device for MiWi protocol. User is able to communicate with device using socket network interface. The theoretical part provides fundamentals of operations and creation of new device drivers for Linux kernel. The main parts of this work consist of design, implementation, final testing and evaluation of results.

Klíčová slova

Linux, ovladač zařízení, SPI, prostor jádra, prostor uživatele, MiWi, IoT, protokolový ovládač, síťové rozhraní

Keywords

Linux, device driver, SPI, kernel space, user space, MiWi, IoT, protocol driver, network interface

Citace

Richard Wolfert: Ovladač protokolu MiWi pro Linux, bakalářská práce, Brno, FIT VUT v Brně, 2015

Ovládač protokolu MiWi pre Linux

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Tomáše Novotného. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Richard Wolfert
18. 5. 2015

Poděkování

Rád by som sa poďakoval vedúcemu tejto bakalárskej práce Ing. Tomášovi Novotnému za poskytnuté odborné konzultácie, pomoc a pripomienky pri písaní tejto práce.

Obsah

1 Úvod

V dobe prudkého rozvoja oblasti informačných technológií vzniká čo raz väčší požiadavok na integráciu výpočtovej techniky do oblastí každodenného života. Čo raz viac sú informačné technológie vnímané ako prostriedok kontroly a správy takmer v akomkoľvek odvetví. Jednou z týchto oblastí je aj správa bežnej domácnosti. Týmto konkrétnym požiadavkom sa venuje projekt s názvom IoT pôsobiaci na tejto fakulte. Táto práca rieši problém v rámci tohoto projektu. Ako nedeliteľná súčasť systému inteligentnej domácnosti je bezdrôtová senzorová sieť. K správe takejto siete sa používa zariadenie, ktoré súčasne tvorí komunikačné rozhranie so zvyškom systému. Toto zariadenie sa nazýva adaptér a je implementované formou vstavaného Linuxového systému. Táto práca sa venuje návrhu a tvorbe ovládača zariadenia určeného pre tento adaptér. Spomínaný ovládač má spravovať hardvérový modul, určený pre priamu komunikáciu so senzormi v bezdrôtovej sieti. Bezdrôtový modul je s adaptérom fyzicky spojený pomocou sériovej zbernice SPI. Súčasné riešenie využíva všeobecný ovládač pre komunikáciu pomocou SPI zbernice s názvom spidev.

Táto práca je rozdelená celkovo do siedmych kapitol, pri čom prvou kapitolou je tento úvod. Druhá kapitola je čisto teoretická, vysvetľuje základné znalosti o ovládačoch v jadre operačného systému Linux. Vysvetľuje tiež základné typy zariadení a poukazuje na ich rozdiely. Porovnáva riešenie problému v priestore jadra a v priestore používateľa. Súčasťou tejto kapitoly je tiež použitý hardvér, pre ktorý je daný ovládač cielený. V závere sa nachádza popis súčasného riešenia spidev. Tretiu kapitolu tvorí návrh na základe znalostí uvedených v druhej kapitole. Vysvetľuje koncepciu sieťového rozhrania a rozdelenie problému na protokolový ovládač a ovládač sieťovej karty. Okrem toho obsahuje predpokladaný postup pri tvorbe z vyšších softvérových vrstiev až po hardvér. Na záver popisuje formu výsledného modulu a spôsob prekladu pomocou krížového prekladu. Štvrtá kapitola sa venuje implementačným problémom a princípom činnosti protokolového ovládača a ovládača sieťovej karty. Ukazuje akým spôsobom jednotlivé vrstvy medzi sebou komunikujú a tiež akým spôsobom sú riešené jednotlivé vybrané problémy. Piata kapitola ukazuje použitie ovládača z hľadiska používateľa. Šiesta kapitola je venovaná testovaniu implementovaného ovládača a obsahuje štatistiky testov. Záverečná siedma kapitola reprezentuje zhrnutie dosiahnutých výsledkov, možné rozšírenia a plány do budúcnosti.

2 Ovládač zariadenia v Linuxe

Táto kapitola sa zaoberá spôsobom akým jadro operačného systému Linux pristupuje k samotnému hardvéru a perifériám. Bude vysvetlené čo vlastne ovládač je a k čomu slúži.

2.1 Priestor používateľa a priestor jadra

V prvom rade je potrebné spomenúť že Linux pracuje na koncepte rozdelenia priestoru jadra (*kernel space*) od priestoru používateľa (*user space*) [1]. Tento princíp funguje na základe rozdelenia virtuálnej pamäti procesu, ktorý je spustený v danom priestore. Procesy bežiace v režime jadra majú obvykle plný a priamy prístup k hardvéru stroja. Hlavnou úlohou procesov bežiacich v režime jadra je poskytnutie akého si rozhrania medzi priestorom používateľa a samotného hardvéru. Procesy bežiace v používateľskom priestore majú obmedzené možnosti prístupu do virtuálnej pamäti a nedokážu pracovať priamo s hardvérom. Ich jedinou možnosťou ako komunikovať s hardvérom stroja je využiť rozhranie, ktoré poskytujú procesy bežiace v režime jadra.

Jednou z hlavných výhod tohoto princípu je bezpečný beh aplikácií v režime používateľa z dôvodu obmedzených možností a priameho prístupu k hardvéru [5]. Jadrový kód vyžaduje stabilitu, pretože akákoľvek chyba vyskytujúca sa na tejto úrovni môže mať fatálne následky pre celý systém.

Obrázok 2-1: Ilustrácia komunikácie pomocou systémových knižníc.

2.2 Ovládač zariadenia

Ovládač zariadenia je softvérová vrstva, patriaca do priestoru jadra operačného systému, ktorej hlavnou úlohou je poskytnúť mechanizmy, pomocou ktorých je aplikačná vrstva schopná komunikácie s hardvérom. Ovládač zariadenia by mal skutočne poskytovať len rozhranie, nemal by teda riešiť čo je možné s hardvérom robiť, ale ako to má robiť.

2.3 Typy zariadení

Z pohľadu Linuxu sú zariadenia všeobecne rozdelené do troch tried: znakové zariadenia, blokové zariadenia a sieťové rozhrania [1].

2.3.1 Znakové zariadenia

Znakové zariadenia sú charakteristické tým, že je k nim možné pristupovať ako ku prúdu jednotlivých bytov. Ovládač implementujúci znakové zariadenie musí poskytovať minimálne 4 základné operácie (*read*, *write*, *open* a *close*). Tieto zariadenia zvyčajne nepodporujú možnosť vrátiť sa späť a znovu načítať už raz načítané dáta. Typickým príkladom znakových zariadení sú textové konzoly, sériové rozhrania, a tak podobne. Ovládač znakového zariadenia v Linuxe implementuje prístup k tomuto zariadeniu pomocou konkrétneho uzlu v adresári „/dev“, napríklad „/dev/tty0“.

2.3.2 Blokové zariadenia

Blokové zariadenia na rozdiel od znakových dokážu pracovať s najmenšou jednotkou o veľkosti niekoľko bytov, ktorá sa nazýva blok. Veľkosť jedného bloku je vo väčšine prípadov 512 bytov, v ostatných prípadoch sa však môže jednať o ľubovolnú hodnotu rovnú mocnine dvojky (1024 bytov, 2048 bytov, ...). V používateľskom režime Linux umožňuje pracovať s blokovým zariadením ako so znakovým, pri čom využíva mechanizmus vyrovnávacej pamäti (cache pamäť). Z toho vyplýva že rozdiel medzi znakovým a blokovým zariadením z používateľského hľadiska nepoznať. Rozdiel je vo vnútornej komunikácii jadra operačného systému s konkrétnym zariadením. Tak isto ako znakové zariadenie je blokové sprístupnené pomocou uzlu v adresári „/dev“. Typickým príkladom blokového zariadenia je akékoľvek diskové zariadenie.

2.3.3 Sieťové rozhrania

Sieťové rozhranie v Linuxe je úplne odlišný typ zariadenia od znakového/blokového. Jeho hlavnou úlohou je prijímať a odosielať dátové pakety. Táto činnosť podlieha a zároveň je riadená sieťovým subsystémom operačného systému. Aplikačný programátor teda nemá možnosť využívať služby sieťového ovládača priamo. Ovládač sieťového rozhrania nemá žiadne informácie o jednotlivých spojeniach, jeho úlohou je len spracovanie dátových paketov. Prístup k sieťovému rozhraniu nie je možné jednoducho namapovať do súborového systému tak ako to funguje pri blokových a znakových zariadeniach. Napriek tomu má každé sieťové rozhranie v Linuxe svoj jedinečný identifikátor, napríklad eth0, wlan0, ten však nie je prístupný zo súborového systému ako to býva zvykom. Hoci sa sieťové zariadenie väčšinou viaže na nejaký hardvér, existujú aj čisto softvérové zariadenia ako napríklad loopback sieťové rozhranie.

2.4 Komunikácia priestoru používateľa s priestorom jadra

Pre každý typ zariadení existuje skupina univerzálnych operácií, slúžiacich ku komunikácii medzi aplikačným procesom a jadrovým procesom riadiacim konkrétne zariadenie [14]. Často je však potrebné použiť nejakú špecifickú operáciu daného zariadenia, na ktorú všeobecné rozhranie nestačí. K riešeniu takéhoto problému ponúka Linux viacero spôsobov. Dvom z nich sa venuje táto podkapitola.

2.4.1 Volanie ioctl()

Prvý spôsob riešenia je jednoduchý. Pre riadenie a zisťovanie stavu daného ovládača sa použije funkcia v priestore používateľa. Prvým argumentom tejto funkcie je deskriptor súboru. Z toho vyplýva, že je tento spôsob veľmi vhodné použiť v prípade ovládača, ktorý sprístupňuje operácie formou súboru. Druhý argument je číselná hodnota, ktorá reprezentuje nejaký konkrétny príkaz pre daný ovládač. Posledným argumentom tejto funkcie je nastavovaná hodnota alebo ukazovateľ na premennú z ktorej sa dáta získavajú alebo kam sú zapisované.
Po vykonaní príkazu funkcia vracia nezápornú návratovú hodnotu nezmenenú. V prípade zápornej vracia hodnotu -1 a súčasne nastaví premennú errno na absolútnu hodnotu tejto zápornej hodnoty.

2.4.2 Netlink

Druhým a novším spôsobom je použitie Netlinku. Netlink vznikol ako alternatívne riešenie k funkcii `ioctl()`. Slúži ako prostriedok medziprocesovej komunikácie medzi procesmi jadra a používateľskými procesmi. Jeho hlavnou výhodou je využitie štandardného socketového rozhrania v priestore používateľa. Používa vlastnú rodinu adries označených konštantou AF_NETLINK, je datagramovo orientovaný a má preddefinovaných niekoľko protokolov, ku ktorým je samozrejme možné pridať ďalšie. Jeho použitie je potom v zásade rovnaké ako použitie internetového socketu.

2.5 Moduly

V rámci operačných systémov môžme pojem jadrový modul (*kernel module*) definovať ako akýsi objekt [7]. Tieto objekty typicky obsahujú binárny kód a je pomocou nich možné rozšíriť základnú funkcionalitu jadra. Moduly môžu byť nakonfigurované dvoma rôznymi spôsobmi v súvislosti so spôsobom vloženia do jadra. Prvou konfiguráciou sú vstavané moduly (*build-in kernel module*). Takto nakonfigurované moduly je možné do jadra vložiť pri štarte operačného systému.

Druhou možnosťou sú dynamické moduly jadra (*loadable kernel module*). Takto preložené moduly je možné vkladať do aktuálne bežiaceho jadra operačného systému priamo v prípade ich potreby. Neskôr ak už moduly nie sú potrebné je ich možné z jadra odobrať. Z dôvodu flexibility, použiteľnosti a ďalších výhod som sa rozhodol výslednú prácu implementovať týmto spôsobom.

2.6 Vývoj pod odlišnou architektúrou

Pri preklade jadrového modulu pre vstavaný linuxový systém je potrebné vyriešiť zásadný problém. Tento problém súvisí s tým, že cieľová architektúra stroju na ktorom prekladaný modul pobeží môže byť iná (napr. arm) od architektúry bežných pracovných strojov (napr. x86, x86_64). Všeobecne poznáme dve možné riešenia takéhoto problému. V závere podkapitoly sa nachádza obrázok 2-2 ilustrujúci obe popísané metódy.

2.6.1 Krížový preklad

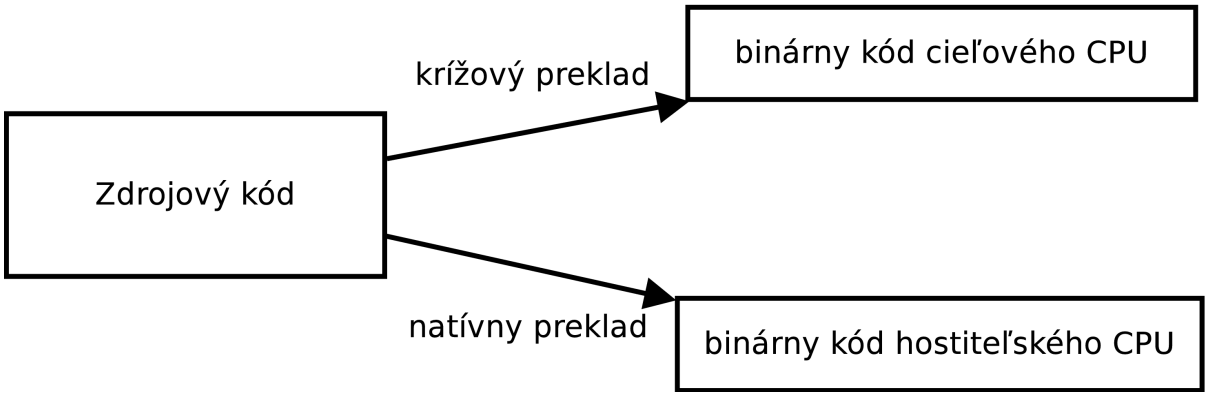
Krížový preklad (*Cross-compilation*) je metóda prekladu zdrojových súborov, ktorá sa používa hlavne pri vývoji vstavaných systémov. Typickým atribútom tejto metódy je, že cieľová architektúra prekladu je odlišná od tej na ktorej beží samotný prekladač. Vývoj softvéru v tomto prípade prebieha na úplne odlišnom stroji s inou architektúrou (napr. x86_64), než stroj na ktorom bude samotný program bežať (napr. arm). Aby táto metóda mohla fungovať, hostiteľský stroj musí obsahovať prekladač schopný transformovať zdrojové súbory na binárne súbory cieľovej architektúry. Takýto prekladač sa nazýva krížový prekladač. Takto preložený program pochopiteľne nie je možné spustiť na hostiteľskom stroji ihneď po preklade, ale binárny súbor sa musí presunúť na cieľový stroj kde ho je možné spustiť.

Táto metóda má niekoľko výhod. Jednou z nich je typicky mnohonásobne vyššia rýchlosť hostiteľských vývojových strojov oproti cieľovým strojom. Dôvodom tohoto je že väčšina vstavaných systémov sú navrhnuté na miniatúrnu veľkosť, spotrebu energie alebo cenu. To všetko sa robí na úkor rýchlosti, ktorá nie je až taká podstatná. Od tohoto faktu súvisí aj ďalšia výhoda ktorou je nepotrebnosť prítomnosti prakticky žiadnych vývojových nástrojov na cieľovom stroji.

2.6.2 Natívny preklad

Jednou z možností ako prekladať softvér na vstavaný linuxový systém je natívny preklad. Pri natívnom preklade sú zavedené všetky vývojové prostriedky potrebné k prekladu, vrátane prekladača na cieľový stroj. Takýto prekladač môžme označiť za natívny prekladač alebo len prekladač. Zdrojové texty sa preložia na binárne súbory spustiteľné pre hostiteľskú architektúru prekladača. V tomto prípade je cieľová architektúra rovnaká ako tá hostiteľská.

Tento spôsob prekladu má prirodzene zmysel ak je architektúra na ktorej sa daný software vyvíja rovnaká ako cieľová architektúra prekladu. Ak sa architektúry líšia a súčasne na cieľovom stroji neprekáža prítomnosť vývojových nástrojov, tiež možno uvažovať nad touto metódou.



Obrázok 2-2: Ilustrácia rozdielu rôznych metód prekladu.

2.7 Preklad jadrového modulu

V nasledujúcej podkapitole sa budem venovať prekladu všeobecného modulu a jeho vloženiu do jadra. Tak tiež to budem demonštrovať na jednoduchom príklade.

2.7.1 Linkovanie s jadrom

Preklad jadrového objektu sa významne líši od prekladu aplikácie bežiacej v užívateľskom priestore [1]. Je dobré si uvedomiť že modul má byť súčasťou jadra. To znamená že pri preklade musí linkér vytvoriť väzby medzi prekladaným modulom a samotným jadrom. Z tohoto dôvodu musí mať prekladač k dispozícii prístup k preloženému jadru do ktorého sa bude daný modul vkladať. Je pri tom podstatne dôležité aby verzia jadra ktorú má k dispozícii prekladač pri preklade modulu bola rovnaká ako verzia jadra na bežiacom cieľovom stroji. Jadro pri tom musí byť preložené s rovnakým konfiguračným súborom a súčasne je vhodné použiť rovnakú verziu prekladacích nástrojov.

```c
#include <linux/init.h>
#include <linux/module.h>

static int hello_init(void)
{
    printk(KERN_ALERT "Hello world\n");
    return 0;
}

static void hello_exit(void)
{
    printk(KERN_ALERT "Goodbye world\n");
}
module_init(hello_init);
module_exit(hello_exit);
```

Ukážka 2-1: Jednoduchý jadrový modul.

Ukážkový modul v podstate nedokáže nič viac než sa v jadre zaregistrovať a neskôr ho opustiť. Je však postačujúci na otestovanie správnosti prekladu funkčnosti vkladania.

O samotný preklad modulu sa už postará prekladový systém konkrétneho jadra. Pre vloženie takto preloženého jadrového objektu do bežiaceho jadra je možné použiť štandardný linuxový nástroj modprobe. Podľa manuálových stránok sa modprobe stará o pridávanie a odoberanie modulov z linuxového jadra.

2.8 Popis hardvéru pre jadro Linuxu

Jadro operačného systému Linux je multiplatformové a podporuje veľké množstvo zariadení. Pre tento účel musí poskytovať mechanizmy, pomocou ktorých je možné špecifikovať konkrétny hardvér na ktorom má jadro bežať. Pre niektoré architektúry (napr. x86) existuje možnosť automatickej detekcie zariadení pripojených na zbernici. Pre iné architektúry (napr. arm) však táto možnosť nie je a zariadenia je nutné priamo špecifikovať. Táto podkapitola bude venovaná možnostiam pre architektúru arm.

2.8.1 ATAGS

V minulosti sa používala dnes už zastaralá metóda, ktorá využíva takzvané ATAGS. V tomto prípade sa popisy všetkých podporovaných hardvérov nachádzajú priamo v kóde jadra. Hlavnú úlohu tu zohráva *bootloader*, ktorý pri štarte jadru oznámi o aký konkrétny typ dosky sa jedná. Okrem toho pripraví dodatočné informácie ako napríklad veľkosť a umiestnenie pamäti. Tieto informácie sa nazývajú ATAGS [16].

2.8.2 Device Tree

Dnešná moderná metóda využíva k popisu hardvéru štruktúru, ktorá sa nazýva *Device Tree*. Zo samotného názvu jasne vyplýva, že sa jedná o stromovú štruktúru. Strom je zložený z pomenovaných uzlov, pričom každý uzol môže obsahovať nejaké vlastnosti alebo ďalšie synovské uzly.

Pre každé podporované zariadenie architektúry arm je možné nájsť daný textový súbor, obsahujúci *Device Tree*, nachádzajúci sa v adresári „arch/arm/boot/dts“ v zdrojových textoch jadra. Tieto súbory majú príponu „.dts“ a pred použitím ich je potrebné skompilovať do binárnej podoby. Tieto skompilované binárne súbory sa nazývajú *Device Tree Blob* súbory a majú typicky príponu „.dtb“. Táto metóda má množstvo výhod, pri čom väčšina vychádza z faktu, že popis hardvéru nie je súčasťou jadra, ale nachádza sa v samostatnom súbore [15][16].

2.9 Prerušenie a odložené spracovanie

Mechanizmus prerušenia (*interrupt*) je v moderných systémoch spôsob, ktorým zariadenie asynchrónne oznamuje procesoru vznik nejakej udalosti, na ktorú je typicky potrebné reagovať [1]. Z pohľadu jadra operačného systému, ovládač daného zariadenia ktoré produkuje prerušenia poskytuje obslužné rutiny, ktoré prebehnú v prípade vzniku tejto udalosti. Spracovanie prerušenia má v systéme najvyššiu prioritu.

Počas obsluhy prerušenia systém nemôže reagovať na bežné udalosti. Z tohoto dôvodu je nutné aby trvalo čo najkratšie a akékoľvek vstupno/výstupné operácie sú tu neprípustné. Reakcia na prerušenie však často vyžaduje vstupno/výstupnú alebo inak časovo náročnú operáciu. Tento problém rieši metóda takzvaného odloženého spracovania a jej formy popísané v tejto podkapitole.

2.9.1 Tasklety

Tasklet je forma metódy odloženého spracovania [18]. Jeho činnosť spočíva v tom, že obsluha prerušenia naplánuje činnosť, ktorá sa musí vykonať ako reakcia na danú udalosť. Beh taskletu má vyššiu priority ako používateľské procesy, no nižšiu než obsluha prerušenia. Počas jeho behu je možné obsluhovať prichádzajúce prerušenia. Tasklety sú spracovávané v poradí v akom prichádzajú a môžu byť naplánované s nízkou alebo vysokou prioritou. Táto forma je vhodná pre menej časovo náročné a neblokujúce operácie.

2.9.2 Pracovné fronty (*Workqueues*)

Pracovné fronty oproti taskletom bežia vo zvláštnom vlákne a ich priorita behu je rovnaká ako používateľské procesy. Hodia sa predovšetkým pre časovo náročnejšie vstupno/výstupné, či

blokujúce operácie. Počas čakania na vykonanie takejto operácie je možné prepnúť kontext procesoru a riešiť iné udalosti. Jadro Linuxu poskytuje jednu spoločnú pracovnú frontu, ktorú je možné využiť. K plánovaniu extrémne časovo náročných operácií sa však odporúča použiť vlastnú pracovnú frontu, aby nebolo obmedzené spracovanie úloh ostatných modulov jadra [17].

2.10 Hardvér

V nasledujúcej podkapitole budú popísané a vysvetlené hardvérové prvky, ktoré sa momentálne používajú vo vyvíjanom systéme a ako celok tvoria adaptér. Tento celok sa dá rozdeliť na dve logické časti.

2.10.1 Adaptér

Základom implementácie koncepcie adaptéru je použitý mikropočítač A10-OLinuXino-LIME [9]. Tento produkt vyrába a vyvíja spoločnosť Olimex. Je postavená na vstavanom linuxovom systéme s architektúrou arm. Konkrétne je doska osadená procesorom Allwinner A10 s jadrom Cortex-A8 bežiaci na maximálnej frekvencii 1 GHz. Ďalej poskytuje operačnú pamäť typu DDR3 o veľkosti 512 MB. K uloženiu externých dát vrátane operačného systému doska poskytuje dve rôzne riešenia:

1. NAND Flash:

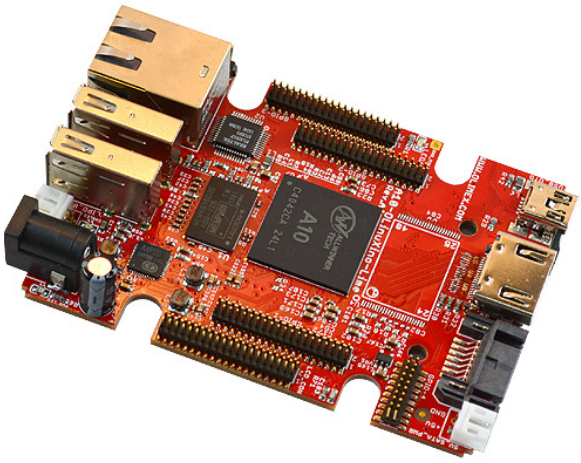
V tomto prípade je pamäťový modul integrovaný na pevno priamo na doske a nie je možné ho jednoducho odobrať, či pridať. Nevolatílna flash pamäť poskytuje dostatočnú veľkosť a vysoké rýchlosti zápisu. Oproti druhej možnosti je toto riešenie cenovo vyššie položené.

2. MicroSD pamäťová karta:

Druhou alternatívou oproti vstavanej flash pamäti je využitie MicroSD pamäťovej karty. Doska je osadená čítačkou tohoto typu pamäťovej karty a plne ju podporuje. Nespornou výhodou tohoto riešenia je možnosť kartu jednoducho odobrať a pridať. Moderné pamäťové karty poskytujú viac než dostačujúce veľkosti. Toto riešenie poskytuje pomalšie prenosové rýchlosti oproti vstavanému NAND modulu, ktoré sú však postačujúce pre daný účel.

Okrem iného musím spomenúť, že sa na doske nachádza integrované sériové rozhranie SPI, ktoré bude slúžiť na komunikáciu s bezdrôtovým modulom. Toto rozhranie je vyvedené na jedno z GPIO (*General-purpose input/output*) fyzických rozhraní.

Ku koncu roku 2014 je cena tohoto zariadenia podľa oficiálneho obchodu spoločnosti Olimex 30,00 EUR za verziu bez NAND pamäti a 40,00 EUR za verziu s integrovanou 4 GB NAND pamäťou.



Obrázok 2-3: A10-OLinuXino-LIME [9].

2.10.2 PAN koordinátor

Hlavnou úlohou adaptéru je spravovať a komunikovať s bezdrôtovou senzorovou sieťou. Adaptér sám o sebe potrebuje rozširujúci hardvérový modul, slúžiaci na komunikáciu v takejto sieti. Projekt Inteligentná domácnosť rieši tento problém s použitím protokolu MiWi. Väčšina informácií uvedených v tejto podkapitole pochádza z oficiálnych web stránok spoločnosti Microchip Technology a technických dokumentácií k popisovaným produktom [8].

2.10.2.1 Protokol MiWi

Protokol MiWi je proprietárny bezdrôtový protokol, ktorý je vyvíjaný firmou Microchip Technology [8]. Jeho princíp je založený na štandarde IEEE 802.15.4, ktorý je určený na definovanie vlastností bezdrôtových osobných sietí (*wireless personal area network*, WPAN). Jedná sa teda o protokol určený pre komunikáciu, ktorá sa vyznačuje krátkymi prenosovými vzdialenosťami, nízkou rýchlosťou prenosu a nízkou spotrebou energie. Protokol je vyvinutý pre prácu na frekvenciách 2,4 GHz a SubGHz (menej než 1 GHz). Medzi hlavné výhody tohoto protokolu je podľa oficiálnych webových stránok firmy Microchip jednoduchosť vývoja a konfigurácie bezdrôtovej siete implementujúcej tento protokol a tým pádom ušetrený čas používateľov tohoto riešenia.

Štandard IEEE 802.15.4 rozlišuje 2 typy zariadení, podľa ich funkcie v sieti:

Typ zariadenia	**Ponúkané služby**	**Typický zdroj napájania**	**Typická konfigurácia prijímača**
Zariadenie s plnou funkcionalitou (FDD)	Všetky	Vlastný zdroj	Zapnutý
Zariadenie s limitovanou funkcionalitou (RFD)	Obmedzené	Batéria	Vypnutý

Tabuľka 2-1: Rozdelenie typov zariadení WPAN podľa IEEE 802.15.4 [8].

Sieť implementujúca protokol MiWi sa môže skladať z 3 typov zariadení, ktoré sú popísané v tabuľke nižšie:

Typ zariadenia	**Typ zariadenia podľa IEEE 802.15.4**	**Typická funkcia**
PAN Koordinátor	FFD	Len jedno zariadenie tohoto typu v sieti. Spravuje sieť, prideľuje sieťové adresy.
Koordinátor	FFD	Zariadenie tohoto typu umožňuje rozšíriť fyzický rozsah siete, umožňuje pripojenie nových uzlov. Môže vykonávať aj funkcie koncového zariadenia. Jeho použitie je voliteľné.
Koncové zariadenie	FFD, RFD	Toto zariadenie vykonáva danú činnosť, ku ktorej bola sieť implementovaná. V tomto prípade sa jedná o senzory alebo aktory.

Tabuľka 2-2: Typy zariadenia podľa protokolu MiWi [8].

Zariadenie ktorému sa zaoberá táto práca je typu PAN Koordinátor. Toto zariadenie tvorí kľúčový prvok v sieti. Vytvára sieť a všetky ostatné prvky nachádzajúce sa v sieti musia poslúchať príkazy tohoto zariadenia.

2.10.2.2 Ponuka hardvéru

Keďže protokol MiWi je licencovaný spoločnosťou Microchip Technology, je nutné použiť hardvér len od danej firmy. Táto spoločnosť poskytuje niekoľko integrovaných obvodov, ktoré dokážu implementovať protokol MiWi. Tieto obvody sú rozdelené do dvoch kategórií v závislosti od ich pracovnej frekvencie.

Prvou skupinou sú obvody pracujúce okolo frekvencie 2,4 GHz, ktoré sú popísané v tabuľke:

Integrovaný obvod	Šírka dátového prenosu	Frekvenčný rozsah (MHz)	Senzitivita (dBm)	Spotreba Tx	Spotreba Rx	výstupný výkon (dBm)
MRF24J40	250 kbps	2,405-2,48	-94	23 mA	19 mA	+0
MRF24XA	125 kbps – 2 Mbps	2,405-2,48	-103	25 mA	13,5 mA	+0

Tabuľka 2-3: Ponuka 2.4 GHz integrovaných obvodov od spoločnosti Microchip [8].

Druhou skupinou sú obvody pracujúce na frekvencii SubGHz, teda frekvencii nižšej ako 1 GHz:

Integrovaný obvod	Modulácia	Šírka dátového prenosu	Frekvenčný rozsah (MHz)	Citlivosť (dBm)	Tx Power (dBm)
MRF89XA	FSK/OOK	200 kbps	868/915/955	-113	+12,5
MRF49XA	FSK	256 kbps	434/868/915	-110	+7

Tabuľka 2-4: Ponuka SubGHz integrovaných obvodov od spoločnosti Microchip [8].

2.10.2.3 MMRF89XAM8A

Keďže medzi hlavné výhody použitia protokolu MiWi patrí jednoduché zostavenie a konfigurácia takejto siete, sú tieto integrované obvody ponúkané ako súčasť samostatných, rozširujúcich hardvérových modulov. Jedným z takýchto produktov je aj doska ktorá nesie označenie „MMRF89XAM8A“. Táto hardwarová komponenta je použitá v projekte Inteligentná domácnosť, a preto som sa jej rozhodol venovať túto podkapitolu.

Ako už bolo vyššie spomenuté, za týmto označením sa skrýva hardvérová komponenta, ktorá je postavená na integrovanom obvode MRF89XA určenú pre SubGHz komunikáciu plne podporujúcu MiWi protokol. Má v sebe zabudovanú PCB (*printed circuit board*) anténu a integrované SPI rozhranie určené na pripojenie mikrokontroléru, ktorý riadi jeho činnosť. Princíp činnosti SPI rozhrania je popísaná v nasledujúcej podkapitole.

Názov vlastnosti	Hodnota
Typ	SubGHz Modul
Výstupný výkon (dBm)	10,0
Rozhranie	4-drôtové SPI
Počet pinov	12
RF Modul	Áno
RF Transciever	Áno
Frekvenčný rozsah	868
Citlivosť vstupu (mVpp)	-113,00
RSSI	Áno
Tx Príkon (mA)	25,00
Rx Príkon (mA)	3,00
Spánok	Áno
MAC	Áno

Tabuľka 2-5: Popis vlastností modulu MMRF89XAM8A [8].

Ako samotný výrobca tejto súčiastky tvrdí, jej použitie je predovšetkým určené pre budovanie domácej bezdrôtovej senzorovej siete s nízkou spotrebou.

2.10.3 SPI

SPI (*Serial Peripheral Interface*) je synchrónne sériové rozhranie pôvodne určené na komunikáciu mikrokontroléru s periférnymi zariadeniami. Je ho však možné využiť ku komunikácii medzi mikrokontrolérmi [10]. Je určené na plne duplexnú komunikáciu, takže v jednom časovom okamžiku je možné prijímať a súčasne odosielať dáta. Umožňuje „point-to-point“ spojenie dvoch zariadení. Okrem toho dokáže vytvoriť SPI zbernicu, do ktorej je možné pripojiť viacero zariadení. V jednom okamžiku je však možná komunikácie len medzi dvoma zariadeniami.

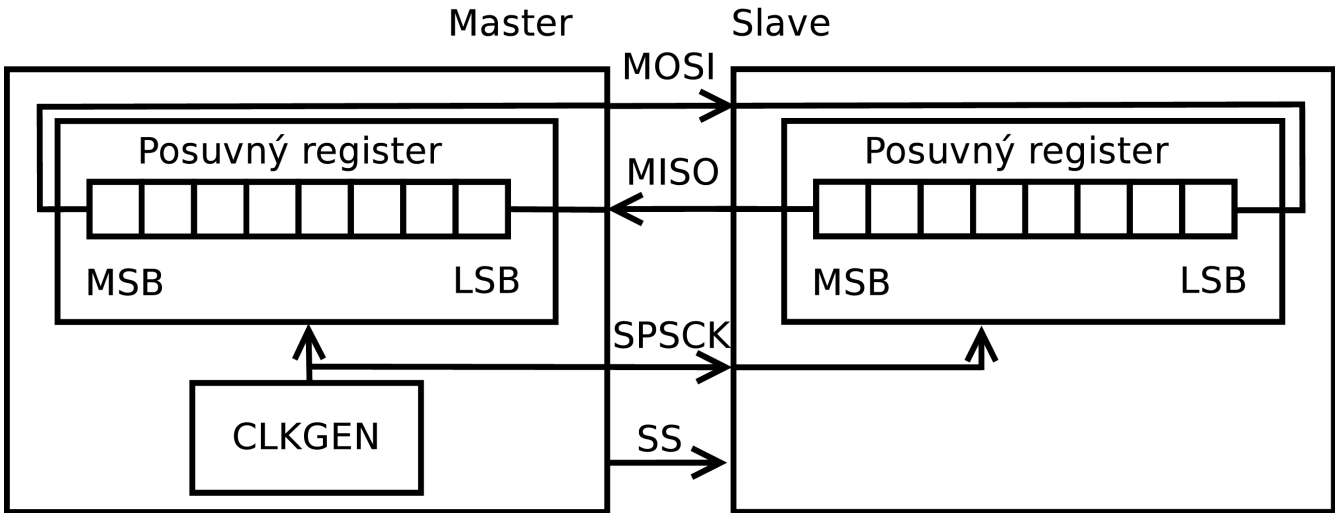
Toto rozhranie funguje na princípe Master/Slave, čo znamená že zariadenia pripojené k takejto zbernici môžu byť dvojakého druhu:

1. Master – Na SPI zbernici sa nachádza vždy práve jedno zariadenie typu master. Jeho úlohou je generovať synchronizačný hodinový signál a riadiť komunikáciu na zbernici. V typickom prípade je zariadenie tohoto typu práve mikrokontrolér.
2. Slave – Všetky ostatné zariadenia na zbernici sú typu slave a svoju činnosť riadia na základe pokynov od zariadenia typu master. Zariadenie tohoto typu býva v typickom prípade nejaké periférne zariadenie.

Fyzické rozhranie je tvorené dvojicou dátových vodičov (MISO, MOSI), jedným synchronizačným vodičom (SPSCK) a jedným riadiacim vodičom (SS). Princíp činnosti je znázornený na obrázku, ktorý je doplnený o tabuľku vysvetľujúcu jednotlivé vodiče rozhrania.

MOSI	Master Output, Slave Input. Pre zariadenie typu master tento vodič slúži ako výstup pre zariadenie typu slave, ktoré naopak dáta prijíma.
MISO	Master Input, Slave Output. Tento vodič má opačný význam ako MOSI. Zariadenie typu slave sem zapisuje dáta, ktoré prijíma zariadenie typu master.
SPSCK	Synchronizačný hodinový signál, ktorý je generovaný v zariadení typu master.
SS	Slave Select. Tento vodič je riadiaci a určuje s ktorým zariadením na zbernici chce master komunikovať.

Tabuľka 2-6: Popis vodičov rozhrania SPI.



Obrázok 2-4: Ukážka činnosti SPI rozhrania [10].

Ako je na obrázku vidno, komunikácia prebieha pomocou posuvných registrov. Typickou vlastnosťou komunikácie pomocou SPI je výmena dát v týchto registroch medzi komunikujúcimi zariadeniami počas prenosu jedného bytu. Komunikácia je teda vždy obojsmerná. Jednosmernú komunikáciu je možné realizovať tým, že sa na druhú stranu prenesie, respektíve príme bezvýznamná hodnota.

2.11 Existujúce riešenie (spidev)

Súčasné riešenie využíva modul určený na komunikáciu pomocou SPI zbernice ktorý sa nazýva spidev. Jedná sa o modul implementujúci rozhranie k znakovému zariadeniu. Podľa oficiálnej dokumentácie ponúka spidev limitované rozhranie a základné operácie v polovičnom duplexe. Použitím ioctl príkazov je možná konfigurácia zbernice [11]. Keďže spidev implementuje ovládač k znakovému zariadeniu, prístup k nemu je pomocou uzlu v súborovom systéme Linuxu. Tento uzol má charakter výlučného prístupu. To znamená, že k nemu môže v jednej chvíli pristupovať najviac jeden proces. Ku komunikácii v plnom duplexe musí používateľ poskytnúť dvojicu pamäťových miest označených ako „tx_buf“ a „rx_buf“.

3 Návrh

Pred tým než začnem popisovať a rozoberať návrh daného ovládača je potrebné zodpovedať základnú otázku: „Akým spôsobom môže navrhovaný ovládač slúžiť lepšie ako súčasne používané riešenie (spidev)?“. Odpovedí na túto otázku je viacero. Všetky však majú spoločnú jednu vec: Vytvorenie komplexného a flexibilného rozhrania k danému zariadeniu a celú komunikáciu a konfiguráciu zabaliť do priestoru jadra, čím sa zjednoduší práca aplikačného programátora. Okrem toho by bolo vhodné implementovať možnosť viacnásobného prístupu z hľadiska používateľa. Z hľadiska výkonnosti je spracovanie prerušenia v jadre lepšia a rýchlejšia varianta, než je súčasné spracovanie v priestore používateľa. V tejto kapitole sa budem venovať návrhu ovládača a popíšem vlastnosti, ktoré by mal obsahovať.

3.1 Typ ovládača

Ako už bolo spomenuté v druhej kapitole tejto práce Linux rozdeľuje zariadenia do troch tried, pri čom pre každú triedu sa implementuje typický ovládač. Najdôležitejšou časťou návrhu je teda rozhodnúť do ktorej triedy naše zariadenie zaradíme.

Fyzické SPI rozhranie je schopné odoslať, respektíve prijať samostatný byte, pracuje teda prúdovo a nie blokovo. Pozerať sa na zariadenie ako na blokové tu nedáva zmysel. Z tohoto dôvodu môžme bezpečne vylúčiť blokový charakter zariadenia a viac nad ním neuvažovať.

Po vylúčení blokového zariadenia je potrebné sa rozhodnúť medzi znakovým zariadením a sieťovým rozhraním. Obe varianty v tomto prípade dávajú zmysel. Znakové zariadenie by malo opäť charakter výlučného prístupu pre prístupový uzol v súborovom systéme a nebolo by možné abstrahovať rámcovú komunikáciu aplikačného protokolu.

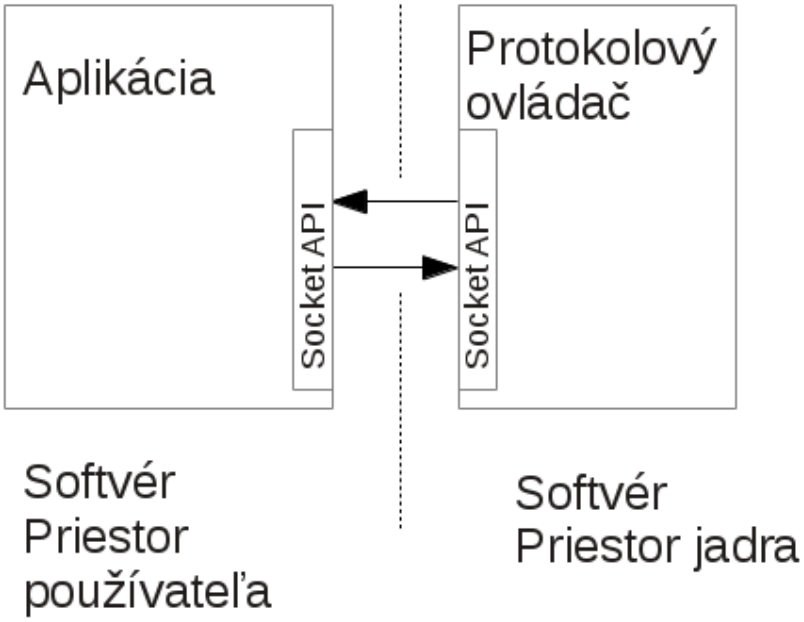
Implementovať zariadenie ako sieťovú kartu by naopak malo prirodzený charakter viacnásobného prístupu. Komunikácia by bola riadená sieťovým subsystémom Linuxu. Dátové rámce by mohli byť zapuzdrené v paketoch vlastného protokolu. Tento uhol pohľadu má svoje nesporné výhody, ktoré by aplikačnému programátorovi poskytli jednoduché rozhranie k danému zariadeniu. Napriek tomu, že sa jedná o implementačne náročnejší spôsob rozhodol som sa ho použiť a PAN koordinátor protokolu MiWi implementovať ako sieťové rozhranie.

3.2 Sieťové rozhranie

Na úvod musím spomenúť že návrh tohoto ovládača vychádza z projektu SocketCAN [13]. Jedná sa o jadrové moduly implementujúce komunikáciu prostredníctvom zbernice CAN (*controller area network*) vyvinuté skupinou *Volkswagen Research*. Pre využitie takejto metódy je nutné implementovať vlastnú rodinu protokolu ku komunikácii s MiWi PAN koordinátorom. S tým je spojená aj vlastná štruktúra použitých socketov. Oproti znakovému zariadeniu môžeme problém rozdeliť na dve samostatné časti.

3.2.1 Protokolový ovládač

Keďže sa nejedná o komunikáciu klasického internetového protokolu, je potrebné implementovať vlastný protokolový ovládač. Takýto ovládač je čisto softvérová vrstva v jadre Linuxu, ktorá je súčasťou sieťového subsystému. Čisto softvérová vrstva v tom zmysle, že so samotným hardvérom priamo ani nekomunikuje. Na nasledujúcom obrázku 3-1 je znázornené rozhranie medzi aplikáciou a protokolovým ovládačom. Ako vidno táto komunikácia prebieha s využitím socket API.



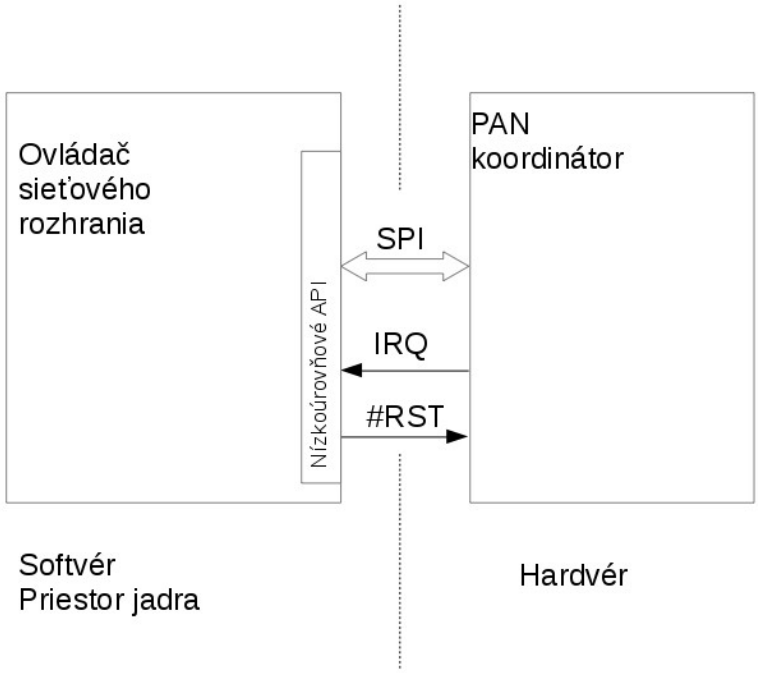
Obrázok 3-1: Komunikácia aplikácie a protokolového ovládača.

3.2.2 Ovládač sieťového rozhrania

Činnosť protokolového ovládača je spracovanie a smerovanie požiadavkov na nejaké rozhranie. V tomto prípade pôjde o ovládač nášho sieťového rozhrania, ktorý bude okrem iného implementovať komunikáciu s hardvérom pomocou nízko úrovňových knižníc. Na obrázku 3-2 je zachytené

rozhranie medzi ovládačom sieťového rozhrania a samotným hardvérom. Ovládač pre komunikáciu s hardvérom využije nízkoúrovňové API, pomocou ktorého v prvom rade získa prístup k SPI zbernici a v druhom rade prístup k GPIO vodičom, na ktorých sa nachádzajú signály RST (reset) a IRQ (interrupt request).

Keďže tento modul bude mať ako jediný prístup k samotnému hardvéru, jeho úlohou bude okrem iného poskytnutie rozhrania ku konfigurácii zariadenia. Pre tento konkrétny účel sa použijú ioctl príkazy.



Obrázok 3-2: Komunikácia ovládača zariadenia s hardvérom.

3.3 Spracovanie signálu IRQ a RST

Ku komunikácii so zariadením okrem SPI zbernice patrí aj dvojica signálov IRQ a RST. Pri súčasnom riešení sa tieto signály riešia samostatne v priestore používateľa. Vzniká teda požiadavka tento problém riešiť v priestore jadra a pokiaľ možno zapúzdriť to do ovládača samotného zariadenia.

Signál IRQ je z pohľadu adaptéru vstupný signál oznamujúci požiadavku na prečítanie vstupných dát. Signál RST je naproti tomu signálom výstupným. Je prítomný z dôvodu výskytu nekonzistentného stavu PAN koordinátora. Pomocou neho je možné vykonať reset hardvérového modulu.

3.4 Dekompozícia problému a implementačný postup

Keďže podstata návrhu rozdeľuje problém na dva nezávislé celky, je dekompozícia problému očividná. Pri implementácii budem postupovať od najvyšších softvérových vrstiev, teda protokolovému ovládača až k samotnému hardvéru, v tomto prípade ovládača sieťového rozhrania.

3.5 Preklad a spôsob implementácie

Ďalšia vec, ktorú treba upresniť je spôsob implementácie ovládača. Na základe kapitoly 2, kde sú popísané moduly je jasným favoritom ovládač vo forme dynamického jadrového modulu. V tomto prípade sa bude vlastne jednať o dva samostatné moduly. Toto riešenie je najlepšie hlavne z pohľadu jednoduchosti a flexibilnosti vývoja.

V súčasnom projekte je použitý operačný systém s linuxovým jadrom vo verzii 3.17.2. Vývoj celého tohoto systému je založený na projekte *OpenEmbedded*, ktorý poskytuje nástroje na krížový preklad softvéru [12]. Tento fakt rieši ďalší problém, teda preklad zdrojových textov do binárnej podoby. Spôsoby prekladu boli vysvetlené v kapitole 2. Najlepšou možnosťou prekladu bude použitie krížových prekladacích nástrojov, ktoré boli vygenerované pomocou prekladového systému *OpenEmbedded*.

4 Implementácia

Medzi kľúčovú časť tejto práce nepochybne patrí implementácia ovládačov, ktorých návrh bol popísaný v predchádzajúcej kapitole. Táto kapitola sa bude venovať rôznym implementačným detailom a problémom, ktoré vznikli počas vývoja a bolo potrebné nájsť adekvátne riešenie.

4.1 Protokolový ovládač

Jadro operačného systému Linux implementuje subsystém určený na sieťovú komunikáciu. Tento subsystém je navrhnutý a vytvorený s dôrazom na transparentnosť, flexibilnosť a hlavne protokolovú nezávislosť.

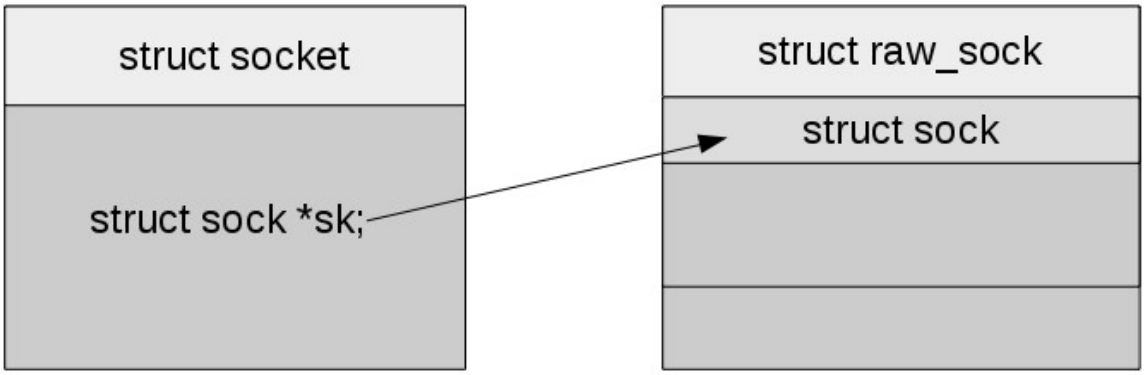
Protokolový ovládač je softvérová vrstva nachádzajúca sa v priestore jadra operačného systému. Vo svojej podstate implementuje transportnú vrstvu sieťovej komunikácie. Z hľadiska operačného systému tvorí transparentné rozhranie medzi aplikačným procesom a ovládačom sieťového zariadenia. Z tohoto faktu vyplývajú jeho dve hlavné úlohy bližšie vysvetlené v nasledujúcich podkapitolách.

4.1.1 Komunikácia s aplikáciou

Prvou úlohou protokolového ovládača je vytvorenie rozhrania a komunikácia s priestorom používateľa. Linux pre tento účel využíva princíp schránok (*socketov*). Schránka je koncový uzol komunikácie v počítačovej sieti. Jej použitie vytvára transparentné rozhranie komunikácie pre priestor používateľa. Na základe použitej protokolovej rodiny a konkrétneho protokolu existuje viacero typov schránok, pri čom sa najčastejšie používajú internetové schránky implementujúce internetový protokol. Podstatou protokolového ovládača je vytvoriť vlastnú protokolovú rodinu. Za týmto účelom je potrebné implementovať protokolové a schránkové operácie. Existuje obrovské množstvo univerzálnych operácií, ktoré je možné implementovať, nie všetky sú však potrebné. V nasledujúcich podkapitolách budú vysvetlené základné operácie implementované v tejto práci.

4.1.1.1 Reprezentácia schránky v jadre

Keďže protokolový ovládač je založený na schránkových operáciách je potrebné si najskôr ukázať akým spôsobom sa na schránku pozerá jadro operačného systému Linux.



Obrázok 4-1: Reprezentácia schránky.

Na obrázku 4-1 sú znázornené najdôležitejšie štruktúry a ich vzťahy. Štruktúra `struct socket` predstavuje všeobecnú BSD schránku. Priamo obsahuje najzákladnejšie informácie ako je napríklad stav, typ, príznaky alebo množina operácií, ktoré je možné nad danou schránkou vykonávať. Okrem toho ešte obsahuje odkaz na štruktúru typu `struct sock` ako je možné vidieť aj na obrázku. Tá obsahuje veľké množstvo konkrétnejších, dodatočných informácií. Napriek tomu je potrebné do reprezentácie schránky zapúzdriť ďalšie, dodatočné informácie. Z tohoto dôvodu je tu tretia dôležitá štruktúra `struct raw_sock`, tá je definovaná a používaná iba v rozsahu modulu protokolového ovládača. Zaujímavosťou je, že sa v programovacom jazyku C využije princíp podobný dedičnosti. Ako je možné vidieť na obrázku na začiatku `struct raw_sock` sa nachádza `struct sock`, a za ňou sa nachádzajú ďalšie informácie. Štruktúra `struct raw_sock` je teda konkrétnejšia, no súčasne referenciu na takýto objekt je možné pretypovať na všeobecnejšiu `struct sock` a bez problémov ju používať a distribuovať.

4.1.1.2 Vytvorenie a deštrukcia schránky

Prvá vec, ktorú musí protokolový ovládač vykonať po svojom zavedení do systému, je registrácia daného protokolu v jadre. Táto operácia je v celku triviálna a vyžaduje jedinú inicializovanú štruktúru `struct net_proto_family`.

```
struct net_proto_family {
    int family;
    int (*create)(struct net *net, struct socket *sock,
          int protocol, int kern );
    struct module *owner;
}
```

Ukážka 4-1: Definícia štruktúry protokolovej rodiny.

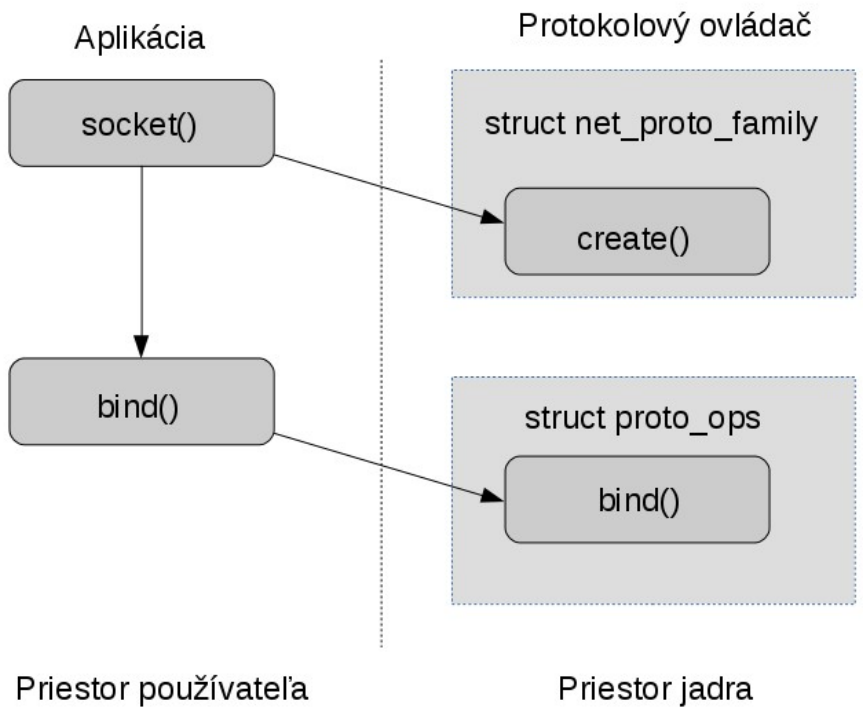
Ako je možné z definície vidieť táto štruktúra obsahuje iba najzákladnejšie veci potrebné ku registrácii protokolu. Premenná `family` obsahuje číselnú konštantu reprezentujúcu typ danej

protokolovej rodiny. Tieto konštanty sú tradične definované v hlavičkovom súbore, nachádzajúcom sa v zdrojových textoch Linuxového jadra „include/linux/socket.h“. Tento súbor bolo potrebné doplniť o definíciu konštanty, ktorá reprezentuje náš vlastný protokol.

Druhá dôležitá premenná v štruktúre je `create`, ktorá obsahuje referenciu na inicializačnú funkciu. Táto funkcia sa volá po vytvorení novej schránky (v priestore používateľa funkcia `socket()`) a je zodpovedná za alokáciu a inicializáciu potrebných štruktúr, dôležitých pre jej fungovanie.

Dôležitá vec ktorú je potrebné v tejto funkcii vyriešiť je nastavenie protokolových operácií v schránke. K tomto účelu sa použije štruktúra `struct proto_ops`, ktorá obsahuje veľké množstvo funkcií, ktoré môže protokol implementovať.

Nasledujúci obrázok ilustruje proces vzniku schránky. Aplikácia požiada jadro o vytvorenie schránky pomocou štandardnej funkcie `socket()`. Na základe požadovanej protokolovej rodiny schránky sa v jadre vyhľadá príslušný protokolový ovládač, ktorý je zaregistrovaný k takejto obsluhe. Zvyšok je už v jeho réžii, napríklad operácia bind, ktorá je tu ilustrovaná.



Obrázok 4-2: Proces vzniku schránky.

4.1.1.3 Bind

Táto operácia je kľúčová, jej úlohou je vytvorenie spojenia medzi existujúcou schránkou a konkrétnym sieťovým rozhraním. Inými slovami je to inicializácia komunikácie medzi protokolovým ovládačom a ovládačom sieťového rozhrania.

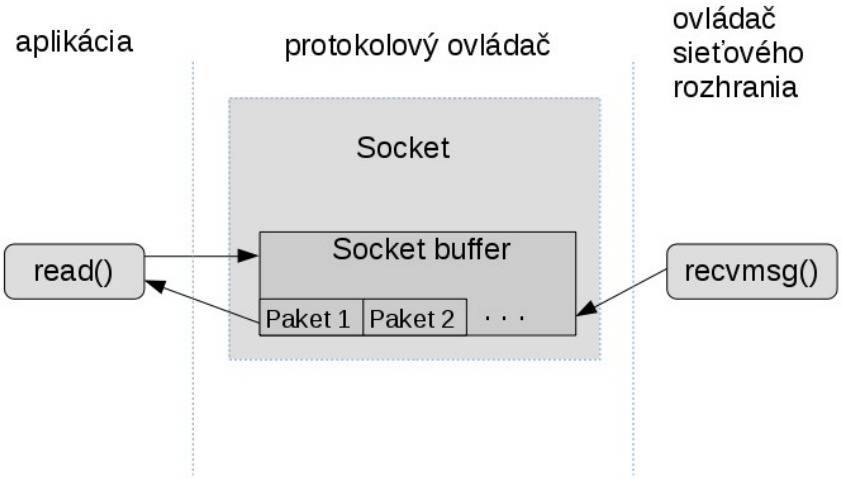
4.1.1.4 Zápis

Operácia zápisu je iniciovaná zo strany aplikačného programu. Keďže na tejto vrstve pre naše účely nie je potrebné dodatočné spracovanie prenášaných paketov, je táto operácia jednoduchá.

Protokolový ovládač len prevezme paket od používateľa a predá ho jadru spolu s informáciou na ktoré rozhranie má byť paket smerovaný. Pomocou odloženého spracovania sa neskôr sieťový subsystém postará o jeho doručenie nižšej vrstve, v tomto prípade ovládaču sieťového rozhrania. Dôležitý je fakt, že samotný prenos prebieha nezávisle. Protokolový ovládač teda nečaká na fyzické vykonanie prenosu. V okamžiku ukončenia operácie zápisu na tejto úrovni sa paket ani nemusel dostať k samotnému ovládaču sieťovej karty.

4.1.1.5 Čítanie

Oproti zápisu je táto operácia komplikovanejšia. Tak isto je iniciovaná zo strany priestoru používateľa, no prichádzajúce pakety z ovládača sieťového rozhrania nemožno priamym spôsobom vynútiť.



Obrázok 4-3: Princíp operácie `read()` v protokolovom ovládači.

Ako je možné vidieť na obrázku, nové pakety prichádzajú zo strany ovládača sieťového rozhrania. Ten vyvolá prenos a protokolový ovládač potom daný paket spracuje a uloží. Každá schránka obsahuje vyrovnávaciu pamäť pre prichádzajúce pakety vo forme fronty. Práve sem sa prichádzajúci paket uloží až do požiadavky na nový paket zo strany používateľa.

Po takomto požiadavku sa z fronty vyberie ďalší paket a ten sa vráti do aplikačného programu. Operácia `read()` môže byť blokujúca, to znamená, že v prípade prázdnej fronty na prichádzajúce pakety je program pozastavený do príchodu najbližšieho paketu.

4.1.2 Komunikácia s ovládačom sieťového rozhrania

Z druhej strany od aplikácie protokolový ovládač zabezpečuje komunikáciu s nižšou sieťovou vrstvou, konkrétne s ovládačom sieťového rozhrania. Z tohoto hľadiska musí poskytovať dve základné operácie: prijatie paketu a odoslanie paketu. V tejto podkapitole budú tieto operácie vysvetlené.

4.1.2.1 Odoslanie paketu

Ako bolo v predchádzajúcej podkapitole vysvetlené operácia zápisu je jednoduchá, iniciuje ju aplikácia a protokolový ovládač sa musí postarať o doručenie paketu nižšej vrstve. K tomu sa použije jednoduchá funkcia `dev_queue_xmit()`, ktorú poskytuje jadro. Funkcia má jediný parameter, referenciu na štruktúru, ktorá predstavuje prvok `struct sk_buff`. Tá obsahuje všetky potrebné informácie k doručeniu paketu konkrétnemu sieťovému rozhraniu. O tento proces sa už postará samotný sieťový subsystém Linuxu.

4.1.2.2 Prijatie paketu

K prijímaniu paketov zo sieťového rozhrania je potrebné aby protokolový ovládač zaregistroval v jadre obsluhu daného typu paketu, ktorý má prijímať. Prijatie paketu funguje tak, že sa zavolá ovládačom poskytnutá obslužná rutina. Tá paket spracuje a vloží ho do vyrovnávacej pamäti všetkých otvorených schránok nad daným protokolom. Tento dej bol ukázaný v predchádzajúcej podkapitole venujúcej sa operácii čítania.

V obsluhe prijatého paketu musí byť informácia o všetkých otvorených schránkach, k tomu slúži naša vlastná definovaná štruktúra:

```
struct miwi_receiver {
    struct list_head head;
    struct sock *sk;
}
```

Ukážka 4-2: Definícia štruktúry prijímača.

K uloženiu týchto prvkov sa použije obojsmerne viazaný zoznam, ktorého implementácia je poskytnutá priamo v jadre operačného systému Linux.

Z dôvodu, že k tomuto zoznamu bude pristupované konkurentne, je potrebné zabezpečiť synchronizáciu prístupu. Operácie nad zoznamom trvajú veľmi krátko a možné konflikty sú len v prípade vytvorenia a deštrukcie novej schránky, čo nie je nijako masívne opakujúca sa operácia. Na základe toho som sa rozhodol použiť metódu aktívneho čakania (spinlock), ktorá zabezpečuje vzájomné vylúčenie prístupu dvoch rôznych procesov k jednému zdroju.

4.2 Ovládač sieťového rozhrania

Ovládač sieťovej karty poskytuje rozhranie medzi fyzickým zariadením a vyššou sieťovou vrstvou. Každý ovládač sieťového rozhrania musí implementovať operácie sieťového zariadenia, ktoré sú definované v štruktúre `struct net_device_ops`. Podobne ako to bolo pri protokolovom ovládači aj tu možno rozdeliť jeho činnosť na dve časti, ktoré budú vysvetlené v tejto podkapitole.

4.2.1 Komunikácia s vyššou sieťovou vrstvou

O prenos paketov medzi protokolovým ovládačom a ovládačom sieťovej karty sa opäť stará sieťový subsystém jadra Linux, ktorý k tomuto účelu poskytuje dané funkcie. Ich použitie je potom triviálne.

4.2.1.1 Zápis

Funkciu pre zápis, ktorá sa nachádza v štruktúre operácií sieťového rozhrania volá jadro pri predávaní paketu. Z neho je potom možné extrahovať konkrétne dáta a ďalej s nimi pracovať. Jedná sa o ten istý paket, ktorý vytvoril protokolový ovládač pri operácii zápisu a predal ho jadru.

4.2.1.2 Čítanie

Táto operácia je vlastne vytvorenie a odovzdanie paketu do vyššej sieťovej vrstvy. Jedná sa viacmenej o presne opačnú operáciu k zápisu. Vytvorí sa štruktúra `struct sk_buff`, ktorá sa naplní patričnými dátami a následne sa predá sieťovému subsystému. Ten na základe typu paketu zistí, ktorý protokolový ovládač je schopný takýto paket prijať a vykoná sa obslužná rutina príjmu v protokolovom ovládači.

4.2.1.3 Štatistiky prevádzky

Z hľadiska funkčnosti zhromažďovanie a poskytovanie štatistiky prevádzky sieťového rozhrania nie je nutná operácia, napriek tomu som sa ju rozhodol implementovať z dôvodu lepšiemu prehľadu používania. Ovládač zariadenia počíta zvlášť prenesené pakety a prenesené byty. Tieto hodnoty pomocou špeciálnej operácie poskytuje jadru. Vo výsledku ich je možné vidieť napríklad príkazom ifconfig priamo v príkazovom riadku.

4.2.2 Komunikácia s hardvérom

Prístup k samotnej SPI zbernici je realizovaný s použitím nízko úrovňového API, ktoré poskytuje jadro operačného systému Linux. Ovládač sa už musí postarať len o inicializáciu konkrétnych štruktúr a vyvolávanie samotných prenosov na zbernici.

4.2.2.1 Inicializácia (funkcia probe)

V prvom rade je nutné aby operačný systém vedel, že náš ovládač má schopnosť obsluhovať a riadiť konkrétny hardvér, v tomto prípade sa jedná o SPI zbernicu. Táto asociácia musí byť súčasťou definície zariadenia v *Device Tree* súbore. Časť tejto definície je možné vidieť nižšie:

```
spidev@0x00 {
    compatible = “linux,miwi_spi“;
    spi-max-frequency = <1000000>;
    reg = <0>;
    gpios = <@pio 8 19 0>, /* IRQ pin */
            <@pio 8 14 0>; /* RESET pin */
}
```

Ukážka 4-3: *Device Tree* uzol popisujúci rozhranie SPI.

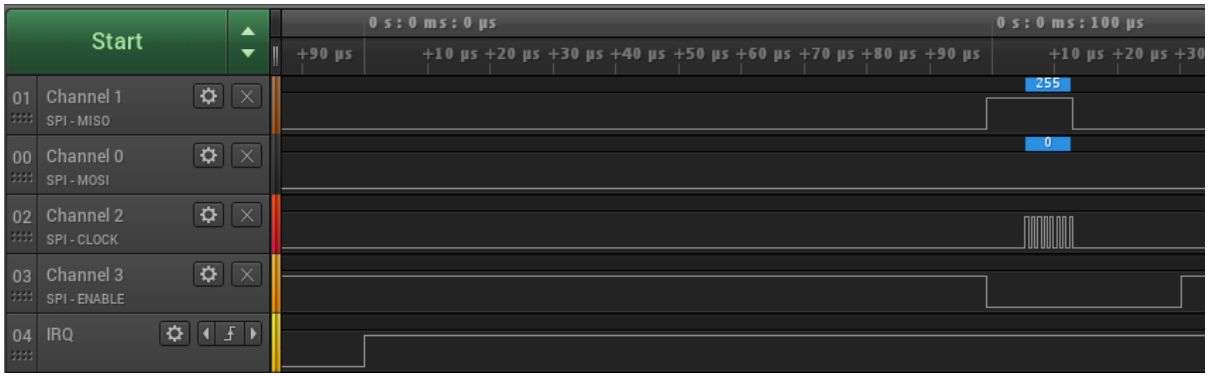
Z ukážky je možné vidieť *Device Tree* uzol, ktorý má nejaké definované vlastnosti. Dôležitá je položka `compatible`. Jej hodnota je textový reťazec na základe ktorého dokáže jadro nájsť príslušný ovládač pre jeho obsluhu. Ovládač sa musí v jadre tak isto zaregistrovať s použitím rovnakého reťazca. Okrem toho musí poskytnúť funkciu (nazývanú probe), ktorú jadro zavolá pri detekcii hardvéru.

Úlohou tejto funkcie je zjednodušene povedané inicializácia činnosti ovládača a hardvéru. Musia sa tu alokovať a inicializovať potrebné štruktúry.

Keďže zariadenie bude využívať princíp prerušenia, je potrebné toto prerušenie zaregistrovať. Tu sa dostávame k ďalšiemu dôležitému prvku v *Device Tree* uzle s označením `gpios`. Ten v sebe nesie adresy potrebných GPIO vodičov. V tomto prípade sa jedná o signály IRQ a RST. Podľa toho už v probe funkcii jednoducho zaregistrujeme obsluhu prerušenia generovanom na danom vodiči.

4.2.2.2 Načítavanie dát – spracovanie prerušenia

Operácia čítania je iniciovaná na základe signálu prerušenia. Práve pomocou prerušenia PAN koordinátor adaptéru oznamuje, že je k dispozícii nová správa, ktorú treba načítať.



Obrázok 4-4: Začiatok prenosu správy z PAN koordinátora.

Obrázok 4-4 je výstupom analyzátoru logických obvodov napojeného na SPI zbernicu počas prenosu správy s použitím už implementovaného ovládača. Konkrétnejšie je tu znázornený začiatok komunikácie. Je možné vidieť, že v čase 0 príde nástupná hrana signálu IRQ, ktorá signalizuje, že je k dispozícii nová správa a je ju potrebné načítať. Spracovanie tejto udalosti adaptéru trvá približne 100 mikrosekúnd po ktorom je možné vidieť prenos prvého bytu s hodnotou 255.

Za tento čas sa vygeneruje prerušenie. V kontexte prerušenia sa zavolá zaregistrovaná obslužná funkcia, ktorej úlohou je do pracovnej fronty naplánovať operáciu načítania správy zo zbernice. Systém potom z pracovnej fronty vyberie úlohu, v ktorej je fyzická operácia nad SPI zbernicou. Prenos potom pokračuje načítaním ďalších bytov z hlavičky správy. Hlavička má vždy fixnú dĺžku, a obsahuje informáciu o počte bytov správy, ktorá ju nasleduje.

Obrázok 4-5: Koniec prenosu hlavičky správy z PAN koordinátora.

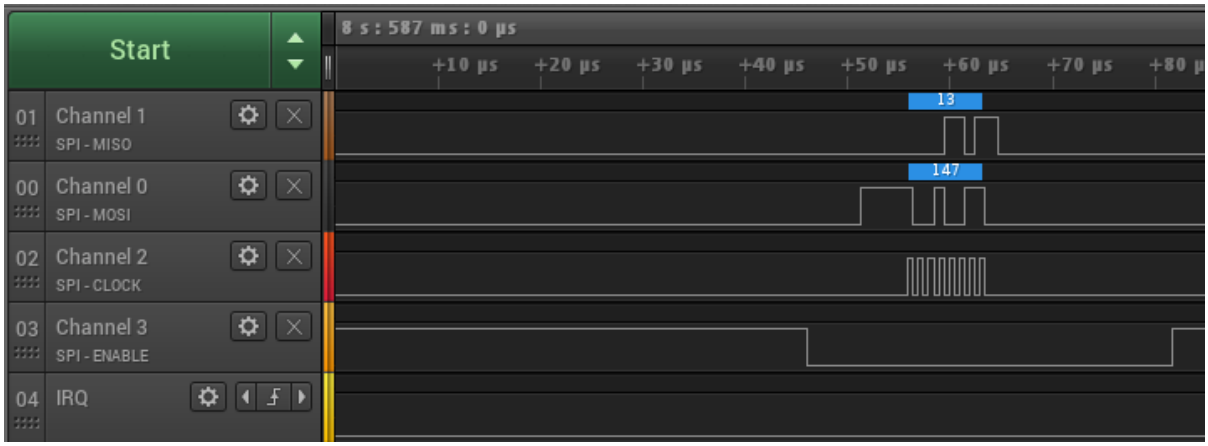
Na obrázku 4-5 je naopak možné vidieť koniec prenosu hlavičky. Po načítaní posledného bytu sa signál prerušenia vráti do logickej nuly. Po veľmi krátkom časovom úseku, ktorý je vyznačený na obrázku signál prerušenia opäť signalizuje dostupnosť dát. V tomto prípade sa jedná o dáta nasledujúce hlavičku správy. Na rozdiel od hlavičky tieto dáta nemajú fixnú dĺžku, ale ich dĺžka je rovná poslednému bytu hlavičky. Z obrázku je jasne vidieť, že sú k dispozícii 4 byty.

Pri prvotných testoch implementácie som zistil, že časový úsek medzi vzostupnou a nástupnou hranou signálu IRQ medzi hlavičkou a dátami je príliš krátky na to aby ho bolo možné spoľahlivo

detegovať. Tento problém som vyriešil uspaním procesu na veľmi krátku, no dostatočne dlhú dobu medzi načítaním hlavičky a načítavaním dát. Akékoľvek zachytené prerušenie zo signálu IRQ je až do dokončenia prenosu ignorované. Po dokončení prenosu sa dáta zabalia do paketu a predajú vyššej sieťovej vrstve.

4.2.2.3 Odosielanie dát

Zápis začína príchodom dát zo sieťovej vrstvy. Tu sa do pracovnej fronty sa naplánuje úloha zápisu a uloží sa spolu s paketom, ktorý je určený na odoslanie. Systém potom opäť z pracovnej fronty vyberie úlohu a dáta z paketu sa zapíšu na zbernicu. Začiatok takéhoto prenosu je možné vidieť na nasledujúcom obrázku 4-6.



Obrázok 4-6: Zápis jedného bytu na PAN koordinátor.

Na zbernicu sa zapíše hodnota 147 (signál MOSI). Hodnota na vodiči MISO je v tomto prípade bezvýznamná.

Keďže operácia zápisu a čítania sú na sebe nezávislé, je potrebné riadiť prístup ku zbernici SPI, k tomu je použitá jednoduchá metóda vzájomného vylúčenia označovaná tiež ako mutex.

5 Použitie

Po implementácii je nutné vysvetliť a určiť akým spôsobom môžu dané riešenie využívať používateľské procesy. V digitálnej prílohe sú zahrnuté jednoduché programy a skripty, ktoré demonštrujú použitie ovládačov v systéme. Táto kapitola bližšie vysvetlí jednotlivé dôležité operácie z pohľadu aplikačného programu.

5.1 Vytvorenie prístupu

Ešte pred samotným použitím ovládačov je potrebné systém pripraviť. Táto príprava sa dá rozdeliť na dve časti. Prvou časťou je vloženie dynamických modulov do bežiaceho jadra pomocou nástroja modprobe.

Druhou časťou je vytvorenie virtuálneho sieťového rozhrania typu „spi“. Pre tento účel sa využije nástroj ip-link, tak ako je to v ukážke:

```
1: ip link add dev spi0 type spi
2: ip link set spi0 up
```

Ukážka 5-1: Skript pre pridanie SPI sieťového rozhrania.

Príkaz na prvom riadku vytvorí a inicializuje sieťové rozhranie s názvom spi0, ktoré je typu spi. Práve podľa typu príkazu systém vyberie práve náš ovládač, ktorý ho bude obsluhovať ďalej obsluhovať. Druhým riadok sa už len vytvorené rozhranie nastaví do stavu up, takže ho je možné používať.

5.2 Vytvorenie schránky

Nová schránka vlastného protokolu sa vytvára takmer rovnako ako schránka bežne používaného internetového protokolu. Použije sa funkcia `socket()`, no požadovaná rodina protokolu bude odlišná.

Po vytvorení schránky je ešte potrebné vytvoriť spojenie schránky so sieťovým rozhraním, takáto operácia sa označuje ako bind a v aplikácii sa použije funkcia `bind()`. Aby bind vedel na ktoré rozhranie má schránku naviazať, je potrebné mu predať identifikačné číslo rozhrania.

5.3 Komunikácia

Po spojení vytvorenej schránky na dané rozhranie je možné posielať, respektíve prijímať správy použitím štandardných systémových volaní `read()` a `write()`.

6 Testovanie

Na implementovanom ovládači bolo potrebné otestovať jeho funkčnosť a stabilitu. Počas vývoja som mal k dispozícii špeciálne upravený PAN koordinátor, ktorý periodicky každú sekundu posielal pevne danú správu. Ten plne vystačoval k overeniu práve vyvíjanej časti ovládača.

6.1 Únik pamäti

Všetky pamäťové zdroje alokované a následne neuvoľnené v priestore jadra znamenajú pamäťové úniky až do reštartu celého systému. Pre systém určený na nepretržitú prevádzku sú takéto úniky neprípustné. Linux ponúka nástroj na detekciu pamäťových únikov v jadre *kmemleak*, ktorý bol použitý počas všetkých uvedených testov v tejto kapitole [19].

6.2 Testovanie komunikácie

Po implementácii ovládača som ho musel otestovať s plne funkčným PAN koordinátorom a reálnymi senzormi, ktoré zaznamenávali teplotu. Prvý test bol s jedným senzorom a trval približne 24 hodín. Počas tejto doby nebol zaznamenaný žiadny pád ani žiadne výkyvy stability operačného systému.

```
spi0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00
          UP RUNNING NOARP  MTU:36  Metric:1
          RX packets:28567 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:542739 (530.0 KiB)  TX bytes:32 (32.0 B)
```

Obrázok 5-1: Štatistika prevádzky po 24 hodinách, 1 senzor.

Keďže ovládač sieťovej karty implementuje počítanie štatistiky prevádzky, bolo možné pomocou príkazu ifconfig tieto štatistiky získať. Na obrázku 5-1 je možné vidieť práve tento výpis. Jediný prenesený paket smerom do PAN koordinátora predstavuje príkaz na párovanie nového senzoru. Po úspešnom spárovaní senzor periodicky posielal správy, ktoré okrem iného, obsahovali aj hodnotu meranej teploty.

Druhý test bol s použitím dvojice teplotných senzorov a tiež trval približne 24 hodín. Na základe prvého testu bol priebeh očakávane stabilný a hodnoty prijatých paketov boli približne dvojnásobné.

Oba testy však preukázali pamäťové úniky pri prenášaní paketu od ovládača sieťovej karty do protokolového ovládača. Zdroj problému bol úspešne odhalený a opravený.

6.3 Testovanie vkladania a odoberania

Operácia vloženia modulu do jadra je dôležitá a tiež ju bolo potrebné otestovať. Na druhej strane je potrebné aby sa dal modul z jadra kedykoľvek odobrať.

Tieto fakty som sa rozhodol spojiť do jediného automatizovaného testu, ktorý periodicky pridáva a odoberá ovládače z jadra. Medzi pridaním a odobraním sa pritom spustí program, ktorý príjme správu od PAN koordinátora. Pri samotnom teste sa táto akcia opakovala 50 000 krát.

Tento test odhalil závažné chyby, ktoré spôsobovali pád celého systému. Tie boli úspešne odhalené a opravené.

7 Záver

Cieľom tejto bakalárskej práce bola implementácia ovládača PAN koordinátora protokolu MiWi pre Linux, komunikujúceho pomocou zbernice SPI. Prvotným predpokladom zvládnutia tejto úlohy bolo naštudovať princípy a spôsoby tvorby ovládačov a programovanie jadrových modulov pre jadro operačného systému Linux. Súčasne bolo potrebné sa zoznámiť s prekladovým systémom jadra Linuxu. Z praktického hľadiska bolo tiež nutné zoznámiť sa so samotným hardvérom ako aj pochopiť princíp a použitie rôznych vývojových softvérových aj hardvérových nástrojov pre vývoj vstavaných systémov. Na základe získaných znalostí bolo navrhnuté riešenie problému zahrňujúce dvojicu ovládačov. Oba ovládače boli úspešne implementované tak, aby splnili svoj účel pre ktorý boli vytvorené. Riešenie bolo úspešne testované, pri čom boli odhalené a následne opravené drobné chyby v implementácii.

Výsledok oproti súčasnému riešeniu zapúzdruje všetky činnosti obsluhy PAN koordinátora do jednotného rozhrania, ktorého implementácia je skrytá v priestore jadra. Riešenie je pri tom flexibilnejšie, rýchlejšie a v prípade požiadavky použiť v projekte iný zdroj dát než je SPI zbernica, stačí doimplementovať daný ovládač sieťovej karty bez nutnosti modifikovať aplikáciu priestoru používateľa.

7.1 Rozšírenie projektu

Prácu by bolo možné v prípade potreby do budúcnosti rozšíriť napríklad implementáciou ovládača sieťového rozhrania pre iné fyzické rozhranie než SPI, napríklad USB vysielače a podobne.
Samotný protokolový ovládač by sa dal rozšíriť o pokročilú štatistiky prevádzky, prístupnú napríklad cez súborový systém sysfs.

Keďže samotný projekt IoT sa v dobe dokončenia tejto práce stále nachádza vo fáze vývoja, implementovaný ovládač bude zrejme potrebné aktualizovať tak aby podporoval najnovšie verzie použitých protokolov.

Literatúra

[1] CORBET, Jonathan, Alessandro RUBINI, Greg KROAH-HARTMAN a Alessandro RUBINI. *Linux device drivers*. 3rd ed. Sebastopol, CA: O'Reilly, 2005, xviii, 615 p. ISBN 0596005903.

[2] YAGHMOUR, Karim. *Building embedded Linux systems*. 2nd ed. Cambridge: O'Reilly, c2008, xx, 439 p. ISBN 9780596529680.

[3] BOVET, Daniel P a Marco CESATI. *Understanding the Linux kernel*. 3rd ed. Sebastopol, CA: O'Reilly, c2006, 923 p. ISBN 0596005652.

[4] RUSLING, David A. Device Drivers. In: *The Linux Documentation Project.* [online], [cit. 3. 1. 2015], URL: http://www.tldp.org/LDP/tlk/dd/drivers.html

[5] SIMMONDS, Chris. Kernel and user space. In: *The Embedded linux Quick Start Guide.* [online], [cit. 3. 1. 2015], URL: http://elinux.org/images/4/4f/02-linux-quick-start.pdf

[6] CALBET, Xavier. Writing device drivers in Linux: A brief tutorial. In: *Free Software Magazine.* [online], [cit. 3. 1. 2015], URL: http://fsmsh.com/1238

[7] Kernel modules, In: *archlinux wiki.* [online], [cit. 3. 1. 2015] URL: https://wiki.archlinux.org/index.php/kernel_modules

[8] MiWi™ Protocol, In: *Web stránky Microchip Technology Inc.* [online], [cit. 3. 1. 2015] URL: http://www.microchip.com/miwi/

[9] A10-OlinuXino-LIME, In:*Web stránky spoločnosti Olimex.* [online], [cit. 3. 1. 2015] URL: https://www.olimex.com/Products/OLinuXino/A10/A10-OLinuXino-LIME/open-source-hardware

[10] BIDLO, Michal. Slidy k prednáške č. 4 *Synchronní sériová rozhraní: SPI, IIC* k predmetu IMP na Fakulte Informačních Technologií Vysokého Učení Technického v Brně, Neverejne dostupné online v informačnom systéme FIT VUT v Brně [cit. 3. 1. 2015]

[11] spidev, In: *Linux Kernel Documentation.* [online], [cit. 22. 1. 2015] URL: https://www.kernel.org/doc/Documentation/spi/spidev

[12] *OpenEmbedded wiki*. [online], [cit. 1. 2. 2015], URL: http://www.openembedded.org/

[13] SocketCAN, In: *Linux Kernel Documentation.* [online], [cit. 1. 2. 2015] URL: https://www.kernel.org/doc/Documentation/networking/can.txt

[14] JELÍNEK, Lukáš. *Jádro systému Linux: kompletní průvodce programátora*. Vyd. 1. Brno: Computer Press, 2008, 686 s. Programování. ISBN 9788025120842.

[15] *Device Tree wiki.* [online], [cit. 8. 5. 2015], URL: http://www.devicetree.org/

[16] PETAZZONI, Thomas. *Device Tree for Dummies.* [online], [cit. 8. 5. 2015] URL: http://events.linuxfoundation.org/sites/events/files/slides/petazzoni-device-tree-dummies.pdf

[17] JELÍNEK, Lukáš. Vývoj jádra VIII. - přerušení a odložené zpracování, In: *LinuxEXPRESS.* [online], [8. 5. 2015]
URL: http://www.linuxexpres.cz/praxe/vyvoj-jadra-viii-preruseni-a-odlozene-zpracovani

[18] LOGINOVA,Vita. *Multitasking in the Linux Kernel. Interrupts and Tasktlets.* [online], [cit. 8. 5. 2015]
URL: http://kukuruku.co/hub/nix/multitasking-in-the-linux-kernel-interrupts-and-tasklets

[19] kmemleak, In: *Linux Kernel Documentation.* [online], [cit. 15. 5. 2015]
URL: https://www.kernel.org/doc/Documentation/kmemleak.txt

Príloha A

Obsah priloženého Cd

Koreňový adresár obsahuje 3 priečinky:

/doc	Obsahuje programovú dokumentáciu ovládačov vygenerovanú nástrojom Kernel-doc.
/kernel	Obsahuje konkrétnu verziu jadra, ktorá bola použitá pri vývoji. Okrem toho obsahuje aj patch súbory a stručný návod ako ich aplikovať.
/src	Obsahuje zdrojové texty ovládačov, ukážkové príklady a skripty pre použitie.