VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

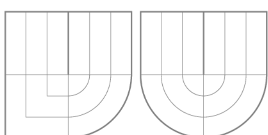
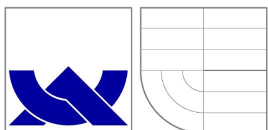
NÁVRH SÍŤOVÝCH APLIKACÍ NA PLATFORMĚ NETCOPE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

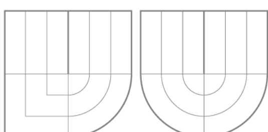
AUTOR PRÁCE
AUTHOR

Bc. ANDREJ HANK

BRNO 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁVRH SÍŤOVÝCH APLIKACÍ NA PLATFORMĚ NETCOPE

DESIGN OF NETWORK APPLICATIONS FOR A NETCOPE PLATFORM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. ANDREJ HANK
AUTHOR

VEDOUCÍ PRÁCE — Ing. TOMÁŠ MARTÍNEK
SUPERVISOR

BRNO 2009

Abstrakt

Monitorování a zaručení bezpečnosti vysoko-propustných sítí s rychlostmi od 1 do 100 Gb/s si vyžaduje hardwarovou akceleraci. Platforma NetCOPE pro rychlý vývoj síťových aplikací využívá akceleraci hardwarovou kartou s technologií FPGA přístupem "Hardware/Software Codesign". Zvýšení výkonnosti softwarové části platformy je podmíněno paralelním zpracováním aplikací pro využití více jader procesoru. Tato práce analyzuje architekturu platformy NetCOPE a možnosti paralelního zpracování standardních síťových aplikací, navrhuje modely využití více jader procesoru umožněním souběžného zpracování dat nad platformou NetCOPE, a následně je implementuje. Podpora modelů je integrovaná do vrstvy systémových ovladačů systému Linux a uživatelských knihoven, které vytváří jednoduché rozhraní pro využití této podpory. Pro dosáhnutí vysoké propustnosti řešení se práce věnuje jeho optimalizacím. Dosáhnuté výsledky jsou změřené vytvořenými testovacími nástroji.

Abstract

Monitoring and security in multigigabit networks with speeds 1 - 100 Gb/s needs hardware acceleration. NetCOPE platform for rapid development of network applications uses hardware acceleration card with FPGA technology by means of hardware/software codesign. Increas in performance of platform's software part is dependent of parallel processing in applications to take advantage of utilising more processor cores. This thesis analyses NetCOPE platform architecture and possibilities of parallelising classic network applications and creates models of concurrent access to data in NetCOPE platform to utilize more processor cores. These models are subsequently implemented as extensions to platform's Linux system drivers. Userspace libraries are created to provide simple interface for applications to use these new features. To achieve high throughput of this solution several optimizations are performed. Results are measured by created testing tools.

Klíčová slova

FPGA, SMP, paralelizace, síťové aplikace, NetCOPE, Liberouter, Linux, ovladače, PCAP, szedata, MISD, SPMD

Keywords

FPGA, SMP, parallelism, network applications, NetCOPE, Liberouter, Linux, drivers, PCAP, szedata, MISD, SPMD

Citácia

Andrej Hank: Návrh síťových aplikací na platformě
NetCOPE, diplomová práce, Brno, FIT VUT v Brně, 2009

Návrh síťových aplikací na platformě NetCOPE

Prehlásenie

Prehlasujem, že som túto prácu vypracoval samostatne pod vedením pána Ing. Tomáša Martínka.

. . . .
Andrej Hank
25. mája 2009

Poďakovanie

Chcem poďakovať účastníkom projektu Liberouter za spoluprácu.

© Andrej Hank, 2009.
Táto práca vznikla ako školské dielo na Vysokom učení technickom v Brne, Fakulte informačných technológií. Práca je chránená autorským zákonom a jej použitie bez udelenia oprávnenia autorom je nezákonné, s výnimkou zákonom definovaných prípadov.

Obsah

1 Úvod **4**

2 Platforma NetCOPE **7**
2.1 Projekt Liberouter 7
2.2 NetCOPE - platforma pre rýchly vývoj sieťových aplikácií 7
2.3 Celková architektúra 8
2.4 Firmvérová architektúra 9
2.5 Softvérová architektúra 9
2.6 Komunikácia medzi hardvérom a softvérom 11
2.6.1 DMA prenosy 12
2.6.2 Kruhový buffer v pamäti RAM 13
2.6.3 Formát prenášaných dát 15
2.6.4 Rozhranie szedata2 15

3 Paralelné spracovanie **18**
3.1 Základné rozdelenie architektúr 18
3.1.1 SISD (Single Instruction, Single Data) 19
3.1.2 SIMD (Single Instruction, Multiple Data) 19
3.1.3 MISD (Multiple Instruction, Single Data) 19
3.1.4 MIMD (Multiple Instruction, Multiple Data) 19
3.2 Pamäť cache 20
3.2.1 Koherencia 21
3.3 Synchronizačné nástroje 21
3.3.1 Race conditions 22
3.3.2 Determinizmus 22
3.4 Paralelizmus z pohľadu operačného systému 22
3.4.1 Multitasking 22
3.4.2 Procesy a vlákna 23
3.5 Paralelizmus aplikácií na systémoch so zdieľanou pamäťou 23
3.5.1 Metodika tvorby paralelných programov 24
3.5.2 Pthreads 26
3.5.3 OpenMP 26
3.5.4 Reentrantný a „thread safe" kód 26

4 Paralelné spracovanie softvérových aplikácií nad platformou NetCOPE **28**
4.1 Typické sieťové (monitorovacie) aplikácie a možnosti ich paralelizácie 28
4.1.1 Firewall 28
4.1.2 IDS 32
4.1.3 Monitorovanie sieťových tokov 33

4.1.4 UTM 34
4.1.5 Router 34
4.2 Podpora aplikačnej paralelizácie platformou NetCOPE 35
4.2.1 RX smer 35
4.2.2 TX smer 37

5 Systémové ovládače **39**

6 Ovládač szedata2 **44**
6.1 Architektúra ovládačov projektu Liberouter 44
6.2 Architektúra ovládačov szedata2 46
6.3 Hardvérovo závislá časť 46
6.4 Hardvérovo nezávislá časť 47
6.4.1 Použité štruktúry 47
6.4.2 Správa zariadení 47
6.4.3 Kruhové buffery 49
6.4.4 Komunikácia s „userspace” 49
6.4.5 Správa aplikácií 50
6.4.6 Príjem a odosielanie dát 51
6.4.7 RX 52
6.4.8 TX 54
6.4.9 Synchronizácia 55

7 Knižnice libsze2 a PCAP **57**
7.1 Libsze2 57
7.1.1 Úlohy a štruktúra knižnice 57
7.1.2 Módy prenosu dát 57
7.1.3 Blokujúce a neblokujúce operácie 59
7.1.4 Zalomenie kruhového bufferu 59
7.1.5 Vývoj a súčasný stav 60
7.2 Rozšírenie PCAP 61

8 Vývoj, testovanie a ladenie výkonnosti **62**
8.1 Nástroje 63
8.1.1 szetest.ko 63
8.1.2 szetest2 63
8.1.3 sze2read 64
8.1.4 crctest 64
8.1.5 sze2write 64
8.1.6 test_tx_burst 65
8.1.7 sze2loopback 65
8.1.8 sze2read-mt 65
8.1.9 sze2write-mt 65
8.1.10 sze2oversize_test 65
8.2 Ladenie výkonnosti 65
8.3 Dosiahnuté výsledky 68
8.3.1 Buffer oversize 68
8.3.2 SPMD RX 69
8.3.3 Paralelný TX 71

9 Záver **72**

A Rozhranie knižnice libsze2 **75**

B Rozhranie knižnice PCAP **76**

C UML diagramy **77**

Kapitola 1

Úvod

Súčasný rozvoj výpočtovej techniky a komunikácie cez počítačové siete priniesol využívanie široko-priepustných sietí. Tieto siete majú svoje uplatnenie najmä u väčších subjektov akými sú poskytovatelia internetového pripojenia a v tzv. chrbtových sieťach[1]. 1 Gb/s a 10 Gb/s ethernet je v tejto oblasti reálne používaný a vývoj smeruje k riešeniam s rýchlosťami 40 Gb/s až 100 Gb/s.

Dôležitou činnosťou v takomto prostredí je monitoring a analýza dát prenášaných sieťami. Tieto úkony sa zmenami legislatívy v poslednom období začínajú stávať dokonca povinnými. Problematika bezpečnosti sietí získava stále viac na význame. Systémy ako firewall, IDS alebo komplexné riešenia UTM prispievajú k zvyšovaniu sieťovej bezpečnosti. Podstatou práce týchto systémov je analýza sieťovej premávky, dát sieťových protokolov a samotného obsahu prenášaných dát. Pri práci na sieťach 1 Gb/s ethernet je v najkritickejšom prípade, spracovanie najkratších ethernetových rámcov veľkosti 64 bajtov, nutné spracovať 2 M rámcov za sekundu. Pri spracovaní týchto dát pomocou štandardného procesoru s frekvenciou 3 GHz môžeme v priemere pre spracovanie jedného rámca využiť 1536 taktov procesoru. Pri rovnakom prístupe na sieťach s rýchlosťami 10 Gb/s a 40 Gb/s sa počet taktov k dispozícii znižuje na čísla 153 a 38 na jeden rámec. Tento výpočtový výkon daný počtom dostupných taktov procesoru je nedostačujúci a preto je nutné hľadať rôzne riešenia, ktoré dokážu poskytnúť zlepšenie tohoto stavu.

Základným trendom posledných dekád vývoja mikroprocesorov bolo zvyšovanie pracovnej frekvencie. Tento trend však postupne stráca svoj význam, pretože prináša negatívne javy ako vysoký príkon či zahrievanie. Preto sa najvýznamnejším trendom vývoja štandardne používaných procesorov posledných rokov stáva zvyšovanie počtu jadier na jednom čipe. Pre využitie všetkých jadier procesoru je nutná paralelizácia programov, ktorá nie je triviálnym procesom, ale môže priniesť významný nárast výkonu aplikácií.

Iným veľmi účinným prostriedkom pre akceleráciu výpočtov a spracovania dát je hardvér. Čisto hardvérové riešenia prinášajú obrovské zrýchlenia avšak bývajú úzko špecializované a finančne veľmi nákladné. Z tohoto dôvodu je využívaná technika hardvér-softvér kodizajn, ktorá, ako jej názov naznačuje, využíva výhody softvérového i hardvérového prístupu. Hardvérová akcelerácia je použitá pre najkritickejšie časti celkového riešenia, komplexnejšie úlohy vyžadujúce flexibilitu sú riešené na úrovni softvéru. Možnosť využiť štandardne dostupného hardvéru osobných počítačov a hardvérovú akceleračnú kartu je s týmto prístupom možná a zároveň veľmi perspektívna, pretože môže priniesť flexibilné a finančne výhodné riešenia.

Projekt Liberouter a platforma NetCOPE prinášajú hardvérové akceleračné karty využívajúce technológiu FPGA[2] použiteľné so súčasným štandardným hardvérom osobných počí-

[1] anglicky backbone networks

[2] Field-programmable gate array vyvíjané spoločnosťou Xilinx

tačov. Táto platforma poskytuje základné prostriedky potrebné pre vývoj sieťových aplikácií a umožňuje rýchly vývoj technikami hardvér-softvér kodizajn. Softvérová časť platformy však zatiaľ nevyužíva prístup používania viacerých jadier procesoru. Táto práca sa zameriava na možnosti využitia viacerých jadier procesoru pre zvýšenie výkonu softvérových sieťových aplikácií umožnením súbežného spracovania dát nad platformou NetCOPE.

Pre komunikáciu s hardvérovou časťou platformy, akceleračnou kartou, slúži vrstva systémových ovládačov platformy spolupracujúca s jadrom operačného systému Linux. Ovládač szedata2 tejto vrstvy vytvára rozhranie, prostredníctvom ktorého môžu aplikácie uskutočňovať prenosy dát medzi užívateľským adresovým priestorom a hardvérovou akceleračnou kartou. Riadenie prenosov dát medzi týmito dvomi entitami je plne v moci ovládača, preto je práve systémový ovládač riadiaci prenosy dát kľúčovým miestom pre implementáciu podpory súbežného spracovania dát. Práca analyzuje architektúru vrstvy ovládačov platformy NetCOPE pričom sa sústreďuje práve na ovládač szedata2 zabezpečujúci prenos dát. Navrhnuté sú možnosti podpory súbežného spracovania dát aplikáciami v kontexte teórie systémových ovládačov.

Rozhranie systémových volaní pre komunikáciu s operačným systémom a systémovými ovládačmi je vytvorené tak, aby pomocou neho bolo možné ovládať všetky existujúce zariadenia rôznych druhov. Pre rýchly a efektívny vývoj aplikácií využívajúcich platformu NetCOPE je však príliš zložité. Rozhranie ovládača pre prenos dát je veľmi špecifické. Jednoduché rozhranie, pomocou ktorého je možné využívať prenosy dát medzi aplikáciami a platformou, vytvára užívateľská knižnica libsze2. Široko používaným štandardným rozhraním pre spracovanie sieťových dát na linkovej vrstve je knižnica PCAP. Možnosť využívať prenosy dát nad platformou cez toto rozhranie umožňuje hardverovú akceleráciu štandardných, široko používaných sieťových aplikácií. Táto práca zahŕňa vytvorenie knižnice libsze2 a rozšírenie knižnice PCAP o podporu platformy NetCOPE.

Ako už bolo uvedené, pri vysokých priepustnostiach v chrbtových sieťach a extrémne veľkému množstvu tečúcich paketov je možné pre spracovanie jediného paketu na súčasných výkonných procesoroch využiť len stovky, prípadne desiatky taktov procesoru. Aplikácie pracujúce s týmito množstvami dát sú výkonovo veľmi náročné. Preto je nutné pri vývoji softvérovej vrstvy platformy dbať na nízku výpočtovú náročnosť tejto vrstvy, keďže spotrebovaný výpočtový výkon uberá ďalšie dôležité takty procesoru pre spracovanie paketu samotnou aplikáciou. Softvérová vrstva využíva optimalizácií a je systematicky analyzovaná pre dosiahnutie vysokej priepustnosti dátových prenosov spolu s nízkou výkonovou náročnosťou.

Práca je členená do deviatich kapitol. V druhej kapitole je popísaná platforma NetCOPE a podrobnejšie je vysvetlený proces prenosu dát k aplikáciám. Tretia kapitola všeobecne pojednáva o problematike paralelného spracovania. Sú tu popísané základné architektúry pre paralelné spracovanie, princípy tvorby a problémy paralelných programov a implementačné mechanizmy architektúry SMP. Kapitola štvrtá analyzuje možnosti paralelného spracovania typických sieťových aplikácií a v jej závere sú prezentované modely vhodné pre podporu možností paralelného spracovania sieťových aplikácií v platforme NetCOPE. V piatej kapitole je zhrnutá základná teória oblasti systémových ovládačov systému Linux. Sú tu vysvetlené pojmy a techniky používané v tejto oblasti a teória nutná pre následnú analýzu architektúry a funkcionality systémových ovládačov platformy NetCOPE, ktorá je podaná v kapitole šiestej. Táto kapitola tiež obsahuje podrobný popis systémového ovládača zabezpečujúceho prenos dát medzi aplikáciami a platformou. Tento ovládač je cieľom niekoľkých vylepšení. Kapitola siedma vysvetľuje základné úlohy knižnice libsze2 pre vytvorenie jednoduchého rozhrania pre využívanie prenosov dát nad platformou. Taktiež je tu podaný spôsob, akým je rozšírená knižnica PCAP o podporu týchto prenosov. Predposledná kapitola popisuje prostredie projektu Liberouter, organizáciu práce, štandardne

používané nástroje a proces hľadania chýb v platforme NetCOPE. Na tieto účely slúžia vytvorené testovacie nástroje, ktoré sú v tejto kapitole podrobne popísané. Taktiež voľba niektorých významných parametrov majúcich vplyv na celkovú výkonnosť platformy je tu objasnená. Výsledky, ktoré prinášajú niektoré úpravy ovládača pre prenos dát a užívateľskej knižnice, sú tu podané v tabuľkovej forme. Posledná kapitola stručne sumarizuje obsah práce a dosiahnuté výsledky.

Kapitola 2

Platforma NetCOPE

Táto kapitola popisuje architektúru platformy NetCOPE vyvíjanej na projekte Liberouter z rôznych uhlov pohľadu. Taktiež popisuje princíp prenosu dát medzi softvérovými aplikáciami a hardvérovou kartou.

2.1 Projekt Liberouter

Tento výskumný projekt existuje vďaka spolupráci viacerých vysokých škôl v ČR (najmä FIT VUT v Brne a FI MUNI v Brne) a organizácii CESNET. Prvotným zámerom projektu, ako naznačuje jeho názov, bol vývoj multigigabitového IPv6 a IPv4 smerovača. Výskumná činnosť sa v rámci projektu vykonáva už niekoľko rokov, projekt má dostatok vývojárov a má dobré technologické i administratívne zázemie.

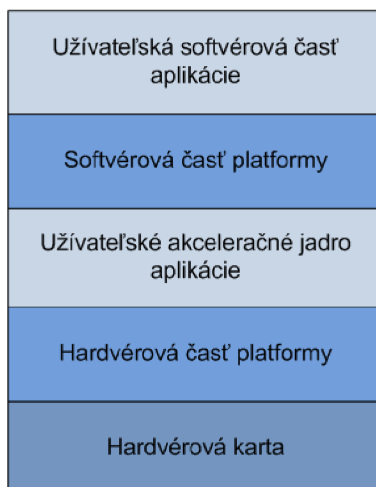
Hlavnými cieľmi projektu je vývoj rôznych sieťových minitorovacích zariadení. Pre dosiahnutie vysokého výkonu a schopnosti spracovávať dáta vo vysoko-priepustných sieťach sa využíva hardvérovej akcelerácie s pomocou technológie FPGA. Pre tieto účely bola v rámci projektu vyvinutá rada akceleračných kariet COMBO6 pracujúcich na zbernici PCI a PCI-X a len nedávno pribudla nová generácia kariet COMBOv2 pracujúca na zbernici PCI Express, ktorá poskytuje významne vyššie prenosové kapacity a mala by slúžiť pre výskum v nasledujúcich rokoch.

2.2 NetCOPE - platforma pre rýchly vývoj sieťových aplikácií

Skúsenosti z vývoja sieťových zariadení na projekte Liberouter vyústili v snahu zjednodušiť a urýchliť vývoj sieťových zariadení vytvorením platformy poskytujúcej potrebné základné funkcie. Platforma NetCOPE vytvára abstraktnú vrstvu nad kartou, odtieni nízkoúrovňové hardvérovo závislé vlastnosti a poskytuje jednoduché využívanie jej zdrojov – sieťové rozhrania, rýchle vnútroplatformové prepojenie a rýchle spojenie s hostiteľským počítačom. Zároveň platforma zahŕňa viacero pred-pripravených funkčných blokov tzv. „IP cores”, napríklad pre analýzu paketov, pre komunikáciu s pamäťami alebo blok precíznych časových značiek.

Platforma je implementovaná pre viacero hardvérových kariet od rôznych výrobcov. Karty sa líšia počtom a rýchlosťou rozhraní a pripojením na zbernicu. Pre užívateľa platformy – vývojára aplikácie, tento fakt prináša možnosť použiť vhodnú kartu s minimálnym úsilím. Aplikáciu nie je nutné meniť, platforma by mala zabezpečiť transparentnosť.

V súčasnosti platforma existuje už v jej tretej verzii s akceleračnými kartami COMBOv2.



Obrázok 2.1: Vrstvy architektúry platformy NetCOPE

2.3 Celková architektúra

Koncept platformy ako aj užívateľskej aplikácie môže byť rozdelený do niekoľkých vrstiev s definovaným rozhraním. Toto rozdelenie je zobrazené na obrázku 2.1.

- **hardvérová karta** – tvorí najnižšiu vrstvu, nesie čip FPGA. V súčasnosti sú podporovanými kartami COMBO6x, ML555 a COMBOv2.
- **hardvérová časť platformy** – táto vrstva obsahuje závislé časti platformy pre obsluhu hardvérovej karty a pre každú kartu sú rôzne. Patria sem bloky sieťového rozhrania, pripojenia na systémovú zbernicu a ďalšie jednotky pre obsluhu karty. S príchodom novej karty musí byť táto vrstva pre ňu osobitne implementovaná, ale s výhodou sa často dá použiť komponentov pripravených pre iné karty. Do hardvérovo nezávislej časti patria komponenty pre DMA prenosy a vnútorný prepojovací systém FrameLink.
- **akceleračné jadro aplikácie** – táto vrstva je implementovaná užívateľom platformy. Poskytuje priestor pre hardvérovú akceleráciu výkonovo kritických funkcií aplikácie. Rozhranie s hardvérovou časťou platformy je dostatočne abstraktné aby nebolo závislé na použitej hardvérovej karte.
- **softvérová časť platformy** – táto vrstva obsahuje systémové ovládače a poskytuje rôzne rozhrania pre softvérovú časť užívateľskej aplikácie. Zároveň obsahuje podporné knižnice, podporné a testovacie nástroje a dokumentáciu. Podrobne o tejto vrstve bude pojednávať samostatná sekcia 2.5.
- **softvérová časť aplikácie** – do tejto vrstvy patrí softvérová časť užívateľskej aplikácie. Vďaka nižším vrstvám dokáže komunikovať rozhraniami s akceleračným jadrom. Do tejto vrstvy je vhodné umiestniť komplexnú, avšak nie výkonovo veľmi náročnú funkcionalitu aplikácie. Táto vrstva poskytuje dostatočnú flexibilitu pre celkové riešenie.

2.4 Firmvérová architektúra

Celková architektúra hardvérovej časti platformy, tzv. firmvéru, je zobrazená na obrázku 2.2.

- **sieťové rozhrania** – komunikácia so sieťovými rozhraniami prebieha pomocou vstupno-výstupných blokov. Tieto bloky obsahujú štruktúry typu FIFO pre prijímanie a posielanie dát/paketov. Prijaté dáta ďalej putujú do aplikačného jadra internou zbernicou protokolom FrameLink.

- **akceleračné jadro** – dáta sú v tejto časti spracúvané a pre ďalšie spracovanie v softvérovej časti aplikácie môžu byť transformované (napr. orezanie paketu), alebo k nim môžu byť pridané metadáta ako výsledok spracovania (napr. príznak prítomnosti a číslo vyhľadávaného reťazca v pakete).

- **interný prepojovací systém** – Popisuje abstraktné rozhranie pre pripojenie aplikačného jadra k systémovej zbernici nezávislé na type použitej hardvérovej karty. Priepustnosť tejto zbernice je dostatočná pre prenos všetkých dát pri plne vyťažených sieťových rozhraniach aj s prípadnými užívateľskými metadátami. Architektúra prepojovacieho systému je podobná ako architektúra PCI Express, skladá sa z 3 základných stavebných prvkov - Root, Switch a Endpoint.

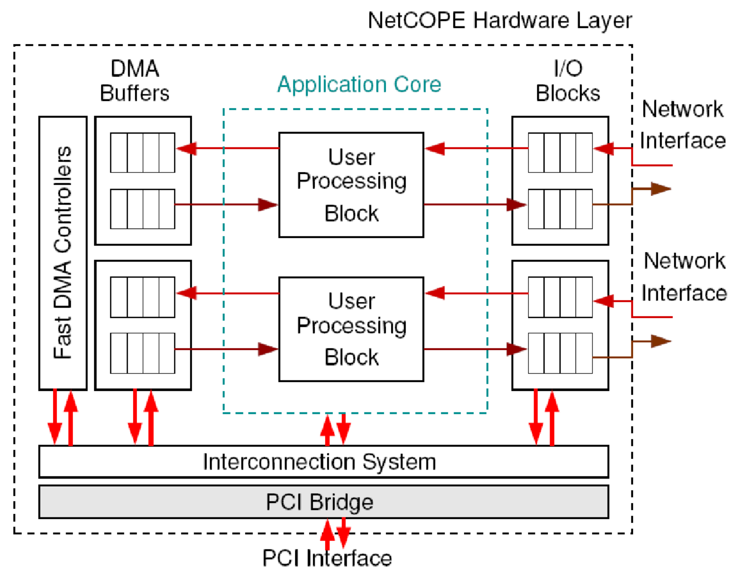
- **rýchle DMA prenosy** – v prvej verzii platformy NetCOPE bola práve táto časť úzkym hrdlom celého riešenia. DMA prenosy boli realizované mikroprocesorom PowerPC, ktoré však túto úlohu nedokázalo vykonávať dostatočne rýchlo pri príjme veľmi krátkych paketov. Preto je v súčasnej verzii platformy implementovaný úplne nový blok DMA prenosov tomuto komponentu je priraďovaný veľký význam. DMA prenosy slúžia pre prenos dát medzi užívateľskou aplikáciou a akceleračným jadrom. Sieťové aplikácie typicky prenášajú paketové dáta, niekedy obohatené aj o metadáta vytvorené pri spracovaní. Akceleračné jadro dáta určené pre softvérovú časť aplikácie posiela do DMA bufferov. DMA radič vykonáva samotný prenos a softvérová aplikácia si prenesené dáta prečíta v pamäti RAM. Podľa použitého rozhrania pre softvérovú časť aplikácie existujú dva typy DMA radičov. Prvým prípadom je prenos paketových dát cez štandardné systémové sieťové rozhranie a komunikácia prostredníctvom soketov. Druhým typom je agregovaný prenos aplikačne špecifických dát a paketových dát s metadátami. Podrobnejšie bude popísaný iba druhý typ prenosov v sekcii 2.6.

2.5 Softvérová architektúra

Softvérová vrstva platformy NetCOPE musí zabezpečovať spoluprácu hardvérovej akceleračnej karty s operačným systémom a taktiež ponúknuť rozhranie, cez ktoré bude možné s hardvérovou akceleračnou kartou pracovať a využívať jej funkcionalitu.

Softvérová architektúra je tvorená tromi vrstvami. Komunikáciu hardvérovej akceleračnej karty s operačným systémom zabezpečujú systémové ovládače. Projektové knižnice vytvárajú rozhranie pre tvorbu aplikácií pracujúcich s hardvérovou akceleračnou kartou. Vrstva nástrojov obsahuje niekoľko pred-pripravených nástrojov pre konfiguráciu štandardných hardvérových komponentov platformy a nástroje pre jej testovanie a ladenie. Táto architektúra je znázornená na obrázku 2.3.

Softvér platformy možno rozdeliť do dvoch logicky rozdielnych skupín. Prvou a základnou z nich je zabezpečenie základných vstupno-výstupných operácií nad hardvérovými



Obrázok 2.2: Architektúra firmvéru platformy NetCOPE

kartami, inicializácia kariet, bootovací proces pre nahrávanie firmvéru a zabezpečenie monitorovania a konfigurácie hardvérových komponentov. Skupina SZE je druhou skupinou softvérovej architektúry. Jej primárnou funkciou je podpora rýchlych DMA prenosov agregovaných dát do užívateľských aplikácií.

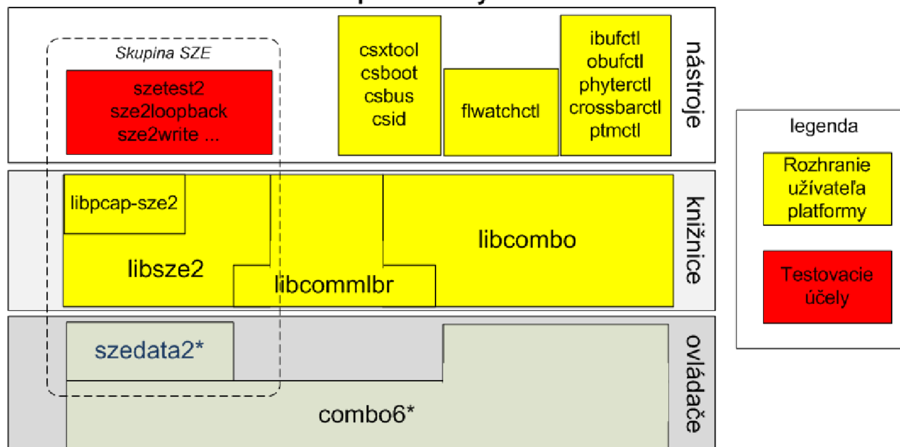
Vertikálne je softvérová vrstva členená do troch skupín, konkrétne artefakty tu budú podrobnejšie popísané:

1. Systémové ovládače – tvoria základné a najnižšie spojenie operačného systému s hardvérovou kartou a firmvérom.

    - combo6* – ovládače pre prácu s kartami COMBO6, COMBO6x, ML555 a COMBOv2. Zabezpečujú základný vstup a výstup a slúžia pre nadväzujúce knižnice.
    - szedata2* – ovládače slúžia pre využívanie rýchlych agregovaných DMA prenosov. Poskytujú základnú funkcionalitu. Podrobnejšie bude architektúra a implementácia ovládačov szedata2 popísaná v samostatnej kapitole.

2. Knižnice – tvoria nadstavbu nad ovládačmi, sprístupňujú ich funkcionalitu v užívateľsky príjemnejšej forme. Vytvárajú abstrakciu pre softvérové ovládanie hardvérových komponentov.

    - libcommlbr – najnižšie ležiaca knižnica. Obsahuje makrá, štruktúry a funkcie pre ladenie, správu verzií a taktiež sú v nej obsiahnuté rutiny pre často sa opakujúce úlohy ako spracovanie parametrov a príkazov aplikácií, konverzie medzi dátovými typmi, práca s časom, výpisy vyrovnávacích pamätí, práca s PCAP súbormi a výpis paketov.
    - libcombo – knižnica umožňuje bootovanie firmvéru, jeho inicializáciu, vstupno-výstupné operácie, prácu s firmvérom a jeho komponentmi a mapovanie adresového priestoru komponentov do užívateľských aplikácií.
    - libsze2 – poskytuje nízkoúrovňové rozhranie pre prácu s rýchlymi DMA prenosmi a rozhraním szedata2 pre oba smery. V rámci tejto práce bola táto knižnica



Obrázok 2.3: Softvérová vrstva platformy NetCOPE

rozšírená o užívateľsky príjemnejšie rozhranie vyššej úrovne. Tejto knižnici sa venuje samostatná kapitola práce.

- libpcap-sze2 – knižnica PCAP rozšírená o podporu práce cez rozhranie szedata2. Umožňuje štandardným aplikáciám, ako napríklad *tcpdump*, *wireshark* a *snort*, pracovať bez žiadnych zmien využívajúc rýchlosť agregovaných DMA prenosov. Toto rozšírenie vzniklo ako súčasť tejto práce a knižnica PCAP bude podrobnejšie popísaná v podsekcii 2.6.4.2, rozšírenie bude podrobnejšie popísané v samostatnej sekcii 7.2.

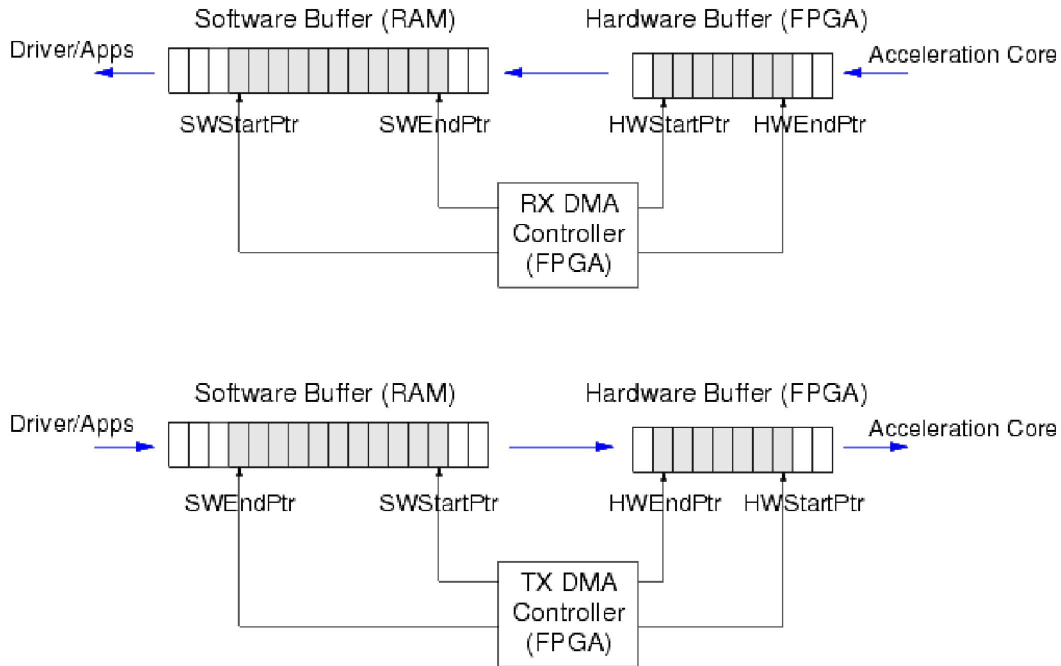
3. Nástroje – vytvárajú užívateľské rozhranie príkazového riadku pre bootovanie, ladenie, testovanie, monitorovanie a konfiguráciu hardvérových komponentov.

   - csxtool, csboot, csbus, csid – sada nástrojov pre bootovanie firmvéru a výpis informácií o aktuálnom firmvéri.
   - flwatchctl – nástroj pre monitorovanie stavu FrameLink komponentov interného prepojovacieho systému.
   - ibufctl, obufctl, phyterctl, crossbarctl, ptmctl – nástroje pre monitorovanie a konfiguráciu rovnomenných komponentov. Zabezpečujú konfiguráciu rýchlostí, duplexu, autonegociácie sieťových rozhraní a správu modulu PTM[1].
   - nástroje skupiny SZE – budú popísané v sekcii 8.1.

2.6 Komunikácia medzi hardvérom a softvérom

V tejto sekcii bude popísaný princíp komunikácie medzi firmvérom platformy NetCOPE a softvérovými aplikáciami prostredníctvom rýchlych DMA prenosov agregovaných dát. Prenosy sú realizované firmvérovým komponentom DMA radičom, s ktorým komunikuje ovládač operačného systému.

[1]Precise Timestamp Module - modul presných časových značiek



Obrázok 2.4: DMA prenosy medzi RAM a akceleračnou kartou - oba smery

2.6.1 DMA prenosy

Rýchly DMA prenos medzi pamäťou RAM a akceleračnou kartou je založený na princípe prenosu dát medzi dvoma kruhovými buffermi. Jeden je umiestnený v pamäti RAM, druhým je DMA buffer akceleračnej karty. Pre identifikáciu zaplnenia kruhových bufferov slúži v každom znich dvojica ukazateľov ukazujúcich na začiatok a koniec dátami zaplnenej časti. Na základe kontroly týchto štyroch údajov DMA radič na akceleračnej karte inicializuje DMA prenosy. Tento proces je znázornený na obrázku 2.4.

2.6.1.1 Prenos dát akceleračná karta -> RAM

Tento prenos sa riadi nasledovným algoritmom:

1. Akceleračné jadro pošle dáta, ktoré chce preniesť do softvéru, do DMA bufferu, kde sú uložené.

2. Informácia o nových dátach je predaná DMA radiču. Ten posunie hodnotu ukazateľa HwEndPtr, skontroluje dostatok miesta pre nové dáta v softvérovom bufferi (SWEndPtr - SWStartPtr >= NewDataSize) a pokiaľ miesto je, inicializuje DMA prenos dát.

3. Keď je prenos ukončený, DMA radič:

    - Posunie ukazateľ SWEndPtr o veľkosť prenesených dát.
    - Predá informáciu DMA bufferu o veľkosti prenesených dát a DMA buffer následne tieto dáta uvoľní presunom HWStartPtr.
    - Pokiaľ je to požadované, vygeneruje prerušenie informujúce o nových dátach.

4. Užívateľská softvérová aplikácia periodicky žiada prostredníctvom knižnice libsze2 ovládač o nové dáta. Ovládač overí prítomnosť dát kontrolou ukazateľov a pokiaľ

má nové dáta, dá ich aplikácii k dispozícii. Pokiaľ nové dáta nemá, aplikácia je blokovaná vo volaní *poll()*, až pokým neuplynie nastavený timeout a potom je aplikácii predaný chybový návratový kód.

5. Po spracovaní dát aplikáciou ovládač posunie ukazateľ SWStartPtr o veľkosť spracovaných dát a tým túto informáciu oznámi DMA radiču.

6. Body 1-4 sa neustále opakujú.

2.6.1.2 Prenos dát RAM -> akceleračná karta

1. Softvérová aplikácia vloží do bufferu v RAM nové dáta a predá túto informáciu ovládaču.

2. Táto informácia je propagovaná ovládačom DMA radiču posunutím SWEndPointeru o hodnotu veľkosti nových dát.

3. DMA radič skontroluje, či je v DMA bufferi dostatok miesta pre nové dáta (HWEndPtr - HWStartPtr >= NewDataSize), pokiaľ je, inicializuje DMA prenos.

4. Keď je prenos ukončený, DMA radič:

   - Posunie hodnotu SWStartPtr o veľkosť prenesených dát, tým sa dáta v RAM uvoľnia. Pokiaľ je požadované, vygeneruje prerušenie informujúce o novom voľnom priestore v RAM.
   - Predá DMA bufferu informáciu o novo prenesených dátach a posunie hodnotu HWEndPtr.

5. DMA buffer dáta odošle, uvoľní, predá DMA radiču informáciu o novo vzniknutom mieste a DMA radič posunie hodnotu HWEndPtr na príslušnú pozíciu.

6. Body 1-5 sa neustále opakujú.

2.6.2 Kruhový buffer v pamäti RAM

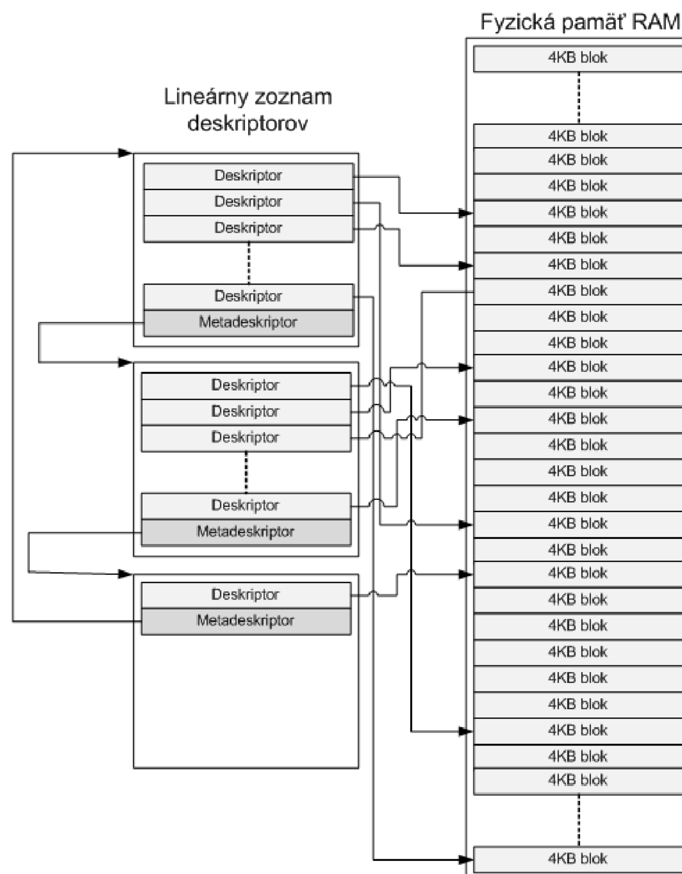
2.6.2.1 Štruktúra bufferu

Buffer sa skladá z blokov pamäte. Tieto bloky musia mať špeciálne vlastnosti pre DMA prenosy, nesmú byť nikdy odswapované ani presunuté na inú adresu vo fyzickej pamäti, inak by mohlo prísť k chybe pri DMA prenosoch. Veľkosť blokov pre DMA prenosy na architektúre x86 je 4 KB a zodpovedá štandardne používanej veľkosti stránky na tejto architektúre. Stránka je základnou jednotkou alokácie pamäte a teda táto veľkosť je pre alokáciu bezproblémová, alokácia väčších súvislých blokov pamäte už bezproblémová nie je kvôli fragmentácii pamäte v operačnom systéme.

Celková veľkosť kruhového bufferu sa pohybuje rádovo v desiatkach až stovkách megabajtov, v závislosti od použitého sieťového rozhrania a konkrétnej aplikácie. Odvodenie veľkosti bufferu je podané v sekcii 8.2.

2.6.2.2 Popis deskriptormi

Celkový kruhový buffer o veľkosti v desiatkach megabajtov je tvorený stovkami až tisícmi základných blokov. Každý z blokov môže začínať na rôznej fyzickej adrese. Pre popis tohoto



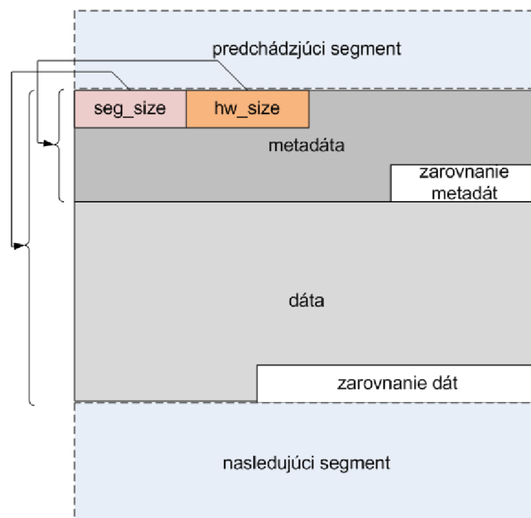
Obrázok 2.5: Popis kruhového bufferu lineárnym zoznamom deskriptorov

bufferu je možné použiť lineárny zoznam s ukazateľmi na každý alokovaný blok pamäte. Tieto ukazatele nazývame deskriptormi.

Tento zoznam musí byť prístupný DMA radiču na akceleračnej karte. Radič musí vedieť, odkiaľ a kam v rámci fyzickej pamäte má prenášať dáta. Preto musí byť celý zoznam deskriptorov uložený v pamäti schopnej DMA prenosov. Rozdelenie na bloky o veľkosti stránky je žiaduce aj pre pamäť tejto dátovej štruktúry.

Veľkosť deskriptoru je 64 bitov s ohľadom pre použitie na 64 bitových systémoch. Do 4 KB bloku sa takýchto deskriptorov zmestí 512. Keďže tento počet typicky nemusí stačiť pre popis celého kruhového bufferu, bloky deskriptorov sú prepojené do lineárneho zoznamu tak, že posledný deskriptor v danom bloku ukazuje na nasledujúci blok deskriptorov (metadeskriptor). Posledný blok deskriptorov nemusí byť plne obsadený a preto metadeskriptor, ukazujúci na začiatok kruhového bufferu, nemusí byť na poslednom mieste v bloku. Pre rozlíšenie deskriptorov a metadeskriptorov slúži ich najspodnejší bit. Táto štruktúra deskriptorov popisujúcich kruhový buffer je zobrazená na obrázku 2.5.

Aj keď je veľkosť deskriptoru 64 bitov, pre adresáciu v 64 b systémoch slúži len 52 horných bitov. Tento fakt je daný tým, že každá 4 KB stránka v systéme je zarovnaná na hranicu 4 KB. Preto možno použiť spodných 12 bitov pre ľubovolné účely, ako napríklad už spomenuté rozlíšenie deskriptorov a metadeskriptorov.



Obrázok 2.6: Štruktúra segmentu a bloku dát prenášaného DMA prenosmi

2.6.3 Formát prenášaných dát

DMA prenosy prenášajú dáta v špecifikovanom formáte. Typicky sa v jednom prenose prenáša blok dát, ktorý pozostáva z niekoľkých samostatných segmentov. Segmenty sú rôznej dĺžky. Štruktúra jedného segmentu a celého bloku dát je zobrazená na obrázku 2.6.

Hlavička segmentu je tvorená údajmi seg_size, hw_size a metadátami**.** Jej veľkosť je zarovnaná na 8 bajtov (**zarovnanie metadát**). Prvé dva bajty, **seg_size,** obsahujú dĺžku celého segmentu. Nasledujúce dva bajty, **hw_size,** obsahujú dĺžku hardvérových metadát. Veľkosť jedného segmentu je týmto obmedzená na 64KB, čo je dostatočná veľkosť. Za týmito dôležitými údajmi nasledujú metadáta vytvorené v hardvérovom akceleračnom jadre.

Za zarovnanou hlavičkou nasledujú samotné dáta, štandardne sa tu nachádza rámec linkovej vrstvy. Veľkosť dát je zarovnaná na 8 bajtov. Pokiaľ sú samotné dáta menšie ako zarovnaná hodnota, priestor je vyplnený nulami (**zarovnanie dát).**

Blok dát obsahuje viacero segmentov rôznych veľkostí. Jednotlivé segmenty sa po prenose musia spracovávať sekvenčne, pretože umiestnenie nasledujúceho segmentu je známe len po spracovaní hlavičky segmentu predchádzajúceho.

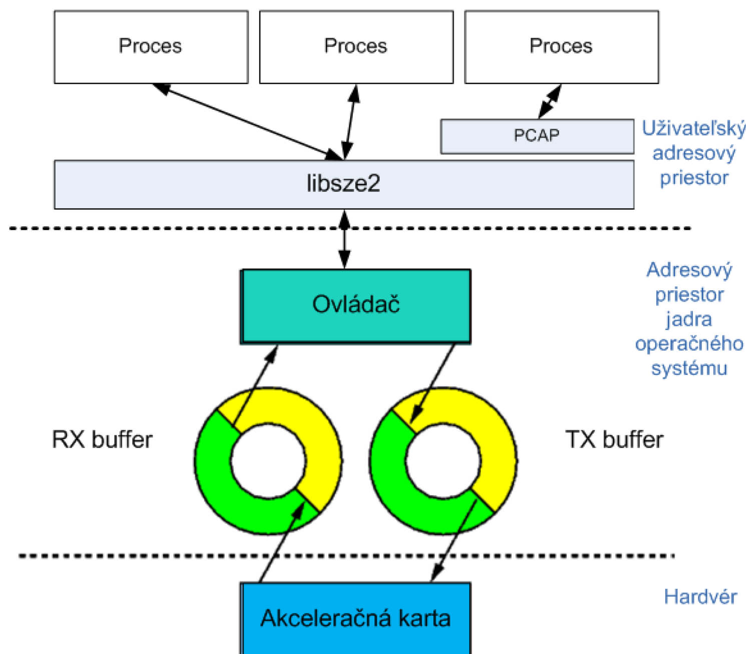
2.6.4 Rozhranie szedata2

Rozhranie szedata2 zabezpečuje softvérovým aplikáciám prístup k príjmu a odosielaniu aplikačne špecifických dát. Skratka SZE znamená „straight zero copy"[2], vyjadruje základný princíp tohoto rozhrania. Dáta sú k aplikáciám prenášané cez kruhový buffer prostredníctvom DMA prenosov.

Tento princíp je veľmi inovatívny na rozdiel od klasického prístupu cez systémové sieťové rozhrania. Prístup založený na prechode dát TCP stackom operačného systému a rozhraní soketov kopíruje dáta medzi vyrovnávacími pamäťami operačného systému a užívateľského procesu. Kopírovanie dát všeobecne znamená veľkú réžiu. SZE princíp prináša veľký nárast priepustnosti.

Rozhranie szedata2 je tvorené firmvérovými komponentmi zabezpečujúcimi DMA prenosy, systémovým ovládačom a užívateľskou knižnicou libsze2. Princíp komunikácie je zobrazený na obrázku 2.7.

[2]priame, bez kopírovania



Obrázok 2.7: Princíp prenosu dát medzi aplikáciami a ovládačom

Pokiaľ je pre prenos dát do/z hardvéru používaných viac oddelených prenosových kanálov, každý kanál pre každý smer obsahuje svoj kruhový buffer v RAM.

2.6.4.1 Komunikácia aplikácií, knižnice libsze2 a ovládača

Priame volanie ovládača aplikáciami je sprostredkované knižnicou libsze2, použitá môže byť aj knižnica PCAP, ktorá bude popísaná v nasledujúcej podsekcii. Ovládač zabezpečí, že každá aplikácia pracujúca s rozhraním szedata2 má vo svojom logickom adresovom priestore namapovaný kruhový buffer v pamäti RAM, v ktorom prebiehajú DMA prenosy.

Prenos dát prebieha periodickým povoľovaním prístupu aplikácii do časti kruhového bufferu v pamäti RAM ovládačom. Tým umožní spracovanie prijatých dát, alebo umožní zápis dát na odoslanie.

Komunikácia zo strany aplikácií prebieha volaním funkcií knižnice libsze2. Rozhranie tejto knižnice môže byť rozdelené na nízkoúrovňové funkcie priamo volajúce rozhranie ovládača a funkcie vyššej úrovne obaľujúce nízkoúrovňové funkcie spôsobom vhodným pre použitie v aplikáciách. Vyššie rozhranie knižnice libsze2 vzniklo ako súčasť tejto práce.

Rozhranie knižnice libsze2 je podrobnejšie popísané v prílohe A. Podrobne túto problematiku riešia kapitoly 6 a 7.

2.6.4.2 Rozhranie PCAP

PCAP je skratka od anglického názvu tejto knižnice – **P**acket **Cap**ture library. Táto široko používaná knižnica, ako značí jej názov, slúži primárne pre zachytávanie sieťových dát. Knižnica poskytuje rozhranie pre jazyky C a C++.

PCAP pracuje na linkovej vrstve a je používaná najmä monitorovacími aplikáciami. Medzi najznámejšie patria programy *tcpdump*, *nmap*, *wireshark* a IDS systémy *snort* a *bro*. V posledných verziách bola knižnica okrem príjmu sieťových dát obohatená aj o možnosť odosielania dát.

Knižnica je primárne vyvinutá pre UNIXové operačné systémy, existuje však aj port pre operačné systémy rodiny Windows pod názvom WinPcap. Pre použitie v iných programovacích jazykoch je možné využiť tzv. wrappery existujúce pre všetky širšie používané vyššie programovacie jazyky. Knižnica podporuje filtrovanie paketov pravidlami BPF[3].

Väčšina nástrojov využívajúcich knižnicu PCAP je skompilovaná s dynamickým linkovaním tejto knižnice. Toho sa dá úspešne využiť pre použitie platformy NetCOPE ako akcelerátoru týchto aplikácií. Knižnica PCAP bola rozšírená o podporu rozhrania szedata2 využitím knižnice libsze2. Bez rekompilácie existujúcich nástrojov, s použitím rozšírenej dynamickej knižnice označenej ako libpcap-sze2, tieto nástroje pracujú s výhodami rýchlosti platformy NetCOPE. Základné rozhranie knižnice PCAP je popísané v prílohe B a podrobnejšie o rozšírení libpcap-sze referuje sekcia 7.2.

[3]Berkeley Packet Filter

Kapitola 3

Paralelné spracovanie

Paralelizácia je spôsob výpočtu, ktorý pozostáva z viacero čiastkových výpočtov vykonávaných súbežne. Hlavným princípom je rozloženie celkovej úlohy na niekoľko podúloh, ktoré môžu byť spracované paralelne.

Paralelizácia sa používa na rôznych úrovniach začínajúc od bitovej paralelizácie (šírka registrov, zberníc). Nasleduje paralelizácia inštrukcií, ktorá môže byť interná - paralelné vykonávanie jednej inštrukcie, a medziinštrukčná – paralelné vykonávanie viacerých inštrukcií. Ďalšími stupňami sú paralelizácia dát a paralelizácia úloh.

Významným dôvodom pre používanie paralelných výpočtov je dnešný stav vývoja mikroprocesorov. Zvyšovanie frekvencie už nie je hlavným cieľom výrobcov pre zvýšenie celkového výkonu, pretože prináša vysoký príkon a nechcené zahrievanie. Hlavným trendom je zvyšovanie počtu jadier na čipe, čo poskytuje prostriedky pre paralelné spracovanie.

Paralelizácia systémov prináša nasledujúce výhody:

- rýchlejšie vyriešenie úlohy – potrebný čas na vyriešenie úlohy je menší.
- vyššia priepustnosť spracovania – vyšší počet transakcií za jednotku času.
- zložitejšie problémy, detailnejšie riešenia – za rovnaký čas.

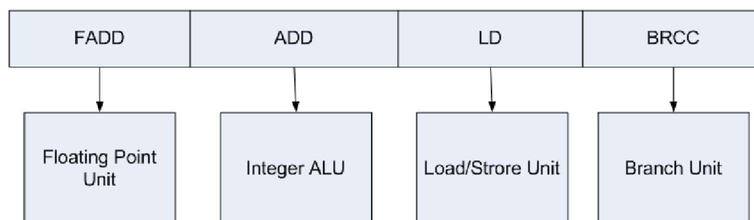
Vývoj softvéru, schopného pracovať paralelne, je náročnejší ako vývoj klasického sekvenčne pracujúceho softvéru. Paralelizácia okrem výhod naopľátku vyžaduje komunikáciu medzi súbežne vykonávanými úlohami a vzájomnú synchronizáciu (časové zosúladenie).

V tejto kapitole budú všeobecne popísané základné architektúry pre paralelné spracovanie a ich vlastnosti, princípy, techniky a prostriedky používané pre paralelné spracovanie a súvisiace problémy. Tento prehľad bude slúžiť v nasledujúcej kapitole pre návrh podpory paralelizácie pre softvérové aplikácie nad platformou NetCOPE.

3.1 Základné rozdelenie architektúr

Architektúry sa podľa prístupu k paralelnému spracovaniu delia do troch základných kategórií:

- dataflow – interpretuje graf toku dát.
- redukčný počítač – má stromovú architektúru, redukcia nahrádza časť výrazu jeho významom.
- Von Neumannovská architektúra – obsahuje spoločnú jednotku pre uchovávanie inštrukcií a dát a spracúvaciu jednotku. Túto architektúru využívajú dnešné osobné počítače a preto bude ďalší text na ňu zameraný.



Obrázok 3.1: Inštrukcia a príslušné jednotky procesoru architektúry VLIW

Pre základné rozdelenie Von Neumannovskej architektúry sa používa Flynova klasifikácia. Tá rozlišuje štyri typy podrobnejšie popísané v nasledujúcich sekciách.

3.1.1 SISD (Single Instruction, Single Data)

Jediný procesor spracúva jediný inštrukčný tok. Architektúra tohoto typu, ktorá využíva paralelizáciu na úrovni inštrukcií je VLIW[1]. Každá inštrukcia obsahuje operačné kódy pre súčasné riadenie všetkých jednotiek procesoru. Tieto inštrukcie sú generované kompilátorom s ohľadom na časovanie a vzájomné závislosti. Vďaka tomuto prístupu samotný hardvér nie je príliš komplexný, táto zložitosť je obsiahnutá v kompilátore. Využitie procesoru je závislé od stupňa paralelizácie každej inštrukcie.

Inštrukčná sada je pri architektúre VLIW volená tak, aby zachovala rovnováhu medzi jej zložitosťou a náročnosťou použitia. Príliš zložité inštrukčné sady nemusia byť vďaka zložitosti kompilátorom efektívne využívané. Ukážka inštrukcie a jednotiek procesoru vykonávajúcich jednotlivé zložky je na obrázku 3.1.

3.1.2 SIMD (Single Instruction, Multiple Data)

Táto architektúra, často označovaná ako **vektorový procesor,** využíva paralelizmu úrovne dát. Rovnaké operácie sa vykonávajú s väčším množstvom rôznych dát. Hlavné využitie má v spracovaní grafických a multimediálnych dát. Konkrétnym príkladom môže byť zmena jasu obrázka, všetkých jeho pixelov, kedy sa viac pixelov spracúva paralelne.

Zrejme najznámejším príkladom tejto architektúry je rozšírenie inštrukčnej sady x86 sadou MMX[2]. Do tejto kategórie patria aj DSP[3] – špecializované mikroprocesory pre spracovanie digitálnych signálov.

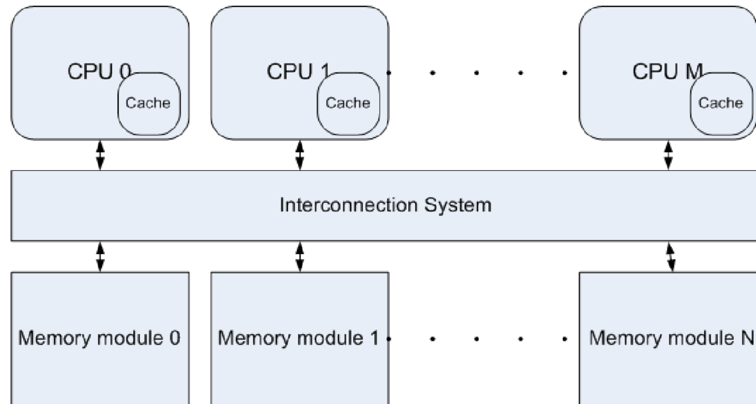
3.1.3 MISD (Multiple Instruction, Single Data)

Paralelizmus spočíva vo vykonávaní viacerých rôznych inštrukcií s rovnakými dátami. Do tejto kategórie patria systémy používajúce zreťazené spracovanie, tzv. pipeline. Riešené úlohy majú prúdový charakter – dáta sú postupne spracovávané nasledujúcimi procesormi. Pred MISD architektúrou sú často uprednostňované architektúry SIMD a MIMD pre ich lepšiu škálovateľnosť a iné vlastnosti.

3.1.4 MIMD (Multiple Instruction, Multiple Data)

Architektúra obsahuje viacero procesorov, ktoré pracujú asynchrónne a nezávisle na ostatných. Každý z nich môže vykonávať rôzne operácie s rôznymi dátami.

[1] Very Long Instruction Word
[2] MultiMedia eXtensions
[3] Data Signal Processors



Obrázok 3.2: Architektúra SMP

Táto skupina sa ďalej delí podľa Johnsonového rozšírenia Flynnovej klasifikácie. Toto rozšírenie delí MIMD architektúry podľa typu pamäte – globálna[4] alebo distribuovaná[5], a podľa spôsobu komunikácie – cez zdieľanú pamäť alebo prostredníctvom predávania správ.

3.1.4.1 Distribuovaná zdieľaná pamäť

Pamäť je fyzicky distribuovaná, logicky však zdieľaná, používa sa jeden adresový priestor. Prístup do pamäte netrvá vždy rovnaký čas, závisí od fyzického umiestnenia pamäťového miesta.

Tento mechanizmus môže byť implementovaný na rôznych úrovniach – na úrovni hardvéru (multiprocesory), operačného systému alebo na úrovni knižníc. Implementácia na úrovni operačného systému skrýva distribuovanosť a poskytuje transparentnosť pre vývojárov aplikácií, je však pomalá.

Koherencia cache je založená na domovských adresároch.

3.1.4.2 Distribuovaná pamäť

Architektúra vytvára viac-procesorový systém. Pamäťové priestory sú oddelené a nezávislé.

3.1.4.3 Globálna zdieľaná pamäť

Niekoľko procesorov je spojených so zdieľanou hlavnou pamäťou. Tento typ architektúry sa označuje aj ako SMP[6]. Zvyšovanie počtu jadier na čipoch je hlavným trendom súčasného vývoja procesorov. Najmodernejšie procesory však využívajú hybridný prístup – prístup do pamäte je typu NUMA, pamäťové moduly sú lokalizované bližšie k jednotlivým jadrám. Operačný systém však tieto typy procesorov chápe ako UMA SMP.

Bežne dostupné procesory osobných počítačov využívajú túto architektúru, preto bude ďalší text zameraný práve na ňu.

3.2 Pamäť cache

Systém pamätí počítača je hierarchický a heterogénny. So zvyšujúcou sa kapacitou, klesá rýchlosť odozvy a priepustnosť pamäťového média a tiež jeho cena. Medzi najpomalšie

[4]UMA - Uniform Memory Access
[5]NUMA - Non-uniform Memory Access
[6]Symmetric multiprocessing

patria externé pamäťové média, ako napríklad CD alebo magnetické pásky. Rýchlejší prístup k dátam ponúka pevný disk a na ďalšej úrovni stojí operačná pamäť.

Softvérové aplikácie sa vyznačujú dvomi dôležitými vlastnosťami. Prvou z nich je **časová lokalita**. Pokiaľ bola daná adresa momentálne používaná, je vysoká pravdepodobnosť, že bude používaná znova.

Druhou vlastnosťou je **priestorová lokalita.** Aplikácia zvyčajne v krátkom časovom okamihu pristupuje k susedným pamäťovým miestam.

Pre jednoduchú ilustráciu týchto vlastností môže slúžiť program spracúvajúci maticu čísel. Prvky sú spracúvané sekvenčne (priestorová lokalita) a každý z nich rovnakými inštrukciami (časová lokalita).

Pre využitie týchto vlastností pre zrýchlenie dátových prenosov slúžia pamäte cache. Fungujú ako vyrovnávacie pamäte medzi relatívne pomalou hlavnou pamäťou RAM a rýchlym procesorom. Systém pamätí cache býva hierarchický, používa sa viac úrovní s rastúcou rýchlosťou a klesajúcou kapacitou označovaných L1C až L4C.

Pamäť cache znižuje priemernú dobu prístupu do pamäte a znižuje zaťaženie zbernice.

Pri nesprávnom používaní pamäte cache môže dôjsť k nasledujúcim negatívnym javom:

- **falošné zdieľanie** – viacero procesorov pristupuje do rovnakého bloku pamäte, a aspoň jeden zapisuje. Spôsobuje to neustále výpadky.
- **cache thrashing** – rôzne dáta, ktoré sú často opakovane potrebné aplikáciou, sa mapujú do toho istého bloku cache. Vždy po načítaní dát do bloku cache, tieto prepíšu ďalšie potrebné dáta, čo spôsobuje výpadky. Tento problém sa rieši pridaním zarovnania dát, aby sa predišlo mapovaniu do rovnakého bloku cache.

Pre spätný zápis dát zmenených v pamäti cache sa používajú techniky **Write through**, zmenené dáta sa ihneď zapisujú do hlavnej pamäte, a **Write back**, dáta sa zapisujú do hlavnej pamäte až pri ich nahradení.

3.2.1 Koherencia

V paralelných systémoch s viacerými procesormi vlastní každý procesor (jadro) svoju pamäť cache, Pokiaľ viacero procesorov pracuje s rovnakým dátovým miestom, a jeden z nich mení jeho hodnotu, dáta v ostatných pamätiach cache zostanú neaktuálne a celý systém sa dostáva do nekonzistentného stavu. To znamená, že v systéme existuje viac ako jediná verzia dát pre konkrétnu adresu. Tieto problémy rieši koherencia pamäte cache.

Pre koherenciu pamäte cache sa využívajú rôzne protokoly podľa použitej architektúry.

3.3 Synchronizačné nástroje

Slúžia pre zaistenie časového zosúladenia medzi súbežne vykonávanými procesmi. Základnou úlohou je zabezpečenie **vzájomného vylúčenia**[7]**,** ktoré zabezpečí prístup ku kódu len jednému z procesov. Kód, ktorého vykonávanie je vzájomne vylúčené sa nazýva **kritická sekcia.**

Implementácia vzájomného vylúčenia musí spĺňať tieto podmienky:

- bezpečnosť – zaručenie vylúčenia
- živosť – nedochádza k uviaznutiu, blokovaniu ani starnutiu

[7] anglicky mutual exclusion

Uviaznutie označované aj ako „deadlock" je stav vzájomného zablokovania. Procesy navzájom čakajú na stav, ktorý by mal nastať pokiaľ by jeden z nich pokračoval, avšak to sa už nestane.

Blokovanie kritická sekcia je voľná, avšak proces čaká na vstup do nej. Proces je teda blokovaný iným procesom. Zabraňuje prístupu do kritickej sekcie v konečnom čase.

Starnutie proces testuje a čaká na stav, ktorý nemusí nikdy nastať.

Podmienky vzájomného vylúčenia splňuje Petersonov a Dekkerov algoritmus pre 2 procesy. Vzájomné vylúčenie obecne N procesov zaručuje Dijskrov a „bakery" algoritmus.

Medzi základné synchronizačné nástroje patrí **spinlock** (využíva aktívne čakanie), **binárny semafor (tzv. mutex), obecný semafor, monitor** a **bariéra**. Pre ich realizáciu slúži hardvérová podpora – inštrukcie typu **test and set, swap, compare and swap**.

3.3.1 Race conditions

Týmto pojmom sa označujú časovo závislé chyby. Výsledok procesu je kriticky závislí na poradí vykonávania úloh.

3.3.2 Determinizmus

Pokiaľ bude výsledok paralelného behu paralelného systému vždy rovnaký, nazývame systém deterministickým. Výsledok behu nedeterministického systému je závisí na poradí vykonávania úloh.

3.4 Paralelizmus z pohľadu operačného systému

3.4.1 Multitasking

Väčšina moderných operačných systémov je schopná vykonávať niekoľko úloh pseudoparalelne. Princípom je abstrakcia virtuálneho procesoru, prideľovanie procesoru jednotlivým procesom v časových kvantách a teda jednotlivé procesy sa v behu striedajú. Proces nevie o tom, že beží pseudoparalelne. Pojem multitasking sa označuje aj pojmom multiprocessing.

Multitasking môže byť samozrejme realizovaný aj na systémoch SMP, kde sa už však jedná aj o skutočne paralelné spracovanie.

Rozlišujeme nepreemptívny a preemptívny multitasking. Proces s aktuálne prideleným CPU sa sám procesoru vzdá a umožní tak pridelenie procesoru ďalšiemu procesu pri nepreemptívnom multitaskingu. Pri druhom type proces nemôže ovplyvniť prepnutie kontextu. Samotné prepínanie kontextu je realizované tzv. prideľovačom – „dispatcherom".

3.4.1.1 Scheduling

„Scheduling" alebo plánovanie je zodpovedné za výber procesov pre pridelenie procesoru. Plánovač je charakterizovaný intervalom rozhodovania, prioritnou funkciou a výberovým pravidlom.

Intervaly rozhodovania určujú čas aktivity plánovača, medzi týmito aktivitami nie je možná zmena pridelenia procesoru.

Prioritná funkcia určuje prioritu procesu na základe časových údajov – spotrebovaný čas, čas v systéme, čas čakania na procesor a podľa ďalších údajov.

Výberová funkcia slúži pre výber procesu pri existencii viacerých procesov s rovnakou prioritou. Využívajú sa tieto prístupy založené na časových údajoch – spotrebovaný čas procesoru, čas strávený v systéme a celková doba trvania procesu:

- FIFO
- LIFO
- Round Robin
- Shortest Job Next
- Multilevel Feedback Queue

3.4.2 Procesy a vlákna

Proces je samostatne bežiaci program vo vlastnom adresovom priestore. Stav procesu je definovaný stavom registrov procesoru, stavom dát, zásobníku, systémových prostriedkov a stavom vykonávania.

Vlákna slúžia pre využitie viacero procesorov a teda pre paralelné vykonávanie jedného programu. Adresový priestor vlákien jedného procesu je spoločný, taktiež ako systémové prostriedky. Kódový a dátový segment je zdieľaný, naopak každé vlákno má vlastný stav registrov, zásobník a stav vykonávania.

Vlákna sú jednotkou prideľovania procesoru. Prepínanie kontextu v rámci vlákien jedného procesu je rádovo rýchlejšie než prepínanie kontextu medzi viacerými procesmi. Zmena TLB tabuľky jednotky MMU[8], nie je nutná, nakoľko adresový priestor sa nemení, taktiež ako uvoľnenie pamäte cache.

Implementácie vlákien sú realizované na rôznych úrovniach.

- úroveň jadra (N:N) – užívateľské vlákna sú reprezentované v jadre.
- úroveň knižníc (N:1) – jadro o vláknach nevie, sú spravované kódom knižníc. Tento prístup však neumožňuje paralelné spracovanie, nakoľko iba jediné vlákno môže byť spracovávané.
- hybridný prístup (N:M) – kombinácia hore-uvedených prístupov pre dosiahnutie čo najnižšej réžie operačného systému a tým najefektívnejšieho paralelného spracovania.

3.5 Paralelizmus aplikácií na systémoch so zdieľanou pamäťou

Všeobecne pre paralelné programovanie existujú nasledujúce možnosti:

- automatická paralelizácia – automatická transformácia sekvenčného kódu na kód schopný pracovať paralelne kompilátorom. Zatiaľ tento prístup, ktorý by bol ideálny, nie je veľmi úspešný. Pre zlepšenie tohoto prístupu je potrebná komplexná analýza programu. Automatická paralelizácia je zameraná na paralelné prevádzanie slučiek, ako základných štruktúr programovacích jazykov, v ktorých aplikácie trávia najviac času.
- rozšírenie sekvenčného jazyka o príkazy pre synchronizáciu a paralelné spracovanie. Príkladom je UPC[9].

[8]Memory Management Unit

[9]Unified Parallel C

- programovanie s vláknami – pthreads (popísané v 3.5.2), Java threads, Windows fibers, Windows threads. Pomerne nízke rozhranie je poskytované knižnicami pre prácu s vláknami.
- vyššie rozhranie pre paralelné programovanie. Príkladom je OpenMP (popísané v 3.5.3).

3.5.1 Metodika tvorby paralelných programov

Metodika umožňuje definovaným postupom vytvoriť paralelne pracujúci model danej úlohy. Metodika obsahuje tieto hlavné body:

3.5.1.1 Odhalenie súbežností

Analyzovaná úloha je dekomponovaná a následne sú pre dekomponované časti vyhodnotené závislosti. Bližšie budú popísané jednotlivé časti tohoto procesu:

Dekompozícia Pre dekompozíciu problému sa používajú dva základné prístupy – dekompozícia funkčná a dekompozícia dátová.

Dekompozícia výpočtu na podúlohy umožňuje podúlohy riešiť samostatne a nezávislo. Jednotlivé podúlohy pre vyriešenie celkovej úlohy potrebujú komunikovať, čo prináša réžiu výpočtu.

Počet podúloh by mal byť rovnaký alebo väčší ako počet procesorov. Podúlohy by mali mať približne rovnakú náročnosť. Niekedy sa dá úloha rozdeliť parametrizovateľne na počet podúloh a ich náročnosť.

Dátová dekompozícia je vhodným prístupom pokiaľ sa najväčšia časť výpočtu týka veľkej dátovej štruktúry a časti výpočtu sa aplikujú nezávisle na časti tejto štruktúry.

Analýza závislostí Prvou fázou analýzy závislostí je zoskupenie úloh do logických skupín s rovnakými obmedzeniami. Závislosťou je potreba výsledkov z inej skupiny, alebo súčasné vykonávanie úloh rôznych skupín. Na poradí behu úloh v niektorých skupinách nemusí záležať.

Následne musia byť úlohy zoradené tak, aby boli splnené obmedzenia a dodržané Bernsteinove podmienky paralelného behu úloh.

Poslednou fázou analýzy závislostí je vyriešenie zdieľania dát. Dáta môžu byť lokálne alebo zdieľané. Podľa prístupu k zdieľaným dátam je nutné použiť synchronizačné prostriedky. Pre prístup k lokálnym dátam slúži komunikácia.

Vyhodnotenie Vypracovaný návrh riešenia problému z predchádzajúcich krokov musí byť ohodnotený. Hodnotí vhodnosť pre danú architektúru a tiež účinnosť závislá na vyvážení procesorov a réžii.

3.5.1.2 Štrukturálne vzory algoritmov

Vzory paralelne pracujúcich algoritmov môžu byť použité pre implementáciu konkrétneho algoritmu. V nasledujúcej podsekcii si uvedieme základné rozdelenie týchto vzorov a stručne popíšeme niektoré z nich.

- Štruktúra založená na toku dát

- reťazené spracovanie – vhodné použiť pre statický a pravidelný tok dát. Rovnaké dáta sú spracúvané postupne vo viacerých stupňoch. Najlepšej účinnosti sa dosahuje pri rovnakej časovej náročnosti v jednotlivých stupňoch.
  - koordinácia udalosťami – tok dát je nepravidelný a môže byť viac-smerný. Tok dát je definovaný udalosťami, keď jedna úloha generuje udalosť a druhá ju spracováva.

- Štruktúra založená na delení úloh
  - funkčný paralelizmus úloh – úloha je rozdelená na podúlohy, ktoré môžu bežať paralelne. Pokiaľ je časová zložitosť jednotlivých úloh dopredu známa, je možné podľa nej priradiť úlohy ku konkrétnym procesorom pre vyváženie záťaže ešte pred samotným výpočtom. Pokiaľ túto náročnosť dopredu nepoznáme je vhodné použiť dynamické prideľovanie úloh procesorom za behu programu.
  - rozdeľ a panuj – problém je rozdelený na podproblémy, ktoré môžu byť riešené paralelne. Typicky je toto delenie prevádzané rekurzívne pre získanie dostatočného počtu úloh.

- Štruktúra založená na delení dát
  - geometrická dekompozícia dát – typicky sa dekomponuje matica buď podľa stĺpcov alebo podľa blokov. Pre výpočty je často nutné pracovať s hraničnými dátami z nelokálnych častí. Komunikácia je potrebná pre prístup k týmto dátam, čo predstavuje réžiu výpočtu. Pre obmedzenie réžie komunikácie sa používa technika prekrytia komunikácie a výpočtu.
  - rekurzívne dáta – je technika práce s natívne sekvenčnými štruktúrami. Celkový výpočet je síce náročnejší ako jeho sekvenčný variant, avšak vďaka paralelnému prevádzaniu prináša zrýchlenie.

3.5.1.3 Podporné štruktúry

Pre realizáciu paralelných algoritmov existujú podporné programové a dátové štruktúry.

Programové štruktúry

- SPMD – single program, multiple data - dáta sú rovnomerne rozdelené. Každé vlákno má jedinečné ID. Dáta sú spracúvané rovnakým programom, chovanie je však diferencované podľa ID vlákna.
- Master/Worker – vzor pre úlohy s nezávislými podúlohami. Proces master udržuje úlohy čakajúce na spracovanie a prideľuje ich pracujúcim vláknam (Workers).
- Fork/Join – úloha vytvára nové úlohy dynamicky a čaká kým nové úlohy dokončia a potom pokračuje.

Dátové štruktúry

- zdieľané dáta – zápisy do zdieľaných dát musia byť synchronizované. Zlepšenie výkonnosti prinesie optimalizácia kritickej sekcie na najmenšiu možnú veľkosť a rozdelenie operácií do rôznych kritických sekcií, pokiaľ je to možné.
- zdieľaná fronta – je to abstraktný dátový typ obsahujúci zoradené objekty rovnaké typu. Existuje v blokujúcej a neblokujúcej variante.

- distribuované polia – delenie matíc podľa viacerých stĺpcov, cyklicky podľa jediného stĺpca, cyklicky po blokoch stĺpcov, na bloky a cyklicky po blokoch.

3.5.2 Pthreads

POSIX implementácia vlákien. Pthreads sú implementované na úrovni knižnice, ktorá je prenositeľná a implementovaná pre systémy UNIX a tiež Windows.

Pthreads poskytujú nízkoúrovňové rozhranie pre prácu s vláknami – ich vytváranie, správu a vzájomnú synchronizáciu.

3.5.3 OpenMP

OpenMP[10] poskytuje aplikačne programové rozhranie pre vývoj paralelných programov na architektúrach so zdieľanou pamäťou. Hlavným cieľom je poskytnutie vysokej škálovateľnosti prostredníctvom rozhrania vyššej úrovne. OpenMP je definované skupinou hlavných výrobcov hardvéru a softvéru. Podporované jazyky sú C, C++ a Fortran.

Použitie rozhrania OpenMP v kóde zahŕňa používanie direktív prekladača (#pragma omp), knižničných funkcií a premenných prostredia. Paralelizácia sekvenčných programov prebieha inkrementálne. Preklad programov používajúcich toto rozhranie si vyžaduje použitie OpenMP-kompatibilného prekladača. Vlákna vytvárané rozhraním OpenMP sú abstraktné, mapujú sa prostriedky podľa konkrétnej použite platformy.

3.5.4 Reentrantný a „thread safe”[11] kód

V multivláknových aplikáciách rovnaké časti kódu a prostriedky môžu byť používané v rovnakom čase viacerými vláknami. Pre zachovanie integrity prostriedkov musí byť kód reentrantný a vláknovo bezpečný.

Reentrantnosť a vláknová bezpečnosť sú dva rozdielne pojmy a podmienky. Program môže spĺňať obe, jednu z nich, alebo žiadnu. Niekedy je však nemožné spraviť nereentrantú funkciu vláknovo bezpečnú.

Reentrantná funkcia neudržiava statické dáta medzi volaniami a ani nevracia ukazatele na statické dáta. Všetky dáta musia byť poskytnuté volajúcim funkcie a funkcia samotná nemôže volať ďalšie nereentrantné funkcie. Implementácia reentrantnej funkcie vyžaduje zmenu rozhrania oproti jej nereentrantnému ekvivalentu.

Vyhnutie sa vracaniu ukazateľa na statické dáta môže byť realizované dynamickou alokáciou dát, v tomto prípade však je volajúci zodpovedný za ich uvoľnenie. Tento spôsob dokonca zachováva rozhranie, avšak existujúce aplikácie musia byť ošetrené, aby sa zabránilo tzv. „memory leakom” teda strácaniu alokovanej pamäte. Ďalším spôsobom riešenia tohoto problému je poskytnutie pamäte volajúcim, tento variant si však už zmenu rozhrania vyžiada.

Funkcia nesmie uchovávať statické dáta medzi nasledujúcimi volaniami, pretože môže byť volaná inými vláknami. Pokiaľ potrebuje funkcia nejaké dáta udržiavať potrebuje, musí to zaistiť volajúci vhodnými návratovými parametrami.

Pre zaistenie používania reentrantných funkcií v jazyku C musia byť použité makrá #define _POSIX_C_SOURCE 199506L a #define _REENTRANT.

[10] MP značí multi programming
[11] vláknovo bezpečný

„Thread safe” funkcia ochraňuje zdieľané dáta a prostriedky pred súbežným prístupom synchronizáciou prístupu. Na rozdiel od reentrantnosti nevyžaduje zmenu rozhrania, ale len implementačné zmeny. Keďže lokálne premenné a argumenty funkcie sú uložené na zásobníku, ktorý má každé vlákno vlastný, prístup k týmto prostriedkom je bezpečný. Vláknovo bezpečné nie je používanie globálnych dát, prístup k nim musí byť synchronizovaný.

Pre zabezpečenie vláknovej bezpečnosti je možné použiť dočasné riešenia, ktoré však vedú k veľmi nízkemu výkonu. Tieto možnosti však môžu byť úspešne využité pri ladení, alebo pri čakaní na vydanie „thread safe” kódu. Princíp je veľmi jednoduchý, prístup k celému kódu, alebo k jeho rôznym nezávislým častiam je synchronizovaný.

Kapitola 4

Paralelné spracovanie softvérových aplikácií nad platformou NetCOPE

V tejto kapitole budú preskúmané možnosti paralelného spracovania typických sieťových aplikácií. Na základe tejto analýzy budú vytvorené modely vhodné pre podporu tejto paralelizácie pre softvérové rozhranie platformy NetCOPE.

4.1 Typické sieťové (monitorovacie) aplikácie a možnosti ich paralelizácie

Sieťový tok Sieťový tok je dôležitý pojem v oblasti sietí a je definovaný ako obojsmerná postupnosť IP paketov, ktoré prejdú zariadením za určitý časový interval a obsahujú rovnaké adresy IP, porty, rovnaký transportný protokol a IP typ služby. Keďže sieťový tok predstavuje oddelenú interakciu systémov cez počítačovú sieť, rôzne sieťové toky sú nezávislé. Táto nezávislosť prináša možnosti paralelného spracovania sieťových tokov.

Formát dát Základným faktom, ktorý musíme brať do úvahy pri analýze možností paralelizácie, je formát dát poskytovaných rozhraniami platformy NetCOPE. Rozhranie knižnice PCAP pracuje s dátami na linkovej vrstve. Rozhranie szedata2 naproti tomu umožňuje prácu s dátami v aplikačne špecifickom formáte, závislom od implementácie akceleračného jadra NetCOPE aplikácie.

Akceleračné jadro je vhodným miestom pre predprípravu dát pre dosiahnutie lepších možností paralelizácie. Pre prácu s týmto druhom dát bude musieť byť použité rozhranie szedata2. Príkladom nech je IP core, modul riešiaci fragmentáciu IP paketov, vhodný pre použitie v systémoch IDS popísaných v sekcii 4.1.2. Ďalším modulom by mohol byť komponent oddeľujúci jednotlivé sieťové toky.

4.1.1 Firewall

Firewall je základným bezpečnostným prostriedkom. Kontroluje sieťovú premávku medzi zónami s rôznym stupňom dôvery (Internet – LAN).

Firewall koná podľa predom definovaných pravidiel, ktoré rozhodujú na základe analýzy hlavičky IP datagramov – zdrojovej a cieľovej IP adresy, čísel portov a použitého protokolu. Pracuje teda na sieťovej vrstve. S paketmi spĺňajúcimi určité podmienku na základe analýzy je vykonaná akcia. Medzi akcie patrí príjem paketu, jeho zahodenie alebo preposlanie.

Štandardnou vlastnosťou moderných firewallov je stavové filtrovanie, keď firewall udržuje záznamy o stave všetkých relácií a filtrovacie pravidlá sa používajú na základe stavu konkrétnej relácie.

Pre ilustráciu popíšem pravidlo firewallu „iptables”, ktorý je štandardnou súčasťou systému Linux od verzie 2.4.

```
/sbin/iptables -A INPUT -p tcp -i eth0 --dport 21 -j ACCEPT
```

Všeobecne pravidlo pozostáva z troch častí. Prvá značí udalosť, pri ktorej sa má dané pravidlo vyhodnotiť. Firewall iptables pracuje so štruktúrami nazývanými „chains”, tieto obsahujú súbory pravidiel pre určité udalosti. Pre jednoduchosť vysvetlenia parameter *-A INPUT* znamená, že toto pravidlo sa pridá a bude vyhodnocovať pri príjme paketu.

Časť druhá *-p tcp -i eth0 –dport 21* obsahuje podmienku, ktorá je kontrolovaná. V tomto konkrétnom pravidle je kontrolovaný typ protokolu TCP, rozhranie eth0, na ktorom bol paket detegovaný a číslo cieľového portu 21. Jedná sa pravdepodobne o pravidlo príjmajúce protokol FTP.

Posledná časť pravidla *-j ACCEPT* značí akciu, ktorá sa má pri úspešnom vyhodnotení vykonať, v tomto prípade akcia značí úspešný príjem paketu.

Pravidlá sú na sebe typicky závislé. Paket môže spĺňať podmienky viacerých pravidiel, avšak výsledná akcia je určená pravidlom s najvyššou prioritou.

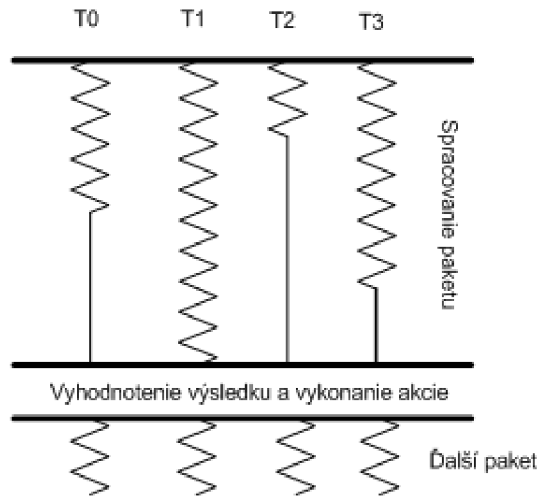
Ďalšia závislosť existuje v poradí príchodu paketov v prípade nasadenia stavového firewallu. Pakety nesmú byť spracovávané mimo poradia, pretože by nereflektovali stav relácií a teda poradie spracovania paketov musí byť zachované.

Možnosti paralelizácie

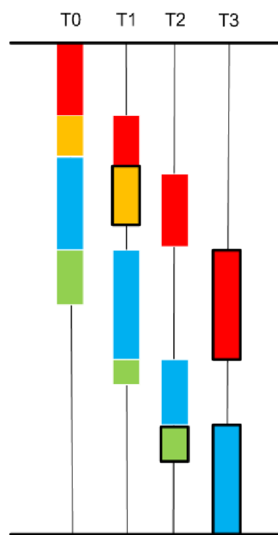
1. MISD model 1 – paralelizácia spočíva v rozdelení pravidiel na podmnožiny pre spracovanie jednotlivými procesmi alebo vláknami. Keďže sú pravidlá na sebe závislé, rozdelenie na výpočtovo ekvivalentne náročné a nezávislé podmnožiny nemusí byť možné. O výslednej akcii musí teda byť rozhodnuté po skončení výpočtov všetkých pracujúcich vlákien a vyhodnotení ich čiastkových výsledkov. Tento model je zobrazený na obrázku 4.1. Dĺžka spracovania celého paketu je závislá na najdlhšie pracujúcom vlákne. Ďalší paket sa začne spracovávať po vyhodnotení celkového výsledku a vykonaní príslušnej akcie.

2. MISD model 2 (pipeline) – rozdelíme súbor pravidiel zoradený podľa priority na časti. Samotné dáta budú spracovávané zreťazene – najprv vláknom s pravidlami najvyššej priority. Pokiaľ prvé vlákno nevykoná žiadnu akciu, na rovnakých dátach začne pracovať vlákno ďalšie s pravidlami s menšou prioritou. Súčasne prvé vlákno môže začať spracúvať paket ďalší. Výpočtová náročnosť jednotlivých podmnožín pravidiel sa môže riadiť podľa faktu, že najčastejšie budú kontrolované pravidlá z prvej skupiny, a preto by mala byť výpočtovo menej náročná ako ostatné. Najmenej často sa budú vykonávať pravidlá z poslednej skupiny a preto môže byť výpočtovo najnáročnejšia. Model práce štyroch vlákien a spracovanie štyroch paketov je znázornený na obrázku 4.2. Miesto ukončenia spracúvania paketu je zvýraznené čiernym obrysom. Bohužiaľ ani tento model nezaručuje sekvenčné spracovanie paketov. Na obrázku môžeme vidieť, že paket štvrtý v poradí (zelený) je spracovaný skôr vláknom T2, ako tretí paket (modrý), spracovaný vláknom T3.

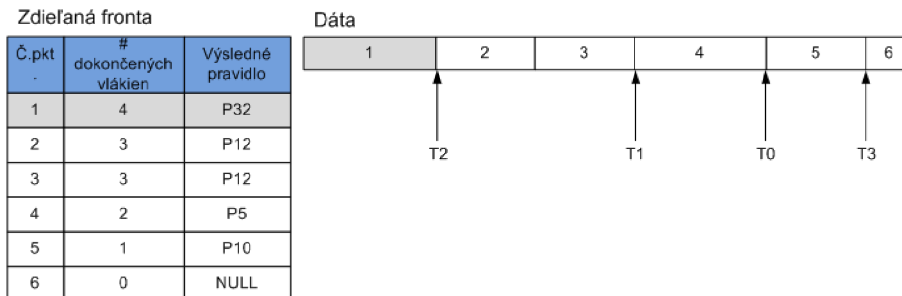
3. MISD model 3 (so zdieľanou frontou) – vylepšenie MISD modelu 1, pridanie zdieľanej fronty a spracovávanie bloku dát. Zdieľaná fronta bude nahrádzať synchronizáciu



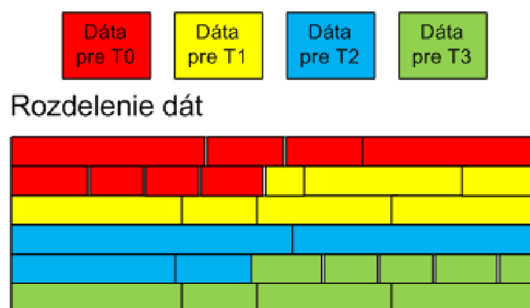
Obrázok 4.1: Model paralelnej práce firewallu - MISD 1



Obrázok 4.2: Model paralelnej práce firewallu - MISD 2



Obrázok 4.3: Model paralelnej práce firewallu so zdieľanou frontou MISD 3. Rámec 1 je už z fronty vyradený.



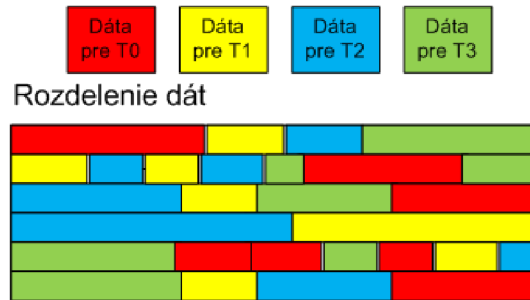
Obrázok 4.4: Rozdelenie dát pri modele SPMD 1

medzi vláknami po každom spracovanom rámci, fázu vyhodnotenia výsledku, vlákna sa môžu predbiehať. Nakoľko spracovanie rôznych paketov trvá rôznu dobu, toto chovanie nie je prekážkou. Každá položka zdieľanej fronty reprezentuje informácie o jednom rámci, o počte dokončených vlákien nad daným rámcom a taktiež obsahuje výsledné pravidlo. Prístup do tejto štruktúry musí byť synchronizovaný, ako značí jej názov, výsledné pravidlo bude vždy pravidlo s najvyššou prioritou. Pokiaľ aktuálne dokončené vlákno priradilo pravidlo s nižšou prioritou ako uložené pravidlo, len inkrementuje čítač vlákien v štruktúre. Pokiaľ hodnota čítača dosiahla počtu všetkých vlákien, je položka z fronty vyradená. Ak je získané pravidlo s vyššou prioritou, alebo ak v štruktúre nie je uložené zatiaľ žiadne pravidlo, aktualizuje výsledné pravidlo štruktúry. Princíp tohoto modelu je zobrazený na obrázku 4.3 .

4. SPMD model 1 (blokový) – každé vlákno kontroluje všetky pravidlá firewallu. Dáta sú rozdelené po blokoch medzi rôzne vlákna a jeden paket je spracovávaný vždy len jedným z vlákien. Tento model porušuje poradie spracovania.

5. SPMD model 2 (po jednom) – v princípe rovnaký ako model 4). Rozdiel je len v prideľovaní dát, každý paket sa prideľuje zvlášť voľnému vláknu. Mohlo by sa zdať, že tento model zachováva poradie spracovania, avšak kvôli rôznej dĺžke spracovania paketov ani tu nie je táto vlastnosť zachovaná. Keďže dĺžka spracovania paketu je rôzna, tento prístup umožňuje lepšie vyváženie záťaže pre jednotlivé vlákna ako model 4).

6. Funkčná dekompozícia – hľadanie výsledného pravidla a vykonanie akcie sú činnosti, ktoré môžu byť vykonávané paralelne. Po získaní výsledného pravidla akcia môže byť realizovaná nezávislým vláknom.



Obrázok 4.5: Rozdelenie dát pri modele SPMD 2

4.1.2 IDS

Intrusion Detection System (systém detekcie narušení) získava informácie o sledovanom systéme. Jeho účelom je sledovať bezpečnostné prieniky, pokusy o ne, zistiť systémové zraniteľnosti, ktoré by k prienikom mohli viesť. V súvislosti s týmito systémami sa vyskytujú pojmy „false negative“, značí situáciu keď' IDS nezachytí útok alebo pokus oň, a „false positive“, značí situáciu pokiaľ IDS označí za útok neškodnú akciu.

Podľa umiestnenia zdrojov informácii pre IDS zariadenie rozlišujeme „host based” (IDS pracujúce a zamerané na hostiteľský systém) a „network based” (umiestnené v počítačovej sieti, spracúvajúce sieťovú premávku). Ďalší text o IDS systémoch bude zameraný len na „network based” IDS systémy.

4.1.2.1 Knowledge vs. behavior based IDS

Podľa základného rozdelenia založeného na detekčnej metóde rozlišujeme dva komplementárne smery IDS zariadení – založené na správaní („behavior based”) a založené na báze znalostí o útokoch („knowledge based”).

Knowledge based IDS: Systémy tohoto druhu sú založené na dlhodobo akumulovanej báze znalostí o útokoch a systémových zraniteľnostiach. IDS využíva tieto informácie pre detegovanie týchto útokov a zneužitie zraniteľností. Je tu uplatnený rovnaký princíp ako v antivírových programoch – v dátach sú vyhľadávané vzorky špecifické pre určité víry, tiež nazývané „signatúry”. Zariadenie vyhľadáva vzorky dát špecifické pre útoky z bázy znalostí. Akcia, ktorá nie je považovaná za útok, je považovaná za neškodnú. Báza znalostí sa musí pravidelne aktualizovať.

Behavior based IDS: Hlavnou myšlienkou týchto systémov je odhalenie útoku pozorovaním odchýlky od bežného očakávaného správania. Aktivita systému sa porovnáva s referenčným modelom, ktorý musí byť v počiatku vytvorený. Zväčša býva založený na štatistikách – priemerných hodnotách dôležitých ukazateľov a ich odchýlok (počet otvorení významných súborov, počet spustení príkazu, využitie zdrojov). Po vytvorení modelu (databáze) systém sleduje aktivitu systému, porovnáva výsledky s referenčným modelom a model sa vyvíja. Pokiaľ deteguje odchýlky, vyskytla sa anomália. Všetko čo sa vymyká vopred naučenému chovaniu je považované za odchýlku, a teda generuje poplach.

V prostredí sieťových IDS systémov medzi sledované metriky patria informácie o sieťových tokoch. Podrobnejšie o sieťových tokoch pojednáva nasledujúca sekcia.

Snort Je štandardom v oblasti IDS systémov, patrí do kategórií knowledge based ale môže byť rozšírený i o modul detekcie anomálií. Snort spracúva sieťovú premávku, je to teda „network based" IDS.

Snort využíva zásuvné moduly tzv. preprocesory, ktoré spracúvajú prijatý sieťový tok pred samotnou analýzou. Jedným z najvýznamnejších modulov je modul riešiaci fragmentáciu. Fragmentácia je často využívaná útočníkmi pre rozloženie signatúr do viacerých paketov pre vyhnutie sa odhaleniu.

Výstup systému je spracovaný výstupnými pluginmi, ktoré riešia formát výstupných dát a následne prechádza logovacím a poplachovým zariadením.

Špecifikácia signatúry je vyjadrená snort pravidlom, ktoré si podrobnejšie popíšeme na nasledujúcom príklade:

```
alert tcp 192.168.0.0/24 any -> 213.219.122.11/32 80
(msg: 'ATTACK RESPONSE Zone-H.org defacement notification';
flow: established, to server; content:'notify '; nocase;
classtype: trojan-activity; sid: 2001616;
pcre:'/notify (defacer|domain|hackmode|reason)=/i'; rev:6; )
```

Pravidlo sa skladá z hlavičky a tela. Hlavička pravidla, podobne ako vo firewalle, podmieňuje použitie pravidla podľa IP adries, portov a transportného protokolu a určuje akciu, ktorá sa má vykonať v prípade potvrdenia pravidla. V ukážkovom pravidle sa pravidlo použije pri TCP toku z podsiete 192.168.0 na akomkoľvek porte na adresu 213.219.122.11 na port 80.

V tele pravidla sa nachádzajú informácie o útoku. Hlavným údajom je položka pcre, ktorá definuje regulárny výraz vo formáte PCRE [1]. Vyhľadávanie podľa regulárnych výrazov, daných súborom pravidiel, je hlavnou úlohou IDS systémov.

Možnosti paralelizácie

1. MISD model – princíp je rovnaký ako u modelu firewallu 1). Paralelizácia spočíva v rozdelení súboru pravidiel na podmnožiny vykonávané jednotlivými vláknami. Kedže pravidlá IDS systémov sú vzájomne nezávislé, nemusí sa fáza vyhodnotenia celkového výsledku vykonávať. Výsledná rýchlosť spracovania je potom daná najpomalším z vlákien.

2. SPMD model – princíp je rovnaký ako u modelov firewallu 4) a 5). Dáta môžu byť rozdelené jednotlivým vláknam buď po blokoch, alebo každý paket zvlášť. Tieto modely nezachovávajú poradie spracovania paketov, čo však pri IDS systémoch nie je nevyhnutné.

4.1.3 Monitorovanie sieťových tokov

Pasívne monitorovanie sieťových tokov zhromažďuje štatistické informácie o sieťových tokoch. Tieto informácie môžu slúžiť pre účtovanie (napríklad meranie počtu prenesených dát klienta), analýzu správania sa siete alebo pre detekciu anomálií (ako bolo spomenuté v sekcii o zariadeniach IDS).

Informácie o existujúcich tokoch sú udržiavané v tabuľke sieťových tokov. Kľúčom tabuľky sú údaje definujúce tok, ostatné informácie sú vypočítavané dynamicky vždy pri obdržaní ďalšieho paketu toku. Typickými informáciami záznamu sú použité sieťové rozhrania, časové značky začiatku a konca toku, počet bajtov a počet paketov toku, pri protokole TCP pozorované flagy tohoto protokolu. Tieto informácie sú závislé podľa použitého protokolu, najznámejšími sú netflow v1 až v9 a IPFIX.

[1] Perl Compatible Regular Expression – využíva sa knižnica libpcre

Možnosti paralelizácie

1. SPMD – dáta sú vláknam rozdelené buď po blokoch, alebo po jednom pakete. Rôzne vlákna pracujú s tabuľkou sieťových tokov a aktualizujú jej stav. Prístup k záznamom musí byť synchronizovaný. Tento prístup nezachováva poradie spracovania paketov a preto pravdepodobne nemôže byť reálne použitý.
2. MISD – všetky vlákna spracúvajú rovnaké dáta. Každému záznamu v tabuľke tokov je priradené obslužné vlákno. Iba obslužné vlákno spracúva rámec prislúchajúci priradenému toku. Pri spracovaní každého paketu musí každé vlákno najprv zistiť, či je jeho obslužným vláknom. Tento prístup zachováva poradie spracovania tokov avšak vyžaduje veľmi vysokú réžiu pri kontrole obslužného vlákna. Prístup by bol výhodný, pokiaľ by bola výpočtová náročnosť kontroly obslužného vlákna ďaleko menšia ako výpočtová náročnosť spracovania toku. Ďalšou výzvou tohoto prístupu je rovnomerné vyváženie tokov pre jednotlivé obslužné vlákna. Z tohoto dôvodu by v systéme musel existovať pomerne zložitý mechanizmus, realizovaný centralizovane arbítrom, alebo formou vzájomnej dohody, pre kontrolu nových tokov a určenie obslužných vlákien, ktorý by vyžadoval množstvo synchronizácie a bol by pravdepodobne úzkym hrdlom modelu. Tento mechanizmus v podobe arbítra by mohol byť vykonávaný samostatným vláknom.

4.1.4 UTM

UTM [2] zariadenia slúžia pre komplexnú ochranu počítačových sietí. Riešenia v sebe zahŕňajú zariadenia ako firewall, IDS/IPS, antispyware, antivírus, detektor nevyžiadanej pošty, filtrovanie obsahu, NAT a VPN. V poslednom čase nadobúdajú stále viac na význame.

Možnosti paralelizácie Možnosti paralelizácie budú podané z pohľadu celkového riešenia, nie z pohľadu jednotlivých zariadení UTM.

1. MISD model 1 – viacero aplikácií, zložiek UTM, spracúva rovnaké dáta.
2. MISD model 2 (pipeline) – viacero aplikácií, zložiek UTM, spracúva dáta zreťazene.
3. SPMD model – princíp je rovnaký ako v predchádzajúcich zariadeniach.

4.1.5 Router

Smerovanie v paketových sieťach je proces výberu cesty pre doručenie dát na ich ceste zo zdroja k cieľu. Smerovanie spracúva dáta na sieťovej úrovni. Smerovač udržuje smerovaciu tabuľku a podľa nej a cieľovej IP adresy paketu vyberá cestu. Smerovacie tabuľky sú zvyčajne vytvárané dynamicky a na tento účel slúžia protokoly založené na dvoch základných algoritmoch – „distance vector” a „link state”.

Možnosti paralelizácie

1. Funkčná dekompozícia – úloha udržovania smerovacej tabuľky a samotné smerovanie podľa tabuľky sú nezávislé a preto je možné tieto úlohy vykonávať samostatne.

[2] Unified Threat Management

2. SPMD model – viacero vlákien spracúva rôzne pakety. Pakety môžu byť prideľované po blokoch, alebo každý zvlášť. Tento model nezachováva poradie spracovania paketov, preto by bolo vhodné paralelizovať dáta na úrovni sieťových tokov pričom o jeden tok by sa staralo vždy len jedno vlákno a tým by zaistilo zachovanie poradia spracovania.

4.2 Podpora aplikačnej paralelizácie platformou NetCOPE

Na základe znalosti architektúry NetCOPE, ktorá bola popísaná v kapitole 2 a na základe analýzy možností paralelizácie typických sieťových aplikácií z predchádzajúcej sekcie, budú popísané vhodné modely realizovateľné softvérovými rozhraniami platformy NetCOPE pre podporu paralelizácie sieťových aplikácií. Podpora spočíva v umožnení prístupu softvérovým aplikáciám k dátam spôsobom vhodným pre paralelné spracovanie. Modely navrhnem zvlášť pre príjem paketov do aplikácií (RX smer) a pre odosielanie (TX smer).

Keďže aplikácie pracujúce so softvérovým rozhraním szedata2 alebo rozhraním PCAP volajú vždy priamo alebo sprostredkovane funkcie knižnice libsze2, je táto možným miestom pre implementáciu modelov podpory paralelizmu.

Za volaniami knižnice libsze2 sa skrýva práca s ovládačom szedata2 a preto je tento ovládač ďalším kandidátom pre miesto implementácie mechanizmov.

Súčasná implementácia ovládača szedata2 podporuje MISD model pre RX smer, ktorý bude popísaný v nasledujúcej sekcii. Preto je ovládač preferovaným miestom pre implementáciu. Existujúca implementácia si vyžiada zmeny architektúry ovládača – vytvorenie abstrakcie vykonávajúcej modely paralelizácie pre aplikácie. Keďže modelov paralelizácie bude typicky viac, musí byť zaručené jednotné rozhranie medzi modulmi a jednoduchá zmena používaného modelu paralelizácie.

V nasledujúcom texte bude používaný pojem proces v zmysle jednotky vykonávania kódu, čiže ako proces alebo vlákno v zmysle operačných systémov.

4.2.1 RX smer

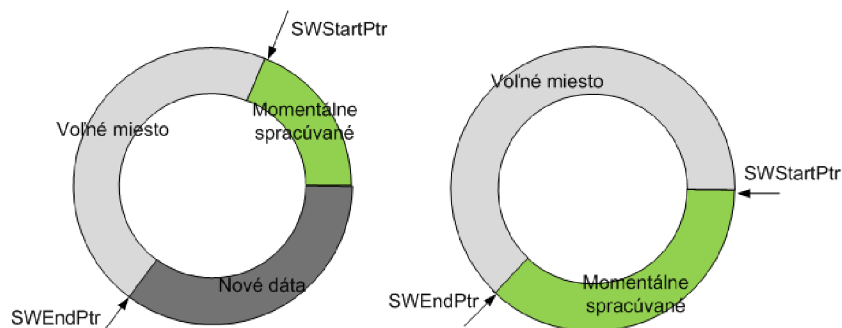
Analýza typických sieťových aplikácií ukázala užitočnosť podpory pre nasledujúce modely paralelizácie:

4.2.1.1 MISD

Platforma musí zaručiť prístup k rovnakým dátam viacerým procesom. Rôzne procesy spracúvajú paralelne rovnaké alebo rôzne časti dát, podľa rýchlosti práce jednotlivých procesov. Platí ale, že všetky procesy musia spracovať všetky dostupné dáta. Keďže dáta sú prístupné len na čítanie, nie je nutné použiť synchronizáciu prístupu. Spracovanie rovnakého množstva dát rôznymi procesmi trvá rôzne dlhú dobu. Výsledná rýchlosť paralelného spracovania dát je priamo závislá na najpomalšie pracujúcom procese.

Voľba veľkosti bloku dát MISD model môžeme ovplyvniť voľbou rovnako veľkého alebo rozdielne veľkého bloku dát pre rôzne procesy. Pri prideľovaní blokov dát rovnakej veľkosti môžeme procesy synchronizovať po spracovaní bloku tak, že ďalší blok začnú spracúvať naraz, alebo nechať procesy predbiehať sa v rámci kruhového bufferu.

Voľba veľkosti prideleného bloku dát môže tiež významne ovplyvniť výkonnosť. Voľba príliš veľkého bloku dát môže spôsobiť dlhú dobu spracovania, neuvoľňovanie miesta v kruhovom bufferi a stratu príchodzích dát z firmvéru. Voľba príliš malej veľkosti bloku dát zas môže viesť na vysokú réžiu.



Obrázok 4.6: a) stav kruhového bufferu pri prístupe MISD b) stav kruhového bufferu pri prístup MISD po spracovaní bloku

Veľkosť bloku dát môžeme nastaviť staticky, pri inicializácii programu. Možné by bolo realizovať aj dynamickú voľbu veľkosti bloku vypočítavanú z údajov získaných pri behu procesov.

Správa kruhového bufferu Z hľadiska správy RX kruhového bufferu v pamäti RAM je nutné evidovať procesy pracujúce s rozhraním szedata2 a ich pozíciu v kruhovom bufferi. Dáta môžu byť z kruhového bufferu uvoľnené až po spracovaní najpomalším vláknom. Táto situácia je zobrazená na obrázku 4.6.

Po spracovaní bloku dát všetkými dátami je ukazateľ SWStartPtr posunutý o veľkosť spracovaných dát a dáta sú uvoľnené. Následne sa začne spracovávať ďalší blok nových dát.

Tento model je implementovaný v súčasnej verzii ovládača szedata2.

4.2.1.2 SPMD

Pre podporu tohoto modelu paralelizácie je nutné zaručiť primerané rozdelenie dát medzi pracujúce vlákna. Každý paket je spracovaný len jediným vláknom a rôzne vlákna súčasne spracúvajú rôzne dáta. Túto situáciu ilustruje obrázok 4.7.

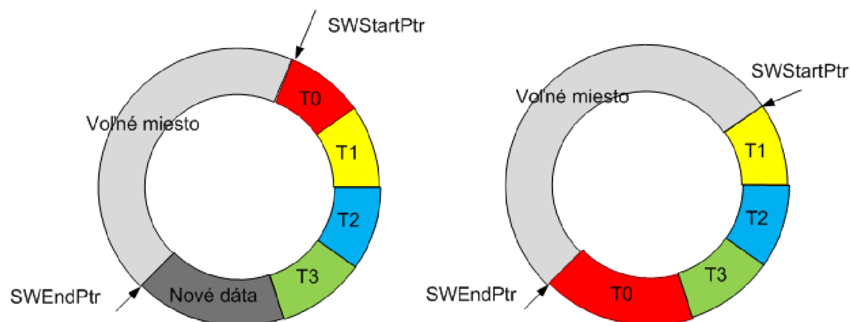
Pre veľkosť prideľovaného bloku dát platia podobné pravidlá, ako pri predchádzajúcom modele.

Správa kruhového bufferu Z hľadiska správy RX kruhového bufferu je nutné evidovať vlákna pracujúce s rozhraním szedata2, pozíciu a veľkosť aktuálne spracúvaných dát pre každé vlákno.

Pri požiadavke procesu o nové dáta, je mu pridelený prvý dostupný blok dát. V priebehu práce procesov môže nastať stav, keď jeden proces predbehne v spracovaní proces spracúvajúci blok dát bližšie začiatku. Spracované dáta sú pripravené na uvoľnenie, avšak uvoľnené zatiaľ byť nemôžu, vzniká fragmentácia kruhového bufferu. Tento jav je nežiadúci. Uvoľnenie dát môže nastať iba pri ukončení práce procesu spracúvajúceho počiatočný blok dát (za ukazateľom SWStartPtr).

4.2.1.3 Zreťazené spracovanie

Model zreťazeného spracovania je veľmi podobný modelu MISD, avšak vyžaduje prístup po jednotlivých paketoch. Zároveň procesy musia obsahovať identifikáciu, ktorá reprezentuje ich poradie v zreťazenom spracovaní. Ďalej by muselo byť obohatené rozhranie ovládača, pre možnosť vyradenia jednotlivého paketu zo zreťazeného spracovania v niektorom z jeho stupňov.



Obrázok 4.7: a) stav kruhového bufferu pri prístupe SPMD b) stav kruhového bufferu pri prístupe SPMD po spracovaní bloku vláknom T0

Uvoľnenie dát by mohlo nastať vždy pri dokončení spracovania prvého paketu (za ukazateľom SWStartPtr). Tiež sa tu vyskytuje problém fragmentácie.

4.2.2 TX smer

Paralelné odosielanie dát môže byť realizované jedným z nasledujúcich dvoch modelov:

Známa veľkosť dát Pokiaľ proces pozná veľkosť dát pre odoslanie, ovládač mu poskytne miesto v kruhovom bufferi a proces môže do tohoto priestoru pripraviť dáta. Súbežne s touto činnosťou môže o odoslanie dát požiadať ďalší proces. Požiadavky musia byť vzájomne vylúčené.

Keďže ovládač pozná veľkosť dát už prebiehajúceho zápisu, môže procesu poskytnúť miesto v kruhovom bufferi nasledujúce za miestom rezervovaným pre predchádzajúci požiadavok. Zápis dát takto môže prebiehať paralelne.

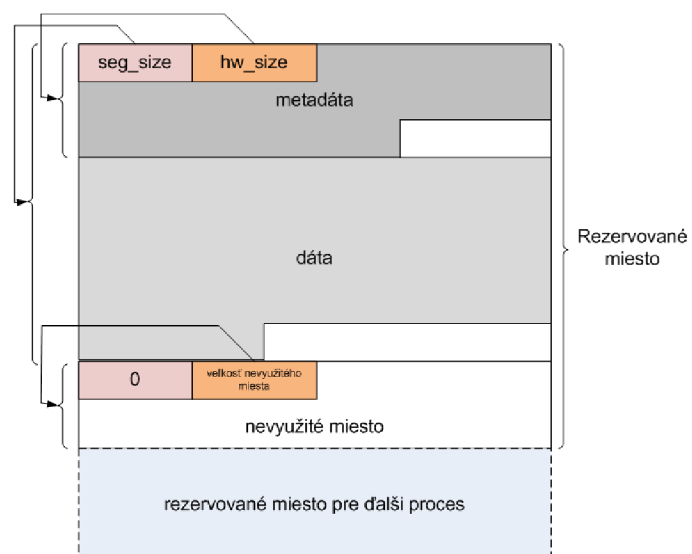
Pri uvedenom prístupe môže nastávať fragmentácia kruhového bufferu, pokiaľ sa procesy predbiehajú. Dáta však môžu byť odoslané vždy až po ukončení zápisu na začiatok kruhového bufferu (za ukazateľ SWStartPtr).

Neznáma veľkosť dát Pokiaľ proces veľkosť dát pre odoslanie nepozná (typicky pri blokových zápisoch), môže ovládač požiadať o miesto, ktoré považuje za dostatočné a určite bude vyhovovať požiadavkám odoslania. Súčasne môže o odoslanie dát požiadať ďalší proces, ktorý dostane miesto za miestom rezervovaným pre prvý proces.

Prvý proces rezervované miesto zaplní podľa svojich požiadavkov. Pokiaľ je však veľkosť skutočne zapísaných dát menšia ako poskytnuté miesto, vzniká problém s týmto nevyužitým miestom. Formát dát (zobrazený na obrázku 2.6) predpokladá následnosť segmentov jedeného za druhým, avšak nevyužité miesto nemá potrebný formát, dáta sú nedefinované, a mohli by tak spôsobiť chybné spracovanie vo firmvéri.

S nevyužitým miestom sa dá vysporiadať vyplnením tohoto miesta špeciálnym segmentom, ktorým firmvér deteguje, že sa jedná o nevyužité miesto a tieto dáta nebudú ďalej spracované. Možné je použiť segment s hodnotami hlavičky seg_size 0 a hw_size veľkosť nevyužitého miesta. Táto situácia je znázornená na obrázku 4.8.

Cache Keďže pri paralelizácii odosielania dát sú dáta zapisované, bude dôležité venovať sa problémom súvisiacim s pamäťou cache popísaným v sekcii 3.2. Tento problém bude riešený v pokračovaní tejto práce.



Obrázok 4.8: Paralelný zápis pri neznámej veľkosti dát

Kapitola 5

Systémové ovládače

Táto kapitola popisuje základné pojmy v oblasti vývoja ovládačov zariadení operačného systému Linux. Táto oblasť je špecifická, jej pochopenie si vyžaduje naštudovanie problematiky viacerých pojmov, princípov a nástrojov vyskytujúcich sa v tejto oblasti, s ktorými programátor nepríde bežne do styku pri vývoji užívateľských aplikácií. Taktiež táto oblasť vyžaduje vyššie nároky na bezpečnosť kódu a jeho kvalitu. Obdobná chyba, ktorá v užívateľskej aplikácií spôsobí pád aplikácie, v priestore jadra systému môže spôsobiť jeho pád.

Základnou úlohou ovládača zariadenia je vytvorenie mostu medzi operačným systémom, ktorý je využívaný aplikáciami cez štandardné rozhrania a ovládaným zariadením. Väčšinou je pod pojmom zariadenie myslený určitý hardvér, ale existujú aj čisto softvérové zariadenia ako virtuálne terminály alebo loopback sieťové rozhranie.

Moduly jadra Linux ako moderný operačný systém umožňuje rozšíriť funkcionalitu operačného systému dynamicky za behu. Pre tento účel sa používa rozhranie modulov špecifikované v súbore *linux/module.h.* Ovládače zariadení sú typicky používané ako moduly jadra nahrávané a odstraňované zo systému podľa potreby. Pre nahranie modulu slúžia príkazy *insmod a modprobe,* naopak pre odstránenie modulu slúži *rmmod.* Je dôležité zmieniť, že každý modul musí špecifikovať inicializačnú a ukončovaciu funkciu, tieto sú vykonané pri nahraní respektíve odstránení modulu.

Synchronizácia, stav prevádzania procesu a kontext O problematike súbežného spracovania, determinizmu, synchronizácie a vzájomného vylúčenia pojednáva sekcia 3.4. Jadro operačného systému je vysoko paralelný systém, súčasne vykonávajúci veľké množstvo procesov pracujúcich so zdieľanými zdrojmi. V tomto prostredí je používanie synchronizačných mechanizmov nutné a bežné. Paralelizmus vzniká pseudoparalelným behom aplikácií (multitasking) a tiež behom na viacerých procesorových jadrách pri SMP systéme.

Pojem stav prevádzania procesu súvisí s problematikou procesov a vlákien z pohľadu operačného systému podanej v sekcii 3.4.2. Dôležité je že pri preemptívnom prideľovaní procesoru môže byť práve vykonávaný proces kedykoľvek odstavený od procesoru. Tým je pozastavený až do doby, pokiaľ mu nebude procesor znovu pridelený. Situáciu komplikuje ešte aj existencia prerušenia, ktoré asynchrónne môže kedykoľvek prerušiť vykonávanie procesu. V tomto okamihu je nutné ozrejmiť i ďalší dôležitý pojem a tým je kontext. Väčšina procesov vykonávajúcich program jadra beží v prospech nejakého užívateľského procesu, ktorý zavolal niektoré zo systémových volaní. Kontext programu v jadre je teda užívateľský proces. Naopak, pri vykonávaní obsluhy prerušenia alebo pri oneskorenom spracovaní (technika spracovania úloh v jadre) tento proces nebeží v kontexte užívateľského procesu a nemôže

spať.

Tieto skutočnosti je nutné brať do úvahy pri voľbe synchronizačných nástrojov. V zásade sa synchronizačné mechanizmu delia na dve významné skupiny podľa toho, ako dlhá je kritická sekcia a či v nej môže proces spať. Pre dlhšie kritické sekcie, v ktorých môže nastať prerušenie behu procesu, sa používajú semafóry (*struct semaphore*), mutexy (*struct mutex*) a „read write” semafóry (*struct rw_semaphore*). Pre krátke kritické sekcie, v ktorých je prerušenie procesu zvyčajne veľkou chybou a spôsobuje degradáciu výkonu, sa používajú spinloky (*spinlock_t*) a read write spinloky (*rwlock_t*).

Správa a mapovanie pamäte Linux využíva systém virtuálnej pamäte, ktorého základný princíp je, že adresy z priestoru užívateľských aplikácií nie sú priamo zhodné s fyzickými adresami, kde sú dáta uložené. Tento mechanizmus pre každý proces vytvára abstrakciu vlastnej pamäte a umožňuje procesu alokovať požadovanú veľkosť pamäte, dokonca väčšiu ako veľkosť fyzickej pamäte systému. Podpora tohoto mechanizmu na úrovni procesoru je sústredená v jednote MMU, ktorá slúži k rýchlemu prekladu virtuálnych adries na fyzické.

Pamäť systému je rozdelená na logické jednotky zvané stránky. Veľkosť stránky je závislá od architektúry, špecifikuje ju makro *PAGE_SIZE,* typická veľkosť na architektúre x86 je 4 KB. Správa pamäte pracuje so stránkami ako elementárnymi jednotkami prideľovania pamäte. Každá adresa do pamäte sa skladá z adresy stránky a offsetu v rámci stránky, ktorý je závislý od jej veľkosti. Stránky prislúchajúce jednému procesu sú organizované v tabuľkách. Tabuľky stránok vytvárajú viacúrovňovú štruktúru. Nepotrebné stránky sú z hlavnej pamäte odkladané mechanizmom zvaným „swapping” do pamäte s pomalším prístupom, zvyčajne na pevný disk. Odložené stránky sú v prípade potreby[1] pri výpadku nahrávané späť do hlavnej pamäte.

Všeobecne sa pre prístup do pamäte využíva viac druhov adries. Užívateľské adresy sú vždy virtuálne a každý užívateľský proces má svoj virtuálny adresový priestor. Fyzické adresy využíva procesor pri prístupe do fyzickej pamäte. Logické adresy jadra vytvárajú bežný adresový priestor jadra, väčšinou sú priamo zhodné s fyzickými adresami alebo sa líšia len v offsete. Virtuálne adresy jadra mapujú adresový priestor jadra na fyzické adresy, toto mapovanie však nie je priame ako pri logických adresách.

Logické a virtuálne adresy jadra nám celkovú pamäť rozdeľujú na tzv. „high” a „low memory”. „Low memory” je oblasť pamäte, pre ktorú existujú logické adresy jadra. Naopak oblasť pamäte, pre ktorú tieto adresy neexistujú sa nazýva „high memory”.

Adresový priestor každého procesu sa skladá z oblastí virtuálnej pamäte. Sú to súvislé oblasti pamäte s rovnakými právami, ktoré sa vzťahujú k jednému objektu. Každá oblasť je v jadre reprezentovaná štruktúrou *struct vma_area_struct* a vzťahujú sa k nej operácie nad touto štruktúrou typu *struct vm_operations_struct.*

Mapovanie pamäte je mechanizmus, ktorý môže zabezpečiť prístup z užívateľských programov do adresového priestoru jadra. Vo výkonovo náročných aplikáciách je tento mechanizmus nevyhnutný. Mapovanie prebieha systémovým volaním *mmap()* z aplikácie, ovládač sa postará o vytvorenie mapovaných stránok a ak je to nutné, môže definovať štruktúru *struct vm_operations_struct.*

Prerušenie Prerušenie je asynchrónny signál, ktorým dáva zariadenie najavo že vyžaduje obsluhu. Tento mechanizmus ďalej komplikuje synchronizačné problémy. Pre zabráneniu prerušenia slúži mechanizmus maskovania prerušení, ktorý sa používa najmä pri spinlockoch. Ovládač zariadenia si alokuje IRQ[2] číslo a zaregistruje si obslužnú rutinu tohoto

[1] anglicky on demand
[2] interrupt request

prerušenia, ktorá sa vykoná vždy keď prerušenie dorazí. Obslužná rutina musí byť krátka, aby neblokovala ostatné prerušenia, pre zložitejšie výpočty slúžia mechanizmy oneskoreného spracovania.

Zariadenia V operačnom systéme Linux existujú tri základné typy zariadení – znakové, blokové a sieťové. Ovládač zariadenia implementuje jedno z týchto chovaní. Keďže v rámci tejto práce ovládač pracuje práve so znakovým zariadením, tento typ zariadenia tu bude podrobnejšie popísaný a aj na operačný systém, jeho štruktúry a mechanizmy bude nazerané práve z perspektívy znakového zariadenia.

Znakové zariadenie pracuje s prúdmi bajtov, podobne ako rozhranie súborov s tým rozdielom, že zariadenie neumožňuje pohyb vzad a vpred v rámci prúdu a prístup je teda len sekvenčný. K znakovým zariadeniam sa pristupuje prostredníctvom súborového systému – cez špeciálne súbory z adresára */dev*. Výstup príkazu *ls -l* poskytuje tri základné údaje o súbore. Hneď prvé písmeno „*c*" signalizuje, že sa jedná o špeciálny súbor znakového zariadenia. Ďalšími významnými údajmi sú major a minor čísla zariadenia. Prvé z nich uvádza, ktorý ovládač sa o zariadenie stará, druhé číslo umožňuje tomuto ovládaču presne špecifikovať konkrétnu inštanciu zariadenia, keďže rovnakých zariadení, o ktoré sa stará jediný ovládač, typicky existuje v systéme viacero.

```
crw-rw-rw- 1 root wheel 251, 0 May 3 15:23 /dev/szedataII0
```

Interne v jadre systému sú minor a major čísla zariadení reprezentované typom *dev_t*. Získanie týchto čísel je jednou z prvých úloh ovládača. Funkcia *int alloc_chrdev_region (dev_t *dev, unsigned int firstminor, unsigned int count, char *name)* dynamicky alokuje od jadra systému špecifikovaný rozsah čísel zariadení a asociuje s týmito číslami meno zariadenia v systéme */proc/devices* a sysfs.

Základné štruktúry jadra Pre vysvetlenie funkcie ovládača v operačnom systéme je nutné zmieniť funkciu základných štruktúr používaných v jadre systému Linux.

file_operations Táto štruktúra mapuje štandardné systémové volania operačného systému na funkcie implementujúce toto chovanie v konkrétnom ovládači. Jedná sa o súbor ukazateľov na funkcie, pokiaľ má niektorý ukazateľ nulovú hodnotu, je operácia s týmto volaním spojená považovaná za nepodporovanú. Táto technika je v jadre systému Linux široko používaná a môžeme o nej hovoriť ako o črte objektovo orientovaného prístupu. Štruktúra ukazateľov na funkcie vytvára rozhranie, samotná implementácia týchto funkcií je však volajúcemu skrytá, jedná sa teda o zapúzdrenie. Niektoré členy tejto štruktúry, systémové volania, ktoré sú významné z hľadiska implementácie ovládača szedata2 si popíšeme neskôr v kontexte tohoto ovládača.

struct file V jadre reprezentuje každý otvorený súbor, či už obyčajný, alebo špeciálny súbor zariadenia. Štruktúra je vytvorená pri volaní *open()* a je predávaná všetkým systémovým volaniam nad súborom, zrušená je pri volaní *close()*. Z položiek štruktúry môžeme zmieniť len najvýznamnejšie:

- *unsigned int f_flags* - príznaky špecifikujúce prístup a mód otvorenia súboru z hľadiska blokujúcich a neblokujúcich operácií.
- *struct file_operations *f_op* - štruktúra mapujúca systémové volania nad každým otvoreným súborom na ich konkrétnu implementáciu. Pri špeciálnych súboroch, ako

bolo spomenuté, sa jedná o mapovanie na obslužné rutiny ovládačov, pri obyčajných súboroch sú volania mapované na rutiny modulov konkrétneho súborového systému obsahujúceho daný súbor.

inode Je internou štruktúrou jadra reprezentujúcou súbor v rámci súborového systému. Z pohľadu systémových ovládačov sú dôležitými položkami štruktúry *dev_t i_rdev* – major a minor čísla zariadenia a *struct cdev *i_cdev* – ukazateľ na konkrétne znakové zariadenie.

struct cdev Jedná sa o internú štruktúru jadra reprezentujúcu jediné znakové zariadenie. Asociuje štruktúru *file_operations* s týmto zariadením. Registrácia tejto štruktúry v jadre je nutná pre umožnenie samotnej práce so znakovým zariadením. Pre tento účel sa využíva funkcie *int cdev_add(struct cdev *dev, dev_t num, unsigned int count).* Pred samotnou registráciou zariadenia je nutné vykonať všetky inicializačné kroky nad zariadením, pretože okamžite po registrácii je zariadenie k dispozícii k použitiu.

Proces inicializácie a používania ovládača Zhrnieme celý proces inicializácie a používania systémového ovládača. Ovládač alokuje množinu minor a major čísel, vytvorí štruktúru *struct cdev* asociujúcu toto zariadenie s týmito číslami a zariadenie zaregistruje. Aplikácie používajúce zariadenie otvoria systémovým volaním *open()* súbor v */dev* reprezentujúci dané zariadenie pričom všetky systémové volania z aplikácie sú mapované na rutiny ovládača. Aplikácie pracujú so zariadením, ovládač vybavuje požiadavky v ich prospech.

V tomto procese sme zatiaľ vynechali jednu časť a tou je vytvorenie súboru zariadenia v */dev.* Tento súbor je možné vytvoriť manuálne príkazom *mknod* pri znalosti major a minor čísla zariadenia. Pre zjednodušenie a automatizáciu tohoto kroku je možné použiť mechanizmu *udev*, ktorý na základe jednoduchých konfiguračných súborov automaticky vytvorí konkrétny špeciálny súbor po registrácii zariadenia.

Rozhranie aplikácií a ovládača Interakcia medzi aplikáciami z užívateľského priestoru a ovládačmi prebieha viacerými spôsobmi. Prvým z nich sú štandardné systémové volania. Významným systémovým volaním je funkcia *ioctl(),* ktorá ponúka rozhranie pre vytvorenie interakcie rozdielnej od štandardných systémových volaní, umožňuje definovať *ioctl* príkazy špecifické pre zariadenie, smer (vstupný alebo výstupný) a typ parametrov príkazov. Na tomto mieste je nutné zmieniť, že predávanie parametrov z užívateľského priestoru do priestoru jadra nie je triviálne (na rozdiel od predávania parametrov cez zásobník v užívateľských aplikáciách), je nutné kopírovať dáta medzi týmito dvomi adresovými priestormi. Ukazatele označené ako *__user* sú ukazatele do adresového priestoru procesu, v ktorého prospech sa vykonáva aktuálne systémové volanie. Pre kopírovanie dát medzi dvomi priestormi slúžia makrá *get_user()* a *put_user().*

Ďalším zo špecifických systémových volaní je *mmap(),* umožňuje namapovanie adresového priestoru jadra do adresového priestoru aplikácie. Následne môže interakcia medzi aplikáciou a ovládačom prebiehať cez tento priestor.

/proc virtuálny súborový systém slúži hlavne pre získavanie informácií o jadre systému a ovládačoch a tiež pre čítanie a nastavovanie hodnôt parametrov ovládačov. Podobnú funkciu plní aj systém sysfs, ktorý je zmienený v nasledujúcom odstavci.

Model zariadení systému Linux a sysfs Vytvára unifikovaný pohľad na štruktúru zariadení operačného systému, postihuje vzťahy a závislosti medzi zariadeniami. Táto štruktúra sa využíva pre účely riadenia napájania, vypínania zariadení, dynamického pripájania

a odpájania zariadení za behu systému (tzv. „hotplug”), modelovania subsystémov, zberníc a tried zariadení a riadenia životného cyklu objektov jadra vzťahujúcich sa k zariadeniam. Ďalšou funkciou tejto štruktúry je komunikácia s užívateľskými aplikáciami cez virtuálny súborový systém sysfs, ktorý poskytuje komplexné informácie o zariadeniach a umožňuje zmenu parametrov ovládačov.

Rozhranie ovládačov medzi sebou Ovládače zložitejších zariadení zvyčajne tvoria vrstvenú[3] štruktúru, kde nižšie vrstvy ponúkajú určité služby tým vyšším. Tento princíp modularity, vlastný celej oblasti informačných technológií, zamedzuje duplikácii kódu a vedie k jednoduchšej implementácii ovládačov. Interakcia medzi ovládačmi prebieha v adresovom priestore jadra, ovládače poskytujú svoje rozhranie deklaráciou *EXPORT_SYMBOL (function_name)*, ktorá sprístupní volanie funkcie všetkým ostatným súčastiam jadra. Počty použití modulu inými modulmi upravujú funkcie *try_module_get()* a *module_put()*. Modul nemôže byť odstránený pokiaľ je používaný inými modulmi.

[3] anglicky stack

Kapitola 6

Ovládač szedata2

Táto kapitola sa venuje vrstve ovládačov platformy NetCOPE, ich celkovej spolupráci a podrobnejšie skupine ovládačov szedata2, ktoré zabezpečujú rýchle DMA prenosy a vytvárajú rozhranie k užívateľským aplikáciám. Modely pre súbežný prístup k dátam pre podporu aplikačnej paralelizácie definované v sekcii 4.2 sú realizované pomocou ovládača szedata2. Jeho štruktúra a celkový princíp práce a princíp práce pre každý z modelov sú v tejto kapitole podrobne popísané.

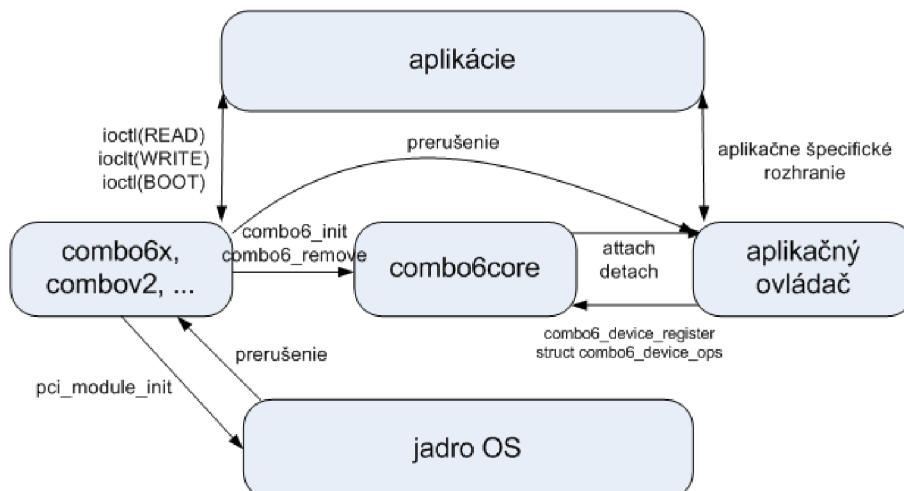
6.1 Architektúra ovládačov projektu Liberouter

Vrstvená štruktúra Skôr ako popíšeme architektúru ovládačov na projekte Liberouter, je nutné sa zmieniť o spôsobu použitia technológie FPGA. Projekt paralelne vyvíja viacero rôznych hardvérových zariadení. V súčasnej dobe to sú NIC – sieťová karta, NIFIC – firewall a FlowMon – zariadenie pre monitorovanie sieťových tokov. Tieto projekty zdieľajú veľké množstvo kódu implementujúceho hardvérové komponenty, projekty založené na platforme NetCOPE dokonca celé prostredie platformy. Kód popisujúci hardvér je syntetizovaný do binárnej reprezentácie zvanej dizajn vhodnej pre nahranie do čipu FPGA na hardvérovej karte, kde následne realizuje svoju funkciu. Každý dizajn je charakterizovaný identifikátorom zvaným SW_ID. Tá istá karta môže v krátkom okamihu zmeniť funkciu určitého zariadenia na inú. Taktiež, ako bolo zmienené v sekcii 2.3, sú používané rôzne hardvérové akceleračné karty, ktorých ovládanie je rôzne.

Tento spôsob vývoja projektov reflektujú aj systémové ovládače. Vrstvená štruktúra umožňuje zabrániť duplikácii kódu pre rôzne projekty a poskytuje rozhrania, ktoré umožnia odtieniť závislosť na použitom hardvéri.

Architektúra ovládačov je zobrazená na obrázku 6.1. Najnižšia vrstva hardvérovo závislých ovládačov, tvorená ovládačmi combo6x a combov2, implementuje operácie špecifické pre konkrétne hardvérové karty. Tieto moduly sú úzko závislé od použitej zbernice a FPGA čipu. Taktiež proces bootovania je značne rozdielny pre rôzne karty. Moduly sa v jadre zaregistrujú ako PCI zariadenia pre hardvérové karty so špecifickou PCI identifikáciou. Ovládače naplnia a zaregistrujú štruktúru *struct combo6*, ktorá slúži ako rozhranie odtieňujúce hardvérové špecifiká pre vyššie vrstvy. Súčasťou tejto štruktúry je položka *struct combo6_ops*, súčasťou ktorej je ukazateľ na rutinu obsluhujúcu *ioctl()* systémové volanie. V systéme je vytvorené znakové zariadenie */dev/combosix/X* (kde X je číslo zariadenia), ktoré týmto *ioctl* rozhraním umožňuje aplikáciám využívať vstupno-výstupné operácie a nahrávať aplikačný dizajn.

Strednú vrstvu tvorí ovládač combo6core, ktorý spravuje zoznam všetkých pripojených kariet a všetkých aplikačných ovládačov. Do týchto zoznamov sa jednotlivé moduly registru-



Obrázok 6.1: Vrstvená architektúra ovládačov na projekte Liberouter

jú samostatne, combo6core pre tieto účely exportuje funkcie *combo6_init(), combo6_remove()* (pre spodnú vrstvu), *combo6_device_register() a combo6_device_unregister()* (pre aplikačnú vrstvu). Stará sa o prepojenie spodnej vrstvy s vrstvou aplikačnou. Zabezpečuje priradenie správneho aplikačného ovládača aktuálnemu dizajnu, podľa zhody identifikátora SW_ID.

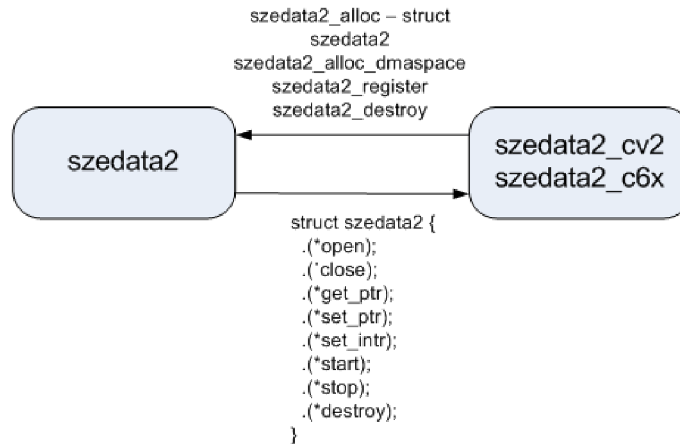
Aplikačný ovládač, bez starostí o použitú hardvérovú kartu, vytvára špecifické rozhranie pre aplikácie. Rozhranie vytvára štruktúra *struct combo6_device_ops*, ktorú ovládač vytvorí a zaregistruje. Môže sa jednať o sieťové systémové zariadenie, ktoré sprístupní dáta z hardvérovej karty prostredníctvom soketov, alebo znakové zariadenie rôzne implementujúce systémové volania pre dosiahnutie požadovanej funkcionality.

Štruktúra zdrojových kódov Vývoj ovládačov na projekte Liberouter prebieha s pomocou nástroja git[1] v repozitári (combo-driver.git). Tento repozitár slúži pre samotný vývoj, obsahuje pomocné testovacie moduly, súbory zabezpečujúce kompatibilitu so staršími verziami jadra a zdrojové súbory kompatibilné s aktuálnou verziou jadra.

Adresárová štruktúra repozitáru reflektuje vrstvovú štruktúru ovládačov a je nasledujúca:

```
build/ - pomocné súbory
kernel/ - zdrojové kódy kompatibilné s aktuálnou verziou jadra systému
          Linux
*/drivers - zdrojové .c súbory
*/include - hlavičkové .h súbory
*/drivers/base - stredná core vrstva modelu
*/drivers/first - spodná vrstva modelu pre prvú generáciu hardvérových
                  kariet
*/drivers/second - spodná vrstva modelu pre druhú generáciu hardvérových
                   kariet
*/drivers/third - spodná vrstva modelu pre tretiu generáciu hardvérových
                  kariet
*/drivers/szedata - szedata verzie 1
*/drivers/szedata2 - hardvérovo nezávislá časť szedata verzie 2
```

[1]distribuovaný systém pre správu revízií používaný pre vývoj jadra systému Linux



Obrázok 6.2: Architektúra ovládačov szedata2

6.2 Architektúra ovládačov szedata2

Logicky je skupina ovládačov szedata2 rozdelená na hardvérovo závislú a hardvérovo nezávislú časť. Prvá znich je tvorená modulmi szedata2_cv2, pre karty ML555 a COMBOv2 a szedata2_c6x pre kartu COMBO6x. Hardvérovo nezávislá časť je tvorená modulom szedata2.

6.3 Hardvérovo závislá časť

Zabezpečuje komunikáciu medzi hardvérovou kartou a hardvérovo nezávislou časťou. Pre komunikáciu s hardvérom využíva služieb rozhrania ovládača combo6core. Hardvérovo závislá časť ovládača je aplikačným ovládačom v ponímaní architektúry ovládačov platformy.

Komunikácia s hw nezávislou časťou prebieha volaním jej exportovaných funkcií. Závislá časť si u nezávislej zaregistruje štruktúru *struct szedata2* a nastaví jej zložky ukazateľov na funkcie na jej hardvérovo špecifické rutiny. Závislá časť je tiež zodpovedná za inicializáciu hardvérovo nezávislej časti - alokáciu špecifického počtu kruhových bufferov podľa použitej hardvérovej karty - volaniami *szedata2_alloc*(), *szedata2_alloc_dmaspace*(), *szedata2_register()* a *szedata2_destroy*(). Situáciu ilustruje obrázok 6.2. Ďalšou z úloh hardvérovo závislej vrstvy je alokácia deskriptorov a ich prenos do hardvérovej akceleračnej karty ako bolo popísané v podsekcii 2.6.2.2.

Ovládač szedata2_cv2 je špecifický aplikačný ovládač, pretože je v jadre systému zaregistrovaný navyše ako ovládač PCI zariadenia. Celkový proces inicializácie ovládačov nad kartou COMBOv2 prebieha nasledovne. Hardvérovo závislý ovládač sa u jadra operačného systému zaregistruje ako PCI ovládač svojou štruktúrou *struct pci_driver*. Táto štruktúra obsahuje tabuľku ním ovládaných PCI zariadení. Pokiaľ jadro nájde niektoré PCI zariadenie z tejto tabuľky, zavolá *attach* funkciu štruktúry. V obsluhe tejto funkcie prebehne alokácia a naplnenie štruktúry *struct combo6,* ktorá reprezentuje combo6 zariadenie. Následne je zaregistrovaná u ovládača combo6core. Zároveň sú zaregistrované u tohoto ovládača aj štruktúry *struct szedata2x_ops,* ktoré mapujú rozsahy SW_ID firmvéru na obslužné rutiny im prislúchajúce. Volanie *attach* ovládača combo6core spôsobí porovnanie SW_ID aktuálneho firmvéru a rozsahov SW_ID zaregistrovaných combo6 zariadení, pri zhode je volaná príslušná rutina hardvérovo závislého ovládača. Táto rutina už zabezpečí inicializáciu hardvérovo nezávislej časti ovládača, tiež alokáciu a prenos deskriptorov.

6.4 Hardvérovo nezávislá časť

Táto časť celkového riešenia, ovládač szedata2, slúži ako most medzi aplikáciami bežiacimi v užívateľskom adresovom priestore a hardvérovo závislou časťou. Aplikácie komunikujú s ovládačom cez štandardné systémové volania operačného systému a ďalej ovládač komunikuje s hardvérovo závislou časťou volaním funkcií zaregistrovanej štruktúry *struct szedata2*.

6.4.1 Použité štruktúry

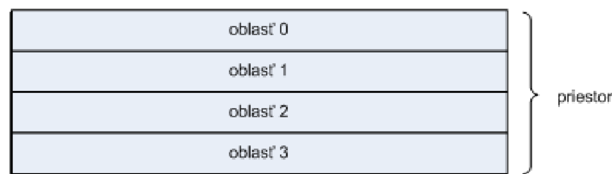
Pre vysvetlenie činnosti hardvérovo nezávislej časti ovládača je nutné špecifikovať interné štruktúry, ktoré vytvárajú abstrakcie domény a tiež ukázať, ku ktorým štruktúram jadra (ako boli popísané v odseku 5) sa vzťahujú. Abstrakcie budú podané v poradí od najjednoduchších až po komplexné abstrakcie zastrešujúce tie jednoduchšie.

- *struct szedata2_block* – elementárna štruktúra popisujúca jediný blok pamäte ako zložku kruhového bufferu. Reprezentuje mapovanie virtuálnej pamäte na alokovanú fyzickú pamäť (tento princíp je popísaný v sekcii 2.6.2.2).
- *struct szedata2_ring* – abstrakcia jedného priestoru. Jediný priestor („space" – typicky RX a TX) sa môže skladať z viacerých oblastí („area") ako zobrazuje obrázok 6.3. Každá oblasť tvorí samostatný kruhový buffer. Samotný priestor sa skladá z väčšieho počtu elementárnych blokov (*struct szedata2_block*).
- *struct szedata2_instance_info* – slúži ako časť rozhrania ovládača a aplikácie. Aplikácia má túto štruktúru namapovanú v jej adresovom priestore cez volanie *mmap()* a číta jej hodnoty. Štruktúra obsahuje údaje o priestoroch, kruhových bufferoch, využívaných rozhraniach a momentálne poskytnutých zámkoch pre danú aplikáciu.
- *struct szedata2_app* – je abstrakciou obsluhovanej (klientskej) aplikácie v ponímaní ovládača. Vytvára obal nad štruktúrou jadra *struct file*, ktorá vytvára kontext konkrétnej aplikácie v ovládači. Obsahuje premenné potrebné pre obsluhu aplikácie za jej behu, najmä ukazatele reflektujúce pozíciu v kruhovom bufferi. Ovládač nerozlišuje či je volaný vláknami jediného procesu, alebo rôznymi procesmi. Abstrakcia *struct szedata2_app* je vytváraná pre každé vlákno vykonávania zvlášť. Keďže pojem vlákno je často používaný v prostredí operačných systémov i pre procesy jadra, v ďalšom texte budú zhodne užívateľské vlákna i procesy využívajúce ovládač označované pojmom aplikácie.
- *struct szedata2* – jedná sa o komplexnú abstrakciu jedného szedata2 zariadenia. Obaľuje štruktúru systémového znakového zariadenia (*struct cdev*), ktoré slúži ako vstupný bod systémových volaní. Zariadenie obsahuje niekoľko priestorov (*struct szedata2_ring*), zoznam obsluhovaných aplikácií a zoznam aplikácií čakajúcich v systémovom volaní *poll()*.

Celkový pohľad na základné štruktúry a ich vzťahy použité v hardvérovo nezávislom ovládači je podaný na obrázku 6.4.

6.4.2 Správa zariadení

Táto sekcia popisuje inicializačný proces hardvérovo nezávislého ovládača samotného a inicializačné kroky zo strany hardvérovo závislého ovládača. Každá inicializačná funkcia má väčšinou aj funkciu reverznú, ktorá odstráni inicializačné efekty, preto pokiaľ je to možné sú funkcie uvádzané po takýchto dvojiciach.



Obrázok 6.3: Štruktúra priestoru

adresový priestor jadra

užívateľský adresový priestor

struct filp
struct filp
struct filp
szedata2_app
sze2_instalnce_info status_page
APP RX ptr
head tail
APP TX ptr
head tail
poll threshold
RX TX

struct szedata
struct sze2_instalnce_info
struct sze2_adesc RX0 adesc
struct sze2_adesc RX1 adesc
struct sze2_adesc TX0 adesc
struct sze2_adesc TX1 adesc
struct szedata_lock
struct szedata_lock
struct szedata_lock
struct szedata_lock
space[SZE2_MMIO_RX]
space[SZE2_MMIO_RX]
RX0 RX1 TX0 TX1

struct szedata2
app_list
poll_wait
struct cdev (/dev/szedataIIX) ukazatele na funkcie: struct file_ops
ukazatele na funkcie hw závislého ovládača
struct szedata2_ring ring[SZE2_MMIO_RX]
struct szedata2_block * RX0
struct szedata2_block * RX1
struct szedata2_ring ring[SZE2_MMIO_TX]
struct szedata2_block * TX0
struct szedata2_block * TX1

legenda
mmap-ované RO
mmap-ované RW
užívateľské rozhranie
rozhranie k HW závislému ovládaču
štruktúry per aplikáciu
štruktúry per zariadenie
štruktúry jadra OS

fyzická pamäť - stránky

Obrázok 6.4: Štruktúry hardvérovo nezávislého ovládača szedata2

- *szedata_init()/szedata_exit()* – vykonáva sa pri nahraní ovládača. Dynamicky alokuje priestor major a minor čísel a vytvorí triedu zariadení.
- *szedata2_alloc()* – alokuje štruktúru struct szedata, inicializuje jej položky.
- *szedata2_alloc_dmaspace()/szedata2_free_dmaspace()* – alokuje jeden priestor pre buffery schopný DMA prenosov.
- *szedata2_register()* – vytvorenie znakového zariadenia a pridanie tohoto zariadenia do triedy. Po tomto kroku by mal mechanizmus udev vytvoriť súbor */dev/szedataIIX* (X je číslo zariadenia). *struct cdev* má nastavené správne ukazatele na obslužné rutiny systémových volaní.

6.4.3 Kruhové buffery

Štruktúra kruhových bufferov je popísaná na všeobecnej úrovni v podsekcii 2.6.2 a v podsekcii 6.4.1 na úrovni štruktúr ovládača. Jednotlivé bloky, ako zložky priestorov a oblastí, sú tvorené blokmi konzistentnej pamäte. Zápis do konzistentnej pamäte, či už od procesoru, alebo zariadenia, je pre oboch okamžite viditeľný bez starostí o mechanizmy súvisiace s pamäťou cache. Bloky tejto pamäte sú získané volaním funkcie *void * pci_alloc_consistent (struct pci_dev *dev, size_t size, dma_addr_t *dma_handle)*, ktorá je súčasťou DMA API systému Linux. Funkcia vracia virtuálnu adresu jadra alokovanej pamäte a tiež jej fyzickú adresu, ktorá je použitá ako deskriptor pre DMA prenosy.

6.4.4 Komunikácia s „userspace"

Táto sekcia špecifikuje všetky rozhrania, cez ktoré nastáva komunikácia medzi ovládačom a klientskými aplikáciami.

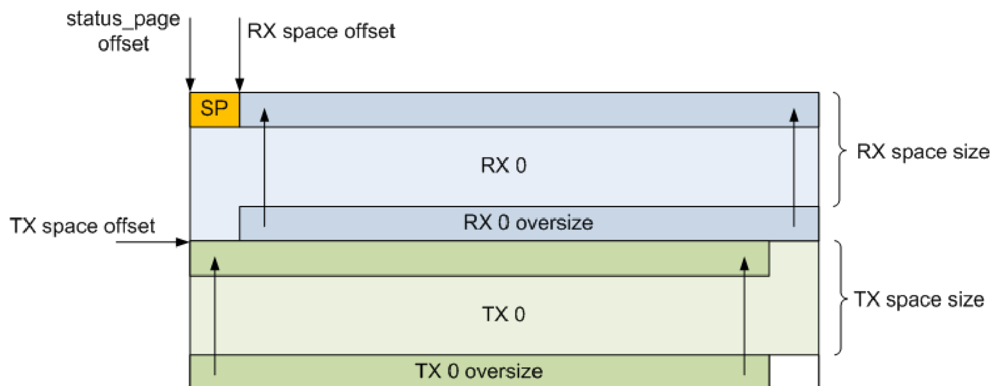
Systémové volania Ovládač podporuje len nasledujúce systémové volania: *open()*, *release()*, *poll()*, *mmap()*, *ioctl()*. Na rozdiel od bežného ovládača znakového zariadenia, ovládač szedata2 nepodporuje základné volania *read()* a *write()*. Toto je dôsledok špecifického prístupu ovládača k prenosu dát, ktorého mechanizmus a rozhranie bude podané v nasledujúcom texte.

Volania *open()*, *release()* a *poll()* súvisia so správou klientských aplikácií a sú popísané v nasledujúcej podsekcii. Volania *mmap()* a *ioctl()* vytvárajú špecifické rozhranie a preto sú popísané v nasledujúcom texte v samostatných odstavcoch.

ioctl() Rozhranie ioctl pri prístupe szedata2 nahradzuje tradičné volania *read()* a *write()*, ktoré kopírujú dáta medzi adresovými priestormi aplikácie a jadra, a poskytuje operácie špecifické pre szedata2 zariadenie. Jednotlivé ioctl príkazy budú popísané v nasledujúcich podsekciách o správe aplikácií, RX a TX smere, kde sú využívané.

mmap() a namapovaný priestor Ako bolo zmienené v odstavci 5, mapovanie pamäte jadra do užívateľského adresového priestoru je mocnou technikou pre dosiahnutie vysokej priepustnosti medzi hardvérovými zariadeniami a aplikáciami. Tento prístup je využitý v prostredí platformy NetCOPE.

Aplikácia si do svojho užívateľského priestoru mapuje dve základné štruktúry. Prvou z nich je štruktúra typu *struct szedata2_instance_info* zvaná *status_page*. Do tejto štruktúry ovládač zapisuje hodnoty popisujúce aktuálne zámky, štruktúry popisujúce aplikácii aktuálne poskytnuté časti kruhových bufferov pre spracovanie. *status_page* obsahuje i hodnoty



Obrázok 6.5: Mapované oblasti pamäte

popisujúce štruktúru priestorov kruhových bufferov, ktoré si aplikácia podľa týchto hodnôt v zápätí namapuje ako druhú štruktúru. Pamäť kruhových bufferov je týmto pripravená k prístupu zo strany aplikácie.

Mapovanie kruhových bufferov využíva predefinovanie funkcie *.fault* štruktúry *struct vm_operations_struct* oblasti virtuálnej pamäte reprezentujúcej kruhové buffery. Táto funkcia pri výpadku stránky rozhodne o nahratí príslušnej stránky bloku konzistentnej pamäte kruhového bufferu.

Pre zjednodušenie práce s namapovanými kruhovými buffermi zo strany užívateľských aplikácií a obaľujúcich knižníc bola použitá technika nazvaná **„buffer oversize"**. Toto vylepšenie ovládača szedata2 vzniklo v rámci tejto diplomovej práce. Každý kruhový buffer je mapovaný na spojitú oblasť virtuálnej pamäte, ktorá je väčšia ako jeho skutočná veľkosť. Toto predĺženie, „oversize", má veľkosť maximálne veľkého segmentu dát podľa formátu popísaného v podsekcii 2.6.3. Predĺženie kruhového bufferu je namapované na stránky zo začiatku kruhového bufferu. Táto technika umožňuje jednoduchšiu prácu s dátami pri zalomení kruhového bufferu, čo sa javí ako jediná kritická situácia pri práci s kruhovým bufferom z pohľadu aplikácií.

Situáciu pri mapovaní pamäte aplikáciou ilustruje obrázok 6.5, pre jednoduchosť priestory RX aj TX obsahujú každý iba jeden kruhový buffer.

Typický scenár komunikácie z pohľadu aplikácie Každá aplikácia komunikuje s ovládačom podľa typického scenáru. Postupnosť týchto krokov udáva UML diagram sekvencie C.1. Komunikácia začína inicializačnou časťou. Aplikácia otvára zariadenie cez špeciálny súbor súborového systému v */dev* systémovým volaním *open()*, čím získa súborový deskriptor (fd) otvoreného súboru. Všetky následné systémové volania sa vzťahujú k tomuto súborovému deskriptoru. Nasleduje mapovanie štruktúr status_page a priestorov kruhových bufferov cez volania mmap(). ioctl príkazom SZE2_IOC_SUBSCRIBE_AREA aplikácia špecifikuje kruhové buffery, s ktorými chce pracovať. Nasledujúci ioctl príkaz SZE2_IOC_START odštartuje samotný príjem a odosielanie dát pre danú aplikáciu. Tým je ukončená inicializačná časť a typicky v slučke prebieha príjem a odosielanie dát.

Po skončení spracovania dát aplikácia volaním ioctl príkazu SZE2_IOC_STOP ovládaču túto skutočnosť oznámi a systémovým volaním *close()* ukončí spojenie s ovládačom.

6.4.5 Správa aplikácií

Táto sekcia popisuje základné operácie týkajúce sa správy aplikácií z pohľadu ovládača. Interakcia nastáva pri systémových volaniach z aplikácie ako bolo popísané v typickom

scenári komunikácie. Opäť je väčšina funkcií uvádzaná spolu s ich inverznými dvojičkami.

- *szedata_open()/szedata_release()* – sú vykonávané ako obslužné rutiny systémových volaní *open()/close()*. Prevádzajú vytvorenie štruktúry *szedata2_app* reprezentujúcej klientsku aplikáciu a štruktúry *szedata2_instance_info* reprezentujúcej časť rozhrania aplikácie. Zvýšenie počítadla referencií ovládača.
- ioctl príkaz SZE2_IOC_SUBSCRIBE – nastavenie bitov reprezentujúcich rozhrania, s ktorými aplikácia pracuje, v príslušnej štruktúre *szedata2_instance_info* a hodnoty *poll threshold* (vysvetlené v odstavci 8.2).
- ioctl príkaz SZE2_IOC_START/SZE2_IOC_STOP – abstrakcia klientskej aplikácie je pridaná do zoznamu bežiacich aplikácií szedata2 zariadenia.

6.4.6 Príjem a odosielanie dát

Komunikácia nahradzujúca klasické systémové volania *read()* a *write()* pre príjem a odosielanie dát je nahradená ioctl príkazmi špecifickými pre prístup szedata2. Príjem dát (označovaný RX) aplikáciou je realizovaný ako umožnenie čítania určitej časti kruhového bufferu. Pre tento účel slúžia ioctl príkazy SZE2_IOC_RXLOCKDATA (ďalej označované aj ako požiadanie o dáta), ktorý ovládač požiada o poskytnutie dát, a SZE2_IOC_RXUNLOCKDATA (ďalej označované aj ako uvoľnenie dát), ktorý oznámi ovládaču, že dáta sú spracované. Rovnakým spôsobom funguje aj odosielanie dát (označované TX). Aplikácia žiada ioctl príkazom SZE2_IOC_TXLOCKDATA a priestor v kruhovom bufferi pre zapisovanie dát, príkazom SZE2_IOC_TXUNLOCKDATA oznámi koľko bajtov skutočne do tohoto priestoru zapísala a dáta sú pripravené na odoslanie. Tento proces je zobrazený na diagrame C.2.

Pre každú klientskú aplikáciu musí teda ovládač zaznamenávať jej súčasnú polohu v kruhových bufferoch. Tieto dáta sú uložené vo forme offsetov v príslušnej *szedata2_app* štruktúre. V nasledujúcom texte budeme tieto údaje označovať ako APP TAIL pre počiatok oblasti spracúvanej aplikáciou a APP HEAD pre jej koniec.

Hlavnou úlohou ovládača je teda sprístupňovanie oblastí kruhových bufferov aplikáciám pre príjem a odosielanie dát. Aby toto mohol ovládač vykonávať, musí mať informácie o stave kruhových bufferov z pohľadu hardvéru. Táto problematika je popísaná v sekcii 2.6. Hardvér pre popis stavu bufferov využíva dva ukazatele, ďalej nazývané ako HW TAIL ako ukazateľ na začiatok dát a HW HEAD ako ukazateľ na ich koniec v bufferi[2]. Oblasť medzi týmito dvomi ukazateľmi v tomto poradí teda reprezentuje dáta pripravené k spracovaniu a oblasť medzi ukazateľmi v opačnom poradí (ostatné miesto v kruhovom bufferi) je voľné miesto.

Podľa smeru spracovania dát RX alebo TX rozlišujeme dve chovania ovládača a hardvéru a rovnako týmito infixami označujeme aj príslušné ukazatele. Pri RX smere hardvér prijíma dáta cez sieťové rozhrania, ukladá ich do kruhového bufferu a upravuje pozíciu HW RX HEAD. Úlohou ovládača je po spracovaní dát posúvať HW RX TAIL ukazateľ, čím vznikne voľné miesto v bufferi, kde môžu hardvér opäť zapísať prijaté dáta. Pri TX smere ovládač po zápise dát do voľného miesta aplikáciami posúva HW TX HEAD ukazateľ. Hardvér postupne spracúva zapísané dáta, odosiela ich cez sieťové rozhrania a posúva HW TX TAIL ukazateľ.

Spätná komunikácia hardvéru smerom k ovládaču prebieha prostredníctvom prerušenia. Ovládač môže hardvéru zdeliť, že pri dosiahnutí určitých hodnôt ukazateľov HW RX HEAD a HW TX TAIL, ktorých posúvanie má hardvér nastarosti a ich zmeny sú pre ovládač

[2]ukazateľ HW HEAD zodpovedá ukazateľu SWEndPtr zo sekcie 2.6.1, HW TAIL zodpovedá ukazateľu SWStartPtr

významné, má byť informovaný prerušením. Tejto vlastnosti sa využíva pri implementácii systémového volania *poll()*, pokiaľ v kruhových bufferoch nie je dostatok dát, ovládač nastaví pozície pre vyvolanie prerušenia podľa požadovanej veľkosti bloku dát špecifikovanej hodnotou *poll threshold*. Proces nastavenia pozície v kruhovom bufferi, pri dosiahnutí ktorej má prísť k prerušeniu, sa nazýva aj ako nastavenie zarážky. Pokiaľ pri volaní *poll()* kruhový buffer neobsahuje dostatok dát, je aplikácia po dobu *poll timeout* blokovaná vo fronte spiacich procesov. Táto fronta je udržiavaná ako položka *poll_wait* štruktúry *struct szedata2*. Po obdržaní prerušenia sú všetky aplikácie blokované vo volaní *poll()* prebudené a opätovne je skontrolovaná veľkosť dostupných dát každou z nich.

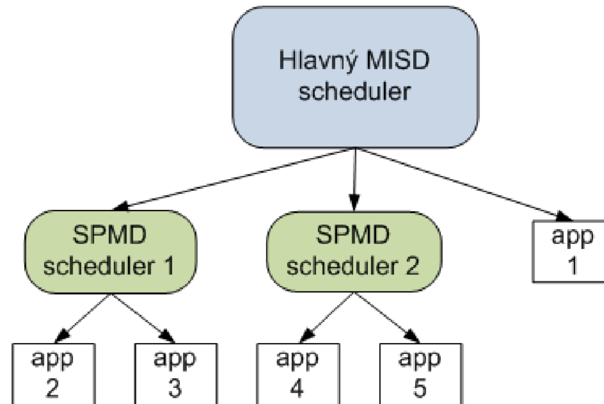
Na tomto mieste je nutné zmieniť s akými veľkými blokmi dát ovládač pracuje. Pri RX smere má ovládač jedinú informáciu o blokoch dát od hardvéru vyjadrenú pozíciou HW RX HEAD ukazateľa. Rozdelenie tohoto bloku na menšie bloky, pri použitom formáte dát, by znamenalo časovo náročné sekvenčné prechádzanie jednotlivých segmentov dát. Preto ovládač prideľuje bloky dát o veľkostiach pokrývajúcich rozsah po hodnotu ukazateľa HW RX HEAD. Pri TX smere je voľba veľkosti bloku v moci ovládača.

6.4.7 RX

Ovládač szedata2 podporuje dva druhy prístupu k dátam z pohľadu podpory paralelného spracovania dát. Tieto módy sa líšia v spôsobe, akým sa chová ovládač pri ioctl príkazoch SZE2_IOC_RXLOCKDATA , SZE2_IOC_RXUNLOCKDATA a pri systémovom volaní *poll()*. Rozdielne je prideľovanie dát aplikáciám a tiež prístup pri posúvaní HW RX TAIL ukazateľa. Implementácia podpory SPMD modelu vznikla v rámci tejto diplomovej práce. Keďže väčšia časť kódu pri obsluhe oboch módov RX je rovnaká, bola vytvorená abstrakcia nazvaná „scheduler" reprezentovaná štruktúrou *struct szedata2_sched*. Technikou ukazateľov na funkcie umožňuje definovať správanie špecifické pre rôzne módy obsluhy a tiež vytvára rozhranie pre rozšíriteľnosť do budúcnosti o iné módy spracovania dát. Ovládač naraz používa len jedinú abstrakciu „scheduleru", teda buď prideľuje dáta v móde MISD alebo SPMD pre všetky aplikácie. Pre možnosť výberu z dvoch módov RX prideľovania dát slúži ioctl príkaz SZE2_IOC_SETSCHED, ktorý je volaný v typickom scenári (podľa diagramu C.1) tesne pred ioctl príkazom SZE2_IOC_START.

Princíp práce ovládača pri volaní *poll()* (C.3) a ioctl príkazov SZE2_IOC_RXLOCKDATA (C.4) a SZE2_IOC_RXUNLOCKDATA (C.5) je zobrazený na UML diagramoch v prílohe C.

V rámci tejto práce bola rozvíjaná i myšlienka „spolupracujúcich schedulerov", ktorá predpokladá existenciu a prácu viacerých „schedulerov" naraz v hierarchickej stromovej štruktúre. Vízia takejto štruktúry je podaná na obrázku 6.6. Hlavný „scheduler" by bol vždy typu MISD aby mohli byť dáta spracúvané viacerými „schedulermi" i aplikáciami naraz. Na každý zo „schedulerov" by sa mohol pripájať iný „scheduler" či aplikácia. Rôzne aplikácie by mohli využívať rôzne typy prideľovania dát. Nutnosťou by bolo vytvoriť rozhranie v rámci ovládača, ktoré by zlučovalo aplikácie i „schedulery" do tzv. „scheduling element" abstrakcie, aby nadradený „scheduler" mohol so všetkými podriadenými štruktúrami pracovať jednotne. Po analýze tohoto prístupu však bolo usúdené, že výsledné riešenie by bolo značne zložitejšie ako to existujúce a zároveň by veľmi stúpla réžia synchronizácie, pretože by bolo nutné zosúladiť viacero spolupracujúcich komponentov. Neblahý by mohol byť i vplyv na výkonnosť riešenia. Nutná by bola i implementácia vytvárania nových „schedulerov", ich prepájanie do stromovej štruktúry a pripájanie aplikácii ku konkrétnym „schedulerom". Otázna je i použiteľnosť takéhoto riešenia. Môžeme predpokladať z účelu platformy NetCOPE ako hardvérového akcelerátoru sieťových aplikácií, že akcelerovaná bude jediná, časovo kritická aplikácia a teda takéto komplexné využitie súbežného spracovania dát nemusí



Obrázok 6.6: Vízia štruktúry „spolupracujúcich schedulerov"

byť potrebné.

6.4.7.1 MISD RX

MISD mód prideľovania dát zaručuje spracovanie všetkých dát všetkými aplikáciami súčasne. Aplikácia teda pri požiadaní o nové dáta dostane všetky dáta v kruhovom bufferi – od pozície ňou posledne spracovaných dát, ukazateľa APP RX HEAD, až po HW RX HEAD. Pri uvoľnení dát aplikáciou ovládač nastaví HW RX TAIL ukazateľ na hodnotu minima APP RX TAIL pre všetky klientské aplikácie. To znamená iterovanie cez frontu klientských aplikácií.

Pokiaľ systémové volanie *poll()* pri MISD móde uspeje, ovládač deteguje, že dostupná veľkosť dát je rovná či väčšia ako špecifikuje hodnota *poll threshold*, pri následnom požiadaní o dáta aplikácia vždy uspeje, je jej poskytnutý dostatočný počet dát. Toto správanie vyplýva z toho, že každej aplikácii sú dáta priraďované nezávisle na druhých.

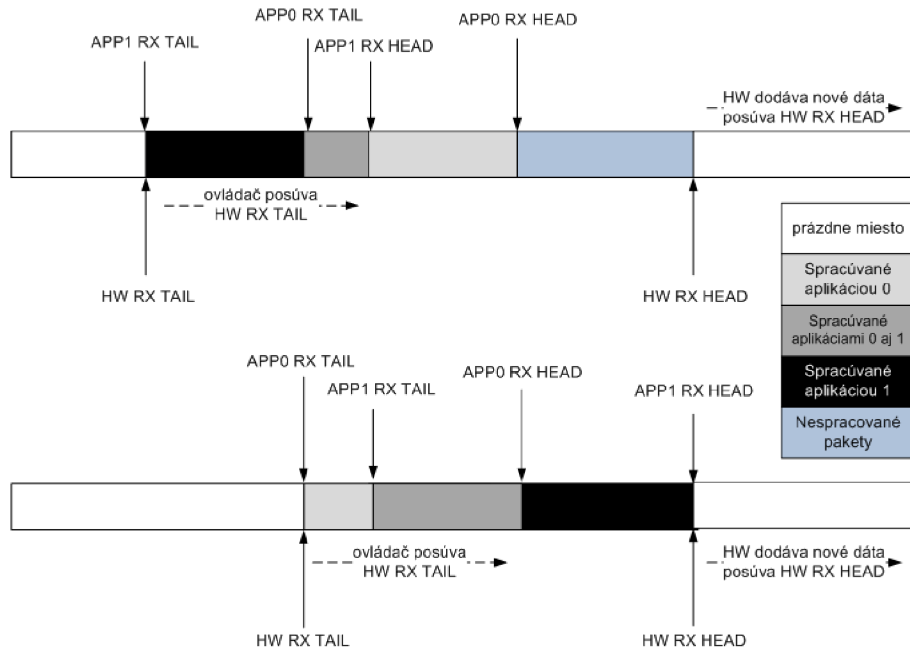
Obrázok 6.7 zobrazuje situáciu v kruhovom bufferi, keď aplikácia APP1 dokončí spracovanie jej bloku dát. Ovládač jej poskytne ďalší blok dát od ukazateľa APP1 RX HEAD až po ukazateľ HW RX HEAD.

6.4.7.2 SPMD RX

SPMD mód prideľovania dát zaručí, že každý segment dát je vždy spracovaný len jedenkrát, jedinou aplikáciou. Prideľovanie dát už nenastáva v závislosti od ich APP RX TAIL a HEAD ukazateľov, ale nastáva v závislosti na globálnej pozícii dát už pridelených k spracovaniu aplikáciám. Túto pozíciu ovládač uchovával vo svojej štruktúre *struct szedata2_sched* a v nasledujúcom texte ju nazývame SCHED RX HEAD ukazateľ. Aplikácia žiadajúca dáta teda dostane všetky dáta od SCHED RX HEAD ukazateľa až po HW RX HEAD pričom SCHED RX HEAD bude posunutý na hodnotu HW RX HEAD, čo znamená, že všetky dáta až po tento ukazateľ sú momentálne spracúvané niektorou z aplikácií.

Pri uvoľnení dát aplikáciou je jej hodnota APP RX TAIL nastavená na hodnotu APP RX HEAD. Pokiaľ sú tieto dva ukazatele aplikácie zhodné, znamená to, že aktuálne nespracúva žiaden blok dát. Pri uvoľňovaní dát ovládač posunie hodnotu HW RX TAIL ukazateľa na minimum z APP RX TAIL z aplikácií, ktoré momentálne spracúvajú niektorý z blokov.

Systémové volanie *poll()* pri SPMD móde sa správa inak ako pri predchádzajúcom móde. Keďže prideľovanie dát je závislé od viacerých aplikácií využívajúcich rozhranie, nie je zaručené, že keď volanie *poll()* uspeje v momente keď ovládač deteguje dostupnosť dostatočného množstva dát, nasledujúca požiadavka o tieto dáta bude uspokojená. Na úkor



Obrázok 6.7: MISD RX

neuspokojenej aplikácie môžu byť dáta pridelené inej aplikácii, ktorá o ne požiadala v momente medzi ukončením volania *poll()* a volaním ioctl príkazu SZE2_IOC_RXLOCKDATA neuspokojenou aplikáciou.

SPMD mód prideľuje aplikáciám bloky dát rôznych veľkostí. Je to spôsobené princípom komunikácie medzi ovládačom a hardvérom. Hardvér informuje prerušením ovládač pri dosiahnutí zarážky v kruhovom bufferi. V tomto momente ovládač musí získať pozície hardvérových ukazateľov pričom nie je zaručené, že sa od momentu signalizácie prerušenia nezmenili a teda požadovaná veľkosť bloku dát sa mohla navýšiť.

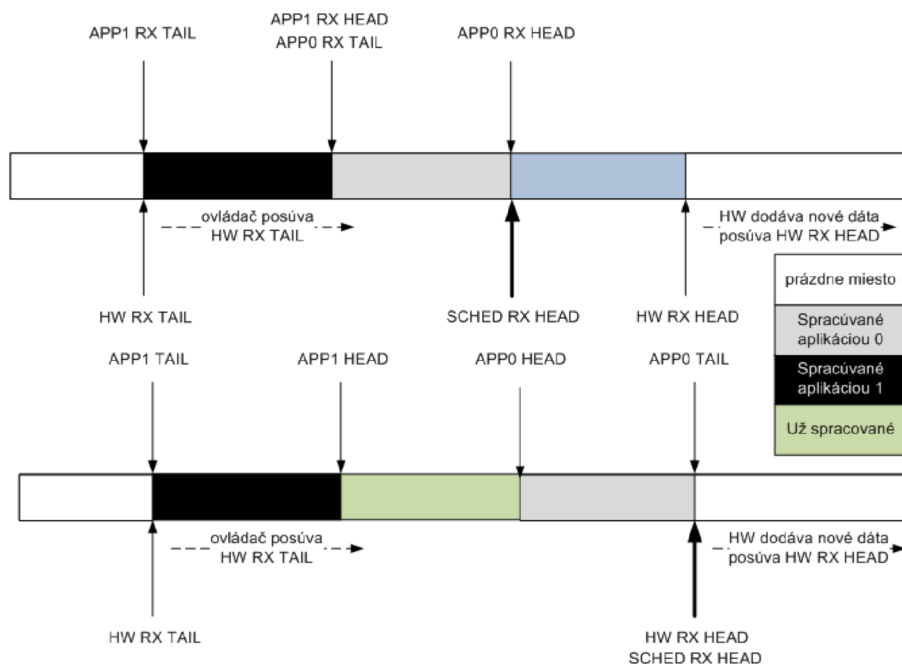
Obrázok 6.8 ukazuje situáciu v kruhovom bufferi keď aplikácia APP0 ukončila spracovanie dát jej prideleného bloku. Po požiadaní o ďalší blok sú jej pridelené dáta od ukazateľa SCHED RX HEAD až po HW RX HEAD, pričom je SCHED RX HEAD posunutá na túto hodnotu.

6.4.8 TX

Zápis dát, všeobecne kvôli jeho podstate, nemôže prebiehať paralelne v rovnakom priestore na rozdiel od čítania (MISD RX mód). Ovládač preto poskytuje pre zápis iba bloky kruhového bufferu určitej veľkosti. Pokiaľ ovládač zaručí, že sa bloky neprekrývajú, môže zápis prebiehať súčasne. Prideľovanie blokov dát pre zápis ovládačom veľmi pripomína SPMD RX mód, len rola ovládača a hardvéru je rozdielna.

Ovládač musí poznať globálnu pozíciu závislú od aplikácií momentálne zapisujúcich dáta. Táto pozícia ukazuje až za posledný bajt rezervovaný niektorou z aplikácií k zápisu a ďalej ju budeme nazývať SCHED TX HEAD. Pokiaľ aplikácia žiada o pridelenie miesta k zápisu, ovládač jej poskytne blok dát špecifikovanej veľkosti od pozície SCHED TX HEAD, pokiaľ sa v kruhovom bufferi nachádza dostatok voľného miesta – veľkosť od SCHED TX HEAD po HW TX TAIL.

Podobne ako pri SPMD RX móde pri uvoľnení dát ovládač nastaví APP RX TAIL na hodnotu APP_TX_HEAD, čím značí, že aplikácia momentálne nezapisuje. Zároveň



Obrázok 6.8: SPMD RX

ovládač posunie ukazateľ HW TX HEAD na hodnotu minima APP TX TAIL z momentálne zapisujúcich aplikácií, čím hardvéru dáva najavo, že dáta až po túto hodnotu sú pripravené k spracovaniu.

Princíp práce ovládača pri volaní ioctl príkazov SZE2_IOC_TXLOCKDATA (C.6) a SZE2_IOC_TXUNLOCKDATA (C.7) je zobrazený na UML diagramoch v prílohe C.

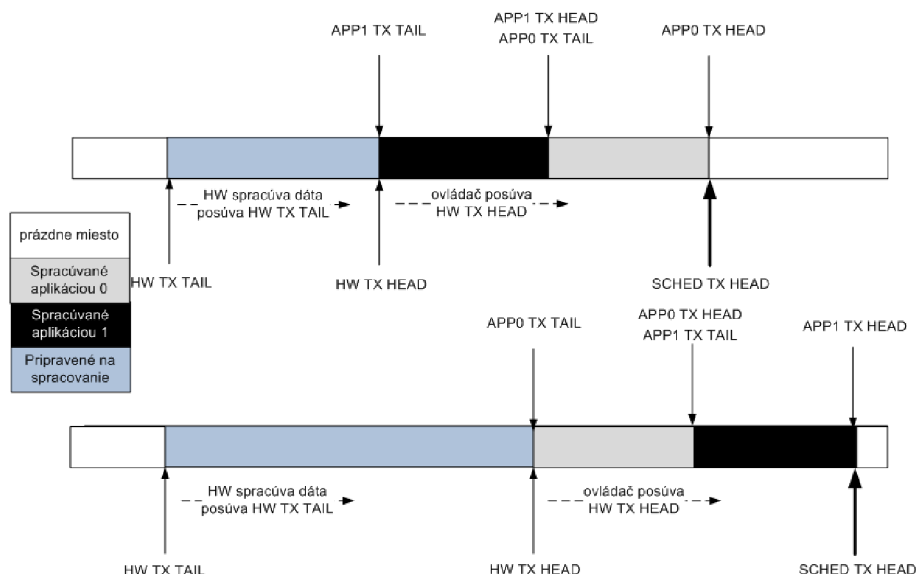
Pokiaľ veľkosť zapísaných dát aplikáciou je menšia ako veľkosť poskytnutého bloku dát, ovládač musí túto medzeru špecifikovať, aby firmvér správne klasifikoval túto oblasť a preskočil ju. Táto oblasť môže byť špecifikovaná ako špeciálny szedata2 paket, ktorého formát bol prezentovaný v časti práce 4.2.2. Spracovanie tejto medzery v zmienenom formáte nie je v súčasnej verzii firmvéru podporované. Po pridaní tejto funkcionality bude ovládač upravený pre jej pokrytie.

Správanie systémového volania *poll()* pri TX smere je rovnaké ako pri SPMD móde RX. I tu platí, že prideľovanie dát je závislé na všetkých aplikáciách využívajúcich rozhranie a preto nie je po úspešnom volaní *poll()* zaručený úspech pri následnom pokuse o poskytnutie priestoru pre zápis.

Obrázok 6.9 ukazuje situáciu v kruhovom bufferi keď jedna z dvoch aplikácií, APP1, ukončí zapisovanie dát v jej pridelenom bloku. Pri žiadaní o ďalší priestor ovládač poskytne blok špecifikovanej veľkosti od ukazateľa SCHED TX HEAD a zároveň mu nastaví novú hodnotu. Posunutím ukazateľa HW TX HEAD zdelí hardvéru, že sú pre spracovanie dostupné ďalšie dáta.

6.4.9 Synchronizácia

Keďže v rámci ovládača viacero bežiacich procesov súťaží o spoločné zdroje pre zaručenie determinizmu je nutné ochrániť tieto zdroje synchronizačnými prostriedkami. V tejto sekcii budú v tabuľke 6.1 stručne zhrnuté zdieľané zdroje, ich funkcia a synchronizačný prostriedok, ktorý je použitý pre ich ochranu a granularita synchronizácie.



Obrázok 6.9: Paralelný TX

Zdroj	Funkcia	Synch. prostriedok	Typ synch. prostriedku	Granularita
szedata2_minors (globálna premenná)	registrácia zariadení	szedata2_lock (globálna premenná)	struct mutex	globálna
inštancia štruktúry struct szedata2	funkcie jedného zariadenia	lock (člen štruktúry struct szedata2)	struct mutex	per zariadenie
fronta klientských aplikácií	fronta klientských aplikácií	list_lock (člen štruktúry struct szedata2)	rwlock_t	per zariadenie
SCHED RX HEAD, APP RX HEAD, APP RX TAIL	pozície v kruhovom bufferi RX	rx_lock (člen štruktúry struct szedata2)	spinlock_t	per zariadenie
SCHED TX HEAD, APP TX HEAD, APP TX TAIL	pozície v kruhovom bufferi TX	tx_locks[] (člen štruktúry struct szedata2)	spinlock_t	per TX buffer

Tabuľka 6.1: Zdieľané zdroje a synchronizačné prostriedky ovládača szedata2

Kapitola 7

Knižnice libsze2 a PCAP

V tejto kapitole bude popísaný postup vývoja knižnice libsze2 a rozšírenie PCAP knižnice o podporu szedata2 rozhrania. Budú tu zmienené hlavné úlohy knižníc, súčasný stav a nutné zmeny do budúcnosti.

7.1 Libsze2

Knižnica libsze2 poskytuje užívateľským aplikáciám rozhranie pre využívanie szedata2 prenosov jednoduchým a užívateľsky prívetivým spôsobom.

7.1.1 Úlohy a štruktúra knižnice

Knižnica je implementovaná v jazyku C. Logicky je členená do niekoľkých modulov, ktoré sa starajú o splnenie úloh vyžadovaných pre prácu s rozhraním szedata2. Základný modul nazývaný „common” obsahuje nízkoúrovňové funkcie, ktoré obaľujú systémové volania ovládača szedata2. Ďalej tento modul obsahuje inicializačné funkcie vyššej úrovne, ktoré sú používané pri inicializácii akejkoľvek práce s rozhraním. Jednou zo základných úloh je tiež rozparsovanie szedata2 paketov, ktoré aplikáciám zaručí jednoduchý prístup a prácu s jednotlivými zložkami paketu. Navyše sa v module nachádzajú funkcie slúžiace pre ladiace účely.

Moduly „read” a „write”, ako ich názov naznačuje, sústreďujú v sebe funkcionalitu týkajúcu sa separátne prijímania a odosielania dát. Hlavičkový súbor *private.h* obsahuje štruktúry, ktoré sú viditeľné len v rámci knižnice libsze2. Tento súbor nie je používaný užívateľskými aplikáciami a zaručuje zapúzdrenie štruktúr, ktoré aplikácia môže používať pre volania knižničných funkcií, avšak nemôže priamo manipulovať so zložkami štruktúr.

Typický scenár práce s knižnicou libsze2 je zobrazený sekvenčným diagramom C.8.

7.1.2 Módy prenosu dát

Pri prenose dát pomocou szedata2 je nutné sa zmieniť o dvoch druhoch používaných prenosov. Prvým z nich je tzv. „single” prenos, pri ktorom pre prenos každého szedata2 paketu prebieha komunikácia s ovládačom a každý paket je spracovaný separátne. Tento mód vykazuje vysokú réžiu spojenú s prenosom avšak prináša nízku latenciu a umožňuje rýchlo reagovať na základe spracúvaných dát. Druhým typom prenosu je takzvaný „burst” prenos označovaný aj ako prenos agregovaných dát. Pri tomto druhu prenosu je pri jednej interakcii s ovládačom a následne hardvérom prenesených viacero szedata2 paketov naraz. „Burst” prenosy prinášajú vysokú priepustnosť, avšak spôsobujú vyššiu latenciu a pri veľkých hodnotách bloku

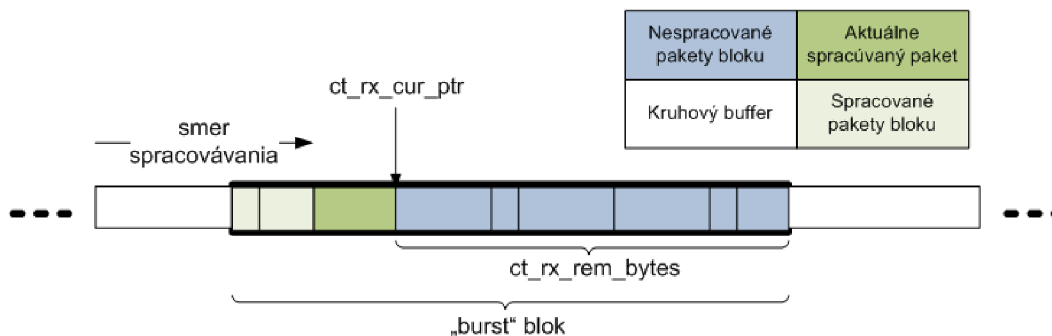
prenášaného v rámci jedného prenosu, môžu vykazovať vyššie nároky na veľkosť kruhových bufferov.

Príjem dát implicitne využíva „burst” prenosy. Je to dané princípom práce firmvéru a ovládača szedata2, ktorému je pomocou hardvérových ukazateľov zdelený blok momentálne všetkých prístupných dát v bufferi. Granularita interakcie hardvéru a ovládača v RX smere je teda na úrovni súvislých blokov szedata2 paketov. Naopak granularita TX smeru je plne v moci ovládača, respektíve aplikácií využívajúcich tento ovládač. Podľa potreby aplikácie môžu používať „single” i „burst” typ odosielania dát.

Knižnica libsze2 ponúka jednoduché rozhranie pre využívanie týchto typov prenosov popísané v prílohe A. Implementácia „single” prenosov je menej náročná, keďže každý szedata2 paket je spracovaný osobitne. V prípade „burst” prenosov je však situácia mierne komplikovanejšia, keďže knižnica musí udržiavať aktuálnu pozíciu aplikácie v rámci bloku aktuálne spracúvaných dát. Táto pozícia je v rámci knižnice nazývaná ako kontext aplikácie. Keďže aplikácia prijímajúca dáta na viacerých rozhraniach zároveň môže spracúvať okamžite len jediný blok dát na jednom rozhraní (čo vyplýva z princípu práce ovládača), kontext je teda nutné udržiavať len pre tento blok dát. Pri odosielaní dát však aplikácia môže postupne napĺňať viacero blokov dát prislúchajúcim rôznym rozhraniam naraz. Preto je pri TX smere nutné udržiavať kontext prenosu dát aplikácie pre všetky rozhrania samostatne.

Najjednoduchším prostriedkom jazyka C pre udržiavanie kontextu v priebehu viacnásobného volania funkcií je použitie statických premenných keď definícii premennej predchádza kľúčové slovo „static”. Tento prístup je veľmi jednoduchý, avšak kód nie je reentrantný a knižnica môže byť využívaná nanajvýš jediným vláknom súčasne. Preto premenné udržujúce kontext prenosov boli zahrnuté do štruktúry *struct szedata*, ktorá je vytvorená pre každé vlákno využívajúce knižnicu zvlášť pri inicializácii. S využitím tejto štruktúry spracovanie „burst” RX prenosu prebieha nasledovne. Aplikácia volá knižničnú funkciu *szedata_read_next()* pre obdržanie jediného paketu. Pokiaľ v kontexte prenosu dát aplikácie neexistuje žiadny rozpracovaný blok dát, knižnica pomocou ioctl príkazu SZE2_IOC_RXLOCKDATA obdrží povolenie pre prístup k bloku dát od ovládača. Kontextová premenná *ct_rx_cur_ptr*, ktorá značí aktuálnu pozíciu v bloku dát, je nastavená na počiatok bloku a kontextová premenná *ct_rx_rem_bytes*, ktorá udáva počet ostávajúcich bajtov dát v rámci bloku, je nastavená na veľkosť bloku dát. Aplikácii je poskytnutý prvý szedata2 paket bloku pre spracovanie, kontextové premenné sú adekvátne upravené. Pri následných volaniach sú pakety bloku poskytované aplikácii bez komunikácie s ovládačom až pokým nie je hodnota kontextovej premennej *ct_rx_rem_bytes* nulová. To indikuje koniec spracovania bloku dát a pri následnom volaní funkcie *szedata_read_next()* je spracovaný blok dát uvoľnený volaním ioctl príkazu SZE2_IOC_RXUNLOCKDATA. Situácia v kruhovom bufferi pri spracovaní „burst” bloku dát je zobrazená na obrázku 7.1.

Princíp „burst” prenosov pri TX smere je skoro rovnaký ako pri RX. Proces pracuje s rovnomennými premennými avšak s prefixom *ct_tx*, ktoré sú polia týchto kontextových premenných pre všetky výstupné rozhrania. Aplikácia volá funkciu *szedata_write_next_burst()* s argumentom číslom rozhrania, cez ktoré má byť paket odoslaný. Knižnica požiada ovládač ioctl príkazom SZE2_IOC_TXLOCKDATA o priestor o veľkosti „burst” bloku, len pokiaľ na tomto rozhraní nie je rozpracovaný prenos bloku dát. Pokiaľ je priestor poskytnutý ovládačom dostatočný pre zápis paketu, je táto operácia vykonaná a príslušné kontextové premenné sú upravené. Pri následných volaniach funkcie sú dáta zapisované do poskytnutého bloku až pokým ostávajúci priestor v bloku je nedostatočný pre aktuálny paket. Vtedy knižnica ioctl príkazom SZE2_IOC_TXUNLOCKDATA oznámi ovládaču ukončenie spracovania bloku dát, skutočne využitá veľkosť bloku je predaná ako argument a dáta sú prístupné pre spracovanie hardvéru. Pokiaľ chce aplikácia odoslať dáta v rozpracovanom bloku



Obrázok 7.1: RX „burst” prenos - situácia v kruhovom bufferi

predčasne, alebo pokiaľ aplikácia ukončí prácu so szedata2 rozhraním, rozpracovaný blok je odoslaný volaním funkcie *szedata_burst_write_flush*(). „Single” prenos v TX smere je realizovaný funkciou *szedata_try_write_next()*. Explicitnú veľkosť bloku „burst” prenosu pri TX je možné v aplikácii špecifikovať premennou prostredia *SZE2_BURST_WRITE_SIZE* pri inicializácii knižnice.

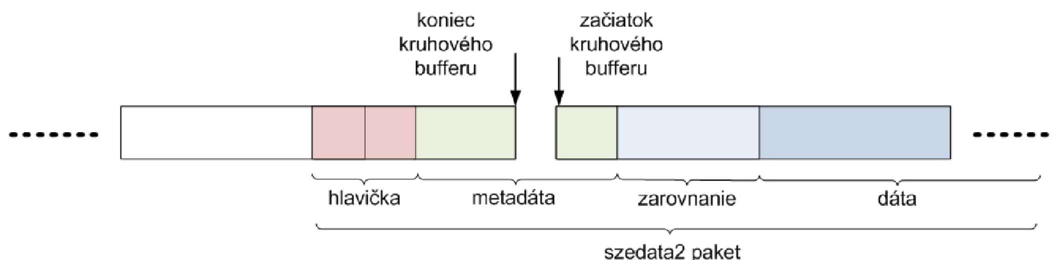
S veľkosťami „burst” blokov súvisí hodnota *poll threshold*, ktorá špecifikuje minimálnu veľkosť bloku dát, ktorú aplikácia vyžaduje od ovládača. Táto hodnota je definovaná aplikáciou pri inicializácii – pri volaní funkcie *szedata_subscribe*(), zvlášť pre RX aj TX smer ako argumenty funkcie. Pre testovacie nástroje, ktoré ako tieto argumenty používajú knižničné premenné reprezentujúce tieto hodnoty, možno pri inicializácii knižnice nastaviť hodnoty *poll threshold* premennými prostredia *SZE2_RX_POLL_CNT* a *SZE2_TX_POLL_CNT*.

7.1.3 Blokujúce a neblokujúce operácie

V teórii informačných technológií sa často používa pojmu blokujúce a neblokujúce operácie. V oblasti operačných systémov blokujúcou operáciou rozumieme operáciu, ktorá spôsobí spánok – odstavenie procesu od procesoru, kým nie je operácia ukončená. Neblokujúca operácia vráti riadenie volajúcemu okamžite, v prípade neúspechu môže vykonávať inú činnosť. Aplikácie môžu vyžadovať rôzne správanie operácií prenosu dát cez rozhranie szedata2. Blokovanie funkcií libsze2 spôsobuje systémové volanie *poll()* ovládača szedata2. Chovanie tohoto volania môže byť ovplyvnené jeho parametrom *poll timeout*, ktorý udáva maximálnu dobu, po ktorú môže byť proces blokovaný vo volaní. Nulová hodnota tohoto argumentu spôsobí neblokujúce chovanie tohoto volania a tým aj neblokujúce chovanie všetkých funkcií knižnice. Aby bolo možné hodnotu *poll timeout* zmeniť, knižnica pri jej inicializácii umožňuje túto hodnotu definovať, pre každý smer zvlášť, nastavením premenných prostredia *SZE2_RX_POLL_TIMEOUT* a *SZE2_TX_POLL_TIMEOUT*.

7.1.4 Zalomenie kruhového bufferu

Funkcie knižnice pre obsluhu RX a TX prenosu dát pracujú priamo s dátami v kruhovom bufferi. Pri príjme dát sú knižnicou aplikácii poskytnuté ukazatele na polia metadáta a dáta priamo ukazujúce do priestoru kruhového bufferu. Spolu s týmito ukazateľmi sú aplikácii predané aj údaje o veľkostiach týchto polí. Pri odosielaní dát cez rozhranie aplikácia špecifikuje ukazatele na príslušné polia, ich veľkosti a úlohou knižnice je vytvoriť v dostupnom priestore kruhového bufferu szedata2 paket s týmito poľami. Je nutné si uvedomiť, že oveľa jednoduchšie by bolo vytvoriť szedata2 paket mimo priestor kruhového bufferu, respektíve pri príjme skopírovať obsah kruhového bufferu do privátneho bufferu knižnice a následne



Obrázok 7.2: Rozdelenie paketu v časti metadát na konci kruhového bufferu

paket spracovať, či už kopírovaním do TX kruhového bufferu, alebo predaním paketu pre spracovanie aplikácii. Keďže poznáme počiatok a veľkosť paketu, prenos paketu by si vyžiadal jediné volanie funkcie *memcpy*(), v prípade že je paket v kruhovom bufferi rozdelený, museli by sa obe časti kopírovať zvlášť, čiže dohromady by si kopírovanie vyžiadalo dve volania *memcpy*(). Táto jednoduchá implementácia by si však vyžiadala nadbytočné kopírovanie všetkých dát medzi kruhovým bufferom a privátnym bufferom knižnice, čo by malo neblahý vplyv na výkonnosť a šlo by proti princípom rýchlych SZE prenosov.

Z týchto dôvodov knižnica používa jemne zložitejšiu implementáciu, ktorá umožní vyhnutie sa kopírovaniu medzi buffermi. Pri nerozdelených paketoch v kruhových bufferoch, ich čítanie i vytváranie nie sú zložité operácie. Zložitejšie prebiehajú prenosy rozdelených paketov ležiacich na konci i začiatku kruhového bufferu. Tento adresový priestor nie je spojitý. Obrázok 7.2 ukazuje szedata2 paket rozdelený na konci kruhového bufferu v časti metadát paketu. Pri príjme dát je takýto rozdelený paket najskôr po častiach skopírovaný do privátneho bufferu knižnice, ktorý je vytvorený ako kontext každej aplikácie a je súčasťou štruktúry *struct szedata*. Tento už spojitý paket môže byť rozparsovaný a predaný aplikácii k spracovaniu. Pri odosielaní, zostavovaní szedata2 paketu priamo v kruhovom bufferi, je nutné počítať s tým, že szedata2 paket môže byť rozdelený kdekoľvek v rámci tohoto paketu. Preto je nutné pokryť všetky možnosti rozdelenia paketu. Tieto úlohy, aj keď sa nejavia ako priveľmi komplikované, sú kritickým miestom v knižnici libsze2, nakoľko už niekoľko odhalených chýb malo pôvod práve v tejto časti kódu. Závažnosť chýb zvyšuje ich veľmi obtiažna reprodukcia a detekcia. Nie je zriedkavé, že sa chyba vyskytne pri jedinom zo stámiliónov prijatých či odoslaných paketov. Na odhaľovaní takýchto chýb sa podieľajú členovia týmu od vývojárov firmvérových komponentov, cez testerov, vývojárov ovládačov a softvéru.

Vďaka vylepšeniu „buffer oversize” ovládača popísaného v odstavci 6.4.4 už však nie je nutné prevádzať operácie pri rozdelení paketu na konci a začiatku kruhového bufferu keďže i priestor na konci kruhového bufferu sa knižnici i aplikáciám javí ako spojitý. Toto vylepšenie umožnilo výrazne zredukovať kód knižnice libsze2 pre obsluhu RX i TX prenosov a prispelo k vyššej výkonnosti. Zmena výkonnosti je zmeraná v nasledujúcej kapitole.

7.1.5 Vývoj a súčasný stav

Vývoj knižnice libsze2 započal diplomovou prácou Jiřího Slabého [2], ktorý ako súčasť práce implementoval nízkoúrovňové funkcie pre prácu s rozhraním szedata2. V priebehu leta 2008 bola knižnica libsze2 intenzívne vyvíjaná a menená v spolupráci s členmi týmu zodpovednými za vývoj firmvéru, keďže sú obe oblasti úzko prepojené. Bolo vytvorené rozhranie vyššej úrovne, niekoľko funkcií pre ladenie a knižnica si podobu získanú počas tohoto obdobia ponechala dodnes. Vývoj knižnice sa na istú dobu utlmil, nakoľko sa riešenie javilo ako dostatočne funkčné a bolo použité v projektoch NIC a NIFIC projektu Liberouter. Ďalší

vývoj, najmä ladenie a opravy chýb a analýza kľúčových hodnôt nastal pred niekoľkými mesiacmi, keďže sa firmvér platformy NetCOPE nad kartou COMBOv2 dostal do štádia značnej funkčnosti. Táto situácia si vyžiadala vytvorenie a systematické intenzívne prevádzanie testov nad platformou, následnú správu a opravu chýb. Toto úsilie bolo vyvíjané najmä testerskou skupinou. Určité zmeny si vyžiadala aj podpora rozhrania PCAP.

V rámci tejto práce bol kód knižnice libsze2 revidovaný a tieto zmeny majú zaručiť reentrantnosť a vláknovú bezpečnosť knižnice (o týchto vlastnostiach pojednáva sekcia 3.5.4). Úpravy spočívali zväčša v odstránení statických premenných, ich funkcionalita bola prevedená na členy štruktúry *struct szedata*, ktorá vytvára kontext aplikácie v knižnici libsze2. Implementácia SPMD RX módu ovládača szedata2 si vynútila rozšírenie rozhrania o možnosť voľby SPMD či MISD módu príjmu dát. Nakoniec implementácia vylepšenia „buffer oversize” spôsobila výraznú redukciu kritického kódu knižnice vykonávajúceho príjem a odosielanie dát.

Úpravy knižnice vykonané ako súčasť vylepšení ovládača szedata2 existujú len vo vedľajších vývojových vetvách, keďže kvôli kontinuite vývoja na projekte nie je možné stabilnú verziu knižnice nahradiť novou. Integrácia týchto zmien do stabilnej verzie knižnice bude v budúcnosti nutná.

7.2 Rozšírenie PCAP

Rozšírenie knižnice PCAP, predstavenej v časti práce 2.6.4.2, pre prácu so szedata2 rozhraním znamená realizovať rozhranie knižnice PCAP prostriedkami knižnice libsze2. Toto rozšírenie je označované aj názvom libpcap-sze. Podobné rozšírenia už v knižnici PCAP existujú, napríklad od spoločnosti Endace pre prácu s DAG kartami a knižnica má pre tento účel definované rozhranie. Pridanie rozšírenia znamená používanie špecifických funkcií špecifického rozhrania pre vybavenie štandardných funkcií knižnice PCAP. Toto sa realizuje známou technikou ukazateľov na funkcie, kde každé rozšírenie nastaví ukazatele na funkcie štruktúry *struct pcap* na funkcie špecifické pre dané rozhranie. Rozšírenie nemusí nevyhnutne podporovať celé rozhranie, nepodporované operácie sú špecifikované nulovým príslušným ukazateľom. Rozhodnutie o tom, ktoré sady funkcií použiť pre špecifikované rozhranie pri inicializácii knižnice PCAP je realizované na základe názvu rozhrania. Rozhranie je špecifikované krátkym reťazcom a číslom (typicky napríklad *eth0*). Pokiaľ chce aplikácia cez PCAP používať szedata2 rozhranie, použije ako prefix názvu rozhrania reťazec „sze”, nasledujúca číslica špecifikuje číslo akceleračnej karty na danom stroji. Reťazec *sze0* použitý pri inicializácii knižnice PCAP zabezpečí prácu so všetkými rozhraniami na prvej akceleračnej karte. Názov PCAP rozhrania ďalej môže špecifikovať i konkrétne číslo rozhrania na danej akceleračnej karte. Reťazcu sa priradí sufix s dvojbodkou a číslom konkrétneho rozhrania. Reťazec *sze0:1* špecifikuje druhé rozhranie na prvej akceleračnej karte.

Knižnica PCAP vďaka jej široko používanému rozhraniu umožňuje podporu prenosov szedata2 pre už existujúce aplikácie. Táto podpora však spôsobuje vyššie nároky na spotrebovaný výpočtový výkon.

Kapitola 8

Vývoj, testovanie a ladenie výkonnosti

V tejto kapitole budú podané podrobnosti vývoja, testovania a ladenia výkonnosti rýchlych prenosov medzi hardvérom a aplikáciami na platforme NetCOPE. Ďalej tu budú popísané použité softvérové a hardvérové nástroje a významné hodnoty majúce vplyv na celkovú výkonnosť riešenia. V závere budú v tabuľkách zhrnuté dosiahnuté výsledky pri zvolených významných hodnotách.

Prostredie projektu Liberouter Na projekte Liberouter pracuje niekoľko desiatok vývojárov, najmä študentov a akademických pracovníkov. V rámci projektu sa vyvíja niekoľko rôznych sieťových zariadení – projektov. Členovia sú primárne organizovaní do skupín podľa zamerania ich činností (firmvér, softvér, testovanie, verifikácia a iné) v rámci projektu. Sekundárne sú z členov týchto skupín vytvorené tímy riešiace niektorý z projektov. Pre spoluprácu a komunikáciu medzi členmi tímu slúži viacero nástrojov s rôznymi funkciami. Vývoj prebieha s pomocou nástrojov pre správu revízií zdrojových kódov. SVN[1] je používaná pre správu zdrojových kódov firmvéru a tiež softvéru. Pre každý projekt existuje samostatný SVN modul, pre zdieľanie spoločného kódu spoločných komponentov medzi viacerými projektmi sa využíva vlastnosť SVN „svn:externals". Všetky softvérové nástroje sú vyvíjané v samostatnom SVN module. Vývoj ovládačov bol pred niekoľkými mesiacmi presunutý z nástroja SVN do nástroja git kvôli do budúcnosti plánovanému začleneniu ovládačov do jadra systému Linux, ktoré využíva tento nástroj. Pri vývoji sa všeobecne využíva vytváranie vývojových vetiev („branch"). Vetvy slúžia primárne pre stabilizáciu, vyladenie chýb určitej verzie projektu, a hlavná vetva (označovaná ako „trunk" alebo „master") slúži pre pridávanie novej funkcionality.

Členova tímu medzi sebou komunikujú prostredníctvom niekoľkých „mailing listov" rozdelených podľa projektov alebo tém. Správy sa týmto spôsobom dostanú ku všetkým zainteresovaným členom projektu a ostávajú archivované. Instantnú komunikáciu zabezpečuje projektový IM jabber server. Pre zhromažďovanie dôležitých informácií, postupov, návodov a pripomienok slúži interný CMS[2] označovaný ako wiki. Pre koordináciu činností členov, priraďovanie jednotlivých úloh, evidenciu a správu chýb na projekte slúži systém Trac. Tiež je tu vedené jednoduché plánovanie vývoja projektov.

Hardvérové zázemie projektu zahŕňa desiatky pomerne výkonných serverov, pričom niektoré z nich sú vybavené hardvérovými akceleračnými kartami. Práca s týmito strojmi pre-

[1] Subversion - subversion.trigris.org

[2] Content Management System

bieha vzdialene cez ssh[3]. Nutnosťou pre vývoj a testovanie je aj sieťová infraštruktúra zabezpečujúca prepojenie hardvérových akceleračných kariet rôznymi médiami so zdrojmi sieťovej premávky. Pre jednoduché testy možno využiť ako zdroj jeden z prepojených serverov a zasielať špecifikovanú premávku nástrojom *tcpreplay*, naopak pre príjem a analýzu premávky vygenerovanej akceleračnou kartou je používaný nástroj *tcpdump* alebo *wireshark*. Pre zložitejšie testy sa na úkony generovania a analýzy sieťovej premávky využíva zariadenie Spirent AX/4000. Toto špecializované hardvérové zariadenie sa štandardne používa pre analýzu širokopásmových sietí. Obsahuje niekoľko portov schopných generovať sieťovú premávku o toku až 10 Gb/s v závislosti na použitom médiu. Zariadenie je vzdialene konfigurovateľné prostredníctvom rozhrania v jazyku tcl[4]. Toto rozhranie je používané testerskou skupinou pre vytváranie automatizovaných testov.

Lokalizácia chýb Pri ladení a testovaní platformy NetCOPE z pohľadu vývojára softvérovej vrstvy je nutné rozlíšiť tri základné vrstvy platformy, každá z nich môže byť potenciálnym zdrojom chyby. Bohužiaľ neplatí že by ich bolo možné testovať samostatne, aj keď niekedy je možné niektorú vrstvu z tohoto procesu vynechať. Pri pohľade zdola nahor prvú vrstvu vytvára firmvér, ďalej naväzuje vrstva systémových ovládačov a poslednú vrstvu tvorí vrstva knižníc. Kedže pre ladenie a testovanie sa používajú vytvorené nástroje, tieto samotné sú možným zdrojom chyby.

8.1 Nástroje

Pre testovanie a ladenie platformy NetCOPE vzniklo niekoľko nástrojov, väčšina z nich v rámci tejto práce. Sú implementované v jazyku C a využívajú rozhranie príkazového riadku, ktoré nie je interaktívne a zväčša obsahujú množstvo prepínačov, ktoré špecifikujú ich argumenty a chovanie. Toto rozhranie je vhodné pre následné skriptovanie. Nástroje tvoria ďalej základ pre komplexné automatizované testy vytvárané testerskou skupinou. Nástroje a naväzujúce testy sa sústreďujú na dve základné vlastnosti platformy. Prvou z nich je overenie správnej funkčnosti prenosu dát. Druhou meranou veličinou je výkonnosť – priepustnosť. Testovanie a ladenie pre RX a TX smer prebieha zväčša nezávisle. Je nutné tiež podotknúť že nástroje pracujú na linkovej vrstve a teda spracúvajú rámce linkovej vrstvy, ktoré tvoria súčasť szedata2 paketov. Ďalej bude podaný výčet a vlastnosti použitých nástrojov.

8.1.1 szetest.ko

Tento testovací ovládač simuluje funkcionalitu hardvérovo závislého ovládača. Pre simuláciu príjmu a vysielania dát cyklicky pomocou časovačov generuje dáta a posúva hardvérovo závislé ukazatele do kruhových bufferov. Tento nástroj slúži primárne pre ladenie hardvérovo nezávislého ovládača. Výhodou je, že ladenie môže prebiehať aj na stroji bez hardvérovej akceleračnej karty. Nástroj pomáha oddeliť hardvér od softvérovej vrstvy a teda umožňuje ladenie chýb v nej.

8.1.2 szetest2

szetest2 prijíma dáta od ovládača, počíta počet prijatých szedtata2 paketov a odosiela náhodne generované dáta vo forme szedata2 paketov. Tento nástroj je nízkoúrovňový – komunikuje s ovládačom priamo cez systémové volania a nepoužíva vyššie ležiace knižnice

[3]Secure Shell
[4]Tool Command Language - dynamický skriptovací jazyk používaný testerskou skupinou

softvérovej vrstvy. Slúži pre overenie základnej funkcionality ovládačov a firmvéru platformy. Niekedy je užitočný pre lokalizáciu chýb vo vyššie ležiacich knižniciach.

8.1.3 sze2read

Tento jednoduchý nástroj prijíma a počíta dáta ako szedata2 pakety. Pracuje cez rozhrania knižníc libsze2 a PCAP. Pri testoch slúži na overenie toho, že všetky rámce odoslané do hardvérovej akceleračnej karty sú preposlané až do softvéru, pričom sa porovnáva len ich počet, nie identita.

8.1.4 crctest

Tento nástroj prijíma dáta, počíta kontrolný súčet CRC32 z obsahu rámca a porovnáva ho s touto hodnotou priamo v rámci. Preposielanie CRC32 rámca do softvéru vyžaduje špeciálne nastavenie firmvéru, implicitne je totiž toto pole vo firmvéri, v jeho vstupnej jednotke (ibuf), skontrolované a odstránené ako zbytočné. Zhodu kontrolného súčtu obsahu rámca prevádza už firmvér hneď po prijatí rámca, pokiaľ zhoda nenastane, prijatý rámec je zahodený. Nástroj *crctest* umožňuje týmto prístupom skontrolovať, že sa obsah rámca od jeho prijatia na vstupnom rozhraní akceleračnej karty až po jeho prijatie softvérovou aplikáciou nezmenil. Aplikácia pracuje s rozhraním knižnice PCAP.

Ďalšia možnosť pre testovanie, ktorú tento nástroj ponúka, pracuje s tzv. vzorkovým súborom. Tento súbor obsahuje rámce, ktoré sú zasielané do hardvérovej akceleračnej karty a prijímané aplikáciou. Každý z rámcov má rôznu veľkosť. *crctest* pred začiatkom prijímania dát spracuje vzorkový súbor, predpočíta hodnotu CRC32 pre každý z rámcov a vytvorí tabuľku s týmito hodnotami. Pri príjme už nie je počítané CRC32 z obsahu rámca, ale dĺžka rámca je použitá ako index do tabuľky predpočítaných hodnôt a táto hodnota je porovnaná s hodnotou poľa kontrolného súčtu rámca. Tento prístup umožňuje zvýšiť priepustnosť pri tomto teste.

8.1.5 sze2write

sze2write je nástrojom pre testovanie TX smeru. Nástroj slúži pre odosielanie dát cez szedata2 rozhranie. Pracuje s rozhraním knižnice libsze2. Dáta pre odoslanie sú špecifikované vo formáte PCAP knižnice v tzv. „PCAP dump" súbore. Aplikácia podporuje dva základné módy odosielania korešpondujúce so schopnosťami knižnice libsze2. Prvým z nich je „single" mód, keď je posielaný každý rámec samostatne. „Burst" mód posiela viacero rámcov naraz, čo zabezpečuje vyššiu priepustnosť. Veľkosť „burst" bloku dát môže byť špecifikovaná.

Keďže PCAP formát rámcov je rozdielny od formátu szedata2 paketov, konverzia medzi týmito formátmi spotrebúva výpočtový výkon, čo je pri testovaní priepustnosti nežiadúce. Preto *sze2write* umožňuje „offline" konverziu medzi týmito formátmi, keď z „PCAP dump" súboru vytvorí „dump" súbor so szedata2 paketmi. Ďalej je možné parametrami špecifikovať aby bol používaný súbor v szedata2 formáte a odosielanie prebieha už bez konverzie. Keďže formát szedata2 paketov obsahuje položku metadát, ktorej obsah môže byť rozdielny pre rôzne projekty nad platformou NetCOPE, nástroj umožňuje túto položku špecifikovať.

Podobne ako nástroj *crctest*, ktorý však pracuje v RX smere, aj *sze2write* umožňuje prácu s kontrolným súčtom CRC32. Prepínač príkazového riadku zabezpečí počítanie CRC32 z obsahu rámca a pridá túto hodnotu na jeho koniec. Táto možnosť testovania vyžaduje špeciálne nastavenie firmvéru, jednotky výstupného rozhrania (obuf). Štandardne nie je hodnota CRC32 počítaná na aplikačnej úrovni, ale práve touto firmvérovou jednotkou, preto je nutné v nej zamedziť tomuto výpočtu, keďže je už táto hodnota dodaná

testovacím nástrojom. Zhoda pri overovaní kontrolného súčtu na zariadení prijímajúcom rámce odoslané z hardvérovej akceleračnej karty značí, že počas cesty rámca z aplikácie cez firmvér a sieťové médium ostal jeho obsah neporušený.

8.1.6 test_tx_burst

Keďže nástroj *sze2write* disponuje väčším množstvom volieb, stal sa menej flexibilným pre použitie pri odhaľovaní určitých druhov chýb. *test_tx_burst* je veľmi jednoduchý nástroj posielajúci testovacie dáta „burst” prenosmi pre ladenie tohoto druhu TX prenosov.

8.1.7 sze2loopback

sze2loopback ako jeho názov hovorí slúži ako „loopback” rozhranie, ktoré všetky prijaté pakety bez spracovania odošle späť. Nástroj ako prvý pracoval s rozhraním knižnice libsze2 v dobe, keď ešte neexistovali nástroje špecializované pre ladenie jednotlivých smerov RX a TX samostatne. Umožňuje overiť príjem i odosielanie dát jedinou aplikáciou. Nástroj môže tiež pracovať cez PCAP rozhranie.

8.1.8 sze2read-mt

Vznik tohoto nástroja si vynútila implementácia SPMD módu príjmu dát. Nástroj pracuje s knižnicou libpthread, zvolí SPMD mód spracovania dát u ovládača a vytvorí viacero vlákien prijímajúcich dáta v tomto móde. Test správnosti SPMD módu prebieha pri použití testovacieho ovládača *szetest.ko*, ktorý každému szedata2 paketu do poľa metadát priradí sekvenčné číslo paketu. Pre evidenciu prijatých paketov slúži bitmapa, v ktorej jednotlivé bity zodpovedajú spracovaniu paketu so sekvenčným číslom poradia bitu. Spracovanie paketu zabezpečí vzájomne vylúčené zmenenie hodnoty príslušného bitu. Po skončení príjmu paketov je bitmapa skúmaná pre obsah „dier”, ktoré znamenajú nespracovanie niektorého z paketov. Naopak bitmapa spojito vyplnená zhodnými hodnotami značí úspešené prijatie všetkých paketov.

8.1.9 sze2write-mt

Tento nástroj využíva paralelného odosielania dát a vznikol spolu s implementáciou tohoto módu v ovládači. Využíva knižnicu libpthread, vytvorí niekoľko vlákien súčasne odosielajúcich testovacie rámce cez rozhranie libsze2. Pri súčasnom stave firmvéru je možné s týmto nástrojom testovať iba za pomoci testovacieho ovládača *szetest.ko*.

8.1.10 sze2oversize_test

Pre overenie správnej implementácie „buffer oversize” vlastnosti ovládača vznikol nástroj *sze2oversize_test*. Nástroj namapuje volaním *mmap()* potrebné štruktúry kruhových bufferov. Zapísanie ľubovoľnej vzorky do TX bufferu na jeho začiatok je overené s vyčítaním tejto hodnoty v „oversize” oblasti na konci bufferu. Kruhové buffery sú vyplnené náhodnou vzorkou dát a následne je kontrolovaná zhoda dát počiatkov a „oversize” oblastí.

8.2 Ladenie výkonnosti

V tejto sekcii budú popísané niektoré voľby hodnôt, ktoré môžu mať výrazný vplyv na výkonnosť pri spracovaní špecifických dát aj na celkovú výkonnosť platformy. Tieto hodnoty sú konfigurovateľné, ale zväčša aplikácie používajú implicitné hodnoty a nemusia tieto

Rýchlosť rozhrania [Gb/s]	1	10	40
Tok na vstupné rozhranie [MB/s]	128	1280	5120
Max. počet ethernet rámcov [Mpkt/s]	2	20	80
Tok do kruhového bufferu [MB/s]	144	1440	5760
Veľkosť kruhového bufferu [MB]	14.4	144	576

Tabuľka 8.1: Teoretické veľkosti kruhových bufferov

hodnoty špecifikovať. Niektoré použité hodnoty sú zvolené na základe teoretického výpočtu, avšak tieto teoretické hodnoty v praxi treba overiť testovaním. Tiež bude vysvetlené aké ďalšie skutočnosti majú vplyv na celkovú výkonnosť platformy.

Veľkosť kruhových bufferov Pre určenie veľkosti kruhových bufferov ovládača bola použitá nasledujúca teoretická úvaha. Kruhový buffer musí byť dostatočne veľký na to, aby zachytil všetky príchodzie dáta pri plnej priepustnosti za časový okamih, cez ktorý je aplikácia spracúvajúca dáta odstavená od procesoru. Plánovač operačného systému poskytne v tejto chvíli procesorový čas iným procesom systému. Tento časový okamih je nazývaný aj ako „system scheduling gap" (ďalej označované ako SSG) a pre teoretické výpočty veľkosti kruhového bufferu bola zvolená hodnota 0.05 s. Pretože kruhový buffer nemusí byť v okamihu prepnutia kontextu prázdny, ako výsledná hodnota je uvádzaný dvojnásobok hodnoty vypočítanej predchádzajúcou úvahou. Pre rôzne rýchlosti sieťových rozhraní a teda maximálne priepustnosti udáva výsledné hodnoty tabuľka 8.1. Za najmenšiu možnú veľkosť rámca je považovaná hodnota 64 B ako udáva štandard IEEE 802.3. Tento rámec však v kvôli formátu szedata2 paketov vytvorí szedata2 paket o veľkosti minimálne 72 B v závislosti od veľkosti metadát.

Tieto výsledky však značne závisia na hodnote SSG, ktorá sa v skutočnosti môže výrazne líšiť. Z tohoto dôvodu bola vykonaná testerskou skupinou projektu NetCOPE sada testov, ktorá mala za úlohu zmerať priepustnosť do softvéru pri použití rôznych veľkostí kruhových bufferov. Test spočíval v zasielaní dátového toku najmenších ethernetových rámcov počas 60 s intervalu zariadením Spirent AX/4000, dáta boli prijímané nástrojom *sze2read*. Výsledky testov udávajú dve významné hodnoty priepustnosti. Prvou z nich je takzvaná hrubá priepustnosť, ktorá značí priemerný počet bitov dát prijatých za sekundu. Druhá hodnota značí priepustnosť ako ju definuje štandard RFC2544. Priepustnosť je definovaná ako tok prijatý aplikáciou bez straty jediného rámca. Určenie tejto hodnoty prebieha metódou podobnou binárnemu vyhľadávaniu, pokiaľ pri určitej priepustnosti zariadenie nie je schopné prijať všetky odoslané rámce bez straty, je priepustnosť znížená na polovicu z prehľadávaného intervalu. Pokiaľ naopak zariadenie príjme všetky rámce bez straty, je priepustnosť testu zvýšená o polovicu z prehľadávaného intervalu. Je nutné poznamenať, že aj keď bolo meranie prevedené na hardvérovej akceleračnej karte s 10 Gb/s rozhraním, výsledné hodnoty priepustnosti sú nižšie. Úzkym hrdlom platformy je momentálne komponent PCI bridge, ktorý umožňuje prenos o teoretickej rýchlosti maximálne 8 Gb/s, pričom samotný protokol PCI si vyžaduje réžiu, ktorá z výslednej priepustnosti časť ukrojí a maximálna priepustnosť szedata2 paketov do softvéru sa pohybuje okolo hodnoty 7 Gb/s.

Z nameraných hodnôt možno usúdiť niekoľko záverov. Keďže priepustnosť podľa RFC2544 je približne rovnaká pre kruhové buffery o veľkostiach 128 a 64 MB a nedosahuje hodnoty hrubých priepustností pri týchto veľkostiach, zrejme tieto nižšie hodnoty budú spôsobené metodikou binárneho vyhľadávania podľa RFC2544. Pokiaľ za maximálnu priepustnosť dát do softvéru vyberieme hodnotu 6.768 Gb/s z hodnôt tabuľky môžeme určiť, že kruhové buffery o veľkostiach 128 a 64 MB sú dostatočne veľké pre pokrytie „system scheduling gap".

Veľkosť bufferu [MB]	Hrubá priepustnosť [Gb/s]	Priepustnosť podľa RFC2544 [Gb/s]	Tok do softvéru [MB/s]	Max. SSG [ms]
128	6.933	6.748	1079.68	
64	6.933	6.768	1082.88	
32	6.925	6.621	1059.36	37.0
16	6.932	5.977	956.32	20.9
8	6.930	3.291	526.56	19.0
4	6.880	1.123	179.68	27.8
2	6.867	0.498	79.68	31.4
1	4.242	0.273	43.68	28.6
0.5	2.644	0.098	15.68	39.9

Tabuľka 8.2: Priepustnosti v závislosti na veľkosti kruhového bufferu

smer	poll threshold [B]	poll timeout [ms]
RX	1/16 veľkosti kruhového bufferu	50
TX	1/16 veľkosti kruhového bufferu	25

Tabuľka 8.3: Implicitné hodnoty *poll threshold* a *poll timeout*

Buffer o veľkosti 32 MB už nie je dostatočný a spôsobuje stratu niekoľkých paketov, čo sa odráža v nižšej výslednej priepustnosti podľa RFC2544. Na základe veľkosti kruhového bufferu a toku do softvéru pri priepustnosti podľa RFC2544 je možné určiť maximálnu hodnotu SSG pokiaľ uvažujeme, za aký časový okamih sa pri danom toku zaplní kruhový buffer. Z výsledkov vyplýva, že maximálna hodnota SSG sa v rámci 60 s intervalu pohybuje v rozmedzí 19-40 ms, čo je menšia hodnota ako 50 ms zvolená pri teoretickom výpočte, preto veľkosti bufferov získané teoretickým výpočtom by mali byť postačujúce. Hodnoty *poll threshold* musia byť vhodne volené, aby aktuálne využívaná časť bufferu „neukrajovala" z časti bufferu určenej pre vykrytie SSG.

poll threshold a poll timeout Tieto voľby majú vplyv na chovanie systémového volania *poll()*, ktoré je obsluhované ovládačom. Obe voľby sú špecifikované zvlášť pre RX a TX smer. *poll timeout* špecifikuje maximálne časové kvantum v milisekundách, počas ktorého zotrváva aplikácia vo volaní *poll()*. Pokiaľ v priebehu tohoto okamihu je k dispozícii pre čítanie či odosielanie aspoň počet bajtov dát špecifikovaný hodnotami *poll threshold*, nastane návrat z volania *poll()* pred uplynutím časového kvanta.

Vyššie hodnoty oboch nastavení spôsobia možnosť spracovania väčšieho množstva dát v priemere na počet systémových volaní, komunikácia s ovládačom sa teda znižuje. Zároveň však stúpa veľkosť aktuálne spracúvaných dát v kruhových bufferoch, sú prideľované väčšie bloky dát, čo môže v konečnom dôsledku spôsobiť stratu dát v dôsledku zaplnenosti bufferov a teda potrebu väčších kruhových bufferov. Empiricky boli zvolené hodnoty pre tieto nastavenia udané tabuľkou 8.3.

Veľkosť bloku dát pri „burst" TX Táto hodnota špecifikuje aký maximálne veľký blok dát bude posielaný v rámci jednej operácie „burst" prenosu. Táto hodnota by mala priamo súvisieť s hodnotou *poll threshold* pre TX smer a nemala by byť väčšia ako táto hodnota. Väčšia hodnota tohoto parametru umožňuje redukovať komunikačnú réžiu s ovládačom, avšak môže spôsobiť vyššiu latenciu pri odosielaní dát. Ako implicitná hodnota tohoto

parametra je zvolená hodnota 4 KB.

Zdrojový kód a kompilátor Zdrojový kód knižnice libsze2 obsahuje množstvo ladiacich informácií, ktoré slúžia pri ladení rôznych druhov chýb, zväčša v spolupráci s firmvérom. Preklad tohoto kódu je podmienený na základe definície príslušných makier. Štandardne všetok softvér na projekte Liberouter je prekladaný s prepínačom prekladača „-g", ktorý začlení do kódu informácie potrebné pre ladenie, k tomuto účelu je používaný nástroj *gdb*. Všetky ladiace informácie v zdrojovom kóde sú potrebné pri hľadaní chýb, avšak spotrebúvajú nemalé množstvo výpočtového výkonu, čo sa odráža na výslednej priepustnosti. Preto je nutné pre účely testovania používať knižnice a softvérové nástroje „čisté", preložené bez akýchkoľvek ladiacich informácií.

Na výkonnosť výsledného kódu môže mať veľký vplyv použitý kompilátor a jeho nastavenia. Na projekte Liberouter je používaný prekladač *gcc* a jeho štandardne používaný prepínač „-O2" zabezpečí zapnutie väčšiny optimalizácií počas kompilácie. Špecifikácia určitých funkcií ako „inline" je štandardnou metódou ako zvýšiť výkonnosť softvérových aplikácií, keďže odpadá réžia spojená s volaním funkcií. Táto metóda je využívaná aj v knižnici libsze2.

8.3 Dosiahnuté výsledky

V tejto sekcii budú podané výsledky meraní riešení a vylepšení vytvorených v rámci tejto práce.

8.3.1 Buffer oversize

Toto vylepšenie ovládača szedata2 umožnilo vynechať významnú časť kódu knižnice libsze2, ktorá ošetrovala situáciu prechodu z konca na začiatok kruhového bufferu. Pre zmeranie predpokladaného zlepšenia vo výkonnosti, teda v spotrebe menšieho procesorového výkonu, bolo využité profilovanie. Profilované knižnice i nástroje boli preložené s prepínačmi prekladača i linkeru „-pg", ktoré zaručia generovanie profilovacích informácií. Nástroje boli staticky linkované s profilovanými knižnicami. Beh programu preloženého s prepínačmi „-pg" vyprodukuje súbor „gmon.out", ktorý obsahuje profilovacie informácie o behu programu. Nástroj *gprof* dokáže analyzovať tieto informácie a poskytne štatistické informácie o behu programu – absolútne časy, ktoré strávil program v jednotlivých funkciách, počty ich volaní.

Pre profilovanie RX smeru bol použitý nástroj *sze2read*. Dáta boli zasielané v intervale 10 minút pri plnej priepustnosti 10 Gb/s zariadením Spirent AX/4000 na jeden port akceleračnej karty COMBOv2 na stroji mossel.liberouter.org. Počet odoslaných rámcov nebol pri jednotlivých pokusoch presne rovnaký. Vo výsledkoch profilácie bola sledovaná veľkosť časového intervalu stráveného programom vo volaní funkcie *szedata_read_next()* z knižnice libsze2, ktorá zabezpečuje príjem dát a počet volaní tejto funkcie, ktorý zodpovedá počtu prijatých paketov. Tento test bol vykonaný pre pôvodnú verziu ovládača a jej príslušiacu knižnicu a pre ovládač s vylepšením „buffer oversize" s redukovanou libsze2 knižnicou. Test navyše zahŕňal dva prípady príjmu dát. V prvom z nich boli zasielané najmenšie rámce ethernetu o veľkosti 64 B s použitím bufferu o implicitnej veľkosti 64 MB. Pri najkratších rámcoch je počet volaní funkcie najvyšší, preto sa pri tejto možnosti najviac prejaví vylepšenie v zjednodušení kódu tejto funkcie. V druhom testovacom prípade boli zasielané rámce najvyššie s veľkosťou 1518 B do bufferu o veľkosti 4 MB. Pri najväčších rámcoch možno sledovať vplyv réžie pri rozdelení szedata2 paketu medzi koniec a začiatok kruhového bufferu. Menšia veľkosť kruhového bufferu bola zvolená pre zvýšenie výsledného vplyvu tohoto faktu.

Veľkosť kruhového bufferu [MB]	Veľkosť rámca [B]	Bez „buffer oversize" [ns]	S „buffer oversize" [ns]
64	64	67.8	62.6
4	1518	270.7	272.3

Tabuľka 8.4: Profilácia vylepšenia „buffer oversize" - priemerný čas vybavenia funkcie *szedata_read_next()*

Každý z prípadov testu bol vykonaný 3-krát pre dosiahnutie hodnovernejších štatistických výsledkov. Tabuľka 8.4 ukazuje priemerné hodnoty trvania funkcie *szedata_read_next()* vypočítané na základe získaných profilovacích informácií.

Z výsledkov vyplýva, že pri najkratších rámcoch vylepšenie „buffer oversize" prináša približne 10 % zlepšenie. Výkonnosť pri tomto type dát je kľúčová pre platformu NetCOPE, pretože je predpoklad, že po vylepšeniach firmvéru, práve vrstva knižnice libsze2 sa môže stať úzkym hrdlom platformy a kľúčovou vrstvou pre celkovú priepustnosť platformy. Prekvapivé výsledky prináša test s najväčšími rámcami kde riešenie s „buffer oversize" prináša zhoršenie približne o 1 %. Tento fakt však môže byť spôsobený nepresnosťou profilovacieho procesu.

8.3.2 SPMD RX

Pre zvolenie vhodného testu pre SPMD mód RX je nutné zmieniť, že jediné vlákno je schopné spracovať všetky prijaté pakety pri plnej priepustnosti pri toku 10 Gb/s na jednom rozhraní, pričom je platforma schopná preniesť sieťový tok približne 7 Gb/s podľa tabuľky 8.2. Práca viacerých vlákien pre prenos dát nemôže túto hodnotu zvýšiť, keďže úzke hrdlo platformy je mimo softvérovú vrstvu.

Ako test SPMD módu bola zvolená nasledujúca úloha. Zariadením Spirent AX/4000 je zasielaný do hardvérovej akceleračnej karty tok 1 Gb/s najmenších 64 B veľkých rámcov po dobu 60 s. Testovanie priepustnosti SPMD módu prebieha nástrojom *sze2read-mt*. Spracovanie každého prijatého rámca musí byť výkonovo náročné, aby jediné vlákno bolo schopné spracovať len menšiu časť zo zasielaných rámcov. Za túto výpočtovo náročnú úlohu bolo zvolené počítanie kontrolného súčtu CRC32 nad dátami rámca cyklicky opakované 10-krát. Počet spracovaných rámcov je meraný. Test je cyklicky opakovaný pre vzrastajúci počet vlákien od jediného vlákna po 4 vlákna. Sú sledované rozdiely v počte spracovaných rámcov podľa počtu pracujúcich vlákien a zároveň rozdelenie zátaže medzi jednotlivé vlákna – pomer spracovaných paketov rôznymi vláknami.

Tento test bol vykonaný dvakrát na rôznych strojoch. Prvý test prebiehal na stroji morgon.liberouter.org s akceleračnou kartou COMBO6x. Stroj je vybavený dvomi štvorjadrovými procesormi Intel Xeon E5335 o frekvencii 2 GHz. Výsledky tohoto testu udáva tabuľka 8.5. Druhý test prebiehal na stroji mossel.liberouter.org s akceleračnou kartou COMBOv2 osadenom procesorom Intel Xeon E5420 o frekvencii 2.5 GHz. Výsledky tohoto testu udáva tabuľka 8.6. Hodnota *poll threshold* bola nastavená na hodnotu 4 MB a *poll timeout* 50 ms. Pre lepšiu vierohodnosť výsledkov bol každý z testov vykonaný 3-krát a tabuľky udávajú priemerné hodnoty. Štvrtý stĺpec tabuľky udáva percentuálny nárast spracovaných rámcov v porovnaní s počtom rámcov spracovaných jediným vláknom. Piaty stĺpec tabuľky udáva priemernú odchýlku počtov rámcov spracovaných rôznymi vláknami a stĺpec šiesty udáva túto hodnotu percentuálne v pomere k počtu spracovaných rámcov.

Výsledky meraní na stroji morgon.liberouter.org ukazujú, že dve vlákna pracujúce v móde SPMD spracovali približne o 59 % viac rámcov ako jediné vlákno. Pridanie ďalšieho vlák-

Počet vlákien	Poslané rámce	Spracované rámce	Nárast [%]	Priemerná odchýlka	Priemerná odchýlka [%]
1	90977727	22755562			
2	90992243	36125877	58.76	218300	0.60
3	90998433	45993628	102.15	210952	0.46
4	90992116	72890422	220.32	154730	0.21

Tabuľka 8.5: Test SPMD RX módu na stroji morgon.liberouter.org

Počet vlákien	Poslané rámce	Spracované rámce	Nárast [%]	Priemerná odchýlka	Priemerná odchýlka [%]
1	90987729	27943251			
2	90977725	46471909	66.31	274940	0.59
3	90988136	71673765	156.5	138813	0.19
4	90986928	90849715	225.12	40640	0.04

Tabuľka 8.6: Test SPMD RX módu na stroji mossel.liberouter.org

na zdvihlo výkon približne na dvojnásobok výkonu jediného vlákna. Veľmi zaujímavé je, že rozdiel výkonu pri troch a štyroch pracujúcich vláknach je rapídny, až približne 120 % výkonu jediného vlákna. Tento fakt navádza k teórii, že i keď samotný stroj disponuje viacerými jadrami procesoru, operačný systém nemusí priradiť vykonávanie každého výkonovo veľmi náročného vlákna separátnemu procesorovému jadru. Celkový nárast výkonu pri štyroch pracujúcich vlákna je približne 225 % z čisto teoretických 300 %. Časť z tohoto rozdielu bola zrejme vynaložená na réžiu ovládača a operačného systému pri viacvláknovom spracovaní. Tieto výsledky zrejme môžu úzko súvisieť aj so špecifickejšou architektúrou dvoch štvorjadrových procesorov a hlavnej pamäte na stroji morgon.liberouter.org. Pozitívne je, že priemerná odchýlka počtov rámcov spracovaných rôznymi vláknami je pomerne nízka, zhruba 0.5 % celkovo spracovaných rámcov a vlákna sú pomerne rovnomerne zaťažené.

Výsledky druhej sady meraní, tentokrát na stroji mossel.liberouter.org, ukazujú mierne pozitívnejšie výsledky. Zdvihnutie počtu pracujúcich vlákien na 2 zvýšilo výkon o približne 66 %. Pridanie tretieho vlákna spôsobilo ďalší kumulatívny nárast o približne 90 % na hodnotu o 157 % vyššiu ako výkon jediného vlákna. Pridanie posledného vlákna spôsobilo nárast výkonu na 325.12 % jediného vlákna a pokrytie približne 99.85 % odoslaných rámcov. Toto výsledné zvýšenie výkonu môže byť skreslené, pretože pokrýva takmer 100 % posielaných dát. 0.15 % nespracovaných rámcov mohlo byť spôsobené zaplnením kruhového bufferu pri „system scheduling gap”. Pre overenie tohoto faktu bola práca štyroch vlákien testovaná pri priepustnosti 1.2 Gb/s kde vlákna zvládli spracovať približne 95.5 M rámcov, čo činí zhruba 342 % výkonu jediného vlákna. Priemerná odchýlka počtov rámcov spracovaných rôznymi vláknami je pri dvoch vláknach približne rovnaká ako v prvej sade testov a pri vyššom počte vlákien dokonca ešte nižšia.

Z prvotných testov SPMD módu RX možno konštatovať zvýšenie výkonnosti aplikácií používajúcich tento mód spracovania dát. Pri práci štyroch vlákien bol nameraný nárast priepustnosti testovacej aplikácie približne o 242 % čo sa môže javiť ako podstatný nárast výkonu aplikácie. Ďalšie prehĺbenie znalostí o prideľovaní vlákien k výpočtovým jadrám SMP operačného systému Linux bude nutné tak isto ako ďalšie sady pokročilých testov.

8.3.3 Paralelný TX

Žiaľ paralelné odosielanie dát nemohlo byť otestované pretože podpora pre spracovanie medzery (vysvetlené 6.4.8) ešte nie je implementovaná v hardvéri platformy v jej súčasnej verzii. Testy paralelného odosielania dát budú vykonané v rámci pokračovania tejto práce po doplnení implementácie tejto vlastnosti.

Kapitola 9

Záver

V tejto práci bola popísaná architektúra platformy NetCOPE so zameraním na komunikáciu medzi aplikáciami a hardvérovou kartou. Naštudovaná bola teória vytvárania paralelne pracujúcich aplikácií, ktorá bola použitá pre analýzu možností paralelného spracovania typických sieťových aplikácií. Pre podporu paralelného spracovania sieťových aplikácií platformou NetCOPE boli vypracované základné modely podpory pre príjem a odosielanie dát.

Vhodnými modelmi pre podporu spracovania prijatých dát sa javia modely MISD a SPMD. Prvý z modelov umožňuje funkčnú dekompozíciu užívateľských aplikácií. Model SPMD prináša dekompozíciu dátovú. V oboch modeloch má na výkonnosť vplyv voľba veľkosti dátového bloku prideľovaného procesom. Model zreťazeného spracovania sa nejaví ako najvhodnejší. Podpora paralelného odosielania paketov môže byť realizovaná modelmi so známou alebo neznámou veľkosťou dát.

Naštudovaná a podaná bola problematika tvorby systémových ovládačov operačného systému Linux. Zvládnutie tejto teórie bolo nevyhnutné pre pochopenie architektúry vrstvy ovládačov platformy NetCOPE. Hardvérovo nezávislý ovládač szedata2 vytvára rozhranie k aplikáciám pre využívanie rýchlych DMA prenosov dát medzi užívateľskými aplikáciami a hardvérom platformy NetCOPE. V implementácii tohoto ovládača bolo vykonaných niekoľko zmien a rozšírení, ktoré pridávajú podporu SPMD modelu spracovania príjmu dát k už existujúcemu MISD modelu. SPMD model umožnil akceleráciu testovacej aplikácie a zvýšenie jej priepustnosti pri využití štyroch jadier procesoru o približne 242 %. Tieto zmeny taktiež rozširujú ovládač szedata2 o podporu paralelného odosielania dát.

Keďže rozhranie systémových volaní operačného systému Linux pre komunikáciu užívateľských aplikácií s ovládačom je príliš komplexné, ako súčasť tejto práce bola implementovaná časť knižnice libsze2 poskytujúca aplikáciám rozhranie vyššej úrovne pre príjem a odosielanie dát. Toto jednoduché rozhranie umožňuje rýchly vývoj aplikácií využívajúcich rýchle DMA prenosy bez hlbších znalostí teórie komunikácie s ovládačmi operačného systému. Pre podporu súbežného viacvláknového spracovania dát táto knižnica prešla úpravou, ktorá zaručuje jej reentrantnosť a vláknovú bezpečnosť. Všeobecne široko používaným štandardným rozhraním pre príjem a odosielanie sieťových dát je knižnica PCAP. Toto rozhranie používa veľké množstvo rozšírených a užitočných nástrojov pre monitorovanie sieťovej premávky. V rámci tejto práce bola knižnica PCAP rozšírená o možnosť práce nad rozhraním szedata2. Tým umožňuje akceleráciu týchto aplikácií hardvérovými akceleračnými kartami platformy NetCOPE.

Pri priepustnostiach chrbtových sietí 10, 40 a 100 Gb/s si spracovanie príslušného množstva sieťových dát vyžaduje vysoké nároky na spotrebovaný výpočtový výkon. Takéto rýchlosti umožňujú venovať spracovaniu jediného paketu len desiatky až stovky taktov procesoru. Časť z tohoto výpočtového výkonu je použitá na samotný prenos dát medzi

hardvérom a užívateľským adresovým priestorom. Túto úlohu zabezpečuje ovládač szedata2 a knižnica libsze2. Cieľom pri implementácii týchto dvoch artefaktov je ich vysoká rýchlosť a malé nároky na potrebný výpočtový výkon. Úprava ovládača szedata2 pomenovaná ako „buffer oversize” využíva princíp virtuálneho pamäťového priestoru aplikácie pre jednoduchšie spracovanie paketov rozdelených medzi koniec a začiatok kruhového bufferu. Toto vylepšenie umožnilo odstrániť veľkú časť kódu knižnice libsze2, ktorá riešila spracovanie takýchto rozdelených paketov. Zároveň touto redukciou kódu došlo k urýchleniu práce knižničných funkcií, pričom potrebný výpočtový výkon v kritickom prípade pri príjme najkratších paketov klesol o 10%. K optimalizácii výkonu knižnice boli použité i špecifické nastavenia kompilátora. Pre dosiahnutie vysokej priepustnosti prenosov dát knižnica libsze2 využíva „burst” prenosy. Na výslednú priepustnosť má vplyv i voľba veľkosti kruhových bufferov a hodnôt iných nastavení. Voľby týchto hodnôt sú v práci vysvetlené.

Testovanie funkčnosti i výkonnosti platformy NetCOPE je významnou súčasťou vývojového procesu. V rámci tejto práce vzniklo množstvo nástrojov umožňujúcich testovanie špecifických vlastností rozhrania szedata2.

Pokračovanie tejto práce je predpokladané v nasledujúcich úlohách. Kvôli kontinuite vývoja platformy NetCOPE sa ako štandardné zložky platformy používajú stabilné verzie ovládačov, knižníc a nástrojov. Stabilné verzie väčšinou prešli niekoľkými procesmi testovania, ktoré odhalili chyby v nich a tieto chyby boli odstránené. Novovzniknuté rozšírenia a zmeny ovládača szedata2 a knižnice libsze2 je pre nahradenie štandardných stabilných verzií nutné spoľahlivo otestovať vývojármi a testerskou skupinou projektu. Momentálnym úzkym miestom platformy je komponent PCI bridge, ktorý znižuje priepustnosť prenosov na akceleračnej karte COMBOv2 na hodnotu približne 7 Gb/s. Teoretický potenciál karty pri dvoch 10 Gb rozhraniach je na úrovni až 20 Gb/s. Na vylepšeniach tohoto komponentu sa intenzívne pracuje. Softvér v súčasnom štádiu stíha vykonávať prenosy na maximálnej priepustnosti. Zvýšenie priepustnosti komponentu PCI bridge a plánovaný príchod akceleračných kariet s 40 Gb rozhraniami pravdepodobne spôsobia presun úzkeho hrdla platformy práve do softvérovej vrstvy. Tieto podmienky budú vyvolávať tlak na optimalizácie ovládača a príslušných knižníc.

Literatúra

[1] CESNET, z.s.p.o., Zaikova 4, 160 00 Praha 6: NetCOPE Architektonický dokument, 2008.

[2] Slabý, Jiří: Rapid Data Transfers on COMBO Platform [Master's thesis]. Brno, Masaryk University Faculty of Informatics, 2008.

[3] Martínek, Tomáš: Komunikace medzi hardware a software. NetCOPE Trac, 2008.

[4] Endace Limited, Ellerslie, Auckland, New Zealand: Solutions [Overview]. Dokument dostupný na URL http://www. endace.com/solutions.html, 2008.

[5] PCAP manual page. Dokument dostupný na URL http://www.tcpdump.org/pcap3_man.html, 2003.

[6] Dvořák, Václav: Architektura a programování paralelních systémů. Skripta FIT, VUTIUM, 2004.

[7] Lampa, Petr: Výukové materiály k predmetu POS. Brno, FIT VUT, 2007.

[8] Hanáček, Petr: Výukové materiály k predmetu PRL. Brno, FIT VUT, 2008.

[9] Corbet, Jonathan; Rubini, Alessandro; Kroah-Hartman, Greg: Linux Device Drivers, Third Edition. O'Reilly, 2005.

Dodatok A

Rozhranie knižnice libsze2

Nízkoúrovňové funkcie

- szedata_open() – namapuje kruhový buffer do adresového priestoru aplikácie a prevedie inicializáciu.
- szedata_subscribe() – umožňuje výber ľubovoľných prenosových kanálov pre prácu. Taktiež špecifikuje požadovanú veľkosť bloku dát pre prácu.
- szedata_start() – započne prenosy.
- szedata_close() – zastaví prenosy, uvoľní použitú pamäť.
- szedata_poll() – započne čakanie na hardvér dokým neobdrží dostatočné množstvo dát. Vlákno je blokované po špecifikovanú dobu.
- szedata_rx_lock_data() – požiadanie ovládača o pridelenie dát na čítanie.
- szedata_rx_unlock_data() – oznámenie ovládača skončenie spracovania prijatých dát.
- szedata_tx_lock_data() – požiadanie ovládača o pridelenie priestoru pre zápis a odoslanie dát.
- szedata_tx_unlock_data() – oznámenie ovládača o skončení zápisu dát.

Vyššie funkcie

- szedata_read_next() – požiadavok pre poskytnutie jedného prijatého paketu. Toto volanie skrýva nízkoúrovňové funkcie szedata_rx_un/lock() a szedata_poll() a skrýva pred aplikáciou blokové prenosy prijatých dát.
- szedata_decode_packet() – slúži pre konverziu dát z formátu použitého pri prenose do formátu vhodného pre spracovanie aplikáciou.
- szedata_prepare_packet() – slúži pre konverziu z formátu vhodného pre spracovanie aplikáciou do formátu použitého pri prenose.
- szedata_try_write_next() – pokúsi sa o prenos jedného paketu v „single” móde.
- szedata_burst_write_next() – odošle jediný paket v „burst” móde.
- szedata_burst_write_flush() – explicitné odoslanie momentálne rozpracovaného „burst” bloku odosielaných dát.
- szedata_set_sched() – nastavenie SPMD alebo MISD módu príjmu dát.

Dodatok B

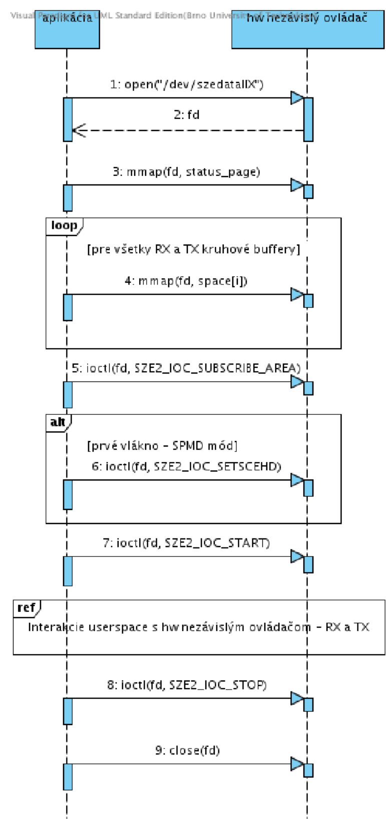
Rozhranie knižnice PCAP

Rozhranie knižnice je jednoduché, väčšina aplikácií potrebuje k práci so sieťovými dátami len nasledujúce základné funkcie:

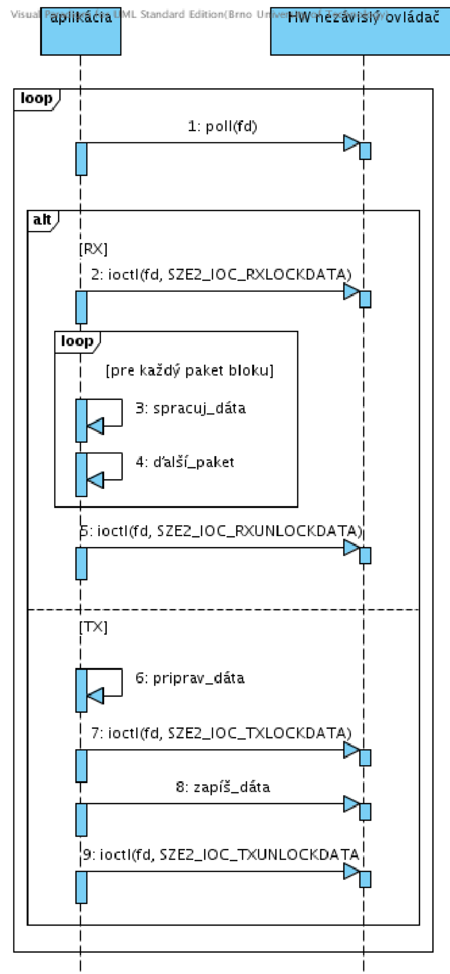
- pcap_open_live() – inicializuje knižnicu a vráti deskriptor.
- pcap_dispatch()/pcap_loop() – zaháji slučku príjmu paketov z inicializovaného rozhrania. Jedným z parametrov tejto funkcie je ukazateľ na obslužnú rutinu, ktorá sa vykoná pri každom príjme paketu s dátami paketu ako argumentmi tejto funkcie.
- pcap_next() – je alternatívou príjmu paketov k predchádzajúcim funkciám. Funkcia získa prvý dostupný paket.
- pcap_inject() – odošle jeden paket.
- pcap_close() – ukončí prácu s knižnicou.

Dodatok C

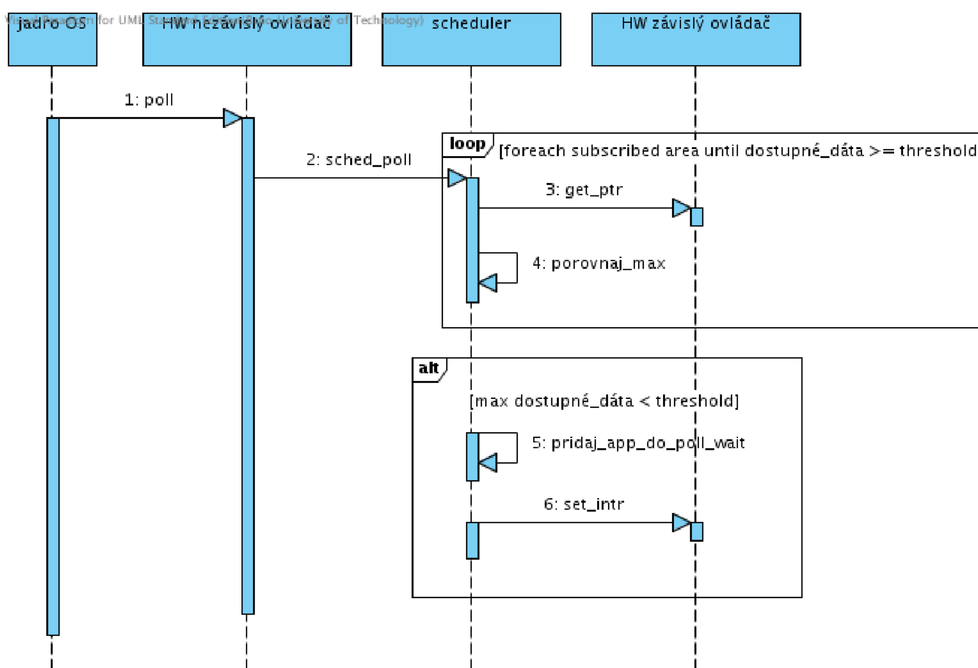
UML diagramy



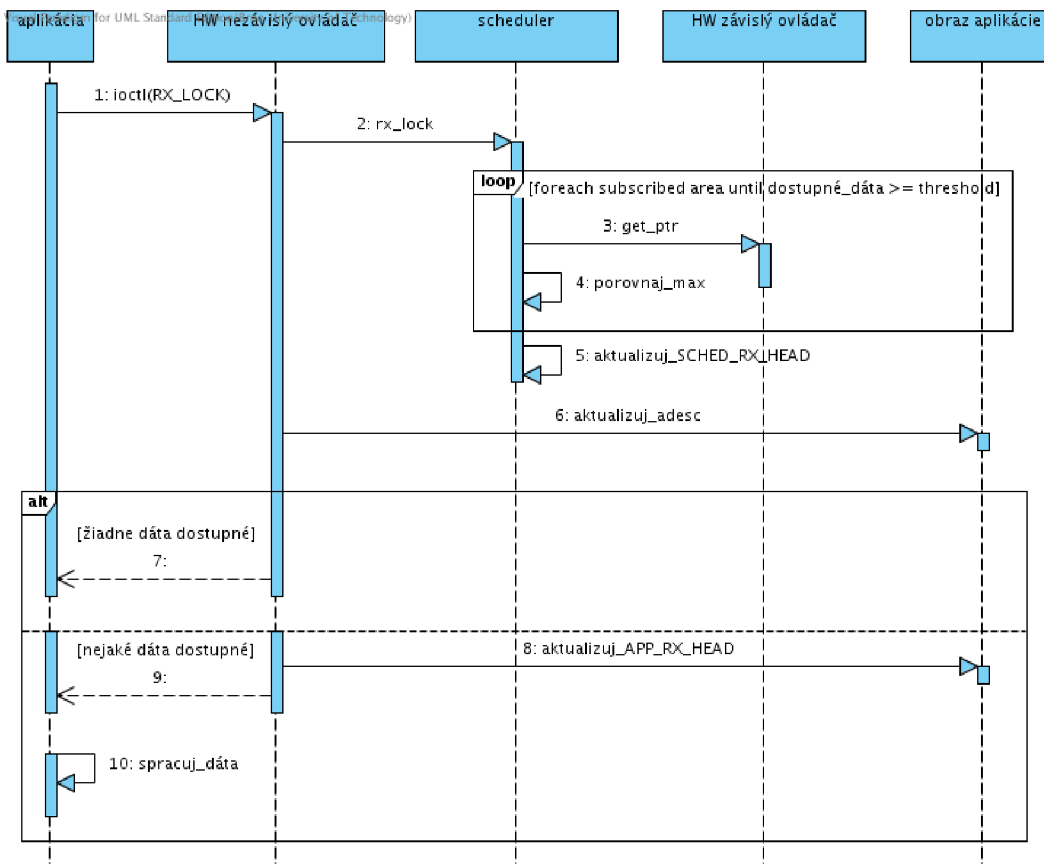
Obrázok C.1: Interakcia aplikácie s hw nezávislým ovládačom – typický scenár



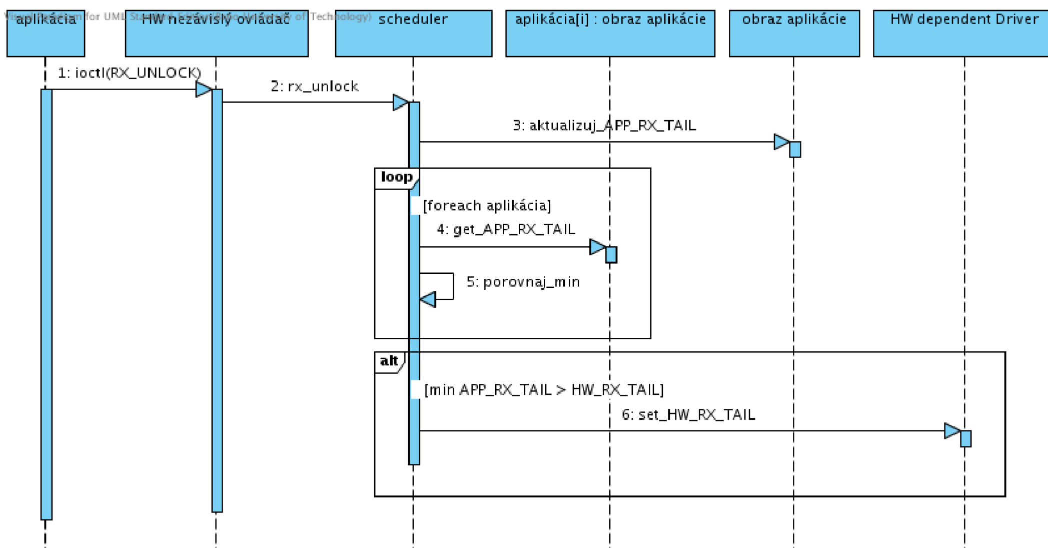
Obrázok C.2: Interakcia aplikácie s hw nezávislým ovládačom - RX a TX



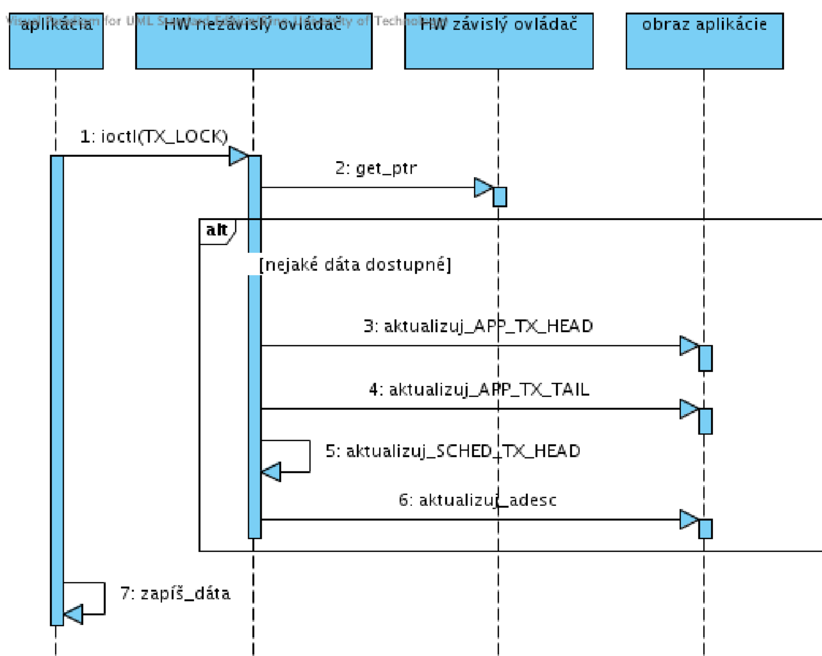
Obrázok C.3: Realizácia volania *poll()* ovládačmi szedata2



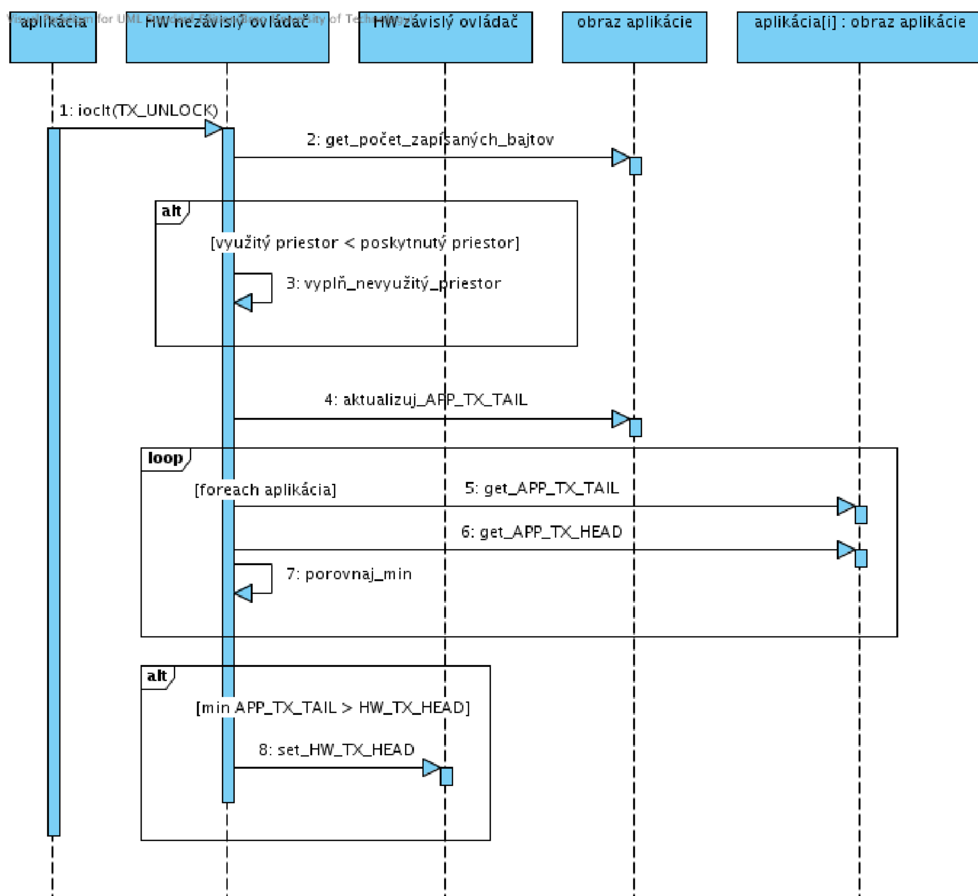
Obrázok C.4: Realizácia požiadavku pre poskytnutie dát RX ovládačmi szedata2



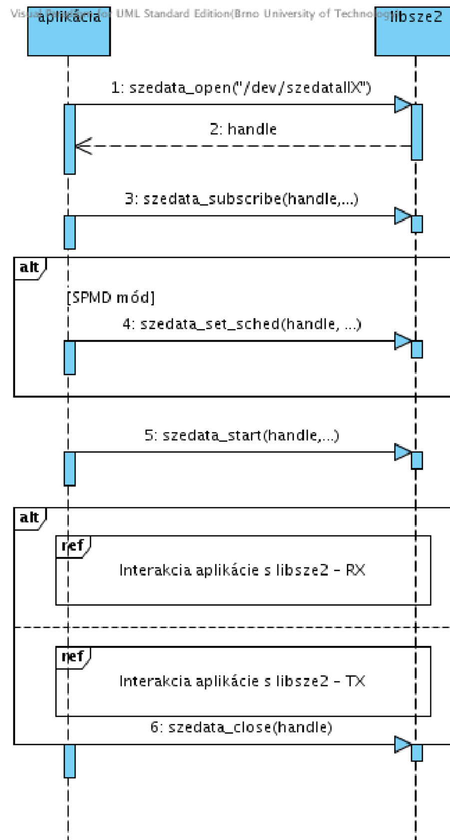
Obrázok C.5: Realizácia dokončenia spracovania dát RX ovládačmi szedata2



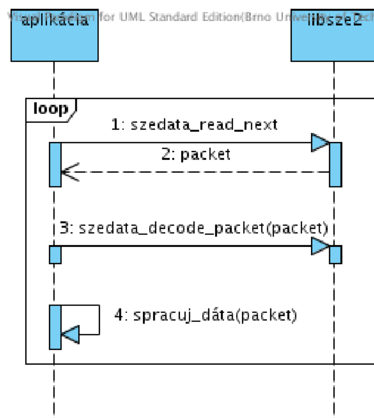
Obrázok C.6: Realizácia požiadavky o priestor pre zápis dát TX ovládačmi szedata2



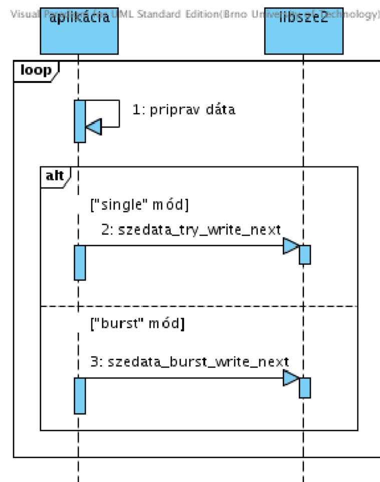
Obrázok C.7: Realizácia ukončenia zápisu dát TX ovládačmi szedata2



Obrázok C.8: Interakcia aplikácie s knižnicou libsze2 - typické použitie



Obrázok C.9: Interakcia aplikácie s libsze2 - RX



Obrázok C.10: Interakcia aplikácie s libsze2 - TX