FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Bakalářská práce

Komunikační protokol CubeSat Space protokol pro referenční distribuci OS Linux

Studijní program:	B0613A140005 Informační technologie
Studijní obor:	Aplikovaná informatika
Autor práce:	**Peter Spurný**
Vedoucí práce:	Ing. Lenka Kosková Třísková, Ph.D. Ústav nových technologií a aplikované informatiky

Liberec 2024

FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Zadání bakalářské práce

Komunikační protokol CubeSat Space protokol pro referenční distribuci OS Linux

Jméno a příjmení:	**Peter Spurný**
Osobní číslo:	M21000134
Studijní program:	B0613A140005 Informační technologie
Specializace:	Aplikovaná informatika
Zadávající katedra:	Ústav nových technologií a aplikované informatiky
Akademický rok:	2023/2024

Zásady pro vypracování:

1. Pro projekt Linux4Space.org navrhněte a realizujte integraci komunikačního protokolu CubeSat Space Protocol.
2. Pro řešení využijte existující knihovnu (https://github.com/libcsp/libcsp).
3. Integraci realizujte formou receptů pro vrstvu meta-linux4space.
4. Funkčnost řešení demonstrujte na vzorovém HW.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30 – 40 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	čeština

Seznam odborné literatury:

[1] SIMMONDS, Chris. Mastering embedded Linux programming: harness the power of Linux to create versatile and robust embedded solutions. Community experience distilled. Birmingham: Packt Publishing, 2015. ISBN 978-1-78439-253-6.

[2] Salvador O., Angolini D.: Embedded Linux Development with Yocto Project, Packt Publishing 2014, ISBN: 9781783282340.

Vedoucí práce:	Ing. Lenka Kosková Třísková, Ph.D. Ústav nových technologií a aplikované informatiky
Datum zadání práce:	12. října 2023
Předpokládaný termín odevzdání:	14. května 2024

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Josef Chaloupka, Ph.D.
garant studijního programu

V Liberci dne 19. října 2023

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

6. května 2024 Peter Spurný

Komunikační protokol CubeSat Space protokol pro referenční distribuci OS Linux

Abstrakt

Cílem této práce je integrovat knihovnu libcsp pro protokol CSP do referenční distribuce operačního systému Linux. Integrace je realizována pomocí receptů a konfiguračních souborů BitBake od projektu Yocto. Cílem druhé části práce je demonstrovat funkčnost knihovny na ukázkovém hardwaru. Je proveden přehled úskalí a problémů vesmírných aplikací a družic na základě jejich prostředí. Je uveden popis kosmických protokolů a důvody jejich použití. Je také popsána práce s projektem Yocto a nástrojem pro automatizaci sestavování BitBake. Dále je představen případ použití projektu Linux4Space. Práce řeší systém dvou jednodeskových počítačů Raspberry Pi 3b+. Zabývá se použitým ovladačem a dalším kódem potřebným k vyvolání restartu systému. Následně je vytvořen demonstrační kód. Analýza je zakončena návrhem, kam recept vložit do projektu Linux4Space. V implementační části práce řeší a demonstruje zapojení počítače Raspberry Pi 3b+ prostřednictvím převodníků USB-UART. Práce také ukazuje provedení demonstračního kódu a ukázku funkční komunikace. Ve shrnutí je uveden výsledek práce. V závěru jsou diskutovány možné další kroky k rozšíření práce.

Klíčová slova: Yocto, BitBake, Raspberry Pi, CSP, Linux4Space

Communication protocol CubeSat Space protocol for Linux OS reference distribution

Abstract

The goal of this work is to integrate the libcsp library for the CSP protocol into a reference distribution of the Linux operating system. The integration is implemented using the BitBake recipes and configuration files from the Yocto project. The second part of the work aims to demonstrate the functionality of the library on sample hardware. An overview of the pitfalls and problems of space applications and satellites based on their environment is made. A description of space protocols and the reasons for their use is given. Work with the Yocto project and the BitBake build automation tool is also described. A use case for the Linux4Space project is also presented. The thesis deals with a system of two Raspberry Pi 3b+ single board computers. It deals with the driver used and other code needed to trigger a system reboot. Subsequently, demonstration code is created. The analysis concludes with a suggestion of where to put the recipe in the Linux4Space project. In the implementation part, the thesis solves and demonstrates the wiring of a Raspberry Pi 3b+ computer via USB-to-UART converters. The thesis also shows the execution of the demonstration code and a demonstration of functional communication. In the summary, the result of the work is presented. Finally, possible next steps to extend the work are discussed.

Keywords: Yocto, BitBake, Raspberry Pi, CSP, Linux4Space

Poděkování

Rád bych poděkoval paní doktorce Lence Koskové Třískové za její vstřícnost, zpětnou vazbu a skvělé vedení. Dále bych chtěl poděkovat svým blízkým za trpělivost a vstřícnost. V neposlední řadě bych rád poděkoval týmu Linux4Space.

Obsah

Seznam zkratek 11

Úvod **12**

1 Satelity a jejich jedinečné podmínky **13**
1.1 CubeSat 13
1.2 Jedinečné problémy vesmírné aplikace 15
1.3 Protokoly používané ve vesmíru 17
1.3.1 SPP - space packet protocol agentury CCSDS 18
1.3.2 SpaceWire 18
1.3.3 AX.25 18
1.4 Technologie použité v bakalářské práci 19
1.4.1 Libcsp 19
1.4.2 Yocto 20
1.4.3 Bitbake 21
1.4.4 Pojmenovací konvence projektu Yocto 21
1.5 Linux4Space 22

2 Počáteční analýza způsobu integrace libcsp do Linux4Space **23**
2.1 Použitý ovladač 23
2.2 UART 23
2.3 Restart v libcsp 25
2.3.1 Restart s libcsp-develop 25
2.3.2 Restart s libcsp-1.6 26
2.4 Demonstrační kód 26
2.4.1 Návrh demonstračního kódu v jazyce C 26
2.5 Vrstvy Linux4Space 28
2.5.1 Recepty BitBake a jejich místo ve vrstvách Linux4Space . . . 28
2.5.2 Vrstvy projektu Linux4Space 28

3 Implementace **29**
3.1 Hardwarová implementace 29
3.2 Operační systém 29
3.3 Úpravy a vytvořené soubory 30
3.3.1 Konfigurační soubory 30
3.3.2 Recepty 31

3.4 Instalace, provoz a předvedení funkčnosti 33
3.4.1 Loopback 33
3.4.2 Jednosměrná komunikace 34
3.4.3 Obousměrná komunikace 37
3.4.4 Výsledky práce 38

4 Závěr **39**

Použitá literatura **40**

Seznam obrázků

1.1 Družice VZLUSAT-1 velikosti 2U [17] 14
1.2 Graf vyslaných nanosatelitů podle oběžné dráhy [12] 15
1.3 Porovnání vrstev modelu TCP/IP a CSP 19
1.4 CSP hlavička 20

2.1 Nastavení programu Minicom 24
2.2 Zapojení Raspberry Pi přes GPIO loopback 24
2.3 Propojení dvou Raspberry Pi prostřednictvím GPIO 25
2.4 Diagram funkčnosti klientského kódu 27
2.5 Diagram funkčnosti kódu na straně serveru 27
2.6 Vizualizace vrstev projektu Linux4Space 28

3.1 Schéma vrstev a souborů, z nichž byl operační systém vytvořen . . . 30
3.2 Zapojení jednosměrné komunikace pomocí převodníků USB-UART . 34
3.3 Ukázka komunikace mezi dvěma Raspberry Pi, pohled zepředu 36
3.4 Ukázka komunikace mezi dvěma Raspberry Pi, pohled zezadu 36
3.5 Diagram obousměrné komunikace mezi dvěma Raspberry Pi 37

Seznam zkratek

APID - Identifikátor aplikačního procesu
BER - Vysoká bitová chybovost
BJT - Bipolární tranzistor
CAN - Síť řídicí jednotky
CCSDS - Poradní výbor pro kosmické datové systémy
CMP - CSP Management Protocol
CRC - Cyklická redundantní kontrola
CSP - CubeSat Space Protocol
ECSS - Evropská spolupráce v oblasti vesmírné standardizace
ELISA - Enabling Linux In Safety Applications
ESA - Evropská kosmická agentura
FreeRTOS - Open-source operační systém reálného času
GEO - Geostacionární oběžná dráha Země
Gnd - Uzemění
GPIO - Univerzální vstupní/výstupní pin
ISS - Mezinárodní vesmírná stanice
I2C - Inter-integrovaný obvod, sériová počítačová sběrnice
JAXA - Japonská národní vesmírná agentura
KISS - Keep It Simple, Stupid, amaterský rádiový protokol
LEO - Nízká oběžná dráha Země
MOSFET - Druh tranzistorů řízených elektrickým polem
NASA - Národní úřad pro letectví a vesmír
POSIX - Přenosné rozhraní operačního systému, standard Unixových OS
RDP - Reliable Datagram Protocol
RKA - Státní korporace pro kosmické aktivity „Roskosmos“
RTT - Proměnná doba zpáteční cesty
Rx - Příjem dat
SEE - Single event effect, typ poruchy
SEB - Single event burnout, typ poruchy
SEL - Single event latch-up, typ poruchy
SEGR - Single event gate rupture, typ poruchy
SEU - Single event upset, typ poruchy
SPP - Space Packet Protocol
SpW - Space Wire
TCP/IP - Transmission Control Protocol / Internet Protocol, sada protokolů pro komunikaci v počítačových sítích
TID - Celková ionizující dávka
TM/TC - Telemetrická jednotka (TM) a jednotka dálkového ovládání (TC)
Tx - Přenos dat
UDP - Unreliable Datagram Protocol
USB - Univerzální sériová sběrnice
UART - Univerzální asynchronní přijímač-vysílač
VZLÚ - Výzkumný a zkušební letecký ústav
WDT - Watchdog timer
WSL - Windows Subsystem for Linux

Úvod

Tato práce se zabývá integrací vesmírného protokolu CubeSat pomocí receptu Yocto do vestavěného operačního systému Linux. Práce je napsána pro projekt Linux4Space, jehož cílem je vytvořit referenční distribuci operačního systému určenou speciálně pro vesmírné aplikace. Z tohoto důvodu práce začíná popisem nanodružic CubeSat a podmínek, kterým musí ve vesmíru čelit. Čtenář tak získá zdůvodnění, proč je CSP integrován. Text práce pokračuje úskalími, která se vyskytla při vývoji výsledného kódu, a končí příkladem funkčního receptu a jeho demonstrací v praxi.

Čtenář má možnost seznámit se s výsledky práce v repozitáři Linux4Space Gitlab, kde má také možnost stáhnout a spustit distribuci Linux4Space podle návodu.

Rozhodnutí vydat se na tuto výzkumnou cestu vychází z kombinace osobního zájmu autora a touhy vyvinout něco trvalejšího - řešení, které přesáhne hranice akademického prostředí a najde odezvu v reálných systémech. Právě tato touha formovala text této práce, a to nejen jako akademickou povinnost, ale jako snahu přispět k rozvoji leteckého průmyslu.

1 Satelity a jejich jedinečné podmínky

Před vysvětlením a popisem výsledků práce je důležité, aby se čtenář seznámil s jedinečnými podmínkami a problémy, s nimiž se musí vývojáři a tvůrci satelitů vypořádat.

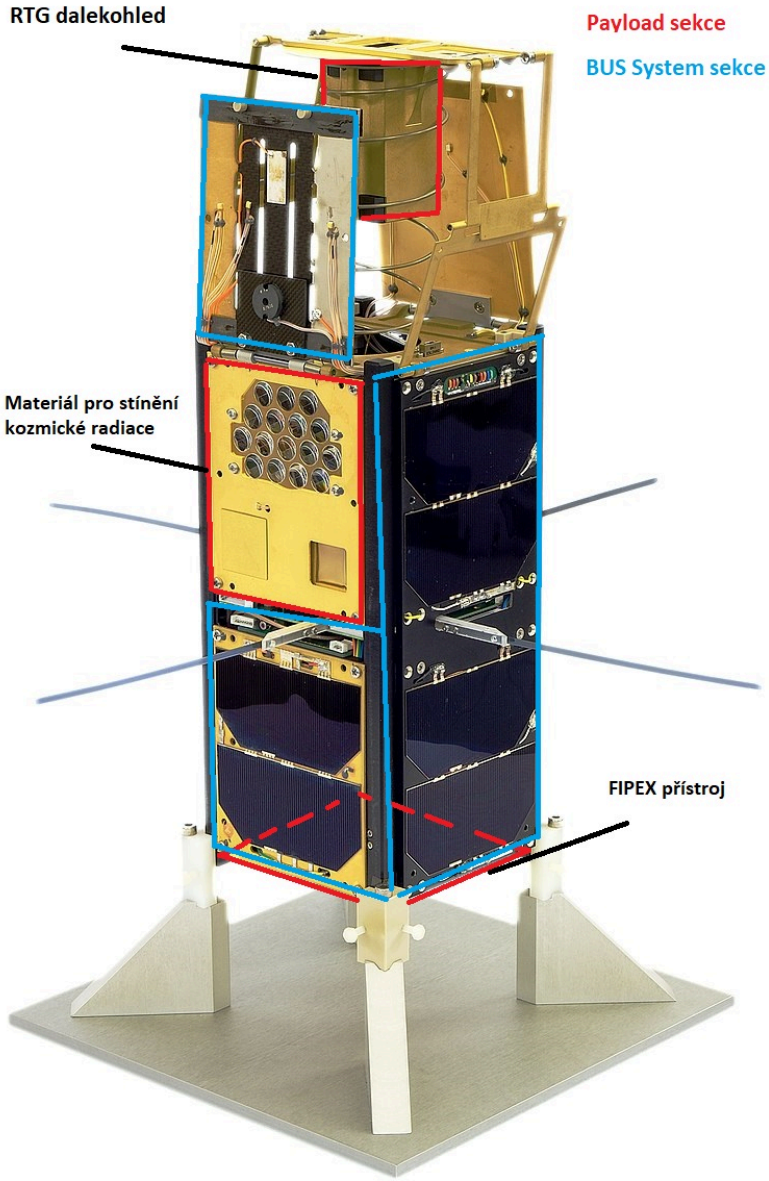
1.1 CubeSat

CubeSat je druh nanodružic, které nepřesahují rozměry 100x100x133 mm a hmotnost 1,33 kg na 1U (standardní jednotka CubeSat). Družice CubeSat se vyrábějí v různých velikostech: 1U, 2U, 3U, 6U, 12U [7]. Specifické standardy pro CubeSat pomáhají snižovat cenu [7]. Jsou menší a levnější než konvenční družice. Cena družice CubeSat se pohybuje v mezi 100 a 200 tisíc USD. Náklady na konvenční satelit se mohou vyšplhat až na 300 milionů USD [5]. Na oběžnou dráhu jsou vysílány z ISS nebo se připojují k satelitu vyslanému z povrchu Země [20]. Hlavním důvodem malých rozměrů družic CubeSat jsou náklady. Malé rozměry znamenají méně použitého materiálu a také méně paliva potřebného k vynesení družice CubeSat na oběžnou dráhu [15].

Navzdory své malé velikosti jsou CubeSat satelity vybaveny řadou subsystémů, včetně energetických systémů, komunikačních systémů a systémů pro řízení polohy. Tyto subsystémy jsou často miniaturizovanými verzemi těch, které se nacházejí ve větších družicích, a využívají pokroky v elektronice a materiálech k maximalizaci funkčnosti v rámci omezení velikosti družice CubeSat.

Družice CubeSat slouží především k testování nových technologií, vědeckým experimentům, komunikaci a pořizování snímků zemského povrchu. CubeSat má dvě části, bus a payload. Bus je část družice, která je zodpovědná za přežití ve vesmíru. Obsahuje systémy, jako je napájení družice, pohon, komunikaci se zemí a řízení nadmorské výšky. Zahrnuje strukturální systém a systémy, které chrání CubeSat před nástrahami vesmíru [21].

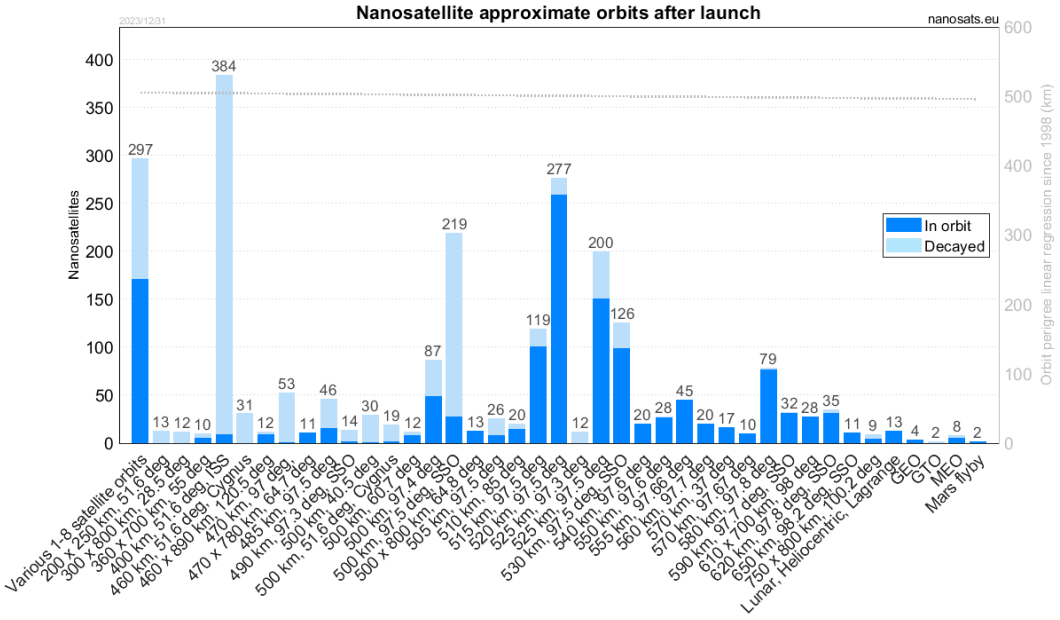
Payload je část družice CubeSat, která je zodpovědná za provedení samotné mise. Ovládá kamery, vědecké přístroje, senzory a další přístroje potřebné k dosažení cíle mise. Družice CubeSat může obsahovat několik payload systémů, nemusí být nutně omezena pouze na jeden [21].



Obrázek 1.1: Družice VZLUSAT-1 velikosti 2U [17]

Družice CubeSat jsou standardně vysílány na LEO což je do 1000 km od povrchu Země. K dnešnímu dni bylo na jiné místo než LEO vysláno pouze 29 nanodružic [12]. Může vyvstat otázka, proč vůbec používat jiný druh družic než CubeSat, když jsou tak nákladově efektivní. Důvodů je celá řada, od výkonu přes pokrytí zemské-

ho povrchu až po omezení dostupného hardwaru. Nejdražší druh satelitů, satelity, které jsou vysílány na GEO a jejichž cena dosahuje částky 300 milionů USD, mají propustnost stovek Gb/s, jsou schopny pokrýt $\frac{1}{3}$ zemského povrchu a mohou být navrženy bez omezení hardwaru a softwaru. Naproti tomu nanodružice na LEO mají propustnost 1 Kb/s - 40 Mb/s, jsou schopny pokrýt malý zlomek zemského povrchu (k pokrytí celé Země by bylo třeba vypustit stovky až tisíce nanodružic) a mají velká omezení v oblasti hardwaru a softwaru [5].



Obrázek 1.2: Graf vyslaných nanosatelitů podle oběžné dráhy [12]

1.2 Jedinečné problémy vesmírné aplikace

Satelity, jejich součásti a software čelí ve vesmíru jedinečným výzvám. Jednou z prvních překážek, které musí překonat, je přežití cesty do vesmíru. Po startu rakety musí družice a její součásti vydržet extrémní vibrace a zrychlení rakety. Během letu na oběžnou dráhu musí družice přežít pyrotechnické šoky. Pyrotechnický šok je dynamický strukturální šok. Tento výbuch je způsoben rychlým uvolněním energie při událostech, jako je oddělení družice nebo raketových stupňů. Při aktivaci pyrotechnických zařízení, která tyto události vyvolávají, vznikají vysokofrekvenční napěťové vlny, které se šíří konstrukcí družice. Tyto napěťové vlny mohou potenciálně poškodit citlivé elektronické součásti, narušit obvody nebo ohrozit celkovou integritu družice [14].

Ve vesmíru je družice vystavena několika typům záření, a to slunečnímu, ultrafialovému a kosmickému. Družice je na oběžné dráze bombardována kosmickým zářením, jehož energetické částice dopadají na povrch a občas pronikají stěnami

družice. Tyto částice mohou způsobit defekty a degradaci součástí a funkcí družice. Defekty způsobené kosmickým zářením se dělí na celkovovou ionizující dávku (TID) a single event effect (SEE) [14]. TID je efekt, při kterém dochází k postupné ionizaci materiálu, která následně způsobuje jeho rozpad a modifikaci. To má za následek například posun prahového napětí, ovlivňuje časové charakteristiky nebo zvyšuje svodové proudy zařízení. SEE je jev, při kterém dochází k zásahu ojedinělou vysokoenergetickou částicí. SEE se dělí na SEU, SEL, SEB a SEGR. SEU způsobuje změnu stavu mikroprocesoru nebo tranzistoru [16]. SEL označuje typ zkratu, při kterém náraz částic vede k abnormálně vysokému proudu a ztrátě funkčnosti zařízení [13]. SEB je stav vysokého proudu ve výkonovém tranzistoru a způsobuje zamrzlé bity, šum a selhání zařízení. SEGR je vytvoření vodivé cesty, která vede k vyhoření tranzistorů BJT, MOSFET a CMOS. SEU a SEL lze opravit restartem, SEB a SEGR je neopravitelná závada [16].

Družice a její hardware a software se musí vyrovnat s vakuem, slabou gravitací a extrémními teplotními rozdíly. Družice na nízké oběžné dráze mohou zažívat extrémní teploty v rozmezí přibližně od -60 °C do +150 °C [1]. Družice CubeSat jsou chráněny proti teplotě a záření několika mechanismy. Desky plošných spojů CubeSat procházejí procesem zpevnění nanesením vrstev pryskyřičného povlaku, který zvyšuje jejich odolnost a chrání je před mechanickým namáháním a tepelnými výkyvy [18]. Kromě toho jsou přijímána opatření k posílení paměťových komponent integrací funkcí detekce a korekce chyb. Podobně jsou posíleny komunikační cesty pomocí mechanismů detekce a opravy chyb zabudovaných do komunikačních protokolů a souvisejícího hardwaru, aby byla zajištěna integrita dat během přenosu. Konkrétně u paměti jsou zavedeny ochrany proti přechodným chybám převrácení bitů v důsledku působení kosmického záření. Náchylnost paměťových čipů k selhání SEU, která je způsobena jejich velkou křemíkovou plochou vystavenou záření, vyžaduje taková ochranná opatření. Desky CubeSat používají k opravě jednobitových chyb Hammingův kód. Oprava rozšířeným Hammingovým kódem probíhá na hardwarové úrovni. Pokud je zjištěna dvoubitová chyba, musí být chyba ošetřena softwarově [18]. Nejčastěji je taková chyba způsobena SEU a opravuje se restartem. Druhou metodou detekce chyb používanou družicemi CubeSat jsou časovače WDT. WDT detekují odchylky v chování softwaru, které mění jeho časové charakteristiky, a v případě havárie nebo smyčky WDT resetují systém. WDT mohou být volány systémovým i aplikačním softwarem [18].

Satelity CubeSat používají k detekci chyb při přenosu dat cyklickou redundantní kontrolu. CRC se připojuje před segmentací na konec každého datagramu. Přijímač používá CRC k ověření integrity předávaných dat. Datagramy, které kontrolou CRC neprojdou, jsou obvykle zahozeny [3]. CRC má několik nevýhod. Zvyšují složitost systému a spotřebovávají značné množství již tak omezené paměti [3]. Druhou nevýhodou je, že nechrání data před úmyslným poškozením [6].

Kryptografické šifry potenciálně chrání družice CubeSat před úmyslnou manipulací s daty. Družice CubeSat jsou náchylné k odposlechu kvůli použití bezdrátových

vysílacích médií. Jakmile je zachyceno dostatečné množství dat, útočník je schopen je analyzovat a napadnout družici formou tzv. útoku přehráním [6].

1.3 Protokoly používané ve vesmíru

V následujících odstavcích je uveden přehled komunikačních protokolů podobných CSP. Čtenář tak získá představu o tom, jak jiné protokoly než CSP řeší problémy uvedené v předchozí kapitole.

Družicové sítě se od standardních pozemních sítí liší několika důležitými způsoby. Trpí větším zpožděním, vyšší chybovostí a v mnoha případech nižší přenosovou rychlostí. Kromě toho může být kvůli pohybu družic a krátkému dosahu antén často přerušen kontakt s pozemní stanicí. Z těchto důvodů jsou některé protokoly typu TCP/IP pro použití v nanodružicích nevhodné. Konkrétně TCP je nevhodný z následujících důvodů [5]:

- RTT je čas v milisekundách, který udává, jak dlouho trvá síťovému požadavku, než se dostane z výchozího místa do místa určení a zpět. Vzhledem k pohybu družice se vzdálenost od pozemní stanice mění, a proto se mění i RTT, a to v rozmezí 40-400 ms.
- Produkty s velkou a proměnlivou latencí a šířkou pásma vedou k plýtvání šířkou pásma kvůli mechanismu potvrzování TCP.
- Spojení mezi družicemi je výrazně asymetrické a negativně ovlivňuje řízení toku TCP.
- Atmosférické podmínky, záření či umělé rušení mohou způsobit BER a ztrátu paketů. TCP interpretuje tyto ztráty jako přetížení a reaguje snížením přenosové rychlosti.

Reakcí na tyto problémy s protokoly TCP v kombinaci s problémy způsobenými vesmírem bylo vytvoření protokolů určených speciálně pro vesmírnou komunikaci. Kvůli omezením v hardwaru a finančních omezením musí nanodružice využívat jednodušší komunikační protokoly a optimalizovat své využití už tak omezené kapacity pásma [5].

1.3.1 SPP - space packet protocol agentury CCSDS

Space packet protocol je protokol vyvinutý ve spolupráci významných kosmických agentur včetně NASA, JAXA, ESA atd. SPP je samostatný nosič schopný přenášet data různých druhů a typů mezi uzly a podsítěmi obsahujícími alespoň jedno z možných spojení, ať už mezi pozemní stanicí a družicí, spojením mezi kosmickými stanicemi nebo spojením uvnitř družice. V zásobníku protokolů se SPP nachází buď v aplikační vrstvě, nebo poskytuje službu shim protokolu - protokolu, jehož cílem je propojit dvě vrstvy úpravou příchozích dat. SPP obsahuje APID, což je identifikátor, který slouží k určení obsahu, zdroje a případně uživatele. Různé typy dat někdy potřebují další informace, aby se maximálně využil jejich potenciál, a tyto informace lze identifikovat právě pomocí APID [4].

1.3.2 SpaceWire

SpaceWire je typ telekomunikačního standardu a protokolu vytvořeného ve spolupráci těchto mezinárodních kosmických agentur: ESA, NASA, JAXA a RKA. Síť SpaceWire využívá sériové spoje typu bod-bod a topologii libovolného směrovacího přepínače, aby se minimalizovalo zpoždění.

Protokol je rozdělen do 6 vrstev [19]:

- Fyzická vrstva: odpovídá za konektory a kabely.
- Signálová vrstva: definuje kódování, úrovně napětí, rychlost signalizace.
- Znaková vrstva: definuje řídicí znaky používané pro správu dat.
- Výměnná vrstva: definuje protokol pro inicializaci spoje, řízení toku, detekci a obnovu chyb na spoji.
- Paketová vrstva: řídí přenos dat přes SpW linku.
- Síťová vrstva: definuje strukturu sítě SpW, přenosy dat od zdroje k cíli a řízení chyb v síťové vrstvě.

SpaceWire je jedním z komunikačních protokolů specifikovaných v požadavcích Linux4Space a jeho zařazení je předmětem další bakalářské práce [10].

1.3.3 AX.25

Amatérský X.25 je protokol vrstvy datového spoje. Pakety jsou přenášeny v malých blocích dat zvaných rámce. Existují 3 typy rámců: informační rámec, dohledový rámec, nečíslovaný rámec. Rámce se skládají mimo jiné z adresy, řídicích bitů, datových bitů a CRC [2]. Tento protokol se používá kvůli své spolehlivosti, nezávislosti na ostatních vrstvách, kompatibilitě s omezeními rádiového pásma a nízké spotřebě energie. AX.25 lze použít například k zapouzdření paketů CSP do svých rámců [5].

1.4 Technologie použité v bakalářské práci

V následujících odstavcích je uveden přehled technologií použitých v této práci. Obsahuje jak popis samotné knihovny, tak popis systému použitého k vytvoření obrazu operačního systému.

1.4.1 Libcsp

CubeSat Space Protocol je sada protokolů napsaných v jazyce C, které jsou primárně určeny pro komunikaci mezi distribuovanými vestavěnými systémy. CSP je v současné době portován na systémy FreeRTOS, POSIX a systémy používající pthreads. Je založen na modelu TCP/IP a zahrnuje mimo jiné transportní protokol, směrovač a protokoly zpráv [8]. Jeho výhodou je malá hlavička obsahující transportní a síťovou vrstvu. Dává vývojářům vestavných systémů možnost využívat možnosti zásobníku TCP/IP bez nutnosti režie v podobě hlavičky IP. Implementace je určena pro systémy s omezenou pamětí [9].

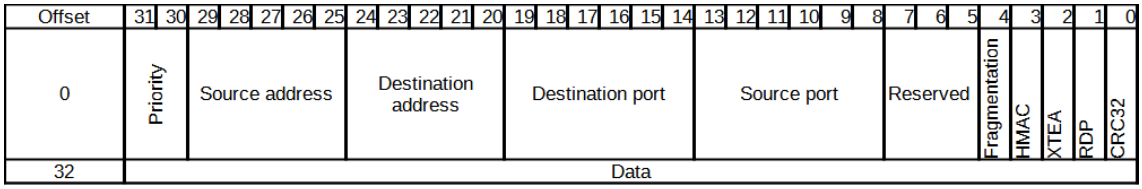
TCP/IP Layers	TCP/IP Description	CSP Layers	CSP Description
Application Layer	HTTP, FTP, DNS, DHCP, TLS, SMTP	Layer 1 - Drivers	CAN, I2C, UART
Transport Layer	TCP,UDP	Layer 2 - MAC Interfaces	I2C, CAN, RS232(KISS), Loopback
Network Layer	IPv4, IPv6	Layer 3 - Network Router	32 nodes
Link Layer	Ethernet, Wireless, LAN	Layer 4 - Transport Layer	UDP, RDP

Obrázek 1.3: Porovnání vrstev modelu TCP/IP a CSP

CSP se stejně jako TCP/IP skládá ze 4 vrstev [9].
Vrstva 1:
CSP poskytuje několik ovladačů, a to CAN, I2C, UART. Nejsou poskytovány pro všechny platformy, ale jsou vyžadovány pro provoz libcp. Pokud neexistuje ovladač, který by podporoval platformu, na které má být libcsp provozována, je nutné naprogramovat vlastní ovladač.

Vrstva 2:
CSP má rozhraní pro I2C, CAN, RS232 (KISS) a loopback. Soubor csp_interface.h obsahuje definice funkcí rx a tx potřebných k vytvoření vlastního rozhraní. Během inicializace se jednotlivá rozhraní načtou do propojeného listu, který je přístupný směrovači.

Vrstva 3:
Cílem routeru je přijímat příchozí pakety a přiřazovat je do správné fronty Za tímto účelem musí každá úloha vytvořit soket a zavolat funkci *accept()*. Tím může úloha naslouchat na portu číslo v síti. Po zavolání funkce *accept()* se úloha zablokuje a čeká na příchozí zprávu. Jakmile se spojení otevře, úloha se probudí. Router se za běhu podívá na 32-bitovou hlavičku, která obsahuje zdrojovou adresu, doručovací adresu a čísla portů pro spojení. Všechny směrovací tabulky jsou předem naprogramovány v subsystémech. Tabulka obsahuje jednotlivé cesty do každého z 32 možných uzlů v síti a další výchozí trasu.



Obrázek 1.4: CSP hlavička

Vrstva 4:
Libcsp poskytuje dva protokoly transportní vrstvy, RDP a UDP. UDP je jednoduchý protokol, který nezaručuje integritu a správné pořadí odesílaných dat. Datagramy odeslané pomocí protokolu UDP mohou dorazit nesprávně seřazené, duplicitní nebo nemusí dorazit vůbec. Detekci a ošetření chyb musí řešit samotná aplikace. Protokol UDP je vhodný pro systémy reálného času, kde se upřednostňuje zahazování paketů před čekáním na doručení. Protokol RDP je implementací protokolů RFC908 (reliable data protocol) a RFC1151 (reliable data protocol v2). Poskytuje funkce, jako je třícestný handshake, řízení toku a vyrovnávací paměť pro data.

Libcsp má sedm rezervovaných portů pro funkce csp. Porty 0 - 6 se používají pro vzdálenou správu programů. Obsahují funkce pro restart, uvolnění paměti, seznam aktuálních procesů, uvolnění vyrovnávací paměti, uptime, ping a obecnou správu.

1.4.2 Yocto

Yocto je open-source projekt, který slouží jako komplexní rámec pro vytváření vlastních distribucí Linuxu pro vestavěné systémy. Vznikl v roce 2011 ve spolupráci OpenEmbedded, nadace Linux Foundation a 21 dalších organizací. Yocto se zaměřuje na flexibilitu, přizpůsobitelnost a škálovatelnost a umožňuje vývojářům vytvářet vlastní platformy založené na Linuxu, které přesně splňují požadavky různých vestavných aplikací. Jádro Yocto používá strukturovanou metodiku, která zjednodušuje proces vývoje. Poskytuje sadu nástrojů, šablon a komponent, které vývojářům umožňují efektivně konstruovat a nasazovat vestavné systémy Linux. Architektura

projektu se točí kolem konceptu vrstev, což uživatelům umožňuje efektivně organizovat a spravovat jejich úpravy. Díky tomuto vrstevnatému přístupu Yocto podporuje modularitu a opakované použití. Vrstvy lze libovolně upravovat, přidávat a odebírat. Klíčovou součástí je nástroj pro automatizaci sestavování BitBake vyvinutý v rámci OpenEmbedded, který vývojářům umožňuje vytvořit vlastní distribuci Linuxu specifickou pro jejich prostředí. Jako referenci poskytuje Yocto Poky, velké množství upravitelných receptů, s jejichž pomocí lze vytvořit vlastní vestavěný operační systém. Yocto nabízí možnost spouštět a testovat vytvořené obrazy operačního systému pomocí emulátoru QEMU [22].

1.4.3 Bitbake

Bitbake je nástroj podobný Pythonu a Bash, který slouží k vytváření balíčků a potenciálně i celých operačních systémů. Bitbake se řídí tzv. recepty s příponou .bb. Recepty obsahují popis, zdrojový kód, ze kterého je balíček sestaven, závislosti, nastavení a pokyny k instalaci. Bitbake je schopen sledovat závislosti a na jejich základě určovat pořadí, v jakém se balíčky instalují. Bitbake je také schopen vytvořit obraz operačního systému obsahující jádro a kořenový souborový systém. Kromě receptů existují také soubory .bbappend a .bbclass. Přípona .bbappend je přípona receptu, kterou nástroj pro automatizaci sestavování BitBake připojí na konec receptu se stejným názvem. Soubory s rozšířením receptu umožňují upravit nastavení pro konkrétní distribuci, aniž by bylo nutné upravovat původní recept. Na adrese lze pomocí rozšíření receptů upravit funkce a proměnné z původního receptu a přizpůsobit. Soubory .bbclasses jsou soubory tříd, které slouží k abstrahování společných funkcí a jejich následnému sdílení mezi více recepty. Příkladem .bbclass je použití CMake v receptu, které je řešeno pomocí cmake.bbclass [22].

1.4.4 Pojmenovací konvence projektu Yocto

Podle dokumentace Yocto by se adresáře a recepty v nich měly řídit konvencí recipes-category/name_recipe/name_recipe_version.bb. Kategorie a název mohou obsahovat podtržítka, verze nikoli. Kromě těchto vlastností by měl každý recept obsahovat proměnné SUMMARY, DESCRIPTION, HOMEPAGE a BUGTRACKER (pokud existuje systém oznamování chyb). K dispozici je také úplný seznam proměnných poskytovaných systémem Yocto a pořadí, v jakém by se měly objevit v jednotlivých receptech. Úlohy jako *do_compile()* a *do_install()* by měly být zapsány v pořadí, v jakém jsou obvykle prováděny. Úplný seznam proměnných a úloh naleznete v dokumentaci Yocto. Každý recept Yocto musí obsahovat proměnné LICENSE a LIC_FILES_CHECKSUM. LIC_FILES_CHECKSUM je hash, který slouží ke kontrole, zda se změnila licence [22].

Celá práce je vytvořena pro projekt Linux4Space. CSP je jedním z komunikačních protokolů definovaných v implementačním plánu projektu Linux4Space [10] a výsledky jsou konzultovány s jeho členy.

1.5 Linux4Space

Linux4Space je společný projekt s otevřeným zdrojovým kódem, jehož cílem je vytvořit distribuci Linuxu založenou na Yocto a přizpůsobenou pro vesmírné aplikace. Distribuce Linuxu je určena pro payload sekce družic CubeSat a má splňovat standardy ECSS. Na projektu spolupracují Technická univerzita v Liberci, Národní centrum letectví a kosmonautiky a Ústav technické a experimentální fyziky ČVUT [11].

Projekt Linux4Space definoval případ použití pro účely vývoje [10]:

- Payload systém - distribuce Linux4Space je primárně určena pro payload sekci nikoliv pro bus sekci.
- Neobsahuje grafické uživatelské rozhraní - Linux4Space neposkytuje grafické uživatelské rozhraní, je navržen pro ovládání pomocí příkazového řádku nebo jako zcela autonomní systém.
- Operace v reálném čase - Linux4Space zaručuje načasování zpracování událostí.
- Standardy ESA - Linux4Space splňuje standardy stanovené Evropskou kosmickou agenturou.
- Distribuce založená na Yocto - definice systému Linux4Space se skládá z meta-vrstev YOCTO.
- Bezpečnost a spolehlivost - systém Linux4Space je koordinován s open-source projektem ELISA, aby byl systém bezpečný a spolehlivý pro kritické aplikace.
- Komunitní báze - Linux4Space je open-source projekt pro všechny zájemce o vesmírné aplikace.

2 Počáteční analýza způsobu integrace libcsp do Linux4Space

V rámci úkolů, které bylo třeba splnit pro úspěšné dokončení práce, bylo třeba učinit různá rozhodnutí a etapy. Problémy byly konzultovány s dalšími vývojáři Linux4Space a rozhodnutí byla přijímána podle jejich návrhů.

2.1 Použitý ovladač

Pro účely hardwarové demonstrace bylo nutné zvolit jedno rozhraní I2C nebo UART. I2C je typ sériové komunikace typu master-slave vytvořený společností Phillips. Libcsp neobsahuje ovladač potřebný pro provoz I2C, ale má rozhraní určené pro I2C. Pro instalaci modulu I2C na obraz Raspberry Pi vytvořený pomocí yocto bylo nutné v souboru local.conf nastavit ENABLE_I2C = ”1” a KERNEL_MODULE_AUTOLOAD_rpi += ” i2c-dev i2c-bcm2708”. Problém nastal v samotném návrhu I2C a Raspberry Pi. Je to proto, že Raspberry Pi je schopen fungovat pouze jako master a úloha vyžadovala komunikaci mezi dvěma deskami Raspberry Pi. Existuje knihovna Pigpio, která změní nastavení pinů GPIO, ale to by vyžadovalo integraci Pigpio do knihovny libcsp. Také by to později způsobilo problémy v případě obousměrné komunikace. Komunikace by také probíhala prostřednictvím funkce BSCX knihovny Pigpio, nikoliv prostřednictvím knihovny libcsp. Z těchto důvodů bylo od implementace I2C upuštěno. Místo se autor rozhodl použít UART.

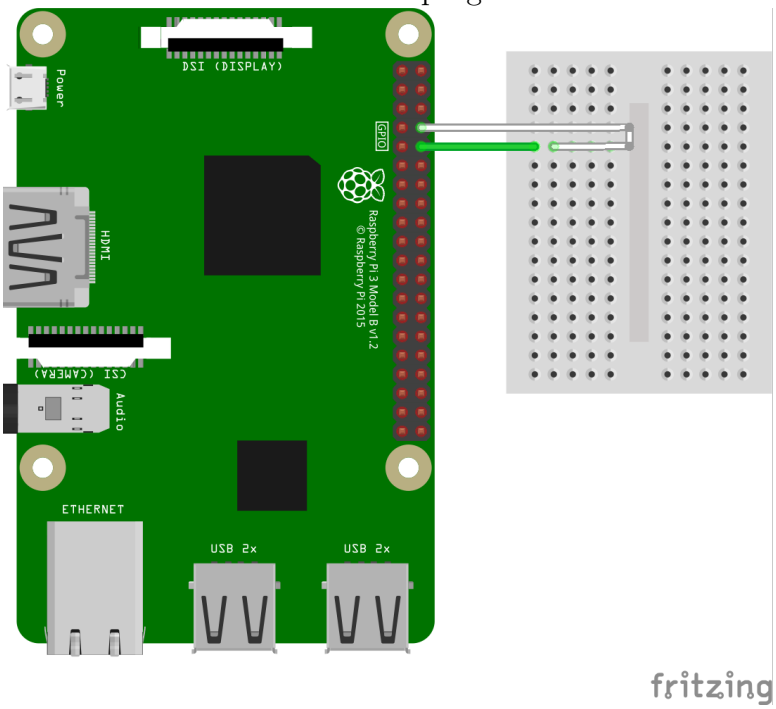
2.2 UART

UART je sériové asynchronní rozhraní. Libcsp poskytuje rozhraní a ovladač pro sériový port RS232(KISS). Na Raspberry Pi jsou za UART zodpovědné piny 8 a 10. V klasickém nastavení operačního systému by uživatel nastavil UART v raspi-config. V případě vytváření obrazu pomocí Yocto je třeba do souboru local.conf vložit ENABLE_UART = ”1” a SERIAL_CONSOLE = ”115200 ttyS0”.

První test UART byl proveden pomocí zapojení loopback a programu Minicom. Nastavení minicomu bylo:

```
+-----------------------------------------------------------------------+
| A -    Serial Device      : /dev/ttyS0                                |
| B - Lockfile Location     : /var/lock                                 |
| C -   Callin Program      :                                           |
| D -  Callout Program      :                                           |
| E -    Bps/Par/Bits       : 115200 8N1                                |
| F - Hardware Flow Control : No                                        |
| G - Software Flow Control : No                                        |
| H -     RS485 Enable      : No                                        |
| I -   RS485 Rts On Send   : No                                        |
| J -  RS485 Rts After Send : No                                        |
| K -  RS485 Rx During Tx   : No                                        |
| L -  RS485 Terminate Bus  : No                                        |
| M - RS485 Delay Rts Before: 0                                         |
| N - RS485 Delay Rts After : 0                                         |
|                                                                       |
|    Change which setting?                                              |
+-----------------------------------------------------------------------+
```

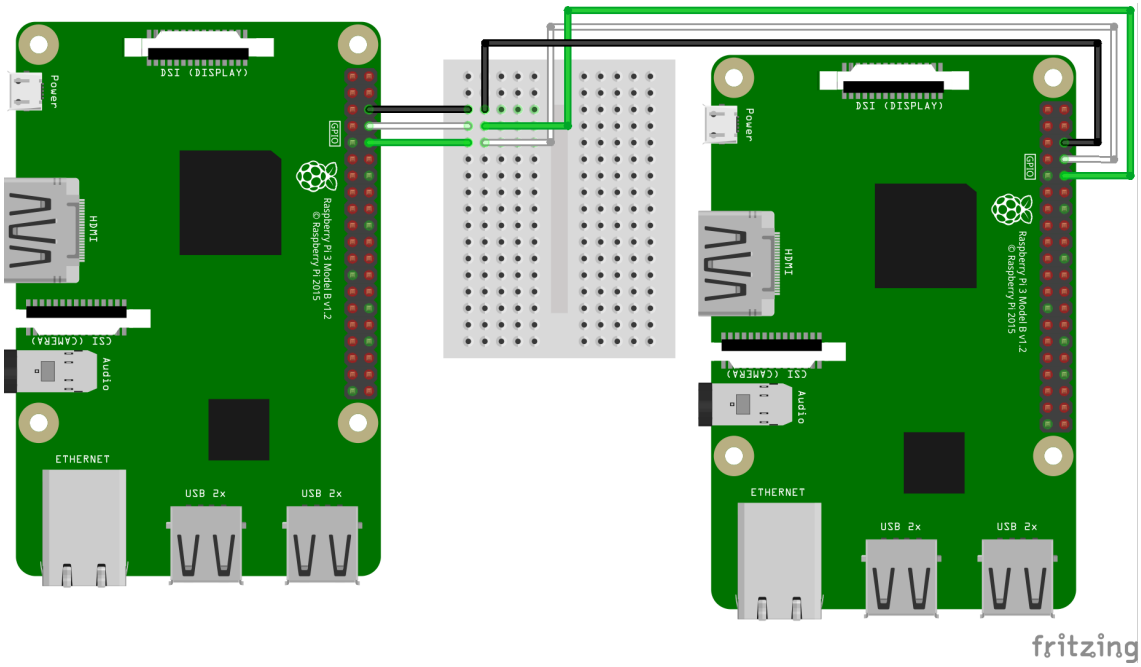
Obrázek 2.1: Nastavení programu Minicom



Obrázek 2.2: Zapojení Raspberry Pi přes GPIO loopback

Pin Tx (8) byl připojen k pinu Rx (10) a program Minicom byl spuštěn na /dev/ttyS0 s přenosovou rychlostí 115200 baud a vypnutým hardwarovým řízením toku. Kromě mezerníku a nového řádku přenášel znaky bez problémů.
Při stisknutí mezerníku nebo nového řádku docházelo k problému se zpětnou smyčkou, kdy se znak nového řádku posílal stále dokola, což vedlo k tomu, že se celá obrazovka vymazala a nebylo možné napsat vůbec žádný znak. Jediné, co pomohlo, bylo vypnutí a zapnutí programu Minicom.

Druhý test zahrnoval vzájemné propojení dvou RPI přes piny GPIO UART. GND je připojeno k GND, Rx a Tx jsou překříženy. Dopadlo to úplně stejně jako první test.



Obrázek 2.3: Propojení dvou Raspberry Pi prostřednictvím GPIO

Komplikace nastaly, když se jednalo o komunikaci prostřednictvím libcsp, nikoliv minicom. Minicom zajišťuje a stará se o synchronizaci. Ovladač libcsp uart je napsán a navržen pro USB-UART převodník, nikoli pro GPIO piny. Pokud je spojení navázáno přes piny GPIO a a není pro ně naprogramován ovladač, dojde k chybě, kdy funkce *read()* vlákna usart_rx po načtení hodnot deskriptoru souboru vrátí prázdnou hodnotu a program se automaticky ukončí.

2.3 Restart v libcsp

2.3.1 Restart s libcsp-develop

První verze receptu integrovala verzi knihovny pocházející z větve develop repozitáře libcsp GitHub. V rámci testování byl spuštěn kód z příkladů poskytnutých vývojáři knihovny, csp_server_client.c, což vedlo k tomu, že se systém pokaždé restartoval hned po spuštění programu. Tento jev se však vyskytl pouze v systému vytvořeném prostřednictvím Yocto, nikoliv ve WSL Ubuntu nebo ve virtuálním systému spuštěném ve Virtual Boxu. Postupně bylo zjištěno, co tento restart způsobuje. První krok spočíval v uložení výstupu programu do textového souboru, takže i v případě restartu bylo možné zobrazit poslední výstup programu. Po přečtení tohoto výstupu bylo zjištěno, že k restartu došlo vždy po zavolání funkce *csp_reboot()*. Do souboru csp_server_client.c však vývojáři vložili komentář, který tvrdil, že systém nemá

funkci *csp_sys_reboot()*, a proto by k restartu nemělo dojít. Postupné prohledávání kódu v repozitáři github odhalilo, že vývojáři funkci *csp_sys_reboot()* z libcsp-2.0 odstranili a nahradili ji funkcí zpětného volání. Komentář vývojářů byl jen součástí neudržovaného starého kódu. Tato funkce je však úspěšně volána pouze v případě, že program běží s dostatečnými právy, což v případě virtuálního systému ve VirtualBoxu a WSL nebylo, ale v případě Yocto program běžel s právy sudo, a proto k restartu docházelo. Tento problém se stal bezpředmětným, když bylo rozhodnuto o integraci libcsp-1.6 z důvodu neexistence stabilní verze libcsp-2.0.

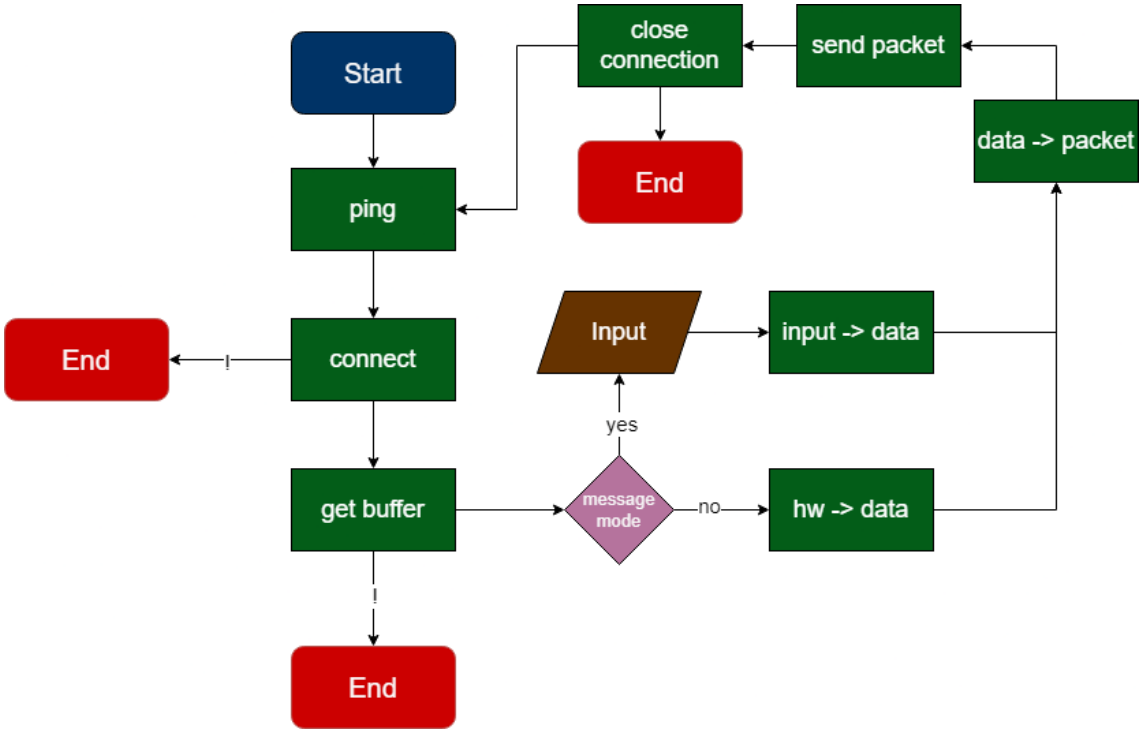
2.3.2 Restart s libcsp-1.6

V některých případech, například při SEE/SEU, je nutné resetovat CubeSat nebo jeho payload. Libcsp 1.6 neobsahuje funkci potřebnou k vyvolání restartu systému. Pokud klient odešle požadavek na restart pomocí funkce *csp_reboot()* nebo pokud na port 4 dorazí správně naformátovaný paket, obsluha služby se pokusí zavolat funkci *csp_sys_reboot()*. Tato funkce není implementována. Každý operační systém, pro který existuje libcsp, používá k vyvolání restartu jinou metodu a je nutné, aby si ji uživatel naprogramoval sám a přidal ji při kompilaci svého programu.V případě operačního systému založeného na Linuxu by to znamenalo zavolat ve vytvořené funkci *reboot(LINUX_REBOOT_CMD_RESTART)*. Je třeba poznamenat, že tato metoda funguje pouze tehdy, když má server práva volat *reboot()* z linux/reboot.h.

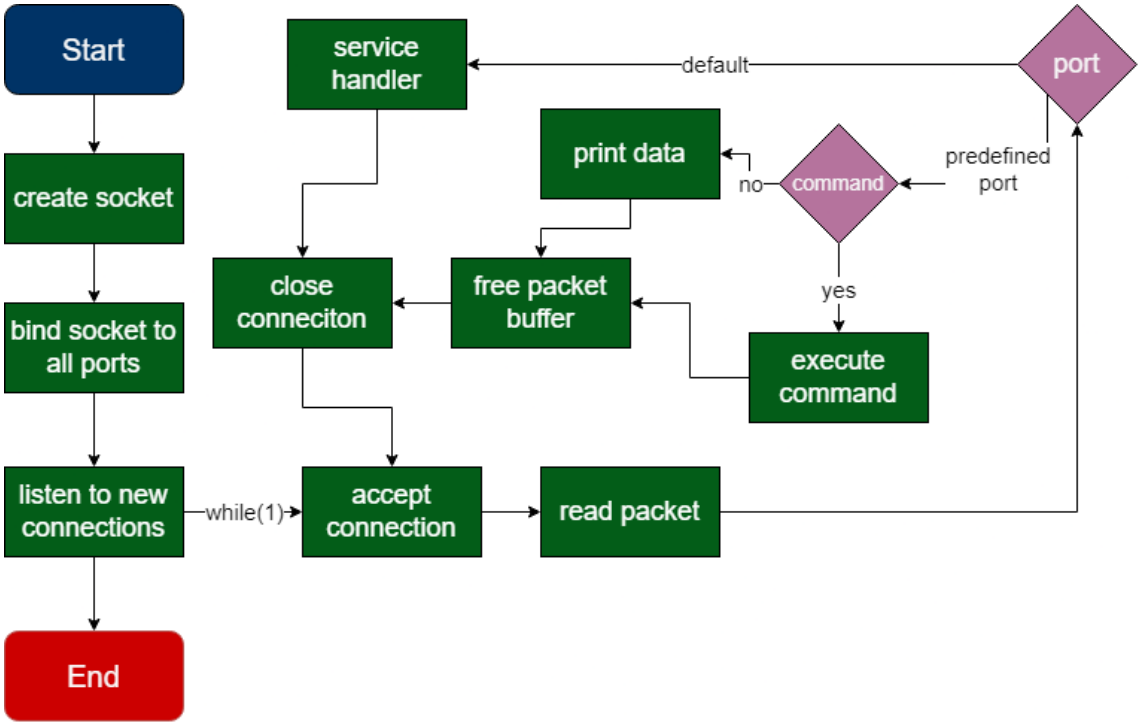
2.4 Demonstrační kód

2.4.1 Návrh demonstračního kódu v jazyce C

Autor této práce se rozhodl použít jako příklad z velké části kód, který je dostupný z knihovny libcsp. Důvodem tohoto rozhodnutí bylo vyhnout se problémům s chybami v software a usnadnit jeho údržbu. Kód napsaný vývojáři knihovny libcsp byl testován a demonstruje správné použití protokolu a osvědčené postupy. Je také snazší udržovat jej v případě vydání nové verze knihovny.
V repozitáři libcsp na githubu je adresář examples, který obsahuje několik různých příkladů použití knihovny libcsp. Jedním z těchto příkladů je soubor csp_server_client.c. V tomto souboru je napsán kód, který vytváří server a klienta ve dvou vláknech současně a provádí mezi nimi loopback test. Úprava provedená v této práci rozdělila tento soubor na dvě části, z nichž jedna vytváří vlákno klienta a druhá vlákno serveru. Druhá úprava přišla v podobě přidaného režimu zpráv, kdy při zadání argumentu -m klient vždy po vytvoření spojení vyzve k odeslání zprávy. Uživatel má pak možnost odeslat textovou zprávu, která se pak zobrazí jako data odeslaná v paketu na straně serveru. Třetí modifikace obsahuje funkci, do které je paket vložen, a pokud zpráva obsahuje znak ”c ” před odesílanými daty, pak je zpráva předána na příkazový řádek, kde je následně provedena jako příkaz.



Obrázek 2.4: Diagram funkčnosti klientského kódu



Obrázek 2.5: Diagram funkčnosti kódu na straně serveru

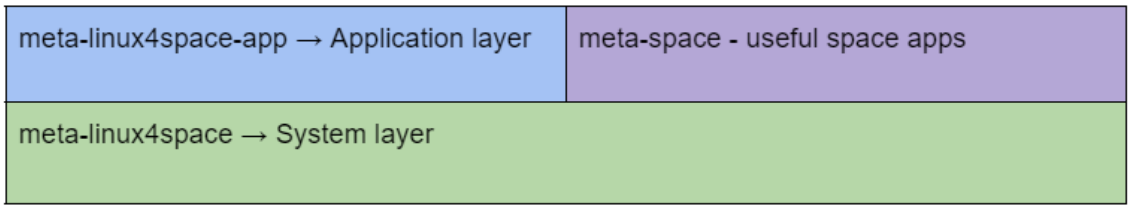
2.5 Vrstvy Linux4Space

2.5.1 Recepty BitBake a jejich místo ve vrstvách Linux4Space

První verze receptu libcsp čerpala kód z vývojové větve GitHub repozitáře libcsp. Recept obsahoval automatickou kontrolu commit verzí, integraci nástroje CMake pro instalaci knihoven a postup Yocto pro instalaci knihoven bez verzí. Linux4Space jako seriózní projekt vyžadoval použití stabilní verze a knihovna libcsp má ve svém repozitáři označené stabilní verze, takže recept musel být upraven. V rámci integrace knihovny libcsp do bylo tedy nakonec nutné napsat jeden recept a soubor .bbappend, který jej doplňuje. Soubor .bbappend se v dokumentaci nachází pouze jako ukázkový příklad, jak upravit recept knihovny, aniž by bylo nutné jej přepisovat. Kromě receptu pro integraci knihovny libcsp byl vytvořen také recept pro přesun a instalaci demo souborů a programů do výsledného obrazu. Recept se nazývá libcsp-demo. Při vytváření nových verzí operačního systému používaných pro demonstraci HW došlo vždy k úplnému obnovení všech nastavení a změn v rootfs. Místo kopírování a následné kompilace programů pro testování knihovny libcsp bylo efektivnější celý proces automatizovat a vytvořit recept, který tyto úkoly provede za uživatele.

2.5.2 Vrstvy projektu Linux4Space

Referenční distribuce Linux4Space se skládá ze dvou hlavních vrstev. Systémovou vrstvu - meta-Linux4Space a aplikační vrstvu - meta-Linux4Space-app. Systémová vrstva obsahuje zavaděč, jádro a služby a recepty kernel-space. V aplikační vrstvě se nacházejí recepty pro uživatelské aplikace, které vyžadují prostorový use-case. Recept libcsp jako komunikační protokol se nachází v systémové vrstvě v části recipes-connectivity. Spolu s ním se v této vrstvě nacházejí recepty pro protokol SpaceWire a protokol KISS [10].



Obrázek 2.6: Vizualizace vrstev projektu Linux4Space

3 Implementace

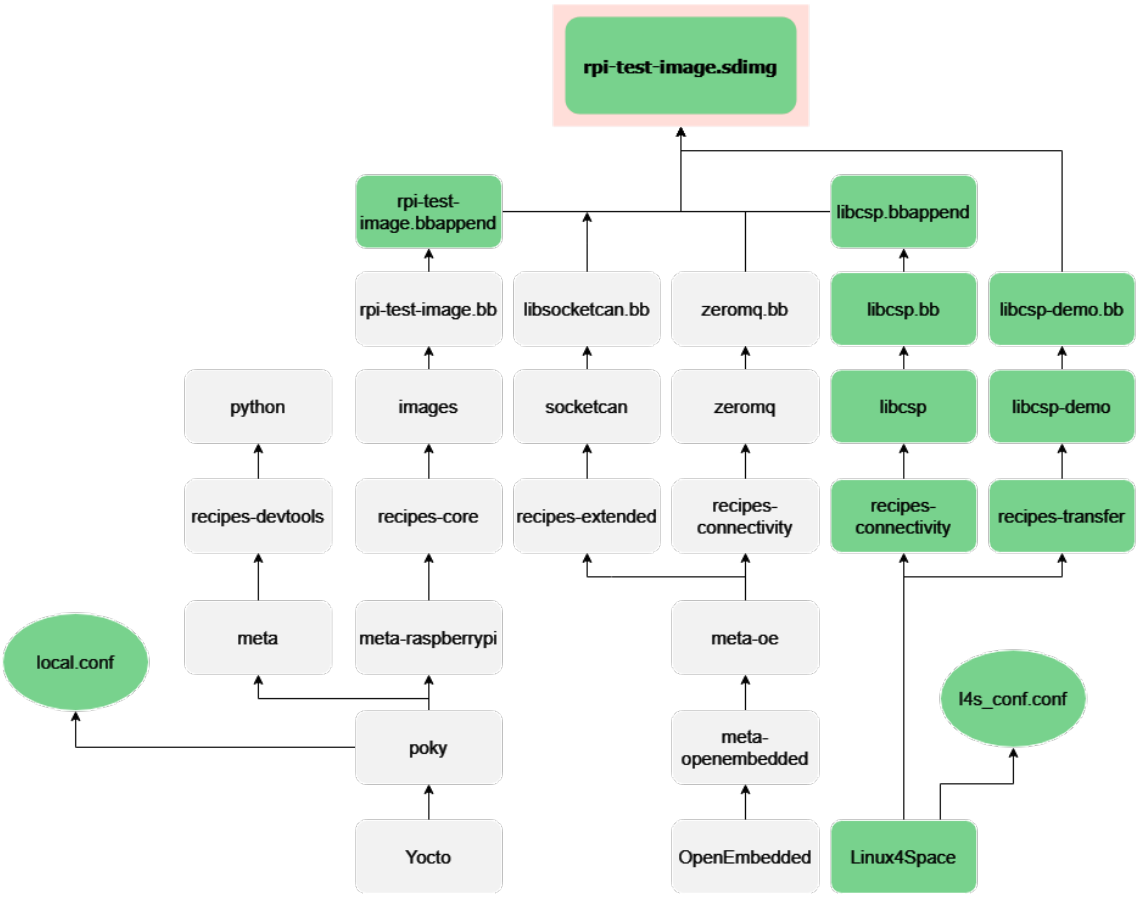
V této kapitole je popsáno, jak byl realizován konečný recept a ukázka hardwaru. Obsahuje vytvoření obrazu operačního systému, vysvětlení receptů a ovládání programů nainstalovaných v rámci demonstrace.

3.1 Hardwarová implementace

Hardwarová implementace a demonstrace se skládá ze dvou desek Raspberry Pi 3b, 6 kabelů DuPont female-to-female a 4 převodníků PL2303 USB na TTL/UART. Převodníky mají piny Tx, Rx a GND připojené pomocí kabelů a jsou připojeny přes porty USB k deskám Raspberry Pi. Kromě těchto komponent desky ještě doplňují dotykové obrazovky a klávesnice pro ovládání a zadávání příkazů. Desky jsou napájeny přes adaptéry USB-B 5V a do každé desky je vložena 32Gb paměťová karta, na které je nahrán operační systém. Obraz operačního systému má příponu rpi-sdimg. K nahrání operačního systému na kartu byl použit program Raspberry Pi Imager.

3.2 Operační systém

Samotný operační systém se skládá z kombinace vrstev Yocto, vrstev OpenEmbedded a vrstvy vytvořené pro účely této práce. Vrstva Yocto Raspberry Pi obsahuje vše potřebné k vytvoření vlastního operačního systému Raspberry Pi. Z vrstvy OpenEmbedded byly použity recepty pro libsocketcan, zeromq. Vrstva Linux4Space obsahuje recepty libcsp a libcsp-demo. Kromě receptů byly vytvořeny soubory .bbappend pro přidání některých funkcí zmíněných v dalších částech a konfigurační soubory .conf. Na obrázku vidíte strukturu a vrstvy celého projektu. Soubory a adresáře, které byly vytvořeny nebo upraveny od základu, jsou na obrázku 3.1 označeny zeleně.



Obrázek 3.1: Schéma vrstev a souborů, z nichž byl operační systém vytvořen

3.3 Úpravy a vytvořené soubory

3.3.1 Konfigurační soubory

V souboru local.conf byla upravena proměnná MACHINE Tato proměnná určuje, pro který stroj bude finální sestavení systému. Pokud proměnná není změněna, je její výchozí nastavení qemux86-64, což je architektura emulovaná emulátorem QEMU.

Na konec souboru bylo přidáno několik proměnných IMAGE_INSTALL:append. Ty slouží k instalaci programů pro testování připojení, jako jsou minicom a picocom, programů pro přepisování a čtení souborů, jako jsou vim nebo nano,
a packagegroup-core-buildessential, což je soubor programů pro sestavování a kompilaci kódu, jako jsou gcc, g++, make a další. Do IMAGE_FSTYPES:append bylo nutné přidat rpi-sdimg. Bez něj by se obraz operačního systému kompatibilní s Raspberry Pi nevytvořil.

Ve výchozím nastavení některá konfigurační nastavení doporučují balíčky pro firmware z linux-firmware-rpidistro Tyto balíčky obsahují bloby firmwaru, které s sebou nesou licenci od společnosti Synaptics. Součástí této licence je klauzule, která by mohla být potenciálně použita jako killswitch. Z tohoto důvodu lze recept použít pouze v případě, že je do položky LICENSE_FLAGS_ACCEPTED vloženo synaptics-killswitch jako potvrzení rizika, které použití tohoto receptu přináší.

Druhý konfigurační soubor upravený v tomto projektu je l4s_conf.conf. Do tohoto konfiguračního souboru byly vloženy dodatky čistě pro vrstvu meta-linux4space. Důvodem tohoto oddělení je vytvoření modularity, kdy vývojář, který chce používat vrstvu meta-linux4space, nemusí jednotlivě balík po balíku procházet balíky obsažené ve vrstvě a připojovat je, ale stačí mu přidat samotnou vrstvu do nastavení BitBake.

Recepty-připojení a recepty-přenos jako adresáře se řídí konvencemi pro pojmenování adresářů ve vrstvách Yocto. Totéž platí pro adresáře libcsp a libcsp-demo.

3.3.2 Recepty

Libcsp-demo

Recept libcsp-demo.bb v první verzi obsahoval pouze úlohu *do_install*, která ve výsledném rootfs vytvořila adresář testdata a zkopírovala do něj kód pro integrační testování. Používal licenci MIT a neinstaloval zkompilované binární soubory do adresáře /usr/bin. Testovací kód převzal z lokálního adresáře. Nakonec byl přepsán, když byl recept libcsp změněn z větve libcsp develop na větev libcsp-1. Recept byl vylepšen a byly přidány proměnné vyžadované jmennými konvencemi Yocto, jako jsou SUMMARY a DESCRIPTION, se správným pořadím. Testovací kód čerpá z repozitáře Linux4Space Gitlab a používá jednu konkrétní revizi - recept ji najde pomocí hashe revize. Obsahuje 2 úlohy: do_compile a do_install. *Do_compile* zkompiluje kód stažený z repozitáře GitLab. V *do_install* přesune zkompilovaný kód do /usr/bin. Druhá část *do_install* vytvoří adresář testdata v adresáři /home uživatele linuxforspace a nakopíruje do něj všechny stažené soubory s příponou .c.

Libcsp-1.6

První recept libcsp.bb měl několik problémů. Převzal zdrojový kód knihovny z nestabilní větve repozitáře github, používal jakousi instalaci Yocto pro knihovny bez verzí a nedával uživatelům možnost přizpůsobit recept pro své účely. Současná verze receptu libcsp čerpá kód z aktuálně nejnovější stabilní verze libcsp 1.6. Tato verze knihovny používá jinou licenci než libcsp 2.0. Libcsp 2.0 používá licenci MIT, zatímco libcsp 1.6 používá licenci LGPL 2.1. Licence LGPL vyžaduje výrazné označení a komentování, pokud byly ve zdrojovém kódu provedeny nějaké změny. Aby byla zachována funkčnost pro budoucí nová vydání knihovny, byl do SRCREV vložen

hash revize 1.6. Knihovna libcsp 1.6 nepoužívá k sestavení knihovny cmake, ale pouze nástroj pro automatizaci sestavování waf.

Waf je nástroj pro automatizaci sestavování napsaný v jazyce Python a je řízen skriptem v úložišti nazvaným wscript. Wscript obsahuje funkce potřebné k sestavení knihovny, které knihovnu konfigurují, sestavují a instalují. V rámci konfigurace je tedy možné nastavit různé vlajky, které mění výsledný kód. Právě zadáním těchto vlajek si uživatelé vybírají, které ovladače chtějí nainstalovat a jaké funkce má knihovna mít. Příznaky, které by bylo vhodné, aby uživatel v případě potřeby změnil, byly identifikovány a vloženy do proměnných: konkrétně DRIVERS a FLAGS. Proměnná DEPENDS potenciálně závisí na proměnné DRIVERS, a proto také patří do této skupiny. V jedné z dřívějších verzí bylo také možné změnit umístění, kam bude knihovna nainstalována, a operační systém, pro který waf knihovnu sestaví. Problém spočíval v tom, že změna umístění by bojovala s proměnnými prostředí, které Yocto obsahuje. Možnost změnit systém je k ničemu, protože Yocto nelze použít pro sestavení knihovny pro Windows nebo MacOS a samotný recept není přizpůsoben pro sestavení knihovny pro FreeRTOS.

Recept má v sobě zapsány 2 úlohy: do_configure a do_install. *Do_configure* jednak konfiguruje ./waf configure pomocí vlajek, jako je toolchain a příznaky uložené v FLAGS a DRIVERS, a jednak upravuje soubory waf a wscript, které se nacházejí v úložišti libcsp. Tyto soubory mají v prvním řádku nesprávně uvedenou proměnnou prostředí pythonu. Příkaz sed změní #!/usr/bin/env python na #!/usr/bin/env python3. *Do_install* pak nainstaluje knihovnu pomocí sestavení ./waf a odstraní soubory statické knihovny .a, které jsou automaticky generovány programem BitBake.

Poslední část receptu vytvoří 3 balíčky: libcsp, libcsp-dev a libcsp-dbg Libcsp obsahuje soubory .so a libcsp-dev obsahuje hlavičkové soubory /usr/include.
Libcsp-dbg existuje, protože soubory .debug jsou automaticky generovány programem BitBake jako součást funkce *split_and_strip_files* v meta/packages/package.bbclass. Jednou z možností bylo potlačit tuto funkci pomocí vlajek INHIBIT_PACKAGE_DEBUG_SPLIT = ”1” a INHIBIT_PACKAGE_STRIP = ”1”
Druhou možností bylo přibalit ladicí symboly do balíčku libcsp-dbg s tím, že pokud je někdy nějaký uživatel bude potřebovat k ladění svého programu, může je použít.

Soubory .bbappend

Soubor libcsp.bbappend je napsán jako ukázkový soubor pouze pro nastavení proměnných v základním receptu. Knihovna Libcsp obsahuje několik předprogramovaných ovladačů, které je třeba vložit do proměnné DRIVERS. Pokud byly do DRIVERS vloženy ovladače zeromq nebo can, musí být příslušné knihovny přidány do DEPENDS.

Wscript libcsp obsahuje celkem 16 příznaků, které lze konfigurovat v ./waf configure. Patří mezi ně povolení CRC32, deduplikace paketů a další. Bylo rozhodnuto, že vzhledem k nutnému zabezpečení přenosu dat v satelitu bude automaticky použit příznak pro CRC32. Druhým automaticky spouštěným příznakem je oprávnění python3. Všechny tyto příznaky lze konfigurovat pomocí proměnné FLAGS.

Poslední soubor s příponou receptu zapsaný před vytvořením operačního systému je rpi-test-image.bbappend. Nebylo žádoucí spouštět všechny testy a programy pod účtem správce. rpi-test-image.bbappend tento problém řeší. V prvním kroku přidává do příkazu IMAGE_INSTALL příkaz sudo. Poté vytvoří uživatele se jménem linuxforspace a heslem "password". Posledním krokem, který provede, je přidání vytvořeného uživatelského účtu a celé skupiny sudo do souboru sudoers.

3.4 Instalace, provoz a předvedení funkčnosti

3.4.1 Loopback

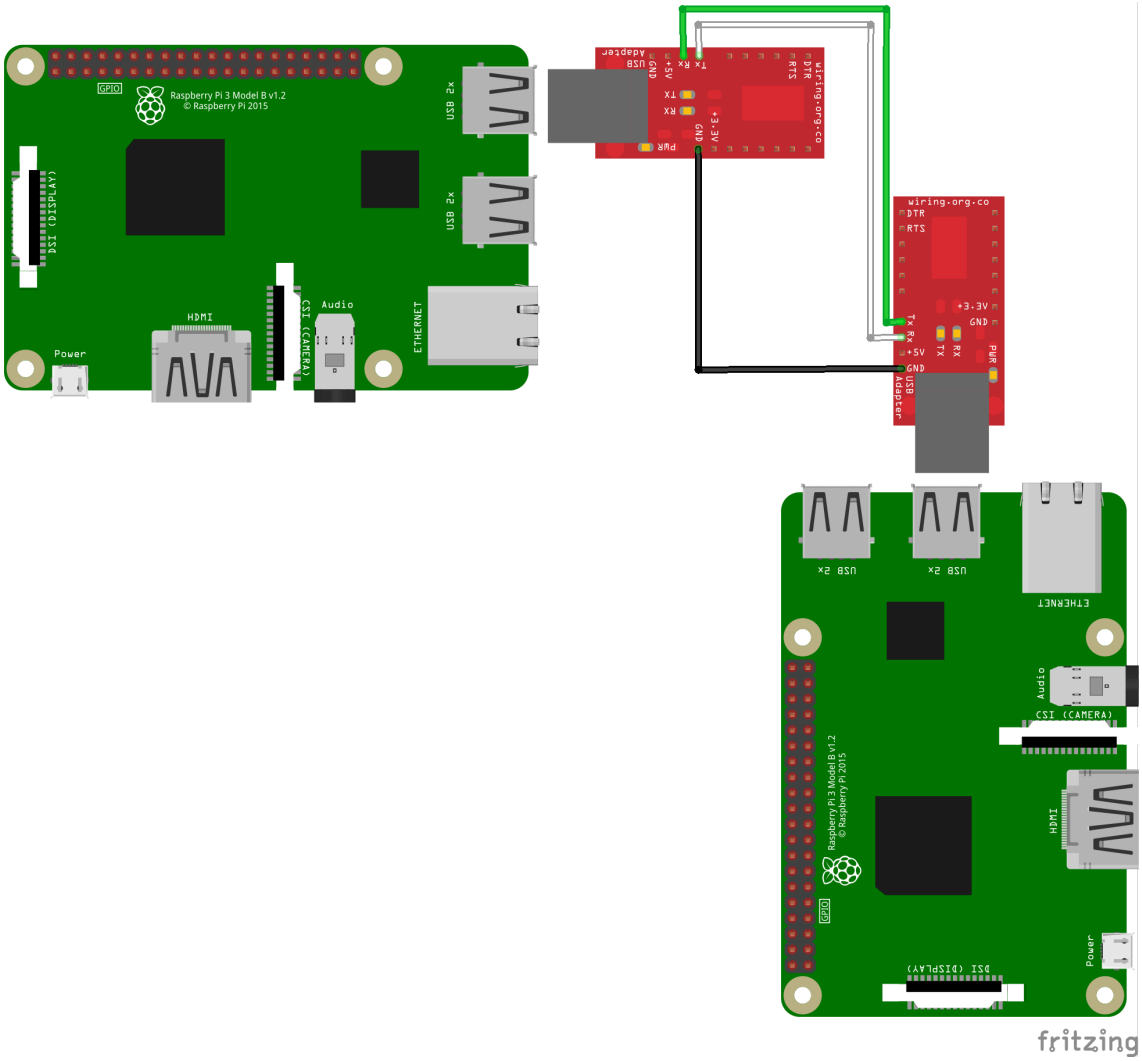
Pomocí sestavovacího systému BitBake byl vytvořen obraz systému rpi-sdimg a nahrán na 32Gb kartu SD pomocí programu Raspberry Pi Imager. Karty byly poté vloženy do dvou počítačů Raspberry Pi 3b. Po připojení k napájení se systém spustil a bylo možné se přihlásit k jednomu ze dvou účtů: root a linuxforspace. Root nevyžaduje heslo, linuxforspace ano. Otestování loopback rozhraní vyžaduje spuštění předinstalovaného programu server-client. Lze jej spustit buď bez argumentů, nebo s argumenty. Pokud je spuštěn bez argumentů, výstupem je nekonečně se opakujících 5 řádků:

```
SERVICE: Ping received
Ping address: 1, result 1 [mS]
reboot system request sent to address: 1
csp_sys_reboot not supported - no user function set
Packet received on MY_SERVER_PORT: Hello World (1)
```

Hello World (x) se při každém výstupu zvětší o 1. Pokud je server-client spuštěn s argumentem -t, bude během 3 sekund odesláno a přijato co nejvíce paketů. Na konci se pak vypíše výsledný stav, kolik paketů bylo úspěšně přijato.

3.4.2 Jednosměrná komunikace

Jednosměrná komunikace vyžaduje, aby byly dva počítače Raspberry Pi propojeny pomocí převodníku USB-UART. Tx a Rx piny jsou překřížené, Gnd piny jsou spojené.



Obrázek 3.2: Zapojení jednosměrné komunikace pomocí převodníků USB-UART

Na jednom Raspberry Pi běží serverový program, na druhém klientský program. Jako argumenty se vkládá -k<adresa převodníku USB-UART>, v případě tohoto projektu je to -k/dev/ttyUSB0. Druhým argumentem je -a<celé číslo>. Tento argument určuje lokální adresu CSP. Tyto adresy musí být různé. Do programu client je třeba vložit ještě 1 nebo 2 další argumenty. Jedním z nich je -r<celé číslo>, do kterého vložíme adresu CSP serveru. Druhý argument se buď vůbec nevkládá, nebo může být -t testovací režim, nebo -m režim zpráv.

Nejprve se zapne server. Pokud je spuštění úspěšné, zobrazí se:

Listing 3.1: Příklad inicializace programu server

```
$ server -k/dev/ttyUSB0 -a1
Initialising CSP
INIT KISS: device: [/dev/ttyUSB0], bitrate: 115200
Connection table
[00 0x14fc560] S:0, 0 -> 0, 0 -> 0, sock: (nil)
Interfaces
LOOP       tx: 00000 rx: 00000 txe: 00000 rxe: 00000
           drop: 00000 autherr: 00000 frame: 00000
           txb: 0 (0.0B) rxb: 0 (0.0B) MTU: 0
KISS       tx: 00000 rx: 00000 txe: 00000 rxe: 00000
           drop: 00000 autherr: 00000 frame: 00000
           txb: 0 (0.0B) rxb: 0 (0.0B) MTU: 252
Route table
1/5 LOOP
0/0 KISS
Server task started
Binding socket 0x55af30cdc338 to port 25
```

Po zapnutí programu client:

Listing 3.2: Příklad inicializace programu client

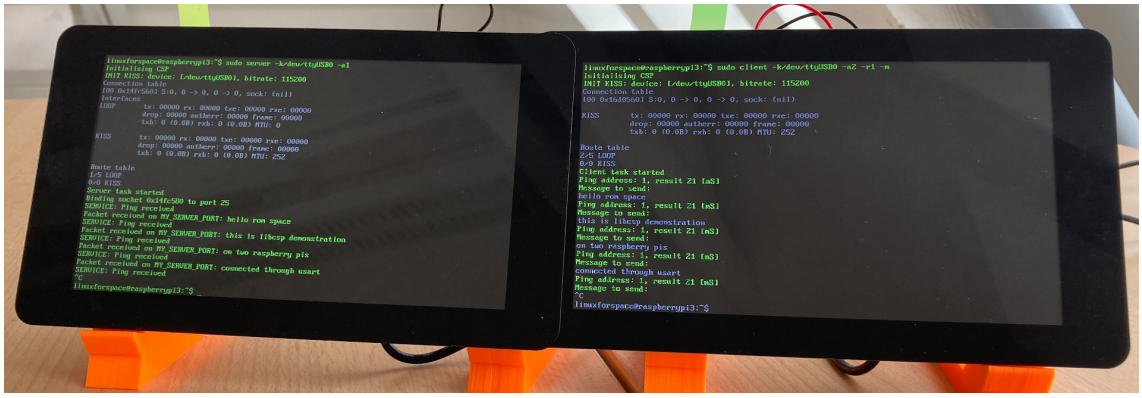
```
$ client -k/dev/ttyUSB0 -a2 -r1 -t
Initialising CSP
INIT KISS: device: [/dev/ttyUSB0], bitrate: 115200
Connection table
[00 0x16d0560] S:0, 0 -> 0, 0 -> 0, sock: (nil)
KISS       tx: 00000 rx: 00000 txe: 00000 rxe: 00000
           drop: 00000 autherr: 00000 frame: 00000
           txb: 0 (0.0B) rxb: 0 (0.0B) MTU: 252
Route table
2/5 LOOP
0/0 KISS
CLient task started
```

Program client pak začne odesílat pakety na server. Pokud je povolen testovací režim, odešle se přibližně 13 paketů a klientský program se ukončí. Pokud je povolen režim zpráv, odešle se paket ping a vyžádá se předání zprávy.

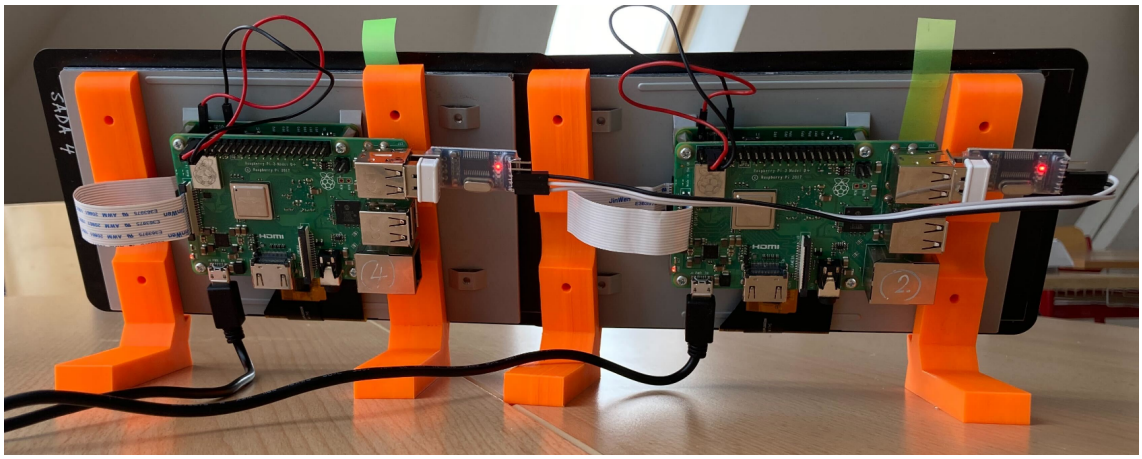
Přeposlání paketu prostřednictvím převodníku USB-UART trvá přibližně 21-22 milisekund.

Listing 3.3: Poslání paketu ping na adresu 1

```
Ping address: 1, result 21 [mS]
```



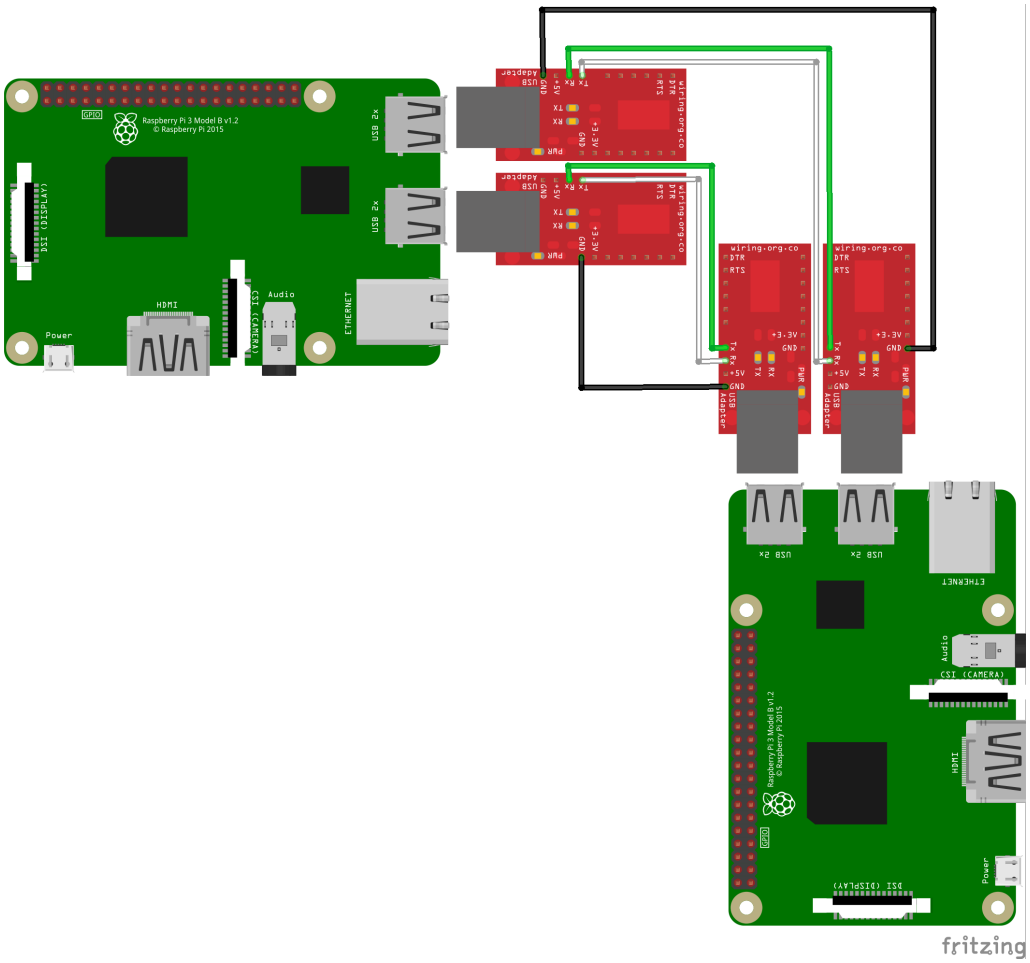
Obrázek 3.3: Ukázka komunikace mezi dvěma Raspberry Pi, pohled zepředu



Obrázek 3.4: Ukázka komunikace mezi dvěma Raspberry Pi, pohled zezadu

3.4.3 Obousměrná komunikace

Obousměrná komunikace vyžaduje 2 další převodníky USB-UART:



Obrázek 3.5: Diagram obousměrné komunikace mezi dvěma Raspberry Pi

Na obou deskách Raspberry Pi je spuštěn program server a jsou jim přiřazeny různé lokální adresy CSP a adresy portů USB, ke kterým jsou klíče připojeny.

Listing 3.4: Nastavení programu server na 1. Raspberry Pi

```
$ server -k/dev/ttyUSB0 -a1
```

Listing 3.5: Nastavení programu server na 2. Raspberry Pi

```
$ server -k/dev/ttyUSB1 -a2
```

Po spuštění programy server pozastavíte pomocí klávesové zkratky CTRL+Z. Poté se znovu spustí, ale na pozadí, pomocí příkazu bg. Na obou deskách je spuštěn také klientský program. Jako argumenty se vloží adresy USB klíče a místní adresa CSP druhé desky RPI.

Listing 3.6: Nastavení programu client na 1. Raspberry Pi

```
$ client -k/dev/ttyUSB0 -a3 -r1
```

Listing 3.7: Nastavení programu client na 2. Raspberry Pi

```
$ client -k/dev/ttyUSB1 -a4 -r2
```

Když je program client zapnutý, probíhá obousměrná komunikace, kdy si desky navzájem vyměňují pakety informací. Vlákna serveru a klienta běží odděleně na pozadí nezávisle na sobě.

3.4.4 Výsledky práce

Výsledek projektu byl představen komunitě Linux4Space dne 10. 04. 2024. V prezentaci byla vysvětlena struktura receptů, důležité konfigurace a modifikovatelné proměnné v přídavném souboru pro recept a funkce přidané do demonstračního kódu knihovny. Bylo také promítnuto video a fotografie z ukázky hardwaru a vysvětlení toku funkcí demonstračního kódu. Ke konci byl prostor pro dotazy a připomínky k projektu s několika návrhy na rozšíření receptů. Demonstrační kódy a recepty jsou umístěny v GitHub repozitáři, který je k dispozici na adrese url https://github.com/Tach3/bachelor-s, a v GitLab repozitáři Linux4Space.

4 Závěr

Cílem práce bylo navrhnout a napsat recept pro začlenění protokolu CSP do vrstvy meta-Linux4Space pro projekt Linux4Space a následně demonstrovat výsledek na ukázkovém hardwaru. Řešení mělo použít existující knihovnu libcsp..

Všechny cíle práce byly splněny, práce popisuje proces od začátku do konce. Byly vytvořeny dva recepty, dva soubory, které recepty rozšiřují, několik konfiguračních souborů a soubory s kódem napsaným v jazyce C. Jeden recept slouží k začlenění samotné knihovny do vlastního operačního systému, druhý slouží k vložení ukázkových kódů a programů do systému. Recepty byly napsány tak, aby je bylo možné snadno zařadit do referenční distribuce a aby je bylo možné upravit podle potřeb budoucích uživatelů. V rámci demonstrace byly napsány a upraveny programy v jazyce C, které byly následně testovány na operačním systému sestaveném z referenční distribuce Poky. Celá demonstrace byla provedena na dvou deskách Raspberry Pi připojených přes USB-UART převodníky.

Práce byla vytvořena s ohledem na budoucí využití v projektu Linux4Space a případné nasazení na oběžné dráze. Projekt je realizován ve spolupráci TUL, ČVUT, VZLÚ a mezinárodního týmu dobrovolníků.

Práce používa verzi libcsp-1.6 a ne verzi 2.0. Verze knihovny libcsp-2.0 byla vydána dne 19. dubna 2024, což jsou téměř 4 roky od poslední verze 1.6. Z časových důvodů nebylo možné integrovat verzi 2.0 a bylo by vhodné na ni eventuálně ukázkový kód a recept upravit. Verze 2.0 totiž obsahuje nové funkce a způsoby odesílání paketů a není kompatibilní s verzí 1.6. Práci lze dále rozšířit, zejména v oblasti využití dostupného ovladače ZeroMQ k vytvoření proxy serveru ZMQ a jeho použití k řízení několika distribuovaných systémů.

Použitá literatura

[1] BATTISTA, Umberto et al. Design of net ejector for space debris capturing. In: 2017. Dostupné také z: https://conference.sdo.esoc.esa.int/proceedings/sdc7/paper/279.

[2] BEECH, William A., Douglas E. NIELSEN a Jack TAYLOR. *AX.25 Link Access Protocol for Amateur Packet Radio*. Tucson Amateur Packet Radio Corporation, 1998. Dostupné také z: https://www.tapr.org/pdf/AX25.2.2.pdf.

[3] CANTILLO, Juan, Jérôme LACAN a Buret ISABELLE. A CRC Usefulness Assessment for Adaptation Layers in Satellite Systems. 2006. Dostupné z DOI: https://arc.aiaa.org/doi/10.2514/6.2006-5358.

[4] CCSDS. *Space Packet Protocols, CCSDS 130.3-G-1*. 2023. Tech. zpr. CCSDS. Dostupné také z: https://public.ccsds.org/Pubs/130x3g1.pdf.

[5] DAVOLI, Franco et al. Small satellites and CubeSats: Survey of structures, architectures, and protocols. *International Journal of Satellite Communications and Networking*. 2019, roč. 37, č. 4, s. 343–359. Dostupné z DOI: https://doi.org/10.1002/sat.1277.

[6] CHALLA, Obulapathi N., Gokul BHAT a Janise MCNAIR. CubeSec and GndSec: A Lightweight Security Solution for CubeSat Communications. In: 2012. Dostupné také z: https://api.semanticscholar.org/CorpusID:52993240.

[7] CHIN, Jamie et al. *CubeSat 101: Basic Concepts and Processes for First-Time CubeSat Developers*. NASA CubeSat Launch Initiative, 2017. Dostupné také z: https://www.nasa.gov/wp-content/uploads/2017/03/nasa_csli_cubesat_101_508.pdf.

[8] CHRISTIANSEN, Johan De Claville. *CubeSat Space Protocol(CSP) - Network-Layer delivery protocol for CubeSats and embedded systems*. GomSpace, 2011. Dostupné také z: https://bytebucket.org/bbruner0/albertasat-on-board-computer/wiki/1.%20Resources/1.1.%20DataSheets/CSP/GS-CSP-1.1.pdf?rev=316ebd49bed49fdbb1d74efdeab74430e7cc726a.

[9] CHRISTIANSEN, Johan De Claville. *Libcsp GitHub* [online]. 2024. [cit. 2024-05-06]. Dostupné z: https://github.com/libcsp/libcsp.

[10] KOSKOVÁ TŘÍSKOVÁ, Lenka a Lukáš MÁZL. *Linux4Space requirements* [online]. 2024. [cit. 2024-05-06]. Dostupné z: https://gitlab.com/linux4space/linux4space_requirements.

[11] KOSKOVÁ TŘÍSKOVÁ, Lenka a Lukáš MÁZL. *Linux4Space webpage* [online]. 2024. [cit. 2024-05-06]. Dostupné z: https://linux4space.org/.

[12] KULU, Erik. *Nanosats Database* [online]. 2024. [cit. 2024-05-06]. Dostupné z: https://www.nanosats.eu/.

[13] LAUENSTEIN, J.-M. Single-Event Gate Rupture in Power MOSFETs: A New Radiation Hardness Assurance Approach. 2011. Dostupné také z: https://ntrs.nasa.gov/api/citations/20110011911/downloads/20110011911.pdf.

[14] LEONARD, Chris. *Challenges for Electronic Circuits in Space Applications* [online]. 2017. [cit. 2024-05-06]. Dostupné z: https://www.analog.com/en/signals/thought-leadership/challenges-for-electronic-circuits-in-space-applications.html.

[15] NASA. *Miniature Satellites with Massive Benefits* [online]. NASA, 2022 [cit. 2024-05-06]. Dostupné z: https://www.nasa.gov/missions/station/miniature-satellites-with-massive-benefits/.

[16] NWANKWO, Victor, N. JIBIRI a Michael KIO. The Impact of Space Radiation Environment on Satellites Operation in Near-Earth Space. In: 2020. ISBN 978-1-78985-996-6. Dostupné z DOI: 10.5772/intechopen.90115.

[17] PAIGER, Karel. *Obrázek českého nanosatelitu VZLUSat-1.* VZLÚ - Výzkumný a zkušební letecký ústav, a.s., 2017. Dostupné také z: https://cs.wikipedia.org/wiki/VZLUSAT-1#/media/Soubor:VZLUSat-1.jpg. Tato fotografie spadá pod licenci CC-BY-SA-2.0. Chcete-li si prohlédnout kopii této licence, navštivte https://creativecommons.org/licenses/by-sa/2.0/.

[18] PAIVA, David et al. Enhanced software development process for CubeSats to cope with space radiation faults. In: 2022. Dostupné z DOI: 10.1109/PRDC55274.2022.00022.

[19] *SpaceWire Handbook*. 4Links Limited, 2012. Dostupné také z: http://spacewire.esa.int/WG/SpaceWire/SpW-WG-Mtg20-Proceedings/4Links-SpaceWire-Handbook-Draft-20130410.pdf.

[20] *State-of-the-Art Small Spacecraft Technology*. 2021. Tech. zpr. NASA - Small Spacecraft Systems Virtual Institute. Dostupné také z: https://www.nasa.gov/wp-content/uploads/2020/06/soa_2021.pdf?emrc=d8d41e.

[21] YAMAZAKI, Masahiko. *Introduction to CubeSat Payload System* [online]. KiboCUBE Academy, 2023 [cit. 2024-05-06]. Dostupné z: https://www.unoosa.org/documents/pdf/psa/access2space4all/KiboCUBE/AcademySeason2/On-demand_Pre-recorded_Lectures/KiboCUBE_Academy_2022_OPL18.pdf.

[22] *Yocto Procejt Documentation* [online]. 2024. [cit. 2024-05-06]. Dostupné z: https://docs.yoctoproject.org/.