České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra počítačů

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Paulík Adam

Studijní program: Otevřená informatika
Obor: Softwarové systémy

Název tématu: Vývoj IoT aplikačního běhového prostředí

Pokyny pro vypracování:

Cílem této práce je vytvořit IoT běhové prostředí pro aplikace třetích stran na Single Board Computer jako je například Raspberry Pi, seznámit se s Single Board Computer s vývojem aplikací, které interagují s vnějším světem. Dále se student naučí vývoj Docker kontejnerů a koncept IoT.

Práce bude obsahovat analýzu současných řešení, analýzu požadavků, návrh a implementaci systému. Systém bude otestován na case study či příkladu.
Požadavky:

1) Běhové prostředí poběží jako Docker kontejner a bude využívat Bulldog knihovnu.
2) Prostředí bude poskytovat komunikační kanál, který budou ostatní aplikace využívat, platformě i jazykově nezávisle.
3) Projekt bude v Github repozitáři.
4) Student musí prozkoumat možnosti běhu v privilegiovaném módu pro přímou komunikaci s pamětí zařízení.

Seznam odborné literatury:

[1] NICKOLOFF, Jeff a Deepak VOHRA. Docker in Action. Manning Publications, 2016. ISBN 1633430235
[2] MOLLOY, Derek. Exploring raspberry PI. Indianapolis, IN: John Wiley and Sons, 2016. ISBN 978-111-9188-681

Vedoucí: Ing. Tomáš Černý, Ph.D.

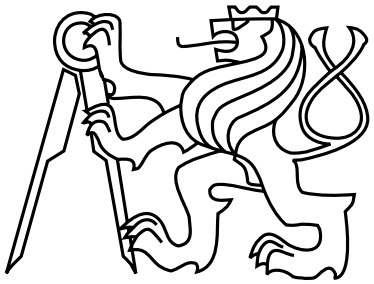
Platnost zadání do konce letního semestru 2017/2018

prof. Dr. Michal Pěchouček, MSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 12.1.2017

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

Vývoj IoT aplikačního běhového prostředí

Adam Paulík

Vedoucí práce: Ing. Tomáš Černý, Ph.D.

Studijní program: Otevřená informatika, Bakalářský

Obor: Softwarové systémy

25. května 2017

Poděkování

Rád bych tímto poděkoval svému vedoucímu a kantorovi mnoha předmětů Ing. Tomáši Černému, Ph.D. za obohacení ze světa moderních softwarových technologií. Dále bych rád poděkoval svému technickému vedoucímu Mgr. Štefanu Bunčiakovi za užitečné rady během zpracování bakalářské práce. Nakonec bych rád poděkoval své rodině za psychickou podporu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 29. 5. 2017 .....

Abstract

This thesis aims to develop platform agnostic IoT runtime using Bulldog Camel and Docker components. Bulldog and Camel components will help to create platform and language agnostic runtime for single board computer (e.g. Raspberry Pi, CubieBoard, BeagleBone Black). Docker technology will be used for easy deployment. This combination of components aims to reduce complexity and bring portability to IoT world.

Keywords: Internet of Things, Runtime, Single-board computer

Abstrakt

Tato práce je zaměřená na vývoj platformě nezávislého IoT běhového prostředí pomocí komponent Bulldog, Camel, Docker. Pomocí Bulldog a Camel komponent jsem řešil platformní a jazykovou nezávislost pro vývoj aplikací pro Single Board Computer např. Raspberry Pi, CubieBoard, BeagleBone Black. Jednoduché nasazení prostředí je řešeno pomocí Docker. Toto spojení komponent si klade za cíl ulehčit vývoj a poskytnout portabilitu aplikací ve světě IoT.

Klíčová slova: internet věcí, běhové prostředí, jednočipový počítač

Obsah

Seznam obrázků

Seznam tabulek

Seznam kódů

Kapitola 1

Úvod

V současné době je velikým trendem internet věcí [10], tato idea se zabývá propojením zařízení do jedné velké sítě pro možnost následovné analýzy nebo analýzy v reálném čase nad daty, která vyprodukují tato zařízení. Při sběru těchto dat nastává problém při veliké rozmanitosti těchto zařízení. Tato zařízení většinou podporují GPIO, I2C, SPI rozhraní. Pro běh jedné aplikace nad několika různými zařízení musí programátor refaktorovat jeho kód pro jiné použité knihovny, nebo měnit pojmenování konstant nastavených v knihovně, která řeší přístup k více zařízení. Možné řešení pro tento problém se správou více duplicitních kódů pro více zařízení, by mohlo řešit běhové prostředí [11], jehož účel by byl odstínit programátora od použití specifických knihoven, napsaných v různých programovacích jazycích a poskytnout mu běhové prostředí s vystaveným komunikačním kanálem. Kód napsaný nad tímto prostředí je možné následně přenášet mezi zařízeními bez refaktorizace nebo přepisování do jiného programovacího jazyka.

1.1 Motivace a cíl této práce

Cílem této práce je navhnout a implementovat běhové prostředí pomocí knihovny Bulldog [13] na Single Board Computer Raspberry Pi, které bude exportovat komunikační kanál přes něhož bude programátor pracovat s periferií Single Board Computer. Hlavní motivací na pozadí je odstranění závislosti programátora na jazyce a i na platformě.

Implementace bude využívat knihovny Bulldog, napsané v Javě, pro práci s interface Single Board Computer. Použití knihovny Bulldog zaručuje komponentální přístup pro práci s Single Board Computer, která aktuálně podporuje RaspberryPi, CubieBoard, BeagleBoneBlack.

1.2 Struktura této práce

Kapitola první prezentuje úvod, motivaci a cíl práce. Kapitola druhá poskytuje informace o IoT, runtime, principy, které se vážou k těmto pojmům. Kapitola třetí představuje aktuální řešení. Ve čtvrté kapitole zanalyzuji možné komunikační protokoly. Kapitola pátá je o sběru požadavků pomocí User stories. Kapitola šestá probírá návrh systému a důvody

pro volby komponent. Kapitola sedmá popisuje implementaci běhového prostředí, pomocí Bulldog knihovny, integračního nástroje Apache Camel a Docker [9], to je nástroj pro izolaci aplikací do kontejnerů. Kapitola osmá popisuje testování. Poslední kapitola otestuje runtime na případu užití.

Kapitola 2

Pozadí

Tato kapitola představuje technické pozadí této práce. Vysvětluje, co to je IoT. Popisuje běžný přístup k vývoji aplikací pro IoT a vývoj pomocí běhového prostředí. V této kapitole jsou vyzdvižené důvody, proč běžný vývoj aplikací pro IoT je těžší na použití a proč vývoj pomocí běhového prostředí poskytuje lepší řešení pro vývoj aplikací.

2.1 Co to je IoT

Internet věcí je síť fyzických zařízení, která je často označována jako chytrá zařízení, budovy a další přístroje připojená na internetové síti umožnující výměnu zpráv. V roce 2013 Global Standards Initiative on Internet of Things (IoT-GSI) definovala IoT jako "infrastrukturu sociální sítě". IoT umožňuje objektům být detekován nebo ovládán vzdáleně přes existující síťovou infrastrukturu vytvářející příležitosti pro více přímou integraci fyzického světa do počítačových systémů mající jako následek zlepšení v efektivitě, přesnosti, snížení finančních nákladů a snížení lidské pracovní síly. V této práci se budu hlavně zabývat přístupu na periferii IoT zařízení [10].

2.2 Co to je SBC

SBC neboli Single-board computer [12] je kompletní malý počítač na jedné desce dispoující procesorem pamětí a vstupními a výstupními rozhraními. Takové desky jsou například Raspberry Pi, Arduino nebo Intel Edison. Jedná se o vložený systém s mnoha výhodamy. Jedna z hlavních výhod je nízká cena a rozšířitelnost systému a proto je voleno SBC v řade projektů od menších až k rozsahu enterprise sféry. Pro komunikaci s vnějším světem většinou používají rozhraní jako GPIO, I2C, SPI, a další.

2.3 Co to je GPIO

Jedná se o obecný pin na integrovaném obvodu. Pin může pracovat v režimu vstupu nebo výstupu. Piny nemají žádné předdefinované chování, většinou jsou nepoužívané v základu. GPIO může být nastaveno jako vstup nebo výstup. Hodnotu při výstupu je možné zapisovat nebo číst. Piny mohou být aktivní či neaktivní. Při pinech v režimu čtení může být z pinů pouze čteno.

2.4 Co to je běhové prostředí

Jedná se o software navržený pro podporu vykonávání operací aplikací napsaný v některém z programovacích jazyků [11]. Runtime obsahuje nízkoúrovňové (práce s hardwarem) operace a někdy vysokoúrovňové (architektonický abstrakt, jako například práce se zařízeními).

Kapitola 3

Současná řešení

V této kapitole zmíním několik knihoven a nástrojů používaných při vývoji na Single Board Computer. Obsahem této kapitoly jsou také ukázky kódu, kde je vidět rozdílnost přístupu k Single Board Computer při použití různých jazyků, které vede k velké náročnosti pro přepisování mezi různými druhy Single Board Computer.

3.1 WiringPi

Wiring Pi [15] je knihovna pro práci s GPIO napsaná v jazyce C pracující na BCM2835 používané v Rasberry Pi a dalších SBC. Knihovna je vydaná pod GNU LGPLv3 licencí a je použitelná z jazyků C, C++, RTB (BASIC) stejně tak v mnoha dalších jazycích s vhodnými wrappery. Knihovna je navržena, aby byla jednoduchá na použití pro vývojáře, kteří používají Arduino "wiring" system.
WiringPi obsahuje aplikaci pro příkazový řádek "gpio", který může být použit pro nastavení GPIO pinů.

```
#include <stdio.h>
#include <wiringPi.h>

// LED Pin - wiringPi pin 0 is BCM_GPIO 17.

#define LED     0

int main (void)
{
  printf ("Raspberry␣Pi␣blink\n") ;

  wiringPiSetup () ;
  pinMode (LED, OUTPUT) ;

  for (;;)
  {
    digitalWrite (LED, HIGH) ;  // On
    delay (500) ;               // mS
    digitalWrite (LED, LOW) ;   // Off
    delay (500) ;
  }
  return 0 ;
}
```

Listing 3.1: Ukázka použití WiringPi

3.2 RPi.GPIO

RPi.GPIO [7, 14] je modul z knihoven jazyka Python pro kontrolu GPIO na Raspberry Pi. Modul není použitelný pro realtime aplikace, protože nelze predikovat, kdy se pustí správa paměti v Pythonu. Dále knihovna běží pod Linuxovým jádrem, které je víceúlohový systém a to může způsobovat zastavování aplikace způsobené přiřazením CPU procesu s vyšší prioritou.

```
import RPi.GPIO as GPIO ## Import GPIO library
import time ## Import 'time' library. Allows us to use 'sleep'

GPIO.setmode(GPIO.BOARD) ## Use board pin numbering
GPIO.setup(7, GPIO.OUT) ## Setup GPIO Pin 7 to OUT

##Define a function named Blink()
def Blink(numTimes,speed):
for i in range(0,numTimes):## Run loop numTimes
print "Iteration " + str(i+1)## Print current loop
GPIO.output(7,True)## Switch on pin 7
time.sleep(speed)## Wait
GPIO.output(7,False)## Switch off pin 7
time.sleep(speed)## Wait
print "Done" ## When loop is complete, print "Done"
GPIO.cleanup()

## Ask user for total number of blinks and length of each blink
iterations = raw_input("Enter total number of times to blink: ")
speed = raw_input("Enter length of each blink(seconds): ")

## Start Blink() function. Convert user input from strings to numeric data types
     and pass to Blink() as parameters
Blink(int(iterations),float(speed))
```

Listing 3.2: Ukázka použití RPi.GPIO

3.3 Bulldog

Bulldog [13] je Java knihovna využívající Java Native Interface, aby poskytla GPIO a low-level IO možnosti embedded linux platform (RaspberryPi, CubieBoard, BeagleBoneBlack). Jako jedna z mála knihoven přistupuje přímo do paměti a proto je daleko rychlejší oproti jiným, např. oproti Java knihovně pro práci s GPIO Pi4J. Dále podporuje většinu běžných sběrnicí vyskytujících se na SBC.

```
import io.silverspoon.bulldog.core.gpio.DigitalOutput;
import io.silverspoon.bulldog.core.platform.Board;
import io.silverspoon.bulldog.core.platform.Platform;
import io.silverspoon.bulldog.core.util.BulldogUtil;
import io.silverspoon.bulldog.raspberrypi.RaspiNames;

public class BulldogLED {

  public static void main(String[] args) {
    //Detect the board we are running on
    Board board = Platform.createBoard();

    //Set up a digital output
    DigitalOutput output = board.getPin(RaspiNames.P1_11).as(DigitalOutput.class
        );

    // Blink the LED
    output.high();
    BulldogUtil.sleepMs(1000);
    output.low();
    }
}
```

Listing 3.3: Ukázka použití Bulldog

3.4 Další řešení

Jedno další řešení by bylo nastudovat si specifikaci mapování paměti na filesystém v linuxu, ale toto řešení je velice pomalé oproti knihovnám, které přistupují přímo do paměti. Pro Single Board Computer existují veliká množství knihoven v různých jazycích, pro každé Single Board Computer má tato knihovna většinou jiné názvy volaných funkcí a jiné pojmenování pinů. Obvykle podporují pouze GPIO rozhraní a nepodporují další jako I2C, SPI, UART atd..

Kapitola 4

Analýza požadavků

V této kapitole se budu zabývat analýzou požadavků na běhové prostředí. Pro tento úkol jsem použil moderní přístup nazývaný "User stories" [5].

"User stories" jsou částí agilního přístupu, které pomáhají přesunout pozornost z psaní o požadavcích k mluvení o nich. Všechny agilní "user stories" obsahují psanou větu nebo dvě, více významně sérii konverzací o chtěné funkcionalitě.

4.1 Obecné "User stories"

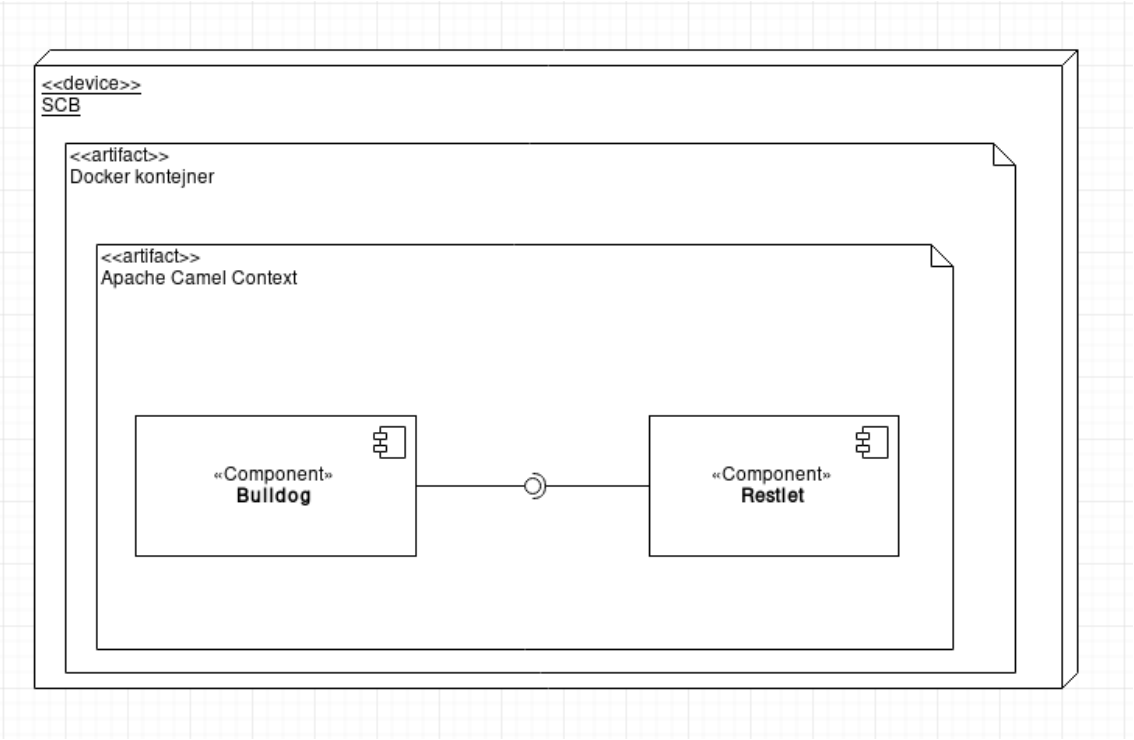
- Jako uživatel chci, aby bylo runtime jednoduché nainstalovat, proto volím Docker, který mi dovoluje všechny závislosti svázat jako samostatnou aplikaci.
- Jako uživatel chci, aby běžel runtime jako servis, jedno spuštění Java VM, jehož spuštění trvá dlouhou dobu.
- Jako uživatel chci, aby bylo možné vytvořit jednoduché obvody a pouštět proud do jednotlivých GPIO podle potřeby.
- Jako uživatel chci, mít možnost připojit zařízení (senzor) a dotázat se nebo zapsat data (pokud to runtime podporuje).
- Jako uživatel chci, mít jednoduché rozhraní/komunikační kanál, který mi dovoluje jednoduše manipulovat s daty, proto volím JSON.

4.2 GPIO "User stories"

- Jako uživatel chci, aby pojmenování pinu bylo jednotné. Aplikace napsaná pro runtime funguje na více druzích SBC. (GPIO <1, N>, SPI, I2C ... jen zapojím podle schématu a pustím stejnou app).
- Jako uživatel chci zjistit stav GPIO pinu (všechny pokud nespecifikováno v dotazu).
- Jako uživatel chci nastavit pin GPIO, z důvodu běhu více aplikací a sdílení pinů musím mít možnost zjistit aktuální stav pinu.

4.3 High level system design

Celá aplikace se bude nasazovat jako Docker kontejner na SBC. Uvnitř kontejneru poběží Camel Context nad Java VM. Do Camel contextu budou nahrané definice cest komponenty, která vytvoří komunikační kanál s vnějším světem a Bulldog komponenty pro komunikaci s SBC. Schéma popisující nasazení, je zobrazeno v obrázku 4.1.



Obrázek 4.1: Schéma popisující nasazení runtime

Kapitola 5

Analýza komunikačních protokolů

5.1 Úvod

Před samotnou implementací běhového prostředí provedu analýzu několika vybraných protokolů. Analýza bude obsahovat přečtení specifikace protokolu a provedení měření nad několika případy užití. Během analýzy budu sledovat celková přenesená data, přenesená data na jednu operaci popsanou v případech užití, průměrnou dobu jedné operace a přívětivost pro vývojáře. Data byla naměřena pomocí Wireshark v Capturing módu.

Použité SBC pro tento účel je Raspberry Pi 3 B s linuxovou distribucí Raspbian ve verzi 2017-03-02, dále Wireshark 2.2.4, Java 1.8 (u65), Camel 2.18.3, Bulldog 0.21. Pro běh JBoss AM-Q 6.3 byl použit notebook Lenovo ThinkPad x230.

5.2 Sledované případy užití

5.2.1 První případ užití

První případ užití, který budu sledovat je nastavení GPIO pinu na hodnotu HIGH/LOW, kde vnější projev bude rozsvícení nebo zhasnutí LED. Tento případ užití se dá považovat za jeden z nejzákladnějších, kdy se pouze uzavírá elektrický obvod. Během testování budu posílat 1200 dotazů s odstupem mezi jednotlivými operacemi 3 sekundy. Naměřené hodnoty pro vybrané protokoly jsou zapsané v tabulce 5.1.

5.2.2 Druhý případ užití

Druhý případ užití ve světě IoT nejvíce se vyskytující, je určitě přečtení hodnoty ze senzoru. Pro tento účel jsem si zvolil senzor bh1750, který měří světelnost. Během testování budu posílat 1200 dotazů s odstupem mezi jednotlivými operacemi 3 sekundy. Naměřené hodnoty jsou zapsané v tabulce 5.2.

5.3 TCP/IP socket

TCP/IP je základní protokol Internetu [4]. TCP/IP je dvouvrstevnatý protokol. Vyšší vrstva TCP spravuje rozložení zprávy na menší pakety přenášené přes Internet, následně složené na původní zprávu pomocí TCP vrstvy. IP vrstva zajišťuje adresní část a doručení packetů na správný cíl. TCP/IP používá klient/server komunikační model. TCP/IP komunikace je primárně "point-to-point", to znamená, že každá komunikace je z jednoho bodu a na další bod v síti nebo samostatným pc. TCP/IP je označovaný jako bezstavový protokol, protože při každém novém spojení se neuvádí v úvahu předchozí spojení. TCP/IP používá mnoho dalších vyšších protokolů jako jsou například Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), Telnet (Telnet), Simple Mail Transfer Protocol (SMTP) a další.

5.3.1 Výhody

- obousměrná komunikace
- binární
- pro více zpráv jedno spojení

5.3.2 Nevýhody

- ne vždy se přenesou všechny bajty (podle stavu sítě)
- pro více hodnot jedno spojení

5.3.3 Klientská část

Jedná se o aplikaci napsanou v jazyce Java, která pomocí jednoho spojení vykonává dotazy podle případu užití.

5.3.4 Serverová část

Serverová část je napsaná v jazyce Java využívající Apache Camel komponentu Netty pro práci s TCP/IP soketem a Bulldog komponentu pro prácí s periferií SBC. Každá zpráva je zakončená odřádkováním a informace je zakódovaná pomocí JSON.

5.4 REST

REST (REpresentational State Transfer) [6] je architektonický návrh, který je často používán k vývoji webových služeb. Použití RESTu je často preferované oproti SOAP, hlavním důvodem je menší použití zdrojů, takže se více hodí pro použití na Internetu. RESTová architekrura ztrácí spojení (low coupling) mezi konzumentem a producentem komunikačního zdroje, což dělá REST velice populární pro cloudová API (Application Programming

Interfaces). REST typicky běží přes HTTP (Hypertext Transfer Protocol) a má tyto omezení/vlastnosti : ztráta spojení (loose coupling) mezi konzumentem a producentem, protokol je bezstavový, možnost použití cache, vrstevnatý systém, Uniform Interface(Methods, Representation). RESTový styl klade důraz na to, že existuje omezený pocet druhů operací. Pružnost rozhraní je zajištěna propojením zdroje s unikátní Universal Resource Identifiers (URIs). Protože každá operace má specifický účel (GET, POST, PUT a DELETE), REST zabraňuje vzniku nejasností.

5.4.1 Výhody

- RESTové služby jsou jednoduše použitelné pomocí velkého množství nástrojů, které jsou zdarma poskytované
- REST je jednoduše škálovatelný
- vývojáři si velice rychle osvojí práci s restovým api, učící křivka je velice strmá
- REST je navržený pro použití na Internetu a pro aplikace škálované na serverových platformách

5.4.2 Nevýhody

- pouze aktive polling

5.4.3 Klientská část

Klientská část pro testování je napsaná ve bash skriptě. Restová volání jsou vykonávané pomocí konzolové aplikace curl. Pro každé restové volání se vytváří nové spojení.

5.4.4 Serverová část

Serverovou část obsluhuje Apache Camel komponenta Restlet.

5.5 MQTT

MQTT (Message Queue Telemetry Transport) [1] je standartizovaný public-subscribe messaging protokol, který funguje nad TCP/IP transportním protokolem. Tento protokol je navržený pro propojení vzdálených zařízení, kde je potřeba mít malý přenos dat. Tento model vyžaduje broker, komponentu zodpovědnou za distribuci zpráv. K této komponentě se připojují klienti, kteří chtějí odebírat určitý druh zpráv, který se rozlišuje pomocí "topic".

Reprezentace integeru je 16 bitů. String je kódovaný UTF-8 (maximální délka 65535 bajtů). Řízení komunikace je pomocí "MQTT Control Packet", skládajících se z 3 částí. Fixed část má 2 bajty a obsahuje informaci o jaký typ contol paketu se jedná, dodatečné flagy pro control packet a zbývající velikost pro variable část packetu a payload. Variable část control packetu obsahuje dodatečné informace, pokud jsou potřeba, payload obsahuje data.

MQTT používá ke komunikaci několik druhů kontrolních paketů, které následně popíši.

5.5.1 Kontrolní pakety MQTT protokolu

- CONNECT: Jedná se o první kontrolní paket, který posílá klient po navázání spojení s brokerem a jeho velikost je minimálně 12 bajtů. Broker na tento kontrolní paket odpovídá s kontrolním paketem CONNACK o velikosti 4 bajty.
- PUBLISH: Tento kontrolní paket posílá klient s obsahem zprávy a následně je tento kontolní paket použit k distribuci zpráv ke klientů, kteří o tuto zprávu mají zájem. Velikost množství přenesených dat se liší podle nastaveného parametru QoS (Quality of service). Tento parametr určuje potvrzovací mechanizmus, který zaručuje tři možnosti doručení zprávy: nejvýše jednou, alespoň jednou, přesně jednou. Tato zpráva má velikost nejméně 4 bajty bez obsahu, který musí být přítomen.
- SUBSCRIBE: Kontrolní paket posílá klient s žádostí o odebírání jeden nebo více topiců. Velikost kontrolního paketu je promněnlivá podle počtu topiců, ale je minimálně 4 bajty veliká. Na tento paket odpovídá server zprávou promněnlivé délky s informací o úspěšnosti čí neúspěšnosti odebírání zprávy.
- UNSUBSCRIBE: Klient posílá žádost o možnost zrušení odebírání topiců. Velikost control packetu je proměnlivý podle počtu topiců. Tato zpráva je minimálně 4 bajty veliká. Broker odpovídá pouze s potvrzením, že žádost přijal a vykonal odebrání všech topiců, které splňují zadané filtry.
- PING: Klient posíla tento kontrolní paket na broker, aby se ujistil, že server je zdravý a že klient má zájem pokračovat v udržování komunikace. Broker odpovídá s potvrzením zprávy. Tento kontrolní paket i odpověď od brokeru mají velikost 4 bajty.
- DISCONNECT: Klient posílá kontrolní paket 2 bajty a následně uzavírá spojení klient nebo server.

5.5.2 Výhody

- obousměrná komunikace
- nízký protocol overhead

5.5.3 Nevýhody

- zprávy jsou one to many

5.5.4 Klientská část

Jedná se o aplikaci napsanou v Javě. Aplikace se připojuje k MQTT brokeru pomocí knihovny Java PAHO od Eclipse [2]. Při prvním případu užití posílá zprávy na topic "/pin1"a nevytváří žádný odběr. Při druhém případu užití posílá práznou zprávu na topic "/request/bp1750/readLuminanceNormalized"a přijímá přečtená data z topicu "/response/bp1750/readLuminanceNormalized". Obě komunikace probíhají s QoS, při kterém se přepošle přesně jedna zpráva.

Protokol	Celkem přeneseno dat v bajtech	Doba operace v milisekundách	Počet přenesených dat na jednu operaci
TCP soket	280252	0.36	234
REST	1449973	86.15	1208
MQTT	858427	41.77	715

Tabulka 5.1: První případ užití

5.5.5 Serverová část

Serverová část je napsaná v jazyce Java využívající MQTT komponentu knihovny Apache Camel. Při prvním připadu užití pouze odebírá zprávy z topicu "/pin1"a vykonává operace pomocí knihovny Bulldog. Při druhém případu užití odebírá zprávy z topicu "/request/bp1750/readLuminanceNormalized"a vykonání operace odpovídá odesláním zprávy na topic "/response/bp1750/readLuminanceNormalized".

5.6 STOMP

STOMP(Simple (nebo Streaming) Text Orientated Messaging Protocol) [3] je velice jednoduchý textový protokol podobný HTTP, který vyžaduje, aby se klient připojil k brokeru.

STOMP je rámcový protokol, inspirovaný HTTP. Rámec obsahuje příkaz, množinu nepovinných hlaviček a nepovinné tělo. STOMP je založený na textu, ale dokáže také přenášet binární soubory. Defaultní encoding pro STOMP je UTF-8, ale podporuje alternativní encoding pro tělo zprávy.

STOMP server je modelován jako množina cílů, kam je možné poslat zprávu. STOMP protokol se chová k cílům jako k obyčejným stringům a jejich syntax je plně na implementaci serveru. Dále STOMP nedefinuje chování cílů, podle kterých by se měla doručit zpráva. Zpráva nebo výměna zprávy nad cílem se může měnit podle serveru nebo dokonce podle adresy cíle. Toto dovoluje vytvořit veliké množství různých serverů, které podporují STOMP protokol. STOMP klient je aplikace, která může být ve dvou módech:

- jako producent, který posílá zprávy na cíl na server pomocí rámce SEND
- jako konzument, který pomocí rámce SUBSCRIBE začne odebírat specifický cíl v rámci MESSAGE

Hlavní filosofií designu STOMPu je jednoduchost a možnost zaměnovat poskytovatele STOMP serverů. STOMP je navržen jako lehký protokol jednoduchý k implementaci pro klienta i server nezávisle na programovacím jazyce. Z toho vyplývá zejména to, že neexistuje mnoho omezení na architekturu serveru a mnoho vlastností jako například přístup k cílům a spolehlivost, jsou specifické pro různé implementace. Protokol je v současné verzi v STOMP 1.2. Před použitím STOMP serveru od některého poskytovatele je potřeba přečíst vlastnosti, které nejsou specifikované v protokolu STOMP.

Protokol	**Celkem přeneseno dat v bajtech**	**Doba operace v milisekundách**	**Počet přenesených dat na jednu operaci**
TCP soket	407829	27.27	340
REST	1616402	89.19	1347
MQTT	1634497	64.46	1362

Tabulka 5.2: Druhý případ užití

5.7 Závěr

Protokol STOMP jsem plně vyřadil z měření, protože nemá specifikované chování pro cíl zprávy a typ komunikace request/response, queue nebo topic(pub/sub) a při použití některé s implementací STOMP brokeru by byl vývojář omezen pouze na tuto implementaci. TCP soket je velice efektivní, ale vytvoření custom komunikačního protokolu nad TCP soketem, by nebylo výhodné pro vývoj, již existuje tisíce proprietárních protokolů. Rest protokol je velice jednoduchý na naučení, ve zprávě se odkazuje na zdroj dat, kde vidím "paralelu" s zdroji na desce SBC. Výkon je relativně stabilní ve všech testovaných případech užití. Datová a časová náročnost je u Restu největší, tuto časovou náročnost můžu přikládat parsování již definovaného protokolu, ale tento rozdíl není veliký pokud vemu v úvahu druhý případ užití, který považuji za více běžný. MQTT protokol by měl být vyvinutý specificky pro komunikaci mezi zařízeními pro IoT, bohužel ale při práci s běhovým prostředím by uživatel byl závislý na brokeru a při druhém případu užití bylo potřeba přenést více dat než u restu a proto vybírám pro běhové prostředí rest.

Kapitola 6

Návrh

V této kapitole popíši návrch běhového prostředí před samotnou implementací.

Z analýzy jsem se rozhodl pro použití RESTu jako komunikační protokol s běhovým prostředím a tedy budu navrhovat REST API pro práci s rozhraním SBC. Toto API jsem popsal pomocí formátu pro popis RESTového API Swagger. Kompletní popis API se nachází v souboru na přiloženém CD. Zde zběžně popíši zvolené cesty a metody.

6.1 Přehled API

- Pro zjištění stavu pinů lze použít GET volání na url: "/gpio". Toto volání vrací JSON obsahující proměnnou pins typu pole, obsahem tohoto pole jsou primitiva typu boolean, informujicí o stavu pinu.

- Pro zjištění stavu pinu lze použít GET volání na url: "/gpio/{id}", kde "{id}" je path parametr, který udává číslo pinu dostupného na desce.

- Pro změnu stavu pinu pomocí POST volání je vystavená url: "/gpio/{id}", kde "{id}" je path parametr, který udává číslo pinu dostupného na desce a jako tělo zprávy se posílá JSON ve tvaru {state:true} nebo {state:false}.

- Pro aplikace využívající světelnost bude v runtime vystavena url "/device/bh1750/readLuminanceNo
  Tato adresa vrací JSON obsahující informaci o světelnosti v rozmezí od nuly po jedna.

URL	**Metoda**	**Přijímá**	**Vrací**
/gpio	GET	-	JSON
/gpio/{id}	GET	-	JSON
/gpio/{id}	POST	JSON	-
/device/bh1750/readLuminanceNormalized	GET	-	JSON

Tabulka 6.1: Přehled REST API

```
swagger: "2.0"
info:
  title: "IoT runtim for SBC"
  description: ""
  version: 1.0.0

host: localhost
basePath: /
schemes:
  - http

tags:
  - name: "gpio"
    description: ""
  - name: "device"
    description: ""

paths:
  /gpio:
    get:
      tags:
      - "gpio"
      summary: "Returns states of GPIO pins"
      description: ""
      produces:
        - application/json
      responses:
        200:
          description: "Array of states of pins."
          schema:
            type: object
            properties:
              pins:
                type: array
                example: [true, false, false, true]
```

Listing 6.1: Ukázka popisu REST API pomocí Swagger

Kapitola 7

Implementace

V této kapitole se budu zabívat implementační částí práce. Vysvětlím výběr použitých technologií. Dále popíši řešení pro běh Docker kontejneru v privilegiovaném módu.

7.1 Camel

Apache Camel [8] je univerzální integrační framework založený na známých Enterprise integračních návrhových vzorech. Apache Camel umožňuje definovat cesty zpráv a práci nad nimi pomocí integračních vzorů v specifickém doménovém jazyce, obsahující Java DSL, Spring DSL, Blueprint DSL. Tento nástroj poskytuje elegantní spojení vystaveného komunikačního kanálu se samotnou periferií při použití Bulldog Camel komponenty.

7.2 Docker

Docker [9] je kontejnerová platforma, která zajišťuje aplikaci nezávislost na Linuxové distribuci. Distribuce pouze poskytuje některé systémové prostředky (jako například systémové volání). V základním režimu je kontejner(aplikace) odstíněna od všech nezbytných prostředků jako je samotný filesystem linuxové distribuce, informace o běžících procesech, atd..

7.2.1 Co to je kontejner

Kontejner je běžící izolovaná aplikace. Oproti virtuálním strojům kontejner neobsahuje operační systém, ale pouze knihovny, soubory a nastavení potřebná k běhu zabalené aplikace [9]. To dělá kontejner velice efektivní a nenáročný na systémové zdroje. Dále je zaručeno, že aplikace poběží stejným způsobem bez závislosti na linuxové distribuci.

7.3 Instalace Raspbianu a potřebných nástrojů

Během této práce jsem používal Linuxou distribuci Rasbian ve verzi 2017-03-02 (http://downloads.raspberrypi.org/raspbian/images/raspbian-2017-03-03/). Image je potřeba

uložit na Micro SD kartu například pomocí příkazu "dd bs=4M if=/path/to/2017-03-02-raspbian-jessie.img of=dev/mmcblk0 status=progress && sync". Popis instalace je podrobně vysvětlen na oficiálních stránkách Raspberry Pi (https://www.raspberrypi.org/documentation/installation/installing-images/). Dále je potřeba provést update systému pomocí příkazu "sudo apt-get install update". V neposlední řadě je potřeba nainstalovat Maven příkazem "sudo apt-get install maven", tento nástroj poskytuje projektu jednoduchou správu závislostí.

7.4 Popis a vysvětlení

Pro implementaci runtime jsem se rozhodl pro použití Camel integračního frameworku. Inicializace Camel contextu je zobrazená v ukázce 7.1. Tento framework obsahuje mnoho komponent pro práci se zprávami a knihovna Bulldog poskytuje komponentu pro tento framework a proto mi poskytne jednoduché spojení REST na GPIO piny. Toto spojení je vidět v ukázce kódu 7.2 z běhového prostředí. Bulldog knihovna obsahuje konzumenta komponentu, ale není vhodná pro můj případ užití a proto jsem použil Bulldog v jeho čisté podobě.

```
CamelContext context = new DefaultCamelContext();
            context.addRoutes(new RaspberryPiBh1750RouteBuilder());
            context.addRoutes(new RaspberryPiGpioRouteBuilder());
            context.start();
```

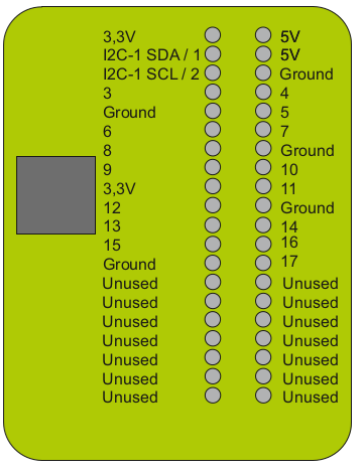
Listing 7.1: Inicializace Camel contextu

```
rest("/").post("/gpio/1").route()
                .choice()
                .when().jsonpath("[?($.state␣==␣true)]").to("bulldog://gpio?pin=
                    P1_3&value=HIGH")
                .process(exchange -> {
                    exchange.getIn().setBody(null);
                    exchange.getOut().setBody(null);
                    exchange.getOut().setHeader(Exchange.HTTP_RESPONSE_CODE,
                        constant(204));
                    exchange.getOut().setHeader(Exchange.HTTP_RESPONSE_TEXT,
                        constant("No␣Content"));
                })
                .when().jsonpath("[?($.state␣==␣false)]").to("bulldog://gpio?pin
                    =P1_3&value=LOW")
                .process(exchange -> {
                    exchange.getIn().setBody(null);
                    exchange.getOut().setBody(null);
                    exchange.getOut().setHeader(Exchange.HTTP_RESPONSE_CODE,
                        constant(204));
                    exchange.getOut().setHeader(Exchange.HTTP_RESPONSE_TEXT,
                        constant("No␣Content"));
                });
```

Listing 7.2: Ukázka spojení RESTu a Bulldogu pomocí Camelu

7.5 Mapování API na SBC

Pro potřebu univerzálnosti GPIO jsem namapoval piny na čísla od 1 po n. V případě Raspberry Pi je podporované rozmezí od 1 po 17. Na toto namapování se pak programátor odkazuje při použití běhového prostředí. Dále je zde zobrazeno mapování i2c rozhraní, pro potřeby využívání světločivného senzoru bh1750, který bude podporovaný v runtime. Jiné SBC by měli trochu jiné mapování, ale pro potřebu běhu aplikace vývojáře pouze zajímá abstrakce, protože zapojení kabeláže není součástí programu a může být převedeno na jiné rozložení pinů při stejné funkcionalitě programu. Schéma zobrazující mapování fyzických pinů na REST API je zobrazeno v obrázku 4.1.



Obrázek 7.1: Schéma popisující mapování API na SBC

7.6 Kontejnerizace pomocí Dockeru

V této kapitole popíši kontejnerizaci runtime pomocí nástroje Docker, kdy v běžném módu není možné pustit docker kontainer, protože nebude mít práva manipulovat s pamětí a proto je třeba vyzkoumat možnosti běhu aplikace v takzvaném privilegiovaném módu.

7.6.1 Privilegiovaný mód

Runtime není možné pustit jako docker kontejner v běžném módu. Při běhu runtime vyžaduje práci s pamětí, ke které potřebuje administrátorská práva. Tato práva můžeme přidat docker kontejneru pomocí přepínače "–privilleged", ale toto řešení přidává zbytečně moc práv programu, který potřebuje pouze pracovat s pamětí a proto je v Dockeru možnost přidávat pouze specifická administrátorská práva přepínačem "–add-cap". Pro potřeby runtime je potřeba nastavit proměnnou SYS_RAWIO, která povolí práci s pamětí, která jí bude přidána s přepínačem "–device". Tímto způsobem bude možné pustit runtime v docker v privilegiovaném módu a poskytnout paměť na ktrerou by neměl práva.

7.6.2 Spuštění runtime

Runtime je možné pustit jako docker kontejner. Pro tento účel je zde připravený soubor Docker file v rootu adresáře projektu. Vytvoření Docker image slouží příkaz "docker build". Runtime je možné zabalit do image pomocí příkazu "docker build . -t iot-runtime". Pro běh docker kontejneru je potřeba přidat možnost využití paměti, na kterou aplikace potřebuje mít administrátorská práva. Tato vlastnost se přidá pomocí přepínačů "–add-cap=SYS_RAWIO –device /dev/mem –device /dev/i2c-1" a celý příkaz pro spuštění runtime jako docker kontejner je tedy "sudo docker run –rm –cap-add SYS_RAWIO –device /dev/mem –device /dev/i2c-1 -it -v /proc/meminfo:/proc/meminfo -p 8080:8080 iot-runtime".

Kapitola 8

Testování

V této kapitole popíši dva testy. První test na případu užití a druhý test, který testuje běhové prostředí pomocí JUnit knihovny, tento test je součástí projektu podle maven konvencí, kde se nachází běhové prostředí.

8.1 Test na příkladu užití pomocí javascriptové aplikace

Funkcionalitu běhového prostředí bych chtěl demonstrovat na jednoduchémpříkladu užití, při kterém budu získávat hodnotu z světlocivného senzoru bh1750 a při snížení světelnosti pod 0.5 rozsvítím led. Senzor bh1750 vrací normalizovanou hodnotu v rozmezí 0 až 1. Příklad je napsaný v jazyce javascript bežící na node.js. Při testu se využívají node.js knihovny sync-request a sleep. Senzor bh1750 zaujímá piny běhového prostředí 1 a 2. Led je připojeno k pinu s číslem 3. Aplikace se nachází v přiloženém CD v souboru "test_priklad_uziti/node.js".

8.2 Test pomocí JUnit

Při předchozích testování na případu užití jsem testoval runtime pouze určité piny a nevyužíval jsem plně funkcionalitu runtime. V tomto testu zkontroluji funkci GPIO pomocí JUnit testování a rozhraní linuxové distribuce pro práci s GPIO, které je vystaveno pomocí filesystému.

8.2.1 Příprava před testem

Před samotným testem je potřeba exportovat pin do filesystému. Tato operace se provádí zapsáním čísla do souboru požadovaného pinu hardwarovým typem číslování. Tento soubor je "/sys/class/gpio/export". Po této operaci se vygeneruje složka "gpio" končící číslovko podle exportovaného pinu. Následně je potřeba nastavit režim pinu pomocí zapsání textu "out" do souboru "direction" v složce exportovaného pinu. V této fázi je připravený exportovaný soubor value obsahující hodnotu 1 nebo 0 podle stavu, ve kterém se nachází a této zkutečnosti využiji při testování. V přípravě testu nastavím všechny piny na hodnotu 0.

8.2.2 Hlavní část testování

V této části testuji funkcionalitu runtime. To znamená nastavování pinů a čtení pinů. Hodnoty nastavuji pomocí runtime a následně je čtu jednak pomocí runtime, ale také pomocí exportovaného pinu do filesystému a tím ověřím správnost všech pinů. Při testování se může vyskytnout chyba, kdy je filesystém zaneprázdněn a ukončí test. Tato chyba neznamená nesprávné chování runtime a při opakovaném provedení testu doběhne test úspěšně.

8.2.3 Úklid po testu

Po testu je dobré odstranit exportované piny do filesystému pomocí zápisu čísla pinu do souboru nacházejícího na adrese "/sys/class/gpio/unexport".

Kapitola 9

Závěr

V této práci bylo navrženo a naimplementováno běhové prostředí pro SBC vystavující REST API jako komunikační kanál, které dává vývojáři volnost při volbě programovacího jazyka. Při této práci bylo použita knihovna Bulldog, pro umožnění platformní nezávislosti.

Běhové prostředí je možné pustit jako Docker kontejner v privilegiovaném módu, kde se aplikaci přiřadí administrátorská práva k periferiím SBC a samotná aplikace běží pod uživatelovi právy, který ji pustí.

Budoucí vývoj běhového prostředí považuji za revoluční myšlenku, kde vývojář píše aplikaci nad abstrakcí SBC interface, kterou může přenášet bez obav o kompatibilitu. Práce se nachází na přiloženém cd ve složce "iot-runtime"a v github repozitáři
https://github.com/kubeIot/iot-runtime.

Literatura

[1] *Dokumentace MQTT* [online]. 2014. [cit. 15. 5. 2017]. Dostupné z: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>.

[2] *Dokumentace Eclipse PAHO Java MQTT klienta* [online]. 2012. [cit. 15. 5. 2017]. Dostupné z: <https://eclipse.org/paho/clients/java/>.

[3] *STOMP Protocol Specification, Version 1.2* [online]. 2012. [cit. 15. 5. 2017]. Dostupné z: <https://stomp.github.io/stomp-specification-1.2.html>.

[4] *Dokumentace TCP/IP* [online]. 1999. [cit. 15. 5. 2017]. Dostupné z: <https://tools.ietf.org/html/rfc2616>.

[5] *User stories* [online]. 2017. [cit. 15. 5. 2017]. Dostupné z: <https://www.mountaingoatsoftware.com/agile/user-stories>.

[6] Alex Rodriguez. *RESTful Web services: The basics* [online]. 2017. [cit. 19. 5. 2017]. Dostupné z: <https://www.ibm.com/developerworks/webservices/library/ws-restful/>.

[7] Dokumentace RPi.GPIO. First GPIO Project. https://pypi.python.org/pypi/RPi.GPIO, stav ze 15. 5. 2017.

[8] IBSEN, C. – ANSTEY, J. *Camel in Action.* Manning Publications, 1st edition, 2011.

[9] NICKOLOFF, J. *Docker in Action.* Manning Publications, 1st edition, 2016.

[10] Přispěvatelé Wikipedie. *Internet of things* [online]. 2017. [cit. 19. 5. 2017]. Dostupné z: <https://en.wikipedia.org/wiki/Internet_of_things>.

[11] Přispěvatelé Wikipedie. *Runtime system* [online]. 2016. [cit. 19. 5. 2017]. Dostupné z: <https://en.wikipedia.org/wiki/Runtime_system>.

[12] Přispěvatelé Wikipedie. *Single-board computer* [online]. 2017. [cit. 19. 5. 2017]. Dostupné z: <https://en.wikipedia.org/wiki/Single-board_computer>.

[13] web:bulldog. Bulldog knihovna. https://github.com/SilverThings/bulldog, stav z 15. 5. 2017.

[14] web:inforpigpio1. First GPIO Project. http://www.thirdeyevis.com/pi-page-2.php, stav ze 15. 5. 2017.

[15] web:infowiringpi. Wiring Pi - GPIO Interface library for the Raspberry Pi. `http://wiringpi.com`, stav ze 15. 5. 2017.

Příloha A

Seznam použitých zkratek

API Aplication Programming Interface

DSL Domain Specific Language

IoT Internet of Things

MQTT Message Queue Telemetry Transport

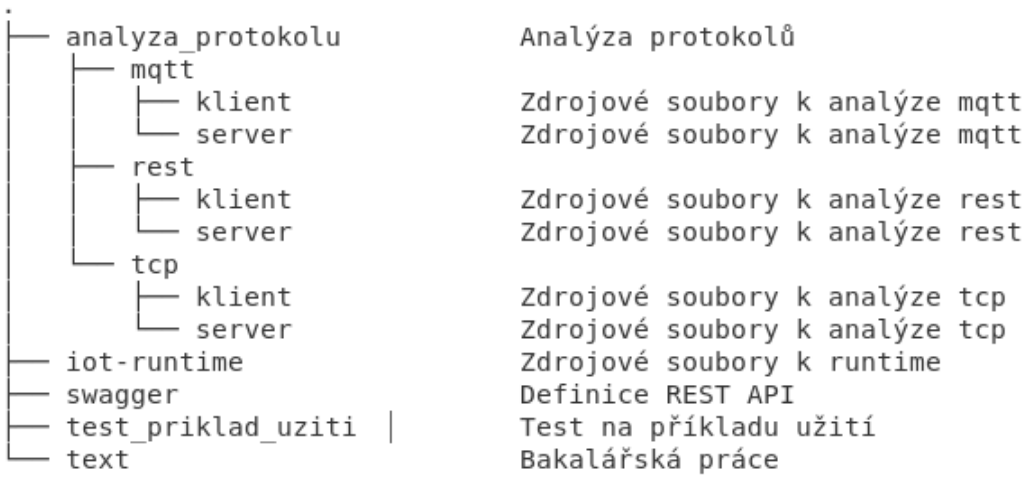
SBC Single Board Computer

STOMP Simple Text Oriented Messaging Protocol

VM Virtual Machine

Příloha B

Obsah přiloženého CD



Obrázek B.1: Seznam přiloženého CD