Herczig Ádám

Szakdolgozat

Hőmérő hálózat

IoT felület

kialakítása

PÉCSI TUDOMÁNY EGYETEM

MŰSZAKI ÉS INFORMATIKAI KAR

VILLAMOSMÉRNÖKI SZAK

Pécs, 2017

Nyilatkozat

Alulírott Herczig Ádám diplomázó hallgató, kijelentem hogy a szakdolgozatomat a Pécsi Tudományegyetem Műszaki és Informatikai karán készítettem a villamosmérnöki BSC diploma megszerzése véget.

Kijelentem, hogy a szakdolgozatom érdemi részét egyedül végeztem el. Az érdemi részen kívül csak meghivatkozott forrásokat használtam fel a szakdolgozatomba (szakirodalom, kód). Tudomásul veszem, hogy a szakdolgozatom leírt forráskódot a Pécsi Tudományegyetem, valamint a szakdolgozat témát kiíró konzulens saját céljaira szabadon felhasználhatja.

2017.10.24.

Herczig Ádám

Tartalomjegyzék helye

1. Bevezetés

A mai hétköznapi életnek lassan már szerves része lesz, hogy mindenféle eszközt rá fogunk tudni csatolni az internet nagy világára. A legáltalánosabb szenzoroktól kezdve, az autóiparban használt robotokon, aktuátorokon át egészen a hétköznapi használatú eszközökig (Televízió, hűtő, riasztóközpont) és még sorolhatnánk tovább is, szinte mindent rá tudunk csatolni az internetre. Ezáltal minden olyan eszköz mely IP címmel rendelkezik, kapcsolatban lehet más egyéb hálózati eszközökkel, amennyiben ez szükséges. Gondoljunk csak bele, megyünk haza munkából, okostelefonunkkal egy pillanat alatt leellenőrizhetjük a hűtőnk tartalmát vagy hogy a lakásunk különböző pontjain milyen hőmérséklet van. Hideg téli estéken akár, ezáltal előre feltudjuk fűteni számunkra megfelelő hőmérsékletűre a lakásunkat hőmérő szenzorok és egy termosztát vezérlésével, mindezt távolról, telefonunk segítségével mire hazaérnénk. Ezt nevezzük a magyar fordításban „Dolgok Internetének”, szakkifejezéssel IoT-nek azaz Internet of Things-nek. „Gyakorlatilag az IoT a fizikai tárgyak, eszközök, járművek, épületek és egyéb beágyazott elektronikai elemek hálózati kapcsolatát és adatcseréjét jelenti.” Az IoT eszközök nagy részének (szinte az összes beágyazott eszköz) az alapját valamilyen Linux alapú rendszer biztosítja.

Szakdolgozatom témájának pont egy hasonló IoT feladatot választottam, melyet az előbb példának hoztam fel. Miután érdekeltnek tartom magam a Linux rendszerek programozásában, fejlesztésében, feladatként egy olyan hálózat kialakítását kellett megterveznem aminek a vezérlője egy Linux alapú vezérlő mely maximum 99 db hőmérő szenzorral tudja tartani a kapcsolatot. Az eszköz programját C nyelven írtam meg. A vezérlő feladata egy konfigurációs fájl felolvasása, majd a kommunikáció inicializálása. Inicializálás után több szálon futva, bizonyos időközönként hőmérséklet adatokat kell lekérnie, felolvasnia majd a felolvasott értékeket ki is kell értékelnie. Végül a kiértékelt adatokat egy webes felületen meg is kell tudnia jeleníteni a vezérlőnek. A szolga eszközökről beolvasott adatokat FIFO1-ba kellett gyűjtenem majd a feldolgozó szállnak a FIFO-ból kivett értékékkel kell tovább dolgoznia kettő különböző mérési algoritmussal.

Hogy le tudjam fordítani a C nyelven megírt forráskódomat, külön egy Makefile-t kellett létrehoznom. A Makefile segítségével tudtam a Linux számára futtatható programmá fordítani a forrás fájlokat. Az adatok feldolgozására vagy a program futása során előforduló hibák logolására a Linux rendszer beépített syslog-ját használtam. Ezen kívül még egy másik beépített programot használtam fel a logok megfelelő tárolására. Ez a program a logrotate. Ennek a segítségével biztosítom, hogy a sok loggolt adat és hiba miatt ne teljen meg az amúgy is kevés tárhellyel rendelkező Linux rendszerem. A fejlesztések során egy git repository-t hoztam létre ennek a projektnek. Ez a repository 2 a [http://github.com](http://github.com/)/Dikeszmen/thesis.git oldalon nyilvánosan elérhető. A lefordított futtatható bináris memóriakezelésére a Valgrind programot használtam.(?) A hardware, melyre a fejlesztés készült egy általam előszeretettel használt Raspberry Pi 2B típusú SoC 3 volt.

2. Hardware

2.1 Raspberry Pi 2 B

Szakdolgozatomat egy Raspberry Pi 2 B típusú egylapkás SoC-re készítettem. Ez az eszköz egy nagyjából bankkártya méretű elektronikai eszköz mely méretéhez képest rengeteg funkcióval és lehetőséggel rendelkezik. A hardware „szíve” egy Broadcom BCM2836 processzor mely mellé 1GB memóriát illesztettek a fejlesztők. A processzor 900MHz-en pörög ezzel is biztosítva a gyors és viszonylagos nagy teljesítményt a méretekhez képest. Magára az eszközre 40 db-os apa aljzatú GPIO\* portot implementáltak a mérnökök, melyek különböző kommunikációs protokollt valamint kimeneti vezérlést biztosítanak további fejlesztőknek. Az eszközön található UART, SPI(2 db) és I2C (2 db) interface. Kommunikációs szinten az eszközön még található egy 100Mbit-es Ethernet csatoló és 4db USB-port. Sajnos a 2-es verzióba még nem található bluetooth és a wifi adapter, de 3-as verzióba ezek hiányát már pótolták a tervezőmérnökök. Az eszköz lehetővé teszi kamera, HDMI-s kijelző, touchpad és 3.5mm-es Jack dugó csatlakozását. Az eszköz alá microSD kártyával tudjuk biztosítani az operációs rendszert mely jellemzően valamilyen Linux-disztribúció. Hivatalosan Raspbian névre hallgató, Linux Debian optimalizált verziója ajánlott mint operációs rendszer, de ezen kívül a Microsoft is bejelentette hogy a 2-es verzióra már lehetővé teszi a Windows 10 IoT Core változatát. Ezektől függetlenül megannyi operációs rendszert lehet ráilleszteni mint Fedora alapú Pidora vagy Minibian. Utóbbi a Raspbian minimális operációs rendszere GUI nélkül.--WIKIPÉDIA

Tápegység gyanánt ajánlott egy 5V-os, 2A-es microUSB-s tápot használni. Megfelelő betáplálás esetén a 4 darab USB portonként képes leadni 1,2 Ampert is anélkül hogy segéd tápot kéne biztosítani hozzá.

A fizikai méretei: 85,60mm x 56,50mm (kiálló csatlakozókat leszámítva)

Súlya: 45gramm

Habár eredetileg oktatási célokra tervezték az eszközt, rengeteg területen bevált már mint céleszköz. Miután egy operációs gépről beszélünk szinte bármit lehet rá illeszteni, fejleszteni. Meggyőződésem a határ csak az adott felhasználó fantáziájától függ. Találkoztam már az eszközzel mint torrent szerver, meteorológiai állomás, otthoni média központ, de olvastam már terveket róla mint okosház szíve is akár aminek akadályát nem nagyon találtam. De nem kell messze menni, akár egy sima asztali gépként is lehet alkalmazni melyen, szörfölhetünk az interneten. Mondjuk nagy elvárásokat nem szabad alátámasztani a kis számítógépnek. Számítási teljesítménye hozzávetőleg egy 300MHz-es Pentium II-es gépnek felel meg összességében.

2.2UART

A program kommunikációja fizikai szinten UART-on valósult meg. Az UART jelentése „Universal Asynchronous Receiver/Transmitter” az az univerzális aszinkron adó –és vevő. Az UART a mikrokontrollereknek, SoC-nek olyan perifériája, amely lehetővé teszi az adatok fogadását és adását aszinkron módon. Az adatok küldése és fogadása soros porton történik. Így a bitek egyesével egymás után kerülnek elküldésre a vonalon. A bitek jelszintei TTL (tranzisztor-tranzisztor-logika) szintnek felelnek meg. A logikai 0 az 0V vagy GND, míg a logikai 1, az a tápfeszültség szintje, mely jellemzően vagy 5 V vagy 3,3V. Összesen 3 vezetékre van csak szükség a kommunikáció biztosításához (Tx, Rx és GND). A fizikai kiépítés során ügyelni kell arra, hogy az adó Tx-e (transzmit-je) a vevő Rx-vel (Requestjével) legyen összekötve és fordítva, különben nem fog működni a kommunikáció. Ahogy a neve is mutatja, aszinkron módon történik a kommunikáció, tehát nincs közös órajel adó és vevő között. Ez okból kifolyólag előre kell definiálni mind az adóban, mind a vevőben a közös Baud rate-t, vagyis a kommunikáció sebességét. UART használat előtt a Baud rate-n kívül pár paramétert mindig előre be kell állítani, hogy biztosítani tudjuk a megfelelő kommunikációt eszközeink között. Ezen paraméterek a következők:

Baud rate: A baud rate egy mértékegység nélküli szám, amely megmondja, hogy 1 másodperc alatt hány jel változás történt. Ha a Baud rate 9600 akkor másodpercenként 9600 bitet továbbítunk a vonalon. Meghatározott értékek lehetnek csak a Baud ratek úgy mint 9600, 19200, 38400, 56000 és 115200 attól függően az adott eszköz mennyit tud biztosítani.

Adatbitek száma: Az adatbitek alatt azokat a biteket értjük melyek a start és a paritás bit között található, már ha használunk paritás bitet. Ha nem akkor értelemszerűen a start és stop között található bitek az adatbitek. Az adatbit 5,6,7 vagy 8 bit lehet. Általában 7 vagy 8 bitet szoktak használni a kommunikációra miután az általános ASCII tábla 7 bites. Így egy keret küldése során egy karaktert tudunk továbbítani.

Paritás: A paritást az adatbitek visszaellenőrzésére használták még régebben, bár újabban erre a célra már inkább crc-t használnak. Ettől függetlenül még a mai napig is szokták használni a paritást. A paritás használata opcionális. Amennyiben szeretnék használni, választhatunk, hogy páros vagy páratlan paritást szeretnénk alkalmazni. Páros paritás során a paritás bit 0, ha az adatbitek közül az „1”-sek száma páros, különben 1 a paritás. Páratlan paritás esetén a paritás bit akkor 0, ha az adatban lévő „1”-sek száma páratlan, viszont ha páros, akkor 1 a paritás bit.

Stopbit: Az adatküldés lezárását stopbittel jelezhetjük. Ilyenkor a vonal visszakerül magasba. A stopbit száma lehet 1,2 vagy 1.5. A stopbit száma meghatározza, hogy mekkora szünetre van szükség két adás között.

Általános jelölés módja egy keretnek soros port alkalmazásakor 8N1 vagy 7N2, ahol az első szám az adatbitek számát, a betű a paritást míg az utolsó szám a stop bit számát adja meg.

Fontos tudni, hogy direktbe nem lehet akármilyen adatátviteli szabványt (RS-232,RS-485) rákötni az UART-ra. Gondoskodnunk kell a megfelelő szintillesztésről, megfelelő átalakító használatával ellenkező esetben visszafordíthatatlan kárt tudunk tenni a hardware-ben.

2.3 Hardware Inicializálás

Operációs rendszer gyanánt egy Minibian-ra esett választásom, egy grafikus felület nélküli optimalizált Linux disztribúcióra. Ebből fakadóan sikeres bootolás után, csak egy terminál ablak fogadott mint munkafelület. Grafikus felület nélkül az egész rendszer elfért egy csupán 2 Gigabyte-os microSD kártyán is ezzel is spórolva a költségeken. Csupán terminálból dolgozni megnehezítette volna a dolgomat így egy pár program telepítésre került. Talán a legfontosabb a Midnight Commander (továbbiakban MC) mely egy egyszerű fájl kezelő program. Kinézetre egy az egyben régi Windows-os Norton Commander-re emlékeztet. Használata rendkívül egyszerű, már csak azért is mert rendelkezik beépített szövegszerkesztővel (MCedit) és így nem kellett bajlódni a Linux alapértelmezett Vi nevezetű szövegszerkesztőjével. Előbbi segítségével tudtam elkészíteni a programhoz tartozó fordító programot, a Makefile-t. MC-n kívül még Git verzió kezelőt és Valgrind debugger program került telepítésre.

Sikeres operációs rendszer telepítése során a rendszer inicializálása következett soron. Kettő nagyon fontos beállítást kellett alkalmazni a rendszeren. Először is engedélyezni kellett a rendszer soros port driver-ét. Alapértelmezett állapotban minden driver tiltva van és a felhasználónak kell beállítani azt amire épp szüksége van.

*//sudo raspi-config*

Majd itt serial interfaces… vagy mit. Ezen kívül még az ssh-t\* is itt lett engedélyezve, hogy továbbiakban vezeték nélkül távolról is eltudjam érni az eszközt.

Sikeres beállítás és mentés után már csak egy helyen kellett változtatást végrehajtani a rendszeren. A \boot\ mappában található egy cmd.txt fájl. Ennek a tartalmából ki kell törölni a következő részt: *„console=ttyAMA0, 115200”.* Különben nem lehet tudni paraméterezni se a soros port típusát, se a Baud ratet. Ahhoz hogy ezt érvényre juttassuk, a rendszernek mindenképp szüksége van egy teljes újraindításra.

A program fejlesztése viszont már laptopon készült a Code::blocks IDE programmal grafikus felületen. Ez egy ingyenes program melyben rengeteg hasznos plugin, és eszköz található. A program beépített eszközeinek köszönhetően jelentősen lerövidítette mind a fejlesztési, mind hiba keresési időt.

3. Segédprogramok

3.1 Make

Szakdolgozatom megírása során fontos szempontnak tartottam, hogy ne csak egy fajta Linux alapú rendszeren lehessen használni az általam tervezett programot, hanem az összes olyan eszközön, melyen valamilyen Linux disztribúció van. Magát a lefordított bináris programot értelemszerűen nem lehet csak úgy másolgatni egyik eszközről a másikra, mert minden egyes eszköz más-más módon lett megtervezve. Egyes eszközöknél a hardware implementáció az mely nagyon eltérhet, másoknál lehet egészen más verziója operációs rendszer, amelyekben különböző headerek lehetnek.

Ettől függetlenül még, ha tesztelés során találtam valami hibát a Raspberry Pi-n és azt egyből tudtam javítani Mceditbe akkor nem szerettem volna újra fordítani a teljes programot és másolgatni a kész binárist vissza a cél eszközre. E célból a projektbe létrehoztam egy Makefile-t. Magát a Makefile-t a make-kel, egy Linux alapú parancssori fordító programmal tudtam fordítani. Fő célja leegyszerűsíteni és automatizálni a fordítást. Tételezzük fel találtunk egy hibát, egyetlen egy forrásfájlban vagy csak szimplán módosítottuk, akkor alapesetben ilyenkor újra kell fordítani az egész programot a main-től kezdve az utolsó forrásfájlig ha valamilyen speciális programmal írjuk a programunkat. A make ezt a procedúrát hidalja át azzal a technikával hogy figyeli melyik fájl vagy fájlok módosult(ak) és csak az(oka)t fordítja le ismét. Így megspóroljuk azt a munkát és időt, amit egy teljes fordításkor használunk fel. Nagyobb programoknál jön ki igazán az előnye, amikor a projektben 5-6, esetleg még több forrásfájl is lehet.

/\*Ide kéne egy mért érték két fordítás között\*/.

Annyi feltétele van, hogy ahol található a projekt, abban a mappában abba létre kell hozni egy fájlt amibe leírjuk a make fordítási metódusát. Célszerű Makefile nevet adni neki, mert akkor nem kell feleslegesen make -f <file> kapcsolót és fájl nevet használnunk. Makefile név esetén elég csak egy make parancsot kiadni és máris ellenőrzi a forráskódokat, hogy történt-e módosítás valamelyik fájlon és ha igen akkor azt fordítja is egyből. Fordítási elvén kívül még az is előnyére írható hogy roppant egyszerű megírni egy Makefile-t.

Egy Makefile sémája a következő:

cél : függőségek

parancs(ok)

Fontos hogy a parancs(okat) ne szóköz hanem egy tabulátor előzze meg!

Egy Makefile-ban lehetőség van változók létrehozására, ezzel is egyszerűbbé téve a fordítási programot.

CC=gcc

A változókra való hivatkozás a következő módon történik.

$(CC)

Továbbá a make rendelkezik automatikus változókkal is mely jelentősen leegyszerűsíti a Makefile megírását és használatát. Néhány példa az automatikus változókra.

$\* Teljes forrásfájl neve kiterjesztés nélkül

$< out-of-date forrásfájl neve kiterjesztéssel

$. forrásfájl teljes neve elérési útvonal nélkül

$&. forrásfájl neve elérési útvonal és kiterjesztés nélkül

$: csak az elérési útvonal

$@ Teljes aktuális cél neve

Ezen felül a make rendelkezik saját függvényekkel is.

$(subst from,to, text)

$(subst oo,OO,book on the roof) → bOOk on the rOOf

A függvény a megadott string mintában kicseréli azokat a karaktereket ahol két darab o van egymás mellett két darab nagy O-ra.

$(patsubst pattern,replacement,text)

$(patsubst %.c,%.o,counting.c reading.c) → counting.o, reading.o

Minden olyan fájl, mely .c-re végződik, kicseréli .o végűre.

Utóbbi függvénnyel nem kell felsorolni a projektben megtalálható összes forrásfájlt, hanem így automatizálva mindig az összesre megcsinálja. Ezáltal ha új forrásfájl kerül be a projektbe a make észre fogja venni és fordítani fogja az összes többivel együtt.

https://www.gnu.org/software/make/manual/html\_node/Text-Functions.html

3.2Git

A Git egy verziókezelő szoftver mely arra szolgál, hogy kisebb-nagyobb projektek esetén nyomon lehessen követni a projekt állapotát, a forráskódok tartalmát és verzióját. Ez egy nyílt forráskódú ingyenes szoftver melyet anno Linus Torvalds fejlesztett ki. Git kezelésről rengeteg magyar és angol forrás található az Interneten, ezért részletesen nem tervezek belemenni a program kifejtésébe, működési elvébe. Nagyvonalakban arról szól a Git, ha módosítás történt egy projekten az „add” paranccsal megmondjuk a Git-nek, melyek azok a fájlok melyek módosítva lettek esetleg új fájl lett létre hozva és ezeket szeretnénk hozzáadni a projekthez. Majd a „commit” parancsot kiadva a helyi könyvtárról csinál egy helyi adatbázist a .git könyvtárba. Lokális adatbázist aztán a „push” paranccsal tudunk feltölteni a szerverre amit majd bárki elérhet aki arra jogosult. Nagy előnye a sebességében valami a fa szerkezetében rejlik. Nem szükséges a fő szálon dolgoznunk végig. Ha eszünkbe jut bármiféle újítás, amit nem akarunk a fő ágon vinni, mert csak „kísérletezgetünk” az új ötlettel, nyugodtan létrehozhatunk al ágat is. Ha az alágon történt fejlesztés jónak tűnik, akkor az alágat össze lehet „merge”-lni\* a fő ággal és onnantól kezdve más is láthatja az újítást. Projekt kezdéskor mindig inicializálni kell egy .git könyvtárat, ami a munka könyvtárunk lesz. További előnye még, hogy ezeket a műveleteket tudja biztosítani titkosított csatornán is (SSH) ezzel is növelve az adatbiztonságot. Használati szinten ugyanolyan parancssoros program, mint a make, viszont sok grafikus alkalmazást fejlesztettek mellyel vissza lehet nézni a history-t ki, mikor, mit csinált az adott projekten. Linuxon eddig, amiket használtam a gitk és gitg program. Eléggé egyszerű programok cserébe annál átláthatóbbak a folyamatok.

3.3Valgrind