Tekninen loppuraportti, nro xx

<PIR>

# Kokoonpano

## 1.1 Raspberry Pi

## 1.2 Tietokantapalvelin

# **Koodit**

Projektia varten kirjoitettut skriptit sekä yksinkertaiset sovellukset toteutettiin Pythonilla ja Bashillä. Projektia varten kirjoitettua koodia esitellään tässä osiossa.

Projektia varten luotiin Git-repositorio, johon kaikki koodit ovat kerättynä. Git-repositorio ei ole julkinen, sillä koodit sisältävät selkokielisiä salasanoja.

## 2.1 Pirsensori.py

Projektin pääkomponentti – PIR-sensori ja sen syöttämää dataa tulkittiin Python-scriptillä.

Ohjelman tarkoitus on käsitellä Raspberry Pi:n infrapunasensorin tulkitsemaa dataa. Kun sensori havaitsee liikettä ohjelma kirjoittaa tapahtumasta aikaleimallisen tiedoston. Ohjelmassa tuotuna RPIO-kirjasto, joka on GPIO-moduuli Rasperry Pi:llä.

Pirsensori.py ajetaan Raspberry Pi:llä aina tietokoneen käynnistyessä.

1. import RPi.GPIO as GPIO
2. import time

Käytetyt kirjastot tuodaan koodiin.

1. GPIO.setmode(GPIO.BCM).

Ohjelman alussa GPIO – kirjasto joka tulkitsee Raspberry Pi:n liitettyjen laitteiden syöttämää dataa, käyttämään BCM GPIO-viitettä fyysisten kiinnitysten numeroiden sijaan.

1. GPIO.setup(GPIO\_PIR,GPIO.IN)
2. GPIO\_PIR = 7

Kerrotaan GPIO:lle millaisia asetuksia käytetään ja määritellään PIR-sensorin kiinnityksen numero numeroksi 7.

Ohjelma käyttää muuttujia:

1. Current\_State  = 0
2. Previous\_State = 0
3. Movement  = 0
4. Timenow **=** time**.**strftime("%Y-%m-%d %H:%M:%S")

Jotka ovat kaikki oletuksellisesti ’0’, lukuunottamatta muuttujaa ”Timenow”.

Muuttuja ”Current\_State” seuraa PIR-sensorin lukeman nykyistä tilaa. Muuttuja ”Previous\_State” seuraa muttujan ”Current\_State” edellistä tilaa. Muuttuja ”Movement” seuraa PIR-sensorin lukemaa, arvoilla 1 = sensori on aktiivinen ja 0 = sensori ei ole aktiivinen.

Muuttuja ”Timenow” hakee nykyisen aikaleiman, ja muuttaa sen MySQL-ystävälliseen datetime-muotoon.

1. **while** GPIO.input(GPIO\_PIR)==1:

Ohjelma pyörii while-silmukassa, jossa ehtona sensorin aktiivisuus. Silmukka toistetaan kun sensori palaa aktiiviseksi passiivisesta tilasta.

1. **while** True :
3. Current\_State = GPIO.input(GPIO\_PIR)
5. **if** Current\_State==1 **and** Previous\_State==0:
6. print " -- Motion detected -- "
7. Previous\_State=1
8. Movement = 1

Ohjelma lukee PIR-sensorin tilan ja jos tila on aktiivinen asetetaan muuttujille oikeat arvot. Aikaisempi tila (previous\_state) saa nyt nykyisen tilan (current\_state) arvon, sekä liike tallennetaan muuttujaan ”Movement”.

1. print "Movement: %d " % Movement
2. print ("Time: " + time.strftime("%Y-%m-%d %H:%M:%S"))
3. #save the sensor data into the CoAP -directory
4. file = open("/home/pi/CoAPthon/lukema.txt","w",0)
5. file.write(time.strftime("%Y-%m-%d %H:%M:%S "))
6. file.write("%d"%Movement)
7. print "Writing file..."
8. time.sleep(2)
9. print "Done!"
10. print "Sending the log to NSA.."
11. file.close()

Kun silmukan ehdot ovat täyttyneet ja muuttujille on asetettu arvot – kirjoitetaan tiedot ylös. ”print” –komennot tulostavat muuttujat konsoliin vianetsintää varten.

Pythonin file-toiminnolla voidaan kirjoittaa tiedosto helposti. Huomataan, että parametreiksi määritelty ”w” tarkoittaa sitä, että mahdollinen vanha tiedosto ylikirjoitetaan, eikä vanha tieto säily paikallisesti.

”file.write” –kirjoittaa tiedostoon aikaleiman ja liikkeen tilan (Movement)

”file.close” –lopettaa tiedoston käsittelyn.

1. elif Current\_State==0 **and** Previous\_State==1:

Lopuksi silmukka asettaa nykyisen tilan (current\_state) passiiviseksi ja on valmis uuteen kierrokseen.

1. except KeyboardInterrupt:
2. print "Quit"
3. # Reset GPIO settings
4. GPIO.cleanup()

Silmukasta poistutaan ctrl-c –komennolla

## 2.2 db.py

Tietokantayhteys toteutetaan db.py –skriptissä. Skripti ajetaan paikallisesti MySQL-tietokantapalvelimessa. Skriptin ajaa paikallisen bash-skriptin ja syöttää sen tuomat tiedot muokattuna tietokantaan.

db.py ajetaan tietokanta palvelimessa saatuaan käskyn dbrun.py skriptiltä.

1. import MySQLdb
2. import os

Db.py käyttää yllä mainittuja Python kirjastoja.

1. os.system("sudo bash /home/pidb/projekti/iot-projekti/get.sh")

Ohjelmassa ajetaan paikallinen bash-skripti. Skripti hakee ohjelmalle tekstitiedoston, joka myöhemmin viedään kantaan.

1. db = MySQLdb.connect("localhost", "pi", "raspberry", "raspberry" )

Sillä projektissa tietoturvan tarkastamiseen ei riittänyt aikaa, otetaan yhteys tietokantaan selkokielisillä salasanalla.

1. cursor = db.cursor()
3. #Reads the file fetched by the .sh script
4. print "Reading PIR-sensor data.."
5. file = open("/home/pidb/projekti/iot-projekti/get.txt", "r")
6. file\_content = file.read()
7. file.close()
9. #Modifies the file to fit into the db insert
10. Datetime = file\_content[:-3]
11. Movement = file\_content[20:-1]
13. #printing variables
14. print "Printing variables to be inserted into DB"
15. print Datetime
16. print Movement
17. quote = "'"
19. #Run query with variables from CoAP
20. query = "INSERT IGNORE INTO RAW\_DATA (Datetime, Movement) VALUES ("+quote+Datetime+quote+","+Movement+")"
21. print query
23. cursor.execute(query)
24. db**.**commit()

Seuraavaksi tehdään MySQL INSERT-lause ja viedään se kantaan. ”cursor” on tietokantayhteyden väli, jossa suoritetaan tietokantaa koskeva osuus.

”file” –komennoilla avataan ja käsitellään tiedosto, jonka bash-skripti loi. Muuttujat luodaan muokkaamalla tekstitiedoston sisältöä haluttuun muotoon (file\_content). Tekstitiedoston sisältö on vakio, jonka takia merkkejä poistamalla saadaan haluttu tieto muuttujiin.

query = tietokannan insert-lauseke.

curose.execute – tietokannan osuus päättyy.

db.commit – koodi ajetaan.

1. db.close()

Yhteys suljetaan.

## 2.3 dbrun.py

Dbrun.py ajaa db.py –koodin tietyin aikavälein. Sillä db.py ei itsessään sisällä silmukkaa, halusimme luoda yksinkertaisen ohjelman, joka suorittaa ohjelman joka kymmenes sekunti.

dbrun.py ajetaan tietokantapalvelimella, heti palvelimen käynnistyessä.

1. **import** os

Koodinpätkän toteuttamiseen tarvitsimme vain os –kirjaston.

timeout **=** 10.0 *#wait = 10 sec*

*#Work loop*

**def** **doWork**():

os**.**system("sudo python /home/pidb/projekti/iot-projekti/db.py")

**pass**

l **=** task**.**LoopingCall(doWork)

l**.**start(timeout) *#Call for def every 10 secs*

*#Run*

reactor**.**run()

Koodi määritellä ajamaan funktio joka kymmenes sekunti. Funktiossa doWork ajetaan db.py ja määritetään silmukan aikaväli.

## 2.4 get.sh

Get.sh on yksinkertainen bash-skripti, jolla otetaan yhteys CoAP-palvelimeen. CoAP-palvelimen tuoma tieto tallennetaan tekstitiedostoon paikallisesti, jota db.py hyödyntää. get.sh on tallennettu /usr/local/bin –kansioon ja sille on annettu ajettavan tiedoston oikeudet – sudo chmod x+r get.sh

get.sh ajetaan tietokantapalvelimella aina db.py –skriptin silmukassa.

1. #/bin/bash
3. sudo coap get coap://172.28.170.197/Pir > /home/pidb/projekti/iot-projekti/get.txt

Tiedosto kokonaisuudessaan.

# Projektin onnistuminen

Laatikaa lyhyt yhteenveto projektin kulusta. Analysoikaa projektille suunniteltua tehtävää suhteessa toteutuneeseen. Kuvatkaa projektin tulosten saavuttamista suhteessa suunniteltuun.