Programmeerimise alused (MTAT.03.256)
Projekt
„Homne elektrihind“ ehk elektritarbimise optimeerimine

Raivo Kasepuu
matrikkel B710710

Projekti eesmärgi kirjeldus

Tänastel elektritarbijatel on võimalik valida mitmete elektrivarustuse pakettide vahel. Kes soovib võib elektrihinna pikaks perioodiks ära fikseerida ja olla nii elektribörsi hindadest sõltumatu. Tegelikkuses tähendab see aga seda, et tarbija „riskid“ maandab elektrimüüja ja ta on võimalikud hinnakõikumise kahjud juba kliendi elektrihinda sisse arvutanud. Ehk sisuliselt maksab tarbija elektrimüüjale oma „unerahu“ eest preemiat.

On võimalik valida ka nn börsihinna pakett, kus tarbija hind iga täistunni kohta määratakse NordPool-i elektribörsil. Klient saab homse päeva iga tunni elektrihinnast teada päev varem kell 13:45. Peab mainima, et ka siin on erinevaid lähenemisi. On pakette, kus tõesti mõõdetakse kliendi iga tunni tarbimist ja seda hinnastatakse tunni täpsusega. On ka selliseid keskmist tarbimist arvestavaid pakette, kus kliendi päevane tarbimine jagatakse võrdsetes osades ööpäeva peale laiali ja hinnastatakse nii „pseudo“ börsi hinnaga.

Elektri hind konkreetsel tunnil sõltub paljudest teguritest. Näiteks sellest, kui suur on hetkel turu koormus. Kui koormus on suur ja elektritootjad peavad nõudluse rahuldamiseks käivitama reservvõimsused, viib see elektrihinna üles. Samuti mõjutavad elektrihinda näiteks erinevad rikked ja planeeritud remonditööd. Kui näiteks Eesti ja Soome vaheline merekaabel on remondis, siis ei jõua Eesti turule sellal odav Skandinaavia hüdroelektrijaamade elekter ja hind on märksa kallim. Tuuliste ja /või pikalt vihmaste ilmade puhul on ette tulnud ka hetki, kus elektri hind on negatiivne. Nimelt ei tasu sellal, kui hüdro- või tuuleenergia suudab katta turu elektrivajaduse tavalisi põlevkivi jms fossiilkütust kasutatavaid jaamu välja lülitada. Selline sisse – välja lülitamine võtab märksa rohkem aega, kui mõni tund ja see on ka seotud mitmete riskidega ja see lihtsalt ei tasu ära. Seetõttu tasub mõnedel tootjatel elektri tarbimise eest kliendile peale maksta. See on Skandinaavia turul olnud aastaid tavaline praktika, kuid 2020 tõi esmakordselt ka sellise juhtumi Eesti turupiirkonda. Vastavat Rahvusringhäälingu vahendatud uudist võib lugeda siit: https://www.err.ee/1109740/negatiivse-elektrihinna-taga-olid-odav-tuule-ja-hudroenergia.

Kui vaadelda erinevate päevade elektrihindasid (näiteks siit: https://dashboard.elering.ee/et), siis näeme, et esineb päevi, kus justkui „asja eest, teist taga“ on mõnel tunnil elektri hind lausa kordi suurem sellele eelnenud või järgnevast tunnist. Näiteks hoone küttesüsteem võimaldab seda ära kasutada. Hooned on reeglina pika soojusinertsiga. Kui hoonet köetakse elektriga, siis reeglina ei tekita ka paaritunnine voolu puudumine hoonele tuntavat temperatuuri langust. Me võime kalli hinnaga tunnil kütte välja lülitada ja kompenseerida seda suurema tarbimisega odavama tunnihinnaga tundidel. Ehk kui me õigel hetkel lülitame küttesüsteemi välja võime saada märgatava säästu.

Käesolev projekt on mõeldud soojuspumba juhtimiseks. Kasutame ära börsihindade kõikumist ja kõrge börsihinnaga tundidel lülitame maasoojuspumba välja. Reaalses maailmas töötab programm töötab Raspberry Pi mikroarvutil. Raspberry Pi erinevatest moodulitest ja nende rakendamisest on veebis tuhandeid lehekülgi infot ja projekte. Algust võib teha näiteks siit: https://www.raspberrypi.org/

Minu projekt koosneb kahest eraldi Python-is kirjutatud programmist: põhiprogramm main.py ja nn „klient“, konkreetselt soojuspumpa juhtiv soojuspump.py. Kuna Raspberry-l on rida juhitavaid väljundeid, saab iga seadme

juhtimiseks kasutada erinevaid väljundeid ja kliendipoolseid programme võib olla teisigi , mis kasutavad küll main.py poolt genereeritud andmefaili, kuid rakendades seal erinevaid sisse ja väljalülitamise vms algoritme.

Projekti failid on tehtud „tarbimiseks“ PC tüüpi arvutil. Raspberry kood on vaid veidi erinev, sest siis tuleb juhtida konkreetseid väljundeid ehk nn PIN-e. PC puhul me trükime lihtsalt ekraanile True või False väärtusi, mis imiteerivad siis kas seade töötab või ei tööta.

Programmide töö kirjeldus

Põhiprogramm main.py:

- Küsib kasutajalt andmefaili nime. Kui kasutaja ei ole 10 sekundi jooksul seda teinud, võtab vaikimisi failinimeks **maindata.txt**

- Kontrollib maindata.txt olemasolu. Faili puudumisel see luuakse ja lisatakse esimene triviaalne timestamp: **[{'timestamp': 1}]**

- Küsib Eleringi API kaudu (https://dashboard.elering.ee/swagger-ui.html#/) NordPool Eesti turupiirkonna elektrihindasid. Konkreetne endpoint on näiteks 1.12.2020 hindade leidmiseks selline:

  https://dashboard.elering.ee/api/nps/price?end=**2020-12-01**%2021%3A00&start=**2020-11-30**%2022%3A00

  Konkreetsete päevade hindade saamiseks anname url stringi allakriipsutatud kuupäevade asemel meile sobivad kuupäevad.

  Vastuseks on json:

  {"success":true,"data":{**"ee"**:[{"timestamp":1606773600,"price":20.9900},{"timestamp":1606777200,"price":23.1900},{"timestamp":1606780800 … ,{"timestamp":1606856400,"price":50.1600}]}}

- Filtreerib saadud json-ist Eesti hinnad („ee“);

- Lisab jsoni olemasolevale timestamp-ile inimlikult loetavamad kuupäeva ja tunni väljad:

  [{'timestamp': 1607590800, 'price': 90.26, **'date': '2020-12-10', 'hour': 11**,

- Leiab kolm kõrgeima hinnaga tundi ja markeerib jsoni-i top_1, top_2 ja top_3 boolean väljad False väärtusega. Kui hind pole 3 kõrgeima hulgas, markeeritakse need True :

  'top_1': True, 'top_2': True, 'top_3': True

- Kustutab jsonist vanad (enam, kui 1h) andmed. See võimaldab klientidel lugeda andmefailist oma „tegevuse“ sisenditeks esimese elemendi väärtused. Tulemus on list, mille elementideks on iga tunni kohta sõnastikud (dict):

  [{'timestamp': 1607590800, 'price': 90.26, 'date': '2020-12-10', 'hour': 11, 'top_1': True, 'top_2': True, 'top_3': True}, {'timestamp': 1607594400, 'price': 90.76, 'date': '2020-12-10', 'hour': 12, 'top_1': ….

{'timestamp': 1607637600, 'price': 22.11, 'date': '2020-12-11', 'hour': 0, 'top_1': True, 'top_2': True, 'top_3': True}]

- Kirjutab uuendatud andmetega üle andmefaili.
- Kogu protsessi alates Eleringi hindade küsimisest korratakse timeri abil iga tunni aja tagant. Praktikas uuendatakse Eleringi andmeid kord päevas 16:00, nii et kord tunni tagant olukorra kontrollimine on enam, kui piisav.

Kliendiprogramm soojuspump.py:

- Küsib kasutajalt andmefaili nime. Kui kasutaja ei ole 10 sekundi jooksul seda teinud, võtab vaikimisi failinimeks **maindata.txt**

- Kontrollib andmefaili olemasolu. Kui fail mingil põhjusel puudub (keegi kogemata kustutab vms), siis jääb programm seda ootama ja juhitav seade tööle (väljund „True“) kuni andmefaili taas leitud või olemas.

- Kontrollib andmefaili Top_3 välja. Top_3 väli on „False“ 3 kõige kõrgema hinnaga tundidel. Nendel tundidel trükitakse terminali False. Kui seade peab olema sisselülitatud, siis True. Reaalses elus juhitakse nii Raspberry konkreetset väljundit, mis seadme välja lülitab nendel tundidel.

- Timeri abil korratakse kogu protsessi alates faili olemasolu kontrollist iga 5min järel.

Olulised moodulid programmide käivitamiseks

Kasutasin oma projektis **inputimeout** moodulit, mis võimaldas kasutada automaatset vaikimisi andmefaili nime sisestamist, kui kasutaja ise ei soovi oma andmefaili määrata. Reaalses elus programmeeriksin andmefaili nime koodi sisse. Siinses projektis oli aga nõue, et „Programm peab midagi küsima kasutajalt (kasvõi failinime)“ ja selle täitmiseks (kuifi tegelikku vajadust pole) tuli leida selleks sobiv meetod.

Inputimeout moodul tuleb käsurealt käsitsi paigaldada:

> pip install inputimeout

Programmide koodid

Põhiprogramm main.py:

```
# Projekt Raivo Kasepuu, matrikkel B710710, MTAT 03 256

import time
import datetime
import json
import requests
from os import path
from requests.exceptions import ConnectionError
from inputimeout import inputimeout, TimeoutOccurred

def getDateTimeNow():
    # käesoleva hetke inimkeeli loetava kuupäeva ja kellaaja leidmine
```

```
    return datetime.datetime.fromtimestamp(time.time()).isoformat()


def getHumanDateTime(timestamp):
    # inimkeeli loetava kuupäeva ja kellaaja leidmine etteantud timestamp-ist
    return datetime.datetime.fromtimestamp(timestamp).isoformat()


def getDateForTomorrow():
    # homse kuupäeva leidmine
    return datetime.datetime.fromtimestamp(time.time() + 24 * 3600).isoformat().split("T")[0]


def getDateForYesterday():
    # eilse kuupäeva leidmine
    return datetime.datetime.fromtimestamp(time.time() - 24 * 3600).isoformat().split("T")[0]


def getDateForToday():
    # tänase kuupäeva leidmine
    return datetime.datetime.fromtimestamp(time.time()).isoformat().split("T")[0]


def getTimestampNow():
    # hetke timestampi leidmine
    return int(time.time())

def makeEleringUrl():
    # genereerib Eleringi turuhindade teenuse küsimiseks url-i
    # Näiteks https://dashboard.elering.ee/api/nps/price?end=2020-12-01%2021%3A00&start=2020-11-
30%2022%3A00
    # annab meile jsoni Eleringi hindadega 1.12.2020 -ks
    start = "start=" + str(getDateForYesterday()) + "%2022%3A00"
    end = "end=" + str(getDateForTomorrow()) + "%2021%3A00"
    return "https://dashboard.elering.ee/api/nps/price?" + end + "&" + start


def getEleringEePrices(url, market):
    while True:
        try:
            result = requests.get(url).json()["data"][market]
            if isinstance(result, list):
                return result
        except ConnectionError as e:
            # prindime erro9ri terminalile
            print(e)
            # ootame 10 sekundit ja proovime uuesti kogy try blokki
            time.sleep(10)


def addDateAndHourToPriceData(priceData):
    for i in range(len(priceData)):
        # lisame priceData sõnastikule kuupäeva:
```

```
        priceData[i]['date'] = str(getHumanDateTime(priceData[i].get('timestamp')).split("T")[0])
        # lisame pricedata sõnastikule kellaaja tunnid integerina:
        priceData[i]['hour'] = int(str(getHumanDateTime(priceData[i].get('timestamp')).split("T")[1].split(":")[0]))
    return priceData


def addTopThreeHours(priceData):
    # Lisame priceData tänase päeva sõnastikele lisaväljad top_1 .. top_3
    # markeerides sõnastikke vastavalt top3 hindadele
    topPriceOne = 0
    maxFirstHour = 0
    topPriceTwo = 0
    maxSecondHour = 0
    topPriceThree = 0
    maxThirdHour = 0
    for i in range(len(priceData)):
        if priceData[i].get('price') > topPriceOne and priceData[i].get('date') == getDateForToday():
            topPriceThree = topPriceTwo
            maxThirdHour = maxSecondHour
            topPriceTwo = topPriceOne
            maxSecondHour = maxFirstHour
            maxFirstHour = priceData[i].get('hour')
            topPriceOne = priceData[i].get('price')

    for i in range(len(priceData)):
        if priceData[i].get('hour') == maxFirstHour:
            priceData[i]['top_1'] = False
            priceData[i]['top_2'] = False
            priceData[i]['top_3'] = False
        elif priceData[i].get('hour') == maxSecondHour:
            priceData[i]['top_1'] = False
            priceData[i]['top_2'] = False
            priceData[i]['top_3'] = True
        elif priceData[i].get('hour') == maxThirdHour:
            priceData[i]['top_1'] = False
            priceData[i]['top_2'] = False
            priceData[i]['top_3'] = True
        else:
            priceData[i]['top_1'] = True
            priceData[i]['top_2'] = True
            priceData[i]['top_3'] = True
    return priceData


def fileCreationIfNeeded(fileName):
    if path.exists(fileName) is False:
        emptyData = [{"timestamp": 1}]
        with open(fileName, 'w') as filehandle:
            json.dump(emptyData, filehandle)
        filehandle.close()


# Määrame failinime, kus hoiame oma andmeid:
```

```
try:
   fileName = inputimeout('Sisesta andmefaili nimi (vaikimisi maindata.txt): ', timeout=10)
except TimeoutOccurred:
   print('valisin vaikimisi andmefailiks maindata.txt')
   fileName = 'maindata.txt'

# Kui selline fail puudub, siis tekitame selle koos triviaalsete algandmetega:
fileCreationIfNeeded(fileName)

# Loeme faili sisu json-i:
with open(fileName, 'r') as filehandle:
   mainData = json.load(filehandle)
   filehandle.close()

# Timeri bloki algus:
while True:
   # küsime viimaseid hindu Eleringi API abil
   url = makeEleringUrl()
   market = "ee"
   priceData = getEleringEePrices(url, market)
   # arvutame ja lisame json-ile kuupäeva ja tunni
   addDateAndHourToPriceData(priceData)
   # markeerime top3 hindadega sõnastikud json-is
   addTopThreeHours(priceData)
   # uuendame mainData väärtusi:
   for i in range(len(priceData)):
      if priceData[i].get('timestamp') > mainData[-1].get('timestamp'):
         mainData.append(priceData[i])
   # Kustutame üle 1h (3600 sec) vanad sõnastikud:
   while mainData[0].get('timestamp') < (getTimestampNow() - 3600):
      mainData.pop(0)
   # Prindime mainData terminalile:
   print(mainData)
   # Kirjutame andmete faili üle uute andmetega
   with open(fileName, 'w') as filehandle:
      json.dump(mainData, filehandle)
      filehandle.close()
   # kordame rutiini teatud aja jooksul sekundites.
   # Antud juhul 1h (3600sec). Testimiseks sobib näiteks 5 sekundit: time.sleep(5)
   time.sleep(3600)
```

Kliendiprogramm soojuspump.py:

```
import json
import time
from inputimeout import inputimeout, TimeoutOccurred

# NB! sisesta käsureal pip install inputimeout
# soojuspump on vaikimisi sisselülitatud.
# Raspberry puhul tähendab see et, konkreetse soojuspumba tööd
# juhtiva pin-i tase on "1":
deviceOn = True
```

```
try:
   fileName = inputimeout('Sisesta andmefaili nimi (vaikimisi maindata.txt): ', timeout=10)
except TimeoutOccurred:
   print('valisin vaikimisi andmefailiks maindata.txt')
   fileName = 'maindata.txt'

print("Alustame tööd. Kui terminalis on True, on seade sisse lülitatud, kui False, siis välja lülitatud")

# Timeri ploki algus:
while True:
   # Mida teha siis, kui faili ei ole:
   try:
      with open(fileName, 'r') as filehandle:
         mainData = json.load(filehandle)
         filehandle.close()
   except IOError:
      print('file not found')
      print('Raspberry puhul anname siin ühele väljundile pinge peale, et LED alarmeeriks')

   # kontrollime, kas seade peab olema sisse- või väljalülitatud
   # antud juhul on soojuspump top3 elektrihinnaga tundidel väljalülitatud
   if mainData[0].get('top_3') is False:
      deviceOn = False
   else:
      deviceOn = True
   # Kontrolliks prindime terminali, mis on hetke seis:
   print(deviceOn)
   # kontrollime olukorda uuesti 5 min (300sec) pärast
   # programmi testimiseks kasuta näiteks 3 sec: time.sleep(3)
   time.sleep(300)
```

Iseseisva programmi tegemine

Iseseisva programmi tegemine. Otseselt vajadust iseseisvaks programmiks selles projektis pole, sest Raspberry saab kenasti hakkama, kui talle Pythoni script ette antakse, kuid sellegipoolest oli huvitav ka see etapp läbi teha.

Meetodeid selleks leidsin mitmeid. Kasutasin pyinstallerit (PyInstaller Quickstart — PyInstaller bundles Python applications):

Näiteks soojuspump.py – st iseseisva programmi tegemine:

Kõigepealt pidin käsurealt pyinstalleri paigaldama:

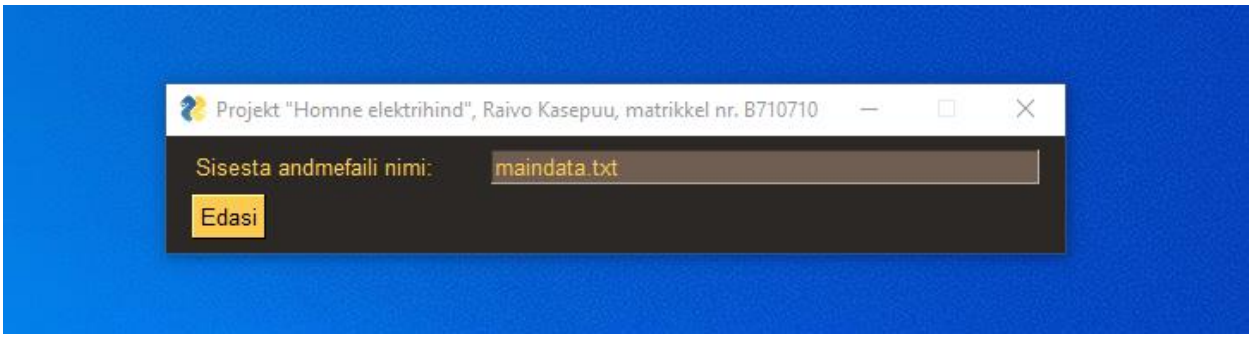
> pip install pyinstaller
Käivitama pyinstalleri:

> pyinstaller –onefile soojuspump.py

Graafilise kasutajaliidese tegemine

Projekti algfaasis plaanisin ka graafilist kasutajaliidest, kuid et kogu projekt peaks töötama ilma sisendita, st kui seadmele toide peale anda, peab seade ise kõikide tekkivate olukordade (pole elektrit, pole netiühendust, andmefail kadunud jne) lahendamisega hakkama saama, siis loobusin GUI arendamisest. Siiski, et märk tehtus jääks, väike tollane koodilõik ja screenshot:

```
# Graafiline kasutajaliides
sg.theme('Dark Brown 1')
layout = [
   [sg.Text('Sisesta andmefaili nimi: ', size=(20, 1)), sg.InputText("maindata.txt")],
   [sg.Submit("Edasi")]
]
window = sg.Window('Projekt "Homne elektrihind", Raivo Kasepuu, matrikkel nr. B710710', layout)
event, values = window.read()
window.close()
dataFile = values[0]
```



Projekti ajakulu aruanne

Kõige rohkem kulus aega teostuse läbimõtlemisele. Kuidas oleks kõige mõistlikum siduda erinevaid seadmeid, kuidas lahendada, mida konkreetselt teeb põhiprogramm, mida kliendiprogramm. Seda ajakulu pole konkreetselt mõõtnud, kuid hinnanguliselt läks kokku 2h.

Veidi kulus aega ka asjaajamisele, sest esialgne valik, NordPool, ei olnud nõus tudengile tasuta API-le ligipääsu tagamast. Nende hind aastaseks litsentsiks on 3500€. Eleringi puhul oli segavaks asjaolu, et nende API kirjelduses puudusid viited, millal nad hindasid uuendavad. Kui NordPool teeb seda 13;45, siis Elering alles 16:00. Kuna mina kirjutan koodi reeglina päevasel ajal, siis oli pikalt kahtlus, et Elering ei avaldagi homseid hindasid. Siiski, sain neilt pöördumisel korrektse info ja sain projektiga edasi minna. Loeksin asjaajamise ajakuluks kokku 2h.

Konkreetselt eesmärgiks seatud algoritmile lahenduste otsimine, Pythoni koodi kirjutamine ja erinevate olukordade testimine võttis kokku ca 6h.

Projekti vormistamine võttis ca 1h

Kokku projekti ajakulu seega 11h.

Projekti nõuete täitmine

Vastavalt moodle-s kirjas olnule saan kommenteerida järgmist:

1. Projekti idee (tähtaeg 16. nov, max 2 punkti) – TEHTUD, punktid saadud.
2. Ülesande täpse püstituse teeb üliõpilane ise. Temaatika peab teile endale huvi pakkuma. – TEHTUD.
3. Peab olema enda tehtud – Kinnitan, et enda TEHTUD.
4. Orienteeruv töömaht lahendamisel 8 tundi. (Ajakulu esitada eraldi ligikaudse aruandena.). Kui teete programmi kahekesi, siis kummalgi 8 tundi – TEHTUD.
5. Programm peab töötlema andmeid – TEHTUD.
6. Andmed võivad olla pärit veebist (stat.ee vm) või enda omad – TEHTUD, main.py.
7. Andmed tuleb lugeda failist (nt csv) – TEHTUD, soojuspump.py.
8. Programm peab midagi küsima kasutajalt (kasvõi failinime) – TEHTUD.
9. Võib eeldada, et kasutaja sisestab sobivate andmetega faili nime – TEHTUD.
10. Programm peab andmete põhjal midagi mõistlikku arvutama ja tulemuse ekraanil esitama. -TEHTUD
11. Järgmistest nõuetest võib jätta kaks täitmata

- graafiline väljund (diagramm vms) – töö käigus on TEHTUD graafiline kasutajaliides, kuid rakendust sellele polnud ja jäi projektist välja.
- filtreeritud andmete teise faili kirjutamine - TEHTUD
- kasutaja valiku põhjal arvutuste tegemine - TEHTUD
- iseseisva (ilma Pythonita käivituva) programmi tegemine - TEHTUD

Lahtiütlemine / Disclaimer

Minu projekti võib avaldada ja kasutada MIT litsentsi (https://opensource.org/licenses/MIT) järgi. Ehk maakeeli lahti seletatult:

- kasuta omal vastutusel minule viidates (Raivo Kasepuu, raivokasepuu@gmail.com).
- Ma ei võta mingit vastutust võimalike tekkida võivate kahjude osas.
- Ma ei anna oma koodile mingeid garantiisid.

Olen tänulik tagasiside eest e-mailil: raivokasepuu@gmail.com

Kokkuvõte

Käesolev projekt on vaid üks osa nn „Targa kodu“ rakendusest, sest pisike ja tubli Raspberry Pi suudab märksa enamat, kui juhtida vaid soojuspumpa. Analoogselt saab juhtida näiteks kodust ventilatsioonisüsteemi - kui maja on valvesüsteemis valve alla pandud ja tegelikult keegi ventilatsiooni ei vaja, võib selle välja lülitada ja automaatselt käivitada, kui keegi hoonesse siseneb. Või siis jälgida päikese tõusu ja loojangut, et automaatselt ette/eest tõmmata kardinaid, juhtida päikest järgivat päikesepaneelide süsteemi jne.