SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

SERVER ZA PRIJENOS PODATAKA IZ POSTROJENJA

Diplomski rad

Damir Jelić

Osijek, 2015.

Sadržaj

1. UVOD

Cilj diplomskog rada je ispitati mogućnost korištenja jeftinih mikroupravljača za jednostavne poslove automatizacije te omogućiti udaljeno upravljanje postrojenjem preko *socket server-a*.

Kako bi se utvrdio cilj diplomskog rada pomoću mikroupravljača je simulirano postrojenje. Komunikacija prema računalu je ostvarena putem *USB* sučelja. Računalo šalje naredbe prema mikroupravljaču pomoću komunikacijskog protokola. *Socket server* je izrađen tako da s jedne strane koristi navedeni protokol da bi komunicirao s mikroupravljačem, a s druge strane omogućuje klijentima upit u stanje postrojenja i postavljanje referentne varijable postrojenja.

Za mikroupravljač je odabran *Arduino Uno* koji preko USB-serijskog sučelja i preko *Firmata* protokola komunicira s računalom. Na *Arduino* je spojena vodena pumpa i senzor koji mjeri razinu vode u boci. Sustav od dvije boce čini postrojenje, u jednoj boci održava se razina vode. Na računalu se nalazi *socket server* pisan u *Haskell-u* koji upravlja mikroupravljačem. Testni klijent pisan u *Python-u* se spaja na *server* te dohvaća mjernu veličinu (razinu vode u boci) i prikazuje trenutno stanje sustava.

Rad je podijeljen u dva dijela. U prvom dijelu razrađene su teorijske pretpostavke vezane uz temu. Važno je napomenuti kako je u ovom dijelu detaljno opisan proces automatizacije i razvoj socket servera. Drugi dio bavi se implementacijom postrojenja, regulatora, *socket server-a* i popratnog klijenta.

2. PREGLED KORIŠTENIH PROGRAMSKIH JEZIKA I IDIOMA

U ovom poglavlju je detaljno opisan dings... bums...

2.1 Haskell

Haskell je funncionalni programski jezik koji

- static typed
- type safety
- laziness

2.1.1 Networking

Socket lib.

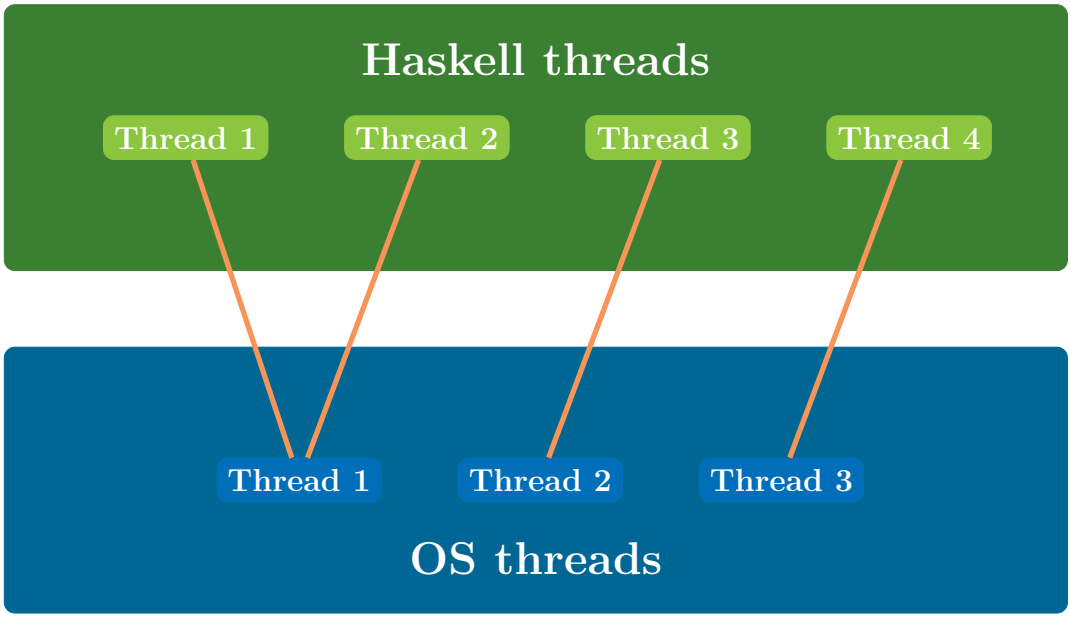
Slika 2.1.: Dings kak se server napravi

```
socket :: Family -> SocketType -> ProtocolNumber -> IO Socket

bind :: Socket -> SockAddr -> IO ()

listen :: Socket -> Int -> IO ()

socketToHandle :: Socket -> IOMode -> IO Handle
```

2.1.2 Istodobnost

Istodobnost (engl. *Concurrency*) je svojstvo sustava s kojim se istodobno mogu izvršavati više računskih operacija ili komputacija.

Istodobnost u Haskell-u je uglavnom izveden pomoću 'Zelenih niti" (engl. *Green threads*). Zelene niti se ne izvršavaju u kernel-u već u *Haskell runtime-u*. Haskell ima hibridni *threading* model s kojim se N Haskell niti mogu vezivati na M *kernel* niti.



Slika 2.2.: N:M višenitni model Haskell-a

Interno se u Haskell-u stvaranje nove niti pretvara u alokaciju strukture koja sprema trenutno stanje niti te se niti pretvaraju u jednu petlju.

Za stvaranje nove niti Haskell nudi `forkIO` funkciju čiji je tip definiran kao:

```
forkIO :: IO () -> IO ThreadId
```

Što znači da funkcija prima kao prvi argument komputaciju (engl. *computation*) i vraća kao rezultat *ThreadId* što nam služi kao referenca na novu nit koju će funkcija stvoriti.

Osim niti za istodobno izvršavanje programa bitna je i komunikacija između niti. Za komunikaciju izmedu niti Haskell nudi mutabilne dijeljene varijable zvane MVar (Mutabel Variables). One se mogu koristiti na razne načine:

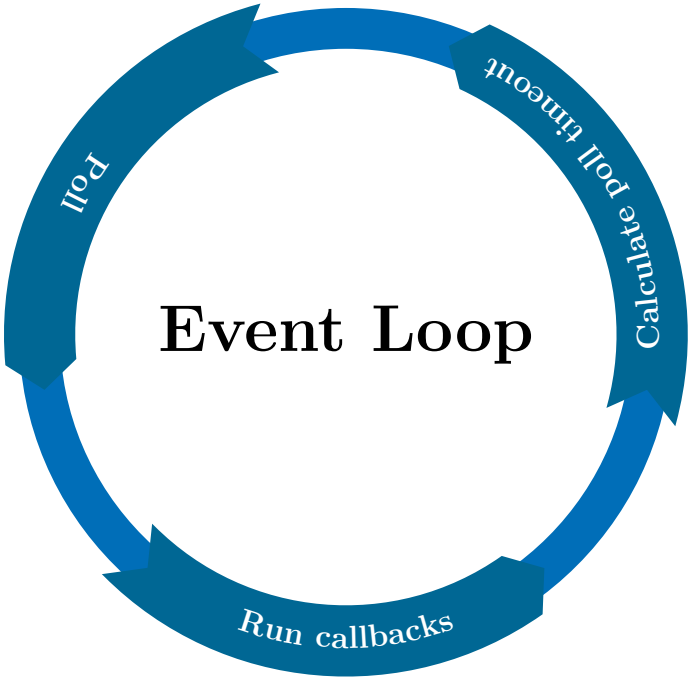
- Sinkronizirane mutabilne varijable.
- Komunikacijski kanali između niti.
- Binarni semafori.

Mvar:

```
newMVar :: a -> IO (MVar a)
```

2.2 Python

2.2.1 Event loop



Slika 2.3.: Event loop schematic

2.2.2 Urwid

2.2.3 Python Networking

Slika 2.4.: Dings kak se client napravi

2.3 Firmata

Naredba	Command?
analog I/O message	0xE0
digital I/O message	0x90
report analog pin	0xC0
report digital port	0xD0
start sysex	0xF0
set pin mode(I/O)	0xF4
set digital pin value	0xF5
sysex end	0xF7
protocol version	0xF9
system reset	0xFF

Tablica 2.1.: Tablica

2.4 Matematički model postrojenja

$$\frac{dV}{dt} = q_{in} - q_{out} \tag{2-1}$$

$$q_{out} = a\sqrt{2gh} \tag{2-2}$$

$$q_{in} = k_p V \tag{2-3}$$

$$A\frac{dh}{dt} = k_p V - a\sqrt{2gh} \tag{2-4}$$

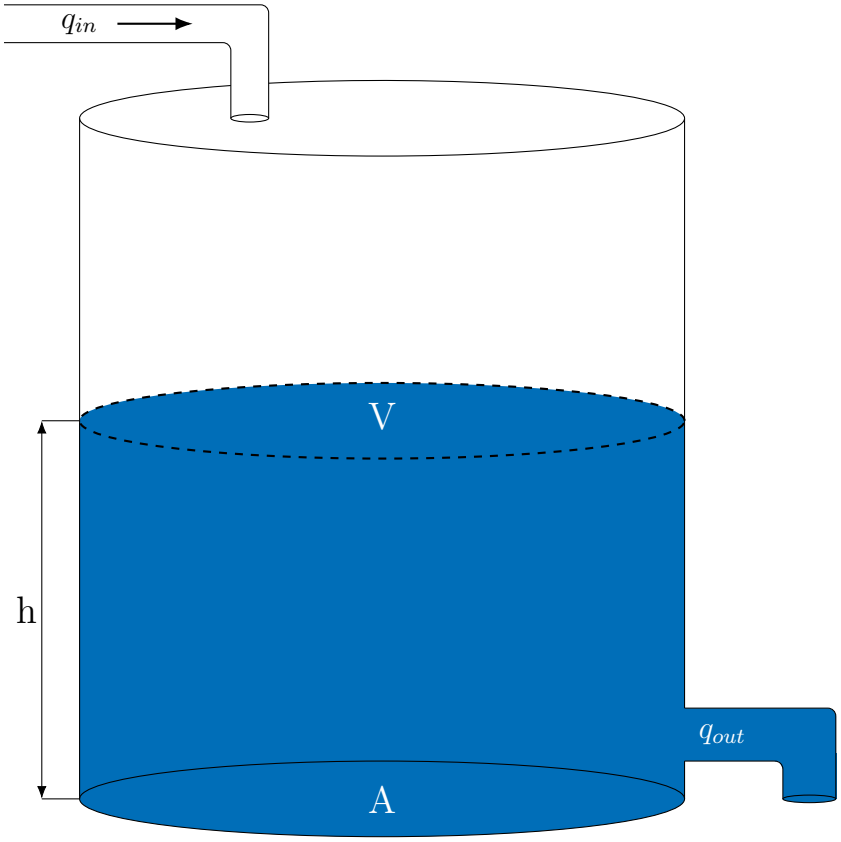
$$\frac{d\delta h}{dt} = \frac{k_p}{A}\delta V - \frac{a\sqrt{2g}}{2A\sqrt{h_0}}\delta h \tag{2-5}$$

$$\frac{d\delta h}{dt} = k_1\delta V - k_2\delta h \tag{2-6}$$

$$sH(s) = k_1 V(s) - k_2 H(s) \tag{2-7}$$

$$H(s)(s + k_2) = k_1 V(s) \tag{2-8}$$

$$G(s) = \frac{H(s)}{V(s)} = \frac{k_1}{s + k_2} \tag{2-9}$$



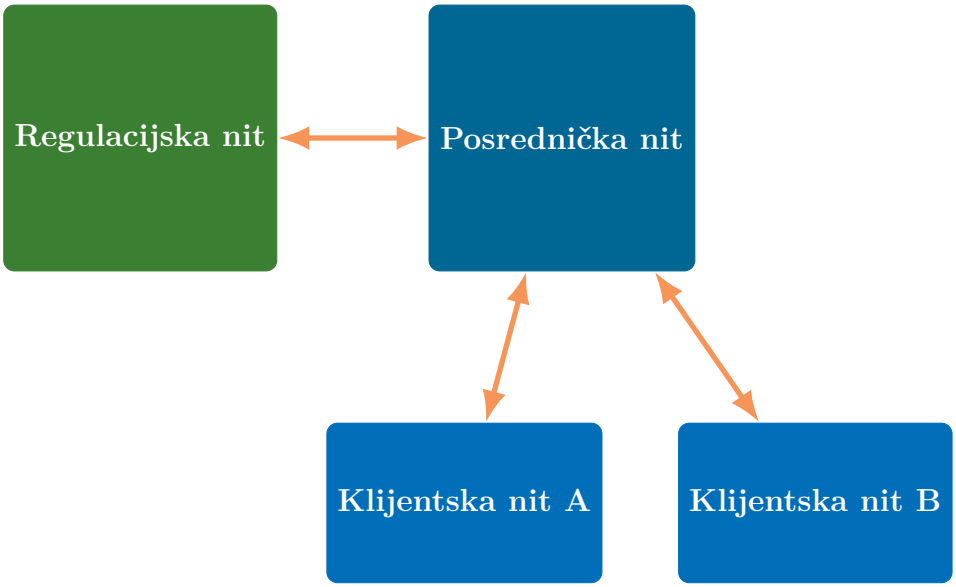
Slika 2.5.: Šematski model postrojenja

3. IMPLEMENTATION

U ovom poglavlju je objašnjena implementacija socket servera, odgovarajućeg klijenta i na kraju samog postrojenja.

3.1 Server

Arhitektura servera.



Slika 3.1.: Arhitektura servera

Na slici 3.1. je prikazana komunikacija između pojedinih niti unutar servera. Niti komuniciraju pomoću djeljenih varijabli. ... je dobro.

```
startDaemon :: Integer -> FilePath -> Bool -> IO ()
startDaemon port arduinoPort simulate = do
    sock <- socket AF_INET Stream 0

    bindSocket sock $ SockAddrInet (fromInteger port) iNADDR_ANY
    listen sock 50

    pv <- newMVar 0
    referenceChan <- newChan
    let com = ProcCom pv referenceChan

    _ <- forkIO $ mainLoop sock com

    controllerBroker arduinoPort simulate com
```

Ispis koda 3.1: Main entry point

Na ispisu koda 3.1 je.

```haskell
mainLoop :: Socket -> ProcCom PVType -> IO ()
mainLoop sock com = do
    (hdl, host, port) <- accept sock
    let hostinfo = host ++ ":" ++ show port
    noticeM rootLoggerName $ "Connected:     " ++ hostinfo
    _ <- forkIO $ runConn hdl com hostinfo

    mainLoop sock com
```

Ispis koda 3.2: Server mainloop

```haskell
runConn :: Handle -> ProcCom PVType -> String -> IO ()
runConn hdl com hostinfo = do
    isEof <- hIsEOF hdl

    if isEof then do
        hClose hdl
        noticeM rootLoggerName $ "Disconnected: " ++ hostinfo
    else do
        contents <- hGetLine hdl
        debugM rootLoggerName $ "RPC request : " ++ contents

        response <- handleMsg com $ C.pack contents
        debugM rootLoggerName $ "RPC response: " ++ C.unpack response

        C.hPutStrLn hdl response
```

Ispis koda 3.3: Client handler thread

```haskell
controllerBroker :: FilePath -> Bool -> ProcCom PVType -> IO ()
controllerBroker arduinoPort simulate (ProcCom pvMVar refChan) = do
    refMVar <- newMVar 0

    _ <- forkIO $ forever $ do
            ref <- readChan refChan
            swapMVar refMVar ref

    if simulate then
        simulatorLoop refMVar pvMVar
    else do
        controlLoop arduinoPort refMVar pvMVar
        shutDownArduino arduinoPort

    noticeM rootLoggerName "Shutting daemon down."
```

Ispis koda 3.4: Controler broker

3.2 Client

```
def main():
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((args.host, args.port))

    tank = TankWidget()
    command_line = CommandLine(cmd_list, remote_cmds, sock)
    top = urwid.Frame(tank, None, command_line, 'footer')

    evl = urwid.AsyncioEventLoop(loop=asyncio.get_event_loop())
    loop = urwid.MainLoop(top, palette, event_loop=evl)

    loop.watch_file(sock.fileno(), read_cb)
    loop.set_alarm_in(0.1, periodic_tasks)

    loop.run()
```

Ispis koda 3.5: Client main loop

```
    def periodic_tasks(loop, data):
        request , _ = lvl_cmd('update-tank', [])
        try:
            sock.send(bytes(request, 'utf-8'))
        except BrokenPipeError as e:
            pass

        tank.update()

        loop.set_alarm_in(args.interval, periodic_tasks)

    loop.watch_file(sock.fileno(), read_cb)
```

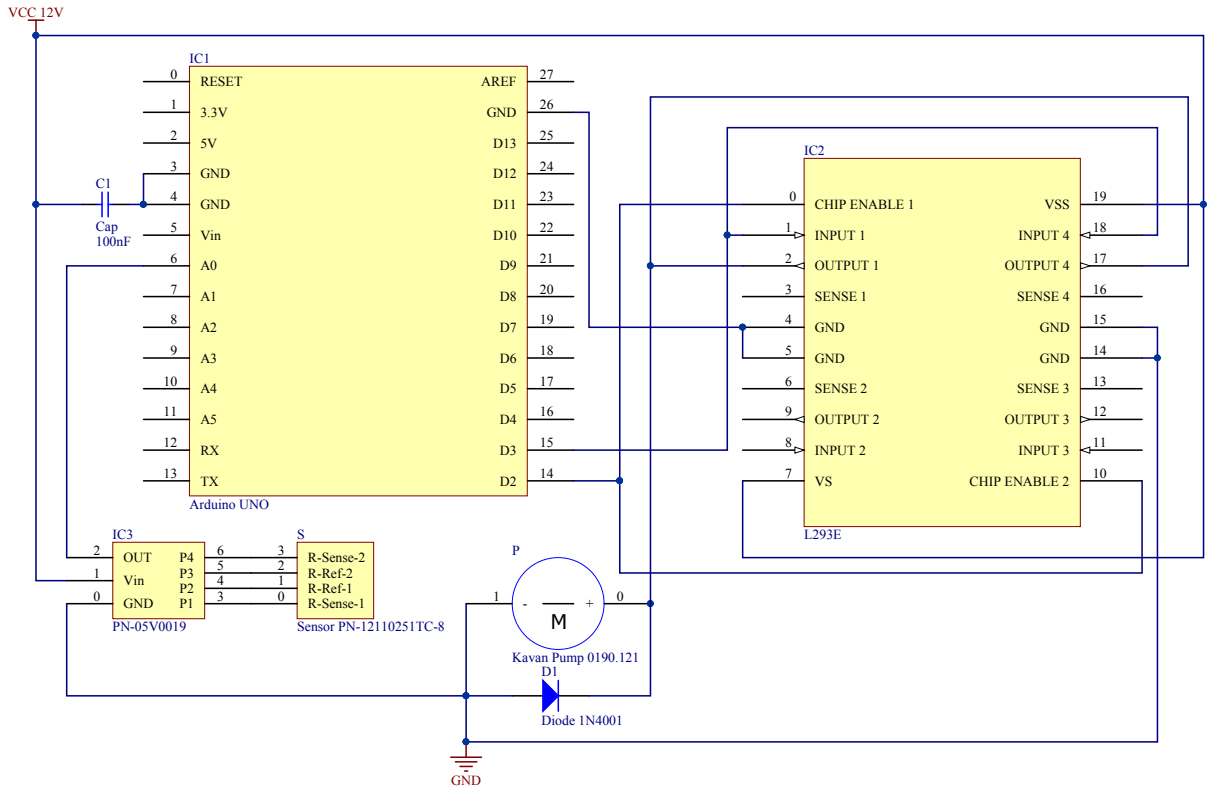
Ispis koda 3.6: Periodic update dings

3.3 Regulator

```
forever $ do
    integral  <- liftIO $ takeMVar integralMVar
    reference <- liftIO $ readMVar refMVar

    sensorValue <- analogRead sensor
    let fillHeight = sensorValueFunc sensorValue
    _ <- liftIO $ swapMVar pvMVar fillHeight

    let error = reference - fillHeight

    let proportional_term = kp * error
    let integral_term     = integral + ki * error / sampleTime

    let output = clampAndScaleOutput $ proportional_term + integral_term

    analogWrite pwm_pin output
    delay sampleTime
```

Ispis koda 3.7: Regulacijska petlja

3.4 Postrojenje



Slika 3.2.: Električna šema postrojenja

BIBLIOGRAFIJA

[1] Paul Adrien Maurice Dirac. *The Principles of Quantum Mechanics*. International series of monographs on physics. Clarendon Press, 1981. ISBN: 9780198520115.

[2] Albert Einstein. "Zur Elektrodynamik bewegter Körper. (German) [On the electrodynamics of moving bodies]". *Annalen der Physik* 322.10 (1905), str. 891–921. DOI: http://dx.doi.org/10.1002/andp.19053221004.

[3] Donald Knuth. *Knuth: Computers and Typesetting*. URL: http://www-cs-faculty.stanford.edu/~uno/abcde.html.

[4] Donald E. Knuth. "Fundamental Algorithms". Addison-Wesley, 1973. Pogl. 1.2.

[5] Norman S. Nise. *Control Systems Engineering*. Wiley, 2010. ISBN: 0470917695.

[6] Simon Peyton Jones, Andrew Gordon i Sigbjorn Finne. "Concurrent Haskell". *Proceedings of the 23rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. POPL '96. St. Petersburg Beach, Florida, USA: ACM, 1996, str. 295–308. ISBN: 0-89791-769-3. DOI: 10.1145/237721.237794. URL: http://doi.acm.org/10.1145/237721.237794.

[7] Andrew M. Rudoff W. Richard Stevens Bill Fenner. *Unix Network Programming Volume 1, Third Edition: The Sockets Networking API*. Addison-Wesley Professional, 2003. ISBN: 0131411551.