

Pràctica 8: Disseny web del control remot d'un led

Aplicacions i serveis d'internet — Enginyeria de Sistemes TIC

Francisco del Àguila López Aleix Llusà Serra

2 de maig de 2023

Índex

1	Organització	2
1.1	Objectius	2
1.2	Condicions	2
1.3	Lliurables	2
1.4	Material necessari	2
2	Introducció al servidor amb Python	2
2.1	Escenari	3
3	Creació de servidors amb Python	3
3.1	socketserver	4
3.2	http.server	4
3.2.1	Servidor simple	4
3.2.2	Servidor propi	5
3.2.3	Atendre les consultes	6
4	Feina a fer	7
4.1	Interacció entre Servidors	8
4.2	Portal Web	8
4.3	Connexió sèrie amb l'Arduino	8
5	Extensions	9
5.1	Atendre en multiprocés	9

Resum

Protocols d'internet: HTTP
Llenguatges web: HTML, JavaScript
Llenguatges de programació: Python

1 Organització

1.1 Objectius

El objectius d'aquesta pràctica són:

1. Entendre el disseny bàsic d'un servidor web.
2. Dissenyar dinàmicament mitjançant servidor web `Python`.
3. Usar el mòdul `http.server` de la llibreria de `Python`.
4. Atendre el control del led de la pràctica de disseny web.
5. Interactuar amb l'entorn, mitjançant un Arduino, des del servei web.

1.2 Condicions

- La pràctica està calibrada per a ésser treballada en equips de dues persones.
- La durada de la pràctica és d'1 setmana.

1.3 Lliurables

Heu d'entregar:

- Els programes que dissenyeu.
- Les pàgines web que dissenyeu.
- Tot allò que calgui per reproduir l'ecosistema de treball, com els fitxers de configuració dels serveis que s'han instal·lat.

1.4 Material necessari

En aquesta pràctica utilitzarem la infraestructura de servei web i el portal web dissenyats a les pràctiques anteriors.

Per a interactuar amb l'entorn és necessari un Arduino.

2 Introducció al servidor amb Python

A la pràctica sobre servei web (P5) heu vist un ús ràpid del servidor web amb `Python` mitjançant el mòdul `http.server`. Aquesta pràctica avalua més profundament aquest mòdul per a dissenyar un servidor web propi a mida.

El disseny d'un servidor web propi és útil per a tasques simples i a on es necessita ser molt flexible en l'atenció del protocol HTTP. A més, permet combinar el servei web amb altres accions en un mateix programa, el qual és molt útil per a encastar tota l'aplicació en dispositius petits com l'Arduino o la RaspberryPi.

A la pràctica sobre servei web també heu vist l'ús d'un servidor web generalista, `Apache`. Els servidors web genèrics incorporen funcionalitats per a totes les tasques web típiques i a més

són extensibles. Són molt còmodes per a ampliar la funcionalitat del servei web amb tasques complicades, com per exemple autenticació o xifratge.

En el disseny d'un servei web és útil aprofitar les funcionalitats que ofereixen els servidors propis i els genèrics. Així doncs, en aquesta pràctica combinarem el disseny d'un servidor web propi amb l'ús d'un servidor web genèric.

2.1 Escenari

En el vostre domini d'autoritat `gXX.asi.itic.cat` heu de dissenyar un portal web que permeti controlar un led remotament. Aquesta interfície web permetrà per una banda visualitzar l'estat actual del led i per altra banda actuar i canviar-ne l'estat.

L'esquema de l'escenari és el de la figura 1:

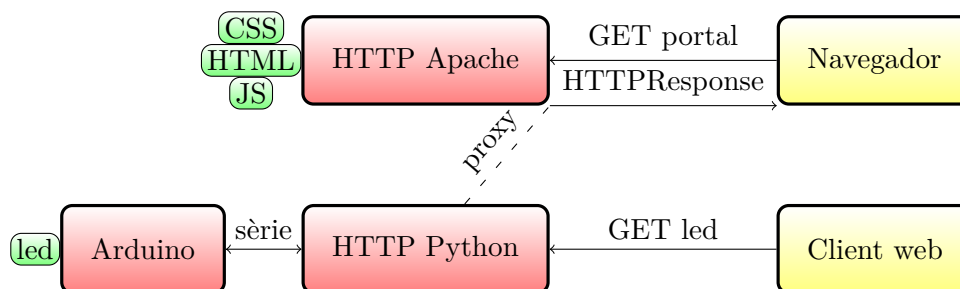


Figura 1: Esquema web per al control remot

- Un Arduino controla un led.
- Un programa Python per una banda estableix una connexió sèrie amb l'Arduino i per altra banda implementa un servidor web específic que permet controlar el led segons l'adreça URL que es visiti.
- Un servidor web amb Apache ofereix una interfície web amable per al control del led. Les sol·licituds de control que es fan a la interfície web s'atenen a través d'un proxy reverse cap al servidor Python.
- Els clients poden controlar directament el led donant les ordres escaients al servidor amb Python o bé usar la interfície web a través d'Apache. En un escenari més complicat podríem introduir seguretat en les accions que els clients poden fer.

3 Creació de servidors amb Python

A la pràctica de *sockets* es va veure com crear tant aplicacions Client com Servidor aprofitant el mòdul *socket* de la llibreria estàndard de Python dins del conjunt de mòduls al grup de *Networking and Interprocess Communication*. Podeu consultar la llibreria estàndard a <https://docs.python.org/3/library/index.html>. Però a l'apartat de *Internet Protocols and Support* hi ha 2 mòduls que ens facilitaran la tasca de desenvolupar *Servidors* a Internet, i en particular *Servidors Web*. Aquests mòduls són:

- `socketserver` catalogat com: *Un framework para servidores de red.*

- `http.server` per definir *Servidors HTTP*.

3.1 socketserver

Aquest mòdul ofereix unes classes per definir servidors de tipus TCP, UDP o bé *sockets* de Unix. En el cas del TCP és `socketserver.TCPServer`. Aquesta classe, a part de necessitar com a paràmetre l'adreça IP i el port TCP, necessita un *RequestHandlerClass*. Aquest *RequestHandlerClass* es crea com a subclasse de *BaseRequestHandler* on s'ha de sobrecarregar el mètode `handle()`. En el mètode `handle()` és on es defineix el tractament que es vol donar a la petició que ve del Client. Per engegar el servei es recomana fer servir la sentència `with`, on es crida al mètode bloquejant `serve_forever()`. Un exemple el teniu a <https://docs.python.org/3/library/socketserver.html#socketserver-tcpserver-example>, on també podeu consultar tota la documentació del mòdul.

3.2 http.server

El mòdul `http.server` de la llibreria de Python serveix per implementar un servidor web, tant de continguts estàtics com dinàmics. En aquesta pràctica l'utilitzarem per a la comunicació de continguts dinàmics. Abans, però, veurem com s'estableix el servei web mitjançant `http.server`.

Amb Python3, disposeu de la documentació a <http://docs.python.org/3/library/http.server.html>.

3.2.1 Servidor simple

A la pràctica del servei web ja vaurem veure com servir documents estàtics amb el servidor web de Python:

```
python -m http.server 8000
```

Per implementar el servidor simple a dins d'un programa propi cal fer servir el mòdul `socketserver`. L'esquema bàsic és el següent:

```
import http.server
import socketserver

PORT = 8000

Handler = http.server.SimpleHTTPRequestHandler

with socketserver.TCPServer(("", PORT), Handler) as httpd:
    print "serving at port", PORT
    httpd.serve_forever()
```

Consulteu la documentació del mòdul per a comprendre'n bé el funcionament. El nucli de funcionament rau en dos objectes:

- `socketserver.TCPServer` Construeix un servidor TCP lligat a una adreça concreta i hi atén les peticions segons l'objecte `Handler` de la classe `http.server.SimpleHTTPRequestHandler`.

- La classe general `RequestHandlerClass` llegeix els missatges HTTP de petició del Client i elabora els missatges HTTP de resposta. El mòdul ofereix tres especialitzacions d'aquest objecte:
 - `BaseHTTPRequestHandler` Implementa un gestor bàsic: ofereix mètodes i atributs genèrics per a la gestió HTTP. De fet, no implementa cap atenció dels missatges HTTP ja que està pensat per a ser utilitzat en subclasses.
 - `SimpleHTTPRequestHandler` És un `BaseHTTPRequestHandler` que sap atendre els mètodes HEAD i GET, aquest últim mapa les URL a noms de fitxers locals i els serveix amb el tipus MIME corresponent o llista el contingut en cas que siguin directoris. Observeu que aquesta és la classe que s'ha fet servir fins ara per muntar els exemples de servidor web amb Python.
 - `CGIHTTPRequestHandler` És un `SimpleHTTPRequestHandler` que a més sap executar scripts CGI.

3.2.2 Servidor propi

Ara implementarem un `RequestHandlerClass` propi. A tal efecte cal crear una subclasse de `BaseHTTPRequestHandler` que com a mínim implementi el mètode `do_GET()`; cal un mètode Python `do_X()` per a cada mètode HTTP que es vulgui atendre – GET, POST, HEAD, etc.

El mètode `do_X()` es crida sense arguments; les dades de la consulta HTTP es troben emmagatzemades en atributs de la instància. Consulteu-ho a la documentació de `BaseHTTPRequestHandler`.

La resposta que elaboren els mètodes `do_X()` ha de seguir el protocol HTTP:

1. Una línia amb el codi de resposta de la forma `<version> <responsecode> <responsestring>`.
Codis de resposta: `http://www.w3schools.com/tags/ref_httpmessages.asp` o vegeu l'atribut `responses` de la classe `BaseHTTPRequestHandler`.
2. Zero o més línies amb les capçaleres (RFC-822); si a la resposta hi ha dades com a mínim hi ha d'haver el tipus MIME: `Content-type: <type>/<subtype>`
3. Línia en blanc
4. Les dades

Per a elaborar la resposta com a instància `BaseHTTPRequestHandler`:

- Amb el mètode `send_response(codi)` s'estableix el codi de resposta.
- Amb el mètode `send_header(clau,valor)` s'estableix cada una de les capçaleres.
- Amb el mètode `end_headers()` es finalitza l'escriptura de capçaleres.
- L'atribut `wfile` conté la sortida per a escriure el cos de la resposta HTTP.

Com a exemple es defineix un `BondiaHTTPRequestHandler` que atén el mètode GET amb un document estàtic generat en el programa:

```
import http.server
import socketserver
```

```

class BondiaHTTPRequestHandler(http.server.BaseHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200)
        self.send_header("Content-type", "text/html")
        self.end_headers()

        resposta = """
        <html>
        <head>
        <title>Bon dia</title>
        </head>
        <body>
        <p>Bon dia</p>
        </body>
        </html>
        """

        self.wfile.write(resposta)

PORT = 8000
Handler = BondiaHTTPRequestHandler

with socketserver.TCPServer(("", PORT), Handler) as httpd:
    print "serving at port", PORT
    httpd.serve_forever()

```

3.2.3 Atendre les consultes

Per a atendre les peticions del client, els objectes `BaseHTTPRequestHandler` tenen els atributs i mètodes següents:

- L'atribut `path` conté la part de camí de la URL que ha sol·licitat el client
- L'atribut `headers` és un diccionari amb les capçaleres de la sol·licitud HTTP
- L'atribut `client_address` conté el tuple (host, port) provinent del servidor TCP
- El mètode `address_string()` retorna el nom que ha sol·licitat el client, el qual d'alguna forma és equivalent a `headers['Host']`.
- L'atribut `rfile` conté l'entrada del cos de la sol·licitud HTTP.

Com a exemple modifiquem l'objecte `BondiaHTTPRequestHandler` per tal que respongui amb el document web només quan se sol·licita per la URL / i amb un document d'error per a les altres URL:

```

class BondiaHTTPRequestHandler(http.server.BaseHTTPRequestHandler):

    def _head_html(self):
        self.send_response(200)
        self.send_header("Content-type", "text/html")

```

```

        self.end_headers()

def _head_error_404(self):
    self.send_error(404, "File not found")

def _bon_dia(self):

    self._head_html()

    resposta = """
    <html>
    <head>
    <title>Bon dia</title>
    </head>
    <body>
    <p>Bon dia</p>
    </body>
    </html>
    """

    self.wfile.write(resposta)

def do_GET(self):
    path = self.path
    host = self.headers['Host']

    print 'URL: {0} al servidor {1}'.format(path,host)

    if path == '/':
        self._bon_dia()
    else:
        self._head_error_404()

```

4 Feina a fer

TASCA PRÈVIA 1 [Llegir, provar i entendre la documentació d'aquesta pràctica.](#)

TASCA 2 Dissenyar i implementar un portal web que controli un led.

- Periòdicament (cada 2 segons) se sol·licita l'estat del led.
- Uns botons permeten encendre i apagar el led.

TASCA 3 Dissenyar i implementar la comunicació entre el portal web, el servidor web Python a mida i un perifèric.

TASCA 4 Comprovar el bon funcionament del portal web i el d'altres companys.

Depenent del maquinari que tingueu, podeu implementar la comunicació entre el servidor web Python i el perifèric d'una manera o altra.

- En el cas d'un Arduino, heu d'utilitzar connexió sèrie entre l'Arduino i el servidor web amb Python.

TASCA 5 Dissenyeu el protocol que creieu més adient.

- En el cas d'una RaspberryPi, aquesta pot executar el programa de servidor web amb Python i accedir directament als ports GPIO.

4.1 Interacció entre Servidors

El led s'ha de poder controlar tant des del servidor web Python fet a mida com des del servidor HTTP genèric fet amb Apache. D'aquesta manera pot interactuar amb una altra màquina directament (client HTTP), com amb un ésser humà a través del portal web fet amb el servidor generalista Apache.

S'ha d'acordar un protocol d'aplicació, implementat pel servidor a mida, per interactuar amb el led. Aquest protocol es pot dissenyar senzillament a partir del mètode GET d'HTTP i els query strings; és a dir que el servidor web actui segons l'adreça URL que se li sol·licita i per exemple retorni un '0' o '1' per indicar l'estat del led.

TASCA PRÈVIA 6 Dissenyeu el protocol per sobre de HTTP que farà servir el servidor a mida per controlar el led. Llisteu les URLs que us faran falta i definiu les accions que farà cada URL amb el led. Determineu quin serà el missatge de resposta a aquestes peticions.

4.2 Portal Web

A la pràctica de HTML i JavaScript va dissenyar una pàgina web amb objectes temporitzadors i XMLHttpRequest. A la pràctica sobre el servei web la va servir amb Apache.

TASCA 7 Modifiqueu ara el que calgui per usar les pràctiques anteriors com a portal web per al control del led.

Recordeu que XMLHttpRequest no pot demanar una URL que no sigui del mateix servidor web, per tant si és el cas haureu de muntar un reverse proxy que dirigeixi els XMLHttpRequest al servidor web Python.

4.3 Connexió sèrie amb l'Arduino

Ja heu treballat aquesta part en altres assignatures. Recordeu com establir-hi una connexió sèrie des de Python mitjançant el mòdul *pyserial* <https://pypi.org/project/pyserial/>

Per a connectar amb l'Arduino hi ha dos escenaris possibles:

- Virtualitzar el port sèrie i executar el servidor Python en una màquina virtual
- Executar el servidor Python en la màquina host i connectar directament al port sèrie

TASCA 8 Definiu el protocol de comunicació per port sèrie de l'Arduino per controlar el led.

5 Extensions

5.1 Atendre en multiprocés

Per a atendre més d'una petició HTTP alhora, cal que el servidor sigui multiprocés. Els objectes `http.server` són subclasses de `SocketServer.TCPServer` i com a tals no són multiprocés. Per a afegir multiprocés es pot crear un nou objecte que sigui subclasse de `http.server` i d'un servidor TCP multiprocés `ThreadingMixIn` o `ForkingMixIn`. A <https://docs.python.org/3/library/socketserver.html#asynchronous-mixins> teniu més informació.

Disposar d'una atenció multiprocés en el servidor web pot semblar que sempre sigui una cosa beneficiosa, però s'ha de tenir en compte que poden aparèixer efectes no desitjats com per exemple problemes de concurrència on els diferents threads / processos vulguin accedir a un únic recurs.

En el cas d'aquesta pràctica, el recurs que s'ha de compartir és la modificació del led. Si el servidor és un únic procés, diferents peticions quedaran encuades pel procés i el socket associat. Per tant s'aniran atenen una darrera l'altra. Aquest mecanisme, per si mateix, evita el conflicte de concurrència de l'accés al led.

En el cas d'haver fet el muntatge amb un servidor web estàndard amb multiprocés / multithread que executa scripts cgi per accedir al led, cal implementar un procés extra dedicat exclusivament a gestionar el led i atendre les peticions dels diferents cgi que es puguin executar en el mateix moment. Algun mecanisme per fer això seria fer cues de processos (pipes), entre d'altres.