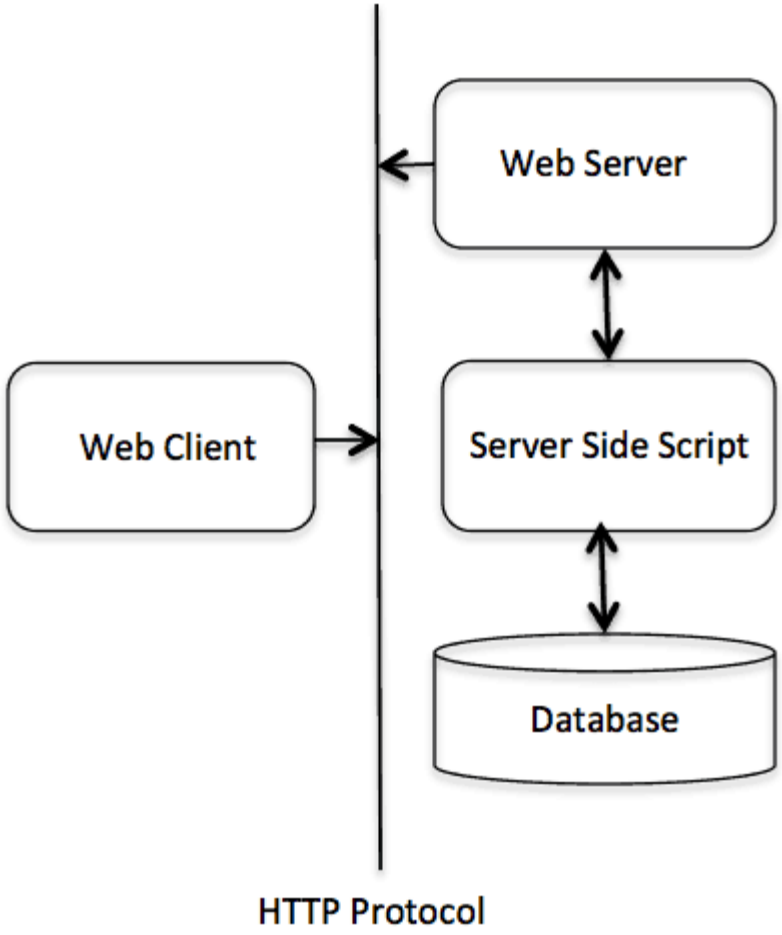
# What is CGI?

- The Common Gateway Interface (CGI)
  - is a set of standards that define how information is exchanged between the web server and a custom script.
  - is a standard for external gateway programs to interface with information servers such as HTTP servers.
- The current version is CGI/1.1 and CGI/1.2 is under progress.
- Web Browsing
  - Your browser contacts the HTTP web server and demands for the URL i.e., filename.
  - Web Server will parse the URL and will look for the filename in if it finds that file then sends it back to the browser, otherwise sends an error message indicating that you have requested a wrong file.

# CGI Architecture Diagram

– Web browser takes response from web server and displays either the received file or error message.



# Web Server Support & Configuration

- Please make sure that your Web Server supports CGI and it is configured to handle CGI Programs.
- All the CGI Programs to be executed by the HTTP server are kept in a pre-configured directory.
  - This directory is called CGI Directory and by convention it is named as /var/www/cgi-bin

```
<Directory "/var/www/cgi-bin">
   AllowOverride None
   Options ExecCGI
   Order allow,deny
   Allow from all
</Directory>

<Directory "/var/www/cgi-bin">
Options All
</Directory>
```

# First CGI Program

```
#!/usr/bin/python

print "Content-type:text/html\r\n\r\n"
print '<html>'
print '<head>'
print '<title>Hello Word - First CGI Program</title>'
print '</head>'
print '<body>'
print '<h2>Hello Word! This is my first CGI program</h2>'
print '</body>'
print '</html>'
```

- If you click hello.py, then this produces the following output:

Hello Word! This is my first CGI program

- There is one important and extra feature available which is first line to be printed **Content-type:text/html\r\n\r\n**.
- This line is sent back to the browser and specify the content type to be displayed on the browser screen.

# HTTP Header

- All the HTTP header will be in the following form:

```
HTTP Field Name: Field Content

For Example
Content-type: text/html\r\n\r\n
```

- There are few other important HTTP headers, which you will use frequently in your CGI Programming.

| Header | Description |
|---|---|
| Content-type: | A MIME string defining the format of the file being returned. Example is Content-type:text/html |
| Expires: Date | The date the information becomes invalid. This should be used by the browser to decide when a page needs to be refreshed. A valid date string should be in the format 01 Jan 1998 12:00:00 GMT. |
| Location: URL | The URL that should be returned instead of the URL requested. You can use this field to redirect a request to any file. |
| Last-modified: Date | The date of last modification of the resource. |
| Content-length: N | The length, in bytes, of the data being returned. The browser uses this value to report the estimated download time for a file. |
| Set-Cookie: String | Set the cookie passed through the *string* |

# CGI Environment Variables

- All the CGI program will have access to the following environment variables.

| Variable Name | Description |
|---|---|
| CONTENT_TYPE | The data type of the content. Used when the client is sending attached content to the server. For example, file upload, etc. |
| CONTENT_LENGTH | The length of the query information. It's available only for POST requests. |
| HTTP_COOKIE | Returns the set cookies in the form of key & value pair. |
| HTTP_USER_AGENT | The User-Agent request-header field contains information about the user agent originating the request. Its name of the web browser. |
| PATH_INFO | The path for the CGI script. |
| QUERY_STRING | The URL-encoded information that is sent with GET method request. |
| REMOTE_ADDR | The IP address of the remote host making the request. This can be useful for logging or for authentication purpose. |
| REMOTE_HOST | The fully qualified name of the host making the request. If this information is not available then REMOTE_ADDR can be used to get IR address. |
| REQUEST_METHOD | The method used to make the request. The most common methods are GET and POST. |
| SCRIPT_FILENAME | The full path to the CGI script. |
| SCRIPT_NAME | The name of the CGI script. |
| SERVER_NAME | The server's hostname or IP Address |
| SERVER_SOFTWARE | The name and version of the software the server is running. |

# Example

```
#!/usr/bin/python

import os

print "Content-type: text/html\r\n\r\n";
print "<font size=+1>Environment</font><\br>";
for param in os.environ.keys():
  print "<b>%20s</b>: %s<\br>" % (param, os.environ[param])
```

## GET and POST Methods

- You must have come across many situations when you need to pass some information from your browser to web server and ultimately to your CGI Program.
- Most frequently, browser uses two methods two pass this information to web server.
  - ✓ **GET**
  - ✓ **POST**

# Passing Information using GET Method

- The GET method sends the encoded user information appended to the page request.
- The page and the encoded information are separated by the **?** character as follows:
  - http://www.test.com/cgi-bin/hello.py?key1=value1&key2=value2
- The GET method is the default method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box.
- Never use GET method if you have password or other sensitive information to pass to the server.
- The GET method has size limitation: only 1024 characters can be sent in a request string.

# Passing Information using GET Method

- The GET method sends information using **QUERY_STRING** header and will be accessible in your CGI Program through **QUERY_STRING** environment variable.
- You can pass information by simply concatenating key and value pairs along with any URL or you can use HTML <FORM> tags to pass information using GET method.

Simple URL Example : Get Method

- Here is a simple URL, which will pass two values to hello_get.py program using GET method.
- /cgi-bin/hello_get.py?**first_name=ZARA**&**last_name=ALI**

# Example

- Below is **hello_get.py** script to handle input given by web browser. We are going to use **cgi** module, which makes it very easy to access passed information:

```
#!/usr/bin/python

# Import modules for CGI handling
import cgi, cgitb

# Create instance of FieldStorage
form = cgi.FieldStorage()

# Get data from fields
first_name = form.getvalue('first_name')
last_name  = form.getvalue('last_name')

print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Hello - Second CGI Program</title>"
print "</head>"
print "<body>"
print "<h2>Hello %s %s</h2>" % (first_name, last_name)
print "</body>"
print "</html>"
```

Hello ZARA ALI

# Simple FORM Example: GET Method

- Here is a simple example which passes two values using HTML FORM and submit button. We are going to use same CGI script hello_get.py to handle this input.

```
<form action="/cgi-bin/hello_get.py" method="get">
First Name: <input type="text" name="first_name">  <br />

Last Name: <input type="text" name="last_name" />
<input type="submit" value="Submit" />
</form>
```

- Here is the actual output of the above form. You enter First and Last Name and then click submit button to see the result.

# POST Method

- This packages the information in exactly the same way as GET methods, but instead of sending it as a text string after a **?** in the URL it sends it as a separate message.
- This message comes into the CGI script in the form of the standard input.
- Below is same hello_get.py script which handles GET as well as POST method.

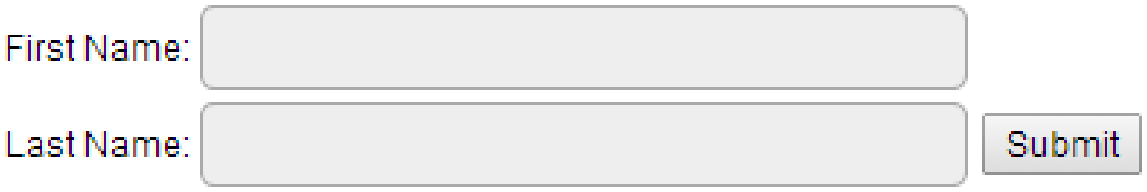
```
#!/usr/bin/python

# Import modules for CGI handling
import cgi, cgitb

# Create instance of FieldStorage
form = cgi.FieldStorage()

# Get data from fields
first_name = form.getvalue('first_name')
last_name  = form.getvalue('last_name')

print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Hello - Second CGI Program</title>"
print "</head>"
print "<body>"
print "<h2>Hello %s %s</h2>" % (first_name, last_name)
print "</body>"
print "</html>"
```

# POST Method

- Let us take again same example as above which passes two values using HTML FORM and submit button.
- We are going to use same CGI script hello_get.py to handle this input.

```
<form action="/cgi-bin/hello_get.py" method="post">
First Name: <input type="text" name="first_name"><br />
Last Name: <input type="text" name="last_name" />

<input type="submit" value="Submit" />
</form>
```

- Here is the actual output of the above form. You enter First and Last Name and then click submit button to see the result.

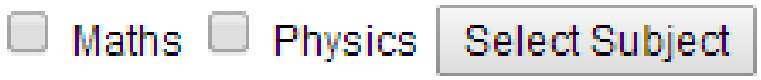


# Passing Checkbox Data to CGI Program

- Checkboxes are used when more than one option is required to be selected.
- Here is example HTML code for a form with two checkboxes:

```
<form action="/cgi-bin/checkbox.cgi" method="POST" target="_blank">
<input type="checkbox" name="maths" value="on" /> Maths
<input type="checkbox" name="physics" value="on" /> Physics
<input type="submit" value="Select Subject" />
</form>
```

- The result of this code is the following form:




- Below is checkbox.cgi script to handle input given by web browser for checkbox button.

# Checkbox Data

```
#!/usr/bin/python

# Import modules for CGI handling
import cgi, cgitb

# Create instance of FieldStorage
form = cgi.FieldStorage()

# Get data from fields
if form.getvalue('maths'):
   math_flag = "ON"
else:
   math_flag = "OFF"

if form.getvalue('physics'):
   physics_flag = "ON"
else:
   physics_flag = "OFF"

print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Checkbox - Third CGI Program</title>"
print "</head>"
print "<body>"
print "<h2> CheckBox Maths is : %s</h2>" % math_flag
print "<h2> CheckBox Physics is : %s</h2>" % physics_flag
print "</body>"
print "</html>"
```

# Using Cookies in CGI

- HTTP protocol is a stateless protocol.
  - But for a commercial website, it is required to maintain session information among different pages.
- For example, one user registration ends after completing many pages.
  - But how to maintain user's session information across all the web pages.
- In many situations, using **cookies** is the most efficient method of remembering and tracking preferences, purchases, commissions, and other information required for better visitor experience or site statistics.

# Using Cookies in CGI

- Your server sends some data to the visitor's browser in the form of a cookie.
- The browser may accept the cookie. If it does, it is stored as a plain text record on the visitor's hard drive.
- Now, when the visitor arrives at another page on your site, the cookie is available for retrieval.
- Once retrieved, your server knows/remembers what was stored.
- Cookies are a plain text data record of 5 variable-length fields:

1. **Expires :** The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.
2. **Domain :** The domain name of your site.
3. **Path :** The path to the directory or web page that sets the cookie. This may be blank if you want to retrieve the cookie from any directory or page.
4. **Secure :** If this field contains the word "secure", then the cookie may only be retrieved with a secure server. If this field is blank, no such restriction exists.
5. **Name=Value :** Cookies are set and retrieved in the form of key and value pairs.

# Setting up Cookies

- It is very easy to send cookies to browser.
- These cookies will be sent along with HTTP Header before to Content-type field.
- Assuming you want to set UserID and Password as cookies. So cookies setting will be done as follows:

```
#!/usr/bin/python

print "Set-Cookie:UserID=XYZ;\r\n"
print "Set-Cookie:Password=XYZ123;\r\n"
print "Set-Cookie:Expires=Tuesday, 31-Dec-2007 23:12:40 GMT";\r\n"
print "Set-Cookie:Domain=www.tutorialspoint.com;\r\n"
print "Set-Cookie:Path=/perl;\n"
print "Content-type:text/html\r\n\r\n"
.....Rest of the HTML Content....
```

- We use **Set-Cookie** HTTP header to set cookies.
- Here, it is optional to set cookies attributes like Expires, Domain and Path. It is notable that cookies are set before sending magic line **"Content-type:text/html\r\n\r\n**.

# Retrieving Cookies

- Cookies are stored in CGI environment variable HTTP_COOKIE and they will have following form:
  - **key1=value1;key2=value2;key3=value3**....
- Here is an example of how to retrieve cookies.

```
#!/usr/bin/python

# Import modules for CGI handling
from os import environ
import cgi, cgitb

if environ.has_key('HTTP_COOKIE'):
   for cookie in map(strip, split(environ['HTTP_COOKIE'], ';')):
      (key, value ) = split(cookie, '=');
      if key == "UserID":
         user_id = value

      if key == "Password":
         password = value

print "User ID  = %s" % user_id
print "Password = %s" % password
```

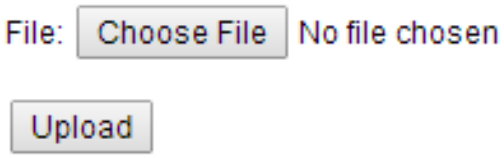
```
User ID = XYZ
Password = XYZ123
```

# File Upload Example

- To upload a file, the HTML form must have the **enctype** attribute set to **multipart/form-data**.
- The input tag with the file type will create a "Browse" button.

```
<html>
<body>
   <form enctype="multipart/form-data"
                     action="save_file.py" method="post">
   <p>File: <input type="file" name="filename" /></p>
   <p><input type="submit" value="Upload" /></p>
   </form>
</body>
</html>
```

- The result of this code is the following form:



# Example

- Here is the script **save_file.py** to handle file upload:

```
#!/usr/bin/python

import cgi, os
import cgitb; cgitb.enable()

form = cgi.FieldStorage()

# Get filename here.
fileitem = form['filename']

# Test if the file was uploaded
if fileitem.filename:
   # strip leading path from file name to avoid
   # directory traversal attacks
   fn = os.path.basename(fileitem.filename)
   open('/tmp/' + fn, 'wb').write(fileitem.file.read())

   message = 'The file "' + fn + '" was uploaded successfully'

else:
   message = 'No file was uploaded'

print """\
Content-Type: text/html\n
<html>
<body>
   <p>%s</p>
</body>
</html>
""" % (message,)
```

If you are running above script on Unix/Linux, then you would have to take care of replacing file separator as follows, otherwise on your windows machine above open() statement should work fine.

```
fn = os.path.basename(fileitem.filename.replace("\\", "/" ))
```

# How To Raise a "File Download" Dialog Box ?

- A user will click a link and it will pop up a "File Download" dialogue box to the user instead of displaying actual content.
- This is very easy and will be achieved through HTTP header. This HTTP header will be different from the header mentioned in previous section.
- For example, if you want make a **FileName** file downloadable from a given link, then its syntax will be as follows:

```
#!/usr/bin/python

# HTTP Header
print "Content-Type:application/octet-stream; name=\"FileName\"\r\n";
print "Content-Disposition: attachment; filename=\"FileName\"\r\n\n";

# Actual File Content will go hear.
fo = open("foo.txt", "rb")

str = fo.read();
print str

# Close opend file
fo.close()
```

# Thread

- Running several threads is similar to running several different programs concurrently, but with the following benefits:

- Multiple threads within a process share the same data space with the main thread and can therefore share information or communicate with each other more easily than if they were separate processes.

- Threads sometimes called light-weight processes and they do not require much memory overhead; they care cheaper than processes.

# Thread

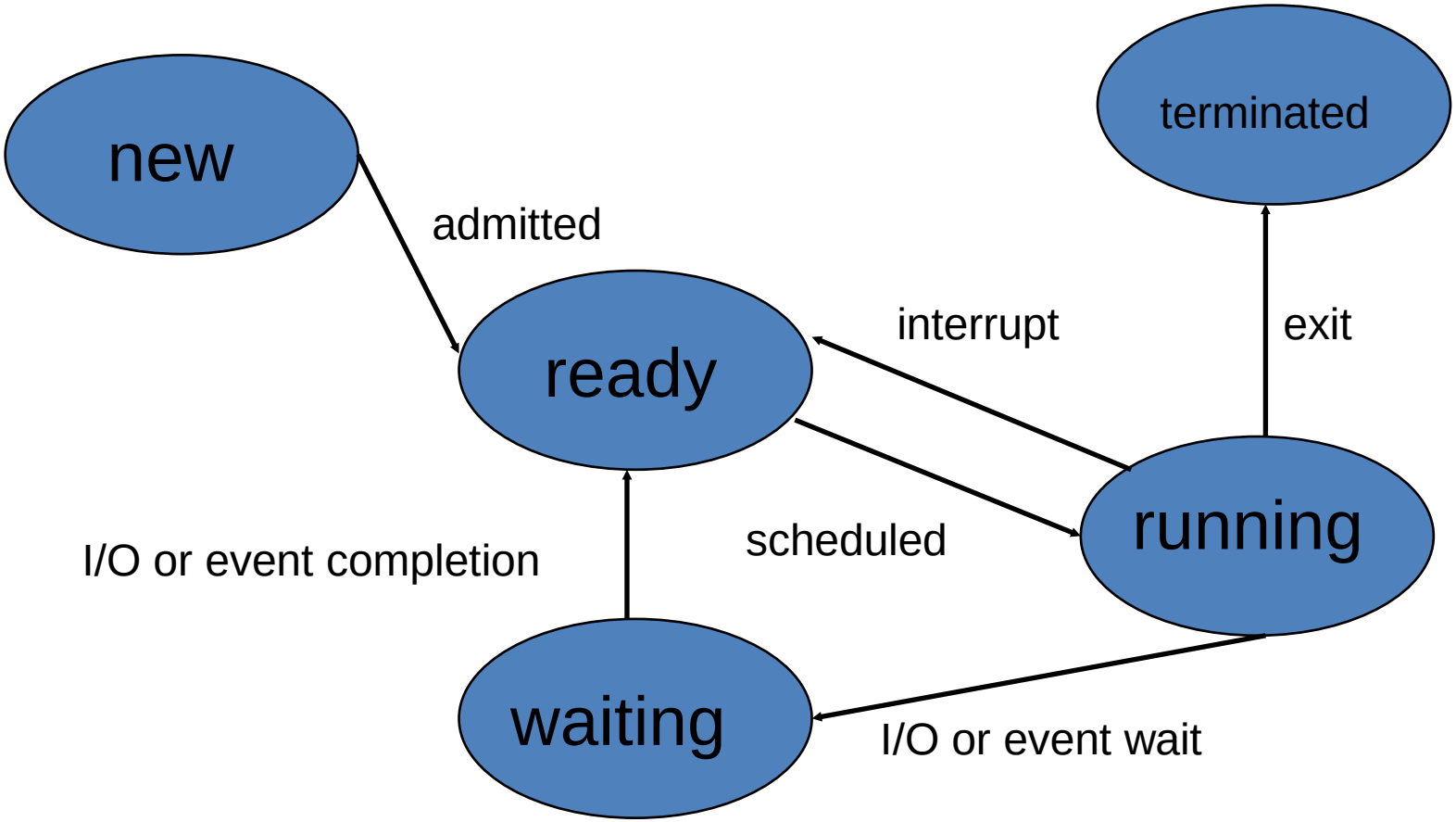
- A thread has a beginning, an execution sequence, and a conclusion.
- It has an instruction pointer that keeps track of where within its context it is currently running.
- It can be pre-empted (interrupted)
- It can temporarily be put on hold (also known as sleeping) while other threads are running - this is called yielding.

# Processes

- Process
  - A Basic Unit of Work from the Viewpoint of OS
  - Types:
    - Sequential processes: an activity resulted from the execution of a program by a processor
    - Multi-thread processes
  - An Active Entity
    - Program Code – A Passive Entity
    - Stack and Data Segments
  - The Current Activity
    - PC, Registers, Contents in the Stack and Data Segments

# Processes

- Process State

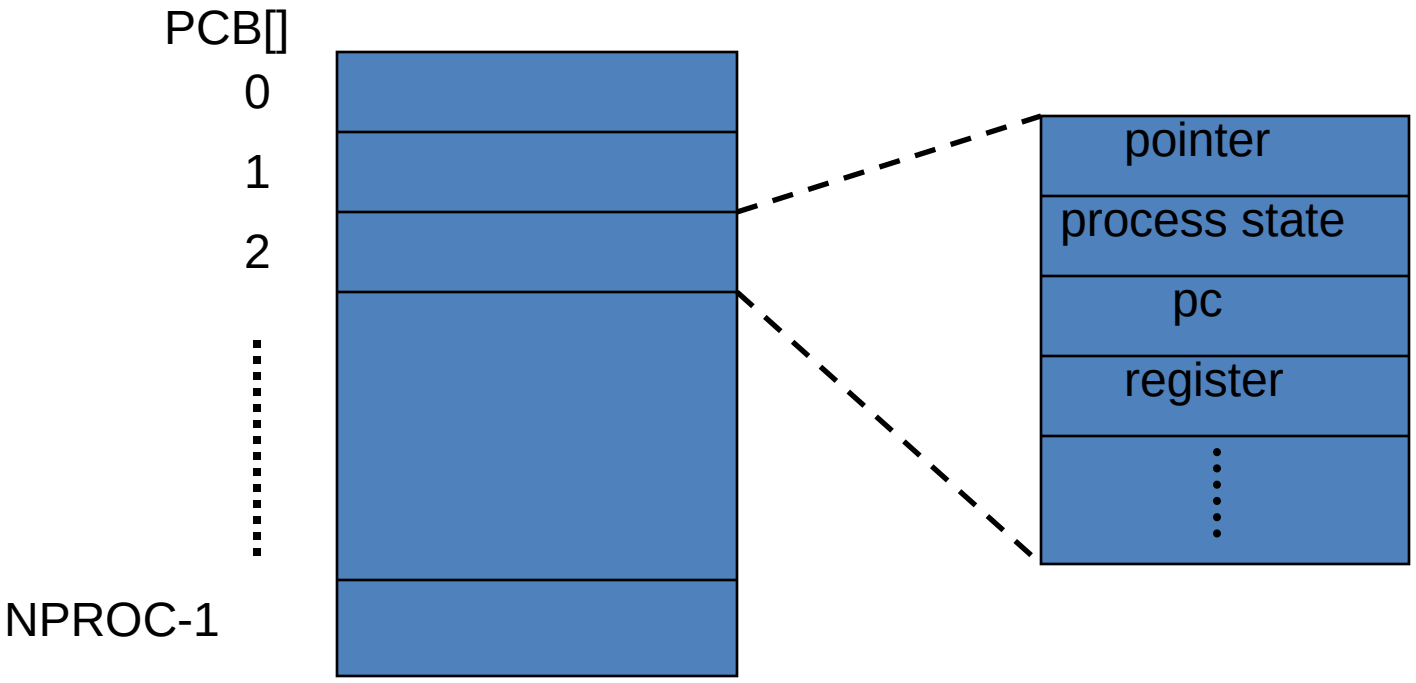


# Processes

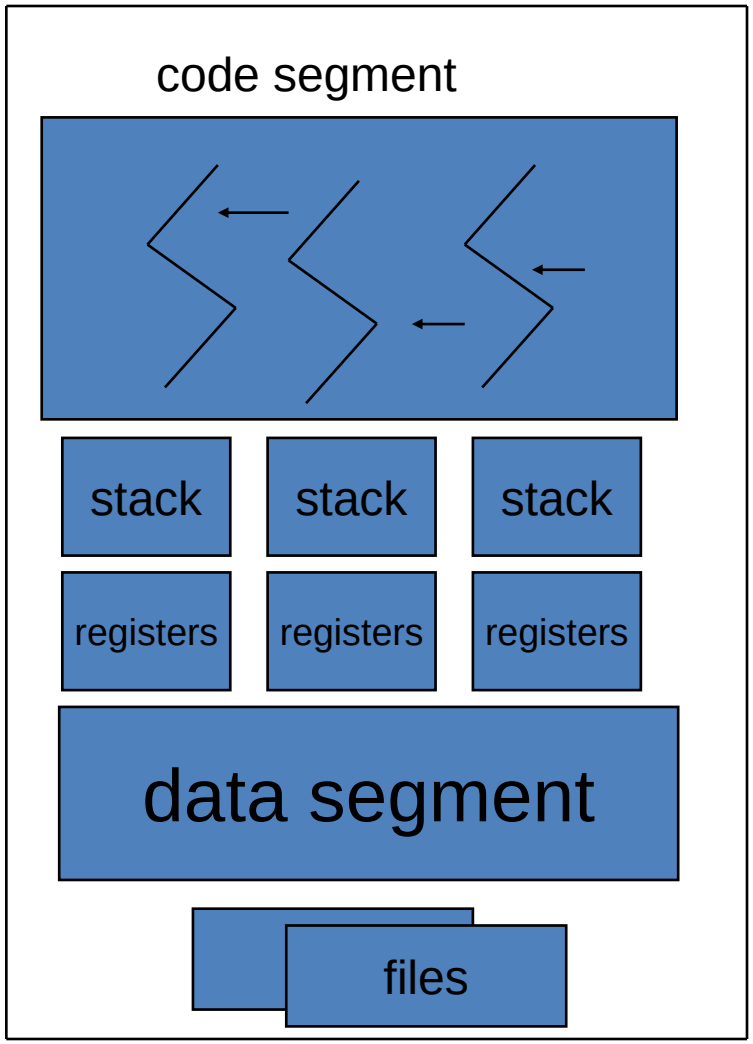
- Process Control Block (PCB)
  - Process State
  - Program Counter
  - CPU Registers
  - CPU Scheduling Information
  - Memory Management Information
  - Accounting Information
  - I/O Status Information

# Processes

- PCB: The repository for any information that may vary from process to process



# Threads

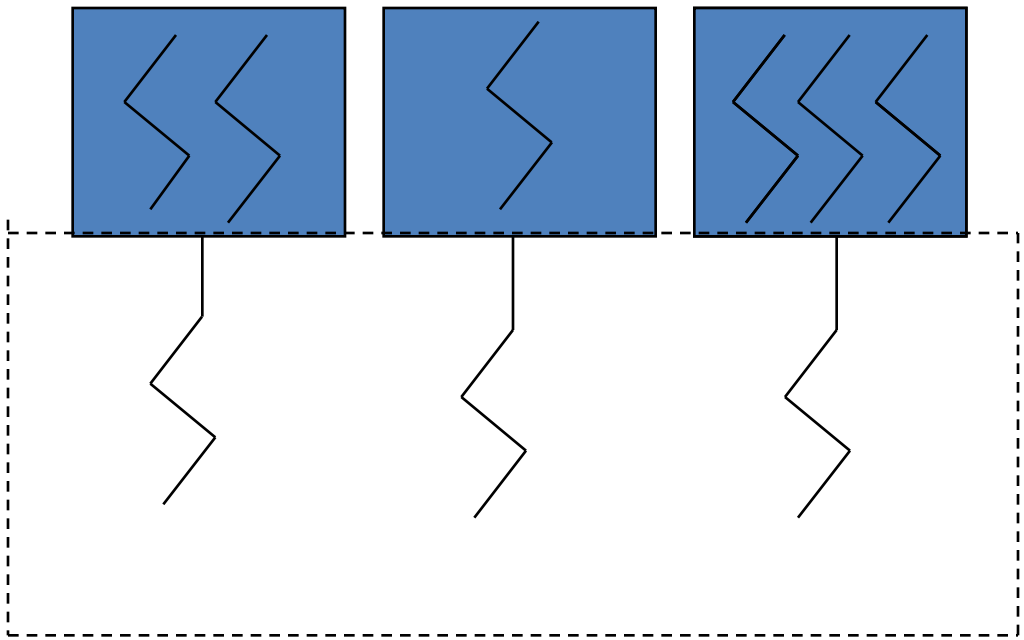



- Motivation
  - A web browser
    - Data retrieval
    - Text/image displaying
  - A word processor
    - Displaying
    - Keystroke reading
    - Spelling and grammar checking
  - A web server
    - Clients' services
    - Request listening

# Threads

- Benefits
  - Responsiveness
  - Resource Sharing
  - Economy
    - Creation and context switching
      - 30 times slower in process creation in Solaris 2
      - 5 times slower in process context switching in Solaris 2
  - Utilization of Multiprocessor Architectures

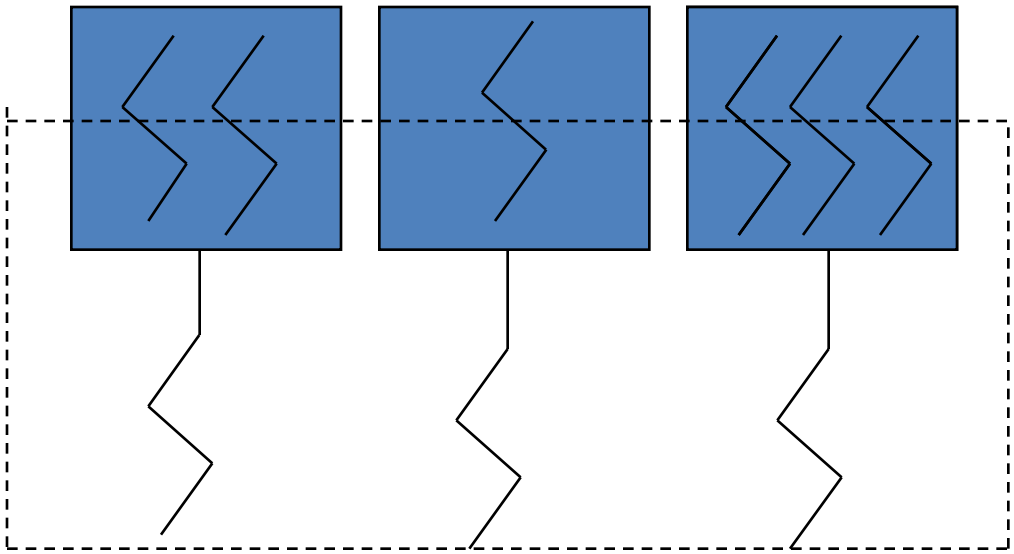
# User-Level Threads



- User-level threads are implemented by a thread library at the user level.
- Examples:
  - POSIX Pthreads, Mach C-threads, Solaris 2 UI-threads

- Advantages
  - Context switching among them is extremely fast
- Disadvantages
  - Blocking of a thread in executing a system call can block the entire process.

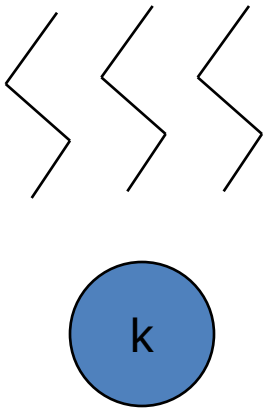
# Kernel-Level Threads



- Kernel-level threads are provided a set of system calls similar to those of processes
- Examples
  - Windows 2000, Solaris 2, True64UNIX

- Advantage
  - Blocking of a thread will not block its entire task.
- Disadvantage
  - Context switching cost is a little bit higher because the kernel must do the switching.

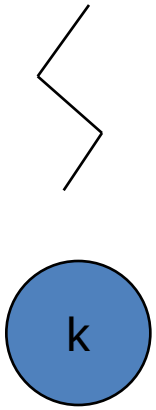
# Multithreading Models



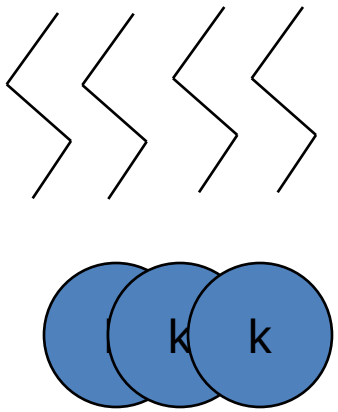

- Many-to-One Model
  - Many user-level threads to one kernel thread
  - Advantage:
    - Efficiency
  - Disadvantage:
    - One blocking system call blocks all.
    - No parallelism for multiple processors
  - Example: Green threads for Solaris 2

# Multithreading Models

- One-to-One Model
  - One user-level thread to one kernel thread
  - Advantage: One system call blocks one thread.
  - Disadvantage: Overheads in creating a kernel thread.
  - Example: Windows NT, Windows 2000, OS/2



# Multithreading Models




- Many-to-Many Model
  - Many user-level threads to many kernel threads
  - Advantage:
    - A combination of parallelism and efficiency
  - Example: Solaris 2, IRIX, HP-UX,Tru64 UNIX

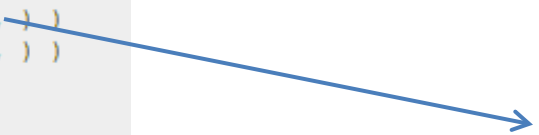
# Starting a New Thread

- To spawn another thread, you need to call following method available in *thread* module:
  - thread.start_new_thread ( function, args[, kwargs] )
- This method call enables a fast and efficient way to create new threads in both Linux and Windows.
- The method call returns immediately and the child thread starts and calls function with the passed list of *agrs*.
- When function returns, the thread terminates.
- Here, *args* is a tuple of arguments; use an empty tuple to call function without passing any arguments.
- *kwargs* is an optional dictionary of keyword arguments.

# EXAMPLE

```
#!/usr/bin/python

import thread
import time

# Define a function for the thread
def print_time( threadName, delay):
   count = 0
   while count < 5:
      time.sleep(delay)
      count += 1
      print "%s: %s" % ( threadName, time.ctime(time.time()) )

# Create two threads as follows
try:
   thread.start_new_thread( print_time, ("Thread-1", 2, ) )
   thread.start_new_thread( print_time, ("Thread-2", 4, ) )
except:
   print "Error: unable to start thread"

while 1:
   pass
```

```
Thread-1: Thu Aug 21 09:54:08 2014
Thread-2: Thu Aug 21 09:54:10 2014
Thread-1: Thu Aug 21 09:54:10 2014
Thread-1: Thu Aug 21 09:54:12 2014
Thread-2: Thu Aug 21 09:54:14 2014
Thread-1: Thu Aug 21 09:54:14 2014
Thread-1: Thu Aug 21 09:54:16 2014
Thread-2: Thu Aug 21 09:54:18 2014
Thread-2: Thu Aug 21 09:54:22 2014
Thread-2: Thu Aug 21 09:54:26 2014
```




- Although it is very effective for low-level threading, but the *thread* module is very limited compared to the newer threading module.

# The *Threading* Module

- The newer threading module included with Python 2.4 provides much more powerful, high-level support for threads than the *thread* module discussed in the previous section.
- The threading module exposes all the methods of the *thread* module and provides some additional methods:
- **threading.activeCount():** Returns the number of thread objects that are active.
- **threading.currentThread():** Returns the number of thread objects in the caller's thread control.
- **threading.enumerate():** Returns a list of all thread objects that are currently active.

# The *Threading* Module

- In addition to the methods, the ***threading*** module has the *Thread* class that implements threading.
- The methods provided by the *Thread* class are as follows:
  - **run():** The run() method is the entry point for a thread.
  - **start():** The start() method starts a thread by calling the run method.
  - **join([time]):** The join() waits for threads to terminate.
  - **isAlive():** The isAlive() method checks whether a thread is still executing.
  - **getName():** The getName() method returns the name of a thread.
  - **setName():** The setName() method sets the name of a thread.

# Creating Thread using *Threading* Module

- To implement a new thread using the threading module, you have to do the following:

1. Define a new subclass of the *Thread* class.
2. Override the *__init__(self [,args])* method to add additional arguments.
3. Then, override the run(self [,args]) method to implement what the thread should do when started.
4. Once you have created the new *Thread* subclass, you can create an instance of it and then start a new thread by invoking the *start()*, which will in turn call *run()* method.

# EXAMPLE

```
#!/usr/bin/python

import threading
import time

exitFlag = 0

class myThread (threading.Thread):
    def __init__(self, threadID, name, counter):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.counter = counter
    def run(self):
        print "Starting " + self.name
        print_time(self.name, self.counter, 5)
        print "Exiting " + self.name

def print_time(threadName, delay, counter):
    while counter:
        if exitFlag:
            thread.exit()
        time.sleep(delay)
        print "%s: %s" % (threadName, time.ctime(time.time()))
        counter -= 1

# Create new threads
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)

# Start new Threads
thread1.start()
thread2.start()

print "Exiting Main Thread"
```

```
Starting Thread-1Starting Thread-2Exiting Main Thread


>>> Thread-1: Thu Aug 21 09:52:58 2014
Thread-2: Thu Aug 21 09:52:59 2014
Thread-1: Thu Aug 21 09:52:59 2014
Thread-1: Thu Aug 21 09:53:00 2014
Thread-2: Thu Aug 21 09:53:01 2014
Thread-1: Thu Aug 21 09:53:01 2014
Thread-1: Thu Aug 21 09:53:02 2014
Exiting Thread-1
Thread-2: Thu Aug 21 09:53:03 2014
Thread-2: Thu Aug 21 09:53:05 2014
Thread-2: Thu Aug 21 09:53:07 2014
Exiting Thread-2
```

# Process Synchronization

- Why Synchronization?
  - To ensure data consistency for concurrent access to shared data!
- Contents:
  - Various mechanisms to ensure the orderly execution of cooperating processes

# Process Synchronization

## – A Consumer-Producer Example

- Producer

```
while (1) {
    while (counter == BUFFER_SIZE)
        ;
    produce an item in nextp;
    ….
    buffer[in] = nextp;
    in = (in+1) % BUFFER_SIZE;
    counter++;
}
```

- Consumer:

```
while (1) {
    while (counter == 0)
        ;
    nextc = buffer[out];
    out = (out +1) % BUFFER_SIZE;
    counter--;
    consume an item in nextc;
}
```

# Process Synchronization

- counter++ vs counter—

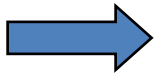
  r1 = counter  r2 = counter

  r1 = r1 + 1  r2 = r2 - 1

  counter = r1  counter = r2

- Initially, let counter = 5.

1. P: r1 = counter
2. P: r1 = r1 + 1
3. C: r2 = counter
4. C: r2 = r2 – 1
5. P: counter = r1
6. C: counter = r2



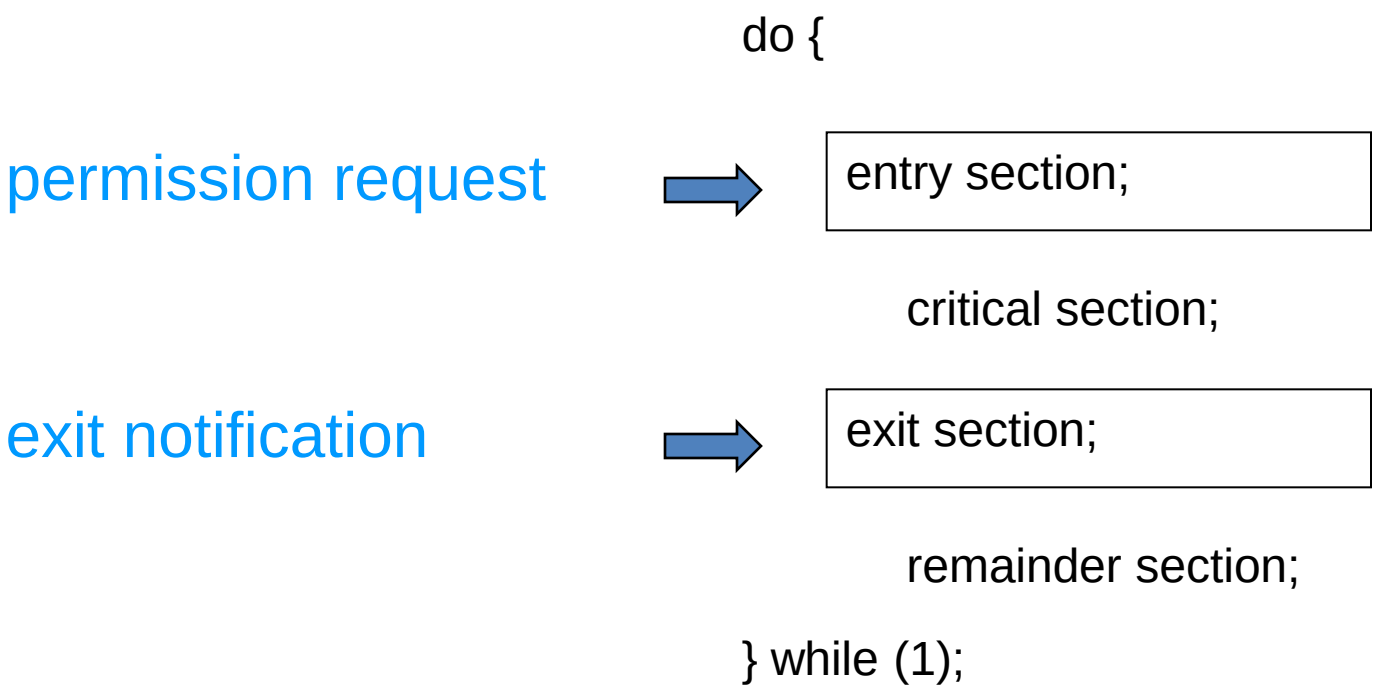
A Race Condition!

# Process Synchronization

- A Race Condition:
  - A situation where the outcome of the execution depends on the particular order of process scheduling.
- The Critical-Section Problem:
  - Design a protocol that processes can use to cooperate.
    - Each process has a segment of code, called a critical section, whose execution must be mutually exclusive.

# Process Synchronization

- A General Structure for the Critical-Section Problem



# The Critical-Section Problem

- Three Requirements

1. Mutual Exclusion
   a. Only one process can be in its critical section.
2. Progress
   a. Only processes not in their remainder section can decide which will enter its critical section.
   b. The selection cannot be postponed indefinitely.
3. Bounded Waiting
   a. A waiting process only waits for a bounded number of processes to enter their critical sections.

# Synchronizing Threads

- The threading module provided with Python includes a simple-to-implement locking mechanism that will allow you to synchronize threads.

- A new lock is created by calling the *Lock()* method, which returns the new lock.

- The *acquire(blocking)* method of the new lock object would be used to force threads to run synchronously.

- The optional *blocking* parameter enables you to control whether the thread will wait to acquire the lock.

# Synchronizing Threads

- If *blocking* is set to 0, the thread will return immediately with a 0 value if the lock cannot be acquired and with a 1 if the lock was acquired.

- If blocking is set to 1, the thread will block and wait for the lock to be released.

- The *release()* method of the new lock object would be used to release the lock when it is no longer required.

# MPLE

```
#!/usr/bin/python

import threading
import time

class myThread (threading.Thread):
    def __init__(self, threadID, name, counter):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.counter = counter
    def run(self):
        print "Starting " + self.name
        # Get lock to synchronize threads
        threadLock.acquire()
        print_time(self.name, self.counter, 3)
        # Free lock to release next thread
        threadLock.release()

def print_time(threadName, delay, counter):
    while counter:
        time.sleep(delay)
        print "%s: %s" % (threadName, time.ctime(time.time()))
        counter -= 1

threadLock = threading.Lock()
threads = []

# Create new threads
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)

# Start new Threads
thread1.start()
thread2.start()

# Add threads to thread list
threads.append(thread1)
threads.append(thread2)

# Wait for all threads to complete
for t in threads:
    t.join()
print "Exiting Main Thread"
```

```
Starting Thread-1Starting Thread-2

Thread-1: Thu Aug 21 09:50:43 2014
Thread-1: Thu Aug 21 09:50:44 2014
Thread-1: Thu Aug 21 09:50:45 2014
Thread-2: Thu Aug 21 09:50:47 2014
Thread-2: Thu Aug 21 09:50:49 2014
Thread-2: Thu Aug 21 09:50:51 2014
Exiting Main Thread
```

# Multithreaded Priority Queue

- The *Queue* module allows you to create a new queue object that can hold a specific number of items.
- There are following methods to control the Queue:
  - **get():** The get() removes and returns an item from the queue.
  - **put():** The put adds item to a queue.
  - **qsize() :** The qsize() returns the number of items that are currently in the queue.
  - **empty():** The empty( ) returns True if queue is empty; otherwise, False.
  - **full():** the full() returns True if queue is full; otherwise, False.

# Example

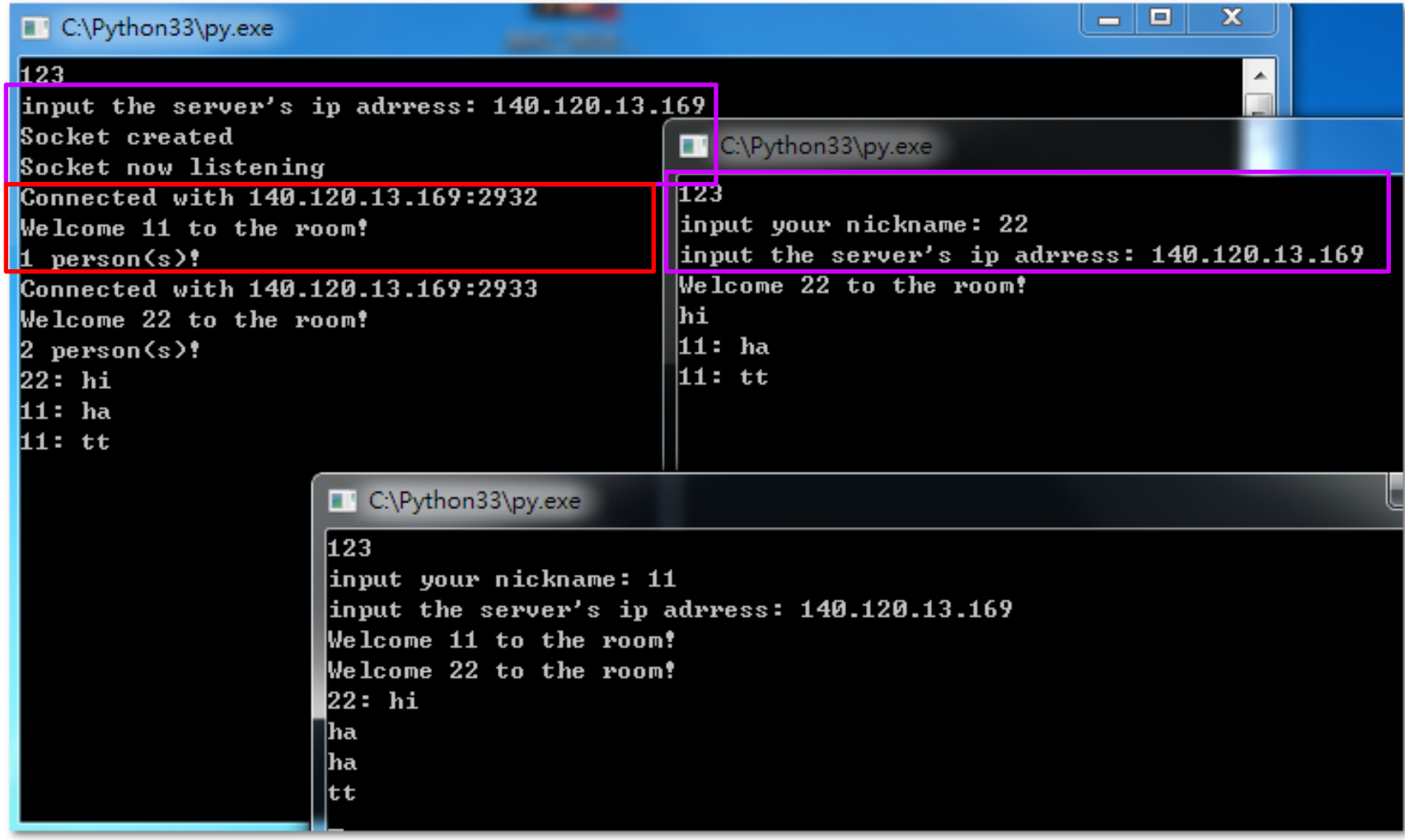
```
import threading
import time

exitFlag = 0

class myThread (threading.Thread):
    def __init__(self, threadID, name, q):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.q = q
    def run(self):
        print "Starting " + self.name
        process_data(self.name, self.q)
        print "Exiting " + self.name

def process_data(threadName, q):
    while not exitFlag:
        queueLock.acquire()
        if not workQueue.empty():
            data = q.get()
            queueLock.release()
            print "%s processing %s" % (threadName, data)
        else:
            queueLock.release()
        time.sleep(1)

threadList = ["Thread-1", "Thread-2", "Thread-3"]
nameList = ["One", "Two", "Three", "Four", "Five"]
queueLock = threading.Lock()
workQueue = Queue.Queue(10)
threads = []
threadID = 1

# Create new threads
for tName in threadList:
    thread = myThread(threadID, tName, workQueue)
    thread.start()
    threads.append(thread)
    threadID += 1

# Fill the queue
queueLock.acquire()
for word in nameList:
    workQueue.put(word)
queueLock.release()

# Wait for queue to empty
```

```
Starting Thread-1Starting Thread-2Starting Thread-3


Thread-1 processing OneThread-2 processing TwoThread-3 processing Three


Thread-1 processing FourThread-2 processing Five

Exiting Thread-3
 Exiting Thread-2Exiting Thread-1

Exiting Main Thread
```

# Example

```
# Wait for queue to empty
while not workQueue.empty():
    pass

# Notify threads it's time to exit
exitFlag = 1

# Wait for all threads to complete
for t in threads:
    t.join()
print "Exiting Main Thread"
```

# Talking Room (Console)



# Server (1)

```
import socket
import sys
import threading

con = threading.Condition()
HOST = raw_input("input the server's ip adrress: ")     # Symbolic name meaning all available interfaces
PORT = 8888     # Arbitrary non-privileged port
data = ''

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print 'Socket created'
s.bind((HOST, PORT))
s.listen(10)
print 'Socket now listening'

#Function for handling connections. This will be used to create threads
def clientThreadIn(conn, nick):
    global data
#infinite loop so that function do not terminate and thread do not end.
    while True:
    #Receiving from client
        try:
            temp = conn.recv(1024)
            if not temp:
                conn.close()
                return
            NotifyAll(temp)
            print data
        except:
            NotifyAll(nick + " leaves the room!")
            print data
            return

        #came out of loop
```

# Server (2)

```
def NotifyAll(sss):
    global data
    if con.acquire():
        data = sss
        con.notifyAll()
        con.release()

def ClientThreadOut(conn, nick):
    global data
    while True:
        if con.acquire():
            con.wait()
            if data:
                try:
                    conn.send(data)
                    con.release()
                except:
                    con.release()
                    return


while 1:
    #wait to accept a connection - blocking call
    conn, addr = s.accept()
    print 'Connected with ' + addr[0] + ':' + str(addr[1])
    nick = conn.recv(1024)
     #send only takes string
    #start new thread takes 1st argument as a function name to be run, second is the tuple of arguments to th
    NotifyAll('Welcome ' + nick + ' to the room!')
    print data
    print str((threading.activeCount() + 1) / 2) + ' person(s)!'
    conn.send(data)
    threading.Thread(target = clientThreadIn , args = (conn, nick)).start()
    threading.Thread(target = ClientThreadOut , args = (conn, nick)).start()

s.close()
raw_input( )
```

# threading.Condition

- This is a synchronization mechanism where a thread waits for a specific condition and another thread signals that this condition has happened.

- Once the condition happened, the thread acquires the lock to get exclusive access to the shared resource.

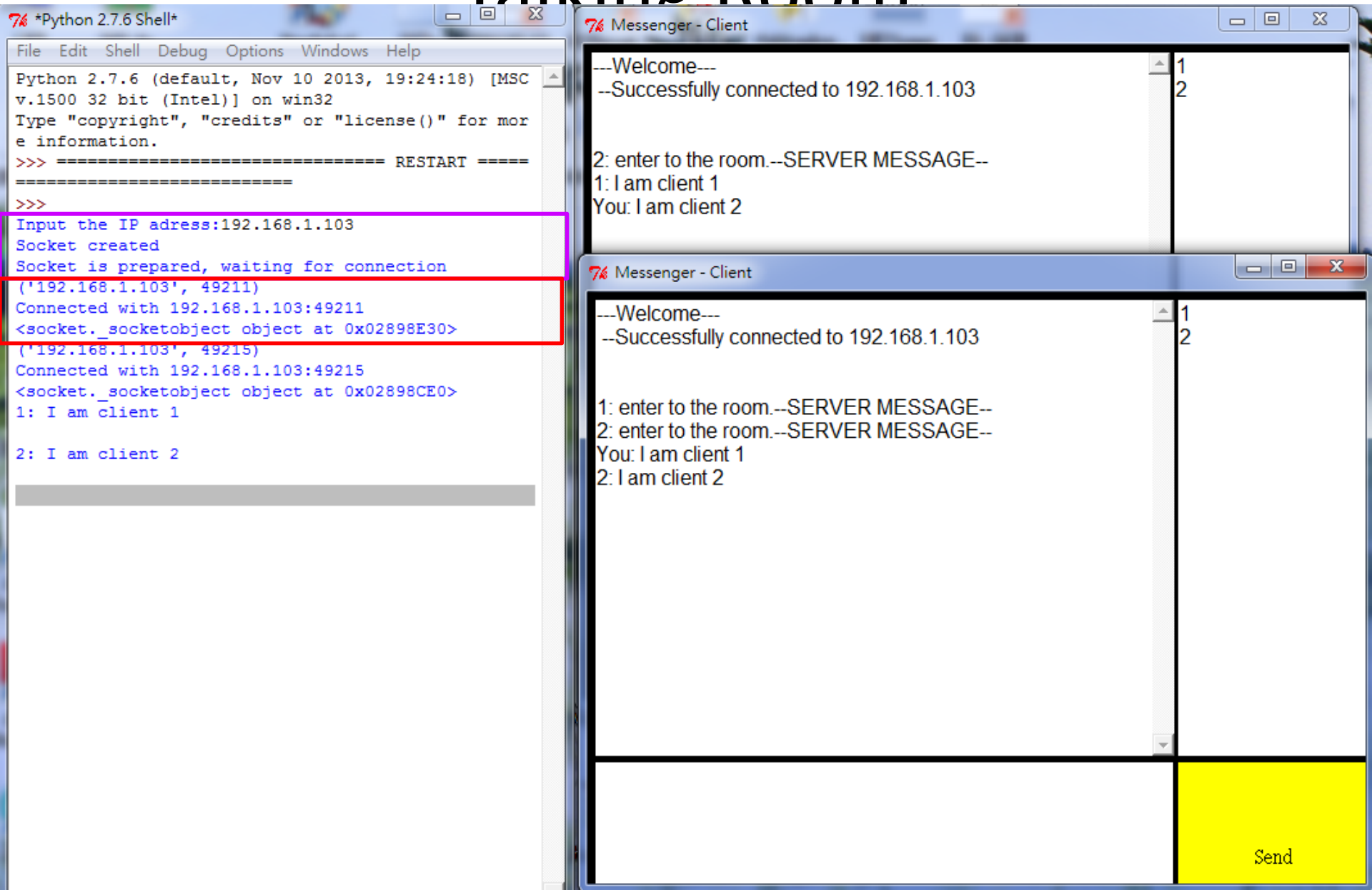
# Client (1)

```
import socket
import threading


inString = ''
outString = ''
nick = ''

def DealOut(s):
    global nick, outString
    while True:
        outString = raw_input()
        outString = nick + ': ' + outString
        s.send(outString)

def DealIn(s):
    global inString
    while True:
        try:
            inString = s.recv(1024)
            if not inString:
                break
            if outString != inString:
                print inString
        except:
            break
```

# Client (2)

```
nick = raw_input("input your nickname: ")
ip = raw_input("input the server's ip adrress: ")
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((ip, 8888))
sock.send(nick)

thin = threading.Thread(target = DealIn, args = (sock,))
thin.start()
thout = threading.Thread(target = DealOut, args = (sock,))
thout.start()

#sock.close()
raw_input( )
```

# Talking Room



project_server.pyw - C:\Python27\project_server.pyw

File Edit Format Run Options Windows Help

```
import socket
import threading

HOST = raw_input("Input the IP adress:")
#HOST= "140.120.221.139"
PORT = 8888
data = ''
outc = threading.Condition()

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print 'Socket created'
s.bind((HOST, PORT))
s.listen(10)
print 'Socket is prepared, waiting for connection'

def incomemessage(acc,nickname):

    global data
    while True:
        try:
            mess = acc.recv(1024)
            if not mess:
                acc.close()
                return
            sendtoall(mess)
            print mess
        except:
            sendtoall(nickname +':' + ' left the room.--SERVER MESSAGE--\n' )
            print data
            return

def sendtoall(dej):
    global data
    if outc.acquire():
        data=dej
        outc.notifyAll()
        outc.release()
```

# Server (2)

```
def outcomemessage(acc):
    global data
    while True:
        if outc.acquire():
            outc.wait()
            if data:
                try:
                    acc.send(data)
                    outc.release()
                except:
                    outc.release()
                    return


while True:
    acc, info = s.accept()
    print info
    print 'Connected with ' + info[0] + ':' + str(info[1])
    nickname = acc.recv(1024)
    sendtoall(nickname +':' + ' enter to the room.--SERVER MESSAGE--\n')
    #print data
    #data=acc.recv()
    print acc
    acc.send(data)
    threading.Thread(target = incomemessage , args = (acc,nickname)).start()
    threading.Thread(target = outcomemessage , args = (acc,)).start()
    #s.sendall("ahoj")

s.close()
raw_input( )
```

project_client.pyw - C:\Python27\project_client.pyw

File Edit Format Run Options Windows Help

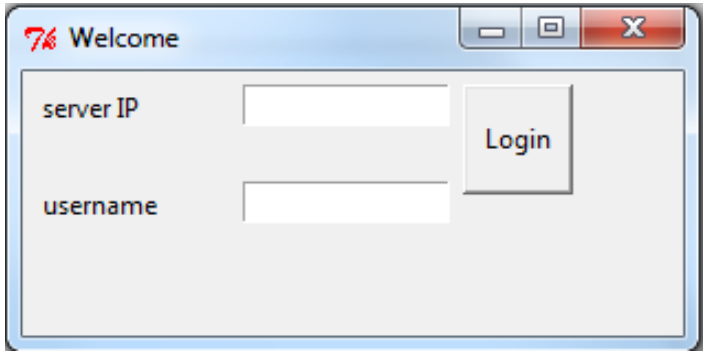
```
import socket
from Tkinter import *
import threading

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mymessage=''
import string

users=[]

def adduser(message):
    global users

    ldata = string.split(message, ':')
    problem = ' left the room.--SERVER MESSAGE--\n'
    if  ldata[0] not in users:
        users.append(ldata[0])
        users.sort()
        seeuser.delete("0.0",END)
        for us in users:
            tisk=us+'\n'
            seeuser.insert(END,tisk)
    elif ldata[1]== problem:
        users.remove(ldata[0])
        seeuser.delete("0.0",END)
        for us in users:
            tisk=us+'\n'
            seeuser.insert(END,tisk)

    else:
        return

def showcomingmessage(message):
    mainchat.config(state=NORMAL)
    mainchat.insert(END, message)
    mainchat.config(state=DISABLED)
```

# Client (2)

```
def SendMessage():
    global s
    global mymessage
    mymessage = chat.get("0.0",END)
    if len(mymessage) == 1:
        return
    #print len(mymessage)
    mainchat.config(state=NORMAL)
    mainchat.insert(END, "You" + ': ' + mymessage)
    mainchat.yview(END)
    chat.delete("0.0",END)
    mainchat.config(state=DISABLED)
    sendmessage= username + ": " + mymessage
    s.sendall(sendmessage)



def connectserver():
    global s
    s.connect((ipadress, 8888))
    s.send(username)
    entrys.destroy()
    entrys.quit()


def press():
    global s
    global  username,ipadress
    adresa=ent1.get()
    jmeno=ent2.get()
    ipadress=adresa
    username=jmeno

    connectserver()
```

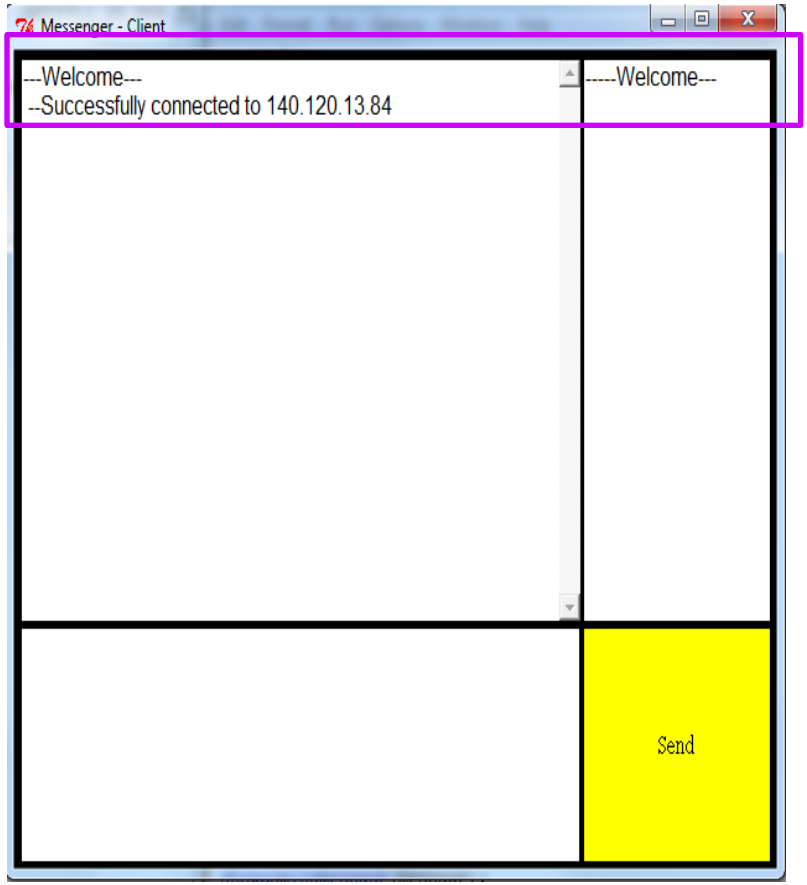
# Client (3)

```
def entry_GUI():

    global ent1,ent2
    global entrys

    entrys = Tk()
    entrys.title("Welcome")
    entrys.geometry("300x120")

    lab1 = Label(entrys,  text = "server IP")
    ent1 = Entry(entrys, width = 15)
    lab2 = Label(entrys, text = "username")
    ent2= Entry(entrys, width = 15)
    buttonlog=Button(entrys, text = 'Login',command=press)

    lab1.place(x=6,y=6)
    lab2.place(x=6,y=50)
    ent1.place(x=100,y=6,)
    ent2.place(x=100,y=50)
    buttonlog.place(x=200,y=6,height=50,width=50)

    entrys.mainloop()
```



# Client (4)

```
entry_GUI()

form = Tk()

form.title("Messenger - Client")
form.geometry("615x510")
form.configure(background = "Black")

mainchat = Text(form, bd=0, bg="white", font="Arial",)
chat = Text(form, bd=0, bg="white", font="Arial")
seeuser = Text(form, bd = 0,  bg="white", font="Arial")
sendb = Button(form, font=30, text="Send",bd=0, bg="#FFFF00",
               activebackground="#00FF00",command=SendMessage)
scrollbar = Scrollbar(form, command=mainchat.yview)
mainchat['yscrollcommand'] = scrollbar.set

mainchat.config(state=DISABLED)
mainchat.config(state=NORMAL)
mainchat.place(x=6,y=6, height=350, width=450)
chat.place(x=6, y=361, height=145, width=450)
sendb.place(x=460, y=361, height=145,width=150)
seeuser.place(x=460, y=6, height=350,width=150)

scrollbar.place(x=440,y=6, height=350)
```




```
mainchat.config(state=NORMAL)
mainchat.insert(END,'---Welcome--- \n --Successfully connected to ' + ipadress + '\n\n\n')
mainchat.config(state=DISABLED)
seeuser.insert(END,'-----Welcome---')
```

# Client (5)

```
def incomemessage2(s):
    global mymessage

    while True:
        try:
            message = s.recv(1024)
            if not message:
                break

            if (username + ": " + mymessage) != message:
                adduser(message)
                showcomingmessage(message)
                #print 'jsem v cili'

            if message == '':

                break
        except:
            break


threading.Thread(target = incomemessage2 , args = (s,)).start()
form.mainloop()
```


# Bibliography


- http://it.metr.ou.edu/byteofpython/features-of-python.html
- http://codesyntax.netfirms.com/lang-python.htm
- http://www.python.org/
- Sebesta, Robert W., Concepts of Programming Languages: 8th ed. 2007
- http://www.python.org/~guido/