CherryPy: Common Programmer Issues with Solutions

Prof.S.A.Shinde[1] & Mr. Kushal S. Patel[2]

ABSTRACT—

Python is rich with a large number of web libraries and frameworks. CherryPy has its own web (HTTP) server previously. Now it can be accessible from any browser. The web server is the gateway to a CherryPy application through which all traffic of HTTP requests and responses has to go. CherryPy Engine is responsible for the controlling, managing, and to monitor of CherryPy process. Even though CherryPy is a rich module of python, it has some issues with some lower versions of python. Earlier version than python 2.5 had issues with this module. This paper proposes some common problem and their solutions in this regard. Experimental results are also shown for the convenience of Python programmer to deal with these solved issues. This paper covers server and client errors along with channel errors associated with CherryPy and Python2.5.

KEYWORDS-

Python, CherryPy, Engine, Channel Allocation, Ports.

[1] Vidya Pratishthan's College of Engg. Maharashtra, India
Meetsan_shinde@yahoo.com

[2] Vidya Pratishthan's College of Engg Baramati, Maharashtra
Patelkushal4444@gmail.com

INTRODUCTION

Now days, World Wide Web is growing exponentially, and has become a key component of the way people live. There are several languages which supports client server architecture. Python language is an emerging scripting language in the field of automation. To have python client server architectural support; some necessary modules are need to be installed. One of those modules is CherryPy. CherryPy is a Python library providing a user friendly interface to the HTTP protocol. Web applications have grown exponentially in the last few years. Due to this it becomes necessity to have web interaction in python. This explosion is helpful to a large number of toolkits, libraries, and frameworks released. CherryPy has started using Python's strengths as a dynamic language to model and bind the HTTP protocol into an API that follows Python idioms.

Python is rich with a large number of web libraries and frameworks. Some of the features of CherryPy are described ahead: Simplicity: To use CherryPy is very simple. The narrow scope covered by the library, the developers have been able to concentrate on the API and community feedback. Self-contained: From the very beginning, The core of CherryPy would not require any third-party Python packages to work. This is pure and independent module designed by developers.
Not intrusive: This is a critical aspect of the library; the idea was to provide a set of tools to any developer making no assumptions about the way in which user may choose to use them.

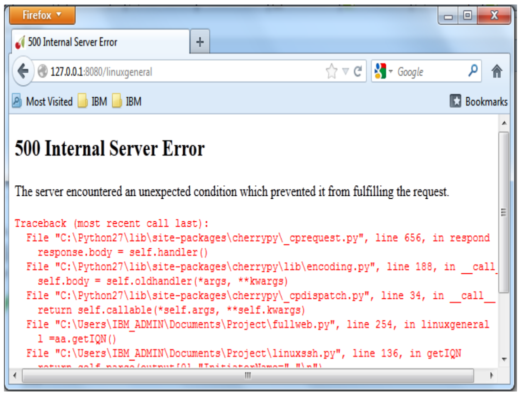
CherryPy has its own web (HTTP) server previously. Now it can be accessible from any browser. The web server is the gateway to a CherryPy application through which all traffic of HTTP requests and responses has to go. Therefore it is up to lower ISO-OSI layer to handle the low-level TCP sockets. These low-level TCP sockets are used to convey the information from the client to the server and vice versa. The internal CherryPy engine is responsible for Creating and managing Request and Response objects in communication. The Response object constructs response by internal computations and validates the response before providing it to the server. Another main important work of engine is to control, manage, and monitor the CherryPy process.

Caching is an important side of any web application. It reduces the load and stress of the different servers in action. Servers in the system can be—HTTP, application, and database servers. Generic caching is provided by this module which can help in achieving decent improvements in application's overall performance. The CherryPy caching module works at application level i.e. at the HTTP server level. It will cache the generated output to be sent to the destination and will retrieve a cached resource based on a predefined key. The cache data is stored at the server memory and is therefore it may lose when stopping the server. We can also pass own caching class to handle the underlying process differently while keeping the same high-level interface.

At the time of building an application using CherryPy; it is useful to understand the path taken by the application based on the input it processes. This helps to determine potential bottlenecks in the systems and also application runs in expected mode. The coverage module in CherryPy does this and provides a browseable output i.e. output can be taken directly in browser during application run.

Server And Client Error

CherryPy returns an HTTP error code 500 when it catches an error that is not handled otherwise by the application developer. The HTTP specification defines two sets of error codes, client errors in the 4xx range and server errors in the 5xx range. The client errors indicate that the user agent has sent an invalid request (e.g. missing authentication credentials, requested resource not found or gone, etc.). If you got error message as 50X where X can be any integer from 0 to 9; it means that either CherryPy serv


Firefox
500 Internal Server Error
127.0.0.1:8080/linuxgeneral
Most Visited IBM IBM Bookmarks

500 Internal Server Error

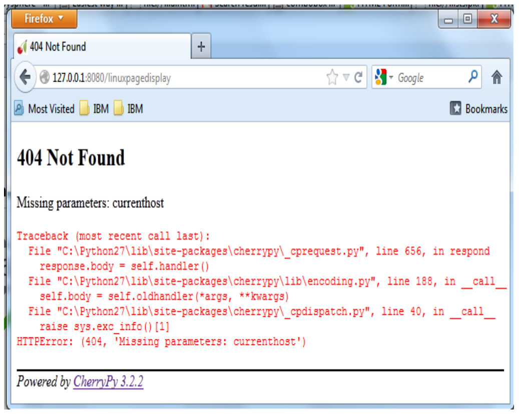
The server encountered an unexpected condition which prevented it from fulfilling the request.

```
Traceback (most recent call last):
  File "C:\Python27\lib\site-packages\cherrypy\_cprequest.py", line 656, in respond
    response.body = self.handler()
  File "C:\Python27\lib\site-packages\cherrypy\lib\encoding.py", line 188, in __call_
    self.body = self.oldhandler(*args, **kwargs)
  File "C:\Python27\lib\site-packages\cherrypy\_cpdispatch.py", line 34, in __call__
    return self.callable(*self.args, **self.kwargs)
  File "C:\Users\IBM_ADMIN\Documents\Project\fullweb.py", line 254, in linuxgeneral
    l =aa.getIQN()
  File "C:\Users\IBM_ADMIN\Documents\Project\linuxssh.py", line 136, in getIQN
```

er has not found method or most probably there is any error in parameters of methods. In CherryPy, each method has to be explored independently. If method is not exposed then it is treated as private method and CherryPy server cannot import that directly. Below snapshot is an example of common server error.

The Cherrypy Server can be accessed by different HTTP methods. GET: This returns the representation of the requested resource depending on the Accept header. Our application allows application/xml, application/atom+xml, text/json, or text/x-json. We use a function called accept, which returns the acceptable header found or raises a cherrypy.HTTPError (406, 'Not Acceptable') error immediately to inform the user agent that our application cannot deal with its request. Then we verify if the resource still exists; if not, we raise a cherrypy.NotFound error, which is a shortcut to cherrypy. HTTPError(404, 'Not Found'). Once we have our pre-conditions checked, we return the requested representation of the resource. Note that this is equivalent to the index() method with the default dispatcher. Bear in mind though that there is no equivalent to the default() method when using the method dispatcher. POST: The HTTP POST method allows a user agent to create a new resource. The first step is to check if the photoblog that will handle that resource exists. Then we create the resource and we return a status code 201 Created along with the Location header indicating the URI to retrieve the newly created resource. PUT: The HTTP PUT method allows the user agent to replace a resource with the one provided in the request body. It is often considered as an update operation. Although RFC 2616 does not forbid PUT to also create a new resource, we will not use it that way in our application as we will explain later.

Like CherrPy server error; client side error can also be happen with this model.


Firefox
404 Not Found
127.0.0.1:8080/linuxpagedisplay
Most Visited IBM IBM Bookmarks

404 Not Found

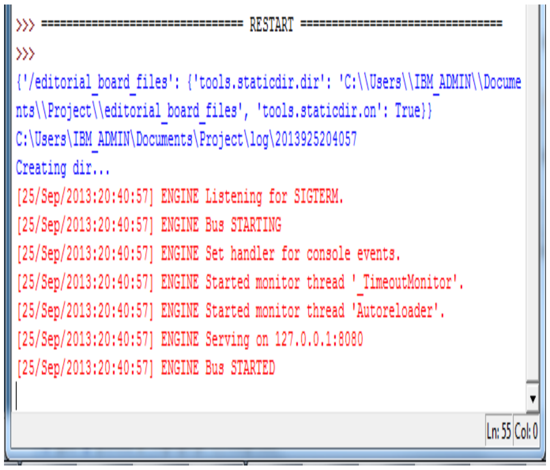
Missing parameters: currenthost

```
Traceback (most recent call last):
  File "C:\Python27\lib\site-packages\cherrypy\_cprequest.py", line 656, in respond
    response.body = self.handler()
  File "C:\Python27\lib\site-packages\cherrypy\lib\encoding.py", line 188, in __call__
    self.body = self.oldhandler(*args, **kwargs)
  File "C:\Python27\lib\site-packages\cherrypy\_cpdispatch.py", line 40, in __call__
    raise sys.exc_info()[1]
HTTPError: (404, 'Missing parameters: currenthost')
```

Powered by CherryPy 3.2.2

In this type of errors; the sources can be at client side probably. Client side issues consist of invalid URL requested. If user produces an invalid URL request then CherryPy interpreter at server end detects that URL is invalid and client gets response as Error 40X where X is any integer between 0 to 9. Another source of error can be invalid parameters in actions. This can include insufficient or variable parameter lists provided to HTML server by client. Mismatching of sequence of parameters can also raise error in execution.

CHANNEL BUS ALLOCATION



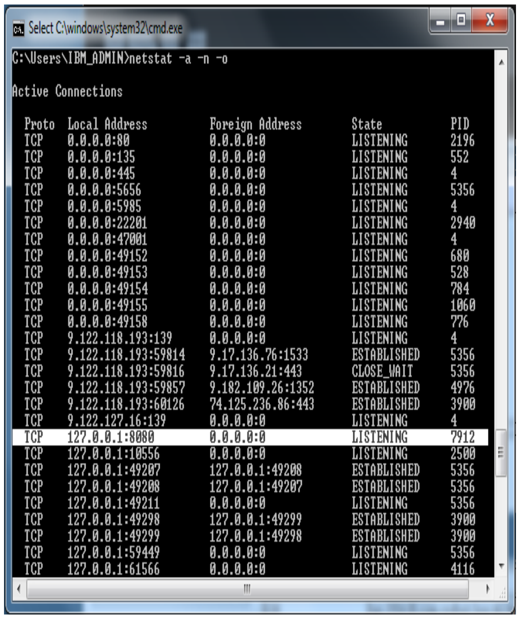
```
>>> ================================ RESTART ================================
>>>
{'/editorial_board_files': {'tools.staticdir.dir': 'C:\\Users\\IBM_ADMIN\\Documents\\Project\\editorial_board_files', 'tools.staticdir.on': True}}
C:\Users\IBM_ADMIN\Documents\Project\log\2013925204057
Creating dir...
[25/Sep/2013:20:40:57] ENGINE Listening for SIGTERM.
[25/Sep/2013:20:40:57] ENGINE Bus STARTING
[25/Sep/2013:20:40:57] ENGINE Set handler for console events.
[25/Sep/2013:20:40:57] ENGINE Started monitor thread '_TimeoutMonitor'.
[25/Sep/2013:20:40:57] ENGINE Started monitor thread 'Autoreloader'.
[25/Sep/2013:20:40:57] ENGINE Serving on 127.0.0.1:8080
[25/Sep/2013:20:40:57] ENGINE Bus STARTED
```

Above figure shows how Engine Bus allocation is done by CherryPy. In this starting process; all the CherryPy variables are initialized. These variables are used to get and set the system environment. Main thread is started at this stage and as per requests sub threads are created and maintained. Main thread is killed by operating system at the time of application closing process. If there are some error in application close by CherryPy then the channel allocation fault is occurred which is explained in detail at later part of this paper.

CherryPy uses the concept of channel allocation at protocol level. In this concept of channel allocation, one dedicated virtual data pipe is allocated to each application. In this when cherrypy server started execution, it determines on which channel it has to establish connection. Once the channel pipe is detected, a virtual channel is created using IP address and port number. If it is not mentioned in server's python script then the default allocation is done using virtual channel of IP address as 'loaclhost' or "127.0.0.1" and default port as 8080.

This channel allocation has several advantages when security of cherrypy is considered. Channel system is very much useful in multiprogramming environment. Every application has different channel, due to this number of applications can be run successfully on same machine without interference to each other. Ideally infinite allocations can be done but due to some practical issues the channel and ultimately applications of cherrypy are limited to 65K i.e. 65535. This method is also known as Engine Bus system as different data follows different bus to travel.

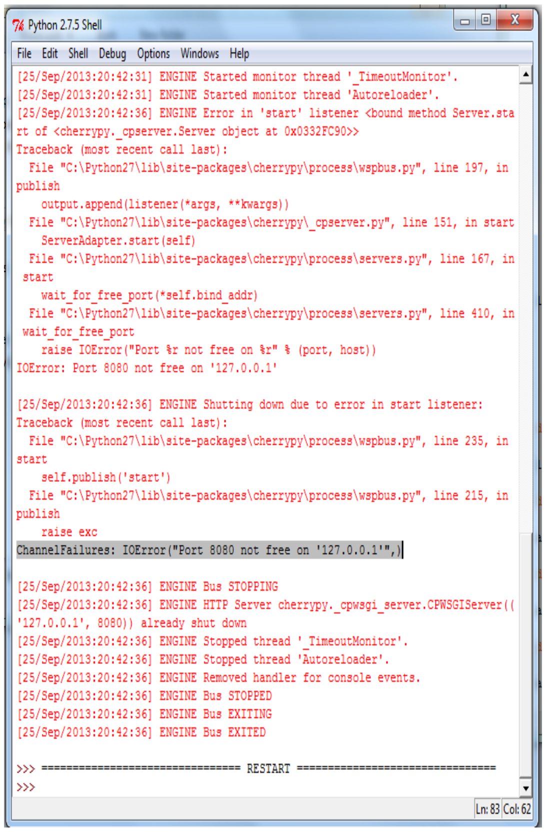
This method has some problems associated with it. Some of problems are illustrated in later part of this paper. Once the application of cherrypy server closed, it is expected that it should relese all the channels allocated to it. If Python code associated with application is dependent on older version than Py 2.7, unfortunately Python code not allwed to return all the channel resources to the underlying operating system. Due to this, new application cannot use this channel for data communication even though it is not used by any other application. If this happens repetedly, number of dead resources increases and it may cause some problem with server machine.



```
C:\Users\IBM_ADMIN>netstat -a -n -o

Active Connections

  Proto  Local Address          Foreign Address        State           PID
  TCP    0.0.0.0:80             0.0.0.0:0              LISTENING       2196
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING       552
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING       4
  TCP    0.0.0.0:5656           0.0.0.0:0              LISTENING       5356
  TCP    0.0.0.0:5985           0.0.0.0:0              LISTENING       4
  TCP    0.0.0.0:22201          0.0.0.0:0              LISTENING       2940
  TCP    0.0.0.0:47001          0.0.0.0:0              LISTENING       4
  TCP    0.0.0.0:49152          0.0.0.0:0              LISTENING       680
  TCP    0.0.0.0:49153          0.0.0.0:0              LISTENING       528
  TCP    0.0.0.0:49154          0.0.0.0:0              LISTENING       784
  TCP    0.0.0.0:49155          0.0.0.0:0              LISTENING       1060
  TCP    0.0.0.0:49158          0.0.0.0:0              LISTENING       776
  TCP    9.122.118.193:139      0.0.0.0:0              LISTENING       4
  TCP    9.122.118.193:59814    9.17.136.76:1533       ESTABLISHED     5356
  TCP    9.122.118.193:59816    9.17.136.21:443        CLOSE_WAIT      5356
  TCP    9.122.118.193:59857    9.182.109.26:1352      ESTABLISHED     4976
  TCP    9.122.118.193:60126    74.125.236.86:443      ESTABLISHED     3900
  TCP    9.122.127.16:139       0.0.0.0:0              LISTENING       4
  TCP    127.0.0.1:8080         0.0.0.0:0              LISTENING       7912
  TCP    127.0.0.1:10556        0.0.0.0:0              LISTENING       2500
  TCP    127.0.0.1:49207        127.0.0.1:49208        ESTABLISHED     5356
  TCP    127.0.0.1:49208        127.0.0.1:49207        ESTABLISHED     5356
  TCP    127.0.0.1:49211        0.0.0.0:0              LISTENING       5356
  TCP    127.0.0.1:49298        127.0.0.1:49299        ESTABLISHED     3900
  TCP    127.0.0.1:49299        127.0.0.1:49298        ESTABLISHED     3900
  TCP    127.0.0.1:59449        0.0.0.0:0              LISTENING       5356
  TCP    127.0.0.1:61566        0.0.0.0:0              LISTENING       4116
```



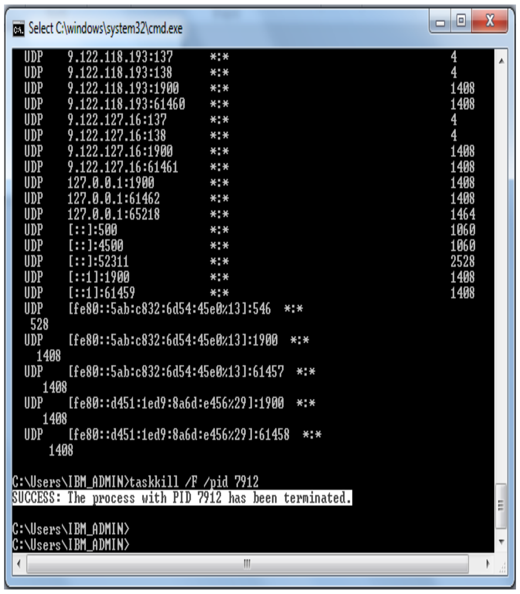
```
[25/Sep/2013:20:42:31] ENGINE Started monitor thread '_TimeoutMonitor'.
[25/Sep/2013:20:42:31] ENGINE Started monitor thread 'Autoreloader'.
[25/Sep/2013:20:42:36] ENGINE Error in 'start' listener <bound method Server.sta
rt of <cherrypy._cpserver.Server object at 0x0332FC90>>
Traceback (most recent call last):
  File "C:\Python27\lib\site-packages\cherrypy\process\wspbus.py", line 197, in
publish
    output.append(listener(*args, **kwargs))
  File "C:\Python27\lib\site-packages\cherrypy\_cpserver.py", line 151, in start
    ServerAdapter.start(self)
  File "C:\Python27\lib\site-packages\cherrypy\process\servers.py", line 167, in
 start
    wait_for_free_port(*self.bind_addr)
  File "C:\Python27\lib\site-packages\cherrypy\process\servers.py", line 410, in
 wait_for_free_port
    raise IOError("Port %r not free on %r" % (port, host))
IOError: Port 8080 not free on '127.0.0.1'

[25/Sep/2013:20:42:36] ENGINE Shutting down due to error in start listener:
Traceback (most recent call last):
  File "C:\Python27\lib\site-packages\cherrypy\process\wspbus.py", line 235, in
start
    self.publish('start')
  File "C:\Python27\lib\site-packages\cherrypy\process\wspbus.py", line 215, in
publish
    raise exc
ChannelFailures: IOError("Port 8080 not free on '127.0.0.1'",)

[25/Sep/2013:20:42:36] ENGINE Bus STOPPING
[25/Sep/2013:20:42:36] ENGINE HTTP Server cherrypy._cpwsgi_server.CPWSGIServer((
'127.0.0.1', 8080)) already shut down
[25/Sep/2013:20:42:36] ENGINE Stopped thread '_TimeoutMonitor'.
[25/Sep/2013:20:42:36] ENGINE Stopped thread 'Autoreloader'.
[25/Sep/2013:20:42:36] ENGINE Removed handler for console events.
[25/Sep/2013:20:42:36] ENGINE Bus STOPPED
[25/Sep/2013:20:42:36] ENGINE Bus EXITING
[25/Sep/2013:20:42:36] ENGINE Bus EXITED

>>> ================================ RESTART ================================
>>>
```

This problem mostly observed by cherrypy programmer as intermediate runnig causing error of channel is already allocated. This problem has a simple solution of restarting the server machine. It works well with dedicated server environment or virtual server envonment. This solution is very much less effective when cherrypy programmer is concern. Programmer cannot restart his machine on every run.



```
  UDP    9.122.118.193:137      *:*                                    4
  UDP    9.122.118.193:138      *:*                                    4
  UDP    9.122.118.193:1900     *:*                                    1408
  UDP    9.122.118.193:61460    *:*                                    1408
  UDP    9.122.127.16:137       *:*                                    4
  UDP    9.122.127.16:138       *:*                                    4
  UDP    9.122.127.16:1900      *:*                                    1408
  UDP    9.122.127.16:61461     *:*                                    1408
  UDP    127.0.0.1:1900         *:*                                    1408
  UDP    127.0.0.1:61462        *:*                                    1408
  UDP    127.0.0.1:65218        *:*                                    1464
  UDP    [::]:500               *:*                                    1060
  UDP    [::]:4500              *:*                                    1060
  UDP    [::]:52311             *:*                                    2528
  UDP    [::1]:1900             *:*                                    1408
  UDP    [::1]:61459            *:*                                    1408
  UDP    [fe80::5ab:c832:6d54:45e0%13]:546  *:*
   528
  UDP    [fe80::5ab:c832:6d54:45e0%13]:1900  *:*
    1408
  UDP    [fe80::5ab:c832:6d54:45e0%13]:61457  *:*
     1408
  UDP    [fe80::d451:1ed9:8a6d:e456%29]:1900  *:*
     1408
  UDP    [fe80::d451:1ed9:8a6d:e456%29]:61458  *:*
      1408

C:\Users\IBM_ADMIN>taskkill /F /pid 7912
SUCCESS: The process with PID 7912 has been terminated.

C:\Users\IBM_ADMIN>
C:\Users\IBM_ADMIN>
```

On the Windows platform, netstat information can be retrieved by calling the **GetTcpTable** and **GetUdpTable** functions. These functions are present in the IP Helper API, or IPHLPAPI.DLL. Returned Information includes local and remote IP addresses, local and remote ports, and (for GetTcpTable) status codes of TCP . In addition to the command-line netstat.exe tool that ships with Windows, GUI -based netstat programs are also available. netstat command is available only if the Internet Protocol is installed as a component in the properties of a network adapter in Network Connections. This command is useful to see the allocation of channel. Programmer knows that his program is allocating which IP address and port number. Using this he can get the channel details. Here import channel detail is process number of that cherrypy application. Once the application is closed, if it is visible in netstat, then this process has to be deleted. On deletion of this task, the channel is relesed forcefully and be available for next application. Snapshots shows detail explaiation of this story.

CONCLUSIONS

Once these steps are done, the channel is now free. Port is also free. This logical port is now added in free pool of operating system and online for allocation. This will add a more functionality in CherryPy module. This concept is already reported to Python.org. Due to this, new versions of python are overcome with this problem.

REFERENCES

[1] Lindstrom, G., "Programming with Python," Journal of IT Professional Sept.-Oct. 2005

[2] Courtwright E., Chuan Yue, Haining Wang,"Efficient resource management on template-based web servers," Dependable Systems & Networks, 2009. DSN '09. IEEE/IFIP International Conference June - July 2009

[3] Edwards, Stephen H."Adding software testing to programming assignments ," 2013 IEEE 26th Conference on Digital Object Identifier: 2013

[4] Xu, Zhaogui, Qian, Ju, Chen, Lin, Chen, Zhifei, Xu, Baowen,"Static Slicing for Python First-Class Objects," 13th International Conference on Digital Object Identifier: 2013

[5] Begosso, L.C. ; Begosso, L.R. ; Goncalves, E.M. ; Goncalves, J.R.,"An approach for teaching algorithms and computer programming using Greenfoot and Python,"Frontiers in Education Conference (FIE), 2012 Digital Object Identifier, 2012

[6] Villar, J.I., Juan, J., Bellido, M.J., Viejo, J., Guerrero, D., Decaluwe, J. ,"Python as a hardware description language: A case study," Digital Object Identifier, 2011