Fun with Interactive Data Language (IDL) Sockets

Dominic Zarro (ADNET Systems, Inc. and NASA/GSFC Code 671)

GSFC Poster Session
January 2016

Abstract:

Interactive Data Language (IDL) provides a low-level socket procedure that enables executing HTTP functions such as GET, HEAD, POST, and PUT requests. Such functions facilitate developing network-based applications such as searching distributed archives and downloading remote datasets - core functions of Data Centers and Virtual Observatories.

Until recently, the socket procedure has been limited to uni-directional communication with a server. However, with the release of IDL 8.5, a new client "listener" feature provides the interesting possibility of bi-directional communication between a client and server. This feature opens the door to developing powerful capabilities such as exchanging client-server data and, even more exciting, executing IDL commands on a remote server without an intermediate API or bridge. This poster will demonstrate a novel prototype client-server application that we have developed to perform remote data processing of datasets using this new socket feature.

Anatomy of a HTTP Get Request:

Open a socket to server and send a GET request

```
server="imgsrc.hubblesite.org"
file="hu/db/images/hs-2007-19-a-xlarge_web.jpg"

socket,lun,server,80,/get_lun
printf,lun,"GET "+file+" HTTP/1.1"
printf,lun,"Host:imgsrc.hubblesite.org:80"
printf,lun,""
```

Read HTTP response

```
header="" & text="xxx"
while text ne "" do begin
 readf,lun,text
 header=[header,text]
endwhile
```

Examine HTTP response

```
print, header

HTTP/1.1 200 OK Date: Mon, 25 Jan 2016
02:56:09 GMT
Last-Modified: Tue, 11 Dec 2007 19:41:14
Accept-Ranges: bytes
Content-Length: 1081679
Content-Type: image/jpeg)
Connection: close
```

Download and read file

```
data=bytarr(1081679)
readu,lun,data
close,lun

openw,lun,"output.jpeg",/get_lun
writeu,lun,data
close,lun

read_jpeg,"output.jpeg",image,/true
tv,image,/true
```

Building a Client-Server System:

Start remote server

```
pro SockServer,port

If n_elements(port) eq 0 then port=21038
socket, ListenerLUN, port, /listen, /get_lun
!null = Timer.Set (.1, "ListenerCallback", ListenerLUN)
return & end
```

Listen for client connections

```
pro ListenerCallback,ID,ListenerLUN

status = File_Poll_Input(ListenerLUN, Timeout = .1)
if (status) then begin
  socket, ClientLUN, accept = ListenerLUN, /get_lun
 !null = Timer.Set(.1, "SockServerCallback", ClientLUN)
endif else begin
 !null = Timer.Set(.1, "ListenerCallback", ListenerLUN)
endelse
return & end
```

Read client data and execute command

```
pro SockServerCallback,ID,ClientLUN

status=File_Poll_Input(ClientLUN, Timeout = .01)
if status then begin
 command=""
 readu, ClientLun, nbytes
 data=bytarr(nbytes)
 readu, ClientLun, data
 readf,Clientlun,command
 status=execute(command)
endif
!null=Timer.Set(.1, "SockServerCallback", ClientLUN)
return & end
```

Start remote client

```
pro SockClient, ID, port,server, lun=ServerLUN

if n_elements(port) eq 0 then port = 21038
if n_elements(server) eq 0  then server="localhost"
socket, ServerLUN, server, port, /get_lun, error=error
If error ne 0 then !null=timer.set(1,"SockClient",port)
return & end
```

Example Use Case:

Send FITS data to server

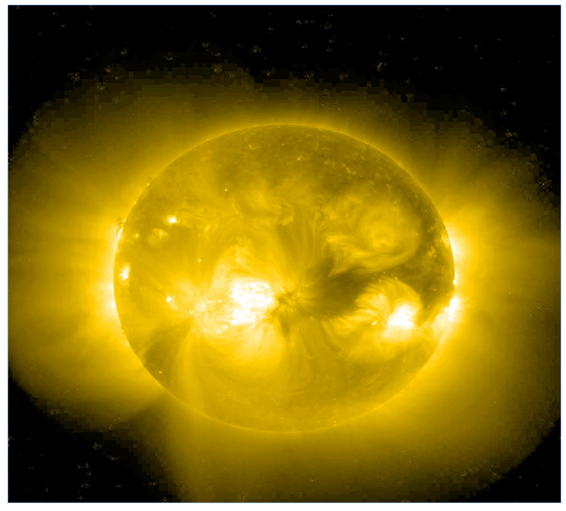
```
file="20141201_044004_n7euA.fts"
openr,lun,file,/get_lun
nbytes=(fstat(lun)).size
data=bytarr(nbytes)
readu,lun,data

writeu, ServerLUN, nbytes
writeu, ServerLUN, data
```

Send IDL command to server

```
c="s=obj_new('euvi') & s->read,data & s=>plot"

printf,ServerLUN,c
```



Advantages and Applications:

- Pure IDL solution.
- Platform/OS Independent
- Remote Procedure Calls (RPC's)
- Multiple server-clients for distributed computer processing