In a previous post, I demonstrated how to write a client-server system using IDL sockets. In this post, I will demonstrate how to write a simple server that accepts HTTP requests. Most of the coding has already been done in the SockClient and SockServer programs. Here, I introduce a new program SockHTTP that does the work of interpreting and executing a GET request to download a file:

```
pro SockHTTP,lun

;-- (1) read HTTP header

value='' & header='' & text='xxx'
while text ne  '' do begin
 readf,lun,text
 header=[header,text]
endwhile
nhead=n_elements(header)
if nhead gt 1 then value=header[1:nhead-1] else value=''

;-- (2) bail if not GET

request=strsplit(value[0],' ',/extract)
print,request
cr=''
if !version.os_family ne 'Windows' then cr=string(13b)

if (request[0] ne 'GET') then begin
 printf,lun,'HTTP/1.1 400 Bad Request'+cr
 printf,lun,'Connection: close'+cr
 free_lun,lun,/force
 return
endif

;-- (3) check if requested file exists in current directory and get
its size.
;-- bail if not found

fname='.'+request[1]
bsize=0l
```

```
info=file_info(fname)
if info.exists && info.regular then bsize=info.size
if bsize eq 0 then begin
 printf,lun,'HTTP/1.1 404 Not Found'+cr
 printf,lun,'Connection: close'+cr
 free_lun,lun,/force
 return
endif

;-- (4) send requested file by reading it and writing bytes to socket

printf,lun,'HTTP/1.1 200 OK'+cr
printf,lun,'Content-Length: '+strtrim(bsize,2)+cr
printf,lun,'Connection: close'+cr
printf,lun,cr

openr,flun,fname,/get_lun
bdata=bytarr(bsize,/nozero)
readu,flun,bdata
writeu,lun,bdata
free_lun,flun,/force

free_lun,lun,/force

return & end
```

There is quite a bit of code here. Let's take it one section at a time:

(1) The input argument for SockHTTP is the socket logical unit number `lun` that has been previously opened when the client connects to the server. I sequentially read the ASCII headers that are being sent by the client over this socket into the string array `value`. For a GET request, the first entry in the headers would look something like:

```
GET /path/filename HTTP/1.1
```

where path and filename is the location and name of the file that is being requested. Note that I am assuming that the server is running on a Unix-like operating system.

(2) From the first header line, I parse out the GET request and filename into the string array

`request` by using strsplit. If the request is not GET, I exit gracefully by printing the following HTTP error message to the client:

```
HTTP/1.1 400 Bad Request
Connection: close
```

Note that I add a carriage-return string (represented by the byte value 13b) to signal the end of each response line. This is not necessary in Windows which automatically appends a carriage-return at the end of the print statement. I subsequently call free_lun to physically close the socket and return.

(3) Using the filename in the second element of request, I call file_info to determine the file size in bytes. For simplicity, I'll assume that the file is in the current directory which I designate by using a '.' period. If the file doesn't exist (or is not a regular file), I send the following HTTP file not found message, close the socket, and return:

```
HTTP/1.1 404 Not Found
Connection: close
```

(4) Having verified that the requested file exists, I am now ready to communicate with the client. Before doing so, I send the following headers to the client:

```
HTTP/1.1 200 OK
Content-Length: '+strtrim(bsize,2)
Connection: close
```

where the HTTP status code 200 signals that the request is successful, the *Content-Length* header contains the size of the file in bytes (converted to a string) which the client needs to read the file, and the *Connection: close* header instructs the client to close the connection once the file is sent. The last line is a blank string (or carriage return) which alerts the client that the server has completed sending header information. I next send the file to the client by opening it, reading it into a byte array, and writing it to the open socket:

```
openr,flun,fname,/get_lun
bdata=bytarr(bsize,/nozero)
readu,flun,bdata
writeu,lun,bdata
```

Finally, I wrap up by closing the logical unit numbers for the opened file and socket. I am now ready to test the server by replacing the ServerCallback function in SockServer

discussed in the post with the following code:

```
pro ServerCallback,ID,ClientLUN
status=File_Poll_Input(ClientLUN, Timeout = .01)
if status then sockHTTP,clientlun
ID=Timer.Set(.1, "ServerCallback", ClientLUN)
return & end
```

You can download a version of SockServer with SockHTTP in the ServerCallback function from GitHub. This version is called SockServerHTTP. Download it together with the file galaxy.jpg into your local directory, and start the server in an IDL session:

```
IDL> SockServerHTTP
SOCKSERVERHTTP: Server listening on port 8000
```

Next open your favorite browser and enter the URL:

```
http://localhost:8000/galaxy.jpg
```

If all goes well, you should see the following image: