NAME

curl_multi_socket − reads/writes available data

SYNOPSIS

#include <curl/curl.h>

CURLMcode curl_multi_socket(CURLM * multi_handle, curl_socket_t sockfd);

CURLMcode curl_multi_socket_all(CURLM *multi_handle);

DESCRIPTION

Alternative versions of *curl_multi_perform()* that allows the application to pass in one of the file descriptors/sockets that have been detected to have "action" on them and let libcurl perform. This allows libcurl to not have to scan through all possible file descriptors to check for action. When the application has detected action on a socket handled by libcurl, it should call *curl_multi_perform()* with the **sockfd** argument set to the socket with the action.

These functions inform the application about updates in the socket (file descriptor) status by doing none, one or multiple calls to the curl_socket_callback given with the CURLMOPT_SOCKETFUNCTION option to *curl_multi_setopt(3)*. They update the status with changes since the previous time this function was called.

If you want to force libcurl to (re-)check all its internal sockets and transfers instead of just a single one, you call **curl_multi_socket_all(3)** instead.

An application should call **curl_multi_timeout(3)** to figure out how long it should wait for socket actions − at most − before doing the timeout action: call the **curl_multi_socket(3)** function with the **sockfd** argument set to CURL_SOCKET_TIMEOUT.

The socket **callback** function uses a prototype like this

```
int curl_socket_callback(CURL *easy,      /* easy handle */
                         curl_socket_t s, /* socket */
                         int action,      /* see values below */
                         void *userp);    /* "private" pointer */
```

The callback MUST return 0.

The *action* (third) argument to the callback has one of five values:

CURL_POLL_NONE (0)
: register, not interested in readiness (yet)

CURL_POLL_IN (1)
: register, interested in read readiness

CURL_POLL_OUT (2)
: register, interested in write readiness

CURL_POLL_INOUT (3)
: register, interested in both read and write readiness

CURL_POLL_REMOVE (4)
: deregister

RETURN VALUE

CURLMcode type, general libcurl multi interface error code.

If you receive *CURLM_CALL_MULTI_PERFORM*, this basically means that you should call

curl_multi_perform again, before you wait for more actions on libcurl's sockets. You don't have to do it immediately, but the return code means that libcurl may have more data available to return or that there may be more data to send off before it is "satisfied".

NOTE that this only returns errors etc regarding the whole multi stack. There might still have occurred problems on individual transfers even when this function returns OK.

TYPICAL USAGE

1. Create a multi handle

2. Set the socket callback with CURLMOPT_SOCKETFUNCTION

3. Add easy handles

4. Call curl_multi_socket_all() first once

5. Setup a "collection" of sockets to supervise when your socket callback is called.

6. Use curl_multi_timeout() to figure out how long to wait for action

7. Wait for action on any of libcurl's sockets

8, When action happens, call curl_multi_socket() for the socket(s) that got action.

9. Go back to step 6.

AVAILABILITY

This function was added in libcurl 7.16.0

SEE ALSO

curl_multi_cleanup(3), **curl_multi_init**(3), **curl_multi_fdset**(3), **curl_multi_info_read**(3)