NAME

curl_multi_socket_action - reads/writes available data given an action

SYNOPSIS

```
#include <curl/curl.h>

CURLMcode curl_multi_socket_action(CURLM *multi_handle,
                                   curl_socket_t sockfd,
                                   int ev_bitmask,
                                   int *running_handles);
```

DESCRIPTION

When the application has detected action on a socket handled by libcurl, it should call *curl_multi_socket_action(3)* with the **sockfd** argument set to the socket with the action. When the events on a socket are known, they can be passed as an events bitmask **ev_bitmask** by first setting **ev_bitmask** to 0, and then adding using bitwise OR (|) any combination of events to be chosen from CURL_CSELECT_IN, CURL_CSELECT_OUT or CURL_CSELECT_ERR. When the events on a socket are unknown, pass 0 instead, and libcurl tests the descriptor internally. It is also permissible to pass CURL_SOCKET_TIMEOUT to the **sockfd** parameter in order to initiate the whole process or when a timeout occurs.

At return, **running_handles** points to the number of running easy handles within the multi handle. When this number reaches zero, all transfers are complete/done. When you call *curl_multi_socket_action(3)* on a specific socket and the counter decreases by one, it DOES NOT necessarily mean that this exact socket/transfer is the one that completed. Use *curl_multi_info_read(3)* to figure out which easy handle that completed.

The *curl_multi_socket_action(3)* function informs the application about updates in the socket (file descriptor) status by doing none, one, or multiple calls to the socket callback function set with the *CURLMOPT_SOCKETFUNCTION(3)* option to *curl_multi_setopt(3)*. They update the status with changes since the previous time the callback was called.

Get the timeout time by setting the *CURLMOPT_TIMERFUNCTION(3)* option with *curl_multi_setopt(3)*. Your application then gets called with information on how long to wait for socket actions at most before doing the timeout action: call the *curl_multi_socket_action(3)* function with the **sockfd** argument set to CURL_SOCKET_TIMEOUT. You can also use the *curl_multi_timeout(3)* function to poll the value at any given time, but for an event-based system using the callback is far better than relying on polling the timeout value.

When this function returns error, the state of all transfers are uncertain and they cannot be continued.

curl_multi_socket_action(3) should not be called again on the same multi handle after an error has been returned, unless first removing all the handles and adding new ones.

TYPICAL USAGE

1. Create a multi handle

2. Set the socket callback with *CURLMOPT_SOCKETFUNCTION(3)*

3. Set the timeout callback with *CURLMOPT_TIMERFUNCTION(3)*, to get to know what timeout value to use when waiting for socket activities.

4. Add easy handles with curl_multi_add_handle()

5. Provide some means to manage the sockets libcurl is using, so you can check them for activity. This can be done through your application code, or by way of an external library such as libevent or glib.

6. Call curl_multi_socket_action(..., CURL_SOCKET_TIMEOUT, 0, ...) to kickstart everything. To get one or more callbacks called.

7. Wait for activity on any of libcurl's sockets, use the timeout value your callback has been told.

8, When activity is detected, call curl_multi_socket_action() for the socket(s) that got action. If no activity is detected and the timeout expires, call *curl_multi_socket_action(3)* with *CURL_SOCKET_TIMEOUT*.

EXAMPLE

```
int main(void)
{
  /* the event-library gets told when there activity on the socket 'fd',
     which we translate to a call to curl_multi_socket_action() */
  int running;
  CURLM *multi; /* the stack we work with */
  int fd; /* the descriptor that had action */
  int bitmask; /* what activity that happened */
  CURLMcode mc = curl_multi_socket_action(multi, fd, bitmask, &running);
  if(mc)
    printf("error: %s\n", curl_multi_strerror(mc));
}
```

AVAILABILITY

This function was added in libcurl 7.15.4, and is deemed stable since 7.16.0.

RETURN VALUE

CURLMcode type, general libcurl multi interface error code. See *libcurl-errors(3)*

SEE ALSO

curl_multi_cleanup(3), **curl_multi_fdset**(3), **curl_multi_info_read**(3), **curl_multi_init**(3), **the**hiperfifo.c**example**