NAME

curl_multi_perform - reads/writes available data from easy handles

SYNOPSIS

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

CURLMcode curl_multi_perform(CURLM *multi_handle, int *running_handles);
```

DESCRIPTION

This function performs transfers on all the added handles that need attention in a non-blocking fashion. The easy handles have previously been added to the multi handle with *curl_multi_add_handle(3)*.

When an application has found out there is data available for the multi_handle or a timeout has elapsed, the application should call this function to read/write whatever there is to read or write right now etc. *curl_multi_perform(3)* returns as soon as the reads/writes are done. This function does not require that there actually is any data available for reading or that data can be written, it can be called just in case. It stores the number of handles that still transfer data in the second argument's integer-pointer.

If the amount of *running_handles* is changed from the previous call (or is less than the amount of easy handles you have added to the multi handle), you know that there is one or more transfers less "running". You can then call *curl_multi_info_read(3)* to get information about each individual completed transfer, and that returned info includes CURLcode and more. If an added handle fails quickly, it may never be counted as a running_handle. You could use *curl_multi_info_read(3)* to track actual status of the added handles in that case.

When *running_handles* is set to zero (0) on the return of this function, there is no longer any transfers in progress.

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

EXAMPLE

```
int main(void)
{
  int still_running;
  CURL *multi = curl_multi_init();
  CURL *curl = curl_easy_init();
  if(curl) {
    curl_multi_add_handle(multi, curl);
```

```
    do {
      CURLMcode mc = curl_multi_perform(multi, &still_running);

      if(!mc && still_running)
        /* wait for activity, timeout or "nothing" */
        mc = curl_multi_poll(multi, NULL, 0, 1000, NULL);

      if(mc) {
        fprintf(stderr, "curl_multi_poll() failed, code %d.\n", (int)mc);
        break;
      }

    /* if there are still transfers, loop! */
    } while(still_running);
  }
}
```

AVAILABILITY

Added in 7.9.6

RETURN VALUE

CURLMcode type, general libcurl multi interface error code.

This function returns errors regarding the whole multi stack. Problems on individual transfers may have occurred even when this function returns *CURLM_OK*. Use *curl_multi_info_read(3)* to figure out how individual transfers did.

TYPICAL USAGE

Most applications use *curl_multi_poll(3)* to make libcurl wait for activity on any of the ongoing transfers. As soon as one or more file descriptor has activity or the function times out, the application calls *curl_multi_perform(3)*.

SEE ALSO

curl_multi_add_handle(3), **curl_multi_cleanup**(3), **curl_multi_fdset**(3), **curl_multi_info_read**(3), **curl_multi_init**(3), **curl_multi_wait**(3), **libcurl-errors**(3)