NAME

curl_multi_fdset - extracts file descriptor information from a multi handle

SYNOPSIS

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

CURLMcode curl_multi_fdset(CURLM *multi_handle,
                           fd_set *read_fd_set,
                           fd_set *write_fd_set,
                           fd_set *exc_fd_set,
                           int *max_fd);
```

DESCRIPTION

This function extracts file descriptor information from a given multi_handle. libcurl returns its *fd_set* sets. The application can use these to select() on, but be sure to *FD_ZERO* them before calling this function as *curl_multi_fdset(3)* only adds its own descriptors, it does not zero or otherwise remove any others. The *curl_multi_perform(3)* function should be called as soon as one of them is ready to be read from or written to.

If the *read_fd_set* argument is not a null pointer, it points to an object of type **fd_set** that on returns specifies the file descriptors to be checked for being ready to read.

If the *write_fd_set* argument is not a null pointer, it points to an object of type **fd_set** that on return specifies the file descriptors to be checked for being ready to write.

If the *exc_fd_set* argument is not a null pointer, it points to an object of type **fd_set** that on return specifies the file descriptors to be checked for error conditions pending.

If no file descriptors are set by libcurl, *max_fd* contain -1 when this function returns. Otherwise it contains the highest descriptor number libcurl set. When libcurl returns -1 in *max_fd*, it is because libcurl currently does something that is not possible for your application to monitor with a socket and unfortunately you can then not know exactly when the current action is completed using select(). You then need to wait a while before you proceed and call *curl_multi_perform(3)* anyway. How long to wait? Unless *curl_multi_timeout(3)* gives you a lower number, we suggest 100 milliseconds or so, but you may want to test it out in your own particular conditions to find a suitable value.

When doing select(), you should use *curl_multi_timeout(3)* to figure out how long to wait for action. Call *curl_multi_perform(3)* even if no activity has been seen on the **fd_sets** after the timeout expires as otherwise internal retries and timeouts may not work as you would think and want.

If one of the sockets used by libcurl happens to be larger than what can be set in an **fd_set**, which on POSIX systems means that the file descriptor is larger than **FD_SETSIZE**, then libcurl tries to not set it. Setting a too large file descriptor in an **fd_set** implies an out of bounds write which can cause crashes, or worse. The effect of NOT storing it might possibly save you from the crash, but makes your program NOT wait for sockets it should wait for...

EXAMPLE

```
int main(void)
{
  fd_set fdread;
  fd_set fdwrite;
  fd_set fdexcep;
  int maxfd;
  int rc;
  CURLMcode mc;
  struct timeval timeout = {1, 0};

  CURLM *multi = curl_multi_init();

  do {

    /* call curl_multi_perform() */

    /* get file descriptors from the transfers */
    mc = curl_multi_fdset(multi, &fdread, &fdwrite, &fdexcep, &maxfd);

    if(mc != CURLM_OK) {
      fprintf(stderr, "curl_multi_fdset() failed, code %d.\n", mc);
      break;
    }

    /* wait for activity on one of the sockets */
    rc = select(maxfd + 1, &fdread, &fdwrite, &fdexcep, &timeout);

  } while(!mc);
}
```

AVAILABILITY

Added in 7.9.6

RETURN VALUE

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

SEE ALSO

curl_multi_cleanup(3), **curl_multi_init**(3), **curl_multi_perform**(3), **curl_multi_timeout**(3), **curl_multi_wait**(3), **select**(2)