NAME

curl_multi_wait - polls on all easy handles in a multi handle

SYNOPSIS

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

CURLMcode curl_multi_wait(CURLM *multi_handle,
                          struct curl_waitfd extra_fds[],
                          unsigned int extra_nfds,
                          int timeout_ms,
                          int *numfds);
```

DESCRIPTION

curl_multi_wait(3) polls all file descriptors used by the curl easy handles contained in the given multi handle set. It blocks until activity is detected on at least one of the handles or *timeout_ms* has passed. Alternatively, if the multi handle has a pending internal timeout that has a shorter expiry time than *timeout_ms*, that shorter time is be used instead to make sure timeout accuracy is reasonably kept.

The calling application may pass additional *curl_waitfd* structures which are similar to *poll(2)*'s *pollfd* structure to be waited on in the same call.

On completion, if *numfds* is non-NULL, it gets populated with the total number of file descriptors on which interesting events occurred. This number can include both libcurl internal descriptors as well as descriptors provided in *extra_fds*.

If no extra file descriptors are provided and libcurl has no file descriptor to offer to wait for, this function returns immediately. (Consider using *curl_multi_poll(3)* to avoid this behavior.)

This function is encouraged to be used instead of select(3) when using the multi interface to allow applications to easier circumvent the common problem with 1024 maximum file descriptors.

curl_waitfd

```
struct curl_waitfd {
  curl_socket_t fd;
  short events;
  short revents;
};
```

CURL_WAIT_POLLIN

Bit flag to *curl_waitfd.events* indicating the socket should poll on read events such as new data received.

CURL_WAIT_POLLPRI

Bit flag to *curl_waitfd.events* indicating the socket should poll on high priority read events such as out of band data.

CURL_WAIT_POLLOUT

Bit flag to *curl_waitfd.events* indicating the socket should poll on write events such as the socket being clear to write without blocking.

EXAMPLE

```
int main(void)
{
  CURL *easy;
  CURLM *multi = curl_multi_init();
  int still_running;

  /* add the individual easy handle */
  curl_multi_add_handle(multi, easy);

  do {
    CURLMcode mc;
    int numfds;

    mc = curl_multi_perform(multi, &still_running);

    if(mc == CURLM_OK) {
      /* wait for activity, timeout or "nothing" */
      mc = curl_multi_wait(multi, NULL, 0, 1000, &numfds);
    }

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

  } while(still_running);

  curl_multi_remove_handle(multi, easy);
}
```

AVAILABILITY

This function was added in libcurl 7.28.0.

RETURN VALUE

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

SEE ALSO

curl_multi_fdset(3), **curl_multi_perform**(3), **curl_multi_poll**(3)