

5c1d3c55dd ▾

...

[mc](#) / [src](#) / [vfs](#) / [sftpfs](#) / [connection.c](#) **aborodin** Ticket #4179: code clean up before 4.8.27 release. ... History 4 contributors

558 lines (443 sloc) | 17.4 KB

...

```
1  /* Virtual File System: SFTP file system.
2     The internal functions: connections
3
4     Copyright (C) 2011-2021
5     Free Software Foundation, Inc.
6
7     Written by:
8     Ilia Maslakov <il.smind@gmail.com>, 2011
9     Slava Zanko <slavazanko@gmail.com>, 2011, 2012, 2013
10
11    This file is part of the Midnight Commander.
12
13    The Midnight Commander is free software: you can redistribute it
14    and/or modify it under the terms of the GNU General Public License as
15    published by the Free Software Foundation, either version 3 of the License,
16    or (at your option) any later version.
17
18    The Midnight Commander is distributed in the hope that it will be useful,
19    but WITHOUT ANY WARRANTY; without even the implied warranty of
20    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
21    GNU General Public License for more details.
22
23    You should have received a copy of the GNU General Public License
24    along with this program. If not, see <http://www.gnu.org/licenses/>.
25 */
26
27 #include <config.h>
28 #include <errno.h>
29
30 #include <netdb.h>          /* struct hostent */
31 #include <sys/socket.h>     /* AF_INET */
32 #include <netinet/in.h>     /* struct in_addr */
33 #ifdef HAVE_ARPA_INET_H
34 #include <arpa/inet.h>
35 #endif
36
37 #include <libssh2.h>
38 #include <libssh2_sftp.h>
39
40 #include "lib/global.h"
41
42 #include "lib/util.h"
43 #include "lib/tty/tty.h"    /* tty_enable_interrupt_key () */
44 #include "lib/vfs/utlrvfs.h"
45
46 #include "internal.h"
47
48 /** global variables *****/
49
50 /** file scope macro definitions *****/
51
52 /** file scope type declarations *****/
53
54 /** file scope variables *****/
55
56 static const char *kbi_passwd = NULL;
57 static const struct vfs_s_super *kbi_super = NULL;
58
59 /* ----- */
60 /** file scope functions *****/
61 /* ----- */
62 /**
63  * Create socket to host.
64  *
65  * @param super    connection data
66  * @param merror pointer to the error handler
67  * @return socket descriptor number, -1 if any error was occurred
68  */
69
70 static int
71 sftpfs_open_socket (struct vfs_s_super *super, GError ** merror)
72 {
73     struct addrinfo hints, *res = NULL, *curr_res;
74     int my_socket = 0;
75     char port[BUF_TINY];
76     int e;
77
78     mc_return_val_if_error (merror, LIBSSH2_INVALID_SOCKET);
```

```

79
80     if (super->path_element->host == NULL || *super->path_element->host == '\0')
81     {
82         mc_propagate_error (mcerror, 0, "%s", _("sftp: Invalid host name."));
83         return LIBSSH2_INVALID_SOCKET;
84     }
85
86     sprintf (port, "%hu", (unsigned short) super->path_element->port);
87
88     tty_enable_interrupt_key ();          /* clear the interrupt flag */
89
90     memset (&hints, 0, sizeof (hints));
91     hints.ai_family = AF_UNSPEC;
92     hints.ai_socktype = SOCK_STREAM;
93
94     #ifdef AI_ADDRCONFIG
95     /* By default, only look up addresses using address types for
96      * which a local interface is configured (i.e. no IPv6 if no IPv6
97      * interfaces, likewise for IPv4 (see RFC 3493 for details). */
98     hints.ai_flags = AI_ADDRCONFIG;
99     #endif
100
101     e = getaddrinfo (super->path_element->host, port, &hints, &res);
102
103     #ifdef AI_ADDRCONFIG
104     if (e == EAI_BADFLAGS)
105     {
106         /* Retry with no flags if AI_ADDRCONFIG was rejected. */
107         hints.ai_flags = 0;
108         e = getaddrinfo (super->path_element->host, port, &hints, &res);
109     }
110     #endif
111
112     if (e != 0)
113     {
114         mc_propagate_error (mcerror, e, _("sftp: %s"), gai_strerror (e));
115         my_socket = LIBSSH2_INVALID_SOCKET;
116         goto ret;
117     }
118
119     for (curr_res = res; curr_res != NULL; curr_res = curr_res->ai_next)
120     {
121         int save_errno;
122
123         my_socket = socket (curr_res->ai_family, curr_res->ai_socktype, curr_res->ai_protocol);
124
125         if (my_socket < 0)
126         {
127             if (curr_res->ai_next != NULL)
128                 continue;
129
130             vfs_print_message (_("sftp: %s"), unix_error_string (errno));
131             my_socket = LIBSSH2_INVALID_SOCKET;
132             goto ret;
133         }
134
135         vfs_print_message (_("sftp: making connection to %s"), super->path_element->host);
136
137         if (connect (my_socket, curr_res->ai_addr, curr_res->ai_addrlen) >= 0)
138             break;
139
140         save_errno = errno;
141
142         close (my_socket);
143
144         if (save_errno == EINTR && tty_got_interrupt ())
145             mc_propagate_error (mcerror, 0, "%s", _("sftp: connection interrupted by user"));
146         else if (res->ai_next == NULL)
147             mc_propagate_error (mcerror, save_errno, _("sftp: connection to server failed: %s"),
148                               unix_error_string (save_errno));
149         else
150             continue;
151
152         my_socket = LIBSSH2_INVALID_SOCKET;
153         break;
154     }
155
156 ret:
157     if (res != NULL)
158         freeaddrinfo (res);
159     tty_disable_interrupt_key ();
160     return my_socket;
161 }
162
163 /* ----- */
164 /**
165  * Recognize authentication types supported by remote side and filling internal 'super' structure by
166  * proper enum's values.
167  *
168  * @param super connection data
169  * @return TRUE if some of authentication methods is available, FALSE otherwise
170  */
171 static gboolean
172 sftpfs_recognize_auth_types (struct vfs_s_super *super)
173 {
174     char *userauthlist;
175     sftpfs_super_t *sftpfs_super = SFTP_SUPER (super);
176

```

```

177     /* check what authentication methods are available */
178     /* userauthlist is internally managed by libssh2 and freed by libssh2_session_free() */
179     userauthlist = libssh2_userauth_list (sftpfs_super->session, super->path_element->user,
180                                         strlen (super->path_element->user));
181
182     if (userauthlist == NULL)
183         return FALSE;
184
185     if ((strstr (userauthlist, "password") != NULL
186         || strstr (userauthlist, "keyboard-interactive") != NULL)
187         && (sftpfs_super->config_auth_type & PASSWORD) != 0)
188         sftpfs_super->auth_type |= PASSWORD;
189
190     if (strstr (userauthlist, "publickey") != NULL
191         && (sftpfs_super->config_auth_type & PUBKEY) != 0)
192         sftpfs_super->auth_type |= PUBKEY;
193
194     if ((sftpfs_super->config_auth_type & AGENT) != 0)
195         sftpfs_super->auth_type |= AGENT;
196
197     return TRUE;
198 }
199
200 /* ----- */
201 /**
202  * Open connection to host using SSH-agent helper.
203  *
204  * @param super    connection data
205  * @param merror pointer to the error handler
206  * @return TRUE if connection was successfully opened, FALSE otherwise
207  */
208
209 static gboolean
210 sftpfs_open_connection_ssh_agent (struct vfs_s_super *super, GError ** merror)
211 {
212     sftpfs_super_t *sftpfs_super = SFTP_SUPER (super);
213     struct libssh2_agent_publickey *identity, *prev_identity = NULL;
214     int rc;
215
216     mc_return_val_if_error (merror, FALSE);
217
218     sftpfs_super->agent = NULL;
219
220     if ((sftpfs_super->auth_type & AGENT) == 0)
221         return FALSE;
222
223     /* Connect to the ssh-agent */
224     sftpfs_super->agent = libssh2_agent_init (sftpfs_super->session);
225     if (sftpfs_super->agent == NULL)
226         return FALSE;
227
228     if (libssh2_agent_connect (sftpfs_super->agent) != 0)
229         return FALSE;
230
231     if (libssh2_agent_list_identities (sftpfs_super->agent) != 0)
232         return FALSE;
233
234     while (TRUE)
235     {
236         rc = libssh2_agent_get_identity (sftpfs_super->agent, &identity, prev_identity);
237         if (rc == 1)
238             break;
239
240         if (rc < 0)
241             return FALSE;
242
243         if (libssh2_agent_userauth (sftpfs_super->agent, super->path_element->user, identity) == 0)
244             break;
245
246         prev_identity = identity;
247     }
248
249     return (rc == 0);
250 }
251
252 /* ----- */
253 /**
254  * Open connection to host using SSH-keypair.
255  *
256  * @param super    connection data
257  * @param merror pointer to the error handler
258  * @return TRUE if connection was successfully opened, FALSE otherwise
259  */
260
261 static gboolean
262 sftpfs_open_connection_ssh_key (struct vfs_s_super *super, GError ** merror)
263 {
264     sftpfs_super_t *sftpfs_super = SFTP_SUPER (super);
265     char *p, *passwd;
266     gboolean ret_value = FALSE;
267
268     mc_return_val_if_error (merror, FALSE);
269
270     if ((sftpfs_super->auth_type & PUBKEY) == 0)
271         return FALSE;
272
273     if (sftpfs_super->xprivkey == NULL)
274         return FALSE;

```

```

275
276     if (libssh2_userauth_publickey_fromfile (sftpfs_super->session, super->path_element->user,
277                                             sftpfs_super->pubkey, sftpfs_super->privkey,
278                                             super->path_element->password) == 0)
279         return TRUE;
280
281     p = g_strdup_printf (_("sftp: Enter passphrase for %s "), super->path_element->user);
282     passwd = vfs_get_password (p);
283     g_free (p);
284
285     if (passwd == NULL)
286         mc_propagate_error (mcerror, 0, "%s", _("sftp: Passphrase is empty."));
287     else
288     {
289         ret_value = (libssh2_userauth_publickey_fromfile (sftpfs_super->session,
290                                                         super->path_element->user,
291                                                         sftpfs_super->pubkey,
292                                                         sftpfs_super->privkey, passwd) == 0);
293         g_free (passwd);
294     }
295
296     return ret_value;
297 }
298
299 /* ----- */
300
301 /**
302  * Keyboard-interactive password helper for opening connection to host by
303  * sftpfs_open_connection_ssh_password
304  *
305  * Uses global kbi_super (data with existing connection) and kbi_passwd (password)
306  *
307  * @param name      username
308  * @param name_len  length of @name
309  * @param instruction unused
310  * @param instruction_len unused
311  * @param num_prompts number of possible problems to process
312  * @param prompts   array of prompts to process
313  * @param responses array of responses, one per prompt
314  * @param abstract  unused
315  */
316
317 static
318 LIBSSH2_USERAUTH_KBDINT_RESPONSE_FUNC (sftpfs_keyboard_interactive_helper)
319 {
320     int i;
321     size_t len;
322
323     (void) instruction;
324     (void) instruction_len;
325     (void) abstract;
326
327     if (kbi_super == NULL || kbi_passwd == NULL)
328         return;
329
330     if (strcmp (name, kbi_super->path_element->user, name_len) != 0)
331         return;
332
333     /* assume these are password prompts */
334     len = strlen (kbi_passwd);
335
336     for (i = 0; i < num_prompts; ++i)
337         if (strcmp (prompts[i].text, "Password: ", prompts[i].length) == 0)
338         {
339             responses[i].text = strdup (kbi_passwd);
340             responses[i].length = len;
341         }
342 }
343
344 /* ----- */
345 /**
346  * Open connection to host using password.
347  *
348  * @param super      connection data
349  * @param mcerror pointer to the error handler
350  * @return TRUE if connection was successfully opened, FALSE otherwise
351  */
352
353 static gboolean
354 sftpfs_open_connection_ssh_password (struct vfs_s_super *super, GError ** mcerror)
355 {
356     sftpfs_super_t *sftpfs_super = SFTP_SUPER (super);
357     char *p, *passwd;
358     gboolean ret_value = FALSE;
359     int rc;
360
361     mc_return_val_if_error (mcerror, FALSE);
362
363     if ((sftpfs_super->auth_type & PASSWORD) == 0)
364         return FALSE;
365
366     if (super->path_element->password != NULL)
367     {
368         while ((rc = libssh2_userauth_password (sftpfs_super->session, super->path_element->user,
369                                                super->path_element->password)) ==
370              LIBSSH2_ERROR_EAGAIN);
371         if (rc == 0)
372             return TRUE;
373     }

```

```

373
374     kbi_super = super;
375     kbi_passwd = super->path_element->password;
376
377     while ((rc =
378             libssh2_userauth_keyboard_interactive (sftpfs_super->session,
379             super->path_element->user,
380             sftpfs_keyboard_interactive_helper)) ==
381             LIBSSH2_ERROR_EAGAIN)
382     ;
383
384     kbi_super = NULL;
385     kbi_passwd = NULL;
386
387     if (rc == 0)
388         return TRUE;
389 }
390
391 p = g_strdup_printf (_("sftp: Enter password for %s "), super->path_element->user);
392 passwd = vfs_get_password (p);
393 g_free (p);
394
395 if (passwd == NULL)
396     mc_propagate_error (mcerror, 0, "%s", _("sftp: Password is empty."));
397 else
398 {
399     while ((rc = libssh2_userauth_password (sftpfs_super->session, super->path_element->user,
400     passwd)) == LIBSSH2_ERROR_EAGAIN)
401     ;
402
403     if (rc != 0)
404     {
405         kbi_super = super;
406         kbi_passwd = passwd;
407
408         while ((rc =
409                 libssh2_userauth_keyboard_interactive (sftpfs_super->session,
410                 super->path_element->user,
411                 sftpfs_keyboard_interactive_helper)) ==
412                 LIBSSH2_ERROR_EAGAIN)
413         ;
414
415         kbi_super = NULL;
416         kbi_passwd = NULL;
417     }
418
419     if (rc == 0)
420     {
421         ret_value = TRUE;
422         g_free (super->path_element->password);
423         super->path_element->password = passwd;
424     }
425     else
426         g_free (passwd);
427 }
428
429 return ret_value;
430 }
431
432 /* ----- */
433 /** public functions ----- */
434 /* ----- */
435 /**
436  * Open new connection.
437  *
438  * @param super    connection data
439  * @param mcerror pointer to the error handler
440  * @return 0 if success, -1 otherwise
441  */
442
443 int
444 sftpfs_open_connection (struct vfs_s_super *super, GError ** mcerror)
445 {
446     int rc;
447     sftpfs_super_t *sftpfs_super = SFTPFS_SUPER (super);
448
449     mc_return_val_if_error (mcerror, -1);
450
451     /*
452      * The application code is responsible for creating the socket
453      * and establishing the connection
454      */
455     sftpfs_super->socket_handle = sftpfs_open_socket (super, mcerror);
456     if (sftpfs_super->socket_handle == LIBSSH2_INVALID_SOCKET)
457         return (-1);
458
459     /* Create a session instance */
460     sftpfs_super->session = libssh2_session_init ();
461     if (sftpfs_super->session == NULL)
462         return (-1);
463
464     /* ... start it up. This will trade welcome banners, exchange keys,
465      * and setup crypto, compression, and MAC layers
466      */
467     #if LIBSSH2_VERSION_NUM < 0x010208
468     rc = libssh2_session_startup (sftpfs_super->session, sftpfs_super->socket_handle);
469     #else
470     rc = libssh2_session_handshake (sftpfs_super->session,

```

```

471         (libssh2_socket_t) sftpfs_super->socket_handle);
472     #endif
473     if (rc != 0)
474     {
475         mc_propagate_error (mcerror, rc, "%s", _("sftp: Failure establishing SSH session"));
476         return (-1);
477     }
478
479     /* At this point we haven't yet authenticated. The first thing to do
480      * is check the hostkey's fingerprint against our known hosts Your app
481      * may have it hard coded, may go to a file, may present it to the
482      * user, that's your call
483      */
484     sftpfs_super->fingerprint =
485         libssh2_hostkey_hash (sftpfs_super->session, LIBSSH2_HOSTKEY_HASH_SHA1);
486
487     if (!sftpfs_recognize_auth_types (super))
488     {
489         int sftp_errno;
490
491         sftp_errno = libssh2_session_last_errno (sftpfs_super->session);
492         sftpfs_ssherror_to_gliberror (sftpfs_super, sftp_errno, mcerror);
493         return (-1);
494     }
495
496     if (!sftpfs_open_connection_ssh_agent (super, mcerror)
497         && !sftpfs_open_connection_ssh_key (super, mcerror)
498         && !sftpfs_open_connection_ssh_password (super, mcerror))
499         return (-1);
500
501     sftpfs_super->sftp_session = libssh2_sftp_init (sftpfs_super->session);
502
503     if (sftpfs_super->sftp_session == NULL)
504         return (-1);
505
506     /* Since we have not set non-blocking, tell libssh2 we are blocking */
507     libssh2_session_set_blocking (sftpfs_super->session, 1);
508
509     return 0;
510 }
511
512 /* ----- */
513 /**
514  * Close connection.
515  *
516  * @param super          connection data
517  * @param shutdown_message message for shutdown functions
518  * @param mcerror        pointer to the error handler
519  */
520
521 void
522 sftpfs_close_connection (struct vfs_s_super *super, const char *shutdown_message, GError ** mcerror)
523 {
524     sftpfs_super_t *sftpfs_super = SFTP_SUPER (super);
525
526     /* no mc_return_*_if_error() here because of abort open_connection handling too */
527     (void) mcerror;
528
529     if (sftpfs_super->sftp_session != NULL)
530     {
531         libssh2_sftp_shutdown (sftpfs_super->sftp_session);
532         sftpfs_super->sftp_session = NULL;
533     }
534
535     if (sftpfs_super->agent != NULL)
536     {
537         libssh2_agent_disconnect (sftpfs_super->agent);
538         libssh2_agent_free (sftpfs_super->agent);
539         sftpfs_super->agent = NULL;
540     }
541
542     sftpfs_super->fingerprint = NULL;
543
544     if (sftpfs_super->session != NULL)
545     {
546         libssh2_session_disconnect (sftpfs_super->session, shutdown_message);
547         libssh2_session_free (sftpfs_super->session);
548         sftpfs_super->session = NULL;
549     }
550
551     if (sftpfs_super->socket_handle != LIBSSH2_INVALID_SOCKET)
552     {
553         close (sftpfs_super->socket_handle);
554         sftpfs_super->socket_handle = LIBSSH2_INVALID_SOCKET;
555     }
556 }
557
558 /* ----- */

```