


master ▾

...

mc / src / vfs / sftpfs / connection.c

 **aborodin** (sftpfs\_read\_known\_hosts): clarify displayed value of unknown host ke... .. History4 contributors 

968 lines (803 sloc) | 31.2 KB ...

```
1  /* Virtual File System: SFTP file system.
2  The internal functions: connections
3
4  Copyright (C) 2011-2022
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 #include "lib/mcconfig.h"   /* mc_config_get_home_dir () */
46 #include "lib/widget.h"     /* query_dialog () */
47
48 #include "internal.h"
49
50 /** global variables *****/
51
52 /** file scope macro definitions *****/
53
54 #define SHA1_DIGEST_LENGTH 20
55
56 /** file scope type declarations *****/
57
58 /** file scope variables *****/
59
60 #ifdef LIBSSH2_KNOWNHOST_KEY_ED25519
61 static const char *const hostkey_method_ssh_ed25519 = "ssh-ed25519";
62 #endif
63 #ifdef LIBSSH2_KNOWNHOST_KEY_ECDSA_521
64 static const char *const hostkey_method_ssh_ecdsa_521 = "ecdsa-sha2-nistp521";
65 #endif
66 #ifdef LIBSSH2_KNOWNHOST_KEY_ECDSA_384
67 static const char *const hostkey_method_ssh_ecdsa_384 = "ecdsa-sha2-nistp384";
68 #endif
69 #ifdef LIBSSH2_KNOWNHOST_KEY_ECDSA_256
70 static const char *const hostkey_method_ssh_ecdsa_256 = "ecdsa-sha2-nistp256";
71 #endif
72 static const char *const hostkey_method_ssh_rsa = "ssh-rsa";
73 static const char *const hostkey_method_ssh_dss = "ssh-dss";
74
75 /**
76  *
77  * The current implementation of know host key checking has following limitations:
78  *
```

```

79  * - Only plain-text entries are supported ('HashKnownHosts no' OpenSSH option)
80  * - Only HEX-encoded SHA1 fingerprint display is supported ('FingerprintHash' OpenSSH option)
81  * - Resolved IP addresses are *not* saved/validated along with the hostnames
82  *
83  */
84
85  static const char *kbi_passwd = NULL;
86  static const struct vfs_s_super *kbi_super = NULL;
87
88  /* ----- */
89  /** file scope functions **** */
90  /* ----- */
91  /**
92   * Create socket to host.
93   *
94   * @param super    connection data
95   * @param merror pointer to the error handler
96   * @return socket descriptor number, -1 if any error was occurred
97   */
98
99  static int
100  sftpfs_open_socket (struct vfs_s_super *super, GError ** merror)
101  {
102      sftpfs_super_t *sftpfs_super = SFTP_SUPER (super);
103      struct addrinfo hints, *res = NULL, *curr_res;
104      int my_socket = 0;
105      char port[BUF_TINY];
106      static char address_ipv4[INET_ADDRSTRLEN];
107      static char address_ipv6[INET6_ADDRSTRLEN];
108      int e;
109
110      mc_return_val_if_error (merror, LIBSSH2_INVALID_SOCKET);
111
112      if (super->path_element->host == NULL || *super->path_element->host == '\0')
113      {
114          mc_propagate_error (merror, 0, "%s", _("sftp: Invalid host name."));
115          return LIBSSH2_INVALID_SOCKET;
116      }
117
118      sprintf (port, "%hu", (unsigned short) super->path_element->port);
119
120      tty_enable_interrupt_key ();      /* clear the interrupt flag */
121
122      memset (&hints, 0, sizeof (hints));
123      hints.ai_family = AF_UNSPEC;
124      hints.ai_socktype = SOCK_STREAM;
125
126      #ifdef AI_ADDRCONFIG
127      /* By default, only look up addresses using address types for
128       * which a local interface is configured (i.e. no IPv6 if no IPv6
129       * interfaces, likewise for IPv4 (see RFC 3493 for details). */
130      hints.ai_flags = AI_ADDRCONFIG;
131      #endif
132
133      e = getaddrinfo (super->path_element->host, port, &hints, &res);
134
135      #ifdef AI_ADDRCONFIG
136      if (e == EAI_BADFLAGS)
137      {
138          /* Retry with no flags if AI_ADDRCONFIG was rejected. */
139          hints.ai_flags = 0;
140          e = getaddrinfo (super->path_element->host, port, &hints, &res);
141      }
142      #endif
143
144      if (e != 0)
145      {
146          mc_propagate_error (merror, e, _("sftp: %s"), gai_strerror (e));
147          my_socket = LIBSSH2_INVALID_SOCKET;
148          goto ret;
149      }
150
151      for (curr_res = res; curr_res != NULL; curr_res = curr_res->ai_next)
152      {
153          int save_errno;
154
155          switch (curr_res->ai_addr->sa_family)
156          {
157              case AF_INET:
158                  sftpfs_super->ip_address =
159                      inet_ntop (AF_INET, &((struct sockaddr_in *) curr_res->ai_addr)->sin_addr,
160                                address_ipv4, INET_ADDRSTRLEN);
161                  break;
162              case AF_INET6:
163                  sftpfs_super->ip_address =
164                      inet_ntop (AF_INET6, &((struct sockaddr_in6 *) curr_res->ai_addr)->sin6_addr,
165                                address_ipv6, INET6_ADDRSTRLEN);
166                  break;
167              default:
168                  sftpfs_super->ip_address = NULL;
169          }
170
171          if (sftpfs_super->ip_address == NULL)
172          {
173              mc_propagate_error (merror, 0, "%s",
174                                  _("sftp: failed to convert remote host IP address into text form"));
175              my_socket = LIBSSH2_INVALID_SOCKET;
176              goto ret;
177          }
178      }

```

```

177     }
178
179     my_socket = socket (curr_res->ai_family, curr_res->ai_socktype, curr_res->ai_protocol);
180
181     if (my_socket < 0)
182     {
183         if (curr_res->ai_next != NULL)
184             continue;
185
186         vfs_print_message (_("sftp: %s"), unix_error_string (errno));
187         my_socket = LIBSSH2_INVALID_SOCKET;
188         goto ret;
189     }
190
191     vfs_print_message (_("sftp: making connection to %s"), super->path_element->host);
192
193     if (connect (my_socket, curr_res->ai_addr, curr_res->ai_addrlen) >= 0)
194         break;
195
196     save_errno = errno;
197
198     close (my_socket);
199
200     if (save_errno == EINTR && tty_got_interrupt ())
201         mc_propagate_error (mcerror, 0, "%s", _("sftp: connection interrupted by user"));
202     else if (res->ai_next == NULL)
203         mc_propagate_error (mcerror, save_errno, _("sftp: connection to server failed: %s"),
204                             unix_error_string (save_errno));
205     else
206         continue;
207
208     my_socket = LIBSSH2_INVALID_SOCKET;
209     break;
210 }
211
212 ret:
213 if (res != NULL)
214     freeaddrinfo (res);
215 tty_disable_interrupt_key ();
216 return my_socket;
217 }
218
219 /* ----- */
220
221 /**
222  * Read ~/.ssh/known_hosts file.
223  *
224  * @param super connection data
225  * @param mcerror pointer to the error handler
226  * @return TRUE on success, FALSE otherwise
227  *
228  * Thanks the Curl project for the code used in this function.
229  */
230 static gboolean
231 sftpfs_read_known_hosts (struct vfs_s_super *super, GError ** mcerror)
232 {
233     sftpfs_super_t *sftpfs_super = SFTP_SUPER (super);
234     struct libssh2_knownhost *store = NULL;
235     int rc;
236     gboolean found = FALSE;
237
238     sftpfs_super->known_hosts = libssh2_knownhost_init (sftpfs_super->session);
239     if (sftpfs_super->known_hosts == NULL)
240         goto err;
241
242     sftpfs_super->known_hosts_file =
243         mc_build_filename (mc_config_get_home_dir (), ".ssh", "known_hosts", (char *) NULL);
244     rc = libssh2_knownhost_readfile (sftpfs_super->known_hosts, sftpfs_super->known_hosts_file,
245                                     LIBSSH2_KNOWNHOST_FILE_OPENSSH);
246
247     if (rc > 0)
248     {
249         const char *kh_name_end = NULL;
250
251         while (!found && libssh2_knownhost_get (sftpfs_super->known_hosts, &store, store) == 0)
252         {
253             /* For non-standard ports, the name will be enclosed in
254              * square brackets, followed by a colon and the port */
255             if (store == NULL)
256                 continue;
257
258             if (store->name == NULL)
259                 found = TRUE;
260             else if (store->name[0] != '[')
261                 found = strcmp (store->name, super->path_element->host) == 0;
262             else
263             {
264                 int port;
265
266                 kh_name_end = strstr (store->name, "]:");
267                 if (kh_name_end == NULL)
268                     /* Invalid host pattern */
269                     continue;
270
271                 port = (int) g_ascii_strtoll (kh_name_end + 2, NULL, 10);
272                 if (port == super->path_element->port)
273                 {
274                     size_t kh_name_size;

```

```

275         kh_name_size = strlen (store->name) - 1 - strlen (kh_name_end);
276         found = strncmp (store->name + 1, super->path_element->host, kh_name_size) == 0;
277     }
278 }
279 }
280 }
281
282 if (found)
283 {
284     int mask;
285     const char *hostkey_method = NULL;
286
287     mask = store->typemask & LIBSSH2_KNOWNHOST_KEY_MASK;
288
289     switch (mask)
290     {
291 #ifdef LIBSSH2_KNOWNHOST_KEY_ED25519
292         case LIBSSH2_KNOWNHOST_KEY_ED25519:
293             hostkey_method = hostkey_method_ssh_ed25519;
294             break;
295 #endif
296 #ifdef LIBSSH2_KNOWNHOST_KEY_ECDSA_521
297         case LIBSSH2_KNOWNHOST_KEY_ECDSA_521:
298             hostkey_method = hostkey_method_ssh_ecdsa_521;
299             break;
300 #endif
301 #ifdef LIBSSH2_KNOWNHOST_KEY_ECDSA_384
302         case LIBSSH2_KNOWNHOST_KEY_ECDSA_384:
303             hostkey_method = hostkey_method_ssh_ecdsa_384;
304             break;
305 #endif
306 #ifdef LIBSSH2_KNOWNHOST_KEY_ECDSA_256
307         case LIBSSH2_KNOWNHOST_KEY_ECDSA_256:
308             hostkey_method = hostkey_method_ssh_ecdsa_256;
309             break;
310 #endif
311         case LIBSSH2_KNOWNHOST_KEY_SSHRSA:
312             hostkey_method = hostkey_method_ssh_rsa;
313             break;
314         case LIBSSH2_KNOWNHOST_KEY_SSHDSS:
315             hostkey_method = hostkey_method_ssh_dss;
316             break;
317         case LIBSSH2_KNOWNHOST_KEY_RSA1:
318             mc_propagate_error (mcerror, 0, "%s",
319                               _("sftp: found host key of unsupported type: RSA1"));
320             return FALSE;
321         default:
322             mc_propagate_error (mcerror, 0, "%s %u", _("sftp: unknown host key type:"),
323                               (unsigned int) mask);
324             return FALSE;
325     }
326
327     rc = libssh2_session_method_pref (sftpfs_super->session, LIBSSH2_METHOD_HOSTKEY,
328                                     hostkey_method);
329
330     if (rc < 0)
331         goto err;
332 }
333
334 return TRUE;
335
336 err:
337 {
338     int sftp_errno;
339
340     sftp_errno = libssh2_session_last_errno (sftpfs_super->session);
341     sftpfs_ssherror_to_gliberror (sftpfs_super, sftp_errno, mcerror);
342 }
343
344 return FALSE;
345 }
346
347 /* ----- */
348
349 /**
350  * Write new host + key pair to the ~/.ssh/known_hosts file.
351  *
352  * @param super connection data
353  * @param remote_key the key for the remote host
354  * @param remote_key_len length of @remote_key
355  * @param type_mask info about format of host name, key and key type
356  * @return 0 on success, regular libssh2 error code otherwise
357  *
358  * Thanks the Curl project for the code used in this function.
359  */
360 static int
361 sftpfs_update_known_hosts (struct vfs_s_super *super, const char *remote_key, size_t remote_key_len,
362                           int type_mask)
363 {
364     sftpfs_super_t *sftpfs_super = SFTPFS_SUPER (super);
365     int rc;
366
367     /* add this host + key pair */
368     rc = libssh2_knownhost_addc (sftpfs_super->known_hosts, super->path_element->host, NULL,
369                                remote_key, remote_key_len, NULL, 0, type_mask, NULL);
370
371     if (rc < 0)
372         return rc;
373
374     /* write the entire in-memory list of known hosts to the known_hosts file */
375     rc = libssh2_knownhost_writefile (sftpfs_super->known_hosts, sftpfs_super->known_hosts_file,

```

```

373 LIBSSH2_KNOWNHOST_FILE_OPENSSSH);
374
375 if (rc < 0)
376     return rc;
377
378 (void) message (D_NORMAL, _("Information"),
379               _("Permanently added\n%s (%s)\nto the list of known hosts."),
380               super->path_element->host, sftpfs_super->ip_address);
381
382     return 0;
383 }
384
385 /* ----- */
386 /**
387  * Compute and return readable host key fingerprint hash.
388  *
389  * @param session libssh2 session handle
390  * @return pointer to static buffer on success, NULL otherwise
391  */
392 static const char *
393 sftpfs_compute_fingerprint_hash (LIBSSH2_SESSION * session)
394 {
395     static char result[SHA1_DIGEST_LENGTH * 3 + 1]; /* "XX:" for each byte, and EOL */
396     const char *fingerprint;
397     size_t i;
398
399     /* The fingerprint points to static storage (!), don't free() it. */
400     fingerprint = libssh2_hostkey_hash (session, LIBSSH2_HOSTKEY_HASH_SHA1);
401     if (fingerprint == NULL)
402         return NULL;
403
404     for (i = 0; i < SHA1_DIGEST_LENGTH && i * 3 < sizeof (result) - 1; i++)
405         g_snprintf ((gchar *) (result + i * 3), 4, "%02x:", (guint8) fingerprint[i]);
406
407     /* remove last ":" */
408     result[i * 3 - 1] = '\0';
409
410     return result;
411 }
412
413 /* ----- */
414 /**
415  * Process host info found in ~/.ssh/known_hosts file.
416  *
417  * @param super connection data
418  * @param merror pointer to the error handler
419  * @return TRUE on success, FALSE otherwise
420  *
421  * Thanks the Curl project for the code used in this function.
422  */
423 static gboolean
424 sftpfs_process_known_host (struct vfs_s_super *super, GError ** merror)
425 {
426     sftpfs_super_t *sftpfs_super = SFTP_SUPER (super);
427     const char *remote_key;
428     const char *key_type;
429     const char *fingerprint_hash;
430     size_t remote_key_len = 0;
431     int remote_key_type = LIBSSH2_HOSTKEY_TYPE_UNKNOWN;
432     int keybit = 0;
433     struct libssh2_knownhost *host = NULL;
434     int rc;
435     char *msg = NULL;
436     gboolean handle_query = FALSE;
437
438     remote_key = libssh2_session_hostkey (sftpfs_super->session, &remote_key_len, &remote_key_type);
439     if (remote_key == NULL || remote_key_len == 0
440         || remote_key_type == LIBSSH2_HOSTKEY_TYPE_UNKNOWN)
441     {
442         mc_propagate_error (merror, 0, "%s", _("sftp: cannot get the remote host key"));
443         return FALSE;
444     }
445
446     switch (remote_key_type)
447     {
448     case LIBSSH2_HOSTKEY_TYPE_RSA:
449         keybit = LIBSSH2_KNOWNHOST_KEY_SSHRSA;
450         key_type = "RSA";
451         break;
452     case LIBSSH2_HOSTKEY_TYPE_DSS:
453         keybit = LIBSSH2_KNOWNHOST_KEY_SSHDSS;
454         key_type = "DSS";
455         break;
456     #ifdef LIBSSH2_HOSTKEY_TYPE_ECDSA_256
457     case LIBSSH2_HOSTKEY_TYPE_ECDSA_256:
458         keybit = LIBSSH2_KNOWNHOST_KEY_ECDSA_256;
459         key_type = "ECDSA";
460         break;
461     #endif
462     #ifdef LIBSSH2_HOSTKEY_TYPE_ECDSA_384
463     case LIBSSH2_HOSTKEY_TYPE_ECDSA_384:
464         keybit = LIBSSH2_KNOWNHOST_KEY_ECDSA_384;
465         key_type = "ECDSA";
466         break;
467     #endif
468     #ifdef LIBSSH2_HOSTKEY_TYPE_ECDSA_521
469     case LIBSSH2_HOSTKEY_TYPE_ECDSA_521:
470

```

```

471     keybit = LIBSSH2_KNOWNHOST_KEY_ECDSA_521;
472     key_type = "ECDSA";
473     break;
474 #endif
475 #ifdef LIBSSH2_HOSTKEY_TYPE_ED25519
476     case LIBSSH2_HOSTKEY_TYPE_ED25519:
477         keybit = LIBSSH2_KNOWNHOST_KEY_ED25519;
478         key_type = "ED25519";
479         break;
480 #endif
481     default:
482         mc_propagate_error (mcerror, 0, "%s",
483                             _("sftp: unsupported key type, can't check remote host key"));
484         return FALSE;
485     }
486
487     fingerprint_hash = sftpfs_compute_fingerprint_hash (sftpfs_super->session);
488     if (fingerprint_hash == NULL)
489     {
490         mc_propagate_error (mcerror, 0, "%s", _("sftp: can't compute host key fingerprint hash"));
491         return FALSE;
492     }
493
494     rc = libssh2_knownhost_checkp (sftpfs_super->known_hosts, super->path_element->host,
495                                   super->path_element->port, remote_key, remote_key_len,
496                                   LIBSSH2_KNOWNHOST_TYPE_PLAIN | LIBSSH2_KNOWNHOST_KEYENC_RAW |
497                                   keybit, &host);
498
499     switch (rc)
500     {
501     default:
502     case LIBSSH2_KNOWNHOST_CHECK_FAILURE:
503         /* something prevented the check to be made */
504         goto err;
505
506     case LIBSSH2_KNOWNHOST_CHECK_MATCH:
507         /* host + key pair matched -- OK */
508         break;
509
510     case LIBSSH2_KNOWNHOST_CHECK_NOTFOUND:
511         /* no host match was found -- add it to the known_hosts file */
512         msg = g_strdup_printf (_("The authenticity of host\n%s (%s)\n"
513                                "can't be established\n"
514                                "%s key fingerprint hash is\nSHA1:%s.\n"
515                                "Do you want to add it to the list of known hosts and continue connecting?"),
516                                super->path_element->host, sftpfs_super->ip_address,
517                                key_type, fingerprint_hash);
518         /* Select "No" initially */
519         query_set_sel (2);
520         rc = query_dialog (_("Warning"), msg, D_NORMAL, 3, _("&Yes"), _("&Ignore"), _("&No"));
521         g_free (msg);
522         handle_query = TRUE;
523         break;
524
525     case LIBSSH2_KNOWNHOST_CHECK_MISMATCH:
526         msg = g_strdup_printf (_("%s (%s)\n"
527                                "is found in the list of known hosts but\n"
528                                "KEYS DO NOT MATCH! THIS COULD BE A MITM ATTACK!\n"
529                                "Are you sure you want to add it to the list of known hosts and continue connecting?"),
530                                super->path_element->host, sftpfs_super->ip_address);
531         /* Select "No" initially */
532         query_set_sel (2);
533         rc = query_dialog (MSG_ERROR, msg, D_ERROR, 3, _("&Yes"), _("&Ignore"), _("&No"));
534         g_free (msg);
535         handle_query = TRUE;
536         break;
537     }
538
539     if (handle_query)
540     {
541         switch (rc)
542         {
543         case 0:
544             /* Yes: add this host + key pair, continue connecting */
545             if (sftpfs_update_known_hosts (super, remote_key, remote_key_len,
546                                           LIBSSH2_KNOWNHOST_TYPE_PLAIN
547                                           | LIBSSH2_KNOWNHOST_KEYENC_RAW | keybit) < 0)
548             {
549                 goto err;
550             }
551             break;
552         case 1:
553             /* Ignore: do not add this host + key pair, continue connecting anyway */
554             break;
555         case 2:
556         default:
557             mc_propagate_error (mcerror, 0, "%s", _("sftp: host key verification failed"));
558             /* No: abort connection */
559             goto err;
560         }
561     }
562
563     return TRUE;
564
565 err:
566     {
567         int sftp_errno;
568
569         sftp_errno = libssh2_session_last_errno (sftpfs_super->session);
570         sftpfs_ssherror_to_gliberror (sftpfs_super, sftp_errno, mcerror);
571     }
572
573     return FALSE;
574 }

```

```

569
570 /* ----- */
571 /**
572  * Recognize authentication types supported by remote side and filling internal 'super' structure by
573  * proper enum's values.
574  *
575  * @param super connection data
576  * @return TRUE if some of authentication methods is available, FALSE otherwise
577  */
578 static gboolean
579 sftpfs_recognize_auth_types (struct vfs_s_super *super)
580 {
581     char *userauthlist;
582     sftpfs_super_t *sftpfs_super = SFTP_SUPER (super);
583
584     /* check what authentication methods are available */
585     /* userauthlist is internally managed by libssh2 and freed by libssh2_session_free() */
586     userauthlist = libssh2_userauth_list (sftpfs_super->session, super->path_element->user,
587                                         strlen (super->path_element->user));
588
589     if (userauthlist == NULL)
590         return FALSE;
591
592     if ((strstr (userauthlist, "password") != NULL
593          || strstr (userauthlist, "keyboard-interactive") != NULL)
594         && (sftpfs_super->config_auth_type & PASSWORD) != 0)
595         sftpfs_super->auth_type |= PASSWORD;
596
597     if (strstr (userauthlist, "publickey") != NULL
598         && (sftpfs_super->config_auth_type & PUBKEY) != 0)
599         sftpfs_super->auth_type |= PUBKEY;
600
601     if ((sftpfs_super->config_auth_type & AGENT) != 0)
602         sftpfs_super->auth_type |= AGENT;
603
604     return TRUE;
605 }
606
607 /* ----- */
608 /**
609  * Open connection to host using SSH-agent helper.
610  *
611  * @param super connection data
612  * @param mcerror pointer to the error handler
613  * @return TRUE if connection was successfully opened, FALSE otherwise
614  */
615
616 static gboolean
617 sftpfs_open_connection_ssh_agent (struct vfs_s_super *super, GError ** mcerror)
618 {
619     sftpfs_super_t *sftpfs_super = SFTP_SUPER (super);
620     struct libssh2_agent_publickey *identity, *prev_identity = NULL;
621     int rc;
622
623     mc_return_val_if_error (mcerror, FALSE);
624
625     sftpfs_super->agent = NULL;
626
627     if ((sftpfs_super->auth_type & AGENT) == 0)
628         return FALSE;
629
630     /* Connect to the ssh-agent */
631     sftpfs_super->agent = libssh2_agent_init (sftpfs_super->session);
632     if (sftpfs_super->agent == NULL)
633         return FALSE;
634
635     if (libssh2_agent_connect (sftpfs_super->agent) != 0)
636         return FALSE;
637
638     if (libssh2_agent_list_identities (sftpfs_super->agent) != 0)
639         return FALSE;
640
641     while (TRUE)
642     {
643         rc = libssh2_agent_get_identity (sftpfs_super->agent, &identity, prev_identity);
644         if (rc == 1)
645             break;
646
647         if (rc < 0)
648             return FALSE;
649
650         if (libssh2_agent_userauth (sftpfs_super->agent, super->path_element->user, identity) == 0)
651             break;
652
653         prev_identity = identity;
654     }
655
656     return (rc == 0);
657 }
658
659 /* ----- */
660 /**
661  * Open connection to host using SSH-keypair.
662  *
663  * @param super connection data
664  * @param mcerror pointer to the error handler
665  * @return TRUE if connection was successfully opened, FALSE otherwise
666  */

```

```

667
668 static gboolean
669 sftpfs_open_connection_ssh_key (struct vfs_s_super *super, GError ** mcerror)
670 {
671     sftpfs_super_t *sftpfs_super = SFTP_SUPER (super);
672     char *p, *passwd;
673     gboolean ret_value = FALSE;
674
675     mc_return_val_if_error (mcerror, FALSE);
676
677     if ((sftpfs_super->auth_type & PUBKEY) == 0)
678         return FALSE;
679
680     if (sftpfs_super->privkey == NULL)
681         return FALSE;
682
683     if (libssh2_userauth_publickey_fromfile (sftpfs_super->session, super->path_element->user,
684                                             sftpfs_super->pubkey, sftpfs_super->privkey,
685                                             super->path_element->password) == 0)
686         return TRUE;
687
688     p = g_strdup_printf (_("sftp: Enter passphrase for %s "), super->path_element->user);
689     passwd = vfs_get_password (p);
690     g_free (p);
691
692     if (passwd == NULL)
693         mc_propagate_error (mcerror, 0, "%s", _("sftp: Passphrase is empty."));
694     else
695     {
696         ret_value = (libssh2_userauth_publickey_fromfile (sftpfs_super->session,
697                                                         super->path_element->user,
698                                                         sftpfs_super->pubkey,
699                                                         sftpfs_super->privkey, passwd) == 0);
700         g_free (passwd);
701     }
702     return ret_value;
703 }
704
705 /* ----- */
706
707 /**
708  * Keyboard-interactive password helper for opening connection to host by
709  * sftpfs_open_connection_ssh_password
710  *
711  * Uses global kbi_super (data with existing connection) and kbi_passwd (password)
712  *
713  * @param name      username
714  * @param name_len  length of @name
715  * @param instruction unused
716  * @param instruction_len unused
717  * @param num_prompts number of possible problems to process
718  * @param prompts   array of prompts to process
719  * @param responses array of responses, one per prompt
720  * @param abstract  unused
721  */
722
723 static
724 LIBSSH2_USERAUTH_KBDINT_RESPONSE_FUNC (sftpfs_keyboard_interactive_helper)
725 {
726     int i;
727     size_t len;
728
729     (void) instruction;
730     (void) instruction_len;
731     (void) abstract;
732
733     if (kbi_super == NULL || kbi_passwd == NULL)
734         return;
735
736     if (strcmp (name, kbi_super->path_element->user, name_len) != 0)
737         return;
738
739     /* assume these are password prompts */
740     len = strlen (kbi_passwd);
741
742     for (i = 0; i < num_prompts; ++i)
743         if (strcmp (prompts[i].text, "Password: ", prompts[i].length) == 0)
744         {
745             responses[i].text = strdup (kbi_passwd);
746             responses[i].length = len;
747         }
748 }
749
750 /* ----- */
751
752 /**
753  * Open connection to host using password.
754  *
755  * @param super      connection data
756  * @param mcerror pointer to the error handler
757  * @return TRUE if connection was successfully opened, FALSE otherwise
758  */
759
760 static gboolean
761 sftpfs_open_connection_ssh_password (struct vfs_s_super *super, GError ** mcerror)
762 {
763     sftpfs_super_t *sftpfs_super = SFTP_SUPER (super);
764     char *p, *passwd;

```



```

765     gboolean ret_value = FALSE;
766     int rc;
767
768     mc_return_val_if_error (merror, FALSE);
769
770     if ((sftpfs_super->auth_type & PASSWORD) == 0)
771         return FALSE;
772
773     if (super->path_element->password != NULL)
774     {
775         while ((rc = libssh2_userauth_password (sftpfs_super->session, super->path_element->user,
776                                                 super->path_element->password)) ==
777                LIBSSH2_ERROR_EAGAIN);
778         if (rc == 0)
779             return TRUE;
780
781         kbi_super = super;
782         kbi_passwd = super->path_element->password;
783
784         while ((rc =
785                 libssh2_userauth_keyboard_interactive (sftpfs_super->session,
786                                                         super->path_element->user,
787                                                         sftpfs_keyboard_interactive_helper)) ==
788                LIBSSH2_ERROR_EAGAIN)
789             ;
790
791         kbi_super = NULL;
792         kbi_passwd = NULL;
793
794         if (rc == 0)
795             return TRUE;
796     }
797
798     p = g_strdup_printf (_("sftp: Enter password for %s "), super->path_element->user);
799     passwd = vfs_get_password (p);
800     g_free (p);
801
802     if (passwd == NULL)
803         mc_propagate_error (merror, 0, "%s", _("sftp: Password is empty."));
804     else
805     {
806         while ((rc = libssh2_userauth_password (sftpfs_super->session, super->path_element->user,
807                                                 passwd)) == LIBSSH2_ERROR_EAGAIN)
808             ;
809
810         if (rc != 0)
811         {
812             kbi_super = super;
813             kbi_passwd = passwd;
814
815             while ((rc =
816                     libssh2_userauth_keyboard_interactive (sftpfs_super->session,
817                                                             super->path_element->user,
818                                                             sftpfs_keyboard_interactive_helper)) ==
819                    LIBSSH2_ERROR_EAGAIN)
820                 ;
821
822             kbi_super = NULL;
823             kbi_passwd = NULL;
824         }
825
826         if (rc == 0)
827         {
828             ret_value = TRUE;
829             g_free (super->path_element->password);
830             super->path_element->password = passwd;
831         }
832         else
833             g_free (passwd);
834     }
835
836     return ret_value;
837 }
838
839 /* ----- */
840 /** public functions ----- */
841 /* ----- */
842 /**
843  * Open new connection.
844  *
845  * @param super    connection data
846  * @param merror pointer to the error handler
847  * @return 0 if success, -1 otherwise
848  */
849
850 int
851 sftpfs_open_connection (struct vfs_s_super *super, GError ** merror)
852 {
853     int rc;
854     sftpfs_super_t *sftpfs_super = SFTP_SUPER (super);
855
856     mc_return_val_if_error (merror, -1);
857
858     /*
859      * The application code is responsible for creating the socket
860      * and establishing the connection
861      */
862     sftpfs_super->socket_handle = sftpfs_open_socket (super, merror);

```

```

863     if (sftpfs_super->socket_handle == LIBSSH2_INVALID_SOCKET)
864         return (-1);
865
866     /* Create a session instance */
867     sftpfs_super->session = libssh2_session_init ();
868     if (sftpfs_super->session == NULL)
869         return (-1);
870
871     if (!sftpfs_read_known_hosts (super, mcerror))
872         return (-1);
873
874     /* ... start it up. This will trade welcome banners, exchange keys,
875      * and setup crypto, compression, and MAC layers
876      */
877     while ((rc =
878             libssh2_session_handshake (sftpfs_super->session,
879                                       (libssh2_socket_t) sftpfs_super->socket_handle)) ==
880            LIBSSH2_ERROR_EAGAIN)
881     ;
882     if (rc != 0)
883     {
884         mc_propagate_error (mcerror, rc, "%s", _("sftp: failure establishing SSH session"));
885         return (-1);
886     }
887
888     if (!sftpfs_process_known_host (super, mcerror))
889         return (-1);
890
891     if (!sftpfs_recognize_auth_types (super))
892     {
893         int sftp_errno;
894
895         sftp_errno = libssh2_session_last_errno (sftpfs_super->session);
896         sftpfs_ssherror_to_gliberror (sftpfs_super, sftp_errno, mcerror);
897         return (-1);
898     }
899
900     if (!sftpfs_open_connection_ssh_agent (super, mcerror)
901         && !sftpfs_open_connection_ssh_key (super, mcerror)
902         && !sftpfs_open_connection_ssh_password (super, mcerror))
903         return (-1);
904
905     sftpfs_super->sftp_session = libssh2_sftp_init (sftpfs_super->session);
906
907     if (sftpfs_super->sftp_session == NULL)
908         return (-1);
909
910     /* Since we have not set non-blocking, tell libssh2 we are blocking */
911     libssh2_session_set_blocking (sftpfs_super->session, 1);
912
913     return 0;
914 }
915
916 /* ----- */
917 /**
918  * Close connection.
919  *
920  * @param super      connection data
921  * @param shutdown_message message for shutdown functions
922  * @param mcerror     pointer to the error handler
923  */
924
925 void
926 sftpfs_close_connection (struct vfs_s_super *super, const char *shutdown_message, GError ** mcerror)
927 {
928     sftpfs_super_t *sftpfs_super = SFTP_SUPER (super);
929
930     /* no mc_return_*_if_error() here because of abort open_connection handling too */
931     (void) mcerror;
932
933     if (sftpfs_super->sftp_session != NULL)
934     {
935         libssh2_sftp_shutdown (sftpfs_super->sftp_session);
936         sftpfs_super->sftp_session = NULL;
937     }
938
939     if (sftpfs_super->agent != NULL)
940     {
941         libssh2_agent_disconnect (sftpfs_super->agent);
942         libssh2_agent_free (sftpfs_super->agent);
943         sftpfs_super->agent = NULL;
944     }
945
946     if (sftpfs_super->known_hosts != NULL)
947     {
948         libssh2_knownhost_free (sftpfs_super->known_hosts);
949         sftpfs_super->known_hosts = NULL;
950     }
951
952     MC_PTR_FREE (sftpfs_super->known_hosts_file);
953
954     if (sftpfs_super->session != NULL)
955     {
956         libssh2_session_disconnect (sftpfs_super->session, shutdown_message);
957         libssh2_session_free (sftpfs_super->session);
958         sftpfs_super->session = NULL;
959     }
960

```

```
961     if (sftpfs_super->socket_handle != LIBSSH2_INVALID_SOCKET)
962     {
963         close (sftpfs_super->socket_handle);
964         sftpfs_super->socket_handle = LIBSSH2_INVALID_SOCKET;
965     }
966 }
967
968 /* ----- */
```