968 lines (803 sloc) | 31.2 KB

```c
/* Virtual File System: SFTP file system.
   The internal functions: connections

   Copyright (C) 2011-2022
   Free Software Foundation, Inc.

   Written by:
   Ilia Maslakov <il.smind@gmail.com>, 2011
   Slava Zanko <slavazanko@gmail.com>, 2011, 2012, 2013

   This file is part of the Midnight Commander.

   The Midnight Commander is free software: you can redistribute it
   and/or modify it under the terms of the GNU General Public License as
   published by the Free Software Foundation, either version 3 of the License,
   or (at your option) any later version.

   The Midnight Commander is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#include <config.h>
#include <errno.h>

#include <netdb.h>              /* struct hostent */
#include <sys/socket.h>         /* AF_INET */
#include <netinet/in.h>         /* struct in_addr */
#ifdef HAVE_ARPA_INET_H
#include <arpa/inet.h>
#endif

#include <libssh2.h>
#include <libssh2_sftp.h>

#include "lib/global.h"

#include "lib/util.h"
#include "lib/tty/tty.h"        /* tty_enable_interrupt_key () */
#include "lib/vfs/utilvfs.h"
#include "lib/mcconfig.h"       /* mc_config_get_home_dir () */
#include "lib/widget.h"         /* query_dialog () */

#include "internal.h"

/*** global variables ************************************************************************/

/*** file scope macro definitions ************************************************************/

#define SHA1_DIGEST_LENGTH 20

/*** file scope type declarations ************************************************************/

/*** file scope variables ********************************************************************/

#ifdef LIBSSH2_KNOWNHOST_KEY_ED25519
static const char *const hostkey_method_ssh_ed25519 = "ssh-ed25519";
#endif
#ifdef LIBSSH2_KNOWNHOST_KEY_ECDSA_521
static const char *const hostkey_method_ssh_ecdsa_521 = "ecdsa-sha2-nistp521";
#endif
#ifdef LIBSSH2_KNOWNHOST_KEY_ECDSA_384
static const char *const hostkey_method_ssh_ecdsa_384 = "ecdsa-sha2-nistp384";
#endif
#ifdef LIBSSH2_KNOWNHOST_KEY_ECDSA_256
static const char *const hostkey_method_ssh_ecdsa_256 = "ecdsa-sha2-nistp256";
#endif
static const char *const hostkey_method_ssh_rsa = "ssh-rsa";
static const char *const hostkey_method_ssh_dss = "ssh-dss";

/**
 *
 * The current implementation of know host key checking has following limitations:
 *
```

```
 *   - Only plain-text entries are supported (`HashKnownHosts no` OpenSSH option)
 *   - Only HEX-encoded SHA1 fingerprint display is supported (`FingerprintHash` OpenSSH option)
 *   - Resolved IP addresses are *not* saved/validated along with the hostnames
 *
 */

static const char *kbi_passwd = NULL;
static const struct vfs_s_super *kbi_super = NULL;

/* ------------------------------------------------------------------------------------------ */
/*** file scope functions ******************************************************************/
/* ------------------------------------------------------------------------------------------ */
/**
 * Create socket to host.
 *
 * @param super   connection data
 * @param mcerror pointer to the error handler
 * @return socket descriptor number, -1 if any error was occurred
 */

static int
sftpfs_open_socket (struct vfs_s_super *super, GError ** mcerror)
{
    sftpfs_super_t *sftpfs_super = SFTP_SUPER (super);
    struct addrinfo hints, *res = NULL, *curr_res;
    int my_socket = 0;
    char port[BUF_TINY];
    static char address_ipv4[INET_ADDRSTRLEN];
    static char address_ipv6[INET6_ADDRSTRLEN];
    int e;

    mc_return_val_if_error (mcerror, LIBSSH2_INVALID_SOCKET);

    if (super->path_element->host == NULL || *super->path_element->host == '\0')
    {
        mc_propagate_error (mcerror, 0, "%s", _("sftp: Invalid host name."));
        return LIBSSH2_INVALID_SOCKET;
    }

    sprintf (port, "%hu", (unsigned short) super->path_element->port);

    tty_enable_interrupt_key ();        /* clear the interrupt flag */

    memset (&hints, 0, sizeof (hints));
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;

#ifdef AI_ADDRCONFIG
    /* By default, only look up addresses using address types for
     * which a local interface is configured (i.e. no IPv6 if no IPv6
     * interfaces, likewise for IPv4 (see RFC 3493 for details). */
    hints.ai_flags = AI_ADDRCONFIG;
#endif

    e = getaddrinfo (super->path_element->host, port, &hints, &res);

#ifdef AI_ADDRCONFIG
    if (e == EAI_BADFLAGS)
    {
        /* Retry with no flags if AI_ADDRCONFIG was rejected. */
        hints.ai_flags = 0;
        e = getaddrinfo (super->path_element->host, port, &hints, &res);
    }
#endif

    if (e != 0)
    {
        mc_propagate_error (mcerror, e, _("sftp: %s"), gai_strerror (e));
        my_socket = LIBSSH2_INVALID_SOCKET;
        goto ret;
    }

    for (curr_res = res; curr_res != NULL; curr_res = curr_res->ai_next)
    {
        int save_errno;

        switch (curr_res->ai_addr->sa_family)
        {
        case AF_INET:
            sftpfs_super->ip_address =
                inet_ntop (AF_INET, &((struct sockaddr_in *) curr_res->ai_addr)->sin_addr,
                           address_ipv4, INET_ADDRSTRLEN);
            break;
        case AF_INET6:
            sftpfs_super->ip_address =
                inet_ntop (AF_INET6, &((struct sockaddr_in6 *) curr_res->ai_addr)->sin6_addr,
                           address_ipv6, INET6_ADDRSTRLEN);
            break;
        default:
            sftpfs_super->ip_address = NULL;
        }

        if (sftpfs_super->ip_address == NULL)
        {
            mc_propagate_error (mcerror, 0, "%s",
                                _("sftp: failed to convert remote host IP address into text form"));
            my_socket = LIBSSH2_INVALID_SOCKET;
            goto ret;
```

```
        }

        my_socket = socket (curr_res->ai_family, curr_res->ai_socktype, curr_res->ai_protocol);

        if (my_socket < 0)
        {
            if (curr_res->ai_next != NULL)
                continue;

            vfs_print_message (_("sftp: %s"), unix_error_string (errno));
            my_socket = LIBSSH2_INVALID_SOCKET;
            goto ret;
        }

        vfs_print_message (_("sftp: making connection to %s"), super->path_element->host);

        if (connect (my_socket, curr_res->ai_addr, curr_res->ai_addrlen) >= 0)
            break;

        save_errno = errno;

        close (my_socket);

        if (save_errno == EINTR && tty_got_interrupt ())
            mc_propagate_error (mcerror, 0, "%s", _("sftp: connection interrupted by user"));
        else if (res->ai_next == NULL)
            mc_propagate_error (mcerror, save_errno, _("sftp: connection to server failed: %s"),
                                unix_error_string (save_errno));
        else
            continue;

        my_socket = LIBSSH2_INVALID_SOCKET;
        break;
    }

  ret:
    if (res != NULL)
        freeaddrinfo (res);
    tty_disable_interrupt_key ();
    return my_socket;
}

/* ------------------------------------------------------------------------------------------ */

/**
 * Read ~/.ssh/known_hosts file.
 *
 * @param super connection data
 * @param mcerror pointer to the error handler
 * @return TRUE on success, FALSE otherwise
 *
 * Thanks the Curl project for the code used in this function.
 */
static gboolean
sftpfs_read_known_hosts (struct vfs_s_super *super, GError ** mcerror)
{
    sftpfs_super_t *sftpfs_super = SFTP_SUPER (super);
    struct libssh2_knownhost *store = NULL;
    int rc;
    gboolean found = FALSE;

    sftpfs_super->known_hosts = libssh2_knownhost_init (sftpfs_super->session);
    if (sftpfs_super->known_hosts == NULL)
        goto err;

    sftpfs_super->known_hosts_file =
        mc_build_filename (mc_config_get_home_dir (), ".ssh", "known_hosts", (char *) NULL);
    rc = libssh2_knownhost_readfile (sftpfs_super->known_hosts, sftpfs_super->known_hosts_file,
                                     LIBSSH2_KNOWNHOST_FILE_OPENSSH);
    if (rc > 0)
    {
        const char *kh_name_end = NULL;

        while (!found && libssh2_knownhost_get (sftpfs_super->known_hosts, &store, store) == 0)
        {
            /* For non-standard ports, the name will be enclosed in
             * square brackets, followed by a colon and the port */
            if (store == NULL)
                continue;

            if (store->name == NULL)
                found = TRUE;
            else if (store->name[0] != '[')
                found = strcmp (store->name, super->path_element->host) == 0;
            else
            {
                int port;

                kh_name_end = strstr (store->name, "]:");
                if (kh_name_end == NULL)
                    /* Invalid host pattern */
                    continue;

                port = (int) g_ascii_strtoll (kh_name_end + 2, NULL, 10);
                if (port == super->path_element->port)
                {
                    size_t kh_name_size;

```

```
                    kh_name_size = strlen (store->name) - 1 - strlen (kh_name_end);
                    found = strncmp (store->name + 1, super->path_element->host, kh_name_size) == 0;
                }
            }
        }
    }

    if (found)
    {
        int mask;
        const char *hostkey_method = NULL;

        mask = store->typemask & LIBSSH2_KNOWNHOST_KEY_MASK;

        switch (mask)
        {
#ifdef LIBSSH2_KNOWNHOST_KEY_ED25519
        case LIBSSH2_KNOWNHOST_KEY_ED25519:
            hostkey_method = hostkey_method_ssh_ed25519;
            break;
#endif
#ifdef LIBSSH2_KNOWNHOST_KEY_ECDSA_521
        case LIBSSH2_KNOWNHOST_KEY_ECDSA_521:
            hostkey_method = hostkey_method_ssh_ecdsa_521;
            break;
#endif
#ifdef LIBSSH2_KNOWNHOST_KEY_ECDSA_384
        case LIBSSH2_KNOWNHOST_KEY_ECDSA_384:
            hostkey_method = hostkey_method_ssh_ecdsa_384;
            break;
#endif
#ifdef LIBSSH2_KNOWNHOST_KEY_ECDSA_256
        case LIBSSH2_KNOWNHOST_KEY_ECDSA_256:
            hostkey_method = hostkey_method_ssh_ecdsa_256;
            break;
#endif
        case LIBSSH2_KNOWNHOST_KEY_SSHRSA:
            hostkey_method = hostkey_method_ssh_rsa;
            break;
        case LIBSSH2_KNOWNHOST_KEY_SSHDSS:
            hostkey_method = hostkey_method_ssh_dss;
            break;
        case LIBSSH2_KNOWNHOST_KEY_RSA1:
            mc_propagate_error (mcerror, 0, "%s",
                                _("sftp: found host key of unsupported type: RSA1"));
            return FALSE;
        default:
            mc_propagate_error (mcerror, 0, "%s %u", _("sftp: unknown host key type:"),
                                (unsigned int) mask);
            return FALSE;
        }

        rc = libssh2_session_method_pref (sftpfs_super->session, LIBSSH2_METHOD_HOSTKEY,
                                          hostkey_method);
        if (rc < 0)
            goto err;
    }

    return TRUE;

  err:
    {
        int sftp_errno;

        sftp_errno = libssh2_session_last_errno (sftpfs_super->session);
        sftpfs_ssherror_to_gliberror (sftpfs_super, sftp_errno, mcerror);
    }
    return FALSE;
}

/* ------------------------------------------------------------------------------------------ */

/**
 * Write new host + key pair to the ~/.ssh/known_hosts file.
 *
 * @param super connection data
 * @param remote_key he key for the remote host
 * @param remote_key_len length of @remote_key
 * @param type_mask info about format of host name, key and key type
 * @return 0 on success, regular libssh2 error code otherwise
 *
 * Thanks the Curl project for the code used in this function.
 */
static int
sftpfs_update_known_hosts (struct vfs_s_super *super, const char *remote_key, size_t remote_key_len,
                           int type_mask)
{
    sftpfs_super_t *sftpfs_super = SFTP_SUPER (super);
    int rc;

    /* add this host + key pair  */
    rc = libssh2_knownhost_addc (sftpfs_super->known_hosts, super->path_element->host, NULL,
                                 remote_key, remote_key_len, NULL, 0, type_mask, NULL);
    if (rc < 0)
        return rc;

    /* write the entire in-memory list of known hosts to the known_hosts file */
    rc = libssh2_knownhost_writefile (sftpfs_super->known_hosts, sftpfs_super->known_hosts_file,
```

```
                                    LIBSSH2_KNOWNHOST_FILE_OPENSSH);

    if (rc < 0)
        return rc;

    (void) message (D_NORMAL, _("Information"),
                    _("Permanently added\n%s (%s)\nto the list of known hosts."),
                    super->path_element->host, sftpfs_super->ip_address);

    return 0;
}

/* ------------------------------------------------------------------------------------------ */
/**
 * Compute and return readable host key fingerprint hash.
 *
 * @param session libssh2 session handle
 * @return pointer to static buffer on success, NULL otherwise
 */
static const char *
sftpfs_compute_fingerprint_hash (LIBSSH2_SESSION * session)
{
    static char result[SHA1_DIGEST_LENGTH * 3 + 1];     /* "XX:" for each byte, and EOL */
    const char *fingerprint;
    size_t i;

    /* The fingerprint points to static storage (!), don't free() it. */
    fingerprint = libssh2_hostkey_hash (session, LIBSSH2_HOSTKEY_HASH_SHA1);
    if (fingerprint == NULL)
        return NULL;

    for (i = 0; i < SHA1_DIGEST_LENGTH && i * 3 < sizeof (result) - 1; i++)
        g_snprintf ((gchar *) (result + i * 3), 4, "%02x:", (guint8) fingerprint[i]);

    /* remove last ":" */
    result[i * 3 - 1] = '\0';

    return result;
}

/* ------------------------------------------------------------------------------------------ */

/**
 * Process host info found in ~/.ssh/known_hosts file.
 *
 * @param super connection data
 * @param mcerror pointer to the error handler
 * @return TRUE on success, FALSE otherwise
 *
 * Thanks the Curl project for the code used in this function.
 */
static gboolean
sftpfs_process_known_host (struct vfs_s_super *super, GError ** mcerror)
{
    sftpfs_super_t *sftpfs_super = SFTP_SUPER (super);
    const char *remote_key;
    const char *key_type;
    const char *fingerprint_hash;
    size_t remote_key_len = 0;
    int remote_key_type = LIBSSH2_HOSTKEY_TYPE_UNKNOWN;
    int keybit = 0;
    struct libssh2_knownhost *host = NULL;
    int rc;
    char *msg = NULL;
    gboolean handle_query = FALSE;

    remote_key = libssh2_session_hostkey (sftpfs_super->session, &remote_key_len, &remote_key_type);
    if (remote_key == NULL || remote_key_len == 0
        || remote_key_type == LIBSSH2_HOSTKEY_TYPE_UNKNOWN)
    {
        mc_propagate_error (mcerror, 0, "%s", _("sftp: cannot get the remote host key"));
        return FALSE;
    }

    switch (remote_key_type)
    {
    case LIBSSH2_HOSTKEY_TYPE_RSA:
        keybit = LIBSSH2_KNOWNHOST_KEY_SSHRSA;
        key_type = "RSA";
        break;
    case LIBSSH2_HOSTKEY_TYPE_DSS:
        keybit = LIBSSH2_KNOWNHOST_KEY_SSHDSS;
        key_type = "DSS";
        break;
#ifdef LIBSSH2_HOSTKEY_TYPE_ECDSA_256
    case LIBSSH2_HOSTKEY_TYPE_ECDSA_256:
        keybit = LIBSSH2_KNOWNHOST_KEY_ECDSA_256;
        key_type = "ECDSA";
        break;
#endif
#ifdef LIBSSH2_HOSTKEY_TYPE_ECDSA_384
    case LIBSSH2_HOSTKEY_TYPE_ECDSA_384:
        keybit = LIBSSH2_KNOWNHOST_KEY_ECDSA_384;
        key_type = "ECDSA";
        break;
#endif
#ifdef LIBSSH2_HOSTKEY_TYPE_ECDSA_521
    case LIBSSH2_HOSTKEY_TYPE_ECDSA_521:
```

```
        keybit = LIBSSH2_KNOWNHOST_KEY_ECDSA_521;
        key_type = "ECDSA";
        break;
#endif
#ifdef LIBSSH2_HOSTKEY_TYPE_ED25519
    case LIBSSH2_HOSTKEY_TYPE_ED25519:
        keybit = LIBSSH2_KNOWNHOST_KEY_ED25519;
        key_type = "ED25519";
        break;
#endif
    default:
        mc_propagate_error (mcerror, 0, "%s",
                            _("sftp: unsupported key type, can't check remote host key"));
        return FALSE;
    }

    fingerprint_hash = sftpfs_compute_fingerprint_hash (sftpfs_super->session);
    if (fingerprint_hash == NULL)
    {
        mc_propagate_error (mcerror, 0, "%s", _("sftp: can't compute host key fingerprint hash"));
        return FALSE;
    }

    rc = libssh2_knownhost_checkp (sftpfs_super->known_hosts, super->path_element->host,
                                   super->path_element->port, remote_key, remote_key_len,
                                   LIBSSH2_KNOWNHOST_TYPE_PLAIN | LIBSSH2_KNOWNHOST_KEYENC_RAW |
                                   keybit, &host);

    switch (rc)
    {
    default:
    case LIBSSH2_KNOWNHOST_CHECK_FAILURE:
        /* something prevented the check to be made */
        goto err;

    case LIBSSH2_KNOWNHOST_CHECK_MATCH:
        /* host + key pair matched -- OK */
        break;

    case LIBSSH2_KNOWNHOST_CHECK_NOTFOUND:
        /* no host match was found -- add it to the known_hosts file */
        msg = g_strdup_printf (_("The authenticity of host\n%s (%s)\ncan't be established!\n"
                                 "%s key fingerprint hash is\nSHA1:%s.\n"
                                 "Do you want to add it to the list of known hosts and continue connecting?"),
                               super->path_element->host, sftpfs_super->ip_address,
                               key_type, fingerprint_hash);
        /* Select "No" initially */
        query_set_sel (2);
        rc = query_dialog (_("Warning"), msg, D_NORMAL, 3, _("&Yes"), _("&Ignore"), _("&No"));
        g_free (msg);
        handle_query = TRUE;
        break;

    case LIBSSH2_KNOWNHOST_CHECK_MISMATCH:
        msg = g_strdup_printf (_("%s (%s)\nis found in the list of known hosts but\n"
                                 "KEYS DO NOT MATCH! THIS COULD BE A MITM ATTACK!\n"
                                 "Are you sure you want to add it to the list of known hosts and continue connecting?"),
                               super->path_element->host, sftpfs_super->ip_address);
        /* Select "No" initially */
        query_set_sel (2);
        rc = query_dialog (MSG_ERROR, msg, D_ERROR, 3, _("&Yes"), _("&Ignore"), _("&No"));
        g_free (msg);
        handle_query = TRUE;
        break;
    }

    if (handle_query)
        switch (rc)
        {
        case 0:
            /* Yes: add this host + key pair, continue connecting */
            if (sftpfs_update_known_hosts (super, remote_key, remote_key_len,
                                           LIBSSH2_KNOWNHOST_TYPE_PLAIN
                                           | LIBSSH2_KNOWNHOST_KEYENC_RAW | keybit) < 0)
                goto err;
            break;
        case 1:
            /* Ignore: do not add this host + key pair, continue connecting anyway */
            break;
        case 2:
        default:
            mc_propagate_error (mcerror, 0, "%s", _("sftp: host key verification failed"));
            /* No: abort connection */
            goto err;
        }

    return TRUE;

  err:
    {
        int sftp_errno;

        sftp_errno = libssh2_session_last_errno (sftpfs_super->session);
        sftpfs_ssherror_to_gliberror (sftpfs_super, sftp_errno, mcerror);
    }

    return FALSE;
}
```

```

/* ---------------------------------------------------------------------------------------- */
/**
 * Recognize authentication types supported by remote side and filling internal 'super' structure by
 * proper enum's values.
 *
 * @param super connection data
 * @return TRUE if some of authentication methods is available, FALSE otherwise
 */
static gboolean
sftpfs_recognize_auth_types (struct vfs_s_super *super)
{
    char *userauthlist;
    sftpfs_super_t *sftpfs_super = SFTP_SUPER (super);

    /* check what authentication methods are available */
    /* userauthlist is internally managed by libssh2 and freed by libssh2_session_free() */
    userauthlist = libssh2_userauth_list (sftpfs_super->session, super->path_element->user,
                                          strlen (super->path_element->user));

    if (userauthlist == NULL)
        return FALSE;

    if ((strstr (userauthlist, "password") != NULL
         || strstr (userauthlist, "keyboard-interactive") != NULL)
        && (sftpfs_super->config_auth_type & PASSWORD) != 0)
        sftpfs_super->auth_type |= PASSWORD;

    if (strstr (userauthlist, "publickey") != NULL
        && (sftpfs_super->config_auth_type & PUBKEY) != 0)
        sftpfs_super->auth_type |= PUBKEY;

    if ((sftpfs_super->config_auth_type & AGENT) != 0)
        sftpfs_super->auth_type |= AGENT;

    return TRUE;
}

/* ---------------------------------------------------------------------------------------- */
/**
 * Open connection to host using SSH-agent helper.
 *
 * @param super   connection data
 * @param mcerror pointer to the error handler
 * @return TRUE if connection was successfully opened, FALSE otherwise
 */

static gboolean
sftpfs_open_connection_ssh_agent (struct vfs_s_super *super, GError ** mcerror)
{
    sftpfs_super_t *sftpfs_super = SFTP_SUPER (super);
    struct libssh2_agent_publickey *identity, *prev_identity = NULL;
    int rc;

    mc_return_val_if_error (mcerror, FALSE);

    sftpfs_super->agent = NULL;

    if ((sftpfs_super->auth_type & AGENT) == 0)
        return FALSE;

    /* Connect to the ssh-agent */
    sftpfs_super->agent = libssh2_agent_init (sftpfs_super->session);
    if (sftpfs_super->agent == NULL)
        return FALSE;

    if (libssh2_agent_connect (sftpfs_super->agent) != 0)
        return FALSE;

    if (libssh2_agent_list_identities (sftpfs_super->agent) != 0)
        return FALSE;

    while (TRUE)
    {
        rc = libssh2_agent_get_identity (sftpfs_super->agent, &identity, prev_identity);
        if (rc == 1)
            break;

        if (rc < 0)
            return FALSE;

        if (libssh2_agent_userauth (sftpfs_super->agent, super->path_element->user, identity) == 0)
            break;

        prev_identity = identity;
    }

    return (rc == 0);
}

/* ---------------------------------------------------------------------------------------- */
/**
 * Open connection to host using SSH-keypair.
 *
 * @param super   connection data
 * @param mcerror pointer to the error handler
 * @return TRUE if connection was successfully opened, FALSE otherwise
 */
```

```c

static gboolean
sftpfs_open_connection_ssh_key (struct vfs_s_super *super, GError ** mcerror)
{
    sftpfs_super_t *sftpfs_super = SFTP_SUPER (super);
    char *p, *passwd;
    gboolean ret_value = FALSE;

    mc_return_val_if_error (mcerror, FALSE);

    if ((sftpfs_super->auth_type & PUBKEY) == 0)
        return FALSE;

    if (sftpfs_super->privkey == NULL)
        return FALSE;

    if (libssh2_userauth_publickey_fromfile (sftpfs_super->session, super->path_element->user,
                                             sftpfs_super->pubkey, sftpfs_super->privkey,
                                             super->path_element->password) == 0)
        return TRUE;

    p = g_strdup_printf (_("sftp: Enter passphrase for %s "), super->path_element->user);
    passwd = vfs_get_password (p);
    g_free (p);

    if (passwd == NULL)
        mc_propagate_error (mcerror, 0, "%s", _("sftp: Passphrase is empty."));
    else
    {
        ret_value = (libssh2_userauth_publickey_fromfile (sftpfs_super->session,
                                                          super->path_element->user,
                                                          sftpfs_super->pubkey,
                                                          sftpfs_super->privkey, passwd) == 0);
        g_free (passwd);
    }

    return ret_value;
}

/* ---------------------------------------------------------------------------------------- */

/**
 * Keyboard-interactive password helper for opening connection to host by
 * sftpfs_open_connection_ssh_password
 *
 * Uses global kbi_super (data with existing connection) and kbi_passwd (password)
 *
 * @param name             username
 * @param name_len         length of @name
 * @param instruction      unused
 * @param instruction_len  unused
 * @param num_prompts      number of possible problems to process
 * @param prompts          array of prompts to process
 * @param responses        array of responses, one per prompt
 * @param abstract         unused
 */

static
LIBSSH2_USERAUTH_KBDINT_RESPONSE_FUNC (sftpfs_keyboard_interactive_helper)
{
    int i;
    size_t len;

    (void) instruction;
    (void) instruction_len;
    (void) abstract;

    if (kbi_super == NULL || kbi_passwd == NULL)
        return;

    if (strncmp (name, kbi_super->path_element->user, name_len) != 0)
        return;

    /* assume these are password prompts */
    len = strlen (kbi_passwd);

    for (i = 0; i < num_prompts; ++i)
        if (strncmp (prompts[i].text, "Password: ", prompts[i].length) == 0)
        {
            responses[i].text = strdup (kbi_passwd);
            responses[i].length = len;
        }
}

/* ---------------------------------------------------------------------------------------- */
/**
 * Open connection to host using password.
 *
 * @param super   connection data
 * @param mcerror pointer to the error handler
 * @return TRUE if connection was successfully opened, FALSE otherwise
 */

static gboolean
sftpfs_open_connection_ssh_password (struct vfs_s_super *super, GError ** mcerror)
{
    sftpfs_super_t *sftpfs_super = SFTP_SUPER (super);
    char *p, *passwd;
```

```
765     gboolean ret_value = FALSE;
766     int rc;
767
768     mc_return_val_if_error (mcerror, 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 (mcerror, 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 mcerror 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 ** mcerror)
852 {
853     int rc;
854     sftpfs_super_t *sftpfs_super = SFTP_SUPER (super);
855
856     mc_return_val_if_error (mcerror, -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, mcerror);
```

```
    if (sftpfs_super->socket_handle == LIBSSH2_INVALID_SOCKET)
        return (-1);

    /* Create a session instance */
    sftpfs_super->session = libssh2_session_init ();
    if (sftpfs_super->session == NULL)
        return (-1);

    if (!sftpfs_read_known_hosts (super, mcerror))
        return (-1);

    /* ... start it up. This will trade welcome banners, exchange keys,
     * and setup crypto, compression, and MAC layers
     */
    while ((rc =
            libssh2_session_handshake (sftpfs_super->session,
                                       (libssh2_socket_t) sftpfs_super->socket_handle)) ==
            LIBSSH2_ERROR_EAGAIN)
        ;
    if (rc != 0)
    {
        mc_propagate_error (mcerror, rc, "%s", _("sftp: failure establishing SSH session"));
        return (-1);
    }

    if (!sftpfs_process_known_host (super, mcerror))
        return (-1);

    if (!sftpfs_recognize_auth_types (super))
    {
        int sftp_errno;

        sftp_errno = libssh2_session_last_errno (sftpfs_super->session);
        sftpfs_ssherror_to_gliberror (sftpfs_super, sftp_errno, mcerror);
        return (-1);
    }

    if (!sftpfs_open_connection_ssh_agent (super, mcerror)
        && !sftpfs_open_connection_ssh_key (super, mcerror)
        && !sftpfs_open_connection_ssh_password (super, mcerror))
        return (-1);

    sftpfs_super->sftp_session = libssh2_sftp_init (sftpfs_super->session);

    if (sftpfs_super->sftp_session == NULL)
        return (-1);

    /* Since we have not set non-blocking, tell libssh2 we are blocking */
    libssh2_session_set_blocking (sftpfs_super->session, 1);

    return 0;
}

/* ------------------------------------------------------------------------------------------ */
/**
 * Close connection.
 *
 * @param super            connection data
 * @param shutdown_message message for shutdown functions
 * @param mcerror          pointer to the error handler
 */

void
sftpfs_close_connection (struct vfs_s_super *super, const char *shutdown_message, GError ** mcerror)
{
    sftpfs_super_t *sftpfs_super = SFTP_SUPER (super);

    /* no mc_return_*_if_error() here because of abort open_connection handling too */
    (void) mcerror;

    if (sftpfs_super->sftp_session != NULL)
    {
        libssh2_sftp_shutdown (sftpfs_super->sftp_session);
        sftpfs_super->sftp_session = NULL;
    }

    if (sftpfs_super->agent != NULL)
    {
        libssh2_agent_disconnect (sftpfs_super->agent);
        libssh2_agent_free (sftpfs_super->agent);
        sftpfs_super->agent = NULL;
    }

    if (sftpfs_super->known_hosts != NULL)
    {
        libssh2_knownhost_free (sftpfs_super->known_hosts);
        sftpfs_super->known_hosts = NULL;
    }

    MC_PTR_FREE (sftpfs_super->known_hosts_file);

    if (sftpfs_super->session != NULL)
    {
        libssh2_session_disconnect (sftpfs_super->session, shutdown_message);
        libssh2_session_free (sftpfs_super->session);
        sftpfs_super->session = NULL;
    }

```

```
    if (sftpfs_super->socket_handle != LIBSSH2_INVALID_SOCKET)
    {
        close (sftpfs_super->socket_handle);
        sftpfs_super->socket_handle = LIBSSH2_INVALID_SOCKET;
    }
}

/* ---------------------------------------------------------------------------------------- */
```