aborodin Ticket #4179: code clean up before 4.8.27 release. … History

4 contributors

558 lines (443 sloc) | 17.4 KB

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

   Copyright (C) 2011-2021
   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 "internal.h"

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

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

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

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

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)
{
    struct addrinfo hints, *res = NULL, *curr_res;
    int my_socket = 0;
    char port[BUF_TINY];
    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;

        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;
}

/* --------------------------------------------------------------------------------------------- */
/**
 * Recognize authenticaion 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
 */

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;
    gboolean ret_value = FALSE;
    int rc;

    mc_return_val_if_error (mcerror, FALSE);

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

    if (super->path_element->password != NULL)
    {
        while ((rc = libssh2_userauth_password (sftpfs_super->session, super->path_element->user,
                                                super->path_element->password)) ==
               LIBSSH2_ERROR_EAGAIN);
        if (rc == 0)
            return TRUE;
```

```c

        kbi_super = super;
        kbi_passwd = super->path_element->password;

        while ((rc =
                libssh2_userauth_keyboard_interactive (sftpfs_super->session,
                                                       super->path_element->user,
                                                       sftpfs_keyboard_interactive_helper)) ==
               LIBSSH2_ERROR_EAGAIN)
            ;

        kbi_super = NULL;
        kbi_passwd = NULL;

        if (rc == 0)
            return TRUE;
    }

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

    if (passwd == NULL)
        mc_propagate_error (mcerror, 0, "%s", _("sftp: Password is empty."));
    else
    {
        while ((rc = libssh2_userauth_password (sftpfs_super->session, super->path_element->user,
                                                passwd)) == LIBSSH2_ERROR_EAGAIN)
            ;

        if (rc != 0)
        {
            kbi_super = super;
            kbi_passwd = passwd;

            while ((rc =
                    libssh2_userauth_keyboard_interactive (sftpfs_super->session,
                                                           super->path_element->user,
                                                           sftpfs_keyboard_interactive_helper)) ==
                   LIBSSH2_ERROR_EAGAIN)
                ;

            kbi_super = NULL;
            kbi_passwd = NULL;
        }

        if (rc == 0)
        {
            ret_value = TRUE;
            g_free (super->path_element->password);
            super->path_element->password = passwd;
        }
        else
            g_free (passwd);
    }

    return ret_value;
}

/* --------------------------------------------------------------------------------------------- */
/*** public functions ****************************************************************************/
/* --------------------------------------------------------------------------------------------- */
/**
 * Open new connection.
 *
 * @param super   connection data
 * @param mcerror pointer to the error handler
 * @return 0 if success, -1 otherwise
 */

int
sftpfs_open_connection (struct vfs_s_super *super, GError ** mcerror)
{
    int rc;
    sftpfs_super_t *sftpfs_super = SFTP_SUPER (super);

    mc_return_val_if_error (mcerror, -1);

    /*
     * The application code is responsible for creating the socket
     * and establishing the connection
     */
    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);

    /* ... start it up. This will trade welcome banners, exchange keys,
     * and setup crypto, compression, and MAC layers
     */
#if LIBSSH2_VERSION_NUM < 0x010208
    rc = libssh2_session_startup (sftpfs_super->session, sftpfs_super->socket_handle);
#else
    rc = libssh2_session_handshake (sftpfs_super->session,
                                     (libssh2_socket_t) sftpfs_super->socket_handle);
#endif
    if (rc != 0)
    {
        mc_propagate_error (mcerror, rc, "%s", _("sftp: Failure establishing SSH session"));
        return (-1);
    }

    /* At this point we havn't yet authenticated.  The first thing to do
     * is check the hostkey's fingerprint against our known hosts Your app
     * may have it hard coded, may go to a file, may present it to the
     * user, that's your call
     */
    sftpfs_super->fingerprint =
        libssh2_hostkey_hash (sftpfs_super->session, LIBSSH2_HOSTKEY_HASH_SHA1);

    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;
    }

    sftpfs_super->fingerprint = NULL;

    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;
    }
}

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