sshmount ソースリスト

美都

2026 年 2 月 1 日

目次

1　メインモジュール　main.rs

```rust
mod cmdline_opt;
mod fuse_util;
mod ssh_connect;
mod ssh_filesystem;

use anyhow::{Context, Result};
use clap::Parser;
use cmdline_opt::Opt;
use daemonize::Daemonize;
use fuse_util::{make_full_path, make_mount_option, make_remote_path};
use ssh_connect::make_ssh_session;
//use log::debug;

fn main() -> Result<()> {
    env_logger::init();
    let opt = Opt::parse();

    let ssh = make_ssh_session(&opt).context("Failed to generate ssh session.")?;

    let path = make_remote_path(&opt, &ssh).context("Failed to generate remote path.")?;
    let options = make_mount_option(&opt);
    let mount_point = make_full_path(&opt.mount_point)?;

    // プロセスのデーモン化
    if opt.daemon {
        let daemonize = Daemonize::new();
        if let Err(e) = daemonize.start() {
            eprintln!("daemonization filed.(error: {})", e);
        }
    }
    // ファイルシステムへのマウント実行
    let fs = ssh_filesystem::Sshfs::new(ssh, &path)?;
    fuser::mount2(fs, mount_point, &options).context("Failed to mount FUSE.")?;
    Ok(())
}
```

2　コマンドラインオプションの定義 cmdline_opt.rs

```rust
pub mod remote_host;

use anyhow::{anyhow, Context};
use clap::Parser;
use std::path::PathBuf;

use remote_host::RemoteName;

/// コマンドラインオプション
#[derive(Parser, Debug)]
#[command(author, version, about)]
pub struct Opt {
    /// Distination [user@]host:[path] or scp://[user@]host[:port][/path]
    pub remote: RemoteName,
    /// Path to mount
    #[arg(value_parser = exist_dir)]
    pub mount_point: String,
    /// Path to config file
    #[arg(short = 'F', long)]
    pub config_file: Option<PathBuf>,
    /// Login name
    #[arg(short, long)]
    pub login_name: Option<String>,
    /// File name of secret key file
    #[arg(short, long)]
    pub identity: Option<PathBuf>,
    /// Port no
    #[arg(short, long)]
    pub port: Option<u16>,
    /// Read only
    #[arg(short, long)]
    pub readonly: bool,
    /// Not executable
    #[arg(long)]
    pub no_exec: bool,
    /// Do not change access date and time(atime)
    #[arg(long)]
    pub no_atime: bool,
    /// run in daemon mode
    #[arg(short, long)]
    pub daemon: bool,
}

```

```rust
/// 指定されたディレクトリが存在し、中にファイルがないことを確認する。
fn exist_dir(s: &str) -> anyhow::Result<String> {
    match std::fs::read_dir(s) {
        Ok(mut dir) => match dir.next() {
            None => Ok(s.to_string()),
            Some(_) => Err(anyhow!("Mount destination directory is not empty.")),
        },
        Err(e) => match e.kind() {
            std::io::ErrorKind::NotFound => Err(anyhow!("The mount directory does not exist.")),
            std::io::ErrorKind::NotConnected => Err(anyhow!(
                "The network of the mount directory is disconnected. (Did you forget to umount?)."
            )),
            _ => Err(e).context("Unexpected error.(check mount directory)"),
        },
    }
}

#[cfg(test)]
mod test {
    use super::*;
    #[test]
    fn verify_cli() {
        use clap::CommandFactory;
        Opt::command().debug_assert()
    }
}
```

3　コマンドライン　remotehost の解析部 cmdline_opt/remote_host.rs

```rust
//! コマンドラインオプションにおける RemoteHost 構造体

use fluent_uri::{component::Host, Uri};
use log::debug;
use std::net::IpAddr;

/// コマンドラインの接続先ホスト情報
#[derive(Clone, Debug, PartialEq)]
pub struct RemoteName {
    /// ユーザー名
    pub user: Option<String>,
    /// IP アドレス
    pub host: HostInfo,
    /// ポート番号
    pub port: Option<u16>,
    /// 接続先パス
    pub path: Option<std::path::PathBuf>,
}

/// ホスト情報 (ホスト名または、IP アドレス)
#[derive(Clone, Debug, PartialEq)]
pub enum HostInfo {
    Name(String),
    Ip(IpAddr),
}

impl std::fmt::Display for HostInfo {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        match self {
            HostInfo::Name(name) => write!(f, "{}", name),
            HostInfo::Ip(ip) => write!(f, "{}", ip),
        }
    }
}

impl RemoteName {
    /// remote 引数の解析 (URI 形式の場合)
    fn parse_uri(s: &str) -> Result<Self, ErrorRemoteName> {
        // URI の解析
        let uri = Uri::parse(s)?;
        let info = uri
            .authority()
            .ok_or(ErrorRemoteName::NotFoundHostInformationInURI)?;
```

```
        // ポート番号の取得
        let port = info
            .port_to_u16()
            .map_err(|_| ErrorRemoteName::InvalidPortNo)?;
        // ホストアドレスの取得
        let host = match info.host_parsed() {
            Host::Ipv4(ip) => HostInfo::Ip(ip.into()),
            Host::Ipv6(ip) => HostInfo::Ip(ip.into()),
            _ => HostInfo::Name(info.host().to_string()),
        };
        // パス名の取得
        let path_str = uri.path().as_str();
        let path = if path_str.is_empty() {
            None
        } else {
            Some(
                path_str
                    .trim()
                    .parse::<std::path::PathBuf>()
                    .map_err(|_| ErrorRemoteName::InvalidPath)?,
            )
        };
        // ユーザー名の取得
        let user = info.userinfo().and_then(|s| {
            let trimed = s.as_str().trim();
            if trimed.is_empty() {
                None
            } else {
                Some(trimed.to_string())
            }
        });

        Ok(Self {
            user,
            host,
            port,
            path,
        })
    }

    /// remote引数の解析（URI形式でない場合）
    fn parse_non_uri(s: &str) -> Result<Self, ErrorRemoteName> {
        let mut rest_str = s.trim();
        // ユーザー名の取得
        let user = match rest_str.split_once("@") {
            Some((u, r)) => {
```

```
                rest_str = r;
                Some(u.trim().to_string())
            }
            None => None,
        };
        // ホストアドレスの取得
        let host_str: &str;
        // IPv6の判定
        rest_str = rest_str.trim_start();
        if rest_str.starts_with('[') {
            let (ip6_str, rest) = match rest_str.split_once(']') {
                Some((l, r)) => (l.trim_start_matches('['), r),
                None => return Err(ErrorRemoteName::MissingClosingBracketInIPv6),
            };
            host_str = ip6_str.trim();
            let (_, r) = rest.split_once(":").ok_or(ErrorRemoteName::NoColon)?;
            rest_str = r;
        } else {
            let (h, r) = rest_str.split_once(':').ok_or(ErrorRemoteName::NoColon)?;
            if h.trim().is_empty() {
                return Err(ErrorRemoteName::NoHostName);
            }
            host_str = h.trim();
            rest_str = r;
        }

        let try_ip = host_str.parse::<IpAddr>();
        let host = match try_ip {
            Ok(addr) => HostInfo::Ip(addr),
            Err(_) => HostInfo::Name(host_str.to_string()),
        };

        // パス名の取得
        rest_str = rest_str.trim();
        let path = if rest_str.is_empty() {
            None
        } else {
            Some(
                rest_str
                    .parse::<std::path::PathBuf>()
                    .map_err(|_| ErrorRemoteName::InvalidPath)?,
            )
        };

        Ok(Self {
            host,
```

```
            port: None,
            user,
            path,
        })
    }
}

impl std::fmt::Display for RemoteName {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        let s = format!(
            "<{:?}><{:?}><{:?}><{:?}>",
            &self.user, &self.host, &self.port, &self.path
        );
        s.fmt(f)
    }
}

impl std::str::FromStr for RemoteName {
    type Err = ErrorRemoteName;
    fn from_str(s: &str) -> Result<Self, Self::Err> {
        let ret = if s.trim_start().starts_with("scp://") {
            Self::parse_uri(s)
        } else {
            Self::parse_non_uri(s)
        };
        debug!("RemoteName: {:?}", ret);
        ret
    }
}

#[derive(thiserror::Error, Debug, PartialEq, Eq)]
//#[error("The format of the host to connect to is \"[user@]host:[path]\" or
↪  \"scp://[user@]host[:port][/path]\".")]
pub enum ErrorRemoteName {
    #[error("URI parse error.")]
    UriParse(#[from] fluent_uri::ParseError),
    #[error("Host information not found at the specified URI.")]
    NotFoundHostInformationInURI,
    #[error("Invalid port number.")]
    InvalidPortNo,
    #[error("Invalid path name")]
    InvalidPath,
    #[error("The closing bracket is missing from IPv6.")]
    MissingClosingBracketInIPv6,
    #[error("In the host information, there is no subsequent colon (:).")]
    NoColon,
```

```
    #[error("No hostname specified.")]
    NoHostName,
}

#[cfg(test)]
mod test {
    use fluent_uri::ParseErrorKind;

    use super::*;

    #[test]
    fn test_from_str_remotename() {
        use std::path::Path;
        let s = "mito@reterminal.local:/home/mito";
        let r: RemoteName = s.parse().unwrap();
        let k = RemoteName {
            user: Some("mito".to_string()),
            host: HostInfo::Name("reterminal.local".to_string()),
            port: None,
            path: Some(Path::new("/home/mito").into()),
        };
        assert_eq!(r, k);

        let s = "mito@reterminal.local:/home/mito/";
        let r: RemoteName = s.parse().unwrap();
        let k = RemoteName {
            user: Some("mito".to_string()),
            host: HostInfo::Name("reterminal.local".to_string()),
            port: None,
            path: Some(Path::new("/home/mito").into()),
        };
        assert_eq!(r, k);

        let s = "mito@[fe80::a00:27ff:fe0e:8c0c]:/home/mito/";
        let r: RemoteName = s.parse().unwrap();
        let k = RemoteName {
            user: Some("mito".to_string()),
            host: HostInfo::Ip("fe80::a00:27ff:fe0e:8c0c".parse().unwrap()),
            port: None,
            path: Some(Path::new("/home/mito").into()),
        };
        assert_eq!(r, k);

        let s = "mito@[::1]:/home/mito/";
        let r: RemoteName = s.parse().unwrap();
        let k = RemoteName {
```

```
227             user: Some("mito".to_string()),
228             host: HostInfo::Ip("::1".parse().unwrap()),
229             port: None,
230             path: Some(Path::new("/home/mito").into()),
231         };
232         assert_eq!(r, k);
233 
234         let s = "reterminal.local:";
235         let r: RemoteName = s.parse().unwrap();
236         let k = RemoteName {
237             user: None,
238             host: HostInfo::Name("reterminal.local".to_string()),
239             port: None,
240             path: None,
241         };
242         assert_eq!(r, k);
243 
244         let s = " mito @reterminal.local: ";
245         let r: RemoteName = s.parse().unwrap();
246         let k = RemoteName {
247             user: Some("mito".to_string()),
248             host: HostInfo::Name("reterminal.local".to_string()),
249             port: None,
250             path: None,
251         };
252         assert_eq!(r, k);
253 
254         let s = "reterminal.local";
255         let r: Result<RemoteName, ErrorRemoteName> = s.parse();
256         assert_eq!(r, Err(ErrorRemoteName::NoColon));
257 
258         let s = "mito@reterminal.local";
259         let r: Result<RemoteName, ErrorRemoteName> = s.parse();
260         assert_eq!(r, Err(ErrorRemoteName::NoColon));
261 
262         let s = " mito @: ";
263         let r: Result<RemoteName, ErrorRemoteName> = s.parse();
264         assert_eq!(r, Err(ErrorRemoteName::NoHostName));
265     }
266 
267     #[test]
268     // scp://[user@]host[:port][/path]  の解析テスト
269     fn test_from_str_remotename_uri() {
270         let s = "scp://name@hostname.hoge:22/test_path";
271         let a = RemoteName {
272             user: Some("name".to_string()),
```

```
            host: HostInfo::Name("hostname.hoge".to_string()),
            port: Some(22),
            path: Some(std::path::PathBuf::from("/test_path")),
        };
        let r: RemoteName = s.parse().unwrap();
        assert_eq!(r, a);

        let s = "scp://name@192.168.0.1:22/test_path/path";
        let a = RemoteName {
            user: Some("name".to_string()),
            host: HostInfo::Ip("192.168.0.1".parse().unwrap()),
            port: Some(22),
            path: Some(std::path::PathBuf::from("/test_path/path")),
        };
        let r: RemoteName = s.parse().unwrap();
        assert_eq!(r, a);

        let s = "scp://[::1]:22/test";
        let a = RemoteName {
            user: None,
            host: HostInfo::Ip("::1".parse().unwrap()),
            port: Some(22),
            path: Some(std::path::PathBuf::from("/test")),
        };
        let r: RemoteName = s.parse().unwrap();
        assert_eq!(r, a);

        let s = "scp://localhost";
        let a = RemoteName {
            user: None,
            host: HostInfo::Name("localhost".to_string()),
            port: None,
            path: None,
        };
        let r: RemoteName = s.parse().unwrap();
        assert_eq!(r, a);
    }

    #[test]
    fn test_from_str_with_extra_space() {
        let s = "name@192.168.0.1:  /test_path/path   ";
        let a = RemoteName {
            user: Some("name".to_string()),
            host: HostInfo::Ip("192.168.0.1".parse().unwrap()),
            port: None,
            path: Some(std::path::PathBuf::from("/test_path/path")),
```

```
319         };
320         let r: RemoteName = s.parse().unwrap();
321         assert_eq!(r, a);
322 
323         let s = "scp://[::1]:22/  /test";
324         let r: Result<RemoteName, ErrorRemoteName> = s.parse();
325         match r {
326             Ok(_) => {
327                 unreachable!("この形式はエラーであるはず。");
328             }
329             Err(ErrorRemoteName::UriParse(e)) => {
330                 assert_eq!(e.kind(), ParseErrorKind::UnexpectedChar);
331             }
332             Err(e) => {
333                 unreachable!("何が起こった?({:?}", e);
334             }
335         }
336 
337         let s = "scp:// name @192.168.0.1:22/test_path/path";
338         let r: Result<RemoteName, ErrorRemoteName> = s.parse();
339         match r {
340             Ok(_) => {
341                 unreachable!("この形式はエラーであるはず。");
342             }
343             Err(ErrorRemoteName::UriParse(e)) => {
344                 assert_eq!(e.kind(), ParseErrorKind::UnexpectedChar);
345             }
346             Err(e) => {
347                 unreachable!("何が起こった?({:?})", e);
348             }
349         }
350     }
351 }
```

4　ssh2 ログイン処理モジュール ssh_connect.rs

```rust
//! ssh 接続関連関数モジュール

use crate::cmdline_opt::Opt;
use anyhow::{anyhow, Context, Result};
use dialoguer::Password;
use dns_lookup::lookup_host;
use log::{debug, error, info, warn};
use ssh2::Session;
use ssh2_config::{HostParams, ParseRule, SshConfig};
use std::{
    fs::File,
    io::BufReader,
    net::TcpStream,
    path::{Path, PathBuf},
    str,
};

/// ssh 認証で試行するキーファイルの最大数。
/// あまりに、トライ数が多いとサーバー負荷等に影響があるかもしれないので制限する。
const MAX_IDENTITY_TRY: usize = 10;
/// デフォルトのポート番号
const DEFAULT_PORT: u16 = 22;

/// セッションを生成する。
pub fn make_ssh_session(opt: &Opt) -> Result<Session> {
    let host_params = make_host_params(opt).context("Failed to make host parameters.")?;
    let addresses = get_address(&host_params)?;
    let user_name = host_params
        .user
        .as_ref()
        .ok_or(anyhow!("User name is not specified."))?;
    info!(
        "[main] info connection-> user name:\"{}\", ip address:{:?}",
        &user_name, &addresses
    );
    let identity_file = &host_params.identity_file;

    let ssh = connect_ssh(&addresses[..]).context("The ssh connection failed.")?;
    userauth(&ssh, user_name, identity_file).context("User authentication failed.")?;
    info!("success connect ssh: ip=>{:?}", addresses);
    Ok(ssh)
}

```

```rust
/// ホストパラメータの生成
/// config ファイルより取得したホスト情報をもとに、コマンドラインオプションで上書きしたホストパラメータを生成する。
/// ホスト情報は、コマンドラインオプション>config ファイル>remote_host 引数の順で上書きする。
fn make_host_params(opt: &Opt) -> Result<HostParams> {
    let mut host_params = get_ssh_config(&opt.config_file).query(opt.remote.host.to_string());
    // ホスト名の解決
    if host_params.host_name.is_none() {
        host_params.host_name = Some(opt.remote.host.to_string());
    }
    // ユーザー名の解決
    host_params.user = Some(get_username(opt, &host_params).context("Failed to get user name.")?);
    // 秘密キーファイルの解決
    host_params.identity_file = get_identity_file(opt, &host_params)?;
    // ポート番号の解決
    host_params.port = Some(
        opt.port.unwrap_or(
            host_params
                .port
                .unwrap_or(opt.remote.port.unwrap_or(DEFAULT_PORT)),
        ),
    );
    Ok(host_params)
}

/// ホストの ip アドレス解決
fn get_address(host_params: &HostParams) -> Result<Vec<std::net::SocketAddr>> {
    let dns = host_params
        .host_name
        .as_ref()
        .ok_or(anyhow!("Host name is not specified."))?;
    let port = host_params
        .port
        .ok_or(anyhow!("Port number is not specified."))?;
    let addrs = lookup_host(dns)
        .inspect_err(|e| error!("get_address : Failed lookup_host[{}]", e))
        .context("Cannot find host to connect to.")?
        .map(|addr| std::net::SocketAddr::from((addr, port)))
        .collect::<Vec<_>>();
    if addrs.is_empty() {
        return Err(anyhow!("No address found for the specified host."));
    }
    Ok(addrs)
}

/// ssh-config の取得と解析
```

```rust
/// ファイル名が指定されていない場合は"~/.ssh/config"を使用
/// config ファイルのエラー及びファイルがない場合、デフォルト値を返す。
fn get_ssh_config(file_opt: &Option<PathBuf>) -> SshConfig {
    get_config_file(file_opt)
        .map(BufReader::new)
        .map_or(SshConfig::default(), |mut f| {
            SshConfig::default()
                .parse(&mut f, ParseRule::ALLOW_UNKNOWN_FIELDS)
                .unwrap_or_else(|e| {
                    eprintln!("Warning: Failed to parse ssh_config file. Using default settings.
                    ↪  (error: {})", e);
                    SshConfig::default()
                })
        })
}

/// ssh_config ファイルがあれば、オープンする。
/// ファイル名の指定がなければ、$Home/.ssh/config を想定する。
fn get_config_file(file_name: &Option<PathBuf>) -> Option<std::fs::File> {
    let file_name = file_name.clone().or_else(|| {
        home::home_dir().map(|p| {
            let mut p = p;
            p.push(".ssh/config");
            p
        })
    });

    file_name.and_then(|p| File::open(p).ok())
}

/// ログイン名を確定し、取得する。
/// ログイン名指定の優先順位は、1. -u 引数指定, 2.remote 引数, 3.ssh_config 指定, 4. 現在のユーザー名
fn get_username(opt: &Opt, params: &HostParams) -> Result<String> {
    if let Some(n) = &opt.login_name {
        Ok(n.clone())
    } else if let Some(n) = &opt.remote.user {
        Ok(n.clone())
    } else if let Some(n) = &params.user {
        Ok(n.clone())
    } else if let Some(n) = users::get_current_username() {
        n.to_str()
            .map(|s| s.to_string())
            .ok_or(anyhow!("Invalid login user name. -- {n:?}"))
    } else {
        Err(anyhow!("Could not obtain user name."))
    }
```

```rust
}

/// 秘密キーファイルのパスを取得する
fn get_identity_file(opt: &Opt, host_params: &HostParams) -> Result<Option<Vec<PathBuf>>> {
    if let Some(n) = &opt.identity {
        let path = expand_tilde_in_path(n);
        std::fs::File::open(&path).with_context(|| {
            format!(
                "Unable to access the secret key file specified by the \"-i\" option. [{}]",
                &path.to_string_lossy()
            )
        })?;
        Ok(Some(vec![path]))
    } else {
        let name = host_params.identity_file.as_ref();
        match name {
            Some(n) => {
                let paths = n
                    .iter()
                    .map(expand_tilde_in_path)
                    .filter(|p|  match std::fs::File::open(p) {
                        Ok(_) => true,
                        Err(e) => {
                            warn!(
                                "IdentityFile '{:?}' from ssh-config is not accessible. skipping.
                                ↪  (io error: {})",
                                p, e
                            );
                            eprintln!(
                                "Warning: IdentityFile '{:?}' from ssh-config is not accessible.
                                ↪  skipping.",
                                p
                            );
                            false
                        }
                    })
                    .collect::<Vec<_>>();
                if paths.is_empty() {
                    Err(anyhow!(
                        "No usable identity files found for host {:?} (checked {} entries from
                        ↪  ssh-config).",
                        host_params.host_name.as_deref().unwrap_or("<unknown>"),
                        n.len()
                    ))
                } else {
                    Ok(Some(paths))
```

```
                }
            }
            None => Ok(None),
        }
    }
}

// ファイル名の~記号を展開する。
fn expand_tilde_in_path(path: impl AsRef<Path>) -> PathBuf {
    let path_str = path.as_ref().to_string_lossy();
    let expanded_path = shellexpand::tilde(&path_str);
    PathBuf::from(expanded_path.as_ref())
}

/// リモートの ssh に接続し、セッションを生成する。
fn connect_ssh<A: std::net::ToSocketAddrs>(address: A) -> Result<Session> {
    let tcp = TcpStream::connect(address).context("Failed to connect to TCP/IP.")?;
    let mut ssh = Session::new().context("Failed to connect to ssh.")?;
    ssh.set_tcp_stream(tcp);
    ssh.handshake().context("Failed to hanshake ssh.")?;
    Ok(ssh)
}

/// ssh 認証を実施する。
fn userauth(sess: &Session, username: &str, identity: &Option<Vec<PathBuf>>) -> Result<()> {
    if user_auth_agent(sess, username).is_ok() {
        return Ok(());
    }
    if let Some(f) = identity {
        let ret = f
            .iter()
            .take(MAX_IDENTITY_TRY)
            .any(|f| user_auth_identity(sess, username, f).is_ok());
        if ret {
            return Ok(());
        }
    }
    user_auth_password(sess, username)
        .map_err(|_| anyhow!("All user authentication methods failed."))
}

/// agent 認証
fn user_auth_agent(sess: &Session, username: &str) -> Result<(), ssh2::Error> {
    let ret = sess.userauth_agent(username);
    if let Err(e) = &ret {
        debug!("認証失敗 (agent)->{:?}", e);
```

```
    };
    ret
}

/// 公開キー認証
fn user_auth_identity(sess: &Session, username: &str, key_file: &Path) -> Result<()> {
    let mut ret = sess.userauth_pubkey_file(username, None, key_file, None);
    if ret.is_ok() {
        return Ok(());
    };
    if let ssh2::ErrorCode::Session(-16) = ret.as_ref().unwrap_err().code() {
        // error_code -16 ->
        // LIBSSH2_ERROR_FILE:PUBLIC_KEYの取得失敗。多分、秘密キーのパスフレーズ
        for _i in 0..3 {
            let password = Password::new()
                .with_prompt("Enter the passphrase for the secret key.")
                .allow_empty_password(true)
                .interact()?;
            ret = sess.userauth_pubkey_file(username, None, key_file, Some(&password));
            if ret.is_ok() {
                return Ok(());
            }
            eprintln!("The passphrase is different.");
        }
    }
    debug!(
        "Authentication failed(pubkey)->{:?}",
        ret.as_ref().unwrap_err()
    );
    Err(anyhow!("Public key authentication failed."))
}

/// パスワード認証
fn user_auth_password(sess: &Session, username: &str) -> Result<()> {
    for _i in 0..3 {
        let password = Password::new()
            .with_prompt("Enter your login password.")
            .allow_empty_password(true)
            .interact()?;
        let ret = sess.userauth_password(username, &password);
        if ret.is_ok() {
            return Ok(());
        }
        let ssh2::ErrorCode::Session(-18) = ret.as_ref().unwrap_err().code() else {
            break;
        };
```

```
269             // ssh2エラーコード  -18 ->
270             // LIBSSH2_ERROR_AUTHENTICATION_FAILED: パスワードが違うんでしょう。
271             eprintln!("The password is different.");
272             debug!("Authentication failed(password)->{:?}", ret.unwrap_err());
273         }
274         Err(anyhow!("Password authentication failed."))
275     }
276 
277     #[cfg(test)]
278     mod test {
279         use super::*;
280         use clap::Parser;
281         #[test]
282         #[ignore]
283         fn make_host_params_test() {
284             let config_file_path = test_config_file_path();
285             let identify = make_dummyidentity_file(1);
286             let opt = make_dummy_opt(format!(
287                 "sshmount -F {} -i {} -p 2223 test_host:/remote/path mnt",
288                 config_file_path.to_string_lossy(),
289                 identify.to_string_lossy()
290             ));
291             let host_param = make_host_params(&opt).unwrap();
292             assert_eq!(host_param.host_name.unwrap(), "example.com");
293             assert_eq!(host_param.port.unwrap(), 2223);
294             assert_eq!(host_param.user.unwrap(), "testuser");
295         }
296 
297         #[test]
298         #[ignore]
299         fn test_make_host_params_default_port() {
300             let config_file_path = test_config_file_path();
301             let opt = make_dummy_opt(format!(
302                 "sshmount -F {} default_port:/remote/path mnt",
303                 config_file_path.to_string_lossy(),
304             ));
305             let host_param = make_host_params(&opt).unwrap();
306             assert_eq!(host_param.host_name.unwrap(), "default.example.com");
307             assert_eq!(host_param.port.unwrap(), DEFAULT_PORT);
308             assert_eq!(host_param.user.unwrap(), "defaultuser");
309         }
310 
311         #[test]
312         #[ignore]
313         fn test_make_host_params_ip_address_config() {
314             let config_file_path = test_config_file_path();
```

```
315         let opt = make_dummy_opt(format!(
316             "sshmount -F {} 192.168.0.100:/remote/path mnt",
317             config_file_path.to_string_lossy(),
318         ));
319         let host_param = make_host_params(&opt).unwrap();
320         assert_eq!(host_param.host_name.unwrap(), "192.168.0.101");
321         assert_eq!(host_param.port.unwrap(), 2200);
322         assert_eq!(host_param.user.unwrap(), "admin");
323         assert_eq!(
324             host_param.identity_file.unwrap()[0],
325             PathBuf::from("/home/mito/develop/rust/sshmount/test_data/dummy2_rsa")
326         );
327     }
328
329     #[test]
330     #[ignore]
331     fn test_make_host_params_multi_identify() {
332         let config_file_path = test_config_file_path();
333         let opt = make_dummy_opt(format!(
334             "sshmount -F {} multi_identity:/remote/path mnt",
335             config_file_path.to_string_lossy(),
336         ));
337         let host_param = make_host_params(&opt).unwrap();
338         assert_eq!(host_param.host_name.unwrap(), "multi.example.com");
339         assert_eq!(
340             host_param.identity_file.as_ref().unwrap()[0],
341             PathBuf::from("/home/mito/develop/rust/sshmount/test_data/dummy1_rsa")
342         );
343         assert_eq!(
344             host_param.identity_file.as_ref().unwrap()[1],
345             PathBuf::from("/home/mito/develop/rust/sshmount/test_data/dummy2_rsa")
346         );
347         assert_eq!(host_param.identity_file.as_ref().unwrap().len(), 2);
348     }
349
350     fn test_config_file_path() -> PathBuf {
351         let d = env!("CARGO_MANIFEST_DIR");
352         let mut p = PathBuf::new();
353         p.push(d);
354         p.push("test_data/config");
355         p
356     }
357
358     fn make_dummyidentity_file(no: u16) -> PathBuf {
359         let d = env!("CARGO_MANIFEST_DIR");
360         let mut p = PathBuf::new();
```

```
361             p.push(d);
362             p.push(format!("test_data/dummy{}_rsa", no));
363             p
364         }
365 
366         fn make_dummy_opt(cmdline: impl AsRef<str>) -> Opt {
367             let args = cmdline.as_ref().split_whitespace();
368             Opt::try_parse_from(args).unwrap()
369         }
370     }
```

5　FUSE 接続オプション生成モジュール fuse_util.rs

```rust
//! FUSEパラメータ関係　ユーティリティ

use crate::cmdline_opt::Opt;
use anyhow::{ensure, Context, Result};
use ssh2::Session;
use std::env::current_dir;
use std::{
    io::Read,
    path::{Path, PathBuf},
    str,
};

/// マウントポイントのフルパスを生成する
pub fn make_full_path<P: AsRef<Path>>(path: P) -> Result<PathBuf> {
    if path.as_ref().is_absolute() {
        Ok(path.as_ref().to_path_buf())
    } else {
        let mut full_path = current_dir().context("cannot access current directory.")?;
        full_path.push(path);
        Ok(full_path)
    }
}

/// リモート接続先の path の生成
pub fn make_remote_path(opt: &Opt, session: &Session) -> Result<PathBuf> {
    // パスの生成
    const MSG_ERRORHOME: &str = "Fail to generate path name.";
    let mut path = match opt.remote.path {
        Some(ref p) => {
            if p.is_absolute() {
                p.clone()
            } else {
                let mut h = get_home_on_remote(session).context(MSG_ERRORHOME)?;
                h.push(p);
                h
            }
        }
        None => get_home_on_remote(session).context(MSG_ERRORHOME)?,
    };
    // 生成したパスが実在するかを確認する
    let sftp = session
        .sftp()
        .context("Connection to SFTP failed when checking for existence of a path.")?;
```

```
44      let file_stat = sftp
45          .stat(&path)
46          .with_context(|| format!("Cannot find path to connect to. path={:?}", &path))?;
47      ensure!(
48          file_stat.is_dir(),
49          "The path to connect to is not a directory."
50      );
51      // 生成したパスがシンボリックリンクのときは、リンク先を解決する
52      let file_stat = sftp
53          .lstat(&path)
54          .context("Failed to obtain the attributes of the destination directory.")?;
55      if file_stat.file_type().is_symlink() {
56          path = sftp
57              .readlink(&path)
58              .context("Failed to resolve symbolic link to connect to.")?;
59          if !path.is_absolute() {
60              let tmp = path;
61              path = get_home_on_remote(session)
62                  .context("Failed to complete the symbolic link to connect to.")?;
63              path.push(tmp);
64          };
65      };
66
67      Ok(path)
68  }
69
70  /// FUSE の接続時オプションを生成する
71  pub fn make_mount_option(cmd_opt: &Opt) -> Vec<fuser::MountOption> {
72      use fuser::MountOption;
73
74      let mut options = vec![MountOption::FSName("sshfs".to_string())];
75      options.push(MountOption::NoDev);
76      options.push(MountOption::DirSync);
77      options.push(MountOption::Sync);
78      match cmd_opt.readonly {
79          true => options.push(MountOption::RO),
80          false => options.push(MountOption::RW),
81      }
82      match cmd_opt.no_exec {
83          true => options.push(MountOption::NoExec),
84          false => options.push(MountOption::Exec),
85      }
86      match cmd_opt.no_atime {
87          true => options.push(MountOption::NoAtime),
88          false => options.push(MountOption::Atime),
89      }
```

```
    options
}

/// ssh 接続先のカレントディレクトリを取得する
fn get_home_on_remote(session: &Session) -> Result<PathBuf> {
    let mut channel = session
        .channel_session()
        .context("Fail to build ssh channel.")?;
    channel
        .exec("pwd")
        .context("Fail to execute \"pwd\" command.")?;
    let mut buf = Vec::<u8>::new();
    channel
        .read_to_end(&mut buf)
        .context("Fail to get response for \"pwd\" command.")?;
    channel.close().context("Fail to close ssh channel.")?;
    str::from_utf8(&buf)
        .context("The pwd result contains non-utf8 characters.")?
        .trim()
        .parse::<PathBuf>()
        .context("Fail to build path name.")
}
```

6　ファイルシステムモジュール ssh_filesystem.rs

```rust
mod bi_hash_map;
mod file_handle;
mod inode;

use file_handle::Fhandles;
use inode::Inodes;

use anyhow::Context;
use fuser::{FileAttr, Filesystem, ReplyAttr, ReplyData, ReplyDirectory, ReplyEntry, Request};
use libc::ENOENT;
use log::{debug, error, warn};
use ssh2::{ErrorCode, OpenFlags, OpenType, Session, Sftp};
use std::{
    ffi::OsStr,
    io::{Read, Seek, SeekFrom, Write},
    path::{Path, PathBuf},
    time::{Duration, SystemTime, UNIX_EPOCH},
};

/// FUSE ファイルシステム実装
pub struct Sshfs {
    _session: Session,
    sftp: Sftp,
    inodes: Inodes,
    fhandls: Fhandles,
    _top_path: PathBuf,
}

impl Sshfs {
    pub fn new<P: AsRef<Path>>(session: Session, path: P) -> anyhow::Result<Self> {
        let mut inodes = Inodes::new();
        let top_path: PathBuf = path.as_ref().into();
        inodes.add(&top_path);
        let sftp = session
            .sftp()
            .inspect_err(|_| {
                error!("Failed to create sftp from session.");
            })
            .context("Failed to create sftp from session.(Sshfs::new)")?;
        debug!(
            "[Sshfs::new] connect path: <{:?}>, inodes=<{:?}>",
            &top_path, &inodes
        );
```

```
44          Ok(Self {
45              _session: session,
46              sftp,
47              inodes,
48              fhandls: Fhandles::new(),
49              _top_path: top_path,
50          })
51      }
52
53      /// ssh2経由でファイルのステータスを取得する。
54      /// 副作用: 取得に成功した場合、inodesにパスを登録する。
55      fn getattr_from_ssh2(&mut self, path: &Path, uid: u32, gid: u32) -> Result<FileAttr, Error> {
56          let attr_ssh2 = self.sftp.lstat(path)?;
57          let kind = Self::conv_file_kind_ssh2fuser(&attr_ssh2.file_type())?;
58          let ino = self.inodes.add(path);
59          Ok(FileAttr {
60              ino,
61              size: attr_ssh2.size.unwrap_or(0),
62              blocks: attr_ssh2.size.unwrap_or(0) / 512 + 1,
63              atime: UNIX_EPOCH + Duration::from_secs(attr_ssh2.atime.unwrap_or(0)),
64              mtime: UNIX_EPOCH + Duration::from_secs(attr_ssh2.mtime.unwrap_or(0)),
65              ctime: UNIX_EPOCH + Duration::from_secs(attr_ssh2.mtime.unwrap_or(0)),
66              crtime: UNIX_EPOCH,
67              kind,
68              perm: attr_ssh2.perm.unwrap_or(0o666) as u16,
69              nlink: 1,
70              uid,
71              gid,
72              rdev: 0,
73              blksize: 512,
74              flags: 0,
75          })
76      }
77
78      fn conv_file_kind_ssh2fuser(filetype: &ssh2::FileType) -> Result<fuser::FileType, Error> {
79          match filetype {
80              ssh2::FileType::NamedPipe => Ok(fuser::FileType::NamedPipe),
81              ssh2::FileType::CharDevice => Ok(fuser::FileType::CharDevice),
82              ssh2::FileType::BlockDevice => Ok(fuser::FileType::BlockDevice),
83              ssh2::FileType::Directory => Ok(fuser::FileType::Directory),
84              ssh2::FileType::RegularFile => Ok(fuser::FileType::RegularFile),
85              ssh2::FileType::Symlink => Ok(fuser::FileType::Symlink),
86              ssh2::FileType::Socket => Ok(fuser::FileType::Socket),
87              ssh2::FileType::Other(_) => Err(Error(libc::EBADF)),
88          }
89      }
```

```
90
91      fn conv_timeornow2systemtime(time: &fuser::TimeOrNow) -> SystemTime {
92          match time {
93              fuser::TimeOrNow::SpecificTime(t) => *t,
94              fuser::TimeOrNow::Now => SystemTime::now(),
95          }
96      }
97  }
98
99  impl Filesystem for Sshfs {
100     fn lookup(&mut self, req: &Request, parent: u64, name: &OsStr, reply: ReplyEntry) {
101         debug!("[lookup] start: parent={}, name={:?}", parent, name);
102         let Some(mut path) = self.inodes.get_path(parent) else {
103             debug!("[lookup] 親ディレクトリの検索に失敗 inode={}", parent);
104             reply.error(ENOENT);
105             return;
106         };
107         path.push(Path::new(name));
108         match self.getattr_from_ssh2(&path, req.uid(), req.gid()) {
109             Ok(attr) => reply.entry(&Duration::from_secs(1), &attr, 0),
110             Err(e) => {
111                 reply.error(e.0);
112             }
113         };
114     }
115
116     fn getattr(&mut self, req: &Request, ino: u64, _fh: Option<u64>, reply: ReplyAttr) {
117         debug!("[getattr] start: ino={}", ino);
118         let Some(path) = self.inodes.get_path(ino) else {
119             debug!("[getattr] path 取得失敗: inode={}", ino);
120             reply.error(ENOENT);
121             return;
122         };
123         match self.getattr_from_ssh2(&path, req.uid(), req.gid()) {
124             Ok(attr) => {
125                 //debug!("[getattr]retrun attr: {:?}", &attr);
126                 reply.attr(&Duration::from_secs(1), &attr);
127             }
128             Err(e) => {
129                 warn!("[getattr] getattr_from_ssh2 エラー: {:?}", &e);
130                 reply.error(e.0)
131             }
132         };
133     }
134
135     fn readdir(
```

```
        &mut self,
        _req: &Request,
        ino: u64,
        _fh: u64,
        offset: i64,
        mut reply: ReplyDirectory,
    ) {
        debug!("[readdir] start: ino={}, offset={}", ino, offset);
        let Some(path) = self.inodes.get_path(ino) else {
            reply.error(libc::ENOENT);
            return;
        };
        match self.sftp.readdir(&path) {
            Ok(mut dir) => {
                let cur_file_attr = ssh2::FileStat {
                    size: None,
                    uid: None,
                    gid: None,
                    perm: Some(libc::S_IFDIR),
                    atime: None,
                    mtime: None,
                }; // "." ".."の解決用。 attr ディレクトリであることのみを示す。
                dir.insert(0, (Path::new("..").into(), cur_file_attr.clone()));
                dir.insert(0, (Path::new(".").into(), cur_file_attr));
                let mut i = offset + 1;
                for f in dir.iter().skip(offset as usize) {
                    let ino = if f.0 == Path::new("..") || f.0 == Path::new(".") {
                        1
                    } else {
                        self.inodes.add(&f.0)
                    };
                    let name = match f.0.file_name() {
                        Some(n) => n,
                        None => f.0.as_os_str(),
                    };
                    let filetype = &f.1.file_type();
                    let filetype = match Self::conv_file_kind_ssh2fuser(filetype) {
                        Ok(t) => t,
                        Err(e) => {
                            warn!(
                                "[readdir] ファイルタイプ解析失敗: inode={}, name={:?}",
                                ino, name
                            );
                            reply.error(e.0);
                            return;
                        }
```

```
                    };
                    if reply.add(ino, i, filetype, name) {
                        break;
                    }
                    i += 1;
                }
                reply.ok();
            }
            Err(e) => {
                warn!("[readdir]ssh2::readdir 内でエラー発生-- {:?}", e);
                reply.error(Error::from(e).0);
            }
        };
    }

    fn readlink(&mut self, _req: &Request<'_>, ino: u64, reply: ReplyData) {
        debug!("[readlink] start: ino={}", ino);
        let Some(path) = self.inodes.get_path(ino) else {
            error!("[readlink] 親ディレクトリの検索に失敗 {ino}");
            reply.error(libc::ENOENT);
            return;
        };
        match self.sftp.readlink(&path) {
            Ok(p) => {
                //debug!("[readlink] ret_path => {:?}", &p);
                reply.data(p.as_os_str().to_string_lossy().as_bytes());
            }
            Err(e) => {
                //debug!("[readlink] ssh2::readlink error => {e:?}");
                reply.error(Error::from(e).0);
            }
        }
    }

    fn open(&mut self, _req: &Request<'_>, ino: u64, flags: i32, reply: fuser::ReplyOpen) {
        debug!("[open] start: ino={}, flags={:x}", ino, flags);
        let Some(file_name) = self.inodes.get_path(ino) else {
            reply.error(libc::ENOENT);
            return;
        };

        let mut flags_ssh2 = OpenFlags::empty();
        if flags & libc::O_WRONLY != 0 {
            flags_ssh2.insert(OpenFlags::WRITE);
        } else if flags & libc::O_RDWR != 0 {
            flags_ssh2.insert(OpenFlags::READ);
```

```
            flags_ssh2.insert(OpenFlags::WRITE);
        } else {
            flags_ssh2.insert(OpenFlags::READ);
        }
        if flags & libc::O_APPEND != 0 {
            flags_ssh2.insert(OpenFlags::APPEND);
        }
        if flags & libc::O_CREAT != 0 {
            flags_ssh2.insert(OpenFlags::CREATE);
        }
        if flags & libc::O_TRUNC != 0 {
            flags_ssh2.insert(OpenFlags::TRUNCATE);
        }
        if flags & libc::O_EXCL != 0 {
            flags_ssh2.insert(OpenFlags::EXCLUSIVE);
        }

        debug!(
            "[open] filename='{:?}', openflag = {:?}, bit = {:x}",
            &file_name,
            &flags_ssh2,
            flags_ssh2.bits()
        );
        match self
            .sftp
            .open_mode(&file_name, flags_ssh2, 0o777, ssh2::OpenType::File)
        {
            Ok(file) => {
                let fh = self.fhandls.add_file(file);
                reply.opened(fh, flags as u32);
            }
            Err(e) => {
                log::error!(
                    "file-open error: filename='{:?}', mode={:?}, err={}",
                    &file_name,
                    &flags_ssh2,
                    &e
                );
                reply.error(Error::from(e).0);
            }
        }
    }

    fn release(
        &mut self,
        _req: &Request<'_>,
```

```
        _ino: u64,
        fh: u64,
        _flags: i32,
        _lock_owner: Option<u64>,
        _flush: bool,
        reply: fuser::ReplyEmpty,
    ) {
        debug!("[release] start: fh={}", fh);
        self.fhandls.del_file(fh);
        reply.ok();
    }

    fn read(
        &mut self,
        _req: &Request,
        _ino: u64,
        fh: u64,
        offset: i64,
        size: u32,
        _flags: i32,
        _lock_owner: Option<u64>,
        reply: ReplyData,
    ) {
        debug!("[read] start: fh={}, offset={}, size={}", fh, offset, size);
        let Some(file_mutex) = self.fhandls.get_file(fh) else {
            reply.error(libc::EBADF);
            return;
        };
        let mut file = file_mutex.lock().unwrap();
        // 注釈: このデータの出所である Fhandles 構造体内のデータに毒化があるということは、
        // システム全域の他のファイルハンドルの正当性も保証できないことを意味する。
        // ファイル操作を失敗させることより、システム全体を落とすことが正しい選択と思われる。
        // よって、lock().unwrap() とする。write() 関数他においても同様。

        if let Err(e) = file.seek(std::io::SeekFrom::Start(offset as u64)) {
            reply.error(Error::from(e).0);
            return;
        }
        let mut buff = vec![0; size as usize];
        let mut read_size: usize = 0;
        while read_size < size as usize {
            match file.read(&mut buff[read_size..]) {
                Ok(s) => {
                    if s == 0 {
                        break;
                    };
```

```
                    read_size += s;
                }
                Err(e) => {
                    reply.error(Error::from(e).0);
                    return;
                }
            }
        }
        buff.resize(read_size, 0u8);
        reply.data(&buff);
    }

    fn write(
        &mut self,
        _req: &Request<'_>,
        _ino: u64,
        fh: u64,
        offset: i64,
        data: &[u8],
        _write_flags: u32,
        _flags: i32,
        _lock_owner: Option<u64>,
        reply: fuser::ReplyWrite,
    ) {
        debug!(
            "[write] start: fh={}, offset={}, size={}",
            fh,
            offset,
            data.len()
        );
        let Some(file_mutex) = self.fhandls.get_file(fh) else {
            reply.error(libc::EBADF);
            return;
        };
        let file = &mut file_mutex.lock().unwrap();

        if let Err(e) = file.seek(std::io::SeekFrom::Start(offset as u64)) {
            reply.error(Error::from(e).0);
            return;
        }
        let mut buf = data;
        while !buf.is_empty() {
            let cnt = match file.write(buf) {
                Ok(cnt) => cnt,
                Err(e) => {
                    reply.error(Error::from(e).0);
```

```
                    return;
                }
            };
            buf = &buf[cnt..];
        }
        reply.written(data.len() as u32);
    }

    fn lseek(
        &mut self,
        _req: &Request<'_>,
        _ino: u64,
        fh: u64,
        offset: i64,
        whence: i32,
        reply: fuser::ReplyLseek,
    ) {
        debug!(
            "[lseek] start: fh={}, offset={}, whence={}",
            fh, offset, whence
        );
        let seek_from = match whence {
            libc::SEEK_SET => {
                if offset >= 0 {
                    SeekFrom::Start(offset as u64)
                } else {
                    reply.error(libc::EINVAL);
                    return;
                }
            }
            libc::SEEK_CUR => SeekFrom::Current(offset),
            libc::SEEK_END => SeekFrom::End(offset),
            _ => {
                reply.error(libc::EINVAL);
                return;
            }
        };
        let Some(file_mutex) = self.fhandls.get_file(fh) else {
            reply.error(libc::EBADF);
            return;
        };
        let file = &mut file_mutex.lock().unwrap();
        let pos = match file.seek(seek_from) {
            Ok(p) => p,
            Err(e) => {
                reply.error(Error::from(e).0);
```

```
                return;
            }
        };
        reply.offset(pos as i64);
    }

    fn mknod(
        &mut self,
        req: &Request<'_>,
        parent: u64,
        name: &OsStr,
        mode: u32,
        umask: u32,
        _rdev: u32,
        reply: ReplyEntry,
    ) {
        debug!(
            "[mknod] start: parent={}, name={:?}, mode={:o}, umask={:o}",
            parent, name, mode, umask
        );
        if mode & libc::S_IFMT != libc::S_IFREG {
            reply.error(libc::EPERM);
            return;
        }
        let mode = mode & (!umask | libc::S_IFMT);
        let Some(mut new_name) = self.inodes.get_path(parent) else {
            reply.error(libc::ENOENT);
            return;
        };
        new_name.push(name);
        if let Err(e) =
            self.sftp
                .open_mode(&new_name, OpenFlags::CREATE, mode as i32, OpenType::File)
        {
            reply.error(Error::from(e).0);
            return;
        }
        let new_attr = match self.getattr_from_ssh2(&new_name, req.uid(), req.gid()) {
            Ok(a) => a,
            Err(e) => {
                reply.error(e.0);
                return;
            }
        };
        reply.entry(&Duration::from_secs(1), &new_attr, 0);
    }
```

```rust

    fn unlink(&mut self, _req: &Request<'_>, parent: u64, name: &OsStr, reply: fuser::ReplyEmpty) {
        debug!("[unlink] start: parent={}, name={:?}", parent, name);
        let Some(mut path) = self.inodes.get_path(parent) else {
            reply.error(libc::ENOENT);
            return;
        };
        path.push(name);
        match self.sftp.unlink(&path) {
            Ok(_) => {
                self.inodes.del_inode_with_path(&path);
                reply.ok();
            }
            Err(e) => reply.error(Error::from(e).0),
        }
    }

    fn mkdir(
        &mut self,
        req: &Request<'_>,
        parent: u64,
        name: &OsStr,
        mode: u32,
        umask: u32,
        reply: ReplyEntry,
    ) {
        debug!(
            "[mkdir] start: parent={}, name={:?}, mode={:o}, umask={:o}",
            parent, name, mode, umask
        );
        let Some(mut path) = self.inodes.get_path(parent) else {
            reply.error(libc::ENOENT);
            return;
        };
        path.push(name);

        let mode = (mode & (!umask) & 0o777) as i32;

        match self.sftp.mkdir(&path, mode) {
            Ok(_) => match self.getattr_from_ssh2(&path, req.uid(), req.gid()) {
                Ok(attr) => reply.entry(&Duration::from_secs(1), &attr, 0),
                Err(e) => reply.error(e.0),
            },
            Err(e) => reply.error(Error::from(e).0),
        }
    }
```

```
504
505     fn rmdir(&mut self, _req: &Request<'_>, parent: u64, name: &OsStr, reply: fuser::ReplyEmpty) {
506         debug!("[rmdir] start: parent={}, name={:?}", parent, name);
507         let Some(mut path) = self.inodes.get_path(parent) else {
508             reply.error(libc::ENOENT);
509             return;
510         };
511         path.push(name);
512         match self.sftp.rmdir(&path) {
513             Ok(_) => {
514                 self.inodes.del_inode_with_path(&path);
515                 reply.ok()
516             }
517             Err(e) => {
518                 if e.code() == ErrorCode::Session(-31) {
519                     // ssh2 ライブラリの返すエラーが妙。置換しておく。
520                     reply.error(libc::ENOTEMPTY);
521                 } else {
522                     reply.error(Error::from(e).0)
523                 }
524             }
525         }
526     }
527
528     fn symlink(
529         &mut self,
530         req: &Request<'_>,
531         parent: u64,
532         name: &OsStr,
533         link: &Path,
534         reply: ReplyEntry,
535     ) {
536         debug!(
537             "[symlink] start: parent={}, name={:?}, link={:?}",
538             parent, name, link
539         );
540         let Some(mut target) = self.inodes.get_path(parent) else {
541             reply.error(libc::ENOENT);
542             return;
543         };
544         target.push(name);
545         match self.sftp.symlink(link, &target) {
546             Ok(_) => match self.getattr_from_ssh2(&target, req.uid(), req.gid()) {
547                 Ok(attr) => reply.entry(&Duration::from_secs(1), &attr, 0),
548                 Err(e) => reply.error(e.0),
549             },
```

```
            Err(e) => reply.error(Error::from(e).0),
        }
    }

    fn setattr(
        &mut self,
        req: &Request<'_>,
        ino: u64,
        mode: Option<u32>,
        _uid: Option<u32>,
        _gid: Option<u32>,
        size: Option<u64>,
        atime: Option<fuser::TimeOrNow>,
        mtime: Option<fuser::TimeOrNow>,
        _ctime: Option<std::time::SystemTime>,
        _fh: Option<u64>,
        _crtime: Option<std::time::SystemTime>,
        _chgtime: Option<std::time::SystemTime>,
        _bkuptime: Option<std::time::SystemTime>,
        _flags: Option<u32>,
        reply: ReplyAttr,
    ) {
        debug!(
            "[setattr] start: ino={}, mode={:?}, size={:?}, atime={:?}, mtime={:?}",
            ino, mode, size, atime, mtime
        );
        let stat = ssh2::FileStat {
            size,
            uid: None,
            gid: None,
            perm: mode,
            atime: atime.map(|t| {
                Self::conv_timeornow2systemtime(&t)
                    .duration_since(UNIX_EPOCH)
                    .unwrap_or_default()
                    .as_secs()
            }),
            mtime: mtime.map(|t| {
                Self::conv_timeornow2systemtime(&t)
                    .duration_since(UNIX_EPOCH)
                    .unwrap_or_default()
                    .as_secs()
            }),
        };
        let Some(filename) = self.inodes.get_path(ino) else {
            reply.error(ENOENT);
```

```
            return;
        };
        match self.sftp.setstat(&filename, stat) {
            Ok(_) => {
                let stat = self.getattr_from_ssh2(&filename, req.uid(), req.gid());
                match stat {
                    Ok(s) => reply.attr(&Duration::from_secs(1), &s),
                    Err(e) => reply.error(e.0),
                }
            }
            Err(e) => reply.error(Error::from(e).0),
        }
    }

    fn rename(
        &mut self,
        _req: &Request<'_>,
        parent: u64,
        name: &OsStr,
        newparent: u64,
        newname: &OsStr,
        flags: u32,
        reply: fuser::ReplyEmpty,
    ) {
        debug!(
            "[rename] start: parent={}, name={:?}, newparent={}, newname={:?}, flags={}",
            parent, name, newparent, newname, flags
        );
        let Some(mut old_path) = self.inodes.get_path(parent) else {
            reply.error(libc::ENOENT);
            return;
        };
        old_path.push(name);

        let Some(mut new_path) = self.inodes.get_path(newparent) else {
            reply.error(libc::ENOENT);
            return;
        };
        new_path.push(newname);

        let mut rename_flag = ssh2::RenameFlags::NATIVE;
        if flags & libc::RENAME_EXCHANGE != 0 {
            rename_flag.insert(ssh2::RenameFlags::ATOMIC);
        }
        if flags & libc::RENAME_NOREPLACE == 0 {
            // rename の OVERWRITE が効いてない。手動で消す。
```

```
642             if let Ok(stat) = self.sftp.lstat(&new_path) {
643                 if stat.is_dir() {
644                     if let Err(e) = self.sftp.rmdir(&new_path) {
645                         reply.error(Error::from(e).0);
646                         return;
647                     }
648                 } else if let Err(e) = self.sftp.unlink(&new_path) {
649                     reply.error(Error::from(e).0);
650                     return;
651                 }
652                 self.inodes.del_inode_with_path(&new_path);
653             }
654         }
655 
656         match self.sftp.rename(&old_path, &new_path, Some(rename_flag)) {
657             Ok(_) => {
658                 self.inodes.rename(&old_path, &new_path);
659                 reply.ok();
660             }
661             Err(e) => reply.error(Error::from(e).0),
662         }
663     }
664 }
665 
666 #[derive(Debug, Clone, Copy)]
667 struct Error(i32);
668 
669 impl From<ssh2::Error> for Error {
670     fn from(value: ssh2::Error) -> Self {
671         let eno = match value.code() {
672             ssh2::ErrorCode::Session(_) => libc::ENXIO,
673             ssh2::ErrorCode::SFTP(i) => match i {
674                 // libssh2の libssh2_sftp.hにて定義されている。
675                 2 => libc::ENOENT,          // NO_SUCH_FILE
676                 3 => libc::EACCES,          // permission_denied
677                 4 => libc::EIO,             // failure
678                 5 => libc::ENODEV,          // bad message
679                 6 => libc::ENXIO,           // no connection
680                 7 => libc::ENETDOWN,        // connection lost
681                 8 => libc::ENODEV,          // unsported
682                 9 => libc::EBADF,           // invalid handle
683                 10 => libc::ENOENT,         //no such path
684                 11 => libc::EEXIST,         // file already exists
685                 12 => libc::EACCES,         // write protected
686                 13 => libc::ENXIO,          // no media
687                 14 => libc::ENOSPC,         // no space on filesystem
```

```
                15 => libc::EDQUOT,       // quota exceeded
                16 => libc::ENODEV,       // unknown principal
                17 => libc::ENOLCK,       // lock conflict
                18 => libc::ENOTEMPTY,    // dir not empty
                19 => libc::ENOTDIR,      // not a directory
                20 => libc::ENAMETOOLONG, // invalid file name
                21 => libc::ELOOP,        // link loop
                _ => {
                    error!("An unknown error occurred during SSH2.[{}]", i);
                    libc::EIO
                }
            },
        };
        Self(eno)
    }
}

impl From<std::io::Error> for Error {
    fn from(value: std::io::Error) -> Self {
        use std::io::ErrorKind::*;
        let eno = match value.kind() {
            NotFound => libc::ENOENT,
            PermissionDenied => libc::EACCES,
            ConnectionRefused => libc::ECONNREFUSED,
            ConnectionReset => libc::ECONNRESET,
            ConnectionAborted => libc::ECONNABORTED,
            NotConnected => libc::ENOTCONN,
            AddrInUse => libc::EADDRINUSE,
            AddrNotAvailable => libc::EADDRNOTAVAIL,
            BrokenPipe => libc::EPIPE,
            AlreadyExists => libc::EEXIST,
            WouldBlock => libc::EWOULDBLOCK,
            InvalidInput => libc::EINVAL,
            InvalidData => libc::EILSEQ,
            TimedOut => libc::ETIMEDOUT,
            WriteZero => libc::EIO,
            Interrupted => libc::EINTR,
            Unsupported => libc::ENOTSUP,
            UnexpectedEof => libc::EOF,
            OutOfMemory => libc::ENOMEM,
            _ => {
                error!(
                    "An unknown error occurred during std::io.[{}]",
                    value.kind()
                );
                libc::EIO
```

```
734                 }
735             };
736             Self(eno)
737         }
738     }
```

7　ファイルハンドル管理モジュール ssh_filesystem/file_handle.rs

```rust
//! ファイルハンドル管理モジュール

use std::collections::HashMap;
use std::sync::{
    atomic::{AtomicU64, Ordering},
    Arc, Mutex,
};

/// ファイルハンドル管理構造体
pub(super) struct Fhandles {
    list: Mutex<HashMap<u64, Arc<Mutex<ssh2::File>>>>,
    next_handle: AtomicU64,
}

impl Fhandles {
    pub(super) fn new() -> Self {
        Self {
            list: Mutex::new(HashMap::new()),
            next_handle: AtomicU64::new(0),
        }
    }

    pub(super) fn add_file(&mut self, file: ssh2::File) -> u64 {
        let handle = self.next_handle.fetch_add(1, Ordering::AcqRel);
        self.list
            .lock()
            .unwrap()
            .insert(handle, Arc::new(Mutex::new(file)));
        handle
        // 注釈: このリストが毒化されたら、もはや、全システムにわたり、ファイル操作の正当性を保証できない。
        // プログラムとしてできることは即座にシステムを落とすことだけである。
        // よって、このモジュール内において、lock().unwrap() とする。
    }

    pub(super) fn get_file(&self, fh: u64) -> Option<Arc<Mutex<ssh2::File>>> {
        self.list.lock().unwrap().get(&fh).cloned()
    }

    pub(super) fn del_file(&mut self, fh: u64) {
        self.list.lock().unwrap().remove(&fh); // 戻り値は捨てる。この時点でファイルはクローズ。
    }
}
```

8 Inode 管理モジュール ssh_filesystem/inode.rs

```rust
//! Inode 管理モジュール

use super::bi_hash_map::BiHashMap;

use std::path::{Path, PathBuf};
use std::sync::{
    atomic::{AtomicU64, Ordering},
    Mutex,
};

/// Inode 管理構造体
#[derive(Debug, Default)]
pub(super) struct Inodes {
    list: Mutex<BiHashMap<u64, PathBuf>>,
    next_inode: AtomicU64,
}

impl Inodes {
    /// Inodes を生成する
    pub(super) fn new() -> Self {
        Self {
            list: Mutex::new(BiHashMap::new()),
            next_inode: AtomicU64::new(1),
        }
    }

    /// path で指定された inode を生成し、登録する。
    /// すでに path の登録が存在する場合、追加はせず、登録済みの inode を返す。
    /// 初めて、この関数が呼ばれるときは、ファイルシステムにおけるルートであり、inode 番号 1 が割り当てられる。
    pub(super) fn add<P: AsRef<Path>>(&mut self, path: P) -> u64 {
        let mut list_guard = self.list.lock().unwrap();
        // 注釈: このリストが毒化されたら、もはや、全システムにわたり、inode 管理の正当性を保証できない。
        // 最善の方法が、即時システムを落とすことである。
        // 以下、このモジュール全体に共通。
        let path = PathBuf::from(path.as_ref());
        match list_guard.get_left(&path) {
            Some(i) => *i,
            None => {
                let inode = self.next_inode.fetch_add(1, Ordering::AcqRel);
                if list_guard.insert_no_overwrite(inode, path.clone()).is_err() {
                    unreachable!("Unexpected duplicate inode {} or path {:?}", inode, path);
                    // 既に重複がチェックされているので、ありえない。
```

```
                }
                inode
            }
        }
    }

    /// path から inode を取得する
    /// 主用途が消滅したが、将来のために残しておく
    #[allow(dead_code)]
    pub(super) fn get_inode<P: AsRef<Path>>(&self, path: P) -> Option<u64> {
        let path = PathBuf::from(path.as_ref());
        self.list.lock().unwrap().get_left(&path).copied()
    }

    /// inode から path を取得する
    pub(super) fn get_path(&self, inode: u64) -> Option<PathBuf> {
        self.list.lock().unwrap().get_right(&inode).cloned()
    }

    /// inodes から、inode の登録を削除する
    /// (主用途がなくなっちゃったけど、将来のために残しておく)
    #[allow(dead_code)]
    pub(super) fn del_inode(&mut self, inode: u64) -> Option<u64> {
        self.list.lock().unwrap().remove_left(&inode).map(|_| inode)
    }

    /// path 名から iNode の登録を削除する
    pub(super) fn del_inode_with_path<P: AsRef<Path>>(&mut self, path: P) -> Option<u64> {
        let path = PathBuf::from(path.as_ref());
        self.list.lock().unwrap().remove_right(&path)
    }

    /// 登録されている inode の path を変更する。
    /// old_path が存在しなければ、なにもしない。
    pub(super) fn rename<P: AsRef<Path>>(&mut self, old_path: P, new_path: P) {
        let old_path = PathBuf::from(old_path.as_ref());
        let new_path = PathBuf::from(new_path.as_ref());
        let mut list_guard = self.list.lock().unwrap();
        let Some(ino) = list_guard.get_left(&old_path).copied() else {
            return;
        };
        list_guard.remove_left(&ino);
        list_guard.insert(ino, new_path);
    }
}
```

```rust
#[cfg(test)]
mod inode_test {
    use super::Inodes;
    use std::path::Path;

    #[test]
    fn inode_add_test() {
        let mut inodes = Inodes::new();
        assert_eq!(inodes.add(""), 1);
        assert_eq!(inodes.add(Path::new("test")), 2);
        assert_eq!(inodes.add(Path::new("")), 1);
        assert_eq!(inodes.add(Path::new("test")), 2);
        assert_eq!(inodes.add(Path::new("test3")), 3);
        assert_eq!(inodes.add(Path::new("/test")), 4);
        assert_eq!(inodes.add(Path::new("test/")), 2);
    }

    fn make_inodes() -> Inodes {
        let mut inodes = Inodes::new();
        inodes.add(Path::new(""));
        inodes.add(Path::new("test"));
        inodes.add(Path::new("test2"));
        inodes.add(Path::new("test3/"));
        inodes
    }

    #[test]
    fn inodes_get_inode_test() {
        let inodes = make_inodes();
        assert_eq!(inodes.get_inode(Path::new("")), Some(1));
        assert_eq!(inodes.get_inode(Path::new("test4")), None);
        assert_eq!(inodes.get_inode(Path::new("/test")), None);
        assert_eq!(inodes.get_inode(Path::new("test3")), Some(4));
    }

    #[test]
    fn inodes_get_path_test() {
        let inodes = make_inodes();
        assert_eq!(inodes.get_path(1), Some(Path::new("").into()));
        assert_eq!(inodes.get_path(3), Some(Path::new("test2").into()));
        assert_eq!(inodes.get_path(5), None);
        assert_eq!(inodes.get_path(3), Some(Path::new("test2/").into()));
    }

    #[test]
    fn inodes_rename() {
```

```
135         let mut inodes = make_inodes();
136         let old = Path::new("test2");
137         let new = Path::new("new_test");
138         let ino = inodes.get_inode(old).unwrap();
139         inodes.rename(old, new);
140         assert_eq!(inodes.get_path(ino), Some(new.into()));
141
142         let mut inodes = make_inodes();
143         let inodes2 = make_inodes();
144         inodes.rename(Path::new("nai"), Path::new("kawattenai"));
145         assert_eq!(*inodes.list.lock().unwrap(), *inodes2.list.lock().unwrap());
146     }
147 }
```

9　双方向ハッシュマップモジュール ssh_filesystem/bi_hash_map.rs

```rust
//! bidirectional hash map
//! 双方向ハッシュマップ

use std::{collections::HashMap, hash::Hash, sync::Arc};

/// 双方向ハッシュマップ
#[derive(Debug, Default, PartialEq, Eq)]
pub(super) struct BiHashMap<L, R>
where
    L: Hash + Eq + Clone,
    R: Hash + Eq + Clone,
{
    left: HashMap<Arc<L>, Arc<R>>,
    right: HashMap<Arc<R>, Arc<L>>,
}

impl<L, R> BiHashMap<L, R>
where
    L: Hash + Eq + Clone,
    R: Hash + Eq + Clone,
{
    /// 新しい双方向ハッシュマップを生成する
    pub fn new() -> Self {
        BiHashMap {
            right: HashMap::new(),
            left: HashMap::new(),
        }
    }

    /// チェック無しで挿入する。
    /// この時点で与えられる引数は、R,L のいずれも既存のキーと重複しないことが保証されている必要がある。
    fn insert_no_check(&mut self, left: L, right: R) {
        let right = Arc::new(right);
        let left = Arc::new(left);
        self.right.insert(right.clone(), left.clone());
        self.left.insert(left, right);
    }

    /// マップに新しい要素を挿入する。
    /// 既存のキーと重複する場合、対応する値を上書きし、OverwriteResult で通知する。
    pub fn insert(&mut self, left: L, right: R) -> OverwriteResult<L, R> {
        let old_left = self.right.remove(&right);
        let old_right = self.left.remove(&left);
```

```
        let result = match (old_left, old_right) {
            (None, None) => OverwriteResult::NoOverwrite,
            (Some(old_l), None) => {
                self.left.remove(old_l.as_ref());
                OverwriteResult::OverwriteLeft((*old_l).clone())
            }
            (None, Some(old_r)) => {
                self.right.remove(old_r.as_ref());
                OverwriteResult::OverwriteRight((*old_r).clone())
            }
            (Some(old_l), Some(old_r)) => {
                self.left.remove(old_l.as_ref());
                self.right.remove(old_r.as_ref());
                OverwriteResult::OverwriteBoth(
                    (left.clone(), (*old_r).clone()),
                    ((*old_l).clone(), right.clone()),
                )
            }
        };
        self.insert_no_check(left, right);
        result
    }

    /// マップに新しい値を挿入する
    /// 既存のキーが存在する場合は、エラーを返す。
    pub fn insert_no_overwrite(&mut self, left: L, right: R) -> Result<(), ()> {
        if self.contains_left(&left) || self.contains_right(&right) {
            return Err(());
        }
        self.insert_no_check(left, right);
        Ok(())
    }

    /// 左側のキーから右側の値を取得する
    /// 存在しない場合は *None* を返す
    pub fn get_right(&self, left: &L) -> Option<&R> {
        self.left.get(left).map(|arc_r| arc_r.as_ref())
    }

    /// 右側のキーから左側の値を取得する
    /// 存在しない場合は *None* を返す
    pub fn get_left(&self, right: &R) -> Option<&L> {
        self.right.get(right).map(|arc_l| arc_l.as_ref())
    }

    /// 左側のキーが存在するかどうかを返す
```

```rust
    /// 存在する場合は true、存在しない場合は false を返す
    pub fn contains_left(&self, left: &L) -> bool {
        self.left.contains_key(left)
    }

    /// 右側のキーが存在するかどうかを返す
    /// 存在する場合は true、存在しない場合は false を返す
    pub fn contains_right(&self, right: &R) -> bool {
        self.right.contains_key(right)
    }

    /// 左側の値から、リストの項目を削除する
    /// 存在しない場合は、なにもしない。
    /// 以前の値を返す。(存在しない場合は None)
    pub fn remove_left(&mut self, left: &L) -> Option<R> {
        let result = self.left.remove(left).map(|arc_r| (*arc_r).clone());
        if let Some(right) = result.as_ref() {
            self.right.remove(right);
        };
        result
    }

    /// 右側の値から、リストの項目を削除する
    /// 存在しない場合は、なにもしない。
    /// 以前の値を返す。(存在しない場合は None)
    pub fn remove_right(&mut self, right: &R) -> Option<L> {
        let result = self.right.remove(right).map(|arc_l| (*arc_l).clone());
        if let Some(left) = result.as_ref() {
            self.left.remove(left);
        };
        result
    }
}

/// 挿入時の上書き結果
#[derive(Debug, PartialEq, Eq)]
pub enum OverwriteResult<L, R> {
    NoOverwrite,
    OverwriteRight(R),
    OverwriteLeft(L),
    OverwriteBoth((L, R), (L, R)),
}

#[cfg(test)]
mod tests {
    use super::*;
```

```rust

    #[test]
    /// 単純な挿入と、値の取得のテスト
    fn tanjyunna_insert() {
        let mut bimap = BiHashMap::new();
        assert_eq!(bimap.insert(1, "a"), OverwriteResult::NoOverwrite);
        assert_eq!(bimap.get_right(&1), Some(&"a"));
        assert_eq!(bimap.get_left(&"a"), Some(&1));
        assert_eq!(bimap.insert_no_overwrite(2, "b"), Ok(()));
        assert_eq!(bimap.get_right(&2), Some(&"b"));
        assert_eq!(bimap.get_left(&"b"), Some(&2));
        assert!(bimap.contains_left(&1));
        assert!(bimap.contains_right(&"b"));
    }

    #[test]
    /// 上書き挿入のテスト
    /// 左側、右側、両側の上書きのケースを確認する
    fn overwrite_insert() {
        let mut bimap = BiHashMap::new();
        assert_eq!(bimap.insert(1, "a"), OverwriteResult::NoOverwrite);
        print_hash_map(&bimap, "insert (1, 'a')");
        assert_eq!(bimap.insert(1, "b"), OverwriteResult::OverwriteRight("a"));
        print_hash_map(&bimap, "insert (1, 'b')");
        assert_eq!(bimap.get_right(&1), Some(&"b"));
        assert_eq!(bimap.insert(2, "b"), OverwriteResult::OverwriteLeft(1));
        print_hash_map(&bimap, "insert (2, 'b')");
        assert_eq!(bimap.get_left(&"b"), Some(&2));
        assert_eq!(bimap.get_right(&2), Some(&"b"));
        assert_eq!(bimap.insert_no_overwrite(3, "c"), Ok(()));
        print_hash_map(&bimap, "insert (3, 'c')");
        assert_eq!(
            bimap.insert(2, "b"),
            OverwriteResult::OverwriteBoth((2, "b"), (2, "b"))
        );
        print_hash_map(&bimap, "insert (2, 'b') again");
        assert_eq!(
            bimap.insert(3, "b"),
            OverwriteResult::OverwriteBoth((3, "c"), (2, "b"))
        );
        print_hash_map(&bimap, "insert (3, 'b')");
        assert_eq!(bimap.get_right(&3), Some(&"b"));
        assert_eq!(bimap.get_left(&"b"), Some(&3));
        assert_eq!(bimap.left.len(), bimap.right.len());
        assert_eq!(bimap.get_right(&2), None);
        assert_eq!(bimap.insert_no_overwrite(3, "d"), Err(()));
```

```
182         print_hash_map(&bimap, "insert_no_overwright (3, 'd') [fail]");
183         assert_eq!(bimap.insert_no_overwrite(5, "e"), Ok(()));
184         print_hash_map(&bimap, "insert_no_overwright (5, 'e') [ok]");
185         assert_eq!(bimap.insert_no_overwrite(5, "d"), Err(()));
186         print_hash_map(&bimap, "insert_no_overwright (5, 'd') [fail]");
187     }
188
189     /// 左右の削除のテスト
190     #[test]
191     fn remove_test() {
192         let mut bimap = BiHashMap::new();
193         bimap.insert_no_check(1, "a");
194         bimap.insert_no_check(2, "b");
195         bimap.insert_no_check(3, "c");
196         print_hash_map(&bimap, "initial map");
197         assert_eq!(bimap.remove_left(&2), Some("b"));
198         print_hash_map(&bimap, "after remove_left(2)");
199         assert_eq!(bimap.get_left(&"b"), None);
200         assert_eq!(bimap.get_right(&2), None);
201         assert_eq!(bimap.remove_right(&"c"), Some(3));
202         print_hash_map(&bimap, "after remove_right('c')");
203         assert_eq!(bimap.get_left(&"c"), None);
204         assert_eq!(bimap.get_right(&3), None);
205         assert_eq!(bimap.remove_left(&4), None);
206         print_hash_map(&bimap, "after remove_left(4) [no op]");
207         assert_eq!(bimap.remove_right(&"d"), None);
208         print_hash_map(&bimap, "after remove_right('d') [no op]");
209     }
210
211     use std::fmt::Debug;
212     fn print_hash_map<R, L>(bimap: &BiHashMap<R, L>, mes: &str)
213     where
214         R: Debug + Hash + Eq + Clone,
215         L: Debug + Hash + Eq + Clone,
216     {
217         println!("=== {} ===", mes);
218         println!("Left to Right:");
219         for (l, r) in &bimap.left {
220             println!("  {:?} => {:?}", l, r);
221         }
222         println!("Right to Left:");
223         for (r, l) in &bimap.right {
224             println!("  {:?} => {:?}", r, l);
225         }
226     }
227 }
```