sshmount ソースリスト

美都

2025 年 10 月 19 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
use anyhow::{anyhow, Context};
use clap::Parser;
use std::path::PathBuf;

/// コマンドラインオプション
#[derive(Parser)]
#[command(author, version, about)]
pub struct Opt {
    /// Distination [user@]host:[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, default_value_t = 22)]
    pub port: 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,
}

/// 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)"),
        },
    }
}

/// 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>,
    /// ホスト名　または　IPアドレス
    pub host: String,
    /// 接続先パス
    pub path: Option<std::path::PathBuf>,
}

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.path);
        s.fmt(f)
    }
}

impl std::str::FromStr for RemoteName {
    type Err = ErrorRemoteName;
    fn from_str(s: &str) -> Result<Self, Self::Err> {
        let mut rest_str = s;
        let user = match rest_str.split_once('@') {
            Some((u, r)) => {
                rest_str = r;
                if !u.trim().is_empty() {
                    Some(u.trim().to_string())
                } else {
                    None
                }
            }
            None => None,
        };
```

```
        let (host, path) = match rest_str.split_once(':') {
            Some((h, p)) => (
                if !h.trim().is_empty() {
                    h.trim().to_string()
                } else {
                    return Err(ErrorRemoteName);
                },
                if !p.trim().is_empty() {
                    Some(std::path::PathBuf::from(p.trim().to_string()))
                } else {
                    None
                },
            ),
            None => return Err(ErrorRemoteName),
        };
        Ok(Self { user, host, path })
    }
}

#[derive(thiserror::Error, Debug, PartialEq, Eq)]
#[error("The format of the host to connect to is \"[user@]host:[path]\".")]
pub struct ErrorRemoteName;

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

    #[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: "reterminal.local".to_string(),
            path: Some(Path::new("/home/mito").into()),
        };
        assert_eq!(r, k);

        let s = "mito@reterminal.local:/home/mito/";
        let r: RemoteName = s.parse().unwrap();
```

```
136         let k = RemoteName {
137             user: Some("mito".to_string()),
138             host: "reterminal.local".to_string(),
139             path: Some(Path::new("/home/mito").into()),
140         };
141         assert_eq!(r, k);
142 
143         let s = "reterminal.local:";
144         let r: RemoteName = s.parse().unwrap();
145         let k = RemoteName {
146             user: None,
147             host: "reterminal.local".to_string(),
148             path: None,
149         };
150         assert_eq!(r, k);
151 
152         let s = " mito @reterminal.local: ";
153         let r: RemoteName = s.parse().unwrap();
154         let k = RemoteName {
155             user: Some("mito".to_string()),
156             host: "reterminal.local".to_string(),
157             path: None,
158         };
159         assert_eq!(r, k);
160 
161         let s = "reterminal.local";
162         let r: Result<RemoteName, ErrorRemoteName> = s.parse();
163         assert_eq!(r, Err(ErrorRemoteName));
164 
165         let s = "mito@reterminal.local";
166         let r: Result<RemoteName, ErrorRemoteName> = s.parse();
167         assert_eq!(r, Err(ErrorRemoteName));
168 
169         let s = " mito @: ";
170         let r: Result<RemoteName, ErrorRemoteName> = s.parse();
171         assert_eq!(r, Err(ErrorRemoteName));
172     }
173 }
```

3　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;
use ssh2::Session;
use ssh2_config::{HostParams, ParseRule, SshConfig};
use std::{
    fs::File,
    io::BufReader,
    net::TcpStream,
    path::{Path, PathBuf},
    str,
};

/// セッションを生成する。
pub fn make_ssh_session(opt: &Opt) -> Result<Session> {
    let host_params = get_ssh_config(&opt.config_file).query(&opt.remote.host);
    let address = get_address(opt, &host_params).context("Failed to get host address")?;
    let username = get_username(opt, &host_params).context("Failed to get user name.")?;
    debug!(
        "[main] 接続先情報-> ユーザー:\"{}\", ip address:{:?}",
        &username, &address
    );
    let identity_file = get_identity_file(opt, &host_params)?;

    let ssh = connect_ssh(address).context("The ssh connection failed.")?;
    userauth(&ssh, &username, &identity_file).context("User authentication failed.")?;
    Ok(ssh)
}

/// ホストの ip アドレス解決
fn get_address(opt: &Opt, host_params: &HostParams) -> Result<std::net::SocketAddr> {
    let dns = host_params.host_name.as_deref().unwrap_or(&opt.remote.host);
    let addr = lookup_host(dns)
        .context("Cannot find host to connect to.")?
        .collect::<Vec<_>>();
    Ok(std::net::SocketAddr::from((addr[0], opt.port)))
}

/// 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!("警告:config ファイル内にエラー -- {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<PathBuf>> {
    if let Some(n) = &opt.identity {
        std::fs::File::open(n).with_context(|| {
            format!(
                "Unable to access the secret key file specified by the \"-i\" option. [{:?}]",
                &n
            )
        })?;
        Ok(Some(n.clone()))
    } else {
        let name = host_params.identity_file.as_ref().map(|p| p[0].clone());
        if let Some(ref n) = name {
            std::fs::File::open(n).with_context(|| {
                format!(
                    "Unnable to access the secret file specified by the ssh-config. [{:?}]",
                    &n
                )
            })?;
        }
        Ok(name)
    }
}

/// 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<PathBuf>) -> Result<()> {
    if user_auth_agent(sess, username).is_ok() {
        return Ok(());
    }
    if let Some(f) = identity {
        if user_auth_identity(sess, username, f).is_ok() {
            return Ok(());
        }
    }
    user_auth_password(sess, username)
        .map_err(|_| anyhow!("All user authentication methods failed."))
```

```rust
}

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

/// 公開キー認証
fn user_auth_identity(sess: &Session, username: &str, key_file: &Path) -> Result<(), String> {
    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()
                .map_err(|e| e.to_string())?;
            ret = sess.userauth_pubkey_file(username, None, key_file, Some(&password));
            if ret.is_ok() {
                return Ok(());
            }
            eprintln!("The passphrase is different.");
        }
    }
    debug!("認証失敗 (pubkey)->{:?}", ret.as_ref().unwrap_err());
    Err("公開キー認証失敗".to_string())
}

/// パスワード認証
fn user_auth_password(sess: &Session, username: &str) -> Result<(), String> {
    for _i in 0..3 {
        let password = Password::new()
            .with_prompt("Enter your login password.")
            .allow_empty_password(true)
            .interact()
            .map_err(|e| e.to_string())?;
        let ret = sess.userauth_password(username, &password);
```

```
182             if ret.is_ok() {
183                 return Ok(());
184             }
185             let ssh2::ErrorCode::Session(-18) = ret.as_ref().unwrap_err().code() else {
186                 break;
187             };
188             // ssh2エラーコード  -18 ->
189             // LIBSSH2_ERROR_AUTHENTICATION_FAILED: パスワードが違うんでしょう。
190             eprintln!("The password is different.");
191             debug!("認証失敗(password)->{:?}", ret.unwrap_err());
192         }
193         Err("パスワード認証失敗".to_string())
194     }
```

4　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      }
```

```rust
    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.")
}
```

5　ファイルシステムモジュール ssh_filesystem.rs

```rust
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::{
    collections::HashMap,
    ffi::OsStr,
    io::{Read, Seek, 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(session: Session, path: &Path) -> anyhow::Result<Self> {
        let mut inodes = Inodes::new();
        let top_path: PathBuf = path.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.list
        );
        Ok(Self {
            _session: session,
            sftp,
            inodes,
            fhandls: Fhandles::new(),
            _top_path: top_path,
```

```
        })
    }

    /// ssh2経由でファイルのステータスを取得する。
    /// 副作用: 取得に成功した場合、inodesにパスを登録する。
    fn getattr_from_ssh2(&mut self, path: &Path, uid: u32, gid: u32) -> Result<FileAttr, Error> {
        let attr_ssh2 = self.sftp.lstat(path)?;
        let kind = Self::conv_file_kind_ssh2fuser(&attr_ssh2.file_type())?;
        let ino = self.inodes.add(path);
        Ok(FileAttr {
            ino,
            size: attr_ssh2.size.unwrap_or(0),
            blocks: attr_ssh2.size.unwrap_or(0) / 512 + 1,
            atime: UNIX_EPOCH + Duration::from_secs(attr_ssh2.atime.unwrap_or(0)),
            mtime: UNIX_EPOCH + Duration::from_secs(attr_ssh2.mtime.unwrap_or(0)),
            ctime: UNIX_EPOCH + Duration::from_secs(attr_ssh2.mtime.unwrap_or(0)),
            crtime: UNIX_EPOCH,
            kind,
            perm: attr_ssh2.perm.unwrap_or(0o666) as u16,
            nlink: 1,
            uid,
            gid,
            rdev: 0,
            blksize: 512,
            flags: 0,
        })
    }

    fn conv_file_kind_ssh2fuser(filetype: &ssh2::FileType) -> Result<fuser::FileType, Error> {
        match filetype {
            ssh2::FileType::NamedPipe => Ok(fuser::FileType::NamedPipe),
            ssh2::FileType::CharDevice => Ok(fuser::FileType::CharDevice),
            ssh2::FileType::BlockDevice => Ok(fuser::FileType::BlockDevice),
            ssh2::FileType::Directory => Ok(fuser::FileType::Directory),
            ssh2::FileType::RegularFile => Ok(fuser::FileType::RegularFile),
            ssh2::FileType::Symlink => Ok(fuser::FileType::Symlink),
            ssh2::FileType::Socket => Ok(fuser::FileType::Socket),
            ssh2::FileType::Other(_) => Err(Error(libc::EBADF)),
        }
    }

    fn conv_timeornow2systemtime(time: &fuser::TimeOrNow) -> SystemTime {
        match time {
            fuser::TimeOrNow::SpecificTime(t) => *t,
            fuser::TimeOrNow::Now => SystemTime::now(),
        }
```

```
    }
}

impl Filesystem for Sshfs {
    fn lookup(&mut self, req: &Request, parent: u64, name: &OsStr, reply: ReplyEntry) {
        let Some(mut path) = self.inodes.get_path(parent) else {
            debug!("[lookup] 親ディレクトリの検索に失敗 inode={}", parent);
            reply.error(ENOENT);
            return;
        };
        path.push(Path::new(name));
        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);
            }
        };
    }

    fn getattr(&mut self, req: &Request, ino: u64, _fh: Option<u64>, reply: ReplyAttr) {
        let Some(path) = self.inodes.get_path(ino) else {
            debug!("[getattr] path 取得失敗: inode={}", ino);
            reply.error(ENOENT);
            return;
        };
        match self.getattr_from_ssh2(&path, req.uid(), req.gid()) {
            Ok(attr) => {
                //debug!("[getattr]retrun attr: {:?}", &attr);
                reply.attr(&Duration::from_secs(1), &attr);
            }
            Err(e) => {
                warn!("[getattr] getattr_from_ssh2 エラー: {:?}", &e);
                reply.error(e.0)
            }
        };
    }

    fn readdir(
        &mut self,
        _req: &Request,
        ino: u64,
        _fh: u64,
        offset: i64,
        mut reply: ReplyDirectory,
    ) {
        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) => {
```

```
182                 warn!("[readdir]ssh2::readdir 内でエラー発生-- {:?}", e);
183                 reply.error(Error::from(e).0);
184             }
185         };
186     }
187 
188     fn readlink(&mut self, _req: &Request<'_>, ino: u64, reply: ReplyData) {
189         let Some(path) = self.inodes.get_path(ino) else {
190             error!("[readlink] 親ディレクトリの検索に失敗 {ino}");
191             reply.error(libc::ENOENT);
192             return;
193         };
194         match self.sftp.readlink(&path) {
195             Ok(p) => {
196                 //debug!("[readlink] ret_path => {:?}", &p);
197                 reply.data(p.as_os_str().to_str().unwrap().as_bytes());
198             }
199             Err(e) => {
200                 //debug!("[readlink] ssh2::readlink error => {e:?}");
201                 reply.error(Error::from(e).0);
202             }
203         }
204     }
205 
206     fn open(&mut self, _req: &Request<'_>, ino: u64, flags: i32, reply: fuser::ReplyOpen) {
207         let Some(file_name) = self.inodes.get_path(ino) else {
208             reply.error(libc::ENOENT);
209             return;
210         };
211 
212         let mut flags_ssh2 = OpenFlags::empty();
213         if flags & libc::O_WRONLY != 0 {
214             flags_ssh2.insert(OpenFlags::WRITE);
215         } else if flags & libc::O_RDWR != 0 {
216             flags_ssh2.insert(OpenFlags::READ);
217             flags_ssh2.insert(OpenFlags::WRITE);
218         } else {
219             flags_ssh2.insert(OpenFlags::READ);
220         }
221         if flags & libc::O_APPEND != 0 {
222             flags_ssh2.insert(OpenFlags::APPEND);
223         }
224         if flags & libc::O_CREAT != 0 {
225             flags_ssh2.insert(OpenFlags::CREATE);
226         }
227         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,
    ) {
        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,
    ) {
        let Some(file) = self.fhandls.get_file(fh) else {
            reply.error(libc::EINVAL);
            return;
        };

        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,
    ) {
        let Some(file) = self.fhandls.get_file(fh) else {
            reply.error(libc::EINVAL);
            return;
        };

        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 mknod(
        &mut self,
        req: &Request<'_>,
        parent: u64,
        name: &OsStr,
        mode: u32,
        umask: u32,
        _rdev: u32,
        reply: ReplyEntry,
    ) {
        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);
    }

    fn unlink(&mut self, _req: &Request<'_>, parent: u64, name: &OsStr, reply: fuser::ReplyEmpty) {
        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,
    ) {
        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),
        }
    }

    fn rmdir(&mut self, _req: &Request<'_>, parent: u64, name: &OsStr, reply: fuser::ReplyEmpty) {
        let Some(mut path) = self.inodes.get_path(parent) else {
            reply.error(libc::ENOENT);
            return;
        };
        path.push(name);
        match self.sftp.rmdir(&path) {
            Ok(_) => {
                self.inodes.del_inode_with_path(&path);
                reply.ok()
            }
            Err(e) => {
                if e.code() == ErrorCode::Session(-31) {
                    // ssh2ライブラリの返すエラーが妙。置換しておく。
                    reply.error(libc::ENOTEMPTY);
                } else {
                    reply.error(Error::from(e).0)
                }
            }
        }
    }

    fn symlink(
        &mut self,
        req: &Request<'_>,
        parent: u64,
        name: &OsStr,
        link: &Path,
        reply: ReplyEntry,
    ) {
        let Some(mut target) = self.inodes.get_path(parent) else {
```

```
458             reply.error(libc::ENOENT);
459             return;
460         };
461         target.push(name);
462         match self.sftp.symlink(link, &target) {
463             Ok(_) => match self.getattr_from_ssh2(&target, req.uid(), req.gid()) {
464                 Ok(attr) => reply.entry(&Duration::from_secs(1), &attr, 0),
465                 Err(e) => reply.error(e.0),
466             },
467             Err(e) => reply.error(Error::from(e).0),
468         }
469     }
470 
471     fn setattr(
472         &mut self,
473         req: &Request<'_>,
474         ino: u64,
475         mode: Option<u32>,
476         _uid: Option<u32>,
477         _gid: Option<u32>,
478         size: Option<u64>,
479         atime: Option<fuser::TimeOrNow>,
480         mtime: Option<fuser::TimeOrNow>,
481         _ctime: Option<std::time::SystemTime>,
482         _fh: Option<u64>,
483         _crtime: Option<std::time::SystemTime>,
484         _chgtime: Option<std::time::SystemTime>,
485         _bkuptime: Option<std::time::SystemTime>,
486         _flags: Option<u32>,
487         reply: ReplyAttr,
488     ) {
489         let stat = ssh2::FileStat {
490             size,
491             uid: None,
492             gid: None,
493             perm: mode,
494             atime: atime.map(|t| {
495                 Self::conv_timeornow2systemtime(&t)
496                     .duration_since(UNIX_EPOCH)
497                     .unwrap()
498                     .as_secs()
499             }),
500             mtime: mtime.map(|t| {
501                 Self::conv_timeornow2systemtime(&t)
502                     .duration_since(UNIX_EPOCH)
503                     .unwrap()
```

```
                    .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,
    ) {
        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。
            if let Ok(stat) = self.sftp.lstat(&new_path) {
                if stat.is_dir() {
                    if let Err(e) = self.sftp.rmdir(&new_path) {
                        reply.error(Error::from(e).0);
                        return;
                    }
                } else if let Err(e) = self.sftp.unlink(&new_path) {
                    reply.error(Error::from(e).0);
                    return;
                }
                self.inodes.del_inode_with_path(&new_path);
            }
        }

        match self.sftp.rename(&old_path, &new_path, Some(rename_flag)) {
            Ok(_) => {
                self.inodes.rename(&old_path, &new_path);
                reply.ok();
            }
            Err(e) => reply.error(Error::from(e).0),
        }
    }
}

#[derive(Debug, Default)]
struct Inodes {
    list: HashMap<u64, PathBuf>,
    max_inode: u64,
}

impl Inodes {
    /// Inode を生成する
    fn new() -> Self {
        Self {
            list: std::collections::HashMap::new(),
            max_inode: 0,
        }
    }

    /// 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。
    fn add(&mut self, path: &Path) -> u64 {
        match self.get_inode(path) {
            Some(i) => i,
            None => {
```

```
                self.max_inode += 1;
                self.list.insert(self.max_inode, path.into());
                self.max_inode
            }
        }
    }

    /// pathから inodeを取得する
    fn get_inode(&self, path: &Path) -> Option<u64> {
        self.list.iter().find(|(_, p)| path == *p).map(|(i, _)| *i)
    }

    /// inodeから pathを取得する
    fn get_path(&self, inode: u64) -> Option<PathBuf> {
        self.list.get(&inode).map(|p| (*p).clone())
    }

    /// inodesから、inodeの登録を削除する
    fn del_inode(&mut self, inode: u64) -> Option<u64> {
        self.list.remove(&inode).map(|_| inode)
    }

    /// inodesから、pathの名前の登録を削除する
    fn del_inode_with_path(&mut self, path: &Path) -> Option<u64> {
        self.get_inode(path).map(|ino| self.del_inode(ino).unwrap())
    }

    /// 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。
    fn rename(&mut self, old_path: &Path, new_path: &Path) {
        let Some(ino) = self.get_inode(old_path) else {
            return;
        };
        if let Some(val) = self.list.get_mut(&ino) {
            *val = new_path.into();
        }
    }
}

struct Fhandles {
    list: HashMap<u64, ssh2::File>,
    next_handle: u64,
}

impl Fhandles {
    fn new() -> Self {
```

```
642             Self {
643                 list: HashMap::new(),
644                 next_handle: 0,
645             }
646         }
647 
648         fn add_file(&mut self, file: ssh2::File) -> u64 {
649             let handle = self.next_handle;
650             self.list.insert(handle, file);
651             self.next_handle += 1;
652             handle
653         }
654 
655         fn get_file(&mut self, fh: u64) -> Option<&mut ssh2::File> {
656             self.list.get_mut(&fh)
657         }
658 
659         fn del_file(&mut self, fh: u64) {
660             self.list.remove(&fh); // 戻り値は捨てる。この時点でファイルはクローズ。
661                                    // ハンドルの再利用のため、次回ハンドルを調整
662             match self.list.keys().max() {
663                 Some(&i) => self.next_handle = i + 1,
664                 None => self.next_handle = 0,
665             }
666         }
667 }
668 
669 #[derive(Debug, Clone, Copy)]
670 struct Error(i32);
671 
672 impl From<ssh2::Error> for Error {
673     fn from(value: ssh2::Error) -> Self {
674         let eno = match value.code() {
675             ssh2::ErrorCode::Session(_) => libc::ENXIO,
676             ssh2::ErrorCode::SFTP(i) => match i {
677                 // libssh2の libssh2_sftp.hにて定義されている。
678                 2 => libc::ENOENT,          // NO_SUCH_FILE
679                 3 => libc::EACCES,          // permission_denied
680                 4 => libc::EIO,             // failure
681                 5 => libc::ENODEV,          // bad message
682                 6 => libc::ENXIO,           // no connection
683                 7 => libc::ENETDOWN,        // connection lost
684                 8 => libc::ENODEV,          // unsported
685                 9 => libc::EBADF,           // invalid handle
686                 10 => libc::ENOENT,         //no such path
687                 11 => libc::EEXIST,         // file already exists
```

```
                12 => libc::EACCES,       // write protected
                13 => libc::ENXIO,        // no media
                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
                _ => 0,
            },
        };
        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,
            _ => 0,
        };
        Self(eno)
    }
}
```

```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(Path::new("")), 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() {
```

```
780         let mut inodes = make_inodes();
781         let old = Path::new("test2");
782         let new = Path::new("new_test");
783         let ino = inodes.get_inode(old).unwrap();
784         inodes.rename(old, new);
785         assert_eq!(inodes.get_path(ino), Some(new.into()));
786 
787         let mut inodes = make_inodes();
788         let inodes2 = make_inodes();
789         inodes.rename(Path::new("nai"), Path::new("kawattenai"));
790         assert_eq!(inodes.list, inodes2.list);
791     }
792 }
```