

# sshmount ソースリスト

美都

2023 年 2 月 26 日

## 目次

1	メインモジュール main.rs .....	2
2	コマンドラインオプションの定義 cmdline_opt.rs .....	3
3	ssh2 ログイン処理モジュール ssh_connect.rs .....	8
4	FUSE 接続オプション生成モジュール fuse_util.rs .....	12
5	ファイルシステムモジュール ssh_filesystem.rs .....	15

## 1 メインモジュール main.rs

```
1  mod cmdline_opt;
2  mod fuse_util;
3  mod ssh_connect;
4  mod ssh_filesystem;
5
6  use anyhow::{anyhow, Context, Result};
7  use clap::Parser;
8  use cmdline_opt::Opt;
9  use fuse_util::{make_mount_option, make_remote_path};
10 use ssh_connect::make_ssh_session;
11 //use log::debug;
12
13 fn main() -> Result<()> {
14     env_logger::init();
15     let opt = Opt::parse();
16
17     let ssh = make_ssh_session(&opt).map_err(|e| anyhow!(e))?;
18
19     let path = make_remote_path(&opt, &ssh).context("Failed to generate remote path.")?;
20     let options = make_mount_option(&opt);
21
22     // ファイルシステムへのマウント実行
23     let fs = ssh_filesystem::Sshfs::new(ssh, &path);
24     fuser::mount2(fs, opt.mount_point, &options).context("Failed to mount FUSE.")?;
25     Ok(())
26 }
```

## 2 コマンドラインオプションの定義 cmdline\_opt.rs

```
1 use clap::Parser;
2 use std::path::PathBuf;
3
4 /// コマンドラインオプション
5 #[derive(Parser)]
6 #[command(author, version, about)]
7 pub struct Opt {
8     /// Distination [user@]host:[path]
9     pub remote: RemoteName,
10    /// Path to mount
11    #[arg(value_parser = exist_dir)]
12    pub mount_point: String,
13    /// Path to config file
14    #[arg(short = 'F', long)]
15    pub config_file: Option<PathBuf>,
16    /// Login name
17    #[arg(short, long)]
18    pub login_name: Option<String>,
19    /// File name of secret key file
20    #[arg(short, long)]
21    pub identity: Option<PathBuf>,
22    /// Port no
23    #[arg(short, long, default_value_t = 22)]
24    pub port: u16,
25    /// Read only
26    #[arg(short, long)]
27    pub readonly: bool,
28    /// Not executable
29    #[arg(long)]
30    pub no_exec: bool,
31    /// Do not change access date and time(ctime)
32    #[arg(long)]
33    pub no_atime: bool,
34 }
35
36 /// 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。
37 fn exist_dir(s: &str) -> Result<String, String> {
38     match std::fs::read_dir(s) {
39         Ok(mut dir) => match dir.next() {
40             None => Ok(s.to_string()),
41             Some(_) => Err("マウント先ディレクトリが空ではありません".to_string()),
42         },
43     }
44 }
```

```

43     Err(e) => match e.kind() {
44         std::io::ErrorKind::NotFound => {
45             Err("マウント先ディレクトリが存在しません.".to_string())
46         }
47         std::io::ErrorKind::NotConnected => {
48             Err("マウント先ディレクトリのネットワークが切断されています。(umount を忘れていませ
49                 ↳ んか?).to_string())
50         }
51         _ => Err(format!("計り知れないエラーです。--{e:?}")),
52     },
53 }
54
55 /// コマンドラインの接続先ホスト情報
56 #[derive(Clone, Debug, PartialEq)]
57 pub struct RemoteName {
58     /// ユーザー名
59     pub user: Option<String>,
60     /// ホスト名 または IP アドレス
61     pub host: String,
62     /// 接続先パス
63     pub path: Option<std::path::PathBuf>,
64 }
65
66 impl std::fmt::Display for RemoteName {
67     fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
68         let s = format!("<{:?}><{:?}><{:?}>", &self.user, &self.host, &self.path);
69         s.fmt(f)
70     }
71 }
72
73 impl std::str::FromStr for RemoteName {
74     type Err = Error;
75     fn from_str(s: &str) -> Result<Self, Self::Err> {
76         let mut rest_str = s;
77         let user = match rest_str.split_once('@') {
78             Some((u, r)) => {
79                 rest_str = r;
80                 if !u.trim().is_empty() {
81                     Some(u.trim().to_string())
82                 } else {
83                     None
84                 }
85             }
86             None => None,

```

```

87     };
88     let (host, path) = match rest_str.split_once(':') {
89         Some((h, p)) => (
90             if !h.trim().is_empty() {
91                 h.trim().to_string()
92             } else {
93                 return Err(Error);
94             },
95             if !p.trim().is_empty() {
96                 Some(std::path::PathBuf::from(p.trim().to_string()))
97             } else {
98                 None
99             },
100         ),
101         None => return Err(Error),
102     };
103     Ok(Self { user, host, path })
104 }
105 }
106
107 #[derive(thiserror::Error, Debug)]
108 #[error("接続先ホストの形式は、\"[user@]host:[path]\"です。")]
109 pub struct Error;
110
111 #[cfg(test)]
112 mod test {
113     use super::*;
114     #[test]
115     fn verify_cli() {
116         use clap::CommandFactory;
117         Opt::command().debug_assert()
118     }
119
120     #[test]
121     fn test_from_str_remotename() {
122         use std::path::Path;
123         let s = "mito@reterminal.local:/home/mito";
124         let r: RemoteName = s.parse().unwrap();
125         let k = RemoteName {
126             user: Some("mito".to_string()),
127             host: "reterminal.local".to_string(),
128             path: Some(Path::new("/home/mito").into()),
129         };
130         assert_eq!(r, k);
131     }

```

```

132 let s = "mito@reterminal.local:/home/mito/";
133 let r: RemoteName = s.parse().unwrap();
134 let k = RemoteName {
135     user: Some("mito".to_string()),
136     host: "reterminal.local".to_string(),
137     path: Some(Path::new("/home/mito").into()),
138 };
139 assert_eq!(r, k);
140
141 let s = "reterminal.local: ";
142 let r: RemoteName = s.parse().unwrap();
143 let k = RemoteName {
144     user: None,
145     host: "reterminal.local".to_string(),
146     path: None,
147 };
148 assert_eq!(r, k);
149
150 let s = " mito @reterminal.local: ";
151 let r: RemoteName = s.parse().unwrap();
152 let k = RemoteName {
153     user: Some("mito".to_string()),
154     host: "reterminal.local".to_string(),
155     path: None,
156 };
157 assert_eq!(r, k);
158
159 let s = "reterminal.local";
160 let r: Result<RemoteName, String> = s.parse();
161 assert_eq!(
162     r,
163     Err("接続先ホストの形式は、\"[user@]host:[path]\"です。".to_string())
164 );
165
166 let s = "mito@reterminal.local";
167 let r: Result<RemoteName, String> = s.parse();
168 assert_eq!(
169     r,
170     Err("接続先ホストの形式は、\"[user@]host:[path]\"です。".to_string())
171 );
172
173 let s = " mito @: ";
174 let r: Result<RemoteName, String> = s.parse();
175 assert_eq!(
176     r,

```

```
177         Err("接続先ホストの形式は、\"[user@]host:[path]\"です。".to_string())
178     );
179 }
180 }
```

### 3 ssh2 ログイン処理モジュール ssh\_connect.rs

```
1  ///! ssh 接続関連関数モジュール
2
3  use crate::cmdline_opt::Opt;
4  use dialoguer::Password;
5  use dns_lookup::lookup_host;
6  use log::debug;
7  use ssh2::Session;
8  use ssh2_config::{HostParams, SshConfig};
9  use std::{
10     fs::File,
11     io::BufReader,
12     net::TcpStream,
13     path::{Path, PathBuf},
14     str,
15 };
16
17  /// セッションを生成する。
18  pub fn make_ssh_session(opt: &Opt) -> Result<Session, String> {
19     let host_params = get_ssh_config(&opt.config_file).query(&opt.remote.host);
20     let address = get_address(opt, &host_params)?;
21     let username = get_username(opt, &host_params)?;
22     debug!(
23         "[main] 接続先情報-> ユーザー:\"{}\", ip address:{:?}\",
24         &username, &address
25     );
26     let identity_file = get_identity_file(opt, &host_params);
27
28     let ssh = connect_ssh(address)?;
29     userauth(&ssh, &username, &identity_file)?;
30     Ok(ssh)
31 }
32
33  /// ホストの ip アドレス解決
34  fn get_address(opt: &Opt, host_params: &HostParams) -> Result<std::net::SocketAddr, String> {
35     let dns = host_params.host_name.as_deref().unwrap_or(&opt.remote.host);
36     let addr = lookup_host(dns).map_err(|_| format!("接続先ホストが見つかりません。({dns})"))?;
37     Ok(std::net::SocketAddr::from((addr[0], opt.port)))
38 }
39
40  /// ssh-config の取得と解析
41  /// ファイル名が指定されていない場合は "~/.ssh/config" を使用
42  /// 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。
```



```

43 fn get_ssh_config(file_opt: &Option<PathBuf>) -> SshConfig {
44     get_config_file(file_opt)
45         .map(BufReader::new)
46         .map_or(SshConfig::default(), |mut f| {
47             SshConfig::default().parse(&mut f).unwrap_or_else(|e| {
48                 eprintln!("警告:config ファイル内にエラー -- {e}");
49                 SshConfig::default()
50             })
51         })
52 }
53
54 /// ssh_config ファイルがあれば、オープンする。
55 /// ファイル名の指定がなければ、$Home/.ssh/config を想定する。
56 fn get_config_file(file_name: &Option<PathBuf>) -> Option<std::fs::File> {
57     let file_name = file_name.clone().or_else(|| {
58         home::home_dir().map(|p| {
59             let mut p = p;
60             p.push(".ssh/config");
61             p
62         })
63     });
64
65     file_name.and_then(|p| File::open(p).ok())
66 }
67
68 /// ログイン名を確定し、取得する。
69 /// 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
70 fn get_username(opt: &Opt, params: &HostParams) -> Result<String, String> {
71     if let Some(n) = &opt.login_name {
72         Ok(n.clone())
73     } else if let Some(n) = &opt.remote.user {
74         Ok(n.clone())
75     } else if let Some(n) = &params.user {
76         Ok(n.clone())
77     } else if let Some(n) = users::get_current_username() {
78         n.to_str()
79             .ok_or_else(|| format!("ログインユーザ名不正。({n:?})"))
80             .map(|s| s.to_string())
81     } else {
82         Err("ユーザー名が取得できませんでした。")?
83     }
84 }
85
86 /// 秘密キーファイルのパスを取得する
87 fn get_identity_file(opt: &Opt, host_params: &HostParams) -> Option<PathBuf> {

```

```

88     if opt.identity.is_some() {
89         opt.identity.clone()
90     } else {
91         host_params.identity_file.as_ref().map(|p| p[0].clone())
92     }
93 }
94
95 /// リモートの ssh に接続し、セッションを生成する。
96 fn connect_ssh<A: std::net::ToSocketAddrs>(address: A) -> Result<Session, String> {
97     let tcp = TcpStream::connect(address)
98         .map_err(|e| format!("TCP/IP の接続に失敗しました -- {:?}", &e))?;
99     let mut ssh = Session::new().map_err(|e| format!("ssh の接続に失敗しました。 -- {:?}", &e))?;
100     ssh.set_tcp_stream(tcp);
101     ssh.handshake()
102         .map_err(|e| format!("ssh の接続に失敗しました。(2) -- {:?}", &e))?;
103     Ok(ssh)
104 }
105
106 /// ssh 認証を実施する。
107 fn userauth(sess: &Session, username: &str, identity: &Option<PathBuf>) -> Result<(), String> {
108     if user_auth_agent(sess, username).is_ok() {
109         return Ok(());
110     }
111     if let Some(f) = identity {
112         if user_auth_identity(sess, username, f).is_ok() {
113             return Ok(());
114         }
115     }
116     user_auth_password(sess, username)
117 }
118
119 /// agent 認証
120 fn user_auth_agent(sess: &Session, username: &str) -> Result<(), ssh2::Error> {
121     let ret = sess.userauth_agent(username);
122     if ret.is_err() {
123         debug!("認証失敗 (agent)->{:?}", ret.as_ref().unwrap_err());
124     };
125     ret
126 }
127
128 /// 公開キー認証
129 fn user_auth_identity(sess: &Session, username: &str, key_file: &Path) -> Result<(), String> {
130     let mut ret = sess.userauth_pubkey_file(username, None, key_file, None);
131     if ret.is_ok() {
132         return Ok(());

```

```

133     };
134     if let ssh2::ErrorCode::Session(-16) = ret.as_ref().unwrap_err().code() {
135         // error_code -16 ->
136         // LIBSSH2_ERROR_FILE:PUBLIC_KEYの取得失敗。多分、秘密キーのパスフレーズ
137         for _i in 0..3 {
138             let password = Password::new()
139                 .with_prompt("秘密キーのパスフレーズを入力してください。")
140                 .allow_empty_password(true)
141                 .interact()
142                 .map_err(|e| e.to_string())?;
143             ret = sess.userauth_pubkey_file(username, None, key_file, Some(&password));
144             if ret.is_ok() {
145                 return Ok(());
146             }
147             eprintln!("パスフレーズが違います。");
148         }
149     }
150     debug!("認証失敗 (pubkey)->{:?}", ret.as_ref().unwrap_err());
151     Err("公開キー認証失敗".to_string())
152 }
153
154 /// パスワード認証
155 fn user_auth_password(sess: &Session, username: &str) -> Result<(), String> {
156     for _i in 0..3 {
157         let password = Password::new()
158             .with_prompt("ログインパスワードを入力してください。")
159             .allow_empty_password(true)
160             .interact()
161             .map_err(|e| e.to_string())?;
162         let ret = sess.userauth_password(username, &password);
163         if ret.is_ok() {
164             return Ok(());
165         }
166         let ssh2::ErrorCode::Session(-18) = ret.as_ref().unwrap_err().code() else { break; };
167         // ssh2 エラーコード -18 ->
168         // LIBSSH2_ERROR_AUTHENTICATION_FAILED: パスワードが違うんでしょう。
169         eprintln!("パスワードが違います。");
170         debug!("認証失敗 (password)->{:?}", ret.unwrap_err());
171     }
172     Err("パスワード認証失敗".to_string())
173 }

```

## 4 FUSE 接続オプション生成モジュール fuse\_util.rs

```
1  /// FUSE パラメータ関係 ユーティリティ
2
3  use crate::cmdline_opt::Opt;
4  use anyhow::{ensure, Context, Result};
5  use ssh2::Session;
6  use std::{io::Read, path::PathBuf, str};
7
8  /// リモート接続先の path の生成
9  pub fn make_remote_path(opt: &Opt, session: &Session) -> Result<PathBuf> {
10     // パスの生成
11     const MSG_ERRORHOME: &str = "Fail to generate path name.";
12     let mut path = match opt.remote.path {
13         Some(ref p) => {
14             if p.is_absolute() {
15                 p.clone()
16             } else {
17                 let mut h = get_home_on_remote(session).context(MSG_ERRORHOME)?;
18                 h.push(p);
19                 h
20             }
21         }
22         None => get_home_on_remote(session).context(MSG_ERRORHOME)?,
23     };
24     // 生成したパスが実在するかを確認する
25     let sftp = session
26         .sftp()
27         .context("Connection to SFTP failed when checking for existence of a path.")?;
28     let file_stat = sftp
29         .stat(&path)
30         .with_context(|| format!("Cannot find path to connect to. path={:?}", &path))?;
31     ensure!(
32         file_stat.is_dir(),
33         "The path to connect to is not a directory."
34     );
35     // 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
36     let file_stat = sftp
37         .lstat(&path)
38         .context("Failed to obtain the attributes of the destination directory.")?;
39     if file_stat.file_type().is_symlink() {
40         path = sftp
41             .readlink(&path)
42             .context("Failed to resolve symbolic link to connect to.")?;
```

```

43     if !path.is_absolute() {
44         let tmp = path;
45         path = get_home_on_remote(session)
46             .context("Failed to complete the symbolic link to connect to.")?;
47         path.push(tmp);
48     };
49 };
50
51 Ok(path)
52 }
53
54 /// FUSEの接続時オプションを生成する
55 pub fn make_mount_option(cmd_opt: &Opt) -> Vec<fuser::MountOption> {
56     use fuser::MountOption;
57
58     let mut options = vec![MountOption::FSName("sshfs".to_string())];
59     options.push(MountOption::NoDev);
60     options.push(MountOption::DirSync);
61     options.push(MountOption::Sync);
62     match cmd_opt.readonly {
63         true => options.push(MountOption::RD),
64         false => options.push(MountOption::RW),
65     }
66     match cmd_opt.no_exec {
67         true => options.push(MountOption::NoExec),
68         false => options.push(MountOption::Exec),
69     }
70     match cmd_opt.no_atime {
71         true => options.push(MountOption::NoAtime),
72         false => options.push(MountOption::Atime),
73     }
74     options
75 }
76
77 /// ssh接続先のカレントディレクトリを取得する
78 fn get_home_on_remote(session: &Session) -> Result<PathBuf> {
79     let mut channel = session
80         .channel_session()
81         .context("Fail to build ssh channel.")?;
82     channel
83         .exec("pwd")
84         .context("Fail to execute \"pwd\" command.")?;
85     let mut buf = Vec::<u8>::new();
86     channel
87         .read_to_end(&mut buf)

```

```
88         .context("Fail to get response for \"pwd\" command.")?;
89     channel.close().context("Fail to close ssh channel.")?;
90     str::from_utf8(&buf)
91         .context("The pwd result contains non-utf8 characters.")?
92         .trim()
93         .parse::<PathBuf>()
94         .context("Fail to build path name.")
95     }
```

## 5 ファイルシステムモジュール ssh\_filesystem.rs

```
1  /// FUSE ファイルシステム実装
2  use fuser::{FileAttr, Filesystem, ReplyAttr, ReplyData, ReplyDirectory, ReplyEntry, Request};
3  use libc::ENOENT;
4  use log::{debug, error, warn};
5  use ssh2::{ErrorCode, OpenFlags, OpenType, Session, Sftp};
6  use std::{
7      collections::HashMap,
8      ffi::OsStr,
9      io::{Read, Seek, Write},
10     path::{Path, PathBuf},
11     time::{Duration, SystemTime, UNIX_EPOCH},
12 };
13
14 pub struct Sshfs {
15     _session: Session,
16     sftp: Sftp,
17     inodes: Inodes,
18     fhandles: Fhandles,
19     _top_path: PathBuf,
20 }
21
22 impl Sshfs {
23     pub fn new(session: Session, path: &Path) -> Self {
24         let mut inodes = Inodes::new();
25         let top_path: PathBuf = path.into();
26         inodes.add(&top_path);
27         let sftp = session.sftp().unwrap();
28         debug!(
29             "[Sshfs::new] connect path: <{:?}>, inodes=<{:?}>",
30             &top_path, &inodes.list
31         );
32         Self {
33             _session: session,
34             sftp,
35             inodes,
36             fhandles: Fhandles::new(),
37             _top_path: top_path,
38         }
39     }
40
41     /// ssh2経由でファイルのステータスを取得する。
42     /// 副作用: 取得に成功した場合、inodes にパスを登録する。
```

```

43 fn getattr_from_ssh2(&mut self, path: &Path, uid: u32, gid: u32) -> Result<FileAttr, Error> {
44     let attr_ssh2 = self.sftp.lstat(path)?;
45     let kind = Self::conv_file_kind_ssh2fuser(&attr_ssh2.file_type())?;
46     let ino = self.inodes.add(path);
47     Ok(FileAttr {
48         ino,
49         size: attr_ssh2.size.unwrap_or(0),
50         blocks: attr_ssh2.size.unwrap_or(0) / 512 + 1,
51         atime: UNIX_EPOCH + Duration::from_secs(attr_ssh2.atime.unwrap_or(0)),
52         mtime: UNIX_EPOCH + Duration::from_secs(attr_ssh2.mtime.unwrap_or(0)),
53         ctime: UNIX_EPOCH + Duration::from_secs(attr_ssh2.mtime.unwrap_or(0)),
54         crtime: UNIX_EPOCH,
55         kind,
56         perm: attr_ssh2.perm.unwrap_or(0o666) as u16,
57         nlink: 1,
58         uid,
59         gid,
60         rdev: 0,
61         blksize: 512,
62         flags: 0,
63     })
64 }
65
66 fn conv_file_kind_ssh2fuser(filetype: &ssh2::FileType) -> Result<fuser::FileType, Error> {
67     match filetype {
68         ssh2::FileType::NamedPipe => Ok(fuser::FileType::NamedPipe),
69         ssh2::FileType::CharDevice => Ok(fuser::FileType::CharDevice),
70         ssh2::FileType::BlockDevice => Ok(fuser::FileType::BlockDevice),
71         ssh2::FileType::Directory => Ok(fuser::FileType::Directory),
72         ssh2::FileType::RegularFile => Ok(fuser::FileType::RegularFile),
73         ssh2::FileType::Symlink => Ok(fuser::FileType::Symlink),
74         ssh2::FileType::Socket => Ok(fuser::FileType::Socket),
75         ssh2::FileType::Other(_) => Err(Error(libc::EBADF)),
76     }
77 }
78
79 fn conv_timeornow2systemtime(time: &fuser::TimeOrNow) -> SystemTime {
80     match time {
81         fuser::TimeOrNow::SpecificTime(t) => *t,
82         fuser::TimeOrNow::Now => SystemTime::now(),
83     }
84 }
85 }
86
87 impl Filesystem for Sshfs {

```



```

88 fn lookup(&mut self, req: &Request, parent: u64, name: &OsStr, reply: ReplyEntry) {
89     let Some(mut path) = self.inodes.get_path(parent) else {
90         debug!("[lookup] 親ディレクトリの検索に失敗 inode={}", parent);
91         reply.error(ENOENT);
92         return;
93     };
94     path.push(Path::new(name));
95     match self.getattr_from_ssh2(&path, req.uid(), req.gid()) {
96         Ok(attr) => reply.entry(&Duration::from_secs(1), &attr, 0),
97         Err(e) => {
98             reply.error(e.0);
99         }
100     };
101 }
102
103 fn getattr(&mut self, req: &Request, ino: u64, reply: ReplyAttr) {
104     let Some(path) = self.inodes.get_path(ino) else {
105         debug!("[getattr] path 取得失敗: inode={}", ino);
106         reply.error(ENOENT);
107         return;
108     };
109     match self.getattr_from_ssh2(&path, req.uid(), req.gid()) {
110         Ok(attr) => {
111             //debug!("[getattr] retrun attr: {:?}", &attr);
112             reply.attr(&Duration::from_secs(1), &attr);
113         }
114         Err(e) => {
115             warn!("[getattr] getattr_from_ssh2 エラー: {:?}", &e);
116             reply.error(e.0)
117         }
118     };
119 }
120
121 fn readdir(
122     &mut self,
123     _req: &Request,
124     ino: u64,
125     _fh: u64,
126     offset: i64,
127     mut reply: ReplyDirectory,
128 ) {
129     let Some(path) = self.inodes.get_path(ino) else {
130         reply.error(libc::ENOENT);
131         return;
132     };

```

```

133 match self.sftp.readdir(&path) {
134     Ok(mut dir) => {
135         let cur_file_attr = ssh2::FileStat {
136             size: None,
137             uid: None,
138             gid: None,
139             perm: Some(libc::S_IFDIR),
140             atime: None,
141             mtime: None,
142         }; // "." ".."の解決用。 attr ディレクトリであることのみを示す。
143         dir.insert(0, (Path::new("..").into(), cur_file_attr.clone()));
144         dir.insert(0, (Path::new(".").into(), cur_file_attr));
145         let mut i = offset + 1;
146         for f in dir.iter().skip(offset as usize) {
147             let ino = if f.0 == Path::new("..") || f.0 == Path::new(".") {
148                 1
149             } else {
150                 self.inodes.add(&f.0)
151             };
152             let name = match f.0.file_name() {
153                 Some(n) => n,
154                 None => f.0.as_os_str(),
155             };
156             let filetype = &f.1.file_type();
157             let filetype = match Self::conv_file_kind_ssh2fuser(filetype) {
158                 Ok(t) => t,
159                 Err(e) => {
160                     warn!(
161                         "[readdir] ファイルタイプ解析失敗: inode={}, name={:?}",
162                         ino, name
163                     );
164                     reply.error(e.0);
165                     return;
166                 }
167             };
168             if reply.add(ino, i, filetype, name) {
169                 break;
170             }
171             i += 1;
172         }
173         reply.ok();
174     }
175     Err(e) => {
176         warn!("[readdir] ssh2::readdir 内でエラー発生-- {:?}", e);
177         reply.error(Error::from(e).0);

```

```

178         }
179     };
180 }
181
182 fn readlink(&mut self, _req: &Request<'_>, ino: u64, reply: ReplyData) {
183     let Some(path) = self.inodes.get_path(ino) else {
184         error!("[readlink] 親ディレクトリの検索に失敗 {ino}");
185         reply.error(libc::ENOENT);
186         return;
187     };
188     match self.sftp.readlink(&path) {
189         Ok(p) => {
190             //debug!("[readlink] ret_path => {:?}", &p);
191             reply.data(p.as_os_str().to_str().unwrap().as_bytes());
192         }
193         Err(e) => {
194             //debug!("[readlink] ssh2::readlink error => {e:?}");
195             reply.error(Error::from(e).0);
196         }
197     }
198 }
199
200 fn open(&mut self, _req: &Request<'_>, ino: u64, flags: i32, reply: fuser::ReplyOpen) {
201     let Some(file_name) = self.inodes.get_path(ino) else {
202         reply.error(libc::ENOENT);
203         return;
204     };
205
206     let mut flags_ssh2 = OpenFlags::empty();
207     if flags & libc::O_WRONLY != 0 {
208         flags_ssh2.insert(OpenFlags::WRITE);
209     } else if flags & libc::O_RDWR != 0 {
210         flags_ssh2.insert(OpenFlags::READ);
211         flags_ssh2.insert(OpenFlags::WRITE);
212     } else {
213         flags_ssh2.insert(OpenFlags::READ);
214     }
215     if flags & libc::O_APPEND != 0 {
216         flags_ssh2.insert(OpenFlags::APPEND);
217     }
218     if flags & libc::O_CREAT != 0 {
219         flags_ssh2.insert(OpenFlags::CREATE);
220     }
221     if flags & libc::O_TRUNC != 0 {
222         flags_ssh2.insert(OpenFlags::TRUNCATE);

```

```

223     }
224     if flags & libc::O_EXCL != 0 {
225         flags_ssh2.insert(OpenFlags::EXCLUSIVE);
226     }
227
228     debug!(
229         "[open] openflag = {:?}, bit = {:x}",
230         &flags_ssh2,
231         flags_ssh2.bits()
232     );
233     match self
234         .sftp
235         .open_mode(&file_name, flags_ssh2, 0o777, ssh2::OpenType::File)
236     {
237         Ok(file) => {
238             let fh = self.fhandls.add_file(file);
239             reply.opened(fh, flags as u32);
240         }
241         Err(e) => reply.error(Error::from(e).0),
242     }
243 }
244
245 fn release(
246     &mut self,
247     _req: &Request<'_>,
248     _ino: u64,
249     fh: u64,
250     _flags: i32,
251     _lock_owner: Option<u64>,
252     _flush: bool,
253     reply: fuser::ReplyEmpty,
254 ) {
255     self.fhandls.del_file(fh);
256     reply.ok();
257 }
258
259 fn read(
260     &mut self,
261     _req: &Request,
262     _ino: u64,
263     fh: u64,
264     offset: i64,
265     size: u32,
266     _flags: i32,
267     _lock_owner: Option<u64>,

```

```

268     reply: ReplyData,
269 ) {
270     let Some(file) = self.fhandls.get_file(fh) else {
271         reply.error(libc::EINVAL);
272         return;
273     };
274
275     if let Err(e) = file.seek(std::io::SeekFrom::Start(offset as u64)) {
276         reply.error(Error::from(e).0);
277         return;
278     }
279     let mut buff = Vec::<u8>::new();
280     buff.resize(size as usize, 0u8);
281     let mut read_size: usize = 0;
282     while read_size < size as usize {
283         match file.read(&mut buff[read_size..]) {
284             Ok(s) => {
285                 if s == 0 {
286                     break;
287                 };
288                 read_size += s;
289             }
290             Err(e) => {
291                 reply.error(Error::from(e).0);
292                 return;
293             }
294         }
295     }
296     buff.resize(read_size, 0u8);
297     reply.data(&buff);
298 }
299
300 fn write(
301     &mut self,
302     _req: &Request<'_,>,
303     _ino: u64,
304     fh: u64,
305     offset: i64,
306     data: &[u8],
307     _write_flags: u32,
308     _flags: i32,
309     _lock_owner: Option<u64>,
310     reply: fuser::ReplyWrite,
311 ) {
312     let Some(file) = self.fhandls.get_file(fh) else {

```

```

313         reply.error(libc::EINVAL);
314         return ;
315     };
316
317     if let Err(e) = file.seek(std::io::SeekFrom::Start(offset as u64)) {
318         reply.error(Error::from(e).0);
319         return;
320     }
321     let mut buf = data;
322     while !buf.is_empty() {
323         let cnt = match file.write(buf) {
324             Ok(cnt) => cnt,
325             Err(e) => {
326                 reply.error(Error::from(e).0);
327                 return;
328             }
329         };
330         buf = &buf[cnt..];
331     }
332     reply.written(data.len() as u32);
333 }
334
335 fn mknod(
336     &mut self,
337     req: &Request<'_,>,
338     parent: u64,
339     name: &OsStr,
340     mode: u32,
341     umask: u32,
342     _rdev: u32,
343     reply: ReplyEntry,
344 ) {
345     if mode & libc::S_IFMT != libc::S_IFREG {
346         reply.error(libc::EPERM);
347         return;
348     }
349     let mode = mode & (!umask | libc::S_IFMT);
350     let Some(mut new_name) = self.inodes.get_path(parent) else {
351         reply.error(libc::ENOENT);
352         return;
353     };
354     new_name.push(name);
355     if let Err(e) =
356         self.sftp
357             .open_mode(&new_name, OpenFlags::CREATE, mode as i32, OpenType::File)

```

```

358     {
359         reply.error(Error::from(e).0);
360         return;
361     }
362     let new_attr = match self.getattr_from_ssh2(&new_name, req.uid(), req.gid()) {
363         Ok(a) => a,
364         Err(e) => {
365             reply.error(e.0);
366             return;
367         }
368     };
369     reply.entry(&Duration::from_secs(1), &new_attr, 0);
370 }
371
372 fn unlink(&mut self, _req: &Request<'_>, parent: u64, name: &OsStr, reply: fuser::ReplyEmpty) {
373     let Some(mut path) = self.inodes.get_path(parent) else {
374         reply.error(libc::ENOENT);
375         return;
376     };
377     path.push(name);
378     match self.sftp.unlink(&path) {
379         Ok(_) => {
380             self.inodes.del_inode_with_path(&path);
381             reply.ok();
382         }
383         Err(e) => reply.error(Error::from(e).0),
384     }
385 }
386
387 fn mkdir(
388     &mut self,
389     req: &Request<'_>,
390     parent: u64,
391     name: &OsStr,
392     mode: u32,
393     umask: u32,
394     reply: ReplyEntry,
395 ) {
396     let Some(mut path) = self.inodes.get_path(parent) else {
397         reply.error(libc::ENOENT);
398         return;
399     };
400     path.push(name);
401
402     let mode = (mode & (!umask) & 0o777) as i32;

```

```

403
404     match self.sftp.mkdir(&path, mode) {
405         Ok(_) => match self.getattr_from_ssh2(&path, req.uid(), req.gid()) {
406             Ok(attr) => reply.entry(&Duration::from_secs(1), &attr, 0),
407             Err(e) => reply.error(e.0),
408         },
409         Err(e) => reply.error(Error::from(e).0),
410     }
411 }
412
413 fn rmdir(&mut self, _req: &Request<'_>, parent: u64, name: &OsStr, reply: fuser::ReplyEmpty) {
414     let Some(mut path) = self.inodes.get_path(parent) else {
415         reply.error(libc::ENOENT);
416         return;
417     };
418     path.push(name);
419     match self.sftp.rmdir(&path) {
420         Ok(_) => {
421             self.inodes.del_inode_with_path(&path);
422             reply.ok()
423         }
424         Err(e) => {
425             if e.code() == ErrorCode::Session(-31) {
426                 // ssh2 ライブラリの返すエラーが妙。置換しておく。
427                 reply.error(libc::ENOTEMPTY);
428             } else {
429                 reply.error(Error::from(e).0)
430             }
431         }
432     }
433 }
434
435 fn symlink(
436     &mut self,
437     req: &Request<'_>,
438     parent: u64,
439     name: &OsStr,
440     link: &Path,
441     reply: ReplyEntry,
442 ) {
443     let Some(mut target) = self.inodes.get_path(parent) else {
444         reply.error(libc::ENOENT);
445         return;
446     };
447     target.push(name);

```



```

448     match self.sftp.symlink(link, &target) {
449         Ok(_) => match self.getattr_from_ssh2(&target, req.uid(), req.gid()) {
450             Ok(attr) => reply.entry(&Duration::from_secs(1), &attr, 0),
451             Err(e) => reply.error(e.0),
452         },
453         Err(e) => reply.error(Error::from(e).0),
454     }
455 }
456
457 fn setattr(
458     &mut self,
459     req: &Request<'_,>,
460     ino: u64,
461     mode: Option<u32>,
462     _uid: Option<u32>,
463     _gid: Option<u32>,
464     size: Option<u64>,
465     atime: Option<fuser::TimeOrNow>,
466     mtime: Option<fuser::TimeOrNow>,
467     _ctime: Option<std::time::SystemTime>,
468     _fh: Option<u64>,
469     _crttime: Option<std::time::SystemTime>,
470     _chmtime: Option<std::time::SystemTime>,
471     _bkuptime: Option<std::time::SystemTime>,
472     _flags: Option<u32>,
473     reply: ReplyAttr,
474 ) {
475     let stat = ssh2::FileStat {
476         size,
477         uid: None,
478         gid: None,
479         perm: mode,
480         atime: atime.map(|t| {
481             Self::conv_timeornow2systemtime(&t)
482                 .duration_since(UNIX_EPOCH)
483                 .unwrap()
484                 .as_secs()
485         }),
486         mtime: mtime.map(|t| {
487             Self::conv_timeornow2systemtime(&t)
488                 .duration_since(UNIX_EPOCH)
489                 .unwrap()
490                 .as_secs()
491         }),
492     };

```

```

493     let Some(filename) = self.inodes.get_path(ino) else {
494         reply.error(ENOENT);
495         return;
496     };
497     match self.sftp.setstat(&filename, stat) {
498         Ok(_) => {
499             let stat = self.getattr_from_ssh2(&filename, req.uid(), req.gid());
500             match stat {
501                 Ok(s) => reply.attr(&Duration::from_secs(1), &s),
502                 Err(e) => reply.error(e.0),
503             }
504         }
505         Err(e) => reply.error(Error::from(e).0),
506     }
507 }
508
509 fn rename(
510     &mut self,
511     _req: &Request<'_,>,
512     parent: u64,
513     name: &OsStr,
514     newparent: u64,
515     newname: &OsStr,
516     flags: u32,
517     reply: fuser::ReplyEmpty,
518 ) {
519     let Some(mut old_path) = self.inodes.get_path(parent) else {
520         reply.error(libc::ENOENT);
521         return;
522     };
523     old_path.push(name);
524
525     let Some(mut new_path) = self.inodes.get_path(newparent) else {
526         reply.error(libc::ENOENT);
527         return;
528     };
529     new_path.push(newname);
530
531     let mut rename_flag = ssh2::RenameFlags::NATIVE;
532     if flags & libc::RENAME_EXCHANGE != 0 {
533         rename_flag.insert(ssh2::RenameFlags::ATOMIC);
534     }
535     if flags & libc::RENAME_NOREPLACE == 0 {
536         // rename の OVERWRITE が効いてない。手動で消す。
537         if let Ok(stat) = self.sftp.lstat(&new_path) {

```

```

538         if stat.is_dir() {
539             if let Err(e) = self.sftp.rmdir(&new_path) {
540                 reply.error(Error::from(e).0);
541                 return;
542             }
543         } else if let Err(e) = self.sftp.unlink(&new_path) {
544             reply.error(Error::from(e).0);
545             return;
546         }
547         self.inodes.del_inode_with_path(&new_path);
548     }
549 }
550
551 match self.sftp.rename(&old_path, &new_path, Some(rename_flag)) {
552     Ok(_) => {
553         self.inodes.rename(&old_path, &new_path);
554         reply.ok();
555     }
556     Err(e) => reply.error(Error::from(e).0),
557 }
558 }
559 }
560
561 #[derive(Debug, Default)]
562 struct Inodes {
563     list: HashMap<u64, PathBuf>,
564     max_inode: u64,
565 }
566
567 impl Inodes {
568     /// Inode を生成する
569     fn new() -> Self {
570         Self {
571             list: std::collections::HashMap::new(),
572             max_inode: 0,
573         }
574     }
575
576     /// path で指定された inode を生成し、登録する。
577     /// 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。
578     fn add(&mut self, path: &Path) -> u64 {
579         match self.get_inode(path) {
580             Some(i) => i,
581             None => {
582                 self.max_inode += 1;

```

```

583         self.list.insert(self.max_inode, path.into());
584         self.max_inode
585     }
586 }
587 }
588
589 /// path から inode を取得する
590 fn get_inode(&self, path: &Path) -> Option<u64> {
591     self.list.iter().find(|(_, p)| path == *p).map(|(i, _)| *i)
592 }
593
594 /// inode から path を取得する
595 fn get_path(&self, inode: u64) -> Option<PathBuf> {
596     self.list.get(&inode).map(|p| (*p).clone())
597 }
598
599 /// inodes から、inode の登録を削除する
600 fn del_inode(&mut self, inode: u64) -> Option<u64> {
601     self.list.remove(&inode).map(|_| inode)
602 }
603
604 /// inodes から、path の名前の登録を削除する
605 fn del_inode_with_path(&mut self, path: &Path) -> Option<u64> {
606     self.get_inode(path).map(|ino| self.del_inode(ino).unwrap())
607 }
608
609 /// 登録されている inode の path を変更する。
610 /// old_path が存在しなければ、なにもしない。
611 fn rename(&mut self, old_path: &Path, new_path: &Path) {
612     let Some(ino) = self.get_inode(old_path) else {
613         return;
614     };
615     if let Some(val) = self.list.get_mut(&ino) {
616         *val = new_path.into();
617     }
618 }
619 }
620
621 struct Fhandles {
622     list: HashMap<u64, ssh2::File>,
623     next_handle: u64,
624 }
625
626 impl Fhandles {
627     fn new() -> Self {

```

```

628     Self {
629         list: HashMap::new(),
630         next_handle: 0,
631     }
632 }
633
634 fn add_file(&mut self, file: ssh2::File) -> u64 {
635     let handle = self.next_handle;
636     self.list.insert(handle, file);
637     self.next_handle += 1;
638     handle
639 }
640
641 fn get_file(&mut self, fh: u64) -> Option<&mut ssh2::File> {
642     self.list.get_mut(&fh)
643 }
644
645 fn del_file(&mut self, fh: u64) {
646     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。
647                             // ハンドルの再利用のため、次回ハンドルを調整
648     match self.list.keys().max() {
649         Some(&i) => self.next_handle = i + 1,
650         None => self.next_handle = 0,
651     }
652 }
653 }
654
655 #[derive(Debug, Clone, Copy)]
656 struct Error(i32);
657
658 impl From<ssh2::Error> for Error {
659     fn from(value: ssh2::Error) -> Self {
660         let eno = match value.code() {
661             ssh2::ErrorCode::Session(_) => libc::ENXIO,
662             ssh2::ErrorCode::SFTP(i) => match i {
663                 // libssh2のlibssh2_sftp.hにて定義されている。
664                 2 => libc::ENOENT,           // NO_SUCH_FILE
665                 3 => libc::EACCES,           // permission_denied
666                 4 => libc::EIO,              // failure
667                 5 => libc::ENODEV,           // bad message
668                 6 => libc::ENXIO,            // no connection
669                 7 => libc::ENETDOWN,         // connection lost
670                 8 => libc::ENODEV,           // unsported
671                 9 => libc::EBADF,            // invalid handle
672                 10 => libc::ENOENT,          //no such path

```

```

673         11 => libc::EEXIST,      // file already exists
674         12 => libc::EACCES,      // write protected
675         13 => libc::ENXIO,        // no media
676         14 => libc::ENOSPC,       // no space on filesystem
677         15 => libc::EDQUOT,       // quota exceeded
678         16 => libc::ENODEV,       // unknown principal
679         17 => libc::ENOLCK,       // lock conflict
680         18 => libc::ENOTEMPTY,    // dir not empty
681         19 => libc::ENOTDIR,      // not a directory
682         20 => libc::ENAMETOOLONG, // invalid file name
683         21 => libc::ELOOP,        // link loop
684         _ => 0,
685     },
686 };
687 Self(eno)
688 }
689 }
690
691 impl From<std::io::Error> for Error {
692     fn from(value: std::io::Error) -> Self {
693         use std::io::ErrorKind::*;
694         let eno = match value.kind() {
695             NotFound => libc::ENOENT,
696             PermissionDenied => libc::EACCES,
697             ConnectionRefused => libc::ECONNREFUSED,
698             ConnectionReset => libc::ECONNRESET,
699             ConnectionAborted => libc::ECONNABORTED,
700             NotConnected => libc::ENOTCONN,
701             AddrInUse => libc::EADDRINUSE,
702             AddrNotAvailable => libc::EADDRNOTAVAIL,
703             BrokenPipe => libc::EPIPE,
704             AlreadyExists => libc::EEXIST,
705             WouldBlock => libc::EWOULDBLOCK,
706             InvalidInput => libc::EINVAL,
707             InvalidData => libc::EILSEQ,
708             TimedOut => libc::ETIMEDOUT,
709             WriteZero => libc::EIO,
710             Interrupted => libc::EINTR,
711             Unsupported => libc::ENOTSUP,
712             UnexpectedEof => libc::EOF,
713             OutOfMemory => libc::ENOMEM,
714             _ => 0,
715         };
716         Self(eno)
717     }

```

```

718 }
719
720 #[cfg(test)]
721 mod inode_test {
722     use super::Inodes;
723     use std::path::Path;
724
725     #[test]
726     fn inode_add_test() {
727         let mut inodes = Inodes::new();
728         assert_eq!(inodes.add(Path::new("")), 1);
729         assert_eq!(inodes.add(Path::new("test")), 2);
730         assert_eq!(inodes.add(Path::new("")), 1);
731         assert_eq!(inodes.add(Path::new("test")), 2);
732         assert_eq!(inodes.add(Path::new("test3")), 3);
733         assert_eq!(inodes.add(Path::new("/test")), 4);
734         assert_eq!(inodes.add(Path::new("test/")), 2);
735     }
736
737     fn make_inodes() -> Inodes {
738         let mut inodes = Inodes::new();
739         inodes.add(Path::new(""));
740         inodes.add(Path::new("test"));
741         inodes.add(Path::new("test2"));
742         inodes.add(Path::new("test3/"));
743         inodes
744     }
745
746     #[test]
747     fn inodes_get_inode_test() {
748         let inodes = make_inodes();
749         assert_eq!(inodes.get_inode(Path::new("")), Some(1));
750         assert_eq!(inodes.get_inode(Path::new("test4")), None);
751         assert_eq!(inodes.get_inode(Path::new("/test")), None);
752         assert_eq!(inodes.get_inode(Path::new("test3")), Some(4));
753     }
754
755     #[test]
756     fn inodes_get_path_test() {
757         let inodes = make_inodes();
758         assert_eq!(inodes.get_path(1), Some(Path::new("").into()));
759         assert_eq!(inodes.get_path(3), Some(Path::new("test2").into()));
760         assert_eq!(inodes.get_path(5), None);
761         assert_eq!(inodes.get_path(3), Some(Path::new("test2/").into()));
762     }

```

```

763
764     #[test]
765     fn inodes_rename() {
766         let mut inodes = make_inodes();
767         let old = Path::new("test2");
768         let new = Path::new("new_test");
769         let ino = inodes.get_inode(old).unwrap();
770         inodes.rename(old, new);
771         assert_eq!(inodes.get_path(ino), Some(new.into()));
772
773         let mut inodes = make_inodes();
774         let inodes2 = make_inodes();
775         inodes.rename(Path::new("nai"), Path::new("kawattenai"));
776         assert_eq!(inodes.list, inodes2.list);
777     }
778 }

```