

sshmount ソースリスト

美都

2025年11月3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	7
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
6	ファイルハンドル管理モジュール ssh_filesystem/file_handle.rs	31
7	Inode 管理モジュール ssh_filesystem/inode.rs	32
8	双方向ハッシュマップモジュール ssh_filesystem/bi_hash_map.rs	36

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::{Context, Result};
7 use clap::Parser;
8 use cmdline_opt::Opt;
9 use daemonize::Daemonize;
10 use fuse_util::{make_full_path, make_mount_option, make_remote_path};
11 use ssh_connect::make_ssh_session;
12 //use log::debug;
13
14 fn main() -> Result<()> {
15     env_logger::init();
16     let opt = Opt::parse();
17
18     let ssh = make_ssh_session(&opt).context("Failed to generate ssh session.")?;
19
20     let path = make_remote_path(&opt, &ssh).context("Failed to generate remote path.")?;
21     let options = make_mount_option(&opt);
22     let mount_point = make_full_path(&opt.mount_point)?;
23
24     // プロセスのデーモン化
25     if opt.daemon {
26         let daemonize = Daemonize::new();
27         if let Err(e) = daemonize.start() {
28             eprintln!("daemonization failed.(error: {})", e);
29         }
30     }
31     // ファイルシステムへのマウント実行
32     let fs = ssh_filesystem::Sshfs::new(ssh, &path)?;
33     fuser::mount2(fs, mount_point, &options).context("Failed to mount FUSE.")?;
34     Ok(())
35 }
```

2 コマンドラインオプションの定義 cmdline_opt.rs

```
1 use anyhow::anyhow, Context;
2 use clap::Parser;
3 use std::path::PathBuf;
4
5 /// コマンドラインオプション
6 #[derive(Parser)]
7 #[command(author, version, about)]
8 pub struct Opt {
9     /// Destination [user@]host:[path]
10    pub remote: RemoteName,
11    /// Path to mount
12    #[arg(value_parser = exist_dir)]
13    pub mount_point: String,
14    /// Path to config file
15    #[arg(short = 'F', long)]
16    pub config_file: Option<PathBuf>,
17    /// Login name
18    #[arg(short, long)]
19    pub login_name: Option<String>,
20    /// File name of secret key file
21    #[arg(short, long)]
22    pub identity: Option<PathBuf>,
23    /// Port no
24    #[arg(short, long, default_value_t = 22)]
25    pub port: u16,
26    /// Read only
27    #[arg(short, long)]
28    pub readonly: bool,
29    /// Not executable
30    #[arg(long)]
31    pub no_exec: bool,
32    /// Do not change access date and time(atime)
33    #[arg(long)]
34    pub no_atime: bool,
35    /// run in daemon mode
36    #[arg(short, long)]
37    pub daemon: bool,
38 }
39
40 /// 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。
41 fn exist_dir(s: &str) -> anyhow::Result<String> {
42     match std::fs::read_dir(s) {
43         Ok(mut dir) => match dir.next() {
```

```

44     None => Ok(s.to_string()),
45     Some(_) => Err(anyhow!("Mount destination directory is not empty.")),
46   },
47   Err(e) => match e.kind() {
48     std::io::ErrorKind::NotFound => Err(anyhow!("The mount directory does not exist.")),
49     std::io::ErrorKind::NotConnected => Err(anyhow!(
50       "The network of the mount directory is disconnected. (Did you forget to umount?)."
51     )),
52     _ => Err(e).context("Unexpected error.(check mount directory)"),
53   },
54 }
55 }

56

57 /// コマンドラインの接続先ホスト情報
58 #[derive(Clone, Debug, PartialEq)]
59 pub struct RemoteName {
60   /// ユーザー名
61   pub user: Option<String>,
62   /// ホスト名 または IP アドレス
63   pub host: String,
64   /// 接続先パス
65   pub path: Option<std::path::PathBuf>,
66 }

67

68 impl std::fmt::Display for RemoteName {
69   fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
70     let s = format!("<{:?}><{:?}><{:?}>", &self.user, &self.host, &self.path);
71     s.fmt(f)
72   }
73 }

74

75 impl std::str::FromStr for RemoteName {
76   type Err = ErrorRemoteName;
77   fn from_str(s: &str) -> Result<Self, Self::Err> {
78     let mut rest_str = s;
79     let user = match rest_str.split_once('@') {
80       Some((u, r)) => {
81         rest_str = r;
82         if !u.trim().is_empty() {
83           Some(u.trim().to_string())
84         } else {
85           None
86         }
87       }
88       None => None,
89     };

```

```

90     let (host, path) = match rest_str.split_once(':') {
91         Some((h, p)) => (
92             if !h.trim().is_empty() {
93                 h.trim().to_string()
94             } else {
95                 return Err(ErrorRemoteName);
96             },
97             if !p.trim().is_empty() {
98                 Some(std::path::PathBuf::from(p.trim().to_string()))
99             } else {
100                 None
101             },
102         ),
103         None => return Err(ErrorRemoteName),
104     };
105     Ok(Self { user, host, path })
106 }
107 }
108
109 #[derive(thiserror::Error, Debug, PartialEq, Eq)]
110 #[error("The format of the host to connect to is \"[user@]host:[path]\\\"")]
111 pub struct ErrorRemoteName;
112
113 #[cfg(test)]
114 mod test {
115     use super::*;

116     #[test]
117     fn verify_cli() {
118         use clap::CommandFactory;
119         Opt::command().debug_assert()
120     }
121
122     #[test]
123     fn test_from_str_remotename() {
124         use std::path::Path;
125         let s = "mito@reterminal.local:/home/mito";
126         let r: RemoteName = s.parse().unwrap();
127         let k = RemoteName {
128             user: Some("mito".to_string()),
129             host: "reterminal.local".to_string(),
130             path: Some(Path::new("/home/mito").into()),
131         };
132         assert_eq!(r, k);
133
134         let s = "mito@reterminal.local:/home/mito/";
135         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

```
1 //! ssh 接続関連関数モジュール
2
3 use crate::cmdline_opt::Opt;
4 use anyhow::{anyhow, Context, Result};
5 use dialoguer::Password;
6 use dns_lookup::lookup_host;
7 use log::{debug, error};
8 use ssh2::Session;
9 use ssh2_config::{HostParams, ParseRule, SshConfig};
10 use std::{
11     fs::File,
12     io::BufReader,
13     net::TcpStream,
14     path::{Path, PathBuf},
15     str,
16 };
17
18 /// セッションを生成する。
19 pub fn make_ssh_session(opt: &Opt) -> Result<Session> {
20     let host_params = get_ssh_config(&opt.config_file).query(&opt.remote.host);
21     let address = get_address(opt, &host_params).context("Failed to get host address")?;
22     let username = get_username(opt, &host_params).context("Failed to get user name.")?;
23     debug!(
24         "[main] 接続先情報-> ユーザー:\"{}\", ip address:{}",
25         &username, &address
26     );
27     let identity_file = get_identity_file(opt, &host_params)?;
28
29     let ssh = connect_ssh(address).context("The ssh connection failed.")?;
30     userauth(&ssh, &username, &identity_file).context("User authentication failed.")?;
31     Ok(ssh)
32 }
33
34 /// ホストの ip アドレス解決
35 fn get_address(opt: &Opt, host_params: &HostParams) -> Result<std::net::SocketAddr> {
36     let dns = host_params.host_name.as_deref().unwrap_or(&opt.remote.host);
37     let addr = lookup_host(dns)
38         .inspect_err(|e| error!("get_address : Failed lookup_host[{}]", e))
39         .context("Cannot find host to connect to.")?
40         .collect::<Vec<_>>();
41     let addr = addr
42         .first()
43         .ok_or(anyhow!("Unable to obtain DNS address."))
44 }
```

```

44     .inspect_err(|e| error!("get_address : {}", e))?;
45     Ok(std::net::SocketAddr::from((*addr, opt.port)))
46 }
47
48 /// ssh-config の取得と解析
49 /// ファイル名が指定されていない場合は "~/.ssh/config" を使用
50 /// 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。
51 fn get_ssh_config(file_opt: &Option<PathBuf>) -> SshConfig {
52     get_config_file(file_opt)
53         .map(BufReader::new)
54         .map_or(SshConfig::default(), |mut f| {
55             SshConfig::default()
56                 .parse(&mut f, ParseRule::ALLOW_UNKNOWN_FIELDS)
57                 .unwrap_or_else(|e| {
58                     eprintln!("警告:config ファイル内にエラー -- {}", e);
59                     SshConfig::default()
60                 })
61         })
62 }
63
64 /// ssh_config ファイルがあれば、オープンする。
65 /// ファイル名の指定がなければ、$Home/.ssh/config を想定する。
66 fn get_config_file(file_name: &Option<PathBuf>) -> Option<std::fs::File> {
67     let file_name = file_name.clone().or_else(|| {
68         home::home_dir().map(|p| {
69             let mut p = p;
70             p.push(".ssh/config");
71             p
72         })
73     });
74
75     file_name.and_then(|p| File::open(p).ok())
76 }
77
78 /// ログイン名を確定し、取得する。
79 /// 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
80 fn get_username(opt: &Opt, params: &HostParams) -> Result<String> {
81     if let Some(n) = &opt.login_name {
82         Ok(n.clone())
83     } else if let Some(n) = &opt.remote.user {
84         Ok(n.clone())
85     } else if let Some(n) = &params.user {
86         Ok(n.clone())
87     } else if let Some(n) = users::get_current_username() {
88         n.to_str()
89         .map(|s| s.to_string())

```

```

90         .ok_or(anyhow!("Invalid login user name. -- {n:?}"))
91     } else {
92         Err(anyhow!("Could not obtain user name."))
93     }
94 }
95
96 ///////////////////////////////////////////////////////////////////
97 fn get_identity_file(opt: &Opt, host_params: &HostParams) -> Result<Option<PathBuf>> {
98     if let Some(n) = &opt.identity {
99         std::fs::File::open(n).with_context(|| {
100             format!(
101                 "Unable to access the secret key file specified by the \"-i\" option. [{:?}]",
102                 &n
103             )
104         })?;
105     Ok(Some(n.clone()))
106 } else {
107     let name = host_params.identity_file.as_ref().map(|p| p[0].clone());
108     if let Some(ref n) = name {
109         std::fs::File::open(n).with_context(|| {
110             format!(
111                 "Unnable to access the secret file specified by the ssh-config. [{:?}]",
112                 &n
113             )
114         })?;
115     }
116     Ok(name)
117 }
118 }
119
120 ///////////////////////////////////////////////////////////////////
121 fn connect_ssh<A: std::net::ToSocketAddrs>(address: A) -> Result<Session> {
122     let tcp = TcpStream::connect(address).context("Failed to connect to TCP/IP.")?;
123     let mut ssh = Session::new().context("Failed to connect to ssh.")?;
124     ssh.set_tcp_stream(tcp);
125     ssh.handshake().context("Failed to handshake ssh.")?;
126     Ok(ssh)
127 }
128
129 ///////////////////////////////////////////////////////////////////
130 fn userauth(sess: &Session, username: &str, identity: &Option<PathBuf>) -> Result<()> {
131     if user_auth_agent(sess, username).is_ok() {
132         return Ok(());
133     }
134     if let Some(f) = identity {
135         if user_auth_identity(sess, username, f).is_ok() {

```

```

136         return Ok(());
137     }
138 }
139 user_auth_password(sess, username)
140     .map_err(|_| anyhow!("All user authentication methods failed."))
141 }
142
143 /// agent 認証
144 fn user_auth_agent(sess: &Session, username: &str) -> Result<(), ssh2::Error> {
145     let ret = sess.userauth_agent(username);
146     if ret.is_err() {
147         debug!("認証失敗 (agent)->{:?}", ret.as_ref().unwrap_err());
148     };
149     ret
150 }
151
152 /// 公開キー認証
153 fn user_auth_identity(sess: &Session, username: &str, key_file: &Path) -> Result<(), String> {
154     let mut ret = sess.userauth_pubkey_file(username, None, key_file, None);
155     if ret.is_ok() {
156         return Ok(());
157     };
158     if let ErrorCode::Session(-16) = ret.as_ref().unwrap_err().code() {
159         // error_code -16 -
160         // LIBSSH2_ERROR_FILE:PUBLIC_KEY の取得失敗。多分、秘密キーのパスフレーズ
161         for _i in 0..3 {
162             let password = Password::new()
163                 .with_prompt("Enter the passphrase for the secret key.")
164                 .allow_empty_password(true)
165                 .interact()
166                 .map_err(|e| e.to_string())?;
167             ret = sess.userauth_pubkey_file(username, None, key_file, Some(&password));
168             if ret.is_ok() {
169                 return Ok(());
170             }
171             eprintln!("The passphrase is different.");
172         }
173     }
174     debug!("認証失敗 (pubkey)->{:?}", ret.as_ref().unwrap_err());
175     Err("公開キー認証失敗".to_string())
176 }
177
178 /// パスワード認証
179 fn user_auth_password(sess: &Session, username: &str) -> Result<(), String> {
180     for _i in 0..3 {
181         let password = Password::new()

```

```
182     .with_prompt("Enter your login password.")
183     .allow_empty_password(true)
184     .interact()
185     .map_err(|e| e.to_string())?;
186 
187     let ret = sess.userauth_password(username, &password);
188 
189     if ret.is_ok() {
190         return Ok(());
191     }
192 
193     let ssh2::ErrorCode::Session(-18) = ret.as_ref().unwrap_err().code() else {
194         break;
195     };
196 
197     // ssh2エラーコード -18 ->
198     // LIBSSH2_ERROR_AUTHENTICATION_FAILED: パスワードが違うんでしょう。
199     eprintln!("The password is different.");
200     debug!("認証失敗 (password)->{:?}", ret.unwrap_err());
201 
202     Err("パスワード認証失敗".to_string())
203 }
```

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::env::current_dir;
7  use std:: {
8      io::Read,
9      path::{Path, PathBuf},
10     str,
11 };
12
13  ///! マウントポイントのフルパスを生成する
14  pub fn make_full_path<P: AsRef<Path>>(path: P) -> Result<PathBuf> {
15      if path.as_ref().is_absolute() {
16          Ok(path.as_ref().to_path_buf())
17      } else {
18          let mut full_path = current_dir().context("cannot access current directory.")?;
19          full_path.push(path);
20          Ok(full_path)
21      }
22  }
23
24  ///! リモート接続先の path の生成
25  pub fn make_remote_path(opt: &Opt, session: &Session) -> Result<PathBuf> {
26      // パスの生成
27      const MSG_ERRORHOME: &str = "Fail to generate path name.";
28      let mut path = match opt.remote.path {
29          Some(ref p) => {
30              if p.is_absolute() {
31                  p.clone()
32              } else {
33                  let mut h = get_home_on_remote(session).context(MSG_ERRORHOME)?;
34                  h.push(p);
35                  h
36              }
37          }
38          None => get_home_on_remote(session).context(MSG_ERRORHOME)?,
39      };
40      // 生成したパスが実在するかを確認する
41      let sftp = session
42          .sftp()
43          .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     }

```

```
90     options
91 }
92
93     /// ssh接続先のカレントディレクトリを取得する
94 fn get_home_on_remote(session: &Session) -> Result<PathBuf> {
95     let mut channel = session
96         .channel_session()
97         .context("Fail to build ssh channel.")?;
98     channel
99         .exec("pwd")
100        .context("Fail to execute \"pwd\" command.")?;
101    let mut buf = Vec::<u8>::new();
102    channel
103        .read_to_end(&mut buf)
104        .context("Fail to get response for \"pwd\" command.")?;
105    channel.close().context("Fail to close ssh channel.")?;
106    str::from_utf8(&buf)
107        .context("The pwd result contains non-utf8 characters.")?
108        .trim()
109        .parse::<PathBuf>()
110        .context("Fail to build path name.")
111 }
```

5 ファイルシステムモジュール ssh_filesystem.rs

```
1 mod bi_hash_map;
2 mod file_handle;
3 mod inode;
4
5 use file_handle::Fhandles;
6 use inode::Inodes;
7
8 use anyhow::Context;
9 use fuser::{FileAttr, Filesystem, ReplyAttr, ReplyData, ReplyDirectory, ReplyEntry, Request};
10 use libc::ENOENT;
11 use log::{debug, error, warn};
12 use ssh2::{ErrorCode, OpenFlags, OpenType, Session, Sftp};
13 use std::{
14     ffi::OsStr,
15     io::{Read, Seek, SeekFrom, Write},
16     path::{Path, PathBuf},
17     time::{Duration, SystemTime, UNIX_EPOCH},
18 };
19
20 /// FUSE ファイルシステム実装
21 pub struct Sshfs {
22     _session: Session,
23     sftp: Sftp,
24     inodes: Inodes,
25     fhandles: Fhandles,
26     _top_path: PathBuf,
27 }
28
29 impl Sshfs {
30     pub fn new<P: AsRef<Path>>(session: Session, path: P) -> anyhow::Result<Self> {
31         let mut inodes = Inodes::new();
32         let top_path: PathBuf = path.as_ref().into();
33         inodes.add(&top_path);
34         let sftp = session
35             .sftp()
36             .inspect_err(|_| {
37                 error!("Failed to create sftp from session.");
38             })
39             .context("Failed to create sftp from session.(Sshfs::new)")?;
40         debug!(
41             "[Sshfs::new] connect path: <{:?}>, inodes=<{:?}>",
42             &top_path, &inodes
43         );
44     }
45 }
```

```

44     Ok(Self {
45         _session: session,
46         sftp,
47         inodes,
48         fhandles: 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 setattr_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         let Some(mut path) = self.inodes.get_path(parent) else {
102             debug!("[lookup] 親ディレクトリの検索に失敗 inode={}", parent);
103             reply.error(ENOENT);
104             return;
105         };
106         path.push(Path::new(name));
107         match self.getattr_from_ssh2(&path, req.uid(), req.gid()) {
108             Ok(attr) => reply.entry(&Duration::from_secs(1), &attr, 0),
109             Err(e) => {
110                 reply.error(e.0);
111             }
112         };
113     }
114
115     fn getattr(&mut self, req: &Request, ino: u64, _fh: Option<u64>, reply: ReplyAttr) {
116         let Some(path) = self.inodes.get_path(ino) else {
117             debug!("[getattr] path 取得失敗: inode={}", ino);
118             reply.error(ENOENT);
119             return;
120         };
121         match self.getattr_from_ssh2(&path, req.uid(), req.gid()) {
122             Ok(attr) => {
123                 //debug!("[getattr] retrun attr: {:?}", &attr);
124                 reply.attr(&Duration::from_secs(1), &attr);
125             }
126             Err(e) => {
127                 warn!("[getattr] getattr_from_ssh2 エラー: {:?}", &e);
128                 reply.error(e.0)
129             }
130         };
131     }
132
133     fn readdir(
134         &mut self,
135         _req: &Request,

```

```

136     ino: u64,
137     _fh: u64,
138     offset: i64,
139     mut reply: ReplyDirectory,
140 ) {
141     let Some(path) = self.inodes.get_path(ino) else {
142         reply.error(libc::ENOENT);
143         return;
144     };
145     match self.sftp.readdir(&path) {
146         Ok(mut dir) => {
147             let cur_file_attr = ssh2::FileStat {
148                 size: None,
149                 uid: None,
150                 gid: None,
151                 perm: Some(libc::S_IFDIR),
152                 atime: None,
153                 mtime: None,
154             }; // ".." の解決用。 attr ディレクトリであることのみを示す。
155             dir.insert(0, (Path::new("..").into(), cur_file_attr.clone()));
156             dir.insert(0, (Path::new(".").into(), cur_file_attr));
157             let mut i = offset + 1;
158             for f in dir.iter().skip(offset as usize) {
159                 let ino = if f.0 == Path::new(..) || f.0 == Path::new(.) {
160                     1
161                 } else {
162                     self.inodes.add(&f.0)
163                 };
164                 let name = match f.0.file_name() {
165                     Some(n) => n,
166                     None => f.0.as_os_str(),
167                 };
168                 let filetype = &f.1.file_type();
169                 let filetype = match Self::conv_file_kind_ssh2fuser(filetype) {
170                     Ok(t) => t,
171                     Err(e) => {
172                         warn!(
173                             "[readdir] ファイルタイプ解析失敗: inode={}, name={:{}?}",
174                             ino, name
175                         );
176                         reply.error(e.0);
177                         return;
178                     }
179                 };
180                 if reply.add(ino, i, filetype, name) {
181                     break;

```

```

182         }
183         i += 1;
184     }
185     reply.ok();
186 }
187 Err(e) => {
188     warn!("[readdir] ssh2::readdir 内でエラー発生-- {:?}", e);
189     reply.error(Error::from(e).0);
190 }
191 }
192 }
193
194 fn readlink(&mut self, _req: &Request<'_>, ino: u64, reply: ReplyData) {
195     let Some(path) = self.inodes.get_path(ino) else {
196         error!("[readlink] 親ディレクトリの検索に失敗 {:?}", ino);
197         reply.error(libc::ENOENT);
198         return;
199     };
200     match self.sftp.readlink(&path) {
201         Ok(p) => {
202             //debug!("[readlink] ret_path => {:?}", &p);
203             reply.data(p.as_os_str().to_string_lossy().as_bytes());
204         }
205         Err(e) => {
206             //debug!("[readlink] ssh2::readlink error => {:?}", e);
207             reply.error(Error::from(e).0);
208         }
209     }
210 }
211
212 fn open(&mut self, _req: &Request<'_>, ino: u64, flags: i32, reply: fuser::ReplyOpen) {
213     let Some(file_name) = self.inodes.get_path(ino) else {
214         reply.error(libc::ENOENT);
215         return;
216     };
217
218     let mut flags_ssh2 = OpenFlags::empty();
219     if flags & libc::O_WRONLY != 0 {
220         flags_ssh2.insert(OpenFlags::WRITE);
221     } else if flags & libc::O_RDWR != 0 {
222         flags_ssh2.insert(OpenFlags::READ);
223         flags_ssh2.insert(OpenFlags::WRITE);
224     } else {
225         flags_ssh2.insert(OpenFlags::READ);
226     }
227     if flags & libc::O_APPEND != 0 {

```

```

228     flags_ssh2.insert(OpenFlags::APPEND);
229 }
230 if flags & libc::O_CREAT != 0 {
231     flags_ssh2.insert(OpenFlags::CREATE);
232 }
233 if flags & libc::O_TRUNC != 0 {
234     flags_ssh2.insert(OpenFlags::TRUNCATE);
235 }
236 if flags & libc::O_EXCL != 0 {
237     flags_ssh2.insert(OpenFlags::EXCLUSIVE);
238 }
239
240 debug!(
241     "[open] filename='{:?}', openflag = {:?}, bit = {:x}",
242     &file_name,
243     &flags_ssh2,
244     flags_ssh2.bits()
245 );
246 match self
247     .sftp
248     .open_mode(&file_name, flags_ssh2, 0o777, ssh2::OpenType::File)
249 {
250     Ok(file) => {
251         let fh = self.fhandles.add_file(file);
252         reply.opened(fh, flags as u32);
253     }
254     Err(e) => {
255         log::error!(
256             "file-open error: filename='{:?}', mode={:?}", err={}",
257             &file_name,
258             &flags_ssh2,
259             &e
260         );
261         reply.error(Error::from(e).0);
262     }
263 }
264
265
266 fn release(
267     &mut self,
268     _req: &Request<'_>,
269     _ino: u64,
270     fh: u64,
271     _flags: i32,
272     _lock_owner: Option<u64>,
273     _flush: bool,

```

```

274         reply: fuser::ReplyEmpty,
275     ) {
276         self.fhandles.del_file(fh);
277         reply.ok();
278     }
279
280     fn read(
281         &mut self,
282         _req: &Request,
283         _ino: u64,
284         fh: u64,
285         offset: i64,
286         size: u32,
287         _flags: i32,
288         _lock_owner: Option<u64>,
289         reply: ReplyData,
290     ) {
291         let Some(file_mutex) = self.fhandles.get_file(fh) else {
292             reply.error(libc::EINVAL);
293             return;
294         };
295         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。
296
297         if let Err(e) = file.seek(std::io::SeekFrom::Start(offset as u64)) {
298             reply.error(Error::from(e).0);
299             return;
300         }
301         let mut buff = vec![0; size as usize];
302         let mut read_size: usize = 0;
303         while read_size < size as usize {
304             match file.read(&mut buff[read_size..]) {
305                 Ok(s) => {
306                     if s == 0 {
307                         break;
308                     };
309                     read_size += s;
310                 }
311                 Err(e) => {
312                     reply.error(Error::from(e).0);
313                     return;
314                 }
315             }
316         }
317     }
318 }
319

```

```

320     }
321 
322     buff.resize(read_size, 0u8);
323     reply.data(&buff);
324 }
325 
326 fn write(
327     &mut self,
328     _req: &Request<'_>,
329     _ino: u64,
330     fh: u64,
331     offset: i64,
332     data: &[u8],
333     _write_flags: u32,
334     _flags: i32,
335     _lock_owner: Option<u64>,
336     reply: fuser::ReplyWrite,
337 ) {
338 
339     let Some(file_mutex) = self.fhandls.get_file(fh) else {
340         reply.error(libc::EINVAL);
341         return;
342     };
343 
344     let file = &mut file_mutex.lock().unwrap();
345 
346     if let Err(e) = file.seek(std::io::SeekFrom::Start(offset as u64)) {
347         reply.error(Error::from(e).0);
348         return;
349     }
350 
351     let mut buf = data;
352     while !buf.is_empty() {
353 
354         let cnt = match file.write(buf) {
355             Ok(cnt) => cnt,
356             Err(e) => {
357                 reply.error(Error::from(e).0);
358                 return;
359             }
360         };
361 
362         buf = &buf[cnt..];
363     }
364 
365     reply.written(data.len() as u32);
366 }
367 
368 fn lseek(
369     &mut self,
370     _req: &Request<'_>,
371     _ino: u64,
372     fh: u64,

```

```

366     offset: i64,
367     whence: i32,
368     reply: fuser::ReplyLseek,
369 ) {
370     let seek_from = match whence {
371         libc::SEEK_SET => SeekFrom::Start(offset as u64),
372         libc::SEEK_CUR => SeekFrom::Current(offset),
373         libc::SEEK_END => SeekFrom::End(offset),
374         _ => {
375             reply.error(libc::EINVAL);
376             return;
377         }
378     };
379     let Some(file_mutex) = self.fhandles.get_file(fh) else {
380         reply.error(libc::EBADF);
381         return;
382     };
383     let file = &mut file_mutex.lock().unwrap();
384     let pos = match file.seek(seek_from) {
385         Ok(p) => p,
386         Err(e) => {
387             reply.error(Error::from(e).0);
388             return;
389         }
390     };
391     reply.offset(pos as i64);
392 }
393
394 fn mknod(
395     &mut self,
396     req: &Request<'_>,
397     parent: u64,
398     name: &OsStr,
399     mode: u32,
400     umask: u32,
401     _rdev: u32,
402     reply: ReplyEntry,
403 ) {
404     if mode & libc::S_IFMT != libc::S_IFREG {
405         reply.error(libc::EPERM);
406         return;
407     }
408     let mode = mode & (!umask | libc::S_IFMT);
409     let Some(mut new_name) = self.inodes.get_path(parent) else {
410         reply.error(libc::ENOENT);
411         return;

```

```

412     };
413 
414     new_name.push(name);
415 
416     if let Err(e) =
417         self.sftp
418             .open_mode(&new_name, OpenFlags::CREATE, mode as i32, OpenType::File)
419     {
420 
421         reply.error(Error::from(e).0);
422 
423         return;
424     }
425 
426     let new_attr = match self.getattr_from_ssh2(&new_name, req.uid(), req.gid()) {
427 
428         Ok(a) => a,
429 
430         Err(e) => {
431 
432             reply.error(e.0);
433 
434             return;
435         }
436     };
437 
438     reply.entry(&Duration::from_secs(1), &new_attr, 0);
439 }
440 
441 fn unlink(&mut self, _req: &Request<'_>, parent: u64, name: &OsStr, reply: fuser::ReplyEmpty) {
442 
443     let Some(mut path) = self.inodes.get_path(parent) else {
444 
445         reply.error(libc::ENOENT);
446 
447         return;
448     };
449 
450     path.push(name);
451 
452     match self.sftp.unlink(&path) {
453 
454         Ok(_) => {
455 
456             self.inodes.del_inode_with_path(&path);
457 
458             reply.ok();
459         }
460 
461         Err(e) => reply.error(Error::from(e).0),
462     }
463 }
464 
465 fn mkdir(
466     &mut self,
467 
468     req: &Request<'_>,
469 
470     parent: u64,
471 
472     name: &OsStr,
473 
474     mode: u32,
475 
476     umask: u32,
477 
478     reply: ReplyEntry,
479 )
480 {
481 
482     let Some(mut path) = self.inodes.get_path(parent) else {
483 
484         reply.error(libc::ENOENT);
485 
486         return;
487     }
488 
```

```

458     };
459     path.push(name);
460
461     let mode = (mode & (!umask) & 0o777) as i32;
462
463     match self.sftp.mkdir(&path, mode) {
464         Ok(_) => match self.getattr_from_ssh2(&path, req.uid(), req.gid()) {
465             Ok(attr) => reply.entry(&Duration::from_secs(1), &attr, 0),
466             Err(e) => reply.error(e.0),
467         },
468         Err(e) => reply.error(Error::from(e.0)),
469     }
470 }
471
472 fn rmdir(&mut self, _req: &Request<'_>, parent: u64, name: &OsStr, reply: fuser::ReplyEmpty) {
473     let Some(mut path) = self.inodes.get_path(parent) else {
474         reply.error(libc::ENOENT);
475         return;
476     };
477     path.push(name);
478     match self.sftp.rmdir(&path) {
479         Ok(_) => {
480             self.inodes.del_inode_with_path(&path);
481             reply.ok()
482         }
483         Err(e) => {
484             if e.code() == ErrorCode::Session(-31) {
485                 // ssh2 ライブラリの返すエラーが妙。置換しておく。
486                 reply.error(libc::ENOTEMPTY);
487             } else {
488                 reply.error(Error::from(e.0))
489             }
490         }
491     }
492 }
493
494 fn symlink(
495     &mut self,
496     req: &Request<'_>,
497     parent: u64,
498     name: &OsStr,
499     link: &Path,
500     reply: ReplyEntry,
501 ) {
502     let Some(mut target) = self.inodes.get_path(parent) else {
503         reply.error(libc::ENOENT);

```

```

504     return;
505 }
506 target.push(name);
507 match self.sftp.symlink(link, &target) {
508     Ok(_) => match self.getattr_from_ssh2(&target, req.uid(), req.gid()) {
509         Ok(attr) => reply.entry(&Duration::from_secs(1), &attr, 0),
510         Err(e) => reply.error(e.0),
511     },
512     Err(e) => reply.error(Error::from(e).0),
513 }
514 }
515
516 fn setattr(
517     &mut self,
518     req: &Request<'_>,
519     ino: u64,
520     mode: Option<u32>,
521     _uid: Option<u32>,
522     _gid: Option<u32>,
523     size: Option<u64>,
524     atime: Option<fuser::TimeOrNow>,
525     mtime: Option<fuser::TimeOrNow>,
526     _ctime: Option<std::time::SystemTime>,
527     _fh: Option<u64>,
528     _crttime: Option<std::time::SystemTime>,
529     _chgttime: Option<std::time::SystemTime>,
530     _bkuptime: Option<std::time::SystemTime>,
531     _flags: Option<u32>,
532     reply: ReplyAttr,
533 ) {
534     let stat = ssh2::FileStat {
535         size,
536         uid: None,
537         gid: None,
538         perm: mode,
539         atime: atime.map(|t| {
540             Self::conv_timeornow2systemtime(&t)
541                 .duration_since(UNIX_EPOCH)
542                 .unwrap_or_default()
543                 .as_secs()
544         }),
545         mtime: mtime.map(|t| {
546             Self::conv_timeornow2systemtime(&t)
547                 .duration_since(UNIX_EPOCH)
548                 .unwrap_or_default()
549                 .as_secs()

```

```

550         },
551     );
552     let Some(filename) = self.inodes.get_path(ino) else {
553         reply.error(ENOENT);
554         return;
555     };
556     match self.sftp.setstat(&filename, stat) {
557         Ok(_) => {
558             let stat = self.getattr_from_ssh2(&filename, req.uid(), req.gid());
559             match stat {
560                 Ok(s) => reply.attr(&Duration::from_secs(1), &s),
561                 Err(e) => reply.error(e.0),
562             }
563         }
564         Err(e) => reply.error(Error::from(e).0),
565     }
566 }
567
568 fn rename(
569     &mut self,
570     _req: &Request<'_>,
571     parent: u64,
572     name: &OsStr,
573     newparent: u64,
574     newname: &OsStr,
575     flags: u32,
576     reply: fuser::ReplyEmpty,
577 ) {
578     let Some(mut old_path) = self.inodes.get_path(parent) else {
579         reply.error(libc::ENOENT);
580         return;
581     };
582     old_path.push(name);
583
584     let Some(mut new_path) = self.inodes.get_path(newparent) else {
585         reply.error(libc::ENOENT);
586         return;
587     };
588     new_path.push(newname);
589
590     let mut rename_flag = ssh2::RenameFlags::NATIVE;
591     if flags & libc::RENAME_EXCHANGE != 0 {
592         rename_flag.insert(ssh2::RenameFlags::ATOMIC);
593     }
594     if flags & libc::RENAME_NOREPLACE == 0 {
595         // rename の OVERWRITE が効いてない。手動で消す。

```

```

596     if let Ok(stat) = self.sftp.lstat(&new_path) {
597         if stat.is_dir() {
598             if let Err(e) = self.sftp.rmdir(&new_path) {
599                 reply.error(Error::from(e).0);
600                 return;
601             }
602         } else if let Err(e) = self.sftp.unlink(&new_path) {
603             reply.error(Error::from(e).0);
604             return;
605         }
606         self.inodes.del_inode_with_path(&new_path);
607     }
608 }
609
610 match self.sftp.rename(&old_path, &new_path, Some(rename_flag)) {
611     Ok(_) => {
612         self.inodes.rename(&old_path, &new_path);
613         reply.ok();
614     }
615     Err(e) => reply.error(Error::from(e).0),
616 }
617 }
618 }
619
620 #[derive(Debug, Clone, Copy)]
621 struct Error(i32);
622
623 impl From<ssh2::Error> for Error {
624     fn from(value: ssh2::Error) -> Self {
625         let eno = match value.code() {
626             ssh2::ErrorCode::Session(_) => libc::ENXIO,
627             ssh2::ErrorCode::SFTP(i) => match i {
628                 // libssh2 の libssh2_sftp.h にて定義されている。
629                 2 => libc::ENOENT,           // NO_SUCH_FILE
630                 3 => libc::EACCES,          // permission_denied
631                 4 => libc::EIO,              // failure
632                 5 => libc::ENODEV,           // bad message
633                 6 => libc::ENXIO,            // no connection
634                 7 => libc::ENETDOWN,          // connection lost
635                 8 => libc::ENODEV,           // unsupported
636                 9 => libc::EBADF,             // invalid handle
637                 10 => libc::ENOENT,           // no such path
638                 11 => libc::EEXIST,            // file already exists
639                 12 => libc::EACCES,           // write protected
640                 13 => libc::ENXIO,            // no media
641                 14 => libc::ENOSPC,           // no space on filesystem

```

```

642             15 => libc::EDQUOT,           // quota exceeded
643             16 => libc::ENODEV,          // unknown principal
644             17 => libc::ENOLCK,          // lock conflict
645             18 => libc::ENOTEMPTY,        // dir not empty
646             19 => libc::ENOTDIR,          // not a directory
647             20 => libc::ENAMETOOLONG,      // invalid file name
648             21 => libc::ELOOP,            // link loop
649             _ => {
650                 error!("An unknown error occurred during SSH2.[{}]", i);
651                 libc::EIO
652             }
653         },
654     );
655     Self(eno)
656 }
657 }
658
659 impl From<std::io::Error> for Error {
660     fn from(value: std::io::Error) -> Self {
661         use std::io::ErrorKind::*;
662         let eno = match value.kind() {
663             NotFound => libc::ENOENT,
664             PermissionDenied => libc::EACCES,
665             ConnectionRefused => libc::ECONNREFUSED,
666             ConnectionReset => libc::ECONNRESET,
667             ConnectionAborted => libc::ECONNABORTED,
668             NotConnected => libc::ENOTCONN,
669             AddrInUse => libc::EADDRINUSE,
670             AddrNotAvailable => libc::EADDRNOTAVAIL,
671             BrokenPipe => libc::EPIPE,
672             AlreadyExists => libc::EEXIST,
673             WouldBlock => libc::EWOULDBLOCK,
674             InvalidInput => libc::EINVAL,
675             InvalidData => libc::EILSEQ,
676             TimedOut => libc::ETIMEDOUT,
677             WriteZero => libc::EIO,
678             Interrupted => libc::EINTR,
679             Unsupported => libc::ENOTSUP,
680             UnexpectedEof => libc::EOF,
681             OutOfMemory => libc::ENOMEM,
682             _ => {
683                 error!(
684                     "An unknown error occurred during std::io.[{}]",
685                     value.kind()
686                 );
687                 libc::EIO

```

```
688     }
689     };
690     Self(eno)
691 }
692 }
```

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

```
1  ///! ファイルハンドル管理モジュール
2
3  use std::collections::HashMap;
4  use std::sync::{
5      atomic::{AtomicU64, Ordering},
6      Arc, Mutex,
7  };
8
9  ///! ファイルハンドル管理構造体
10 pub(super) struct Fhandles {
11     list: Mutex<HashMap<u64, Arc<Mutex<ssh2::File>>>,
12     next_handle: AtomicU64,
13 }
14
15 impl Fhandles {
16     pub(super) fn new() -> Self {
17         Self {
18             list: Mutex::new(HashMap::new()),
19             next_handle: AtomicU64::new(0),
20         }
21     }
22
23     pub(super) fn add_file(&mut self, file: ssh2::File) -> u64 {
24         let handle = self.next_handle.fetch_add(1, Ordering::AcqRel);
25         self.list
26             .lock()
27             .unwrap()
28             .insert(handle, Arc::new(Mutex::new(file)));
29         handle
30         // 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
31         // → ない。
32         // 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。
33         // よって、このモジュール内において、lock().unwrap() とする。
34     }
35
36     pub(super) fn get_file(&self, fh: u64) -> Option<Arc<Mutex<ssh2::File>>> {
37         self.list.lock().unwrap().get(&fh).cloned()
38     }
39
40     pub(super) fn del_file(&mut self, fh: u64) {
41         self.list.lock().unwrap().remove(&fh); // 戻り値は捨てる。この時点ではファイルはクローズ。
42     }
}
```

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

```
1  ///! Inode 管理モジュール
2
3  use super::bi_hash_map::BiHashMap;
4
5  use std::path::{Path, PathBuf};
6  use std::sync::{
7      atomic::{AtomicU64, Ordering},
8      Mutex,
9  };
10
11 /// Inode 管理構造体
12 #[derive(Debug, Default)]
13 pub(super) struct Inodes {
14     list: Mutex<BiHashMap<u64, PathBuf>>,
15     next_inode: AtomicU64,
16 }
17
18 impl Inodes {
19     /// Inodes を生成する
20     pub(super) fn new() -> Self {
21         Self {
22             list: Mutex::new(BiHashMap::new()),
23             next_inode: AtomicU64::new(2),
24         }
25     }
26
27     /// path で指定された inode を生成し、登録する。
28     /// 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。
29     /// 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 2 が割り当たら
30     /// れる。
31     pub(super) fn add<P: AsRef<Path>>(&mut self, path: P) -> u64 {
32         let mut list_guard = self.list.lock().unwrap();
33         // 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。
34         // 最善の方法が、即時システムを落とすことである。
35         // 以下、このモジュール全体に共通。
36         let path = PathBuf::from(path.as_ref());
37         match list_guard.get_left(&path) {
38             Some(i) => *i,
39             None => {
40                 let inode = self.next_inode.fetch_add(1, Ordering::AcqRel);
41                 if list_guard.insert_no_overwrite(inode, path.clone()).is_err() {
42                     unreachable!("Unexpected duplicate inode {} or path {:?}", inode, path);
43                 }
44                 // 既に重複がチェックされているので、ありえない。
45             }
46         }
47     }
48 }
```

```

43         }
44         inode
45     }
46 }
47 }

48

49 /// pathから inodeを取得する
50 /// 主用途が消滅したが、将来のために残しておく
51 #[allow(dead_code)]
52 pub(super) fn get_inode<P: AsRef<Path>>(&self, path: P) -> Option<u64> {
53     let path = PathBuf::from(path.as_ref());
54     self.list.lock().unwrap().get_left(&path).copied()
55 }

56

57 /// inodeから pathを取得する
58 pub(super) fn get_path(&self, inode: u64) -> Option<PathBuf> {
59     self.list.lock().unwrap().get_right(&inode).cloned()
60 }

61

62 /// inodesから、inodeの登録を削除する
63 /// (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)
64 #[allow(dead_code)]
65 pub(super) fn del_inode(&mut self, inode: u64) -> Option<u64> {
66     self.list.lock().unwrap().remove_left(&inode).map(|_| inode)
67 }

68

69 /// path名から iNodeの登録を削除する
70 pub(super) fn del_inode_with_path<P: AsRef<Path>>(&mut self, path: P) -> Option<u64> {
71     let path = PathBuf::from(path.as_ref());
72     self.list.lock().unwrap().remove_right(&path)
73 }

74

75 /// 登録されている inodeの pathを変更する。
76 /// old_pathが存在しなければ、なにもしない。
77 pub(super) fn rename<P: AsRef<Path>>(&mut self, old_path: P, new_path: P) {
78     let old_path = PathBuf::from(old_path.as_ref());
79     let new_path = PathBuf::from(new_path.as_ref());
80     let mut list_guard = self.list.lock().unwrap();
81     let Some(ino) = list_guard.get_left(&old_path).copied() else {
82         return;
83     };
84     list_guard.remove_left(&ino);
85     list_guard.insert(ino, new_path);
86 }
87 }

88

```

```

89 #[cfg(test)]
90 mod inode_test {
91     use super::Inodes;
92     use std::path::Path;
93
94     #[test]
95     fn inode_add_test() {
96         let mut inodes = Inodes::new();
97         assert_eq!(inodes.add(""), 2);
98         assert_eq!(inodes.add(Path::new("test")), 3);
99         assert_eq!(inodes.add(Path::new(""))), 2);
100        assert_eq!(inodes.add(Path::new("test")), 3);
101        assert_eq!(inodes.add(Path::new("test3")), 4);
102        assert_eq!(inodes.add(Path::new("/test")), 5);
103        assert_eq!(inodes.add(Path::new("test/")), 3);
104    }
105
106    fn make_inodes() -> Inodes {
107        let mut inodes = Inodes::new();
108        inodes.add(Path::new(""));
109        inodes.add(Path::new("test"));
110        inodes.add(Path::new("test2"));
111        inodes.add(Path::new("test3/"));
112        inodes
113    }
114
115    #[test]
116    fn inodes_get_inode_test() {
117        let inodes = make_inodes();
118        assert_eq!(inodes.get_inode(Path::new("")), Some(2));
119        assert_eq!(inodes.get_inode(Path::new("test4")), None);
120        assert_eq!(inodes.get_inode(Path::new("/test")), None);
121        assert_eq!(inodes.get_inode(Path::new("test3")), Some(5));
122    }
123
124    #[test]
125    fn inodes_get_path_test() {
126        let inodes = make_inodes();
127        assert_eq!(inodes.get_path(2), Some(Path::new("").into()));
128        assert_eq!(inodes.get_path(4), Some(Path::new("test2").into()));
129        assert_eq!(inodes.get_path(6), None);
130        assert_eq!(inodes.get_path(4), Some(Path::new("test2/").into()));
131    }
132
133    #[test]
134    fn inodes_rename() {

```

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

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

```
1  /// bidirectional hash map
2  /// 双方向ハッシュマップ
3
4  use std::collections::HashMap, hash::Hash, sync::Arc;
5
6  /// 双方向ハッシュマップ
7  #[derive(Debug, Default, PartialEq, Eq)]
8  pub(super) struct BiHashMap<L, R>
9  where
10     L: Hash + Eq + Clone,
11     R: Hash + Eq + Clone,
12 {
13     left: HashMap<Arc<L>, Arc<R>>,
14     right: HashMap<Arc<R>, Arc<L>>,
15 }
16
17 impl<L, R> BiHashMap<L, R>
18 where
19     L: Hash + Eq + Clone,
20     R: Hash + Eq + Clone,
21 {
22     /// 新しい双方向ハッシュマップを生成する
23     pub fn new() -> Self {
24         BiHashMap {
25             right: HashMap::new(),
26             left: HashMap::new(),
27         }
28     }
29
30     /// チェック無しで挿入する。
31     /// 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。
32     fn insert_no_check(&mut self, left: L, right: R) {
33         let right = Arc::new(right);
34         let left = Arc::new(left);
35         self.right.insert(right.clone(), left.clone());
36         self.left.insert(left, right);
37     }
38
39     /// マップに新しい要素を挿入する。
40     /// 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。
41     pub fn insert(&mut self, left: L, right: R) -> OverwriteResult<L, R> {
42         let old_left = self.right.remove(&right);
43         let old_right = self.left.remove(&left);
```

```

44     let result = match (old_left, old_right) {
45         (None, None) => OverwriteResult::NoOverwrite,
46         (Some(old_l), None) => {
47             self.left.remove(old_l.as_ref());
48             OverwriteResult::OverwriteLeft((*old_l).clone())
49         }
50         (None, Some(old_r)) => {
51             self.right.remove(old_r.as_ref());
52             OverwriteResult::OverwriteRight((*old_r).clone())
53         }
54         (Some(old_l), Some(old_r)) => {
55             self.left.remove(old_l.as_ref());
56             self.right.remove(old_r.as_ref());
57             OverwriteResult::OverwriteBoth(
58                 (left.clone(), (*old_r).clone()),
59                 ((*old_l).clone(), right.clone()),
60             )
61         }
62     };
63     self.insert_no_check(left, right);
64     result
65 }
66
67 ///////////////////////////////////////////////////////////////////
68 ///////////////////////////////////////////////////////////////////
69 pub fn insert_no_overwrite(&mut self, left: L, right: R) -> Result<(), ()> {
70     if self.contains_left(&left) || self.contains_right(&right) {
71         return Err(());
72     }
73     self.insert_no_check(left, right);
74     Ok(())
75 }
76
77 ///////////////////////////////////////////////////////////////////
78 ///////////////////////////////////////////////////////////////////
79 pub fn get_right(&self, left: &L) -> Option<&R> {
80     self.left.get(left).map(|arc_r| arc_r.as_ref())
81 }
82
83 ///////////////////////////////////////////////////////////////////
84 ///////////////////////////////////////////////////////////////////
85 pub fn get_left(&self, right: &R) -> Option<&L> {
86     self.right.get(right).map(|arc_l| arc_l.as_ref())
87 }
88
89 ///////////////////////////////////////////////////////////////////

```

```

90  /// 存在する場合は true、存在しない場合は false を返す
91  pub fn contains_left(&self, left: &L) -> bool {
92      self.left.contains_key(left)
93  }
94
95  /// 右側のキーが存在するかどうかを返す
96  /// 存在する場合は true、存在しない場合は false を返す
97  pub fn contains_right(&self, right: &R) -> bool {
98      self.right.contains_key(right)
99  }
100
101 /// 左側の値から、リストの項目を削除する
102 /// 存在しない場合は、なにもしない。
103 /// 以前の値を返す。(存在しない場合は None)
104 pub fn remove_left(&mut self, left: &L) -> Option<R> {
105     let result = self.left.remove(left).map(|arc_r| (*arc_r).clone());
106     if let Some(right) = result.as_ref() {
107         self.right.remove(right);
108     };
109     result
110 }
111
112 /// 右側の値から、リストの項目を削除する
113 /// 存在しない場合は、なにもしない。
114 /// 以前の値を返す。(存在しない場合は None)
115 pub fn remove_right(&mut self, right: &R) -> Option<L> {
116     let result = self.right.remove(right).map(|arc_l| (*arc_l).clone());
117     if let Some(left) = result.as_ref() {
118         self.left.remove(left);
119     };
120     result
121 }
122 }
123
124 /// 挿入時の上書き結果
125 #[derive(Debug, PartialEq, Eq)]
126 pub enum OverwriteResult<L, R> {
127     NoOverwrite,
128     OverwriteRight(R),
129     OverwriteLeft(L),
130     OverwriteBoth((L, R), (L, R)),
131 }
132
133 #[cfg(test)]
134 mod tests {
135     use super::*;


```

```

137 #[test]
138 /// 単純な挿入と、値の取得のテスト
139 fn tanjyunna_insert() {
140     let mut bimap = BiHashMap::new();
141     assert_eq!(bimap.insert(1, "a"), OverwriteResult::NoOverwrite);
142     assert_eq!(bimap.get_right(&1), Some(&"a"));
143     assert_eq!(bimap.get_left(&"a"), Some(&1));
144     assert_eq!(bimap.insert_no_overwrite(2, "b"), Ok(()));
145     assert_eq!(bimap.get_right(&2), Some(&"b"));
146     assert_eq!(bimap.get_left(&"b"), Some(&2));
147     assert!(bimap.contains_left(&1));
148     assert!(bimap.contains_right(&"b"));
149 }
150
151 #[test]
152 /// 上書き挿入のテスト
153 /// 左側、右側、両側の上書きのケースを確認する
154 fn overwrite_insert() {
155     let mut bimap = BiHashMap::new();
156     assert_eq!(bimap.insert(1, "a"), OverwriteResult::NoOverwrite);
157     print_hash_map(&bimap, "insert (1, 'a')");
158     assert_eq!(bimap.insert(1, "b"), OverwriteResult::OverwriteRight("a"));
159     print_hash_map(&bimap, "insert (1, 'b')");
160     assert_eq!(bimap.get_right(&1), Some(&"b"));
161     assert_eq!(bimap.insert(2, "b"), OverwriteResult::OverwriteLeft(1));
162     print_hash_map(&bimap, "insert (2, 'b')");
163     assert_eq!(bimap.get_left(&"b"), Some(&2));
164     assert_eq!(bimap.get_right(&2), Some(&"b"));
165     assert_eq!(bimap.insert_no_overwrite(3, "c"), Ok(()));
166     print_hash_map(&bimap, "insert (3, 'c')");
167     assert_eq!(
168         bimap.insert(2, "b"),
169         OverwriteResult::OverwriteBoth((2, "b"), (2, "b"))
170     );
171     print_hash_map(&bimap, "insert (2, 'b') again");
172     assert_eq!(
173         bimap.insert(3, "b"),
174         OverwriteResult::OverwriteBoth((3, "c"), (2, "b"))
175     );
176     print_hash_map(&bimap, "insert (3, 'b')");
177     assert_eq!(bimap.get_right(&3), Some(&"b"));
178     assert_eq!(bimap.get_left(&"b"), Some(&3));
179     assert_eq!(bimap.left.len(), bimap.right.len());
180     assert_eq!(bimap.get_right(&2), None);
181     assert_eq!(bimap.insert_no_overwrite(3, "d"), Err(()));

```

```

182     print_hash_map(&bimap, "insert_no_overwrite (3, 'd') [fail]");
183     assert_eq!(bimap.insert_no_overwrite(5, "e"), Ok(()));
184     print_hash_map(&bimap, "insert_no_overwrite (5, 'e') [ok]");
185     assert_eq!(bimap.insert_no_overwrite(5, "d"), Err(()));
186     print_hash_map(&bimap, "insert_no_overwrite (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 }

```