sshmount ソースリスト

美都

2023 年 1 月 25 日

目次

1　メインモジュール　main.rs

```rust
mod ssh_filesystem;

use clap::Parser;
use dialoguer::Password;
use dns_lookup::lookup_host;
use log::debug;
use ssh2::Session;
use ssh2_config::SshConfig;
use std::{
    fs::File,
    io::{BufReader, Read},
    net::TcpStream,
    path::PathBuf,
    str,
};

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

    // ssh config ファイルの取得と解析
    let mut ssh_config = SshConfig::default();
    {
        let file = get_config_file(&opt.config_file).map(BufReader::new);
        if let Some(mut f) = file {
            match SshConfig::default().parse(&mut f) {
                Ok(c) => ssh_config = c,
                Err(e) => eprintln!("警告:config ファイル内にエラー -- {}", e),
            };
        };
    }

    // ssh config のエイリアスを解決し、接続アドレスの逆引き。
    let mut dns = &opt.remote.host;
    let host_params = ssh_config.query(dns);
    if let Some(ref n) = host_params.host_name {
        dns = n
    };
    let address =
        lookup_host(dns).map_err(|_| format!("接続先ホストが見つかりません。({})", dns))?;

    // ログイン名の確定
```

```
43      let username: String = if let Some(n) = &opt.login_name {
44          n.clone()
45      } else if let Some(n) = &opt.remote.user {
46          n.clone()
47      } else if let Some(n) = &host_params.user {
48          n.clone()
49      } else if let Some(n) = users::get_current_username() {
50          n.to_str()
51              .ok_or_else(|| format!("ログインユーザ名不正。({:?})", n))?
52              .to_string()
53      } else {
54          Err("ユーザー名が取得できませんでした。")?
55      };
56      debug!("[main] ログインユーザー名: {}", &username);
57
58      // 秘密キーファイル名の取得
59      let identity_file: Option<PathBuf> = if let Some(ref i) = host_params.identity_file {
60          Some(i[0].clone())
61      } else {
62          opt.identity.as_ref().cloned()
63      };
64
65      // ssh 接続作業
66      let socketaddr = std::net::SocketAddr::from((address[0], opt.port));
67      debug!("接続先: {:?}", socketaddr);
68      let tcp = TcpStream::connect(socketaddr).unwrap();
69      let mut ssh = Session::new().unwrap();
70      ssh.set_tcp_stream(tcp);
71      ssh.handshake().unwrap();
72      // ssh 認証
73      userauth(&ssh, &username, &identity_file)?;
74
75      // リモートホストのトップディレクトリの生成
76      let path = make_remote_path(&opt, &ssh)?;
77
78      let fs = ssh_filesystem::Sshfs::new(ssh, &path);
79      let options = vec![fuser::MountOption::FSName("sshfs".to_string())];
80      fuser::mount2(fs, opt.mount_point, &options).unwrap();
81      Ok(())
82  }
83
84  /// ssh 認証を実施する。
85  fn userauth(sess: &Session, username: &str, identity: &Option<PathBuf>) -> Result<(), String> {
86      let ret = sess.userauth_agent(username);
87      if ret.is_ok() {
```

```
88          return Ok(());
89      }
90      debug!("認証失敗(agent)->{:?}", ret.unwrap_err());
91      if let Some(f) = identity {
92          let ret = sess.userauth_pubkey_file(username, None, f, None);
93          if ret.is_ok() {
94              return Ok(());
95          }
96          if let ssh2::ErrorCode::Session(-16) = ret.as_ref().unwrap_err().code() {
97              // error_code -16 ->
98              // LIBSSH2_ERROR_FILE:PUBLIC_KEYの取得失敗。多分、秘密キーのパスフレーズ
99              for _i in 0..3 {
100                 let password = Password::new()
101                     .with_prompt("秘密キーのパスフレーズを入力してください。")
102                     .allow_empty_password(true)
103                     .interact()
104                     .map_err(|e| e.to_string())?;
105                 let ret = sess.userauth_pubkey_file(username, None, f, Some(&password));
106                 if ret.is_ok() {
107                     return Ok(());
108                 }
109                 eprintln!("パスフレーズが違います。");
110             }
111         }
112         debug!("認証失敗(pubkey)->{:?}", ret.unwrap_err());
113     }
114     for _i in 0..3 {
115         let password = Password::new()
116             .with_prompt("ログインパスワードを入力してください。")
117             .allow_empty_password(true)
118             .interact()
119             .map_err(|e| e.to_string())?;
120         let ret = sess.userauth_password(username, &password);
121         if ret.is_ok() {
122             return Ok(());
123         }
124         let ssh2::ErrorCode::Session(-18) = ret.as_ref().unwrap_err().code() else { break; };
125         // ssh2エラーコード  -18 ->
126         // LIBSSH2_ERROR_AUTHENTICATION_FAILED: パスワードが違うんでしょう。
127         eprintln!("パスワードが違います。");
128         debug!("認証失敗(password)->{:?}", ret.unwrap_err());
129     }
130     Err("sshの認証に失敗しました。".to_string())
131 }
132
```

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

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

/// リモート接続先の path の生成
fn make_remote_path(opt: &Opt, session: &Session) -> Result<PathBuf, String> {
    // パスの生成
    let mut path = match opt.remote.path {
        Some(ref p) => {
            if p.is_absolute() {
                p.clone()
            } else {
                let mut h = get_home_on_remote(session)?;
                h.push(p);
                h
            }
        }
        None => get_home_on_remote(session)?,
    };
    // 生成したパスが実在するかを確認する
    let sftp = session
        .sftp()
        .map_err(|e| format!("接続作業中、リモートへの sftp 接続に失敗しました。-- {}", e))?;
    let file_stat = sftp
        .stat(&path)
        .map_err(|_| format!("接続先のパスが見つかりません。{:?}", &path))?;
    if !file_stat.is_dir() {
        Err("接続先のパスはディレクトリではありません。")?;
    };
    // 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
    let file_stat = sftp.lstat(&path).unwrap();
    if file_stat.file_type().is_symlink() {
        path = sftp
            .readlink(&path)
            .map_err(|e| format!("接続先のシンボリックリンクの解決に失敗しました。-- {}", e))?;
```

```
178         if !path.is_absolute() {
179             let tmp = path;
180             path = get_home_on_remote(session)?;
181             path.push(tmp);
182         };
183     };
184 
185     Ok(path)
186 }
187 
188 /// ssh 接続先のカレントディレクトリを取得する
189 fn get_home_on_remote(session: &Session) -> Result<PathBuf, String> {
190     let mut channel = session
191         .channel_session()
192         .map_err(|e| format!("接続作業中、ssh のチャンネル構築に失敗しました。-- {}", e))?;
193     channel
194         .exec("pwd")
195         .map_err(|e| format!("HOME ディレクトリの取得に失敗しました。-- {}", e))?;
196     let mut buf = Vec::<u8>::new();
197     channel
198         .read_to_end(&mut buf)
199         .map_err(|e| format!("HOME ディレクトリの取得に失敗しました (2) -- {}", e))?;
200     channel.close().map_err(|e| {
201         format!(
202             "接続作業中、ssh チャンネルのクローズに失敗しました。-- {}",
203             e
204         )
205     })?;
206     str::from_utf8(&buf)
207         .map_err(|e| format!("HOME ディレクトリの取得に失敗しました (3) -- {}", e))?
208         .trim()
209         .parse::<PathBuf>()
210         .map_err(|e| format!("HOME ディレクトリの取得に失敗しました (4) -- {}", e))
211 }
212 
213 /// コマンドラインオプション
214 #[derive(Parser)]
215 #[command(author, version, about)]
216 struct Opt {
217     /// 接続先 [user@]host:[path]
218     remote: RemoteName,
219     /// マウント先のパス
220     #[arg(value_parser = exist_dir)]
221     mount_point: String,
222     /// config ファイルのパス指定
```

```rust
    #[arg(short = 'F', long)]
    config_file: Option<PathBuf>,
    /// ログイン名
    #[arg(short, long)]
    login_name: Option<String>,
    /// 秘密キーファイル名
    #[arg(short, long)]
    identity: Option<PathBuf>,
    /// ポート番号
    #[arg(short, long, default_value_t = 22)]
    port: u16,
}

/// 指定されたディレクトリが存在し、中にファイルがないことを確認する。
fn exist_dir(s: &str) -> Result<String, String> {
    match std::fs::read_dir(s) {
        Ok(mut dir) => match dir.next() {
            None => Ok(s.to_string()),
            Some(_) => Err("マウント先ディレクトリが空ではありません".to_string()),
        },
        Err(e) => match e.kind() {
            std::io::ErrorKind::NotFound => {
                Err("マウント先ディレクトリが存在しません。".to_string())
            }
            _ => Err("計り知れないエラーです。".to_string()),
        },
    }
}

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

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

```
}

impl std::str::FromStr for RemoteName {
    type Err = String;
    fn from_str(s: &str) -> Result<Self, Self::Err> {
        let mut rest_str = s;
        let user = match rest_str.split_once('@') {
            Some((u, r)) => {
                rest_str = r;
                if !u.trim().is_empty() {
                    Some(u.trim().to_string())
                } else {
                    None
                }
            }
            None => None,
        };
        let (host, path) = match rest_str.split_once(':') {
            Some((h, p)) => (
                if !h.trim().is_empty() {
                    h.trim().to_string()
                } else {
                    return Err("接続先ホストの形式は、\"[user@]host:[path]\"です。".to_string());
                },
                if !p.trim().is_empty() {
                    Some(std::path::PathBuf::from(p.trim().to_string()))
                } else {
                    None
                },
            ),
            None => return Err("接続先ホストの形式は、\"[user@]host:[path]\"です。".to_string()),
        };
        Ok(Self { user, host, path })
    }
}

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

```

```
313     #[test]
314     fn test_from_str_remotename() {
315         use std::path::Path;
316         let s = "mito@reterminal.local:/home/mito";
317         let r: RemoteName = s.parse().unwrap();
318         let k = RemoteName {
319             user: Some("mito".to_string()),
320             host: "reterminal.local".to_string(),
321             path: Some(Path::new("/home/mito").into()),
322         };
323         assert_eq!(r, k);
324 
325         let s = "mito@reterminal.local:/home/mito/";
326         let r: RemoteName = s.parse().unwrap();
327         let k = RemoteName {
328             user: Some("mito".to_string()),
329             host: "reterminal.local".to_string(),
330             path: Some(Path::new("/home/mito").into()),
331         };
332         assert_eq!(r, k);
333 
334         let s = "reterminal.local:";
335         let r: RemoteName = s.parse().unwrap();
336         let k = RemoteName {
337             user: None,
338             host: "reterminal.local".to_string(),
339             path: None,
340         };
341         assert_eq!(r, k);
342 
343         let s = " mito @reterminal.local: ";
344         let r: RemoteName = s.parse().unwrap();
345         let k = RemoteName {
346             user: Some("mito".to_string()),
347             host: "reterminal.local".to_string(),
348             path: None,
349         };
350         assert_eq!(r, k);
351 
352         let s = "reterminal.local";
353         let r: Result<RemoteName, String> = s.parse();
354         assert_eq!(
355             r,
356             Err("接続先ホストの形式は、\"[user@]host:[path]\"です。".to_string())
357         );
```

```
358
359         let s = "mito@reterminal.local";
360         let r: Result<RemoteName, String> = s.parse();
361         assert_eq!(
362             r,
363             Err("接続先ホストの形式は、\"[user@]host:[path]\"です。".to_string())
364         );
365
366         let s = " mito @: ";
367         let r: Result<RemoteName, String> = s.parse();
368         assert_eq!(
369             r,
370             Err("接続先ホストの形式は、\"[user@]host:[path]\"です。".to_string())
371         );
372     }
373 }
```

2　ファイルシステムモジュール ssh_filesystem.rs

```rust
/// FUSE ファイルシステム実装
use fuser::{
    FileAttr, Filesystem, ReplyAttr, ReplyData, ReplyDirectory, ReplyEntry, Request,
};
use libc::ENOENT;
use log::{warn, debug};
use ssh2::{Session, Sftp, OpenType, OpenFlags};
use std::{
    ffi::OsStr,
    path::{Path, PathBuf},
    time::{SystemTime,Duration, UNIX_EPOCH},
    io::{Seek, Read, Write},
};

pub struct Sshfs {
    _session: Session,
    sftp: Sftp,
    inodes: Inodes,
    _top_path: PathBuf,
}

impl Sshfs {
    pub fn new(session: Session, path: &Path) -> Self {
        let mut inodes = Inodes::new();
        let top_path: PathBuf = path.into();
        inodes.add(&top_path);
        let sftp = session.sftp().unwrap();
        debug!("[Sshfs::new] connect path: <{:?}>, inodes=<{:?}>", &top_path, &inodes.list);
        Self {
            _session: session,
            sftp,
            inodes,
            _top_path: top_path,
        }
    }

    /// ssh2経由でファイルのステータスを取得する。
    /// 副作用：取得に成功した場合、inodes にパスを登録する。
    fn getattr_from_ssh2(
        &mut self,
        path: &Path,
        uid: u32,
```

```
        gid: u32,
    ) -> Result<FileAttr, Error> {
        let attr_ssh2 = self.sftp.lstat(path)?;
        let kind = Self::conv_file_kind_ssh2fuser(&attr_ssh2.file_type())?;
        let ino = self.inodes.add(path);
        Ok(FileAttr {
            ino,
            size: attr_ssh2.size.unwrap_or(0),
            blocks: attr_ssh2.size.unwrap_or(0) / 512 + 1,
            atime: UNIX_EPOCH + Duration::from_secs(attr_ssh2.atime.unwrap_or(0)),
            mtime: UNIX_EPOCH + Duration::from_secs(attr_ssh2.mtime.unwrap_or(0)),
            ctime: UNIX_EPOCH + Duration::from_secs(attr_ssh2.mtime.unwrap_or(0)),
            crtime: UNIX_EPOCH,
            kind,
            perm: attr_ssh2.perm.unwrap_or(0o666) as u16,
            nlink: 1,
            uid,
            gid,
            rdev: 0,
            blksize: 512,
            flags: 0,
        })
    }

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

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

```

```rust
impl Filesystem for Sshfs {
    fn lookup(&mut self, req: &Request, parent: u64, name: &OsStr, reply: ReplyEntry) {
        let Some(mut path) = self.inodes.get_path(parent) else {
                debug!("[lookup] 親ディレクトリの検索に失敗 inode={}", parent);
                reply.error(ENOENT);
                return;
        };
        path.push(Path::new(name));
        match self.getattr_from_ssh2(&path, req.uid(), req.gid()) {
            Ok(attr) => reply.entry(&Duration::from_secs(1), &attr, 0),
            Err(e) => {
                reply.error(e.0);
            }
        };
    }

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

    fn readdir(
        &mut self,
        _req: &Request,
        ino: u64,
        _fh: u64,
        offset: i64,
        mut reply: ReplyDirectory,
    ) {
        let Some(path) = self.inodes.get_path(ino) else {
            reply.error(libc::ENOENT);
            return;
```

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

```
177
178     fn readlink(&mut self, _req: &Request<'_>, ino: u64, reply: ReplyData) {
179         let Some(path) = self.inodes.get_path(ino) else {
180             reply.error(libc::ENOENT);
181             return;
182         };
183         match self.sftp.readlink(&path) {
184             Ok(p) => reply.data(p.as_os_str().to_str().unwrap().as_bytes()),
185             Err(e) => reply.error(Error::from(e).0),
186         }
187     }
188
189     fn read(
190         &mut self,
191         _req: &Request,
192         ino: u64,
193         _fh: u64,
194         offset: i64,
195         size: u32,
196         _flags: i32,
197         _lock_owner: Option<u64>,
198         reply: ReplyData,
199     ) {
200         let Some(path) = self.inodes.get_path(ino) else {
201             reply.error(libc::ENOENT);
202             return;
203         };
204         match self.sftp.open(&path) {
205             Ok(mut f) => {
206                 if let Err(e) = f.seek(std::io::SeekFrom::Start(offset as u64)) {
207                     reply.error(Error::from(e).0);
208                     return;
209                 }
210                 let mut buff = Vec::<u8>::new();
211                 buff.resize(size as usize, 0u8);
212                 let mut read_size : usize = 0;
213                 while read_size < size as usize {
214                     match f.read(&mut buff[read_size..]) {
215                         Ok(s) => {
216                            if s == 0 {break;};
217                            read_size += s;
218                         }
219                         Err(e) => {
220                             reply.error(Error::from(e).0);
221                             return;
```

```
                    }
                }
            }
            reply.data(&buff);
        }
        Err(e) => {
            reply.error(Error::from(e).0);
        }
    }
}

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

fn write(
        &mut self,
```

```
            _req: &Request<'_>,
            ino: u64,
            _fh: u64,
            offset: i64,
            data: &[u8],
            _write_flags: u32,
            _flags: i32,
            _lock_owner: Option<u64>,
            reply: fuser::ReplyWrite,
        ) {
        let Some(filename) = self.inodes.get_path(ino) else {
            reply.error(ENOENT);
            return;
        };
        let mut file = match self.sftp.open_mode(&filename, ssh2::OpenFlags::WRITE, 0,
        ↪  ssh2::OpenType::File) {
            Ok(file) => file,
            Err(e) => {
                reply.error(Error::from(e).0);
                return;
            },
        };
        if let Err(e) = file.seek(std::io::SeekFrom::Start(offset as u64)) {
            reply.error(Error::from(e).0);
            return;
        }
        let mut buf = data;
        while !buf.is_empty() {
            let cnt = match file.write(buf) {
                Ok(cnt) => cnt,
                Err(e) => {
                    reply.error(Error::from(e).0);
                    return;
                }
            };
            buf = &buf[cnt..];
        }
        reply.written(data.len() as u32);
    }

    fn setattr(
            &mut self,
            req: &Request<'_>,
            ino: u64,
            mode: Option<u32>,
```

```
            _uid: Option<u32>,
            _gid: Option<u32>,
            size: Option<u64>,
            atime: Option<fuser::TimeOrNow>,
            mtime: Option<fuser::TimeOrNow>,
            _ctime: Option<std::time::SystemTime>,
            _fh: Option<u64>,
            _crtime: Option<std::time::SystemTime>,
            _chgtime: Option<std::time::SystemTime>,
            _bkuptime: Option<std::time::SystemTime>,
            _flags: Option<u32>,
            reply: ReplyAttr,
        ) {
        let stat = ssh2::FileStat{
            size,
            uid: None,
            gid: None,
            perm: mode,
            atime: atime.map(|t|
                Self::conv_timeornow2systemtime(&t).duration_since(UNIX_EPOCH).unwrap().as_secs()
            ),
            mtime: mtime.map(|t|
                Self::conv_timeornow2systemtime(&t).duration_since(UNIX_EPOCH).unwrap().as_secs()
            ),
        };
        let Some(filename) = self.inodes.get_path(ino) else {
             reply.error(ENOENT);
             return;
        };
        match self.sftp.setstat(&filename, stat) {
            Ok(_) => {
                let stat = self.getattr_from_ssh2(&filename, req.uid(), req.gid());
                match stat {
                    Ok(s) => reply.attr(&Duration::from_secs(1), &s),
                    Err(e) => reply.error(e.0),
                }
            },
            Err(e) => reply.error(Error::from(e).0),
        }
    }
}

#[derive(Debug, Default)]
struct Inodes {
    list: std::collections::HashMap<u64, PathBuf>,
```

```
355     max_inode: u64,
356 }
357 
358 impl Inodes {
359     /// Inodeを生成する
360     fn new() -> Self {
361         Self {
362             list: std::collections::HashMap::new(),
363             max_inode: 0,
364         }
365     }
366 
367     /// pathで指定されたinodeを生成し、登録する。
368     /// 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。
369     fn add(&mut self, path: &Path) -> u64 {
370         match self.get_inode(path){
371             Some(i) => i,
372             None => {
373                 self.max_inode += 1;
374                 self.list.insert(self.max_inode, path.into());
375                 self.max_inode
376             }
377         }
378     }
379 
380     /// pathからinodeを取得する
381     fn get_inode(&self, path: &Path) -> Option<u64> {
382         self.list.iter().find(|(_, p)| path == *p).map(|(i, _)| *i)
383     }
384 
385     /// inodeからpathを取得する
386     fn get_path(&self, inode: u64) -> Option<PathBuf> {
387         self.list.get(&inode).map(|p| (*p).clone())
388     }
389 }
390 
391 #[derive(Debug, Clone, Copy)]
392 struct Error(i32);
393 
394 impl From<ssh2::Error> for Error {
395     fn from(value : ssh2::Error) -> Self {
396         let eno = match value.code() {
397             ssh2::ErrorCode::Session(_) => libc::ENXIO,
398             ssh2::ErrorCode::SFTP(i) =>
399                 match i {
```

```
                    // libssh2の libssh2_sftp.hにて定義されている。
                    2 => libc::ENOENT,  // NO_SUCH_FILE
                    3 => libc::EACCES,  // permission_denied
                    4 => libc::EIO,     // failure
                    5 => libc::ENODEV,  // bad message
                    6 => libc::ENXIO,   // no connection
                    7 => libc::ENETDOWN,// connection lost
                    8 => libc::ENODEV,  // unsported
                    9 => libc::EBADF,   // invalid handle
                    10 => libc::ENOENT, //no such path
                    11 => libc::EEXIST, // file already exists
                    12 => libc::EACCES, // write protected
                    13 => libc::ENXIO,  // no media
                    14 => libc::ENOSPC, // no space on filesystem
                    15 => libc::EDQUOT, // quota exceeded
                    16 => libc::ENODEV, // unknown principal
                    17 => libc::ENOLCK, // lock conflict
                    18 => libc::ENOTEMPTY, // dir not empty
                    19 => libc::ENOTDIR,// not a directory
                    20 => libc::ENAMETOOLONG,// invalid file name
                    21 => libc::ELOOP, // link loop
                    _ => 0,
                }
        };
        Self(eno)
    }
}

impl From<std::io::Error> for Error {
    fn from(value : std::io::Error) -> Self {
        use std::io::ErrorKind::*;
        let eno = match value.kind() {
            NotFound => libc::ENOENT,
            PermissionDenied => libc::EACCES,
            ConnectionRefused => libc::ECONNREFUSED,
            ConnectionReset => libc::ECONNRESET,
            ConnectionAborted => libc::ECONNABORTED,
            NotConnected => libc::ENOTCONN,
            AddrInUse => libc::EADDRINUSE,
            AddrNotAvailable => libc::EADDRNOTAVAIL,
            BrokenPipe => libc::EPIPE,
            AlreadyExists => libc::EEXIST,
            WouldBlock => libc::EWOULDBLOCK,
            InvalidInput => libc::EINVAL,
            InvalidData => libc::EILSEQ,
```

```
445                 TimedOut => libc::ETIMEDOUT,
446                 WriteZero => libc::EIO,
447                 Interrupted => libc::EINTR,
448                 Unsupported => libc::ENOTSUP,
449                 UnexpectedEof => libc::EOF,
450                 OutOfMemory => libc::ENOMEM,
451                 _ => 0,
452             };
453             Self(eno)
454         }
455     }
456
457     #[cfg(test)]
458     mod inode_test {
459         use super::Inodes;
460         use std::path::Path;
461
462         #[test]
463         fn inode_add_test() {
464             let mut inodes = Inodes::new();
465             assert_eq!(inodes.add(Path::new("")), 1);
466             assert_eq!(inodes.add(Path::new("test")), 2);
467             assert_eq!(inodes.add(Path::new("")), 1);
468             assert_eq!(inodes.add(Path::new("test")), 2);
469             assert_eq!(inodes.add(Path::new("test3")), 3);
470             assert_eq!(inodes.add(Path::new("/test")), 4);
471             assert_eq!(inodes.add(Path::new("test/")), 2);
472         }
473
474         fn make_inodes() -> Inodes {
475             let mut inodes = Inodes::new();
476             inodes.add(Path::new(""));
477             inodes.add(Path::new("test"));
478             inodes.add(Path::new("test2"));
479             inodes.add(Path::new("test3/"));
480             inodes
481         }
482
483         #[test]
484         fn inodes_get_inode_test() {
485             let inodes = make_inodes();
486             assert_eq!(inodes.get_inode(Path::new("")), Some(1));
487             assert_eq!(inodes.get_inode(Path::new("test4")), None);
488             assert_eq!(inodes.get_inode(Path::new("/test")), None);
489             assert_eq!(inodes.get_inode(Path::new("test3")), Some(4));
```

```
490         }
491
492         #[test]
493         fn inodes_get_path_test() {
494             let inodes = make_inodes();
495             assert_eq!(inodes.get_path(1), Some(Path::new("").into()));
496             assert_eq!(inodes.get_path(3), Some(Path::new("test2").into()));
497             assert_eq!(inodes.get_path(5), None);
498             assert_eq!(inodes.get_path(3), Some(Path::new("test2/").into()));
499         }
500     }
```