sshmount ソースリスト

美都

2023 年 2 月 4 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
    let username: String = if let Some(n) = &opt.login_name {
```

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

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

```rust
/// 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}"))?;
        if !path.is_absolute() {
```

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

```
223     login_name: Option<String>,
224     /// 秘密キーファイル名
225     #[arg(short, long)]
226     identity: Option<PathBuf>,
227     /// ポート番号
228     #[arg(short, long, default_value_t = 22)]
229     port: u16,
230 }
231 
232 /// 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。
233 fn exist_dir(s: &str) -> Result<String, String> {
234     match std::fs::read_dir(s) {
235         Ok(mut dir) => match dir.next() {
236             None => Ok(s.to_string()),
237             Some(_) => Err("マウント先ディレクトリが空ではありません".to_string()),
238         },
239         Err(e) => match e.kind() {
240             std::io::ErrorKind::NotFound => {
241                 Err("マウント先ディレクトリが存在しません。".to_string())
242             }
243             _ => Err("計り知れないエラーです。".to_string()),
244         },
245     }
246 }
247 
248 /// コマンドラインの接続先ホスト情報
249 #[derive(Clone, Debug, PartialEq)]
250 struct RemoteName {
251     /// ユーザー名
252     user: Option<String>,
253     /// ホスト名  または  IPアドレス
254     host: String,
255     /// 接続先パス
256     path: Option<std::path::PathBuf>,
257 }
258 
259 impl std::fmt::Display for RemoteName {
260     fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
261         let s = format!("<{:?}><{:?}><{:?}>", &self.user, &self.host, &self.path);
262         s.fmt(f)
263     }
264 }
265 
266 impl std::str::FromStr for RemoteName {
267     type Err = String;
```

```rust
    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()
    }

    #[test]
    fn test_from_str_remotename() {
        use std::path::Path;
        let s = "mito@reterminal.local:/home/mito";
```

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

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

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, ErrorCode};
use std::{
    ffi::OsStr,
    path::{Path, PathBuf},
    time::{SystemTime,Duration, UNIX_EPOCH},
    io::{Seek, Read, Write},
    collections::HashMap,
};

pub struct Sshfs {
    _session: Session,
    sftp: Sftp,
    inodes: Inodes,
    fhandls: Fhandles,
    _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,
            fhandls: Fhandles::new(),
            _top_path: top_path,
        }
    }

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

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

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

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

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

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

```
            }
        };
    }

    fn readlink(&mut self, _req: &Request<'_>, ino: u64, reply: ReplyData) {
        let Some(path) = self.inodes.get_path(ino) else {
            reply.error(libc::ENOENT);
            return;
        };
        match self.sftp.readlink(&path) {
            Ok(p) => reply.data(p.as_os_str().to_str().unwrap().as_bytes()),
            Err(e) => reply.error(Error::from(e).0),
        }
    }

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

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

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

    fn release(
            &mut self,
            _req: &Request<'_>,
            _ino: u64,
```

```
            fh: u64,
            _flags: i32,
            _lock_owner: Option<u64>,
            _flush: bool,
            reply: fuser::ReplyEmpty,
        ) {
        self.fhandls.del_file(fh);
        reply.ok();
    }

    fn read(
        &mut self,
        _req: &Request,
        _ino: u64,
        fh: u64,
        offset: i64,
        size: u32,
        _flags: i32,
        _lock_owner: Option<u64>,
        reply: ReplyData,
    ) {
        let Some(file) = self.fhandls.get_file(fh) else {
            reply.error(libc::EINVAL);
            return;
        };

        if let Err(e) = file.seek(std::io::SeekFrom::Start(offset as u64)) {
            reply.error(Error::from(e).0);
            return;
        }
        let mut buff = Vec::<u8>::new();
        buff.resize(size as usize, 0u8);
        let mut read_size : usize = 0;
        while read_size < size as usize {
            match file.read(&mut buff[read_size..]) {
                Ok(s) => {
                   if s == 0 {break;};
                   read_size += s;
                }
                Err(e) => {
                    reply.error(Error::from(e).0);
                    return;
                }
            }
        }
```

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

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

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

    fn mkdir(
            &mut self,
            req: &Request<'_>,
            parent: u64,
```

```
355             name: &OsStr,
356             mode: u32,
357             umask: u32,
358             reply: ReplyEntry,
359         ) {
360         let Some(mut path) = self.inodes.get_path(parent) else {
361             reply.error(libc::ENOENT);
362             return;
363         };
364         path.push(name);
365 
366         let mode = (mode & (!umask) & 0o777) as i32;
367 
368         match self.sftp.mkdir(&path, mode) {
369             Ok(_) => {
370                 match self.getattr_from_ssh2(&path, req.uid(), req.gid()) {
371                     Ok(attr) => reply.entry(&Duration::from_secs(1), &attr, 0),
372                     Err(e) => reply.error(e.0),
373                 }
374             }
375             Err(e) => reply.error(Error::from(e).0),
376         }
377     }
378 
379     fn rmdir(&mut self, _req: &Request<'_>, parent: u64, name: &OsStr, reply: fuser::ReplyEmpty) {
380         let Some(mut path) = self.inodes.get_path(parent) else {
381             reply.error(libc::ENOENT);
382             return;
383         };
384         path.push(name);
385         match self.sftp.rmdir(&path) {
386             Ok(_) => {
387                 self.inodes.del_inode_with_path(&path);
388                 reply.ok()
389             }
390             Err(e) => {
391                 if e.code() == ErrorCode::Session(-31) {
392                     // ssh2ライブラリの返すエラーが妙。置換しておく。
393                     reply.error(libc::ENOTEMPTY);
394                 } else {
395                     reply.error(Error::from(e).0)
396                 }
397             }
398         }
399     }
```

```
400
401     fn setattr(
402             &mut self,
403             req: &Request<'_>,
404             ino: u64,
405             mode: Option<u32>,
406             _uid: Option<u32>,
407             _gid: Option<u32>,
408             size: Option<u64>,
409             atime: Option<fuser::TimeOrNow>,
410             mtime: Option<fuser::TimeOrNow>,
411             _ctime: Option<std::time::SystemTime>,
412             _fh: Option<u64>,
413             _crtime: Option<std::time::SystemTime>,
414             _chgtime: Option<std::time::SystemTime>,
415             _bkuptime: Option<std::time::SystemTime>,
416             _flags: Option<u32>,
417             reply: ReplyAttr,
418         ) {
419         let stat = ssh2::FileStat{
420             size,
421             uid: None,
422             gid: None,
423             perm: mode,
424             atime: atime.map(|t|
425                 Self::conv_timeornow2systemtime(&t).duration_since(UNIX_EPOCH).unwrap().as_secs()
426             ),
427             mtime: mtime.map(|t|
428                 Self::conv_timeornow2systemtime(&t).duration_since(UNIX_EPOCH).unwrap().as_secs()
429             ),
430         };
431         let Some(filename) = self.inodes.get_path(ino) else {
432              reply.error(ENOENT);
433              return;
434         };
435         match self.sftp.setstat(&filename, stat) {
436             Ok(_) => {
437                 let stat = self.getattr_from_ssh2(&filename, req.uid(), req.gid());
438                 match stat {
439                     Ok(s) => reply.attr(&Duration::from_secs(1), &s),
440                     Err(e) => reply.error(e.0),
441                 }
442             },
443             Err(e) => reply.error(Error::from(e).0),
444         }
```

```
445     }
446
447     fn rename(
448             &mut self,
449             _req: &Request<'_>,
450             parent: u64,
451             name: &OsStr,
452             newparent: u64,
453             newname: &OsStr,
454             flags: u32,
455             reply: fuser::ReplyEmpty,
456         ) {
457         let Some(mut old_path) = self.inodes.get_path(parent) else {
458             reply.error(libc::ENOENT);
459             return;
460         };
461         old_path.push(name);
462
463         let Some(mut new_path) = self.inodes.get_path(newparent) else {
464             reply.error(libc::ENOENT);
465             return;
466         };
467         new_path.push(newname);
468
469         let mut rename_flag = ssh2::RenameFlags::NATIVE;
470         if flags & libc::RENAME_EXCHANGE != 0 { rename_flag.insert(ssh2::RenameFlags::ATOMIC); }
471         if flags & libc::RENAME_NOREPLACE == 0 { // rename の OVERWRITE が効いてない。手動で消す。
472             if let Ok(stat) = self.sftp.lstat(&new_path) {
473                 if stat.is_dir() {
474                     if let Err(e) = self.sftp.rmdir(&new_path) {
475                         reply.error(Error::from(e).0);
476                         return;
477                     }
478                 } else if let Err(e) = self.sftp.unlink(&new_path) {
479                     reply.error(Error::from(e).0);
480                     return;
481                 }
482                 self.inodes.del_inode_with_path(&new_path);
483             }
484         }
485
486         match self.sftp.rename(&old_path, &new_path, Some(rename_flag)) {
487             Ok(_) => {
488                 self.inodes.rename(&old_path, &new_path);
489                 reply.ok();
```

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

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

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

    /// path で指定された inode を生成し、登録する。
    /// すでに path の登録が存在する場合、追加はせず、登録済みの inode を返す。
    fn add(&mut self, path: &Path) -> u64 {
        match self.get_inode(path){
            Some(i) => i,
            None => {
                self.max_inode += 1;
                self.list.insert(self.max_inode, path.into());
                self.max_inode
            }
        }
    }

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

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

    /// inodes から、inode の登録を削除する
```

```
535     fn del_inode(&mut self, inode: u64) -> Option<u64> {
536         self.list.remove(&inode).map(|_| inode)
537     }
538 
539     /// inodes から、path の名前の登録を削除する
540     fn del_inode_with_path(&mut self, path: &Path) -> Option<u64> {
541         self.get_inode(path).map(|ino| self.del_inode(ino).unwrap())
542     }
543 
544     /// 登録されている inode の path を変更する。
545     /// old_path が存在しなければ、なにもしない。
546     fn rename(&mut self, old_path: &Path, new_path: &Path)  {
547         let Some(ino) = self.get_inode(old_path) else {
548             return;
549         };
550         if let Some(val) = self.list.get_mut(&ino) {
551             *val = new_path.into();
552         }
553     }
554 }
555 
556 struct Fhandles {
557     list: HashMap<u64, ssh2::File>,
558     next_handle: u64,
559 }
560 
561 impl Fhandles {
562     fn new() -> Self {
563         Self {
564             list: HashMap::new(),
565             next_handle: 0,
566         }
567     }
568 
569     fn add_file(&mut self, file: ssh2::File) -> u64 {
570         let handle = self.next_handle;
571         self.list.insert(handle, file);
572         self.next_handle += 1;
573         handle
574     }
575 
576     fn get_file(&mut self, fh: u64) -> Option<&mut ssh2::File> {
577         self.list.get_mut(&fh)
578     }
579 
```

```
580     fn del_file(&mut self, fh: u64) {
581         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。
582         // ハンドルの再利用のため、次回ハンドルを調整
583         match self.list.keys().max() {
584             Some(&i) => self.next_handle = i + 1,
585             None => self.next_handle = 0,
586         }
587     }
588 
589 }
590 
591 #[derive(Debug, Clone, Copy)]
592 struct Error(i32);
593 
594 impl From<ssh2::Error> for Error {
595     fn from(value : ssh2::Error) -> Self {
596         let eno = match value.code() {
597             ssh2::ErrorCode::Session(_) => libc::ENXIO,
598             ssh2::ErrorCode::SFTP(i) =>
599                 match i {
600                     // libssh2の libssh2_sftp.hにて定義されている。
601                     2 => libc::ENOENT,  // NO_SUCH_FILE
602                     3 => libc::EACCES,  // permission_denied
603                     4 => libc::EIO,     // failure
604                     5 => libc::ENODEV,  // bad message
605                     6 => libc::ENXIO,   // no connection
606                     7 => libc::ENETDOWN,// connection lost
607                     8 => libc::ENODEV,  // unsported
608                     9 => libc::EBADF,   // invalid handle
609                     10 => libc::ENOENT, //no such path
610                     11 => libc::EEXIST, // file already exists
611                     12 => libc::EACCES, // write protected
612                     13 => libc::ENXIO,  // no media
613                     14 => libc::ENOSPC, // no space on filesystem
614                     15 => libc::EDQUOT, // quota exceeded
615                     16 => libc::ENODEV, // unknown principal
616                     17 => libc::ENOLCK, // lock conflict
617                     18 => libc::ENOTEMPTY, // dir not empty
618                     19 => libc::ENOTDIR,// not a directory
619                     20 => libc::ENAMETOOLONG,// invalid file name
620                     21 => libc::ELOOP, // link loop
621                     _ => 0,
622                 }
623         };
624         Self(eno)
```

```
625     }
626 }
627 
628 impl From<std::io::Error> for Error {
629     fn from(value : std::io::Error) -> Self {
630         use std::io::ErrorKind::*;
631         let eno = match value.kind() {
632             NotFound => libc::ENOENT,
633             PermissionDenied => libc::EACCES,
634             ConnectionRefused => libc::ECONNREFUSED,
635             ConnectionReset => libc::ECONNRESET,
636             ConnectionAborted => libc::ECONNABORTED,
637             NotConnected => libc::ENOTCONN,
638             AddrInUse => libc::EADDRINUSE,
639             AddrNotAvailable => libc::EADDRNOTAVAIL,
640             BrokenPipe => libc::EPIPE,
641             AlreadyExists => libc::EEXIST,
642             WouldBlock => libc::EWOULDBLOCK,
643             InvalidInput => libc::EINVAL,
644             InvalidData => libc::EILSEQ,
645             TimedOut => libc::ETIMEDOUT,
646             WriteZero => libc::EIO,
647             Interrupted => libc::EINTR,
648             Unsupported => libc::ENOTSUP,
649             UnexpectedEof => libc::EOF,
650             OutOfMemory => libc::ENOMEM,
651             _ => 0,
652         };
653         Self(eno)
654     }
655 }
656 
657 #[cfg(test)]
658 mod inode_test {
659     use super::Inodes;
660     use std::path::Path;
661 
662     #[test]
663     fn inode_add_test() {
664         let mut inodes = Inodes::new();
665         assert_eq!(inodes.add(Path::new("")), 1);
666         assert_eq!(inodes.add(Path::new("test")), 2);
667         assert_eq!(inodes.add(Path::new("")), 1);
668         assert_eq!(inodes.add(Path::new("test")), 2);
669         assert_eq!(inodes.add(Path::new("test3")), 3);
```

```
670         assert_eq!(inodes.add(Path::new("/test")), 4);
671         assert_eq!(inodes.add(Path::new("test/")), 2);
672     }
673 
674     fn make_inodes() -> Inodes {
675         let mut inodes = Inodes::new();
676         inodes.add(Path::new(""));
677         inodes.add(Path::new("test"));
678         inodes.add(Path::new("test2"));
679         inodes.add(Path::new("test3/"));
680         inodes
681     }
682 
683     #[test]
684     fn inodes_get_inode_test() {
685         let inodes = make_inodes();
686         assert_eq!(inodes.get_inode(Path::new("")), Some(1));
687         assert_eq!(inodes.get_inode(Path::new("test4")), None);
688         assert_eq!(inodes.get_inode(Path::new("/test")), None);
689         assert_eq!(inodes.get_inode(Path::new("test3")), Some(4));
690     }
691 
692     #[test]
693     fn inodes_get_path_test() {
694         let inodes = make_inodes();
695         assert_eq!(inodes.get_path(1), Some(Path::new("").into()));
696         assert_eq!(inodes.get_path(3), Some(Path::new("test2").into()));
697         assert_eq!(inodes.get_path(5), None);
698         assert_eq!(inodes.get_path(3), Some(Path::new("test2/").into()));
699     }
700 
701     #[test]
702     fn inodes_rename() {
703         let mut inodes = make_inodes();
704         let old = Path::new("test2");
705         let new = Path::new("new_test");
706         let ino = inodes.get_inode(old).unwrap();
707         inodes.rename(old, new);
708         assert_eq!(inodes.get_path(ino), Some(new.into()));
709 
710         let mut inodes = make_inodes();
711         let inodes2 = make_inodes();
712         inodes.rename(Path::new("nai"), Path::new("kawattenai"));
713         assert_eq!(inodes.list, inodes2.list);
714     }
```

```
715
716
717
718 }
```