sshmount ソースリスト

美都

2023 年 1 月 8 日

目次

1　メインモジュール　main.rs

```rust
mod ssh_filesystem;

use clap::Parser;
use dns_lookup::lookup_host;
use log::debug;
use ssh2::Session;
use std::net::TcpStream;

fn main() {
    let opt = Opt::parse();
    env_logger::init();
    let options = vec![fuser::MountOption::FSName("sshfs".to_string())];

    println!("\"{}\"をマウントする予定", opt.remote);
    // 今は、固定接続先に、固定接続
    let address = lookup_host("reterminal.local").unwrap();
    if address.is_empty() {
        panic!("not found");
    }
    let socketaddr = std::net::SocketAddr::from((address[0], 22));
    debug!("接続先: {:?}", socketaddr);
    let tcp = TcpStream::connect(socketaddr).unwrap();
    let mut ssh = Session::new().unwrap();
    ssh.set_tcp_stream(tcp);
    ssh.handshake().unwrap();
    let key = std::path::Path::new("/home/mito/.ssh/id_rsa");
    ssh.userauth_pubkey_file("mito", None, key, None).unwrap();
    let fs = ssh_filesystem::Sshfs::new(ssh);

    fuser::mount2(fs, opt.mount_point, &options).unwrap();
}

/// コマンドラインオプション
#[derive(Parser)]
#[command(author, version, about)]
struct Opt {
    /// 接続先のアドレスまたは DNS 名
    remote: String,
    /// マウント先のパス
    #[arg(value_parser = exist_dir)]
    mount_point: String,
}
```

```rust

/// 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()),
        },
    }
}

#[test]
fn verify_cli() {
    use clap::CommandFactory;
    Opt::command().debug_assert()
}
```

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::debug;
use ssh2::{Session, Sftp};
use std::{
    cmp::min,
    ffi::OsStr,
    path::{Path, PathBuf},
    time::{Duration, UNIX_EPOCH},
};

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

impl Sshfs {
    pub fn new(session: Session) -> Self {
        let mut inodes = Inodes::new();
        inodes.add(Path::new(""));
        let top_path: PathBuf = Path::new("/home/mito").into();
        let sftp = session.sftp().unwrap();
        Self {
            _session: session,
            sftp,
            inodes,
            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 mut full_path = self.top_path.clone();
        full_path.push(path);
        let attr_ssh2 = self.sftp.lstat(full_path.as_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)),
        }
    }
}

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;
```

```
88          };
89          path.push(Path::new(name));
90          match self.getattr_from_ssh2(&path, req.uid(), req.gid()) {
91              Ok(attr) => reply.entry(&Duration::from_secs(1), &attr, 0),
92              Err(e) => {
93                  debug!("[lookup] アトリビュートの取得に失敗 path={:?}", &path);
94                  reply.error(e.0);
95              }
96          };
97      }
98
99      fn getattr(&mut self, req: &Request, ino: u64, reply: ReplyAttr) {
100         let Some(path) = self.inodes.get_path(ino) else {
101             reply.error(ENOENT);
102             return;
103         };
104         match self.getattr_from_ssh2(&path, req.uid(), req.gid()) {
105             Ok(attr) => reply.attr(&Duration::from_secs(1), &attr),
106             Err(e) => reply.error(e.0),
107         };
108     }
109
110     fn readdir(
111         &mut self,
112         _req: &Request,
113         ino: u64,
114         _fh: u64,
115         offset: i64,
116         mut reply: ReplyDirectory,
117     ) {
118         let Some(path) = self.inodes.get_path(ino) else {
119             reply.error(libc::ENOENT);
120             return;
121         };
122         let mut full_path = self.top_path.clone();
123         full_path.push(path);
124         debug!("[readdir] inode={}, paht={:?}, offset={}",ino, &full_path, offset);
125         match self.sftp.readdir(&full_path) {
126             Ok(dir) => {
127                 let mut i = offset+1;
128                 for f in dir.iter().skip(offset as usize) {
129                     let path : PathBuf = f.0.strip_prefix(&self.top_path).unwrap().into();
130                     let ino = self.inodes.add(&path);
131                     let name = path.file_name().unwrap();
132                     let filetype = match Self::conv_file_kind_ssh2fuser(&f.1.file_type()) {
```

```
                        Ok(t) => t,
                        Err(e) => {
                            debug!("[readdir] ファイルタイプ解析失敗: inode={}, name={:?}", ino,
                            ↪  name);
                            reply.error(e.0);
                            return;
                        }
                    };
                    if reply.add(ino, i, filetype, name) {break;}
                    i += 1;
                }
                reply.ok();
            }
            Err(e) => {
                debug!("ssh2::readdir 内でエラー発生");
                reply.error(Error::from(e).0);
            }
        };
    }

    fn read(
        &mut self,
        _req: &Request,
        ino: u64,
        _fh: u64,
        offset: i64,
        size: u32,
        _flags: i32,
        _lock_owner: Option<u64>,
        reply: ReplyData,
    ) {
        if ino == 2 {
            let offset = offset as usize;
            let size = size as usize;
            let ret_str = &CONTENTS[offset..min(CONTENTS.len() - 1, offset + size - 1)];
            reply.data(ret_str.as_bytes());
        } else {
            reply.error(ENOENT);
        }
    }
}

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

```
177     max_inode: u64,
178 }
179 
180 impl Inodes {
181     /// Inode を生成する
182     fn new() -> Self {
183         Self {
184             list: std::collections::HashMap::new(),
185             max_inode: 0,
186         }
187     }
188 
189     /// path で指定された inode を生成し、登録する。
190     /// 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。
191     fn add(&mut self, path: &Path) -> u64 {
192         match self.get_inode(path){
193             Some(i) => i,
194             None => {
195                 self.max_inode += 1;
196                 self.list.insert(self.max_inode, path.into());
197                 self.max_inode
198             }
199         }
200     }
201 
202     /// path から inode を取得する
203     fn get_inode(&self, path: &Path) -> Option<u64> {
204         self.list.iter().find(|(_, p)| path == *p).map(|(i, _)| *i)
205     }
206 
207     /// inode から path を取得する
208     fn get_path(&self, inode: u64) -> Option<PathBuf> {
209         self.list.get(&inode).map(|p| (*p).clone())
210     }
211 }
212 
213 #[derive(Debug, Clone, Copy)]
214 struct Error(i32);
215 
216 impl From<ssh2::Error> for Error {
217     fn from(value : ssh2::Error) -> Self {
218         let eno = match value.code() {
219             ssh2::ErrorCode::Session(_) => libc::ENXIO,
220             ssh2::ErrorCode::SFTP(i) =>
221                 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)
    }
}

const CONTENTS: &str = "Hello fuse!\n";

#[cfg(test)]
mod inode_test {
    use super::Inodes;
    use std::path::Path;

    #[test]
    fn inode_add_test() {
        let mut inodes = Inodes::new();
        assert_eq!(inodes.add(Path::new("")), 1);
        assert_eq!(inodes.add(Path::new("test")), 2);
        assert_eq!(inodes.add(Path::new("")), 1);
        assert_eq!(inodes.add(Path::new("test")), 2);
        assert_eq!(inodes.add(Path::new("test3")), 3);
        assert_eq!(inodes.add(Path::new("/test")), 4);
        assert_eq!(inodes.add(Path::new("test/")), 2);
```

```
267     }
268
269     fn make_inodes() -> Inodes {
270         let mut inodes = Inodes::new();
271         inodes.add(Path::new(""));
272         inodes.add(Path::new("test"));
273         inodes.add(Path::new("test2"));
274         inodes.add(Path::new("test3/"));
275         inodes
276     }
277
278     #[test]
279     fn inodes_get_inode_test() {
280         let inodes = make_inodes();
281         assert_eq!(inodes.get_inode(Path::new("")), Some(1));
282         assert_eq!(inodes.get_inode(Path::new("test4")), None);
283         assert_eq!(inodes.get_inode(Path::new("/test")), None);
284         assert_eq!(inodes.get_inode(Path::new("test3")), Some(4));
285     }
286
287     #[test]
288     fn inodes_get_path_test() {
289         let inodes = make_inodes();
290         assert_eq!(inodes.get_path(1), Some(Path::new("").into()));
291         assert_eq!(inodes.get_path(3), Some(Path::new("test2").into()));
292         assert_eq!(inodes.get_path(5), None);
293         assert_eq!(inodes.get_path(3), Some(Path::new("test2/").into()));
294     }
295 }
```