THE FILE TRANSFER PROTOCOL
RFC172

田端 利宏

1999 年 2 月 2 日

1　概要

ftp とは、ユーザレベルでのホストコンピュータ間のファイル転送のためのプロトコルである。RFC171 では、FTP は、データの転送のために、RFC171 で述べられている data transfer protocol を使用すると書かれている。現在の FTP は、RFC959 に基づき実装が行われているので、RFC959 のについての説明を中心に行なう。

ftp のサーバーとクライアントの接続の様子を図 1に示す。

USER-PI と SERVER-PI の間は、コマンドとその応答のやりとりが行なわれ、この接続は、the Telnet Protocol を利用している。FTP commands は、user-FTP プロセスから、server-FTP プロセスへの制御情報のながれをなす命令の集合である。DTP とは、the data transfer process の略で、DTP 間のデータコネクションを確立し、管理する。a full duplex connection でデータは転送される。PI とは、the protocol interpreter の略で、user と server のプロトコルは、user-PI と server-PI で別の役割を持つように実装されている。

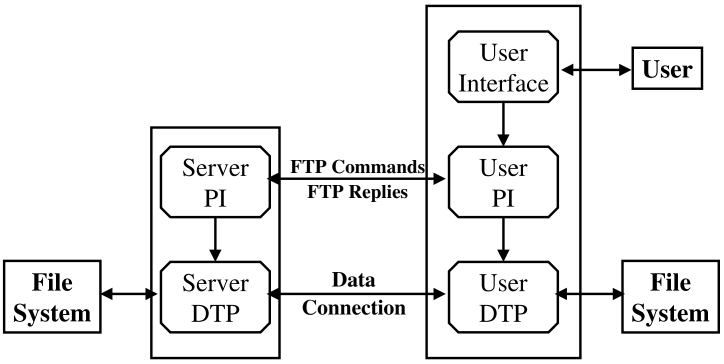

図 1: ftp のサーバーとクライアントの接続

2 ftpd

実装は、ftp://www.jp.freebsd.org/pub/FreeBSD/FreeBSD-current/src/libexec/ftpd/ のソースコードを調査した。

ftpd の親プロセス動作を図 2で示す。

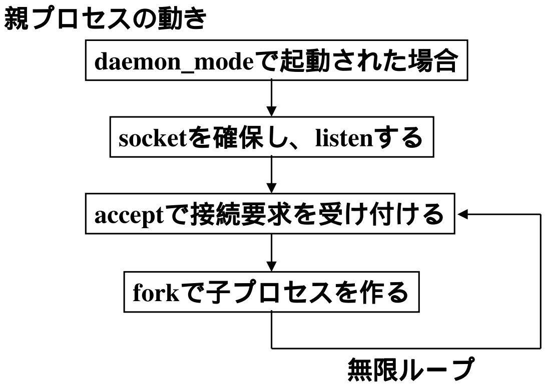

図 2: ftpd の処理

daemon_mode で起動した場合に、サーバーの動作をする。まず、socket を確保し、listen してクライアントの接続を待つ。次に、接続を受け取ると fork で子プロセスを作り、クライアントの処理を子プロセスにかませる。親プロセスは、また listen し、次の接続を待機する処理を無限に繰り返す。

一方、子プロセスは、クライアントからの接続を受け、ユーザの認証を行ない、ユーザからのコマンドを受け処理結果を返す処理を、ユーザが接続を終了するまで繰り返す。

RFC765 では、クライアントから受け取るコマンドが定義されている。これらのコマンドを以下に示す。サーバーは、受け取ったコマンドを実行したあとに、応答をクライアントに返す。各コマンドは、以下の tab という構造体により定義されている。*name は、コマンドの名前である。implemented は、コマンドが実装されているとき 1 である。*help は、表示するヘルプの内容である。ftpd では、tab の構造体配列で各コマンドを定義している。

ftpd は、クライアントから、cmdtab[] で定義されているコマンドが送られてくると、各コマンドに対応した処理を行ない、その結果をクライアントに返す処理を行なう。

```
   struct tab {
     char *name;
     short token;
     short state;
     short implemented; /* 1 if command is implemented */
     char *help;
};

   struct tab cmdtab[] = { /* In order defined in RFC 765 */
     { ”USER”, USER, STR1, 1, ”¡sp¿ username” },
```

```
    { ”PASS”, PASS, ZSTR1, 1, ”¡sp¿ password” },
    { ”ACCT”, ACCT, STR1, 0, ”(specify account)” },
    { ”SMNT”, SMNT, ARGS, 0, ”(structure mount)” },
    { ”REIN”, REIN, ARGS, 0, ”(reinitialize server state)” },
    { ”QUIT”, QUIT, ARGS, 1, ”(terminate service)”, },
    { ”PORT”, PORT, ARGS, 1, ”¡sp¿ b0, b1, b2, b3, b4” },
    { ”PASV”, PASV, ARGS, 1, ”(set server in passive mode)” },
    { ”TYPE”, TYPE, ARGS, 1, ”¡sp¿ [ A — E — I — L ]” },
    { ”STRU”, STRU, ARGS, 1, ”(specify file structure)” },
    { ”MODE”, MODE, ARGS, 1, ”(specify transfer mode)” },
    { ”RETR”, RETR, STR1, 1, ”¡sp¿ file-name” },
    { ”STOR”, STOR, STR1, 1, ”¡sp¿ file-name” },
    { ”APPE”, APPE, STR1, 1, ”¡sp¿ file-name” },
    { ”MLFL”, MLFL, OSTR, 0, ”(mail file)” },
    { ”MAIL”, MAIL, OSTR, 0, ”(mail to user)” },
    { ”MSND”, MSND, OSTR, 0, ”(mail send to terminal)” },
    { ”MSOM”, MSOM, OSTR, 0, ”(mail send to terminal or mailbox)” },
    { ”MSAM”, MSAM, OSTR, 0, ”(mail send to terminal and mailbox)” },
    { ”MRSQ”, MRSQ, OSTR, 0, ”(mail recipient scheme question)” },
    { ”MRCP”, MRCP, STR1, 0, ”(mail recipient)” },
    { ”ALLO”, ALLO, ARGS, 1, ”allocate storage (vacuously)” },
    { ”REST”, REST, ARGS, 1, ”¡sp¿ offset (restart command)” },
    { ”RNFR”, RNFR, STR1, 1, ”¡sp¿ file-name” },
    { ”RNTO”, RNTO, STR1, 1, ”¡sp¿ file-name” },
    { ”ABOR”, ABOR, ARGS, 1, ”(abort operation)” },
    { ”DELE”, DELE, STR1, 1, ”¡sp¿ file-name” },
    { ”CWD”, CWD, OSTR, 1, ”[ ¡sp¿ directory-name ]” },
    { ”XCWD”, CWD, OSTR, 1, ”[ ¡sp¿ directory-name ]” },
    { ”LIST”, LIST, OSTR, 1, ”[ ¡sp¿ path-name ]” },
    { ”NLST”, NLST, OSTR, 1, ”[ ¡sp¿ path-name ]” },
    { ”SITE”, SITE, SITECMD, 1, ”site-cmd [ ¡sp¿ arguments ]” },
    { ”SYST”, SYST, ARGS, 1, ”(get type of operating system)” },
    { ”STAT”, STAT, OSTR, 1, ”[ ¡sp¿ path-name ]” },
    { ”HELP”, HELP, OSTR, 1, ”[ ¡sp¿ ¡string¿ ]” },
    { ”NOOP”, NOOP, ARGS, 1, ”” },
    { ”MKD”, MKD, STR1, 1, ”¡sp¿ path-name” },
    { ”XMKD”, MKD, STR1, 1, ”¡sp¿ path-name” },
    { ”RMD”, RMD, STR1, 1, ”¡sp¿ path-name” },
    { ”XRMD”, RMD, STR1, 1, ”¡sp¿ path-name” },
    { ”PWD”, PWD, ARGS, 1, ”(return current directory)” },
    { ”XPWD”, PWD, ARGS, 1, ”(return current directory)” },
    { ”CDUP”, CDUP, ARGS, 1, ”(change to parent directory)” },
    { ”XCUP”, CDUP, ARGS, 1, ”(change to parent directory)” },
    { ”STOU”, STOU, STR1, 1, ”¡sp¿ file-name” },
    { ”SIZE”, SIZE, OSTR, 1, ”¡sp¿ path-name” },
    { ”MDTM”, MDTM, OSTR, 1, ”¡sp¿ path-name” },
    { NULL, 0, 0, 0, 0 }
};
```

3 ftp

ftp の実装は、BSD/OS 3.1 の ftp コマンドを調査した。ftp の処理の流れを図 3に示す。ftp コマンドは、最初にサーバーに接続し、ユーザの認証を行なう。認証に成功すると、ユーザからのコマンドを受け付ける。このコマンドは、RFC には定義されておらず、その内容は ftp のクライアントの実装に依存する。

BSD/OS 3.1 の ftp コマンドのコマンドを登録した構造体配列を以下に示す。この構造体には、最初にコマンド名が登録されている。そして、構造体の最後には、そのコマンドの関数の実体の関数名が登録されている。

ユーザから、コマンドが入力された時に、入力されたコマンドと一致するものが、この構造体配列にあるかを調べる。もし、一致するものがあれば、そのコマンドの関数を実行する。

これらのコマンドの関数は、2 章の ftpd についての説明で紹介した RFC765 で定義されたコマンドをサーバーに送る処理を行なっている。各コマンドは、RFC765 で定義されたコマンドを組み合わせて ftp のコマンドの機能を実現する。具体的な例を 4 章で説明する。

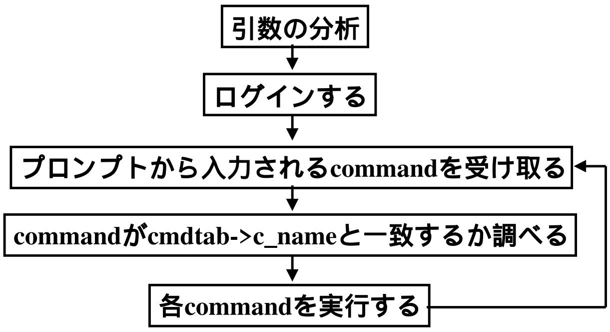

図 3: ftp の処理

```
struct cmd cmdtab[] = {
 { ”!”, shellhelp, 0, 0, 0, shell },
 { ”$”, domachelp, 1, 0, 0, domacro },
  { ”account”, accounthelp, 0, 1, 1, account},
 { ”append”, appendhelp, 1, 1, 1, put },
 { ”ascii”, asciihelp, 0, 1, 1, setascii },
 { ”bell”, beephelp, 0, 0, 0, setbell },
 { ”binary”, binaryhelp, 0, 1, 1, setbinary },
 { ”bye”, quithelp, 0, 0, 0, quit },
 { ”case”, casehelp, 0, 0, 1, setcase },
 { ”cd”, cdhelp, 0, 1, 1, cd },
 { ”cdup”, cduphelp, 0, 1, 1, cdup },
 { ”chmod”, chmodhelp, 0, 1, 1, do_chmod },
 { ”close”, disconhelp, 0, 1, 1, disconnect },
 { ”cr”, crhelp, 0, 0, 0, setcr },
 { ”delete”, deletehelp, 0, 1, 1, delete },
 { ”debug”, debughelp, 0, 0, 0, setdebug },
 { ”dir”, dirhelp, 1, 1, 1, ls },
 { ”disconnect”, disconhelp, 0, 1, 1, disconnect },
 { ”form”, formhelp, 0, 1, 1, setform },
 { ”get”, receivehelp, 1, 1, 1, get },
```

```
    { ”glob”, globhelp, 0, 0, 0, setglob },
    { ”hash”, hashhelp, 0, 0, 0, sethash },
    { ”help”, helphelp, 0, 0, , help },
    { ”idle”, idlehelp, 0, 1, 1, idle },
    { ”image”, binaryhelp, 0, 1, 1, setbinary },
    { ”lcd”, lcdhelp, 0, 0, 0, lcd },
    { ”ls”, lshelp, 1, 1, 1, ls },
    { ”macdef”, macdefhelp, 0, 0, 0, macdef },
    { ”mdelete”, mdeletehelp, 1, 1, 1, mdelete },
    { ”mdir”, mdirhelp, 1, 1, 1, mls },
    { ”mget”, mgethelp, 1, 1, 1, mget },
    { ”mkdir”, mkdirhelp, 0, 1, 1, makedir },
    { ”mls”, mlshelp, 1, 1, 1, mls },
    { ”mode”, modehelp, 0, 1, 1, setftmode },
    { ”modtime”, modtimehelp, 0, 1, 1, modtime },
    { ”mput”, mputhelp, 1, 1, 1, mput },
    { ”newer”, newerhelp, 1, 1, 1, newer },
    { ”nmap”, nmaphelp, 0, 0, 1, setnmap },
    { ”nlist”, nlisthelp, 1, 1, 1, ls },
    { ”ntrans”, ntranshelp, 0, 0, 1, setntrans },
    { ”open”, connecthelp, 0, 0, 1, setpeer },
    { ”passive”, passivehelp, 0, 0, 0, setpassive },
    { ”prompt”, prompthelp, 0, 0, 0, setprompt },
    { ”proxy”, proxyhelp, 0, 0, 1, doproxy },
    { ”sendport”, porthelp, 0, 0, 0, setport },
    { ”put”, sendhelp, 1, 1, 1, put },
    { ”pwd”, pwdhelp, 0, 1, 1, pwd },
    { ”quit”, quithelp, 0, 0, 0, quit },
    { ”quote”, quotehelp, 1, 1, 1, quote },
    { ”recv”, receivehelp, 1, 1, 1, get },
    { ”reget”, regethelp, 1, 1, 1, reget },
    { ”rstatus”, rmtstatushelp, 0, 1, 1, rmtstatus },
    { ”rhelp”, remotehelp, 0, 1, 1, rmthelp },
    { ”rename”, renamehelp, 0, 1, 1, renamefile },
    { ”reset”, resethelp, 0, 1, 1, reset },
    { ”restart”, restarthelp, 1, 1, 1, restart },
    { ”rmdir”, rmdirhelp, 0, 1, 1, removedir },
    { ”runique”, runiquehelp, 0, 0, 1, setrunique },
    { ”send”, sendhelp, 1, 1, 1, put },
    { ”site”, sitehelp, 0, 1, 1, site },
    { ”size”, sizecmdhelp, 1, 1, 1, sizecmd },
    { ”status”, statushelp, 0, 0, 1, status },
    { ”struct”, structhelp, 0, 1, 1, setstruct },
    { ”system”, systemhelp, 0, 1, 1, syst },
    { ”sunique”, suniquehelp, 0, 0, 1, setsunique },
    { ”tenex”, tenexhelp, 0, 1, 1, settenex },
    { ”trace”, tracehelp, 0, 0, 0, settrace },
    { ”type”, typehelp, 0, 1, 1, settype },
    { ”user”, userhelp, 0, 1, 1, user },
    { ”umask”, umaskhelp, 0, 1, 1, do_umask },
    { ”verbose”, verbosehelp, 0, 0, 0, setverbose },
    { ”?”, helphelp, 0, 0, 1, help },
    { 0 },
};
```

4　例 rename

図 4に、rename コマンドの例を示し、説明する。ユーザが、rename a.file b.file return と入力したとする。そこで、ftp のプログラムは、rename という文字列に一致するコマンドがあるか、構造体配列 cmdtab[] を検索する。rename と一致する名前のエントリを見つけるので、そこに登録されている renamefile という関数を実行する。

renamefile の中では、引数をチェックする。もし、引数が違っていたら、コマンドの使い方を表示してエラーで終了する。引数が正しければ、名前を変更するファイルを指定する RNFR (RENAME FROM) コマンドをサーバーに送る。

サーバーは、RNFR を受けとると、renamefrom コマンドを実行する。renamefrom コマンドの中で、stat 関数を実行しファイルが存在するかを調べる。その結果をクライアントに返す。

クライアントは、結果を受けとり、結果が CONTINUE だった場合に、RNTO(RENAME TO) コマンドをサーバーに送る。

サーバーでは、renamecmd 関数を実行する。その中で、rename 関数を実行し、成功すれば、ack(“RNTO ”); を実行し、処理が成功したことをクライアントに通知する。

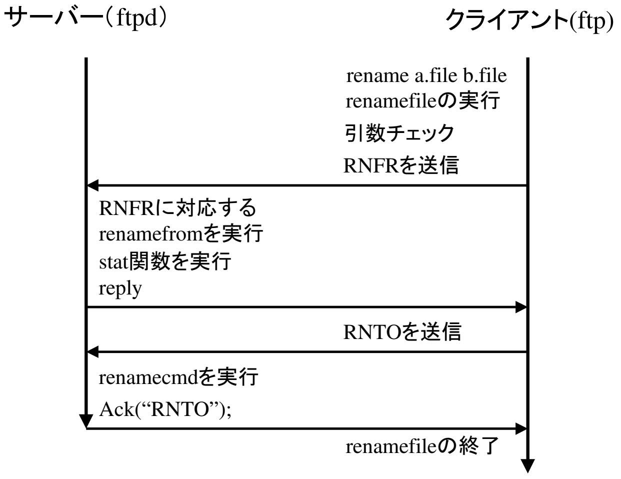

図 4: rename コマンドの処理