```
1     /*
2      * メールを受信する(POP)/socket 版
3      *
4      * Copyright 1998, Shuji Ishii, shuji@ccs.mt.nec.co.jp
5      */
6
7     #include <stdio.h>
8     #include <unistd.h>
9     #include <string.h>
10    #include <sys/types.h>
11    #include <sys/socket.h>
12    #include <netinet/in.h>
13    #include <netdb.h>
14    #include <pwd.h>
15
16    #define POP_RESULT_LEN (256)
17
18    typedef struct _popserver {
19            FILE *in, *out;         /* POPサーバ入出力用 */
20            char result[POP_RESULT_LEN]; /* 結果格納領域 */
21    } POPSERVER;
22
23    #define POP_OK (0)
24    #define POP_ERR (-1)
25
26    static int transaction(POPSERVER *pop, char *command);
27
28    /*
29     * pop サーバに接続する.
30     * 接続に成功すれば POPSERVER 構造体へのポインタを返す
31     * エラーの場合 NULL を返す
32     */
33    static POPSERVER *open_popserver(char *popserver)
34    {
35            struct hostent *hp;     /* ホストエントリ */
36            struct protoent *pp;    /* プロトコルエントリ */
37            struct servent *sp;     /* サービスエントリ */
38            int s = -1;             /* ソケット */
39            struct sockaddr_in sin; /* サーバ */
40            POPSERVER *pop = NULL;
41
42            /* pop サーバのエントリを取得する */
43            if ((hp = gethostbyname(popserver)) == NULL) {
44                    goto err;
45            }
46            /* tcp のエントリを取得する */
47            if ((pp = getprotobyname("tcp")) == NULL) {
48                    goto err;
49            }
50            /* pop3 サービスのエントリを取得する */
51            if ((sp = getservbyname("pop3", "tcp")) == NULL) {
52                    goto err;
53            }
54
55            if ((pop = (POPSERVER *)malloc(sizeof(POPSERVER))) == NULL) {
56                    goto err;
57            }
58
59            /* socket を作る */
60            if ((s = socket(PF_INET, SOCK_STREAM, pp->p_proto)) < 0) {
61                    goto err;
62            }
63
64            /* sockaddr_in 構造体を設定する */
65            memset(&sin, 0, sizeof(sin));
66            sin.sin_len = sizeof(sin);
```



```
67            sin.sin_family = PF_INET;
68            sin.sin_port = sp->s_port;
69            memcpy(&sin.sin_addr, *(hp->h_addr_list), sizeof(sin.sin_addr));
70
71            /* pop サーバに接続する */
72            if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
73                    goto err;
74            }
75
76            /* pop サーバへの通信路を fdopen で open する */
77            pop->in = fdopen(s, "r");
78            pop->out = fdopen(s, "w");
79
80            /* greeting message を読み込む */
81            transaction(pop, NULL);
82
83            return pop;
84
85        err:
86            if (pop) {
87                    free(pop);
88            }
89            if (s >= 0) {
90                    close(s);
91            }
92            return NULL;
93    }
94
95    /*
96     * pop サーバへの接続を閉じる
97     */
98    static void close_popserver(POPSERVER *pop)
99    {
100           fclose(pop->in);
101           fclose(pop->out);
102           free(pop);
103   }
104
105   /*
106    * コマンドを送信し、結果が +OK なら POP_OK を、それ以外なら POP_ERR を
107    * 返す
108    */
109   static int transaction(POPSERVER *pop, char *command)
110   {
111           if (command != NULL) {
112                   fprintf(pop->out, "%s\r\n", command);
113                   fflush(pop->out);
114           }
115           fgets(pop->result, POP_RESULT_LEN, pop->in);
116
117           return (pop->result[0] == '+') ? POP_OK : POP_ERR;
118   }
119
120   /*
121    * pop サーバに login する
122    */
123   static int pop_login(POPSERVER *pop, char *user, char *pass)
124   {
125           char buf[256];
126
127                                   /* USER コマンド */
128           snprintf(buf, sizeof(buf), "USER %s", user);
129           if (transaction(pop, buf) == POP_ERR) {
130                   return POP_ERR;
131           }
132
```

```
133                                 /* PASS コマンド */
134             snprintf(buf, sizeof(buf), "PASS %s", pass);
135             if (transaction(pop, buf) == POP_ERR) {
136                     return POP_ERR;
137             }
138
139             /*パスワードを消す*/
140             memset(buf, 0, sizeof(buf));
141
142             return POP_OK;
143     }
144
145     /*
146      * 現在のメッセージ数を返す
147      */
148     static int pop_msgcnt(POPSERVER *pop)
149     {
150             int msgcnt, ttlsize;
151
152                                 /* STAT コマンド */
153             if (transaction(pop, "STAT") == POP_ERR) {
154                     return -1;
155             }
156                                 /* 結果を解析 */
157             if (sscanf(pop->result, "+OK %d %d", &msgcnt, &ttlsize) != 2) {
158                     return -1;
159             }
160
161             return msgcnt;
162     }
163
164     /*
165      * 指定されたメッセージ番号を持つメールを out に指定された出力に
166      * 出力する
167      */
168     static int pop_retr(POPSERVER *pop, int msg, FILE *out)
169     {
170             char buf[256];
171
172                                 /* RETR コマンド */
173             snprintf(buf, sizeof(buf), "RETR %d", msg);
174             if (transaction(pop, buf) == POP_ERR) {
175                     return POP_ERR;
176             }
177
178             for (;;) {
179                     fgets(buf, sizeof(buf), pop->in);
180
181                     /* 行頭が '.'の時の処理 */
182                     if (buf[0] == '.') {
183                             if (buf[1] == '.') {
184                                     fprintf(out, "%s", buf + 1);
185                                     continue;
186                             } else if (buf[1] == '\r' && buf[2] == '\n') {
187                                     /* end of file */
188                                     break;
189                             }
190                     }
191                     fprintf(out, "%s", buf);
192             }
193
194             return POP_OK;
195     }
196
197     void main(int argc, char **argv)
198     {
```



```
199             POPSERVER *pop;
200             char *pass;
201             int msgcnt;
202             int i;
203
204             if ((pop = open_popserver("mailserver")) == NULL) {
205                     fprintf(stderr, "error: %s\n", *argv);
206                     exit(2);
207             }
208
209             if (pop_login(pop, "shuji", "password")) == POP_ERR) {
210                     fprintf(stderr, "error: pop_login\n");
211                     exit(3);
212             }
213
214             if ((msgcnt = pop_msgcnt(pop)) < 0) {
215                     fprintf(stderr, "error: pop_msgcnt\n");
216                     exit(4);
217             }
218
219             for (i = 1; i <= msgcnt; i++) {
220                     if (pop_retr(pop, i, stdout) == POP_ERR) {
221                             fprintf(stderr, "error: pop_retr\n");
222                             break;
223                     }
224             }
225
226             close_popserver(pop);
227             exit(0);
228     }
229
230     /* ----- end of fetchmail-socket.c ----- */
```

```
1     /*
2      * メールを送信する(SMTP)/socket 版
3      *
4      * Copyright 1998, Shuji Ishii, shuji@ccs.mt.nec.co.jp
5      */
6
7     #include <stdio.h>
8     #include <unistd.h>
9     #include <string.h>
10    #include <sys/types.h>
11    #include <sys/socket.h>
12    #include <netinet/in.h>
13    #include <netdb.h>
14
15    #define SMTP_RESULT_LEN (256)
16
17    typedef struct _smtpserver {
18            FILE *in, *out;         /* サーバ入出力用 */
19            char result[SMTP_RESULT_LEN]; /* 結果格納領域 */
20    } SMTPSERVER;
21
22    typedef struct _mail {
23            char **headers;         /* ヘッダ */
24            FILE *body;             /* 本文 */
25    } MAIL;
26
27    #define SMTP_OK (0)
28    #define SMTP_ERR (-1)
29
30    static int transaction(SMTPSERVER *smtp, char *command, MAIL *mail);
31
32    /*
33     * SMTP サーバに接続する
34     * 接続に成功すれば SMTPSERVER 構造体へのポインタを返す
35     * エラーの場合 NULL を返す
36     */
37    static SMTPSERVER *open_smtpserver(char *smtpserver)
38    {
39            struct hostent *hp;     /* ホストエントリ */
40            struct protoent *pp;    /* プロトコルエントリ */
41            struct servent *sp;     /* サービスエントリ */
42            int s = -1;             /* ソケット */
43            struct sockaddr_in sin; /* サーバ */
44            SMTPSERVER *smtp = NULL;
45            char buf[128];
46
47            /* smtp サーバのエントリを取得する */
48            if ((hp = gethostbyname(smtpserver)) == NULL) {
49                    goto err;
50            }
51            /* tcp のエントリを取得する */
52            if ((pp = getprotobyname("tcp")) == NULL) {
53                    goto err;
54            }
55            /* smtp サービスのエントリを取得する */
56            if ((sp = getservbyname("smtp", "tcp")) == NULL) {
57                    goto err;
58            }
59
60            if ((smtp = (SMTPSERVER *)malloc(sizeof(SMTPSERVER))) == NULL) {
61                    goto err;
62            }
63
64            /* socket を作る */
65            if ((s = socket(PF_INET, SOCK_STREAM, pp->p_proto)) < 0) {
66                    goto err;
67            }
68
69            /* sockaddr_in 構造体を設定する */
70            memset(&sin, 0, sizeof(sin));
71            sin.sin_len = sizeof(sin);
72            sin.sin_family = PF_INET;
73            sin.sin_port = sp->s_port;
74            memcpy(&sin.sin_addr, *(hp->h_addr_list), sizeof(sin.sin_addr));
75
76            /* smtp サーバに接続する */
77            if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
78                    goto err;
79            }
80
81            /* smtp サーバへの通信路を fdopen で open する */
82            smtp->in = fdopen(s, "r");
83            smtp->out = fdopen(s, "w");
84
85            /* greeting message を読み込む */
86            if (transaction(smtp, NULL, NULL) != SMTP_OK) {
87                    fprintf(stderr, "open_smtpserver: %s\n", smtp->result);
88                    goto err;
89            }
90
91            return smtp;
92
93        err:
94            if (smtp) {
95                    free(smtp);
96            }
97            if (s >= 0) {
98                    close(s);
99            }
100           return NULL;
101   }
102
103   /*
104    * smtp サーバへの接続を閉じる
105    */
106   static void close_smtpserver(SMTPSERVER *smtp)
107   {
108           fclose(smtp->in);
109           fclose(smtp->out);
110           free(smtp);
111   }
112
113   /*
114    * コマンドを送信し、結果が OK なら SMTP_OK を、それ以外なら SMTP_ERR を
115    * 返す
116    *
117    * 2xx: OK
118    * 5xx: ERR
119    * 3xx: need data
120    */
121   static int transaction(SMTPSERVER *smtp, char *command, MAIL *mail)
122   {
123           char buf[256];
124           char *newline;
125           int err = SMTP_OK;
126
127           if (command != NULL) {
128                   fprintf(smtp->out, "%s\r\n", command);
129                   fflush(smtp->out);
130           }
131    getstat:
132           smtp->result[0] = '\0';
```


Nov 16 1998 10:28 **delivermail-socket.c** Page 2

```
133          do {
134                  fgets(buf, sizeof(buf), smtp->in);
135                  if (smtp->result[0] == '\0') {
136                          strncpy(smtp->result, buf, SMTP_RESULT_LEN);
137                  }
138          } while (buf[3] == '-');
139
140          switch (buf[0]) {
141          case '1':
142          case '2':
143          case '4':
144                  break;
145
146          case '3':               /* feed data */
147                  /* ヘッダ */
148                  while (*mail->headers != NULL) {
149                          fprintf(smtp->out, "%s\r\n", *mail->headers);
150                          mail->headers++;
151                  }
152                  fputs("\r\n", smtp->out);
153
154                  /* ボディ */
155                  while (fgets(buf, sizeof(buf), mail->body) != NULL) {
156                          if (buf[0] == '.') {
157                                  fputc('.', smtp->out);
158                          }
159                          if ((newline = strchr(buf, '\n')) != NULL) {
160                                  *newline = '\0';
161                                  fprintf(smtp->out, "%s\r\n", buf);
162                          } else {
163                                  fprintf(smtp->out, "%s", buf);
164                          }
165                  }
166                  fputs(".\r\n", smtp->out);
167                  fflush(smtp->out);
168
169                  goto getstat;
170                  /*NOTREACHED*/
171
172          case '5':
173                  err = SMTP_ERR;
174                  break;
175          }
176
177          return err;
178  }
179
180  void main(int argc, char **argv)
181  {
182          MAIL mail;
183          SMTPSERVER *smtp;
184          static char *headers[] = {
185                  "From: <shuji>",
186                  "To: <nonki>",
187                  "Subject: test",
188                  "Mime-Version: 1.0",
189                  "Content-type: text/plain; charset=us-ascii",
190                  "Content-Transfer-Encoding: 7bit",
191                  NULL,
192          };
193
194          mail.headers = headers;
195          mail.body = stdin;
196
197          if ((smtp = open_smtpserver("mailserver")) == NULL) {
198                  fprintf(stderr, "open_smtpserver: localhost\n");
```



```
199                  exit(1);
200          }
201
202          if (transaction(smtp, "HELO localhost", NULL) == SMTP_ERR) {
203                  fprintf(stderr, "transaction: HELO; %s\n", smtp->result);
204                  goto quit;
205          }
206          if (transaction(smtp, "MAIL FROM: <shuji>", NULL) == SMTP_ERR) {
207                  fprintf(stderr, "transaction: MAIL FROM; %s\n", smtp->result);
208                  goto quit;
209          }
210          if (transaction(smtp, "RCPT TO: <nonki>", NULL) == SMTP_ERR) {
211                  fprintf(stderr, "transaction: RCPT TO; %s\n", smtp->result);
212                  goto quit;
213          }
214          if (transaction(smtp, "DATA", &mail) == SMTP_ERR) {
215                  fprintf(stderr, "transaction: DATA; %s\n", smtp->result);
216                  goto quit;
217          }
218          if (transaction(smtp, "QUIT", NULL) == SMTP_ERR) {
219                  fprintf(stderr, "transaction: QUIT; %s\n", smtp->result);
220                  goto quit;
221          }
222
223   quit:
224          close_smtpserver(smtp);
225          exit(0);
226  }
227
228  /* ----- end of delivermail-socket.c ----- */
```

```
1     #!/usr/local/bin/perl
2
3     #
4     # メールを受信する(POP)/perl4 socket 版
5     #
6     # Copyright 1998, Shuji Ishii, shuji@ccs.mt.nec.co.jp
7     #
8
9     require 'sys/socket.ph';
10
11    $AF_INET = &AF_INET;
12    $SOCK_STREAM = &SOCK_STREAM;
13
14    #
15    # pop サーバに接続する.
16    #
17    sub open_popserver
18    {
19        local($popserver) = @_;
20        local($name, $aliases, $type, $len, $server, $port, $proto);
21        local($sin, $s);
22
23        ($name, $aliases, $type, $len, $server) =  gethostbyname($popserver);
24        ($name, $aliases, $proto) = getprotobyname('tcp');
25        ($name, $aliases, $port) = getservbyname('pop3', 'tcp');
26
27        unless (socket(POP, $AF_INET, $SOCK_STREAM, $proto)) {
28            die $!;
29        }
30
31        $sin = pack('S n a4 x8', $AF_INET, $port, $server);
32
33        unless (connect(POP, $sin)) {
34            die $!;
35        }
36
37        # no buffering
38        select(POP); $| = 1; select(STDOUT);
39
40        return &transaction(undef);
41    }
42
43    sub close_popserver
44    {
45        close(POP);
46    }
47
48    #
49    # コマンドを送信し、結果を返す
50    #
51    sub transaction
52    {
53        local($command) = @_;
54        local($result);
55
56        if (defined($command)) {
57            print POP "$command\r\n";
58        }
59
60        $result = <POP>;
61        chop $result; chop $result;                # drop '\r\n';
62
63        return $result;
64    }
65
66    sub pop_login
```



```
67    {
68        local($user, $pass) = @_;
69        local($result);
70
71        if (($result = &transaction("USER $user")) =~ /^\-ERR/) {
72            return $result;
73        }
74        return &transaction("PASS $pass");
75    }
76
77    sub pop_msgcnt
78    {
79        local($msgcnt, $ttlsize);
80
81        if (&transaction("STAT") =~ /^\+OK (\d+) (\d+)/) {
82            return $1;
83        }
84
85        return -1;
86    }
87
88    sub pop_retr
89    {
90        local($msg) = @_;
91        local($result);
92
93        if (($result = &transaction("RETR $msg")) =~ /^\-ERR/) {
94            return undef;
95        }
96
97        while (<POP>) {
98            chop; chop;             # drop '\r\n';
99            if (/^\.(\..*)$/) {
100               print "$1\n";
101           } elsif (/^\.$/) {
102               # eof
103               last;
104           } else {
105               print "$_\n";
106           }
107       }
108
109       return $result;
110   }
111
112   ##
113   ## main
114   ##
115
116   &open_popserver("mailserver");
117   if (&pop_login("shuji", "password") =~ /^\-/) {
118       die "login error\n";
119   }
120   if (($msgcnt = &pop_msgcnt) < 0) {
121       die "pop_msgcnt: $msgcnt\n";
122   }
123   for ($i = 1; $i <= $msgcnt; $i++) {
124       if (!&pop_retr($i)) {
125           die "pop_retr\n";
126       }
127   }
128   &close_popserver;
129   exit(0);
130
131   ## end of fetchmail-perl4.pl ##
```

```
1     #!/usr/local/bin/perl
2
3     #
4     # メールを送信する(SMTP)/perl4 socket 版
5     #
6     # Copyright 1998, Shuji Ishii, shuji@ccs.mt.nec.co.jp
7     #
8
9     require 'sys/socket.ph';
10
11    $AF_INET = &AF_INET;
12    $SOCK_STREAM = &SOCK_STREAM;
13
14    #
15    # SMTP サーバに接続する.
16    #
17    sub open_smtpserver
18    {
19        local($smtpserver) = @_;
20        local($name, $aliases, $type, $len, $server, $port, $proto);
21        local($sin, $s);
22
23        ($name, $aliases, $type, $len, $server) =  gethostbyname($smtpserver);
24        ($name, $aliases, $proto) = getprotobyname('tcp');
25        ($name, $aliases, $port) = getservbyname('smtp', 'tcp');
26
27        unless (socket(SMTP, $AF_INET, $SOCK_STREAM, $proto)) {
28            die $!;
29        }
30
31        $sin = pack('S n a4 x8', $AF_INET, $port, $server);
32
33        unless (connect(SMTP, $sin)) {
34            die $!;
35        }
36
37        # no buffering
38        select(SMTP); $| = 1; select(STDOUT);
39
40        return &transaction(undef);
41    }
42
43    sub close_smtpserver
44    {
45        close(SMTP);
46    }
47
48    #
49    # コマンドを送信し、結果を返す
50    #
51    # 2xx: OK
52    # 5xx: ERR
53    # 3xx: need data
54    #
55    sub transaction
56    {
57        local($command, @headers) = @_;
58        local($result, $h, $dot);
59
60        if (defined($command)) {
61            print SMTP "$command\r\n";
62    print STDERR "transaction: $command\n";
63        }
64
65    getstat:
66        for (;;) {
```



```
67            undef($result);
68
69            do {
70                $_ = <SMTP>;
71                chop; chop;         # drop '\r\n'
72                unless ($result) {
73                    $result = $_;
74                }
75            } while (/^\d\d\d\-/);
76    print STDERR "transaction: $result\n";
77            if (/^[124]/) {
78                last;               # break;
79            } elsif (/^3/) {        # データ
80                # ヘッダ
81                for $h (@headers) {
82                    print SMTP "$h\r\n";
83                }
84                print SMTP "\r\n";
85                # 本文
86                while (<>) {
87                    chop;           # drop '\n' (when unix)
88                    if (/^\./) {
89                        $dot = ".";
90                    } else {
91                        $dot = '';
92                    }
93                    print SMTP "$dot$_\r\n";
94                }
95                print SMTP ".\r\n"; # 本文の終り
96                next getstat;
97            } elsif (/^5/) {
98                last;                       # break
99            }
100       }
101
102       return $result;
103   }
104
105   ## main
106
107   @headers = (
108               "From: <shuji>",
109               "To: <nonki>",
110               "Subject: test",
111               "Mime-Version: 1.0",
112               "Content-type: text/plain; charset=us-ascii",
113               "Content-Transfer-Encoding: 7bit",
114               );
115
116   if (&open_smtpserver("mailserver") !~ /^2/) {
117       die $!;
118   }
119   $hostname = chop `hostname`;
120   if (&transaction("HELO $hostname") =~ /^5/) {
121       die $!;
122   }
123   if (&transaction("MAIL FROM: <shuji>") =~ /^5/) {
124       die $!;
125   }
126   if (&transaction("RCPT TO: <nonki>") =~ /^5/) {
127       die $!;
128   }
129   if (&transaction("DATA", @headers) =~ /^5/) {
130       die $!;
131   }
132   if (&transaction("QUIT") =~ /^5/) {
```

```
133       die $!;
134   }
135
136   &close_smtpserver;
137   exit(0);
138
139   ## end of delivermail-perl4.pl ##
```

```
1     #!/usr/local/bin/perl5
2
3     use IO::Socket;
4
5     #
6     # メールを受信する(POP)/perl5 IO::Socket 版
7     #
8     # Copyright 1998, Shuji Ishii, shuji@ccs.mt.nec.co.jp
9     #
10
11    #
12    # pop サーバに接続する.
13    #
14    sub open_popserver
15    {
16        local($popserver) = @_;
17
18        $POP = IO::Socket::INET->new(PeerAddr => $popserver,
19                                     PeerPort => 'pop3(110)',
20                                     Proto    => 'tcp');
21        unless ($POP) {
22            die $!;
23        }
24
25        autoflush $POP 1;
26
27        return &transaction(undef);
28    }
29
30    sub close_popserver
31    {
32        close($POP);
33    }
34
35    #
36    # コマンドを送信し、結果を返す
37    #
38    sub transaction
39    {
40        local($command) = @_;
41        local($result);
42
43        if (defined($command)) {
44            print $POP "$command\r\n";
45        }
46
47        $result = <$POP>;
48        chop $result; chop $result;                # drop '\r\n';
49
50        return $result;
51    }
52
53    sub pop_login
54    {
55        local($user, $pass) = @_;
56        local($result);
57
58        if (($result = &transaction("USER $user")) =~ /^\-ERR/) {
59            return $result;
60        }
61        return &transaction("PASS $pass");
62    }
63
64    sub pop_msgcnt
65    {
66        local($msgcnt, $ttlsize);
```



```
67
68        if (&transaction("STAT") =~ /^\+OK (\d+) (\d+)/) {
69            return $1;
70        }
71
72        return -1;
73    }
74
75    sub pop_retr
76    {
77        local($msg) = @_;
78        local($result);
79
80        if (($result = &transaction("RETR $msg")) =~ /^\-ERR/) {
81            return $undef;
82        }
83
84        while (<$POP>) {
85            chop; chop;            # drop '\r\n';
86            if (/^\.(\..*)$/) {
87                print "$1\n";
88            } elsif (/^\.$/) {
89                # eof
90                last;
91            } else {
92                print "$_\n";
93            }
94        }
95
96        return $result;
97    }
98
99    ##
100   ## main
101   ##
102
103   &open_popserver("mailserver");
104   if (&pop_login("shuji", "password") =~ /^\-/) {
105       die "login error\n";
106   }
107   $msgcnt = &pop_msgcnt;
108   if ($msgcnt < 0) {
109       die "pop_msgcnt: $msgcnt\n";
110   }
111
112   for ($i = 1; $i <= $msgcnt; $i++) {
113       if (!&pop_retr($i)) {
114           die "pop_retr\n";
115       }
116   }
117
118   &close_popserver;
119   exit(0);
120
121   ## end of fetchmail-perl5-sock.pl ##
```

```
#!/usr/local/bin/perl5

#
# メールを送信する(SMTP)/perl5 IO::Socket 版
#
# Copyright 1998, Shuji Ishii, shuji@ccs.mt.nec.co.jp
#

use IO::Socket;

#
# SMTP サーバに接続する.
#
sub open_smtpserver
{
    local($smtpserver) = @_;

    $SMTP = IO::Socket::INET->new(PeerAddr => $smtpserver,
                                  PeerPort => 'smtp(25)',
                                  Proto    => 'tcp');
    unless ($SMTP) {
        die $!;
    }
    autoflush $SMTP 1;

    return &transaction(undef);
}

sub close_smtpserver
{
    close($SMTP);
}

#
# コマンドを送信し、結果を返す
#
# 2xx: OK
# 5xx: ERR
# 3xx: need data
#
sub transaction
{
    local($command, @headers) = @_;
    local($result, $h, $dot);

    if (defined($command)) {
        print $SMTP "$command\r\n";
    }

getstat:
    for (;;) {
        undef($result);

        do {
            $_ = <$SMTP>;
            chop; chop;         # drop '\r\n'
            unless ($result) {
                $result = $_;
            }
        } while (/^\d\d\d\-/);
        if (/^[124]/) {
            last;               # break;
        } elsif (/^3/) {        # データ
            # ヘッダ
            for $h (@headers) {
                print $SMTP "$h\r\n";
            }
            print $SMTP "\r\n";
            # 本文
            while (<>) {
                chop;           # drop '\n' (when unix)
                if (/^\./) {
                    $dot = ".";
                } else {
                    $dot = '';
                }
                print $SMTP "$dot$_\r\n";
            }
            print $SMTP ".\r\n";        # 本文の終り
            next getstat;
        } elsif (/^5/) {
            last;                       # break
        }
    }

    return $result;
}

## main

@headers = (
            "From: <shuji>",
            "To: <nonki>",
            "Subject: test",
            "Mime-Version: 1.0",
            "Content-type: text/plain; charset=us-ascii",
            "Content-Transfer-Encoding: 7bit",
            );

if (&open_smtpserver("mailserver") !~ /^2/) {
    die $!;
}
$myname = chop `hostname`;
if (&transaction("HELO $myname") =~ /^5/) {
    die $!;
}
if (&transaction("MAIL FROM: <shuji>") =~ /^5/) {
    die $!;
}
if (&transaction("RCPT TO: <nonki>") =~ /^5/) {
    die $!;
}
if (&transaction("DATA", @headers) =~ /^5/) {
    die $!;
}
if (&transaction("QUIT") =~ /^5/) {
    die $!;
}

&close_smtpserver;
exit(0);

## end of delivermail-perl5-sock.pl ##
```


Nov 16 1998 10:28 **delivermail-perl5-sock.pl** Page 2

```
1     #!/usr/local/bin/perl5
2
3     #
4     # メールを受信する(POP)/perl5 Mail::POP3Client 版
5     #
6     # Copyright 1998, Shuji Ishii, shuji@ccs.mt.nec.co.jp
7     #
8
9     ##
10    ## POP3Client-x.xx
11    ##
12
13    use Mail::POP3Client
14
15    $pop = new Mail::POP3Client("shuji", "password", "mailserver");
16    for ($i = 1; $i <= $pop->Count; $i++) {
17        $_ = $pop->Retrieve($i);
18        tr/\r//d;
19        print "$_\n";
20    }
21
22    ## end of fetchmail-perl5-mailpop3client.pl ##
```

```
1     #!/usr/local/bin/perl5
2
3     #
4     # メールを送信する(SMTP)/perl5 Mail::Internet 版
5     #
6     # Copyright 1998, Shuji Ishii, shuji@ccs.mt.nec.co.jp
7     #
8
9     ##
10    ## libnet-x.xx
11    ## DataDumper-x.xx
12    ## MailTools-x.xx
13    ##
14
15    use Mail::Internet;
16    use Mail::Header;
17
18    @headers = (
19                "From: <shuji>",
20                "To: <nonki>",
21                "Subject: test",
22                "Mime-Version: 1.0",
23                "Content-type: text/plain; charset=us-ascii",
24                "Content-Transfer-Encoding: 7bit",
25                );
26
27    $header = new Mail::Header(\@headers);
28    $msg = new Mail::Internet([<>], Header => $header);
29    $msg->smtpsend({Host => "mailserver"});
30
31    ## end of delivermail-perl5-mailinternet.pl ##
```

```
1     // stdafx.h : 標準のシステム インクルード ファイル、
2     //            または参照回数が多く、かつあまり変更されない
3     //            プロジェクト専用のインクルード ファイルを記述します。
4     //
5
6     #if !defined(AFX_STDAFX_H__A9DB83DB_A9FD_11D0_BFD1_444553540000__INCLUDED_)
7     #define AFX_STDAFX_H__A9DB83DB_A9FD_11D0_BFD1_444553540000__INCLUDED_
8
9     #if _MSC_VER > 1000
10    #pragma once
11    #endif // _MSC_VER > 1000
12
13    #define WIN32_LEAN_AND_MEAN             // Windows ヘッダーから殆ど使用されない
      スタッフを除外します
14
15
16    // Windows ヘッダー ファイル:
17    #include <windows.h>
18
19    // C ランタイム ヘッダー ファイル
20    #include <stdlib.h>
21    #include <malloc.h>
22    #include <memory.h>
23    #include <tchar.h>
24
25    // ローカル ヘッダー ファイル
26
27    // TODO: プログラムで必要なヘッダー参照を追加してください。
28
29    //{{AFX_INSERT_LOCATION}}
30    // Microsoft Visual C++ は前行の直前に追加の宣言を挿入します。
31
32    #endif // !defined(AFX_STDAFX_H__A9DB83DB_A9FD_11D0_BFD1_444553540000__INCLUDED_
      )
```

```
1     // stdafx.cpp : 標準インクルードファイルを含むソース ファイル
2     //              iw98_1.pch 生成されるプリコンパイル済ヘッダー
3     //              stdafx.obj 生成されるプリコンパイル済タイプ情報
4
5     #include "stdafx.h"
6
7     // TODO: STDAFX.H に含まれていて、このファイルに記述されていない
8     // ヘッダーファイルを追加してください。
```

```
1
2     #if !defined(AFX_IW98_1_H__CEE76366_778C_11D2_9763_00400530B261__INCLUDED_)
3     #define AFX_IW98_1_H__CEE76366_778C_11D2_9763_00400530B261__INCLUDED_
4
5     #if _MSC_VER > 1000
6     #pragma once
7     #endif // _MSC_VER > 1000
8
9     #include "resource.h"
10
11
12    #endif // !defined(AFX_IW98_1_H__CEE76366_778C_11D2_9763_00400530B261__INCLUDED_
      )
```

```
1     //{{NO_DEPENDENCIES}}
2     // Microsoft Developer Studio generated include file.
3     // Used by iw98_1.rc
4     //
5     #define IDC_MYICON                      2
6     #define IDD_IW98_1_DIALOG               102
7     #define IDD_ABOUTBOX                    103
8     #define IDS_APP_TITLE                   103
9     #define IDM_ABOUT                       104
10    #define IDM_EXIT                        105
11    #define IDS_HELLO                       106
12    #define IDI_IW98_1                      107
13    #define IDI_SMALL                       108
14    #define IDC_IW98_1                      109
15    #define IDR_MAINFRAME                   128
16    #define IDD_SENDMAIL                    129
17    #define IDD_POP                         130
18    #define IDC_EDIT_TO                     1000
19    #define IDC_EDIT_BODY                   1001
20    #define IDC_EDIT_SUBJECT                1002
21    #define IDC_EDIT_USER                   1003
22    #define IDC_EDIT_PASSWORD               1004
23    #define IDC_LIST                        1005
24    #define ID_SEND_MAIL_WSA                32771
25    #define ID_RECV_MAIL                    32772
26    #define IDC_STATIC                      -1
27
28    // Next default values for new objects
29    //
30    #ifdef APSTUDIO_INVOKED
31    #ifndef APSTUDIO_READONLY_SYMBOLS
32    #define _APS_NEXT_RESOURCE_VALUE        131
33    #define _APS_NEXT_COMMAND_VALUE         32773
34    #define _APS_NEXT_CONTROL_VALUE         1006
35    #define _APS_NEXT_SYMED_VALUE           110
36    #endif
37    #endif
```

```
1     #ifndef __RECVMAIL_H__
2     #define __RECVMAIL_H__
3
4     LRESULT CALLBACK RecvMailProc( HWND hDlg, UINT message, WPARAM wParam, LPARAM lP
      aram );
5
6     #endif // __RECVMAIL_H__
```

```
1     #ifndef __SENDMAIL_H__
2     #define __SENDMAIL_H__
3
4     LRESULT CALLBACK SendMailProc( HWND hDlg, UINT message, WPARAM wParam, LPARAM lP
      aram );
5
6     #endif // __SENDMAIL_H__
```

```
1     // iw98_1.cpp
2     //
3
4     #include "stdafx.h"
5     #include <winsock.h>
6     #include "resource.h"
7
8     #include "sendmail.h"
9     #include "recvmail.h"
10
11    #define MAX_LOADSTRING 100
12
13    // グローバル変数:
14    HINSTANCE hInst;                                // 現在のインス
      タンス
15    TCHAR szTitle[MAX_LOADSTRING];                  // タイトル バー テキスト
16    TCHAR szWindowClass[MAX_LOADSTRING];    // タイトル バー テキスト
17
18    // このコード モジュールに含まれる関数の前宣言:
19    ATOM MyRegisterClass( HINSTANCE hInstance );
20    BOOL InitInstance( HINSTANCE, int );
21    LRESULT CALLBACK WndProc( HWND, UINT, WPARAM, LPARAM );
22    bool InitWinsock(WSADATA* pWsaData);
23
24    int APIENTRY WinMain(HINSTANCE hInstance,
25                         HINSTANCE hPrevInstance,
26                         LPSTR     lpCmdLine,
27                         int       nCmdShow )
28    {
29            MSG msg;
30            HACCEL hAccelTable;
31            WSADATA wsadata;
32
33            LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
34            LoadString(hInstance, IDC_IW98_1, szWindowClass, MAX_LOADSTRING);
35            MyRegisterClass( hInstance );
36
37            if( !InitInstance( hInstance, nCmdShow ) )
38            {
39                    return FALSE;
40            }
41
42            // Winsockの初期化
43            if(!InitWinsock(&wsadata)){
44                    return FALSE;
45            }
46
47            hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_IW98_1);
48
49            while( GetMessage(&msg, NULL, 0, 0) )
50            {
51                    if( !TranslateAccelerator (msg.hwnd, hAccelTable, &msg) )
52                    {
53                            TranslateMessage( &msg );
54                            DispatchMessage( &msg );
55                    }
56            }
57
58            // Winsockの解放
59            WSACleanup();
60
61            return msg.wParam;
62    }
63
64    // Winsockの初期化
65    bool InitWinsock(WSADATA* pWsaData)
```



```
66    {
67            int nRequiredVersion = MAKEWORD(1,1);           // Version 1.1
68            if(WSAStartup(nRequiredVersion, pWsaData) != 0){
69                    return false;
70            }
71            if(pWsaData->wVersion != nRequiredVersion){
72                    WSACleanup();
73                    return false;
74            }
75            return true;
76    }
77
78    //
79    //  関数: MyRegisterClass()
80    //
81    //  用途: ウィンドウ クラスの登録
82    //
83    //
84    ATOM MyRegisterClass( HINSTANCE hInstance )
85    {
86            WNDCLASSEX wcex;
87
88            wcex.cbSize = sizeof(WNDCLASSEX);
89
90            wcex.style                      = CS_HREDRAW | CS_VREDRAW;
91            wcex.lpfnWndProc        = (WNDPROC)WndProc;
92            wcex.cbClsExtra         = 0;
93            wcex.cbWndExtra         = 0;
94            wcex.hInstance          = hInstance;
95            wcex.hIcon                      = LoadIcon(hInstance, (LPCTSTR)IDI_IW98_
      1);
96            wcex.hCursor            = LoadCursor(NULL, IDC_ARROW);
97            wcex.hbrBackground      = (HBRUSH)(COLOR_WINDOW+1);
98            wcex.lpszMenuName       = (LPCSTR)IDC_IW98_1;
99            wcex.lpszClassName      = szWindowClass;
100           wcex.hIconSm            = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);
101
102           return RegisterClassEx( &wcex );
103   }
104
105   //
106   //   関数: InitInstance(HANDLE, int)
107   //
108   //   用途: インスタンス ハンドルの保存とメイン ウィンドウの作成
109   //
110   //
111   BOOL InitInstance( HINSTANCE hInstance, int nCmdShow )
112   {
113      HWND hWnd;
114
115      hInst = hInstance; // グローバル変数にインスタンス ハンドルを保存します
116
117      hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
118         CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);
119
120      if( !hWnd )
121      {
122         return FALSE;
123      }
124
125      ShowWindow( hWnd, nCmdShow );
126      UpdateWindow( hWnd );
127
128      return TRUE;
129   }
130
```

```
131   //
132   //  関数: WndProc(HWND, unsigned, WORD, LONG)
133   //
134   //  用途: メイン ウィンドウのメッセージを処理します。
135   //
136   //  WM_COMMAND  - アプリケーション メニューの処理
137   //  WM_PAINT    - メイン ウィンドウの描画
138   //  WM_DESTROY  - 終了メッセージの通知とリターン
139   //
140   //
141   LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
142   {
143           int wmId, wmEvent;
144           TCHAR szHello[MAX_LOADSTRING];
145           LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);
146
147           switch( message )
148           {
149           case WM_COMMAND:
150                   wmId    = LOWORD(wParam);
151                   wmEvent = HIWORD(wParam);
152                   // メニュー選択の解析:
153                   switch( wmId )
154                   {
155                           case IDM_EXIT:
156                                   DestroyWindow( hWnd );
157                                   break;
158                           case ID_SEND_MAIL_WSA:
159                                   DialogBox(hInst, (LPCTSTR)IDD_SENDMAIL, hWnd, (D
      LGPROC)SendMailProc);
160                                   break;
161                           case ID_RECV_MAIL:
162                                   DialogBox(hInst, (LPCTSTR)IDD_POP, hWnd, (DLGPRO
      C)RecvMailProc);
163                                   break;
164                           default:
165                              return DefWindowProc( hWnd, message, wParam, lParam )
      ;
166                   }
167                   break;
168
169           case WM_DESTROY:
170                   PostQuitMessage( 0 );
171                   break;
172
173           default:
174                   return DefWindowProc( hWnd, message, wParam, lParam );
175      }
176      return 0;
177   }
178
```

```
1     #include "stdafx.h"
2     #include <winsock.h>
3     #include <string>
4
5     #include "resource.h"
6     #include "recvmail.h"
7
8     using namespace std;
9
10    #define UM_SELECT               (WM_USER+100)
11    #define UM_HOSTBYNAME   (WM_USER+101)
12    #define UM_QUIT                 (WM_USER+102)
13
14    #define MAX_USER                1000
15    #define MAX_PASS                1000
16
17    #define POP_HOST                "hogehoge"
18    #define POP_PORT                110
19
20    enum POPSTAT {
21            ST_START,
22            ST_CONNECTTING,
23            ST_POP_GREEDING,
24            ST_POP_USER,
25            ST_POP_PASS,
26            ST_POP_STAT,
27            ST_POP_RESP_STAT,
28            ST_POP_RETR,
29            ST_POP_RESP_RETR,
30            ST_POP_RECV_MESSAGE,
31            ST_POP_QUIT,
32            ST_CLOSE
33    };
34
35    static int              g_state;
36    static char             g_bufHostent[MAXGETHOSTSTRUCT];
37    static char             g_szUser[MAX_USER];
38    static char             g_szPass[MAX_PASS];
39    static SOCKET   g_socket;
40    static char*    g_pszRecvBuffer = NULL;
41    static int              g_nBufSize = 0;
42    static int              g_nMessageNumber;
43    static int              g_nTotalCount;
44    static bool             g_bInHeader;
45
46    static bool OnOk(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
47    {
48            GetWindowText(GetDlgItem(hDlg, IDC_EDIT_USER), g_szUser, MAX_USER);
49            GetWindowText(GetDlgItem(hDlg, IDC_EDIT_PASSWORD), g_szPass, MAX_PASS);
50
51            if(strlen(g_szUser) == 0 || strlen(g_szPass) == 0){
52                    return false;
53            }
54
55            return WSAAsyncGetHostByName(hDlg, UM_HOSTBYNAME, POP_HOST,
56                                    g_bufHostent, MAXGETHOSTSTRUCT) != 0;
57    }
58
59    static bool OnGetHostByName(HWND hDlg, UINT message, WPARAM wParam, LPARAM lPara
      m)
60    {
61            int nError = WSAGETASYNCERROR(lParam);
62            if(nError != 0){
63                    return false;
64            }
65            LPHOSTENT lpHostent = (LPHOSTENT)&g_bufHostent;
```



```
66
67            g_socket = socket(AF_INET, SOCK_STREAM, 0);
68            if(g_socket == INVALID_SOCKET){
69                    return false;
70            }
71
72            // 非同期モードに移行
73            if(WSAAsyncSelect(g_socket, hDlg, UM_SELECT,
74                                    FD_CONNECT|FD_READ|FD_WRITE|FD_CLOSE) == SOCKET_
      ERROR)
75            {
76                    return false;
77            }
78
79            //
80            SOCKADDR_IN sockaddr;
81            sockaddr.sin_family = AF_INET;
82            sockaddr.sin_port = htons(POP_PORT);;
83            sockaddr.sin_addr = *((LPIN_ADDR)*lpHostent->h_addr_list);
84
85            if(connect(g_socket, (LPSOCKADDR)&sockaddr, sizeof(SOCKADDR_IN)) == SOCK
      ET_ERROR)
86            {
87                    if(WSAGetLastError() != WSAEWOULDBLOCK){
88                            return false;
89                    }
90            }
91            g_state = ST_CONNECTTING;
92
93            return true;
94    }
95
96    static bool ParsePopResponse(string& text)
97    {
98            if(text.substr(0, 3) == "+OK"){
99                    return true;
100           }
101           return false;
102   }
103
104   static void CloseSocket()
105   {
106           closesocket(g_socket);
107           g_state = ST_CLOSE;
108   }
109
110   // コマンドの送出
111   static bool SendComand(LPCSTR lpszCommand)
112   {
113           string cmd = lpszCommand;
114           cmd += "\r\n";
115           if(send(g_socket, cmd.c_str(), cmd.length(), 0) == SOCKET_ERROR){
116                   CloseSocket();
117                   return false;
118           }
119           return true;
120   }
121
122   // 行の解析。
123   static bool ParseLine(HWND hDlg, string& text)
124   {
125           char szCommand[256];
126
127           if(g_state != ST_POP_RECV_MESSAGE){
128                   // レスポンスの受信
129                   bool bResult = ParsePopResponse(text);
```

```
130                     if(!bResult){
131                             SendComand("QUIT");
132                             g_state = ST_POP_QUIT;
133                             return false;
134                     }
135             }
136
137             bool bContinueLoop;
138             do{
139                     bContinueLoop = false;
140                     switch(g_state)
141                     {
142                     case ST_POP_GREEDING:
143                             wsprintf(szCommand, "USER %s", g_szUser);
144                             if(!SendComand(szCommand)){
145                                     return false;
146                             }
147                             g_state = ST_POP_PASS;
148                             break;
149
150                     case ST_POP_PASS:
151                             wsprintf(szCommand, "PASS %s", g_szPass);
152                             if(!SendComand(szCommand)){
153                                     return false;
154                             }
155                             g_state = ST_POP_STAT;
156                             break;
157
158                     case ST_POP_STAT:
159                             if(!SendComand("STAT")){
160                                     return false;
161                             }
162                             g_state = ST_POP_RESP_STAT;
163                             break;
164
165                     case ST_POP_RESP_STAT:
166                             g_nTotalCount = atoi(text.c_str()+4);
167                             g_nMessageNumber = 1;
168                             g_state = ST_POP_RETR;
169                             bContinueLoop = true;
170                             break;
171
172                     case ST_POP_RETR:
173                             if(g_nMessageNumber > g_nTotalCount){
174                                     if(!SendComand("QUIT")){
175                                             return false;
176                                     }
177                                     g_state = ST_POP_QUIT;
178                                     break;
179                             }
180
181                             wsprintf(szCommand, "RETR %d", g_nMessageNumber++);
182                             g_bInHeader = true;
183                             if(!SendComand(szCommand)){
184                                     return false;
185                             }
186                             g_state = ST_POP_RESP_RETR;
187                             break;
188
189                     case ST_POP_RESP_RETR:
190                             g_state = ST_POP_RECV_MESSAGE;
191                             break;
192
193                     case ST_POP_RECV_MESSAGE:
194                             if(text == ".\r\n"){
195                                     // end of message
```



```
196                                     g_state = ST_POP_RETR;
197                                     bContinueLoop = true;
198                             }else{
199                                     if(g_bInHeader){
200                                             // ヘッダをリストに表示
201                                             SendMessage(GetDlgItem(hDlg, IDC_LIST),
    LB_ADDSTRING, 0, (LPARAM)text.c_str());
202                                     }
203                                     if(text == "\r\n"){
204                                             g_bInHeader = false;
205                                     }
206                             }
207                             break;
208
209                     case ST_POP_QUIT:
210                             CloseSocket();
211                             PostMessage(hDlg, UM_QUIT, 0, 0);
212                             break;
213                     }
214             }while(bContinueLoop);
215             return true;
216     }
217
218     // 受信したデータを行単位に切り出す
219     static bool ParseData(HWND hDlg, const char* buf)
220     {
221             const int BUF_UNIT_SIZE = 2000;
222             if(g_pszRecvBuffer == NULL){
223                     g_pszRecvBuffer = new char[BUF_UNIT_SIZE];
224                     *g_pszRecvBuffer = '\0';
225                     g_nBufSize = BUF_UNIT_SIZE;
226             }
227
228             // bufをg_pszRecvBufferに追加
229             int cnt = strlen(buf);
230             if(strlen(g_pszRecvBuffer)+cnt+1 > g_nBufSize){
231                     // g_pszRecvBufferを拡大
232                     int nNewBufSize = ((strlen(g_pszRecvBuffer)+cnt+1)/BUF_UNIT_SIZE
    +1)*BUF_UNIT_SIZE;
233                     char* pszNewBuffer = new char[nNewBufSize];
234                     strcpy(pszNewBuffer, g_pszRecvBuffer);
235                     delete g_pszRecvBuffer;
236                     g_pszRecvBuffer = pszNewBuffer;
237                     g_nBufSize = nNewBufSize;
238             }
239             strcat(g_pszRecvBuffer, buf);
240
241             // 行を切り出してParseLineに渡す。
242             while(true){
243                     LPTSTR lpszPos = strchr(g_pszRecvBuffer, '\n');
244                     if(lpszPos == NULL){
245                             break;
246                     }
247                     *lpszPos = '\0';
248                     string text = g_pszRecvBuffer;
249                     text += "\n";
250                     if(strlen(lpszPos+1) > 0){
251                             memmove(g_pszRecvBuffer, lpszPos+1, strlen(lpszPos+1)+1)
    ;
252                     }else{
253                             *g_pszRecvBuffer = '\0';
254                     }
255                     if(!ParseLine(hDlg, text)){
256                             return false;
257                     }
258             }
```

```
259         return true;
260 }
261
262 static bool OnSelect(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
263 {
264         switch(WSAGETSELECTEVENT(lParam))
265         {
266         case FD_CONNECT:
267                 g_state = ST_POP_GREEDING;
268                 g_pszRecvBuffer = NULL;
269                 break;
270
271         case FD_READ:
272                 {
273                         char buf[1000];
274                         int cnt = recv(g_socket, buf, sizeof(buf)-1, 0);
275                         if(cnt == SOCKET_ERROR){
276                                 return false;
277                         }else if(cnt == 0){
278                                 // closed
279                                 return false;
280                         }
281                         *(buf+cnt) = '\0';
282                         ParseData(hDlg, buf);
283                 }
284                 break;
285
286         case FD_WRITE:
287                 break;
288
289         case FD_CLOSE:
290                 break;
291         }
292         return true;
293 }
294
295 LRESULT CALLBACK RecvMailProc( HWND hDlg, UINT message, WPARAM wParam, LPARAM lP
    aram )
296 {
297         switch( message )
298         {
299         case WM_INITDIALOG:
300                 g_state = ST_START;
301                 return TRUE;
302
303         case WM_COMMAND:
304                 if( LOWORD(wParam) == IDOK ){
305                         SendMessage(GetDlgItem(hDlg, IDC_LIST), LB_RESETCONTENT,
     0, 0);
306                         OnOk(hDlg, message, wParam, lParam);
307                 }else if(LOWORD(wParam) == IDCANCEL){
308                         EndDialog(hDlg, LOWORD(wParam));
309                         return TRUE;
310                 }
311                 break;
312
313         case WM_DESTROY:
314                 delete[] g_pszRecvBuffer;
315                 g_pszRecvBuffer = NULL;
316                 break;
317
318         case UM_HOSTBYNAME:
319                 OnGetHostByName(hDlg, message, wParam, lParam);
320                 break;
321
322         case UM_SELECT:
```



```
323                 OnSelect(hDlg, message, wParam, lParam);
324                 break;
325
326         case UM_QUIT:
327                 SendMessage(GetDlgItem(hDlg, IDC_LIST), LB_ADDSTRING, 0, (LPARAM
    )"====");
328                 return TRUE;
329         }
330     return FALSE;
331 }
332
```

```
1     #include "stdafx.h"
2     #include <winsock.h>
3     #include <string>
4
5     #include "resource.h"
6     #include "sendmail.h"
7
8     using namespace std;
9
10    #define UM_SELECT               (WM_USER+100)
11    #define UM_HOSTBYNAME   (WM_USER+101)
12    #define UM_POSTEND              (WM_USER+102)
13
14    #define MAX_TO                  1000
15    #define MAX_SUBJECT             1000
16    #define MAX_BODY                1000
17
18    #define SMTP_HOST               "hogehoge"
19    #define SMTP_PORT               25
20    #define FQDN                    "hogehoge.co.jp"
21    #define MYADDRESS               "hoge@hogehoge.co.jp"
22
23    enum SMTP_STAT {
24            ST_START,
25            ST_CONNECTTING,
26            ST_SMTP_GREEDING,
27            ST_SMTP_HELO,
28            ST_SMTP_MAILFROM,
29            ST_SMTP_RCPT,
30            ST_SMTP_DATA,
31            ST_SMTP_QUIT,
32            ST_CLOSE
33    };
34
35    static int              g_state;
36    static char             g_bufHostent[MAXGETHOSTSTRUCT];
37    static char             g_szTo[MAX_TO];
38    static char             g_szSubject[MAX_SUBJECT];
39    static char             g_szBody[MAX_BODY];
40    static SOCKET   g_socket;
41    static char*    g_pszRecvBuffer = NULL;
42    static int              g_nBufSize = 0;
43
44    static bool OnOk(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
45    {
46            GetWindowText(GetDlgItem(hDlg, IDC_EDIT_TO), g_szTo, MAX_TO);
47            GetWindowText(GetDlgItem(hDlg, IDC_EDIT_SUBJECT), g_szSubject, MAX_SUBJE
      CT);
48            GetWindowText(GetDlgItem(hDlg, IDC_EDIT_BODY), g_szBody, MAX_BODY);
49
50            if(strlen(g_szTo) == 0){
51                    return false;
52            }
53
54            return WSAAsyncGetHostByName(hDlg, UM_HOSTBYNAME, SMTP_HOST,
55                                    g_bufHostent, MAXGETHOSTSTRUCT) != 0;
56    }
57
58    static bool OnGetHostByName(HWND hDlg, UINT message, WPARAM wParam, LPARAM lPara
      m)
59    {
60            int nError = WSAGETASYNCERROR(lParam);
61            if(nError != 0){
62                    return false;
63            }
64            LPHOSTENT lpHostent = (LPHOSTENT)&g_bufHostent;
```



```
65
66            g_socket = socket(AF_INET, SOCK_STREAM, 0);
67            if(g_socket == INVALID_SOCKET){
68                    return false;
69            }
70
71            // 非同期モードに移行
72            if(WSAAsyncSelect(g_socket, hDlg, UM_SELECT,
73                                    FD_CONNECT|FD_READ|FD_WRITE|FD_CLOSE) == SOCKET_
      ERROR)
74            {
75                    return false;
76            }
77
78            //
79            SOCKADDR_IN sockaddr;
80            sockaddr.sin_family = AF_INET;
81            sockaddr.sin_port = htons(SMTP_PORT);;
82            sockaddr.sin_addr = *((LPIN_ADDR)*lpHostent->h_addr_list);
83
84            if(connect(g_socket, (LPSOCKADDR)&sockaddr, sizeof(SOCKADDR_IN)) == SOCK
      ET_ERROR)
85            {
86                    if(WSAGetLastError() != WSAEWOULDBLOCK){
87                            return false;
88                    }
89            }
90            g_state = ST_CONNECTTING;
91
92            return true;
93    }
94
95    static void ParseSmtpResponse(string& text, int& code, bool& bContinue)
96    {
97            string num = text.substr(0, 3);
98            code = atoi(num.c_str());
99            bContinue = text[3] == '-';
100   }
101
102   static void CloseSocket()
103   {
104           closesocket(g_socket);
105           g_state = ST_CLOSE;
106   }
107
108   static bool SendComand(LPCSTR lpszCommand)
109   {
110           string cmd = lpszCommand;
111           cmd += "\r\n";
112           if(send(g_socket, cmd.c_str(), cmd.length(), 0) == SOCKET_ERROR){
113                   CloseSocket();
114                   return false;
115           }
116           return true;
117   }
118
119   static bool SendMail()
120   {
121           string message;
122           message += string("to: ") + g_szTo + "\r\n";
123           message += string("subject: ") + g_szSubject + "\r\n\r\n";
124           message += g_szBody;
125           message += "\r\n.\r\n";
126
127           if(send(g_socket, message.c_str(), message.length(), 0) == SOCKET_ERROR)
      {
```

```
128                     CloseSocket();
129                     return false;
130             }
131             return true;
132     }
133
134     static bool ParseLine(HWND hDlg, string& text)
135     {
136             int code;
137             bool bContinue;
138             char szCommand[256];
139
140             ParseSmtpResponse(text, code, bContinue);
141
142             if(bContinue){
143                     return true;    // 継続行がある。
144             }
145
146             int result = code/100;
147
148             switch(g_state)
149             {
150             case ST_SMTP_GREEDING:
151                     if(result != 2 ){
152                             SendComand("QUIT");
153                             g_state = ST_SMTP_QUIT;
154                             return false;
155                     }
156
157                     wsprintf(szCommand, "HELO %s", FQDN);
158                     if(!SendComand(szCommand)){
159                             return false;
160                     }
161                     g_state = ST_SMTP_HELO;
162                     break;
163
164             case ST_SMTP_HELO:
165                     if(result != 2 ){
166                             SendComand("QUIT");
167                             g_state = ST_SMTP_QUIT;
168                             return false;
169                     }
170
171                     wsprintf(szCommand, "MAIL FROM: <%s>", MYADDRESS);
172                     if(!SendComand(szCommand)){
173                             return false;
174                     }
175                     g_state = ST_SMTP_MAILFROM;
176                     break;
177
178             case ST_SMTP_MAILFROM:
179                     if(result != 2 ){
180                             SendComand("QUIT");
181                             g_state = ST_SMTP_QUIT;
182                             return false;
183                     }
184
185                     wsprintf(szCommand, "RCPT TO: <%s>", g_szTo);
186                     if(!SendComand(szCommand)){
187                             return false;
188                     }
189                     g_state = ST_SMTP_RCPT;
190                     break;
191
192             case ST_SMTP_RCPT:
193                     // "2yz"以外はQUITコマンドを送出して終了。
```



```
194                     if(result != 2 ){
195                             SendComand("QUIT");
196                             g_state = ST_SMTP_QUIT;
197                             return false;
198                     }
199
200                     if(!SendComand("DATA")){
201                             return false;
202                     }
203                     g_state = ST_SMTP_DATA;
204                     break;
205
206             case ST_SMTP_DATA:
207                     // "2yz"以外はQUITコマンドを送出して終了。
208                     if(result != 3 ){
209                             SendComand("QUIT");
210                             g_state = ST_SMTP_QUIT;
211                             return false;
212                     }
213
214                     if(!SendMail()){
215                             CloseSocket();
216                             return false;
217                     }
218
219                     if(!SendComand("QUIT")){
220                             return false;
221                     }
222                     g_state = ST_SMTP_QUIT;
223                     break;
224
225             case ST_SMTP_QUIT:
226                     CloseSocket();
227                     PostMessage(hDlg, UM_POSTEND, 0, 0);
228                     break;
229             }
230
231             return true;
232     }
233
234     // 行を切り出す
235     static bool ParseData(HWND hDlg, const char* buf)
236     {
237             const int BUF_UNIT_SIZE = 2000;
238             if(g_pszRecvBuffer == NULL){
239                     g_pszRecvBuffer = new char[BUF_UNIT_SIZE];
240                     *g_pszRecvBuffer = '\0';
241                     g_nBufSize = BUF_UNIT_SIZE;
242             }
243
244             // bufをg_pszRecvBufferに追加
245             int cnt = strlen(buf);
246             if(strlen(g_pszRecvBuffer)+cnt+1 > g_nBufSize){
247                     // g_pszRecvBufferを拡大
248                     int nNewBufSize = ((strlen(g_pszRecvBuffer)+cnt+1)/BUF_UNIT_SIZE
    +1)*BUF_UNIT_SIZE;
249                     char* pszNewBuffer = new char[nNewBufSize];
250                     strcpy(pszNewBuffer, g_pszRecvBuffer);
251                     delete g_pszRecvBuffer;
252                     g_pszRecvBuffer = pszNewBuffer;
253                     g_nBufSize = nNewBufSize;
254             }
255             strcat(g_pszRecvBuffer, buf);
256
257             // 行を切り出してParseLineに渡す。
258             while(true){
```

```
259                 LPTSTR lpszPos = strchr(g_pszRecvBuffer, '\n');
260                 if(lpszPos == NULL){
261                         break;
262                 }
263                 *lpszPos = '\0';
264                 string text = g_pszRecvBuffer;
265                 text += "\n";
266                 if(strlen(lpszPos+1) > 0){
267                         memmove(g_pszRecvBuffer, lpszPos+1, strlen(lpszPos+1)+1)
    ;
268                 }else{
269                         *g_pszRecvBuffer = '\0';
270                 }
271                 if(!ParseLine(hDlg, text)){
272                         return false;
273                 }
274         }
275         return true;
276   }
277
278   static bool OnSelect(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
279   {
280         switch(WSAGETSELECTEVENT(lParam))
281         {
282         case FD_CONNECT:
283                 g_state = ST_SMTP_GREEDING;
284                 g_pszRecvBuffer = NULL;
285                 break;
286
287         case FD_READ:
288                 {
289                         char buf[1000];
290                         int cnt = recv(g_socket, buf, sizeof(buf)-1, 0);
291                         if(cnt == SOCKET_ERROR){
292                                 return false;
293                         }else if(cnt == 0){
294                                 // closed
295                                 return false;
296                         }
297                         *(buf+cnt) = '\0';
298                         ParseData(hDlg, buf);
299                 }
300                 break;
301
302         case FD_WRITE:
303                 break;
304
305         case FD_CLOSE:
306                 break;
307         }
308         return true;
309   }
310
311   LRESULT CALLBACK SendMailProc( HWND hDlg, UINT message, WPARAM wParam, LPARAM lP
    aram )
312   {
313         switch( message )
314         {
315         case WM_INITDIALOG:
316                 g_state = ST_START;
317                 return TRUE;
318
319         case WM_COMMAND:
320                 if( LOWORD(wParam) == IDOK ){
321                         OnOk(hDlg, message, wParam, lParam);
322                 }else if(LOWORD(wParam) == IDCANCEL){
```



```
323                         EndDialog(hDlg, LOWORD(wParam));
324                         return TRUE;
325                 }
326                 break;
327
328         case WM_DESTROY:
329                 delete[] g_pszRecvBuffer;
330                 g_pszRecvBuffer = NULL;
331                 break;
332
333         case UM_HOSTBYNAME:
334                 OnGetHostByName(hDlg, message, wParam, lParam);
335                 break;
336
337         case UM_SELECT:
338                 OnSelect(hDlg, message, wParam, lParam);
339                 break;
340
341         case UM_POSTEND:
342                 EndDialog(hDlg, LOWORD(wParam));
343                 return TRUE;
344         }
345       return FALSE;
346   }
347
```