```
1     #include <stdio.h>
2     #include <sys/types.h>
3     #include <sys/socket.h>
4     #include <netinet/in.h>
5     #include "bugpop3.h"

7     int main( int argc , char ** argv )
8  1  {
9  1    char labb[MAX_BUF];
10 1    struct sockaddr_in peer;
11 1    socklen_t peerLen = sizeof( peer );

13 1    if( getpeername( 0 , (struct sockaddr*) &peer , &peerLen ) )
14 2    {
15 2      writesockline( "-ERR Internal error : Not a socket\n" );
16 2      exit(1);
17 1    }

19 1    if( pop3authenticate( labb , sizeof(labb) ) )
20 2    {
21 2      pUserName = labb;
22 2      if( !lockuser( ) )
23 3      {
24 3        writesockline( "-ERR Unable to lock user" );
25 2      }
26 2      else
27 3      {
28 3        writesockline( "+OK User %s logged in" , labb );
29 3        readmaildrop();
30 3        pop3transaction();
31 2      }
32 1    }
33 1    return 0;
34    }
```

```
1     /*
2      * BTH/IPD Seclab Buggy POP3 Server
3      * pop3.c -- POP3 Logic Implementation
4      */

6     #include "bugpop3.h"
7     #include <stdio.h>
8     #include <string.h>
9     #include <sys/types.h>
10    #include <sys/param.h>
11    #include <pwd.h>
12    #include <unistd.h>
13    #include <fcntl.h>
14    #include <stdlib.h>
15    #include <errno.h>

17    char * pUserName;
18    _msg * messages;
19    int maildropsize, no_messages, saved_messages, mailboxdirty = 0;

21    void badcmd( char * msg )
22 1  {
23 1    static int nBadCmds = 0;
24 1    if( ++ nBadCmds >= 5 )
25 2    {
26 2      writesockline( "-ERR Too many bad commands; Bye Bye." );
27 2      exit(1);
28 1    }
29 1    else
30 1      writesockline( msg ? msg : "-ERR Invalid command" );
31    }
```

```
33    char * pop3authenticate( char * pUsername , int nMaxUserName )
34  1 {
35  1   int authorized = 0;
36  1   char username[32];
37  1   char buf[1024];
38  1   char * pThisPart = 0,
39  1        * pBuf;

41  1   writesockline(
                "+OK BTH IPD Security Lab Buggy POP3 Server %s Ready" , VERSION );
42  1   username[0] = 0;
43  1   while( !authorized )
44  2   {
45  2     pBuf = &buf[0];
46  2     if( ! readsockline( 0 , buf , MAX_BUF , TIMEOUT ) )
47  2       return NULL;

49  2     if( ! (pThisPart = strsep( &pBuf , " " ) ) )
50  2       badcmd(0);
51  2     else if( strncasecmp( pThisPart , "user" , 4 ) == 0 )
52  3     {
53  3       if( ! (pThisPart = strsep( &pBuf , " " )) )
54  4       {
55  4         badcmd( "-ERR USER requires a user name" );
56  4         continue;
57  3       }

59  3       if( ! username[0] )
60  4       {
61  4         strncpy( username , pThisPart , sizeof(username));
62  4         writesockline( "+OK User name accepted." );
63  3       }
64  3       else
65  3         badcmd( "-ERR User name already provided" );

67  2     }
68  2     else if( strncasecmp( pThisPart , "pass" , 4) == 0 )
69  3     {
70  3       if( ! (pThisPart = strsep( &pBuf , " " ) ) )
71  4       {
72  4         badcmd("-ERR PASS requires a password" );
73  4         continue;
74  3       }

76  3       if( ! username[0] )
77  3         badcmd( "-ERR Select a username first." );
78  3       else
79  4       {
80  4         char * pUserPass;
81  4         struct passwd * pPass =
82  4           getpwnam( username );

84  1         strncpy( pUsername, username,
85  4             nMaxUserName < sizeof( username ) ? nMaxUserName : sizeof(
                                                                  username ) );
86  4         username[0] = 0;

88  5         if( ! pPass ) {
89  5           sleep(1);
90  5           badcmd( "-ERR Authentication Error" );
91  5           continue;
92  4         }

94  4         pUserPass = crypt( pThisPart , pPass->pw_passwd );
```

```
 95  4          endpwent();

 97  4          if( strcmp( pUserPass , pPass->pw_passwd ) != 0 )
 98  5          {
 99  5            sleep(1);
100  5            badcmd( "-ERR Authentication Error" );
101  5            continue;
102  4          }
103  4          else
104  4            authorized = 1;
105  3        }
106  2      }
107  2      else if( strncasecmp( pThisPart , "quit" , 4 ) == 0 )
108  3      {
109  3        writesockline( "+OK Bye Bye" );
110  3        exit(0);
111  2      }
112  2      else
113  2        badcmd(0);
114  1    }

116  1    if( authorized )
117  1      return pUsername;
118  1    else
119  1      return NULL;
120    }
```

```
122     int lockuser( void )
123   1 {
124   1   int fd;
125   1   char buf[1020];
126     
127   1   snprintf( buf , sizeof( buf ) , "%s/locks/%s" , MAIL_BASE , pUserName );
128     
129   1   if( (fd=open( buf , O_WRONLY | O_EXCL | O_CREAT, 0600 ) ) <= -1 )
130   1     return -1; /* Unable to lock user */
131   1   else
132   2   {
133   2     atexit( unlockuser );
134   2     close( fd );
135   2     return 1;
136   1   }
137     }
```

```
139     void unlockuser( void  )
140   1 {
141   1   char buf[1024];
142
143   1   snprintf( buf , sizeof(buf) , "%s/locks/%s" , MAIL_BASE , pUserName );
144   1   unlink( buf );
145     }
```

```
147     int pop3transaction( void )
148   1 {
149   1   char buf[1024];
150   1   char * pThisPart, *pBuf;
151
152   1   while(4)
153   2   {
154   2     pBuf = &(buf[0]);
155   2     if( ! readsockline( 0 , buf , sizeof(buf) , TIMEOUT ) )
156   2       return 0;
157
158   2     if( ! (pThisPart = strsep( &pBuf , " " ) ) )
159   2       badcmd(0);
160   2     else if( strncasecmp( pThisPart , "noop" , 4 ) == 0 )
161   3     {
162   3       writesockline( "+OK " );
163   2     }
164   2     else if( strncasecmp( pThisPart , "rset" , 4 ) == 0 )
165   3     {
166   3       _msg * pMsg = messages->pNext;
167   3       for( ; pMsg ; pMsg = pMsg->pNext )
168   3         pMsg->deleted = 0;
169
170   3       mailboxdirty = 0;
171   2     }
172   2     else if( strncasecmp( pThisPart , "quit" , 4 ) == 0 )
173   3     {
174   3       dotransactionquit();
175   3       unlockuser();
176   3       exit(0);
177   2     }
178   2     else if( strncasecmp( pThisPart , "stat" , 4 ) == 0 )
179   3     {
180   3       writesockline( "+OK %i %i" , no_messages ,  maildropsize );
181   2     }
182   2     else if( strncasecmp( pThisPart , "list" , 4 ) == 0 )
183   3     {
184   3       int msgno = -1,i=0;
185   3       _msg * pMsg = messages->pNext;
186
187   3       if( (pThisPart = strsep( &pBuf , " " )) )
188   3         msgno = atoi( pThisPart );
189
190   3       if( msgno != -1 && ( msgno > no_messages || msgno <= 0 ) )
191   4       {
192   4         writesockline( "-ERR Bad message number" );
193   4         continue;
194   3       }
195
196   3       if( msgno == -1 )
197   4       {
198   4         for( i = 1 ; pMsg ; i++ , pMsg = pMsg->pNext )
199   4           writesockline( "%i %i" , i , pMsg->size );
200   3       }
201   3       else
202   4       {
203   4         for( i = 1 ; i != msgno ; i++ )
204   4           pMsg = pMsg->pNext;
205   4         writesockline( "%i %i" , i , pMsg->size );
206   3       }
207
208   3       writesockline( "." );
209
210   2     }
```

```
211 2     else if( strncasecmp( pThisPart , "top" , 3 ) == 0 )
212 3     {
213 3       int msgno, maxline;
214 3       if( ! (pThisPart = strsep( &pBuf , " " )) )
215 4       {
216 4         badcmd( "-ERR top requires a message number" );
217 4         continue;
218 3       }
219 3       else
220 3         msgno = atoi( pThisPart );

222 3       if( ! (pThisPart = strsep( &pBuf , " " )) )
223 4       {
224 4         badcmd( "-ERR top requires a line number" );
225 4         continue;
226 3       }
227 3       else
228 3         maxline = atoi( pThisPart );

230 3       if( msgno > no_messages )
231 4       {
232 4         badcmd( "-ERR No such message" );
233 4         continue;
234 3       }

236 3       fetchmessage( msgno , maxline );
237 2     }
238 2     else if( strncasecmp( pThisPart , "retr" , 4 ) == 0 )
239 3     {
240 3       int msgno;
241 3       if( ! (pThisPart = strsep( &pBuf , " " )) )
242 4       {
243 4         badcmd( "-ERR retr requires a message number" );
244 4         continue;
245 3       }
246 3       else
247 3         msgno = atoi( pThisPart );

249 3       if( msgno > no_messages )
250 4       {
251 4         badcmd( "-ERR No such message" );
252 4         continue;
253 3       }

255 3       fetchmessage( msgno , -1);
256 2     }
257 2     else if( strncasecmp( pThisPart , "dele" , 4 ) == 0 )
258 3     {
259 3       int msgno;
260 3       _msg * pMsg = messages->pNext;

262 3       if( ! (pThisPart = strsep( &pBuf , " " )) )
263 4       {
264 4         badcmd( "-ERR dele requires a message number" );
265 4         continue;
266 3       }
267 3       else
268 3         msgno = atoi( pThisPart );

270 3       if( msgno > no_messages || msgno <= 0)
271 4       {
272 4         badcmd( "-ERR No such message" );
273 4         continue;
274 3       }
```

```
276  3          while( --msgno )
277  3            pMsg = pMsg->pNext;

279  3          pMsg->deleted++;
280  3          mailboxdirty = 1;

282  3          writesockline( "+OK Done" );
283  2        }
284  2        else
285  3        {
286  3          badcmd(0);
287  2        }
288  1      }
289    }
```

```
291     void cleanup( void )
292   1 {
293   1   char namebuf[MAXPATHLEN+1];
295   1   snprintf( namebuf, sizeof(namebuf), "/var/tmp/%s.tmp", pUserName);
296   1   unlink( namebuf );
297     }
```

```
299     int readmaildrop( void )
300   1 {
301   1   char namebuf[MAXPATHLEN+1];
302   1   FILE *file;
303   1   _msg * curmsg;
304   1   _msgline * curline;
305   1   char * buf;
306   1   size_t nbuf;
307   1   int firstline;

309   1   maildropsize = no_messages = 0;
310   1   snprintf( namebuf , sizeof(namebuf) , "%s/%s" , MAIL_BASE , pUserName );
311   1   if( ! (file = fopen( namebuf , "r" ) ) )
312   1     return 1;

314   1   messages = (_msg*)malloc( sizeof( _msg ) );
315   1   curmsg = messages;
316   1   curmsg->pNext = 0;
317   1   curmsg->firstline = (_msgline *)malloc( sizeof( _msgline ) );
318   1   curmsg->size = 0;
319   1   curline = curmsg->firstline;

321   1   snprintf( namebuf, sizeof(namebuf), "/var/tmp/%s.tmp", pUserName );
322   1   if( (saved_messages=open(
                              namebuf, O_WRONLY | O_CREAT | O_EXCL, 0600 ) ) <= -1)
323   1     return 1; /* No old messages */

325   1   for( ;; firstline = 0 )
326   2   {
327   2     if( ! (buf = fgetln( file , &nbuf )))
328   3     {
329   3       curline->pLine = 0;
330   3       curline->pNext = 0;
331   3       break;
332   2     }

334   2     if( strncmp( buf , "From " , 5 ) == 0 )
335   3     {
336   3       no_messages++;
337   3       curline->pLine = 0;
338   3       curline->pNext = 0;
339   3       curmsg->pNext = (_msg*)malloc( sizeof( _msg ) );
340   3       curmsg = curmsg->pNext;
341   3       curmsg->size = 0;
342   3       curmsg->pNext = 0;
343   3       curmsg->firstline = (_msgline *)malloc( sizeof( _msgline ) );
344   3       curline = curmsg->firstline;
345   2     }
346   2     else
347   3     {
348   3       maildropsize += nbuf;
349   3       curmsg->size += nbuf;
350   2     }

352   2     curline->pLine = (char*)malloc( nbuf );
353   2     memcpy( curline->pLine , buf , nbuf );
354   2     curline->pLine[ nbuf -1 ] = 0;
355   2     curline->pNext = (_msgline *)malloc( sizeof( _msgline ) );
356   2     curline = curline->pNext;
357   1   }
358   1   atexit(cleanup);
359   1   fclose( file );
360   1   return 1;
361     }
```

```
364     void fetchmessage( unsigned msgno , int maxline )
365   1 {
366   1   unsigned no_iterated;
367   1   _msg * pMsg = messages;
368   1   _msgline * pLine;
369   1   int bodylines_sent = 0, in_head = 1;

371   1   if( msgno <= 0 )
372   2   {
373   2     badcmd( "-ERR No such message" );
374   2     return;
375   1   }

377   1   for( no_iterated = 0 ; no_iterated != msgno ; no_iterated++ )
378   1     pMsg = pMsg->pNext;

380   1   writesockline( "+OK " );

382   1   for( pLine = pMsg->firstline->pNext ; pLine->pLine; pLine = pLine->pNext )
383   2   {
384   2     if( strcmp( "" , pLine->pLine ) == 0 && in_head )
385   2       in_head = 0;

387   2     if( '.' == pLine->pLine[0] )
388   2       writesockline( ".%s" , pLine->pLine );
389   2     else
390   2       writesockline( "%s" , pLine->pLine );

392   2     if( ! in_head && maxline != -1  )
393   3     {
394   3       if( bodylines_sent++ >= maxline )
395   3         break;
396   2     }

398   1   }
399   1   writesockline( "." );

401     }
```

```
403     void dotransactionquit( void )
404   1 {
405   1   char filename[ MAXPATHLEN +1 ];
406   1   char mailboxname[ MAXPATHLEN +1 ];
407   1   FILE * pfile;
408   1   _msg * pMsg = messages;
409   1   _msgline * pLine;

411   1   if( ! mailboxdirty || ! messages )
412   2   {
413   2     writesockline( "+OK No messages was deleted in this session" );
414   2     return;
415   1   }

417   1   snprintf( filename , MAXPATHLEN , "/var/tmp/%s.tmp" , pUserName );
418   1   if( ! ( pfile = fopen( filename  , "w" ) ) )
419   2   {
420   2     writesockline( "-ERR Unable to write mailbox temp file" );
421   2     return;
422   1   }

424   1   for( pMsg = pMsg->pNext ; pMsg ; pMsg = pMsg->pNext )
425   2   {
426   2     if( pMsg->deleted )
427   2       continue;

429   2     for( pLine = pMsg->firstline->pNext ; pLine->pLine; pLine = pLine->pNext )
430   3     {
431   3       fprintf( pfile , "%s\n" , pLine->pLine );
432   2     }
433   1   }
434   1   fclose( pfile );
435   1   snprintf( mailboxname , sizeof(
                                  mailboxname) , "%s/%s" , MAIL_BASE , pUserName );

437   1   if( -1 == rename( filename , mailboxname ) )
438   1     writesockline( "+OK Internal error : No Emails removed" );
439   1   else
440   1     writesockline( "+OK Bye Bye " );

442     }
```

```
1    #include "bugpop3.h"
2    #include <sys/types.h>
3    #include <unistd.h>
4    #include <sys/poll.h>
5    #include <sys/socket.h>
6    #include <stdio.h>
7    #include <stdarg.h>
8    #include <string.h>
9    #include <sys/time.h>

11   int wait4io( const int sock , signed int * timeout )
12 1 {
13 1   struct pollfd fds;
14 1   int ret, secdiff, mysecdiff;
15 1   struct timeval start, stop;

17 1   fds.events = POLLIN | POLLERR | POLLHUP ;
18 1   fds.fd = sock;

20 1   gettimeofday( &start , NULL );
21 1   ret = poll( &fds , 1 , *timeout );
22 1   gettimeofday( &stop , NULL );

24 1   secdiff   = stop.tv_sec - start.tv_sec;
25 1   mysecdiff = stop.tv_usec - start.tv_usec;
26 1   *timeout -= ( secdiff * 1000 + mysecdiff / 1000 );
27 1   if( *timeout < 0 ) *timeout = 0;

29 1   if( fds.revents & ( POLLERR | POLLHUP ) )
30 1     return -1;

32 1   switch( ret )
33 2   {
34 2     case -1: /* error */
35 2       exit(1);
36 2     case 0:
37 2       return 0;
38 2     default:
39 2       return 1;
40 1   }
41   }
```

```
43    int writesockline( char * const pFormat , ... )
44  1 {
45  1   va_list args;
46  1   char buf[ MAX_BUF +1 ];
47  1   int len;

49  1   va_start( args , pFormat );

51  1   len = vsnprintf( buf , sizeof(buf) , pFormat , args );
52  1   write( 1 , buf , len );
53  1   write( 1 , "\r\n" , 2 );

55  1   va_end( args );
56  1   return len + 2;
57    }
```

```
 59    int readsockline(
               const int sock , char * pBuf , unsigned int nBuf , signed int timeout )
 60  1 {
 61  1   int nBufLeft = nBuf,
 62  1       nReadSession = 0,
 63  1       i;

 65  1   char * pSessBuf = pBuf;
 66  1   signed int timeoutLeft = timeout;

 68  1   for( ;; )
 69  2   {
 70  2     /*
 71  2      * Make sure we have sufficient buffer space
 72  2      */
 73  2     if( nBufLeft <= 0 )
 74  2       return 0;

 76  2     /*
 77  2      * Wait for incomming data on socket
 78  2      */
 79  2     if( 1 != wait4io( sock , &timeoutLeft ) )
 80  2       return 0;

 82  2     nReadSession = recv( sock, pSessBuf, nBufLeft, MSG_PEEK );
 83  2     if( nReadSession <= 0 )
 84  2       return 0;

 86  2     for( i = 0; i < nReadSession ; i++ )
 87  3     {
 88  3       int toRead;

 90  3       if( pSessBuf[i] == '\r' || pSessBuf[i] == '\n' )
 91  4       {
 92  4         toRead = i+1;
 93  4         if( (i+1 < nReadSession) &&
 94  4             (pSessBuf[i] == '\r' || pSessBuf[i] == '\n' ) )
 95  4           toRead++;

 97  4         recv( sock , pSessBuf , toRead , 0 );
 98  4         pSessBuf[i] = 0;
 99  4         return ( &pSessBuf[i] ) - pBuf ;
100  3       }
101  2     }
102  2     /* Partial input received (i.e. part of a line)
103  2      * Read all data so far and wait for more data
104  2      * in next cycle
105  2      */
106  2     nReadSession = recv( sock , pSessBuf , nReadSession , 0 );
107  2     if( -1 == nReadSession )
108  2       return 0;
109  2     pSessBuf += nReadSession;
110  2     nBufLeft -= nReadSession;
111  1   }

113  1   return 0;
114   }
```

```
1   #define MAX_BUF 1070
2   #define TIMEOUT 10000
3   #define VERSION "0.95.23 (Spring 2007)"
4   #define MAIL_BASE "/var/mail"
5   #define BODY 242

7   1 typedef struct _msgline {
8   1   char * pLine;
9   1   struct _msgline * pNext;
10  }_msgline;

12  1 typedef struct _msg {
13  1   int size;
14  1   int deleted;
15  1   _msgline * firstline;
16  1   struct _msg * pNext;
17  } _msg;


20  void dotransactionquit( void );
21  void fetchmessage( unsigned msgno , int maxline );
22  int pop3transaction( void );
23  int readsockline(
          const int sock , char * pBuf , unsigned int nBuf , signed int timeout );
24  int writesockline( char * const pFormat , ... );
25  char * pop3authenticate( char * pUsername , int nMaxUserName );
26  int lockuser( void );
27  void unlockuser( void );
28  int readmaildrop( void );

30  extern char * pUserName;
```

```
1     #include <stdio.h>
2     #include <sys/types.h>
3     #include <sys/socket.h>
4     #include <netinet/in.h>
5     #include "bugpop3.h"

7     int main( int argc , char ** argv )
8   1 {
9   1   char labb[MAX_BUF];
10  1   struct sockaddr_in peer;
11  1   socklen_t peerLen = sizeof( peer );

13  1   if( getpeername( 0 , (struct sockaddr*) &peer , &peerLen ) )
14  2   {
15  2     writesockline( "-ERR Internal error : Not a socket\n" );
16  2     exit(1);
17  1   }

19  1   if( pop3authenticate( labb , sizeof(labb) ) )
20  2   {
21  2     pUserName = labb;
22  2     if( !lockuser( ) )
23  3     {
24  3       writesockline( "-ERR Unable to lock user" );
25  2     }
26  2     else
27  3     {
28  3       writesockline( "+OK User %s logged in" , labb );
29  3       readmaildrop();
30  3       pop3transaction();
31  2     }
32  1   }
33  1   return 0;
34    }
```

```
 1     /*
 2      * BTH/IPD Seclab Buggy POP3 Server
 3      * pop3.c -- POP3 Logic Implementation
 4      */

 6     #include "bugpop3.h"
 7     #include <stdio.h>
 8     #include <string.h>
 9     #include <sys/types.h>
10     #include <sys/param.h>
11     #include <pwd.h>
12     #include <unistd.h>
13     #include <fcntl.h>
14     #include <stdlib.h>
15     #include <errno.h>

17     char * pUserName;
18     _msg * messages;
19     int maildropsize, no_messages, saved_messages, mailboxdirty = 0;

21     void badcmd( char * msg )
22  1  {
23  1    static int nBadCmds = 0;
24  1    if( ++ nBadCmds >= 5 )
25  2    {
26  2      writesockline( "-ERR Too many bad commands; Bye Bye." );
27  2      exit(1);
28  1    }
29  1    else
30  1      writesockline( msg ? msg : "-ERR Invalid command" );
31     }
```

```
33    char * pop3authenticate( char * pUsername , int nMaxUserName )
34  1 {
35  1   int authorized = 0;
36  1   char username[32];
37  1   char buf[1024];
38  1   char * pThisPart = 0,
39  1        * pBuf;

41  1   writesockline(
                "+OK BTH IPD Security Lab Buggy POP3 Server %s Ready" , VERSION );
42  1   username[0] = 0;
43  1   while( !authorized )
44  2   {
45  2     pBuf = &buf[0];
46  2     if( ! readsockline( 0 , buf , MAX_BUF , TIMEOUT ) )
47  2       return NULL;

49  2     if( ! (pThisPart = strsep( &pBuf , " " ) ) )
50  2       badcmd(0);
51  2     else if( strncasecmp( pThisPart , "user" , 4 ) == 0 )
52  3     {
53  3       if( ! (pThisPart = strsep( &pBuf , " " )) )
54  4       {
55  4         badcmd( "-ERR USER requires a user name" );
56  4         continue;
57  3       }

59  3       if( ! username[0] )
60  4       {
61  4         strncpy( username , pThisPart , sizeof(username));
62  4         writesockline( "+OK User name accepted." );
63  3       }
64  3       else
65  3         badcmd( "-ERR User name already provided" );

67  2     }
68  2     else if( strncasecmp( pThisPart , "pass" , 4) == 0 )
69  3     {
70  3       if( ! (pThisPart = strsep( &pBuf , " " ) ) )
71  4       {
72  4         badcmd("-ERR PASS requires a password" );
73  4         continue;
74  3       }

76  3       if( ! username[0] )
77  3         badcmd( "-ERR Select a username first." );
78  3       else
79  4       {
80  4         char * pUserPass;
81  4         struct passwd * pPass =
82  4           getpwnam( username );

84  4         strncpy( pUsername, username,
85  4             nMaxUserName < sizeof( username ) ? nMaxUserName : sizeof(
                                                                  username ) );
86  4         username[0] = 0;

88  5         if( ! pPass ) {
89  5           sleep(1);
90  5           badcmd( "-ERR Authentication Error" );
91  5           continue;
92  4         }

94  4         pUserPass = crypt( pThisPart , pPass->pw_passwd );
```

```
 95  4          endpwent();

 97  4          if( strcmp( pUserPass , pPass->pw_passwd ) != 0 )
 98  5          {
 99  5            sleep(1);
100  5            badcmd( "-ERR Authentication Error" );
101  5            continue;
102  4          }
103  4          else
104  4            authorized = 1;
105  3        }
106  2      }
107  2      else if( strncasecmp( pThisPart , "quit" , 4 ) == 0 )
108  3      {
109  3        writesockline( "+OK Bye Bye" );
110  3        exit(0);
111  2      }
112  2      else
113  2        badcmd(0);
114  1    }

116  1    if( authorized )
117  1      return pUsername;
118  1    else
119  1      return NULL;
120    }
```

```
122     int lockuser( void )
123   1 {
124   1   int fd;
125   1   char buf[1020];
126   1
127   1   snprintf( buf , sizeof( buf ) , "%s/locks/%s" , MAIL_BASE , pUserName );
128   1
129   1   if( (fd=open( buf , O_WRONLY | O_EXCL | O_CREAT, 0600 ) ) <= -1 )
130   1     return -1; /* Unable to lock user */
131   1   else
132   2   {
133   2     atexit( unlockuser );
134   2     close( fd );
135   2     return 1;
136   1   }
137     }
```

```
139   void unlockuser( void  )
140 1 {
141 1   char buf[1024];
143 1   snprintf( buf , sizeof(buf) , "%s/locks/%s" , MAIL_BASE , pUserName );
144 1   unlink( buf );
145   }
```

```
147     int pop3transaction( void )
148   1 {
149   1   char buf[1024];
150   1   char * pThisPart, *pBuf;

152   1   while(4)
153   2   {
154   2     pBuf = &(buf[0]);
155   2     if( ! readsockline( 0 , buf , sizeof(buf) , TIMEOUT ) )
156   2       return 0;

158   2     if( ! (pThisPart = strsep( &pBuf , " " ) ) )
159   2       badcmd(0);
160   2     else if( strncasecmp( pThisPart , "noop" , 4 ) == 0 )
161   3     {
162   3       writesockline( "+OK " );
163   2     }
164   2     else if( strncasecmp( pThisPart , "rset" , 4 ) == 0 )
165   3     {
166   3       _msg * pMsg = messages->pNext;
167   3       for( ; pMsg ; pMsg = pMsg->pNext )
168   3         pMsg->deleted = 0;

170   3       mailboxdirty = 0;
171   2     }
172   2     else if( strncasecmp( pThisPart , "quit" , 4 ) == 0 )
173   3     {
174   3       dotransactionquit();
175   3       unlockuser();
176   3       exit(0);
177   2     }
178   2     else if( strncasecmp( pThisPart , "stat" , 4 ) == 0 )
179   3     {
180   3       writesockline( "+OK %i %i" , no_messages ,  maildropsize );
181   2     }
182   2     else if( strncasecmp( pThisPart , "list" , 4 ) == 0 )
183   3     {
184   3       int msgno = -1,i=0;
185   3       _msg * pMsg = messages->pNext;

187   3       if( (pThisPart = strsep( &pBuf , " " )) )
188   3         msgno = atoi( pThisPart );

190   3       if( msgno != -1 && ( msgno > no_messages || msgno <= 0 ) )
191   4       {
192   4         writesockline( "-ERR Bad message number" );
193   4         continue;
194   3       }

196   3       if( msgno == -1 )
197   4       {
198   4         for( i = 1 ; pMsg ; i++ , pMsg = pMsg->pNext )
199   4           writesockline( "%i %i" , i , pMsg->size );
200   3       }
201   3       else
202   4       {
203   4         for( i = 1 ; i != msgno ; i++ )
204   4           pMsg = pMsg->pNext;
205   4         writesockline( "%i %i" , i , pMsg->size );
206   3       }

208   3       writesockline( "." );

210   2     }
```

```
211  2      else if( strncasecmp( pThisPart , "top" , 3 ) == 0 )
212  3      {
213  3        int msgno, maxline;
214  3        if( ! (pThisPart = strsep( &pBuf , " " )) )
215  4        {
216  4          badcmd( "-ERR top requires a message number" );
217  4          continue;
218  3        }
219  3        else
220  3          msgno = atoi( pThisPart );

222  3        if( ! (pThisPart = strsep( &pBuf , " " )) )
223  4        {
224  4          badcmd( "-ERR top requires a line number" );
225  4          continue;
226  3        }
227  3        else
228  3          maxline = atoi( pThisPart );

230  3        if( msgno > no_messages )
231  4        {
232  4          badcmd( "-ERR No such message" );
233  4          continue;
234  3        }

236  3        fetchmessage( msgno , maxline );
237  2      }
238  2      else if( strncasecmp( pThisPart , "retr" , 4 ) == 0 )
239  3      {
240  3        int msgno;
241  3        if( ! (pThisPart = strsep( &pBuf , " " )) )
242  4        {
243  4          badcmd( "-ERR retr requires a message number" );
244  4          continue;
245  3        }
246  3        else
247  3          msgno = atoi( pThisPart );

249  3        if( msgno > no_messages )
250  4        {
251  4          badcmd( "-ERR No such message" );
252  4          continue;
253  3        }

255  3        fetchmessage( msgno , -1);
256  2      }
257  2      else if( strncasecmp( pThisPart , "dele" , 4 ) == 0 )
258  3      {
259  3        int msgno;
260  3        _msg * pMsg = messages->pNext;

262  3        if( ! (pThisPart = strsep( &pBuf , " " )) )
263  4        {
264  4          badcmd( "-ERR dele requires a message number" );
265  4          continue;
266  3        }
267  3        else
268  3          msgno = atoi( pThisPart );

270  3        if( msgno > no_messages || msgno <= 0)
271  4        {
272  4          badcmd( "-ERR No such message" );
273  4          continue;
274  3        }
```

```
276  3          while( --msgno )
277  3            pMsg = pMsg->pNext;

279  3          pMsg->deleted++;
280  3          mailboxdirty = 1;

282  3          writesockline( "+OK Done" );
283  2        }
284  2        else
285  3        {
286  3          badcmd(0);
287  2        }
288  1      }
289    }
```

```
291     void cleanup( void )
292   1 {
293   1   char namebuf[MAXPATHLEN+1];
295   1   snprintf( namebuf, sizeof(namebuf), "/var/tmp/%s.tmp", pUserName);
296   1   unlink( namebuf );
297     }
```

```
299     int readmaildrop( void )
300   1 {
301   1   char namebuf[MAXPATHLEN+1];
302   1   FILE *file;
303   1   _msg * curmsg;
304   1   _msgline * curline;
305   1   char * buf;
306   1   size_t nbuf;
307   1   int firstline;

309   1   maildropsize = no_messages = 0;
310   1   snprintf( namebuf , sizeof(namebuf) , "%s/%s" , MAIL_BASE , pUserName );
311   1   if( ! (file = fopen( namebuf , "r" ) ) )
312   1     return 1;

314   1   messages = (_msg*)malloc( sizeof( _msg ) );
315   1   curmsg = messages;
316   1   curmsg->pNext = 0;
317   1   curmsg->firstline = (_msgline *)malloc( sizeof( _msgline ) );
318   1   curmsg->size = 0;
319   1   curline = curmsg->firstline;

321   1   snprintf( namebuf, sizeof(namebuf), "/var/tmp/%s.tmp", pUserName );
322   1   if( (saved_messages=open(
                                  namebuf, O_WRONLY | O_CREAT | O_EXCL, 0600 ) ) <= -1)
323   1     return 1; /* No old messages */

325   1   for( ;; firstline = 0 )
326   2   {
327   2     if( ! (buf = fgetln( file , &nbuf )))
328   3     {
329   3       curline->pLine = 0;
330   3       curline->pNext = 0;
331   3       break;
332   2     }

334   2     if( strncmp( buf , "From " , 5 ) == 0 )
335   3     {
336   3       no_messages++;
337   3       curline->pLine = 0;
338   3       curline->pNext = 0;
339   3       curmsg->pNext = (_msg*)malloc( sizeof( _msg ) );
340   3       curmsg = curmsg->pNext;
341   3       curmsg->size = 0;
342   3       curmsg->pNext = 0;
343   3       curmsg->firstline = (_msgline *)malloc( sizeof( _msgline ) );
344   3       curline = curmsg->firstline;
345   2     }
346   2     else
347   3     {
348   3       maildropsize += nbuf;
349   3       curmsg->size += nbuf;
350   2     }

352   2     curline->pLine = (char*)malloc( nbuf );
353   2     memcpy( curline->pLine , buf , nbuf );
354   2     curline->pLine[ nbuf -1 ] = 0;
355   2     curline->pNext = (_msgline *)malloc( sizeof( _msgline ) );
356   2     curline = curline->pNext;
357   1   }
358   1   atexit(cleanup);
359   1   fclose( file );
360   1   return 1;
361     }
```

```
364     void fetchmessage( unsigned msgno , int maxline )
365   1 {
366   1   unsigned no_iterated;
367   1   _msg * pMsg = messages;
368   1   _msgline * pLine;
369   1   int bodylines_sent = 0, in_head = 1;

371   1   if( msgno <= 0 )
372   2   {
373   2     badcmd( "-ERR No such message" );
374   2     return;
375   1   }

377   1   for( no_iterated = 0 ; no_iterated != msgno ; no_iterated++ )
378   1     pMsg = pMsg->pNext;

380   1   writesockline( "+OK " );

382   1   for( pLine = pMsg->firstline->pNext ; pLine->pLine; pLine = pLine->pNext )
383   2   {
384   2     if( strcmp( "" , pLine->pLine ) == 0 && in_head )
385   2       in_head = 0;

387   2     if( '.' == pLine->pLine[0] )
388   2       writesockline( ".%s" , pLine->pLine );
389   2     else
390   2       writesockline( "%s" , pLine->pLine );

392   2     if( ! in_head && maxline != -1  )
393   3     {
394   3       if( bodylines_sent++ >= maxline )
395   3         break;
396   2     }

398   1   }
399   1   writesockline( "." );

401     }
```

```
403     void dotransactionquit( void )
404   1 {
405   1   char filename[ MAXPATHLEN +1 ];
406   1   char mailboxname[ MAXPATHLEN +1 ];
407   1   FILE * pfile;
408   1   _msg * pMsg = messages;
409   1   _msgline * pLine;

411   1   if( ! mailboxdirty || ! messages )
412   2   {
413   2     writesockline( "+OK No messages was deleted in this session" );
414   2     return;
415   1   }

417   1   snprintf( filename , MAXPATHLEN , "/var/tmp/%s.tmp" , pUserName );
418   1   if( ! ( pfile = fopen( filename  , "w" ) ) )
419   2   {
420   2     writesockline( "-ERR Unable to write mailbox temp file" );
421   2     return;
422   1   }

424   1   for( pMsg = pMsg->pNext ; pMsg ; pMsg = pMsg->pNext )
425   2   {
426   2     if( pMsg->deleted )
427   2       continue;

429   2     for( pLine = pMsg->firstline->pNext ; pLine->pLine; pLine = pLine->pNext )
430   3     {
431   3       fprintf( pfile , "%s\n" , pLine->pLine );
432   2     }
433   1   }
434   1   fclose( pfile );
435   1   snprintf( mailboxname , sizeof(
                                  mailboxname) , "%s/%s" , MAIL_BASE , pUserName );

437   1   if( -1 == rename( filename , mailboxname ) )
438   1     writesockline( "+OK Internal error : No Emails removed" );
439   1   else
440   1     writesockline( "+OK Bye Bye " );

442     }
```

```
1     #include "bugpop3.h"
2     #include <sys/types.h>
3     #include <unistd.h>
4     #include <sys/poll.h>
5     #include <sys/socket.h>
6     #include <stdio.h>
7     #include <stdarg.h>
8     #include <string.h>
9     #include <sys/time.h>

11    int wait4io( const int sock , signed int * timeout )
12 1  {
13 1    struct pollfd fds;
14 1    int ret, secdiff, mysecdiff;
15 1    struct timeval start, stop;

17 1    fds.events = POLLIN | POLLERR | POLLHUP ;
18 1    fds.fd = sock;

20 1    gettimeofday( &start , NULL );
21 1    ret = poll( &fds , 1 , *timeout );
22 1    gettimeofday( &stop , NULL );

24 1    secdiff    = stop.tv_sec - start.tv_sec;
25 1    mysecdiff = stop.tv_usec - start.tv_usec;
26 1    *timeout -= ( secdiff * 1000 + mysecdiff / 1000 );
27 1    if( *timeout < 0 ) *timeout = 0;

29 1    if( fds.revents & ( POLLERR | POLLHUP ) )
30 1      return -1;

32 1    switch( ret )
33 2    {
34 2      case -1: /* error */
35 2        exit(1);
36 2      case 0:
37 2        return 0;
38 2      default:
39 2        return 1;
40 1    }
41    }
```

```
43
44  1  int writesockline( char * const pFormat , ... )
45  1  {
46  1    va_list args;
47  1    char buf[ MAX_BUF +1 ];
48  1    int len;
49  1
50  1    va_start( args , pFormat );
51  1
52  1    len = vsnprintf( buf , sizeof(buf) , pFormat , args );
53  1    write( 1 , buf , len );
54  1    write( 1 , "\r\n" , 2 );
55  1
56  1    va_end( args );
57  1    return len + 2;
58     }
```

```
59    int readsockline(
              const int sock , char * pBuf , unsigned int nBuf , signed int timeout )
60  1 {
61  1   int nBufLeft = nBuf,
62  1       nReadSession = 0,
63  1       i;

65  1   char * pSessBuf = pBuf;
66  1   signed int timeoutLeft = timeout;

68  1   for( ;; )
69  2   {
70  2     /*
71  2      * Make sure we have sufficient buffer space
72  2      */
73  2     if( nBufLeft <= 0 )
74  2       return 0;

76  2     /*
77  2      * Wait for incomming data on socket
78  2      */
79  2     if( 1 != wait4io( sock , &timeoutLeft ) )
80  2       return 0;

82  2     nReadSession = recv( sock, pSessBuf, nBufLeft, MSG_PEEK );
83  2     if( nReadSession <= 0 )
84  2       return 0;

86  2     for( i = 0; i < nReadSession ; i++ )
87  3     {
88  3       int toRead;

90  3       if( pSessBuf[i] == '\r' || pSessBuf[i] == '\n' )
91  4       {
92  4         toRead = i+1;
93  4         if( (i+1 < nReadSession) &&
94  4             (pSessBuf[i] == '\r' || pSessBuf[i] == '\n' ) )
95  4           toRead++;

97  4         recv( sock , pSessBuf , toRead , 0 );
98  4         pSessBuf[i] = 0;
99  4         return ( &pSessBuf[i] ) - pBuf ;
100 3       }
101 2     }
102 2     /* Partial input received (i.e. part of a line)
103 2      * Read all data so far and wait for more data
104 2      * in next cycle
105 2      */
106 2     nReadSession = recv( sock , pSessBuf , nReadSession , 0 );
107 2     if( -1 == nReadSession )
108 2       return 0;
109 2     pSessBuf += nReadSession;
110 2     nBufLeft -= nReadSession;
111 1   }

113 1   return 0;
114   }
```

```
1   #define MAX_BUF 1070
2   #define TIMEOUT 10000
3   #define VERSION "0.95.23 (Spring 2007)"
4   #define MAIL_BASE "/var/mail"
5   #define BODY 242

7   typedef struct _msgline {
8     char * pLine;
9     struct _msgline * pNext;
10  }_msgline;

12  typedef struct _msg {
13    int size;
14    int deleted;
15    _msgline * firstline;
16    struct _msg * pNext;
17  } _msg;


20  void dotransactionquit( void );
21  void fetchmessage( unsigned msgno , int maxline );
22  int pop3transaction( void );
23  int readsockline(
          const int sock , char * pBuf , unsigned int nBuf , signed int timeout );
24  int writesockline( char * const pFormat , ... );
25  char * pop3authenticate( char * pUsername , int nMaxUserName );
26  int lockuser( void );
27  void unlockuser( void );
28  int readmaildrop( void );

30  extern char * pUserName;
```