Dr. M. Grabert
Johannes Mayer

Systemnahe Software (SS 2001)

Abgabetermin: 9. Juli 2001

Beispiellösung

12 Jetzt aber mit Sockets! (5 Punkte)

Nachdem Sie nun schon in der Vorlesung kennengelernt haben, wie man mit Sockets umgeht, können Sie auch das letzte Übungsblatt ohne den Umweg über nc oder telnet lösen. Verändern Sie also Ihre Lösung oder die Beispiellösung zum letzten Übungsblatt so, dass statt des „Forkens“ mit darauffolgendem exec-Aufruf und der Kommunikation über zwei Pipes eine Socket-Verbindung zu dem POP3-Server hergestellt wird.

Lösung:

```
/*
 * AI4 SS01 Blatt 8 (c) mg,jm sai uni ulm 2001
 * Via POP3-Protokoll die Header der letzten ... Mails
 * holen und ausgeben - unter Verwendung von Sockets.
 */
#include <stdio.h>
#include <netdb.h>
#include <strings.h>

#define PORT     110       /* Portnummer */
#define LAST     5         /* von den letzten ... Mail */

/*
 * Kommando cmd (evtl. mit Argumenten arg1 und arg2) nach fpwrite
 * schreiben und auf eine Antwort (eine Zeile warten).
 * Rueckgabewert: true, falls Antwort mit "+OK" beginnt, und sonst false
 */
int putCommand(FILE *fpread, FILE *fpwrite, char *cmd, char *arg1, char *arg2) {
   char line[512], *s = line;
```

```
   if (arg1 == NULL)                          // Kommando senden
      fprintf(fpwrite, "%s\n", cmd);
   else if (arg2 == NULL)
      fprintf(fpwrite, "%s %s\n", cmd, arg1);
   else
      fprintf(fpwrite, "%s %s %s\n", cmd, arg1, arg2);

   fflush(fpwrite);                           // Sende-Puffer leeren

   fgets(line, sizeof(line), fpread);         // Antwort holen
   if (!strncmp("+OK", line, 3)) return 1;    // Antwort = +OK ?

   s += 4;                                    // "-ERR" ueberspringen
   while (*s == ' ') s++;                     // Leerzeichen danach auslassen

   fprintf(stderr, "Server Error: %s", s);    // Fehlermeldung ausgeben

   return 0;
}

/*
 * Gibt die Headerzeilen der Mail Nummer num, die mit From:, Subject: oder
 * Date: beginnen, auf die Standardausgabe aus.
 * Rueckgabewert: true, falls das POP3-Kommando top Erfolg hat, und sonst false
 */
int getMessage(FILE *fpread, FILE *fpwrite, int num) {
   char line[512];

   sprintf(line, "%d", num);          // Kommando top senden
   if (!putCommand(fpread, fpwrite, "top", line, "0")) return 0;

      /*
       * Zeile fuer Zeile lesen, bis eine Zeile kommt, deren
       * erstes Zeichen "." ist (und deren 2. Zeichen nicht "." ist).
       * Nur die relevanten From:-, Subject:- und Date:-Zeilen
       * ausgeben.
       */
   while (1) {
      fgets(line, sizeof(line), fpread);
      if (line[0] == '.' && line[1] != '.') break;
      if (!strncmp("From:", line, 5) ||
          !strncmp("Subject:", line, 8) ||
          !strncmp("Date:", line, 5))
         printf("%s", line);
   }
```

```
   return 1;
}

void pop3client(FILE *fpread, FILE *fpwrite, char *user, char *passwd) {
   char line[512];
   int n, num[LAST], i, j, k;

      /*
       * Am Anfang alle Zeilen weglesen, bis eine Zeile kommt,
       * die mit "+OK" beginnt.
       */
   while (fgets(line, sizeof(line), fpread) && strncmp("+OK", line, 3));

      /*
       * Authentisieren und Liste der Mail-Nummern anfordern
       */
   if (!putCommand(fpread, fpwrite, "user", user, NULL)) return;
   if (!putCommand(fpread, fpwrite, "pass", passwd, NULL)) return;
   if (!putCommand(fpread, fpwrite, "list", NULL, NULL)) return;

      /*
       * Nummern der Mails einlesen (wird durch eine Zeile, die
       * mit "." und nicht ".." beginnt beendet).
       * Nur die letzten LAST Nummern speichern.
       */
   n = 0;
   while (1) {
      fgets(line, sizeof(line), fpread);
      if (line[0] == '.' && line[1] != '.') break;
      sscanf(line, " %d ", &num[n++ % LAST]);
   }

      /*
       * LAST neueste Mails ausgeben (falls es so viele gibt).
       */
   k = n > LAST ? LAST : n;   // Anzahl auszugebender Header
   i = n - k;                 // neueste Mail zuletzt
   for (j = 1; j <= k; j++) {
      if (j > 1) puts("----------------------------------------");
      if (!getMessage(fpread, fpwrite, num[i++ % LAST]))
         return;
   }

      /*
       * Sitzung mit dem POP3-Server beenden.
       */
```

```
   if (!putCommand(fpread, fpwrite, "quit", NULL, NULL)) return;
}

int main(int argc, char **argv) {
   struct sockaddr_in addr;
   struct hostent *hp;
   int fd;
   FILE *fpread, *fpwrite;

   if (argc != 4) {
      fprintf(stderr, "Usage: %s <pop3 server> <user name> <password>\n", argv[0]);
      exit(1);
   }

   if (!(hp = gethostbyname(argv[1])))   // IP-Adresse des Servers holen
      fprintf(stderr, "unknown host: %s\n", argv[1]),
      exit(1);

   bzero(&addr, sizeof(addr));           // mit 0en fuellen
   addr.sin_family = AF_INET;            // Adresse fuer Internet-Verbindung
   addr.sin_port = htons(PORT);          // Port eintragen (Network Byte Order!)
   bcopy(hp->h_addr, &addr.sin_addr, hp->h_length); // Serveradresse eintragen

        /*
         * Endpunkt fuer eine Kommunikation ueber Sockets erzeugen.
         * (zuverlaessige Internetverbindung)
         */
   if ((fd = socket(PF_INET, SOCK_STREAM, 0)) < 0)
      perror("socket"),
      exit(1);

        /*
         * Endpunkt mit dem Port des angegebenen Rechners verbinden.
         */
   if (connect(fd, (struct sockaddr *) &addr, sizeof(addr)) <0) // ... und a
      perror("connect"),
      exit(1);

   fpread = fdopen(fd, "r");
   fpwrite = fdopen(fd, "w");

     /*
      * argv[2] = Benutzername, argv[3] = Passwort
      */
   pop3client(fpread, fpwrite, argv[2], argv[3]);
```

```
   fclose(fpwrite);
   fclose(fpread);

   return 0;
}
```

13 Hol' die Mails, Browser! (15 Punkte)

Wow, jetzt haben Sie also einen eigenen POP3-Client, mit dem Sie die Header (Absender, Betreff und Datum) der letzten Mails abholen können. Das ist doch was, oder?! Naja, aber besonders schön sieht die Ausgabe noch nicht aus. Wie wär's denn, wenn die Ausgabe im Browser erfolgen würde. Das ist doch 'mal eine gute Idee. Und das geht zudem noch verblüffend einfach – wie Sie ja in der Vorlesung gesehen haben.
Also nichts wie ans Werk. Schreiben Sie also einen Server, der auf Port ...[1] lauscht. Sobald sich ein Client mit Ihm verbindet, liest er eine POST-Anfrage gemäß HTTP-Protokoll. Eine solche Anfrage besteht aus einem Header, dessen letzte Zeile eine Leerzeile ist (nur mit „`\r`“ bzw. „`\n`“ am Ende). Danach kommen genau so viele Zeichen, wie im Header angegeben ist (in der Header-Zeile, die mit `Content-Length:` beginnt). Nach diesen Zeichen kommt kein Zeilentrenner mehr! Diese zuletzt gelesenen Zeichen enthalten die Variablen. Die einzelnen Variablen sind voneinander durch `&` getrennt. Zwischen dem Name und dem Wert einer Variablen steht ein `=`-Zeichen. Die Variable `host` enthält den Namen des POP3-Servers. In den Variablen `login` und `pass` sind die Zugangsdaten des POP3-Servers enthalten. (Die Reihenfolge der Variablen kann beliebig sein!)
Beispiel für eine POST-Anfrage nach dem HTTP-Protokoll:

```
POST / HTTP/1.0
User-Agent: Mozilla/4.7 [en] (X11; U; SunOS 5.8 sun4u; Nav)
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: de, en
Accept-Charset: iso-8859-1,*,utf-8
Content-Type: application/x-www-form-urlencoded
Content-Length: 32
Via: 1.0 suckit.rz.uni-ulm.de:3128 (Squid/2.3.STABLE2)
X-Forwarded-For: 134.60.66.5
Host: thales.mathematik.uni-ulm.de:11023
Cache-Control: max-age=259200

host=turing&login=test&pass=test
```

Nun sollen von Ihrem POP3-Server die Header der neuesten Mails von dem angegebenen POP3-Account abgeholt und als HTML-Dokument ausgegeben werden. Hierzu können Sie Ihre Lösung für die vorige Aufgabe als Unterprogramm verwenden. Sie müssen es „nur“ so abändern, dass es ein HTML-Dokument produziert. Der Client, welcher Ihrem Server die POST-Anfrage geschickt

[1] Portnummer > 1024; bedenken Sie bei einem evtl. Fehler, dass dieser bereits verwendet wird und versuchen Sie es mit einem anderen.

hat, erwartet nach dem HTTP-Protokoll aber nicht nur das HTML-Dokument, sondern auch eine kurze Antwort, welche aus einem Header und – getrennt durch eine Leerzeile – einem HTML-Dokument besteht.
Beispiel für eine (minimale) Antwort auf eine POST-Anfrage:

```
HTTP/1.1 200 OK
Content-Type: text/html

<HTML>
   <BODY>
   ... die Header der neuesten Mails ...
   </BODY>
</HTML>
```

Nachdem Ihr Server die POST-Anfrage mit seiner Antwort quittiert hat, soll er die nächste Anfrage entgegennehmen.
Tipps: Um komfortabel eine POST-Anfrage an Ihren Server schicken zu können, benötigen sie ein *HTML-Formular*. Bei diesem müssen sie nur noch
`action="http://thales.mathematik.uni-ulm.de:11023/"`
durch den Rechner, auf dem Ihr Server läuft, und den Port, den Ihr Server verwendet, ersetzen. *Hier* finden Sie außerdem ein kleines HTML-Dokument, das sie als Grundlage für die Ausgabe der Mail-Header verwenden können. Wenn Sie die Ausgabe des Servers aufwendiger gestalten wollen, können Sie den *SELFHTML-Kurs* ja 'mal anschauen.
Die Werte der Variablen, welche Sie via POST erhalten, sind „URL-codiert". Zu deren Decodierung und zur Ausgabe von Text als HTML erhalten Sie zwei *C-Funktionen*. Zum Zerlegen des Strings, der die Variablen (der POST-Anfrage) enthält, können Sei die Funktion `strtok` einsetzen. Bei Verwendung von File-Pointern sollten sie nach jedem Schreiben den Puffer mittels `fflush()` leeren. Es ist empfehlenswert, die File-Deskriptoren mittels `fdopen()` in File-Pointer zu konvertieren, und dann mit diesen zu arbeiten.

Lösung:

```
/*
 * AI4 SS01 Blatt 8 (c) jm sai uni ulm 2001
 * HTTP-Server, welcher ausschliesslich - entsprechend den Parametern
 * von POST - eine Verbindung zu einem POP3-Server aufmacht und die
 * Header der Mails in der Inbox dann als HTML-Seite ausgibt.
 */
#include <stdio.h>
#include <netdb.h>
#include <strings.h>

#define PORT     11023      /* Portnummer fuer den Server */
#define LAST     500        /* die Header aller Mails ausgeben ;-) */

/*
```

```
 * Dekodieren eines URL-codierten Strings, d.h. + wird durch Leerzeichen
 * ersetzt, %xy durch das Zeichen mit dem ASCII-Code hexadezimal xy und
 * alles andere bleibt so wie es ist. Die Veraenderung wird dabei im
 * uebergebenen Stringbuffer durchgefuehrt!
 */
char *decodeURL(char *url) {
   char *read = url;      // Lesezeiger fuer die Decodierung
   char *write = url;     // Schreibzeiger fuer die Decodierung
   char buf[] = "0x..";   // Puffer zur Konvertierung von Hex-Zahlen
   int i;

   while (*read != 0) {
      if (*read == '+')            // '+' durch Leerzeichen ersetzen
         *write = ' ';
      else if (*read == '%') {     // %.. durch Zeichen mit
         buf[2] = *++read;         // ASCII-Code .. (hexadezimal) ersetzen
         buf[3] = *++read;
         sscanf(buf, " %x ", &i);
         *write = i;
      }
      else
         *write = *read;           // ... und den Rest nur kopieren
      read++;
      write++;
   }

   *write = 0;            // konvertierten String mit 0-Byte beenden

   return url;
}

/*
 * Umsetzung von Sonderzeichen in s in HTML-Befehle. Die Aenderungen
 * werden nicht im uebergebenen String durchgefuehrt, sondern in einem
 * eigenen! Ein Zeiger auf diesen String ist dann der Rueckgabewert.
 * VORSICHT: Bei nochmaligem Aufruf dieser Funktion wird der String,
 *   welcher beim vorigen Aufruf zurueckgegeben wurde, veraendert! (=> static!)
 */
char *codeHTML(char *s) {
   static char buf[1000];
   char *read = s;      // Lesezeiger
   char *write = buf;   // Schreibzeiger

   while (*read != 0) {
      if (*read == '<') {         // '<' codieren
         sprintf(write, "&lt;");
```

```
         write += 4;
      }
      else if (*read == '>') {    // '>' codieren
         sprintf(write, "&gt;");
         write += 4;
      }
      else
         *write++ = *read;        // anderen Zeichen kopieren
      read++;
   }

   *write = 0;                    // Ergebnisstring mit einem 0-Byte terminieren

   return buf;
}

/*
 * Kommando cmd (evtl. mit Argumenten arg1 und arg2) nach fpwrite
 * schreiben und auf eine Antwort (eine Zeile warten). Evtl. Fehlermeldung
 * als HTML nach out ausgeben.
 * Rueckgabewert: true, falls Antwort mit "+OK" beginnt, und sonst false
 */
int putCommand(FILE *fpread, FILE *fpwrite, char *cmd, char *arg1, char *arg2, FILE *out)
   char line[512], *s = line;

   if (arg1 == NULL)                            // Kommando senden
      fprintf(fpwrite, "%s\n", cmd);
   else if (arg2 == NULL)
      fprintf(fpwrite, "%s %s\n", cmd, arg1);
   else
      fprintf(fpwrite, "%s %s %s\n", cmd, arg1, arg2);

   fflush(fpwrite);                             // Sende-Puffer leeren

   fgets(line, sizeof(line), fpread);           // Antwort holen
   if (!strncmp("+OK", line, 3)) return 1;      // Antwort = +OK ?

   s += 4;                                      // "-ERR" ueberspringen
   while (*s == ' ') s++;                       // Leerzeichen danach auslassen

      /*
       * Fehlermeldung vom POP3-Server zusammen mit dem
       * Kommando, nach dem der Fehler auftrat, als HTML-Text ausgeben.
       */
   fprintf(out, "<h1><b>POP3-Server Error:</b> <i>%s</i></h1>", s);
   fprintf(out, "<h2>(after the following POP3 command: <code>%s", cmd);
```

```
   if (arg1) {
      fprintf(out, " %s", arg1);
      if (arg2) fprintf(out, " %s", arg2);
   }
   fprintf(out, "</code>)</h2>");

   return 0;
}

/*
 * Gibt die Headerzeilen der Mail Nummer num, die mit From:, Subject: oder
 * Date: beginnen, nach out aus. Die Ausgabe ist fuer eine Zeile einer
 * Tabelle in HTML bestimmt.
 * Rueckgabewert: true, falls das POP3-Kommando top Erfolg hat, und sonst false
 */
int getMessage(FILE *fpread, FILE *fpwrite, int num, FILE *out) {
   char line[512], from[200], subject[200], date[200];
   int isFrom = 0, isSubject = 0, isDate = 0;

   sprintf(line, "%d", num);         // Kommando top senden
   if (!putCommand(fpread, fpwrite, "top", line, "0", out)) return 0;

      /*
       * Zeile fuer Zeile lesen, bis eine Zeile kommt, deren
       * erstes Zeichen "." ist (und deren 2. Zeichen nicht "." ist).
       * Nur die relevanten From:-, Subject:- und Date:-Zeilen
       * ausgeben.
       */
   while (1) {
      fgets(line, sizeof(line), fpread);
      if (line[0] == '.' && line[1] != '.') break;  // bei "."-Zeile abbrechen
      if (!strncmp("From:", line, 5))
         strcpy(from, line+5),
         isFrom = 1;
      if (!strncmp("Subject:", line, 8))
         strcpy(subject, line+8),
         isSubject = 1;
      if (!strncmp("Date:", line, 5))
         strcpy(date, line+5),
         isDate = 1;
   }

      /*
       * Eine Zeile der HTML-Tabelle ausgeben mit: Absender, Betreff und Datum.
       */
   fprintf(out, "<tr><td>");
```

```
    if (isFrom)    fprintf(out, "%s", codeHTML(from));
    fprintf(out, "</td><td>");
    if (isSubject) fprintf(out, "<b>%s</b>", codeHTML(subject));
    fprintf(out, "</td><td>");
    if (isDate)    fprintf(out, "%s", codeHTML(date));
    fprintf(out, "</td></tr>\n");
    fflush(out);

    return 1;
}

/*
 * Fuehrt eine Sitzung nach dem POP3-Protokoll durch, wobei die Header
 * der letzten LAST Mails gelesen und nach out ausgegeben werden (in HTML).
 */
void pop3session(FILE *fpread, FILE *fpwrite, char *user, char *passwd, FILE *out) {
    char line[512];
    int n, num[LAST], i, j, k;

        /*
         * Am Anfang alle Zeilen weglesen, bis eine Zeile kommt,
         * die mit "+OK" beginnt.
         */
    while (fgets(line, sizeof(line), fpread) && strncmp("+OK", line, 3));

        /*
         * Authentisieren und Liste der Mail-Nummern anfordern
         */
    if (!putCommand(fpread, fpwrite, "user", user, NULL, out)) return;
    if (!putCommand(fpread, fpwrite, "pass", passwd, NULL, out)) return;
    if (!putCommand(fpread, fpwrite, "list", NULL, NULL, out)) return;

        /*
         * Nummern der Mails einlesen (wird durch eine Zeile, die
         * mit "." und nicht ".." beginnt beendet).
         * Nur die letzten LAST Nummern speichern.
         */
    n = 0;
    while (1) {
        fgets(line, sizeof(line), fpread);
        if (line[0] == '.' && line[1] != '.') break;
        sscanf(line, " %d ", &num[n++ % LAST]);
    }

        /*
         * LAST neueste Mails ausgeben (falls es so viele gibt).
```

```
     */
   fprintf(out, "<table>"); fflush(out);      // HTML-Tabelle beginnen
   k = n > LAST ? LAST : n;   // Anzahl auszugebender Header
   i = n;
   for (j = 1; j <= k; j++) {
      if (!getMessage(fpread, fpwrite, num[--i % LAST], out))
         return;
   }
   fprintf(out, "</table>"); fflush(out);      // HTML-Tabelle beenden

      /*
       * Sitzung mit dem POP3-Server beenden.
       */
   if (!putCommand(fpread, fpwrite, "quit", NULL, NULL, out)) return;
}

/*
 * Zu server eine Socket-Verbindung herstellen und via POP3-Protokoll
 * die Header der LAST neuesten Mails holen und nach out in HTML ausgeben.
 */
void pop3client(char *server, int port, char *login, char *passwd, FILE *out) {
   struct sockaddr_in addr;
   struct hostent *hp;
   int fd;
   FILE *fpread, *fpwrite;

   if (!(hp = gethostbyname(server)))    // IP-Adresse des Servers holen
      fprintf(stderr, "unknown host: %s\n", server),
      exit(1);

   bzero(&addr, sizeof(addr));           // mit 0en fuellen
   addr.sin_family = AF_INET;            // Adresse fuer Internet-Verbindung
   addr.sin_port = htons(port);          // Port eintragen (Network Byte Order!)
   bcopy(hp->h_addr, &addr.sin_addr, hp->h_length); // Serveradresse eintragen

      /*
       * Endpunkt fuer eine Kommunikation ueber Sockets erzeugen.
       * (zuverlaessige Internetverbindung)
       */
   if ((fd = socket(PF_INET, SOCK_STREAM, 0)) < 0)
      perror("socket"),
      exit(1);

      /*
       * Endpunkt mit dem Port des angegebenen Rechners verbinden.
       */
```

```
   if (connect(fd, (struct sockaddr *) &addr, sizeof(addr)) < 0)
      perror("connect"),
      exit(1);

   fpread = fdopen(fd, "r");
   fpwrite = fdopen(fd, "w");

      /*
       * Geringfuegig modifizierte Loesung zu Aufgabe 12 aufrufen.
       */
   pop3session(fpread, fpwrite, login, passwd, out);

   fclose(fpwrite);
   fclose(fpread);
}

int main() {
   struct sockaddr_in addr;
   int sfd, fd;
   FILE *fpread, *fpwrite;
   int optval = 1, len = 0;
   char buf[500];
   char *token, *host, *login, *pass;
   int isHost = 0, isLogin = 0, isPass = 0;
   int i;

   bzero(&addr, sizeof(addr));    // mit 0en fuellen
   addr.sin_family = AF_INET;     // Adresse fuer Internet-Verbindung
   addr.sin_port = htons(PORT);   // Port eintragen (Network Byte Order!)

      /*
       * Endpunkt fuer eine Socketkommunikation erzeugen.
       */
   if ((sfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
      perror("socket"),
      exit(1);

      /*
       * Rasche Wiederverwendung des Server-Ports einstellen.
       */
   if (setsockopt(sfd, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(optval)) < 0)
       perror("setsockopt"),
       exit(1);

      /*
       * Endpunkt an einen Port auf dem lokalen Rechner "binden".
```

```
     */
if (bind(sfd, (struct sockaddr *) &addr, sizeof(addr)) < 0)
     perror("bind"),
     exit(1);

   /*
    * Maximale Anzahl gleichzeitiger Verbindungen einstellen.
    * (Einer wird bedient und die anderen warten...)
    */
if (listen(sfd, SOMAXCONN) < 0)
   perror("listen"),
   exit(1);

   /*
    * Blockiert so lange bis eine Anfrage kommt. Diese
    * wird dann in der while-Schleife abgearbeitet.
    */
while ((fd = accept(sfd, NULL, NULL)) >= 0) {
   fpread = fdopen(fd, "r");
   fpwrite = fdopen(fd, "w");

      /*
       * Anfrage lesen (HTTP-Protokoll)
       * (=> Ende der Anfrage ist eine Leerzeile - d.h. nur Zeilentrenner)
       */
   while (1) {
      if (!fgets(buf, sizeof(buf), fpread)) break;    // bei EOF ist Schluss
         /*
          * Linefeed und Newline am Stringende entfernen.
          */
      i = strlen(buf);
      while (i > 0 && (buf[i-1] == '\n' || buf[i-1] == '\r')) buf[--i] = 0;
         /*
          * i == 0 => Leerzeile => Ende des Headers
          */
      if (i == 0) break;
         /*
          * "Content-Length:"-Feld enthaelt die Laenge (in Bytes)
          * des Teils nach dem Header (und der Leerzeile).
          */
      if (!strncmp(buf, "Content-Length:", 15))
         sscanf(buf+15, " %d ", &len);
   }

   if (sizeof(buf) < len+1)
      fprintf(stderr, "buffer too small\n"),
```

```
   exit(1);
   /*
    * Genau len Zeichen in buf einlesen. (Dazu kommt noch
    * ein abschliessendes 0-Byte; macht also insgesamt
    * len+1 verwendete Zeichen von buf!)
    */
fgets(buf, len+1, fpread);

   /*
    * Die Antwort zerlegen in die Bestandteile
    * host=<POP3-Server>, login=<Benutzername>
    * und pass=<Password> und DANACH erst
    * die URL-Decodierung durchfuehren!
    */
token = strtok(buf, "&");
while (token) {
   if (!strncmp(token, "host=", 5))
      (host = token + 5),
      decodeURL(host),
      isHost = 1;
   if (!strncmp(token, "login=", 6))
      (login = token + 6),
      decodeURL(login),
      isLogin = 1;
   if (!strncmp(token, "pass=", 5))
      (pass = token + 5),
      decodeURL(pass),
      isPass = 1;
   token = strtok(NULL, "&");
}

   /*
    * Zunaechst den Header nach dem HTTP-Protokoll ...
    */
fprintf(fpwrite, "HTTP/1.1 200 OK\n");
fprintf(fpwrite, "Content-Type: text/html\n\n");
   /*
    * ... dann den Anfang des HTML-Dokuments ...
    */
fprintf(fpwrite, "<html><body>\n");
fflush(fpwrite);

   /*
    * Nur falls Host, Login und Passwort angegeben
    * wurden, wir auch eine Verbindung zum POP3-Server
    * hergestellt. Andernfalls erhaelt der Benutzer
```

```
         * eine Fehlermeldung.
         */
      if (isHost && isLogin && isPass)
         pop3client(host, 110, login, pass, fpwrite);
      else
         fprintf(fpwrite, "<h1>Error: host, login or password unspecified within POST</h1>

         /*
          * ... und zuletzt das Ende des HTML-Dokuments ...
          */
      fprintf(fpwrite, "</body></html>\n");
      fflush(fpwrite);

      fclose(fpwrite);
      fclose(fpread);
   }

   return 0;
}
```