```perl
#!/usr/bin/perl

# MicroWeb - ein minimalistischer Webserver zu Lehrzwecken
# (C) 2006 Veit Wahlich

package MicroWeb;
our $VERSION='1.1';


# Benoetigte Module und Pragmas importieren:
use strict;             # Wir programmieren stets strict
use warnings;           # und haetten gerne Warnmeldungen.
use IO::Socket::INET;   # Wir moechten IP-Sockets
use IO::File;           # und Dateien verwenden.
use POSIX qw(strftime); # Fuer saubere Timestamps benutzen wir strftime().


# globale Variable, speichert die Anzahl der aktuell laufenden Kindprozesse
my $children=0;

# globale Konfiguration
my $conf={
    bind_address       => '0.0.0.0',        # IP-Adresse, an die wir binden.
    bind_port          => 8080,             # TCP-Port, an den wir binden.
    default_root       => 'htdocs/default', # Default-Basisverzeichnis der
                                            #   Dokumente.
    vhost_root         => 'htdocs/%s',      # Basisverzeichnis der Dokumente
                                            #   mit VHost (%s wird durch
                                            #   Hostname ersetzt).
    index_file         => 'index.html',     # Datei zu laden wenn nur ein
                                            #   Verzeichnis angefragt wurde.
    max_connections    => 10,               # Max. Anzahl paralleler
                                            #   Verbindungen (== Kindprozesse).
    max_url_length     => 4096,             # Max. Laenge einer URL in Bytes.
    max_headers_length => 4096,             # Max. Groesse der Headers in Bytes.
    client_timeout     => 5,                # Max. Wartezeit fuer Empfang von
                                            #   Anfragen.
    mime_types => {                         # Relationen Dateierweiterung zu
                                            #   MIME-Typ.
        html  => 'text/html',
        htm   => 'text/html',
        txt   => 'text/plain',
        css   => 'text/css',
        xml   => 'text/xml',
        xsl   => 'text/xml',
        jpg   => 'image/jpeg',
        jpeg  => 'image/jpeg',
        png   => 'image/png',
        gif   => 'image/gif',
        mp3   => 'audio/mpeg',
        wav   => 'audio/x-wav',
        mid   => 'audio/midi',
        mpg   => 'video/mpeg',
        mpeg  => 'video/mpeg',
        avi   => 'video/x-msvideo',
        ogg   => 'application/ogg',
        js    => 'application/x-javascript',
        swf   => 'application/x-shockwave-flash',
        gz    => 'application/x-gzip',
        tgz   => 'application/x-gzip',
        bz2   => 'application/x-bzip2',
        tbz2  => 'application/x-bzip2',
        tar   => 'application/x-tar',
        zip   => 'application/zip',
        '*'   => 'application/octet-stream'
    },
};


# Variablen wegen Verwendung in Signalhandlern in globalem Scope:
my $socket;             # Enthaelt spaeter den Listener.
my $client;             # Enthaelt spaeter das Filehandle des aktuellen Clients.


sub main(){

  my $pid;              # Enthaelt spaeter die PID des Client-Kindprozesses.

  # Ein Signalhandler faengt SIGTERM und SIGINT ab:
  $SIG{TERM}=$SIG{INT}=sub{
      # Listener sauber beenden:
      logError('Server received SIGTERM/SIGINT - exiting gracefully');
      $socket->shutdown(2);
      $socket->close();
      STDOUT->close();
      STDERR->close();
      exit(0);
    };

  # Bei SIGCHLD wurde ein Kindprozess beendet und muss geerntet werden:
  $SIG{CHLD}=\&reapPid;

  # Listener anlegen...
  $socket=new IO::Socket::INET(
      Listen    => 5,
      LocalAddr => $conf->{bind_address},
      LocalPort => $conf->{bind_port},
      Proto     => 'tcp',
      Reuse     => 1
    );

  # und auf Erfolg ueberpruefen.
  unless(defined($socket)){
    die("Unable to bind port to address: $!\n");

  }

  # Server-Endlosschleife betreten
  while(1){

    # Auf neue Client-Verbindung warten und in $client speichern:
    $client=$socket->accept();

    # Moderne Betriebssysteme geben ein $client==undef zurueck, wenn $socket
    # noch nicht wieder bereit ist - dann die Schleife von vorn beginnen:
    # *FIXME*: Hier sollte man eigentlich kurz warten, sonst 100% CPU-Last,
    # bis Socket wieder Verbindungen annimmt!
    next unless(defined($client));

    # Nur neue Verbindungen verarbeiten, wenn nicht zu viele Verbindungen
    # aktiv sind:
    if($children < $conf->{max_connections}){

      # Prozess duplizieren und PID speichern.
      $pid=fork();

      # Wenn fork() erfolgreich ist, ist $pid definiert:
      if(defined($pid)){
```

```
129
130           # Und wenn $pid == 0 ist, sind wir gerade im Kindprozess - sonst sind
131           # wir im Mutterprozess.
132           if($pid == 0){
133
134             # Im Client-Kindprozess brauchen wir den Listener nicht.
135             $socket->close();
136
137             # Signalhandler im Client sind anders als die im Server:
138             $SIG{TERM}=$SIG{INT}=sub{
139                 logError('Child '.$$
140                   .' received SIGTERM/SIGINT - exiting gracefully');
141                 exitChild();
142               };
143
144             # Uebergebe Verarbeitung der Verbindung an die Zulieferer-Funktion:
145             processConnection($client);
146
147             # Ist die Verbindung verarbeitet, kann die Client-Verbindung
148             # geschlossen und der Prozess beendet werden:
149             exitChild();
150
151           }
152           # Wenn wir im Mutterprozess sind:
153           else{
154
155             # Anzahl von Kindprozessen inkrementieren:
156             $children++;
157
158             # Im Mutterprozess benoetigen wir die Client-Verbindung nicht.
159             $client->close();
160
161           }
162
163         }
164         # Wenn fork() fehlschlug, Meldung ausgeben:
165         else{
166           logError('Unable to fork: '.$!);
167         }
168
169       }
170       # Wenn die maximale Anzahl von Verbindungen ueberschritten wurde, geben
171       # wir einfach eine Meldung an den Client, ohne die Verbindung wirklich zu
172       # verarbeiten, und schliessen die Verbindung:
173       else{
174         logError('Maximum count of children reached!');
175         sendError($client,503,'Service Unavailable',
176           'Too many concurrent connections. Please try again later.');
177         $client->shutdown(2);
178         $client->close();
179       }
180
181     }
182
183   }
184
185
186   # Verarbeite eine HTTP-Client-Verbindung:
187   sub processConnection($){
188     my($client)=@_;
189
190     my $headers;          # Enthaelt spaeter den Header-Hash.
191     my $getString;        # Enthaelt spaeter den GET-Parameter-String.
192     my $file;             # Enthaelt spaeter den Pfad zur angeforderten Datei.
193     my $host;             # Enthaelt spaeter den Hostname (Host:-Header).
194
195     # Definiere einen SIGALRM-Handler:
196     $SIG{ALRM}=sub{
197         # Timeout-Meldung senden, Client-Verbindung etc. schliessen und
198         # Kindprozess beenden:
199         sendError($client,408,'Request Time-Out',
200           'The server did not receive a propper request within '
201           .$conf->{client_timeout}.' seconds.');
202         exitChild();
203       };
204
205     # Setze ein Timeout fuer den Empfang der Header vom Client:
206     alarm($conf->{client_timeout});
207
208     # Erste Zeile der Anfrage enthaelt HTTP-Methode, -URL und -Version,
209     # empfange und teile nur die ersten max_url_length+100 Bytes:
210     my($method,$url,$version)
211       =split(/ /,substr(readline($client),0,$conf->{max_url_length}+100),3);
212
213     # Sende Nachricht und beende Verbindung, falls URL zu lang:
214     if(length($url) > $conf->{max_url_length}){
215       sendError($client,414,'Request URL Too Long',
216         'The URL requested is longer than '.$conf->{max_url_length}.' bytes.');
217       exitChild();
218     }
219
220     # Extrahiere GET-Parameter aus dem URL:
221     ($url,$getString)=split(/\?/,$url,2);
222
223     # Konvertiere URI-enkodierte Hex-Werte im URL fuer Pfad zu normalen Zeichen:
224     $file=decodeUri($url);
225
226     # Fahre nur fort, wenn URL ausschliesslich aus "sichere Zeichen" besteht
227     # und mit / beginnt, ausserdem muessen double-dot-Attacken ausgeschlossen
228     # sein:
229     # *FIXME* Ansich sollte ein Server .. aufloesen koennen - das waere z.B.
230     # durch s/\/.*?\/\.\.// moeglich, wird aber hier nicht implementiert um den
231     # Code nicht zu komplex (und damit fehlertraechtig) zu machen.
232     if(not($file=~/^\/[\/a-z0-9\.\_\- ]*$/i) || $file=~/\/\.\.(\/|$)/){
233       sendError($client,400,'Bad Request',
234         'The filename ('.$file.') requested contains bad characters.');
235       exitChild();
236     }
237
238     # Empfange Rest der Headers:
239     $headers=getHeaders($client);
240
241     # Entnehme den Host:-Header und entferne alles hinter ":"
242     # (eventuelle Portangabe):
243     $host=exists($headers->{host})?lc($headers->{host}):'';
244     $host=~s/:.*$//;
245
246     # Ueberpruefe den Host:-Header auf Existenz und Konsistenz
247     # (double dot-Attacken ausschliessen):
248     if($host=~/^[a-z0-9\.\-]*[a-z0-9]$/ && -d sprintf($conf->{vhost_root},$host)){
249       # Setze Pfad zur angeforderten Datei mit VHost zusammen:
250       $file=sprintf($conf->{vhost_root},$host).'/'.$file;
251     }
252     # Sonst Default-VHost verwenden:
253     else{
254       # Setze Pfad zur angeforderten Datei im Default-VHost zusammen:
255       $file=$conf->{default_root}.'/'.$file;
256       # Und setze Hostname auf "_default_"
```

```
    $host='_default_';
  }

  # Entferne alle double slashes:
  $file=~s/\/\//\//g;

  # Wenn URL nicht auf / endet und ein Verzeichnis ist, leite den Client dort
  # hin weiter:
  if(not($url=~/\/$/) && -d $file){
    logAccess($client->peerhost().' 302 '.$url.' => '.$url.'/');
    sendHeader($client,302,'Found',{Location => $url.'/'});
    sendStatus($client,302,'Found',
      'This document is located at <a href="'.$url.'">'.$url.'</a>');
    exitChild();
  }


  # Akzeptiere GET-Methode:
  if(uc($method) eq 'GET'){
    serveFile($client,$file,$url,$host);
  }
  # Alle anderen HTTP-Methoden werden nicht unterstuetzt:
  else{
    sendError($client,405,'Method Not Allowed',
      'The method used in request is not allowed here.');
    exitChild();
  }

}


# Schicke die angeforderte Datei zum Client:
sub serveFile($$$$){
  my($client,$file,$url,$host)=@_;

  my $fh;               # Enthaelt spaeter das Dateihandle.
  my $buffer;           # Buffer fuer das Einlesen von Daten.
  my $fileExt;          # Enthaelt spaeter die Dateierweiterung.
  my $mimeType;         # Enthaelt spaeter den MIME-Type zur Dateierweiterung.

  # Index-Datei an Pfad anhaengen, falls Verzeichnis:
  if(-d $file && $file=~/\/$/){
    $file.=$conf->{index_file};
  }

  # Dateierweiterung aus Dateiname extrahieren und MIME-Type bestimmen:
  ($fileExt)=($file=~/.*\/.*?\.([a-z0-9]+)$/i);
  $mimeType=(defined($fileExt) && exists($conf->{mime_types}->{lc($fileExt)}))
    ? ($conf->{mime_types}->{lc($fileExt)}) : ($conf->{mime_types}->{'*'});

  # Wenn Datei nicht existiert, gebe Fehler aus:
  # *FIXME* Trifft auch zu, falls Datei in einem Verzeichnis ohne Zugriffsrechte
  # liegt!
  unless(-f $file){
    logAccess($client->peerhost().' 404 '.$host.' '.$file);
    sendError($client,404,'Not Found',
      'The file requested ('.$url.') does not exist on '.$host.'.');
    exitChild();
  }

  # Oeffne Datei und gebe Fehler aus, falls ohne Erfolg:
  $fh=new IO::File($file,'r')
    || do{
      logAccess($client->peerhost().' 403 '.$host.' '.$file);
      sendError($client,403,'Forbidden',
        'You are not allowed to access the file requested ('.$url.') on '
        .$host.'.');
      exitChild();
    };

  # Sende 200 OK inkl. Header der Dateigroesse:
  sendHeader($client,200,'OK',
      {
        Length         => -s $file,
        'Content-Type' => $mimeType
      }
    );

  # Schalte Dateihandle und Client-Verbindung in den Binaermodus und schiebe
  # alle Daten aus dem File zum Client durch, schliesse dann das File.
  binmode($fh);
  binmode($client);
  while(read($fh,$buffer,4096)){
    print($client $buffer);
  }
  $fh->close();

  # Erfolg protokollieren und Verbindung schliessen, Prozess beenden.
  logAccess($client->peerhost().' 200 '.$host.' '.$file.' ('.$mimeType.')');
  exitChild();

}


# Empfange alle Headers und erzeuge einen Hash daraus:
sub getHeaders($){
  my($client)=@_;

  my $headers={};       # Hash in dem die Header gespeichert werden.
  my $bytes=0;          # Byte-Counter fuer maximale Header-Groesse.

  # Enthaelt die (erste) zu verarbeitende Header-Zeile.
  my $line=readline($client);

  # Bis zu ersten Leerzeile sind alles Header:
  while(defined($line) && not($line=~/^[\r\n]+$/)){

    # Erhoehe den Byte-Counter und beende Verbindung, wenn uebergelaufen:
    $bytes+=length($line);
    if($bytes > $conf->{max_headers_length}){
      sendError($client,413,'Request Entity Too Large',
        'Your request was bigger than '.$conf->{max_headers_length}.' bytes.');
      exitChild();
    }

    # Ueberpruefe Syntax des Headers und speichere sie in $headers:
    if($line=~/^([a-z0-9\-]+):[\s\t]*(.*?)[\r\n]+$/i){
      $headers->{lc($1)}=$2;
    }
    # Wenn ein Syntaxproblem vorliegt; beende Verbindung mit Nachricht:
    else{
      sendError($client,400,'Bad Request',
        'The server did not understand your request.');
      exitChild();
    }

    # Lese naechste Zeile:
    $line=readline($client);
```

```
  }

  # Gebe $headers zurueck.
  return($headers);
}


# Sende eine vollstaendige Fehlermeldung und erzeuge Log-Eintrag:
sub sendError($$$){
  my($client,$code,$status,$message,)=@_;
  sendHeader($client,$code,$status,{});
  sendStatus($client,$code,$status,$message);
  logError($client->peerhost().' ERROR'.$code.': '.$status);
}


# Erzeuge und sende einen HTTP-Header:
sub sendHeader($$$\%){
  my($client,$code,$status,$addHeaders)=@_;

  # Default-Werte:
  my %headers=(
      'Content-Type' => 'text/html',
      'Server'       => 'MicroWeb/'.$VERSION,
    );

  # Fuege alle Header in $addHeaders %headers hinzu:
  foreach(keys(%{$addHeaders})){
    $headers{$_}=$addHeaders->{$_};
  }

  # Sende HTTP-Header an den Client:
  print($client 'HTTP/1.1 '.$code.' '.$status."\r\n");
  foreach(keys(%headers)){
    print($client $_.': '.$headers{$_}."\r\n");
  }
  print($client "\r\n");

}


# Erzeuge und sende eine HTTP-Statusmeldung in HTML:
sub sendStatus($$$){
  my($client,$code,$status,$message)=@_;
  print($client <<__EOF);
<html>
  <head>
    <title>$code $status</title>
  </head>
  <body>
    <h1>$status</h1>
    <p>
      $message
    </p>
    <hr>
    <p>
      MicroWeb $VERSION, a simple HTTP server implementation &mdash;
      &copy; 2006 Veit Wahlich
    </p>
  </body>
</html>
__EOF
}
```

```

# Erzeuge einen Timestamp mit der aktuellen lokalen Zeit fuer Log-Nachrichten
# unter Verwendung von POSIX::strftime():
sub timestamp(){
  return(strftime('[%Y-%m-%d %H:%M:%S] ',localtime(time)));
}


# Gebe eine Nachricht auf STDERR aus:
sub logError($){
  my($message)=@_;
  print(STDERR timestamp().$message."\n");
}


# Eine Nachricht auf STDOUT ausgeben:
sub logAccess($){
  my($message)=@_;
  print(STDOUT timestamp().$message."\n");
}


# Beendet den Kindprozess und schliesst zuvor sauber alle Handles:
sub exitChild(){
  $client->shutdown(2);
  $client->close();
  STDOUT->close();
  STDERR->close();
  exit(0);
}


# Der SIGCHLD-Handler erntet tote Kinder. Das nennt man wirklich so...
sub reapPid(){

  # Warte auf die PID des toten Kindprozesses...
  my $pid=wait();

  # Wenn sie richtig uebergeben wurde, dekrementiere die Anzahl laufender
  # Kindprozesse.
  if($pid > 0){
    $children--;
  }

  # Setze den SIGCHLD-Handler zur Sicherheit nochmal:
  $SIG{CHLD}=\&reapPid;

}


# Dekodiere URI-encodierten String:
sub decodeUri($){
  my($line)=@_;
  if(defined($line)){
    $line=~tr/+/ /;
    $line=~s/%([a-f0-9]{2})/pack('C',hex($1))/egi;
  }
  return($line);
}


main();
1;
```