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

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

package MicroWeb;
our $VERSION='0.8';


# 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.


# 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.
    root               => 'htdocs',         # Basisverzeichnis der Dokumente.
    index_file         => 'index.html',     # Datei zu laden wenn nur ein
                                            #   Verzeichnis angefragt wurde.
    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:
      $socket->shutdown(2);
      $socket->close();
      STDOUT->close();
      STDERR->close();
      exit(0);
    };

  # 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));

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

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

      # Und wenn $pid == 0 ist, sind wir gerade im Kindprozess - sonst sind
      # wir im Mutterprozess.
      if($pid == 0){

        # Im Client-Kindprozess brauchen wir den Listener nicht.
        $socket->close();

        # Signalhandler im Client sind anders als die im Server:
        $SIG{TERM}=$SIG{INT}=sub{
            exitChild();
          };

        # Uebergebe Verarbeitung der Verbindung an die Zulieferer-Funktion:
        processConnection($client);

        # Ist die Verbindung verarbeitet, kann die Client-Verbindung
        # geschlossen und der Prozess beendet werden:
        exitChild();

      }
      # Wenn wir im Mutterprozess sind:
```

```
      else{

        # Im Mutterprozess benoetigen wir die Client-Verbindung nicht.
        $client->close();

      }

    }

  }

}


# Verarbeite eine HTTP-Client-Verbindung:
sub processConnection($){
  my($client)=@_;

  my $getString;        # Enthaelt spaeter den GET-Parameter-String.
  my $file;             # Enthaelt spaeter den Pfad zur angeforderten Datei.
  my $host;             # Enthaelt spaeter den Hostname (Host:-Header).

  # Erste Zeile der Anfrage enthaelt HTTP-Methode, -URL und -Version:
  my($method,$url,$version)
    =split(/ /,readline($client),3);

  # Extrahiere GET-Parameter aus dem URL:
  ($url,$getString)=split(/\?/,$url,2);

  # Konvertiere URI-enkodierte Hex-Werte im URL fuer Pfad zu normalen Zeichen:
  $file=decodeUri($url);

  # Ueberspringe uebermittelte HTTP-Headers (bis Leerzeile empfangen):
  getHeaders($client);

  # Setze Pfad zur angeforderten Datei im Root-Verzeichnis zusammen:
  $file=$conf->{root}.'/'.$file;
  # Setze Hostname auf gebundene IP-Adresse:
  $host=$conf->{bind_address};

  # Akzeptiere GET-Methode:
  if(uc($method) eq 'GET'){
    serveFile($client,$file,$url,$host);
  }
  # Alle anderen HTTP-Methoden werden nicht unterstuetzt:
  else{
    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}->{'*'});

  # Oeffne Datei und gebe Fehler aus, falls ohne Erfolg:
  $fh=new IO::File($file,'r')
    || exitChild();

  # Sende 200 OK inkl. Header der Dateigroesse:
  print($client "HTTP/1.0 200 OK\r\n");
  print($client "Content-Type: ".$mimeType."\r\n");
  print($client "Length: ".(-s $file)."\r\n");
  # Leerzeile schliesst Header ab:
  print($client "\r\n");

  # 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();

  # Verbindung schliessen, Prozess beenden.
  exitChild();

}

# Empfange alle Headers und verwerfe sie:
sub getHeaders($){
  my($client)=@_;

  # 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]+$/)){

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

  }

}


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


# Dekodiere URI-encodierten String:
sub decodeUri($){
  my($line)=@_;
  if(defined($line)){
```

```
257       $line=~tr/+/ /;
258       $line=~s/%([a-f0-9]{2})/pack('C',hex($1))/egi;
259     }
260     return($line);
261   }
262
263
264   main();
265   1;
```