

```

1 #!/usr/bin/perl
2
3 # MicroWeb - ein minimalistischer Webserver zu Lehrzwecken
4 # (C) 2006 Veit Wahlich
5
6 package MicroWeb;
7
8 our $VERSION='1.0';
9
10 # Benötigte Module und Pragmas importieren:
11 use strict; # Wir programmieren stets strict.
12 use warnings; # und haetten gerne Warnmeldungen.
13 use IO::Socket::INET; # Wir möchten IP-Sockets
14 use IO::File; # und Dateien verwenden.
15 use POSIX qw(strftime); # Fuer saubere Timestamps benutzen wir strftime().
16
17 # globale Variable, speichert die Anzahl der aktuell laufenden Kindprozesse
18 my $children=0;
19
20 # globale Konfiguration
21
22 my $conf={
23   bind_address => '0.0.0.0', # IP-Adresse, an die wir binden.
24   bind_port => 8080, # TCP-Port, an den wir binden.
25   root => 'htdocs/default', # Basisverzeichnis der Dokumente.
26   index_file => 'index.html', # Datei zu laden wenn nur ein # Verzeichnis angefragt wurde.
27
28   max_connections => 10, # Max. Anzahl paralleler Verbindungen (== Kindprozesse).
29   max_url_length => 4096, # Max. Laenge einer URL in Bytes.
30   max_headers_length => 4096, # Max. Groesse der Headers in Bytes.
31   client_timeout => 5, # Max. Wartezeit fuer Empfang von Anfragen.
32
33   mime_types => {
34     html => 'text/html',
35     htm => 'text/html',
36     txt => 'text/plain',
37     css => 'text/css',
38     xml => 'text/xml',
39     xsl => 'text/xml',
40     jpeg => 'image/jpeg',
41     jpg => 'image/jpeg',
42     png => 'image/png',
43     gif => 'image/gif',
44     mp3 => 'audio/mpeg',
45     wav => 'audio/x-wav',
46     mid => 'audio/midi',
47     mpg => 'video/mpeg',
48     mpeq => 'video/mpeg',
49     avi => 'video/x-msvideo',
50     ogg => 'application/ogg',
51     js => 'application/x-javascript',
52     swf => 'application/x-shockwave-flash',
53     gz => 'application/x-gzip',
54     tgz => 'application/x-gzip',
55     bz2 => 'application/x-bzip2',
56     tbz2 => 'application/x-bzip2',
57     tar => 'application/x-tar',
58     zip => 'application/zip',
59     '*' => 'application/octet-stream'
60   },
61
62 };
63
64
65 # Variablen wegen Verwendung in Signalhandlern in globalem Scope:
66 my $socket; # Enthält später den Listener, # Enthält später das Filehandle des aktuellen Clients.
67 my $client; # Enthält später die PID des Client-Kindprozesses.
68
69 sub main(){
70
71   my $pid; # Enthält später die PID des Client-Kindprozesses.
72
73   my $spid; # Ein Signalhandler faengt SIGTERM und SIGINT ab:
74   $SIG{TERM}=$SIG{INT}=\&sub{
75     # Listener sauber beenden:
76     logError('Server received SIGTERM/SIGINT - exiting gracefully');
77     $socket->shutdown(2);
78     $socket->close();
79     STDOUT->close();
80     STDERR->close();
81     exit(0);
82   };
83
84
85   # Bei SIGCHLD wurde ein Kindprozess beendet und muss geerntet werden:
86   $SIG{CHLD}=\&reapPid;
87
88   # Listener anlegen...
89   $socket=new IO::Socket::INET(
90     Listen => 5,
91     LocalAddr => $conf->{bind_address},
92     LocalPort => $conf->{bind_port},
93     Proto => 'tcp',
94     Reuse => 1
95   );
96
97
98   # und auf Erfolg ueberpruefen.
99   unless($socket){
100     die("Unable to bind port to address: $!\n");
101
102   }
103
104   # Server-Endlosschleife betreten
105   while(1){
106
107     # Auf neue Client-Verbindung warten und in $client speichern:
108     $client=$socket->accept();
109
110     # Moderne Betriebssysteme geben ein $client=<undef> zurück, wenn $socket noch nicht wieder bereit ist - dann die Schleife von vorne beginnen:
111     # *FIXME*: Hier sollte man eigentlich kurz warten, sonst 100% CPU-Last, bis Socket wieder Verbindungen annimmt!
112
113     next unless(defined($client));
114
115     # Nur neue Verbindungen verarbeiten, wenn nicht zu viele Verbindungen
116     # aktiv sind:
117     if($children < $conf->{max_connections}){
118
119       # Prozess duplizieren und PID speichern.
120       $pid=fork();
121
122       # Wenn fork() erfolgreich ist, ist $pid definiert:
123
124       if(defined($pid)){
125
126         # Und wenn $pid == 0 ist, sind wir gerade im Kindprozess - sonst sind
127         # wir im Mutterprozess.
128
129       }
130
131     }
132
133   }
134
135 }
```

```

129 # Im Client-Kindprozess brauchen wir den Listener nicht.
130 $socket->close();
131
132 # Signalhandler im Client sind anders als die im Server:
133 $SIG{TERM}=$SIG{INT}=$SIG{HUP}=$SIG{QUIT}=$SIG{STOP}=$SIG{STOP2}=$SIG{CONT}=$SIG{CHLD}=$SIG{POLL}=$SIG{SIGPOLL}=$SIG{SIGPOLLNORM}=$SIG{SIGPOLLNORM2}=$SIG{SIGPOLLNORM3}=$SIG{SIGPOLLNORM4}=$SIG{SIGPOLLNORM5}=$SIG{SIGPOLLNORM6}=$SIG{SIGPOLLNORM7}=$SIG{SIGPOLLNORM8}=$SIG{SIGPOLLNORM9}=$SIG{SIGPOLLNORM10}=$SIG{SIGPOLLNORM11}=$SIG{SIGPOLLNORM12}=$SIG{SIGPOLLNORM13}=$SIG{SIGPOLLNORM14}=$SIG{SIGPOLLNORM15}=$SIG{SIGPOLLNORM16}=$SIG{SIGPOLLNORM17}=$SIG{SIGPOLLNORM18}=$SIG{SIGPOLLNORM19}=$SIG{SIGPOLLNORM20}=$SIG{SIGPOLLNORM21}=$SIG{SIGPOLLNORM22}=$SIG{SIGPOLLNORM23}=$SIG{SIGPOLLNORM24}=$SIG{SIGPOLLNORM25}=$SIG{SIGPOLLNORM26}=$SIG{SIGPOLLNORM27}=$SIG{SIGPOLLNORM28}=$SIG{SIGPOLLNORM29}=$SIG{SIGPOLLNORM30}=$SIG{SIGPOLLNORM31}=$SIG{SIGPOLLNORM32}=$SIG{SIGPOLLNORM33}=$SIG{SIGPOLLNORM34}=$SIG{SIGPOLLNORM35}=$SIG{SIGPOLLNORM36}=$SIG{SIGPOLLNORM37}=$SIG{SIGPOLLNORM38}=$SIG{SIGPOLLNORM39}=$SIG{SIGPOLLNORM40}=$SIG{SIGPOLLNORM41}=$SIG{SIGPOLLNORM42}=$SIG{SIGPOLLNORM43}=$SIG{SIGPOLLNORM44}=$SIG{SIGPOLLNORM45}=$SIG{SIGPOLLNORM46}=$SIG{SIGPOLLNORM47}=$SIG{SIGPOLLNORM48}=$SIG{SIGPOLLNORM49}=$SIG{SIGPOLLNORM50}=$SIG{SIGPOLLNORM51}=$SIG{SIGPOLLNORM52}=$SIG{SIGPOLLNORM53}=$SIG{SIGPOLLNORM54}=$SIG{SIGPOLLNORM55}=$SIG{SIGPOLLNORM56}=$SIG{SIGPOLLNORM57}=$SIG{SIGPOLLNORM58}=$SIG{SIGPOLLNORM59}=$SIG{SIGPOLLNORM60}=$SIG{SIGPOLLNORM61}=$SIG{SIGPOLLNORM62}=$SIG{SIGPOLLNORM63}=$SIG{SIGPOLLNORM64}=$SIG{SIGPOLLNORM65}=$SIG{SIGPOLLNORM66}=$SIG{SIGPOLLNORM67}=$SIG{SIGPOLLNORM68}=$SIG{SIGPOLLNORM69}=$SIG{SIGPOLLNORM70}=$SIG{SIGPOLLNORM71}=$SIG{SIGPOLLNORM72}=$SIG{SIGPOLLNORM73}=$SIG{SIGPOLLNORM74}=$SIG{SIGPOLLNORM75}=$SIG{SIGPOLLNORM76}=$SIG{SIGPOLLNORM77}=$SIG{SIGPOLLNORM78}=$SIG{SIGPOLLNORM79}=$SIG{SIGPOLLNORM80}=$SIG{SIGPOLLNORM81}=$SIG{SIGPOLLNORM82}=$SIG{SIGPOLLNORM83}=$SIG{SIGPOLLNORM84}=$SIG{SIGPOLLNORM85}=$SIG{SIGPOLLNORM86}=$SIG{SIGPOLLNORM87}=$SIG{SIGPOLLNORM88}=$SIG{SIGPOLLNORM89}=$SIG{SIGPOLLNORM90}=$SIG{SIGPOLLNORM91}=$SIG{SIGPOLLNORM92}=$SIG{SIGPOLLNORM93}=$SIG{SIGPOLLNORM94}=$SIG{SIGPOLLNORM95}=$SIG{SIGPOLLNORM96}=$SIG{SIGPOLLNORM97}=$SIG{SIGPOLLNORM98}=$SIG{SIGPOLLNORM99}=$SIG{SIGPOLLNORM100}=$SIG{SIGPOLLNORM101}=$SIG{SIGPOLLNORM102}=$SIG{SIGPOLLNORM103}=$SIG{SIGPOLLNORM104}=$SIG{SIGPOLLNORM105}=$SIG{SIGPOLLNORM106}=$SIG{SIGPOLLNORM107}=$SIG{SIGPOLLNORM108}=$SIG{SIGPOLLNORM109}=$SIG{SIGPOLLNORM110}=$SIG{SIGPOLLNORM111}=$SIG{SIGPOLLNORM112}=$SIG{SIGPOLLNORM113}=$SIG{SIGPOLLNORM114}=$SIG{SIGPOLLNORM115}=$SIG{SIGPOLLNORM116}=$SIG{SIGPOLLNORM117}=$SIG{SIGPOLLNORM118}=$SIG{SIGPOLLNORM119}=$SIG{SIGPOLLNORM120}=$SIG{SIGPOLLNORM121}=$SIG{SIGPOLLNORM122}=$SIG{SIGPOLLNORM123}=$SIG{SIGPOLLNORM124}=$SIG{SIGPOLLNORM125}=$SIG{SIGPOLLNORM126}=$SIG{SIGPOLLNORM127}=$SIG{SIGPOLLNORM128}=$SIG{SIGPOLLNORM129}=$SIG{SIGPOLLNORM130}=$SIG{SIGPOLLNORM131}=$SIG{SIGPOLLNORM132}=$SIG{SIGPOLLNORM133}=$SIG{SIGPOLLNORM134}=$SIG{SIGPOLLNORM135}=$SIG{SIGPOLLNORM136}=$SIG{SIGPOLLNORM137}=$SIG{SIGPOLLNORM138}=$SIG{SIGPOLLNORM139}=$SIG{SIGPOLLNORM140}=$SIG{SIGPOLLNORM141}=$SIG{SIGPOLLNORM142}=$SIG{SIGPOLLNORM143}=$SIG{SIGPOLLNORM144}=$SIG{SIGPOLLNORM145}=$SIG{SIGPOLLNORM146}=$SIG{SIGPOLLNORM147}=$SIG{SIGPOLLNORM148}=$SIG{SIGPOLLNORM149}=$SIG{SIGPOLLNORM150}=$SIG{SIGPOLLNORM151}=$SIG{SIGPOLLNORM152}=$SIG{SIGPOLLNORM153}=$SIG{SIGPOLLNORM154}=$SIG{SIGPOLLNORM155}=$SIG{SIGPOLLNORM156}=$SIG{SIGPOLLNORM157}=$SIG{SIGPOLLNORM158}=$SIG{SIGPOLLNORM159}=$SIG{SIGPOLLNORM160}=$SIG{SIGPOLLNORM161}=$SIG{SIGPOLLNORM162}=$SIG{SIGPOLLNORM163}=$SIG{SIGPOLLNORM164}=$SIG{SIGPOLLNORM165}=$SIG{SIGPOLLNORM166}=$SIG{SIGPOLLNORM167}=$SIG{SIGPOLLNORM168}=$SIG{SIGPOLLNORM169}=$SIG{SIGPOLLNORM170}=$SIG{SIGPOLLNORM171}=$SIG{SIGPOLLNORM172}=$SIG{SIGPOLLNORM173}=$SIG{SIGPOLLNORM174}=$SIG{SIGPOLLNORM175}=$SIG{SIGPOLLNORM176}=$SIG{SIGPOLLNORM177}=$SIG{SIGPOLLNORM178}=$SIG{SIGPOLLNORM179}=$SIG{SIGPOLLNORM180}=$SIG{SIGPOLLNORM181}=$SIG{SIGPOLLNORM182}=$SIG{SIGPOLLNORM183}=$SIG{SIGPOLLNORM184}=$SIG{SIGPOLLNORM185}=$SIG{SIGPOLLNORM186}=$SIG{SIGPOLLNORM187}=$SIG{SIGPOLLNORM188}=$SIG{SIGPOLLNORM189}=$SIG{SIGPOLLNORM190}=$SIG{SIGPOLLNORM191}=$SIG{SIGPOLLNORM192}=$SIG{SIGPOLLNORM193}=$SIG{SIGPOLLNORM194}=$SIG{SIGPOLLNORM195}=$SIG{SIGPOLLNORM196}=$SIG{SIGPOLLNORM197}=$SIG{SIGPOLLNORM198}=$SIG{SIGPOLLNORM199}=$SIG{SIGPOLLNORM200}=$SIG{SIGPOLLNORM201}=$SIG{SIGPOLLNORM202}=$SIG{SIGPOLLNORM203}=$SIG{SIGPOLLNORM204}=$SIG{SIGPOLLNORM205}=$SIG{SIGPOLLNORM206}=$SIG{SIGPOLLNORM207}=$SIG{SIGPOLLNORM208}=$SIG{SIGPOLLNORM209}=$SIG{SIGPOLLNORM210}=$SIG{SIGPOLLNORM211}=$SIG{SIGPOLLNORM212}=$SIG{SIGPOLLNORM213}=$SIG{SIGPOLLNORM214}=$SIG{SIGPOLLNORM215}=$SIG{SIGPOLLNORM216}=$SIG{SIGPOLLNORM217}=$SIG{SIGPOLLNORM218}=$SIG{SIGPOLLNORM219}=$SIG{SIGPOLLNORM220}=$SIG{SIGPOLLNORM221}=$SIG{SIGPOLLNORM222}=$SIG{SIGPOLLNORM223}=$SIG{SIGPOLLNORM224}=$SIG{SIGPOLLNORM225}=$SIG{SIGPOLLNORM226}=$SIG{SIGPOLLNORM227}=$SIG{SIGPOLLNORM228}=$SIG{SIGPOLLNORM229}=$SIG{SIGPOLLNORM230}=$SIG{SIGPOLLNORM231}=$SIG{SIGPOLLNORM232}=$SIG{SIGPOLLNORM233}=$SIG{SIGPOLLNORM234}=$SIG{SIGPOLLNORM235}=$SIG{SIGPOLLNORM236}=$SIG{SIGPOLLNORM237}=$SIG{SIGPOLLNORM238}=$SIG{SIGPOLLNORM239}=$SIG{SIGPOLLNORM240}=$SIG{SIGPOLLNORM241}=$SIG{SIGPOLLNORM242}=$SIG{SIGPOLLNORM243}=$SIG{SIGPOLLNORM244}=$SIG{SIGPOLLNORM245}=$SIG{SIGPOLLNORM246}=$SIG{SIGPOLLNORM247}=$SIG{SIGPOLLNORM248}=$SIG{SIGPOLLNORM249}=$SIG{SIGPOLLNORM250}=$SIG{SIGPOLLNORM251}=$SIG{SIGPOLLNORM252}=$SIG{SIGPOLLNORM253}=$SIG{SIGPOLLNORM254}=$SIG{SIGPOLLNORM255}=$SIG{SIGPOLLNORM256}

193 # Kindprozess beenden:
194 sendError($client,408,'Request Time-out',
195 'The server did not receive a proper request within',
196 '$conf->{client_timeout}', 'seconds. ');
197 exitChild();
198 }

199 # Setze ein Timeout fuer den Empfang der Header vom Client:
200 alarm($conf->{client_timeout});
201
202 # Erste Zeile der Anfrage enthaelt HTTP-Methode, -URL und -Version,
203 # empfange und teile nur die ersten max_url_length+100 Bytes:
204 my($method,$url,$version) =split('/',substr(readline($client),0,$conf->{max_url_length}+100),3);
205
206 # Sende Nachricht und beende Verbindung, falls URL zu lang:
207 if(length($url) > $conf->{max_url_length}){
208 sendError($client,414,'Request URL too long');
209 }
210
211 "The URL requested is longer than '$conf->{max_url_length}. bytes.'";
212 exitChild();

213
214 # Extrahiere GET-Parameter aus dem URL:
215 ($url,$getString)=split('/?',$url,2);
216
217 # Konvertiere URL-erkodierte Hex-Werte im URL fuer Pfad zu normalen Zeichen:
218 $file=decodeURI($url);
219
220 # Fahre nur fort, wenn URL ausschliesslich aus "sichere Zeichen" besteht
221 # und mit / beginnt, außerdem muessen double-dot-Attacken ausgeschlossen
222 # sein:
223 # *FIXME* Ansicht sollte ein Server .. aufloesen koennen - das waere z.B.
224 # durch s/\.\.*?\.\./ moglich, wird aber hier nicht implementiert um den
225 # Code nicht zu komplex (und damit fehlertraechtig) zu machen.
226 if(not($file=~'~/\V\za-z0-9\.\V\-\V\$|\$1') || $file=~'\\\.\\.(\\|\$|/)'){
227 sendError($client,400,'Bad Request');
228
229 "The filename ('.$file.') requested contains bad characters.');
230 exitChild();
231
232 # Ueberspringe uebermittelte HTTP-Headers (bis Leerzeile empfangen):
233 getHeaders($client);
234
235 # Setze Pfad zur angeforderten Datei im Root-Verzeichnis zusammen:
236 $file=$conf->{root}.'/'.$file;
237
238 # Setze Hostname auf gebundene IP-Adresse:
239 $host=$conf->{bind_address};
240
241 # Entferne alle double slashes:
242 $file=~s/\V\//\V/g;
243
244 # Wenn URL nicht auf / endet und ein Verzeichnis ist, leite den Client dort
245 # hin weiter:
246 if(not($url=~'^/[^/]' && -d $file){
247 logAccess($client->peerhost(),'302'.'/'.$url.'=>'.'/'.$url.'/');
248 sendHeader($client,302,'found',{Location => $url.'/'.$url.'/'});
249
250 "This document is located at <a href='".$url."'>".$url.'';
251 }
252 exitChild();
253
254 # Akzeptiere GET-Methode:
255 if($c->method) eq 'GET'){
256 serveFile($client,$file,$url,$host);

257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
623
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267

```

```

258 # Alle anderen HTTP-Methoden werden nicht unterstuetzt:
259 else{
260     sendError($client,405, Method Not Allowed', 'The method used in request is not allowed here.');
261 }
262 exitChild();
263
264
265 }
266
267
268 # Schicke die angeforderte Datei zum Client:
269 sub serveFile($$$$){
270     my($client,$file,$url,$host)=@_;
271     my $fh;          # Enthaltet spaeter das Dateihandle.
272     my $buffer;      # Buffer fuer das Eintesen von Daten.
273     my $fileExt;    # Enthaltet spaeter die Dateierweiterung.
274     my $mimeType;   # Enthaltet spaeter den MIME-Type zur Dateierweiterung.
275
276     # Index-Datei an Pfad anhaengen, falls Verzeichnis:
277     if(-d $file & $file =~/^\$/){
278         $file.=>{$conf->{index_file}};
279     }
280
281     # Wenn Datei nicht existiert, gebe Fehler aus:
282     if($fileExt) {
283         ($fileExt) = ($file =~/^(.*\.(a-z0-9)+\$/);
284         $mimeType=>defined($fileExt) && exists($conf->{mime_types})->{lc($fileExt)})?
285             ($conf->{mime_types}) : ($conf->{mime_types}->{'*'});
286
287     # *FIXME* Trifft auch zu, falls Datei in einem Verzeichnis ohne Zugriffsrechte
288     # liegt!
289
290     unless(-f $file){
291         logAccess($client->peerhost(), '404' . $host . ' ' . $file);
292         sendError($client,404, Not Found', 'The file requested (' . $url . ') does not exist on ' . $host . ' ');
293     }
294     exitChild();
295 }
296
297 # Offene Datei und gebe Fehler aus, falls ohne Erfolg:
298 $fh=new IO::File($file,'r');
299 | do{
300     logAccess($client->peerhost(), '403' . $host . ' ' . $file);
301     sendError($client,403, 'Forbidden', "You are not allowed to access the file requested (' . $url . ') on ' .
302     $host . ' ");
303 }
304 exitChild();
305
306 # Sende 200 OK inkl. Header der Dateigroesse:
307
308 {
309     sendHeader($client,200,'OK',
310     'Content-Type' => $MimeType,
311     );
312
313 }
314
315 # Schalte Dateihandle und Client-Verbindung in den Binaermodus und schiebe
316 # alle Daten aus dem File zum Client durch, schliesse dann das File.
317 binmode($fh);
318 binmode($client);
319 while(read($fh,$buffer,4096)){
320     print($client $buffer);
321
322     $fh->close();
323
324     # Erfolg protokollieren und Verbindung schliessen, Prozess beenden.
325     logAccess($client->peerhost(), '200' . $host . ' ' . $file . ' (' . $MimeType . ')');
326     exitChild();
327 }
328
329 # Empfange alle Headers und verwerde sie:
330 sub getHeaders($${
331     my($client)=@_;
332
333     my $bytes=0;          # Byte-Counter fuer maximale Header-Groesse.
334     my $line;              # Enthaltet die (erste) zu verarbeitende Header-Zeile.
335     my $line=readLine($client);
336     my $line=readLine($client);
337     my $line=readLine($client);
338
339     # Bis zu ersten Leerzeile sind alles Header:
340     while(defined($line) && not($line =~/^(\\r\\n)+$/)){
341         # Erhoehe den Byte-Counter und beende Verbindung, wenn uebergelaufen:
342         $bytes+=length($line);
343         print($line);
344         if($bytes > $conf->{max_headers_length}){
345             sendError($client,413, Request Entity Too Large', "Your request was bigger than " . $conf->{max_headers_length}. ' bytes");
346             exitChild();
347         }
348     }
349 }
350
351     # Lese naechste Zeile:
352     $line=readLine($client);
353
354 }
355
356 }
357
358 # Sende eine volstaendige Fehlermeldung und erzeuge Log-Eintrag:
359 sub sendError($$$$){
360     my($client,$code,$status,$message)=@_;
361     sendHeader($client,$code,$status,$message);
362     sendHeader($client,$code,$status,$message);
363     sendHeader($client,$code,$status,$message);
364     logError($client->peerhost(),'ERROR' . $code . ' ' . $status);
365 }
366
367
368 # Erzeuge und sende einen HTTP-Header:
369 sub sendHeader($$%){
370     my($client,$code,$status,$addHeaders)=@_;
371
372     # Default-Werte:
373     my %headers=(
374         'Content-Type' => 'text/html',
375         'Server'        => 'Microweb/' . $VERSION,
376     );
377
378     # Fuege alle Header in $addHeaders %headers hinzu:
379     foreach(keys(%$addHeaders)){
380         $headers{$_-}>=$addHeaders->{$_-};
381     }
382
383     # Sende HTTP-Header an den Client:
384     print($client 'HTTP/1.0 ' . $code . ' ' . $status . "\r\n");

```

```

385     foreach(keys(%headers)){
386         print($client $_.'_'.$headers{$_}."\r\n");
387     }
388     print($client "\r\n");
389 }
390 }

391 # Erzeuge und sende eine HTTP-Statusmeldung in HTML:
392 sub sendStatus($$$$){
393     my($client,$code,$status,$message) = @_;
394     print $client <__EOF__;
395     <html>
396     <head>
397         <title>$code $status</title>
398     </head>
399     <body>
400         <h1>$status</h1>
401         <p>
402             $message
403         </p>
404         <hr>
405         <hr>
406         <hr>
407         <p>
408             Microweb $VERSION, a simple HTTP server implementation &mdash;
409             &copy; 2006 Veit Wahlich
410         </p>
411         </body>
412     </html>
413     __EOF
414 }
415

416 # Erzeuge einen Timestamp mit der aktuellen lokalen Zeit fuer Log-Nachrichten
417 # unter Verwendung von POSIX::strftime();
418 sub timestamp(){
419     return strftime('[%Y-%m-%d %H:%M:%S]', localtime(time));
420 }
421

422

423 # Gebe eine Nachricht auf STDERR aus:
424 sub logError($){
425     my($message)=@_;
426     print STDERR timestamp(),$message, "\r\n";
427 }
428

429

430 # Eine Nachricht auf STDOUT ausgeben:
431 sub logAccess($){
432     my($message)=@_;
433     print STDOUT timestamp(),$message, "\r\n";
434 }
435

436

437 # Beendet den Kindprozess und schliesst zuvor sauber alle Handles:
438 sub exitChild(){
439     $client->shutdown(2);
440     $client->close();
441     $STDOUT->close();
442     $STDERR->close();
443     $STDERR->close();
444     exit(0);
445 }
446
447
448 # Der SIGCHLD-Handler erntet tote Kinder. Das nennt man wirklich so...

```