*Verteilte Systeme im Sommersemester 2021*

Steffen Herweg, Matr. Nr. 873475

Luca Fabio Kock, Matr. Nr. 879534 Osnabrück, 27.04.2021

# Aufgabenblatt 2

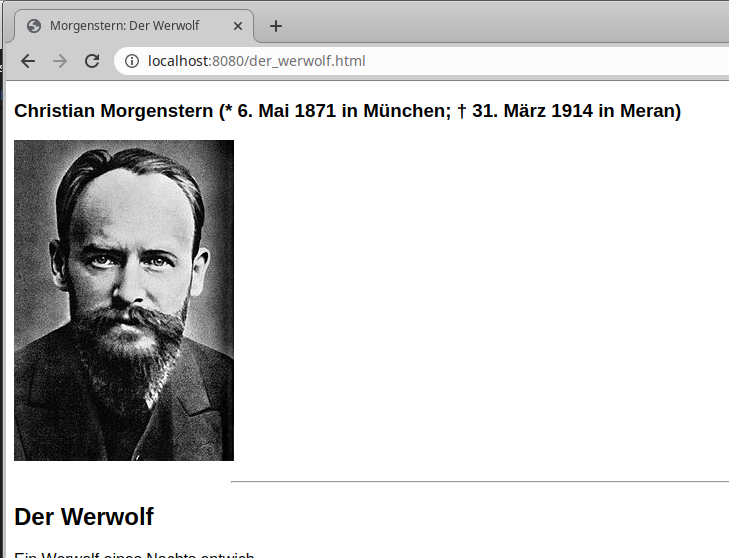
#define MAX\_SOCK 10  
#define MAXREQUESTSIZE 8192  
#define MAXURLSIZE 256  
#define CHUNKSIZE 1024  
#define RESPONSEBUFF 8192  
#define MAXCONTENTSIZE 8192  
  
// Vorwaertsdeklarationen intern  
void html\_serv(int, int);  
  
void response\_get(int, int, const char \*);  
  
void response\_post(int, const char \*);  
  
void response\_file(int, int, const char \*);  
  
void response\_dir(int, int);  
  
void response\_not\_found(int);  
  
void date\_string(char \*, size\_t);  
  
void err\_abort(char \*str);  
  
int main(int argc, char \*argv[]) {  
  
 if (argc != 3) {  
 fprintf(stderr, "Usage: %s <docroot> <port>\n", argv[0]);  
 exit(EXIT\_FAILURE);  
 }  
  
 const char \*docroot = argv[1];  
 int port = atoi(argv[2]);  
  
 // Deskriptoren, Adresslaenge, Prozess-ID  
 int sockfd, newsockfd, alen, pid, rootfd;  
 int reuse = 1;  
  
 rootfd = open(docroot, O\_RDONLY);  
 if (rootfd < 0) {  
 err\_abort("Ordner konnte nicht ge�ffnet werden.");  
 }  
  
 // Socket Adressen  
 struct sockaddr\_in cli\_addr, srv\_addr;  
  
 // TCP-Socket erzeugen  
 if ((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {  
 err\_abort("Kann Stream-Socket nicht oeffnen!");  
 }  
  
 if (setsockopt(sockfd, SOL\_SOCKET, SO\_REUSEADDR, &reuse, sizeof(reuse)) < 0) {  
 err\_abort("Kann Socketoption nicht setzen!");  
 }  
  
 // Binden der lokalen Adresse damit Clients uns erreichen  
 memset((void \*) &srv\_addr, '\0', sizeof(srv\_addr));  
 srv\_addr.sin\_family = AF\_INET;  
 srv\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);  
 srv\_addr.sin\_port = htons(port);  
 if (bind(sockfd, (struct sockaddr \*) &srv\_addr,  
 sizeof(srv\_addr)) < 0) {  
 err\_abort("Kann lokale Adresse nicht binden, laeuft fremder Server?");  
 }  
  
 // Warteschlange fuer TCP-Socket einrichten  
 listen(sockfd, 5);  
 printf("HTML-Server: bereit ...\n");  
  
 for (;;) {  
 alen = sizeof(cli\_addr);  
  
 // Verbindung aufbauen  
 newsockfd = accept(sockfd, (struct sockaddr \*) &cli\_addr, &alen);  
 printf("Got new connection!\n");  
 if (newsockfd < 0) {  
 err\_abort("Fehler beim Verbindungsaufbau!");  
 }  
  
 // fuer jede Verbindung einen Kindprozess erzeugen  
 if ((pid = fork()) < 0) {  
 err\_abort("Fehler beim Erzeugen eines Kindprozesses!");  
 } else if (pid == 0) {  
 close(sockfd);  
 html\_serv(newsockfd, rootfd);  
 exit(0);  
 }  
 close(newsockfd);  
 }  
}  
  
void html\_serv(int sockfd, int rootfd) {  
  
 ssize\_t n;  
 char in[MAXREQUESTSIZE];  
 char url[MAXURLSIZE];  
 char method[8];  
 char version[8];  
  
 memset((void \*) in, '\0', MAXREQUESTSIZE);  
 memset((void \*) url, '\0', MAXURLSIZE);  
 memset((void \*) method, '\0', 8);  
 memset((void \*) version, '\0', 8);  
  
 n = read(sockfd, in, MAXREQUESTSIZE);  
 if (n == 0) {  
 return;  
 } else if (n < 0) {  
 err\_abort("Fehler beim Lesen des Sockets!");  
 }  
 sscanf(in, "%8s /%255s HTTP/%s", method, url, version); //Parameter werden aus Socket ausgelesen  
  
 if (strcmp(method, "GET") == 0) {  
 response\_get(sockfd, rootfd, url);  
 } else if (strcmp(method, "POST") == 0) {  
 response\_post(sockfd, in);  
 }  
}  
  
void response\_get(int sockfd, int rootfd, const char \*url) {  
 int filefd = openat(rootfd, url, O\_RDONLY, "rb"); //RB = Read Binary  
 if (filefd == -1) {  
 printf("Not found\n");  
 response\_not\_found(sockfd);  
 } else {  
 // Determine Extension  
 struct stat filestat;  
 fstat(filefd, &filestat);  
 printf("Found %s\n", url);  
 if (S\_ISDIR(filestat.st\_mode) != 0) {  
 response\_dir(sockfd, filefd);  
 } else {  
 const char \*ext = strrchr(url, '.');  
 printf("Is :[%s]\n", ext);  
 if (strcmp(ext, ".jpg") == 0) {  
 response\_file(sockfd, filefd, "image/jpeg");  
 } else if (strcmp(ext, ".png") == 0) {  
 response\_file(sockfd, filefd, "image/png");  
 } else if (strcmp(ext, ".html") == 0) {  
 response\_file(sockfd, filefd, "text/html");  
 } else if (strcmp(ext, ".txt") == 0) {  
 response\_file(sockfd, filefd, "text/plain");  
 } else {  
 response\_file(sockfd, filefd, "application/octet-stream");  
 }  
 }  
 close(filefd);  
 }  
}  
  
void response\_dir(int sockfd, int filefd) {  
 // Setup content  
 char responseContent[MAXCONTENTSIZE];  
 memset(responseContent, '\0', MAXCONTENTSIZE);  
 int offset = 0;  
  
 offset += snprintf(responseContent + offset, MAXCONTENTSIZE - offset, "<!DOCTYPE HTML><html><body><ul>");  
 DIR \*pDir = fdopendir(filefd);  
 struct dirent \*pDirent;  
 while (pDirent = readdir(pDir)) {  
 offset += snprintf(responseContent + offset, MAXCONTENTSIZE - offset, "<li><a href=\"./%s\">%s</a></li>",  
 pDirent->d\_name, pDirent->d\_name);  
  
 }  
 closedir(pDir);  
 offset += snprintf(responseContent + offset, MAXCONTENTSIZE - offset, "</ul></body></html>");  
  
 // Setup contentLength  
 char contentLength[64];  
 memset(contentLength, '\0', 64);  
 sprintf(contentLength, "Content-Length: %d\r\n", strlen(responseContent));  
  
 // Setup date  
 char date[64];  
 memset(date, '\0', 64);  
 date\_string(date, 64);  
  
 // Setup contentType  
 const char \*contentType = "Content-Type: text/html\r\n";  
  
 // Setting up reponse  
 char response[RESPONSEBUFF];  
 memset(response, '\0', RESPONSEBUFF);  
 sprintf(response, "HTTP/1.1 200 OK\r\n%s%s%s\r\n%s", contentLength, date, contentType, responseContent);  
  
 // Write response  
 write(sockfd, response, strlen(response));  
}  
  
  
void response\_post(int sockfd, const char \*in) {  
 int zahl1, zahl2;  
 printf("In:%s\n", in);  
 sscanf(in, "zahl1=%d&zahl2=%d", &zahl1, &zahl2);  
 int ergebnis = zahl1 \* zahl2;  
 printf("%d\*%d=%d\n", zahl1, zahl2, ergebnis);  
  
 // Setup content  
 char responseContent[256];  
 memset(responseContent, '\0', 256);  
 sprintf(responseContent, "<!DOCTYPE HTML><HTML><BODY><center><h1>Ergebnis: %d </h1></center></BODY></HTML>",  
 ergebnis);  
  
 // Setup contentLength  
 char contentLength[64];  
 memset(contentLength, '\0', 64);  
 sprintf(contentLength, "Content-Length: %lu\r\n", strlen(responseContent));  
  
 // Setup date  
 char date[64];  
 memset(date, '\0', 64);  
 date\_string(date, 64);  
  
 // Setup contentType  
 const char \*contentType = "Content-Type: text/html\r\n";  
  
 // Setting up reponse  
 char response[RESPONSEBUFF];  
 memset(response, '\0', RESPONSEBUFF);  
 sprintf(response, "HTTP/1.1 200 OK\r\n%s%s%s\r\n%s", contentLength, date, contentType, responseContent);  
  
 // Write response  
 write(sockfd, response, strlen(response));  
  
}  
  
  
void response\_not\_found(int sockfd) {  
 // Setup content  
 const char \*content = "404 Not Found";  
  
 // Setup date  
 char date[64];  
 memset(date, '\0', 64);  
 date\_string(date, 64);  
  
 // Setup contentLength  
 char contentLength[64];  
 memset(contentLength, '\0', 64);  
 sprintf(contentLength, "Content-Length: %lu\r\n", strlen(content));  
  
 // Setting up response  
 char response[RESPONSEBUFF];  
 memset(response, '\0', RESPONSEBUFF);  
 sprintf(response, "HTTP/1.1 404 Not Found\r\n%s%s\r\n%s", contentLength, date, content);  
  
 // Write response  
 write(sockfd, response, strlen(response));  
}  
  
void response\_file(int sockfd, int filefd, const char \*type) {  
 // Setup fileinfo  
 struct stat filestat;  
 fstat(filefd, &filestat);  
  
 // Setup date  
 char date[64];  
 memset(date, '\0', 64);  
 date\_string(date, 64);  
  
 // Setup contentLength  
 char contentLength[64];  
 memset(contentLength, '\0', 64);  
 sprintf(contentLength, "Content-Length: %lu\r\n", filestat.st\_size);  
  
 // Setup contentType  
 char contentType[64];  
 memset(contentType, '\0', 64);  
 sprintf(contentType, "Content-Type: %s\r\n", type);  
  
  
 // Setup response  
 char response[RESPONSEBUFF];  
 memset(response, '\0', RESPONSEBUFF);  
 snprintf(response, RESPONSEBUFF, "HTTP/1.1 200 OK\r\n%s%s%s\r\n", contentLength, date, contentType);  
  
 // Send response header  
 write(sockfd, response, strlen(response));  
  
  
 // Send file data  
 char buff[CHUNKSIZE];  
 ssize\_t n;  
 while ((n = read(filefd, buff, CHUNKSIZE)) > 0) {  
 write(sockfd, buff, n);  
 }  
}  
  
void date\_string(char \*str, size\_t maxsize) {  
 time\_t now;  
 time(&now);  
 struct tm \*now\_tm = localtime(&now);  
  
 strftime(str, maxsize, "Date: %a, %d %b %Y %H:%M:%S GMT\r\n", now\_tm);  
}  
  
/\*  
Ausgabe von Fehlermeldungen  
\*/  
void err\_abort(char \*str) {  
 fprintf(stderr, " TCP Echo-Server: %s\n", str);  
 fflush(stdout);  
 fflush(stderr);  
 exit(EXIT\_FAILURE);  
}

**Tests**

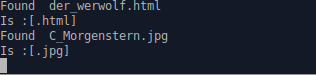
Starten des Servers:



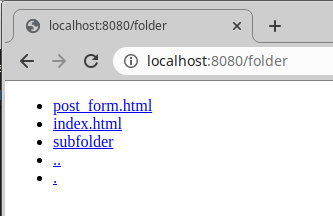
Zugriff auf Webserver in Chrome:



Rückmeldung des Webservers:



Aufruf des Dateisystems in Chrome:



Rückmeldung des Webservers:



Aufruf des Webservers in Firefox:

