Verteilte Systeme im Sommersemester 2021

Steffen Herweg, Matr. Nr. 873475
Luca Fabio Kock, Matr. Nr. 879534

Osnabrück, 27.04.2021

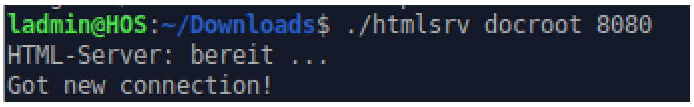
Aufgabenblatt 2

```c
#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,
```

```c
"</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);
}
```

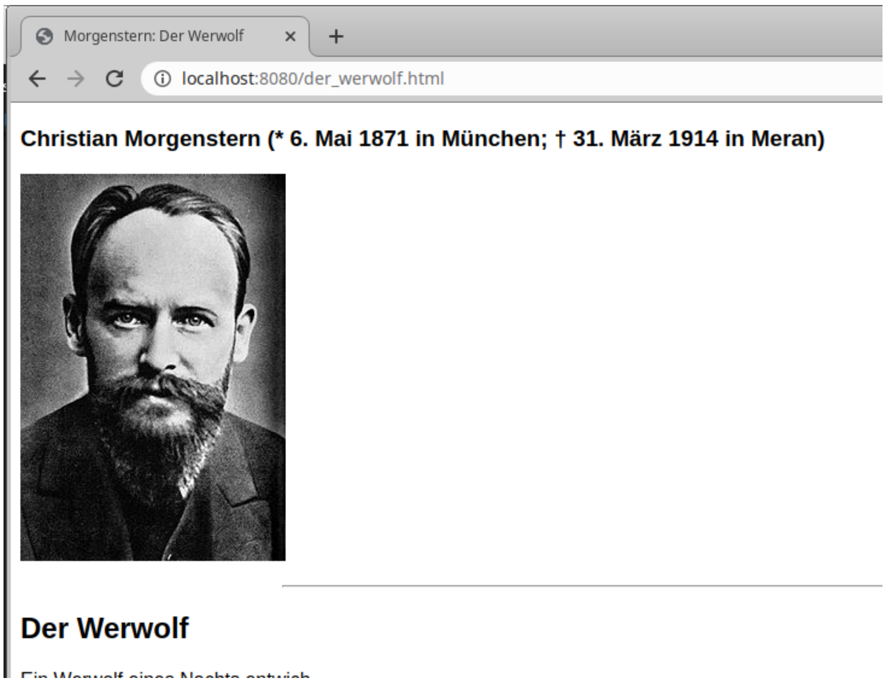
Tests

Starten des Servers:



```
ladmin@HOS:~/Downloads$ ./htmlsrv docroot 8080
HTML-Server: bereit ...
Got new connection!
```

Zugriff auf Webserver in Chrome:

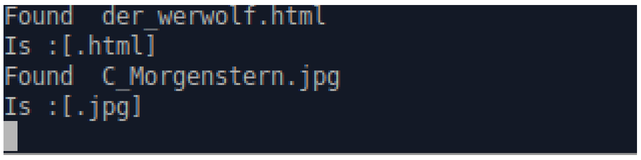


Christian Morgenstern (* 6. Mai 1871 in München; † 31. März 1914 in Meran)

Der Werwolf

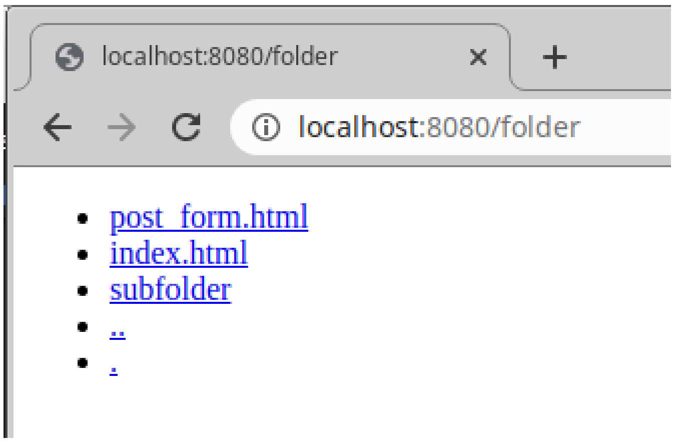
Ein Werwolf eines Nachts entwich

Rückmeldung des Webservers:



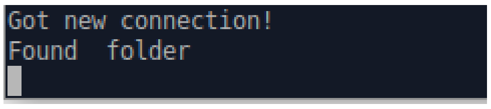
```
Found  der_werwolf.html
Is :[.html]
Found  C_Morgenstern.jpg
Is :[.jpg]
```

Aufruf des Dateisystems in Chrome:



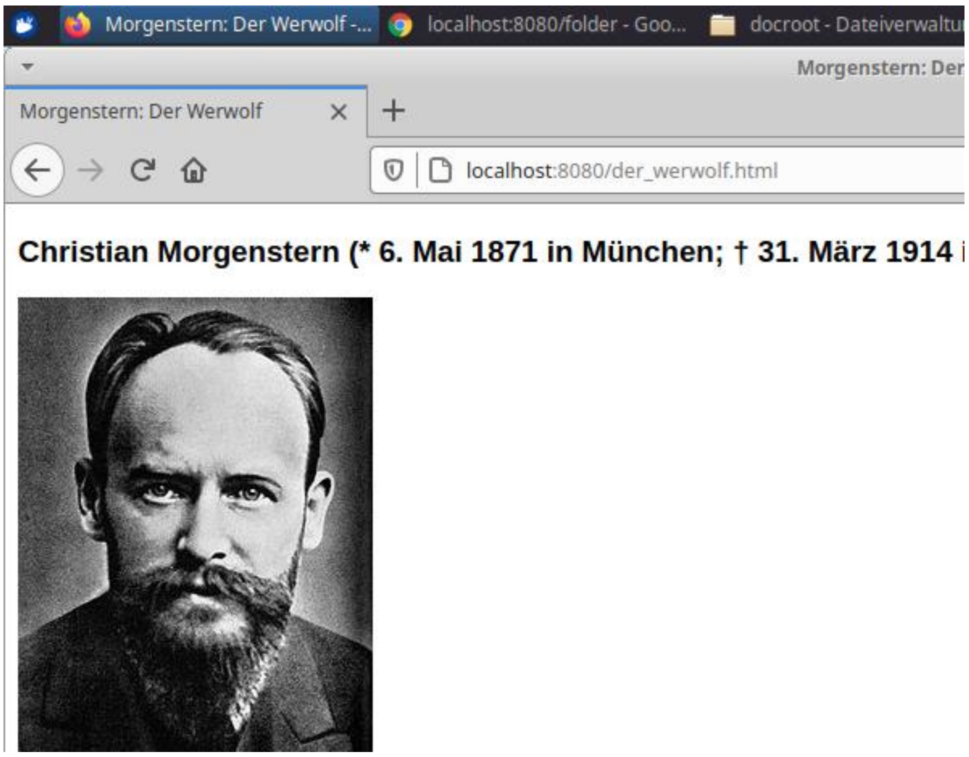
- post_form.html
- index.html
- subfolder
- ..
- .

Rückmeldung des Webservers:



```
Got new connection!
Found  folder
```

Aufruf des Webservers in Firefox:



Morgenstern: Der Werwolf -...
localhost:8080/folder - Goo...
docroot - Dateiverwaltu
Morgenstern: Der
Morgenstern: Der Werwolf
localhost:8080/der_werwolf.html

Christian Morgenstern (* 6. Mai 1871 in München; † 31. März 1914 i