课程编号：A0802052410

《Linux 程序设计》实验报告

姓　　名	
班　　级	
实 验 名 称	
开 设 学 期	
开 设 时 间	
报 告 日 期	
评 定 成 绩	

东北大学软件学院

实验一 多进程编程(8 学时)

一、实验目的

1、理解进程的基本概念和基本的进程函数。
2、掌握网络编程的基本操作和设计原则。
3、掌握客户/服务器协议的原理。
4、服务器设计：使用 fork 来接收多个请求。
5、如何解决僵尸(zombie)问题。

二、实验环境及实验类型

1、操作系统：Linux Ubuntu
2、开发工具、编译器及调试软件：Visual Studio Code,GCC,GNU
3、编程语言及版本号：C++/C
4、版本号：February 2022（version 1.65）
5、机器台号：华硕飞行堡垒 8
6、上机地点：班级未集中进行实验课,居家进行
7、实验类型：设计型

三、实验内容

1、三个主要操作

客户和服务器都是进程。服务器设立服务，然后进入循环接收和处理请求。客户连接到服务器，然后发送、接受或者交换数据，最后退出。该交互过程中主要包含了一下 3 个操作：

- 服务器设立服务。
- 客户连接到服务器。
- 服务器和客户处理事务。

2、服务器的设计问题：DIY 或代理

这里使用了两种服务器的设计方法：

- 自己做(Do It Yourself ,DIY) ------服务器接收请求，自己处理工作。
- 代理------服务器接收请求，然后创建一个新进程来处理工作。

每种方法的优缺点各是什么？

- 自己做用于快速简单的任务
- 代理用于慢速的更加复杂的任务
- 使用 SIGCHLD 来阻止僵尸问题

3、Web 服务器功能

Web 服务器通常要具有 3 种用户操作：

- 列举目录信息
- cat 文件
- 运行程序

4、Web 服务器协议

客户端（浏览器）与 Web 服务器之间的交互主要包含客户的请求和服务器的应答。请求和应答的格式在超文本传输协议（HTTP）中有定义。HTTP 使用纯文本。可以使用 telnet 和 Web 服务器进行交互。

（1）HTTP 请求：GET

telnet 创建了一个 socket 并调用了 connect 来连接到 Web 服务器。服务器接受连接请求，并创建了一个基于 socket 的从客户端的键盘到 Web 服务器进程的数据通道。

接下来，输入请求：

GET /index.html HTTP/1.0

一个 HTTP 请求包含有 3 个字符串。第一个字符串是命令，第二个是参数，第三个是所用协议的版本号。在该例中，使用了 GET 命令，以 index.html 作为参数，使用了 HTTP 版本 1.0 。实际上，一个 Web 服务器只是集成了 cat 和 ls 的 Unix shell。

（2）HTTP 应答：OK

服务器读取请求，检查请求，然后返回一个应答。应答有两个部分：头部和内容。头部以状态行起始，如下所示：

HTTP/1.1 200 OK

状态行含有两个或更多的字符串。第一个串是协议的版本，第二个串是返回码，这里是 200，其文本的解释是 OK。这里请求的文件叫/index.html，而服务器给出应答表示可以得到该文件。如果服务器中没有所请求的文件名，返回码将是 404，其解释将是“未找到”。

头部的其余部分是关于应答的附加信息。在该例子中，附加信息包含服务器名、应答时间、服务器所发送数据类型以及应答的连接类型。一个应答头部可以包含有多行信息，以空行表示结束，空行位于 Connection：close 后面。

应答的其余部分是返回的具体内容。这里，服务器返回了文件/index.html 的内容。

5、编写 Web 服务器

要求Web服务器只支持GET命令，只接收请求行，跳过其余参数，然后处理请求和发送应答。

服务器为每个请求创建一个新的进程来处理。子进程将请求分割成命令和参数。如果命令不是 GET，服务器应答 HTTP 返回码表示未实现的命令。如果命

令是 GET，服务器将期望得到目录名、一个可执行程序或文件名。如果没有该目录或指定的文件名，服务器报错。

如果存在目录或文件，服务器决定所要使用的操作：ls、exex或cat。

6、测试（运行 Web 服务器）

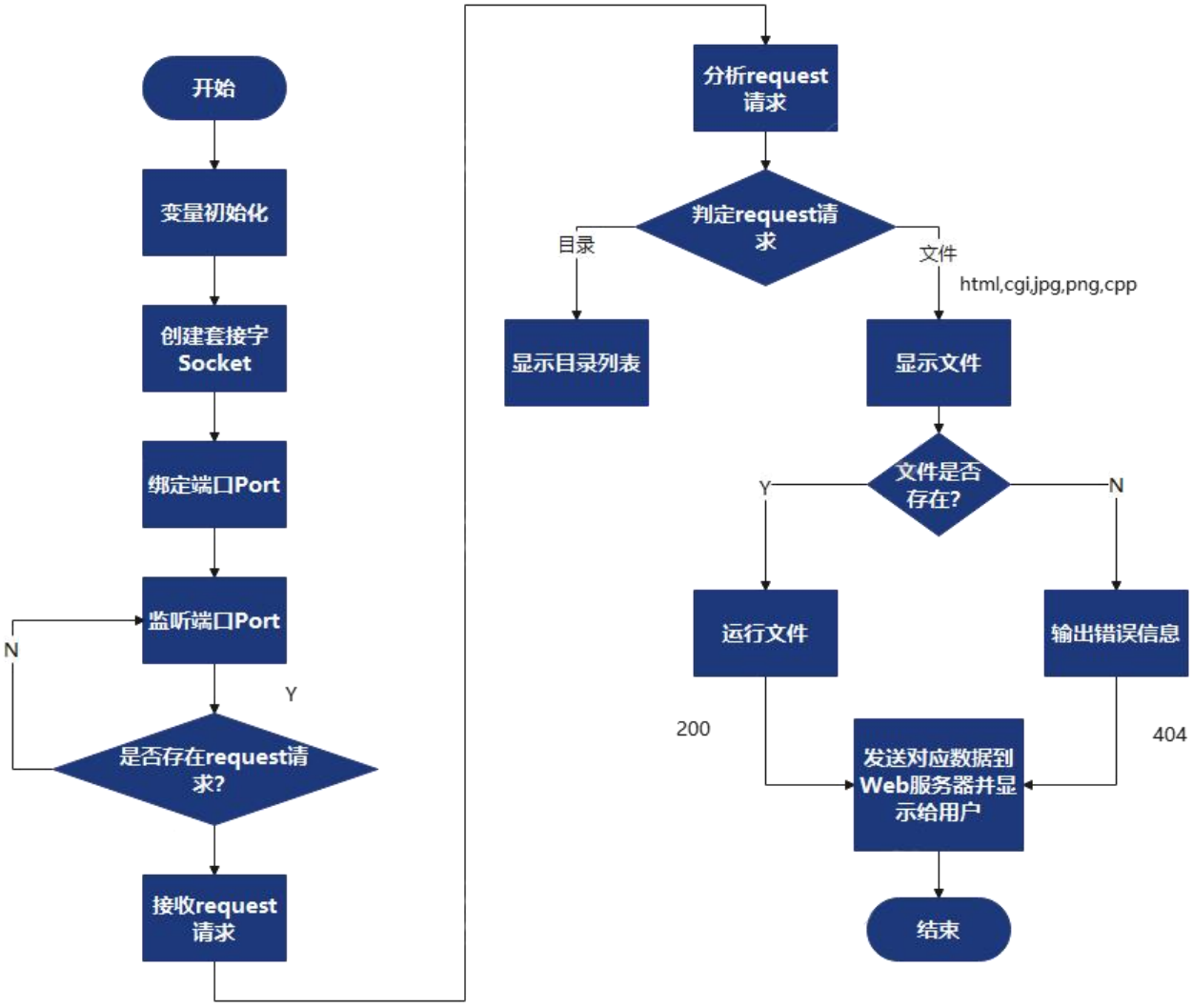
编译程序，在某个端口运行它：

$g++ webserv_linux.cpp -D_REENTRANT -o web -lpthread

$./web 12345

现在可以访问Web服务器，网址为http://yourhostname:12345/。将html文件放到该目录中并且用http://yourhostname:12345//filename.html来打开它。

四、实验流程



五、函数说明及关键代码

1、函数说明

调用函数	函数说明
void* request_handler(void* arg);	处理请求的主函数
void send_data(FILE* fp, char* ct, char* file_name);	向服务器发送数据信息
char* content_type(char* file);	返回文件对于请求的类型
void send_error(FILE* fp);	向服务器发送错误信息
void error_handling(char* message);	对错误的处理
int check_path(char* file_path);	检验文件路径
void child_waiter(int signum);	为SIGCHLD设置一个信号处理函数
bool not_exist(char* arg);	判断文件路径是否存在
void do_404(char* arg,FILE *fp);	发送请求错误 404
bool is_dir(char* arg);	判断文件路径是否为一个目录
void do_ls(char* arg,FILE *fp);	目录列表函数
bool ends_in_cgi(char* arg);	判断文件是否为 cgi 文件
void do_exec(char* arg,FILE *fp);	执行 cgi 文件
void do_cat(char* arg,FILE *fp);	显示当前目录下全部文件或函数
void sendPic(char* arg,FILE *fp);	执行图片类型文件
bool ends_in_html(char* arg);	判断文件是否为 html 文件
bool ends_in_pic(char* arg);	判断文件是否为图片文件
int get_char_times(char* str,char c);	得到字符在字符串中出现的次数
void strmcpy(char *p, char *a, int m);	选择性复制字符串
void get_time(char* time_str);	获取当前时间戳
void log_fun(FILE *log_fp,char *time_log,char *temp);	日志记录函数
int make_server_socket(int portnum);	建立服务器端 socket
int connect_to_server(char* host,int portnum);	建立到服务器的连接

2、关键代码

（1）Web服务器主函数

```
int main(int argc, char *argv[]){
    int serv_sock, clnt_sock;
    struct sockaddr_in serv_adr, clnt_adr;
    int clnt_adr_size;
    char buf[BUF_SIZE];
```

```
    pthread_t t_id;
    if(argc!=2) {
        printf("Usage : %s <port>\n", argv[0]);
        exit(1);
    }
    signal(SIGCHLD,child_waiter);
    serv_sock=socket(PF_INET, SOCK_STREAM, 0);
    memset(&serv_adr, 0, sizeof(serv_adr));
    serv_adr.sin_family=AF_INET;
    serv_adr.sin_addr.s_addr=htonl(INADDR_ANY);
    serv_adr.sin_port = htons(atoi(argv[1]));
    if(bind(serv_sock, (struct sockaddr*)&serv_adr, sizeof(serv_adr))==-1)
        error_handling("bind() error");
    if(listen(serv_sock, 20)==-1)
        error_handling("listen() error");
    FILE *log_fp=fopen("log.txt","a+");
    if(log_fp==NULL){
        fprintf(stderr,"ERROR:Create a log file unsuccessfully!");
        exit(1);
    }
    fclose(log_fp);
    while(1){
        clnt_adr_size=sizeof(clnt_adr);
        clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_adr, (socklen_t*)&clnt_adr_size);
        printf("Waiting for the client to connect......\n");
        printf("Connection Request : %s:%d\n",
            inet_ntoa(clnt_adr.sin_addr), ntohs(clnt_adr.sin_port));
        pthread_create(&t_id, NULL, request_handler, &clnt_sock);
        pthread_detach(t_id);
    }
    close(serv_sock);
    return 0;
}
```

Web服务器的主函数首先检测输入参数是否存在端口号，若不存在则直接退出；若存在，则建立对应的Socket连接和日志文件，循环监听端口，并创建子线程处理对应的请求。

（2）信号处理函数

```
void child_waiter(int signum){
    wait(NULL);
}
```

为SIGCHLD设置一个信号处理函数，避免僵尸进程。

（3）建立服务器端socket

```
int make_server_socket(int portnum){
    int listenfd;//
    struct sockaddr_in serv_addr;
    int temp;
    bzero((void*)&serv_addr,sizeof(serv_addr));
    serv_addr.sin_family=AF_INET;
    serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);
    serv_addr.sin_port=htons(portnum);
    if((listenfd=socket(AF_INET,SOCK_STREAM,0))==-1){
        return -1;
    }
    if(listen(listenfd,QUEUE_SIZE)!=0){
        return -1;
    }
    return listenfd;
}
```

（4）建立到服务器的连接

```
int connect_to_server(char* host,int portnum){
    int sockfd;
    struct sockaddr_in serv_addr;
    struct hostent *hp;
    if((sockfd=socket(AF_INET,SOCK_STREAM,0))==-1){
        return -1;
    }
    bzero(&serv_addr,sizeof(serv_addr));
    if((hp=gethostbyname(host))==NULL){
        return -1;
    }
    bcopy(hp->h_addr,(struct sockaddr*)&serv_addr.sin_addr,hp->h_length);
    serv_addr.sin_port=htons(portnum);
    serv_addr.sin_family=AF_INET;
    if(connect(sockfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr))!=0){
        return -1;
    }
    return sockfd;
}
```

（5）日志文件

```
void log_fun(FILE *log_fp,char *time_log,char *temp){
    log_fp=fopen("log.txt","a+");
    get_time(time_log);
    if(strlen(temp)!=0){
```

```
        fprintf(log_fp,"%s",time_log);
        fprintf(log_fp,"%s",temp);
        memset(temp,0,sizeof(temp));
    }
    fclose(log_fp);
    log_fp=NULL;
}
```

日志文件记录对应操作的时间戳和对应操作的请求内容。

（6）请求出错404

```
void do_404(char* arg,FILE *fp){
    fprintf(fp,"HTTP/1.0 404 Not Found\r\n");
    fprintf(fp,"Content-type:text/plain\r\n");
    fprintf(fp,"\r\n");
    fprintf(fp,"404 Not Found\r\n");
    fprintf(fp,"The origin server could not find a current representation for the target resource
or is not willing to disclose that one exists.(%s)\r\n",arg);
}
```

此函数返回404错误。在本实验中，当传入的文件路径参数arg不存在时，向服务器发送404错误，并提示不存在文件的文件名称。

（7）请求成功200：解析html文件

```
void send_data(FILE* fp, char* ct, char* file_name){
    printf("send data start: HTTP/1.0 200 OK\n");
    char protocol[]="HTTP/1.0 200 OK\r\n";
    char server[]="Server:Linux Web Server \r\n";
    char cnt_len[]="Content-length:2048\r\n";
    char cnt_type[SMALL_BUF];
    char buf[BUF_SIZE];
    FILE* send_file;
    sprintf(cnt_type, "Content-type:%s\r\n\r\n", ct);
    send_file=fopen(file_name, "r");
    if(send_file==NULL) {
        send_error(fp);
        return;
    }
    fputs(protocol, fp);
    fputs(server, fp);
    fputs(cnt_len, fp);
    fputs(cnt_type, fp);
    while(fgets(buf, BUF_SIZE, send_file)!=NULL) {
        fputs(buf, fp);
        fflush(fp);
    }
```

```
    fflush(fp);
    printf("send data end\n");
}
```

此函数完成解析html文件。首先是返回响应头部，然后通过读取对应文件的方式将响应头部和文件内容一起返回给服务器。

（8）请求成功200：解析图片文件

```
int sendPic(FILE *f,char* arg){
    FILE * fp;
    int len;
    char* readBuf=(char*)malloc(1024*1024*2);
    int tlen;
    char * p_bufs = NULL;
    fp = fopen(arg, "rb");
    if (NULL == fp){
        return -1;
    }
    fseek(fp, 0, SEEK_END);
    len = ftell(fp);
    printf("len = %d\n",len);
    if (len <= 0){
        fclose(fp);
        return -1;
    }
    fseek(fp, 0, SEEK_SET);
    len = fread(readBuf, 1, len, fp);
    fclose(fp);
    p_bufs = (char *)malloc(len + 1024);
    if (NULL == p_bufs){
        printf("malloc error!\n");
        return -1;
    }
    tlen = sprintf(p_bufs,  "HTTP/1.1 200 OK\r\n"
                        "Server: hsoap/2.8\r\n"
                        "Content-Type: image/jpeg\r\n"
                        "Content-Length: %d\r\n"
                        "Connection: close\r\n\r\n",
                        len);
    memcpy(p_bufs+tlen, readBuf, len);
    tlen += len;
    send(fileno(f), p_bufs, tlen, 0);
    free(p_bufs);
    free(readBuf);
    return 0;
```

```
}
```

此函数完成解析图片类型文件。首先需要声明一个足够大的字符串空间来容纳图片二进制流（这里声明的是2MB），否则会段错误。然后通过读取二进制流的方法将图片二进制流读取到缓存区。响应头部需要声明Content-Type: image/jpeg，解释返回的是一个图片文件。最后将响应头部和缓存区的二进制流数据一起返回给服务器。

（9）请求成功200：显示目录文件

```
void do_ls(char* arg,FILE *fp){
    int fd=fileno(fp);
    fprintf(fp,"HTTP/1.0 200 OK\r\n");
    fprintf(fp,"Content-type:text/plain");
    fprintf(fp,"\r\n\r\n");
    fflush(fp);
    printf("store origin STDOUT_FILENO\n");
    int savefd = dup(STDOUT_FILENO);
    printf("redirect towards server\n");
    dup2(fd,STDOUT_FILENO);
    pid_t sub=fork();
    if(sub==0){
        execl("/bin/ls","ls","-l",arg,NULL);
    }
    dup2(savefd, STDOUT_FILENO);
    printf("redirect towards STDOUT_FILENO\n");
}
```

此函数完成显示目录文件的功能。由于本函数的思路是通过重定向输出的方式，因此在重定向输出后还需要重定向到原始状态。因此声明savefd来存储标准化输出的描述符，由于exec族函数在执行完成之后会结束进程，因此调用函数fork创建子进程执行execl函数。执行完成后再重定向标准输出到savefd上。

（10）请求成功200：显示文件内容

```
void do_cat(char* arg,FILE *fp){
    int fd=fileno(fp);
    fprintf(fp,"HTTP/1.0 200 OK\r\n");
    fprintf(fp,"Content-type:text/plain");
    fprintf(fp,"\r\n\r\n");
    fflush(fp);
    printf("store origin STDOUT_FILENO\n");
    int savefd = dup(STDOUT_FILENO);
    printf("redirect towards server\n");
    dup2(fd,STDOUT_FILENO);
    pid_t sub=fork();
    if(sub==0){
        execl("/bin/cat","cat",arg,NULL);
```

```
    }
    dup2(savefd, STDOUT_FILENO);
    printf("redirect towards STDOUT_FILENO\n");
}
```

此函数完成显示目标文件内容的功能。由于本函数的思路是通过重定向输出的方式，因此在重定向输出后还需要重定向到原始状态。因此声明savefd来存储标准化输出的描述符，由于exec族函数在执行完成之后会结束进程，因此调用函数fork创建子进程执行execl函数。执行完成后再重定向标准输出到savefd上。

（11）根据文件后缀判断文件类型：图片类型

```
bool ends_in_pic(char* arg){
    const char *pFile=strrchr(arg,'.');
    if(pFile!=NULL){
        if(strcmp(pFile,".jpg")==0||strcmp(pFile,".png")==0){
            return true;
        }
    }
    return false;
}
```

六、实验运行结果

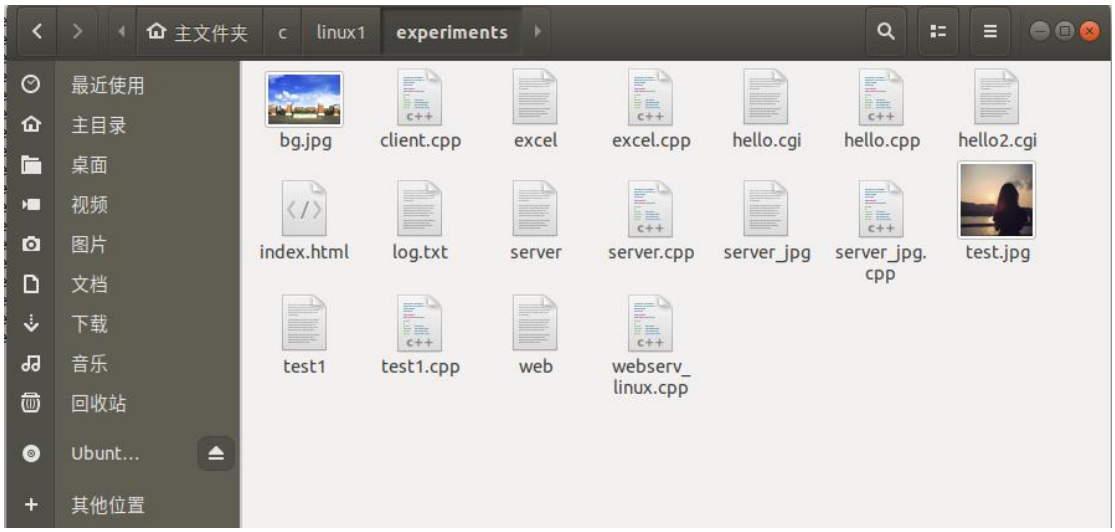
1. 启动Web服务器

```
neuljh@neuljh-virtual-machine:~/c/linux1/experiments$ g++ webserv_linux.cpp -D_REENTRANT -o web -lpthread
webserv_linux.cpp: In function ‘int main(int, char**)’:
webserv_linux.cpp:77:32: warning: ISO C++ forbids converting a string constant to ‘char*’ [-Wwrite-strings]
   error_handling("bind() error");
                  ^
webserv_linux.cpp:79:34: warning: ISO C++ forbids converting a string constant to ‘char*’ [-Wwrite-strings]
   error_handling("listen() error");
                  ^
webserv_linux.cpp: In function ‘char* content_type(char*)’:
webserv_linux.cpp:284:10: warning: ISO C++ forbids converting a string constant to ‘char*’ [-Wwrite-strings]
   return "text/html";
          ^~~~~~~~~~
webserv_linux.cpp:286:10: warning: ISO C++ forbids converting a string constant to ‘char*’ [-Wwrite-strings]
   return "text/plain";
          ^~~~~~~~~~~
webserv_linux.cpp: In function ‘int sendPic(FILE*, char*)’:
webserv_linux.cpp:606:40: warning: format ‘%d’ expects argument of type ‘int’, but argument 2 has type ‘size_t {aka long unsigned int}’ [-Wformat=]
  printf("strlen = %d\n",strlen(readBuf));
                         ~~~~~~~~~~~~~~^
neuljh@neuljh-virtual-machine:~/c/linux1/experiments$ ./web 9190
```

循环监听端口 9190。

2. 显示目录列表

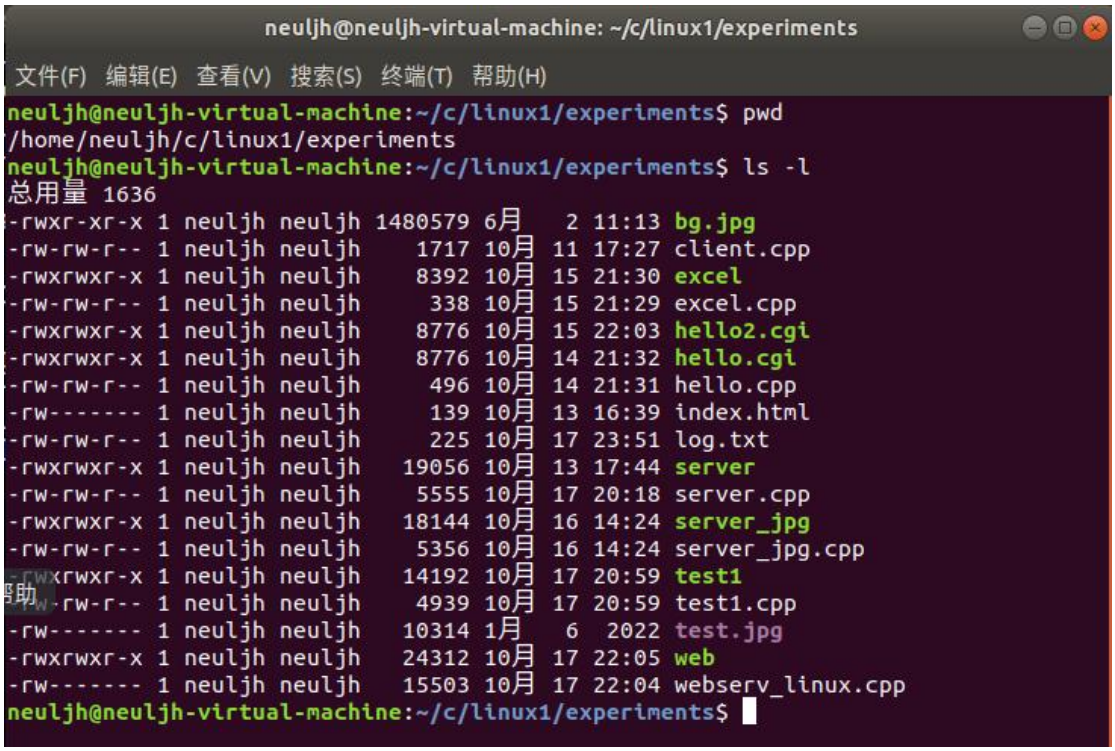
首先随便进入一个目录，这里我们选择文件webserv_linux.cpp（即本实验的c++源文件）所在的目录：



进入终端，查看所在目录：

得到所在目录为：/home/neuljh/c/linux1/experiments

输入命令：ls -l：

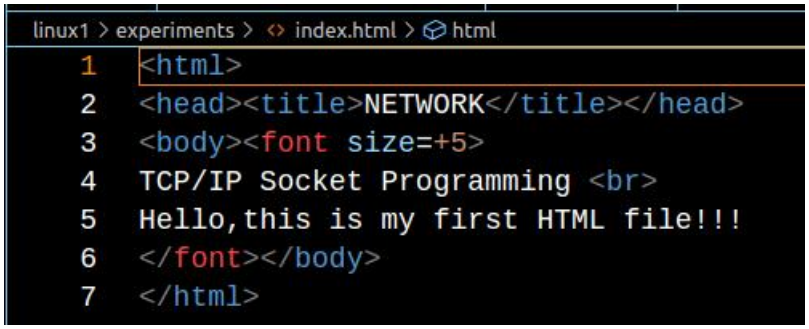


在浏览器中输入网址：
http://localhost:9190/home/neuljh/c/linux1/experiments：

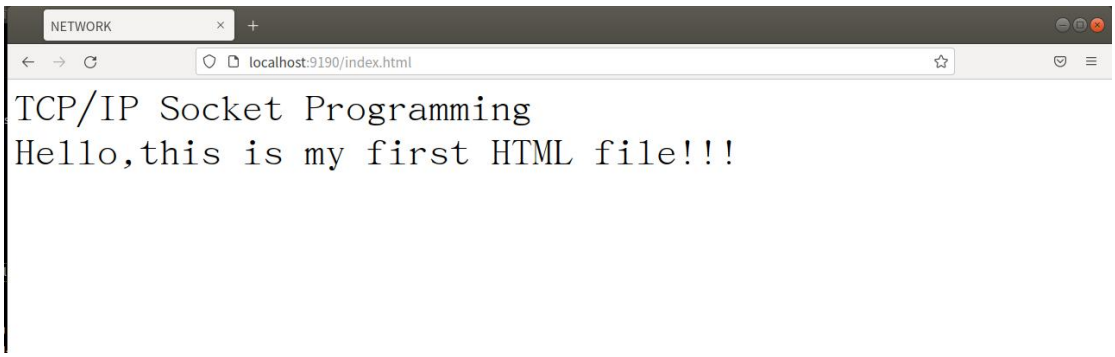
发现浏览器输出内容和在终端的输出内容相同。

3. 解析.html文件

首先查看 index.html 文件源码：



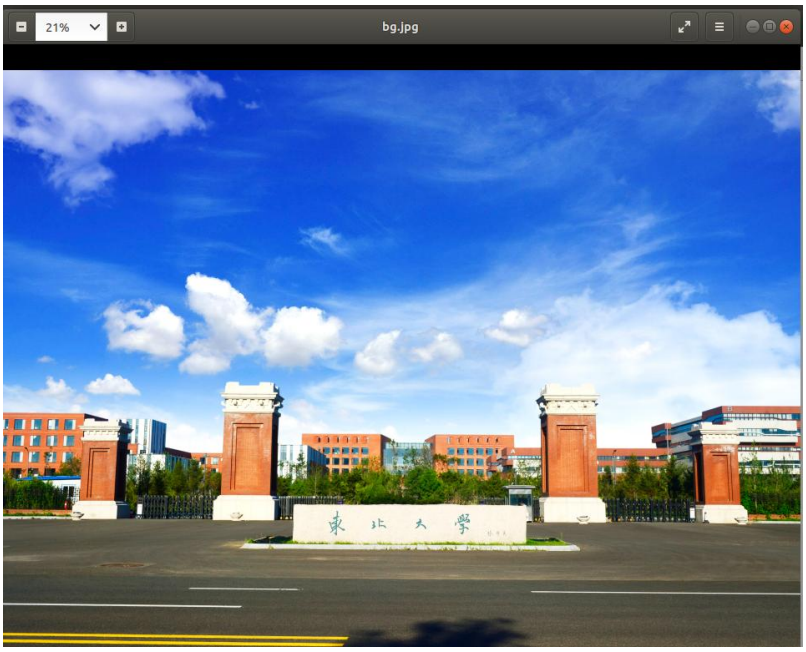
在浏览器中输入网址：http://localhost:9190/index.html：



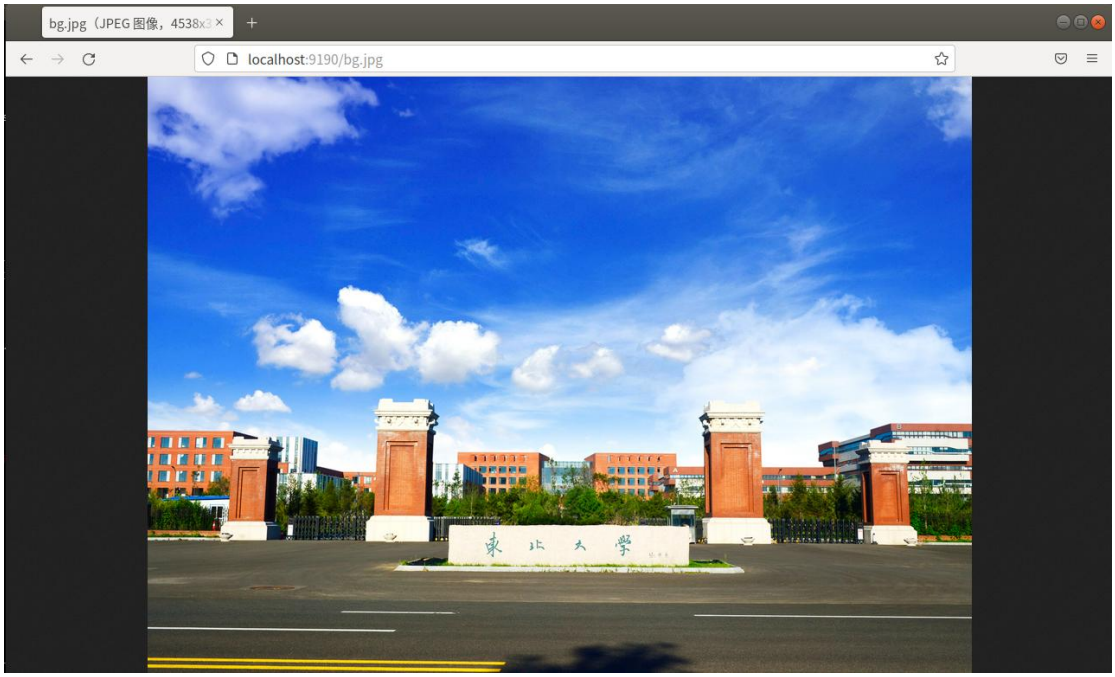
Html文件解析成功。

4. 解析.jpg/.png文件

首先查看图片文件bg.jpg:



在浏览器中输入网址：http://localhost:9190/bg.jpg



解析图片文件成功。

5. 显示文件内容

这里以hello.cpp文件为例子，首先查看hello.cpp文件源码：

linux1 > experiments > hello.cpp > main()

```cpp
#include <iostream>
using namespace std;
int main (){
   cout << "Content-type:text/html\r\n\r\n";
   cout << "<html>\n";
   cout << "<head>\n";
   cout << "<title>Hello World - First CGI Program</title>\n";
   cout << "</head>\n";
   cout << "<body>\n";
   cout << "<h2>Hello World! This is my first CGI program</h2>\n";
   cout << "</body>\n";
   cout << "</html>\n";
}

```

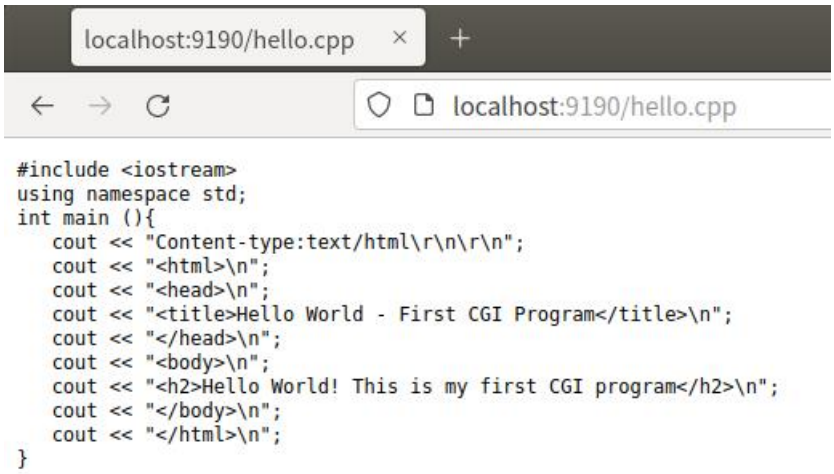
在所在文件目录输入命令：cat hello.cpp

neuljh@neuljh-virtual-machine: ~/c/linux1/experiments

文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
neuljh@neuljh-virtual-machine:~/c/linux1/experiments$ cat hello.cpp
#include <iostream>
using namespace std;
int main (){
   cout << "Content-type:text/html\r\n\r\n";
   cout << "<html>\n";
   cout << "<head>\n";
   cout << "<title>Hello World - First CGI Program</title>\n";
   cout << "</head>\n";
   cout << "<body>\n";
   cout << "<h2>Hello World! This is my first CGI program</h2>\n";
   cout << "</body>\n";
   cout << "</html>\n";
}
neuljh@neuljh-virtual-machine:~/c/linux1/experiments$
```

在浏览器中输入网址：http://localhost:9190/hello.cpp



发现浏览器输出内容和在终端的输出内容相同。

6. 运行.cgi程序

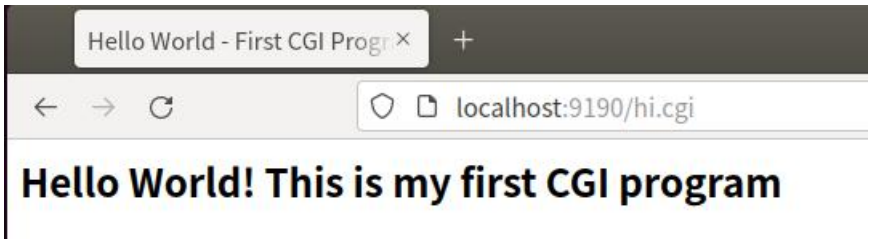
(1)创建hi.cgi文件，并创建shell脚本

linux1 > experiments > hi.cgi

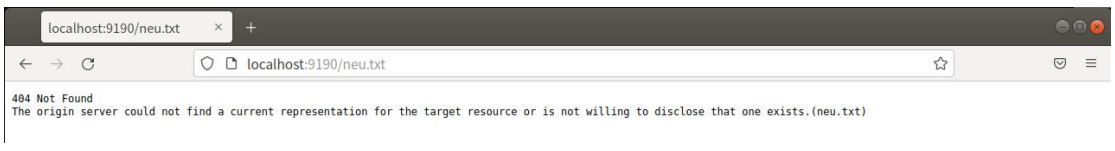
```
1  printf("Content-type:text/html\r\n\r\n")
2  printf("<html>\n");
3  printf("<head>\n");
4  printf("<title>Hello World - First CGI Program</title>\n");
5  printf("</head>\n");
6  printf("<body>\n");
7  printf("<h2>Hello World! This is my first CGI program</h2>\n");
8  printf("</body>\n");
9  printf("</html>\n");
```

(2)使用chmod命令修改hi.cgi权限为755

(3)运行hi.cgi程序



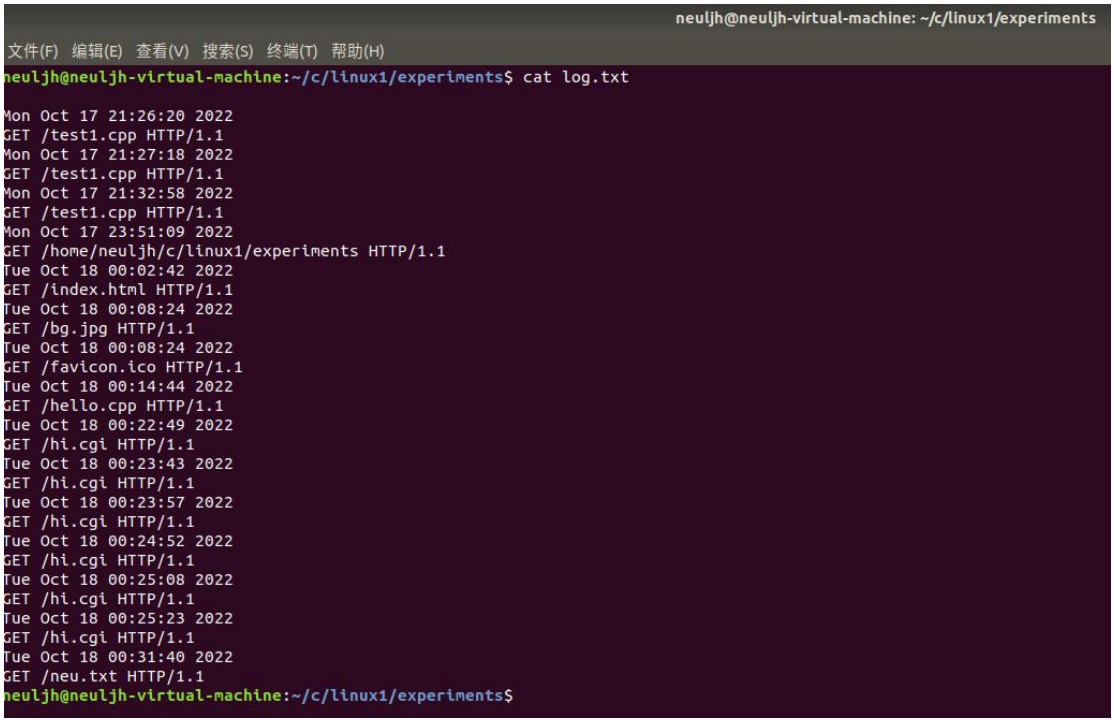
7. 请求无效时显示错误信息



当对应路径或者目录不存在时，此时请求无效，浏览器会显示404错误。

8. 日志记录

在本实验中我还加入了日志记录功能，实时记录浏览器的请求内容及时间戳。相关信息记录在同目录下的log.txt文件中。



使用cat命令查看log.txt文件的日志内容。

七、实验总结

通过本次实践，我了解到了并发服务器的运行模式和套接字编程的基本原理，理解了在Linux中进程的基本概念和基本的进程函数，知道如何利用fork来接收多个请求、创建子进程，以及使用SIGCHLD来阻止僵尸问题。

在本实验中，学习到了网络编程中关于简易HTTP-web服务器编写，掌握了HTTP协议在网络传输中的基本原理和基本规则。更为受益匪浅的是尽管本次实验工程量浩大，但是所需要掌握的知识点都是平时上课所学习的。感受到了理论知识和实际相结合的快乐。

实验课程加深了我对于课堂内容的理解，实践和理论相结合，提高了我的动手能力和编程能力，获益匪浅。

最后感谢老师的辛勤付出！

附录：

《Linux 程序设计》成绩评定表

评价表格

考核标准	得分
（1）正确理解和掌握实验所涉及的概念和原理（20%）；	
（2）按实验要求合理设计数据结构和程序结构，运行结果正确（40%）；	
（3）实验过程中，具有严谨的学习态度和认真、踏实、一丝不苟的科学作风，实验报告规范；认真记录实验数据，原理及实验结果分析准确（40%）；	
合计	