课程编号：B080203060

# 《Linux 程序设计》实验报告


| | | | |
|---|---|---|---|
| 姓　　名 | 薛　　旗 | 学　　号 | 2 0 1 5 5 3 6 2 |
| 班　　级 | 软信-1503 | 指　导　教　师 | 王学毅 |
| 实　验　名　称 | 多进程编程 | | |
| 开　设　学　期 | 2017-2018 第一学期 | | |
| 开　设　时　间 | 第 18 周——第 19 周 | | |
| 报　告　日　期 | 2018 年 1 月 8 日 | | |
| 评　定　成　绩 | | 评　定　人 | 王学毅 |
| | | 评　定　日　期 | 2018 年 1 月 12 日 |

东北大学软件学院

# 实验一　多进程编程

## 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**

- 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。
- 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。Web 服务器在端口 80 监听。

（1）HTTP 请求：GET

（2）HTTP 应答：OK

**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，一个以.cgi

结尾的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：

$cc webserv.c socklib.c –o webserv

$./webserv 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。

## 三、函数说明及关键代码

| 调用函数 | 函数说明 |
|---|---|
| int make_server_socket(int) | 建立服务器端 socket |
| int connect_to_server(char *, int) | 建立到服务器的连接 |
| void get_time(char *) | 获得当前时间字符串 |
| void log_fun(FILE *, char *, char *) | 日志 |
| void child_waiter(int) | 为 SIGCHLD 设置一个信号处理函数 |
| void read_request(FILE*) | 读取完整的请求 |
| void process_rq(char *, int) | 处理请求 |
| void get_header(FILE *, char*) | 获取头部信息 |
| void do_501(int) | 请求 501 错误 |
| void do_404(char *, int) | 请求出错 404 |
| int iscata(char*) | 判断是否为目录 |
| int not_exist(char *) | 判断请求是否存在 |
| void do_ls(char*, int) | 目录列表函数 |
| char* file_type(char *) | 返回文件类型 |
| int ends_in_cgi(char *) | 判断是否为 cgi 类型文件 |
| void do_exec(char *, int) | 执行可执行程序 cgi |
| void do_cat(char*, int) | 显示当前目录下全部文件或目录 |

### 1.Web 服务器主函数

```
1  void main(int ac, char*av[]) {
2
3      char time_log[BUFSIZ];
4      int sock, fd;
5      FILE *fpin;
6      char request[BUFSIZ];
7      if (ac == 1) {
8          fprintf(stderr, "usage:ws portnum\n");
```

```
9           exit(1);
10      }
11      //阻止僵尸进程
12      signal(SIGCHLD, child_waiter);
13      if ((sock = make_server_socket(atoi(av[1]))) == -1)
14          exit(1);
15
16      //创建日志文件
17      FILE *log_fp = fopen("log.txt", "a+");
18      if (log_fp == NULL) {
19          fprintf(stderr, "ERROR: Create a log file failed!\n\n");
20          exit(1);
21      }
22      fclose(log_fp);
23
24      while (1) {
25          fd = accept(sock, NULL, NULL);
26          if (fd == -1)
27              break;
28          fpin = fdopen(fd, "r");
29          fgets(request, BUFSIZ, fpin);
30          printf("Request = %s", request);
31          read_request(fpin);
32          //处理请求
33          process_rq(request, fd);
34          //结束本次请求
35          fclose(fpin);
36          log_fun(log_fp, time_log, request);
37      }
38  }
```

## 2.信号处理函数

```
1  void child_waiter(int signum) {
2      wait(NULL);
3  }
```

## 3.日志文件

```
1  void log_fun(FILE *log_fp, char *time_log, char *temp) {
2      log_fp = fopen("log.txt", "a+");
3      get_time(time_log);
4      if (strlen(temp) != 0) {
5          fprintf(log_fp, "%s", time_log);
```

```
6            fprintf(log_fp, "%s", temp);
7            memset(temp, 0, sizeof(temp));
8        }
9        fclose(log_fp);
10       log_fp = NULL;
11   }
```

## 4.建立服务器端 socket

```
1    int make_server_socket(int portnum) {
2        int listenfd;  //服务器端文件描述符
3        struct sockaddr_in servaddr;  //存放ip 端口信息的结构体
4
5                                          //初始化servaddr
6        bzero((void *)&servaddr, sizeof(servaddr));
7        servaddr.sin_family = AF_INET;
8        servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
9        servaddr.sin_port = htons(portnum);
10
11       //创建服务器socket
12       listenfd = socket(AF_INET, SOCK_STREAM, 0);
13       if (listenfd == -1)
14           return -1;
15
16       //绑定
17       if (bind(listenfd, (struct sockaddr *)&servaddr, sizeof(servaddr)) != 0)
18           return -1;
19
20       //监听
21       if (listen(listenfd, 128) != 0)
22           return -1;
23       return listenfd;
24   }
```

## 5.建立到服务器的连接

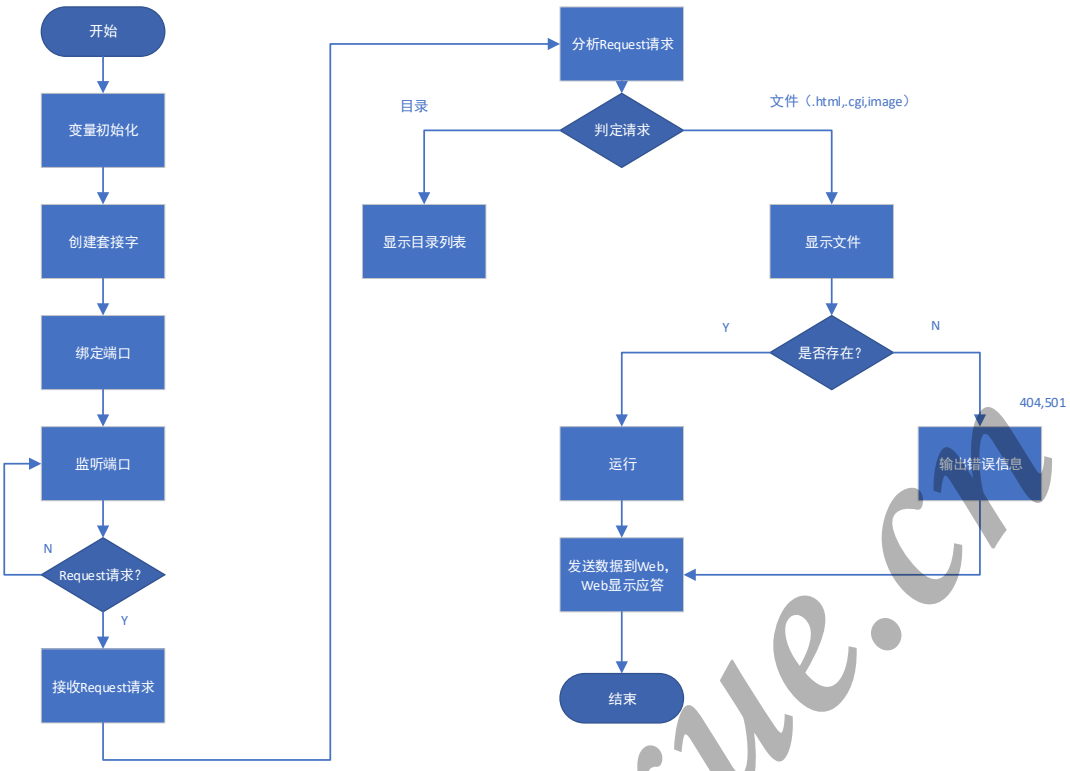
```
1    int connect_to_server(char *host, int portnum) {
2        int sock;
3        struct sockaddr_in servaddr;
4        struct hostent *hp;
5
6        //得到一个socket
7        sock = socket(AF_INET, SOCK_STREAM, 0);//get a line
8        if (sock == -1)
```

```
9           return -1;
10
11      //连接服务器
12      bzero(&servaddr, sizeof(servaddr));
13      hp = gethostbyname(host); //得到主机IP
14      if (hp == NULL)
15          return -1;
16      bcopy(hp->h_addr, (struct sockaddr*)&servaddr.sin_addr, hp->h_length);
17      servaddr.sin_port = htons(portnum);
18      servaddr.sin_family = AF_INET;
19
20      if (connect(sock, (struct sockaddr*)&servaddr, sizeof(servaddr)) != 0)
21          return -1;
22
23      return sock;
24  }
```

### 6.请求出错 404

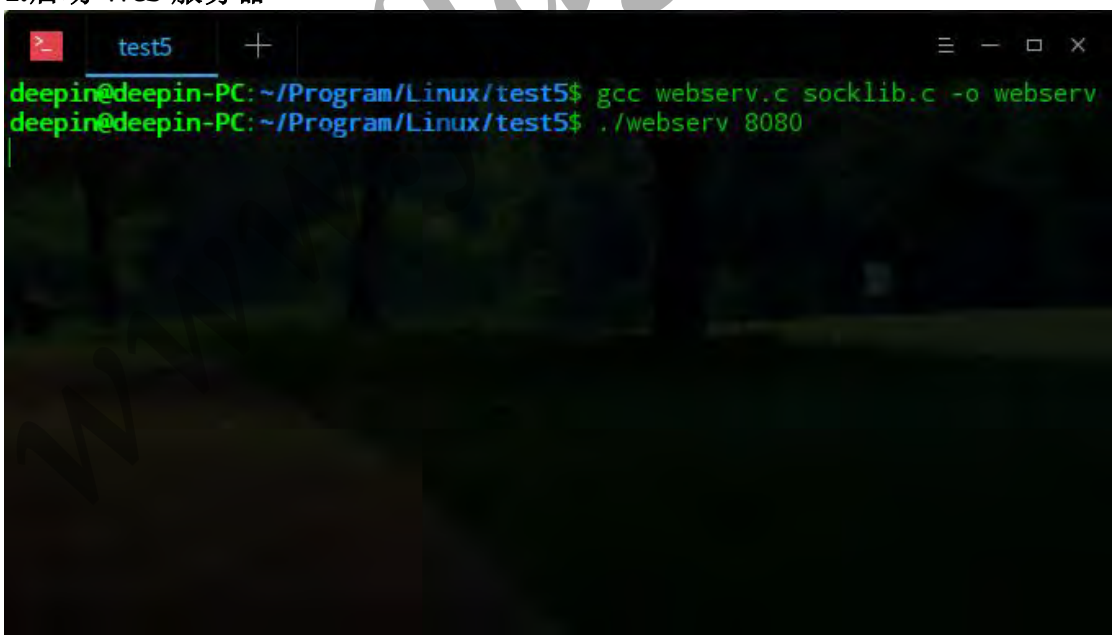
```
1  void do_404(char *item, int fd) {
2      FILE *fp = fdopen(fd, "w");
3      fprintf(fp, "HTTP/1.0 404 Not Found\r\n");
4      fprintf(fp, "Content-type:text/plain\r\n");
5      fprintf(fp, "\r\n");
6      fprintf(fp, "404 Not Found\r\n");
7      fprintf(fp, "The origin server did not find a current representation for the
       target resource or is not willing to disclose that one exists.(%s)\r\n", item);
8      fclose(fp);
9  }
```

## 四、流程图

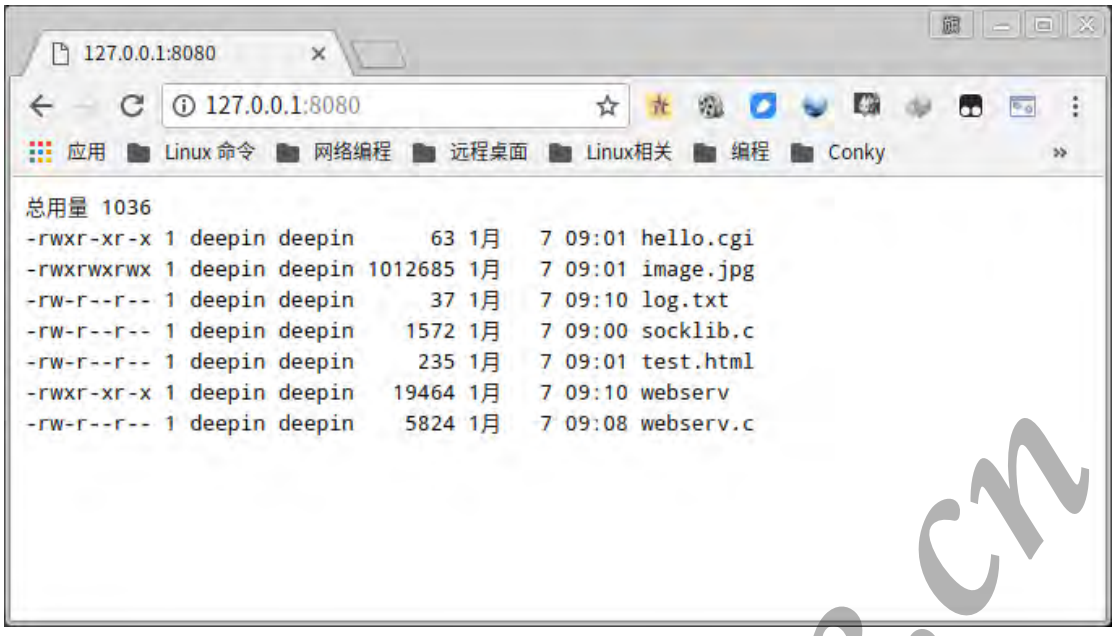



## 五、实验运行结果

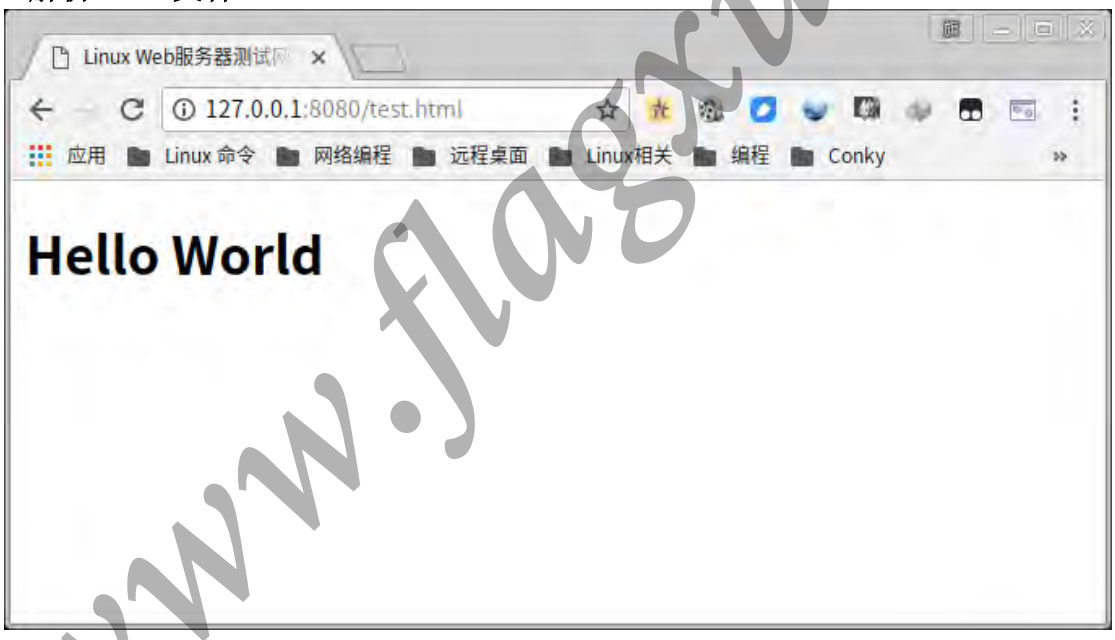
### 1.启动 Web 服务器



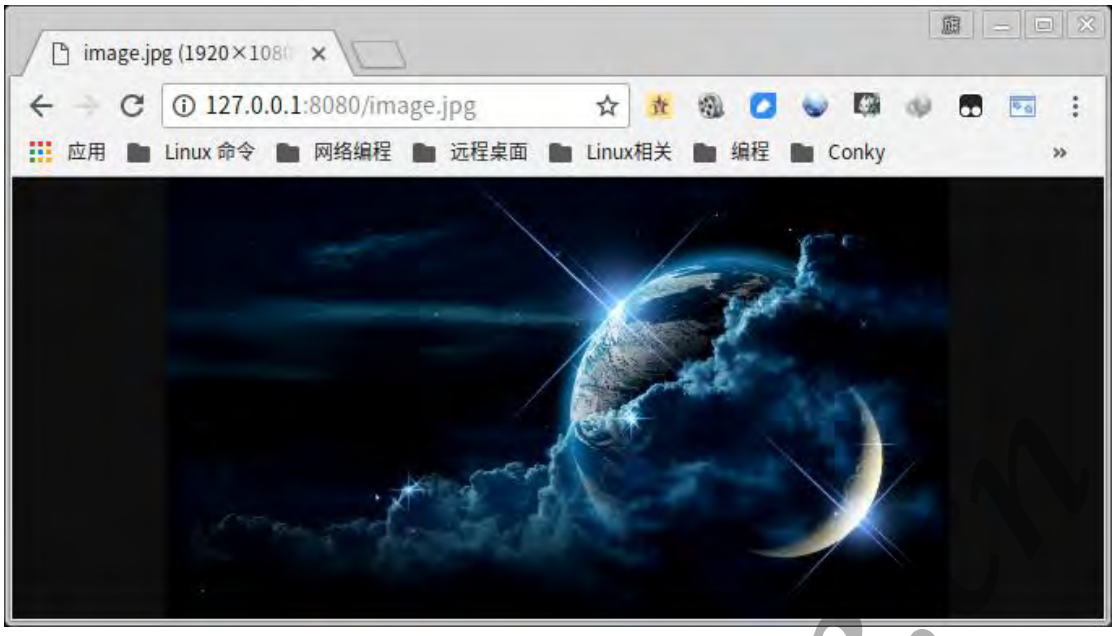

### 2.显示目录列表




**3.解析.html 文件**

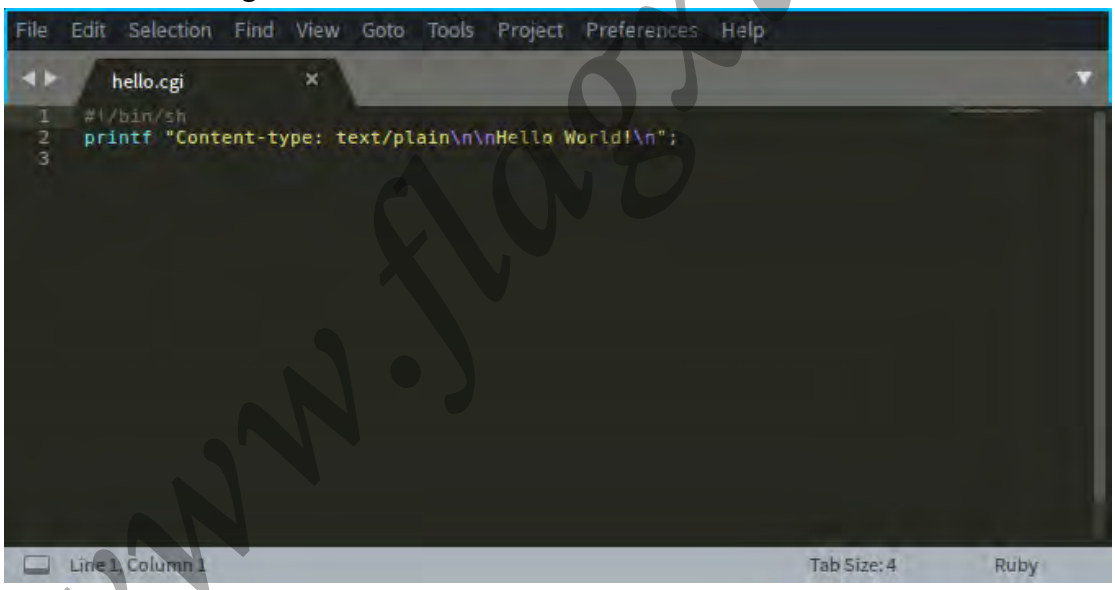



**4.绘图**

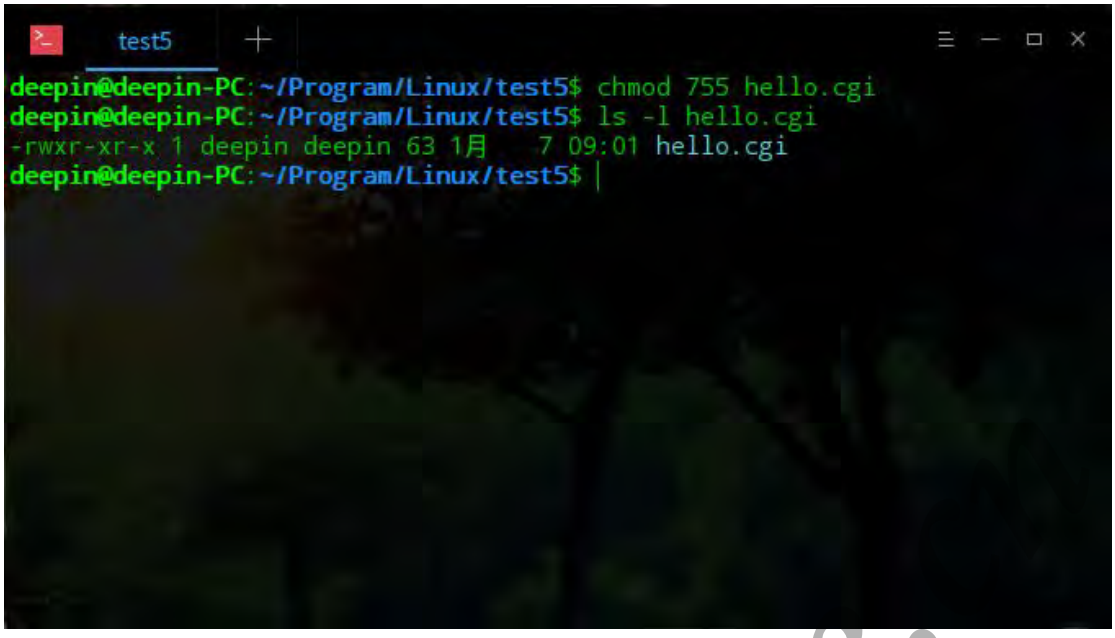



**5.运行.cgi 程序**

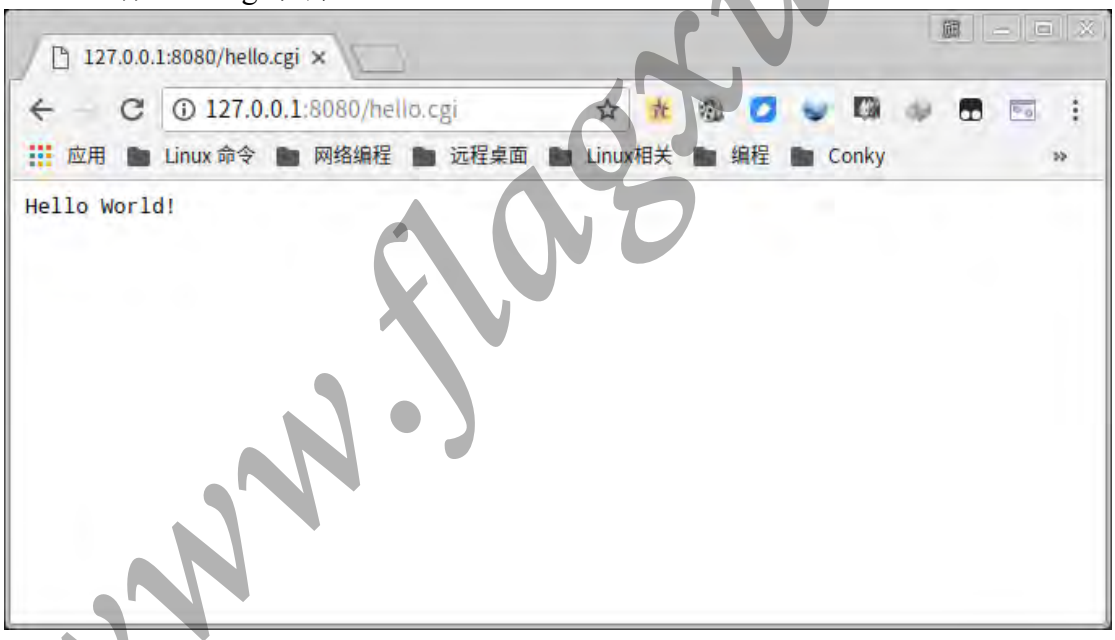
（1）创建 hello.cgi 文件，并创建 shell 脚本




（2）使用 chmod 命令修改 hello.cgi 权限为 755




（3）运行 hello.cgi 程序

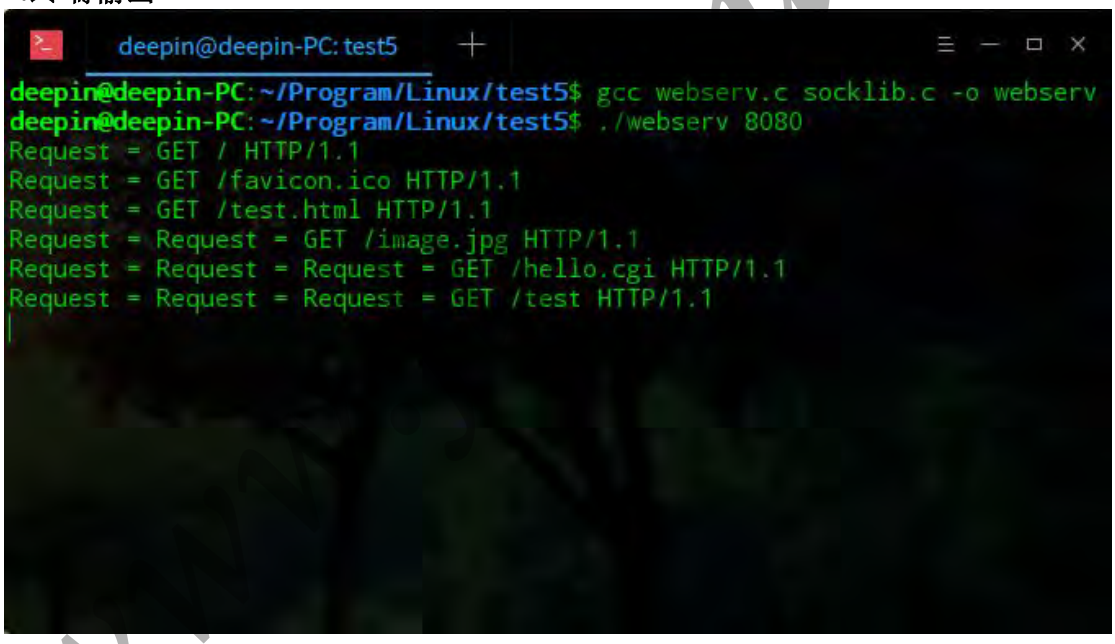




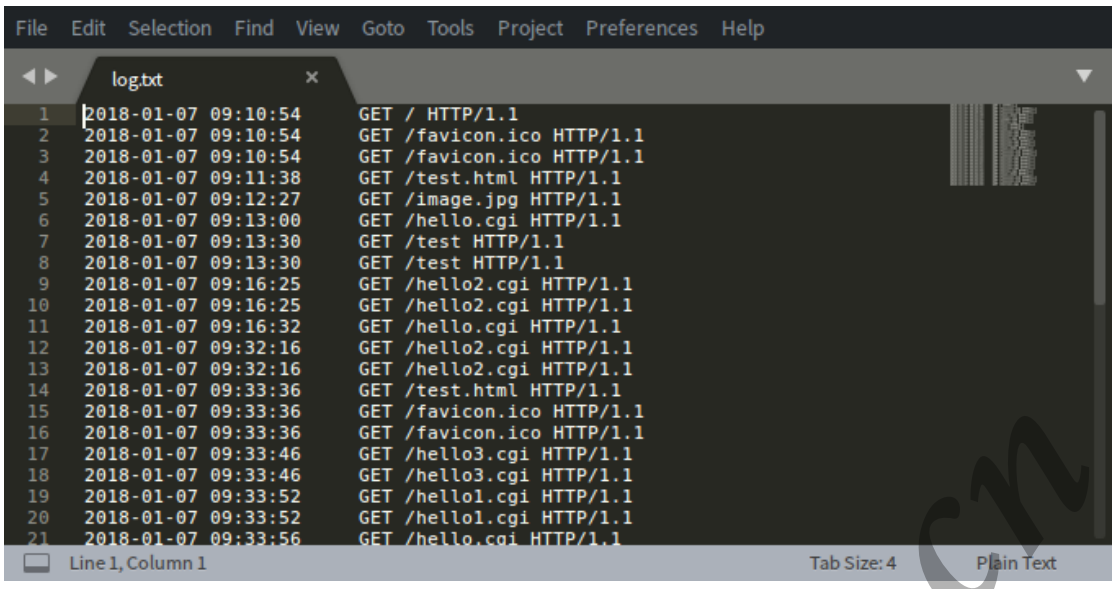

**6.请求无效，显示错误信息**


127.0.0.1:8080/test
127.0.0.1:8080/test
应用
Linux 命令
网络编程
远程桌面
Linux相关
编程
Conky
404 Not Found
The origin server did not find a current representation for the target resource or
is not willing to disclose that one exists.(.//test)

**7.终端输出**


deepin@deepin-PC: test5
deepin@deepin-PC:~/Program/Linux/test5$ gcc webserv.c socklib.c -o webserv
deepin@deepin-PC:~/Program/Linux/test5$ ./webserv 8080
Request = GET / HTTP/1.1
Request = GET /favicon.ico HTTP/1.1
Request = GET /test.html HTTP/1.1
Request = Request = GET /image.jpg HTTP/1.1
Request = Request = Request = GET /hello.cgi HTTP/1.1
Request = Request = Request = GET /test HTTP/1.1

**8.日志记录**




## 五、实验总结

通过本次实践，我了解了并发服务器的运行模式和套接字的使用方法，理解了进程的基本概念和基本的进程函数，知道了如何利用 fork 来接收多个请求，以及使用 SIGCHLD 来阻止僵尸问题。实验课程加深了我对课堂内容的理解，实践与理论相结合，提高了我的动手能力和编程能力，获益匪浅。

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（20%）； | |
| （3）运行结果正确（20%）； | |
| （4）认真记录实验数据，原理及实验结果分析准确（20%）； | |
| (5)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(10%)； | |
| （7）实验报告规范（10%）。 | |
| 合计 | |