



计算机网络

实验三

姓 名	邓语苏
-----	-----

学	22920212204066
---	----------------

日	2023 年 11 月 13 日
---	------------------

学	信息学院
---	------

课程名称	计算机网络
------	-------

实验三

目录

1 编译和运行代码	1
2 任务一：完善 Socket 客户机和服务器	1
2.1 任务要求	1
2.2 程序实现	1
2.2.1 通过命令行参数指定 IP 和端口	1
2.2.2 服务器端处理 SIGINT 信号	2
2.2.3 为 Socket 添加错误处理	2
2.2.4 允许用户输入字符串	3
2.2.5 使用 bye 退出	3
2.2.6 将服务器端改为迭代服务器	3
2.3 运行情况	3
3 任务二：课程查询服务器 (TCP 迭代)	4
3.1 任务要求	4
3.2 程序设计	4
3.3 运行情况	6
4 Server 端的 Backlog	6
5 任务三：课程查询服务器 (TCP 并发)	7
5.1 任务要求	7
5.2 程序实现	7
5.3 并发服务器的运行情况	7
5.4 父进程分支中是否需要关闭 Socket	8
5.4.1 理论分析	8
5.4.2 实验验证	10
6 任务四：获取网页内容 (Python 自学)	10

6.1	任务要求	10
6.2	Socket 爬虫实现	10
6.3	爬虫改进	11
6.4	结果对比	11
7	实验总结	12
	附录：代码清单	13

1 编译和运行代码

实验代码在以下环境中进行了编译和运行：

- 操作系统：Ubuntu 22.04 (on Windows Subsystem for Linux)
- 编译器：GCC 11.3.0
- CMake：3.22.1

为了编译代码，需要使用 CMake，在项目目录下建立目录 build，然后依次运行：

1. cd build
2. cmake ..
3. make

就可以得到 server 和 client 两个可执行文件。运行方式都是”可执行文件名 IP PORT”。分别为程序指定 IP 和端口。

2 任务一：完善 Socket 客户机和服务器

2.1 任务要求

按以下要求，修改范例 *client_example.c*，如代码 10，实现类似 telnet 连接 echo 服务器的效果：

1. 为所有 socket 函数调用添加错误处理代码；
2. 范例中服务器地址和端口是固定值，请将它们改成允许用户以命令行参数形式输入；
3. 范例中客户机发送的是固定文本 “Hello Network!”，请改成允许用户输入字符串，按回车发送；
4. 实现循环，直至客户机输入 “bye” 退出。

2.2 程序实现

2.2.1 通过命令行参数指定 IP 和端口

首先，为了实现通过参数指定服务器 IP 和端口，需要处理命令行参数。先检查主函数的 *argc*，如果不是 3，说明参数不对，打印提示信息并退出。如代码 1。

代码 1: 检查 *argc*

```
1 if (argc != 3) {  
2     printf("Usage: %s <server_ip> <port>\n", argv[0]);  
3     return 1;  
4 }
```

然后在构造 *sever_addr* 结构体的时候根据输入的 IP 和端口号进行赋值。用户输入的端口号是一个数字字符串，首先使用 *atoi* 转换成整数，然后通过 *htons* 转换成网络字节序。而用户输入的 IP 地址是一个点分十进制字符串，需要使用 *inet_aton* 转换成一个整形。如代码 2。

代码 2: 构造 *server_addr* 结构体

```
1  /* 指定服务器地址 */
2  server_addr.sin_family = AF_INET;
3  server_addr.sin_port = htons(atoi(argv[2]));
4  if (inet_aton(argv[1], &server_addr.sin_addr) == 0) {
5      perror("inet_aton");
6      return 1;
7  }
```

这样, 就可以通过命令行参数指定服务器 IP 和端口了。

2.2.2 服务器端处理 SIGINT 信号

由于服务器端使用 *bind* 绑定了端口, 如果退出时没有正确释放资源, 端口就会被占用。之后的运行就需要指定其他端口, 造成了资源的浪费。使用 Ctrl+C 退出时, 服务器端会收到 SIGINT 信号, 可以在主函数中捕获这个信号, 然后在信号处理函数中调用 *close* 释放资源。否则程序会直接退出, 因而无法释放资源, 造成问题。如代码 3。

代码 3: 捕获 SIGINT 信号

```
1  void release_socket(int signo)
2  {
3      close(server_sock_listen);
4      exit(-1);
5  }
6
7  // in int main()
8  signal(SIGINT, release_socket);
```

2.2.3 为 Socket 添加错误处理

为了给 Socket 添加错误处理, 需要在每次调用 Socket 函数的时候检查返回值, Socket 如果返回 -1, 说明调用失败, 那么就打印错误信息并退出。

代码 4: 为 socket 添加错误处理

```
1  /* 连接服务器 */
2  if (connect(client_sock, (struct sockaddr *)&server_addr, sizeof(server_addr)) == -1) {
3      perror("connect");
4      return 1;
5  }
6
7  if (send(client_sock, send_msg, strlen(send_msg), 0) == -1) {
8      perror("send");
9      break;
10 }
11 if (recv(client_sock, recv_msg, sizeof(recv_msg), 0) == -1) {
12     perror("recv");
13     break;
14 }
```

2.2.4 允许用户输入字符串

为了允许用户输入字符串, 可以开辟一定的缓冲区, 然后使用 *fgets* 输入任意字符。使用 *fgets* 的好处是它可以指定缓冲区长度, 可以防止缓冲区溢出, 更为安全。另外, 使用 *fgets* 输入的字符串末尾会有一个 `'\0'`, 所以不需要再手动添加 `'\0'`。也不会因为忘记添加该符号而造成问题。

2.2.5 使用 bye 退出

服务器端需要把处理和返回消息的部分加入一个循环中, 直到收到 `bye` 消息才退出。客户端在服务器返回 `bye` 消息后再释放资源退出。

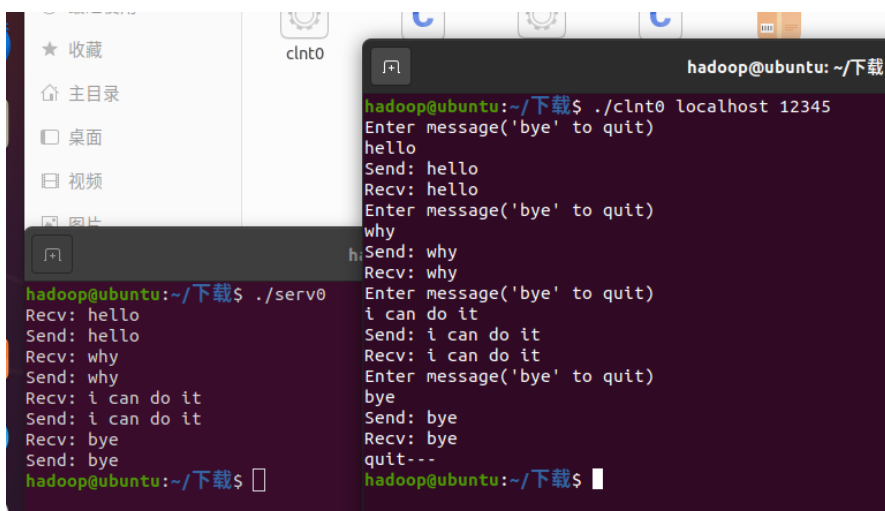
2.2.6 将服务器端改为迭代服务器

迭代服务器可以依次对所有的客户机进行服务。在服务器端, 需要使用一个循环来接收客户机的连接请求。所有的请求都在服务器的单个进程中依次处理。为此, 可以把服务器端的代码改为如代码 13 所示。将 *accept* 调用以及对消息的处理放进一个循环中, 这样就可以依次接收多个客户机的连接请求了。当客户机输入 `bye` 的时候进入下一次循环, 服务器端就会关闭当前的连接, 然后 *accept* 等待下一个客户机的连接请求。

2.3 运行情况

客户机循环输入字符串, 服务器接受并输出相同字符串, 直至接收到 `bye`

图 1: 基础迭代服务器

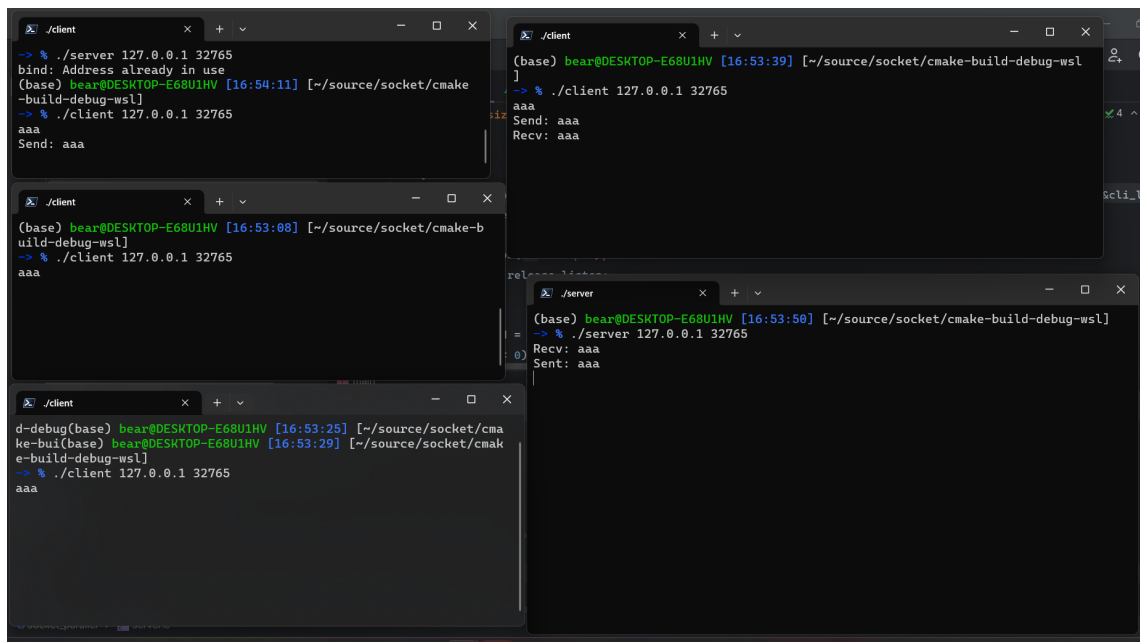


```
hadoop@ubuntu: ~/下载
hadoop@ubuntu:~/下载$ ./clnt0 localhost 12345
Enter message('bye' to quit)
hello
Send: hello
Recv: hello
Enter message('bye' to quit)
why
Send: why
Recv: why
Enter message('bye' to quit)
i can do it
Send: i can do it
Recv: i can do it
Enter message('bye' to quit)
bye
Send: bye
Recv: bye
quit---
hadoop@ubuntu:~/下载$

hadoop@ubuntu:~/下载$ ./serv0
Recv: hello
Send: hello
Recv: why
Send: why
Recv: i can do it
Send: i can do it
Recv: bye
Send: bye
hadoop@ubuntu:~/下载$
```

如图 2, 运行迭代版的服务器, 用 4 个客户端进行连接。可以看到, 服务器端只能依次处理 4 个客户端的请求。每次只有一个服务端能得到处理。当该客户端输入 `bye` 结束后, 下一个客户端才能得到处理。

图 2: 迭代版的服务器



3 任务二：课程查询服务器 (TCP 迭代)

3.1 任务要求

按以下要求，修改范例 `server_example.c`，如代码 11：

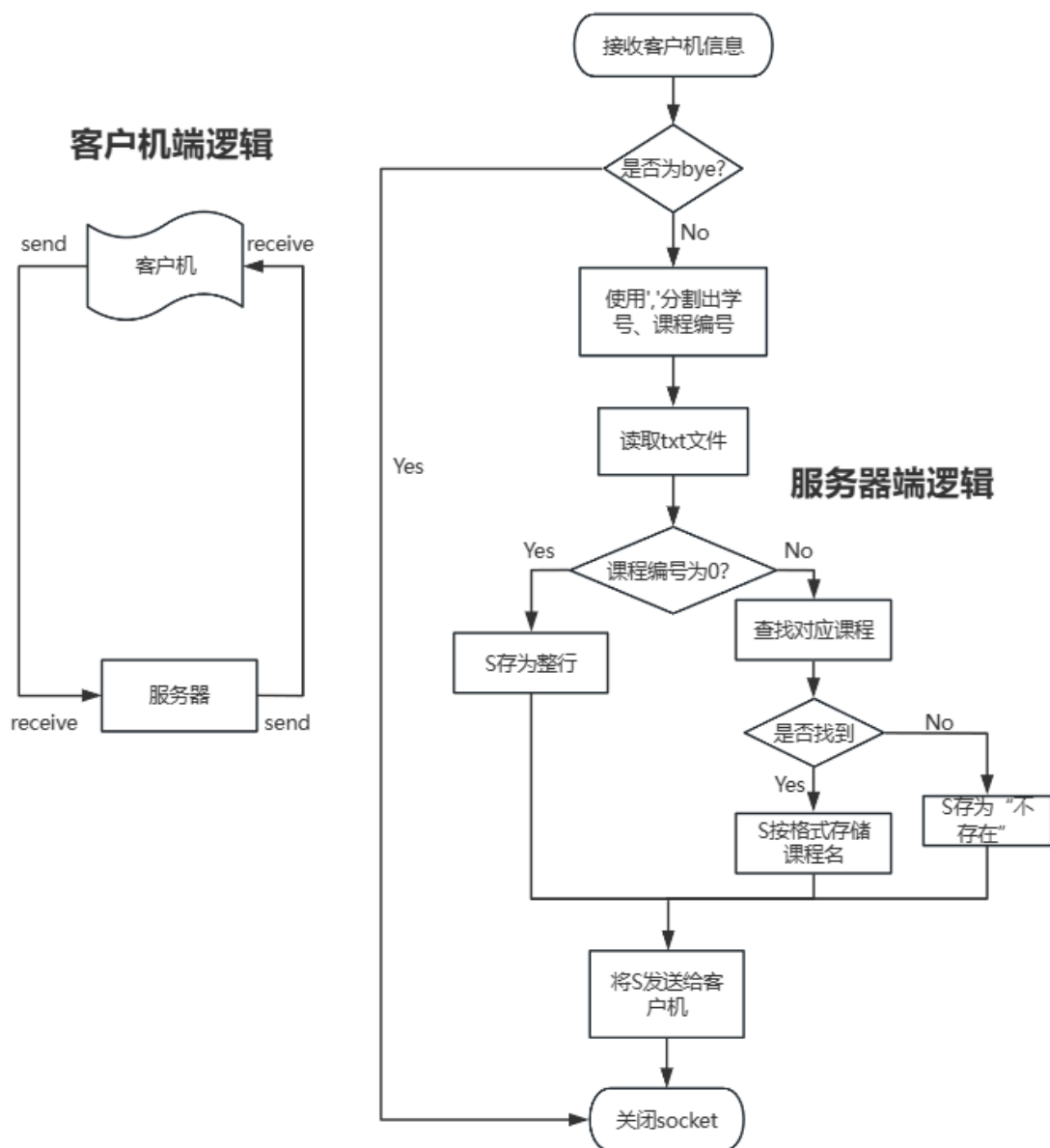
1. 为所有 `socket` 函数调用添加错误处理代码；
2. 范例中服务器地址和端口是固定值，请将它们改成允许用户通过命令行参数形式输入；
3. 实现循环，直至客户机输入 “bye” 退出；
4. 服务器迭代地处理客户机请求：查询给定的 `testlist`，将结果回复客户机；一个客户机退出后、继续接受下一个，按 `Ctrl+C` 可以终止服务器程序。

3.2 程序设计

客户机向服务器发送一个字符串，服务器接收后判断是否为 bye，为 bye 则关闭 Socket；否则查询课程表。

将客户机发送的字符串适用“,”分割出学号,课程编号。若只输入学号,课程编号为0。接着按行读取txt文件,并判断课程编号是否为0。若为0则按行对比txt文件,将学号相同的行整行取出发给客户机。若不为0,在找到具有相同行的条件下查找相应课程,找到则将所需信息整合发送给客户机;否则发送“不存在”给客户机。

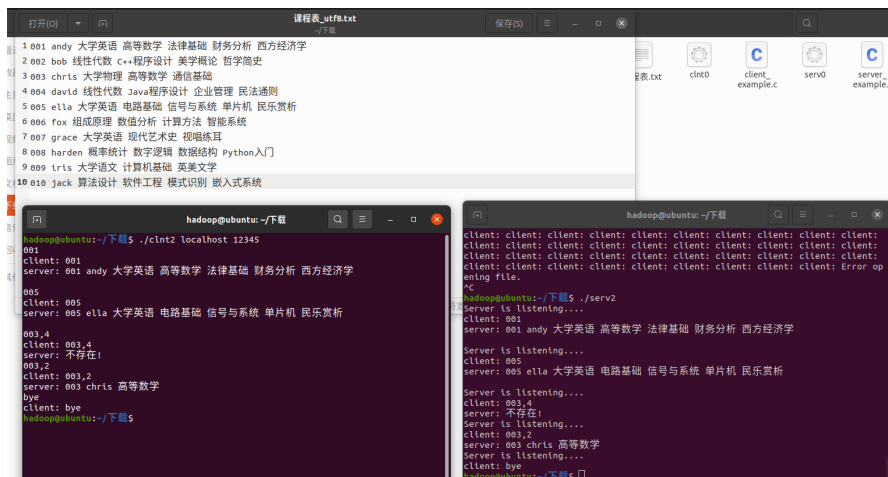
图 3: 任务二流程图



需要注意的是，由于 Windows 和 Linux 系统的编码区别 (UTF 和 ISO)，导致原本编码为 UTF-8 的 txt 文件到 Linux 中会转为 ISO 文件。而 C 语言默认读取 UTF-8 编码的文件，故会乱码。因此需要在 Linux 系统中将文件转化为 UTF-8 的编码才可避免乱码。

3.3 运行情况

图 4: 课程查询服务器 (TCP 迭代)



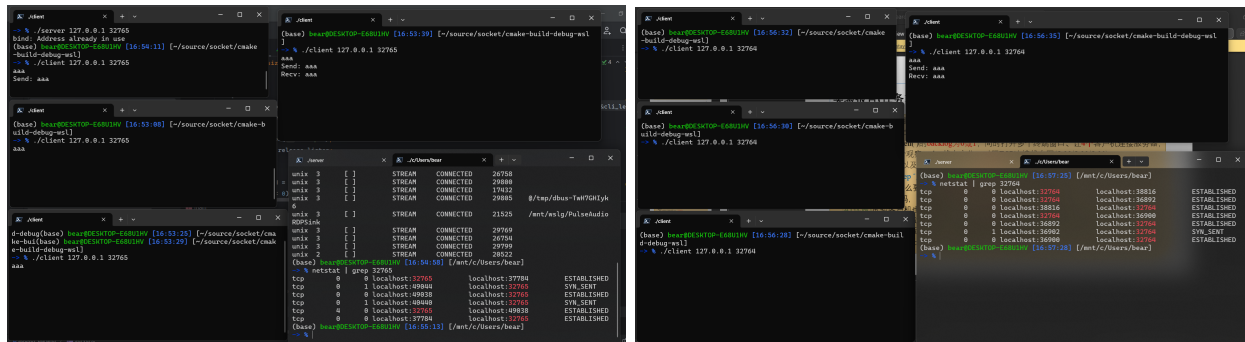
4 Server 端的 Backlog

在服务器端, Socket 建立连接主要分为以下几步:

1. 将 TCP 状态设置为 LISTEN 状态, 开启监听客户端的连接请求
2. 收到客户端发送的 SYN 报文后, TCP 状态切换为 SYN RECEIVED, 并发送 SYN ACK 报文
3. 收到客户端发送的 ACK 报文后, TCP 三次握手完成, 状态切换为 ESTABLISHED

其中, 第一步通过调用 *listen* 完成。其中 *backlog* 是已完成的连接队列 (ESTABLISHED) 与未完成连接队列 (SYN_RCVD) 之和的上限。将参数设置为 0 和 1 的运行结果使用 *netstat* 进行统计, 如图 ?? 所示。

图 5: 不同的 *backlog*



(a) *backlog* = 0

(b) *backlog* = 1

可以发现, 对于 4 个尝试建立连接的客户端, 当 *backlog* = 0 时, 使用 *netstat* 可以发现 2 对连接处于 ESTABLISHED 状态, 当 *backlog* = 1 时, 使用 *netstat* 则有 3 对。可见, *backlog* 确实限制了服务器端的连接数。

5 任务三：课程查询服务器 (TCP 并发)

5.1 任务要求

在任务 2 的基础上修改服务器代码，以多进程的方式提供并发服务，实现“同时”与多个客户机交替对话

5.2 程序实现

为了实现多进程，需要使用 *fork* 系统调用。首先，为了方便建立多个连接，需要将 *listen* 的参数 *n* 设置成较大的数，例如 5。然后循环进行 *accept* 调用，接收客户机的连接请求。

一旦接受了客户机的连接请求，就需要创建一个子进程来处理客户机的请求。在父进程中，需要关闭服务器端的套接字，在子进程中，执行正常的 TCP Echo Server 的执行流程。当收到 *bye* 结束时，子进程需要关闭套接字，释放资源，然后退出。

判断父进程和子进程通过 *fork* 的返回值来判断。如果 *fork* 返回值为 0，说明是子进程，如果返回值大于 0，说明是父进程。

具体实现如代码 5 所示

代码 5: TCP 并发核心代码

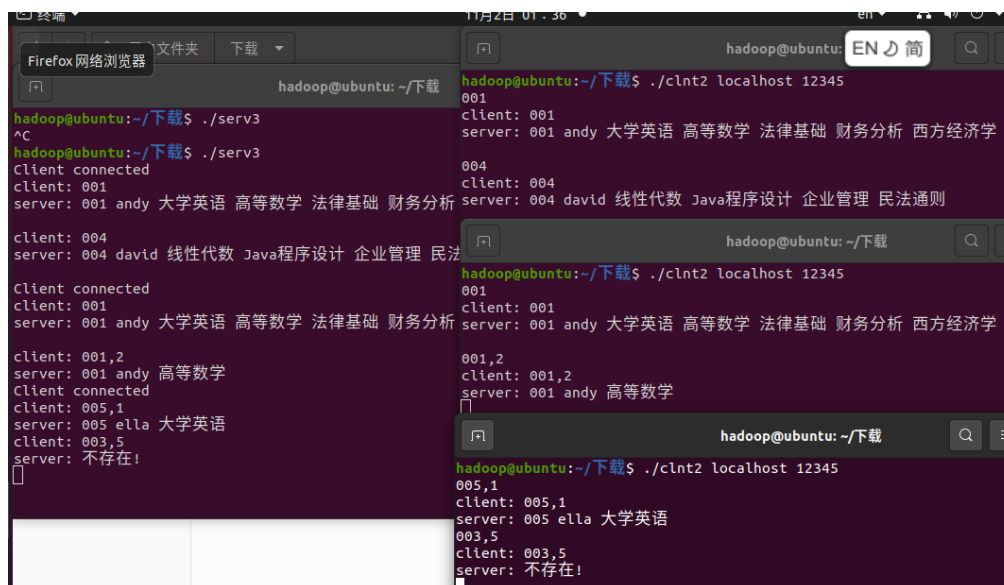
```
1  while (1) {
2      /* 接受客户端连接 */
3      client_socket = accept(server_sock_listen, NULL, NULL);
4      if (client_socket == -1) {
5          perror("accept");
6          continue;
7      }
8
9      printf("Client connected\n");
10
11     /* 创建子进程处理客户端连接 */
12     pid_t pid = fork();
13     if (pid < 0) {
14         perror("fork");
15         close(client_socket);
16         continue;
17     } else if (pid == 0) {
18         // 子进程处理客户端连接
19         close(server_sock_listen); // 关闭在子进程中不需要的套接字
20         handle_client(client_socket);
21     } else {
22         // 父进程继续监听下一个连接
23         close(client_socket); // 父进程不需要处理该连接，关闭套接字
24     }
25 }
```

完整代码见附录：代码清单中的代码 14。

5.3 并发服务器的运行情况

运行并发服务器，结果如图 6 所示。四个客户端都能分别发送消息，服务器端也能正常接收。

图 6: 并发的服务器



5.4 父进程分支中是否需要关闭 Socket

从结论上来说, 父进程分支中必须关闭 *Socket*, 否则会造成系统资源的浪费。

5.4.1 理论分析

为了分析这个问题, 首先要理解 *fork* 的功能。*fork* 的作用是创建一个新的进程, 新的进程是原进程的一个副本。这个系统调用是所有类 Unix 系统进程模型的中心。因此我们可以通过分析一个相对简单的 Unix 操作系统 XV6 的 *fork* 系统调用实现来分析这个问题。

XV6 的 *fork* 系统调用实现如代码 6 所示。在 *fork* 系统调用中, 首先调用了 *allocproc* 函数来分配一个进程结构体。然后对这个进程结构体进行初始化, 包括设置进程的状态, 设置进程的父进程, 设置进程的内存空间等。此时, 父进程的整个内存内容都通过 *copyuvm* 进行了复制, 然后所有父进程打开的文件都通过 *filedup* 进行了复制。最后, 子进程 *trapframe* 的 *eax* 设置成了 1, 这样在子进程中 *fork* 的返回值就会成为 0。

代码 6: XV6 的 *fork* 系统调用实现

```

1
2 int
3 fork(void)
4 {
5     int i, pid;
6     struct proc *np;
7
8     // Allocate process.
9     if((np = allocproc()) == 0)
10         return -1;
11
12     // Copy process state from p.
13     if((np->pgdir = copyuvm(proc->pgdir, proc->sz)) == 0){
14         kfree(np->kstack);
15         np->kstack = 0;
16         np->state = UNUSED;

```

```
17     return -1;
18 }
19 np->sz = proc->sz;
20 np->parent = proc;
21 *np->tf = *proc->tf;
22 np->mode = proc->mode;
23 np->parentfds = proc->parentfds;
24
25 // Clear %eax so that fork returns 0 in the child.
26 np->tf->eax = 0;
27
28 for(i = 0; i < NOFILE; i++)
29     if(proc->ofile[i])
30         np->ofile[i] = filedup(proc->ofile[i]);
31 np->cwd = idup(proc->cwd);
32 np->thr = proc->thr;
33
34 pid = np->pid;
35 np->state = RUNNABLE;
36 safestrcpy(np->name, proc->name, sizeof(proc->name));
37 return pid;
38 }
```

由于 Socket API 的设计遵从 UNIX “一切皆文件”的设计理念, 因此 Socket 调用也是基于文件系统的, 打开一个 Socket 就会创建一个文件描述符。因此, 为了确认是否需要在父进程中关闭相应的 Socket, 需要进一步探究 *fork* 系统调用中的 *filedup* 函数的实现。

filedup 函数的实现如代码 7 所示。可见, 复制文件描述符的时候, 只是将文件描述符的引用计数加 1, 而不是真正意义上的复制。

代码 7: *filedup* 函数的实现

```
1 struct file *
2 filedup(struct file *f)
3 {
4     acquire(&ftable.lock);
5     if (f->ref < 1)
6     {
7         panic("filedup");
8     }
9     f->ref++;
10    release(&ftable.lock);
11    return f;
12 }
```

而当关闭文件的时候, 系统会检查文件的引用计数, 如果引用计数为 0, 才会真正释放文件, 否则不进行任何操作。如代码 8 所示。

代码 8: *fileclose* 函数的实现

```
1 void
2 fileclose(struct file *f)
3 {
4     ...
5     if (--f->ref > 0)
6     {
```

```

7         release(&ftable.lock);
8         return;
9     }
10    ...
11 }

```

这样一来, 如果父进程中的 Socket 没有被关闭, 那么子进程中对 Socket 关闭时, 内核首先对相应文件管理结构的引用计数-1, 然后发现引用计数不为 0, 因此不会真正释放文件。这样一来, 子进程中的 Socket 就会一直保持打开状态, 没有被关闭。因此, 父进程中如果不关闭该 Socket, 就会对系统资源产生浪费, 因为这个 Socket 将永远不会得到关闭。

由于类 UNIX 操作系统的 *fork* 都有相同的功能, 所以这一分析可以很好地推广到常见的 Linux 内核中。当 *fork* 发生时, 相应的描述符会发生逻辑上的复制, 所以如果父进程不关闭相应的 Socket, 就会造成资源的浪费。

5.4.2 实验验证

使用 netstat 工具来对是否关闭 Socket 的服务器进行信息的获取, 如图 ??。

图 7: 是否关闭

(a) 关闭				(b) 不关闭			
<pre> (base) learn@DESKTOP-F0UJ1HW [17:40:23] [~/source/socket_parallel/cmake-build-debug-ssl] \$ netstat grep 32762 tcp 0 0 localhost:32762 localhost:47310 TIME_WAIT tcp 0 0 localhost:32762 localhost:47306 TIME_WAIT tcp 0 0 localhost:32762 localhost:35824 TIME_WAIT tcp 0 0 localhost:32762 localhost:45382 TIME_WAIT </pre>				<pre> (base) learn@DESKTOP-F0UJ1HW [17:40:23] [~/source/socket_parallel/cmake-build-debug-ssl] \$ netstat grep 32761 tcp 1 0 localhost:32761 localhost:44814 CLOSE_WAIT tcp 0 0 localhost:33190 localhost:32761 FIN_WAIT2 tcp 0 0 localhost:39892 localhost:32761 FIN_WAIT2 tcp 1 0 localhost:32761 localhost:43998 CLOSE_WAIT tcp 1 0 localhost:43998 localhost:39892 CLOSE_WAIT tcp 0 0 localhost:44814 localhost:32761 FIN_WAIT2 tcp 0 0 localhost:43998 localhost:32761 FIN_WAIT2 tcp 0 0 localhost:33190 localhost:32761 CLOSE_WAIT </pre>			

可以发现如果不关闭, 那么在进程结束之后也会残留 CLOSE_WAIT 状态的连接。这大大浪费了系统资源。因此, 需要在父进程中关闭相应的 Socket。

6 任务四: 获取网页内容 (Python 自学)

6.1 任务要求

编写程序, 建立与 www.people.com.cn 的 tcp 连接, 请求网页的内容并保存。

1. 将网页的内容以字符串形式保存在 txt 文件中。
2. (选做) 将网页中的图片保存到本地文件夹

6.2 Socket 爬虫实现

会出现 UTF-8 编码不兼容的问题, 如图 ??

图 8: UTF-8 编码不兼容

```

-----
UnicodeDecodeError                                Traceback (most recent call last)
e:\XMU\大三上\计网\实验三\计网实验三爬虫.ipynb 单元格 9 line 3
    27 client_socket.close()
    29 # 将响应数据以字符串形式返回, 忽略无法解码的部分
--> 30 html_content=response.decode("utf-8")
    32 # 将网页内容保存到txt文件中
    33 with open("people_com_cn.txt", "w", encoding="utf-8") as file:

UnicodeDecodeError: 'utf-8' codec can't decode byte 0xc8 in position 826: invalid

```

解决方案如代码 9 所示, 完整代码见 7 中的代码 15

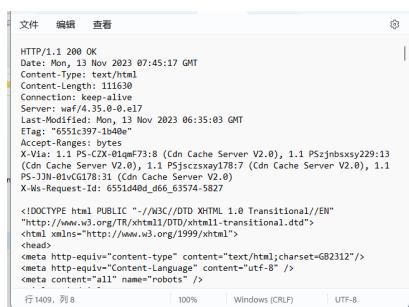
代码 9: 解决方案

```
1 # 将响应数据以字符串形式返回, 忽略无法解码的部分
2 html_content=response.decode("utf-8", errors="ignore")
3
4 # 将网页内容保存到txt文件中
5 with open("people_com_cn.txt", "w", encoding="utf-8") as file:
6     file.write(html_content)
```

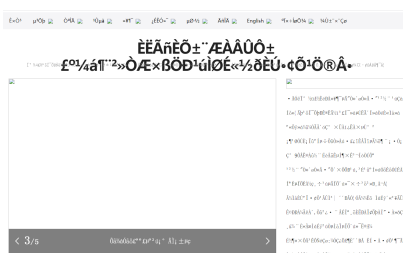
6.3 爬虫改进

使用 socket 爬取网页会偏向底层和繁琐, 且会有编码冲突的问题。将网页的 html 爬取后存为 txt 文件也不易查看。紧接着我尝试了其他两种常见爬虫框架: BeautifulSoup、Selenium。并将爬取 txt 文件取代为爬取 html 文件。完整代码见 7 板块的代码 16、代码 17

6.4 结果对比



socket



Beautifulsoup

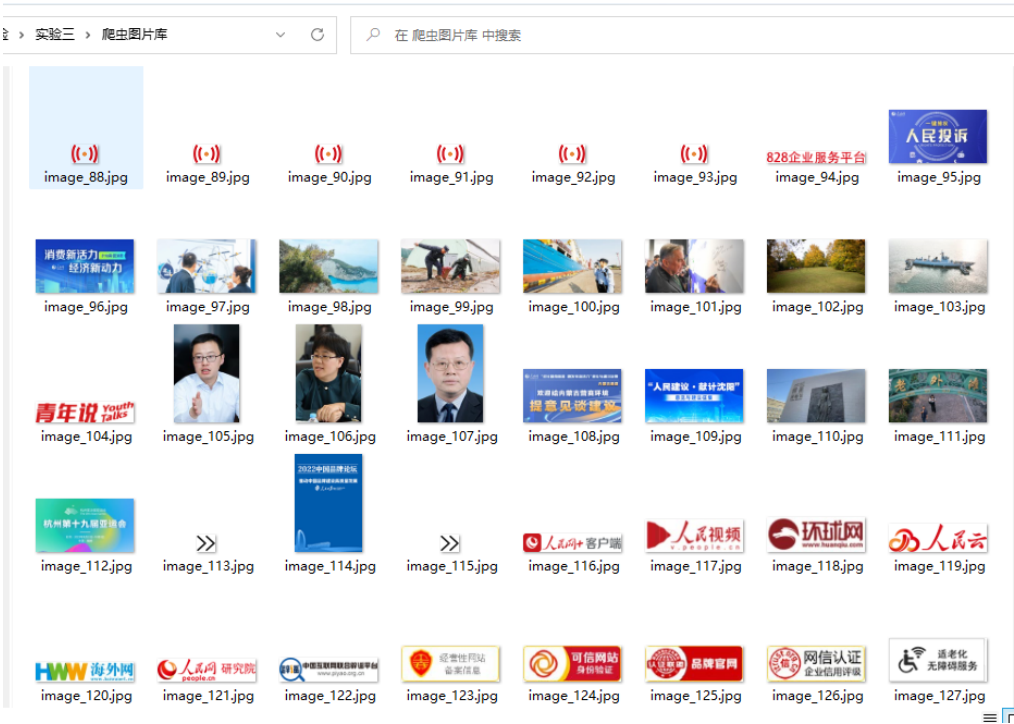


Selenium

图 9: 三种爬虫框架的结果

可以看出 selenium 的效果最佳, BeautifulSoup 虽然可以爬取 html 文件, 但是与 socket 一样存在乱码的问题。因此, 选择使用 selenium 爬取目标网页的图片并存入 images 文件夹, 结果如图 10。完整 selenium 代码见代码 17

图 10: 使用 selenium 爬取图片



7 实验总结

通过这次实验，我学习了如何调用 Socket API 通过 TCP 连接收发网络数据。并实现了迭代版和并发版的不同服务器端。在实验过程中，我学习了如何使用 netstat 工具获得所需要的信息。同时，通过理论分析和实验验证，我学到了在 fork 之后，必须关闭父进程中的 Socket 这一原则。这一原则对于防止网络程序对系统资源的浪费有很大的帮助。

附录：代码清单

代码 10: 客户机原始代码

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <sys/types.h>
4  #include <sys/socket.h>
5  #include <netinet/in.h>
6  #include <arpa/inet.h>
7  #include <unistd.h>
8  #include <stdlib.h>
9  #include <errno.h>
10
11 int main(int argc, char *argv[]) {
12     if (argc != 3) {
13         printf("Usage: %s <server_ip> <port>\n", argv[0]);
14         return 1;
15     }
16     int client_sock;
17     struct sockaddr_in server_addr;
18     char send_msg[255]; // 自定义
19     char recv_msg[255];
20
21     /* 创建socket */
22     if ((client_sock = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
23         perror("socket"); // 错误处理代码
24         return 1;
25     }
26
27     if (strcmp(argv[1], "localhost") == 0) {
28         strcpy(argv[1], "127.0.0.1");
29     }
30
31     /* 指定服务器地址 */
32     server_addr.sin_family = AF_INET;
33     server_addr.sin_port = htons(atoi(argv[2]));
34     if (inet_aton(argv[1], &server_addr.sin_addr) == 0) {
35         perror("inet_aton");
36         return 1;
37     }
38
39     memset(server_addr.sin_zero, 0, sizeof(server_addr.sin_zero)); // 零填充
40
41
42     /* 连接服务器 */
43     if (connect(client_sock, (struct sockaddr *)&server_addr, sizeof(server_addr)) == -1) {
44         perror("connect");
45         return 1;
46     }
47     while (1) {
48         printf("Enter message('bye' to quit)\n");
49         memset(send_msg, 0, sizeof(send_msg)); // 发送数组置零
50         fgets(send_msg, sizeof(send_msg), stdin);
51     }
```



```
52
53     /* 发送消息 */
54     printf("Send: %s", send_msg);
55
56     if(send(client_sock, send_msg, strlen(send_msg), 0)==-1){
57         perror("send");
58         break;
59     }
60
61     /* 接收并显示消息 */
62     memset(recv_msg, 0, sizeof(recv_msg)); //接收数组置零
63     if(recv(client_sock, recv_msg, sizeof(recv_msg), 0)==-1){
64         perror("recv");
65         break;
66     }
67     printf("Recv: %s", recv_msg);
68
69     if(strcmp(recv_msg, "bye\n")==0){
70         printf("quit---\n");
71         break;
72     }
73 }
74
75 /* 关闭socket */
76 close(client_sock);
77 return 0;
78 }
```

代码 11: 服务器

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <sys/types.h>
4  #include <sys/socket.h>
5  #include <netinet/in.h>
6  #include <arpa/inet.h>
7  #include <unistd.h>
8  #include <error.h>
9
10 int main(int argc, char *argv[])
11 {
12     int server_sock_listen, server_sock_data;
13     struct sockaddr_in server_addr;
14     char recv_msg[255];
15
16     /* 创建socket */
17     server_sock_listen = socket(AF_INET, SOCK_STREAM, 0);
18
19     /* 指定服务器地址 */
20     server_addr.sin_family = AF_INET;
21     server_addr.sin_port = htons(12345);
22     server_addr.sin_addr.s_addr = htonl(INADDR_ANY); //INADDR_ANY表示本机所有IP地址
23     memset(&server_addr.sin_zero, 0, sizeof(server_addr.sin_zero)); //零填充
24
25     /* 绑定socket与地址 */
```

```
26     bind(server_sock_listen, (struct sockaddr *)&server_addr, sizeof(server_addr));
27     /* 监听socket */
28     listen(server_sock_listen, 0);
29
30     server_sock_data = accept(server_sock_listen, NULL, NULL);
31
32     while(1){
33
34         /* 接收并显示消息 */
35         memset(recv_msg, 0, sizeof(recv_msg)); //接收数组置零
36         recv(server_sock_data, recv_msg, sizeof(recv_msg), 0);
37         printf("Recv: %s", recv_msg);
38
39         /* 发送消息 */
40         printf("Send: %s", recv_msg);
41         if(send(server_sock_data, recv_msg, strlen(recv_msg), 0)==-1){
42             perror("send");
43             break;
44         }
45
46         if(strcmp(recv_msg, "bye\n")==0)
47             break;
48
49
50
51     }
52     /* 关闭数据socket */
53     close(server_sock_data);
54     /* 关闭监听socket */
55     close(server_sock_listen);
56     return 0;
57 }
```

代码 12: 客户机

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <sys/types.h>
4  #include <sys/socket.h>
5  #include <netinet/in.h>
6  #include <arpa/inet.h>
7  #include <unistd.h>
8  #include <stdlib.h>
9  #include <errno.h>
10 int main(int argc, char *argv[]) {
11     if (argc != 3) {
12         printf("Usage: %s <server_ip> <port>\n", argv[0]);
13         return 1;
14     }
15     int client_sock;
16     struct sockaddr_in server_addr;
17     char send_msg[255]; //自定义
18     char recv_msg[255];
19
20     /* 创建socket */
```

```
21     if ((client_sock = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
22         perror("socket"); //错误处理代码
23         return 1;
24     }
25
26     if (strcmp(argv[1], "localhost") == 0) {
27         strcpy(argv[1], "127.0.0.1");
28     }
29
30     /* 指定服务器地址 */
31     server_addr.sin_family = AF_INET;
32     server_addr.sin_port = htons(atoi(argv[2]));
33     if (inet_aton(argv[1], &server_addr.sin_addr) == 0) {
34         perror("inet_aton");
35         return 1;
36     }
37
38     memset(server_addr.sin_zero, 0, sizeof(server_addr.sin_zero)); //零填充
39
40
41     /* 连接服务器 */
42     if (connect(client_sock, (struct sockaddr *)&server_addr, sizeof(server_addr)) == -1) {
43         perror("connect");
44         return 1;
45     }
46
47     while (1) {
48         //printf("请输入学生ID 课程编号\n");
49         memset(send_msg, 0, sizeof(send_msg)); //发送数组置零
50         fgets(send_msg, sizeof(send_msg), stdin);
51
52
53         /* 发送消息 */
54         printf("client: %s", send_msg);
55
56         if (send(client_sock, send_msg, strlen(send_msg), 0) == -1) {
57             perror("send");
58             break;
59         }
60         if (strcmp(send_msg, "bye\n") == 0)
61             break;
62
63         /* 接收并显示消息 */
64         memset(recv_msg, 0, sizeof(recv_msg)); //接收数组置零
65         if (recv(client_sock, recv_msg, sizeof(recv_msg), 0) == -1) {
66             perror("recv");
67             break;
68         }
69
70
71         /* 发送消息 */
72         printf("server: %s", recv_msg);
73
74         /* 误区--以下功能应该是服务器完成的 */
75         // FILE *file = fopen("课程表_utf8.txt", "r");
```

```
76 //         if (file == NULL) {
77 //             printf("Error opening file.\n");
78 //             return 1;
79 //         }
80 //
81 //
82 //
83 //         char line[256];
84 //         char *StudentID;
85 //         int courseID = 0;
86 //
87 //         char *token2 = strtok(recv_msg, ",");
88 //         if (token2 != NULL) {
89 //             StudentID = token2;
90 //             token2 = strtok(NULL, ",");
91 //             if (token2 != NULL) {
92 //                 courseID = atoi(token2);
93 //             }
94 //         }
95 //         //printf("StudentID:%s courseID:%d\n", StudentID, courseID);
96 //
97 //         while (fgets(line, sizeof(line), file)) {
98 //             char *token = strtok(line, " "); //获取学号
99 //             //printf("token:%d StudentID:%d\n", atoi(token), atoi(StudentID));
100 //             if (atoi(token) == atoi(StudentID)) {
101 //                 if (courseID == 0) {
102 //                     printf("server: ");
103 //                     while (token != NULL) {
104 //                         printf("%s ", token);
105 //                         token = strtok(NULL, " ");
106 //                     }
107 //                     printf("\n");
108 //                     break;
109 //                 } else {
110 //                     token = strtok(NULL, " ");
111 //                     char *StudentName = token;
112 //
113 //                     for (int i = 0; i < courseID && token != NULL; i++)
114 //                         token = strtok(NULL, " ");
115 //                     if (token == NULL) {
116 //                         printf("server: 不存在!\n");
117 //                     } else {
118 //                         char *courseName = token;
119 //
120 //                         printf("server: %s %s %s\n", StudentID, StudentName,
121 //                             courseName);
122 //                     }
123 //                     break;
124 //                 }
125 //             }
126 //         }
127 //         fclose(file);
128 //     }
129 //     /* 关闭socket */
130     close(client_sock);
```

```
130
131     return 0;
132 }
```

代码 13: 服务器 (迭代)

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <sys/types.h>
4  #include <sys/socket.h>
5  #include <netinet/in.h>
6  #include <arpa/inet.h>
7  #include <unistd.h>
8  #include <error.h>
9
10 int main(int argc, char *argv[]) {
11     int server_sock_listen, server_sock_data;
12     struct sockaddr_in server_addr;
13     char recv_msg[255];
14     char send_msg[255];
15     /* 创建socket */
16     server_sock_listen = socket(AF_INET, SOCK_STREAM, 0);
17
18     /* 指定服务器地址 */
19     server_addr.sin_family = AF_INET;
20     server_addr.sin_port = htons(12345);
21     server_addr.sin_addr.s_addr = htonl(INADDR_ANY); //INADDR_ANY表示本机所有IP地址
22     memset(&server_addr.sin_zero, 0, sizeof(server_addr.sin_zero)); //零填充
23
24     /* 绑定socket与地址 */
25     bind(server_sock_listen, (struct sockaddr *)&server_addr, sizeof(server_addr));
26     /* 监听socket */
27     listen(server_sock_listen, 0);
28
29     server_sock_data = accept(server_sock_listen, NULL, NULL);
30     //不断接受客户机发送来的信息,直到接受到bye
31     while (1) {
32         printf("Server is listening...\n");
33
34         /* 接收并显示消息 */
35         memset(recv_msg, 0, sizeof(recv_msg)); //接收数组置零
36         recv(server_sock_data, recv_msg, sizeof(recv_msg), 0);
37         printf("client: %s", recv_msg);
38
39         if (strcmp(recv_msg, "bye\n") == 0) //若接收到bye则退出服务器
40             break;
41
42
43         FILE *file = fopen("课程表_utf8.txt", "r");
44         if (file == NULL) {
45             printf("Error opening file.\n");
46             return 1;
47         }
48
49     }
```

```
50     char line[256];
51     char *StudentID;
52     int courseID = 0;
53
54     //使用逗号分隔接收到的字符串
55     char *token2 = strtok(recv_msg, ",");
56     if (token2 != NULL) {
57         StudentID = token2;
58         token2 = strtok(NULL, ",");
59         if (token2 != NULL) {
60             courseID = atoi(token2);
61         }
62     }
63     //printf("StudentID:%s courseID:%d\n", StudentID, courseID);
64
65     //按行读取txt文件
66     while (fgets(line, sizeof(line), file)) {
67
68         memset(send_msg, 0, sizeof(send_msg)); //接收数组置零
69
70         char *token = strtok(line, " "); //获取学号
71
72
73         //若txt中某行学号与客户机输入的一致
74         if (atoi(token) == atoi(StudentID)) {
75             //若没有输入课程编号--整行输出
76             if (courseID == 0) {
77                 printf("server: ");
78                 while (token != NULL) {
79                     printf("%s ", token);
80                     strcat(send_msg, token);
81                     strcat(send_msg, " ");
82                     token = strtok(NULL, " ");
83                 }
84                 printf("\n");
85                 strcat(send_msg, "\n");
86             } else { //若有输入课程编号--定位到对应位置
87                 token = strtok(NULL, " ");
88                 char *StudentName = token;
89
90                 for (int i = 0; i < courseID && token != NULL; i++)
91                     token = strtok(NULL, " ");
92
93                 //若token查找不到, 则不存在
94                 if (token == NULL) {
95                     printf("server: 不存在!\n");
96                     strcat(send_msg, "不存在!\n");
97                 } else {
98                     char *courseName = token;
99                     printf("server: %s %s %s\n", StudentID, StudentName,
100                        courseName);
101                     strcat(send_msg, StudentID);
102                     strcat(send_msg, " ");
103                     strcat(send_msg, StudentName);
104                     strcat(send_msg, " ");
```

```
104         strcat(send_msg, courseName);
105         strcat(send_msg, "\n");
106     }
107 }
108 //          /* 向客户机发送消息 */
109 printf("%s", send_msg);
110 if (send(server_sock_data, send_msg, strlen(send_msg), 0) == -1) {
111     printf("send");
112     break;
113 }
114 break;
115 }
116
117 }
118 fclose(file);
119
120 }
121 /* 关闭数据socket */
122 close(server_sock_data);
123 /* 关闭监听socket */
124 close(server_sock_listen);
125 return 0;
126 }
```

代码 14: 服务器 (并发)

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <sys/types.h>
4  #include <sys/socket.h>
5  #include <netinet/in.h>
6  #include <arpa/inet.h>
7  #include <unistd.h>
8  #include <error.h>
9  #include <stdlib.h>
10
11 void handle_client(int client_socket) {
12     char recv_msg[255];
13     char send_msg[255];
14
15     while (1) {
16         memset(recv_msg, 0, sizeof(recv_msg)); //接收数组置零
17         recv(client_socket, recv_msg, sizeof(recv_msg), 0);
18         printf("client: %s", recv_msg);
19
20         if (strcmp(recv_msg, "bye\n") == 0) { //若接受到bye则退出服务器
21             close(client_socket);
22             exit(EXIT_SUCCESS);
23         }
24
25         char line[256];
26         char *StudentID;
27         int courseID = 0;
28
29     }
```

```
30     char *token2 = strtok(recv_msg, ",");
31     if (token2 != NULL) {
32         StudentID = token2;
33         token2 = strtok(NULL, ",");
34         if (token2 != NULL) {
35             courseID = atoi(token2);
36         }
37     }
38     //printf("StudentID: %s,courseID: %d\n", StudentID, courseID);
39
40
41     FILE *file = fopen("课程表_utf8.txt", "r");
42
43     if (file == NULL) {
44         printf("Error opening file.\n");
45         close(client_socket);
46         exit(EXIT_FAILURE);
47     }
48
49
50     while (fgets(line, sizeof(line), file)) {
51         memset(send_msg, 0, sizeof(send_msg)); //接收数组置零
52
53         char *token = strtok(line, " ");
54         //printf("StudentID: %d,token: %d\n", atoi(StudentID), atoi(token));
55         if (atoi(token) == atoi(StudentID)) {
56             if (courseID == 0) {
57                 printf("server: ");
58                 while (token != NULL) {
59                     printf("%s ", token);
60                     strcat(send_msg, token);
61                     strcat(send_msg, " ");
62                     token = strtok(NULL, " ");
63                 }
64                 printf("\n");
65                 strcat(send_msg, "\n");
66                 if (send(client_socket, send_msg, strlen(send_msg), 0) == -1) {
67                     perror("send");
68                     break;
69                 }
70                 break;
71             } else {
72                 token = strtok(NULL, " ");
73                 char *StudentName = token;
74
75                 for (int i = 0; i < courseID && token != NULL; i++)
76                     token = strtok(NULL, " ");
77                 if (token == NULL) {
78                     printf("server: 不存在!\n");
79                     strcat(send_msg, "不存在!\n");
80                 } else {
81                     char *courseName = token;
82                     printf("server: %s %s %s\n", StudentID, StudentName,
83                             courseName);
84                     strcat(send_msg, StudentID);
```



```
84         strcat(send_msg, " ");
85         strcat(send_msg, StudentName);
86         strcat(send_msg, " ");
87         strcat(send_msg, courseName);
88         strcat(send_msg, "\n");
89     }
90     if (send(client_socket, send_msg, strlen(send_msg), 0) == -1) {
91         perror("send");
92         break;
93     }
94     break;
95 }
96 }
97 }
98 }
99 }
100
101 int main(int argc, char *argv[]) {
102     int server_sock_listen, client_socket;
103     struct sockaddr_in server_addr;
104
105     /* 创建socket */
106     server_sock_listen = socket(AF_INET, SOCK_STREAM, 0);
107
108     /* 指定服务器地址 */
109     server_addr.sin_family = AF_INET;
110     server_addr.sin_port = htons(12345);
111     server_addr.sin_addr.s_addr = htonl(INADDR_ANY); //INADDR_ANY表示本机所有IP地址
112     memset(&server_addr.sin_zero, 0, sizeof(server_addr.sin_zero)); //零填充
113
114     /* 绑定socket与地址 */
115     bind(server_sock_listen, (struct sockaddr *)&server_addr, sizeof(server_addr));
116     /* 监听socket */
117     listen(server_sock_listen, 5);
118
119
120     while (1) {
121         /* 接受客户端连接 */
122         client_socket = accept(server_sock_listen, NULL, NULL);
123         if (client_socket == -1) {
124             perror("accept");
125             continue;
126         }
127
128         printf("Client connected\n");
129
130         /* 创建子进程处理客户端连接 */
131         pid_t pid = fork();
132         if (pid < 0) {
133             perror("fork");
134             close(client_socket);
135             continue;
136         } else if (pid == 0) {
137             // 子进程处理客户端连接
138             close(server_sock_listen); // 关闭在子进程中不需要的套接字
```

```
139         handle_client(client_socket);
140     } else {
141         // 父进程继续监听下一个连接
142         close(client_socket); // 父进程不需要处理该连接, 关闭套接字
143     }
144 }
145
146 /* 关闭监听socket (在实际应用中通常不会执行到这里) */
147 close(server_sock_listen);
148 return 0;
149 }
```

代码 15: socket 爬取网页内容

```
1 import socket
2 import os
3 import re
4 from urllib.parse import urlparse
5
6 # 定义要连接的目标URL和端口号
7 target_host = "www.people.com.cn"
8 target_port = 80
9
10 # 建立TCP连接
11 client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
12 client_socket.connect((target_host, target_port))
13
14 # 发送GET请求
15 request = f"GET / HTTP/1.1\r\nHost: {target_host}\r\n\r\n"
16 client_socket.send(request.encode())
17
18 # 接收服务器响应数据
19 response = b""
20 while True:
21     data = client_socket.recv(4096)
22     if not data:
23         break
24     response += data
25
26 # 关闭连接
27 client_socket.close()
28
29 # 将响应数据以字符串形式返回, 忽略无法解码的部分
30 html_content=response.decode("utf-8", errors="ignore")
31
32 # 将网页内容保存到txt文件中
33 with open("people_com_cn.txt", "w", encoding="utf-8") as file:
34     file.write(html_content)
35
36 print("网页内容已保存到people_com_cn.txt文件中。")
```

代码 16: BeautifulSoup 爬取网页 html

```
1 import sys
2 import numpy as np
3 import json
```

```
4 import re
5 import requests
6 import pandas as pd
7 import datetime
8 from bs4 import BeautifulSoup
9 import base64
10 import os
11 import random
12 import time
13
14 user_agents = ['Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome
    /39.0.2171.95 Safari/537.36',
15                'Mozilla/5.0 (Windows NT 6.1; WOW64; rv:34.0) Gecko/20100101 Firefox/34.0',
16                'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/534.57.2 (KHTML, like Gecko) Version
    /5.1.7 Safari/534.57.2',
17                'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
    Chrome/91.0.4472.114 Safari/537.36',
18                'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.1 (KHTML, like Gecko) Chrome
    /21.0.1180.71 Safari/537.1 LBBROWSER',
19                'Mozilla/5.0 (Windows NT 5.1) AppleWebKit/535.11 (KHTML, like Gecko) Chrome
    /17.0.963.84 Safari/535.11 SE 2.X MetaSr 1.0'
20            ]
21 cookie={'cookie':'bid=kRRUP5Adrsc; _ga=GA1.3.1583431493.1679359048; _gid=GA1.3.240421151.1679359048; _ga=
    GA1.1.1583431493.1679359048; page_style="mobile"; dbc12="215291240:+lGgZ069L0g"; _pk_ses.100001.a7dd
    ==*; ck=AT7V; _ga_RXNMP372GL=GS1.1.1679406549.4.1.1679408190.60.0.0; _pk_id.100001.a7dd=0
    f38c905a23f4f70.1679359049.4.1679408190.1679402067.; _gat=1'}
22
23 headers = {
24     'User-Agent': random.choice(user_agents)
25 }
26 url='http://www.people.com.cn/'
27 res = requests.get(url,headers=headers,cookies=cookie)
28 sp = BeautifulSoup(res.text,'lxml')
29 # 获取网页的HTML内容
30 html_content = sp.prettify()
31
32 # 将HTML内容保存到txt文件
33 with open("people_com_cn.html", "w", encoding="utf-8") as file:
34     file.write(html_content)
35
36 print("网页内容已保存到people_com_cn.html文件中。")
```

代码 17: Selenium 爬取网页以及图片

```
1 from selenium import webdriver
2 from selenium.webdriver.chrome.options import Options
3 import requests
4 import os
5 import re
6 from selenium.webdriver.common.keys import Keys
7 from selenium.webdriver.common.by import By
8 from selenium.webdriver.support.ui import WebDriverWait
9 from selenium.webdriver.support import expected_conditions as EC
10 from selenium.webdriver.common.action_chains import ActionChains
11 import json
```

```
12 import random
13 import time
14 import win32con
15 import win32api
16 import win32gui
17 # 随机延迟1到3秒
18 delay = random.uniform(1, 3)
19
20 user_agent = "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/605.1.15 (KHTML, like Gecko)
    Version/12.0.3 Safari/605.1.15"
21 # Options参数指定
22 chrome_options = Options()
23 # chrome_options.add_argument('--headless')
24 chrome_options.add_argument('window-size=1920x1080')
25 # chrome_options.add_argument('disable-gpu') # 在无头模式下禁用GPU加速
26 chrome_options.add_argument('--user-agent=%s' % user_agent)
27 chrome_options.add_argument('--save-page-as-mhtml') # 启用保存为MHTML的选项
28
29 # 添加options对象, 剩下和普通selenium基本一致了
30 driver = webdriver.Chrome(executable_path="C:\\Program Files\\Google\\Chrome\\Application\\chromedriver.
    exe", options=chrome_options)
31 start_url="http://www.people.com.cn/"
32 print(start_url)
33 driver.get(start_url)
34
35 '''
36 爬取html
37 '''
38 driver.refresh()
39 #res = driver.execute_cdp_cmd('Page.captureSnapshot', {})
40 time.sleep(delay)
41 win32api.keybd_event(17, 0, 0, 0) # 按下ctrl
42 win32api.keybd_event(83, 0, 0, 0) # 按下s
43 win32api.keybd_event(83, 0, win32con.KEYEVENTF_KEYUP, 0) # 释放s
44 win32api.keybd_event(17, 0, win32con.KEYEVENTF_KEYUP, 0) # 释放ctrl
45 # hd = win32gui.FindWindow(u"#32770", u"另存为")
46 # win32gui.SetForegroundWindow(hd)
47 time.sleep(delay)
48 win32api.keybd_event(13, 0, 0, 0) # 按下enter
49 win32api.keybd_event(13, 0, win32con.KEYEVENTF_KEYUP, 0) # 释放enter
50 time.sleep(delay)
51
52 '''
53 爬取全部图片
54 '''
55 try:
56     # 定位所有图片元素
57     wait = WebDriverWait(driver, 10) # 最大等待时间为10秒
58     img_elements = wait.until(EC.presence_of_all_elements_located((By.TAG_NAME, 'img')))
59 except Exception as e:
60     print("发生异常:", e)
61
62 output_folder="E:\\XMU\\大三上\\计网\\实验\\实验三\\爬虫图片库"
63 # 创建文件夹 (如果不存在的话)
64 os.makedirs(output_folder, exist_ok=True)
```

```
65
66 # 遍历所有图片元素，下载并保存图片
67 for index, img_element in enumerate(img_elements):
68     img_url = img_element.get_attribute('src')
69     img_name = f'image_{index}.jpg' # 可以根据需要修改图片保存的文件名规则
70     img_path = os.path.join(output_folder, img_name)
71     with open(img_path, 'wb') as img_file:
72         img_content = requests.get(img_url).content
73         img_file.write(img_content)
74 driver.close()
```