# 实验 1 Web 服务器初步实现

# 1 compile

- gcc webserver.c -o webserver

# 2 run

- 网页运行：


- webserver.log:

old.log

```
INFO: request:GET / HTTP/1.1**Host: 127.0.0.1:8000**Connection: keep-alive**Cache-Control: max-age=0**sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="96", "Google Chrome";v="96"**sec-ch-ua-mobile: ?0**sec-ch-ua-platform: "Linux"**Upgrade-Insecure-Requests: 1**User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36**Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9**Sec-Fetch-Site: none**Sec-Fetch-Mode: navigate**Sec-Fetch-User: ?1**Sec-Fetch-Dest: document**Accept-Encoding: gzip, deflate, br**Accept-Language: en-CN,en;q=0.9,zh-CN;q=0.8,zh;q=0.7,en-GB;q=0.6,en-US;q=0.5,zh-TW;q=0.4**Cookie: io=anvUJbzL5HBrROpiAAAB****:1$
 INFO: SEND:index.html:1$
 INFO: Header:HTTP/1.1 200 OK$
Server: nweb/23.0$
Content-Length:297$
Connection:close$
Content-Type: text/html$
$
:1$
 INFO: request:GET /lufei.jpg HTTP/1.1**Host: 127.0.0.1:8000**Connection: keep-alive**sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="96", "Google Chrome";v="96"**sec-ch-ua-mobile: ?0**User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36**sec-ch-ua-platform: "Linux"**Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8**Sec-Fetch-Site: same-origin**Sec-Fetch-Mode: no-cors**Sec-Fetch-Dest: image**Referer: http://127.0.0.1:8000/**Accept-Encoding: gzip, deflate, br**Accept-Language: en-CN,en;q=0.9,zh-CN;q=0.8,zh;q=0.7,en-GB;q=0.6,en-US;q=0.5,zh-TW;q=0.4**Cookie: io=anvUJbzL5HBrROpiAAAB****:2$
 INFO: SEND:lufei.jpg:2$
 INFO: Header:HTTP/1.1 200 OK$
Server: nweb/23.0$
Content-Length:129231$
Connection:close$
Content-Type: image/jpg$
$
:2$
 INFO: request:GET /global.ico HTTP/1.1**Host: 127.0.0.1:8000**Connection: keep-alive**sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="96", "Google Chrome";v="96"**sec-ch-ua-mobile: ?0**User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36**sec-ch-ua-platform: "Linux"**Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8**Sec-Fetch-Site: same-origin**Sec-Fetch-Mode: no-cors**Sec-Fetch-Dest: image**Referer: http://127.0.0.1:8000/**Accept-Encoding: gzip, deflate, br**Accept-Language: en-CN,en;q=0.9,zh-CN;q=0.8,zh;q=0.7,en-GB;q=0.6,en-US;q=0.5,zh-TW;q=0.4**Cookie: io=anvUJbzL5HBrROpiAAAB****:3$
 INFO: SEND:global.ico:3$
 INFO: Header:HTTP/1.1 200 OK$
Server: nweb/23.0$
Content-Length:67646$
Connection:close$
Content-Type: image/ico$
$
:3$
```

# 浏览器中仅请求一次网页,而实际上 webserver 接收了很多次从浏览器发出的文件请求?

- 见 `https://developer.mozilla.org/zh-CN/docs/learn/Server-side/First_steps/Client-Server_overview`:

> 当一个HTML页面被返时，页面会被网络浏览器呈现出来。作为处理工作的一部分，浏览器会发现指向其他资源的链接（比如，一个HTML页面通常会参考Javascript和CSS页面），并且会发送独立的HTTP请求来下载这些文件。

- 发现一个好的总结:

> 1. 用户输入网址（假设是个 HTML 页面，并且是第一次访问），浏览器向服务器发出请求，服务器返回 HTML 文件;
> 2. 浏览器开始载入 HTML 代码，发现 <head> 标签内有一个 <link> 标签引用外部 CSS 文件;
> 3. 浏览器又发出 CSS 文件的请求，服务器返回这个 CSS 文件;
> 4. 浏览器继续载入 HTML 中 <body> 部分的代码，并且 CSS 文件已经拿到手了，可以开始渲染页面了;
> 5. 浏览器在代码中发现一个 < img > 标签引用了一张图片，向服务器发出请求。此时浏览器不会等到图片下载完，而是继续渲染后面的代码;
> 6. 服务器返回图片文件，由于图片占用了一定面积，影响了后面段落的排布，因此浏览器需要回过头来重新渲染这部分代码;
> /* 以上足以说明问题，下面是额外的技术，，，
> 7. 浏览器发现了一个包含一行 JavaScript 代码的 <script> 标签，赶快运行它;

```
8. JavaScript 脚本执行了这条语句，它命令浏览器隐藏掉代码中的某个 <div>
（style.display=”none”）。杯具啊，突然就少了这么一个元素，浏览器不得不重新渲染这部分代
码；
9. 终于等到了 </html> 的到来，浏览器泪流满面……
10. 等等，还没完，用户点了一下界面中的“换肤”按钮，JavaScript 让浏览器换了一下 <link> 标
签的 CSS 路径；
11. 浏览器召集了在座的各位 < div><span>< ul>< li> 们，“大伙儿收拾收拾行李，咱得重新来
过……”，浏览器向服务器请求了新的CSS文件，重新渲染页面。

作者：一只好奇的茂
链接：https://www.jianshu.com/p/b2a8f992c4d8
来源：简书
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

- 总而言之：就是浏览器扫描接到的 html 文件，发现有需要的东西，又向服务器发请求。

# 浏览器请求网页文件时,为加快 html 网页显示的速度,都采用了什么样的技术?

`https://www.runoob.com/w3cnote/speed-up-your-website.html`

- 一、 网站程序中采用DIV+CSS这种模式，不用Table 二、 采用Gzip技术对网页进行压缩 三、 减少CSS文件数量（合并）和体积
- 四、使用CDN加速 Content Delivery Network，解释为内容分发网络。 原理思路是尽可能避开互联网上有可能影响数据传输速度和稳定性的瓶颈和环节，使内容传输的更快、更稳定。
- 五、优化代码，减少臃肿结构 六、减少图片大小和数量 七、减少JavaScript脚本文件，尽量存放在一个文件中 八、运用静态的HTML页面
- 在 Google Chrome 中有:

```
- JavaScript: Chrome 使用的 JavaScript 虚拟机，即 V8 JavaScript 引擎，具有动态代码生成、
隐藏类转换和精确垃圾收集等功能。

- HTML排版引擎
在Android团队的建议下，“Google Chrome”使用WebKit引擎。WebKit简易小巧，并能有效率的运用存储
器，符合Google理念，并且对新开发者来说相当容易上手。但从28.0起，Google以Webkit引擎为基础，为
Chrome开发全新的Blink引擎，将比现行的Webkit引擎更简化程序源始码，并对多进程有更好的支持。

- DNS预先截取
DNS预先截取是指“域名系统”预先截取。当浏览网页时，“Google Chrome”可查询或预先截取网页上所有链
接的IP地址。浏览器使用IP地址来加载网页，因此只要事先查询这些资料，当按下网页上的任何链接时，便可
更快速地加载目标网页。

- GPU硬件加速
当激活GPU硬件加速时，使用“Google Chrome”浏览那些含有大量图片之网站时可以更快渲染完成并使页面滚
动时不会出现影像破裂的问题。

- Accelerated Mobile Pages（简称AMP，意为“加速移动页面”）是Google带领开发的开源项目，目的
是为提升移动设备对网站的访问速度。AMP也可指其派生的标准和库等项目成果。AMP在HTML等广泛使用的网
络技术基础上进行改良，它的核心称作AMP HTML，是HTML的一种。

--- from wikipedia
```

# 3 修改 webserver.c 文件中 logger 函数源代码,使得日志文件中每行信息的起始部分均有时间信息,以表示这行信息被写入的时间

- 源码见附件，log 如下：

webserver.log

```
Current local time and data: Sat Nov 20 22:08:26 2021$
$
 INFO: request:GET / HTTP/1.1**Host: 127.0.0.1:8000**Connection: keep-alive**Cache-Control: max-
: " Not A;Brand";v="99", "Chromium";v="96", "Google Chrome";v="96"**sec-ch-ua-mobile: ?0**sec-ch
inux"**Upgrade-Insecure-Requests: 1**User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537
 Gecko) Chrome/96.0.4664.45 Safari/537.36**Accept: text/html,application/xhtml+xml,application/x
vif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9**Sec-Fetch-Site: none
: navigate**Sec-Fetch-User: ?1**Sec-Fetch-Dest: document**Accept-Encoding: gzip, deflate, br**Ac
n-CN,en;q=0.9,zh-CN;q=0.8,zh;q=0.7,en-GB;q=0.6,en-US;q=0.5,zh-TW;q=0.4**Cookie: io=Z9VBW8p8Lg2Br
$
Current local time and data: Sat Nov 20 22:08:26 2021$
$
 INFO: SEND:index.html:1$
$
Current local time and data: Sat Nov 20 22:08:26 2021$
$
 INFO: Header:HTTP/1.1 200 OK$
Server: nweb/23.0$
Content-Length:297$
Connection:close$
Content-Type: text/html$
$
:1$
$
Current local time and data: Sat Nov 20 22:08:27 2021$
$
 INFO: request:GET /lufei.jpg HTTP/1.1**Host: 127.0.0.1:8000**Connection: keep-alive**sec-ch-ua:
v="99", "Chromium";v="96", "Google Chrome";v="96"**sec-ch-ua-mobile: ?0**User-Agent: Mozilla/5.0
_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36**sec-ch-ua-platfor
pt: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8**Sec-Fetch-Site: same-origi
e: no-cors**Sec-Fetch-Dest: image**Referer: http://127.0.0.1:8000/**Accept-Encoding: gzip, defla
anguage: en-CN,en;q=0.9,zh-CN;q=0.8,zh;q=0.7,en-GB;q=0.6,en-US;q=0.5,zh-TW;q=0.4**Cookie: io=Z9V
A****:2$
$
Current local time and data: Sat Nov 20 22:08:27 2021$
$
 INFO: SEND:lufei.jpg:2$
$
Current local time and data: Sat Nov 20 22:08:27 2021$
$
 INFO: Header:HTTP/1.1 200 OK$
Server: nweb/23.0$
Content-Length:129231$
Connection:close$
Content-Type: image/jpg$
$
:2$
$
Current local time and data: Sat Nov 20 22:08:28 2021$
```

# 4 在浏览器中多次快速点击刷新按钮后,为什么浏览器要隔很长一段时间才开始显示页面?请结合日志文件中的信息来分析具体原因

- 在我电脑上运行，直接终止进程：

```
➜ ~/Learing/Cod/C/democ/O/design  master ⇡10 !3 ?4 )
[1]  + 595387 broken pipe  ./webserver 8000 ./HTML
```

- 不单是快速刷新有这个问题，开了别的东西也会导致终止
- 可能是进程终止了，log 啥信息也没有，还是得搜。。。结论：
- 由于页面刷新，致使以前的链接断开，客户端在服务器返回前，就已经断开链接，服务器端写 response 报错，
- client端用户在杀死进程时，接口的 TCP 请求尚未完成，导致 server 端 write 数据时，收到 SIGPIPE 信号，抛出 Broken pipe异常。

# 5 use tool to analysis

使用 http_load 工具对此 webserver 程序进行性能测试,并记录其返回的各种参数数据。
同时在服务器端,使用 vmstat、iostat 和 iotop 等工具收集 webserver 运行时系统的各种数据,
并对 webserver 进行分析,结合它的代码说明其对系统所带来的各种消耗。

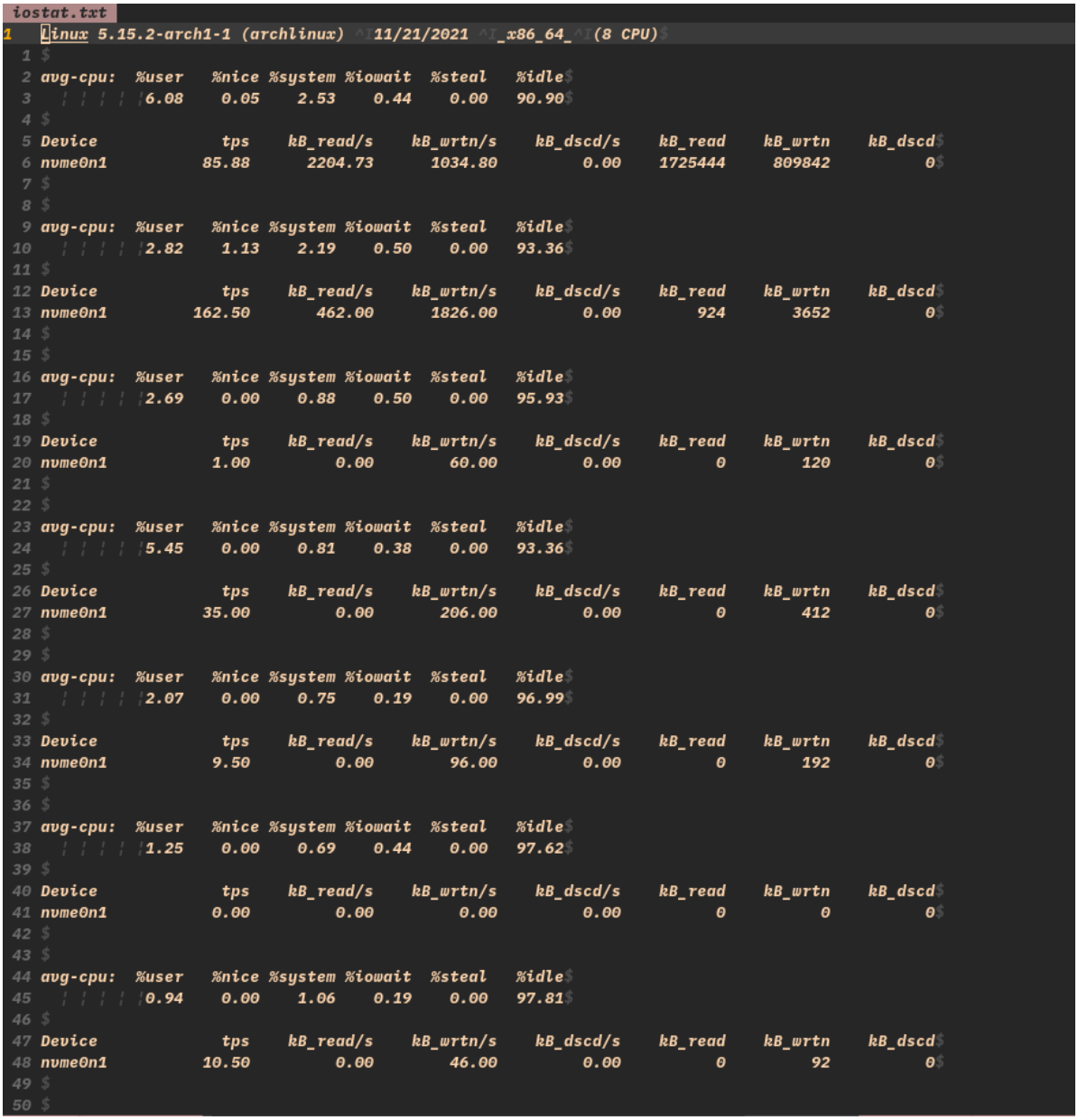
- http_load:

```
❯ ./webserver 8000 ./HTML &
[1] 6861
❯ ./http_load -parallel 5 -fetches 40 -seconds 20 http
19 fetches, 5 max parallel, 5643 bytes, in 20.0016 seconds
297 mean bytes/connection
0.949926 fetches/sec, 282.128 bytes/sec
msecs/connect: 0.261632 mean, 0.41 max, 0.061 min
msecs/first-response: 3476.53 mean, 4003.54 max, 0.605 min
HTTP response codes:
  code 200 -- 19
[1]  + 6861 broken pipe  ./webserver 8000 ./HTML
```

- vmstat

```
❯ vmstat 2 10
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0      0 1946444 146184 2980544    0    0   278   130  244  567  6  3 91  0  0
 0  0      0 1946900 146200 2980684    0    0     0    96 1306 2469  7  2 90  0  0
 2  0      0 1943448 146200 2984148    0    0     0   334  710 2071  2  1 96  0  0
 0  0      0 1936712 146200 2990192    0    0   462  1780  840 2254  4  2 94  0  0
 1  0      0 1932500 146208 2987252    0    0     0    60  859 1854  3  1 96  1  0
 0  0      0 1934064 146208 2986788    0    0     0   206  977 1211  5  1 94  0  0
 0  0      0 1933812 146216 2986852    0    0     0    96  495 1142  2  1 97  0  0
 1  0      0 1933284 146216 2987880    0    0     0    46  763 1636  1  1 97  1  0
 2  0      0 1929004 146216 2988908    0    0     0     0  401  808  1  1 98  0  0
 0  0      0 1928764 146224 2989932    0    0     0    28  405  878  1  1 98  0  0
```

- iostat



```
iostat.txt
1  Linux 5.15.2-arch1-1 (archlinux)  11/21/2021  _x86_64_  (8 CPU)
 1 $
 2 avg-cpu:  %user   %nice %system %iowait  %steal   %idle$
 3            6.08    0.05    2.53    0.44    0.00   90.90$
 4 $
 5 Device             tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd$
 6 nvme0n1          85.88      2204.73      1034.80         0.00    1725444     809842          0$
 7 $
 8 $
 9 avg-cpu:  %user   %nice %system %iowait  %steal   %idle$
10            2.82    1.13    2.19    0.50    0.00   93.36$
11 $
12 Device             tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd$
13 nvme0n1         162.50       462.00      1826.00         0.00        924       3652          0$
14 $
15 $
16 avg-cpu:  %user   %nice %system %iowait  %steal   %idle$
17            2.69    0.00    0.88    0.50    0.00   95.93$
18 $
19 Device             tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd$
20 nvme0n1            1.00         0.00        60.00         0.00          0        120          0$
21 $
22 $
23 avg-cpu:  %user   %nice %system %iowait  %steal   %idle$
24            5.45    0.00    0.81    0.38    0.00   93.36$
25 $
26 Device             tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd$
27 nvme0n1           35.00         0.00       206.00         0.00          0        412          0$
28 $
29 $
30 avg-cpu:  %user   %nice %system %iowait  %steal   %idle$
31            2.07    0.00    0.75    0.19    0.00   96.99$
32 $
33 Device             tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd$
34 nvme0n1            9.50         0.00        96.00         0.00          0        192          0$
35 $
36 $
37 avg-cpu:  %user   %nice %system %iowait  %steal   %idle$
38            1.25    0.00    0.69    0.44    0.00   97.62$
39 $
40 Device             tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd$
41 nvme0n1            0.00         0.00         0.00         0.00          0          0          0$
42 $
43 $
44 avg-cpu:  %user   %nice %system %iowait  %steal   %idle$
45            0.94    0.00    1.06    0.19    0.00   97.81$
46 $
47 Device             tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd$
48 nvme0n1           10.50         0.00        46.00         0.00          0         92          0$
49 $
50 $
```

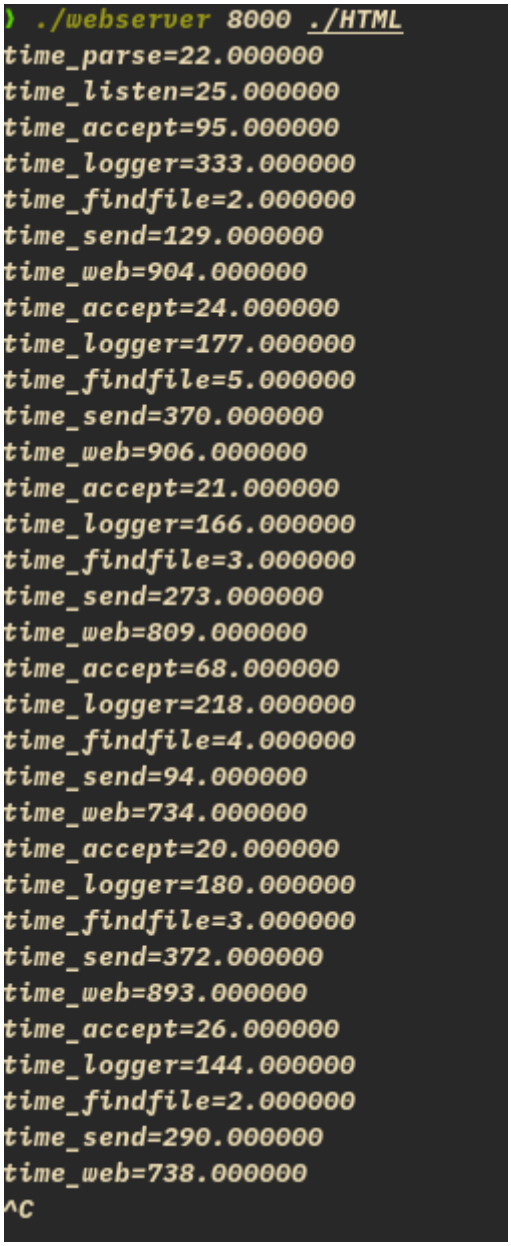
- iotop

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 1961 liukang+  20   0 3039236 210956 140784 S   3.0   2.7   0:12.90 krunner
 1195 liukang+  20   0 2402044 221076 116736 S   2.6   2.8   0:51.93 kwin_x11
  445 root      20   0   10288   6688   5152 S   1.7   0.1   0:16.60 EasyMonitor
  951 root      20   0  595108 152392 119500 S   1.7   1.9   0:44.17 Xorg
  460 root      20   0  333700  10796   8940 S   1.3   0.1   0:13.79 ECAgent
 1305 liukang+  20   0  852052 133912 111516 S   1.3   1.7   0:29.32 yakuake
 1256 liukang+  20   0 2508572 269872 160592 S   1.0   3.4   0:28.35 plasmashell
 1433 liukang+  20   0  249596  33292  28648 S   1.0   0.4   0:10.14 ksystemstats
   60 root      20   0       0      0      0 S   0.3   0.0   0:00.17 ksoftirqd/6
  140 root      20   0       0      0      0 I   0.3   0.0   0:01.81 kworker/4:2-events
  147 root      20   0       0      0      0 I   0.3   0.0   0:00.31 kworker/u16:7-flush-259:0
  213 root       0 -20       0      0      0 I   0.3   0.0   0:00.03 kworker/1:1H-events_highpri
 1299 liukang+  20   0  287700  42428  36316 S   0.3   0.5   0:03.48 sogoupinyinServ
 1760 liukang+  20   0  163488  22496  20388 S   0.3   0.3   0:01.30 kio_http_cache_
 3581 liukang+  20   0   24.4g 207028 108468 S   0.3   2.6   0:23.05 chrome
 6191 root       0 -20       0      0      0 D   0.3   0.0   0:00.58 kworker/u17:0+i915_flip
    1 root      20   0  166112  11120   8256 S   0.0   0.1   0:01.29 systemd
    2 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kthreadd
    3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
    4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
    6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0H-events_highpri
    7 root      20   0       0      0      0 I   0.0   0.0   0:00.75 kworker/u16:0-flush-259:0
    8 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq
    9 root      20   0       0      0      0 I   0.0   0.0   0:00.24 kworker/u16:1-flush-259:0
   10 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_tasks_kthre
   11 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_tasks_rude_
   12 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_tasks_trace
   13 root      20   0       0      0      0 S   0.0   0.0   0:00.18 ksoftirqd/0
   14 root      -2   0       0      0      0 I   0.0   0.0   0:00.71 rcu_preempt
   15 root      -2   0       0      0      0 S   0.0   0.0   0:00.00 rcub/0
   16 root      -2   0       0      0      0 S   0.0   0.0   0:00.00 rcuc/0
   17 root      rt   0       0      0      0 S   0.0   0.0   0:00.00 migration/0
   18 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject/0
   19 root      20   0       0      0      0 I   0.0   0.0   0:00.03 kworker/0:1-cgroup_destroy
   20 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/0
   21 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/1
   22 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject/1
   23 root      rt   0       0      0      0 S   0.0   0.0   0:00.07 migration/1
   24 root      -2   0       0      0      0 S   0.0   0.0   0:00.00 rcuc/1
   25 root      20   0       0      0      0 S   0.0   0.0   0:00.22 ksoftirqd/1
   27 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/1:0H-kblockd
   28 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/2
   29 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject/2
   30 root      rt   0       0      0      0 S   0.0   0.0   0:00.07 migration/2
   31 root      -2   0       0      0      0 S   0.0   0.0   0:00.00 rcuc/2
   32 root      20   0       0      0      0 S   0.0   0.0   0:00.17 ksoftirqd/2
   33 root      20   0       0      0      0 I   0.0   0.0   0:00.00 kworker/2:0-events
```

- http_load中：一秒大概获取 0.9 个网页，每次连接平均传输的数据量 297 bytes，而建立连接时间较短，为 0.26，但网页请求时间就比较大，为 3476，故每个连接都等待了很长时间才得到服务器的响应信息。
- vmstat 中任务数量为 1、2，小于 cpu（8） 数，而 sy 表示系统进程所占用 CPU 的时间百分比；很小，id 表示 CPU 空闲时间的百分比，很大，可以看出 cpu 占用小；
- bo 表示 I/O 块设备(block output)每秒发送的数据量(对应从 I/O 设备读数据)，从 iotop 中也可看出 I/O 的消耗
- in 表示每秒 CPU 中断次数，看出进程中断时间较大

# 6 在 server.c 中增加相关计时函数,分析一下程序的哪个部分最耗时?使用 perf 工具来跟踪 webserver 程序,根据其运行报告进行程序性能分析,请指出 webserver 中比较 耗费时间的函数有哪些?

- 刷新了两次页面，每次请求需要三次 accept：




> 看出 I\O 函数是最耗时的，listen 与 accept 都耗时较小
>
> > 主要是写入 log 与 send 文件耗时，图中看出发送一个图片用了 370，而 logger 也占比较大

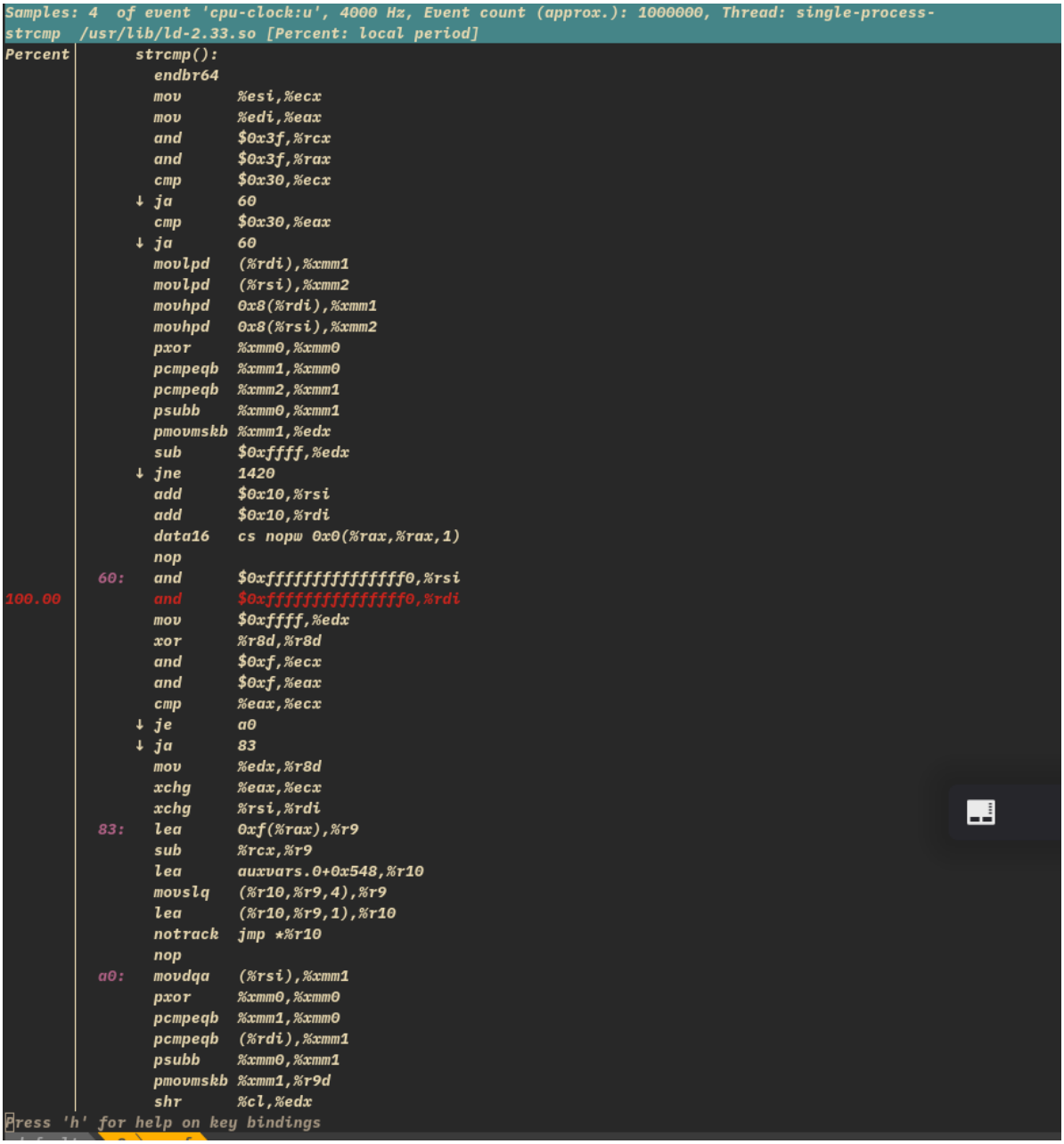
- perf_start:

```
(2/2) Refreshing Package DB...
❯ perf stat ./single-process-server 8000 ./HTML
./single-process-server: Broken pipe

 Performance counter stats for './single-process-server 8000 ./HTML':

             10.57 msec task-clock:u              #    0.000 CPUs utilized
                 0      context-switches:u        #    0.000 /sec
                 0      cpu-migrations:u          #    0.000 /sec
                61      page-faults:u             #    5.772 K/sec
           860,523      cycles:u                  #    0.081 GHz
           328,593      instructions:u            #    0.38  insn per cycle
            64,347      branches:u                #    6.089 M/sec
             6,673      branch-misses:u           #   10.37% of all branches

      21.408291972 seconds time elapsed

       0.002641000 seconds user
       0.007917000 seconds sys
```

- perf_report

```
Samples: 4  of event 'cpu-clock:u', Event count (approx.): 1000000, Thread: single-process-
  Children      Self  Command          Shared Object      Symbol
+   25.00%    25.00%  single-process-  libc-2.33.so       [.] __strlen_avx2
+   25.00%     0.00%  single-process-  [unknown]          [.] 0x00007ffdc5fa9df0
+   25.00%    25.00%  single-process-  ld-2.33.so         [.] strcmp
+   25.00%    25.00%  single-process-  libc-2.33.so       [.] __nanosleep
+   25.00%    25.00%  single-process-  libc-2.33.so       [.] clock_nanosleep@@GLIBC_2.17
```

- annotate 详情：（全是汇编语言）




> 函数耗时， strlen 与 strcmp 耗时长。

# 7 根据题目 5 和题目 6 的结论,能否指出 webserver 性能低下的原因?并给出相应的解决方法?

- 主要问题是写入 log 与传输文件的问题，而函数 strlen 与 strcmp 效率低下
- 解决办法（参考第二题的加速技术）：

1. 可以减少 log 的写入，少些废话，精炼提取，规定格式；
2. 对于大文件的传输，采取压缩技术，合并技术
3. 写高质量的代码，提高速度（难）