

题目1

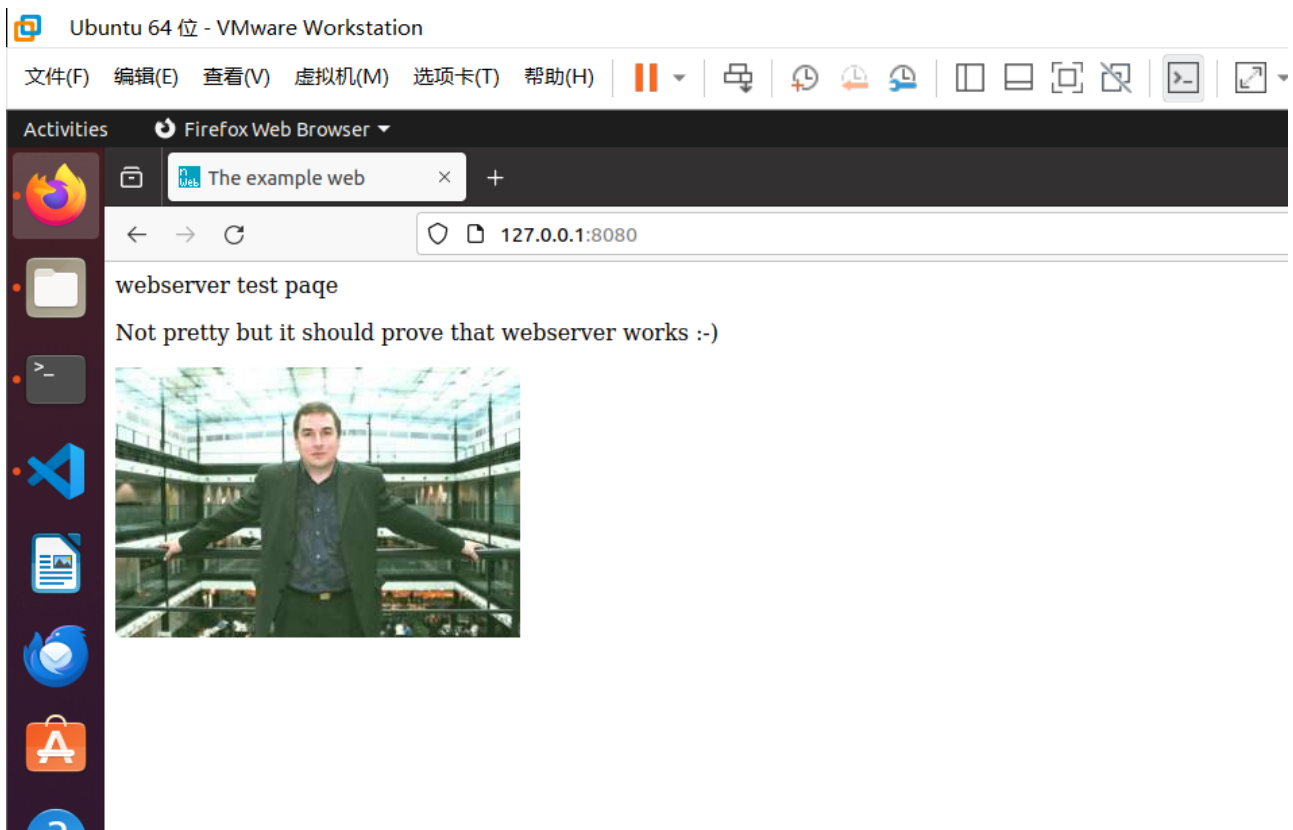
创建 *makefile* 文件，将 *webserver.c* 代码进行编译为 *webserver* 可执行程序。

```
1  # Makefile for webserver.c
2
3  CC = gcc
4  CFLAGS = -Wall
5  TARGET = nweb
6  SOURCE = webserver.c
7  OBJS = $(SOURCE:.c=.o)
8
9  all: $(TARGET)
10
11  $(TARGET): $(OBJS)
12      $(CC) $(CFLAGS) -o $(TARGET) $(OBJS)
13
14  $(OBJS): $(SOURCE)
15      $(CC) $(CFLAGS) -c $<
16
17  clean:
18      rm -f $(TARGET) $(OBJS)
19
```

题目2

启动webserver程序

```
./webserver 8080 ./nwebdir
```



观察日志信息

```
1  INFO: request:GET /index.html HTTP/1.1**Host:
    127.0.0.1:8080**User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux
    x86_64; rv:109.0) Gecko/20100101 Firefox/119.0**Accept:
    text/html,application/xhtml+xml,application/xml;q=0.9,image/avif
    ,image/webp,*/*;q=0.8**Accept-Language: en-US,en;q=0.5**Accept-
    Encoding: gzip, deflate, br**Connection: keep-alive**Upgrade-
    Insecure-Requests: 1**Sec-Fetch-Dest: document**Sec-Fetch-Mode:
    navigate**Sec-Fetch-Site: none**Sec-Fetch-User: ?1****:5
2  INFO: SEND:index.html:5
3  INFO: Header:HTTP/1.1 200 OK
4  Server: nweb/23.0
5  Content-Length: 260
6  Connection: close
7  Content-Type: text/html
8
9  :5
```

```
10  INFO: request:GET /example.jpg HTTP/1.1**Host:
    127.0.0.1:8080**User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux
    x86_64; rv:109.0) Gecko/20100101 Firefox/119.0**Accept:
    image/avif,image/webp,*/***Accept-Language: en-
    US,en;q=0.5**Accept-Encoding: gzip, deflate, br**Connection:
    keep-alive**Referer: http://127.0.0.1:8080/index.html**Sec-
    Fetch-Dest: image**Sec-Fetch-Mode: no-cors**Sec-Fetch-Site:
    same-origin****:6
11  INFO: SEND:example.jpg:6
12  INFO: Header:HTTP/1.1 200 OK
13  Server: nweb/23.0
14  Content-Length: 10184
15  Connection: close
16  Content-Type: image/jpg
17
18  :6
```

为什么在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：

在日志中，可以看到浏览器在加载 `/index.html` 时发出了对 `/example.jpg` 请求。这些请求是浏览器根据 `index.html` 中的资源引用自动发出的。这些额外的请求可能是因为浏览器需要加载网页中的其他资源，如图片、样式表、JavaScript文件等。这些资源通常是通过相对或绝对URL引用的，浏览器会根据这些URL发出额外的请求以获取这些资源。因此，虽然可能只在浏览器中看到一个请求，但实际上可能会有多个请求发送到web服务器，以获取网页所需的所有资源。

为加速 **HTML** 网页显示速度，浏览器采用的技术包括：

1. 并行加载：现代浏览器支持并行加载资源，这意味着它们可以同时下载多个资源，而不是等待一个资源的下载完成后再开始另一个资源的下载。这可以显著提高网页加载速度。
2. 域名分片：浏览器会将网页上的资源请求分配给不同的域名（如使用CDN，Content Delivery Network），这允许更多资源同时下载。这是因为浏览器在单个域名上有一定数量的并行连接限制，通过使用不同的域名，可以绕过这些限制。
3. 延迟加载：浏览器可以延迟加载某些资源，例如图片、广告、或其他不是立即需要的内容。这允许首次加载时尽快显示页面的关键内容，然后在后台加载其他资源。
4. 压缩：浏览器和服务器可以使用压缩算法，如Gzip或Brotli，来减小传输过程中的资源大小。这减少了网络传输时间，从而提高了加载速度。

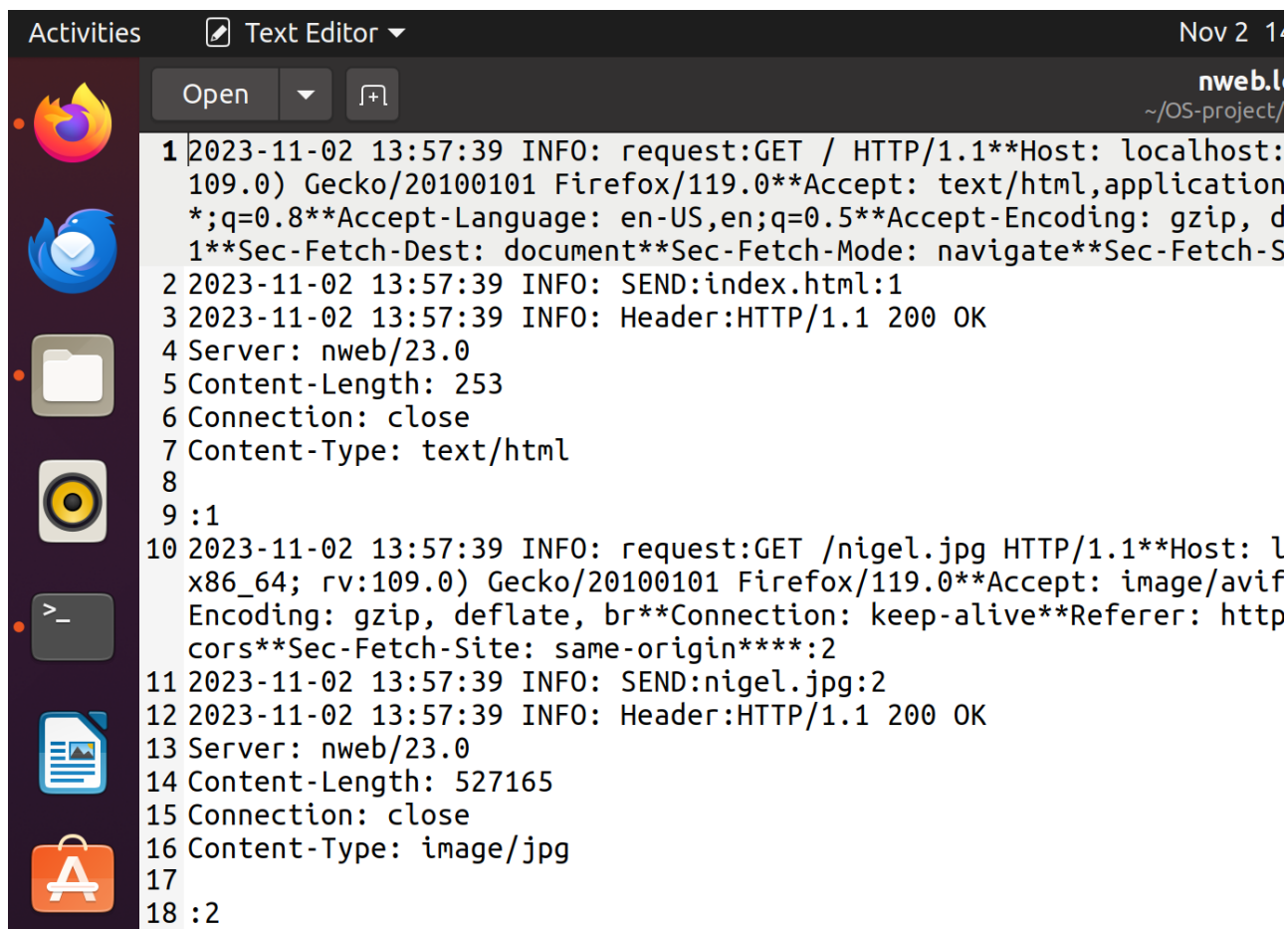
题目3

修改 logger 函数，添加时间戳

```
1 void logger(int type, char *s1, char *s2, int socket_fd)
2 {
3     int fd;
4     char logbuffer[BUFSIZE*2];
5     char timebuffer[80]; // Buffer to store time
6
7     time_t now;
8     struct tm *timeinfo;
9
10    time(&now);
11    timeinfo = localtime(&now);
12    strftime(timebuffer, sizeof(timebuffer), "%Y-%m-%d %H:%M:%S",
13    timeinfo);
14
15    switch (type) {
16        case ERROR:
17            (void)sprintf(logbuffer, "%s ERROR: %s:%s Errno=%d exiting
18            pid=%d", timebuffer, s1, s2, errno, getpid());
19            break;
20        case FORBIDDEN:
21            // ...
22        case NOTFOUND:
23            // ...
24        case LOG:
25            (void)sprintf(logbuffer, "%s INFO: %s:%s:%d", timebuffer,
26            s1, s2, socket_fd);
27            break;
28    }
29
30    if ((fd = open("nweb.log", O_CREAT | O_WRONLY | O_APPEND,
31    0644)) >= 0) {
32        (void)write(fd, logbuffer, strlen(logbuffer));
33        (void)write(fd, "\n", 1);
34        (void)close(fd);
35    }
36 }
```

在这个修改后的代码中，`strftime` 函数用于格式化当前时间，并将其存储在 `timebuffer` 中，然后在 `logbuffer` 中使用 `%s` 占位符将时间信息添加到日志信息之前，然后再将整个 `logbuffer` 写入到日志文件。这样就在日志文件的每个条目之前添加了时间戳。

Output Screenshot



```

1 2023-11-02 13:57:39 INFO: request:GET / HTTP/1.1**Host: localhost:
109.0) Gecko/20100101 Firefox/119.0**Accept: text/html,application
*;q=0.8**Accept-Language: en-US,en;q=0.5**Accept-Encoding: gzip, d
1**Sec-Fetch-Dest: document**Sec-Fetch-Mode: navigate**Sec-Fetch-S
2 2023-11-02 13:57:39 INFO: SEND:index.html:1
3 2023-11-02 13:57:39 INFO: Header:HTTP/1.1 200 OK
4 Server: nweb/23.0
5 Content-Length: 253
6 Connection: close
7 Content-Type: text/html
8
9 :1
10 2023-11-02 13:57:39 INFO: request:GET /nigel.jpg HTTP/1.1**Host: l
x86_64; rv:109.0) Gecko/20100101 Firefox/119.0**Accept: image/avif
Encoding: gzip, deflate, br**Connection: keep-alive**Referer: http
cors**Sec-Fetch-Site: same-origin***:2
11 2023-11-02 13:57:39 INFO: SEND:nigel.jpg:2
12 2023-11-02 13:57:39 INFO: Header:HTTP/1.1 200 OK
13 Server: nweb/23.0
14 Content-Length: 527165
15 Connection: close
16 Content-Type: image/jpg
17
18 :2

```

题目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。

实际出现的实验现象与预期结果有所不同。如果等待一次刷新完成再进行第二次刷新，则可以正常显示页面。当多次快速点击刷新按钮后，浏览器端：`unable to connect`；

这可能是因为服务器无法并发或高效处理多个连接。每个传入连接都在一个循环中处理，如果新连接在前一个连接完全处理之前到达，可能会导致问题。

题目5

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。

webserver 程序性能测试

http_load

```
→ http_load-12mar2006 ./http_load -parallel 10 -fetches 10 urls.txt
10 fetches, 10 max parallel, 2600 bytes, in 9.00752 seconds
260 mean bytes/connection
1.11018 fetches/sec, 288.648 bytes/sec
msecs/connect: 0.091 mean, 0.223 max, 0.016 min
msecs/first-response: 4503.75 mean, 9007.25 max, 0.174 min
HTTP response codes:
code 200 -- 10
```

webserver 运行时系统的各种数据

vmstat

```
→ http_load-12mar2006 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
2   0   1292 300944 220456 2784536    0    0    36    24   160  393  2   1  97   0   0
0   0   1292 300944 220456 2784544    0    0     0     0   152  314  0   0 100   0   0
0   0   1292 300944 220456 2784544    0    0     0     0   132  272  0   1 100   0   0
0   0   1292 300944 220464 2784548    0    0     0    20   142  303  1   0  99   0   0
0   0   1292 300944 220464 2784548    0    0     0     0   165  286  0   0 100   0   0
0   0   1292 300944 220472 2784548    0    0     0    90   191  348  0   1  99   0   0
0   0   1292 300944 220472 2784548    0    0     0     0   232  432  0   0 100   0   0
0   0   1292 302816 220472 2784564    0    0     8     0   927 1806  2   4  94   0   0
0   0   1292 302816 220480 2784564    0    0     0     8   481  928  0   1  99   0   0
0   0   1292 302816 220480 2784564    0    0     0     0   249  424  0   1  99   0   0
```

iostat

```
→ http_load-12mar2006 iostat -k 2 10
Linux 5.15.0-88-generic (ubuntu)      11/09/2023      _x86_64_      (2 CPU)
```

avg-cpu:	%user	%nice	%system	%iowait	%steal	%idle			
	1.70	0.03	1.42	0.02	0.00	96.83			
Device	tps	kB_read/s	kB_wrtn/s	kB_dscd/s	kB_read	kB_wrtn	kB_dscd		
loop0	0.01	0.03	0.00	0.00	851	0	0		
loop1	0.01	0.03	0.00	0.00	989	0	0		
loop10	0.06	0.12	0.00	0.00	3847	0	0		
loop11	0.02	0.78	0.00	0.00	24728	0	0		
loop12	0.02	0.04	0.00	0.00	1165	0	0		
loop13	0.00	0.01	0.00	0.00	367	0	0		
loop14	0.12	0.31	0.00	0.00	9875	0	0		
loop15	0.01	0.11	0.00	0.00	3567	0	0		
loop16	0.00	0.00	0.00	0.00	14	0	0		
loop2	0.00	0.00	0.00	0.00	17	0	0		
loop3	0.02	0.07	0.00	0.00	2315	0	0		
loop4	0.01	0.03	0.00	0.00	917	0	0		
loop5	0.02	0.20	0.00	0.00	6428	0	0		
loop6	0.01	0.03	0.00	0.00	928	0	0		
loop7	0.08	0.85	0.00	0.00	26724	0	0		
loop8	0.02	0.06	0.00	0.00	1949	0	0		
loop9	0.02	0.07	0.00	0.00	2244	0	0		
sda	3.83	68.04	48.00	0.00	2144238	1512853	0		
scd0	0.00	0.00	0.00	0.00	2	0	0		
scd1	0.00	0.03	0.00	0.00	1072	0	0		

```
total DISK READ:      0.00 B/s | Total DISK WRITE:      0.00 B/s
current DISK READ:    0.00 B/s | Current DISK WRITE:    10.75 K/s
```

TID	PRIO	USER	DISK READ	DISK WRITE	SWAPIN	IO>	COMMAND
1	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		init auto noprompt
2	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[kthreadd]
3	be/0	root	0.00 B/s	0.00 B/s	?unavailable?		[rcu_gp]
4	be/0	root	0.00 B/s	0.00 B/s	?unavailable?		[rcu_par_gp]
5	be/0	root	0.00 B/s	0.00 B/s	?unavailable?		[slub_flushwq]
6	be/0	root	0.00 B/s	0.00 B/s	?unavailable?		[netns]
8	be/0	root	0.00 B/s	0.00 B/s	?unavailable?		[kworker/0:0H-events_highpri]
10	be/0	root	0.00 B/s	0.00 B/s	?unavailable?		[mm_percpu_wq]
11	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[rcu_tasks_rude_]
12	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[rcu_tasks_trace]
13	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[ksoftirqd/0]
14	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[rcu_sched]
15	rt/4	root	0.00 B/s	0.00 B/s	?unavailable?		[migration/0]
16	rt/4	root	0.00 B/s	0.00 B/s	?unavailable?		[idle_inject/0]
18	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[cpuhp/0]
19	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[cpuhp/1]
20	rt/4	root	0.00 B/s	0.00 B/s	?unavailable?		[idle_inject/1]
21	rt/4	root	0.00 B/s	0.00 B/s	?unavailable?		[migration/1]
22	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[ksoftirqd/1]
24	be/0	root	0.00 B/s	0.00 B/s	?unavailable?		[kworker/1:0H-events_highpri]
25	be/4	root	0.00 B/s	0.00 B/s	?unavailable?		[kdevtmpfs]
26	be/0	root	0.00 B/s	0.00 B/s	?unavailable?		[inet_frag_wq]

iotop


```
sudo iotop

Total DISK READ:      0.00 B/s | Total DISK WRITE:      17.89 K/s
Current DISK READ:    0.00 B/s | Current DISK WRITE:    42.93 K/s

  TID  PRIO  USER      DISK READ  DISK WRITE  SWAPIN     IO>   COMMAND
  323  be/3  root       0.00 B/s   14.31 K/s   ?unavailable? [jbd2/sda5-8]
15806 be/4  hatsunem  0.00 B/s   3.58 K/s   ?unavailable? ./webserver 8080 ./nwebdir
    1  be/4  root       0.00 B/s   0.00 B/s   ?unavailable? init auto noprompt
    2  be/4  root       0.00 B/s   0.00 B/s   ?unavailable? [kthreadd]
    3  be/0  root       0.00 B/s   0.00 B/s   ?unavailable? [rcu_gp]
    4  be/0  root       0.00 B/s   0.00 B/s   ?unavailable? [rcu_par_gp]
    5  be/0  root       0.00 B/s   0.00 B/s   ?unavailable? [slub_flushwq]
    6  be/0  root       0.00 B/s   0.00 B/s   ?unavailable? [netns]
    8  be/0  root       0.00 B/s   0.00 B/s   ?unavailable? [kworker/0:0-ents_highpri]
   10  be/0  root       0.00 B/s   0.00 B/s   ?unavailable? [mm_percpu_wq]
   11  be/4  root       0.00 B/s   0.00 B/s   ?unavailable? [rcu_tasks_rude_]
   12  be/4  root       0.00 B/s   0.00 B/s   ?unavailable? [rcu_tasks_trace]
   13  be/4  root       0.00 B/s   0.00 B/s   ?unavailable? [ksoftirqd/0]
   14  be/4  root       0.00 B/s   0.00 B/s   ?unavailable? [rcu_sched]
   15  rt/4  root       0.00 B/s   0.00 B/s   ?unavailable? [migration/0]
   16  rt/4  root       0.00 B/s   0.00 B/s   ?unavailable? [idle_inject/0]
   18  be/4  root       0.00 B/s   0.00 B/s   ?unavailable? [cpuhp/0]
   19  be/4  root       0.00 B/s   0.00 B/s   ?unavailable? [cpuhp/1]
   20  rt/4  root       0.00 B/s   0.00 B/s   ?unavailable? [idle_inject/1]

keys:  any: refresh  q: quit  i: ionice  o: active  p: procs  a: accum
sort:  r: asc  left: SWAPIN  right: COMMAND  home: TID  end: COMMAND
CONFIG_TASK_DELAY_ACCT not enabled in kernel, cannot determine SWAPIN and IO %
```

webserver资源消耗分析

1. CPU 资源：

- 服务器主循环(main函数中的无限for循环)不断地接受客户端连接，并调用web函数处理每个连接。这可能导致CPU消耗较高，特别是在有大量并发连接的情况下。
- 文件的读取和写入操作，特别是对大文件的处理，也可能占用一定的CPU资源。

2. 内存 资源：

- 每个连接都需要一定的内存空间来存储请求和响应数据。这些数据主要存储在buffer数组中。
- 文件的读取操作可能需要额外的内存缓冲区。

3. 磁盘 I/O 资源：

- 文件的读取是通过open和read等系统调用完成的。这可能导致对磁盘I/O的需求，尤其是在频繁读取大文件时。
- 服务器日志的写入也可能引起文件I/O操作。

4. 网络 I/O 资源：

- 通过socket进行网络通信，接收和发送数据。这会导致网络I/O消耗，尤其是在高并发情况下。

题目 6

在 `servers` 中增加相关计时函数

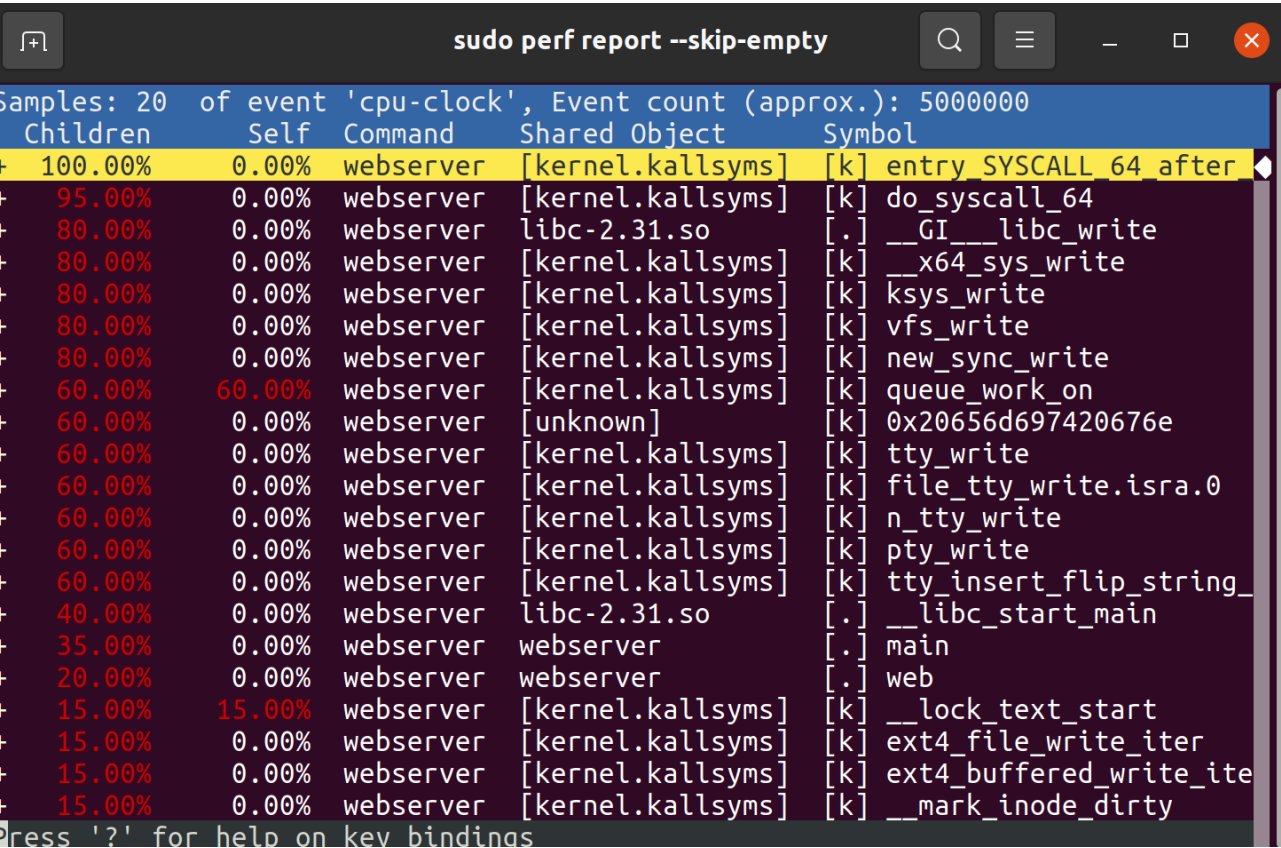
```
1 #include <time.h>
2 // ... (existing code)
3 // Function to get the current time in microseconds
4 long long current_timestamp() {
5     struct timespec ts;
6     clock_gettime(CLOCK_MONOTONIC, &ts);
7     return ts.tv_sec * 1000000LL + ts.tv_nsec / 1000;
8 }
9 // ... (existing code)
10 void web(int fd, int hit) {
11     // Record the start time
12     long long start_time = current_timestamp();
13     // ... (existing code)
14     // Record the end time
15     long long end_time = current_timestamp();
16     // Calculate and print the elapsed time
17     printf("Processing time for hit %d: %lld microseconds\n",
18         hit, end_time - start_time);
19 }
```



```
./webserver 8080 ./nwebdir
OS-project ./webserver 8080 ./nwebdir
Processing time for hit 1: 1000586 microseconds
Processing time for hit 2: 1000588 microseconds
Processing time for hit 3: 1000643 microseconds
Processing time for hit 4: 1001050 microseconds
Processing time for hit 5: 1000846 microseconds
Processing time for hit 6: 1000698 microseconds
Processing time for hit 7: 1000856 microseconds
Processing time for hit 8: 1001197 microseconds
```

分析程序

perf运行报告



根据提供的性能报告，以下是在 `webserver` 中耗费时间较多的函数：

1. `entry_SYSCALL_64_after_` (100.00%)
 - 该函数属于内核，与系统调用处理有关。
2. `do_syscall_64` (95.00%)
 - 另一个与系统调用处理相关的内核函数。
3. `__GI___libc_write` (80.00%)
 - 这是GNU C库（glibc）中用于写入数据的函数。
4. `__x64_sys_write`, `ksys_write`, `vfs_write`, `new_sync_write` (80.00%)
 - 这些是与写操作相关的内核函数。
5. `queue_work_on` (60.00%)
 - 与任务调度相关的内核函数。
6. `tty_write`, `file_tty_write.isra.0`, `n_tty_write`, `pty_write`, `tty_insert_flip_string_` (60.00%)
 - 这些是与终端（tty）和文件写入相关的内核函数。
7. `__libc_start_main` (40.00%)

- C程序的入口点，属于libc。

8. `main` (35.00%)

- `webserver` 应用程序的主函数。

9. `web` (20.00%)

- `webserver` 应用程序中的另一个函数。

这些函数涵盖了大部分执行时间，其中包括内核级别的系统调用处理、写入操作、任务调度以及应用程序级别的主要入口点和功能。

题目7

本次实验提供的代码是一个简单的Web服务器实现，但存在一些导致性能低下的问题。以下是一些存在的性能瓶颈和建议的解决方法：

1. 同步I/O：

- 代码使用同步I/O操作（`read`和`write`）处理客户端请求和提供文件。这意味着服务器一次只能处理一个请求。
- 解决方法：考虑使用异步I/O或多线程/多进程方法以同时处理多个请求。这可以提高服务器的响应性和总体吞吐量。

2. 阻塞文件读取：

- 使用阻塞I/O操作进行文件读取，可能导致服务器在读取大文件时等待。
- 解决方法：使用非阻塞I/O或采用异步I/O进行文件读取。这样，在等待文件I/O完成时，服务器可以继续处理其他请求。

3. 没有内容缓存：

- 服务器对于每个请求都从磁盘读取整个文件而没有缓存。
- 解决方法：实现缓存机制，将频繁请求的文件存储在内存中，减少对重复请求的磁盘I/O。