一旦将文件映射到内存，就不再需要它的描述符了，所以我们关闭这个文件(第 20 行)。执行这项任务失败将导致潜在的致命的内存泄漏。第 21 行执行的是到客户端的实际文件传送。rio_writen 函数复制从 srcp 位置开始的 filesize 个字节(它们当然已经被映射到了所请求的文件)到客户端的已连接描述符。最后，第 22 行释放了映射的虚拟内存区域。这对于避免潜在的致命的内存泄漏是很重要的。

7. serve_dynamic 函数

TINY 通过派生一个子进程并在子进程的上下文中运行一个 CGI 程序，来提供各种类型的动态内容。

图 11-35 中的 serve_dynamic 函数一开始就向客户端发送一个表明成功的响应行，同时还包括带有信息的 Server 报头。CGI 程序负责发送响应的剩余部分。注意，这并不像我们可能希望的那样健壮，因为它没有考虑到 CGI 程序会遇到某些错误的可能性。

code/netp/tiny/tiny.c

```
1  void serve_dynamic(int fd, char *filename, char *cgiargs)
2  {
3      char buf[MAXLINE], *emptylist[] = { NULL };
4
5      /* Return first part of HTTP response */
6      sprintf(buf, "HTTP/1.0 200 OK\r\n");
7      Rio_writen(fd, buf, strlen(buf));
8      sprintf(buf, "Server: Tiny Web Server\r\n");
9      Rio_writen(fd, buf, strlen(buf));
10
11     if (Fork() == 0) { /* Child */
12         /* Real server would set all CGI vars here */
13         setenv("QUERY_STRING", cgiargs, 1);
14         Dup2(fd, STDOUT_FILENO);         /* Redirect stdout to client */
15         Execve(filename, emptylist, environ); /* Run CGI program */
16     }
17     Wait(NULL); /* Parent waits for and reaps child */
18 }
```

code/netp/tiny/tiny.c

图 11-35　TINY serve_dynamic 为客户端提供动态内容

在发送了响应的第一部分后，我们会派生一个新的子进程(第 11 行)。子进程用来自请求 URI 的 CGI 参数初始化 QUERY _ STRING 环境变量(第 13 行)。注意，一个真正的服务器还会在此处设置其他的 CGI 环境变量。为了简短，我们省略了这一步。

接下来，子进程重定向它的标准输出到已连接文件描述符(第 14 行)，然后加载并运行 CGI 程序(第 15 行)。因为 CGI 程序运行在子进程的上下文中，它能够访问所有在调用 execve 函数之前就存在的打开文件和环境变量。因此，CGI 程序写到标准输出上的任何东西都将直接送到客户端进程，不会受到任何来自父进程的干涉。其间，父进程阻塞在对 wait 的调用中，等待当子进程终止的时候，回收操作系统分配给子进程的资源(第 17 行)。

旁注　处理过早关闭的连接

尽管一个 Web 服务器的基本功能非常简单，但是我们不想给你一个假象，以为编写一个实际的 Web 服务器是非常简单的。构造一个长时间运行而不崩溃的健壮的 Web 服务器是一件困难的任务，比起在这里我们已经学习了的内容，它要求对 Linux 系统编程有更加深入的