Problem 1:

Complete the following table:

Hex address	Dotted-decimal address
0x0	0.0.0.0
0xFFFFFF00	255.255.255.0
0x7F000001	127.0.0.1
0xCA780265	202.120.2.101
0x08080808	8.8.8.8
0x0AFFF9FE	10.255.249.254

Problem 2:

In Section 10.9, we warned you about the dangers of using the C standard I/O functions in network applications. Yet the CGI program in Figure 11.26 is able to use standard I/O without any problems. Why?

The reason that standard I/O works in CGI programs is that the CGI program running in the child process does not need to explicitly close any of its input or output streams. When the child terminates, the kernel closes all descriptors automatically.

Problem 3:

Modify Tiny so that it reaps CGI children inside a SIGCHLD handler instead of explicitly waiting for them to terminate.

Note that you only need to show what have been added, deleted or updated.

Add a function:

```
Void handler(int sig)
{
  pid_t pid;
  while((pid = waitpid(-1,NULL,0)) > 0);

  if (!((pid == 0) || (pid == -1 && errno == ECHILD)))
  unix_error("sigchld_handler wait error");
}
```

Add in main():

signal(SIGCHLD, handler);

Delete in serve_dynamic():

wait(NULL);

Problem 4:

Modify Tiny so that when it serves static content, it copies the requested file to the connected descriptor using malloc, rio_readn, and rio_writen, instead of mmap and rio_writen.

Note that you only need to show what have been added, deleted or updated.

Update this piece of code in serve_static():

```
srcfd = Open(filename, O_RDONLY, 0);
srcp = Mmap(0, filesize, PROT_READ, MAP_PRIVATE,
srcfd, 0);
Close(srcfd);
Rio_writen(fd, srcp, filesize);
Munmap(srcp, filesize);
```

To:

```
srcfd = Open(filename, O_RDONLY, 0);
srcp = (char*)malloc(filesize);
Rio_readn(srcfd, srcp, filesize);
Rio_writen(fd, srcp, filesize);
close(srcfd);
free(srcp);
```