附录 A

APPENDIX A

错误处理

程序员应该总是检查系统级函数返回的错误代码。有许多细微的方式会导致出现错误，只有使用内核能够提供给我们的状态信息才能理解为什么有这样的错误。不幸的是，程序员往往不愿意进行错误检查，因为这使他们的代码变得很庞大，将一行代码变成一个多行的条件语句。错误检查也是很令人迷惑的，因为不同的函数以不同的方式表示错误。

在编写本书时，我们面临类似的问题。一方面，我们希望代码示例阅读起来简洁简单；另一方面，我们又不希望给学生们一个错误的印象，以为可以省略错误检查。为了解决这些问题，我们采用了一种基于*错误处理包装函数*(error-handling wrapper)的方法，这是由 W. Richard Stevens 在他的网络编程教材[110]中最先提出的。

其思想是，给定某个基本的系统级函数 foo，我们定义一个有相同参数、只不过开头字母大写了的包装函数 Foo。包装函数调用基本函数并检查错误。如果包装函数发现了错误，那么它就打印一条信息并终止进程。否则，它返回到调用者。注意，如果没有错误，包装函数的行为与基本函数完全一样。换句话说，如果程序使用包装函数运行正确，那么我们把每个包装函数的第一个字母小写并重新编译，也能正确运行。

包装函数被封装在一个源文件(csapp.c)中，这个文件被编译和链接到每个程序中。一个独立的头文件(csapp.h)中包含这些包装函数的函数原型。

本附录给出了一个关于 Unix 系统中不同种类的错误处理的教程，还给出了不同风格的错误处理包装函数的示例。csapp.h 和 csapp.c 文件可以从 CS：APP 网站上获得。

A.1 Unix 系统中的错误处理

本书中我们遇到的系统级函数调用使用三种不同风格的返回错误：*Unix 风格的*、*Posix 风格的*和 *GAI 风格的*。

1. Unix 风格的错误处理

像 fork 和 wait 这样 Unix 早期开发出来的函数(以及一些较老的 Posix 函数)的函数返回值既包括错误代码，也包括有用的结果。例如，当 Unix 风格的 wait 函数遇到一个错误(例如没有子进程要回收)，它就返回 −1，并将全局变量 errno 设置为指明错误原因的错误代码。如果 wait 成功完成，那么它就返回有用的结果，也就是回收的子进程的 PID。Unix 风格的错误处理代码通常具有以下形式：

```
1       if ((pid = wait(NULL)) < 0) {
2           fprintf(stderr, "wait error: %s\n", strerror(errno));
3           exit(0);
4       }
```

strerror 函数返回某个 errno 值的文本描述。

2. Posix 风格的错误处理

许多较新的 Posix 函数，例如 Pthread 函数，只用返回值来表明成功(0)或者失败(非 0)。任何有用的结果都返回在通过引用传递进来的函数参数中。我们称这种方法为 Posix