** 8.20　使用 execve 编写一个叫做 myls 的程序，该程序的行为和 /bin/ls 程序的一样。你的程序应该接受相同的命令行参数，解释同样的环境变量，并产生相同的输出。

ls 程序从 COLUMNS 环境变量中获得屏幕的宽度。如果没有设置 COLUMNS，那么 ls 会假设屏幕宽 80 列。因此，你可以通过把 COLUMNS 环境设置得小于 80，来检查你对环境变量的处理：

```
linux> setenv COLUMNS 40
linux> ./myls
  .
  .  // Output is 40 columns wide
  .
linux> unsetenv COLUMNS
linux> ./myls
  .
  .  // Output is now 80 columns wide
  .
```

** 8.21　下面的程序可能的输出序列是什么？

code/ecf/waitprob3.c

```
1   int main()
2   {
3       if (fork() == 0) {
4           printf("a"); fflush(stdout);
5           exit(0);
6       }
7       else {
8           printf("b"); fflush(stdout);
9           waitpid(-1, NULL, 0);
10      }
11      printf("c"); fflush(stdout);
12      exit(0);
13  }
```

code/ecf/waitprob3.c

** 8.22　编写 Unix system 函数的你自己的版本

```
int mysystem(char *command);
```

mysystem 函数通过调用“/bin/sh-c command”来执行 command，然后在 command 完成后返回。如果 command(通过调用 exit 函数或者执行一条 return 语句)正常退出，那么 mysystem 返回 command 退出状态。例如，如果 command 通过调用 exit(8)终止，那么 mysystem 返回值 8。否则，如果 command 是异常终止的，那么 mysystem 就返回 shell 返回的状态。

** 8.23　你的一个同事想要使用信号来让一个父进程对发生在子进程中的事件计数。其想法是每次发生一个事件时，通过向父进程发送一个信号来通知它，并且让父进程的信号处理程序对一个全局变量 counter 加一，在子进程终止之后，父进程就可以检查这个变量。然而，当他在系统上运行图 8-45 中的测试程序时，发现当父进程调用 printf 时，counter 的值总是 2，即使子进程向父进程发送了 5 个信号也是如此。他很困惑，向你寻求帮助。你能解释这个程序有什么错误吗？

code/ecf/counterprob.c

```
1   #include "csapp.h"
2
3   int counter = 0;
4
5   void handler(int sig)
6   {
7       counter++;
8       sleep(1); /* Do some work in the handler */
9       return;
```

图 8-45　家庭作业 8.23 中引用的计数器程序