

```


/* Bits in the third argument to 'waitpid'. */
#define WNOHANG    1    /* Don't block waiting. */
#define WUNTRACED  2    /* Report status of stopped children. */

```

为了使用这些常量，必须在代码中包含 `wait.h` 头文件：

```
#include <sys/wait.h>
```

每个 Unix 函数的 man 页列出了无论何时你在代码中使用那个函数都要包含的头文件。同时，为了检查诸如 `ECHILD` 和 `EINTR` 之类的返回代码，你必须包含 `errno.h`。为了简化代码示例，我们包含了一个称为 `csapp.h` 的头文件，它包括了本书中使用的所有函数的头文件。`csapp.h` 头文件可以从 CS: APP 网站在线获得。

 **练习题 8.3** 列出下面程序所有可能的输出序列：

```

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

```

code/ecf/waitprob0.c

code/ecf/waitprob0.c

5. wait 函数

`wait` 函数是 `waitpid` 函数的简单版本：

```

#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int *statusp);

```

返回：如果成功，则为子进程的 PID，如果出错，则为 -1。

调用 `wait(&status)` 等价于调用 `waitpid(-1, &status, 0)`。

6. 使用 `waitpid` 的示例

因为 `waitpid` 函数有些复杂，看几个例子会有所帮助。图 8-18 展示了一个程序，它使用 `waitpid`，不按照特定的顺序等待它的所有 N 个子进程终止。在第 11 行，父进程创建 N 个子进程，在第 12 行，每个子进程以一个唯一的退出状态退出。在我们继续讲解之前，请确认你已经理解为什么每个子进程会执行第 12 行，而父进程不会。

在第 15 行，父进程用 `waitpid` 作为 `while` 循环的测试条件，等待它所有的子进程终止。因为第一个参数是 -1，所以对 `waitpid` 的调用会阻塞，直到任意一个子进程终止。在每个子进程终止时，对 `waitpid` 的调用会返回，返回值为该子进程的非零的 PID。第 16 行检查子进程的退出状态。如果子进程是正常终止的——在此是以调用 `exit` 函数终止的——那么父进程就提取出退出状态，把它输出到 `stdout` 上。