code/ecf/shellex.c

```
1  /* parseline - Parse the command line and build the argv array */
2  int parseline(char *buf, char **argv)
3  {
4      char *delim;         /* Points to first space delimiter */
5      int argc;            /* Number of args */
6      int bg;              /* Background job? */
7
8      buf[strlen(buf)-1] = ' ';  /* Replace trailing '\n' with space */
9      while (*buf && (*buf == ' ')) /* Ignore leading spaces */
10         buf++;
11
12     /* Build the argv list */
13     argc = 0;
14     while ((delim = strchr(buf, ' '))) {
15         argv[argc++] = buf;
16         *delim = '\0';
17         buf = delim + 1;
18         while (*buf && (*buf == ' ')) /* Ignore spaces */
19                buf++;
20     }
21     argv[argc] = NULL;
22
23     if (argc == 0)  /* Ignore blank line */
24         return 1;
25
26     /* Should the job run in the background? */
27     if ((bg = (*argv[argc-1] == '&')) != 0)
28         argv[--argc] = NULL;
29
30     return bg;
31 }
```

code/ecf/shellex.c

图 8-25　parseline 解析 shell 的一个输入行

注意，这个简单的 shell 是有缺陷的，因为它并不回收它的后台子进程。修改这个缺陷就要求使用信号，我们将在下一节中讲述信号。

8.5　信号

到目前为止对异常控制流的学习中，我们已经看到了硬件和软件是如何合作以提供基本的低层异常机制的。我们也看到了操作系统如何利用异常来支持进程上下文切换的异常控制流形式。在本节中，我们将研究一种更高层的软件形式的异常，称为 Linux 信号，它允许进程和内核中断其他进程。

一个信号就是一条小消息，它通知进程系统中发生了一个某种类型的事件。比如，图 8-26 展示了 Linux 系统上支持的 30 种不同类型的信号。

每种信号类型都对应于某种系统事件。低层的硬件异常是由内核异常处理程序处理的，正常情况下，对用户进程而言是不可见的。信号提供了一种机制，通知用户进程发生了这些异常。比如，如果一个进程试图除以 0，那么内核就发送给它一个 SIGFPE 信号(号码 8)。如果一个进