含返回值。−4095 到−1 之间的负数返回值表明发生了错误，对应于负的 errno。

编号	名字	描述	编号	名字	描述
0	read	读文件	33	pause	挂起进程直到信号到达
1	write	写文件	37	alarm	调度告警信号的传送
2	open	打开文件	39	getpid	获得进程 ID
3	close	关闭文件	57	fork	创建进程
4	stat	获得文件信息	59	execve	执行一个程序
9	mmap	将内存页映射到文件	60	_exit	终止进程
12	brk	重置堆顶	61	wait4	等待一个进程终止
32	dup2	复制文件描述符	62	kill	发送信号到一个进程

图 8-10　Linux x86-64 系统中常用的系统调用示例

例如，考虑大家熟悉的 hello 程序的下面这个版本，用系统级函数 write(见 10.4 节)来写，而不是用 printf：

```
1  int main()
2  {
3      write(1, "hello, world\n", 13);
4      _exit(0);
5  }
```

write 函数的第一个参数将输出发送到 stdout。第二个参数是要写的字节序列，而第三个参数是要写的字节数。

图 8-11 给出的是 hello 程序的汇编语言版本，直接使用 syscall 指令来调用 write 和 exit 系统调用。第 9～13 行调用 write 函数。首先，第 9 行将系统调用 write 的编号存放在%rax 中，第 10～12 行设置参数列表。然后第 13 行使用 syscall 指令来调用系统调用。类似地，第 14～16 行调用_exit 系统调用。

code/ecf/hello-asm64.sa

```
1   .section .data
2   string:
3     .ascii "hello, world\n"
4   string_end:
5     .equ len, string_end - string
6   .section .text
7   .globl main
8   main:
      First, call write(1, "hello, world\n", 13)
9     movq $1, %rax          write is system call 1
10    movq $1, %rdi          Arg1: stdout has descriptor 1
11    movq $string, %rsi     Arg2: hello world string
12    movq $len, %rdx        Arg3: string length
13    syscall                Make the system call

      Next, call _exit(0)
14    movq $60, %rax         _exit is system call 60
15    movq $0, %rdi          Arg1: exit status is 0
16    syscall                Make the system call
```

code/ecf/hello-asm64.sa

图 8-11　直接用 Linux 系统调用来实现 hello 程序