

江南大学物联网工程学院实验报告

课程名称 《Linux 环境程序设计》 实验名称 实验 6 Linux 环境编程 实验日期 2017.11.29

班级 计科 1404 姓名 阎覃 学号 1030414414

实验报告要求 1. 实验目的 2. 实验内容 3. 实验步骤 4. 运行情况 5. 实验体会

1 实验目的

- 理解系统调用和库函数的异同;
- 学会用系统调用进行编程;
- 掌握一些常用的系统调用的功能及应用。

2 实验内容

- 使用系统调用对文件进行操作;
- 使用系统调用对进程进行控制;
- 使用管道机制进行 I/O;
- 使用信号机制进行进程通信。

3 实验步骤及运行情况

Problem 1 编写一个程序, 把一个文件的内容复制到另一个文件上, 即实现简单的 copy 功能。要求: 只用 open(), read(), write() 和 close() 系统调用, 程序的第一个参数是源文件, 第二个参数是目的文件。

Answer

下面是程序的源代码:

```
1 #include <iostream>
2 #include <fcntl.h>
3 #include <zconf.h>
4
5 #define BUF_SIZE 5
6
7 int main(int argc, char *argv[]) {
8
9     if (argc != 3) {
10         std::cout << "用法: copy 源文件 目标文件" << std::endl;
11         return EPERM;
12     }
13
14     int srcFile = open(argv[1], O_RDONLY); // 打开文件
15     if (srcFile == -1) {
16         perror("无法打开源文件");
17         return errno;
18     }
19
20     int dstFile = open(argv[2], O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
21     // 创建文件
22     if (dstFile == -1) {
23         perror("无法创建目标文件");
24         return errno;
25     }
26
27     char buffer[BUF_SIZE], *ptr;
28     ssize_t bytesRead, bytesWrite;
```

```

28 while ((bytesRead = read(srcFile, buffer, BUF_SIZE)) != 0) {
29     if ((bytesRead == -1) && (errno != EINTR)) {
30         perror("读文件时遇到错误");
31         return errno;
32     } else if (bytesRead > 0) {
33         ptr = buffer;
34         while ((bytesWrite = write(dstFile, ptr, (size_t) bytesRead))
35             != 0) {
36             if ((bytesWrite == -1) && (errno != EINTR)) {
37                 perror("写文件时遇到错误");
38                 return errno;
39             }
40             std::cout << "已复制" << bytesWrite << "字节" << std::
41                 endl;
42             if (bytesWrite == bytesRead) break;
43             else if (bytesWrite > 0) {
44                 ptr += bytesWrite;
45                 bytesRead -= bytesWrite;
46             }
47         }
48     }
49 }
50 std::cout << "操作完成!" << std::endl;
51
52 close(srcFile);
53 close(dstFile);
54 return 0;
55 }

```

运行截图

```

MacBook-Pro:cmake-build-debug ethan$ ./copy
用法: copy 源文件 目标文件
MacBook-Pro:cmake-build-debug ethan$ date >> a
MacBook-Pro:cmake-build-debug ethan$ ls b
ls: b: No such file or directory
MacBook-Pro:cmake-build-debug ethan$ ./copy a b
已复制5字节
已复制5字节
已复制5字节
已复制5字节
已复制5字节
已复制5字节
已复制5字节
已复制5字节
已复制5字节
已复制3字节
操作完成!
MacBook-Pro:cmake-build-debug ethan$ ls b -l
ls: -l: No such file or directory
b
MacBook-Pro:cmake-build-debug ethan$ ls -l b
-rw----- 1 ethan admin 48 11 29 08:20 b
MacBook-Pro:cmake-build-debug ethan$ cat b
2017年11月29日 星期三 08时19分50秒 CST
MacBook-Pro:cmake-build-debug ethan$

```

Problem 2 编写一个程序, 它利用 `fork()` 创建一个子进程; 父进程打开一个文件, 父子进程都向文件写入 (利用 `write()`) 信息, 表明是在哪个进程中; 每个进程都打印两个进程的 ID 号。最后父进程执行 `wait()`。另外, 如果没有 `wait` 调用, 会出现什么情况?

Answer

下面是程序的源代码:

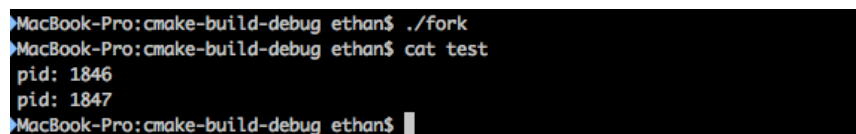
```

1  #include <iostream>
2  #include <fcntl.h>
3  #include <unistd.h>
4
5  int main() {
6
7      int file = open("test", O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR); // 创
          建文件
8      if (file == -1) {
9          perror("无法创建目标文件");
10         return errno;
11     }
12
13     pid_t pid;
14     pid = fork();
15     if (pid < 0) { // error
16         printf("Fork Failed\n");
17         return 1;
18     }
19
20     char buffer[100];
21     sprintf(buffer, "pid: %d \n", getpid());
22
23     write(file, buffer, strlen(buffer));
24
25     if (pid != 0) { // parent process
26         wait(nullptr);
27     }
28     return 0;
29 }

```

如果没有 wait 调用, 子进程将会成为僵死进程。

运行截图



```

MacBook-Pro:cmake-build-debug ethan$ ./fork
MacBook-Pro:cmake-build-debug ethan$ cat test
pid: 1846
pid: 1847
MacBook-Pro:cmake-build-debug ethan$

```

Problem 3 编写一个程序, 它创建一个子进程。父进程向子进程发送一个信号, 然后等待子进程终止; 子进程接受信号, 输出自己的状态信息, 最后终止自己。

Answer

下面是程序的源代码:

```

1  #include <iostream>
2  #include <unistd.h>
3  #include <signal.h>
4
5  void handler(int signo) {
6      std::cout << "[子进程]收到信号:" << signo << "\tpid:" << getpid() <<
          "\t父进程pid:" << getppid() << std::endl;
7      std::cout << "[子进程]子进程退出" << std::endl;
8      exit(0);
9  }
10
11 int main() {
12     int pid;

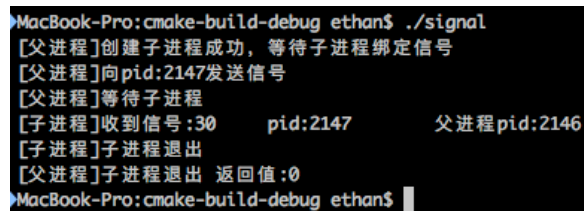
```

```

13 pid = fork();
14 if (pid < 0) {
15     std::cout << "[父进程]创建子进程失败" << std::endl;
16 } else if (pid == 0) {
17     signal(SIGUSR1, handler);
18     while (true);
19 } else {
20     std::cout << "[父进程]创建子进程成功，等待子进程绑定信号" << std
21         ::endl;
22     sleep(2);
23     std::cout << "[父进程]向pid:" << pid << "发送信号" << std::endl;
24     kill(pid, SIGUSR1);
25     std::cout << "[父进程]等待子进程" << std::endl;
26     int status;
27     wait(&status);
28     std::cout << "[父进程]子进程退出 返回值:" << status << std::endl;
29 }
30
31 return 0;
32 }

```

运行截图



```

MacBook-Pro:cmake-build-debug ethan$ ./signal
[父进程]创建子进程成功，等待子进程绑定信号
[父进程]向pid:2147发送信号
[父进程]等待子进程
[子进程]收到信号:30      pid:2147      父进程pid:2146
[子进程]子进程退出
[父进程]子进程退出 返回值:0
MacBook-Pro:cmake-build-debug ethan$

```

Problem 4 编译并运行教材 P220-221 例题 7.5，体会管道机制的应用。

Answer

下面是程序的源代码：

```

1  #include <iostream>
2  #include<unistd.h>
3
4  int main(int argc, char **argv)
5  {
6      static const char mesg[]="Happy New Years to you!";
7      char buf[BUFSIZ];
8      size_t rcount,wcount;
9      int p_fd[2];
10     size_t n;
11
12     if(pipe(p_fd)<0){
13         fprintf(stderr,"%s:pipe failed:%s\n",argv[0],strerror(errno));
14         exit(1);
15     }
16     printf("Read end=fd %d,write end=fd %d\n",p_fd[0],p_fd[1]);
17     n=strlen(mesg);
18     if((wcount=write(p_fd[1],mesg,n))!=n){
19         fprintf(stderr,"%s:write failed:%s\n",argv[0],strerror(errno));
20         exit(1);
21     }
22     if((rcount=read(p_fd[0],buf,BUFSIZ))!=wcount){
23         fprintf(stderr,"%s:read failed:%s\n",argv[0],strerror(errno));
24         exit(1);

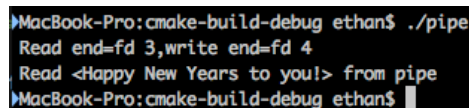
```

```

25     }
26     buf[rcount]='\0';
27     printf("Read <%s> from pipe\n",buf);
28     close(p_fd[0]);
29     close(p_fd[1]);
30     return 0;
31 }

```

运行截图



```

MacBook-Pro:cmake-build-debug ethan$ ./pipe
Read end=fd 3,write end=fd 4
Read <Happy New Years to you!> from pipe
MacBook-Pro:cmake-build-debug ethan$

```

Problem 5 掌握 Unix/Linux 环境下事件驱动编程的一般方法, 能利用 curses 库、间隔计时器 (interval timer)、信号处理编写一个功能完整的视频游戏程序, 并能在 Unix/Linux 环境下正确地运行。

Answer

本游戏是一个躲避障碍的小游戏, 用户可以按空格键操作小人, 跳跃躲避障碍物。随着游戏的进行难度会不断提升。右上角会计分, 当小人遇到障碍没有成功躲避时游戏结束。

```

1  #include <cstdio>
2  #include <curses.h>
3  #include <csignal>
4  #include <cstring>
5  #include <list>
6  #include <stdlib>
7  #include "set_ticker.c"
8  #include <random>
9
10 using std::default_random_engine;
11 using std::uniform_real_distribution;
12
13 const int ROW = 15;
14 const float P = 0.1f; // 出现障碍物的概率
15
16 default_random_engine e;
17 uniform_real_distribution<float> u(0, 1); // 随机数分布对象
18
19 struct Rock {
20     const char *MSG = "@";
21     int col = COLS;
22 };
23
24 std::list<Rock *> rocks;
25
26 struct People {
27     const int COL = 5;
28     const char *MSG = "*";
29
30     int row = ROW;
31     int dir = 0;
32 };
33
34 People people;
35
36 int score;
37 int level;
38 int delay;
39

```

```

40 void tick(int);
41
42 // 画地面
43 void drawGround() {
44     for (int i = 0; i < COLS; i++) {
45         move(ROW + 1, i);
46         addstr(" ");
47     }
48 }
49
50 // 初始化游戏
51 void initGame() {
52     clear();
53     rocks = std::list<Rock *>();
54     people.row = ROW;
55     people.dir = 0;
56     delay = 60;
57     score = 0;
58     level = 1;
59
60     drawGround(); // 画地面
61     move(people.row, people.COL); // 画人
62     addstr(people.MSG);
63     move(5, COLS - 5); // 画分数
64     addstr("0");
65     move(5, COLS - 15);
66     addstr("level 1");
67     signal(SIGALRM, tick);
68     set_ticker(delay);
69 }
70
71 void setLevel(int flag) {
72     if (flag > 0) {
73         if (level >= 9)
74             return;
75         delay -= 5;
76         level++;
77     } else {
78         if (level <= 0)
79             return;
80         delay += 5;
81         level--;
82     }
83
84     char levelStr[10];
85     sprintf(levelStr, "level %d", level);
86     move(5, COLS - 15);
87     addstr(levelStr);
88     set_ticker(delay);
89 }
90
91 int main() {
92     initscr();
93     crmode();
94     noecho();
95
96     initGame(); // 初始化游戏
97
98     while (true) {
99         int c = getch();
100         switch (c) {
101             case 'q':
102                 endwin();
103                 exit(0);
104             case 'r':
105                 for (auto i = rocks.begin(); i != rocks.end(); i++) {

```

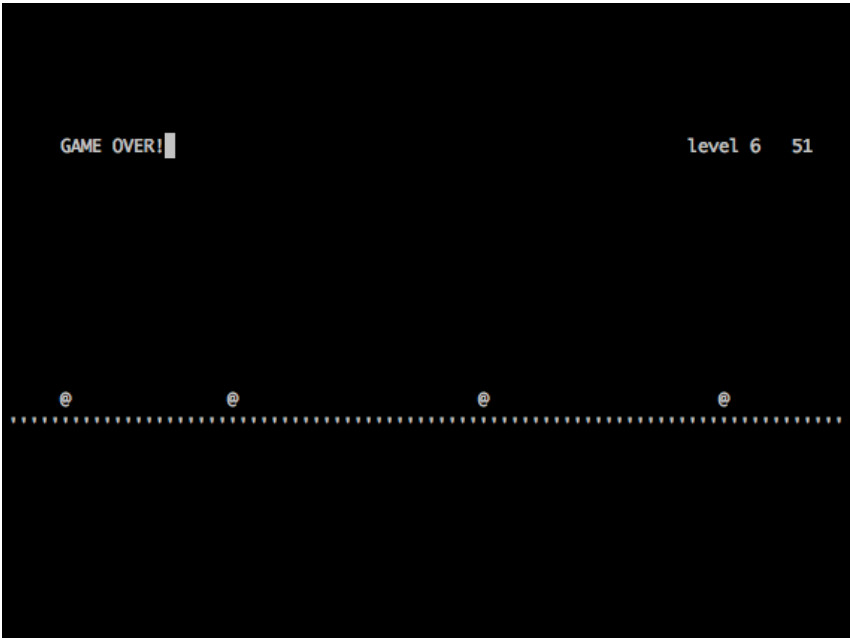
```

106         delete *i;
107     }
108     initGame();
109     break;
110     case ' ':
111         if (people.dir == 0) people.dir = -1;
112         break;
113     case '=':
114         setLevel(1);
115         break;
116     case '_':
117         setLevel(-1);
118         break;
119     default:
120         break;
121 }
122 }
123 }
124
125
126 int flag = 0;
127
128 void tick(int signum) {
129     signal(SIGALRM, tick);          /* reset, just in case */
130
131     if (people.row == ROW && people.dir == 1) // 到达地面
132         people.dir = 0;
133     else if (people.row == ROW - 4 && people.dir == -1) // 到达最高点
134         people.dir = 1;
135     else if (people.dir != 0) {
136         move(people.row, people.COL);      // 删除原来的人
137         addstr(" ");
138
139         people.row += people.dir;
140
141         move(people.row, people.COL);      // 画人
142         addstr(people.MSG);
143     }
144
145     if (--flag <= 0 && u(e) < P) {          // 根据概率创建石头 出现在屏
146         幕右边 并加入队尾
147         rocks.push_back(new Rock);
148         flag = 20 - level;
149     }
150
151     for (auto i = rocks.begin(); i != rocks.end(); i++) {
152         if ((*i)->col != 5 || people.row != ROW) {
153             move(ROW, (*i)->col);          // 清空原来的石头
154             addstr(" ");
155         }
156         (*i)->col -= 1;                    // 左移一位
157
158         if ((*i)->col < 0) {                // 一旦超出屏幕左侧 删除
159             delete *i;
160             i = rocks.erase(i);
161         } else {
162             move(ROW, (*i)->col);          // 画新石头
163             addstr((*i)->MSG);
164         }
165
166         if ((*i)->col == 5) {              // 检测小人位置是否碰撞
167             if (people.row == ROW) {      // 发生碰撞
168                 move(5, 5);
169                 addstr("GAME OVER!");
170                 set_ticker(0);            // 游戏结束
171                 refresh();

```

```
171         return;
172     } else {
173         score++; // 没有碰撞 加分
174         if (score % 10 == 9) setLevel(1);
175         char scoreStr[5];
176         sprintf(scoreStr, "%d", score);
177         move(5, COLS - 5);
178         addstr(scoreStr);
179     }
180 }
181 }
182
183 move(LINES - 1, 0);
184 refresh();
185 }
```

运行截图



4 实验体会

通过这次实验，我练习了许多 Linux 系统调用函数，同时也对一些系统调用得到了进一步的了解。包括文件操作，进程间通信，定时，信号等等，都是 Linux 中的重要概念。最后的事件驱动编程编写了一个完整的小游戏，提升了我的动手能力，并且增强了对 Linux 的兴趣。

实验报告采用 L^AT_EX 排版，代码托管至 GitHub:
<https://github.com/Ethan-yt/JNU-Linux-exp>

教师评价	优	良	中	及格	不及格	教师签名	日期
------	---	---	---	----	-----	------	----