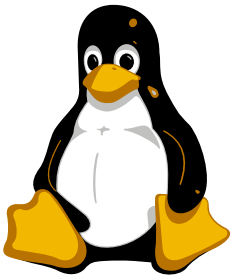


嵌入式 Linux 系统开发入门

练习参考解答

撰稿人：方元

2018 年 5 月

Chapter 1

Linux 基本使用方法

1. 显示环境变量PATH 的命令：

   ```
   $ echo $PATH
   ```

   在环境变量PATH 中增加路径：

   ```
   $ export PATH=/usr/local/bin:$PATH
   ```

2. 创建一个用户账户：

   ```
   # adduser user10
   # chown 700 /home/user100
   ```

   “700” 是访问权限：本人 “rwx”，其他人无权访问。

3. 命令 `ls -l` 列出的是文件最后修改时间。

4. 使用 wget 下载文件：

   ```
   $ wget https://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz
   ```

   使用 curl 下载文件：

   ```
   $ curl -O https://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz
   ```

5. ```
   $ for f in `ls *.c`; do sed -i "1i/* $f */" $f; done
   ```
   ```

Chapter 3

文件读写

1. 制作空文件的方法很多，这里仅举两例。

   - 使用 dd 命令：

     ```
     $ dd if=/dev/zero of=file_8M bs=1K count=8K
     ```

   - 写一个小程序：

清单 3.1: 创建一个 8M 的文件 file8M.c

```
1  #include <stdio.h>
2  int main (int argc, char *argv[])
3  {
4      FILE *fp;
5      int x = 0;
6      fp = fopen("file_8M", "wb");
7      fseek(fp, 8192*1024 - 4, SEEK_SET);
8      fwrite(&x, 4, 1, fp);
9      fclose(fp);
10     return 0;
11 }
```

2. 使用函数 `stat()` 读取文件状态的小程序如下：

清单 3.2: stat.c

```
1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <sys/stat.h>
4 #include <unistd.h>
5 #include <time.h>
6
```

```
 7 int main (int argc, char *argv[])
 8 {
 9     struct stat buf;
10     int ret;
11
12     if(argc < 2) {
13         printf("An exist filename is needed.\n");
14         return -1;
15     }
16     ret = stat(argv[1], &buf);
17     if (ret < 0) {
18         printf("File open error.\n");
19     }
20     printf("%d\n", buf.st_size);
21     printf("%.24s\n", ctime(&buf.st_atim.tv_sec));
22
23     return 0;
24 }
```

以上打印文件的大小和最后访问时间。将 stat() 换成 lstat() 后，软链接文件和链接源打印结果不同。

3. 以下程序每隔 4ms 打印一个 “A”，如果缓冲大小是 1KB，则缓冲区满之前不会显示，装满缓冲区的时间约是 4 秒。程序一次连续打印 “A” 的数量就是缓冲区的大小。

清单 3.3: 缓冲测试 buffersize.c

```
 1 #include <stdio.h>
 2 #include <unistd.h>
 3
 4 int main(int argc, char *argv[])
 5 {
 6     int i = 0;
 7
 8     for (i = 0; i < BUFSIZ*2; i++) {
 9         printf("A");
10         usleep(4000);
11     }
12
13     return 0;
14 }
```

4. 程序清单如下：

清单 3.4: puts.c

```
1  #include <stdio.h>      /* include puts() */
2  #include <stdlib.h>
3  #include <unistd.h>
4
5  /* 重写的 puts() */
6  int puts_STDOUT(const char *s)
7  {
8      char *p = (char *)s;
9      char r = '\n';
10     int len = 0, ret;
11
12     while(1) {
13         if (*p++ == '\0')
14             break;
15         len++;
16     }
17     ret = write(STDOUT_FILENO, s, len);
18     if (ret != len)
19         return ret;
20     ret = write(STDOUT_FILENO, &r, 1);
21     if (ret != 1)
22         return ret;
23     else
24         return len + 1;
25 }
26
27 /* 对比测试程序 */
28 int main(int argc, char *argv[])
29 {
30     puts("This is a test line.\n Hello, world.");
31     puts_STDOUT("This is a test line.\n Hello, world.");
32
33     return 0;
34 }
```

5. 以下是测试程序：

清单 3.5: 对齐 align.c

```
1  #include <stdio.h>
2
3  int main (int argc, char *argv[])
4  {
5      struct tagBitmapFileHeader {
6          unsigned short bmp_Type;
7          unsigned long bmp_Size;
8          unsigned short bmp_Reserved1;
9          unsigned short bmp_Reserved2;
10         unsigned long bmp_OffBits ;
11     };
12     printf("%d\n", sizeof(struct tagBitmapFileHeader));
13     return 0;
14 }
```

在 x86-64 系统中，long 类型变量占 8 个字节。缺省的编译选项下，运行时打印 “32”，表示结构 tagBitmapFileHeader 占 32 字节。在 32 位平台 (或使用 gcc 的-m32 选项)，long 类型变量占 4 个字节，打印结果是 “16”。

出现这个问题的原因来自地址对齐的要求。在地址对齐方式下，处理器对存储器的访问效率最高，因此编译器默认采用地址对齐方式。如果想改变对齐方式，可以在程序中使用 #pragma pack(size)。

注：使用 x86-64 平台的编译器，gcc 选项 -m32 须有 32 位库的支持。

6. 除了将信息输出到标准错误设备stderr，还可以采用下面的方法：

   - 使用 printf("\n");
   - 使用 fflush(stdout);
   - 使用 setbuf() 一组函数。

Chapter 4

多任务机制

1. (略)
2. 下面的程序，子进程创建后立即结束，主进程 20 秒后才读取它的状态，在这 20 秒时间内，子进程是僵尸进程。

清单 4.1: 产生僵尸的程序 zombie.c

```
1  #include <sys/wait.h>
2  #include <unistd.h>
3
4  int main (int argc, char *argv[])
5  {
6      pid_t pid;
7
8      pid = fork();
9      if (pid == 0) {
10         _exit(0);
11     } else {
12         sleep(20);
13         waitpid(pid, NULL, 0);
14     }
15
16     return 0;
17 }
```

3. 服务器：

清单 4.2: 管道服务器 server.c

```
1  #include <sys/types.h>
2  #include <sys/stat.h>
```

```
3  #include <unistd.h>
4  #include <fcntl.h>
5  #include <stdio.h>
6
7  int main (int argc, char *argv[])
8  {
9      int fd_serv, fd_cli, n;
10     int pid, ret;
11     char str[256];
12
13     fd_serv = open("/tmp/serv", O_RDONLY); /* 打开公共管道 */
14
15     n = read(fd_serv, str, 1024);      /* 接收客户端传来的信息 */
16     str[n] = '\0';
17     mkfifo(str, 0666);                /* 创建通信管道 */
18     fd_cli = open(str, O_RDWR);
19
20     while(1) {
21         n = read(STDIN_FILENO, str, 255);
22         write(fd_cli, str, n);        /* 从标准输入设备读，写入管道 */
23         n = read(fd_serv, str, 255);
24         write(STDOUT_FILENO, str, n); /* 从管道读取，写到终端上 */
25     }
26     return 0;
27 }
```

客户端

清单 4.3: 管道客户端 client.c

```
1  #include <sys/types.h>
2  #include <sys/stat.h>
3  #include <unistd.h>
4  #include <fcntl.h>
5  #include <stdio.h>
6  #include <string.h>
7
8  int main (int argc, char *argv[])
9  {
10     int fd_serv, fd_cli, n;
```

```
11      int pid, ret;
12      char str[256];
13
14      fd_serv = open("/tmp/serv", O_RDWR);
15      pid = getpid();
16      sprintf(str, "/tmp/%d", pid);
17      n = strlen(str);
18      write(fd_serv, str, n);
19
20      do {
21          fd_cli = open(str, O_RDWR);
22      } while (fd_cli < 0);
23
24      while(1) {
25          n = read(fd_cli, str, 255);
26          write(STDOUT_FILENO, str, n);
27          n = read(STDIN_FILENO, str, 255);
28          write(fd_serv, str, n);
29      }
30      return 0;
31  }
```

服务器启动之前，使用 `mkfifo` 命令手工创建管道文件 /tmp/serv 。

4. 下面的程序忽略 `SIGTERM` 信号，不能简单地用`kill PID` 终止，必须使用选项 `-9`。

清单 4.4: 忽略 SIGTERM 信号 ignorekill.c

```
1  #include <signal.h>
2  #include <stdio.h>
3  #include <strings.h>
4  #include <unistd.h>
5
6  int main(int argc, char *argv[])
7  {
8      struct sigaction sa;
9      bzero(&sa, sizeof(sa));
10     sa.sa_handler = SIG_IGN;
11     sigaction (SIGTERM, &sa, NULL);
12     while(1) {
13         fprintf(stderr,".");
```

```
14          sleep(1);
15      }
16
17      return 0;
18  }
```

5. 管道的输入端可以通达任意输出端，写入管道的数据可以被任一进程读取。关闭不用的端口，可以避免错误的读/写导致数据传输紊乱。
6. 下面是使用 localtime() 函数的例子：

清单 4.5: 可重入函数测试 reentrance.c

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <time.h>
4
5  int main (int argc, char *argv[])
6  {
7      time_t now;
8      struct tm *t1, *t2;
9
10     now = time(NULL);
11     t1 = localtime(&now);
12
13     sleep(3);          /* 等待3秒 */
14     now += 600;        /* 计时推迟10分钟 */
15     t2 = localtime(&now);
16
17     printf("%02d:%02d:%02d in thread 1.\n",
18            t1->tm_hour, t1->tm_min, t1->tm_sec);
19     printf("%02d:%02d:%02d in thread 2.\n",
20            t2->tm_hour, t2->tm_min, t2->tm_sec);
21
22     return 0;
23 }
```

上例看到，尽管 localtime() 使用的是不同的参数，两个线程打印的结果却相同。原因是返回值指针指向同一个地址，后者将前者的结果覆盖。如果使用了函数 localtime_r()，由于它将返回值作为函数的一个参数，避免了返回值覆盖问题。

Chapter 5

网络套接字编程

1. 下面的程序，小端打印 “78 56 34 12”，大端相反：

清单 5.1: 测试主机端序 get_endian.c

```
1  #include <stdio.h>
2
3  int main (int argc, char *argv[])
4  {
5      int i, a = 0x12345678;
6      char *p = (char *)&a;
7
8      for(i = 0; i < sizeof(a); i++)
9          printf("%02x ", *p++);
10     return 0;
11 }
```

2. 使用函数 htonl() 比较输入参数和返回值，如果二者不同，说明网络字节顺序与本机不同：

清单 5.2: 测试字节顺序 nethost.c

```
1  #include <stdio.h>
2  #include <arpa/inet.h>
3
4  int main ()
5  {
6      int a = 0x12345678, b;
7
8      b = htonl(a);
9      printf("a=%08X, b=%08X\n", a, b);
10     return 0;
```

```
11 }
```

3. IANA 分配的 13 号端口是日时钟服务。
4. 下面是服务器的核心部分：接收来自客户端的请求、创建管道、创建新的进程，父进程将计数器通过管道写给子进程，子进程读取计数器送给客户端：

清单 5.3: 访客统计服务器 guests.c

```
1     int counter = 0;
2     ...
3     while(1) {
4         connfd = accept(sockfd, (struct sockaddr *)&cliaddr,
5                         &addrlen);
6         pipe(pipefd);
7         pid = fork();
8         if(pid == 0) {
9             close(pipefd[1]);
10            read(pipefd[0], &counter, 4);
11            close(pipefd[0]);
12            n = sprintf(buf, "Total visited guests: %d\n", counter);
13            write(connfd, buf, n);
14            close(connfd);
15            exit(0);
16        } else {
17            close(pipefd[0]);
18            counter++;
19            write(pipefd[1], &counter, 4);
20            close(pipefd[1]);
21            close(connfd);
22        }
23    }
24    ...
```

程序功能可使用 telnet 命令测试。(服务器使用多进程，支持并发请求，但未经大规模客户请求测试。)

5. 只需要在循环开始前增加一条语句 `daemon(0, 0);` 便可将上述程序变成守护进程，函数`daemon()` 的第一个参数 0 表示将进程的目录转到根目录，第二个参数 0 表示将标准输入、标准输出和标准错误输出设备重定向到/dev/null 。

Chapter 6

模块与设备驱动

1. (略)
2. `init_module()` 返回 -1 表示模块不驻留，不需要使用`rmmod`。
3. `register_chrdev()` 会被记录在 /proc/devices 中。模块自己也可以使用 `printk()` 作为调试信息，供 `dmesg` 查看。
4. `cp` 命令的执行过程是打开文件、读取文件内容、写入文件内容和关闭文件，它的任务不是读取设备文件名。在这个过程中，得不到设备文件的设备号，所以不能直接复制设备文件名。
5. 可以在读写数据中增加一个特征位，用于区分是数据交换还是 I/O 控制。这样做，增加了读写方法的复杂性，降低了效率。
6. 下面是三个测试程序，结果如下表。

清单 6.1: jiffies 忙等待 jiffies.c

```
1  #include <linux/module.h>
2  #include <linux/sched.h>
3
4  int init_module()
5  {
6      long now = jiffies;
7
8      printk("start at %ld\n", now);
9      while(jiffies < now + 3600*HZ)
10         ;
11
12     printk("stop at %ld\n", jiffies);
13
14     return -1;
15 }
```

清单 6.2: TSC 忙等待 tsc.c (用户空间)

```
1  #include <stdio.h>
2
3  #define rdtsc(low ,high) \
4      __asm__ __volatile__ (" rdtsc " : "=a" (low), "=d" (high ))
5
6  #define CLK  2294720000  /* 2.294GHz */
7
8  int main (int argc, char *argv[])
9  {
10     unsigned long l1, h1, l2, h2;
11     unsigned long long t1, t2;
12
13     rdtsc(l1, h1);
14     printf("%ld %ld\n", h1, l1);
15     t1 = h1*(1L<<32) + l1;
16     while(1) {
17         rdtsc(l2, h2);
18         t2 = h2*(1L<<32) + l2;
19         if (t2 - t1 > CLK*3600)    /* 3600秒 */
20             break;
21     }
22     printf("%ld %ld\n", h2, l2);
23
24     return 0;
25 }
```

清单 6.3: mdelay 忙等待 mdelay.c

```
1  #include <linux/module.h>
2  #include <linux/sched.h>
3
4  int init_module()
5  {
6      int cnt;
7
8      printk("start at %ld\n", jiffies);
9      for(cnt = 0; cnt < 3600; cnt++)
10         mdelay(1000);
```

```
11
12      printk("stop at %ld\n", jiffies);
13
14      return -1;
15  }
```

表 6.1: 不同方法 1 小时定时结果

jiffies	TSC	mdelay
3600.06	3600.08	3572.03

jiffies 和 TSC 是实时读取硬件计时器，而 mdelay() 是软件耗时，会积累误差。

Chapter 14

图形用户接口

本章给出几个小测试程序，抛砖引玉。

14.1 建立基础图形库

使用 fbset 命令，可以看到 Framebuffer 的数据结构：

```
mode "800x480"
    geometry 800 480 800 480 32
    timings 0 0 0 0 0 0 0
    accel true
    rgba 8/16,8/8,8/0,0/0
endmode
```

可以看出，显示分辨率 800×480，32 位色，以及 RGBA 的位移位置。

清单 14.1: 初始化图形环境 graphics.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/ioctl.h>
4  #include <sys/mman.h>
5  #include <fcntl.h>
6  #include <linux/fb.h>
7
8  static struct fb_fix_screeninfo finfo;
9  static struct fb_var_screeninfo vinfo;
10 static unsigned char *fbmem;
11
12 /* 在 (x, y) 处画一个颜色为 color 的点.
```

```
13      本程序只适用于RGBA 32位 Frame buffer 格式. */
14  int drawpixel(int x, int y, int color)
15  {
16      unsigned int ptr;
17
18      if((x >= vinfo.xres) || (x < 0)
19         || (y >= vinfo.yres) || (y < 0))
20          return;
21      ptr = y * finfo.line_length + x * 4;
22      *(unsigned int *)(fbmem + ptr) = color;
23
24      return 0;
25  }
26
27  /* 初始化 Frame Buffer，获得缓冲区首地址 fbmem */
28  int fb_init(char *dev)
29  {
30      int fd;
31      long screensize = 0;
32
33      fd = open(dev, O_RDWR);
34
35      if (fd < 0){
36          printf("Error : Can not open framebuffer device\n");
37          exit(-1);
38      }
39      ioctl(fd, FBIOGET_VSCREENINFO, &vinfo);
40      ioctl(fd, FBIOGET_FSCREENINFO, &finfo);
41      printf("Resolution: x = %d, y = %d\n", vinfo.xres, vinfo.yres);
42
43      screensize = vinfo.xres*vinfo.yres*(vinfo.bits_per_pixel/8);
44      fbmem = (char *)mmap(0, screensize, PROT_READ|PROT_WRITE,
45                           MAP_SHARED, fd,0);
46      if (fbmem == NULL) {
47          printf ("Error: failed to map framebuffer device to memory.\n");
48          exit(-1);
49      }
50
```

```
51      return fd;
52  }
```

清单 14.2: 画直线程序 line.c

```
1   #include "graphics.h"
2
3   /* 画一条 (x0, y0) 到 (x1, y1) 的直线 */
4   int line(int x0, int y0, int x1, int y1, int color)
5   {
6       int dx = x1 - x0;
7       int dy = y1 - y0;
8       int sx, sy;
9       int x, y, temp;
10      float k;
11
12      sx = (dx > 0)? 1: -1;
13      sy = (dy > 0)? 1: -1;
14
15      if(ABS(dx) > ABS(dy)) {      /* 缓斜线，|斜率| < 1 */
16          if(x0 > x1) {
17              temp = x0, x0 = x1, x1=temp;
18              temp = y0, y0 = y1, y1=temp;
19          }
20          k = (float)dy / dx;
21          for(x = x0; x <= x1; x++) {
22              y = y0 + k * (x - x0);
23              drawpixel(x, y, color);
24          }
25      } else if (ABS(dy) > ABS(dx)) {    /* 陡斜线 */
26          if(y0 > y1) {
27              temp = x0, x0 = x1, x1=temp;
28              temp = y0, y0 = y1, y1=temp;
29          }
30          k = (float)dx / dy;
31          for(y = y0; y <= y1; y++) {
32              x = x0 + k * (y - y0);
33              drawpixel(x, y, color);
34          }
```

```
35      } else {      /* (x0, y0) 和 (x1, y1) 重叠 */
36          drawpixel(x0, y0, color);
37      }
38
39      return 0;
40  }
```

清单 14.3: 矩形处理子程序 rectangle.c

```
1   #include "graphics.h"
2
3   /* 两个顶点 (x0, y0) -- (x1, y1) 决定的矩形 */
4   int rectangle(int x0, int y0, int x1, int y1, int color)
5   {
6       int ret = 0;
7
8       ret += line(x0, y0, x1, y0, color);
9       ret += line(x1, y0, x1, y1, color);
10      ret += line(x1, y1, x0, y1, color);
11      ret += line(x0, y1, x0, y0, color);
12
13      return ret;
14  }
15
16  /* 填充矩形 */
17  int fillrect(int x0, int y0, int x1, int y1, int color)
18  {
19      int x, y;
20
21      if (x0 > x1) {
22          temp = x0, x0 = x1, x1 = temp;
23      }
24      if (y0 > y1) {
25          temp = y0, y0 = y1, y1 = temp;
26      }
27      for(y = y0; y <= y1; y++) {
28          for(x = x0; x <= x1; x++) {
29              drawpixel(x, y, color);
30          }
```

```
31      }
32      return 0;
33  }
```

清单 14.4: 画圆或椭圆子程序 ellipse.c

```
1   #include <math.h>
2   #include "graphics.h"
3
4   /* 中心 (x0, y0)、长短半轴为 a、b的椭圆，当 a、b 相等时为正圆形.
5      圆周上的点数约是半径的6.28倍. 利用对称性，只计算第一象限. */
6   int ellipse(int x0, int y0, int a, int b, int color)
7   {
8       int x, y;
9       float theta, dth;
10      int axis_m;
11
12      axis_m = (a > b) ? a : b;
13      if (axis_m <= 0)
14          return -1;
15      dth = 1.57 / axis_m;
16
17      for(theta = 0; theta <= M_PI/2; theta += dth) {
18          x = a * cos(theta);
19          y = b * sin(theta);
20          drawpixel(x0 + x, y0 + y, color);
21          drawpixel(x0 - x, y0 + y, color);
22          drawpixel(x0 + x, y0 - y, color);
23          drawpixel(x0 - x, y0 - y, color);
24      }
25
26      return 0;
27  }
```

清单 14.5: 头文件 graphics.h

```
1   #ifndef _GRAPHICS_H
2   #define _GRAPHICS_H
3
4   #define  ABS(x)    (((x)>=0)? (x): -(x))
```

```
5
6  int fb_init(char *dev);
7  int drawpixel(int x, int y, int color);
8
9  int line(int x0, int y0, int x1, int y1, int color);
10 int ellipse(int x0, int y0, int a, int b, int color);
11
12 int rectangle(int x0, int y0, int x1, int y1, int color);
13 int fillrect(int x0, int y0, int x1, int y1, int color);
14
15 #endif   /* _GRAPHICS_H */
```

清单 14.6: 编译共享库的 Makefile

```
1  CC      = arm-linux-gcc
2  SRC     = graphics.c line.c ellipse.c rectangle.c
3
4  LIBNAME = graphics
5  VERSION = 0
6  MINOR   = 0
7
8  all: lib$(LIBNAME).so lib$(LIBNAME).so.$(VERSION)
9
10 lib$(LIBNAME).so: lib$(LIBNAME).so.$(VERSION).$(MINOR)
11         ln -sf $^ $@
12
13 lib$(LIBNAME).so.$(MINOR): lib$(LIBNAME).so.$(VERSION).$(MINOR)
14         ln -sf $^ $@
15
16 lib$(LIBNAME).so.$(VERSION).$(MINOR): $(SRC)
17         $(CC) -o $@ $^ -shared -fPIC \
18                 -Wl,-soname,lib$(LIBNAME).so.$(VERSION)
19
20 clean:
21         $(RM) lib$(LIBNAME).*
```

动态库编译后，生成 libgraphics.so.0.0、libgraphics.so.0 和 libgraphics.so 三个文件。

14.2 编写简单的应用

下面是一个简单的测试程序：

清单 14.7: 用于测试的主程序 main,c

```
1  #include <unistd.h>
2  #include "graphics.h"
3
4  int main(int argc, char *argv[])
5  {
6      int fd = fb_init("/dev/fb0");
7
8      fillrect(10, 10, 640, 480, 0xffffffff);   /* 白屏 */
9      ellipse(400, 240, 30, 40, 0xffff0000);
10     line(200,200, 600, 450, 0xff0000ff);
11
12     rectangle(300, 120, 500, 360, 0xff00ffff);
13     close(fd);
14
15     return 0;
16 }
```

共享库编译后，用下面的命令编译测试程序：

```
$ arm-linux-gcc -o main main.c -L. -lgraphics -lm
```

将动态库和链接复制到目标系统的 /usr/lib 目录，程序main 在目标系统中运行。

14.3 研究算法效率和动态库的作用

请读者检索 Bresenham 算法，用来替代上节实现的功能。重新生成 libgraphics.so.0.0。在不重新编译主程序的情况下运行。

编写一个有大量画图动作的主程序 (例如，在屏幕区域内随机画大量的圆)，测量程序运行时间，用数据说明不同算法的优劣。

14.4 其他尝试

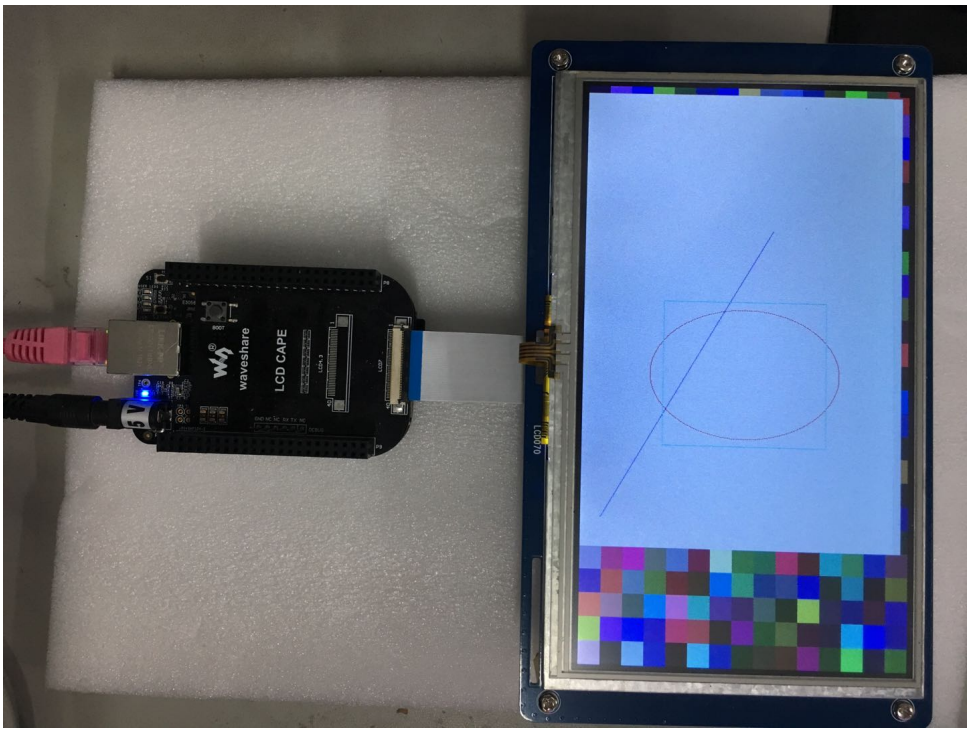
下面是一个利用多线程技术实现的动态马赛克效果 (程序存在竞争冒险)：

清单 14.8: 马赛克效果 mosaic.c

```
1  #include <stdlib.h>
```

```
2  #include <unistd.h>
3  #include <pthread.h>
4  #include "graphics.h"
5
6  #define  BLOCKSIZE 32
7
8  #define  RES_X  800
9  #define  RES_Y  480
10
11 struct coord
12 {
13     int x, y;
14 };
15
16 void *drawblock(void *param)
17 {
18     struct coord *p = (struct coord*)param;
19     int i, j, color;
20
21     while(1) {
22         color = rand();
23         for(i = 0; i < BLOCKSIZE; i++)
24             for(j = 0; j < BLOCKSIZE; j++)
25                 drawpixel(p->x + i, p->y + j, color);
26         usleep((unsigned)rand()/500);
27     }
28 }
29
30 int main (int argc, char *argv[])
31 {
32     int fd = 0;
33     int i, j, k, color = 0;
34     pthread_t threadId;
35     struct coord xy[500];
36
37     fd = fb_init("/dev/fb0");
38
39     k = 0;
```

```
40      for(i = 0; i < RES_X; i += BLOCKSIZE) {
41          for(j = 0; j < RES_Y; j += BLOCKSIZE) {
42              xy[k].x = i;
43              xy[k].y = j;
44              pthread_create (&threadId, NULL, drawblock, &xy[k]);
45              k++;
46          }
47      }
48      sleep(5);
49      close (fd);
50
51      return 0;
52  }
```



Chapter 15

音频接口程序设计

15.1 设置内核支持 USB 声卡

Beaglebone Black 只有 HDMI 支持的声音输出，没有其他板载声音采集和输出设备。我们另外选择了一款 USB 声卡，内核识别为即插即用设备 (Plug and Play, PnP)：

```
usb 2-2: Product: USB PnP Sound Device
usb 2-2: Manufacturer: C-Media Electronics Inc.
input: C-Media Electronics Inc.
```

在配置内核时只需要选中 USB 声卡项中的 USB Audio/MIDI driver 即可。如果作为模块编译，应在内核启动后加载 snd-usb-audio.ko 驱动。插入声卡，可以看到 /dev/snd 下面多出一组 controlCx 和 pcmCxDx 设备文件。如果配置内核时选择兼容 OSS (Open Sound System, 开放声音系统)，同时还可以看到 /dev/dsp、/dev/mixer 等设备。

15.2 开放声音系统

下面是基于 OSS 的音频数据采集和输出。

清单 15.1: OSS 录音，oss_record.c

```
1 /* oss_record.c
2  */
3
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <string.h>
7 #include <sys/ioctl.h>
8 #include <unistd.h>
9 #include <fcntl.h>
```

```
10 #include <sys/soundcard.h>
11
12 #define    BUF    16
13 int main(int argc, char *argv[])
14 {
15     int audio_fd;
16     int mixer_fd;
17
18     int format = AFMT_S16_NE;
19     int channels = 1;
20     int speed = 48000;  //44.1 KHz
21
22     int file_fd;
23     signed short applicbuf[BUF];
24     int count;
25
26     // open device file
27     if ((audio_fd = open("/dev/dsp",O_RDWR,0)) == -1) {
28         perror("/dev/dsp");
29         exit(1);
30     }
31
32     /* 设置数据格式 */
33     if (ioctl(audio_fd,SNDCTL_DSP_SETFMT, &format) == -1) {
34         perror("SNDCTL_DSP_SETFMT");
35         exit(1);
36     }
37
38     /* 设置通道数 */
39     if (ioctl(audio_fd, SNDCTL_DSP_CHANNELS, &channels) == -1) {
40         perror("SNDCTL_DSP_CHANNELS");
41         exit(1);
42     }
43
44     /* 设置采样率 */
45     if (ioctl(audio_fd, SNDCTL_DSP_SPEED, &speed) == -1) {
46         perror("SNDCTL_DSP_SPEED");
47         exit(1);
```

```
48      } else
49          printf("Support 44.1 KHz , Actual Speed : %d \n",speed);
50
51      if ((file_fd = open(argv[1], O_WRONLY | O_CREAT, 0)) == -1) {
52          perror(argv[1]);
53          exit(1);
54      }
55
56      /* 以下采样10秒数据，存入文件 */
57
58      int totalbyte= speed * channels * 2 * 10; // 10 seconds
59      int totalword = totalbyte/2;
60      int total = 0;
61
62      while (total != totalword) {
63          if (totalword - total >= BUF)
64              count = BUF;
65          else
66              count = totalword - total;
67
68          read(audio_fd, applicbuf, count);
69          write(file_fd, applicbuf, count);
70          total += count;
71      }
72      close(audio_fd);
73      close(file_fd);
74
75      return 0;
76  }
```

清单 15.2: OSS 播放音频数据文件 oss_play.c

```
1  /* oss_play.c
2   */
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7  #include <sys/ioctl.h>
```

```
 8 #include <unistd.h>
 9 #include <fcntl.h>
10 #include <sys/soundcard.h>
11
12 #define  BUF    40
13
14 int main(int argc, char *argv[])
15 {
16     int audio_fd;
17     int format = AFMT_S16_NE;
18     int channels = 1;     /* 单声道 */
19     int speed = 16000;    /* 采样率*/
20     int file_fd;
21     signed short applicbuf[BUF];
22     int count;
23
24     // open device file
25     if ((audio_fd = open("/dev/dsp",O_RDWR,0)) == -1) {
26         perror("/dev/dsp");
27         exit(1);
28     }
29
30     /*  设置数据格式 */
31     if (ioctl(audio_fd,SNDCTL_DSP_SETFMT, &format) == -1) {
32         perror("SNDCTL_DSP_SETFMT");
33         exit(1);
34     }
35     /* 设置通道数 */
36     if (ioctl(audio_fd, SNDCTL_DSP_CHANNELS, & channels) == -1) {
37         perror("SNDCTL_DSP_CHANNELS");
38         exit(1);
39     }
40
41     /* 设置采样率 */
42     if (ioctl(audio_fd, SNDCTL_DSP_SPEED, &speed) == -1) {
43         perror("SNDCTL_DSP_SPEED");
44         exit(1);
45     } else
```

```
46          printf("Actual Speed : %d \n",speed);
47
48      if ((file_fd = open(argv[1], O_RDONLY, 0)) == -1) {
49          perror(argv[1]);
50          exit(1);
51      }
52
53      /* 读取数据文件，写入声卡 */
54      while ((count = read(file_fd, applicbuf, BUF)) > 0) {
55          write(audio_fd, applicbuf, count);
56      }
57
58      close(audio_fd);
59      close(file_fd);
60
61      return 0;
62  }
```

15.3 高级 Linux 声音架构

基于 ALSA (Advanced Linux Sound Architecture) 的声卡编程需要先移植 alsa-lib 库。

下载 alsa-lib 源码，交叉编译，生成 libasound。将共享库复制到 Beaglebone black 的 /usr/lib，同时安装在主机的交叉编译安装目录，供主机交叉编译用。

清单 15.3: ALSA 播放正弦波 alsa_play.c

```
1  /* alsa_play.c
2   */
3
4  #include <alsa/asoundlib.h>
5  #include <stdio.h>
6  #include <math.h>
7
8  int main(int argc, char *argv[])
9  {
10     short buffer[4096];
11     int err;
12     unsigned int fs = 16000;
13     snd_pcm_t *handle;
```

```
14      snd_pcm_sframes_t frames;
15      unsigned long ptr = 0, i;
16
17      err = snd_pcm_open(&handle, "plughw:1,0", SND_PCM_STREAM_PLAYBACK, 0);
18      if (err < 0) {
19          perror("Device open error.\n");
20          return -1;
21      }
22
23      err = snd_pcm_set_params(handle,
24                               SND_PCM_FORMAT_S16_LE,
25                               SND_PCM_ACCESS_RW_INTERLEAVED,
26                               2,         /* 通道数 */
27                               fs,        /* 采样率 */
28                               1,         /* 重采样 */
29                               100000     /* 延迟(us) */
30                              );
31      if (err < 0) {
32          perror("Playback open error.\n");
33          return -1;
34      }
35
36      while (1) {
37          for(i = 0; i < 4096; i += 2) {    /* 左右声道 500Hz/800Hz */
38              buffer[i + 0] = 20000*sin(2*M_PI*ptr*500/fs);
39              buffer[i + 1] = 20000*sin(2*M_PI*ptr*800/fs);
40              ptr++;
41          }
42          err = snd_pcm_wait(handle, 1000);
43          if (err < 0) {
44              fprintf (stderr, "poll failed (%d)\n", err);
45              break;
46          }
47
48          frames = snd_pcm_writei(handle, buffer, 2048);
49
50          if (frames < 0)
51              err = snd_pcm_recover(handle, frames, 0);
```

```
52          if (err < 0) {
53              perror("snd_pcm_writei failed.\n");
54              break;
55          }
56      }
57      snd_pcm_close(handle);
58
59      return 0;
60  }
```

清单 15.4: ALSA 录音 alsa_record.c

```
1   /* alsa_record.c
2    */
3
4   #include <alsa/asoundlib.h>
5   #include <unistd.h>
6   #include <stdio.h>
7
8   int main(int argc, char *argv[])
9   {
10      short buffer[4096];
11      int err;
12      unsigned int fs = 16000;
13      snd_pcm_t *handle;
14      snd_pcm_sframes_t frames;
15      unsigned long ptr = 0, i;
16
17      err = snd_pcm_open(&handle, "plughw:1,0", SND_PCM_STREAM_PLAYBACK, 0);
18      if (err < 0) {
19          perror("Device open error.\n");
20          return -1;
21      }
22
23      err = snd_pcm_set_params(handle,
24                               SND_PCM_FORMAT_S16_LE,
25                               SND_PCM_ACCESS_RW_INTERLEAVED,
26                               1,         /* 通道数 */
27                               fs,        /* 采样率 */
```

```
28                                  1,          /* 重采样 */
29                                  100000      /* 延迟(us) */
30                                 );
31      if (err < 0) {
32          perror("Playback open error.\n");
33          return -1;
34      }
35
36      while (1) {
37          frames = snd_pcm_readi(handle, buffer, 4096);
38          if (frames == -EPIPE) {
39              /*  EPIPE means overrun */
40              fprintf(stderr, "overrun occurred\n");
41              snd_pcm_prepare(handle);
42          } else if (frames < 0) {
43              fprintf(stderr, "error from read.\n");
44          } else if (4096 != (int)frames) {
45              fprintf(stderr, "read %ld, expected 4096.\n", frames);
46          }
47
48          err = write(STDOUT_FILENO, buffer, sizeof(buffer));
49      }
50      snd_pcm_close(handle);
51
52      return 0;
53  }
```

Chapter 16

嵌入式系统中的 I/O 接口驱动

下面是实现 GPIO1_12、GPIO1_13 的驱动程序和测试程序。驱动加载后，创建一个字符设备文件，主设备号 223 ，次设备号 1：

```
# mknod /dev/gpio c 223 1
```

在 Beaglebone Black P8 的 11 或 12 脚接一个发光二极管。应用程序运行时可以看看到 LED 闪数下，随后应用程序将这两个端口改成输入方式。可使用导线将 pin11 或 pin12 接地/悬空，观察打印结果。

清单 16.1: GPIO 驱动 gpio.c

```
1  /* gpio.c
2   */
3
4  #include <linux/kernel.h>
5  #include <linux/fs.h>       /* file_operations */
6  #include <linux/io.h>       /* ioremap()、iounmap() */
7  #include <linux/slab.h>     /* kmalloc()、kfree() */
8  #include <linux/uaccess.h> /* copy_to_user()、copy_from_user() */
9
10 #include "gpio.h"
11
12 int led_open (struct inode *inode, struct file *filp)
13 {
14     int minor_dev = MINOR(inode->i_rdev);
15     gpio_t *gpio;
16
17     printk("minor = %d\n", minor_dev);
18     filp->private_data = kmalloc(sizeof(gpio_t), GFP_KERNEL);
```

```
19      gpio = (gpio_t *)filp->private_data;
20
21      gpio->ctrl = ioremap(CONTRL, 128*1024);
22
23      if(minor_dev == 1) {
24          gpio->port= ioremap(GPIO1, 4096);
25          gpio->ctrl[AD12]  = 0x37;       /* GPIO1[12] */
26          gpio->ctrl[AD13]  = 0x37;       /* GPIO1[13] */
27      } else if(minor_dev == 2) {
28          gpio->port = ioremap(GPIO2, 4096);  /* TODO: */
29      }
30
31      return 0;
32  }
33
34  int led_close (struct inode *inode, struct file *filp)
35  {
36      gpio_t *gpio;
37
38      gpio = (gpio_t *)filp->private_data;
39      iounmap(gpio->port);
40      iounmap(gpio->ctrl);
41
42      kfree(filp->private_data);
43      return 0;
44  }
45
46  ssize_t led_read (struct file *filp,
47                    char __user *buf,
48                    size_t size,
49                    loff_t *offset)
50  {
51      gpio_t *gpio = (gpio_t *)filp->private_data;
52      int val, n;
53
54      val = gpio->port[DATIN];
55      n = copy_to_user(buf, &val, sizeof(val));
56
```

```
57      return size;
58  }
59
60  ssize_t led_write (struct file *filp,
61                     const char __user *buf,
62                     size_t size,
63                     loff_t *offset)
64  {
65      gpio_t *gpio = (gpio_t *)filp->private_data;
66      int val, n;
67      n = copy_from_user(&val, buf, size);
68      if (size > 4)  size = 4;
69  //    printk("value  %x write to device\n", val);
70      gpio->port[DATOUT] = val;
71
72      return size;
73  }
74
75  long led_ioctl(struct file *filp,
76                 unsigned int cmd,
77                 unsigned long arg)
78  {
79      //   ioctl(fd, LEDIOSET, &arg);
80      gpio_t *gpio = (gpio_t *)filp->private_data;
81      int val, n;
82
83      switch(cmd) {
84          case LEDIOSET:
85              n = copy_from_user(&val, (unsigned int *)arg, 4);
86              gpio->port[OE] = val;
87              break;
88          case LEDIOGET:
89              val = gpio->port[OE];
90              n = copy_to_user((unsigned int *)arg, &val, 4);
91              break;
92          default:
93              break;
94      }
```

```
95
96      return 0;
97  }
98
99  struct file_operations fop={
100     .open    = led_open,
101     .release = led_close,
102     .read    = led_read,
103     .write   = led_write,
104     .unlocked_ioctl = led_ioctl,
105 };
106
107 int init_module(void)
108 {
109     int val;
110
111     val = register_chrdev(223, "gpio LED", &fop);
112
113     if (val == 0) {
114         printk("Module installed.\n");
115     } else {
116         printk("Module register failed.\n");
117     }
118
119     return val;
120 }
121
122 void cleanup_module(void)
123 {
124     unregister_chrdev(223, "gpio LED");
125     printk("module removed from kernel.\n");
126 }
```

清单 16.2: 驱动程序头文件 gpio.h

```
1  /* gpio.h
2   */
3
4  #ifndef _GPIO_H
```

```
5  #define _GPIO_H
6
7  #include <linux/ioctl.h>
8
9  typedef struct gpio {
10     volatile unsigned int *port;
11     volatile unsigned int *ctrl;
12 } gpio_t;
13
14 /* 以下是寄存器地址。请查阅 AM3358 数据手册 */
15 #define GPIO0   (0x48E07000)
16 #define GPIO1   (0x4804c000)
17 #define GPIO2   (0x481AC000)
18 #define GPIO3   (0x481AE000)
19
20 #define OE      (0x134/4)
21 #define DATIN   (0x138/4)
22 #define DATOUT  (0x13c/4)
23 #define CLR     (0x190/4)
24 #define SET     (0x194/4)
25
26 #define CONTRL  (0x44e10000)
27 #define AD12    (0x830/4)
28 #define AD13    (0x834/4)
29
30 #define  DEVICE_NAME   "/dev/gpio_led"
31
32 #define  LEDIOSET _IOW(221, 0, int)
33 #define  LEDIOGET _IOR(221, 0, int)
34
35 #endif      /* _GPIO_H */
```

清单 16.3: GPIO 测试程序 apps.c

```
1  /* apps.c
2   */
3
4  #include <stdio.h>
5  #include <unistd.h>
```

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>

#include "gpio.h"

int main(int argc, char *argv[])
{
    int fd;
    int val, i;

    fd = open(argv[1], O_RDWR);

    ioctl(fd, LEDIOGET, &val);
    val &= ~(0b11 << 12);          /* GPIO12、GPIO13 设为输出 */
    ioctl(fd, LEDIOSET, &val);
    for(i = 0; i < 10; i++) {
        val = (0b11 << 12);
        write(fd, &val, 4);
        usleep(200000);
        val = (0b00 << 12);
        write(fd, &val, 4);
        usleep(200000);
    }

    ioctl(fd, LEDIOGET, &val);
    val |= (0b11 << 12);           /* GPIO12、GPIO13 设为输入 */
    ioctl(fd, LEDIOSET, &val);
    for(;;) {
        read(fd, &val, 4);
        printf("%08X\n", val);
        usleep(200000);
    }
    close(fd);

    return 0;
}
```