

华中农业大学

学生实验报告

专业: 计科 2102

姓名: 高星杰

学号: 2021307220712

日期: 2024 年 3 月 25 日

成绩:

实验课程	嵌入式系统 A 实验	实验名称	嵌入式编程基础实验	实验类型	验证	设计
					综合	基础

实验目的:

1. 了解开发板 Tiny4412 的功能、资源、开发环境及其基本使用;
2. 掌握 Ubuntu 上 arm-linux-gcc 交叉编译环境的配置和使用;
3. 掌握通过 minicom 实现主机和开发板间串口通信;
4. 掌握 ARM 架构 Linux 嵌入式多线程开发步骤
5. 掌握 pthread 多线程支持库的使用, 设计多线程程序应用
6. 掌握 Linux 嵌入式系统的串口程序设计, 设计串行端口程序
7. 掌握 Linux 环境下 I/O 操作 tcgetattr(), tcsetattr(), tcflush() 等

实验要求:

1. C 语言编程基础。
2. 掌握在 Linux 下常用编辑器 gedit, vi 的使用。
3. 掌握 Makefile 的编写和使用。
4. 掌握 Linux 下的程序编译与交叉编译过程
5. 掌握 minicom 使用
6. 熟悉 Tiny4412 开发环境

实验原理:

1. Pthread 库

pthread 库是 POSIX 线程库 (POSIX Threads Library) 的简称, 是一个为应用程序提供多线程编程接口的标准化线程库。它定义了创建和操作线程的一套 API。使用 pthread 库可以让程序并发执行多个任务, 从而提高应用程序的性能和响应能力。

pthread 库主要提供以下功能:

- a) 线程管理
 - 创建线程(pthread_create)
 - 等待线程结束(pthread_join)
 - 线程分离(pthread_detach)
- b) 互斥量(mutex)
 - 创建互斥量(pthread_mutex_init)
 - 锁定互斥量(pthread_mutex_lock)
 - 解锁互斥量(pthread_mutex_unlock)
- c) 条件变量
 - 创建条件变量(pthread_cond_init)
 - 等待条件变量(pthread_cond_wait)

通知条件变量(pthread_cond_signal pthread_cond_broadcast)

d) 线程同步

线程同步(pthread_barrier_wait)

自旋锁(pthread_spin_init pthread_spin_lock)

e) 线程属性管理

设置分离状态(pthread_attr_setdetachstate)

设置堆栈大小(pthread_attr_setstacksize)

设置优先级(pthread_attr_setschedparam)

pthread 库是符合 POSIX 标准的,可在多种 UNIX 系统和类 UNIX 系统上运行,如 Linux、macOS 等。许多编程语言都提供了使用 pthread 库的接口,如 C、C++等。通过 pthread 库,开发者可以充分利用多核 CPU 的优势,编写高效的多线程应用程序。

2. 什么是串行端口程序设计?

串行端口程序设计就像是你与一个只会说一种语言的外国人交流,你们之间需要一个翻译。

计算机是你,串行设备就是那个外国人。你们虽然都有自己的“语言”,但是彼此无法直接沟通。于是,串行端口就扮演了“翻译”的角色。它规定了一套“通信规则”,比如用什么样的语速(波特率)、用什么样的文字(数据位)、用什么样的停顿(停止位)、用什么样的发音(奇偶校验)等等。

只有计算机和串行设备都遵循同一套规则,才能相互理解对方的“语言”,顺利地进行数据交换和指令传递。

串行端口程序设计就是编写这份“翻译手册”,让计算机和串行设备按照统一的通信协议进行对话,完成所需的工作,就像外国人和你借助翻译在交流一样。

串行端口就是一个“桥梁”,串行端口程序就是搭建这座桥梁的“工程师”,负责制定交流的规范,实现设备与计算机的无缝对接。

实验内容:

1. 准备工作

(1) 连接好开发板,将开发板串口与主机串口相连

(2) 打开终端,激活 sudo 权限

2. 在主机上完成交叉编译环境的配置

(1) 将 arm-linux-gcc-4.6.4- 下载到主机上;

(2) 使用 tar arm-linux-gcc-4.6.4-v6-vfp-20120301.tar -C / 命令将其解压到/opt 路径下;

(3) 将 arm-linux-gcc 加入环境变量中,使用命令 gedit ~/.bashrc, 打开编辑模型;

```
:/home/oseasy# gedit ~/.bashrc
```

(4) 在文件末尾追加 PATH=\$PATH:/opt/FriendlyARM/toolschain/4.6.4/bin,即加入 PATH 环境中;

```

65 ;;
66 esac
67
68 # enable color support of ls and also add handy aliases
69 if [ -x /usr/bin/dircolors ]; then
70     test -r ~/.dircolors && eval "$(dircolors -b ~/.dircolors)" || eval "$(dircolors -b)"
71     alias ls='ls --color=auto'
72     #alias dir='dir --color=auto'
73     #alias vdir='vdir --color=auto'
74
75     alias grep='grep --color=auto'
76     alias fgrep='fgrep --color=auto'
77     alias egrep='egrep --color=auto'
78 fi
79
80 # some more ls aliases
81 alias ll='ls -alF'
82 alias la='ls -A'
83 alias l='ls -CF'
84
85 # Alias definitions.
86 # You may want to put all your additions into a separate file like
87 # ~/.bash_aliases, instead of adding them here directly.
88 # See /usr/share/doc/bash-doc/examples in the bash-doc package.
89
90 if [ -f ~/.bash_aliases ]; then
91     . ~/.bash_aliases
92 fi
93
94 # enable programmable completion features (you don't need to enable
95 # this, if it's already enabled in /etc/bash.bashrc and /etc/profile
96 # sources /etc/bash.bashrc).
97 #if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
98 #    . /etc/bash_completion
99 #fi
100
101 export PATH=$PATH:/opt/TuxamitoSoftToolchains/arm-arm1176jzfssf-linux-gnueabi/gcc-4.6.4/bin

```

(5) 使用 `source ~/.bashrc` 更新环境变量，`echo $PATH` 显示编译器逻辑则正确：

```

(gedit:10434): Tepl-WARNING **: 20:07:37.525: GVfs metadata is not supported. Fa
llback to TeplMetadataManager. Either GVfs is not correctly installed or GVfs me
tadata are not supported on this platform. In the latter case, you should config
ure Tepl with --disable-gvfs-metadata.
export PATH=$PATH:/opt/TuxamitoSoftToolchains/arm-arm1176jzfssf-linux-gnueabi/gc
c-4.6.4/bin
^Z
[1]+ 已停止      gedit ~/.bashrc

```

(6) 使用 `arm-linux-gcc -V` 显示交叉编译器版本则说明配置正确。

```

root@PC05:/home/oseasy# source /root/.bashrc
root@PC05:/home/oseasy# arm-linux-gcc -v
Using built-in specs.
COLLECT_GCC=arm-linux-gcc
COLLECT_LTO_WRAPPER=/opt/TuxamitoSoftToolchains/arm-arm1176jzfssf-linux-gnueabi/gcc-4.6.4/libexec/gcc/arm-arm1176jzfssf-linux-gnueabi/4.6.4/lto-wrapper
Target: arm-arm1176jzfssf-linux-gnueabi
Configured with: /work/buildldir/src/gcc-4.6.4/configure --build=x86_64-build-unknown-linux-gnu --host=x86_64-build-unknown-linux-gnu --target=arm-arm1176jzfssf-linux-gnueabi --prefix=/opt/TuxamitoSoftToolchains/arm-arm1176jzfssf-linux-gnueabi/gcc-4.6.4 --with-sysroot=/opt/TuxamitoSoftToolchains/arm-arm1176jzfssf-linux-gnueabi/gcc-4.6.4/arm-arm1176jzfssf-linux-gnueabi/sysroot --enable-languages=c,c++ --with-arch=armv6zk --with-cpu=arm1176jzf-s --with-tune=arm1176jzf-s --with-fpu=vfp --with-float=softfp --with-pkgversion=crosstool-NG hg+unknown-20130521.154019 - tc0002 --disable-sjlj-exceptions --enable-cxa_atexit --disable-libmudflap --disable-lbgomp --disable-libssp --disable-libquadmath --disable-libquadmath-support --with-gmp=/work/buildldir/arm-arm1176jzfssf-linux-gnueabi/buildtools --with-mpfr=/work/buildldir/arm-arm1176jzfssf-linux-gnueabi/buildtools --with-mpc=/work/buildldir/arm-arm1176jzfssf-linux-gnueabi/buildtools --with-ppc=/work/buildldir/arm-arm1176jzfssf-linux-gnueabi/buildtools --with-cloog=/work/buildldir/arm-arm1176jzfssf-linux-gnueabi/buildtools --with-elf=/work/buildldir/arm-arm1176jzfssf-linux-gnueabi/buildtools --with-host-libstdcxx=static-libgcc -Wl,-Bstatic,-lstdc++,-Bdynamic -ln --enable-threads=posix --enable-target-optspace --without-long-double-128 --disable-nls --disable-multilib --with-local-prefix=/opt/TuxamitoSoftToolchains/arm-arm1176jzfssf-linux-gnueabi/gcc-4.6.4/arm-arm1176jzfssf-linux-gnueabi/sysroot --enable-c99 --enable-long-long
Thread model: posix
gcc version 4.6.4 (crosstool-NG hg+unknown-20130521.154019 - tc0002)
root@PC05:/home/oseasy#

```

3. 修改 minicom 模式，为后续文件传送做准备

(1) 进入 minicom 的设置里

```

Welcome to minicom 2.7.1

OPTIONS: I18n
Compiled on Dec 23 2019, 02:06:26.
Port /dev/ttyS0, 03:06:47

Press CTRL-A Z for help on special keys

+---+[configuration]-----+
| Filenames and paths      |
| File transfer protocols  |
| Serial port setup        |
| Modem and dialing        |
| Screen and keyboard     |
| Save setup as dfl        |
| Save setup as..         |
| Exit                      |
+---+

```

(2) 修改 serial device 和波特率，关闭硬件控制流；

```

Welcome to minicom 2.7.1

OPTI+-----+
Comp| A - Serial Device      : /dev/ttyS0
Port| B - Lockfile Location   : /var/lock
    | C - Callin Program     :
Pres| D - Callout Program    :
    | E - Bps/Par/Bits       : 115200 8N1
    | F - Hardware Flow Control : No
    | G - Software Flow Control : No
    |
    | Change which setting? 
    |-----+
    | | Screen and keyboard |
    | | Save setup as dfl  |
    | | Save setup as..  |
    | | Exit                |
    |-----+

CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyS0

```

(3) 配置同步文件夹

```

Welcome to minicom 2.7.1

OPTIONS: I18n
Comp+-----+
Port| A - Download directory : /home/oseasy
    | B - Upload directory  : /home/oseasy
Pres| C - Script directory  :
    | D - Script program   : /bin/bash
    | E - Kermit program   :
    | F - Logging options
    |
    | Change which setting? 
    |-----+
    | | Screen and keyboard |
    | | Save setup as dfl  |
    | | Save setup as..  |
    | | Exit                |
    |-----+

CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | ttyS0

```

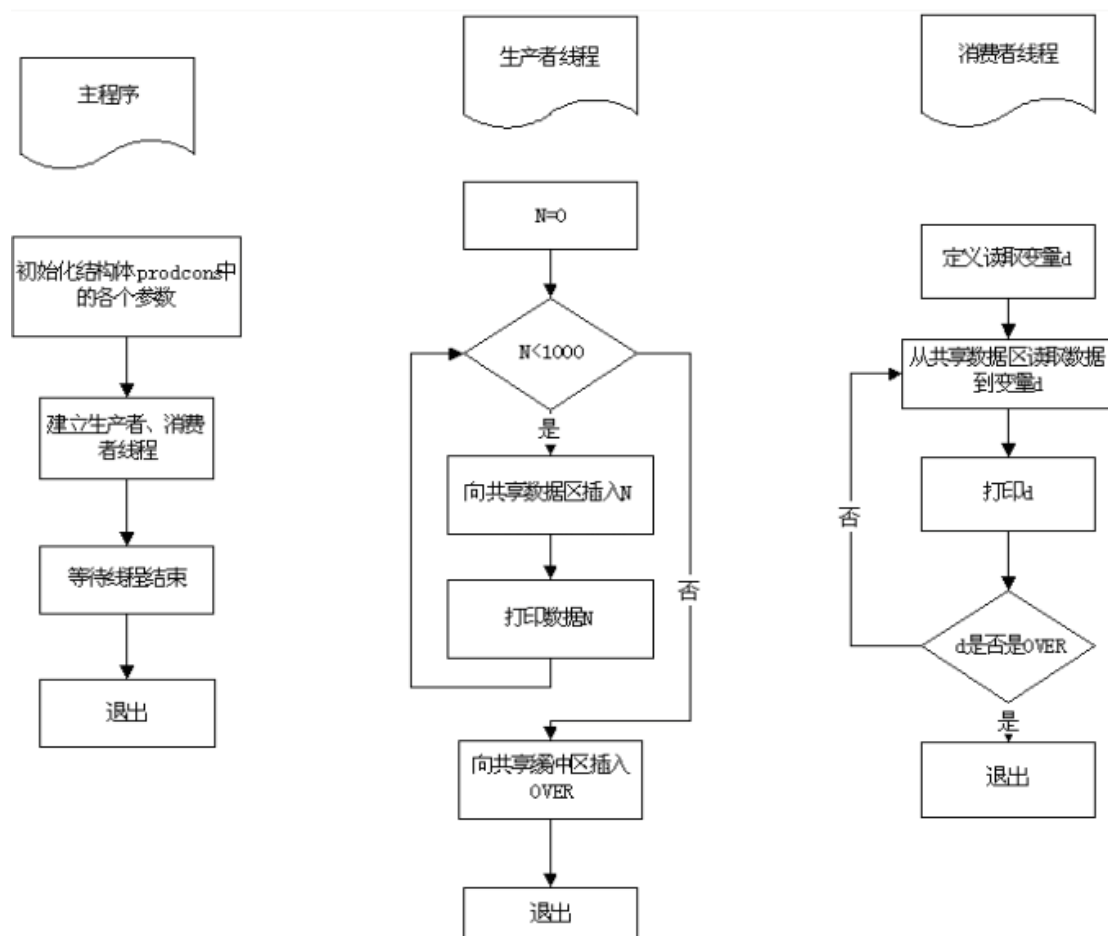
(4) 重启开发板，更新配置

```

[ 3.670000] Failed to init busfreq.
[ 3.670000] s3c-rtc s3c64xx-rtc: setting system clock to 2016-01-01 12:00:01 UTC (1451649601)
[ 3.670000] hotplug_policy_init: initialised with policy : DVFS_NR_BASED_HOTPLUG
[ 3.670000] ALSA device list:
[ 3.670000]   No soundcards found.
[ 3.670000] Freeing init memory: 216K
[ 3.685000] EXT4-fs (mncblkp1): INFO: recovery required on readonly filesystem
[ 3.685000] EXT4-fs (mncblkp1): write access will be enabled during recovery
[ 3.695000] usb 1-2.1: new high-speed USB device number 3 using ssp-ehci
[ 3.720000] EXT4-fs (mncblkp1): recovery complete
[ 3.725000] EXT4-fs (mncblkp1): mounted filesystem with ordered data mode. Opts: (null)
[ 3.730000] EXT4-fs (mncblkp1): re-mounted. Opts: data=ordered
[ 3.805000] usb 1-2.1: New USB device found, idVendor=0424, idProduct=4040
[ 3.805000] usb 1-2.1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[ 3.805000] usb 1-2.1: Product: Ultra Fast Media Reader
[ 3.805000] usb 1-2.1: Manufacturer: Generic
[ 3.805000] usb 1-2.1: SerialNumber: 00000264001
[ 3.815000] scsi0 : usb-storage 1-2.1:1.0
[ 3.900000] usb 1-2.2: new full-speed USB device number 4 using ssp-ehci
[ 4.005000] usb 1-2.2: not running at top speed; connect to a high speed hub
[ 4.075000] usb 1-2.2: New USB device found, idVendor=0846, idProduct=9621
[ 4.075000] usb 1-2.2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[ 4.080000] dm962x: dm_read_reg() 0x29 0x0a
[ 4.080000] dm962x: dm_read_reg() 0x28 0x46
[ 4.080000] dm962x: dm_read_reg() 0x2b 0x96
[ 4.080000] dm962x: dm_read_reg() 0x2c 0x21
[ 4.085000] dm962x: dm_read_reg() 0xF2 0x00
[ 4.090000] dm962x: [Analysis.2] 0xF2, D[7] 0 OK
[ 4.095000] dm962x: [Analysis.2] 0xF2, D[6] 0 OK
[ 4.100000] dm962x: [Analysis.2] 0xF2, D[5] 0 EP1: Empty
[ 4.105000] dm962x: [Analysis.2] 0xF2, D[3] 0 OK
[ 4.110000] dm962x: [Analysis.2] 0xF2, D[2] 0 OK
[ 4.115000] dm962x: [Analysis.2] 0xF2, D[1] 0 OK
[ 4.120000] dm962x: [Analysis.2] 0xF2, D[0] 0 Status: TX buffer 0 pkts
[ 4.130000] dm962x: ethernet MAC address 1c:6f:65:34:51:7e (param data)
[ 4.135000] dm962x: 9620 Mode = 128
[ 4.155000] dm9620 1-2.2:1.0: eth0: register 'dm9620' at usb-spp-ehci-2.2, Davicom DM9620 USB Ethernet, 1c:6f:65:34:51:7e
[ 4.815000] scsi 0:0:0:0: Direct-Access Generic Ultra HS-COMBO 2.01 PQ: 0 ANSI: 0
[ 4.820000] sd 0:0:0:0: Attached scsi generic sgd type 0
[ 4.825000] sd 0:0:0:0: [sda] Attached SCSI removable disk
[01/Jan/2016:04:00:04 +0000] boa: server version Boa/0.94.13
[01/Jan/2016:04:00:04 +0000] boa: server built Dec 30 2010 at 11:18:35.
[01/Jan/2016:04:00:04 +0000] boa: starting server pid=162, port 80

```

4. 实验 2.2——多线程应用程序设计



上图多线程应用程序的程序流程图，实验为著名的生产者—消费者问题模型的实现，主程序中分别启动生产者线程和消费者线程。生产者线程不断顺序地将 0 到 1000 的数字写入共享的循环缓冲区，同时消费者线程不断地从共享的循环缓冲区读取数据。

程序中主要使用的 pthread 多线程 API 如下所示：

- (1) pthread_create() 线程创建函数
- (2) pthread_self() 获取父进程 ID 号
- (3) pthread_exit() 推出指定线程
- (4) pthread_equal() 测试两个线程是否相等
- (5) pthread_join() 等待指定的线程结束
- (6) pthread_mutex_init() 互斥量初始化
- (7) pthread_mutex_destroy() 销毁互斥量
- (8) pthread_mutex_trylock() 再试一次获得对互斥量的锁定(非阻塞)
- (9) pthread_mutex_lock() 锁定互斥量
- (10) pthread_mutex_unlock() 解锁互斥量
- (11) pthread_cond_init() 条件变量的初始化
- (12) pthread_cond_destroy() 销毁条件变量
- (13) pthread_cond_signal() 唤醒线程等待条件变量
- (14) pthread_cond_wait() 等待条件变量(阻塞)
- (15) pthread_cond_timewait() 在指定时间到达前等待条件变量

实验代码具体如下

```
1. // 引入所需的头文件
2. #include <stdio.h>
3. #include <stdlib.h>
4. #include <time.h>
5. #include "pthread.h"
6.
7. // 定义缓冲区大小
8. #define BUFFER_SIZE 16
9.
10. // 定义一个结构体，用于表示生产者和消费者共享的缓冲区
11. struct prodcons
12. {
13.     int buffer[BUFFER_SIZE]; // 实际的数据缓冲区
14.     pthread_mutex_t lock; // 互斥锁，确保对缓冲区的独占访问
15.     int readpos, writepos; // 读写位置
16.     pthread_cond_t notempty; // 条件变量，当缓冲区不为空时发出信号
17.     pthread_cond_t notfull; // 条件变量，当缓冲区不满时发出信号
18. };
19.
20. // 初始化缓冲区
21. void init(struct prodcons *b)
22. {
23.     pthread_mutex_init(&b->lock, NULL); // 初始化互斥锁
24.     pthread_cond_init(&b->notempty, NULL); // 初始化条件变量
25.     pthread_cond_init(&b->notfull, NULL); // 初始化条件变量
26.     b->readpos = 0; // 初始化读位置
27.     b->writepos = 0; // 初始化写位置
28. }
29.
30. // 向缓冲区中添加数据
31. void put(struct prodcons *b, int data)
32. {
33.     pthread_mutex_lock(&b->lock); // 加锁
34.     // 当缓冲区满时，等待
35.     while ((b->writepos + 1) % BUFFER_SIZE == b->readpos)
36.     {
37.         printf("wait for not full\n");
38.         pthread_cond_wait(&b->notfull, &b->lock); // 等待“notfull”条件成
           立
39.     }
40.     // 写入数据并更新写位置
41.     b->buffer[b->writepos] = data;
```

```

42. b->writepos++;
43. if (b->writepos >= BUFFER_SIZE)
44.     b->writepos = 0;
45. // 发出“notempty”信号
46. pthread_cond_signal(&b->notempty);
47. pthread_mutex_unlock(&b->lock); // 解锁
48. }
49.
50. // 从缓冲区中获取数据
51. int get(struct prodcons *b)
52. {
53.     int data;
54.     pthread_mutex_lock(&b->lock); // 加锁
55.     // 当缓冲区空时，等待
56.     while (b->writepos == b->readpos)
57.     {
58.         printf("wait for not empty\n");
59.         pthread_cond_wait(&b->notempty, &b->lock); // 等待“notempty”条件
            成立
60.     }
61.     // 读取数据并更新读位置
62.     data = b->buffer[b->readpos];
63.     b->readpos++;
64.     if (b->readpos >= BUFFER_SIZE)
65.         b->readpos = 0;
66.     // 发出“notfull”信号
67.     pthread_cond_signal(&b->notfull);
68.     pthread_mutex_unlock(&b->lock); // 解锁
69.     return data; // 返回读取的数据
70. }
71.
72. // 主函数
73. int main(void)
74. {
75.     pthread_t th_a, th_b; // 定义线程
76.     void *retval;
77.     init(&buffer); // 初始化缓冲区
78.     pthread_create(&th_a, NULL, producer, 0); // 创建生产者线程
79.     pthread_create(&th_b, NULL, consumer, 0); // 创建消费者线程
80.     // 等待生产者和消费者线程结束
81.     pthread_join(th_a, &retval);
82.     pthread_join(th_b, &retval);
83.     return 0;

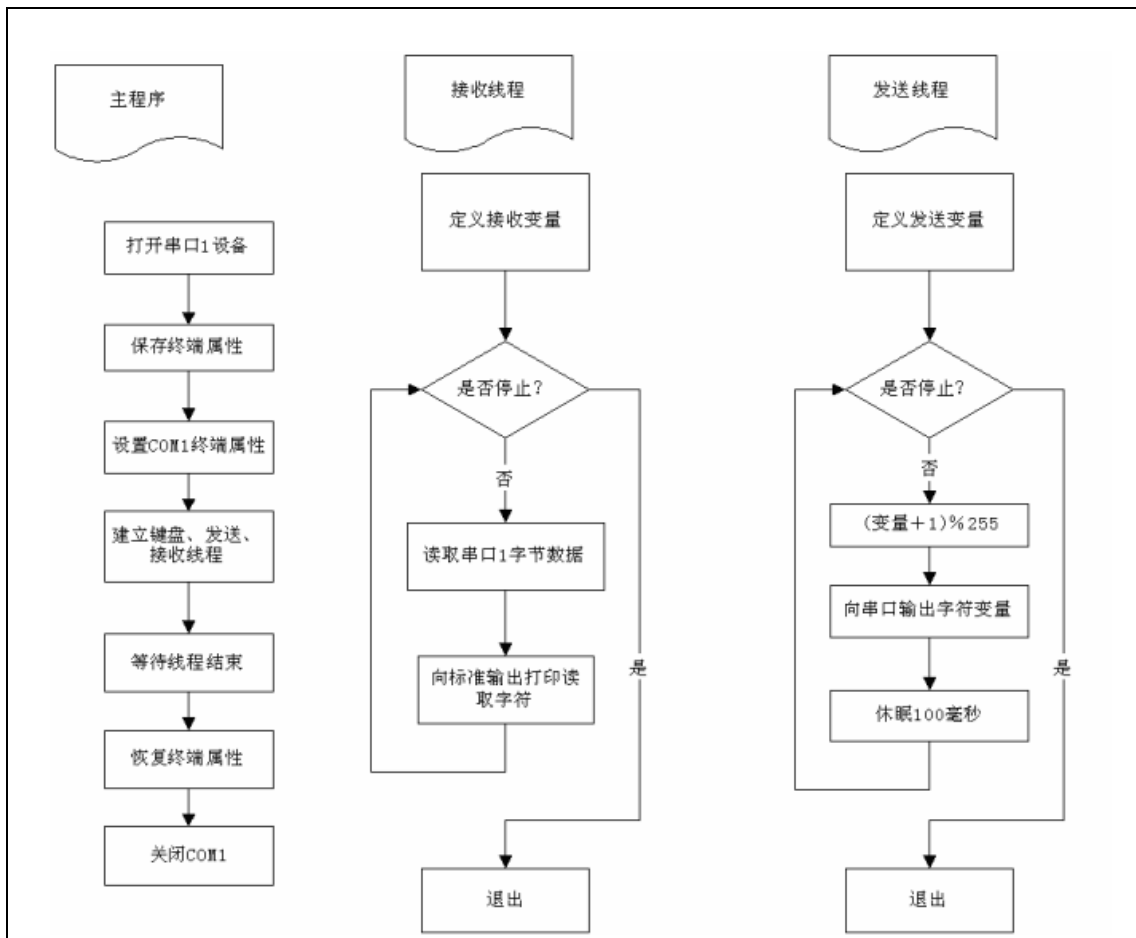
```

```
84. }
```

思考题 1 加入新的线程用于处理键盘输入，并在按键为 ESC 时终止所有线程,将下列代码放入 getChar 函数中，getChar()函数在 mian 函数中创建线程。

```
1. void *input(void *data)
2. {
3.     char ch;
4.     ch = getchar();
5.     if (ch == 0x1b)
6.     {
7.         exit(0);
8.     }
9.     return;
10.}
11.int main(void)
12.{
13.    pthread_t a, b, c;
14.    void *retval;
15.    pthread_create(&a, NULL, producer, 0);
16.    pthread_create(&b, NULL, consumer, 0);
17.    pthread_create(&c, NULL, input, 0);
18.    pthread_join(&a, retval);
19.    pthread_join(&b, retval);
20.    pthread_join(&c, retval);
21.    return 0;
22.}
```

5. 实验 2.3——串口通信应用程序设计



上图为串行端口应用程序的控制流程图，设备运行程序后，主函数打开串口，创建接收线程和发送线程，用于收发信息；两个设备运行同样的程序实现串口通讯。程序详细代码如下：

```

1. // 包含了一些必要的头文件，这些头文件提供了串行通信和多线程所需的函数和
   数据类型
2. #include <termios.h>
3. #include <stdio.h>
4. #include <unistd.h>
5. #include <fcntl.h>
6. #include <sys/signal.h>
7. #include <pthread.h>
8.
9. // 定义了一些常量，包括波特率、串口设备文件路径、退出程序的键盘输入等
10. #define BAUDRATE B115200
11. #define COM1 "/dev/ttyS0"
12. #define COM2 "/dev/ttyS1"
13. #define ENDMINITERM 27 /* ESC to quit miniterm */
14. #define FALSE 0
15. #define TRUE 1
16.
  
```

```

17. // 定义了两个全局变量，用于控制程序的运行和串口的文件描述符
18. volatile int STOP = FALSE;
19. volatile int fd;
20.
21. // 定义了一个信号处理函数，当接收到SIGCHLD信号时，会打印一条消息并设置
    STOP 为 TRUE，使程序停止运行
22. void child_handler(int s)
23. {
24.     printf("stop!!!\n");
25.     STOP = TRUE;
26. }
27.
28. // 定义了一个线程函数，这个线程会不断地从键盘读取输入，当读取到定义的退
    出字符时，会设置STOP 为 TRUE，使程序停止运行
29. void *keyboard(void *data)
30. {
31.     int c;
32.     for (;;)
33.     {
34.         c = getchar();
35.         if (c == ENDMINITERM)
36.         {
37.             STOP = TRUE;
38.             break;
39.         }
40.     }
41.     return NULL;
42. }
43.
44. // 定义了一个线程函数，这个线程会不断地从串口读取数据，并将读取到的数据
    写入到标准输出
45. void *receive(void *data)
46. {
47.     int c;
48.     printf("read modem\n");
49.     while (STOP == FALSE)
50.     {
51.         read(fd, &c, 1); /* com port */
52.         write(1, &c, 1); /* stdout */
53.     }
54.     printf("exit from reading modem\n");
55.     return NULL;
56. }
57.

```

```

58. // 定义了一个线程函数，这个线程会不断地向串口写入数据，数据是一个不断增加的字符
59. void *send(void *data)
60. {
61.     int c = '0';
62.     printf("send data\n");
63.     while (STOP == FALSE) /* modem input handler */
64.     {
65.         c++;
66.         c %= 255;
67.         write(fd, &c, 1); /* stdout */
68.         usleep(100000);
69.     }
70.     return NULL; /* wait for child to die or it will become a zombie */
71. }
72.
73. // 主函数，程序的入口点
74. int main(int argc, char **argv)
75. {
76.     // 定义了一些变量，包括termios 结构体用于设置串口，sigaction 结构体用于设置信号处理函数，pthread_t 用于创建线程
77.     struct termios oldtio, newtio, oldstdtio, newstdtio;
78.     struct sigaction sa;
79.     int ok;
80.     pthread_t th_a, th_b, th_c;
81.     void *retval;
82.
83.     // 根据命令行参数来打开不同的串口设备文件
84.     if (argc > 1)
85.         fd = open(COM2, O_RDWR);
86.     else
87.         fd = open(COM1, O_RDWR); //| O_NOCTTY |O_NONBLOCK);
88.
89.     // 如果打开文件失败，则打印错误信息并退出程序
90.     if (fd < 0)
91.     {
92.         error(COM1);
93.         exit(-1);
94.     }
95.
96.     // 获取并保存当前的串口和标准输入的设置
97.     tcgetattr(0, &oldstdtio);
98.     tcgetattr(fd, &oldtio); /* save current modem settings */

```

```
99.
100. // 获取新的串口和标准输入的设置
101. tcgetattr(fd, &newsttio); /* get working stdtio */
102.
103. // 设置新的串口的各项参数
104. newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD; /*c
    trol flag*/
105. newtio.c_iflag = IGNPAR;          /*input flag*/
106. newtio.c_oflag = 0;              /*output flag*/
107. newtio.c_lflag = 0;
108. newtio.c_cc[VMIN] = 1;
109. newtio.c_cc[VTIME] = 0;
110.
111. // 清空串口的输入和输出缓冲区, 并设置新的串口参数
112. tcflush(fd, TCIFLUSH);
113. tcsetattr(fd, TCSANOW, &newtio); /*set attrib*/
114.
115. // 设置信号处理函数
116. sa.sa_handler = child_handler;
117. sa.sa_flags = 0;
118. sigaction(SIGCHLD, &sa, NULL); /* handle dying child */
119.
120. // 创建三个线程, 分别用于处理键盘输入、串口接收和串口发送
121. pthread_create(&th_a, NULL, keyboard, 0);
122. pthread_create(&th_b, NULL, receive, 0);
123. pthread_create(&th_c, NULL, send, 0);
124.
125. // 等待三个线程结束
126. pthread_join(th_a, &retval);
127. pthread_join(th_b, &retval);
128. pthread_join(th_c, &retval);
129.
130. // 恢复原来的串口和标准输入的设置
131. tcsetattr(fd, TCSANOW, &oldtio); /* restore old modem setings
    */
132. tcsetattr(0, TCSANOW, &oldsttio); /* restore old tty setings *
    /
133.
134. // 关闭串口设备文件
135. close(fd);
136.
137. // 退出程序
138. exit(0);
```

6. 编译文件，将目标文件使用 minicom 传输至开发板

使用交叉编译环境编译程序

```
# arm-linux-gcc pthread.c -o pthread -lpthread
# arm-linux-gcc term.c -o term -lpthread
```

```
root@PC05:/home/oseasy# arm-linux-gcc pthread.c -o pthread -lpthread
root@PC05:/home/oseasy# arm-linux-gcc term.c -o term -lpthread
arm-linux-gcc: error: -lpthread: No such file or directory
root@PC05:/home/oseasy# arm-linux-gcc term.c -o term -lpthread
term.c: In function 'main':
term.c:83:2: warning: incompatible implicit declaration of built-in function 'exit' [enabled by default]
term.c:111:5: warning: incompatible implicit declaration of built-in function 'exit' [enabled by default]
root@PC05:/home/oseasy#
```

使用 minicom 在 zmodern 传输至开发板上

```
root@PC05:/home/oseasy
8.520000] asoc: wm8960-hifi <-> samsung-l2s.4 mapping ok
-----[Select one or more files for upload]-----
8.610000] usbcore: registered new interface [Directory: /home/oseasy]
8.610000] Compat-drivers backport release [..]
8.615000] Backport based on linux-stable. [..cache]
8.640000] compat.git: linux-stable.git [..config]
8.640000] cfg80211: calling CRDA to update [..gnupp]
8.695000] usbcore: registered new interface [..local]
8.715000] usbcore: registered new interface [..mozilla]
8.735000] usbcore: registered new interface [..pk]
8.745000] usbcore: registered new interface [..ssh]
8.765000] libertas_sdio: libertas sdio dri [..thunderbird]
8.765000] libertas_sdio: Copyright Pierre [opt]
8.795000] link_reset() speed: 10 duplex: 1 [u]
8.795000] IPv6: ADDRCONF(NETDEV_UP): eth0 [snap]
Please press Enter to activate this console. [u]
root@FriendlyARM /)# ls
24cXX.o      hello      servi
Makefile    hhh        servi
_new        home      .bash_history
_s.out      i2c       shar
adc-test    led        .bashrc
arm-udp     led-player term
arm-udptalk led1      .sudo_as_admin_successful
arm-udptalkZK ltb      test
bin         linuxrc   test
button     lost+found th
button_test mnt      tiny
button_test_opt opt      tiny
data       p        tiny
dev        proc     tmp
eeprog.o   pthread udisk
etc        pthread.o udpt
fa-network-service pwm      usr
float-test pwm_test var
float-test.soft pwm_test www
float-test.vfp root     x?
fwq       sbin    x??hello?7816
globalmem.ko sdcard
root@FriendlyARM /)#
```

```
_new        home      service.asm
_s.out      i2c       share
adc-test    led       sys
arm-udp     led-player term
arm-udptalk led1      test
arm-udptalkZK ltb      test.sh
bin         linuxrc   test.sh-
button     lost+found th
button_test mnt      tiny4412_buttons.ko
button_test_opt opt      tiny4412_leds.ko
data       p        tiny4412_pwm.ko
dev        proc     tmp
eeprog.o   pthread udisk
etc        pthread.o udptalk.asm
fa-network-service pwm      usr
float-test pwm_test var
float-test.soft pwm_test www
float-test.vfp root     x?
fwq       sbin    x??hello?7816
globalmem.ko sdcard
[roo@FriendlyARM /)# ls
24cXX.o      hello      serve.asm
Makefile    hhh        server.asm
_new        home      service.asm
_s.out      i2c       share
_adc-test    led       sys
_arm-udp     led-player term
_arm-udptalk led1      test
_arm-udptalkZK ltb      test.sh
_bin         linuxrc   test.sh-
_button     lost+found th
_button_test mnt      tiny4412_buttons.ko
_button_test_opt opt      tiny4412_leds.ko
_data       p        tiny4412_pwm.ko
_dev        proc     tmp
_eeprog.o   pthread udisk
_etc        pthread.o udptalk.asm
_fa-network-service pwm      usr
_float-test pwm_test var
_float-test.soft pwm_test www
_float-test.vfp root     x?
_fwq       sbin    x??hello?7816
_globalmem.ko sdcard
```

运行可执行文件

```
[root@FriendlyARM ~]# ./pthread
wait for not empty
put-->0
put-->1
put-->2
put-->3
put-->4
put-->5
put-->6
put-->7
put-->8
put-->9
put-->10
put-->11
put-->12
put-->13
put-->14
put-->15
wait for not full
8-->get
1-->get
2-->get
3-->get
4-->get
5-->get
6-->get
7-->get
8-->get
```

```
wait for not empty
put-->16
put-->17
put-->18
put-->19
put-->20
put-->21
put-->22
put-->23
put-->24
put-->25
put-->26
put-->27
put-->28
put-->29
put-->30
wait for not full
15-->get
16-->get
17-->get
18-->get
19-->get
20-->get
21-->get
22-->get
23-->get
24-->get
25-->get
26-->get
27-->get
28-->get
put-->31
put-->32
put-->33
put-->34
put-->35
put-->36
put-->37
```

```
wait for not full
976-->get
put-->993
wait for not full
977-->get
put-->994
wait for not full
978-->get
put-->995
wait for not full
979-->get
put-->996
wait for not full
980-->get
put-->997
wait for not full
981-->get
put-->998
wait for not full
982-->get
put-->999
wait for not full
983-->get
wait for not full
984-->get
producer stopped!
985-->get
986-->get
987-->get
988-->get
989-->get
990-->get
991-->get
992-->get
993-->get
994-->get
995-->get
996-->get
997-->get
998-->get
999-->get
consumer stopped!
```

```
float-test          read
globalmem.ko       readpos
hello              readpos++
[root@FriendlyARM /]# ./term
send data
read modem
```

思考题运行结果

```
wait for not full
308-->get
  put-->495
wait for not full
309-->get
  put-->496
wait for not full
310-->get
  put-->497
wait for not full
311-->get
  put-->498
wait for not full
312-->get
^[
Stop,get ESC
```

实验环境（含主要设计设备，器材，软件等）

硬件：Ubuntu 主机(硬盘 40G 以上, 内存大于 128M)、Tiny4412 开发板

软件：minicom、gedit

遇到的问题

1. 编译问题

```
root@PC05:/home/oseasy# arm-linux-gcc pthread.c -o pthread -lpthread
arm-linux-gcc: error: -lpthread: No such file or directory
root@PC05:/home/oseasy# arm-linux-gcc pthread.c -o pthread
/tmp/cc0pb11b.o: In function 'main':
pthread.c:(.text+0x348): undefined reference to 'pthread_create'
pthread.c:(.text+0x360): undefined reference to 'pthread_create'
pthread.c:(.text+0x374): undefined reference to 'pthread_join'
pthread.c:(.text+0x388): undefined reference to 'pthread_join'
collect2: ld returned 1 exit status
root@PC05:/home/oseasy# arm-linux-gcc pthread.c -o pthread -lpthread
arm-linux-gcc: error: -lpthread: No such file or directory
root@PC05:/home/oseasy# arm-linux-gcc pthread.c -o pthread -l pthread
arm-linux-gcc: error: -l: No such file or directory
arm-linux-gcc: error: pthread: No such file or directory
root@PC05:/home/oseasy# arm-linux-gcc pthread.c -o pthread -lpthread
arm-linux-gcc: error: -lpthread: No such file or directory
root@PC05:/home/oseasy# arm-linux-gcc pthread.c -o pthread -lpthread
arm-linux-gcc: error: -lpthread: No such file or directory
root@PC05:/home/oseasy#
```

这次是第一次使用指令编译 cpp 文件，并且带有库的编译，所以学习了一下相关指令。GCC 在编译时如果需要链接外部库文件，可以在命令行中使用 `-l` 选项指定要链接的库名。一般的命令格式为：

```
1. gcc source_file.c -l[library_name]
```

其中：

- `source_file.c` 是要编译的 C 源文件名
- `-l` 表示链接库文件
- `library_name` 是需要链接的库名，不需要带“lib”前缀和库文件扩展名(如 .a、.so)

如果一次需要链接多个库，只需要将库名用空格隔开：

```
1. gcc mycode.c -lpthread -lm -lmylib
```

需要注意的是，有些库之间存在依赖关系，链接顺序可能会影响最终编译结果，一般先链接主要依赖库，再链接其他库更为安全。

此外，`-l` 选项只是简化了链接库的命令，完整的命令格式是使用 `-L` 和 `-l` 两个选项，如：

```
1. gcc mycode.c -L/path/to/libs -lmylib
```

其中 `-L` 指定了库文件的路径。

了解了这些后发现是指令格式错误，改正后编译成功

```
root@PC05:/home/oseasy# arm-linux-gcc pthread.c -o pthread -lpthread
root@PC05:/home/oseasy# arm-linux-gcc term.c -o term -lpthread
arm-linux-gcc: error: -lpthread: No such file or directory
root@PC05:/home/oseasy# arm-linux-gcc term.c -o term -lpthread
term.c: In function 'main':
term.c:83:2: warning: incompatible implicit declaration of built-in function 'exit' [enabled by default]
term.c:111:5: warning: incompatible implicit declaration of built-in function 'exit' [enabled by default]
root@PC05:/home/oseasy#
```


实验结果与总结：

实验结果：

- 成功配置了 ARM 交叉编译环境, 可以使用 arm-linux-gcc 进行交叉编译。
- 使用 pthread 库编写的多线程生产者-消费者程序能正确运行, 线程之间通过共享缓冲区、互斥量和条件变量实现了同步, 并在按键为 ESC 时终止所有线程(直接使用 getchar 读取输入的字符)。
- 串口通信程序能在两个开发板之间实现数据收发。
- 使用 minicom 工具能够便捷地在主机和开发板之间传输文件。

总结：

通过本次实验, 我掌握了以下几个方面知识和技能：

- ARM Linux 交叉编译环境的配置及使用方法。
- 使用 pthread 库进行多线程编程, 理解了线程同步的概念和实现方式。
- 串行端口程序的设计原理, 学会了在 Linux 下使用 termios 库进行串口编程。
- 读取键盘的输入的数据 (读取 esc 等按键的符号使用 getchar)
- 会使用 minicom 工具在主机和开发板之间传输文件。
- 熟悉了 Tiny4412 开发板的使用方法和开发环境。

这些技能为我后续的嵌入式 Linux 系统开发打下了基础。本实验也加深了我对并发编程以及设备驱动程序的理解。在实验过程中还遇到了一些问题, 如交叉编译环境变量设置错误、串口设备权限问题等, 通过查阅资料和老师的指导最终都得到了解决, 锻炼了我的问题分析和解决能力。