

华中科技大学

课程实验报告

课程名称: 嵌入式操作系统原理

专业班级: 物联网 1601

学 号: U201614897

姓 名: 潘越

指导教师: 石柯

报告日期: 2018 年 5 月 8 日

计算机科学与技术学院

目录

PART I: 操作系统实验	1
实验一 进程控制	1
1.1 实验目的.....	1
1.2 实验内容.....	1
1.3 程序设计.....	1
1.3.1 总体模式	1
1.3.2 详细设计	2
1.4 实验过程.....	4
1.4.1 开发环境	4
1.4.2 实验步骤	4
1.5 实验结果与分析	4
1.6 补充实验—采用命名管道 FIFO 完成实验一	5
1.6 心得体会与总结	6
实验二 线程同步与通信	8
2.1 实验目的.....	8
2.2 实验内容.....	8
2.3 程序设计.....	8
2.3.1 总体模式	8
2.3.2 详细设计	9
2.4 实验过程.....	9
2.4.1 开发环境	9
2.4.2 实验步骤	9
2.5 实验结果与分析	10
2.6 心得体会与总结	10
实验三 共享内存与进程同步	12
3.1 实验目的.....	12
3.2 实验内容.....	12
3.3 程序设计.....	12
3.3.1 总体模式	12
3.3.2 详细设计	13
3.4 实验过程.....	15

3.4.1 开发环境	15
3.4.2 实验步骤	15
3.5 实验结果与分析	16
3.6 心得体会与总结	16
实验四 LINUX 文件目录	18
4.1 实验目的	18
4.2 实验内容	18
4.3 程序设计	18
4.3.1 总体模式	18
4.3.2 详细设计	18
4.4 实验过程	20
4.4.1 开发环境	20
4.4.2 实验步骤	20
4.5 实验结果与分析	21
4.6 心得体会与总结	21
PART II: TINYOS 实验	23
实验一 二进制 0-7 亮灯程序烧录	23
5.1 实验目的	23
5.2 实验内容	23
5.3 实验过程	23
5.4 实验结果与分析	24
5.5 心得体会与总结	25
实验二 TINYOS SPLIT-PHASE 过程探讨	26
6.1 实验目的	26
6.2 实验内容	26
6.3 实验过程	26
6.4 实验结果与分析	27
6.5 心得体会与总结	27
实验三 TINYOS 感知与通信实验	28
7.1 实验目的	28
7.2 实验内容	28
7.3 实验设计	29

7.4 实验过程.....	31
7.5 实验结果与分析	32
7.6 心得体会与总结	32
PART III: 附录	34
附录 A 实验中用到的库文件说明.....	34
附录 B 在 ARCHLINUX 环境下配置 TINYOS 开发环境.....	42
附录 C 操作系统实验代码	49
附录 D TINYOS 实验代码.....	67
参考文献	78

PART I: 操作系统实验

实验一 进程控制

1.1 实验目的

1. 加深对进程的理解, 进一步认识并发执行的实质;
2. 分析进程争用资源现象, 学习解决进程互斥的方法;
3. 掌握 Linux 进程基本控制;
4. 掌握 Linux 系统中的软中断和管道通信。

1.2 实验内容

编写程序, 演示多进程并发执行和进程软中断、管道通信。

- 父进程使用系统调用 `pipe()` 建立一个管道, 然后使用系统调用 `fork()` 创建两个子进程, 子进程 1 和子进程 2;
- 子进程 1 每隔 1 秒通过管道向子进程 2 发送数据:
I send you x times. (x 初值为 1, 每次发送后做加一操作)
子进程 2 从管道读出信息, 并显示在屏幕上。
- 父进程用系统调用 `signal()` 捕捉来自键盘的中断信号(即按 Ctrl+C 键);
当捕捉到中断信号后, 父进程用系统调用 `Kill()` 向两个子进程发出信号, 子进程捕捉到信号后分别输出下列信息后终止:
Child Process 1 is Killed by Parent!
Child Process 2 is Killed by Parent!
- 父进程等待两个子进程终止后, 释放管道并输出如下的信息后终止
Parent Process is Killed!

1.3 程序设计

1.3.1 总体模式

实验一分为三个部分, 父进程首先创建管道, 然后通过系统调用 `fork()` 创建两个子进程 1 和 2, 子进程 1 通过管道向子进程 2 发送数据, 子进程 2 通过管

道接收数据并将数据传递至标准输出。

实验一的设计模式图如下：

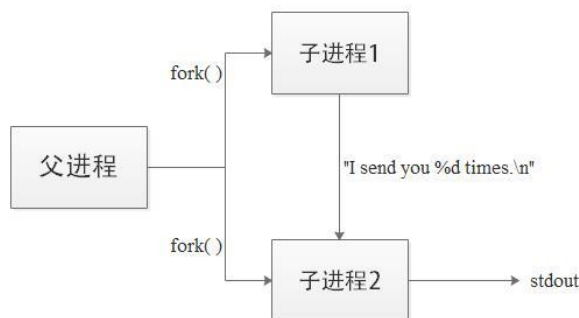


图 1-1 实验一的设计模式图

1.3.2 详细设计

实验一使用无名管道 `pipe()`，信号控制函数使用 `signal()`。

主进程首先通过 `signal()` 设置对 `SIGINT` 信号的处理函数 `sigint_handler`，并通过 `pipe()` 创建无名管道。创建管道成功之后，通过 `fork()` 创建两个子进程 1 和 2。在完成对两个进程的创建之后，关闭父进程管道的写入端和读取端，之后通过 `waitpid()` 来等待两个子进程的结束。

父进程的流程图如下：

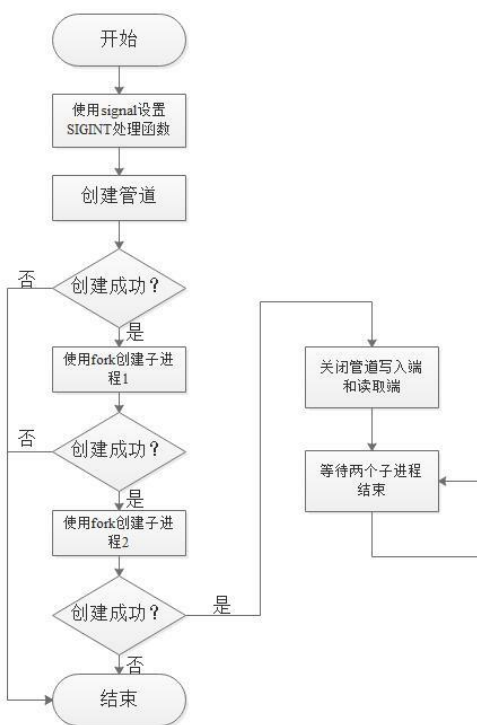


图 1-2 父进程流程图

对于子进程 1，首先要通过 `signal()` 忽略 `SIGINT` 信号，然后再设置对 `SIGUSR1` 信号的处理函数 `sigusr_handler`，并关闭管道的读取端。接下来进入死循环，每隔一秒向管道内写入一行数据，`count` 变量用来计数。

对于子进程 2，同样首先通过 `signal()` 忽略 `SIGINT` 信号，然后再设置对 `SIGUSR2` 信号的处理函数 `sigusr_handler`，并关闭管道的写入端。接下来进入死循环，不断接收管道内的数据，并将数据输出到标准输出。

子进程 1 和 2 的流程图如下：

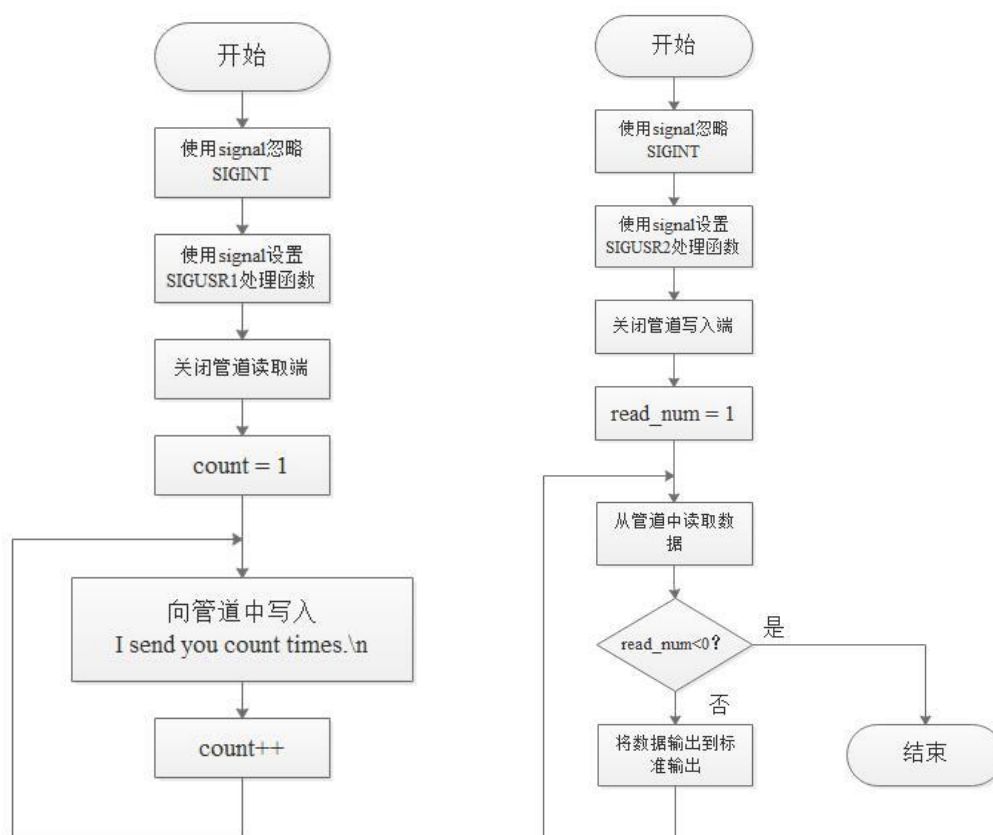


图 1-3 子进程 1 和 2 的流程图

对于处理函数 `sigint_handler` 和 `sigusr_handler`, `sigint_handler` 只需要通过 `kill()` 调用向两个子进程发送信号 `SIGUSR1` 和 `SIGUSR2`, `sigusr_handler` 则通过相应的信号打印进程终止结果，并通过 `exit()` 调用结束进程。

1.4 实验过程

1.4.1 开发环境

操作系统：4.15.15-1-ARCH x86_64 GNU/Linux

编译器：gcc 7.3.1

自动编译工具：GNU Make 4.2.1、cmake version 3.3.2

编辑器：Visual Studio Code

资源监视器：htop 2.2.0

1.4.2 实验步骤

首先编写源码，编译过程使用 CMake 构建 Makefile 来自动编译，编译完成后得到可执行文件 pipe，在终端下执行 pipe 若干秒后按下 Ctrl+C，观察程序的输出结果如下图：

```

panyue@Saltedfish ~/code/my_github/HUST-OperatingSystem-Labs master ./pipe
I send you 1 times.
I send you 2 times.
I send you 3 times.
I send you 4 times.
I send you 5 times.
I send you 6 times.
^C
Child Process 2 is Killed by Parent!
Child Process 1 is Killed by Parent!
Parent Process is Killed!
    
```

图 1-4 实验一输出结果

观察到在两个子进程结束之后父进程才结束，满足实验的要求。

1.5 实验结果与分析

在程序运行的过程中，我们采用 htop 对系统进程进行分析，开启树状图模式，找到 ./pipe 进程，得到如下图所示结果：

```

1  [|||||] 6.6% 5 [|||||] 37.1%
2  [|||||] 8.4% 6 [|||||] 11.0%
3  [|||||] 30.3% 7 [|||||] 11.2%
4  [|||||] 9.2% 8 [|||||] 5.3%
Mem[|||||] 4.30G/15.6G Tasks: 173, 724 thr; 1 running
Swp[|||||] 0K/8.00G Load average: 1.81 1.74 1.55
Uptime: 00:31:33

PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
1541 panyue 20 0 4140 772 708 S 0.0 0.0 0:00.00 ./pipe
1544 panyue 20 0 4140 72 0 S 0.0 0.0 0:00.00 ./pipe
1542 panyue 20 0 4140 72 0 S 0.0 0.0 0:00.00 ./pipe
    
```

图 1-5 实验一结果分析

根据图我们可以很明显看出主进程和两个子进程之间的关系，验证实验结果。

我们再使用 htop 发送信号尝试，对子进程发送 SIGINT 信号：

The screenshot shows the htop interface. At the top, there are process statistics for PID 1, 2, 3, 4, 5, 6, 7, and 8. Below this, a table lists processes with columns: PID, USER, PRI, NI, VIRT, RES, SHR, S, CPU%, MEM%, TIME+, and Command. The table shows three processes with PID 16909, 16911, and 16910, all running under the user 'panyue'. The command for all three is './pipe'. Below the table, a 'Send signal:' menu is open, showing a list of signals from 0 to 9. Signal 2, SIGINT, is highlighted.

Send signal:	PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
0 Cancel	16909	panyue	20	0	4140	752	688	S	0.0	0.0	0:00.00	./pipe
1 SIGHUP	16911	panyue	20	0	4140	72	0	S	0.0	0.0	0:00.00	./pipe
2 SIGINT	16910	panyue	20	0	4140	72	0	S	0.0	0.0	0:00.00	./pipe

图 1-6 发送 SIGINT 信号

结果得到两个子进程仍然存在，说明子进程正确忽略了 SIGINT 信号。

我们再对子进程 2 发送 SIGUSR2 信号尝试，得到如下结果：

The screenshot shows the output of a program in a terminal. It displays a series of messages: 'I send you 13 times.', 'I send you 14 times.', 'I send you 15 times.', 'I send you 16 times.', 'I send you 17 times.', 'I send you 18 times.', 'I send you 19 times.', and 'I send you 20 times.'. The final line of output is 'Child Process 2 is Killed by Parent!'.

图 1-6 发送 SIGUSR2 信号后的结果

子进程 2 接收到了 SIGUSR2 信号并且终止，打印输出。

综上，根据发送信号验证的结果，实验一的行为与预期的结果一直，达到了实验的要求。

1.6 补充实验—采用命名管道 FIFO 完成实验一

使用命名管道基本不需要更改过多内容，设计模式和 pipe 一样，根据源码编译得到可执行文件 fifo，在终端下执行 fifo 若干秒后按下 Ctrl+C，观察程序的输出结果如下图：

```

panyue@Saltedfish ~/code/my_github/HUST-OperatingSystem-Labs  master  ./fifo
I send you 1 times.
I send you 2 times.
I send you 3 times.
I send you 4 times.
I send you 5 times.
^C
Child Process 1 is Killed by Parent!
Child Process 2 is Killed by Parent!
Parent Process is Killed!

```

图 1-7 FIFO 输出结果

执行 fifo 程序后，我们会发现源目录下多出了一个文件：lab01_fifo，执行 ls -l 后查看该文件的信息，得到：

```

panyue@Saltedfish ~/code/my_github/HUST-OperatingSystem-Labs  master  ls -l
总用量 352
drwxr-xr-x  2 panyue panyue  4096 4月  9 13:58 Assignments
drwxr-xr-x  2 panyue panyue  4096 4月  3 20:45 bin
drwxr-xr-x  5 panyue panyue  4096 4月 15 18:38 build
-rw-r--r--  1 panyue panyue 12866 4月  1 13:14 CMakeCache.txt
drwxr-xr-x 10 panyue panyue  4096 5月 12 19:39 CMakeFiles
-rw-r--r--  1 panyue panyue  1746 4月  1 13:14 cmake_install.cmake
-rw-r--r--  1 panyue panyue   588 4月 16 10:09 CMakeLists.txt
-rwxr-xr-x  1 panyue panyue 13912 4月 17 13:59 dir
-rwxr-xr-x  1 panyue panyue 13536 4月 10 21:59 fifo
-rwxr-xr-x  1 panyue panyue 13816 5月 12 19:39 filecp
prw-----  1 panyue panyue    0 5月 12 20:26 lab01_fifo
drwxr-xr-x  2 panyue panyue  4096 4月  1 20:11 lab01-Pipe
drwxr-xr-x  2 panyue panyue  4096 4月  3 16:04 lab02-Sync

```

图 1-7 FIFO 文件属性

我们可以看到，显示文件类型和权限的字符串序列的第一位为 p，正是管道文件的标识。

1.6 心得体会与总结

本次实验作为 OS 第一次实验，内容非常简单，主要练习管道和信号处理函数的使用。实验中需要注意的地方有两个，一个是关闭未使用的管道文件描述符的问题，防止产生阻塞。另一个是在子进程中设置对 SIGINT 信号的忽略，少了这一步会导致子进程被 Ctrl+C 终止。

课程内要求的部分没什么难度，我在阅读 *The Linux Programming Interface* 第 44 章后，更加深入理解了管道的相关细节，理解了基于管道的 C/S 模型，并且自己实现了命名管道 FIFO 版本的实验一，并且测试了用管道来进程进程同步的操作。在阅读 *Computer Systems: A Programmer's Perspective, 3/E* 第 8 章之后，对 Linux 的异常控制流和信号处理有了更多的理解，同时也学会了比 signal 功能更强大的 sigemptyset() 等一系列信号控制 API。

总而言之，在实验部分还是应该多看一些业界经典书籍，这样才能学到更多

东西。

实验二 线程同步与通信

2.1 实验目的

1. 掌握 Linux 下线程的概念；
2. 了解 Linux 线程同步与通信的主要机制；
3. 通过信号灯操作实现线程间的同步与互斥。

2.2 实验内容

通过 Linux 多线程与信号灯机制，设计并实现计算机线程与 I/O 线程共享缓冲区的同步与通信。

程序要求：两个线程，共享公共变量 a

- 线程 1 负责计算 (1 到 100 的累加，每次加一个数)
- 线程 2 负责打印 (输出累加的中间结果)

2.3 程序设计

2.3.1 总体模式

实验二通过父进程创建两个线程，一个 compute 线程负责计算，另一个 print 线程负责输出结果，两个线程合理利用 P、V 操作控制好进程同步。

实验二的设计模式图如下：

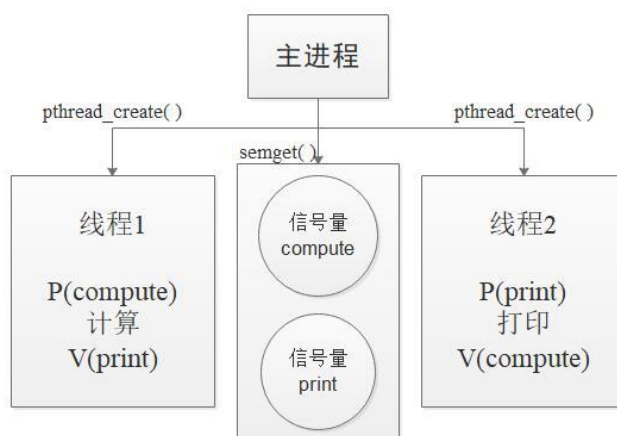


图 2-1 实验二的设计模式图

2.3.2 详细设计

由于实验二需要使用信号量的PV操作,所以需要先封装PV操作的API函数,具体设计在2.6节中再详细说明,这里略去。

实验二主进程首先用 `semget()` 创建一组两个信号量并初始化,再用 `pthread_create()` 创建两个线程分别执行计算任务和打印任务,用 `thread_join()` 函来等待线程的结束,最后销毁信号量并结束进程。

主进程的流程图如下:



图 2-2 主进程流程图

两个线程采用同步的PV操作交叉模型设计,流程类似图2-1中的设计模式。这里不在附流程图。

2.4 实验过程

2.4.1 开发环境

操作系统: 4.15.15-1-ARCH x86_64 GNU/Linux

编译器: gcc 7.3.1

自动编译工具: GNU Make 4.2.1、cmake version 3.3.2

编辑器: Visual Studio Code

2.4.2 实验步骤

首先编写源码,编译过程使用 CMake 构建 Makefile 来自动编译,编译完成

后得到可执行文件 `sync`，在终端下执行 `sync` 后得到如下结果：

```

panyue@Saltedfish > ~/code/my_github/HUST-OperatingSystem-Labs > master > ./sync
The sum is: 1
The sum is: 3
The sum is: 6
The sum is: 10
The sum is: 15
The sum is: 21
The sum is: 28
The sum is: 36
The sum is: 45
The sum is: 4095
The sum is: 4186
The sum is: 4278
The sum is: 4371
The sum is: 4465
The sum is: 4560
The sum is: 4656
The sum is: 4753
The sum is: 4851
The sum is: 4950
The sum is: 5050
panyue@Saltedfish > ~/code/my_github/HUST-OperatingSystem-Labs > master >

```

图 2-3 实验二输出结果

2.5 实验结果与分析

观察结果可知，程序正确地输出了每一步的计算结果，并且在完成计算后退出，说明程序的逻辑是正常的。同时我们去掉 PV 操作后再进行测试，得到如下结果：

```

The sum is: 5050
The sum is: 5050
The sum is: 5050
The sum is: 5050
The sum is: 5050
The sum is: 5050
The sum is: 5050
The sum is: 5050
The sum is: 5050

```

图 2-4 实验二结果分析

所有的输出都变成了 5050, 这也说明我们的程序是正确地实现了同步操作，达到了实验的预期要求。

2.6 心得体会与总结

实验二是一个对课本上的典型同步 PV 操作交叉模型的具体实现，也比较简单。PV 操作的封装是在阅读 *The Linux Programming Interface* 第 45, 47 章之后，参考书上的完备性说明封装的操作，包含

`init_sem_available()`：初始化一个可用信号量；

`init_sem_in_use()`: 初始化一个已使用信号量;

`init_sem_value()`: 按值初始化信号量;

`sem_p()`: P 操作;

`sem_v()`: V 操作。

为我在其他工程中使用之便, 还增加了 `SEM_UNDO` 功能和 `retry` 功能。

`SEM_UNDO` 功能: 在进程终止时, 撤销对信号量更改的操作。

`retry` 功能: 在 `semop()` 被信号量打断后, 重新执行 `semop()` 操作。

这些封装的源码具体见附录 A: `zxcpyplib/zxcryp_sem.c`

关于进程同步这一块, 实验上其实可以多出一些具体经典的问题, 通过问题构建模型, 再到用代码实现, 这样可以更有意思并且复杂些的操作可以熟练对相关 API 的使用。

实验三 共享内存与进程同步

3.1 实验目的

1. 掌握 Linux 下共享内存的概念与使用方法；
2. 掌握环形缓冲的结构与使用方法；
2. 掌握 Linux 下进程同步与通信的主要机制。

3.2 实验内容

利用多个共享内存（有限空间）构成的环形缓冲，将源文件复制到目标文件，实现两个进程的誊抄。

3.3 程序设计

3.3.1 总体模式

实验三使用和实验二相同的 PV 操作。

主进程首先创建三个信号量，用于同步读写和临界区控制。同时主进程还要创建一个环形缓冲区待使用，然后主进程创建两个子进程 1 和 2，分别用于从源文件读数据进入缓冲区和从缓冲区读数据进入标准输出。

实验三的设计模式图如下：

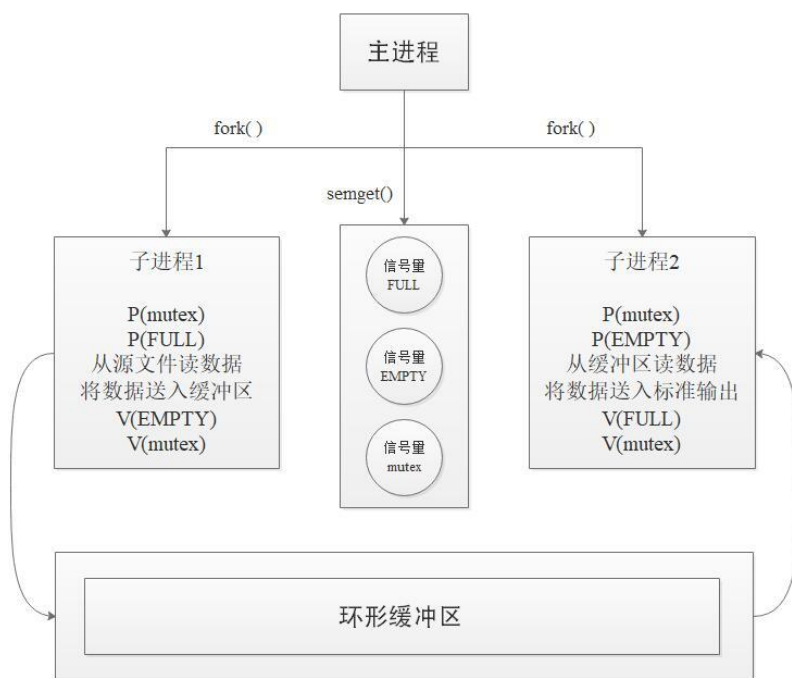


图 2-1 实验二的设计模式图

3.3.2 详细设计

主进程首先打开两个文件，注意在 open 写入的文件时，应该带有以下几个标志：O_CREAT：在文件不存在的时候创建新文件。O_TRUNC：若文件已经存在且为普通文件，那么先清空文件。这两个标志的功能为 CP 命令默认带有的功能。

然后主进程创建三个信号量，

mutex：用来控制临界区，初始化为 1；

FULL：缓冲区满时不允许继续读入，初始化为缓冲区数目；

EMPTY：缓冲区空时不允许读出，初始化为 0。

然后主进程创建并初始化环形缓冲区，供此后使用。

主进程通过 fork() 创建两个子进程 1 和 2，在完成对子进程的创建之后，通过 waitpid() 等待子进程结束。之后销毁缓冲区，销毁信号量，终止进程。

主进程的流程图如下：（由于篇幅的原因，某一步发生错误直接跳转到结束的流程不再给出）



图 2-2 主进程流程图

环形缓冲区则设计成环形链表的形式，设计了 10 个大小为 1024 的数据块，不过这里与链表不同的是，指向下一块的不是指针，而是用于标识共享内存的 `shmid`，只要利用 `shmat()` 函数就可以通过 `shmid` 来获取共享内存块，从而达到指针的效果。在每个块内设计了一个 `status` 成员，当 `status` 为 `available` 时，程序继续运行，当 `status` 为 `finish` 时，文件已经拷贝完毕，则终止两个子进程。

环形缓冲区模式图如下：（截取一部分）

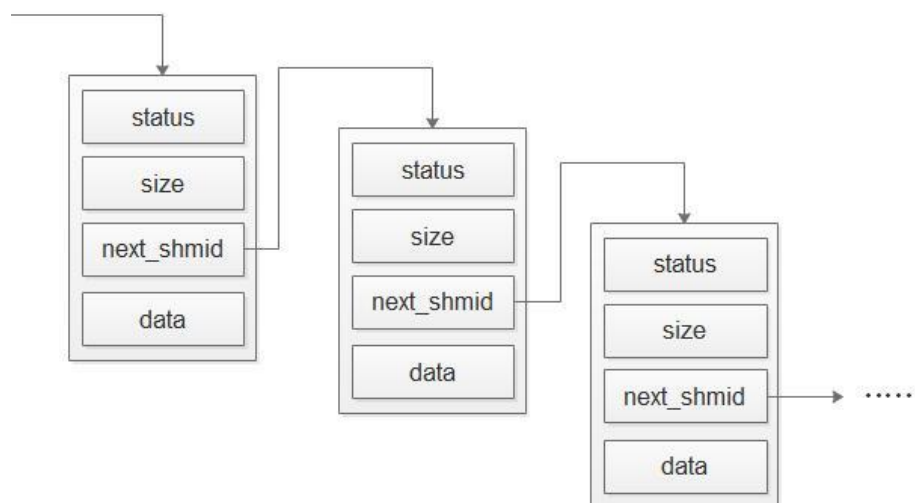


图 2-3 环形缓冲区模式图

两个子进程的模式类似实验一。

子进程 1 负责从源文件里读数据至缓冲区。首先初始化 `ring_buf_tail` 指向第一个数据块，读数据部分为临界区，所以首先要 `P(mutex)`，然后 `P(FULL)`，缓冲区容量减一，然后从源文件读一个块大小的数据，`ring_buf_tail` 指向下一个块。最后 `V(EMPTY)`，`V(mutex)`。当 `read()` 函数的返回值为 0 时，发送一个 `status` 为 `finish` 的块标志着拷贝完毕，终止进程。

子进程 2 负责从缓冲区里读数据至标准输出。首先初始化 `ring_buf_head` 指向第一个数据块，读数据部分为临界区，所以首先要 `P(mutex)`，然后 `P(EMPTY)`，从缓冲区中读取一个块的数据，`ring_buf_head` 指向下一个块，最后 `V(FULL)`，缓冲区容量加一，`V(mutex)`。当接收到一个块的 `status` 是 `finish` 时，终止进程。

子进程 1 和 2 的流程图如下：

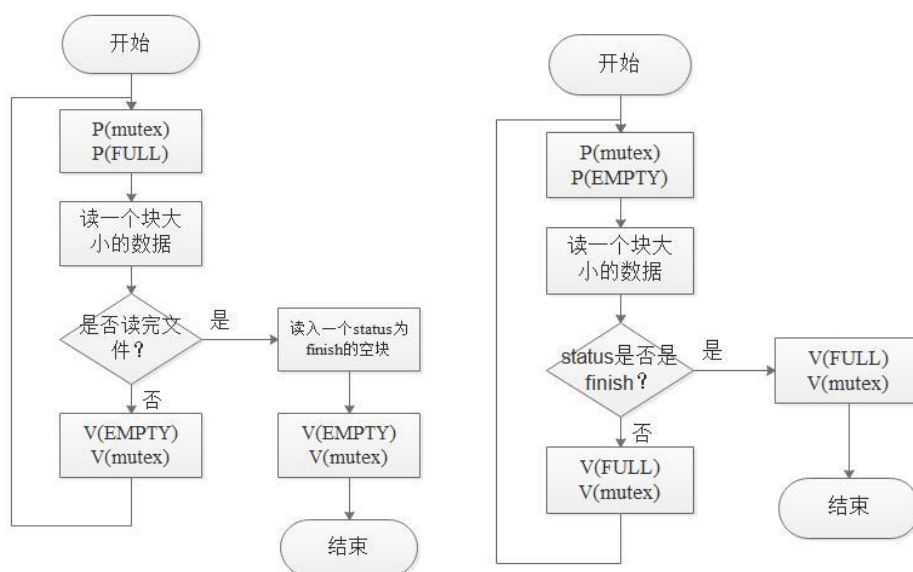


图 2-4 子进程 1 和 2 的流程图

3.4 实验过程

3.4.1 开发环境

操作系统：4.15.15-1-ARCH x86_64 GNU/Linux

编译器：gcc 7.3.1

自动编译工具：GNU Make 4.2.1、cmake version 3.3.2

编辑器：Visual Studio Code

3.4.2 实验步骤

首先编写源码，编译过程使用 CMake 构建 Makefile 来自动编译，编译完成后得到可执行文件 filecp，接下来执行 filecp 去对几种文件进行复制。

利用 filecp 复制实验一中的 pipe 文件为 test，并执行，得到如下结果：

```

panyue@Saltedfish ~/code/my_github/HUST-OperatingSystem-Labs$ ./filecp pipe test
panyue@Saltedfish ~/code/my_github/HUST-OperatingSystem-Labs$ ./test
I send you 1 times.
I send you 2 times.
I send you 3 times.
^C
Child Process 2 is Killed by Parent!
Child Process 1 is Killed by Parent!
Parent Process is Killed!
    
```

图 2-5 filecp 复制可执行文件

从结果中我们可以看到 test 程序执行结果和 pipe 一样。

再利用 filecp 复制一张图片，并比较图片的 md5 值，得到：

```

panyue@SaltedFish: ~/code/my_github/HUST-OperatingSystem-Labs
panyue@SaltedFish: ~/code/my_github/HUST-OperatingSystem-Labs
0ee2e81640ced805f0f7ab198a1409cf Sensor-Network.png
0ee2e81640ced805f0f7ab198a1409cf test.png
panyue@SaltedFish: ~/code/my_github/HUST-OperatingSystem-Labs

```

图 2-6 filecp 复制图片

从 md5 的输出中，我们可以观察到两个图片的 md5 值是一样的。

最后我们尝试用 filecp 复制本次实验的要求 PPT，比较结果：

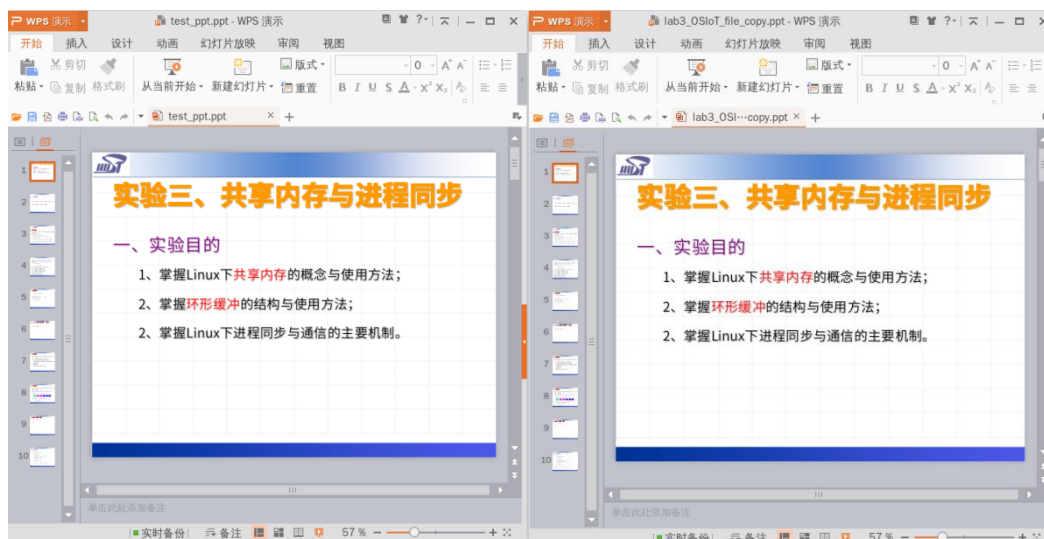


图 2-7 filecp 复制 PPT

根据打开的结果可以知道，我们的复制是成功的。

3.5 实验结果与分析

从以上三种复制结果可知，进程间通过共享内存区是可以传递信息的，我们得到的复制程序也达到实验预期的要求。

3.6 心得体会与总结

实验三相比实验一和二要复杂了很多，主要是涉及到了一个数据结构—环形缓冲区。通过实验也熟练了共享内存 API 的使用。同时在阅读了 *The Linux Programming Interface* 第 48 章后也加深了对共享内存存在虚拟内存中的布局 and 共享内存的使用细节的理解。在实验的过程中，主要遇到的问题是怎么控制拷贝的结束。在调试过几次 bug 之后决定以设计标志位的形式来判定，并且在拷贝完毕后，多发送一个带有结束标记的块进入缓冲区，当写进程接收到这一块时就终止，很好地解决了这个问题。实验中遇到的另一个问题就是如何去模拟系统中真正的 cp 的功能，经过对文档的查询，找到了几个 open() 调用中的标志，使得程序的

操作基本上与 cp 默认选项一致了。

关于缓冲区的设计,一开始使用了数组,然后思考后觉得效率不如大块传送,于是就改成了后来的环形链表式设计。

总的来说,多查 man 文档对实验会有很大帮助。

实验四 Linux 文件目录

4.1 实验目的

1. 了解 Linux 文件系统与目录操作；
2. 了解 Linux 文件系统目录结构；
3. 掌握文件和目录的程序设计方法。

4.2 实验内容

编程实现目录查询功能：

功能类似 `ls -lR`；

- 查询指定目录下的文件及子目录信息；
- 显示文件的类型、大小、时间等信息；
- 递归显示子目录中的所有文件信息。

4.3 程序设计

4.3.1 总体模式

本实验和之前的不同，整个实验只有一个递归结构，重点在于理解和掌握系统的各个接口的使用方法，通过这些接口去获得相应的信息。

4.3.2 详细设计

首先在实验目录下执行 `ls -l`，查看到底有哪些输出结果：

```

panyue@Saltedfish > ~/code/my_github/HUST-OperatingSystem-Labs > master ls -l
总用量 352
drwxr-xr-x  2 panyue panyue  4096 4月  9 13:58 Assignments
drwxr-xr-x  2 panyue panyue  4096 4月  3 20:45 bin
drwxr-xr-x  5 panyue panyue  4096 4月 15 18:38 build
-rw-r--r--  1 panyue panyue 12866 4月  1 13:14 CMakeCache.txt
drwxr-xr-x 10 panyue panyue  4096 5月 12 20:42 CMakeFiles
-rw-r--r--  1 panyue panyue  1746 4月  1 13:14 cmake_install.cmake
-rw-r--r--  1 panyue panyue   588 4月 16 10:09 CMakeLists.txt
-rwxr-xr-x  1 panyue panyue 13912 4月 17 13:59 dir
-rwxr-xr-x  1 panyue panyue 13536 5月 12 20:41 fifo
-rwxr-xr-x  1 panyue panyue 13816 5月 12 19:39 filecp
prw-----  1 panyue panyue    0 5月 12 20:26 lab01_fifo
drwxr-xr-x  2 panyue panyue  4096 4月  1 20:11 lab01-Pipe
drwxr-xr-x  2 panyue panyue  4096 4月  3 16:04 lab02-Sync
drwxr-xr-x  2 panyue panyue  4096 4月  6 21:40 lab03-FileCopy
drwxr-xr-x  2 panyue panyue  4096 4月 16 10:08 lab04-Directory
-rw-r--r--  1 panyue panyue 35147 3月 27 15:10 LICENSE
-rw-r--r--  1 panyue panyue 10553 4月 20 19:25 Makefile
-rwxr-xr-x  1 panyue panyue 13472 4月 10 21:59 pipe
-rw-r--r--  1 panyue panyue  1163 4月 17 13:23 README.md
-rw-r--r--  1 panyue panyue 155116 4月 10 22:09 Sensor-Network.png
-rwxr-xr-x  1 panyue panyue 13552 5月 12 20:42 sync
drwxr-xr-x  8 panyue panyue  4096 4月 10 21:55 TinyOS-Lab
drwxr-xr-x  3 panyue panyue  4096 4月 20 19:25 zxcpyplib
panyue@Saltedfish > ~/code/my_github/HUST-OperatingSystem-Labs > master

```

图 3-1 ls -l 输出结果

根据 ls -l 的输出，我们需要处理以下信息：文件类型，文件权限，硬链接个数，文件所属用户，文件所属用户组，文件大小，文件最后修改时间，文件名，文件夹内总块大小。

而 ls -lR 则是对目录下的所有子目录进行递归执行 ls -l 的结果，而且观察输出发现其中还涉及到了对目录名称排序的问题。综合这几项，将程序设计如下：

1. 首先传入参数，若是普通文件，则对这个文件执行信息打印，否则对文件夹执行递归打印信息；

2. 打印当前目录相对传入参数的相对路径；

3. 打印该目录下所有文件的总块大小；

4. 将该目录下所有文件按照文件名排序；

5. 对该目录下所有文件递归执行操作。

接下来具体说明各个信息怎么获取：

1. 相对路径可以通过递归调用的参数不断连接。

2. total 所表示的块大小在查阅资料后发现是 $total = \text{所有文件占有的物理块数目} * \text{物理块大小} / \text{LS_BLOCK_SIZE}$ ，期中，前两个变量可以通过 stat 函数获取，LS_BLOCK_SIZE 则与 ls 命令的实现有关，在本机上通过实验测试后得知，LS_BLOCK_SIZE 的大小是 1024，那么我们直接遍历所有文件获取总块数即可。

3. 对文件的排序使用 strcoll 函数按照 locale 排序，算法使用 quicksort。

4. 文件类型和权限可以通过 stat 结构体直接获得。

5. 硬链接数目可以通过 stat 结构体获得，但在打印之前需要计算硬链接数目的位数，从而控制打印格式。

6. 用户和用户组可以通过 getpwuid 和 getgrgid 函数获得相应的结构体，再打印相应成员即可。

7. 块大小也可以同 stat 结构体获得，同样需要在打印之前计算位数。

8. 最后修改时间可以用 ctime 函数获取。

4.4 实验过程

4.4.1 开发环境

操作系统：4.15.15-1-ARCH x86_64 GNU/Linux

编译器：gcc 7.3.1

自动编译工具：GNU Make 4.2.1、cmake version 3.3.2

编辑器：Visual Studio Code

4.4.2 实验步骤

首先编写源码，编译过程使用 CMake 构建 Makefile 来自动编译，编译完成后得到可执行文件 dir，接下来对单文件和目录分别执行 dir 测试。

对单文件执行 dir 得到的结果如图：

```
panyue@Saltedfish > ~/code/my_github/HUST-OperatingSystem-Labs | master > ./dir Makefile
-rw-r--r-- 1 panyue panyue 10553 Apr 20 19:25 Makefile
panyue@Saltedfish > ~/code/my_github/HUST-OperatingSystem-Labs | master > ls -l Makefile
-rw-r--r-- 1 panyue panyue 10553 Apr 20 19:25 Makefile
panyue@Saltedfish > ~/code/my_github/HUST-OperatingSystem-Labs | master > |
```

图 4-1 对单文件执行 dir 测试

对目录执行 dir 得到的结果如图：

```
panyue@Saltedfish > ~/code/my_github/HUST-OperatingSystem-Labs | master > ./dir .
..
total 352
drwxr-xr-x 2 panyue panyue 4096 Apr 9 13:58 Assignments
drwxr-xr-x 2 panyue panyue 4096 Apr 3 20:45 bin
drwxr-xr-x 5 panyue panyue 4096 Apr 15 18:38 build
-rw-r--r-- 1 panyue panyue 12866 Apr 1 13:14 CMakeCache.txt
drwxr-xr-x 10 panyue panyue 4096 May 12 20:42 CMakeFiles
-rw-r--r-- 1 panyue panyue 1746 Apr 1 13:14 cmake_install.cmake
-rw-r--r-- 1 panyue panyue 588 Apr 16 10:09 CMakeLists.txt
-rwxr-xr-x 1 panyue panyue 13912 Apr 17 13:59 dir
-rwxr-xr-x 1 panyue panyue 13536 May 12 20:41 fifo
-rwxr-xr-x 1 panyue panyue 13816 May 12 19:39 filecp
prw----- 1 panyue panyue 0 May 12 20:26 lab01_fifo
drwxr-xr-x 2 panyue panyue 4096 Apr 1 20:11 lab01_Pipe
drwxr-xr-x 2 panyue panyue 4096 Apr 3 16:04 lab02-Sync
drwxr-xr-x 2 panyue panyue 4096 Apr 6 21:40 lab03-FileCopy
drwxr-xr-x 2 panyue panyue 4096 Apr 16 10:08 lab04-Directory
-rw-r--r-- 1 panyue panyue 35147 Mar 27 15:10 LICENSE
-rw-r--r-- 1 panyue panyue 10553 Apr 20 19:25 Makefile
-rwxr-xr-x 1 panyue panyue 13472 Apr 10 21:59 pipe
-rw-r--r-- 1 panyue panyue 1163 Apr 17 13:23 README.md
```

图 4-2 对目录执行 dir 测试

由于目录内文件过多，我们采用 diff 命令直接比较我们的 dir 程序和真实的 ls -lR 的输出结果，如下图：


```
-rw-r--r-- 1 panyue panyue 478 Apr  9 14:29 depend.make
-rw-r--r-- 1 panyue panyue 171 Apr  1 13:11 flags.make
-rw-r--r-- 1 panyue panyue 146 Apr  3 20:46 link.txt
-rw-r--r-- 1 panyue panyue  67 Apr 20 19:25 progress.make
-rw-r--r-- 1 panyue panyue 2328 Apr 10 21:59 zxcryp_err.c.o
-rw-r--r-- 1 panyue panyue 2616 Apr 10 21:59 zxcryp_sem.c.o
panyue@Saltedfish: ~/code/my_github/HUST-OperatingSystem-Labs  master diff <(ls -lR) <(/dir .)
panyue@Saltedfish: ~/code/my_github/HUST-OperatingSystem-Labs  master
```

图 4-3 使用 diff 验证结果

图中显示 diff 没有输出，这也说明了程序的正确性。

4.5 实验结果与分析

从以上的结果可以看出，在实验目录下 dir 的输出和 ls -lR 的输出完全一致。但是，在实际测试中发现，对于一些大型目录，例如直接对 “/” 做测试，程序会由于栈溢出而崩溃，这是因为实验要求的实现方式是递归，但是在递归层数过深之后会直接导致栈溢出使程序崩溃。在查阅资料后发现，ls 并不是用递归实现的，因此在执行的时候就没有这个问题，本实验的 dir 程序只能说是在层数较浅的目录里是功能正常的。

4.6 心得体会与总结

这次实验在查询资料上费了很大功夫，也遇到了好几个需要处理的问题。

首先是文件排序的问题，这里使用的是 quicksort 算法，一开始我是使用 strcmp 来比较，在很多目录下测试都没有问题，直到突然发现在一个有中文文件名的目录下测试结果和 ls -lR 不一致，经过查阅文献之后知道，在 LC_COLLATE 和使用 strcoll 根据 locale 来排序可以解决这个问题。

另一个问题是块大小的计算，一开始我直接将所有的 stat 结构体中的 st_blocks 相加，结果发现和 ls -lR 的真实值相差两倍，查询文献资料后发现原来 stat 结构体中的块是物理块，其中的块大小是物理块大小，而 ls 的块大小是 ls 自己规定的，于是经过自己测试，得知本机的 ls 的 LS_BLOCK_SIZE 为 1024，就解决了问题。

第三个是格式控制，通过提前对相应段的长度进程计算，从而在打印的时候统一控制输出格式。

在本次实验中 man stat.2 文档给了我很大的帮助，同时 stackoverflow 也给了我很大的帮助，有不少问题的原因和解决方式是在 stackoverflow 上找到的。实验四做完之后还是对这些 API 和相关结构体内容有了很多的理解的。

PART II: TinyOS 实验

实验一 二进制 0-7 亮灯程序烧录

5.1 实验目的

1. 了解典型 nesC 的程序结构及语法;
2. 了解 tinyos 执行机制, 实现程序异步处理的方法;
3. 了解 tinyos 中 task 抽象及其使用;
4. 在 Blink 程序中使用 printf 输出信息, 使用 task 实现计算和外部设备操作的并发。

5.2 实验内容

1. Blink 程序的编译和下载。
2. Blink 程序加入 printf, 在每次定时器事件触发点亮 LED 的同时通过串口显示信息。
3. 重写 Blink 程序, 只使用一个 Timer, 三个 LED 灯作为 3 位的二进制数表示 (亮灯为 1, 不亮为 0), 按照 0-7 的顺序循环显示, 同时将数值显示在终端上。

5.3 实验过程

首先在 Makefile 中添加如下内容:

```
CFLAGS += -I$(TOSDIR)/lib/printf
```

```
PFLAGS += -DNEW_PRINTF_SEMANTICS
```

```
CFLAGS += -DPRINTF_BUFFER_SIZE=128
```

其中, DPRINTF_BUFFER_SIZE 定义节点传输数据包最大有效荷载。

然后在 BlinkAppC.nc 中加入组件:

```
components PrintfC;
```

```
components SerialStartC;
```

在 BlinkC.nc 中编写源码, 然后编译, 可以看到 LED 等按照二进制表示的 0-8 开始闪烁:

```
make telosb install /dev/ttyUSB0
```

5.4 实验结果与分析

LED 灯的闪烁效果如图：

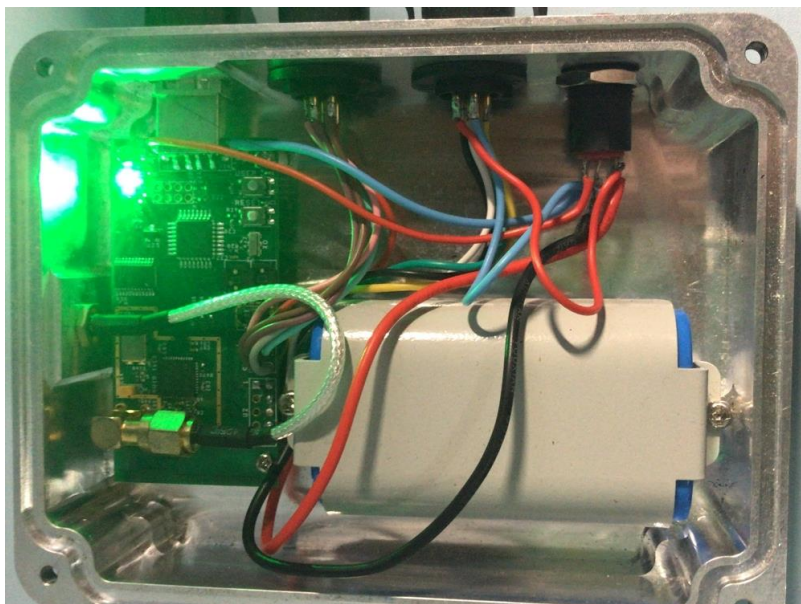


图 5-1 黄灯和红灯亮，代表 3

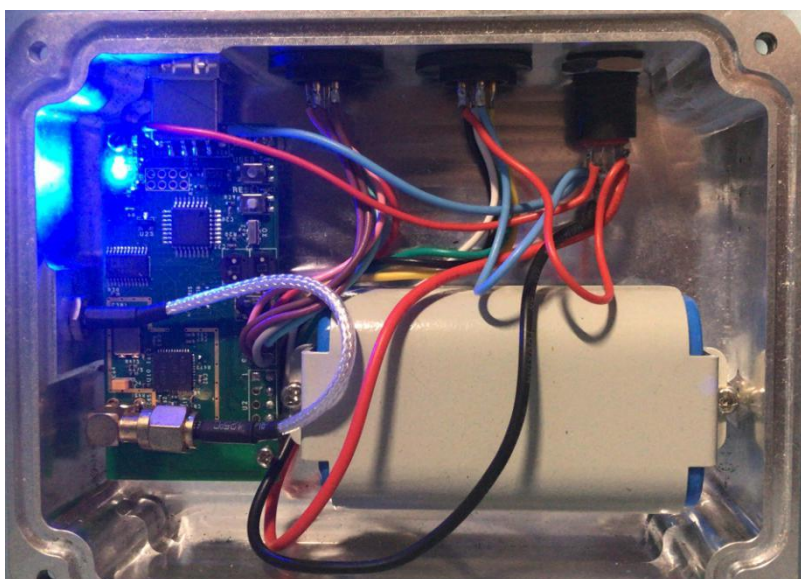


图 3-3 蓝灯亮，代表 4

由图以及实际的观察效果，TelosB 结点按照蓝灯、绿灯、红灯的顺序表示由高到低的三位，达到了实验的预期效果。

5.5 心得体会与总结

这次实验学了一下 nesC 语言的基本语法结构，简单的编程规范和形式。第一次接触这种组件化基于事件驱动的语言，感觉不是很好上手。但实际上写起来类似掉库，操作还不算很复杂，实验的逻辑也很简单。总是算是一个学习 nesC 语言的机会。

实验二 TinyOS Split-phase 过程探讨

6.1 实验目的

1. 了解典型 nesC 的程序结构及语法；
2. 了解 tinyos 执行机制，实现程序异步处理的方法；
3. 了解 tinyos 中 task 抽象及其使用；
4. 在 Blink 程序中使用 printf 输出信息，使用 task 实现计算和外部设备操作的并发。

6.2 实验内容

1. Blink 程序，在 timer0 的触发事件处理中加入计算。

```
event void Timer0.fired()
{
    uint32_t i;
    dbg("BlinkC", "Timer 0 fired @ %s.\n", sim_time_string());
    printf("LED Toggle: 0\n");
    for (i = 0; i < 400001; i++)
        call Leds.led0Toggle();
}
```

2. LED 亮灯的情况，分析其原因，将 400001 改为 10001，再观察并进行分析，加入 printf 进行输出。

3. 修改 Blink 程序，采用 task 实现计算。

4. 观察 LED 亮灯的情况，分析其原因，将 400001 改为 10001，再观察并进行分析。

5. 请修改 computeTask 的内容，将 400001 次计算分割成为若干小的部分，从而使得 LED1 和 LED2 的 fire 事件可以被正常调用，并通过 printf 输出。

6.3 实验过程

1. 直接在 TinyOS 中加入计算，我们可以看到红灯始终亮着。
2. 将 400001 改为 10001 之后，可以观察到 Blink 程序恢复正常。
3. 修改为 task 之后，观察到红灯闪烁。
4. 将 400001 改为 10001 之后，可以观察到 Blink 程序恢复正常。

5. 使用 Split-phase 模式重写 computeTask 的内容之后,编译,观察到 Blink 程序恢复正常。

6.4 实验结果与分析

由于 TinyOS 的非抢占式的进程调度机制,计算从开始运行到结束一直占用 CPU,使得 Timer1 和 Timer2 都被阻塞,没有任何运行的机会,因而会一直观察到红灯亮着(其实是因为两次闪烁之间时间间隔太短而观察不出)。

而 task 是一种类似“后台”的处理方式,调用和执行并不是同时发生。改成 Task 后我们可以看到,红灯不断闪烁,这是因为对红色 LED 的 toggle 是在每次执行 task 之前进行的,此后才执行一个 task,同样由于大量的计算阻塞了其他 Timer。

而在改成每次执行一万次计算,反复调用 task 时,就不会发生阻塞的问题了,可以观察到 Blink 程序正常进行。

6.5 心得体会与总结

通过这次实验的机会理解了 TinyOS 中的 Split-phase 机制,同时也了解了 TinyOS 的进程调度和程序编译与链接等问题,明白了静态链接对于程序优化的优点,也更熟练了 nesC 语言。

实验三 TinyOS 感知与通信实验

7.1 实验目的

1. 了解 Telosb 节点中传感器的类型与使用;
2. 了解 Telosb 节点的传感器数据的获取;
3. 获取的数据通过 printf 传输至电脑;
4. 将节点的传感器数据传输到基站,并在电脑端解析显示,了解数据的采集过程。

7.2 实验内容

1. 传感器种类:

温度传感器、湿度传感器、光照传感器

telosb 的温湿度集中在一个传感器上,该传感器名为 SHT11,是 Sensirion 公司的产品,光照传感器使用的是 Hamamatsu 的 S1087,两者的数据手册都可以从网上搜索到。

温湿度组件: SensirionSht11C()

光照组件: HamamatsuS1087ParC()

2. 温度、湿度、光照强度计算:

温度: (摄氏度)

$temp = -40.1 + 0.01 * val;$

相对湿度: (百分比)

SORH 为 12-bit 的测量值

$C1 = -4, C2 = 0.0405, C3 = -2.8 * 10E-6$

光照强度: (勒克斯 Lux 或 Lx)

$I = AD * 1.5 / 4096 / 10000$

$LUX = 0.625 * 1e6 * I * 1000$ (S1087)

$LUX = 0.769 * 1e6 * I * 1000$ (S1087-01)

其中 AD 为 12-bit 测量值

3. 典型的办公室温度. 湿度. 光照强度测量值

```
serial@/dev/ttyUSB1:115200: resynchronising
```

```
received temperature:24.17°C
```

```
received humidity:55.86%
```

```
received photo:572.20Lux
```

```
received temperature:24.18°C
```

```
received humidity:55.89%
```

```
received photo:549.32Lux
```

7.3 实验设计

实验分为三个部分：

1. 使用 TelosB 节点感知数据。
2. 节点与节点间实现无限通信。
3. 节点与 PC 间实现串口通信。

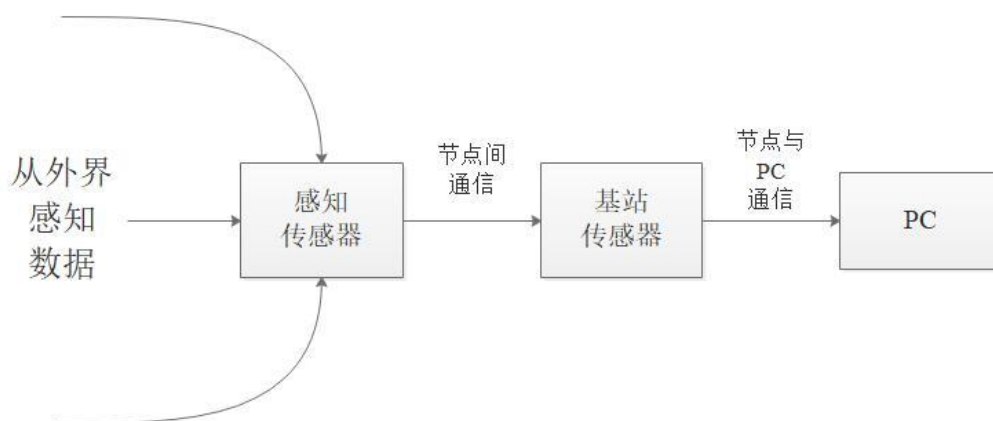


图 7-2 实验三设计模式

第一部分，感知程序的设计（修改 Sense 例程）

我们在 SenseAppC.nc 中添加要使用的传感器组件，本实验中使用的是 SensirionSht11C 和 HamamatsuS1087ParC，然后在 SenseC.nc 中利用 Read 接口就可以实现对传感器数据的读取。但是注意，这里读取的数据并不是真实生活中使用的值，还要通过后续的实现来实现。

第二部分，无限通信的设计（修改 BlinkToRadio 例程）

通过查阅资料，TinyOS 在关于通信的接口中，都使用了一个共同的消息抽象：message_t，其结构如下

```
typedef nx_struct message_t{
    nx_uint8_t header [sizeof(message_header_t)];
    nx_uint8_t data[TOSH_DATA_LENGTH];
    nx_uint8_t footer[sizeof(message_footer_t)];
    nx_uint8_t metadata[message_metadata_t];
}message_t;
```

但其结构对用户不透明，我们需要利用接口来操作这个结构体。

为了传递数据，自定义外部结构体（nx_struct）如下：

```
typedef nx_struct SenseMsg {
    nx_uint16_t nodeid;
    nx_uint16_t kind;
    nx_uint16_t data;
} SenseMsg;
```

这里 kind 代表传感器类型，通过宏定义定义三种类型，data 为传感器获取的数据。

节点间的通信使用 AMPacket 和 AMSend 接口，仿照 BlinkToRadio 例程，实现消息的发送即可，接着将其和第一部分结合，在感知数据之后根据传感器类型和数据设置 SenseMsg 结构体的成员值，然后通过 AMSend.send 将数据发送出去。

第三部分，节点与 PC 串口通信（修改 TestSerial 例程）

节点与 PC 间的串口通信使用 MIG 工具以便于解析数据。

在 Makefile 里添加如下内容，生成解析数据需要使用的 java 文件：

```
BUILD_EXTRA_DEPS += SenseMsg.class
CLEAN_EXTRA = *.class SenseMsg.java
```

```
SenseMsg.class: SenseMsg.java
```

```
    javac SenseMsg.java
```

SenseMsg.java:

```
mig java -target=$(PLATFORM) -java-classname=SenseMsg Sense.h
SenseMsg -o $@
```

编译完成之后生成了一个 java 文件 SenseMsg.java，其中包含了很多方便我们解析的函数。然后模仿例程中的 TestSerial.java 文件，写出只接收串口数据的 java 文件 ReceiveSerial.java。

与 PC 连接的结点要完成接收感知结点发过来的数据包和向串口发送数据包的功能，作为数据的中转站，这里我直接使用了 Basestation 例程。

7.4 实验过程

在第一个结点上烧录 Sense 程序，在第二个结点上烧录 Basestation 程序，在 PC 上执行 ReceiveSerial.java，结果如下：

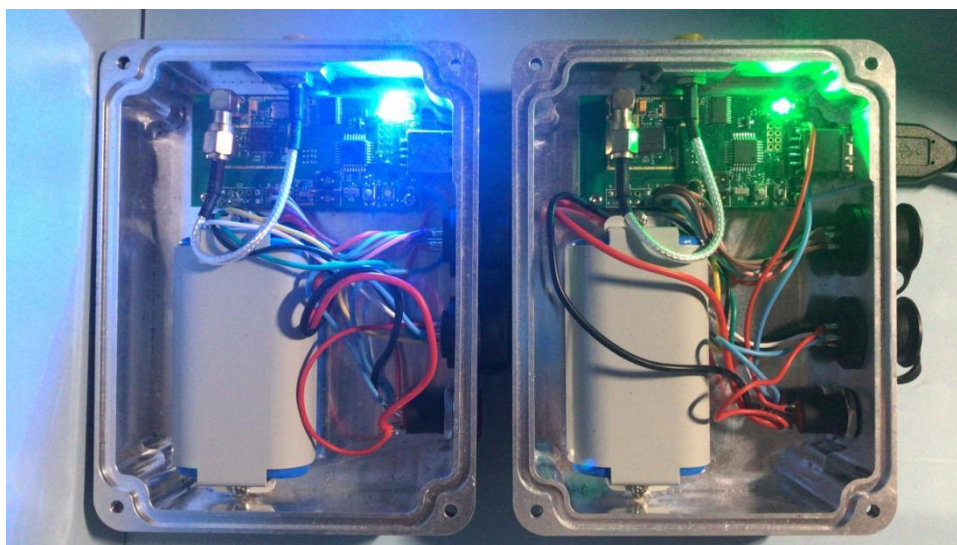
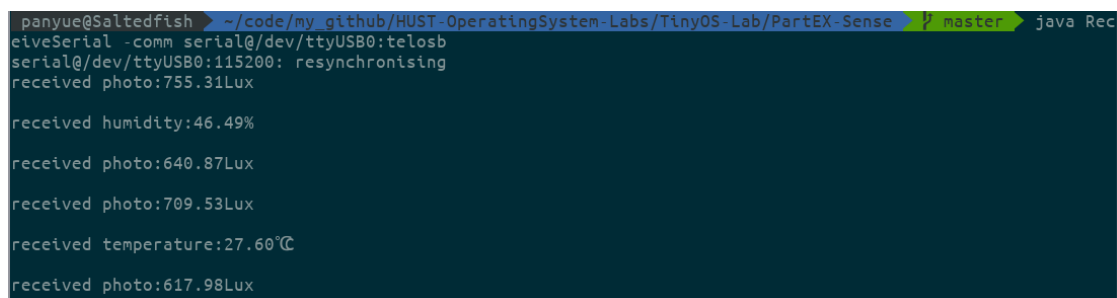


图 7-2 结点显示结果

左侧结点为感知结点，在感知到相应类别是，对应的灯会闪烁，右侧为基站结点，在接受到左侧结点发送的数据之后会闪烁，然后将数据发送到 PC。

执行 ReceiveSerial.java 后，得到的结果如下：



```

panyue@Saltedfish: ~/code/my_github/HUST-OperatingSystem-Labs/TinyOS-Lab/PartEX-Sense
eiveSerial -comm serial@/dev/ttyUSB0:telosb
serial@/dev/ttyUSB0:115200: resynchronising
received photo:755.31Lux

received humidity:46.49%

received photo:640.87Lux

received photo:709.53Lux

received temperature:27.60℃

received photo:617.98Lux
    
```

图 7-3 数据分析显示结果

7.5 实验结果与分析

从最终的结果中可以看到，分析得到的实验数据和实验文档中给的一般数据类似，验证我们的程序达到了实验预期的要求。

7.6 心得体会与总结

这次实验之前还是做了很多的准备工作的，通过查阅相关资料和阅读 TinyOS 官方文档，学习了一下 TinyOS 通信模块的使用，踩的坑也不少。

感知部分的实验很简单，调用一下几个接口就好。

节点间通信部分，一开始直接对着接口写发包，然后节点老是没反应，最后通过查阅资料，了解到 TinyOS 有自带的 8 个 tutorials，开发者可以通过这 8 个例程来学习相关模块的使用，于是根据 BlinkToRadio 例程进行学习，仿照着写出了这部分的程序。

串口通信部分，一开始打算用 Python 实现，后来发现写起来还是相当的麻烦，尤其是解析 `message_t` 结构这部分，通过查阅了一些文档，了解到了可以使用 MIG 工具来解析数据，于是学习了 MIG 的使用，又通过 TestSerial 例程学习了用 java 接受数据的写法，最后实现了通信。不过，这部分出现的问题相当多，debug 用了很长时间，就不细说了。

关于基站程序，我在实验时直接使用了 TinyOS 的 Basestation 例程，其实可以自己写出很简单的转发程序，结合节点间通信和串口通信即可。但是和 Basestation 比较就可以发现，自己的程序在数据包一多时就显得乏力了。分析 Basestation 的代码了解到，例程除了在实现转发数据包以外，又实现了缓冲区，从而处理接收到大量数据包的情况，另外例程还实现了对丢包的判断。关于 TinyOS 的编程这里，还有很多的地方需要继续深入学习啊，主要还是很多东西

该使用什么接口的问题，这个还是要很多的积累的。

由于之前看过也写过计网的很多东西，所以通信实验做起来其实没有那么陌生，上手也比较快。整个实验中对我帮助最大的还是官方 TinyOS 文档 http://tinyos.stanford.edu/tinyos-wiki/index.php/Main_Page，对于很多接口的使用都有详细的解释，另外就是通过 Google 搜到了很多前人们写的博客，对于 TinyOS 语言或是经典的编程模型都有很多讲解。而 nesC 语言的学习也主要依赖于博客去快速上手，直接看详细的书籍其实并不是很好的学习方式，有大块时间了准备详细学习语言的时候再看吧。

总结一下所有的实验，果然仅仅通过做实验去学习，收获的东西并不会有很多，实验本身很简单，没有什么过多的可发掘的内容。真正可以学到很多的是在实验的过程中不断去看书，去 Google，去看文档学到的东西，这些在课外学到的东西才是真的很多啊。

PART III: 附录

附录 A 实验中用到的库文件说明

A.1 zxcpylib/zxcryp_sys.h

文件说明：系统编程通用的头文件，包含系统库文件和几个宏定义源码：

```
/*
 * zxcryp's lib
 *
 * zxcryp sys header
 */

#ifndef ZXCPYP_H
#define ZXCPYP_H

#include <stdio.h>      /* Standard I/O functions */
#include <stdlib.h>     /* Standard library */
#include <string.h>     /* String handling functions */
#include <unistd.h>     /* Unix system calls */
#include <errno.h>      /* Error difinations */
#include <fcntl.h>      /* Micros for I/O */
#include <sys/types.h>  /* System data type */
#include <sys/stat.h>   /* File system status */

#include "zxcryp_err.h" /* Error handling functions */

/* True and false defines */
#ifndef FALSE
#define FALSE 0
#endif // !FALSE
#ifndef TRUE
#define TRUE 1
#endif // !TRUE

/* Max and min defines */
#ifndef min
#define min(m, n) ((m) < (n) ? (m) : (n))
#endif // !min
```

```
#ifndef max
    #define max(m, n) ((m) > (n) ? (m) : (n))
#endif // !max

#endif // !ZXCPYP_H
```

A. 2 zxcpyplib/zxcpyp_err.h

文件说明：错误处理头文件

源码：

```
/*
 * zxcpyp's Lib
 *
 * zxcpyp error message header
 */

#ifndef ZXCPYP_ERR_H
#define ZXCPYP_ERR_H

#include <stdio.h>
#include <stdlib.h>

/*
 * err_msg - Print error message
 */
void err_msg(const char *err);

/*
 * err_exit - Announce error and exit
 */
void err_exit(const char *err);

/*
 * usage_err - Announce usage and exit
 */
void usage_err(const char *err);

/*
 * fatal_err - Handle fatal error
 */
void fatal_err(const char *err);
```

```
#endif // !ZXCYPY_ERR_H
```

A.3 zxcpyplib/zxcpyp_err.c

文件说明：错误处理函数实现

源码：

```
/*
 * zxcpyp's Lib
 *
 * zxcpyp error message lib
 */

#include "zxcpyp_err.h"

void err_msg(const char *err) {
    printf("Error: %s\n", err);
}

void err_exit(const char *err) {
    printf("%s error!\n", err);
    printf("Exit with 1.\n");
    exit(1);
}

void usage_err(const char *err) {
    printf("Usage: %s\n", err);
    exit(1);
}

void fatal_err(const char *err) {
    printf("Fatal: %s\n", err);
    exit(1);
}
```

A.4 zxcpyplib/systemV_ipc.h

文件说明：进程间通信通用的头文件

源码：

```
/*
```



```

* zxcryp's lib
*
* zxcryp system V IPC header
*/

#ifndef ZXCPYP_SYSTRMV_IPC_H
#define ZXCPYP_SYSTRMV_IPC_H

#include <sys/types.h> /* System data type */
#include <sys/msg.h> /* System V Message */
#include <sys/sem.h> /* System V Semaphore */
#include <sys/shm.h> /* System V Shared Memory */
#include <errno.h> /* Error difinations */

/*
+-----+-----+-----+-----+
|      IPC      | Message | Semaphore | Shared Memory |
+-----+-----+-----+-----+
| Head file | <sys/msg.h> | <sys/sem.h> | <sys/shm.h> |
+-----+-----+-----+-----+
| Data structure| msqid_ds | semid_ds | shmid_ds |
+-----+-----+-----+-----+
| Create Object | msgget() | semget() | shmget()shmat()|
+-----+-----+-----+-----+
| Control | msgctl() | semctl() | shmctl() |
+-----+-----+-----+-----+
|      IPC      | msgsnd() | semop() | |
| Operate | msgrcv() | (P/V) | |
+-----+-----+-----+-----+
*/

/* True and false defines */
#ifndef FALSE
#define FALSE 0
#endif // !FALSE
#ifndef TRUE
#define TRUE 1
#endif // !TRUE

#endif // !ZXCPYP_SYSTRMV_IPC_H

```

A.5 zxcpyplib/zxcpyp_sem.h

文件说明：信号量处理封装函数头文件

源码：

```
/*
 * zxcpyp's Lib
 *
 * zxcpyp semaphore header (System V)
 */

#ifndef ZXCYPYP_SEM_H
#define ZXCYPYP_SEM_H

#include "systemV_ipc.h"

/* Used in calls to semctl() */
union semun {
    int val;
    struct semid_ds *buf;
    unsigned short *array;
#ifdef __linux__
    struct seminfo *__buf;
#endif // __linux__
};

extern int UseSemUndo;    /* Whether using SEM_UNDO in semop() */
extern int RetryOnEintr; /* Whether retry when semop() is interrupted */

/*
 * init_sem_available - Initial a semaphore to 1
 */
int init_sem_available(int sem_id, int sem_num);

/*
 * init_sem_in_use - Initial a semaphore to 0
 */
int init_sem_in_use(int sem_id, int sem_num);

/*
 * init_sem_value - Initial a semaphore to sem_value
 */
int init_sem_value(int sem_id, int sem_num, int sem_value);
```

```

/*
 * sem_p - P operate
 */
int sem_p(int sem_id, int sem_num);

/*
 * sem_v - V operate
 */
int sem_v(int sem_id, int sem_num);

#endif // !ZXCYPY_SEM_H

```

A.6 zxcpyplib/zxcypyp_sem.c

文件说明：信号量处理封装函数实现

源码：

```

/*
 * zxcypyp's Lib
 *
 * zxcypyp semaphore lib (System V)
 */

#include "zxcypyp_sem.h"

int UseSemUndo = FALSE; /* Whether using SEM_UNDO in semop() */
int RetryOnEintr = TRUE; /* Whether retry when semop() is interrupted */

int init_sem_available(int sem_id, int sem_num) {
    union semun arg; /* semget() use arg to control */
    arg.val = 1; /* Semaphore available */
    return semctl(sem_id, sem_num, SETVAL, arg);
}

int init_sem_in_use(int sem_id, int sem_num) {
    union semun arg; /* semget() use arg to control */
    arg.val = 0; /* Semaphore in use */
    return semctl(sem_id, sem_num, SETVAL, arg);
}

int init_sem_value(int sem_id, int sem_num, int sem_value) {

```

```

union semun arg;    /* semget() use arg to control */
arg.val = sem_value; /* Ordered semaphore value */
return semctl(sem_id, sem_num, SETVAL, arg);
}

int sem_p(int sem_id, int sem_num) {
    struct sembuf sops; /* semop() use sops to control */
    sops.sem_num = sem_num;
    sops.sem_op = -1;    /* the P operation */
    sops.sem_flg = UseSemUndo ? SEM_UNDO : 0;
    while(semop(sem_id, &sops, 1) == -1)
        /* The operation was interrupted by a signal handler */
        if (errno != EINTR || !RetryOnEintr)
            return -1;
    return 0;
}

int sem_v(int sem_id, int sem_num) {
    struct sembuf sops; /* semop() use sops to control */
    sops.sem_num = sem_num;
    sops.sem_op = 1;    /* the V operation */
    sops.sem_flg = UseSemUndo ? SEM_UNDO : 0;
    return semop(sem_id, &sops, 1);
}

```

A.7 zxcpylib/ring_buf.h

文件说明：环形缓冲区相关定义

源码：

```

/*
 * zxcryp's Lib
 *
 * zxcryp ring buffer header
 */

#ifndef ZPCYP_RINGBUF_H
#define ZPCYP_RINGBUF_H

#include "zxcryp_sys.h"

/* Buffer status */

```

```
#define STATUS_AVALAVLE 0
#define STATUS_FINISH 1

/* Ring buffer blocks num */
#define RING_BUF_NUM 10

/* The ring buffer length */
#define RING_BUF_LEN 1024

typedef struct ring_buf {
    int status;           /* This buffer status */
    int size;             /* This buffer size */
    int next_shmid;       /* The semid of next buffer */
    char data[RING_BUF_LEN]; /* Data stored */
} ring_buf;

#endif // !ZXCPYP_RINGBUF_H
```

附录 B 在 Archlinux 环境下配置 TinyOS 开发环境

一、Arch Linux 系统的安装

Arch 的系统镜像可以在 <https://www.archlinux.org/download/> 处下载，安装过程可以参考官方 wiki: Archwiki。由于 Archlinux 的安装过程及其复杂，涉及到许多 Linux 系统的细节知识，这里不详细展开，推荐参考 viseator 的博客教程 https://www.viseator.com/2017/05/17/arch_install/。桌面环境选择 KDE 或者 Gnome 均可，可以自由安装。

二、安装必要的软件包

Arch 下很多和 tinyos 相关的包都已经在 AUR 里了，我们可以方便的下载如下几个包：

```
yaourt -S tinyos
yaourt -S tinyos-tools
yaourt -S nesc
yaourt -S gcc-avr-tinyos
yaourt -S binutils-avr-tinyos
```

执行完以上几条命令之后，在系统的 /opt 目录下可以发现安装好的 tinyos，在这里对其进行一系列的修改需要 root 权限，为了方便自己使用，建议将其移动到 home 目录下，便于使用。（当然也可以放在 /opt 下）

```
sudo mv -rf /opt/tinyos-2.1.2 ~/tinyos-2.1.2
```

三、交叉编译器安装

安装交叉编译器 msp430，在 AUR 中也已经集成了相关的包，直接安装即可：

```
yaourt -S msp430mcu
yaourt -S mspgcc-ti
yaourt -S mspdebug
yaourt -S msp430-libc
yaourt -S gcc-msp430
yaourt -S binutils-msp430
```

需要注意的是，在 AUR 中有很多和 msp430 相关的包已经过时了，安装时需注意一下 AUR 中的 out of date 标识。

四、环境变量配置

在命令行中执行：

```
export TOSROOT="$HOME/tinyos-2.1.2"
export TOSDIR="$TOSROOT/tos"
export CLASSPATH="$TOSROOT/support/sdk/java/tinyos.jar:."
export MAKERULES="$TOSROOT/support/make/Makerules"
```

上述命令会把环境变量写入当前用户下的 .bashrc 文件，设置在当前目录下有效，但是在重启电脑之后就会失效，需要重新配置，建议将以上命令单独编辑为脚本文件 tinyos.sh，并且导入当前使用的 shell 的配置文件中，例如：

```
#!/usr/bin/env bash

# To setup the environment variables needed by the TinyOS

TOSROOT="$HOME/tinyos-2.1.2"
TOSDIR="$TOSROOT/tos"
CLASSPATH="$TOSROOT/support/sdk/java/tinyos.jar:."
MAKERULES="$TOSROOT/support/make/Makerules"

export TOSROOT TOSDIR CLASSPATH MAKERULES
```

然后在 shell 配置文件中（默认是 .bashrc）中写入如下两行：

```
#Sourcing the TinyOS environment variables
source $HOME/tinyos-2.1.2/tinyos.sh
```

若想设置对全部用户有效，请将上面上面四行内容单独编辑为独立的 tinyos.env 文件，将此文件拷贝到 /etc/profile.d/ 中。

五、检查开发环境

设置完成后，在终端输入

```
tos-check-env
```

检查环境是否安装和设置好，结果应该和如下类似：

```
linux> tos-check-env
```

Path:

```
/usr/local/bin  
/usr/local/sbin  
/usr/bin  
/opt/cuda/bin  
/usr/lib/jvm/default/bin  
/opt/ti/mspgcc/bin  
/usr/bin/site_perl  
/usr/bin/vendor_perl  
/usr/bin/core_perl
```

Classpath:

```
/home/(your user name)/tinyos-2.1.2/support/sdk/java/tinyos.jar  
.
```

rpms:

nesc:

```
/usr/bin/nesc  
Version: nesc: 1.3.6
```

perl:

```
/usr/bin/perl  
Version: v5.26.1) built for x86_64-linux-thread-multi
```

flex:

```
/usr/bin/flex
```

bison:

```
/usr/bin/bison
```

java:

```
/usr/bin/java
```



```
--> WARNING: The JAVA version found first by tos-check-env may not be version
1.4 or version 1.5one of which is required by TOS. Please ensure that the located
Java version is 1.4 or 1.5
```

```
graphviz:
```

```
  /usr/bin/dot
```

```
dot - graphviz version 2.40.1 (20161225.0304)
```

```
--> WARNING: The graphviz (dot) version found by tos-check-env is not 1.10.
Please update your graphviz version if you'd like to use the nescdoc documentation
generator.
```

```
tos-check-env completed with errors:
```

```
--> WARNING: The JAVA version found first by tos-check-env may not be version
1.4 or version 1.5one of which is required by TOS. Please ensure that the located
Java version is 1.4 or 1.5
```

```
--> WARNING: The graphviz (dot) version found by tos-check-env is not 1.10.
Please update your graphviz version if you'd like to use the nescdoc documentation
generator.
```

关于 java 和 graphviz 的警告是由于 Java 版本较新造成的, 可以不必理会。

执行命令

```
printenv MAKERULES
```

应该看到类似如下显示

```
linux> printenv MAKERULES
```

```
/home/(your user name)/tinyos-2.1.2/support/make/Makerules
```

六、Python 串口通信包替换

在 Arch 下配置 TinyOS 时, 必须要进行 Python 串口通信包的替换, 否则在

编译的时候会报错，因为 TinyOS 的串口通信模块是用 Python2 的 serial 包 2.7 版本开发的，而在 Archlinux 下，不论是 Python3 还是 Python2 内的 serial 包都已经达到了 3.x 版本，因此我们必须要先进行包的替换。

首先查看是否已安装 python2-pyserial 包，若已安装，则卸载包。

```
sudo pacman -Ss python-pyserial
```

```
sudo pacman -R python-pyserial
```

由于在现在 Arch 的包管理中，这个 python2-pyserial 包已经变成最新版本的了，所以我们必须到官网去下载 pyserial2.7 版本。

进入官网 <https://pypi.org/project/pyserial/2.7/#files>，点击 Download files，选择 pyserial-2.7.tar.gz 下载。

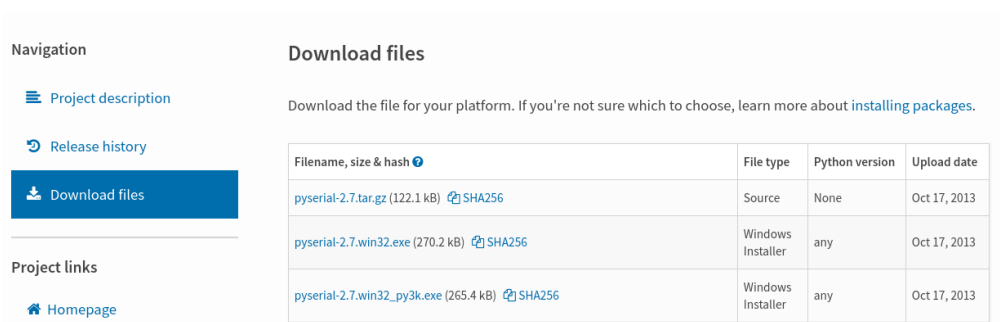


图 B.1 官网下载 pyserial2.7

下载完成后，解压并手动编译即可。

```
tar -xvf pyserial-2.7.tar.gz pyserial-2.7/
```

```
cd pyserial-2.7
```

```
python setup.py install
```

```
sudo python setup.py install
```

七、第一个 TinyOS 程序的编译和烧写

将节点电源断开[注意：非常重要，这里一定不要打开电源，否则会导致节点烧坏]，用 USB 连接到电脑，使用 motelist 命令查看输出。

```
linux > motelist
```

Reference	Device	Description
FT0JLCD	/dev/ttyUSB0	FTDI USB <-> Serial Converter

其中，/dev/ttyUSB0 为该节点在 linux 中对应的设备文件。

命令 `motelist` 仅支持 `telosB` 结点，其他结点请用 `dmesg` 命令，输出类似：

```
[ 709.846133] usb 1-3: FTDI USB Serial Device converter now attached to ttyUSB0
```

然后可以编译和烧写程序，进入 LED 闪烁程序目录：

```
cd $TOSROOT/apps/Blink
```

```
make telosb install /dev/ttyUSB0
```

若已有程序，请将 `install` 换成 `reinstall`

显示如下图，然后便可以看到结点的三个 LED 灯闪烁。

```
panyue@Saltedfish ~/tinyos-2.1.2/apps/Blink$ make telosb install /dev/ttyUSB0
mkdir -p build/telosb
compiling BlinkAppC to a telosb binary
ncc -o build/telosb/main.exe -Os -DNEW_PRINTF_SEMANTICS -fnesc-separator=__ -Wall -Wshadow -Wnesc-all -target=telosb -fnesc-cfile=build/telosb/app.c -board= -DDEFINED_TOS_AM_GROUP=0x22 -I/home/panyue/tinyos-2.1.2/tos/lib/printf -DPRINTF_BUFFER_SIZE=128 -DIDENT_APPNAME="BlinkAppC" -DIDENT_USERNAME="panyue" -DIDENT_HOSTNAME="Saltedfish" -DIDENT_USERHASH=0xedc60628L -DIDENT_TIMESTAMP=0x5afc2b7fL -DIDENT_UIDHASH=0xad2a7f25L BlinkAppC.nc -lm
compiled BlinkAppC to build/telosb/main.exe
6330 bytes in ROM
482 bytes in RAM
msp430-objcopy --output-target=ihex build/telosb/main.exe build/telosb/main.ihex
writing TOS image
cp build/telosb/main.ihex build/telosb/main.ihex.out
found mote on /dev/ttyUSB0 (using bsl,auto)
installing telosb binary using bsl
tos-bsl --telosb -c /dev/ttyUSB0 -r -e -I -p build/telosb/main.ihex.out
MSP430 Bootstrap Loader Version: 1.39-goodfet-8
Mass Erase...
Transmit default password ...
Invoking BSL...
Transmit default password ...
Current bootstrap loader version: 1.61 (Device ID: f16c)
Changing baudrate to 38400 ...
Program ...
6474 bytes programmed.
Reset device ...
rm -f build/telosb/main.exe.out build/telosb/main.ihex.out
panyue@Saltedfish ~/tinyos-2.1.2/apps/Blink$
```

图 B.1 烧写程序至结点

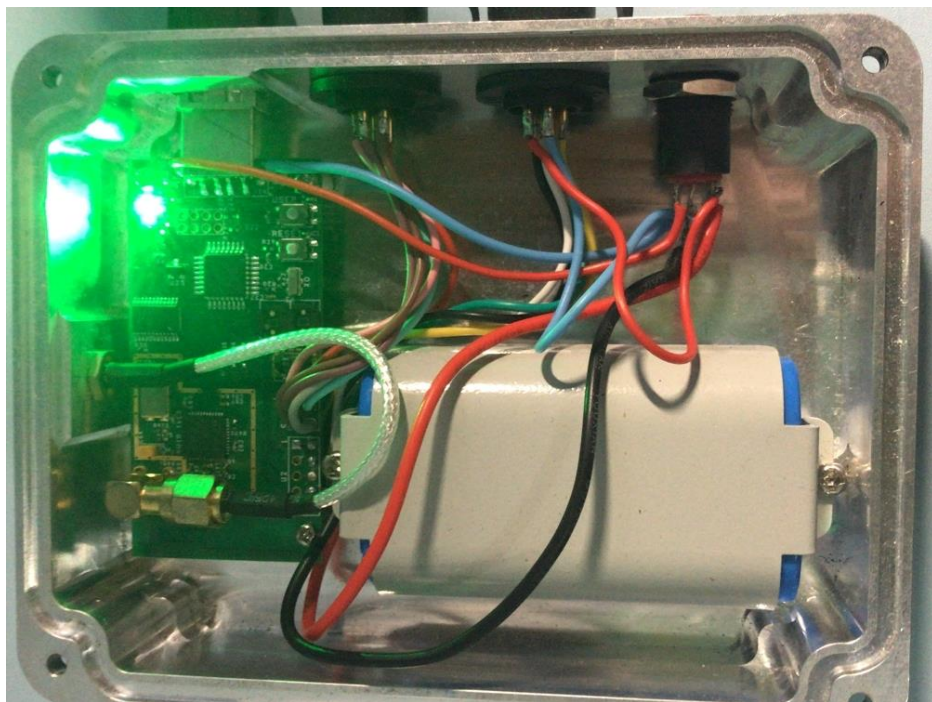


图 B. 2 Blink 程序成功烧写

成功到这里之后，TinyOS 环境就配置好了。

附录 C 操作系统实验代码

C.1 操作系统实验编译文件

文件名: CMakeLists.txt

作用: 自动编译程序

源码:

```
cmake_minimum_required(VERSION 3.9)

project(OS_labs)

# add zxcryp lib
add_subdirectory(./zxcryp_lib)

# build lab01
add_executable(pipe ./lab01-Pipe/pipe.c)
target_link_libraries(pipe zxcryp_lib)

add_executable(fifo ./lab01-Pipe/fifo.c)
target_link_libraries(fifo zxcryp_lib)

# build lab02
add_executable(sync ./lab02-Sync/sync.c)
target_link_libraries(sync zxcryp_lib pthread)

# build lab03
add_executable(filecp ./lab03-FileCopy/filecp.c)
target_link_libraries(filecp zxcryp_lib)

# build lab04
add_executable(dir ./lab04-Directory/directory.c)
target_link_libraries(dir zxcryp_lib)
```

C.2 实验一代码

文件名: pipe.c

作用: 实现无名管道

源码:

```
/*
 * HUST OS Lab 01 - Single Pipe
```

```

*
* By Pan Yue
*/

#include "../zxcpyplib/zxcryp_sys.h"
#include <signal.h>
#include <wait.h>

#define BUF_SIZE 30

int pid1, pid2;

/*
 * pipe(int *pfd)
 * pfd[0]: read
 * pfd[1]: write
 */

/*
 * sigint_handler - handle SIGINT, kill child pid
 */
void sigint_handler(int sig) {
    printf("\n");
    kill(pid1, SIGUSR1);
    kill(pid2, SIGUSR2);
}

/*
 * sigusr_handler - handle SIGUSR1/2, end child pid
 */
void sigusr_handler(int sig) {
    if (sig == SIGUSR1) {
        printf("Child Process 1 is Killed by Parent!\n");
        exit(0);
    }
    else if (sig == SIGUSR2) {
        printf("Child Process 2 is Killed by Parent!\n");
        exit(0);
    }
}

int main(void) {

```

```

int pfd[2];
char read_buf[BUF_SIZE];
int read_num;
int wait_tmp;
signal(SIGINT, sigint_handler);

if (pipe(pfd) == -1)
    err_exit("Pipe");

switch (pid1 = fork()) {
    case -1:
        err_exit("First fork");

        /* Child 1 - write data */
        case 0:
            /* Ignore SIGINT */
            signal(SIGINT, SIG_IGN);
            /* Catch SIGUSR2 */
            signal(SIGUSR1, sigusr_handler);
            if (close(pfd[0]) == -1)
                err_exit("Child 1 close read fd");

            int count = 1;
            char write_buf[BUF_SIZE];
            for (;;) {
                sprintf(write_buf, "I send you %d times.\n", count);
                if (write(pfd[1], write_buf, strlen(write_buf)) !=
strlen(write_buf))
                    err_exit("Child 1 write");
                sleep(1);
                count++;
            }

            /* Parent pid */
            default:
                switch (pid2 = fork()) {
                    case -1:
                        err_exit("Second fork");

                        /* Child 2 - read data */
                        case 0:
                            /* Ignore SIGINT */

```

```

    signal(SIGINT, SIG_IGN);
    /* Catch SIGUSR2 */
    signal(SIGUSR2, sigusr_handler);
    if (close(pfd[1]) == -1)
        err_exit("Child 2 close write fd");

    for (;;) {
        read_num = read(pfd[0], read_buf, BUF_SIZE);
        if (read_num == -1)
            err_exit("Child 2 read");
        if (read_num == 0)
            continue;
        if (write(STDOUT_FILENO, read_buf, read_num) != read_num)
            err_exit("Child 2 write");
    }

    /* Parent pid */
    default:
        if (close(pfd[0]) == -1)
            err_exit("Parent close pfd[0]");
        if (close(pfd[1]) == -1)
            err_exit("Parent close pfd[1]");
        /* Wait for child pid */
        waitpid(pid1, &wait_tmp, 0);
        waitpid(pid1, &wait_tmp, 0);
        printf("Parent Process is Killed!\n");
        exit(0);
    }
}
}
}

```

文件名: fifo.c

作用: 实现命名管道

源码:

```

/*
 * HUST OS Lab 01 - Single Pipe
 *
 * By Pan Yue
 */

#include "../zxcpyplib/zxcryp_sys.h"

```



```

#include <signal.h>
#include <wait.h>

#define FIFO "lab01_fifo"
#define BUF_SIZE 30

int pid1, pid2;

/*
 * pipe(int *pfd)
 * pfd[0]: read
 * pfd[1]: write
 */

/*
 * sigint_handler - handle SIGINT, kill child pid
 */
void sigint_handler(int sig) {
    printf("\n");
    kill(pid1, SIGUSR1);
    kill(pid2, SIGUSR2);
}

/*
 * sigusr_handler - handle SIGUSR1/2, end child pid
 */
void sigusr_handler(int sig) {
    if (sig == SIGUSR1) {
        printf("Child Process 1 is Killed by Parent!\n");
        exit(0);
    }
    else if (sig == SIGUSR2) {
        printf("Child Process 2 is Killed by Parent!\n");
        exit(0);
    }
}

int main(void) {
    int read_fd, write_fd;
    char read_buf[BUF_SIZE];
    int read_num;

```

```

int wait_tmp;
signal(SIGINT, sigint_handler);

if (mkfifo(FIFO, S_IRUSR | S_IWUSR) == -1 && errno != EEXIST)
    err_exit("Pipe");

switch (pid1 = fork()) {
    case -1:
        err_exit("First fork");

        /* Child 1 - write data */
    case 0:
        /* Ignore SIGINT */
        signal(SIGINT, SIG_IGN);
        /* Catch SIGUSR2 */
        signal(SIGUSR1, sigusr_handler);

        /* Open FIFO writer */
        if ((write_fd = open(FIFO, O_WRONLY)) == -1)
            err_exit("FIFO writer open");

        int count = 1;
        char write_buf[BUF_SIZE];
        for (;;) {
            sprintf(write_buf, "I send you %d times.\n", count);
            if (write(write_fd, write_buf, strlen(write_buf)) !=
                strlen(write_buf))
                err_exit("Child 1 write");
            sleep(1);
            count++;
        }

        /* Parent pid */
    default:
        switch (pid2 = fork()) {
            case -1:
                err_exit("Second fork");

                /* Child 2 - read data */
            case 0:
                /* Ignore SIGINT */
                signal(SIGINT, SIG_IGN);

```

```

    /* Catch SIGUSR2 */
    signal(SIGUSR2, sigusr_handler);

    if ((read_fd = open(FIFO, O_RDONLY)) == -1)
        err_exit("FIFO reader open");

    for (;;) {
        read_num = read(read_fd, read_buf, BUF_SIZE);
        if (read_num == -1)
            err_exit("Child 2 read");
        if (read_num == 0)
            continue;
        if (write(STDOUT_FILENO, read_buf, read_num) != read_num)
            err_exit("Child 2 write");
    }

    /* Parent pid */
    default:
        /* Wait for child pid */

        waitpid(pid1, &wait_tmp, 0);
        waitpid(pid1, &wait_tmp, 0);
        printf("Parent Process is Killed!\n");
        exit(0);
    }
}
}

```

C.3 实验二代码

文件名: sync.c

作用: 实现线程同步

源码:

```

/*
 * HUST OS Lab 02 - Thread synchronization
 *
 * By Pan Yue
 */

#include "../zxcpyplib/zxcryp_sys.h"
#include "../zxcpyplib/zxcryp_sem.h"

```

```

#include <pthread.h>

/* Semaphores defines */
#define COMPUTE 0
#define PRINT 1

/* The global sum */
int sum = 0;
/* Semaphores */
int semid;

void* compute(void *arg) {
    for (int i = 1; i <= 100; i++) {
        if (sem_p(semid, COMPUTE) == -1)
            err_exit("P compute");

        sum += i;

        if (sem_v(semid, PRINT) == -1)
            err_exit("V print");
    }
    return NULL;
}

void* print(void *arg) {
    for (int i = 1; i <= 100; i++) {
        if (sem_p(semid, PRINT) == -1)
            err_exit("P print");

        printf("The sum is: %d\n", sum);

        if (sem_v(semid, COMPUTE) == -1)
            err_exit("V compute");
    }
    return NULL;
}

int main(void) {
    /* Thread 0: Computr, Thread 1: Print */
    pthread_t thread[2];
    void *thread_result;
    /* Create and initial 2 semaphore for user */

```

```

if ((semid = semget(IPC_PRIVATE, 2, S_IRUSR | S_IWUSR)) == -1)
    err_exit("Get semaphore");
if (init_sem_available(semid, COMPUTE) == -1)
    err_exit("Initial compute semaphore");
if (init_sem_in_use(semid, PRINT) == -1)
    err_exit("Initial print semaphore");

/* Create compute thread */
if (pthread_create(&thread[0], NULL, compute, NULL) != 0)
    err_exit("Compute thread create");

/* Create print thread */
if (pthread_create(&thread[1], NULL, print, NULL) != 0)
    err_exit("Print thread create");

pthread_join(thread[0], &thread_result);
pthread_join(thread[1], &thread_result);
if (semctl(semid, IPC_RMID, 0) == -1)
    err_exit("Delete semaphore");
return 0;
}

```

C.4 实验三代码

文件名: filecp.c

作用: 实验三头文件

源码:

```

/*
 * HUST OS Lab 03 - File copy
 *
 * By Pan Yue
 */

#include "../zxcpyplib/zxcryp_sys.h"
#include "../zxcpyplib/zxcryp_sem.h"
#include "../zxcpyplib/ring_buf.h"
#include <wait.h>

/* Semaphores defines */
#define EMPTY 0
#define FULL 1

```

```
#define MUTEX 2

/* Semaphores */
int semid;

/* Functions */

/*
 * read_buf - copy buffer from src to ringbuf
 */
int read_buf(int read_fd, int shmid_tail, int semid);

/*
 * write_buf - copy buffer from ring buf to dst
 */
int write_buf(int write_fd, int shmid_head, int semid);
```

文件名: filecp.c

作用: 实现文件拷贝

源码:

```
/*
 * HUST OS Lab 03 - File copy
 *
 * By Pan Yue
 */

#include "filecp.h"

int main(int argc, char **argv) {
    int read_fd, write_fd;
    int shmid_head;
    int wait_tmp;
    pid_t readpid_id, writepid_id;
    /* Check args */
    if (argc != 3)
        usage_err("./filecp <src> <dst>");

    /* Open files */
    read_fd = open(argv[1], O_RDONLY);
```

```

write_fd = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR
| S_IEXEC);

/* Create and initial 3 semaphore for user */
if ((semid = semget(IPC_PRIVATE, 3, S_IRUSR | S_IWUSR)) == -1)
    err_exit("Get semaphore");
if (init_sem_in_use(semid, EMPTY) == -1)
    err_exit("Initial empty");
if (init_sem_value(semid, FULL, RING_BUF_NUM) == -1)
    err_exit("Initial full");
if (init_sem_available(semid, MUTEX) == -1)
    err_exit("Initial mutex");

/* Create a ring buffer
 * The pointer is shmid
 */
if ((shmid_head = shmget(IPC_PRIVATE, sizeof(ring_buf), IPC_CREAT |
S_IRUSR | S_IWUSR)) == -1)
    err_exit("Shared memory get");
ring_buf *ring_buf_tmp = (ring_buf*)shmat(shmid_head, NULL, 0);
if (ring_buf_tmp == (void*)-1)
    err_exit("Shared memory attach");
for (int i = 0; i < RING_BUF_NUM; i++) {
    int new_shmid;
    if ((new_shmid = shmget(IPC_PRIVATE, sizeof(ring_buf), IPC_CREAT |
S_IRUSR | S_IWUSR)) == -1)
        err_exit("Shared memory get");
    ring_buf_tmp->next_shmid = new_shmid;
    ring_buf_tmp->status = STATUS_AVAILABLE;
    ring_buf_tmp = (ring_buf*)shmat(new_shmid, NULL, 0);
    if (ring_buf_tmp == (void*)-1)
        err_exit("Shared memory attach");
}
ring_buf_tmp->next_shmid = shmid_head;

/* Fork reader process */
switch(readpid_id = fork()) {
    case -1:
        err_exit("First fork");

    /* Child 1 - read data */
    case 0:

```

```

    read_buf(read_fd, shmid_head, semid);
    exit(0);

    /* Parent pid */
    default:
        switch(writepid_id = fork()) {
            case -1:
                err_exit("Second fork");

                /* Child 2 - write data */
            case 0:
                write_buf(write_fd, shmid_head, semid);
                exit(0);

                /* Parent pid */
            default:
                waitpid(readpid_id, &wait_tmp, 0);
                waitpid(writepid_id, &wait_tmp, 0);
                if (semctl(semid, semid, IPC_RMID, 0) == -1)
                    err_exit("Delete semaphore");
                exit(0);
        }
    }
}

int read_buf(int read_fd, int shmid_tail, int semid) {
    int read_num;
    ring_buf *ring_buf_tail;
    /* Attach ring buffer tail for read in */
    if ((ring_buf_tail = (ring_buf*)shmat(shmid_tail, NULL, 0)) == (void*)-1)
        err_exit("read_buf: Shared memory attach");
    /* Reading */
    for (;;) {
        sem_p(semid, FULL);
        sem_p(semid, MUTEX);
        /* Read data */
        read_num = read(read_fd, ring_buf_tail->data, RING_BUF_LEN);
        if (read_num == -1)
            err_exit("read_buf: Read");
        /* Read finished */
        if (read_num == 0) {
            ring_buf_tail->status = STATUS_FINISH;

```



```

    ring_buf_tail->size = read_num;
    close(read_fd);
    sem_v(semid, EMPTY);
    sem_v(semid, MUTEX);
    return 0;
}
ring_buf_tail->size = read_num;
/* Attach next */
shmid_tail = ring_buf_tail->next_shmid;
if ((ring_buf_tail = (ring_buf*)shmat(shmid_tail, NULL, 0)) ==
(void*)-1)
    err_exit("read_buf: Shared memory attach");
sem_v(semid, MUTEX);
sem_v(semid, EMPTY);
}
}

int write_buf(int write_fd, int shmid_head, int semid) {
    ring_buf *ring_buf_head;
    /* Attach ring buffer head for write out */
    if ((ring_buf_head = (ring_buf*)shmat(shmid_head, NULL, 0)) == (void*)-1)
        err_exit("write_buf: Shared memory attach");
    /* Writting */
    for (;;) {
        sem_p(semid, EMPTY);
        sem_p(semid, MUTEX);
        /* Write data */
        /* Write finished */
        if (ring_buf_head->status == STATUS_FINISH) {
            write(write_fd, ring_buf_head->data, ring_buf_head->size);
            close(write_fd);
            sem_v(semid, MUTEX);
            sem_v(semid, FULL);
            return 0;
        }
        write(write_fd, ring_buf_head->data, ring_buf_head->size);
        shmid_head = ring_buf_head->next_shmid;
        if ((ring_buf_head = (ring_buf*)shmat(shmid_head, NULL, 0)) ==
(void*)-1)
            err_exit("write_buf: Shared memory attach");
        sem_v(semid, MUTEX);
        sem_v(semid, FULL);
    }
}

```

```
}
}
```

C.5 实验四代码

文件名: directory.c

作用: 实现 ls -lR

源码:

```
/*
 * HUST OS Lab 04 - Directory
 *
 * By Pan Yue
 */

#include "../zxcpyplib/zxcryp_sys.h"
#include <dirent.h>
#include <locale.h> /* Set sorting order using system locale sort */
#include <pwd.h>     /* Get username */
#include <grp.h>     /* Get groupname */
#include <time.h>    /* Get last modify time */

#define LS_BLOCK_SIZE 1024

/* Cmp function for quick sort */
int dir_cmp(const void *a, const void *b) {
    struct dirent *dir_a = (struct dirent*)a;
    struct dirent *dir_b = (struct dirent*)b;
    /* Compare for Chinese */
    return strcoll(dir_a->d_name, dir_b->d_name);
}

/* Print status of a file */
void print_status(char *file, int link_num, int size_num) {
    char modstr[11];
    char *time;
    struct stat s_buf;
    struct passwd *pwd_buf;
    struct group *group_buf;
    if (stat(file, &s_buf) != 0)
        err_exit("Stat");
    /* Set mod str */
```

```

strcpy(modstr, "-----");
/* File type */
if (S_ISDIR(s_buf.st_mode))
    modstr[0] = 'd';
    if (S_ISCHR(s_buf.st_mode))
        modstr[0] = 'c';
    if (S_ISBLK(s_buf.st_mode))
        modstr[0] = 'b';
if (S_ISFIFO(s_buf.st_mode))
    modstr[0] = 'p';
/* User mod */
if (s_buf.st_mode & S_IRUSR)
    modstr[1] = 'r';
if (s_buf.st_mode & S_IWUSR)
    modstr[2] = 'w';
if (s_buf.st_mode & S_IXUSR)
    modstr[3] = 'x';
/* Group mod */
if (s_buf.st_mode & S_IRGRP)
    modstr[4] = 'r';
if (s_buf.st_mode & S_IWGRP)
    modstr[5] = 'w';
if (s_buf.st_mode & S_IXGRP)
    modstr[6] = 'x';
/* Other mod */
if (s_buf.st_mode & S_IROTH)
    modstr[7] = 'r';
if (s_buf.st_mode & S_IWOTH)
    modstr[8] = 'w';
if (s_buf.st_mode & S_IXOTH)
    modstr[9] = 'x';
/* Get username */
if ((pwd_buf = getpwuid(s_buf.st_uid)) == NULL)
    err_exit("Username get");
if ((group_buf = getgrgid(s_buf.st_gid)) == NULL)
    err_exit("Groupname get");
/* Get Last modify time */
time = ctime(&s_buf.st_mtime);
/* Print file info */
printf("%s %d %s %s %ld %.12s %-s\n", \
        modstr, link_num, (int)s_buf.st_nlink, pwd_buf->pw_name,
        group_buf->gr_name, size_num, (long)s_buf.st_size, time + 4, file);

```

```

}

/* List all files */
void list_files(char *fullpath, char *path) {
    struct dirent *entry = NULL;
    struct stat s_buf;

    DIR *directory = opendir(path);
    if (directory == NULL)
        err_exit("Open dir");

    /* Change path */
    chdir(path);
    if (strcmp(path, fullpath) != 0) /* Print '\n' if not first */
        printf("\n");
    printf("%s:\n", fullpath);

    /* Get dirent num */
    int entry_count = 0;
    while ((entry = readdir(directory)) != NULL) {
        /* Skip . and .. */
        if (strncmp(entry->d_name, ".", 1) == 0)
            continue;
        entry_count++;
    }

    /* Get dirent struct */
    int pos = 0;
    rewinddir(directory);
    struct dirent *entry_bufs = (struct dirent*)malloc(sizeof(struct dirent)
* entry_count);
    while ((entry = readdir(directory)) != NULL) {
        /* Skip . and .. */
        if (strncmp(entry->d_name, ".", 1) == 0)
            continue;
        memcpy(entry_bufs + pos, entry, sizeof(struct dirent));
        pos++;
    }

    /* Quick sort */
    qsort(entry_bufs, entry_count, sizeof(struct dirent), dir_cmp);
}

```

```

/* Count total blocks */
int total = 0;
int link_max = 0, size_max = 0;
int link_num = 0, size_num = 0;
for (pos = 0; pos < entry_count; pos++) {
    entry = entry_bufs + pos;
    stat(entry->d_name, &s_buf);
    /* Get max link num and size num */
    if (s_buf.st_nlink > link_max)
        link_max = s_buf.st_nlink;
    if (s_buf.st_size > size_max)
        size_max = s_buf.st_size;
    /* Add total blocks */
    total += s_buf.st_blocks;
}
/* Count num of max link num and max size num */
for (; link_max; link_max /= 10)
    link_num++;
for (; size_max; size_max /= 10)
    size_num++;

printf("total %d\n", total * 512 / LS_BLOCK_SIZE);

/* List file */
for (pos = 0; pos < entry_count; pos++) {
    entry = entry_bufs + pos;
    print_status(entry->d_name, link_num, size_num);
}

/* Recursive */
for (int pos = 0; pos < entry_count; pos++) {
    entry = entry_bufs + pos;
    /* Link new path */
    char *new_path = (char*)malloc(sizeof(*fullpath) + 2 +
sizeof(entry->d_name));
    strcpy(new_path, fullpath);
    strcat(new_path, "/");
    strcat(new_path, entry->d_name);
    if (entry->d_type == 4)
        list_files(new_path, entry->d_name);
}
chdir("../");

```

```

    closedir(directory);
}

int main(int argc, char **argv) {

    /* Check args */
    if (argc != 2) {
        usage_err("./directory <path>");
        exit(1);
    }

    /* Set sorting order using system locale sort
     * LC_COLLATE: local sort order
     */
    setlocale(LC_COLLATE, "");

    /* Get path of the file */
    struct stat s_buf;
    char *path = (char *)malloc(sizeof(argv[1]) + 1);
    strcpy(path, argv[1]);

    /* Judge if this is a dir */
    stat(path, &s_buf);
    if (S_ISDIR(s_buf.st_mode))
        list_files(path, path);
    else {
        int link_num = 0;
        int size_num = 0;
        for (; s_buf.st_nlink; s_buf.st_nlink /= 10)
            link_num++;
        for (; s_buf.st_size; s_buf.st_size /= 10)
            size_num++;
        print_status(path, link_num, size_num);
    }
    return 0;
}

```

附录 D TinyOS 实验代码

D.1 实验一代码

文件名: BlinkC.nc

源码:

```
/*
 * HUST IOT TinyOS Lab Part I - Binary 0 - 7
 *
 * By Pan Yue, modified from tinyos/apps/Blink/BlinkC.nc
 */

#include "Timer.h"
#include "printf.h"

module BlinkC @safe()
{
    uses interface Timer<TMilli> as Timer0;
    uses interface Leds;
    uses interface Boot;
}

implementation
{
    uint32_t count = 0;
    event void Boot.booted()
    {
        call Timer0.startPeriodic( 1000 );
    }

    event void Timer0.fired()
    {
        count++;
        printf("count: %d\n", count);
        dbg("BlinkC", "Timer 0 fired @ %s.\n", sim_time_string());
        if ((count & 0x01) == 0x01) {
            call Leds.led0Toggle();
        }
        else if ((count & 0x02) == 0x02) {
            call Leds.led0Toggle();
            call Leds.led1Toggle();
        }
    }
}
```

```

    }
    else if ((count & 0x04) == 0x04) {
        call Leds.led0Toggle();
        call Leds.led1Toggle();
        call Leds.led2Toggle();
    }
    else {
        call Leds.led0Off();
        call Leds.led1Off();
        call Leds.led2Off();
    }
    if (count == 8)
        count = 0;
}
}

```

D.2 实验二代码

这里仅放上实验二最终结果的源码

文件名: BlinkC.nc

源码:

```

/*
 * HUST IOT TinyOS Lab Part IV - Split phase
 *
 * By Pan Yue, modified from tinyos/apps/Blink/BlinkC.nc
 */

#include "Timer.h"
#include "printf.h"

module BlinkC @safe()
{
    uses interface Timer<TMilli> as Timer0;
    uses interface Timer<TMilli> as Timer1;
    uses interface Timer<TMilli> as Timer2;
    uses interface Leds;
    uses interface Boot;
}

implementation
{

```



```

event void Boot.booted()
{
    call Timer0.startPeriodic( 250 );
    call Timer1.startPeriodic( 500 );
    call Timer2.startPeriodic( 1000 );
}

uint32_t i;
task void computeTask()
{
    uint32_t tmp = i;
    for (;i < tmp + 10000 && i < 400001; i++) {}
    if (i > 400000) {
        i = 0;
    }
    else {
        post computeTask();
    }
}

event void Timer0.fired()
{
    dbg("BlinkC", "Timer 0 fired @ %s.\n", sim_time_string());
    printf("LED Toggle: 0\n");
    call Leds.led0Toggle();
    post computeTask();
}

event void Timer1.fired()
{
    dbg("BlinkC", "Timer 1 fired @ %s \n", sim_time_string());
    printf("LED Toggle: 1\n");
    call Leds.led1Toggle();
}

event void Timer2.fired()
{
    dbg("BlinkC", "Timer 2 fired @ %s.\n", sim_time_string());
    printf("LED Toggle: 2\n");
    call Leds.led2Toggle();
}
}

```

D.3 实验三代码

文件名: Sense.h

作用: 定义数据包结构体

源码:

```
/*
 * HUST IOT TinyOS Lab Part EX - Sense
 *
 * By Pan Yue
 */

#ifndef SENSE_H
#define SENSE_H

#define TEMPORARY 0
#define HUMIDITY 1
#define PHOTOVOLTAIC 2

enum {
    AM_SENSEMSG = 6,
    TIMER_PERIOD_MILLI = 250
};

typedef nx_struct SenseMsg {
    nx_uint16_t nodeid;
    nx_uint16_t kind;
    nx_uint16_t data;
} SenseMsg;

#endif
```

文件名: SenseAppC.nc

作用: 感知程序配置文件

源码:

```
/*
 * HUST IOT TinyOS Lab Part EX - Sense
 *
 * By Pan Yue, modified from tinyos/apps/test/TestSerialAppC.nc
 */
```

```
#include "Sense.h"

configuration SenseAppC
{
}

implementation {

    components SenseC, MainC, LedsC, new TimerMilliC(), new DemoSensorC() as
Sensor;
    components new SensirionSht11C();
    components new HamamatsuS1087ParC();
    components ActiveMessageC;
    components new AMSenderC(AM_SENSEMSG);

    SenseC.Boot -> MainC;
    SenseC.Leds -> LedsC;
    SenseC.Timer -> TimerMilliC;
    SenseC.readTemp -> SensirionSht11C.Temperature;
    SenseC.readHumidity -> SensirionSht11C.Humidity;
    SenseC.readPhoto -> HamamatsuS1087ParC;
    SenseC.Packet -> AMSenderC;
    SenseC.AMPacket -> AMSenderC;
    SenseC.AMControl -> ActiveMessageC;
    SenseC.AMSend -> AMSenderC;
}
```

文件名: SenseC.nc

作用: 感知并发送数据包

源码:

```
/*
 * HUST IOT TinyOS Lab Part EX - Sense
 *
 * By Pan Yue, modified from tinynos/apps/test/TestSerialC.nc
 */

#include "Timer.h"
#include "SensirionSht11.h"

module SenseC
{
```

```

uses {
    interface Boot;
    interface Leds;
    interface Timer<TMilli>;
    interface Read<uint16_t> as readTemp;
    interface Read<uint16_t> as readHumidity;
    interface Read<uint16_t> as readPhoto;
    interface Packet;
    interface AMPacket;
    interface AMSend;
    interface SplitControl as AMControl;
}

implementation
{
    // the message struct
    message_t sense_packet;
    bool busy = FALSE;

    // Sampling frequency in binary milliseconds
    #define SAMPLING_FREQUENCY 100

    // Datas
    uint16_t TempData;
    uint16_t HumidityData;
    uint16_t PhotoData;

    event void Boot.booted() {
        call AMControl.start();
    }

    event void AMControl.startDone(error_t err) {
        if (err == SUCCESS) {
            call Timer.startPeriodic(SAMPLING_FREQUENCY);
        }
        else {
            call AMControl.start();
        }
    }

    event void AMControl.stopDone(error_t err) {

```

```

}

event void Timer.fired()
{
    call readTemp.read(); // Read temporary
    call readHumidity.read(); // Read humidity
    call readPhoto.read(); // Read photovoltaic
}

// Temporary handler
event void readTemp.readDone(error_t result, uint16_t val)
{
    if (!busy && result == SUCCESS) {
        SenseMsg *payload = (SenseMsg*) call Packet.getPayload(&sense_packet,
sizeof(SenseMsg));
        if (payload == NULL) {
            return;
        }
        payload->nodeid = TOS_NODE_ID;
        payload->kind = TEMPORARY;
        payload->data = val;
        if (call AMSend.send(AM_BROADCAST_ADDR, &sense_packet,
sizeof(SenseMsg)) == SUCCESS) {
            call Leds.led0Toggle();
            busy == TRUE;
        }
    }
}

// Humidity handler
event void readHumidity.readDone(error_t result, uint16_t val)
{
    if (!busy && result == SUCCESS) {
        SenseMsg *payload = (SenseMsg*) call Packet.getPayload(&sense_packet,
sizeof(SenseMsg));
        if (payload == NULL) {
            return;
        }
        payload->nodeid = TOS_NODE_ID;
        payload->kind = HUMIDITY;
        payload->data = val;
    }
}

```

```

        if (call AMSend.send(AM_BROADCAST_ADDR, &sense_packet,
sizeof(SenseMsg)) == SUCCESS) {
            call Leds.led1Toggle();
            busy == TRUE;
        }
    }
}

// Photovoltaic handler
event void readPhoto.readDone(error_t result, uint16_t val)
{
    if (!busy && result == SUCCESS) {
        SenseMsg *payload = (SenseMsg*)call Packet.getPayload(&sense_packet,
sizeof(SenseMsg));
        if (payload == NULL) {
            return;
        }
        payload->nodeid = TOS_NODE_ID;
        payload->kind = PHOTOVOLTAIC;
        payload->data = val;
        if (call AMSend.send(AM_BROADCAST_ADDR, &sense_packet,
sizeof(SenseMsg)) == SUCCESS) {
            call Leds.led2Toggle();
            busy == TRUE;
        }
    }
}

event void AMSend.sendDone(message_t* msg, error_t err) {
    if (&sense_packet == msg) {
        busy = FALSE;
    }
}
}

```

文件名: ReceiveSerial.java

作用: 接收并分析数据

源码:

```

/*
 * HUST IOT TinyOS Lab Part EX - Sense
 *

```

```

* By Pan Yue, modified from tinys/apps/test/TestSerial.java
*/

import java.io.IOException;

import net.tinys.message.*;
import net.tinys.packet.*;
import net.tinys.util.*;

public class ReceiveSerial implements MessageListener {

    private MoteIF moteIF;

    public ReceiveSerial(MoteIF moteIF) {
        this.moteIF = moteIF;
        this.moteIF.registerListener(new SenseMsg(), this);
    }

    public void messageReceived(int to, Message message) {
        SenseMsg msg = (SenseMsg)message;
        int type = msg.get_kind();
        double tempature;
        double humidity;
        double photo;
        switch(type){
            case 0:
                tempature = -40.1 + 0.01 * msg.get_data();
                System.out.printf("received temperature:%.2f", tempature);

                System.out.println("°C");

                System.out.println();
                break;
            case 1:
                humidity = -4 + 0.0405 * msg.get_data() + (-2.8/1000000)
                *msg.get_data() * msg.get_data();
                System.out.printf("received humidity:%.2f" , humidity);
                System.out.println("%");
                System.out.println();
                break;
            case 2:
                photo = msg.get_data() * 1.5 / 4096 / 10000;
                photo = 0.625 * 1000000 * photo * 1000;

```

```

        System.out.printf("received photo:%.2f" , photo);
        System.out.println("Lux");
        System.out.println();
        break;
    default:
        System.out.println("received unknow data:" +
msg.get_data());
        break;
    }
    try {Thread.sleep(1000);}
    catch (InterruptedException exception) {}
}

private static void usage() {
    System.err.println("usage: ReceiveSerial [-comm <source>]");
}

public static void main(String[] args) throws Exception {
    String source = null;
    if (args.length == 2) {
        if (!args[0].equals("-comm")) {
            usage();
            System.exit(1);
        }
        source = args[1];
    }
    else if (args.length != 0) {
        usage();
        System.exit(1);
    }

    PhoenixSource phoenix;

    if (source == null) {
        phoenix = BuildSource.makePhoenix(PrintStreamMessenger.err);
    }
    else {
        phoenix = BuildSource.makePhoenix(source, PrintStreamMessenger.err);
    }

    MoteIF mif = new MoteIF(phoenix);
    ReceiveSerial serial = new ReceiveSerial(mif);

```



```
}  
}
```

Basestation 取自 TinyOS-2.1.2 例程，这里不再附上源码。

程序中的其他文件详见附件。

参考文献

- [1] Michael Kerrisk, The Linux Programming Interface. 2010
- [2] Andrew S. Tanenbaum, Modern Operating Systems, 3/E. 2009
- [3] Randal E Bryant and David R O'Hallaron, Computer Systems: A Programmer's Perspective, 3/E. 2017
- [4] Philip Levis and David Gay, TinyOS Programming. 2009
- [5] TinyOS Wiki <http://tinyos.stanford.edu/tinyos-wiki/index.php>
- [6] Arch Wiki <https://wiki.archlinux.org/>