

**课程实验报告**

**课程名称： 嵌入式操作系统**

**专业班级： 物联网1601**

**学 号： U201614890**

**姓 名： 徐光磊**

**指导教师： 石柯**

**报告日期： 2018.05.09**

**计算机科学与技术学院**

目录

[实验一 进程控制 3](#_Toc513816163)

[1 实验目的与要求 3](#_Toc513816164)

[2 实验内容 3](#_Toc513816165)

[3 实验过程与结果 3](#_Toc513816166)

[4实验结果分析 4](#_Toc513816167)

[5心得体会与总结 5](#_Toc513816168)

[实验二 线程同步与通信 6](#_Toc513816169)

[1 实验目的与要求 6](#_Toc513816170)

[2 实验内容 6](#_Toc513816171)

[3 实验过程与结果 6](#_Toc513816172)

[4实验结果分析 7](#_Toc513816173)

[5心得体会与总结 7](#_Toc513816174)

[实验三 共享内存与进程同步 9](#_Toc513816175)

[1 实验目的与要求 9](#_Toc513816176)

[2 实验内容 9](#_Toc513816177)

[3 实验过程与结果 9](#_Toc513816178)

[4实验结果分析 10](#_Toc513816179)

[5心得体会与总结 10](#_Toc513816180)

[实验四 TinyOS实验 12](#_Toc513816181)

[1 实验目的与要求 12](#_Toc513816182)

[2 实验内容 12](#_Toc513816183)

[3 实验过程与结果 13](#_Toc513816184)

[4实验结果分析 19](#_Toc513816185)

[5心得体会与总结 20](#_Toc513816186)

[附加实验 Linux文件目录 21](#_Toc513816187)

[1 实验目的与要求 21](#_Toc513816188)

[2 实验内容 21](#_Toc513816189)

[3 实验过程与结果 21](#_Toc513816190)

[4实验结果分析 23](#_Toc513816191)

[5心得体会与总结 25](#_Toc513816192)

[附录 26](#_Toc513816193)

实验一 进程控制

1 实验目的与要求

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。

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。

1. 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；
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。

1. 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 l is Killed by Parent!

Child Process 2 is Killed by Parent!

1. 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!

3 实验过程与结果

1. 调用pipe函数创建一个无名管道，使两个进程通过这个管道来互相传递数据。
2. 调用两次fork函数创建两个子进程。
3. 主进程创建结束子进程后，调用signal函数，监听SIGINT键盘中断信号，并设置为监听到则执行handle函数来完成指定的结束两个子进程的任务，然后调用wait函数等待两个子进程运行结束。
4. 一号子进程进入其主循环（一个死循环），不停地向管道中写入数据。
5. 二号子进程进入其主循环（一个死循环），不停从管道中读取数据，并输出到控制台中。
6. 当控制台中输入SIGINT中断信号后，主进程结束了两个子进程，关闭管道，结束程序。
7. 运行结果：

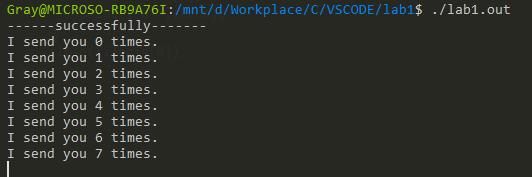


图1.1 程序运行截图

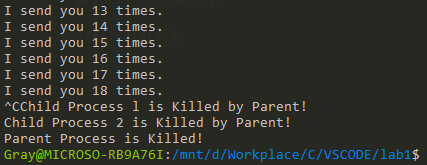


图1.2 自我结束进程截图

4实验结果分析

1. 编译源程序：



图1.3 gcc编译截图

1. **运行过程：**

图1.1，1.2展示了程序的整个运行过程，程序在成功初始化管道、子进程后，按顺序输出了指定的计次字符串。在控制台中按下CTRL+C后，程序成功监听到了SIGINT中断信号，完成了对子进程的结束。

1. **监听SIGINT信号后进行中断：**

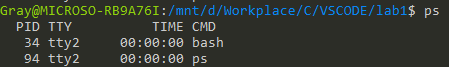


图1.4 后台进程截图

通过ps命令查看后台进程，发现没有实验程序在运行，说明的确成功完成了对进程的结束。

5心得体会与总结

1. 第一次实验中，有大量的Linux C系统库函数是第一次接触，如管道如何创建，使用，读写等等。课程文档上对函数的说明并不是很完整，在搜索了一些较为完整的函数API使用说明后，才摸到了门路。
2. 课程文档中本来是想让我们自定义一个结束进程的函数，但是我在代码中使用了SIGKILL这个特殊信号来直接完成结束进程的工作。进一步体会到Linux中四十多个信号的不同功用。
3. 第一次写C语言的并发代码，感觉虽然通过调用操作系统功能的方式来实现进程的各个功能虽然显得底层一些，但是代码上并不繁杂，内部库的设计还是非常好。

实验二 线程同步与通信

1 实验目的与要求

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 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。

1. 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。
2. 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)。

3 实验过程与结果

1. 主线程中调用semget函数创建一个长度为二的信号量数组。
2. 调用semctl函数来为这个信号量数组完成信号量大小的初始化（第一个初始化为0，第二个初始化为1）。
3. 通过pthread\_create函数来创建两个线程，并绑定上指定的运行函数，然后主线程通过pthread\_join函数阻塞自己等待两个线程运行结束。
4. 一号线程进入100次的循环体，每次循环时，先对第一个信号量进行P操作，再为一个全局变量sum进行加法操作，在对第二个信号量进行V操作。
5. 二号线程进入100次的循环体，每次循环时，先对第二个信号量进行P操作，再输出全局变量sum的值，最后对第一个信号量进行V操作。
6. 待两个线程运行结束后，主线程删除信号量，结束程序
7. 运行结果：

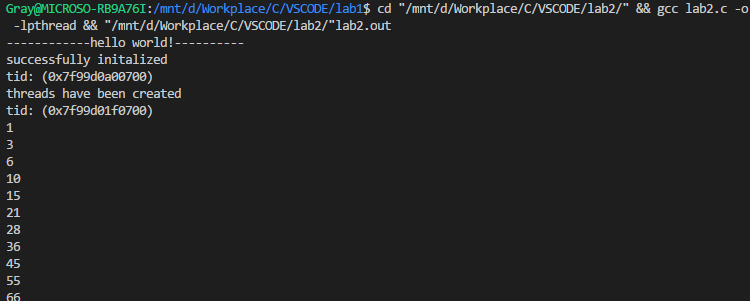


图2.1 程序运行截图

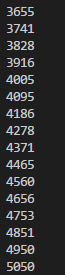


图2.2 程序的输出最后部分截图

4实验结果分析

1. 编译源程序：



图2.3 gcc编译截图

由截图可见，成功完成了编译。

1. **运行结果分析：**

运行的过程截图如图2.1，2.2所示，可见输出的数列符合实验要求中所述的1-100的累加过程。

5心得体会与总结

1. 本次实验关注于线程的同步互斥操作，因为有类似的编程经历，完成代码逻辑的过程并不困难，主要的时间还是花在查阅Linux C的线程库函数API上。
2. 在实验一开始的时候，因为对线程库的功能不是很熟悉，在需要阻塞主线程的时候，不知道可以使用pthread\_join函数来实现，使用了一个全局变量计数器，除非子线程结束的时候更新计数器变量，不然主线程就通过busy-wait 循环的方式进行阻塞，这样实现的方式就不是很优雅。
3. 本程序中的Linux中的信号量P、V操作是使用了semop函数传入结构体的方式，感觉这样的传参方式比较复杂，而且P、V操作的需求仅仅是判断并自加或自减，这样的需求应该有更简单的实现。

实验三 共享内存与进程同步

1 实验目的与要求

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；

3. 掌握Linux下进程同步与通信的主要机制。

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 实验过程与结果

1. 主进程中调用fopen函数打开命令行参数中指定的待读与待写文件。
2. 调用shmget函数来创建一块指定大小的共享内存空间，在这片空间中，分为两块，主要的部分用来做誊抄的缓冲区，另有一个标志位，存有一个数字，用来标记是否读取完毕待读文件的内容，以及最后一个数据块在缓冲区的索引。
3. 调用semget函数来创建一个长度为2的信号量数组，分别为两个信号量赋值大小为0和128（即缓冲区的长度），分别代表缓冲区中有多少数据和有多少空位。
4. 主进程调用两次fork函数创建两个子进程，进行共享内存中的誊抄步骤，然后调用wait函数阻塞自身，等待两个子进程运行结束。
5. 负责将数据从待读文件复制到缓冲区的线程先调用shmat函数获得共享内存空间的地址，为标志位初始化值为-1（即代表本进程尚未完成数据复制）。进入进程的主循环，循环的条件为能通过fread函数读取到数据，在循环体内对信号量2进行P操作，再对共享内存空间的指定位置进行复制，移动索引，最后对信号量1进行V操作。读到最后一个数据的时候，对共享区的索引标志位进行复制。（之所以在互斥区内完成共享索引标志位复制，是为了防止另一个子进程太早读完而则色，没办法得到跳出循环条件，发生死锁）。
6. 负责将数据从缓冲区复制到待写文件的线程先调用shmat函数获得共享内存空间的地址，然后进入其主循环。对信号量1进行P操作，从缓冲区获得数据，调用fwrite函数向待写文件写入数据，判断当前索引是否和标志位中的索引相同，若相同则跳出主循环，最后对信号量2进行V操作。
7. 誊抄结束后，两个子进程也运行结束，主进程结束阻塞，释放共享内存空间，删除信号量，结束程序。
8. 运行结果：

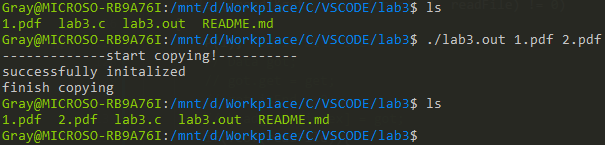


图3.1 程序运行截图

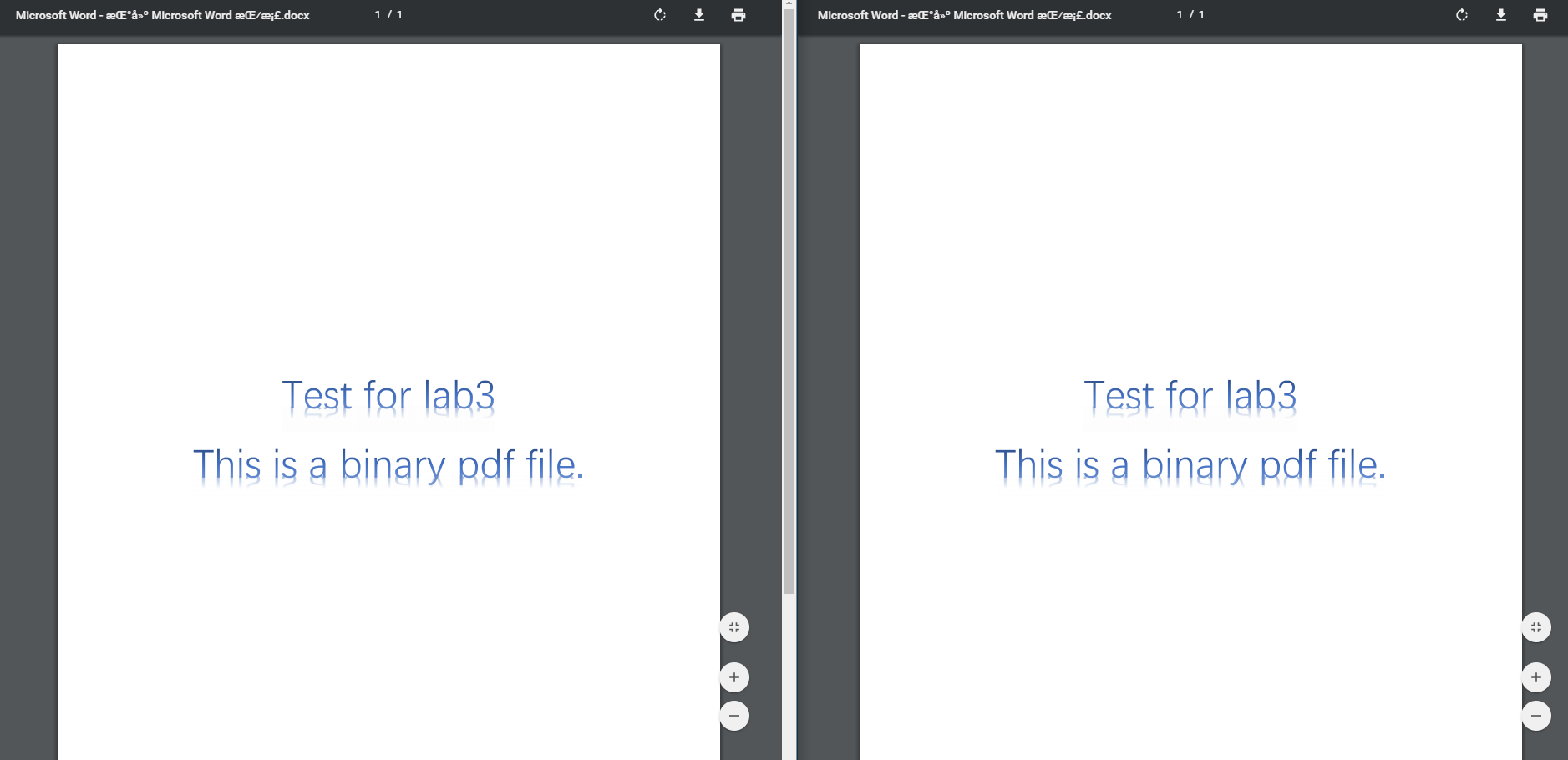


图3.2 复制前后的二进制PDF文件

4实验结果分析

1. 编译源程序：



图3.3 gcc编译截图

由截图可见，成功完成了编译。

1. **运行结果分析：**

运行的过程如图3.1所示，程序提示誊抄成功。通过调用ls函数可以看到在目录中创建了指定的待写目标文件。如图3.2所示，两个实例PDF文件被成功地复制了。

5心得体会与总结

1. 本次实验中比较有意思的环节是如何让从缓冲中复制数据到代写文件的进程知道什么时候应该停止下来，也就是两个进程通信的地方。在第一次写的时候，我采用了向缓冲区写入一个’\0’字符来作为标识符。在老师指出后，我意识到这样的方法并不适合二进制文件的复制。在第二次实现的时候，我用了一个比较暴力的方法，即每个缓冲区的位置都存入一个完整的结构体，这个结构体带有数据和标志位，每次读取这个结构体的时候，都查看标志位以此来检查这个结构体是不是最后一个数据。这个方法的代价就是要花一半的共享内存空间在保存大量的标志位上。最后使用的方法则是上文中提到的只设置一个标志位的方式，用这个标志位来保存最后一个数据的索引，初始化的时候设置为-1，防止冲突。每次进程都去检查一下标志索引是不是和当前索引相同，若相同则说明读到了最后一个数据。
2. 缓冲技术的应用非常广泛，不仅在本次实验中的誊抄程序中，网络通信等等场景中都有大量的缓冲，能够有效地平衡不同设备的不同的速度，从而提高总效率。
3. Linux中进程通信手段不像线程可以直接共享全局变量，需求还是比较苛刻。

实验四 TinyOS实验

1 实验目的与要求

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. 了解Telosb节点中传感器的类型与使用；

6. 了解Telosb节点的传感器数据的获取；

7. 获取的数据通过printf传输至电脑；

8. 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。

2 实验内容

1. （1）Blink程序的编译和下载

（2）给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）修改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。

1. 修改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());  
        for(i=0;i<400001;i++)  
        call Leds.led0Toggle();  
      }

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，加入printf进行输出。

1. 采用task实现计算

uint32\_t i;  
      task void computeTask()  
      {  
        for (i=0;i 400001; i++) {}  
      }  
      event void Timer0.fired()  
      {  
        dbg("BlinkC", "Timer 0 fired @ %s.\n", sim\_time\_string());  
        call Leds.led0Toggle();  
        post computeTask();  
      }

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。

1. 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。
2. 使用节点中的各个传感器，通过温度、湿度、光强强度计算，将数据反馈到电脑端进行输出。

3 实验过程与结果

1. 计数：
2. 基于原有的Blink例程，修改Makefile文件，添加入添加printf组件必需的一些设置（包括缓冲区的大小，组件代码实现地址等）。
3. 在组件实现文件中加入，添加入头文件声明，组件声明。
4. 修改主代码文件中，修改成只有一个计时器，每次计时完毕调用对应fired回调函数的时候，对一个全局计数全局变量进行自加。通过对这个全局变量的多个判断语句，对特定的Led灯组件调用led\*On, led\*Off函数来控制灯组的亮灭。
5. 运行结果：

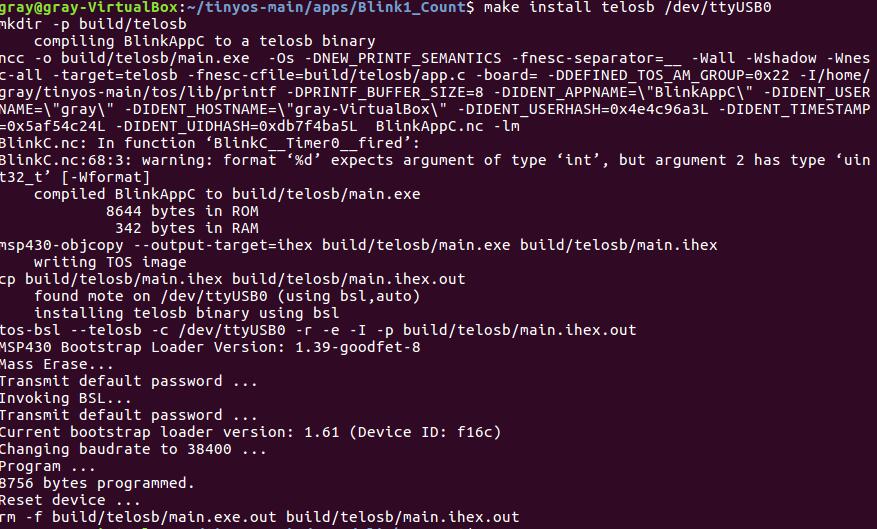


图4.1 程序编译烧录截图

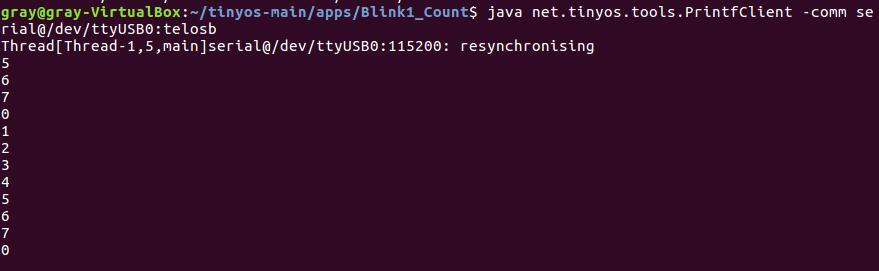


图4.2 电脑端程序监听串口数据截图

1. 循环：
2. 在计数程序上进行修改，去除原自加计数的逻辑，在fired回调函数中添加一个循环次数为400001的for循环，以此模拟复杂计算的耗时。
3. 将循环次数修改为10001后再次进行测试运行。
4. 运行结果：

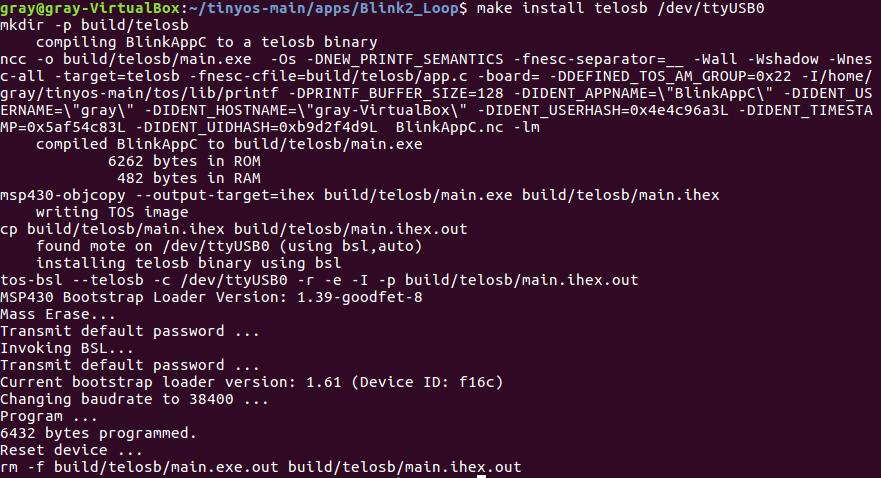


图4.3 程序编译烧录截图

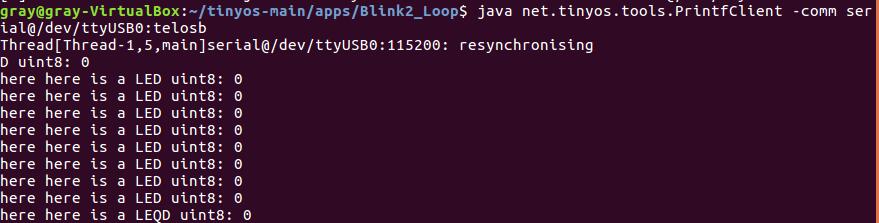


图4.4 电脑端程序监听串口数据（循环400001次）截图

1. 任务：
2. 在循环程序上进行修改，将原有的400001次for循环的逻辑抽分到外部，通过调用一个task来进行实现。
3. 将循环次数修改为10001。
4. 运行结果：

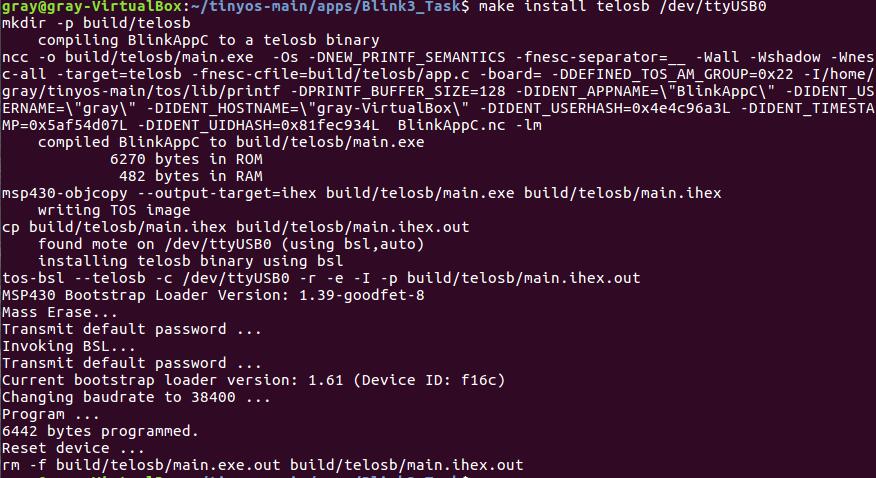


图4.5 程序编译烧录截图

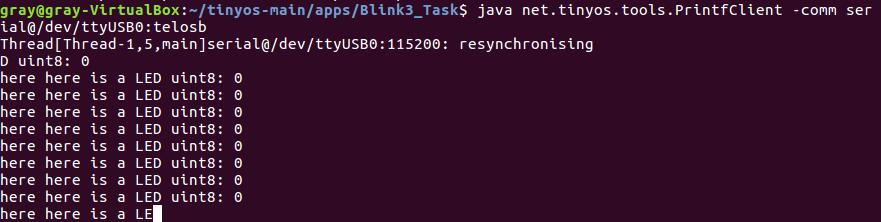


图4.6 电脑端程序监听串口数据（循环400001次）截图

1. 切分任务：
2. 在任务程序上进行修改，将task中的400001次循环模拟复杂计算的过程修改切分为400个循环1000次的小任务。从而使得程序的运行逻辑恢复正常。
3. 运行结果：

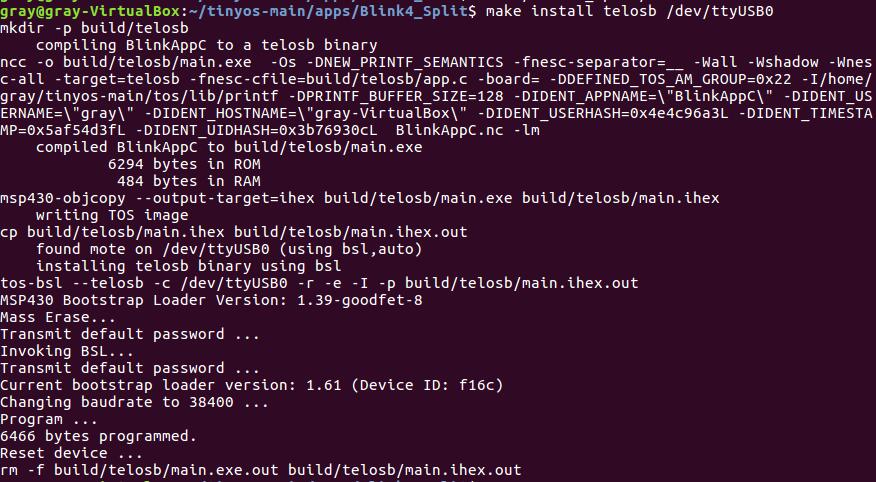


图4.7 程序编译烧录截图

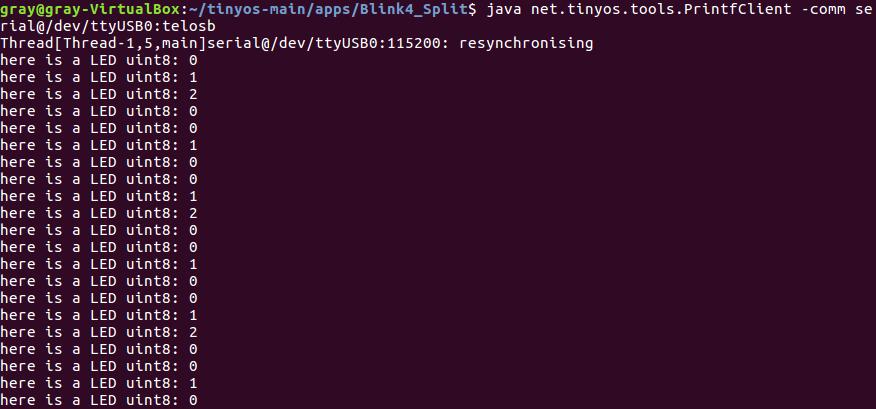


图4.8 电脑端程序监听串口数据截图

1. 传感器读取：
2. Pirntf版本：
3. AppC配置文件中，添加DemoSensorC，SerialStartC，SensirionSht11C，HamamatsuS1087ParC，Printf组件，将程序与传感器的温度、湿度、光照器件链接。
4. Makefile中，对Printf组件进行缓冲区大小、库文件路径等进行配置。
5. 主程序文件中，booted回调函数中为计时器设定频率，fired回调函数中调用三个传感器部件的read功能。每个传感器的readDone回调中，都在内部完成对读取值的预处理后，通过调用Printf，通过串口向外输出字符串，最后再修改Led灯的点亮情况。
6. ActiveMessage版本：
7. 在Makefile中，为JNI通信配置配对指定的nesC结构体和Java类。
8. 在AppC配置文件中，添加DemoSensorC，SensirionSht11C，HamamatsuS1087ParC，SerialActiveMessageC，SerialStartC组件，将程序与所需的组件们进行连接。对ActiveMessage进行管理、包等的配置。
9. 主程序文件中，booted回调中，开启控制器。在控制器成功开启后为计时器初始化计时频率。
10. 每次计时器fired回调后，调用是三个传感器的read功能。
11. 每个传感器的readDone内，调用Packet.getPayload获得一个待发送的包，为包赋值为读取到的信息，然后调用AMSend.send发送这个数据包，切换对应的Led灯状态。
12. 通过一个全局flag变量保证同一时间只有一个包被发送，当包发送结束（AMSend.sendDone回调 ）后才恢复全局flag变量。
13. 电脑端监听的Java Client中，通过命令行命令读取各项参数，初始化包括监听在内的串口各项配置。在messageReceived回调函数中，获得TinyOS发送的数据包，根据不同的种类，对数据值进行不同的处理，最后完成输出。

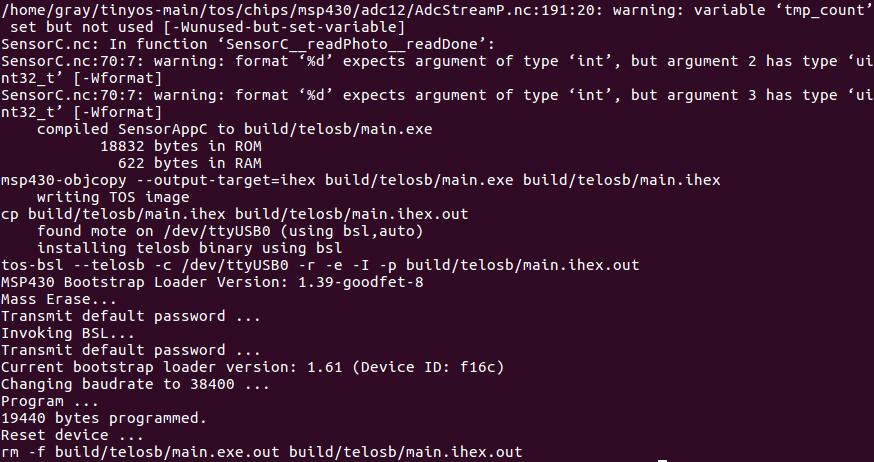


图4.9 程序编译烧录截图



图4.10 电脑端程序监听串口数据截图

4实验结果分析

1. **计数：**

根据图4.1,4.2所示，程序成功编译烧录，从电脑端监听的串口数据来看，也成功读取到了0-7的循环计数。从传感器节点的Led灯组的结果来看，也成功按顺序展示了0-7对应的二进制码

1. **循环：**

根据图4.3,4.4所示，程序成功编译烧录。从电脑端监听的串口数据来看，计时器每次只有0号被调用。从传感器节点的Led灯组的结果来看，也只有0号Led灯亮起。可见程序整个的时间安排，被复杂计算的时间完全打乱。

1. **任务：**

根据图4.5,4.6所示，程序成功编译烧录。从电脑端监听的串口数据来看，计时器每次也只有0号被调用。从传感器节点的Led灯组的结果来看，也只有0号Led灯亮起。可见就算把复杂计算放在任务中，时间也被完全打乱。

1. **切分任务：**

根据图4.7,4.8所示，程序成功编译烧录。从电脑端监听的串口数据来看，正常输出了计时器的轮转输出数据。从传感器界定啊的Led灯组来看，灯组亮起的频率也和程序中指定的频率相同。可见把复杂任务切分成多个小任务，解决了打乱时间线的问题。

1. **传感器读取：**

根据图4.9,4.10所示，程序成功编译烧录，电脑监听的串口数据来看，也成功输出了正确的三个传感器读取的环境数据。从Led灯组的亮灭强开来看，也遵循了正确的频率设置。用物体遮挡传感器后，感光数据也发生了从40到6 lux的变化（见图4.10的红圈位置）。

5心得体会与总结

1. TinyOS实验使用了nesC来进行开发，和C语言完全不同的设计方式，使用了大量的组件和回调。在前四个实验中，大部分的工作都是在原有的Blink上进行修改，其实只要理解程序的运行逻辑的前提下，修改起来其实并没有什么难度。
2. 读取传感器数据的实验中，直接采用Printf的方法比较简单，但是把计算预处理的过程放在了计算能力薄弱的TinyOS传感器节点中，而且Printf的格式化输出能力相较C stdio标准库的printf来说，缺乏很多功能，需求一些额外的代码来进行维护（比如支持输出小数）。
3. 在使用了ActiveMessage的版本中，通过了JNI生成包来进行通讯，在这个方案中，需要显式声明依赖大量的组件，写起来其实比较麻烦，但是把数据的计算处理过程放在了电脑端，并能传输更复杂的数据结构。
4. 其实Printf这个组件的功能已经提供了电脑端设备和TinyOS设备通过串口进行字符通信的功能，完全可以仿照网络通信中的方式进行一定简化后来进行通讯，比如把输出的信息格式化为json格式，在电脑Java端也能快速解译。

附加实验 Linux文件目录

1 实验目的与要求

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。

2 实验内容

编程实现目录查询功能：

1. 功能类似ls -lR；
2. 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；
3. 父递归显示子目录中的所有文件信息。

3 实验过程与结果

1. 从控制行命令中读取参数，以运行指定的逻辑（以下逻辑以传入了-lR为例）。
2. 调用opendir函数，传入指定路径的文件夹，来获得一个DIR指针，将这个指针循环传入readdir函数即可获得该文件夹内所有的文件路径，对循环体中遍历到的所有文件调用lstat获得这个文件的属性信息结构体，再调用getpwuid函数、getgrgid函数、localtime函数来获得更多数据，将所需数据整理为字符串存入缓存，若遍历到的文件为文件夹，则将这个文件夹路径存入缓存。
3. 通过缓存中的文件属性信息的数据计算占用硬盘块数，然后格式化数据输出。最后关闭当前查询的文件夹。
4. 对缓存中的所有未操作的文件夹文件，递归调用处理的过程（即）。
5. 运行结果：

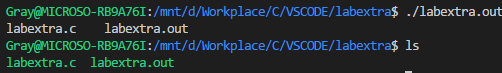


图5.1 无传入参数运行截图

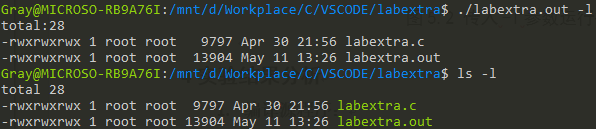


图5.2 传入’-l’参数运行截图

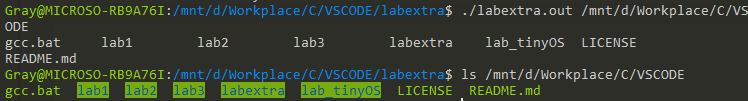
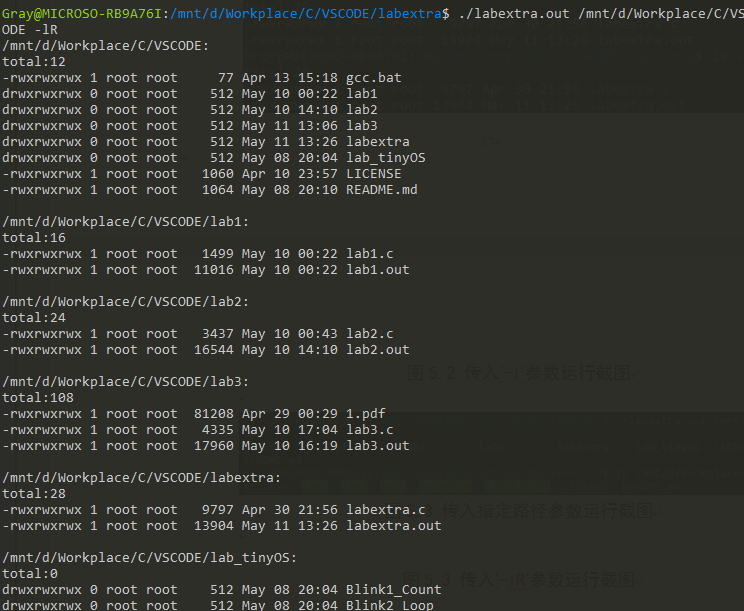


图5.3 传入指定路径参数运行截图



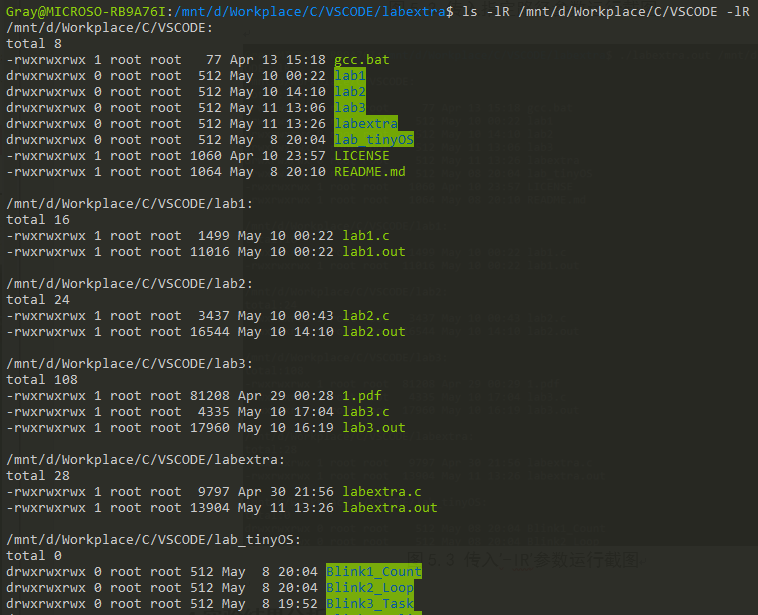


图5.4 传入’-lR’参数运行截图

4实验结果分析

1. 编译源程序：



图5.5 gcc编译截图

由图5.5可见，使用gcc成功完成了编译过程。

1. **运行过程：**
2. 由图5.1可见，在无传入参数的状态下，输出了本文件夹下的所有文件名。
3. 由图5.2可见，在传入’-l’的情况下，程序输出了本文件夹下的所有文件的详细信息，和ls程序的运行情况相同。
4. 由图5.3可见，在传入指定文件夹路径的情况下，程序输出了该文件夹下的所有文件的文件名，且和ls程序的运行情况相同。
5. 由图5.4可见，在传入’-lR’的情况下，程序递归输出了指定文件夹下的所有文件与文件夹的详细信息。
6. **压力测试：**

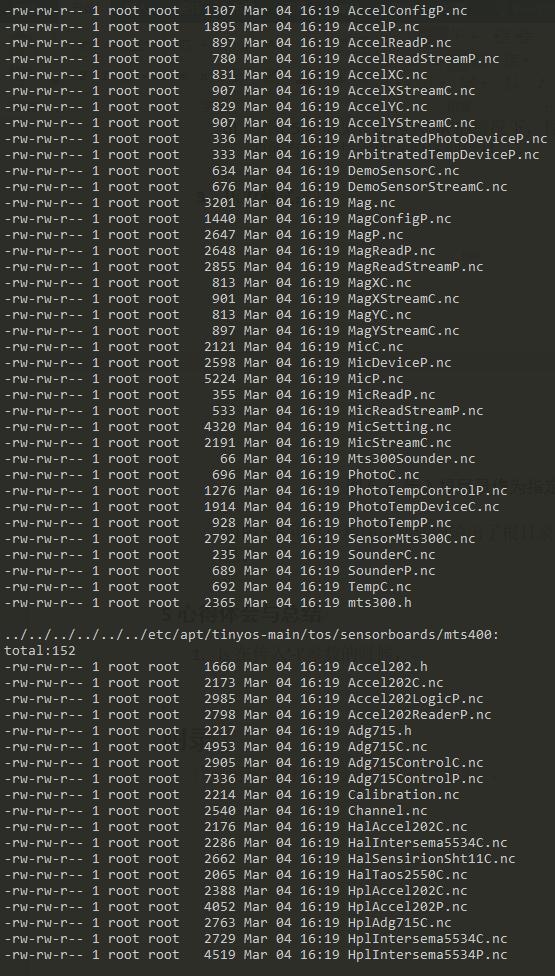


图5.6 传入根目录作为指定路径的参数

由图5.6可见，程序成功递归输出了根目录下的所有的文件的详细信息。

5心得体会与总结

1. ls在传入’-l’参数的时候，会在输出文件夹下文件的详细信息前，会输出本文件夹内所有文件所占用的硬盘块数。在模拟ls程序的过程中，我在计算总块数的时候遇到了不小的问题，通过逐渐的摸索才总结出ls命令在管理文件的时候，默认把硬盘的单块体积设为4096字节，同时计算时默认忽略文件夹、链接文件的体积，过小体积的文件也不被计入。
2. 文件系统是Linux系统中比较复杂的一个环节，在C语言中的库函数系统调用中，整个调用的思路也设计得比较复杂。在搜索了一些有关的例程后，才大概清楚了文件描述结构体的作用。

附录

**lab1.c:**

#include <signal.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

int child1(int \*filedis);

int child2(int \*filedis);

int handle();

int pid1 = 0;

int pid2 = 0;

int pipe\_field[2];// 一读一写 0写1读

int main()

{

// signal(SIGINT,SIG\_IGN);

if (pipe(pipe\_field) < 0)

{

printf("pipe create failed!\n");

return -1;

}

else

{

printf("------successfully-------\n");

pid1 = fork();

if (pid1 == 0)

{

child1(pipe\_field);

// exit(0);

}

pid2 = fork();

if (pid2 == 0)

{

child2(pipe\_field);

// exit(0);

}

//parent

signal(SIGINT,handle);

wait(pid1);

wait(pid2);

//close pipe

close(pipe\_field[0]);

close(pipe\_field[1]);

printf("Parent Process is Killed!\n");

}

return 0;

}

int handle(){

kill(pid1, SIGKILL);

printf("Child Process l is Killed by Parent!\n");

kill(pid2, SIGKILL);

printf("Child Process 2 is Killed by Parent!\n");

}

int child1(int \*filedis)

{

int count = 0;

close(filedis[0]);

while (1)

{

// printf("1 is in loops\n");

char string[50];

sprintf(string, "I send you %d", count);

strcat(string, " times.\n");

write(filedis[1], string, 50);

sleep(1);

count++;

}

close(filedis[1]);

}

int child2(int \*filedis)

{

char cache[50];

cache[0] = 0;

close(filedis[1]);

while (1)

{

// printf("1 is in loops\n");

read(filedis[0], cache, 50);

printf("%s",cache);

}

close(filedis[0]);

}

**lab2.c:**

#include <fcntl.h>

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/sem.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <unistd.h>

void printTid();

void \*myThread();

void delSemvalue();

int setSemvalue();

void \*subp1();

void \*subp2();

int P(int index);

int V(int index);

pthread\_t id1; //thread id

pthread\_t id2;

int sem\_id = 0;

int threadCount = 2;

union semun {

int val;

struct semid\_ds \*buf;

unsigned short \*arry;

};

// struct sembuf

// {

// short sem\_num; // index

// short sem\_op; // Operation for semaphore, -1->P, +1->V.

// short sem\_flg; // Operation flags. Let OS know where is this sem, and delete it when something bad happens

// };

int main()

{

printf("------------hello world!----------\n");

//initalize semaphore (we have two)

sem\_id = semget((key\_t)IPC\_PRIVATE, 2, 0666 | IPC\_CREAT);

if (sem\_id == -1)

{

printf("failed to initalize semaphore\n");

exit(0);

}

if (setSemvalue() == 0)

{

printf("setSemvalue failed\n");

exit(1);

}

printf("successfully initalized\n");

int result1 = pthread\_create(&id1, NULL, subp1, NULL);

int result2 = pthread\_create(&id2, NULL, subp2, NULL);

if (result1 != 0 || result2 != 0)

{

printf("result failed\n");

exit(1);

}

printf("threads have been created\n");

// printTid();

// while (threadCount > 0)

// {

// // sleep(2);

// }

void \*status[2];

pthread\_join(id1, &status[0]);

pthread\_join(id2, &status[1]);

delSemvalue(); //delete

return 0;

}

//init sem's value

int setSemvalue()

{

union semun arg1;

arg1.val = 0;

union semun arg2;

arg2.val = 1;

if (semctl(sem\_id, 0, SETVAL, arg1) == -1)

{

printf("setSemvalue failed\n");

return 0;

}

if (semctl(sem\_id, 1, SETVAL, arg2) == -1)

{

printf("setSemvalue failed\n");

return 0;

}

return 1;

}

void delSemvalue()

{

union semun sem\_union;

if (semctl(sem\_id, 1, IPC\_RMID, sem\_union) == -1)

{

printf("delSemvalue failed\n");

exit(1);

}

}

int sum = 0;

void \*subp1()

{

printTid();

int i;

for (i = 1; i <= 100; i++)

{

P(1);

// printf("thread 1 in loops, %d times\n", i);

sum += i;

V(0);

}

// threadCount--;

return ((void \*)0);

}

void \*subp2()

{

printTid();

int i;

for (i = 1; i <= 100; i++)

{

P(0);

// printf("thread 2 in loops, %d times\n", i);

printf("%d\n", sum);

V(1);

}

// threadCount--;

return ((void \*)0);

}

void printTid()

{

printf("tid: (0x%lx)\n", (unsigned long)pthread\_self());

}

int P(int index)

{

//add 1 to sem

struct sembuf sem\_b;

sem\_b.sem\_num = index;

sem\_b.sem\_op = -1;

sem\_b.sem\_flg = 0;

if (semop(sem\_id, &sem\_b, 1) == -1)

{

printf("P failed\n");

return 0;

}

return 1;

}

int V(int index)

{

//delete 1 from sem

struct sembuf sem\_b;

sem\_b.sem\_num = index;

sem\_b.sem\_op = 1;

sem\_b.sem\_flg = 0;

if (semop(sem\_id, &sem\_b, 1) == -1)

{

printf("V failed\n");

return 0;

}

return 1;

}

**lab3.c**

#include <fcntl.h>

#include <pthread.h>

#include <signal.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/sem.h>

#include <sys/shm.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <unistd.h>

// #include <sys/wait.h>

#define LENGTH 128

int pid1;

int pid2;

int sem\_id = 0;

int shmid;

int shmid2;

FILE\* readFile;

FILE\* writeFile;

void writeChild();

void readChild();

int setSemvalue();

void delSemvalue();

int P(int index);

int V(int index);

union semun {

int val;

struct semid\_ds\* buf;

unsigned short\* arry;

};

int main(int argc, char\* argv[]) {

printf("-------------start copying!----------\n");

if (argc != 3) {

printf("args error\n");

exit(1);

}

readFile = fopen(argv[1], "rb");

writeFile = fopen(argv[2], "wb");

if (readFile == NULL || writeFile == NULL) {

printf("File error.\n");

exit(1);

}

// initalize shared memory

if ((shmid = shmget(IPC\_PRIVATE, (LENGTH + 1) \* sizeof(char),

0666 | IPC\_CREAT)) == -1) // connect

{

printf("Create Share Memory Error");

exit(1);

}

// initalize sem

sem\_id = semget((key\_t)IPC\_PRIVATE, 2, 0666 | IPC\_CREAT);

if (sem\_id == -1) {

printf("failed to initalize semaphore\n");

exit(0);

}

if (setSemvalue() == 0) {

exit(1);

}

printf("successfully initalized\n");

// initalize process

pid1 = fork();

if (pid1 == 0) {

readChild();

exit(0);

}

pid2 = fork();

if (pid2 == 0) {

writeChild();

exit(0);

}

wait(pid1);

wait(pid2);

printf("finish copying\n");

delSemvalue();

// shmdt(head\_addr);//deattach

if (shmctl(shmid, IPC\_RMID, 0) < 0) // release shared memory

{

printf("release error\n");

exit(1);

}

fclose(readFile);

fclose(writeFile);

return 0;

}

void writeChild() {

char\* head\_addr = (char\*)shmat(shmid, 0, 0);

head\_addr[LENGTH] = -1;

int index = 0;

// write

char get = ' ';

while (fread(&get, sizeof(char), 1, readFile) != 0) {

P(1); // init with 1024

head\_addr[index] = get;

// printf("write %d\n", index);

index++;

index = index % LENGTH;

if(feof(readFile))

head\_addr[LENGTH] = index;

V(0);

}

// P(0);

// printf("write end\n");

// head\_addr[LENGTH] = index-1;

// V(0);

}

void readChild() {

char\* head\_addr = (char\*)shmat(shmid, 0, 0);

int index = 0;

// read

while (1) {

P(0); // init with 0

char get = head\_addr[index];

fwrite(&get, sizeof(char), 1, writeFile);

if (index == head\_addr[LENGTH]) {

// printf("read end\n");

break;

}

index++;

index = index % LENGTH;

V(1);

}

}

// init sem's value

int setSemvalue() {

union semun arg1;

arg1.val = 0; // first one refers to "place to write"

union semun arg2;

arg2.val = LENGTH; // refer to "place to read"

if (semctl(sem\_id, 0, SETVAL, arg1) == -1) {

printf("setSemvalue failed\n");

return 0;

}

if (semctl(sem\_id, 1, SETVAL, arg2) == -1) {

printf("setSemvalue failed\n");

return 0;

}

return 1;

}

void delSemvalue() {

union semun sem\_union;

if (semctl(sem\_id, 1, IPC\_RMID, sem\_union) == -1) {

printf("delSemvalue failed\n");

exit(1);

}

}

int P(int index) {

// add 1 to sem

struct sembuf sem\_b;

sem\_b.sem\_num = index;

sem\_b.sem\_op = -1;

sem\_b.sem\_flg = 0;

if (semop(sem\_id, &sem\_b, 1) == -1) {

printf("P failed %d\n", index);

return 0;

}

return 1;

}

int V(int index) {

// delete 1 from sem

struct sembuf sem\_b;

sem\_b.sem\_num = index;

sem\_b.sem\_op = 1;

sem\_b.sem\_flg = 0;

if (semop(sem\_id, &sem\_b, 1) == -1) {

printf("V failed %d\n", index);

return 0;

}

return 1;

}

**Blink1\_Count\_AppC.nc:**

#include "printf.h"

configuration BlinkAppC

{

}

implementation

{

components MainC, BlinkC, LedsC;

components new TimerMilliC() as Timer0;

components PrintfC;

components SerialStartC;

BlinkC -> MainC.Boot;

BlinkC.Timer0 -> Timer0;

BlinkC.Leds -> LedsC;

}

**Blink1\_Count\_C.nc**

#include "Timer.h"

module BlinkC @safe()

{

uses interface Timer<TMilli> as Timer0;

uses interface Leds;

uses interface Boot;

}

implementation

{

uint32\_t i = 0;

event void Boot.booted()

{

call Timer0.startPeriodic( 1000 );

}

event void Timer0.fired()

{

dbg("BlinkC", "Timer 0 fired @ %s.\n", sim\_time\_string());

printf("%d\n",i);

if((i&1) == 1){ call Leds.led0On();}

else call Leds.led0Off();

if((i&2) == 2){ call Leds.led1On();}

else call Leds.led1Off();

if((i&4) == 4){ call Leds.led2On();}

else call Leds.led2Off();

i++;

if(i == 8) i = 0;

}

}

**Blink2\_Loop\_AppC.nc:**

#include "printf.h"

configuration BlinkAppC

{

}

implementation

{

components MainC, BlinkC, LedsC;

components new TimerMilliC() as Timer0;

components new TimerMilliC() as Timer1;

components new TimerMilliC() as Timer2;

components PrintfC;

components SerialStartC;

BlinkC -> MainC.Boot;

BlinkC.Timer0 -> Timer0;

BlinkC.Timer1 -> Timer1;

BlinkC.Timer2 -> Timer2;

BlinkC.Leds -> LedsC;

}

**Blink2\_ Loop \_C.nc**

#include "Timer.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

{

uint32\_t i = 0;

event void Boot.booted()

{

call Timer0.startPeriodic( 250 );

call Timer1.startPeriodic( 500 );

call Timer2.startPeriodic( 1000 );

}

event void Timer0.fired()

{

dbg("BlinkC", "Timer 0 fired @ %s.\n", sim\_time\_string());

for(i = 0;i<10001;i++){

call Leds.led0Toggle();

printf("here is a LED uint8: 0\n");

printfflush();

}

}

event void Timer1.fired()

{

dbg("BlinkC", "Timer 1 fired @ %s \n", sim\_time\_string());

call Leds.led1Toggle();

printf("here is a LED uint8: 1\n");

printfflush();

}

event void Timer2.fired()

{

dbg("BlinkC", "Timer 2 fired @ %s.\n", sim\_time\_string());

call Leds.led2Toggle();

printf("here is a LED uint8: 2\n");

printfflush();

}

}

**Blink3\_Task\_AppC.nc:**

#include "printf.h"

configuration BlinkAppC

{

}

implementation

{

components MainC, BlinkC, LedsC;

components new TimerMilliC() as Timer0;

components new TimerMilliC() as Timer1;

components new TimerMilliC() as Timer2;

components PrintfC;

components SerialStartC;

BlinkC -> MainC.Boot;

BlinkC.Timer0 -> Timer0;

BlinkC.Timer1 -> Timer1;

BlinkC.Timer2 -> Timer2;

BlinkC.Leds -> LedsC;

}

**Blink3\_ Task \_C.nc**

#include "Timer.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

{

uint32\_t i;

task void computeTask(){

for(i=0;i<400001;i++){}//simulate compute cost

}

event void Boot.booted()

{

call Timer0.startPeriodic( 250 );

call Timer1.startPeriodic( 500 );

call Timer2.startPeriodic( 1000 );

}

event void Timer0.fired()

{

dbg("BlinkC", "Timer 0 fired @ %s.\n", sim\_time\_string());

printf("here is a LED uint8: 0\n");

printfflush();

post computeTask();//without this line this program would be normal Blink

call Leds.led0Toggle();

}

event void Timer1.fired()

{

dbg("BlinkC", "Timer 1 fired @ %s \n", sim\_time\_string());

call Leds.led1Toggle();

printf("here is a LED uint8: 1\n");

printfflush();

}

event void Timer2.fired()

{

dbg("BlinkC", "Timer 2 fired @ %s.\n", sim\_time\_string());

call Leds.led2Toggle();

printf("here is a LED uint8: 2\n");

printfflush();

}

}

**Blink4\_Split\_AppC.nc:**

#include "printf.h"

configuration BlinkAppC

{

}

implementation

{

components MainC, BlinkC, LedsC;

components new TimerMilliC() as Timer0;

components new TimerMilliC() as Timer1;

components new TimerMilliC() as Timer2;

components PrintfC;

components SerialStartC;

BlinkC -> MainC.Boot;

BlinkC.Timer0 -> Timer0;

BlinkC.Timer1 -> Timer1;

BlinkC.Timer2 -> Timer2;

BlinkC.Leds -> LedsC;

}

**Blink4\_ Split \_C.nc:**

#include "Timer.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

{

uint32\_t start;

task void computeSmallTask(){

uint32\_t temp = 0;

//int variable start would be updated in another func

for(temp = start;temp<start+1000;temp++){

//do computing

}

}

task void computeTask(){

uint32\_t max = 400001;

uint32\_t iteration = max/1000;

uint32\_t i = 0;

for(i=0;i<iteration;i++){

start = i\*1000;

post computeSmallTask();

}

start = (i+1)\*100;

for(i = iteration\*1000;i<max;i++){

//do computing

}

}

event void Boot.booted()

{

call Timer0.startPeriodic( 250 );

call Timer1.startPeriodic( 500 );

call Timer2.startPeriodic( 1000 );

}

event void Timer0.fired()

{

dbg("BlinkC", "Timer 0 fired @ %s.\n", sim\_time\_string());

printf("here is a LED uint8: 0\n");

printfflush();

post computeTask();//without this line this program would be normal Blink

call Leds.led0Toggle();

}

event void Timer1.fired()

{

dbg("BlinkC", "Timer 1 fired @ %s \n", sim\_time\_string());

call Leds.led1Toggle();

printf("here is a LED uint8: 1\n");

printfflush();

}

event void Timer2.fired()

{

dbg("BlinkC", "Timer 2 fired @ %s.\n", sim\_time\_string());

call Leds.led2Toggle();

printf("here is a LED uint8: 2\n");

printfflush();

}

}

**Makefile(Blink1-4):**

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

PFLAGS += -DNEW\_PRINTF\_SEMANTICS

CFLAGS += -DPRINTF\_BUFFER\_SIZE=128

COMPONENT=BlinkAppC

include $(MAKERULES)

**Sensor\_Printf:**

**Makefile:**

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

PFLAGS += -DNEW\_PRINTF\_SEMANTICS

CFLAGS += -DPRINTF\_BUFFER\_SIZE=128

COMPONENT=SensorAppC

include $(MAKERULES)

**SensorAppC.nc:**

#include "printf.h"

configuration SensorAppC

{

}

implementation

{

components MainC, SensorC, LedsC;

components new TimerMilliC() as Timer0;

components PrintfC;

components SerialStartC;

components new DemoSensorC() as Sensor;

components new SensirionSht11C();

components new HamamatsuS1087ParC();

SensorC -> MainC.Boot;

SensorC.readTemp -> SensirionSht11C.Temperature;

SensorC.readHumidity -> SensirionSht11C.Humidity;

SensorC.readPhoto -> HamamatsuS1087ParC;

SensorC.Timer0 -> Timer0;

SensorC.Leds -> LedsC;

}

**SensorC.nc**

#include "Timer.h"

#include "SensirionSht11.h"

/\*read and print raw data without any process\*/

module SensorC @safe()

{

uses interface Timer<TMilli> as Timer0;

uses interface Leds;

uses interface Boot;

uses interface Read<uint16\_t> as readTemp;

uses interface Read<uint16\_t> as readHumidity;

uses interface Read<uint16\_t> as readPhoto;

}

implementation

{

#define FREQUENCY 1000

uint16\_t tempData;

uint16\_t humidityData;

uint16\_t photoData;

bool locked =FALSE;

event void Boot.booted()

{

call Timer0.startPeriodic(FREQUENCY);

}

event void Timer0.fired()

{

call readTemp.read();

call readHumidity.read();

call readPhoto.read();

}

event void readTemp.readDone(error\_t result, uint16\_t val){

if(result == SUCCESS){

double temp = 10\*(-40.1+0.01\*val);

uint32\_t a = temp/10;

uint32\_t b = temp-temp/10\*10;

printf("temp: %d.%d\n",a,b);

printfflush();

call Leds.led0Toggle();

}

}

event void readHumidity.readDone(error\_t result, uint16\_t val){

if(result == SUCCESS){

double temp = (-4+0.0405\*val+(-2.8/1000000)\*val\*val)\*10;

uint32\_t a = temp/10;

uint32\_t b = temp-temp/10\*10;

printf("humidity: %d.%d%\n",a,b);

printfflush();

call Leds.led1Toggle();

}

}

event void readPhoto.readDone(error\_t result, uint16\_t val){

if(result == SUCCESS){

double temp = 0.0625\*1000000\*(val\*1.5/4096/10000)\*1000\*10;

uint32\_t a = temp/10;

uint32\_t b = temp-temp/10\*10;

printf("photo: %d.%d lux\n",a,b);

printfflush();

call Leds.led2Toggle();

}

}

}

**Sensor\_AM:**

**Makefile:**

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

PFLAGS += -DNEW\_PRINTF\_SEMANTICS

CFLAGS += -DPRINTF\_BUFFER\_SIZE=128

COMPONENT=SensorAppC

BUILD\_EXTRA\_DEPS += SensorMsg.class

CLEAN\_EXTRA = \*.class SensorMsg.java

SensorMsg.class: SensorMsg.java

javac SensorMsg.java

SensorMsg.java:

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

include $(MAKERULES)

**SensorAppC.nc:**

#include "printf.h"

configuration SensorAppC

{

}

implementation

{

components MainC, SensorC, LedsC;

components new TimerMilliC() as Timer0;

components PrintfC;

components SerialStartC;

components new DemoSensorC() as Sensor;

components new SensirionSht11C();

components new HamamatsuS1087ParC();

components SerialActiveMessageC as AM;

SensorC -> MainC.Boot;

SensorC.readTemp -> SensirionSht11C.Temperature;

SensorC.readHumidity -> SensirionSht11C.Humidity;

SensorC.readPhoto -> HamamatsuS1087ParC;

SensorC.Timer0 -> Timer0;

SensorC.Leds -> LedsC;

SensorC.Packet -> AM;

SensorC.AMPacket -> AM;

SensorC.Control -> AM;

SensorC.AMSend -> AM.AMSend[AM\_SENSORMSG];

}

**SensorC.nc:**

#include "Timer.h"

#include "Sensor.h"

#include "SensirionSht11.h"

/\*read and print raw data without any process\*/

module SensorC @safe()

{

uses interface Timer<TMilli> as Timer0;

uses interface Leds;

uses interface Boot;

uses interface Read<uint16\_t> as readTemp;

uses interface Read<uint16\_t> as readHumidity;

uses interface Read<uint16\_t> as readPhoto;

uses interface Packet;

uses interface AMPacket;

uses interface AMSend;

uses interface SplitControl as Control;

}

implementation

{

#define FREQUENCY 100

uint16\_t tempData;

uint16\_t humidityData;

uint16\_t photoData;

message\_t packet;

bool locked =FALSE;

event void Boot.booted()

{

// call Timer0.startPeriodic(FREQUENCY);

call Control.start();

}

event void Control.startDone(error\_t err)

{

if (err == SUCCESS)

call Timer0.startPeriodic(1000);

}

event void Control.stopDone(error\_t err)

{

//do sth

}

event void Timer0.fired()

{

if(locked){

return;

}

call readTemp.read();

call readHumidity.read();

call readPhoto.read();

}

event void readTemp.readDone(error\_t result, uint16\_t val){

if(result == SUCCESS){

SensorMsg \*payload = (SensorMsg\*) call Packet.getPayload(&packet, sizeof(SensorMsg));

if(!payload)return;

payload->nodeid = TOS\_NODE\_ID;

payload->kind = TEMPORARY;//0

payload->data = val;

if (call AMSend.send(AM\_BROADCAST\_ADDR, &packet, sizeof(SensorMsg)) == SUCCESS)

{

call Leds.led0Toggle();

locked = TRUE;

}

}

}

event void readHumidity.readDone(error\_t result, uint16\_t val){

if(result == SUCCESS){

SensorMsg \*payload = (SensorMsg\*) call Packet.getPayload(&packet, sizeof(SensorMsg));

if(!payload)return;

payload->nodeid = TOS\_NODE\_ID;

payload->kind = HUMIDITY;//1

payload->data = val;

if (call AMSend.send(AM\_BROADCAST\_ADDR, &packet, sizeof(SensorMsg)) == SUCCESS)

{

call Leds.led1Toggle();

locked = TRUE;

}

}

}

event void readPhoto.readDone(error\_t result, uint16\_t val){

if(result == SUCCESS){

SensorMsg \*payload = (SensorMsg\*) call Packet.getPayload(&packet, sizeof(SensorMsg));

if(!payload)return;

payload->nodeid = TOS\_NODE\_ID;

payload->kind = 2;//2

payload->data = val;

if (call AMSend.send(AM\_BROADCAST\_ADDR, &packet, sizeof(SensorMsg)) == SUCCESS)

{

call Leds.led2Toggle();

locked = TRUE;

}

}

}

event void AMSend.sendDone(message\_t\* msg, error\_t err)

{

if (&packet == msg)

{

locked = FALSE;

}

}

}

**Sensor.h:**

#ifndef SENSOR\_H

#define SENSOR\_H

#define TEMPORARY 0

#define HUMIDITY 1

#define PHOTOVOLTAIC 2

typedef nx\_struct SensorMsg{

nx\_uint16\_t nodeid;

nx\_uint16\_t kind;

nx\_uint16\_t data;

} SensorMsg;

enum{

AM\_SENSORMSG = 6,

};

#endif

**SensorClient.java:**

import java.io.IOException;

import net.tinyos.message.\*;

import net.tinyos.packet.\*;

import net.tinyos.util.\*;

public class SensorClient implements MessageListener {

private MoteIF moteIF;

public SensorClient(MoteIF moteIF) {

this.moteIF = moteIF;

this.moteIF.registerListener(new SensorMsg(), this);

}

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);

SensorClient serial = new SensorClient(mif);

}

public void messageReceived(int to, Message message) {

SensorMsg msg = (SensorMsg) 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.println("Temperature:" + tempature + "℃");

break;

case 1:

humidity = -4 + 0.0405 \* msg.get\_data() + (-2.8 / 1000000) \* msg.get\_data() \* msg.get\_data();

System.out.println("Humidity:" + humidity + "%");

break;

case 2:

photo = msg.get\_data() \* 1.5 / 4096 / 10000;

photo = 0.625 \* 1000000 \* photo \* 1000;

System.out.println("Photo:" + photo + "Lux");

break;

default:

System.out.println("Unknow data:" + msg.get\_data());

break;

}

// try {

// Thread.sleep(1000);

//} catch (Exception e) {}

}

private static void usage() {

System.err.println("usage: SensorClient [-comm <source>]");

}

}

**labextra.c:**

#include <dirent.h>

#include <grp.h>

#include <pwd.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/ioctl.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <termios.h>

#include <time.h>

#include <unistd.h>

int showDir(char\* dirname, int mode);

char\* getFileInfo(struct stat\* sP, char\* filename, int\* blocks);

char\* num2month(int num);

void printInfoList(char\*\* allFileInfos, int indexForInfos);

void printFormatList(char\*\* allFileNames,

int maxLengthOfFileName,

int indexForFileNames);

int WIDTH = 0; // num of columns of terminal

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

char\* dir = ".";

int mode = 0;

// get width(Columns) of terminal

struct winsize size;

ioctl(STDIN\_FILENO, TIOCGWINSZ, &size);

WIDTH = size.ws\_col;

// read arg's info

if (argc != 1) {

for (int i = 1; i < argc; i++) {

if (argv[i][0] == '-') {

if (strcmp("-lR", argv[i]) == 0) {

mode = 2;

} else {

mode = 1;

}

} else {

dir = argv[i]; // change the path

}

}

}

showDir(dir, mode);

return 0;

}

// called by showDir

// with '-l' further infomation of files

char\* getFileInfo(struct stat\* sP, char\* filename, int\* blocks) {

char\* buf = (char\*)malloc(sizeof(char) \* 1024);

struct stat s = \*sP;

switch (s.st\_mode & S\_IFMT) {

case S\_IFREG:

sprintf(buf, "-");

break;

case S\_IFDIR:

sprintf(buf, "d");

break;

case S\_IFLNK:

sprintf(buf, "l");

break;

case S\_IFBLK:

sprintf(buf, "b");

break;

case S\_IFCHR:

sprintf(buf, "c");

break;

case S\_IFIFO:

sprintf(buf, "p");

break;

case S\_IFSOCK:

sprintf(buf, "s");

break;

}

for (int i = 8; i >= 0; i--) {

if (s.st\_mode & (1 << i)) {

switch (i % 3) {

case 2:

strcat(buf, "r");

break;

case 1:

strcat(buf, "w");

break;

case 0:

strcat(buf, "x");

break;

}

} else {

strcat(buf, "-");

}

}

struct passwd\* p = getpwuid(s.st\_uid);

struct group\* g = getgrgid(s.st\_gid);

char temp[128];

sprintf(temp, " %d %s %s %6ld", (int)s.st\_nlink, p->pw\_name, g->gr\_name,

s.st\_size);

strcat(buf, temp);

struct tm\* t = localtime(&s.st\_ctime);

sprintf(temp, " %s %02d %02d:%02d", num2month(t->tm\_mon + 1), t->tm\_mday,

t->tm\_hour, t->tm\_min);

strcat(buf, temp);

sprintf(temp, " %s\n", filename);

strcat(buf, temp);

int now4Blocks = s.st\_size / 4096;

if (S\_ISLNK(s.st\_mode) || S\_ISDIR(s.st\_mode)) {

now4Blocks = 0;

}else if (s.st\_size % 4096 != 0){

now4Blocks++;

}

\*blocks = \*blocks + now4Blocks\*4;

return buf;

}

// mode: 0-> default

// 1-> '-l'

// 2-> '-lR'

int showDir(char\* dirname, int mode) {

if (mode == 2) {

printf("%s:\n", dirname);

}

int blocks = 0;

DIR\* dir = opendir(dirname);

struct dirent\* dirDescribe;

struct stat st;

char nowDirnameBuf[1024];

/\*for formatting\*/

int indexForFileNames = 0; // index for allFileNames

int maxLengthOfFileName = 0;

char\*\* allFileNames = NULL; // save all file's name to make sure format

if (mode == 0) {

allFileNames = (char\*\*)malloc(sizeof(char\*) \* 2048);

}

/\*for keeping file info (kind of delay for us to get "total block")\*/

int indexForInfos = 0;

char\*\* allFileInfos = NULL;

if (mode != 0) {

allFileInfos = (char\*\*)malloc(sizeof(char\*) \* 2048);

}

/\*for recursion\*/

int indexForDirNames = 0;

char\*\* allDirNames = NULL;

if (mode == 2) {

allDirNames = (char\*\*)malloc(sizeof(char\*) \* 2048);

}

while ((dirDescribe = readdir(dir)) != NULL) { // reading in loop

strcpy(nowDirnameBuf, dirname);

strcat(nowDirnameBuf, "/");

strcat(nowDirnameBuf, dirDescribe->d\_name);

if (lstat(nowDirnameBuf, &st)) {

printf("error\n");

return -1;

}

if (dirDescribe->d\_name[0] != '.') { // hidden files off

if (mode == 0) {

char\* nameBuf = (char\*)malloc(sizeof(char) \* 100);

strcpy(nameBuf, dirDescribe->d\_name);

allFileNames[indexForFileNames] = nameBuf;

indexForFileNames++;

if (strlen(nameBuf) > maxLengthOfFileName)

maxLengthOfFileName = strlen(nameBuf);

} else {

if (mode == 2) {

if (S\_ISDIR(st.st\_mode)) { // is dir or not

char\* nameBuf = (char\*)malloc(sizeof(char) \* 100);

strcpy(nameBuf, nowDirnameBuf);

allDirNames[indexForDirNames] = nameBuf;

indexForDirNames++;

}

}

char\* fileInfo = getFileInfo(&st, dirDescribe->d\_name, &blocks);

allFileInfos[indexForInfos] = fileInfo;

indexForInfos++;

}

}

}

// output formatted infomation when mode is 0

if (mode == 0) {

printFormatList(allFileNames, maxLengthOfFileName, indexForFileNames);

} else {

printf("total:%d\n", blocks);

printInfoList(allFileInfos, indexForInfos);

if (mode == 2) {

printf("\n");

int i = 0;

for (i = 0; i < indexForDirNames; i++) {

showDir(allDirNames[i], 2); // enter recursion here

}

}

}

// free all heap space

if (mode == 0) {

for (int i = 0; i < indexForFileNames; i++) {

free(allFileNames[i]);

}

free(allFileNames);

} else {

if (mode == 2) {

for (int i = 0; i < indexForDirNames; i++) {

free(allDirNames[i]);

}

free(allDirNames);

} else {

for (int i = 0; i < indexForInfos; i++) {

free(allFileInfos[i]);

}

free(allFileInfos);

}

}

closedir(dir);

return 0;

}

char\* num2month(int num) {

switch (num) {

case 1:

return "Jan";

case 2:

return "Feb";

case 3:

return "Mar";

case 4:

return "Apr";

case 5:

return "May";

case 6:

return "Jun";

case 7:

return "Jul";

case 8:

return "Aug";

case 9:

return "Sep";

case 10:

return "Oct";

case 11:

return "Nov";

case 12:

return "Dec";

default:

return "";

}

}

void printFormatList(char\*\* allFileNames,

int maxLengthOfFileName,

int indexForFileNames) {

int i = 0;

int num = WIDTH / (maxLengthOfFileName + 2);

// a little trick here ( guess it wouldn't pass 99

int temp = maxLengthOfFileName;

char format[10];

strcpy(format, "%-");

if (temp > 9) {

format[2] = temp / 10 + 48;

format[3] = temp % 10 + 48;

format[4] = '\0';

} else {

format[2] = temp + 48;

format[3] = '\0';

}

strcat(format, "s ");

for (i = 0; i < indexForFileNames; i++) {

printf(format, allFileNames[i]);

if ((i + 1) % num == 0) {

printf("\n");

}

}

printf("\n");

}

void printInfoList(char\*\* allFileInfos, int indexForInfos) {

for (int i = 0; i < indexForInfos; i++) {

printf("%s", allFileInfos[i]);

}

}