

**课程实验报告**

**课程名称： 嵌入式操作系统**

**专业班级： 物联网工程1601**

**学 号： Ｕ201614898**

**姓 名： 潘翔**

**指导教师： 石柯**

**报告日期： 2018.4.24**

**计算机科学与技术学院**

[实验1 进程控制 4](#_Toc514334882)

[1.1 实验目的与要求 4](#_Toc514334883)

[1.2 实验内容 4](#_Toc514334884)

[1.3 实验过程与结果 4](#_Toc514334885)

[1.4 实验结果分析 5](#_Toc514334886)

[1.5 心得体会与总结 6](#_Toc514334887)

[实验2 线程同步与通信 8](#_Toc514334888)

[2.1 实验目的与要求 8](#_Toc514334889)

[2.2 实验内容 8](#_Toc514334890)

[2.3 实验过程与结果 8](#_Toc514334891)

[2.4 实验结果分析 9](#_Toc514334892)

[2.5 心得体会与总结 9](#_Toc514334893)

[实验3 共享内存与进程同步 11](#_Toc514334894)

[3.1 实验目的与要求 11](#_Toc514334895)

[3.2 实验内容 11](#_Toc514334896)

[3.3 实验过程与结果 11](#_Toc514334897)

[3.4 实验结果分析 12](#_Toc514334898)

[3.5 心得体会与总结 12](#_Toc514334899)

[实验4 TinyOS实验 13](#_Toc514334900)

[4.1 实验目的与要求 13](#_Toc514334901)

[4.1.1 基础实验(Task1-3) 13](#_Toc514334902)

[4.1.2 附加实验(Task4) 13](#_Toc514334903)

[4.2 实验内容 13](#_Toc514334904)

[4.2.1 BlinkBinary(Task1) 13](#_Toc514334905)

[4.2.2 BlinkCompute(Task2) 13](#_Toc514334906)

[4.2.3 BlinkComputeSlicing(Task3) 13](#_Toc514334907)

[4.2.4 TinyOSSensor(Task4) 13](#_Toc514334908)

[4.3 实验过程与结果 14](#_Toc514334909)

[4.2.1 BlinkBinary(Task1) 14](#_Toc514334910)

[4.2.2 BlinkCompute(Task2) 14](#_Toc514334911)

[4.2.3 BlinkComputeSlicing(Task3) 14](#_Toc514334912)

[4.2.4 TinyOSSensor(Task4) 14](#_Toc514334913)

[4.4 实验结果分析 14](#_Toc514334914)

[3.5 心得体会与总结 14](#_Toc514334915)

[实验5 附加实验ls命令的实现 15](#_Toc514334916)

[5.1 实验目的与要求 15](#_Toc514334917)

[5.2 实验内容 15](#_Toc514334918)

[5.3 实验过程与结果 15](#_Toc514334919)

[5.4 实验结果分析 15](#_Toc514334920)

[5.5 心得体会与总结 15](#_Toc514334921)

[附录 16](#_Toc514334922)

[Lab1 code 16](#_Toc514334923)

[Lab2 code 16](#_Toc514334924)

[Lab3 code 16](#_Toc514334925)

[Lab4 code 16](#_Toc514334926)

# 实验1 进程控制

## 1.1 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。

## 1.2 实验内容

1. 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；

1. 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!

## 1.3 实验过程与结果

1. 调用pipe()创建无名管道，用于之后两子进程通信
2. 子进程1进行写管道操作并判断是否成功
3. 子进程2进行读管道操作并判断是否成功
4. 持续上述过程直到收到signal 2(control+c)信号



图1-1 编译C文件

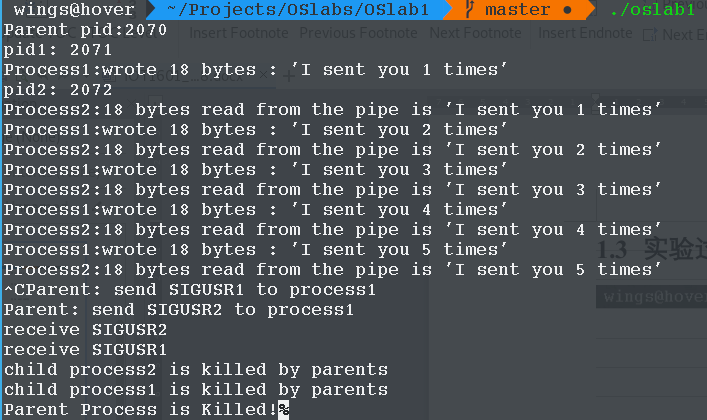


图1-2 执行C文件

## 1.4 实验结果分析

1. 进程12进行管道通信，使用ps aux进行进程查看

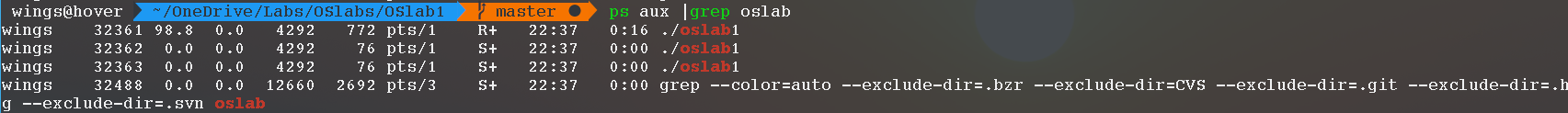


图1-3 进程情况

1. 当收到信号时，从最后的结果来看，进程1，2被父进程杀死

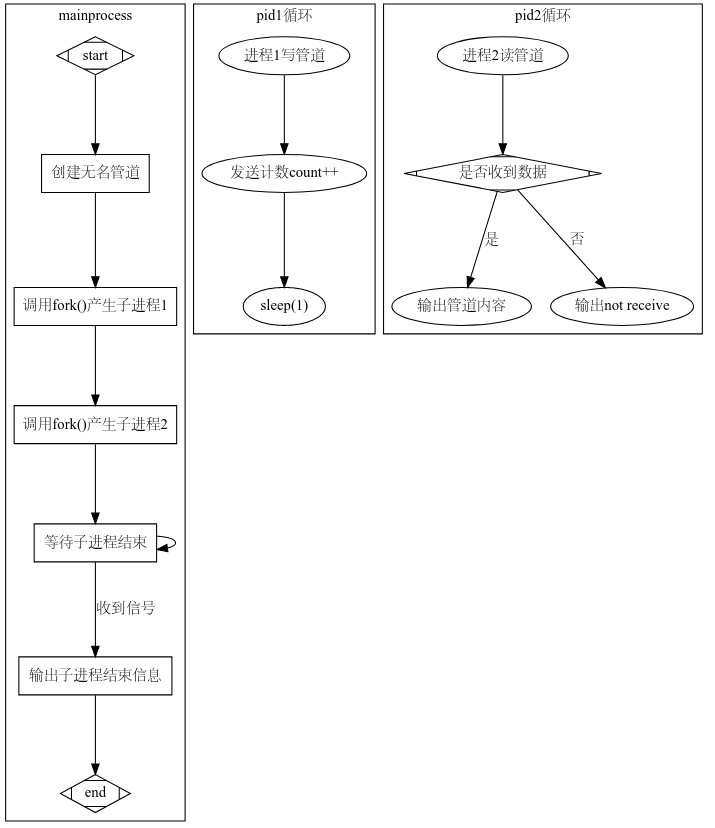


图1-3 函数流程图

## 1.5 心得体会与总结

1. 对于多线程的代码调试尝试了GDB,Clion自带调试，使用了不同的调试跟踪策略的尝试
2. 尝试使用dot语言进行函数流程图的绘制，增强了Linux下的文档能力
3. 了解了不同的中断信号及其系统的处理措施
4. 在绑定信号处理函数的时候发现了无法添加参数，必须使用全局变量，即绑定的信号处理函数无参数传入，即使定义了参数
5. 在进行gdb调试的时候无法使用control+c 发送信号，必须使用signal sigint

# 实验2 线程同步与通信

## 2.1 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。

## 2.2 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:

1. 两个线程,共享公共变量a
2. 线程1负责计算(1到100的累加，每次加一个数)
3. 线程2负责打印（输出累加的中间结果)

## 2.3 实验过程与结果

1. 创建信号量数组并判断创建结果
2. 创建线程
3. 主进程进行count判断线程数是否退出
   1. 线程1进行total累加操作
   2. 线程2进行total输出
   3. 利用PV操作空值进程运行顺序



图2-1 编译C文件



图2-2 执行C文件

## 2.4 实验结果分析

1. 整个程序的运行几乎是瞬时的，可以看出在线程间进行调度在现代操作系统上较快
2. 父进程进行监视的过程中可以采用sleep进行间歇监视，会造成最后完成之后有一定的时间延迟

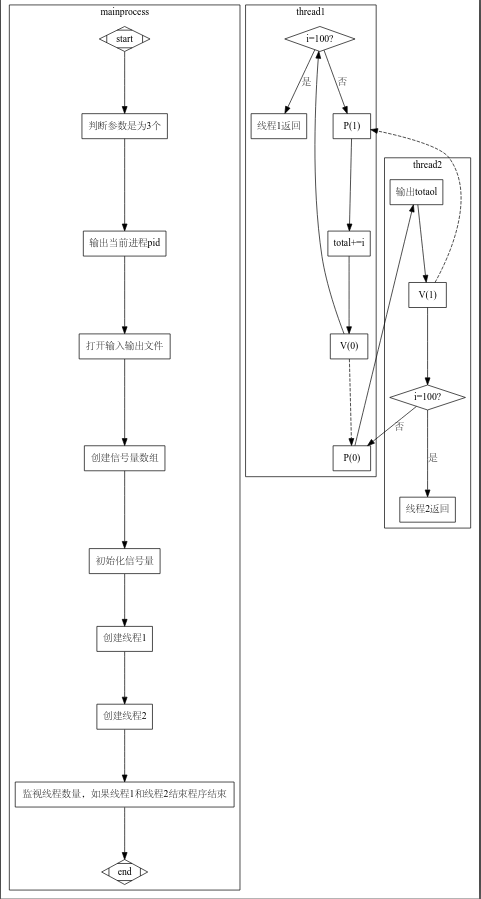


图2-3 函数流程图

## 2.5 心得体会与总结

1. 关于线程库的链接问题：
   1. 对于gcc编译，使用-pthread指令
   2. 对于CMake，使用target\_link\_libraries(oslab2 ${CMAKE\_THREAD\_LIBS\_INIT})
2. 关于P V操作的自行封装：
   1. 于semaphore.h中存在信号量相关函数，也可利用POSIX标准函数自行封装，二者同事include存在冲突会报错
   2. 封装的时候考虑每次需要传入semid十分麻烦，但是如果设置全局比变量又不是很优雅，考虑能否对于每一个线程设定局部变量
3. 对于此类程序，采用主进程监视子线程1和子线程2，在计算时间较长时可以考虑使用sleep函数减少主进程调度次数，但是在计算量较少时，可能造成最后结束时的延迟

# 实验3 共享内存与进程同步

## 3.1 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。

## 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 实验过程与结果

1. 检查参数数量，如果参数数量不为3，输出错误退出
2. 打开输入输出文件并判断
3. 创建共享内存组
4. 设置初始信号量值
5. 设置缓冲区标志位
6. 创建线程1
7. 线程1执行readbuf\_process()
   1. 获取缓冲区地址
   2. 从文件读取一个字节
   3. 缓冲区空闲区信号量P操作
   4. 写缓冲区
   5. 判断文件是否写完，如果写完，将缓冲区完成标志位进行完成标记，完成f操作后退出
   6. 缓冲区满信号量V操作
   7. 继续b操作
8. 线程2执行writebuf\_process()
   1. 获取缓冲区地址
   2. 缓冲区满信号量P操作
   3. 从缓冲区读取一个字节
   4. 判断缓冲区完成标志位，是否完成，如果完成，判断当前缓冲区下标是否等于readbuf\_process下标，如果等于，完成e操作后退出
   5. 缓冲区空闲去信号量V操作
   6. 继续b操作



图3-1 编译C文件

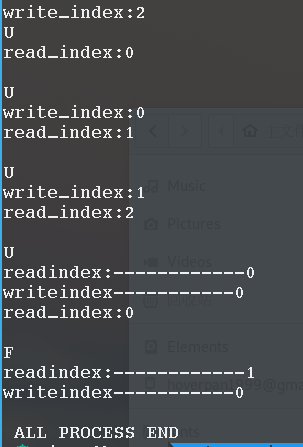


图3-2 执行C文件

## 3.4 实验结果分析

图3-3 函数流程图

## 3.5 心得体会与总结

1. 书写过程中, semget和shmget混淆，造成其中一个进程能够运行但是无法正常输入输出，一定时间的debug
2. 对于拷贝，初始考虑为采用文件末尾的标识符进行终止，但是对于非文本文件的拷贝存在问题，且若文本中若存在此类符号，造成提前终止，类似于SQL中的语句嵌套提前终止，采用读写通信进程的方式进行拷贝
3. 通过调节缓冲区大小MAXLENGTH进行函数的调节3和50，发现整体拷贝速度无明显改变，说明限制拷贝速度的不是缓冲区大小，而是频繁的I/O操作

# 实验4 TinyOS实验

## 4.1 实验目的与要求

### 4.1.1 基础实验(Task1-3)

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

### 4.1.2 附加实验(Task4)

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。

## 4.2 实验内容

### 4.2.1 BlinkBinary(Task1)

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。

### 4.2.2 BlinkCompute(Task2)

修改Blink程序，在timer0的触发事件处理中加入计算

### 4.2.3 BlinkComputeSlicing(Task3)

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。

### 4.2.4 TinyOSSensor(Task4)

## 4.3 实验过程与结果

### 4.2.1 BlinkBinary(Task1)

### 4.2.2 BlinkCompute(Task2)

### 4.2.3 BlinkComputeSlicing(Task3)

### 4.2.4 TinyOSSensor(Task4)

图1-1 编译C文件

图1-2 执行C文件

## 4.4 实验结果分析

图1-3 函数流程图

## 4.5 心得体会与总结

1. TinyOS 中nesC的运作方式类似于基于C语言实现的简易面向对象的程序设计，为了实现模块化和模块的可复用性，对模块进行运行时接口连接
2. 理解nesC中不同模块间的通信机制，基于下行调用，和上行反馈

# 实验5 附加实验ls命令的实现

## 5.1 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。

## 5.2 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；
3. 显示文件的类型、大小、时间等信息；
4. 递归显示子目录中的所有文件信息。

## 5.3 实验过程与结果

1. 使用getopt()和getopt\_long()解析命令行参数，完成基础功能
2. 模式解析：
   1. 树形打印：
      1. 调用opendir()打开当前目标目录
      2. 调用chdir()转到目标目录
      3. 忽略 . 和.. 目录项，对目录下文件进行递归打印
      4. 通过depth递归深度判断前面空格长度
   2. ls输出
      1. 初始化缓冲区进行当前目录项的内容缓存，便于进行目录项输出
      2. 对于缓冲区的目录文件第一项和最后一项进行计算出当前目录不递归的大小并输出在首行
      3. 对于目录项下的内容进行遍历，如果为目录项则递归进入
      4. 为了打印出完整的目录内容，需要使用getcwd()函数获取完整的文件路径，此处自己实现了mycwd()函数利用..文件指针进行向上倒序获取完整路径

图1-1 编译C文件

图1-2 执行C文件

## 5.4 实验结果分析

图1-3 函数流程图

## 5.5 心得体会与总结

1. 编写过程中需要熟悉文件结构，其中Linux的文件结构对于每一个逻辑文件项或者目录项都是以指针形式访问，同时需要关注当前所在的目录项
2. 在编写自己的cwd()函数时，因为向上逆序寻找到根目录获取整个文件路径，在此过程中改变了当前的目录项，需要注意之后chdir返回之前的目录，否则会造成访问失败造成程序崩溃
3. 对于linux基础shell指令的实现能够更好的理解linux操作系统，其中cp,cwd,ls,mkdir等函数编写能够更好的理解文件系统

# 附录

## Lab1 code

## Lab2 code

## Lab3 code

## TinyOSLab Task1 Code

## TinyOSLab Task2 Code

## TinyOSLab Task3 Code

## TinyOSLab Task4 Code