**中国科学技术大学计算机学院**

**《计算机网络》实验报告**



实验题目：lab2\_WireShark Labs(Getting Started & HTTP)

学生姓名：胡毅翔

学生学号：PB18000290

专业：计算机科学与技术

指导老师：张信明

完成日期：2020年11月1日

计算机实验教学中心制

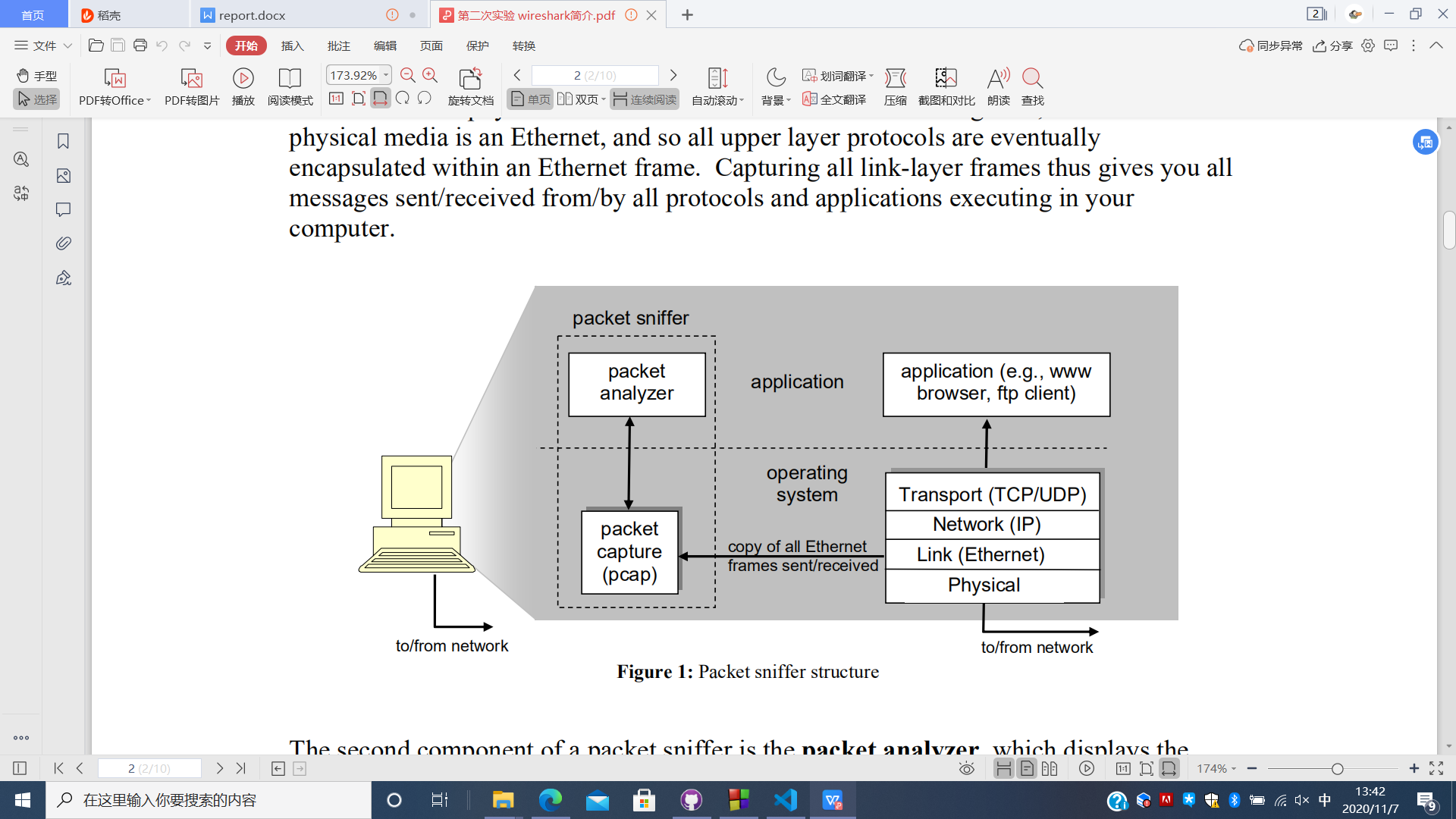
2019年09月

**实验目的**

1. 实现两种调度算法熟悉并掌握Wireshark网络分析工具。
2. 捕获观察并分析HTTP报文结构。
3. 回答第二次实验PDF文件中的问题。
4. 分析HTTP中GET和POST请求方式的区别。

**实验原理**

本次实验使用WireShark工具。其中，用来观察执行协议实体之间交换的报文的基本工具称为分组嗅探器(packet sniffer)。分组嗅探器被动地拷贝(嗅探)由计算机发送和接收的报文；它也能显示出这些被捕获报文的各个协议字段的内容。分组嗅探器从不发送报文，同时接收到的报文也不会显式地发送到分组嗅探器。它接受的是发送/接收的报文的复制。



分组嗅探器的结构如图1所示。在图1的右边是运行在计算机上的协议和应用。分组嗅探器(图中画虚线框部分)是计算机中的附加软件(区别于上述协议和应用)，它包含两个部分。分组捕获库获取每一个链路层接收/发送的帧。第二部分是分组分析器，其中显示了协议所有字段的内容。为了实现这一目的，分组分析器必须理解所有协议所交换的信息的结构。比如，我们对图1 中HTTP协议的各个字段信息感兴趣。分组分析器理解以太网帧的格式，所以可以从以太网帧中区分出IP数据报。同时，它还理解IP数据报格式，所以它能从IP数据报分离出TCP报文段。最后，它还理解TCP报文段格式，从中分离出HTTP报文。又因它理解HTTP协议，所以能在实现WireShark理解封包的功能。

**实验环境**

1. PC一台
2. Windows系统
3. WireShark网络分析工具(版本 3.2.7)
4. Edge浏览器(版本 86.0.622.56)

**实验过程**

**Getting Started**

1. WireShark的安装

实验步骤：

1.前往http://www.wireshark.org/download.html 下载并安装WireShark。

2.下载WireShark用户指南。

尝试运行WireShark

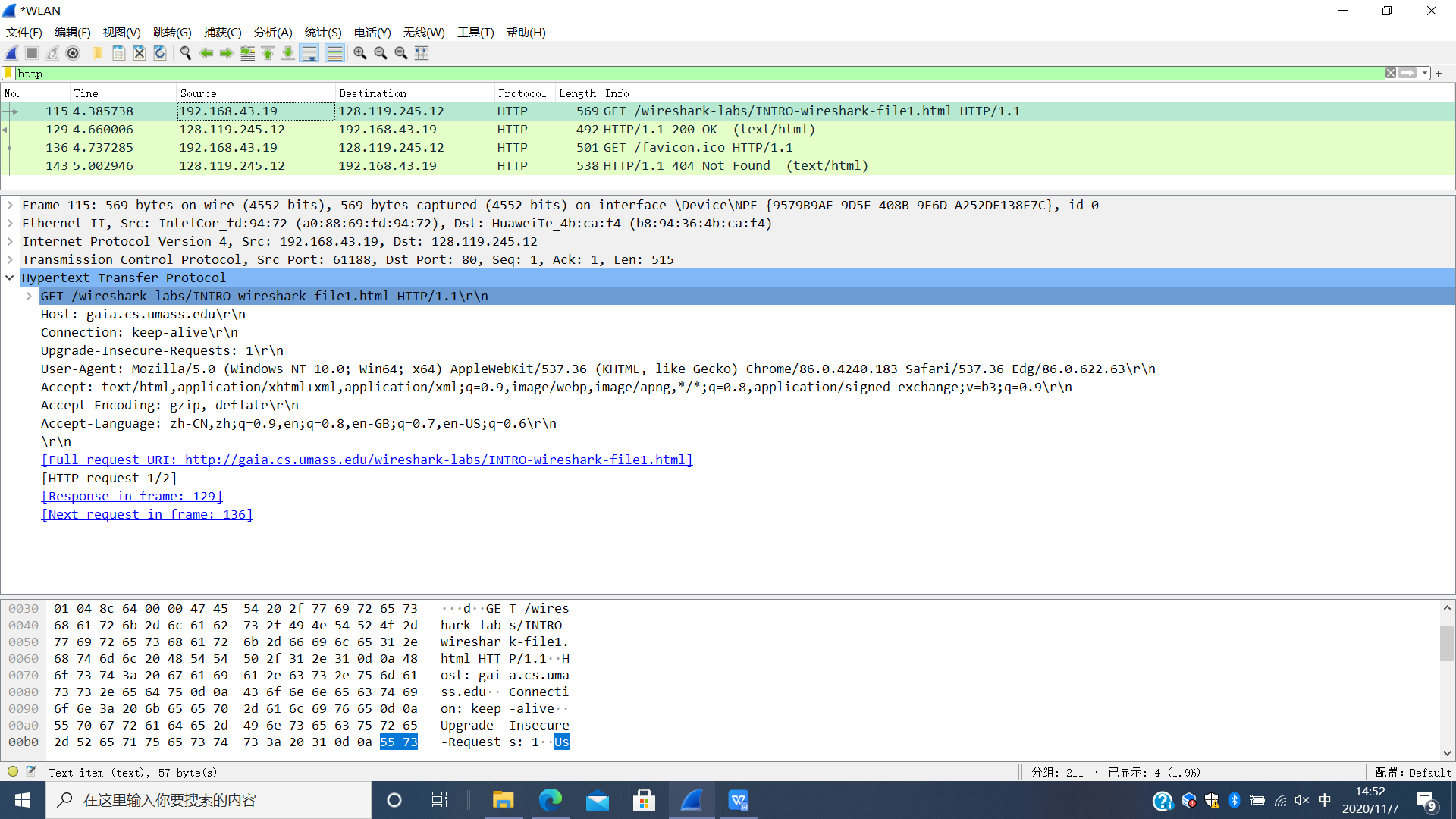
1. 启动WireShark，结合实验指导书，了解WireShark各个界面(命令菜单，显示过滤器，封包列表，封包详细信息以及16进制数据)的功能。
2. 根据实验指导书，对WireShark进行一系列设置。

3.开始运行WireShark，打开链接http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html

4.加载完后，在捕获窗口中停止WireShark捕获分组，利用“http”规则筛选出HTTP条目

5.分析HTTP报文结构

请求报文

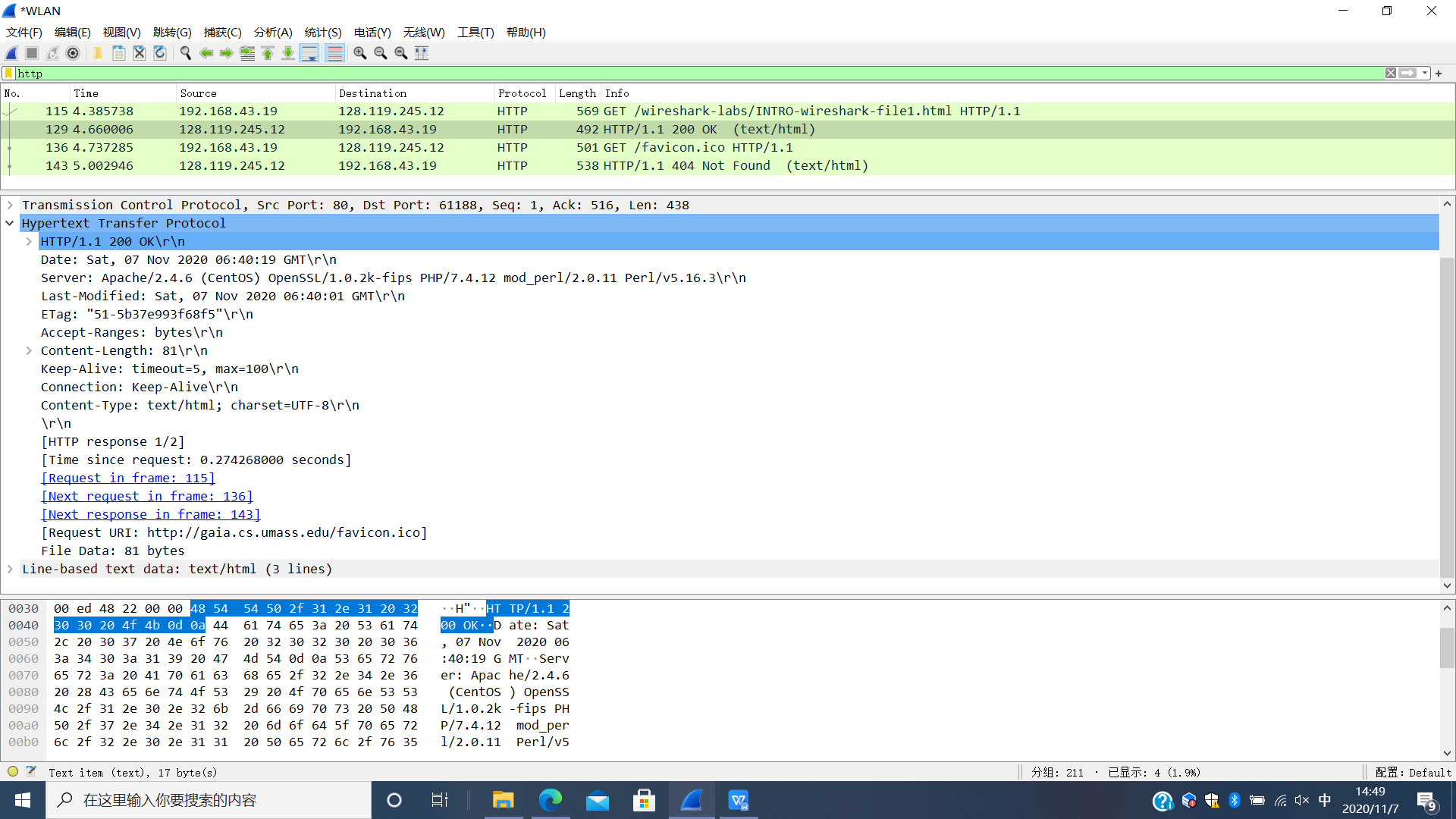


第一部分为请求行，说明请求类型(方法字段)为GET，要访问的资源(URL字段)为 /wireshark-labs/INTRO-wireshark-file1.html，所用的HTTP版本(HTTP版本字段)为1.1版本(即采取持续连接，服务器在发送相应后保持该TCP连接打开，相同客户和服务器之间后续的请求和相应能通过相同的连接进行传送)。

第二部分为请求头部，或称首部行。Host指明了对象所在的主机(gaia.cs.umass.edu)。Connection: keep-alive表示使用持续连接。Upgrade-Insecure-Requests: 1告诉服务器浏览器可以处理https协议。User-Agent用来指明用户代理，即向服务器发送请求的浏览器类型。这里浏览器类型是Edg/86.0.622.63。Accpet表明浏览器希望接收的数据类型。Accpet-Encoding表明浏览器希望接收的数据编码格式。Accpet-Language表明浏览器希望接收的对象的语言版本。这里zh-CN表示希望接收中文版本。

第三部分为实体体。使用GET方法时实体体为空。

响应报文



第一部分为状态行，有3个字段：协议版本字段、状态码和相应状态信息。这里指示服务器正在使用HTTP/1.1，并且一切正常。

第二部分为首部行。Date: 指示服务器产生并发送该响应报文的日期和时间。Server: 指示该报文是由Apache Web服务器产生的。Last-Modified: 指示了对象创建或最后修改的日期和时间。Etag: 指示了对象的标记，可以视为版本标记，主要是为了解决一些Last-Modified无法解决的问题(1、一些文件也许会周期性的更改，但是他的内容并不改变(仅仅改变的修改时间)，这个时候我们并不希望客户端认为这个文件被修改了，而重新GET;

2、某些文件修改非常频繁，比如在秒以下的时间内进行修改，(比方说1s内修改了N次)，If-Modified-Since能检查到的粒度是s级的，这种修改无法判断(或者说UNIX记录MTIME只能精确到秒)

3、某些服务器不能精确的得到文件的最后修改时间；)。响应头Accept-Ranges标识自身支持范围请求(partial requests)。字段的具体值用于定义范围请求的单位。Content-Length: 指示了被发送对象中的字节数。

Keep-Alive: 中timeout指示了一个空闲连接需要保持打开状态的最小时长（以秒为单位），而max指示了在连接关闭之前，在此连接可以发送的请求的最大值。Connection: Keep-Alive指示了 持续连接。 Content-Type: 指示了实体体的对象是html，编码格式为utf-8。

6.退出WireShark

7.回答实验指导书最后的问题：

1.1. List up to 10 different protocols that appear in the protocol column in the

unfiltered packet-listing window in step 7 above.

TCP、DNS TLSv1.2 ICMP NBNS HTTP UDP ARP MDNS BROWSER TLSv1.3

2. How long did it take from when the HTTP GET message was sent until the HTTP

OK reply was received? (By default, the value of the Time column in the packet

listing window is the amount of time, in seconds, since Wireshark tracing began.

To display the Time field in time-of-day format, select the Wireshark *View* pull

down menu, then select Time *Display Format*, then select *Time-of-day*.)

4.660006-4.385738

3. What is the Internet address of the gaia.cs.umass.edu (also known as www

net.cs.umass.edu)? What is the Internet address of your computer?

gaia.cs.umass.edu:128.119.245.12

Mycomputer:192.128.43.19

4. Print the two HTTP messages displayed in step 9 above. To do so, select *Print*

from the Wireshark *File* command menu, and select “*Selected Packet Only”* and

*“Print as displayed”* and then click OK.

HTTP请求和接收报文截图已包含在实验过程中。

WireShark Lab: HTTP

The Basic HTTP GET/response interaction

实验步骤：

1. 启动浏览器
2. 启动WireShark分组嗅探器，在显式过滤器中输入http。
3. 等一分多钟，然后开始捕获分组。

在浏览器中输入http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file1.html

1. 停止WireShark捕获分组。

回答

1. Is your browser running HTTP version 1.0 or 1.1? What version of HTTP is the

server running?

HTTP/1.1

GET /wireshark-labs/HTTP-wireshark-file1.html HTTP/1.1\r\n

HTTP/1.1

HTTP/1.1 200 OK\r\n

2. What languages (if any) does your browser indicate that it can accept to the

server?

Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6\r\n

Chinese/English

简体中文 中文 英语 英式英语 美式英语

zh-CN,zh;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6

1. What is the IP address of your computer? Of the gaia.cs.umass.edu server?

gaia.cs.umass.edu:128.119.245.12

Mycomputer:192.128.43.19

1. What is the status code returned from the server to your browser?

200

1. When was the HTML file that you are retrieving last modified at the server?

Last-Modified: Sat, 07 Nov 2020 06:59:01 GMT\r\n

1. How many bytes of content are being returned to your browser?

128

7. By inspecting the raw data in the packet content window, do you see any headers

within the data that are not displayed in the packet-listing window? If so, name

one.

2.打开Edge浏览器

本实验的软件框图如图所示。软件层次分为multiboot\_header、myOS和userApp三部分。

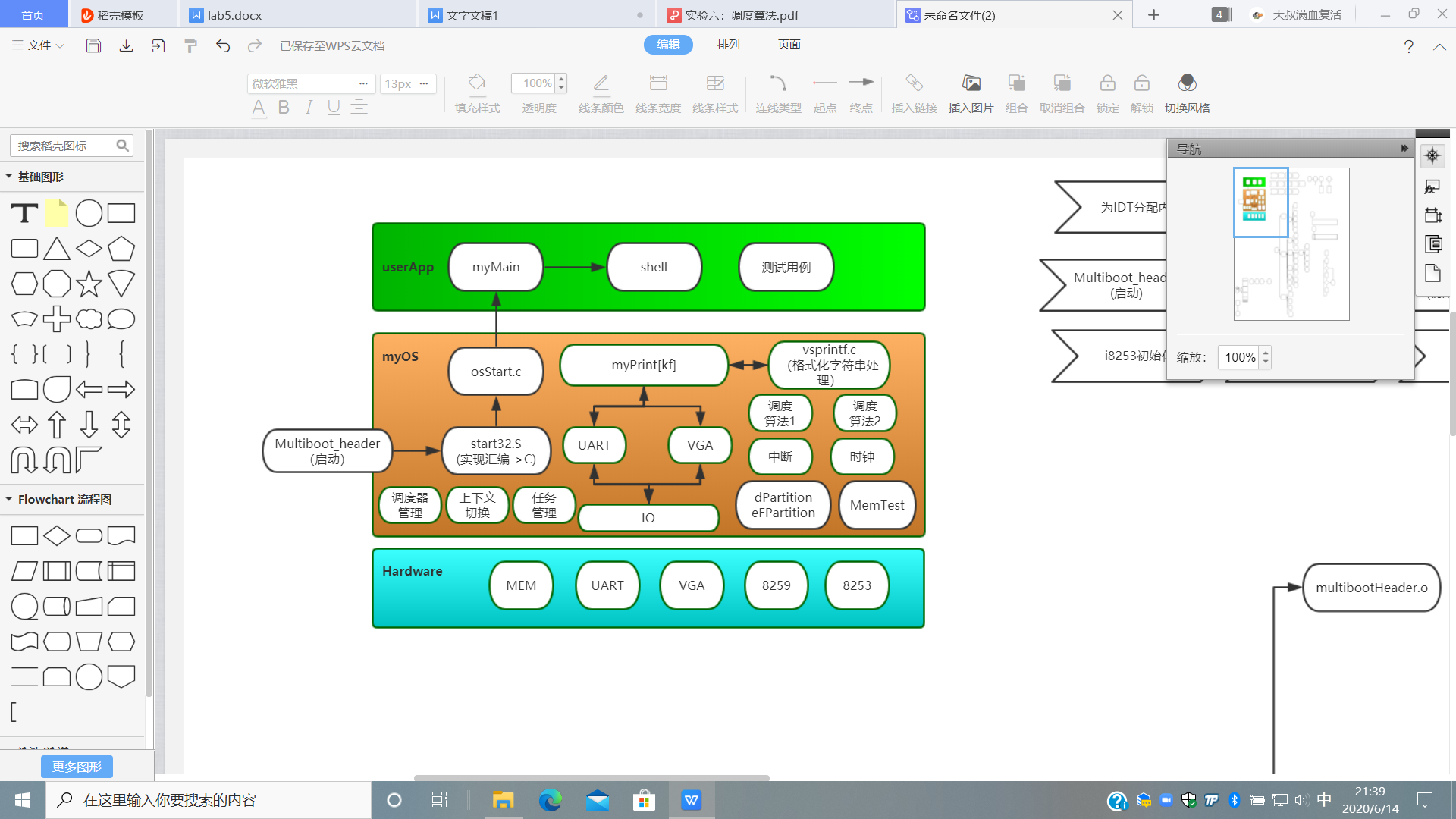


图 1

**主流程**

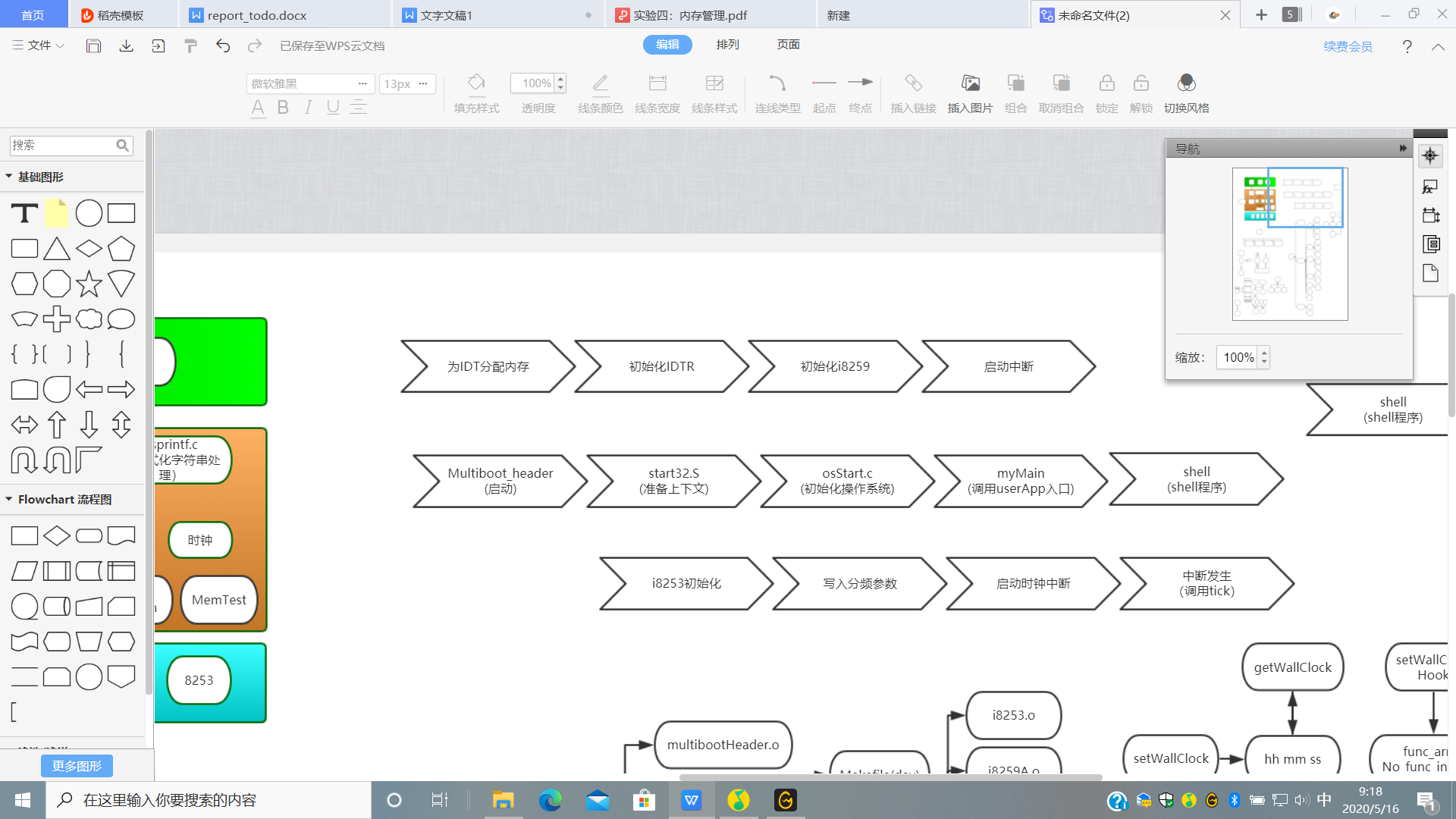


图 2

本实验的主流程如上图所示：

1. 在multiboot\_header中完成系统的启动。
2. 在start32.S中准备好上下文，最后调用osStart.c把进入c程序。
3. 在osStart.c中完成初始化8259A，初始化8253，清屏，内存初始化，任务管理初始化等操作，最后切换至多任务状态。
4. 运行myMain中的代码，完成创建测试任务，shell初始化，内存测试初始化等操作，启动shell。
5. 进入shell程序，等待命令的输入。

**主要功能模块及其实现**

**任务数据结构(TCB)及任务池(tcbPool)**

该功能模块用于对任务的信息的管理及控制，主要包含栈顶指针，栈底指针，任务序号，任务状态。任务池则用TCB数组构成，用next指针链接成链表。

TCB结构如下：

typedef struct myTCB {

    /\* node should be the 1st element\*/

    struct dLink\_node thisNode;

    /\* node body \*/

    unsigned long state;  // 0:rdy

    int tcbIndex;

    struct myTCB \* next;

    unsigned long\* stkTop;

    unsigned long stack[STACK\_SIZE];

    tskPara para;

    unsigned int leftSlice; // for SCHED\_RR or SCHED\_RT\_RR policy

} myTCB;

任务池结构如下：

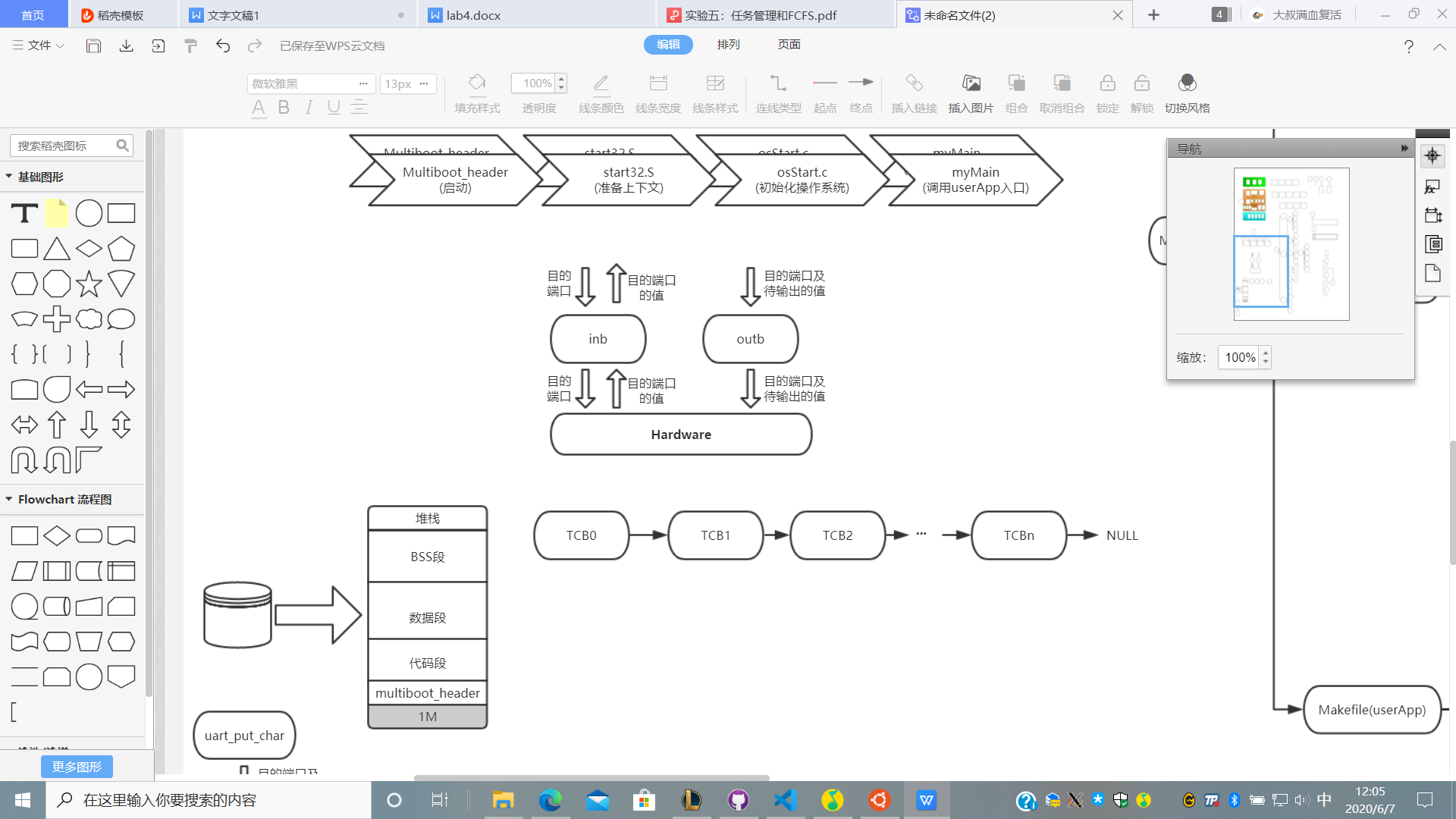


图 3

**任务创建/销毁**

该功能模块用于创建，销毁，主要目的是实现createTsk(),destroyTsk()函数。

createTsk()实现TCB分配，初始化调度参数，初始化栈，对下一空闲TCB进行修改，若此时为到达时间，调用tskStart()，启动任务，否则调用tskStartDelayed()函数，待到达时间到后再执行。

destroyTsk()实现TCB回收，将TCB链接到空闲TCB链表头，修改下一空闲TCB，同时调度新任务执行。

**任务参数**

该功能模块用于任务调度时的任务参数的初始化，设置及读取，主要目的是实现initTskP

ara()，setTskPara()，getTskPara()函数。

具体实现较为简单，详见taskPara.c文件中的代码。

**统一的调度接口**

该模块的主要功能是调度器的接口统一，实现方式为调用对应调度器的结构体的函数，具体实现详见代码。

**调度算法PRIORITY0、PRIORITY的实现**

这两种调度算法的实现与FCFS基本一致，只需在任务入队时，根据任务的优先及进行排序即可。因入队前，任务链表中的优先级是有序的，所以只需顺序查询，找到插入的节点,调用dLinkList.c中的函数完成插入即可。

具体实现代码如下：

void tskEnqueuePRIO0(myTCB \*tsk)

{

    myTCB \*point;

    point = rqPRIO0;

    if (point == NULL)

    {

        dLinkInsertBefore((dLinkedList \*)point, (dLink\_node \*)point, (dLink\_node \*)tsk);

    }

    else

    {

        while (tsk->para.priority > point->para.priority && point->next != 0)

            point = point->next;

        if (tsk->para.priority > point->para.priority)

            dLinkInsertAfter((dLinkedList \*)point, (dLink\_node \*)point, (dLink\_node \*)tsk);

        else

            dLinkInsertBefore((dLinkedList \*)point, (dLink\_node \*)point, (dLink\_node \*)tsk);

    }

}

**调度器管理的实现**

该部分主要实现调度器的设置及相关参数的设定，主要目的是实现getSysScheduler()，setSysScheduler()，getSysSchedulerPara()及setSysSchedulerPara()。

具体实现较为简单，详见代码task\_sched.c部分。

**源代码说明**

**Makefile组织**

Makefile组织结构如下：

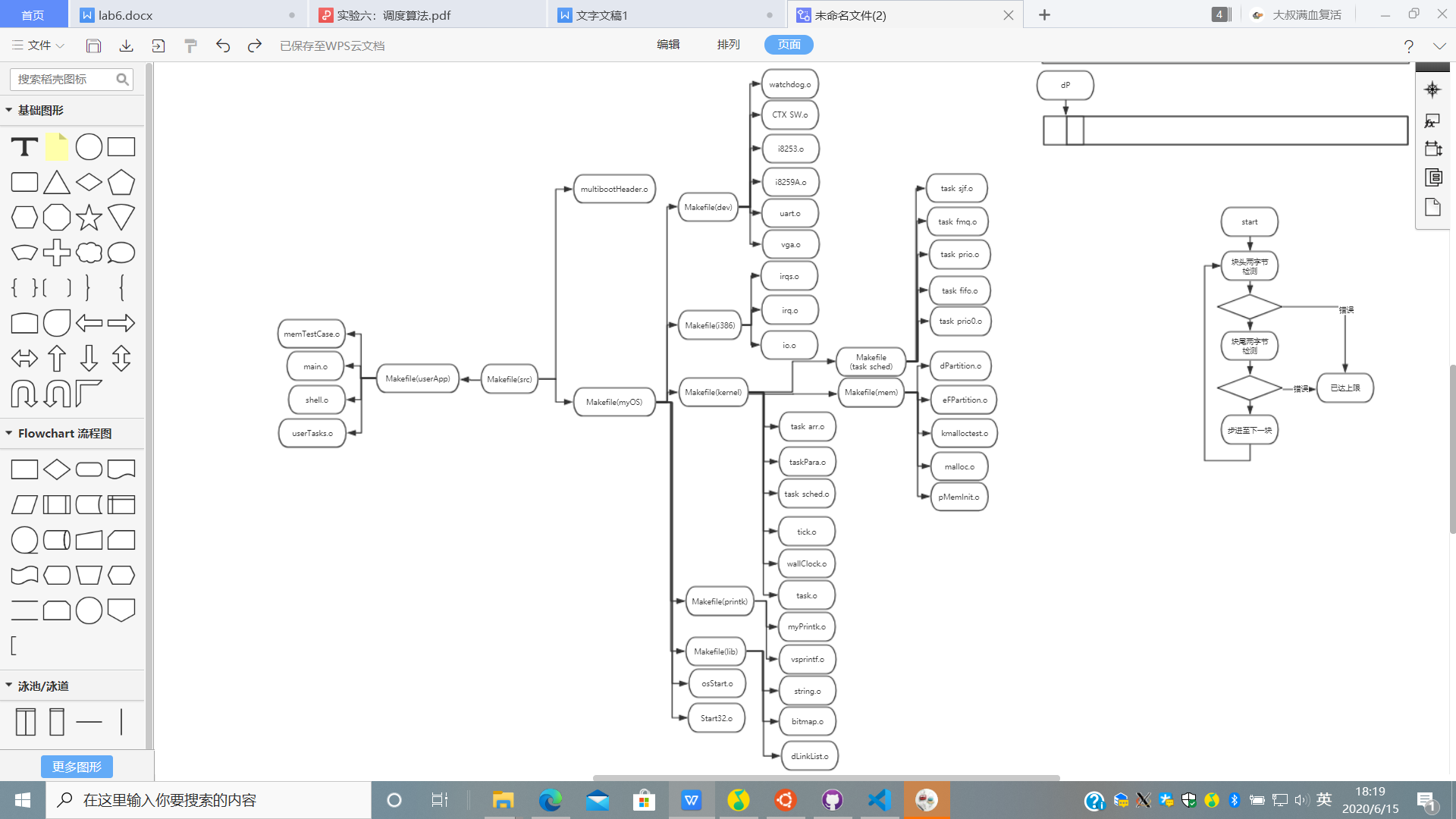


图 4

**目录组织**

目录组织如下图所示，主目录下

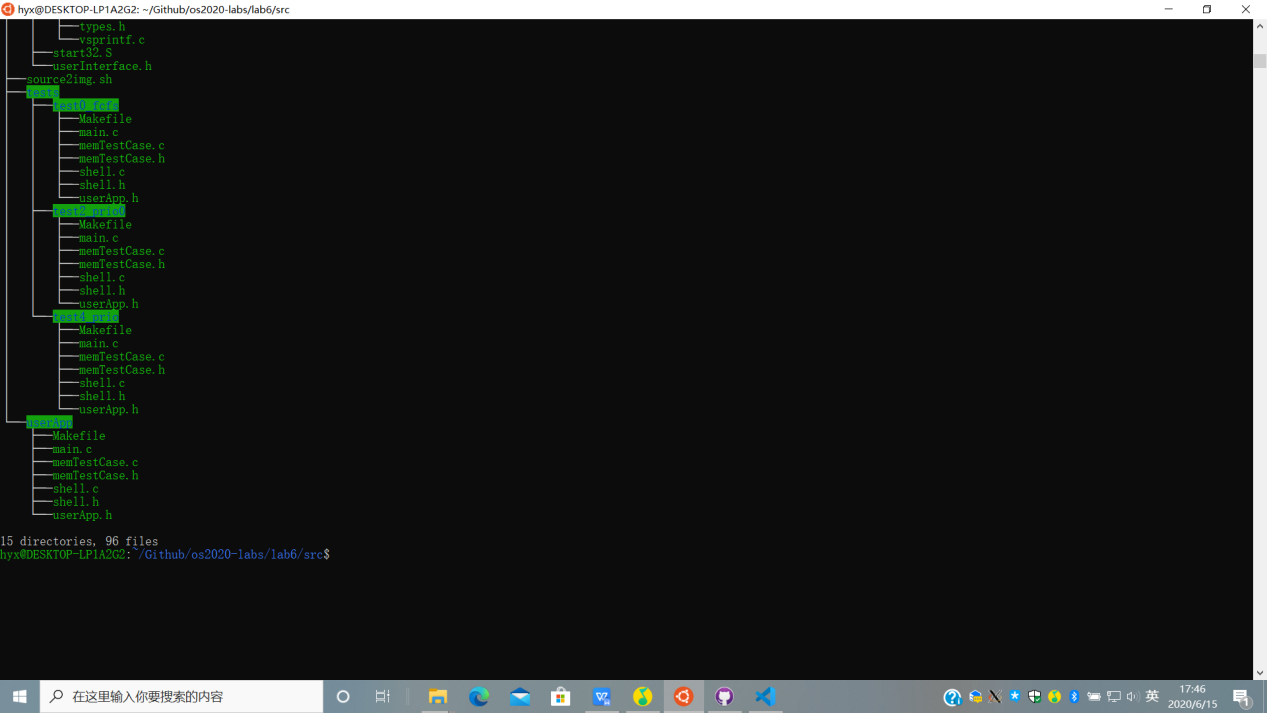
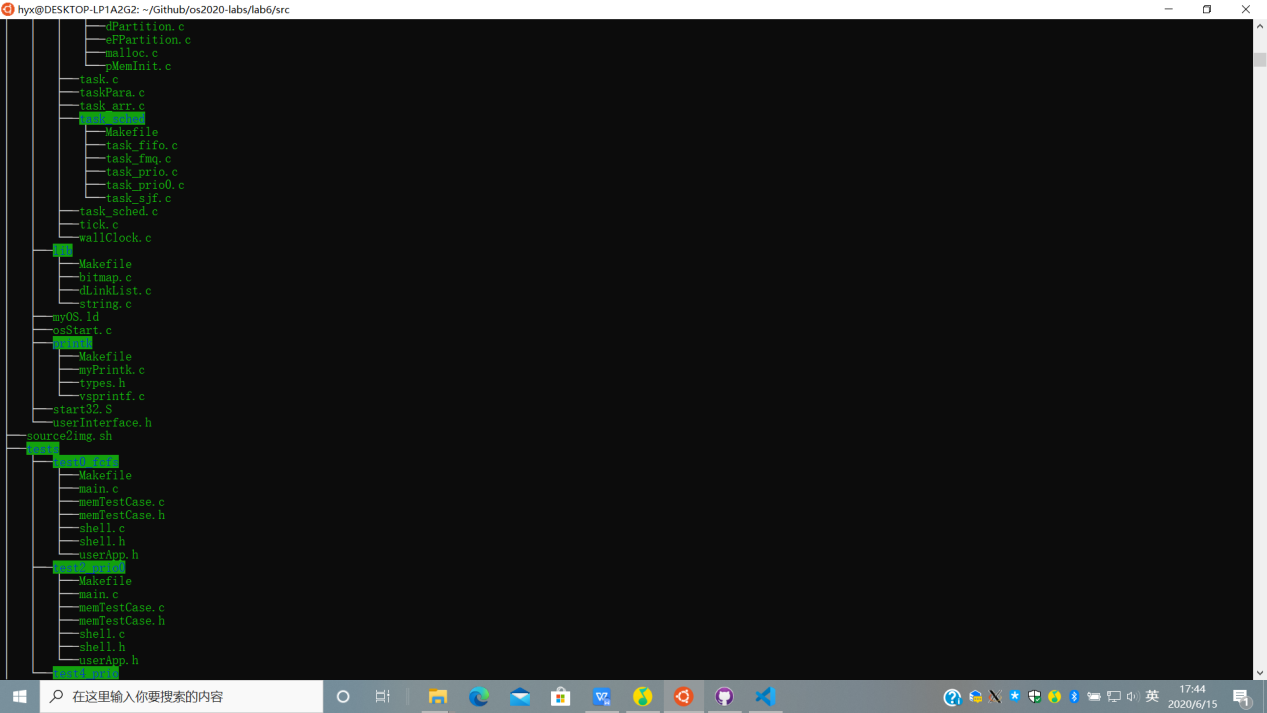
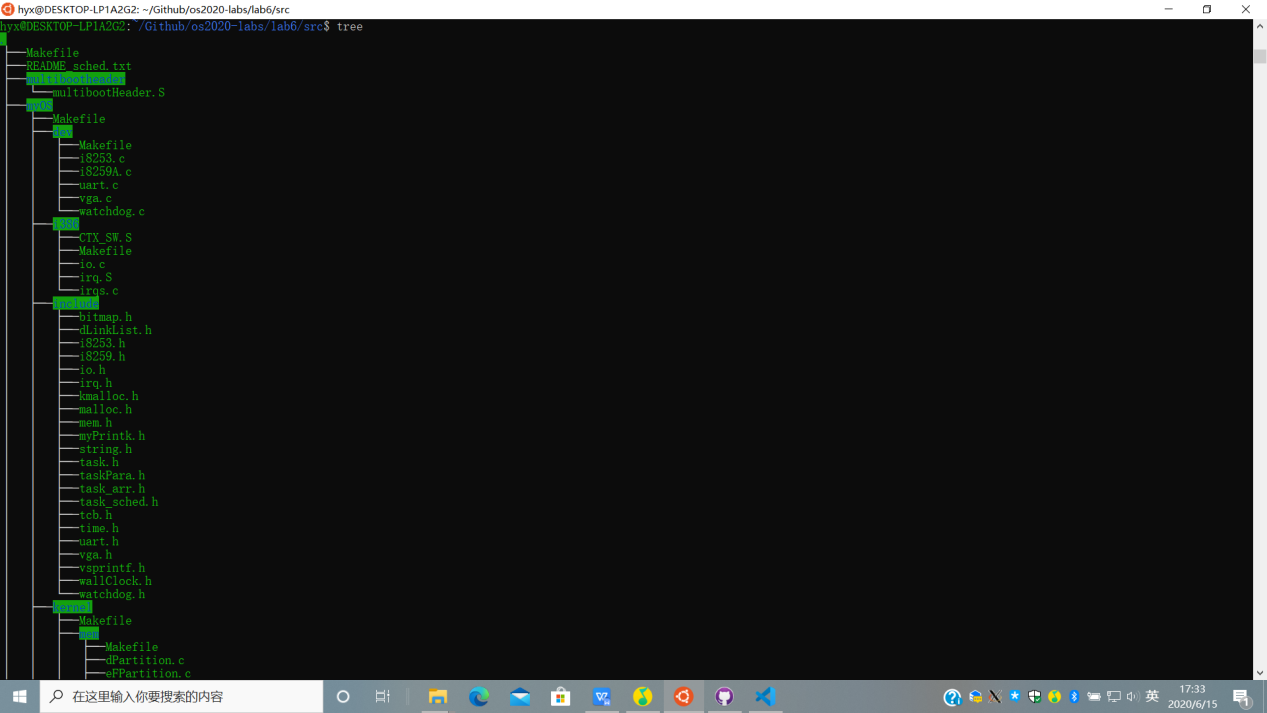


图 4

**代码布局说明（地址空间）**

从物理内存1M的位置开始放代码和数据，前面12个字节为multiboot\_header,向后对齐8个字节，放代码。再向后对齐16个字节，用于放初始化的数据（数据段）。在数据段之后，再向后对齐16个字节。之后为BSS（Block Started by Symbol）段,用于存放程序中未初始化的全局变量和静态变量。并在BSS段后，再向后对齐16个字节。剩余部分为堆栈段。

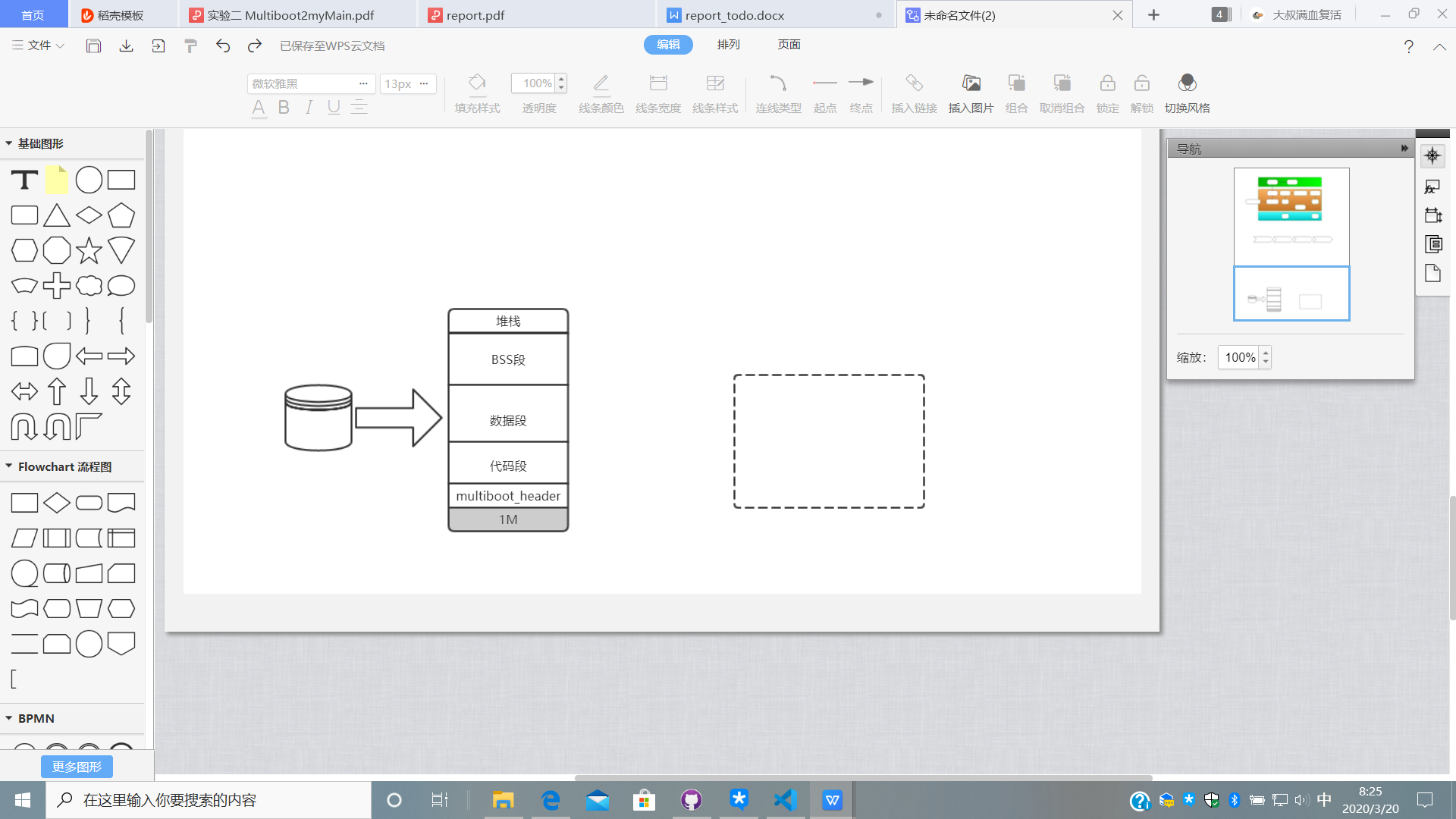


图 5

**编译过程说明**

ASM\_FLAGS= -m32 --pipe -Wall -fasm -g -O1 -fno-stack-protector

-m32用32位机器的编译器来编译这个文件

--pipe使用管道代替编译中临时文档

-Wall 打开警告选项

-fasm 识别asm关键字

-g 使用调试器GDB

-O1 优化生成代码

-fno-stack-protector停止使用stack-protector功能

C\_FLAGS =  -m32 -fno-stack-protector -fno-pic -fno-builtin -g

-fpic 如果支持这种目标机,编译器就生成位置无关目标码.适用于共享库(shared library)

生成multiHeader.o、osStart.o、start32.o、uart.o、vga.o、io.o、myPrintk.o、vsprintf.o及main.o多个目标文件，链接生成myOS.elf文件。

**运行和运行结果说明**

PRIORITY0的测试用例示意如下：

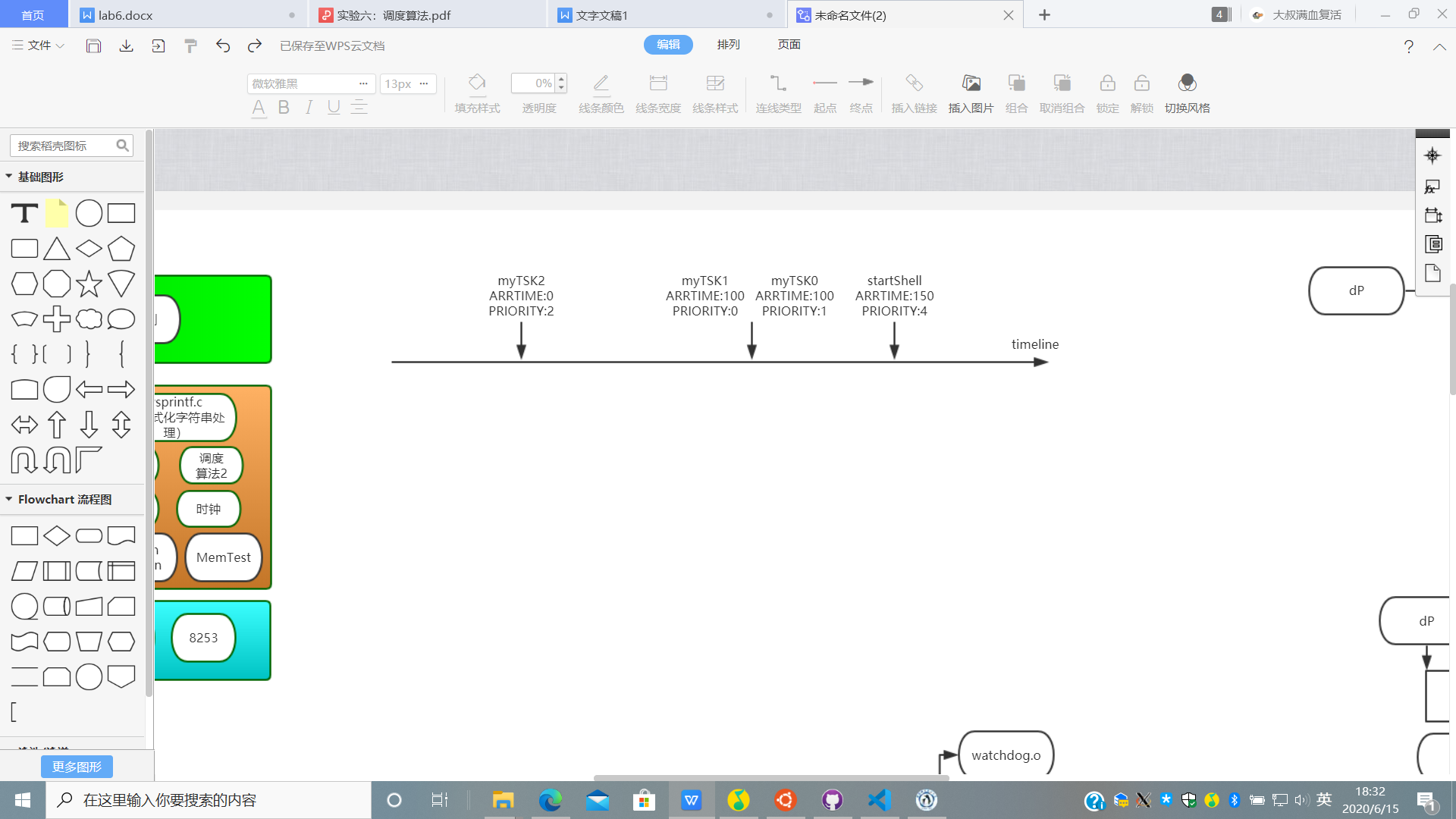


图 6

输入./source2run.sh test2\_prio0指令后，编译，链接，生成myOS.elf文件并运行之，输入sudo screen /dev/pts/0，通过Ubuntu输入。运行结果如下图：

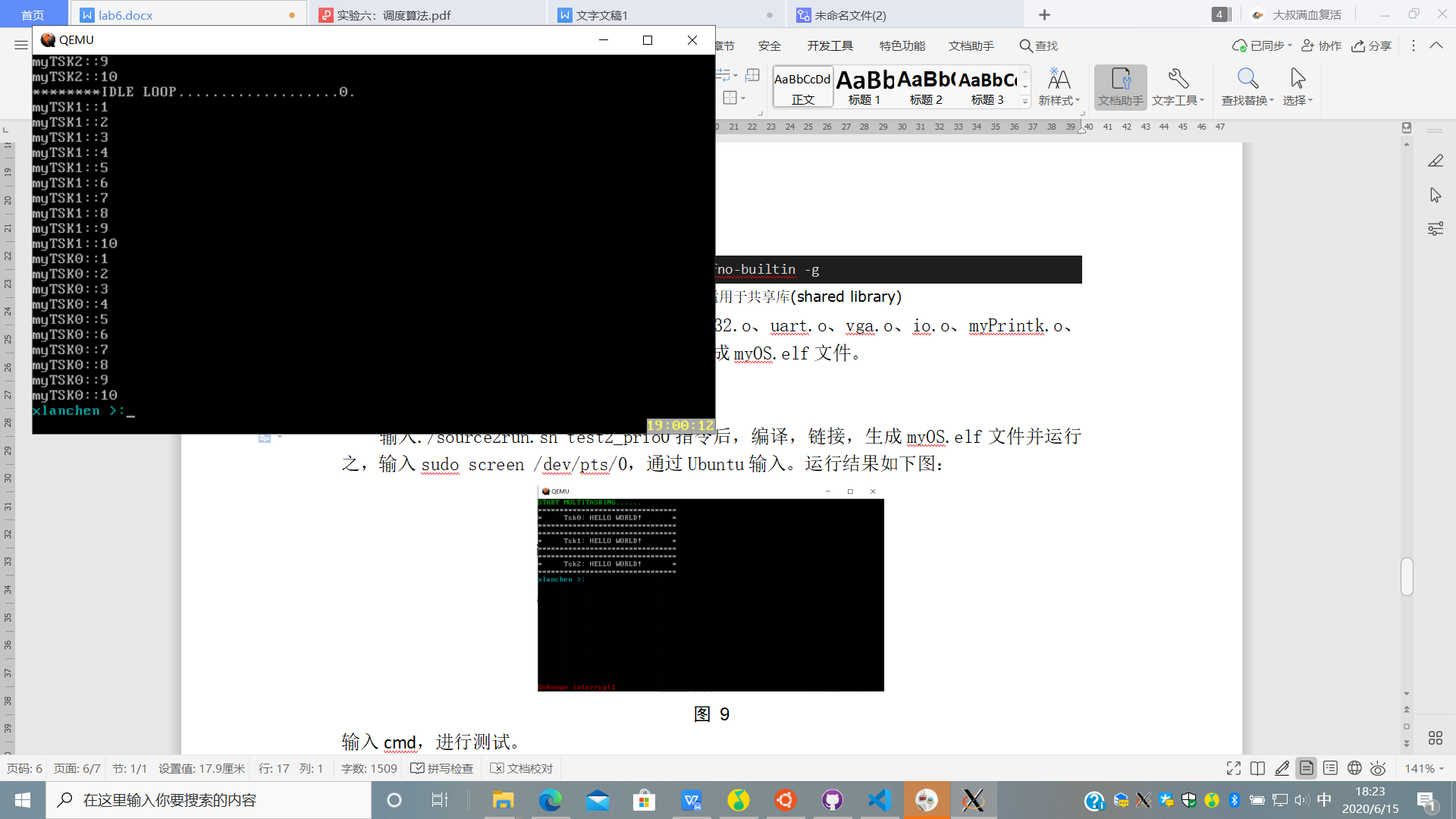


图 7

PRIORITY的测试用例示意如下：

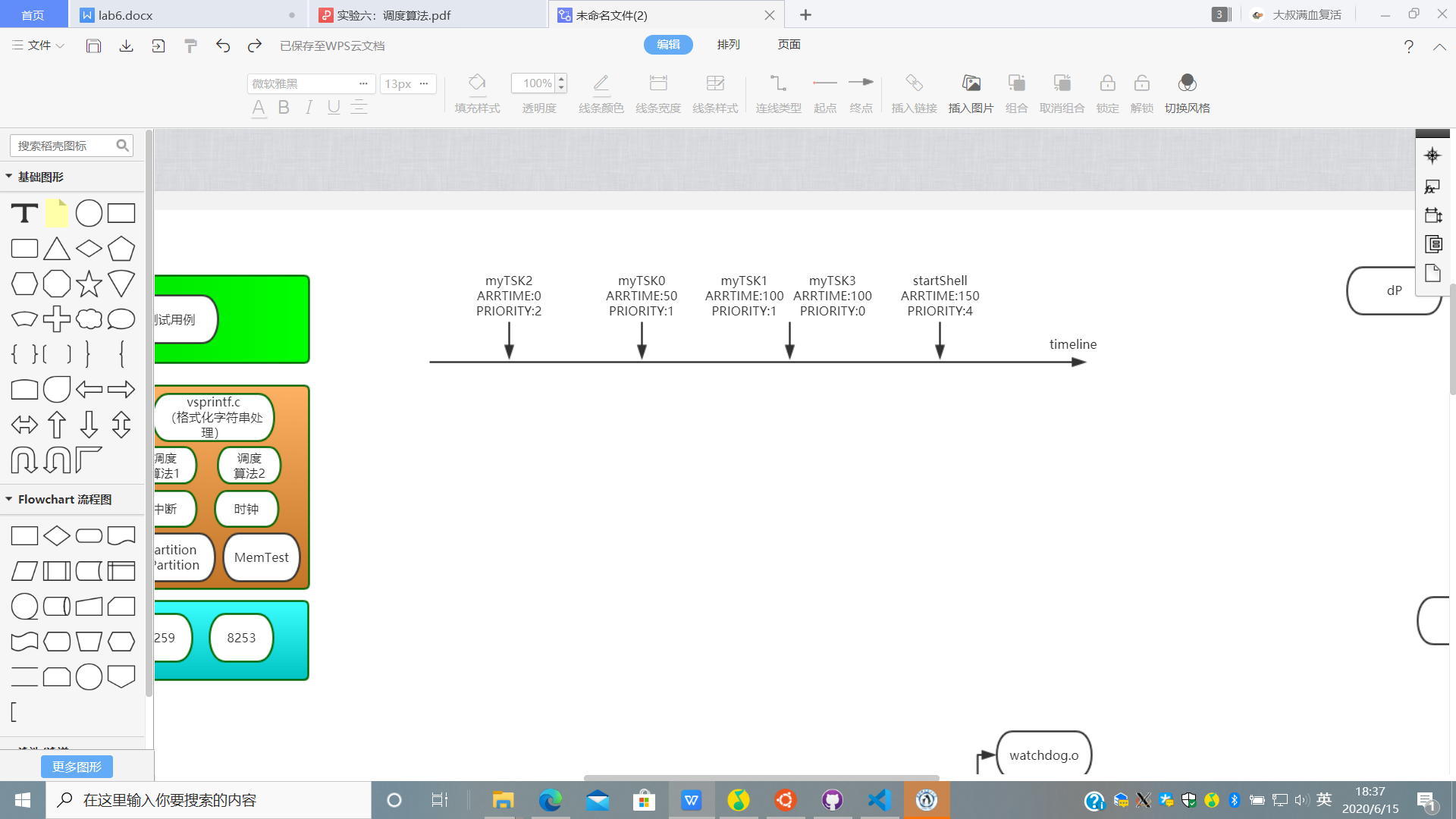


图 8

输入./source2run.sh test4\_prio指令后，编译，链接，生成myOS.elf文件并运行之，输入sudo screen /dev/pts/0，通过Ubuntu输入。运行结果如下图：



图 9

**遇到的问题和解决方案说明**

1. 编译时出现报错“对‘\_GLOBAL\_OFFSET\_TABLE\_’未定义的引用”。

解决方案：在src目录下的Makefile文件的CFLAGS变量中添加-fno-pic.

1. 编译出现fatal error: bits/libc-header-start.h: No such file or directory

解决方案:在Ubuntu中输入apt-get install gcc-multilib，完善编译环境

1. 编译时出现warning: assignment makes pointer from integer without a cast

解决方案：在给指针赋值前进行强制格式转化(unsigned short int\*)

1. 编译时出现warning: conflicting types for built-in function

解决方案：在src目录下的Makefile文件的CFLAGS变量中添加-fno-builtin