《现代交换原理》实验报告

实验名称 ＿＿＿＿＿时间表调度实验＿＿＿＿＿

班　　级 ＿＿＿＿＿＿2017211305＿＿＿＿＿＿

姓　　名 ＿＿＿＿＿＿＿＿于海鑫＿＿＿＿＿＿＿＿

指导教师 ＿＿＿＿＿＿＿＿丁玉荣＿＿＿＿＿＿＿＿

一、 实验目的

驱动交换网络实验用来考查学生对时间表调度原理的掌握情况。

二、 实验原理及设计

在程控数字交换的体系结构中，周期级程序（例如摘挂机检测程序、脉冲识别程序、位间隔识别程序）是由时间表调度实现的。所谓时间表调度，是指每经过交换系统的最短有效时间（这通常是指各周期性程序周期的最大公约数），都会检查调度表的调度要求，如果某个程序在这时需要执行，则调度程序开始执行它。

在我们设计的时间表调度实验中，这个调度表的调度是静态的。所谓静态，是指我们的调度表是在系统初始化的时候就建立起来的，在系统运行的情况下不再改动。 实验要求的就是这个调度表的初始化。这个调度表如下:

时间（10ms） \ 任务	0：摘挂机检测任务	1：脉冲检测任务	2：位间隔检测任务
0	0/1	0/1	0/1
1	0/1	0/1	0/1
....			
18	0/1	0/1	0/1
19	0/1	0/1	0/1

我们这个交换系统提供了三个周期性调度程度（摘挂机检测程序、脉冲识别程序和位间隔识别程序），它们的调用周期分别为 200ms、10ms 和 100ms，所以我们系统的最小调度时间为 10ms。如图所示，每隔 10ms,我们就会检查这个表的一行，如果该行上某一列为 1，我们就执列所对应的任务，如果为 0，就什么都不做。每当执行到这个表的最后一行，调度任务会返回第一行循环执行。而你所要做的就是按照你的理解来填写这个调度表。

三、 实验主要数据结构

函数功能：完成调度表的初始化；

函数原型：initSchTable(int ScheduleTable[SchTabLen][SchTabWdh])； 其中 SchTalLen 和 SchTabWdh 为在 bconstant.h 中的宏定义：

#define SchTabLen 20 //代表这个调度表为 20 行（相邻行之间的时间间隔为 10ms);

#define SchTabWdh 3 //代表三个周期性调度任务——0：摘挂机检测任务；1：脉冲检测任务；2：位间隔检测任务；

四、 实验代码

```
#include "bconstant.h"

extern "C" _declspec(dllexport) void initSchTable(int Sched
uleTable[SchTabLen][SchTabWdh]) {
    for (int i = 0; i < SchTabLen; i++) {
        if (i % 20 == 0) {
            ScheduleTable[i][0] = 1;
        }
        else {
            ScheduleTable[i][0] = 0;
        }
```

```
            ScheduleTable[i][1] = 1;

            if (i % 10 == 0) {
                ScheduleTable[i][2] = 1;
            } else {
                ScheduleTable[i][2] = 0;
            }
        }
        return;
}
```

五、 实验效果检验

当调度表初始化正确时，能够进行正常的通话；如果初始化不正确，可能会造成周期性程序的不正常调用，例如位间隔调度的延迟会造成识别位间隔的延误甚至丢失。

注：由于为循环程序，所以调度表的初始化方案不唯一。

六、 实验结果

程序初始化后能正确检测摘挂机动作并进行通话，与预计结果相符。

七、 实验心得

本次实验比较简单，在理解了时间表调度原理之后，对调度表初始化程序的编程实现也非常容易，很快就可以编写完毕。 通过本次实验，我对时间表调度的具体实现有了初步的理解。

《现代交换原理》实验报告

实验名称 ＿＿＿＿＿摘挂机检测实验＿＿＿＿＿

班　　级 ＿＿＿＿＿＿2017211305＿＿＿＿＿＿

姓　　名 ＿＿＿＿＿＿＿＿于海鑫＿＿＿＿＿＿＿＿

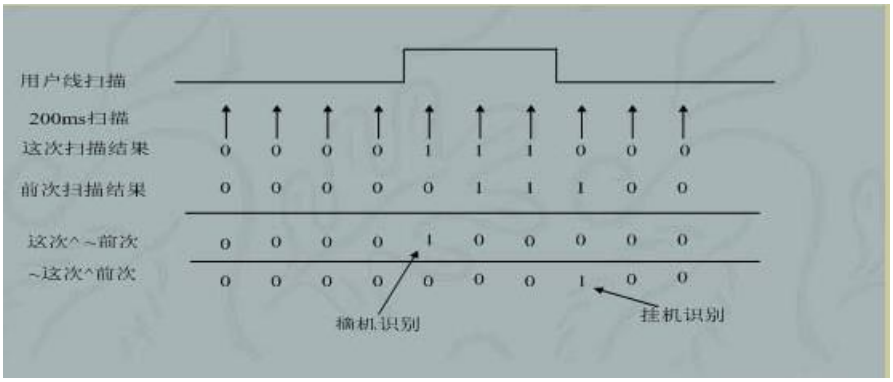
指导教师 ＿＿＿＿＿＿＿＿丁玉荣＿＿＿＿＿＿＿＿

一、 实验目的

摘挂机检测实验用来考查学生对摘挂机检测原理的掌握情况。

二、 实验原理及设计

设用户在挂机状态时扫描输出为“0”，用户在摘机状态时扫描输出为“1”，摘挂机扫描程序的执行周期为 200ms，那么摘机识别，就是在 200ms 的周期性扫描中找到从“0”到“1”的变化点，挂机识别就是在 200ms 的周期性扫描中找到从“1”到“0”的变化点，该原理的示意图如下所示：



在我们的实验中，我们把前 200ms 的线路状态保存以备这次可以读取，同时读出这次的线路状态，把前 200ms 的线路状态取反与这次的线路状态相与，如果为 1，就说明检测到摘机消息了。同理，我们把这次的线路状态取反再与前 200ms 的线路状态相与，如果为 1 就说明检测到挂机消息了，然后把摘挂机信号作为事件放入摘挂机队列中。

三、 实验主要数据结构

函数功能为：检测到摘、挂机事件，并把该事件放入到摘挂机事件队列中。

函数原型：void scanfor200(int linestate200[LINEMAX],int linestate[LINEMAX],UpOnnode * head1, UpOnnode* end1)；其中 LINEMAX 为线路总数，是定义在 "bconstant.h" 中的一个宏，linestate200[LINEMAX] 为已保存的 200ms 前线路状态，linestate[LINEMAX] 为当前的线路状态，head1,end1 为摘挂机队列的首尾指针，该队列已经在主程序中进行了初始化。我们所要做的就是把检测到的摘挂机事件以摘挂机队列节点的形式插入到摘挂机事件队列中。

数据结构说明:

头文件："bconstant.h";(以下的数据结构都已在该文件中定义）

LINEMAX：最大线路数;

int linestate200[LINEMAX],linestate[LINEMAX]：线路从 0 开始编号；状态：1：有电流，0 无电流;

enum UporOn {ehandup,ehandon} :为摘挂机区别符：ehandup 表示摘机，ehandon 表示挂机;

```
struct UpOnnode  {      //摘挂机队列节点结构
        UporOn phonestate;     //摘挂机区别符;
        int linenum;           //线路号（从 0 开始）;
        struct UpOnnode* next;  //指向下一节点的指针;
};
```

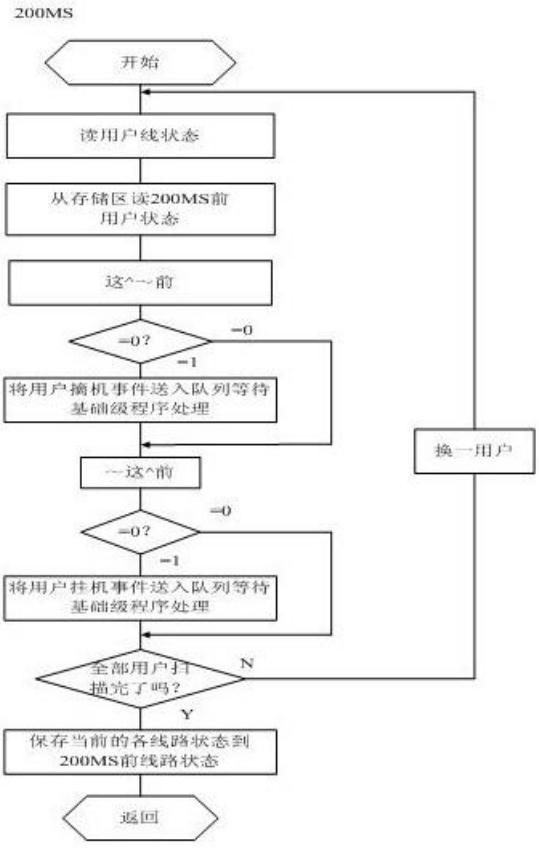
注意事项:

1. 我们编写的模块是基础实验部分预加载的本局交换系统的一个模块而已，在系统中 head1 头指针和 end1 尾指针已经完成初始化。为方便

起见，我们的摘挂机事件队列是一个包含头节点的单向链表，并且头指针指向该头节点，尾指针在初始化时也指向了该节点。所以在我们的函数编写中应保证头指针始终指向该头节点上、尾指针指向摘挂机事件队列的最末一个节点。

2. 注意把这次扫描的线路状态值保存在前 200ms 扫描线路状态数组中,以便主程周期调用。

四、 实验主体流程图



五、 实验代码

```
#include "bconstant.h"
```

```
extern "C" _declspec(dllexport) void scanfor200(int linesta
te200[LINEMAX], int linestate[LINEMAX], UpOnnode *head1, Up
Onnode *end1) {
    int up, down;

    for (int i = 0; i <= LINEMAX; i++) {
        struct UpOnnode *now = new struct UpOnnode;
        up = (!linestate200[i]) && linestate[i];
        down = (!linestate[i]) && linestate200[i];
        if (up || down) {
            if (up) {
                now->phonestate = ehandup;
            } else {
                now->phonestate = ehandon;
            }
            now->linenum = i;
            now->next = 0;
            end1->next = now;
            end1 = now;
        }
    }
    return;
}

extern "C" _declspec(dllexport) void freenode(UpOnnode *nod
e) {
    delete node;
}
```

六、 实验结果

程序执行后能正确检测摘挂机动作并且能进行通话，与预计结果相符，试验成功。

七、 实验心得

本次实验比较简单，很快就可以编写完毕。 通过本次实验，我对摘挂机检测原理的掌握更进一步，也增强了对实验平台的熟悉和对软件的了解。

《现代交换原理》实验报告

实验名称 ＿＿＿＿＿MPLS 编程实验＿＿＿＿＿

班　　级 ＿＿＿＿＿＿2017211305＿＿＿＿＿＿

姓　　名 ＿＿＿＿＿＿＿＿于海鑫＿＿＿＿＿＿＿＿

指导教师 ＿＿＿＿＿＿＿＿丁玉荣＿＿＿＿＿＿＿＿

一、 实验目的

加强学生对 MPLS 交换中标记请求、标记分配与分发、标记分组转发的理解。

二、 实验设计

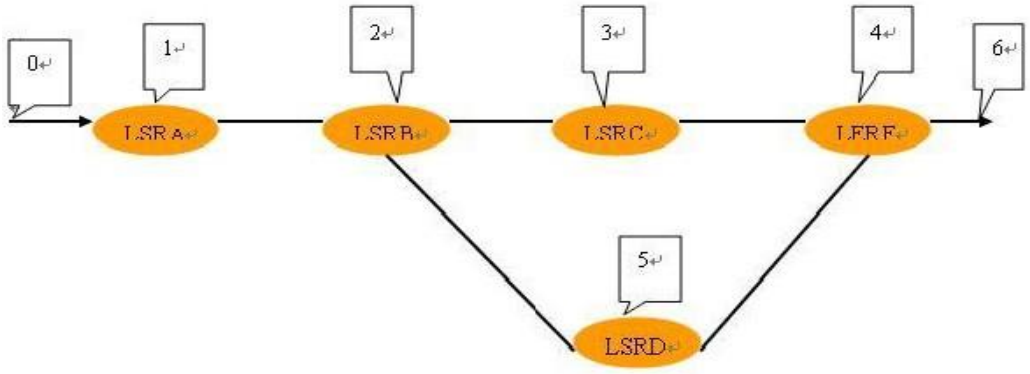
多协议标记交换 MPLS（Multiple Protocol Labeled Switching）技术是将第二层交换和第三层路由结合起来的一种 L2/L3 集成数据传输技术。MPLS 是一项面向连接的交换技术，因此有建立连接的过程。各个 MPLS 设备运行 路由协议，在标记分发协议 LDP 的控制下根据计算得到的路由在相邻的路由器进行标记分配和分发，从而通过标记的拼接建立起从网络入口到出口的标记交换路径 LSP。

在数据转发过程中，入口标记路由器 LER 根据数据流的属性比如网络层目的地址等将分组映射到某一转发等价类 FEC，并为分组绑定标记。核心标记交换路由器 LSR 只需根据分组中所携带的标记进行转发即可。出口标记路由器 LER 弹出标记，根据分组的网络层目的地址将分组转发到下一跳。MPLS 节点（MPLS 标记交换路由器 LSR 或 MPLS 边缘路由器 LER）均要创建和维护传统的路由表和标记信息库 LIB。

路由表记录记录路由信息，用于转发网络层分组和标记分发从而建立标记交换路径。LIB 记录了本地节点分配的标记与从邻接 MPLS 节点收到的标记之间的映射关系，用于标记分组的转发。

MPLS 技术的核心实质在于：（1）网络中分组基于标记的转发（2）LDP 协议控制下的进行标记分发从而建立标记交换路径 LSP。

实验网络的拓扑结构（节点分布示意图）：



三、 实验主要数据结构

所需要的头文件："mplsconstant.h"

其中的主要数据结构为：

```
//发送的请求信息包数据结构
struct ReqType
{
    int iFirstNode;    //请求信息包的源节点
    int iEndNode;      //请求信息包的目的节点
    double ipaddress; //请求信息包包含的网络层目的 IP 地址前缀（例如 197.42）
};

//路由表表项的数据结构
struct routertype
{
    double ipaddress; //网络层目的地址前缀
    int nexthop;       //下一跳节点
    int lasthop;       //上一跳节点
    int inpoint;       //入端口号
    int outpoint;      //出端口号
};

//标记信息表表项的数据结构
```

```
struct libtype
{
    double ipaddress; //网络层目的地址前缀
    int inpoint;      //入端口号
    int outpoint;     //出端口号
    int inlabel;      //入标记值
    int outlabel;     //出标记值
};

//发送的标记信息包数据结构

struct LabelPack
{
    int iFirstNode; //源节点号
    int iEndNode;   //目的节点号
    int labelvalue; //标签值
};

struct funcusedtype
{
    struct libtype libinfo;      //包含的标记信息表项
    struct LabelPack labelinfo; //包含的标记信息包数据结构
};

//发送的标记分组信息包类型

struct LabelledDataPack
{
    int iFirstNode;                 //源节点号
    int iEndNode;                   //目的节点号
    struct MessageType DataInfo; //包含的标记分组类型信息
};

//标记分组类型
struct MessageType
{
    double ipaddress; //网络层目的地址前缀
    int labelvalue;   //输出标签值
};
```

1. 标记请求实验要求函数：

```
extern "C" _declspec(dllexport) struct ReqType req_process(
int idnow, struct routertype routenow) {
    struct ReqType reqtemp;
    return reqtemp;
}
```

参数意义：

int idnow：当前的节点号；

struct routertype routenow：当前所指的路由表的表项；

函数要求：根据提供的当前节点号和路由表表项值产生标记请求包；

过程描述：

标记请求包的源节点号由当前节点号提供，目的节点号和 ip 地址前缀由当前所指的路由表表项的下一跳节点和 ip 地址前缀提供；

2：标记分配与分发实验：

```
extern "C" _declspec(dllexport) struct funcusedtype label_p
rocess(struct routertype routenow, int labelout, int idnow)
 {
    struct funcusedtype tempstruct;
    return tempstruct;
}
```

参数意义：

struct routertype routenow：当前所指的路由表表项；

int labelout：分配的输出标签号；

int idnow：当前的节点号；

函数要求：

该函数要求根据提供的路由表当前表项、分配的输出标签号和当前节点号，构造一 funcusedtype 信息包。注：各节点的输入标签可以自由选定，但必须是 1-9 的整数；

过程描述：

该 funcusedtype 信息包的 libinfo 部分可由当前的路由表表项、当前分配的标签号的有关部分构成； labelinfo 部分由当前节点号和当前的路由表表项的有关部分构成；

3．标记分组转发实验

```
extern "C" _declspec(dllexport) struct LabelledDataPack pac
k_process(struct routertype routenow, struct libtype libnow
, int idnow) {
    struct LabelledDataPack packtemp;
    return packtemp;
}
```

参数意义：

struct routertype routenow： 当前所指的路由表表项；

struct libtype libnow： 当前的标签信息表表项；

int idnow： 当前的节点号；

函数要求：

该函数要求根据提供的路由表表项、标签信息表表项和当前节点号，构造出一个标签数据信息包。

过程描述：

该标签信息包的源节点、目的节点、 IP 地址前缀和标签值均可由当前节点号、路由表表项和标签信息表表项构成；

四、 实验代码

A. 实验 1

```
#include "mplsconstant.h"

extern "C" _declspec(dllexport) struct ReqType req_process(
int idnow, struct routertype routenow) {
    struct ReqType reqtemp;
    reqtemp.iFirstNode = idnow;
    reqtemp.iEndNode = routenow.nexthop;
    reqtemp.ipaddress = routenow.ipaddress;
    return reqtemp;
}
```

B. 实验 2

```
#include "mplsconstant.h"

extern "C" _declspec(dllexport) struct funcusedtype label_p
rocess(struct routertype routenow, int labelout, int idnow)
 {
    struct funcusedtype tempstruct;
    tempstruct.libinfo.ipaddress = routenow.ipaddress;
    tempstruct.libinfo.inpoint = routenow.inpoint;
    tempstruct.libinfo.outpoint = routenow.outpoint;
    tempstruct.libinfo.inlabel = 7;
    tempstruct.libinfo.outlabel = labelout;
    tempstruct.labelinfo.iFirstNode = idnow;
    tempstruct.labelinfo.iEndNode = routenow.lasthop;
    tempstruct.labelinfo.labelvalue = tempstruct.libinfo.in
label;
    return tempstruct;
}
```

C. 实验 3

```
#include "mplsconstant.h"

extern "C" _declspec(dllexport) struct LabelledDataPack pac
k_process(struct routertype routenow, struct libtype libnow
, int idnow) {
    struct LabelledDataPack packtemp;
```

```
    packtemp.iFirstNode = idnow;
    packtemp.iEndNode = routenow.nexthop;
    packtemp.DataInfo.ipaddress = routenow.ipaddress;
    packtemp.DataInfo.labelvalue = libnow.outlabel;

    return packtemp;
}
```

五、 实验结果

实验结果符合预期，实验成功。

六、 实验心得

这次实验比较简单，代码实现基本就是赋值的操作，最重要的是增强了我对MPLS 原理和工作过程的理解 。