湖南大學

HUNAN UNIVERSITY

嵌入式计算机系统实验二报告

目录

一、实验环境、目的、内容

（1）实验环境

个人电脑、Keil、STC-ISP、STC-B 学习板。

（2）实验目的

基于 STC-B 开发板，尝试、体验并实现一个极简嵌入式操作系统及其应用。

（3）实验内容

本次实验内容具体分为 0 级、1 级和 2 级三个档次。其中 0 级属于必做项目，在验收时应能够回答 0 级实验的问题并在实验报告中展现。1 级与 2 级实验只需选择其中之一完成即可，其中 1 级实验的验收分数上限为 85 分，而 2 级实验的验收分数上限为 100 分，相应的 2 级实验内容会更有挑战，请自行选择，在验收时只能从 1 级和 2 级选择其中一个进行现场展示和验收问答，欢迎同时完成 1 级与 2 级实验，但无任何额外加分。

我选择的是**【2 级实验】**。

二、0 级实验

在之前的实验（包括小学期的实践和本课程的第一次实验）中，我们在进行相关代码设计时，总是在 main()函数中使用 while 死循环来执行核心功能，比如这样：

```
while (1)
    {
        MySTC_OS();
    }
```

请考虑，倘若有两个或两个以上的任务（Task1，Task2,……），共享某种资源（例如某全局变量），或者任务之间要进行通信，这时如果还写成这样：

```
while (1) {
    Task1 () ;
    Task2 () ;    }
```

会出现什么问题？请根据前修课程和你的实践经验，明确给出你的答案。

在多任务环境中，如果使用 while (1)死循环来执行多个任务，可能会遇到以下问题：

（1）竞态条件

当两个或多个任务共享同一个资源时，如果没有适当的同步机制，它们可能会在不适当的时刻访问资源，导致数据不一致或不可预测的行为。例如，如果 Task1 和 Task2 都尝试更新同一个全局变量，在 Task1 取出变量后发生了时钟中断，Task2 对变量进行修改，再运行 Task1，那么，虽然两个任务都运行了一次，但 Task2 的运行并没有生效，从而导致我们不希望看到的错误行为发生。

（2）死锁

死锁发生在两个或多个任务都在等待对方释放资源时。如果 Task1 持有资源 A 并等待资源 B，而 Task2 持有资源 B 并等待资源 A，那么两个任务都会无限期等待，导致系统资源无法被有效利用。

（3）饥饿

某些任务可能因为其他任务的无限循环或高优先级任务的持续占用而无法获得必要的资源，导致任务被饿死。

（4）优先级反转

当高优先级任务等待低优先级任务释放资源时，如果中间有中等优先级的任务抢占了 CPU，可能会导致高优先级任务长时间等待，而低优先级任务无法执行，这违反了优先级调度的原则。

为了解决这些问题，通常需要引入操作系统的任务调度器，使用锁、信号量、事件等同步机制来确保任务能够安全、有效地共享资源和进行通信。此外，还可以通过设置任务优先级、采用合适的调度算法来提高系统的响应性和公平性。

也就是我们的 STC-B-OS 中所实现的功能，在该系统中，可以用信号量来实现锁的行为。

三、STC-B-OS 改进

（1）进程调度——优先级调度模式

将原先的轮询调度 RR 修改成了优先级调度，其实我不是很理解老师上课时

候说的由于访问片外存储导致效率并没有上升的问题，我认为优先级调度的目的不是为了让效率变快，而是让事件在处理的时候存在轻重缓急，所以我并没有对性能进行测试，而是仅对优先级的实现上进行了测试。

初始化时增加了优先级处理。

```
void start_process(u16 entry, u16 pid, u32 param, u8 priority)
{
    //XBPH,L
    process_context[pid][13] = (((u16)process_xstack[pid] + PROCESS_XSTACKSIZE)&0xff00)>>8;
    process_context[pid][14] = ((u16)process_xstack[pid] + PROCESS_XSTACKSIZE)&0x00ff;

    //PCH, PCL
    process_context[pid][15] = (entry&0xff00) >> 8;
    process_context[pid][16] = (entry&0x00ff);


    //SP
    if(get_stack_index(pid)!=-1)
        process_context[pid][17] = process_stack[get_stack_index(pid)]; //stack present in memory, use absolute address
    else
        process_context[pid][17] = 0; //stack in stackswap space, use address relative to stack start

    //R4-R7 (param)
    process_context[pid][9] = (param & 0xff000000) >> 24;
    process_context[pid][10] = (param & 0x00ff0000) >> 16;
    process_context[pid][11] = (param & 0x0000ff00) >> 8;
    process_context[pid][12] = (param & 0x000000ff);

    proc_time_share[pid] = DEFAULT_TIMESLICES;
    proc_priority[pid] = priority;
    process_slot |= BIT(pid);
}
```

寻找下一个运行的进程时，定义了一个局部变量 max_priority 存储当前最大优先级，在遍历当前所有进程后选择优先级最大的进行运行。优先级级别定义为 1-8（一共最多支持 8 个进程同时运行），max_priority 初始化为 9。

```
u8 select_process()
{
        XDATA u8 tmp_process;
        XDATA u8 fin_process;
        XDATA u8 max_priority = 9;
        XDATA u8 flag=0;
        //current_process can be 8 (kernel startup) or 9 (idle spin)
        //so we have to set current_process to 0 in that situation.
        if(current_process >= 8) current_process = 0;
        fin_process = current_process;



        while((tmp_process = NEXT(tmp_process)) != current_process)
        {
                if(process_ready(tmp_process))
                    if(proc_priority[tmp_process]<max_priority)//检查优先级
                    {
                            max_priority = proc_priority[tmp_process];
                            fin_process = tmp_process;
                            flag=1;
                    }

        }
        if(flag==1)
            goto SCHEDULER_END;
    //if no other process can run, check if current process can run again
    if (process_ready(current_process))
    {
            fin_process = current_process;
            goto SCHEDULER_END;
        }

  //Can't find a process to run, reuturn 9(invalid)
    //ISR should recognize this and put system to spin until next interrupt
    fin_process = 9;

    SCHEDULER_END:;
    return fin_process;
```

需要注意的是，为了展示效果，需要在.h 文件中修改时间片，否则运行时会出现显示闪动很快的情况。

```
#define DEFAULT_TIMESLICES 255
//修改时间片
```

在 main 文件中设置三个任务，设置优先级进行调用，与未修改前的效果进行对比。一个实现了优先级调度，一个是 RR 随机调度。

```
void task1(u16 param)
{
    while(1){
        seg_set_number(11111111);
    }
}
void task2(u16 param)
{
    while(1){
        seg_set_number(22222222);
    }
}
void task3(u16 param)
{
    while(1){
        seg_set_number(33333333);
    }
}
```

```
    start_process((u16)task1,0,0,3);
    start_process((u16)task2,1,0,2);
    start_process((u16)task3,2,0,1);
```

尝试实现多级队列调度未成功。

（2）缓存访问——LRU

在 stack.c 中，如果调度时没有找到 idata 的 5 个物理栈中的空位，则需要和 xdata 中的栈进行交换。这里选择交换的“victim”时使用的是随机策略，我将其修改为 LRU，即最近最少访问原则策略。

在该交换函数中，我定义了一个 least_recently_used 作为跟踪当前最近最少使用原则栈槽位的索引，然后遍历来寻找当前最少使用栈，再进行交换。

因为这个优化的效果难以演示，我没有设置演示范例。

```
void stackswap(u8 swap_index)
{
    XDATA u8 i, temp, least_recently_used;

    // Find the least recently used stack slot
    least_recently_used = 0; // 跟踪最近最少使用的栈槽位的索引

    //从索引1开始，查找最近最少使用的栈槽位。这是通过比较栈槽位的最后一个元素，即进程ID，来实现的。最小的进程ID表示最近最少使用的栈。
    for (i = 1; i < 5; i++) {
        if (process_stack[i][PROCESS_STACKSIZE-1] < process_stack[least_recently_used][PROCESS_STACKSIZE-1]) {
            least_recently_used = i;
        }
    }

    for (i = 0; i < PROCESS_STACKSIZE; i++) {
        temp = process_stack[least_recently_used][i];
        process_stack[least_recently_used][i] = process_stack_swap[swap_index][i];
        process_stack_swap[swap_index][i] = temp;
    }

    // Convert context SP of process whose stack is being swapped out to a relative address
    process_context[process_stack_swap[swap_index][PROCESS_STACKSIZE-1]][17] -= (u8)process_stack[least_recently_used];

    // Convert context SP of process whose stack is being swapped in to an absolute address
    process_context[process_stack[least_recently_used][PROCESS_STACKSIZE-1]][17] += (u8)process_stack[least_recently_used];
}
```

（3）宏定义——代码复用

注意到原先的 event.c 中有很多格式一致的函数，基本就是复制粘贴然后修改了部分变量来进行的，这样后续如果修改的话会很麻烦，因此在 event.h 中增加了一个宏定义，来增强当前代码的可维护性。

```
43 #define SET_EVENT_FLAG(event)  (curr_events |= (event))
```

```
33 void collect_btnevts()
34 {
35     update_button_state();
36 
37     SET_EVENT_FLAG(btnstate_posedge & BTNSTATE_B1 ? EVT_BTN1_DN : 0);
38     SET_EVENT_FLAG(btnstate_posedge & BTNSTATE_B2 ? EVT_BTN2_DN : 0);
39     SET_EVENT_FLAG(btnstate_posedge & BTNSTATE_B3 ? EVT_BTN3_DN : 0);
40     SET_EVENT_FLAG(btnstate_posedge & BTNSTATE_UP ? EVT_NAV_U : 0);
41     SET_EVENT_FLAG(btnstate_posedge & BTNSTATE_DOWN ? EVT_NAV_D : 0);
42     SET_EVENT_FLAG(btnstate_posedge & BTNSTATE_LEFT ? EVT_NAV_L : 0);
43     SET_EVENT_FLAG(btnstate_posedge & BTNSTATE_RIGHT ? EVT_NAV_R : 0);
44     SET_EVENT_FLAG(btnstate_posedge & BTNSTATE_PUSH ? EVT_NAV_PUSH : 0);
45 
46     SET_EVENT_FLAG(btnstate_posedge & BTNSTATE_B1 ? EVT_BTN1_UP : 0);
47     SET_EVENT_FLAG(btnstate_posedge & BTNSTATE_B2 ? EVT_BTN2_UP : 0);
48     SET_EVENT_FLAG(btnstate_posedge & BTNSTATE_B3 ? EVT_NAV_BTN3_RESET : 0);
49     SET_EVENT_FLAG(btnstate_posedge & BTNSTATE_UP ? EVT_NAV_BTN3_RESET : 0);
50     SET_EVENT_FLAG(btnstate_posedge & BTNSTATE_DOWN ? EVT_NAV_BTN3_RESET : 0);
51     SET_EVENT_FLAG(btnstate_posedge & BTNSTATE_LEFT ? EVT_NAV_BTN3_RESET : 0);
52     SET_EVENT_FLAG(btnstate_posedge & BTNSTATE_RIGHT ? EVT_NAV_BTN3_RESET : 0);
53     SET_EVENT_FLAG(btnstate_posedge & BTNSTATE_PUSH ? EVT_NAV_BTN3_RESET : 0);
54 }
```

```
57 void collect_uartevts() //VOID
58 {
59     if(rs485_evtstate)
60     {
61             rs485_evtstate = 0;
62             SET_EVENT_FLAG(EVT_UART2_RECV);
63     }
64     if(usbcom_evtstate)
65     {
66             usbcom_evtstate = 0;
67             SET_EVENT_FLAG(EVT_UART1_RECV);
68     }
69 }
```

四、STC-B-OS 游戏设计——二人交互的井字棋游戏

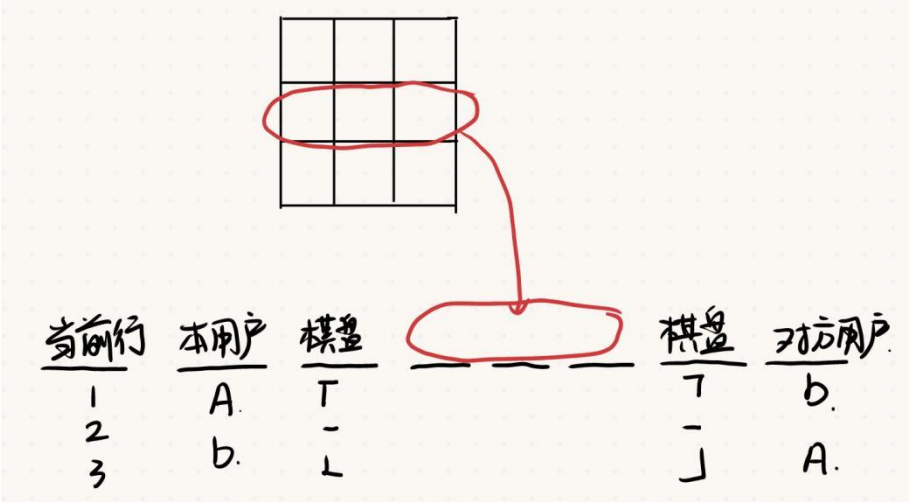
在先前修改版本的基础上，我设计了一个可二人交互的井字棋游戏。游戏规则与平常的井字棋游戏规则相同，三行三列，两个人轮流落子，如果有一人的棋子完成了横、竖、斜任一种三个连线即胜利。

在我的游戏中，二者显示的界面是相同的，首先是滚动的欢迎界面，按 key2 进行用户选择显示，蜂鸣器发声，再次按下选中，两者选择的用户不能相同；然后按 key1 进入游戏，LED 一次显示一行的内容，初始时游戏默认显示第二行，当前选中位置会进行闪动，可以通过导航按键进行当前位置的移动，摁下导航按键将在当前位置落子，随后下棋权将交给对方；一方获胜后，将显示欢迎界面，

此时长按 key2 将显示当前比分，按 key1 将进行下一轮比赛，比赛采取三局两胜制度，积分累计不会超过 3。

总代码量为 700 行左右。

游戏中显示界面如下图，后续内容将进行部分代码解释。



（1）头文件与变量定义

需要说明的是，虽然这里定义了几个延时函数，本意是对于蜂鸣器的控制，但后期测试的时候发现延时函数使用会出现很多莫名其妙的 bug（就像小学期一样），未使用。

需要在 main.c 中调用初始化函数 game_init()和游戏开始函数 gameStart()。

game.h

```
#ifndef __INTRINS_H__
#define __INTRINS_H__

#pragma SAVE

#if defined (__CX2__)
#pragma FUNCTIONS(STATIC)
/* intrinsic functions are reentrant, but need static attribute */
#endif

extern void          _nop_     (void);
extern bit           _testbit_ (bit);
extern unsigned char _cror_    (unsigned char, unsigned char);
extern unsigned int  _iror_    (unsigned int,  unsigned char);
extern unsigned long _lror_    (unsigned long, unsigned char);
extern unsigned char _crol_    (unsigned char, unsigned char);
extern unsigned int  _irol_    (unsigned int,  unsigned char);
extern unsigned long _lrol_    (unsigned long, unsigned char);
extern unsigned char _chkfloat_(float);
#if defined (__CX2__)
extern int           abs       (int);
#endif
#if !defined (__CX2__)
```

```
extern void          _push_    (unsigned char _sfr);
extern void          _pop_     (unsigned char _sfr);
#endif

#pragma RESTORE


void Uart2Init( void );//

void refreshChessBoard();
void refreshBoard();

void game_init();//

void Delay5ms();
void Delay30ms();
void Delay50ms();
void Delay100ms();
void Delay2000ms();
void DelayMs( unsigned int xms );

unsigned char NavKeyCheck();

void tick1();
void tick2();
void tick3();
void tick4();
```

```
void sendDatas();
void displayLED();

unsigned char isOver();
void displayChessBoard();
void displayRoll(unsigned char arr[]);

void gameStart();
void gameOver();

void playChess();

void NavKey_Process();

void selectCharacter();

void getData();//

void Uart2_Process( void );//
void T0_Process();//
void T1_Process();//

#endif
```

```
    //start process
    game_init();
    gameStart();
```

btSendBusy 用于表示当前是否正在通信。

datas 用于存储发送的信息。

i,j,x,y 用作一些下标的选择。

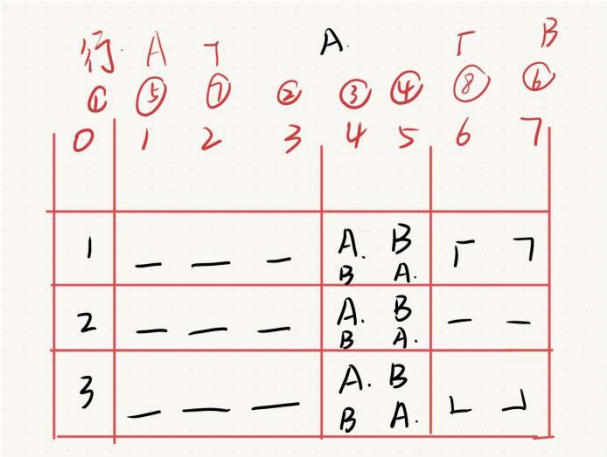
led 是 LED 灯位数指示。

arrRoll[]是数码管滚动显示的内容。

arrSegSelect[22]用于数码管段选，dig[]用于数码管位选。

Board[4][8]记录当前棋盘，scoreBoard[8]记录当前比分板。

以下为当前棋盘 Board[4][8]中存储的内容，在显示中由 dig[]指示显示顺序。



curPlayer 和 myPlayer 指示当前玩家与本玩家角色信息。

gameNum 记录比赛局数。

blinkIndex 指示当前闪烁位（后续该功能未实现）。

point 指示当前小数点在的位置（后续该功能未实现）。

start，over，rest 指示当前游戏状态，用于条件判断。

```
/*****定义变量*****/
bit btSendBusy;                                  //为1时忙（发送数据），为0时闲
uchar datas;                                     //数据向量，用于两个玩家之间的数据交互
uchar code dig[]={0,3,4,5,1,7,2,6};                //数码管位选
uchar digIndex=0;                                //数码管位选下标
uchar i,j,x,y;                                   //循环下标(i,j) 棋盘坐标（x,y）
uchar led=0x01;
uchar arrRoll[]={0x73,0x38,0x77,0x6e,0x00,0x00,0x39,0x76,0x79,0x6D,0x6D,0x00,0x00};
//play  chess
uchar arrSegSelect[22] = {                                                  //段选，显示0-fc
0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f,0x6f, 0x77, 0x7c, 0x39, 0x5e, 0x79, 0x71,0x44,0x50,0x40,0x42,0x60,0x00};
uchar Board[4][8]={
            {0,0,0,0,0x0a,0xb,0x13,0x14},
            {1,0,0,0,0x0a,0xb,0x13,0x14},
            {2,0,0,0,0x0a,0xb,0x12,0x12},
            {3,0,0,0,0x0a,0xb,0x10,0x11}};                                  //棋盘
uchar scoreBoard[8]={0,0,0x12,0xa,0xb,0x12,0,0};                            //比分板
uchar curPlayer=0x0a;                                                       //当前玩家
uchar myPlayer=0x0a;                                                        //己方玩家
uchar myScore=0,otherScore=0;                                               //比分
uchar gameNum;                                                              //比赛局数
uchar blinkIndex=1;                                                         //闪烁位
uchar point=4;                                                              //默认小数点在左侧
uchar count=0;                                                              //定时器辅助计数用
uchar start=0,over=0,rest=0;                                                    //游戏是否开始/结束
uchar winner;                                                               //赢家
bit isSelect=0;                                                         //已经选择完毕
bit btBeepFlag;                                                         //按键声音标志
uchar ucTimerH,ucTimerL;                                                    //定义定时器的重装值


uchar uiLed = 0x01;         //LED灯值寄存
uint  uiLedCnt = 0;         //LED灯累计计数器
```

（2）初始化

对变量和引脚进行了一些初始化。

```
void game_init(){
    P3M0 = 0x10;        //P3.4推挽
    sbtLedSel = 0;      //选择数码管作为输出
    P1ASF = 0x80;       //P1.7作为模拟功能A/D使用
    ADC_RES = 0;        //转换结果清零
    ADC_CONTR = 0x8F;   //cstAdcPower = 1
    CLK_DIV = 0X00;     //ADRJ = 0    ADC_RES存放高八位结果
    btBeepFlag = 0;         //蜂鸣器初始化为0


    x=2;                                    //坐标初始化为（2,2）
    y=2;
    digIndex = 0;               //初始化下标和辅助计数值
    winner=0;
    start=0;                        //开始，结束
    over=0;
    myScore=0;
    myScore=0;
    count = 0;              //滚动辅助
    myPlayer=0x0a;
    datas=0;                //数据初始化
    gameNum=0;

    //定时器T0
    TMOD=0x00;                          //16位可重装模式
    ET0=1;                                  //开启定时器T0中断
    TH0=(65535-1000)/256;   //定时1ms
    TL0=(65535-1000)%256;
    TR0=1;                              //T0开启定时

    //定时器T1
    TH1 = 0xD8;
    TL1 = 0xEF;
    ET1 = 1;
    TR1 = 0;                        //定时器处于关闭状态，当
    EA=1;
```

```
    //485初始化                         波特率生成
    sbtM485_TRN = 0 ;   //初始为接收状态
    P_SW2 |= 0x01 ;     //切换串口2的管脚到P4.6,P4.7
    //Uart2Init();
    btSendBusy = 1 ;
    IE2 |= 0x01 ;       //开串行口2中断
    IP2 |= 0x01 ;       //设置串行口中断：高优先级

    datas=0;                            //初始化交互数据为0
}
```

（3）执行过程

在执行中，大部分功能由定时器 T0 实现调用，小部分由中断调用。

定时器 T0 中断处理函数中，它检测 Key1 的状态来判断是否开始游戏，并在开始后进行通信；检测 Key2 是否选择角色信息并通信；每次调用时对当前位值自加并根据状态位确定当前显示并调用相应的函数；设置蜂鸣器按键发声。

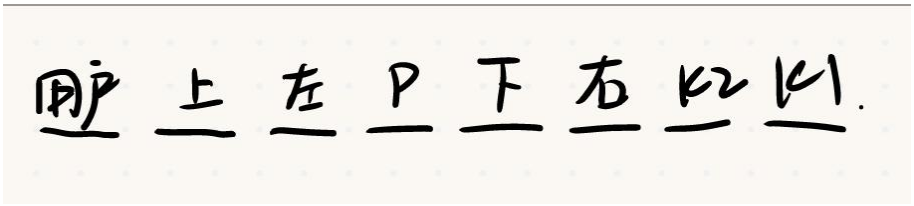
```
/*---------定时器0中断处理函数---------*/
void T0_Process() interrupt 1                   //中断
{
    //检测按键Key1游戏开始
    if(sbtKey1==0 && (!start) && (!over))                   //按键按下并且start=0并且已经选择了游戏玩家角色，则可以提前开始开始
    {
        //tick4();
        if(sbtKey1==0){
            while( !sbtKey1 );
            start=1;                                        //key1按下，游戏开始
            datas=0x01;                                     //设置Key1数据，记住，玩家信息在sendData中添加
            sendDatas();                                    //发送数据
        }
    }

    //选择角色信息
    if( sbtKey2 == 0 && gameNum==0 && (!start) && (!over))  //当游戏未开始，且游戏局数为0的时候，Key2将会选择玩家
    {
        Delay5ms();                                         //延时消抖
        //tick4();
        if( sbtKey2 == 0 )
        {
            while( !sbtKey2 );                              //等待K1放开
            selectCharacter();
            isSelect=1;
            datas=0x02;                                     //发送key2
            sendDatas();                                    //直接发送数据
        }
    }

    /*****显示部分*****/
    digIndex++;                                             //数码管的数组下标
    if(digIndex==8){
        digIndex=0;                                         //下标归零
        count++;
    }
    if(sbtKey2 == 0 && (start||over||gameNum))              //开始或者结束的时候可以查看棋盘
        displayScore();                                     //3局开始后，被按下，显示比分板
    else if((!start&&!over&&gameNum==0&&!isSelect&&!rest)||(start&&over&&!rest))          //未开始或者已经全局结束
        displayRoll(arrRoll);                               //显示滚动字母
    else
        displayChessBoard();                                //显示棋盘

    //蜂鸣器按键发声
    if(btBeepFlag && (!over))
        sbtBeep = ~sbtBeep;                                 //产生方波使得蜂鸣器发声
    else if(TR1==0)
        sbtBeep = 0;                                        //TR1=0的时候，说明没有播放音乐，停止发声，并将sbtBeep端口置于低电平
}
```

串口二中断处理程序中对通信进行处理，这里是 getData()函数唯一的调用点。

```
/*---------串口2中断处理程序，数据接收---------*/
void Uart2_Process( void ) interrupt 8 using 1
{
    if( S2CON & cstUart2Ri )                    //无校验&接收中断请求标志位
    {
        datas = S2BUF ;                         //从串口中接收数据暂存
        S2CON &= ~cstUart2Ri;                   //接收中断标志位清0
        getData();
    }
    if( S2CON & cstUart2Ti )                    //无校验&发送中断请求标志位
    {
        btSendBusy = 0 ;                        //清除忙信号
        S2CON &= ~cstUart2Ti ;                  //发送中断标志位清0
    }
}
```

（4）通信处理与导航按键处理

一次通信中，我们只发送一个字节，8bit，这 8bit 被划分成如下形式。



首位代表当前用户，这里将对 myPlayer 的末位与发送的标志进行异或操作，由于初始化函数中将 myPlayer 都初始化为 1，这里使用异或来保证玩家互斥，检查完后去除标志位；0,1 位处理完成后，将右移两位，对其余位进行检查。

```
//数据解析函数
void getData()
{
    isSelect=1;
    if(!((datas>>7)^(myPlayer&0x01))){                    //当标志相同的时候，确认信息
        if(start) return;                                 //如果交互数据来自自己，那么拒绝接收
        else{
            myPlayer^=0x01;                               //异或，保持玩家互斥
        }
    }
    datas =datas&0x7f;                                    //去掉最高位的来源信息，0111 1111
    if(datas)                                             //如果数据不为空
    {
        //检查key1游戏开始功能
        if(datas&0x01){
            start=1;
            //tick4();
        }
        //key2没有发过来的必要，因为双方没有权利知道对方是否查看比分板，
        //但是Key2在角色选择中发挥了作用,这个时候同步棋盘
        if(datas&0x02){
            for(i=0;i<4;i++){
                Board[i][4]=myPlayer;
                Board[i][5]=myPlayer^0x01;                //第二个玩家
            }
            for(i=1;i<4;i++){                             //更新棋盘第2行
                Board[2][i]=myPlayer;                     //显示选择角色的信息
            }
            Board[2][0]=21;                               //显示为0
            Board[2][4]=21;                               //显示第二行玩家
            Board[2][5]=21;                               //显示第二行的玩家
        }
        datas>>=2;
        //导航键
```

对应完成位置移动操作，和按下键的落子操作。

```
        //导航键
        switch(datas)
        {
            case 0x01:                    //右
                if(y!=3){
                    y++;
                    //tick1();
                }
                break;
            case 0x02:                    //下
                if(x!=3){
                    x++;
                    //tick1();
                }
                break;
            case 0x04:                    //按下
                if(over) return;          //游戏处于结束状态，则直接返回
                if(Board[x][y]==0){
                    playChess();
                }                         //落子
                break;
            case 0x08:
                if(y!=1){
                    y--;
                    //tick1();
                }
                break;                    //左
            case 0x10:                    //上
                if(x!=1){
                    x--;
                    //tick1();
                }
                break;
        }
        datas=0;    //清空数据发送接收，保证只执行一次
```

导航按键处理函数 NavKey_Process()与以上基本相同，但是需要发送数据并判断游戏是否结束。

```
530         //先发送完数据，再结束游戏
531         datas=(1<<(ucNavKeyPast+1));                //编码，设置发送数据
532         sendDatas();
533     }
534     if(over&&start&&gameNum<3){
535         gameOver();
536     }
537     Delay30ms();
538 }
```

```
315 /*-----数据发送函数----*/
316 void sendDatas()
317 {
318     sbtM485_TRN = 1 ;                   //MAX485使能引脚，开启485串口
319     S2BUF = datas | (myPlayer<<7);      //将发送数据写入缓存区,带上来源机,A玩家则是以0开头，B玩家以1开头，1010，1011
320     while( btSendBusy );                //等待数据发送完毕
321     btSendBusy = 1 ;                    //再次设置为1，表示发送忙
322     sbtM485_TRN = 0 ;                   //关闭485串口
323     datas=0;
324 }
```

对导航操作进行支持。

```
260 /*----------获取导航按键值子函数---------*/
261 unsigned char NavKeyCheck()
262 {
263     u8 key;
264     key = adc_read(cstAdcChs17);      //获取AD的值
265     if( key != 255 )     //有按键按下时
266     {
267         Delay5ms();
268         key = adc_read(cstAdcChs17);
269         if( key != 255 )              //按键消抖 仍有按键按下
270         {
271             key = key & 0xE0;         //获取高3位，其他位清零
272             key = _cror_( key, 5 ); //循环右移5位 获取A/D转换高三位值，减小误差
273             return key;
274         }
275     }
276     return 0x07;          //没有按键按下时返回值0x07
277 }
```

（5）落子与游戏结束

被上个函数调用。

如果当前游戏未结束且当前位置未落子，将落子并更新 curPlayer。

这里进行了 point 的更新，该变量是为了实现在当前显示 curPlayer 位下加点的效果，但是后续实现时发现更新总是慢通信一步，检查了一天也没有找到原因，我觉得可能是不小心碰到了这个板子的底层 bug，遂放弃。

```
/*----------落子函数------------*/
void playChess(){
    //tick2();
    if(over) return;                                    //游戏已经结束，则无法落子

    Board[x][y]=curPlayer;                              //落子

    curPlayer^=0x01;                                    //0000 1010  ！0000 1011取反最后一位，交换curPlayer

    over=isOver();    //判断输赢
    if(over && over!=0x10) {
        winner=over;                                    //获取游戏赢家
    }
    else{
    if(curPlayer==myPlayer)                           //左侧点
            point=4;
    else
            point=5;                                    //右侧
    }
}
```

判断是否游戏结束，如果结束了且未到三局则对分数进行更新，并更新当前局数和比分，重新做一些变量的初始化，再次开始。

后续测试的时候将所有的音乐都删除掉了，出于对有限空间的考虑。

```
/*-------------游戏结束处理-------------*/
void gameOver(){
        if(gameNum==3) return;
        if(winner==myPlayer){
            rest=1;
            //PlayMusic(100,arrMusicSuccess);
            myScore++;
            displayLED();
            rest=0;
        }
        else{
            //PlayMusic(100,arrMusicFail);
            otherScore++;
        }
        refreshBoard();                 //刷新比分

        Delay2000ms();


        //开始新的一轮游戏
        gameNum++;                      //局数计数
        curPlayer=0x0a;                 //重新定位当前玩家
        winner=0;
        start=0;
        over=0;
        refreshChessBoard();            //刷新初始化棋盘

        if(gameNum!=3){
            gameStart();
        }
        else                            //游戏结束
        {
            //PlayMusic(200,arrMusicBM);    //播放背景音乐
            start=1;
            over=1;
        }
}
```

一些初始化：

```
//初始化棋盘
void refreshChessBoard(){
    //初始化棋盘
    for(i=1;i<4;i++){
        for(j=0;j<4;j++)
            Board[i][j]=0;
    }

    //初始化玩家信息
    for(i=1;i<4;i++){
        Board[i][4]=myPlayer;                   //显示我的玩家
        Board[i][5]=myPlayer^0x01;              //显示对方玩家
        Board[i][0]=i;                          //行号
    }

    //初始化小数点
    if(myPlayer==0x0b)
        point=5;
    else
        point=4;

    //初始化光标
    x=2;
    y=2;
}
```

```
//刷新比分板
void refreshBoard(){
    scoreBoard[3]=myPlayer;         //左侧显示的是我的得分
    scoreBoard[4]=myPlayer^0x01;

    scoreBoard[1]=myScore;          //更新比分
    scoreBoard[7]=otherScore;
}
```

对输赢逻辑的判断。

```
/*-------------判断输赢-------------*/
unsigned char isOver(){
    //假设一定分出胜负的情况
    for(i=1;i<4;i++){
        if(Board[i][1]!=0&&Board[i][1]==Board[i][2]&&Board[i][1]==Board[i][3]) return Board[i][1];  //行
        if(Board[1][i]!=0&&Board[1][i]==Board[2][i]&&Board[1][i]==Board[3][i]) return Board[1][i];  //列
    }
    if(Board[1][1]!=0&&Board[1][1]==Board[2][2]&&Board[1][1]==Board[3][3]) return Board[1][1];      //左斜对角线
    if(Board[1][3]!=0&&Board[1][3]==Board[2][2]&&Board[2][2]==Board[3][1]) return Board[1][3];      //左斜对角线

    //棋盘空
    for(i=1;i<4;i++)
        for(j=1;j<4;j++)
            if(Board[i][j]==0)  //存在空位，说明还没有下完棋
                return 0;
    //最后一种情况，棋盘满了，平局
    return 0x10;
}
```

（6）所有的显示部分

这里出现了一个 bug，一直无法显示，我怀疑是延时函数搞的鬼，尝试删除延时等方式，上专选的时候蹲在充电插座旁边调了很久，没有成功，最终放弃了这个功能（赢方显示流水灯）。

```
//LED灯光
void displayLED(){
    P2=0;sbtLedSel=1; //开启LED显示
    led=0x01;
    j=0;
    for(i=0;i<10;i++){
        if(led==0x80){
            j=!j;
        }
        else if(led==0x01){
            j=!j;
        }
        if(j)                           //向左右移动
            led=led<<1;
        else
            led=led>>1;
        P0=led;
        Delay30ms();
    }
    sbtLedSel=0;
```

展示棋盘，可以看出来本来是想实现加点指示的功能的，后来注释掉了。这里还实现了当前选择位闪烁的功能。

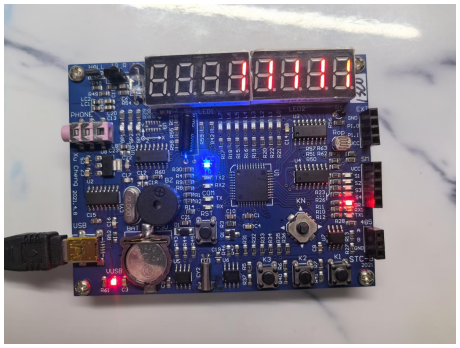
```
void displayChessBoard(){               //展示棋盘
    P0=0;
    blinkIndex=y;                       //设置闪烁
    if(digIndex==y)                     //如果当前位选位和闪烁位一致，说明需要进行闪烁操作
    {
        if(count>20){                   //计满,点亮
            if(count==50) count=0;
        }
        else                            //熄灭（跳过当前位）
            digIndex++;
    }
//  if (digIndex==point && start)               //光标
//      P0 = arrSegSelect[Board[x][digIndex]] | 0x80;
//  else
        P0 = arrSegSelect[Board[x][digIndex]];
    P2=dig[digIndex];                   //位选
}
```

数码管滚动显示，用于游戏开始前和间歇时。

```
403 //数码管滚动显示
404 void displayRoll(unsigned char arr[]){
405     //滚动一次
406     if(count>20){
407         count=0;                                    //分频归零
408         j=arr[0];
409         for(i=0;i<12;i++) arr[i]=arr[i+1];
410         arr[12]=j;
411     }
412     P0=0;
413     P2=digIndex;                                    //位选
414     P0 = arr[digIndex];                             //段选
415 }
```

五、结果展示

（1）优先级调度

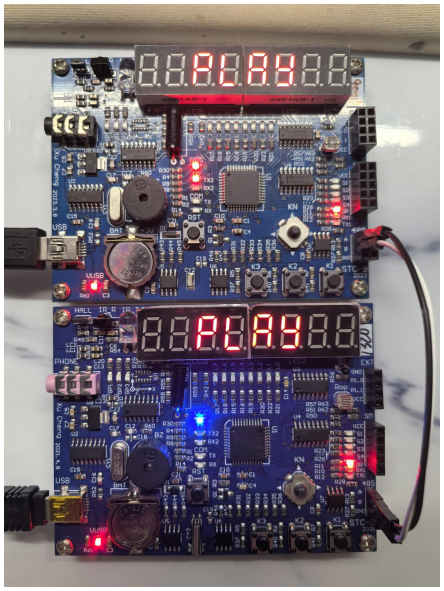


动态过程难以演示，仅作为参考。123 轮流显示，可根据优先级变换顺序。

（2）游戏

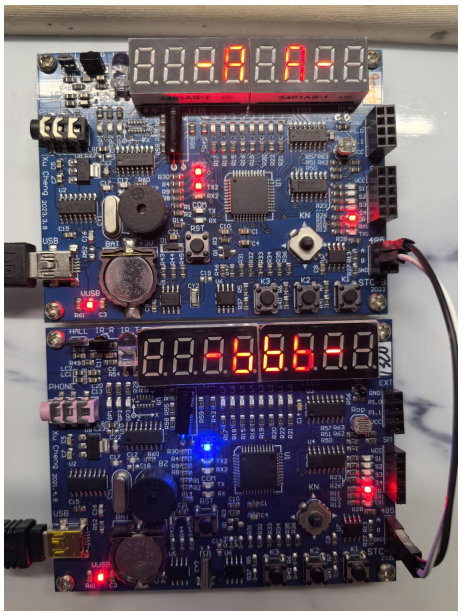
由于 LED 的显示原理，拍照出来会很奇怪，仅作为一个演示，我会标出它的显示内容。

①初始化



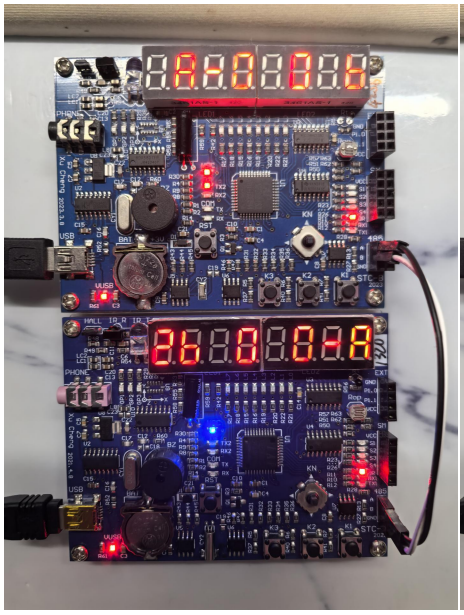
PLAY CHESS，数码管滚动显示。

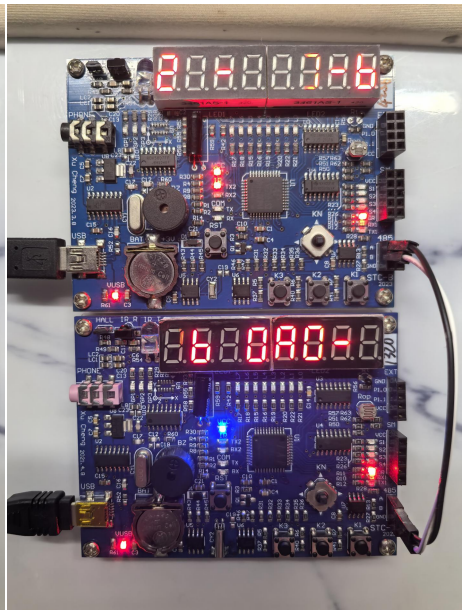
②选择用户



上方 AAA，下方 bbb。

③进入游戏

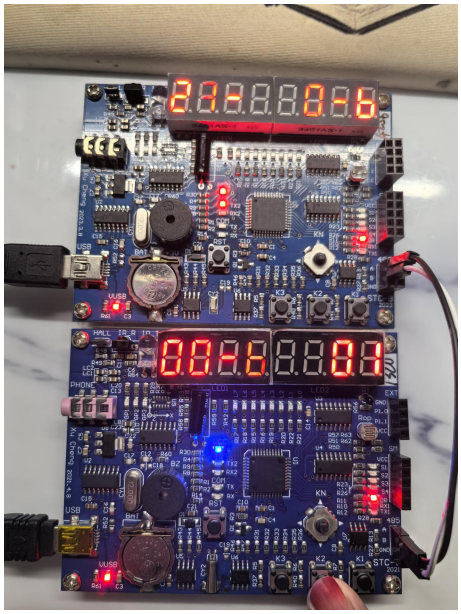




初始化显示，上 2A-000-b，下 2b-000-A。

在正中心下了一个棋子，2A-0A0-b，下 2b-0A0-A。

④游戏结束



摁住下方的 key2 键，下方显示为 00-bA-01，上局 A 获胜，得一分。上方是一句结束后滚动显示完成后，显示初始化的 2A-000-b。

若摁住上方的 key2 键，则显示为 01-Ab-00。

六、实验中遇到的问题与难点

（1）对于优先级的展示需要修改时间片

写的时候忘记时间片这个东西了，调了很久才想起来，展示时发生 LED 闪烁的原因是因为时间片太短了，修改后解决。

（2）奇怪的 bug

总有一些不知道理由的奇奇怪怪的 bug，在先前的报告中我已经展示出部分，比如将延时函数由 5ms 修改成 50ms 就无法正常运行，或者那个对当前 curPlayer 加点，我怀疑也是延时函数的使用引起的，最后也算是取了折中，放弃了一些功能，修改了一些函数。

（3）优化部分

优化真的好难做，感觉之前学长做的系统已经很完善了，连进程隔离和切换的部分都做了四版，实现空间的动态缩放真的太难了，最后放弃对存储空间进行优化，仅对调度方面做了一些改进。

七、实验心得

终于做完了，做了很久，期间一度非常崩溃。实验给了两周的时间，其实做的过程仅不到一周，前面又有小班课又临时有作业之类的任务，一个 DDL 接着一个，心力交瘁，一直没来得及写，DDL 之前的四天每天写这个实验都超过八小时了，验收前一天熬到半夜两点，还好最终还是做完了。

也怀疑过用这么多精力去做这个实验是不是值得，因为投入明显和回报不成正比，实验的内容又和上课的内容，不能说一模一样，只能说 90%都不相关，又没有什么相关的指导，写得非常痛苦，验收完也没有什么特别的成就感，只有一种劫后余生的后怕。

好在是做完了。

做完这个实验对小学期所用的 STC-B 板的认知更深了，也对一个嵌入式操作系统的认知更加清楚，前面写这个 STC-B-OS 的学长真的很厉害，方方面面都考虑得很完全。

对于老师上课时候提到的“由于访问片外存储导致效率并没有上升的问题”我并不是很理解，对于 reentrant 函数来说，只将返回地址放在片内，变量放在片外，那么无论如何调度，都会在片内片外来回访问，唯一能够解释的理由就是遍历访问 event 的栈增加了一些时间，但是我认为优先级调度的主要目的并不是让它更快，而是让事件存在轻重缓急，所以最终还是按照我的想法写了。

或许使用多级队列调度会更好（验收时助教也说他期待看到多级队列调度的方式），当时觉得多级队列调度复杂度可能太高，对于最多只有 8 个的系统可能没有必要，会造成时间上升，再加上没有成功实现，最终还是放弃了，有一点遗憾。