# 基础知识

多线程系统事件的响应也是在中断中完成了，但是事件的处理是在线程中完成的。线程跟中断一样，也具有优先级。

线程： 我们根据功能把整个系统分割成一个个独立的且无法返回的函数。 这个函数我们称之为线程。

定义线程栈：

多线程中，每个线程是独立的，互不干扰的， 所以要为每个线程分配单独的线程栈。 这个栈空间可以是预先定义的全局数组，也可以是动态分配的内存。

设置字节对齐可以提高内存的访问效率。

定义线程控制块：

线程是由系统调用的，系统为了顺利的调度线程， 为每个线程定义了额外的线程控制块。

相当于线程的身份证， 包含线程的堆栈指针， 线程名称，线程参数等等。 对线程的操作可以通过线程控制块来完成。

初始声明：

Struct rt\_thread

{

void \*sp； /\*线程堆栈指针\*/

void \*entry； /\* 入口地址\*/

void \*para; /\*形参\*/

void \*stack\_addr; /\* 栈开始地址\*/

void \*stack\_size; /\* 线程栈大小\*/

rt\_list\_t tlsit; /\* 线程链表节点\*/

};

Typedef stuuct et\_thread \*rt\_thread\_t;

线程创建函数：

rt\_thread\_init(struct rt\_thread \*thread,void(\* entry)(void \*para),void\*para,void \*stack\_start,rt\_uint32\_t statck\_size)

函数的实现：

1. 初始化线程链表节点： rt\_list\_init(&(thread->tlist))

双向链表rt\_list\_t 的定义：

struct rt\_list\_node

{

struct rt\_list\_node \*next; /\*指向后一个节点\*/

struct rt\_list\_node \*prev; /\* 指向前一个节点\*/

}

链表的初始化： 双向的环形链表

链表的初始化就是把节点里的next和prev这两点节点指针指向节点本身。

双向无头循环链表参考代码：

<https://www.cnblogs.com/chengzi123/p/4340471.html>

#include <stdio.h>  
#include <stdlib.h>  
#define OVERFLOW -2  
#define OK 1  
#define ERROR 0  
  
//此双向循环链表无头结点  
typedef int ElemType;  
typedef struct DulNode {  
    ElemType data;  
    struct DulNode \*prior;  
    struct DulNode \*next;  
}DulNode,\*DulLinkList;  
  
DulLinkList InitList\_Dul(DulLinkList L);  
DulLinkList CreateElemP\_Dul(DulLinkList L);  
void PrintList\_Dul(DulLinkList L);  
int GetElemP1\_Dul(DulLinkList L,int i,ElemType \*e);  
DulLinkList ListInsert\_Dul(DulLinkList L,int i,ElemType e);  
int ListLength\_Dul(DulLinkList L);  
DulLinkList GetElemP2\_Dul(DulLinkList L,int i);  
DulLinkList ListInsert\_Dul(DulLinkList L,int i,ElemType e);  
DulLinkList ListDelete\_Dul(DulLinkList L,int i,ElemType \*e);  
int GetLocate\_Dul(DulLinkList L,ElemType e);  
int main()  
{  
    DulLinkList dul;  
    int len,loc;  
    ElemType e;  
    dul = InitList\_Dul(dul);  
    dul = CreateElemP\_Dul(dul);  
    len = ListLength\_Dul(dul);  
    printf("该双向链表的长度为：%d\n",len);  
    PrintList\_Dul(dul);  
    GetElemP1\_Dul(dul,3,&e);//3为元素所在的位序  
    printf("第i个位置的元素为：%d\n",e);  
    printf("向链表中插入一个元素后：");  
    dul = ListInsert\_Dul(dul,1,0);//1为插入元素的位置  
    PrintList\_Dul(dul);  
    printf("从链表删除一个元素后：");  
    dul = ListDelete\_Dul(dul,2,&e);//2为删除元素的位序  
    PrintList\_Dul(dul);  
    printf("删除的元素是：%d\n",e);  
    loc = GetLocate\_Dul(dul,0);//0为元素的值  
    printf("该双向循环链表元素i的位序为:%d\n",loc);  
    return 0;  
}  
  
//初始化一个双向循环链表  
DulLinkList InitList\_Dul(DulLinkList L) {  
    L = (DulLinkList)malloc(sizeof(DulNode));  
    if(L != NULL) {  
        L->next = L->prior = L;  
    } else {  
        exit(OVERFLOW);  
    }  
    return L;  
  
}

//创建一个双向循环链表  
DulLinkList CreateElemP\_Dul(DulLinkList L) {  
    int n,i;  
    DulLinkList p,s;  
    s = L;  
    printf("请输入双向循环链表的元素个数：");  
    scanf("%d",&n);  
    printf("请输入双向循环链表中元素的值：");  
    for(i=0; i<n; i++) {  
        p = (DulLinkList)malloc(sizeof(DulNode));  
        scanf("%d",&p->data);  
        p->prior = s->prior;  
        s->prior->next = p;  
        p->next = s;  
        s->prior = p;  
    }  
    return L;  
}  
  
//输出该双向循环链表  
void PrintList\_Dul(DulLinkList L) {  
    int i;  
    DulLinkList p;  
    p = L;  
    printf("该循环链表中的元素为：");  
    while(p->next != L) {  
        p = p->next;  
        printf("%d ",p->data);  
    }  
    printf("\n");  
}  
  
//返回双向循环链表的长度  
int ListLength\_Dul(DulLinkList L) {  
    DulLinkList p;  
    int i;  
    p = L->next;  
    i = 0;  
    while((p!=L) && (p!=NULL)) {  
        ++i;  
        p = p->next;  
    }  
    return i;  
}  
  
//取得双向循环链表上第i个位置上的元素的值  
int GetElemP1\_Dul(DulLinkList L,int i,ElemType \*e) {  
    DulLinkList p;  
    int j;  
    p = L;  
    j = 0;  
    while(p->next && j<i) {  
        ++j;  
        p = p->next;  
    }  
    if(!p || j>i) return ERROR;  
    \*e = p->data;  
    return OK;  
}  
  
//返回双向循环链表上第i个位置上的指针  
DulLinkList GetElemP2\_Dul(DulLinkList L,int i) {  
    DulLinkList p;  
    int j;  
    p = L;  
    j = 0;  
    while(p->next && j<i) {  
        ++j;  
        p = p->next;  
    }  
    if(!p || j>i) return ERROR;  
    return p;  
}  
  
//向该双向循环链表的第i个位置之前插入元素e  
DulLinkList ListInsert\_Dul(DulLinkList L,int i,ElemType e) {  
    DulLinkList p,s;  
    int m;  
    if(!(p=GetElemP2\_Dul(L,i))) {  
        return ERROR;  
    }  
  
    if(!(s = (DulLinkList)malloc(sizeof(DulNode)))) {  
        return ERROR;  
    }  
    s->data = e;  
    s->prior = p->prior;  
    p->prior->next = s;  
    s->next = p;  
    p->prior = s;  
    return L;  
}  
  
//删除双向循环链表上第i个位置的元素，并返回该元素的值  
DulLinkList ListDelete\_Dul(DulLinkList L,int i,ElemType \*e) {  
    DulLinkList p;  
    if(!(p=GetElemP2\_Dul(L,i))) {  
        return ERROR;  
    }  
    \*e = p->data;  
    p->prior->next = p->next;  
    p->next->prior = p->prior;  
    return L;  
}  
  
//返回双向循环链表上元素e的位序  
int GetLocate\_Dul(DulLinkList L,ElemType e) {  
    DulLinkList p;  
    int j;  
    p = L->next;  
    j = 0;  
    while((p != L) && (p->data!=e)) {  
        j++;  
    }  
    return j;  
}

/\* 初始化线程栈\*/

Rt\_hw\_stack\_init()函数