嵌入式系统概论实验报告

改进的 EDF 调度算法实现

陈自强
141250017
2016-11-17

一、实验目的：

在 uC/OS-II 平台上实现 EDF 的调度器。uC/OS 默认只提供对固定优先级调度的支持，这样虽然很容易实现 RM 调度，但是效率较低，也无法应用于实际情况。而 EDF 算法则支持调度截止期限最短的进程，从而可以实现理论上 100%的 CPU 调用率（当 CPU 不出现过载的情况下）。本实验拟实现基本的 EDF 算法，并在此基础上探索改进的 EDF 算法。

二、实验条件

本实验代码修改自μCOSII_VC 移植版， 使用 Microsoft Visual Studio 2015、Sublime 进行开发，VC 版本为 V140，平台版本为 Windows SDK 10.0.14393.0

三、基本 EDF 调度实现：

3.1 思路

在分析了源码之后，得出如下结论：

- 核心文件为 OS_Core.c,包含调度算法，Timetick 中断处理
- 默认调度算法为 OS_SchedNew
- 每一次 ISR 的执行代码流如下

```
if (suspended == DEF_TRUE) {
    OSIntEnter();
    OSTimeTick();
    OSIntExit();  已用时间 <= 1ms
    OSIntCurTaskResume();
}
```

- 在 OS_Start(), OS_IntExit() 以及 OS_Sched()均会调用调度算法

分析完源码之后，实验的核心也就非常明确了：实现一个新的调度算法，用于替代默认的 OS_SchedNew()方法。

为了实现这一目的，我们首先需要做一些准备工作：

- 生成几个用户的 Task 来模拟进程用于调度， 只需要在 App.c 仿照默认的 AppTaskStart 建立新的 task 就够了
- 建立数据结构来存储 task 的起始时间、消耗时间、截止时间。为此，我们需要在 ucos_ii.h 文件中找到 TCB 的结构定义，增加属性。但是可以发现 Tcb 默认有一个 OSTCBExtPtr 指针指向用户自定义的结构，可以用来扩展，为了减少对原有代码的修改，于是我们新建一个结构在该头文件中（importance 为改进算法使用，此处暂不涉及），并在初始化 task 时将指针传入即可：

```
/*
*********************************************************************************************************
*                                          Qiang implement EDF Support
*********************************************************************************************************
*/
typedef struct edf_task_data{
    INT32U c;       // execute c units in every p units of time
    INT32U p;
    INT32U compTime;    // the task remain to comsume in a period
    INT32U ddl;         // the deadline of a task
    INT32U start;
    INT32U end;
    INT32U importance;
} EDF_TASK_DATA;
```

- 有了这些支持之后，只要修改算法并输出即可。输出使用默认的 Printf，需要包含头文件 “stdio.h”
- 其他修改如函数预声明、数据定义、函数栈定义等此处略

3.2 核心算法分析

调度算法：

- 遍历 TCBList 中的每一个 Task，找出截止期限最早的 task 并调度它。
- 如果所有的 task 都处于 delay 状态，调用默认的空任务。
- 由于默认的 Idle task 没有 OSTCBEXTPTR，因此我对这类型的赋值了一个默认值。
- 当发生任务切换时，打印结果，justZero 位于 timetick()中，指示 comptime 是否在最近一个 ISR 中变为 0，根据这个打印 complete 和 Preempt。

```
static void OS_mySched(void)
{
    OS_TCB* task_current;
    OS_TCB* nextToDone;
    int tempLatestDDL = 1000000;
    int ddl;
    int isAllDelay = 1;

    task_current = OSTCBList;
    nextToDone = OSTCBPrioTbl[OS_TASK_IDLE_PRIO];
    OSPrioHighRdy = OS_TASK_IDLE_PRIO;
    while( task_current->OSTCBPrio != OS_TASK_IDLE_PRIO ) {

        if (task_current->OSTCBExtPtr == 0) {
            task_current->OSTCBExtPtr = &idleEdf;
        }


        if (task_current->OSTCBDly == 0 && ((EDF_TASK_DATA *)task_current->OSTCBExtPtr)->compTime>0)
        {
            ddl = ((EDF_TASK_DATA *)task_current->OSTCBExtPtr)->ddl;
            if (ddl < tempLatestDDL)
            {
                tempLatestDDL = ddl;
                nextToDone = task_current;
            }
            isAllDelay = 0;
        }
        task_current = task_current->OSTCBNext;
    }

    int nextTaskID = nextToDone->OSTCBId;
    if (OSTCBCur->OSTCBId != nextTaskID)
    {
        EDF_TASK_DATA * tempPoint = (EDF_TASK_DATA *)OSTCBCur->OSTCBExtPtr;
        if (justZero == 1)
        {
            printf("%d\tComplete\t%d\t%d\n",OSTimeGet()/100,OSTCBCur->OSTCBId,nextTaskID);
        }else{
            printf("%d\tPreempted\t%d\t%d\n",OSTimeGet()/100,OSTCBCur->OSTCBId,nextTaskID);
        }
    }
    OSPrioHighRdy=nextToDone->OSTCBPrio;
    if(isAllDelay==1){
        OSPrioHighRdy=OS_TASK_IDLE_PRIO;
    }
}
```

DDL、剩余计算时间（CompTime）等的维护：

```
justZero = 0;
EDF_TASK_DATA* tempPoint = (EDF_TASK_DATA*)OSTCBCur->OSTCBExtPtr;
tempPoint->compTime--;
if (tempPoint->compTime == 0)
{
    justZero = 1;
    tempPoint->end = OSTimeGet(); // end time
    int t = tempPoint->p;
    int toDelay =  t - ( tempPoint->end - tempPoint->start);
    if (toDelay <= 0)
    {


        toDelay = 0;
    }
    tempPoint->start = tempPoint->start + t; // next start time
    //printf("  ****%d    ",toDelay);
    //printf("  ****%d    ", tempPoint->ddl);
    tempPoint->ddl = tempPoint->ddl + tempPoint->p;
    tempPoint->compTime = tempPoint->c; // reset the counter (c ticks for computation)
    OSTCBCur->OSTCBDly = toDelay;
}
```

- 此处位于 TimeTick 代码的最后部，用于维护 DDL、compTime（当前任务剩余计算时间）以及 JustZero（指示是 complete 还是 Preempt，见调度算法）变量
- 先将当前进行的 task 的剩余计算时间（compTime） -1

- 在测试时如果系统过载（即不可能所有任务都在 DDL 之前完成，正常的测试用例不会出现这种情况）toDelay 可能会出现为负数的情况，为了避免这个情况，当 todelay<0 时，置为 0

3.3 其他修改

- 搜索并替换原有的 sched_new()方法为新的调度算法
- 声明一个默认的 idleedf，作为 idletask 的 OSTCBExtPtr
- 其他变量声明、测试输出等不赘述

3.4 测试

测试用例 1：
输入：（各参数含义见图片内注释，默认 1s 100ticks，故 100 表示 1s）

```
// c, p, compTime, ddl, start, end, importance
EDF_TASK_DATA taskData[] = {
    {100,300,100,300,0,0},          // task1
    {300,500,300,500,0,0},          // task2
```

```
OSTick     created, Thread ID  7548
Task[ 63] created, Thread ID  6560
Task[ 62] created, Thread ID  1868
Task[ 61] created, Thread ID  4668
Task[  1] created, Thread ID  6416
Task[  2] created, Thread ID  7264
Task[  2] '?' Running
1        Complete        1        2
4        Complete        2        1
5        Complete        1        2
6        Preempted       2        1
7        Complete        1        2
9        Complete        2        1
10       Complete        1        2
13       Complete        2        1
14       Complete        1        65535
Task[ 63] 'uC/OS-II Idle' Running
15       Preempted       65535    1
16       Complete        1        2
19       Complete        2        1
20       Complete        1        2
21       Preempted       2        1
22       Complete        1        2
```

测试用例 2：

输入:

```
// c, p, compTime, ddl, start, end, importance
EDF_TASK_DATA taskData[] = {
    {100,300,100,300,0,0},          // task1
    {300,500,300,500,0,0},          // task2
{ 100,400,100,400,0,0 , 100},        // task3
{ 200,500,200,500,0,0 , 200}         // task4
,
{ 200,1000,200,1000,0,0 , 300}       //task5
};
```

输出：

```
OSTick    created, Thread ID  5476
Task[ 63] created, Thread ID  3768
Task[ 62] created, Thread ID  4320
Task[ 61] created, Thread ID  9024
Task[  3] created, Thread ID  5668
Task[  4] created, Thread ID  6028
Task[  5] created, Thread ID  3648
Task[  5] '?' Running
1       Complete        3       4
3       Complete        4       5
4       Preempted       5       3
5       Complete        3       5
6       Complete        5       4
8       Complete        4       3
9       Complete        3       65535
Task[ 63] 'uC/OS-II Idle' Running
10      Preempted       65535   4
12      Complete        4       3
13      Complete        3       5
15      Complete        5       4
17      Complete        4       3
18      Complete        3       65535
20      Preempted       65535   3
21      Complete        3       4
23      Complete        4       5
24      Preempted       5       3
25      Complete        3       5
26      Complete        5       4
```

四、改进的 EDF 算法实现

4.1 改进思路

默认的 EDF 进行调度的唯一标准是任务的截止期限（DDL），根据截止期的大小映射出对应的报文优先级，即优先级大小是截止期限的一元函数。

这种方法的好处十分明显：操作简单且容易实现，系统实现简单。同时缺点也十分明显，任务的优先级并不能简单的用一个 ddl 来判断，例如对于 ddl 均很大的两个 task（如它们的 DDL 分别为 3000，3100，此时 DDL 不应该是调度优先考虑的因素），

此时他们的 ddl 本身意义不大，task 本身还有其他因素应该纳入考虑，如重要程度、紧急程度、剩余时间、请求出现的顺序等等。

因此本算法在 EDF 算法的基础上进行优化，考虑其他参数，使得调度算法更加的合理，引入相对价值作为优先级的衡量标准，其定义为：

$$相对价值 = 重要程度/(剩余时间 * 到达顺序 * 截止期限)$$

实现该算法后，系统可以允许用户调整任务的先后次序，给某个 task 增加其重要程度可以使得其优先执行。

4.2 改进方法

```
static void OS_mySched(void)
{
    OS_TCB* task_current;
    OS_TCB* nextToDone;
    double highestPriority = 0;
    double tempPriority = 0;
    int isAllDelay = 1;
    int order = 0;                // order defers the coming order of the user tasks, system idle tasks are exclusive
    task_current = OSTCBList;
    nextToDone = OSTCBPrioTbl[OS_TASK_IDLE_PRIO];
    OSPrioHighRdy = OS_TASK_IDLE_PRIO;
    while( task_current->OSTCBPrio != OS_TASK_IDLE_PRIO ) {

        if (task_current->OSTCBExtPtr == 0) {
            task_current->OSTCBExtPtr = &idleEdf;
        }


        if (task_current->OSTCBDly == 0 && ((EDF_TASK_DATA *)task_current->OSTCBExtPtr)->compTime>0)
        {
            EDF_TASK_DATA* tempPoint = (EDF_TASK_DATA *)task_current->OSTCBExtPtr;
            if (tempPoint->compTime > 10000)       //only care about user tasks exclude the idle tasks
            {
                tempPriority = tempPoint->importance / (double)(tempPoint->compTime * tempPoint->ddl*1000);
            }
            else
            {
                order++;
                tempPriority = tempPoint->importance / (double)(tempPoint->compTime * tempPoint->ddl*order);
            }




            if (tempPriority > highestPriority)
            {
                highestPriority = tempPriority;
                nextToDone = task_current;
            }
            isAllDelay = 0;
        }
        task_current = task_current->OSTCBNext;
    }

    int nextTaskID = nextToDone->OSTCBId;
    if (OSTCBCur->OSTCBId != nextTaskID)
    {
        EDF_TASK_DATA * tempPoint = (EDF_TASK_DATA *)OSTCBCur->OSTCBExtPtr;
        if (justZero == 1)
        {
            printf("%d\tComplete\t%d\t%d\n",OSTimeGet()/100,OSTCBCur->OSTCBId,nextTaskID);
            printf("tempid %d  %f\n", nextTaskID, highestPriority);
        }else{
            printf("%d\tPreempted\t%d\t%d\n",OSTimeGet()/100,OSTCBCur->OSTCBId,nextTaskID);
            printf("tempid %d  %f\n", nextTaskID, highestPriority);

        }
    }
    OSPrioHighRdy=nextToDone->OSTCBPrio;
    if(isAllDelay==1){
        OSPrioHighRdy=OS_TASK_IDLE_PRIO;
    }
}
```

1. 在 EDF_TASK_DATA 中增加一个属性 importance

```
typedef struct edf_task_data{
    INT32U c;       // execute c units in every p units of time
    INT32U p;
    INT32U compTime;    // the task remain to comsume in a period
    INT32U ddl;         // the deadline of a task
    INT32U start;
    INT32U end;
    INT32U importance;
} EDF_TASK_DATA;
```

2. 遍历 TCBList 中 task 出现的顺序获得 task 到达进程的 order
3. 计算新的优先级，并找出优先级最高的 task 调用即可

PS：系统 idle 进程 order 调为 1000 以降低其优先级

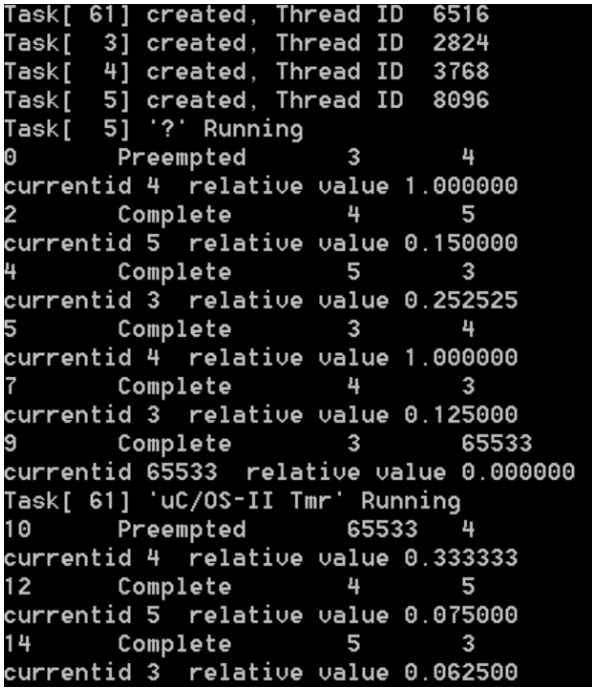
4.3 测试

输入如下：

```
// c, p, compTime, ddl, start, end, importance
EDF_TASK_DATA taskData[] = {
    {100, 300, 100, 300, 0, 0 , 10000},          // task1
    {300, 500, 300, 500, 0, 0 , 20000},
{ 100, 400, 100, 400, 0, 0 , 10000},
{ 200, 500, 200, 500, 0, 0 , 200000}
,
{ 200, 1000, 200, 1000, 0, 0 , 30000}
};
```

输出如下（其中 relative Value 是相对价值）：

可以看出在其他条件不变的情况下增加了 task4 的重要性，使得其一开始抢占了 DDL 比它早的 task3（这里其他参数同基本 EDF 第二个用例，可参考），并且可以看见随着 DDL 增加，每一个 task 的 relative Value 都逐渐减少。



本改进思路参考了：

《改进型 EDF 调度算法的研究与实现>》--萧伟，冯治宝，应启戛

文章编号：1000—3428(2009)18—0231—03