嵌入式实验报告

μcosII 的 edf 调度算法

张文玘

2016-11-25

1. 实验环境

RTOS	μCOSII_VC 移植版
VC 平台工具集	Visual Studio 2015

2. 实验目的

在 VC 移植版本的μCOUII 上实现 EDF 调度。
μCOSII 目前只提供对静态的固定优先级调度支持，很容易实现 RMS 调度，但是不支持动态优先级，而 EDF 调度是一种理论上可以实现 100%CPU 利用率的调度算法

3. EDF 调度实现

1） 思路与方法
   a) 定义用于实现 EDF 算法的数据结构，由于 EDF 调度采用最小截止时限优先的调度算法，需要在 ucos_ii.h 文件中新建一个用于存储 edf 所需数据的数据结构 edf_data，这个数据结构可以通过原代码中已定义的 os_tcb 数据结构中的 OSTCBExtPtr 来链接到 tcb 数据结构中，这个指针就是 ucos_ii 设计用来添加 tcb 扩展数据的指针。edf_data 数据详见下文
   b) 程序每次调度哪个进程，是由 os_core.c 中的 OS_Sched 函数实现的，它根据当前 tcb 列表中最高优先级的任务进行调度，所以我们要实现的就是在它决定所调用的 tcb 之前找到在 edf 算法中优先级最高的任务（截止时间距离现在最近的任务），并将 OSPrioHighRdy 改掉，将 OS_SchedNew()函数替换为 OS_SchedEDF()用于实现找到截止时间距离现在最近的任务。
   c) 新增 getEDFNextId()函数，用于找到下一个要执行任务的 id
   d) 原程序中有一个 OSTimeTick 的函数，它是一个时钟周期调用一次的函数，我在这里判断任务是完成还是抢占，在这里要给 edf_data 所需剩余的完成时间减 1，然后判断这个任务是否已经完成，如果已经执行完了，要对任务的截止时间等数据进行更新，以便之后的调用，并输出当前任务执行完成的信息以及下一个要执行的任务的 id；如果这个任务还没有执行完毕，要判断下一个要执行的任务是不是当前任务，如果不是的话要进行抢占
   e) 一个小改动是要在 getEDFNextID()和 OS_SchedEDF()函数中遍历任务序列时判断其扩展指针是否为空，避免 edf 为空的情况。
   f) 在 app.c 中进行任务初始化，统一将进入系统的时间强制设为 1
   g) 在 os_cfg.h 中修改 OS_TICKS_PER_SEC 变量，将其设为 1，即 1 秒钟 1 个 tick
2） 核心数据结构
   a) Edf_data 的结构体定义（ucosii.h 中）

```
typedef struct edf_data {
    INT32U c_value;   //p_value代表周期长度，c_value代表执行时间，单位都为tick
    INT32U p_value;
    INT32U comp_time;   //在当前周期中此任务还需消费的tick数
    INT32U ddl;             //任务的deadline
    INT32U start;        //初始化为此任务进入系统的时间
    INT32U end;
}EDF_DATA;
```

Start 都初始化为 1

3） 核心算法

a） 调度函数 OS_SchedEDF()

```
static void OS_SchedEDF(void) {
    OS_TCB* p_current;
    OS_TCB* edf_ptcb;
    int temp_earliest_deadline = 1000000;
    int temp_deadline = 0;
    int isAllDelay = 1;

    p_current = OSTCBList;

    edf_ptcb = OSTCBPrioTbl[OS_TASK_IDLE_PRIO];
    //如果什么都不做的话应该指向空闲任务（OSTCBPrioTbl[OS_TASK_IDLE_PRIO]即为系统中的空闲任务）
    OSPrioHighRdy = OS_TASK_IDLE_PRIO;
// APP_TRACE("\n In os_schedEDF, curent id:%d, delay:%d", OSTCBCur->OSTCBId, OSTCBCur->OSTCBDly);
// APP_TRACE(", OS_TASK_IDLE_PRIO: %d, id: %d", OS_TASK_IDLE_PRIO, OSTCBPrioTbl[OS_TASK_IDLE_PRIO]->OSTCBId);
    while ((p_current->OSTCBPrio != OS_TASK_IDLE_PRIO)&&((p_current->OSTCBExtPtr)!=0x00000000)){
//      APP_TRACE("\nid:%d, comptime:%d, delay: %d,", p_current->OSTCBId, ((EDF_DATA*)p_current->OSTCBExtPtr)->comp_time, p_current->OSTCBDly);
//      APP_TRACE("  ddl:%d, prio:%d", ((EDF_DATA*)p_current->OSTCBExtPtr)->ddl,((EDF_DATA*)p_current->OSTCBPrio));
        if (p_current->OSTCBDly == 0 && ((EDF_DATA*)p_current->OSTCBExtPtr)->comp_time > 0) {
            temp_deadline = ((EDF_DATA*)p_current->OSTCBExtPtr)->ddl;
            if (temp_deadline < temp_earliest_deadline) {
                temp_earliest_deadline = temp_deadline;
                edf_ptcb = p_current;
            }
            isAllDelay = 0;
        }
        p_current = p_current->OSTCBNext;
    }

    if (isAllDelay == 1) {
        //如果当前任务即为优先级最高的任务,则应将最高优先级的任务指向空闲任务，由于前面初始化时edf_ptcb指向的就是空闲任务，所以不需要再次赋值
        //edf_ptcb=OSTCBPrioTbl[OS_TASK_IDLE_PRIO];
    }

    OSPrioHighRdy = edf_ptcb->OSTCBPrio;


}
```

b） 获取下一个任务的 id, getEDFNextID()

```
static INT32U getEDFNextID(void) {
    OS_TCB* p_current;
    OS_TCB* edf_ptcb;
    int temp_earliest_deadline = 1000000;
    int temp_deadline = 0;
    int isAllDelay = 1;
    INT32U nextId = 0;

    p_current = OSTCBList;
    //
    edf_ptcb = OSTCBPrioTbl[OS_TASK_IDLE_PRIO];//如果什么都不做的话应该指向空闲任务 （ OSTCBPrioTbl[OS_TASK_IDLE_PRIO]即为系统中的空闲任务 ）
    OSPrioHighRdy = OS_TASK_IDLE_PRIO;
    //APP_TRACE("\n In os_schedEDF, curent id:%d, delay:%d", OSTCBCur->OSTCBId, OSTCBCur->OSTCBDly);

    while ((p_current->OSTCBPrio != OS_TASK_IDLE_PRIO)&&(p_current->OSTCBExtPtr!=0x00000000)) {
        //APP_TRACE
        //APP_TRACE
        if (p_current->OSTCBDly == 0 && ((EDF_DATA*)p_current->OSTCBExtPtr)->comp_time > 0) {
            temp_deadline = ((EDF_DATA*)p_current->OSTCBExtPtr)->ddl;
            if (temp_deadline < temp_earliest_deadline) {
                temp_earliest_deadline = temp_deadline;
                edf_ptcb = p_current;
            }
            isAllDelay = 0;
        }
        p_current = p_current->OSTCBNext;
    }


    if (isAllDelay == 1) {
        //如果当前任务即为优先级最高的任务,则应将最高优先级的任务指向空闲任务，由于前面初始化时edf_ptcb指向的就是空闲任务，所以不需要再次赋值
        //edf_ptcb=OSTCBPrioTbl[OS_TASK_IDLE_PRIO];
    }

    nextId = edf_ptcb->OSTCBId;
    return nextId;

}
```

c) 对截止时间和执行时间的更新，以及输出信息的实现（位于 ostimetick 函数中）

```
    ((EDF_DATA*)OSTCBCur->OSTCBExtPtr)->comp_time--;//当前任务又执行了一个周期，所需要的执行时间减1

    if (((EDF_DATA*)OSTCBCur->OSTCBExtPtr)->comp_time == 0) {//若当前任务执行完了,更新当前任务TCB的各项指标
        ((EDF_DATA*)OSTCBCur->OSTCBExtPtr)->ddl = ((EDF_DATA*)OSTCBCur->OSTCBExtPtr)->ddl + ((EDF_DATA*)OSTCBCur->OSTCBExtPtr)->p_value;
        ((EDF_DATA*)OSTCBCur->OSTCBExtPtr)->comp_time = ((EDF_DATA*)OSTCBCur->OSTCBExtPtr)->c_value;
        ((EDF_DATA*)OSTCBCur->OSTCBExtPtr)->end = OSTimeGet();
        OSTCBCur->OSTCBDly = ((EDF_DATA*)OSTCBCur->OSTCBExtPtr)->p_value - (((EDF_DATA*)OSTCBCur->OSTCBExtPtr)->end - ((EDF_DATA*)OSTCBCur->OSTCBExtPtr)->start);


        ((EDF_DATA*)OSTCBCur->OSTCBExtPtr)->start = ((EDF_DATA*)OSTCBCur->OSTCBExtPtr)->start + ((EDF_DATA*)OSTCBCur->OSTCBExtPtr)->p_value;

        //调用输出函数输出此时的TCB运行信息
        APP_TRACE("\n%d\tCompleted\t%d\t%d", OSTimeGet(), OSTCBCur->OSTCBId, getEDFNextID());
    }
    else {
        if (OSTCBCur->OSTCBId != getEDFNextID()) {
            APP_TRACE("\n%d\t Preempt \t%d\t%d", OSTimeGet(), OSTCBCur->OSTCBId, getEDFNextID());
        }
    }

        }//判断此时kernel正处于运行状态的if右括号
    }
```

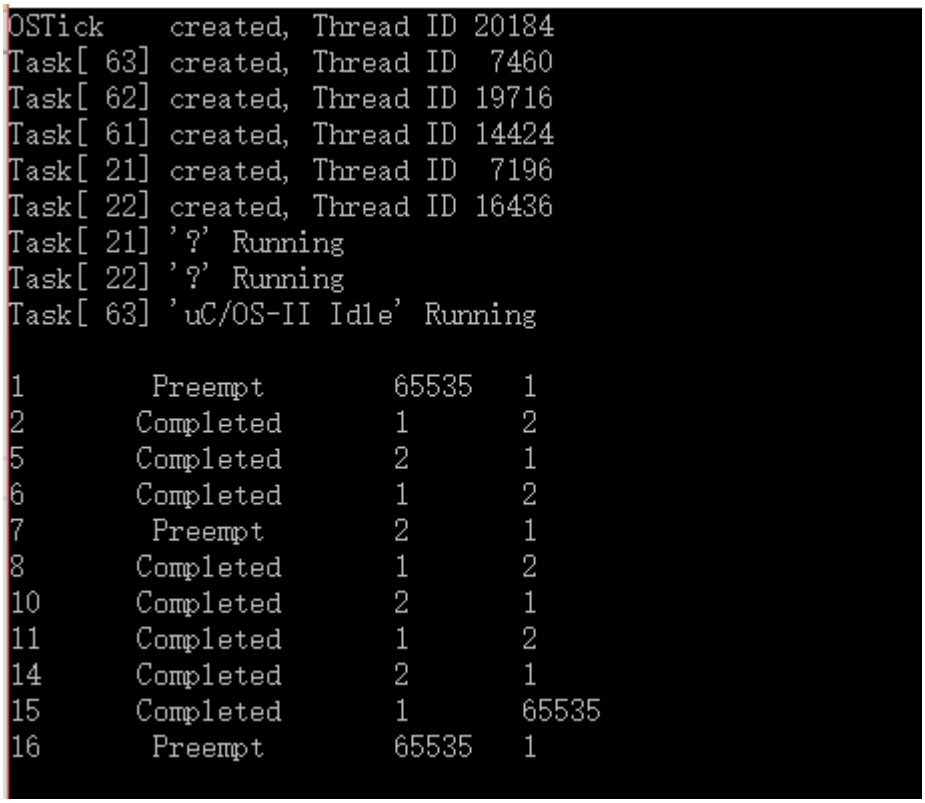
其中的 delay 值代表函数在 end 到周期结束这段时间内不能再次调用此进程

4） 测试用例

```
/*---   edf_datas ---*/
// EDF_DATA{c_value,p_value,comp_time,ddl,start,end}
EDF_DATA edf_datas[] =
{
    { 1,3,1,4,1,1 },
    { 3,5,3,6,1,1 },


};
```

输出


OSTick created, Thread ID 20184
Task[63] created, Thread ID 7460
Task[62] created, Thread ID 19716
Task[61] created, Thread ID 14424
Task[21] created, Thread ID 7196
Task[22] created, Thread ID 16436
Task[21] '?' Running
Task[22] '?' Running
Task[63] 'uC/OS-II Idle' Running
1 Preempt 65535 1
2 Completed 1 2
5 Completed 2 1
6 Completed 1 2
7 Preempt 2 1
8 Completed 1 2
10 Completed 2 1
11 Completed 1 2
14 Completed 2 1
15 Completed 1 65535
16 Preempt 65535 1