

EDF调度实验报告

131130137—董本超



2016-11-3

NJU-SE

目录

[1 实验环境 2](#_Toc466068499)

[2 实验目的 2](#_Toc466068500)

[3 基本EDF的调度实现 2](#_Toc466068501)

[3.1 思路和方法 2](#_Toc466068502)

[3.2 核心数据结构 2](#_Toc466068503)

[3.3 核心算法 3](#_Toc466068504)

[3.3.1 有序任务链表的维护 3](#_Toc466068505)

[3.3.2 调度函数 3](#_Toc466068506)

[3.3.3 对截止时限的实现 4](#_Toc466068507)

[3.4测试用例 4](#_Toc466068508)

[4 改进的EDF调度 5](#_Toc466068509)

[4.1 改进思路 5](#_Toc466068510)

[4.2 实现方法 7](#_Toc466068511)

[4.3 测试用例 9](#_Toc466068512)

[5 参考资料 10](#_Toc466068513)

# 1 实验环境

本次实验使用的RTOS是ucos\_ii的windows移植版，ucos\_ii版本号为2.92.11，使用visual studio 2013进行开发。

# 2 实验目的

修改移植版ucos\_ii的代码，为其添加EDF调度。ucos\_ii原本是使用基于固定优先级的RMS调度，而EDF是基于截至时限的一种可抢占式调度，拥有更高的CPU利用率。希望通过本次实验加深对这两种调度算法的理解。

# 3 基本EDF的调度实现

## 3.1 思路和方法

首先需要定义EDF调度所需要的数据结构。因为EDF调度需要知道每个任务的周期以及截至实现等信息，所以需要在ucos\_ii本身的OS\_TCB中添加这部分的数据。而正好原本的OS\_TCB中就预留了一个(void \*)类型的指针—OSTCBExtPtr，用来指向需要扩展的数据结构，因此可以使用这个来扩充我们需要的数据。

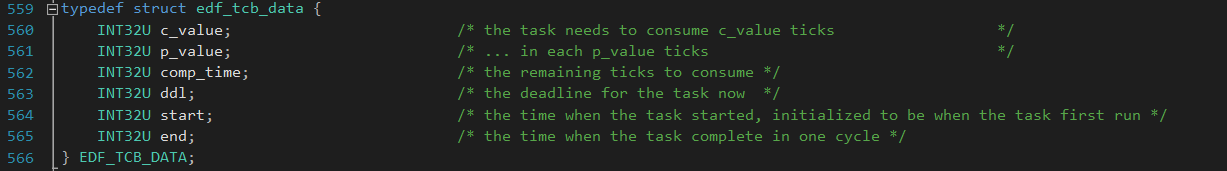
添加一个OS\_TASK\_EDF\_EN的预定义宏，当它不为0时则开启EDF调度，否则还使用原本的RMS调度，这样可以方便切换。

添加一个OS\_TaskIsrt (OS\_TCB \*ptcb)方法，使用原本程序中的OSTCBList数组，将其改为一个有序的链表，其中的OS\_TCB按照其截限时间排序，OS\_TashIsrt方法就负责将执行完等待下一个周期的任务或者刚刚创建的任务插入这个有序链表中。

将原本的代码中使用OS\_SchedNew()方法的地方换成OS\_SchedEDF()，OS\_SchedEDF()方法从已经排好序的OSTCBList中选择最前的非Dely状态的task，并将其优先级赋值给OSPrioHighRdy，然后OS\_Sched进行任务切换。

## 3.2 核心数据结构

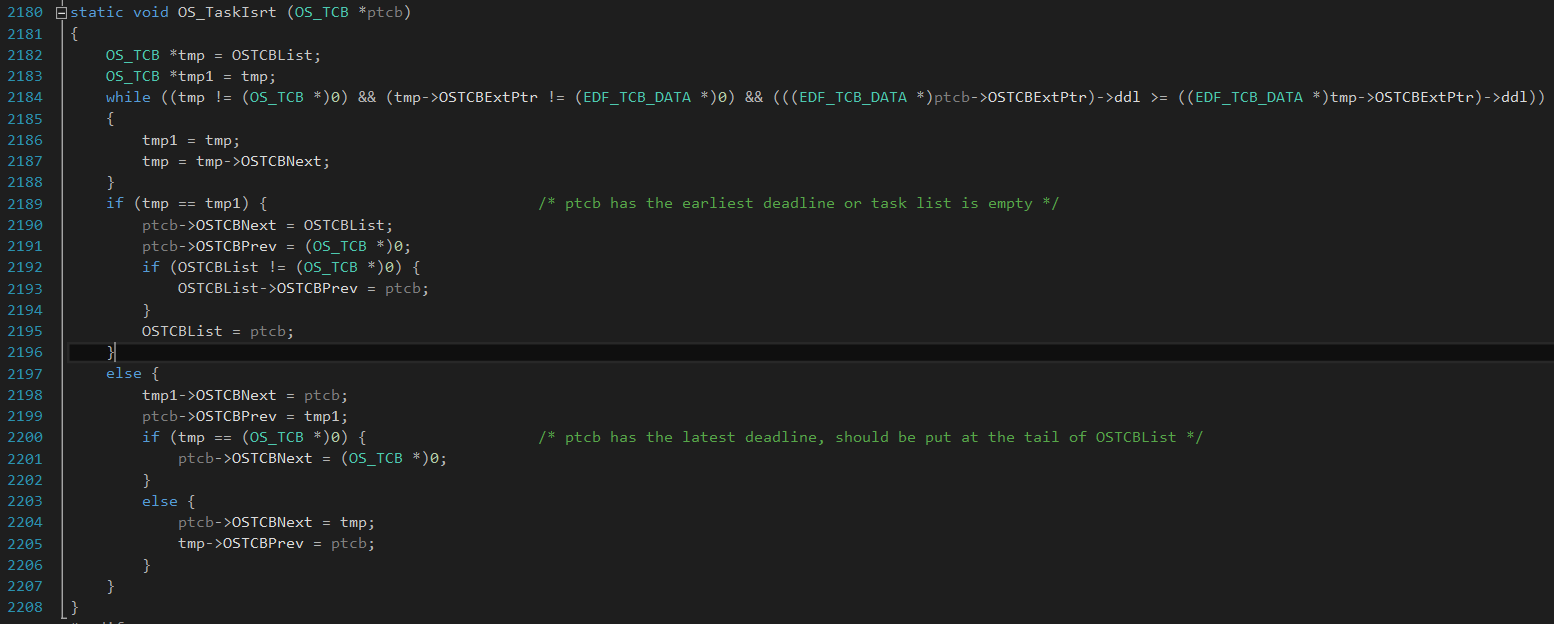
在ucosii.h中定义edf调度所需数据的结构体EDF\_TCB\_DATA：



其中c\_value是任务每个周期中需要运行的tick数，p\_value是任务的周期的tick数，comp\_time是当前周期中剩余需要消耗的tick数，ddl是当前周期的截限时间，start和end是用来辅助计算任务需要delay的tick数量的辅助变量。

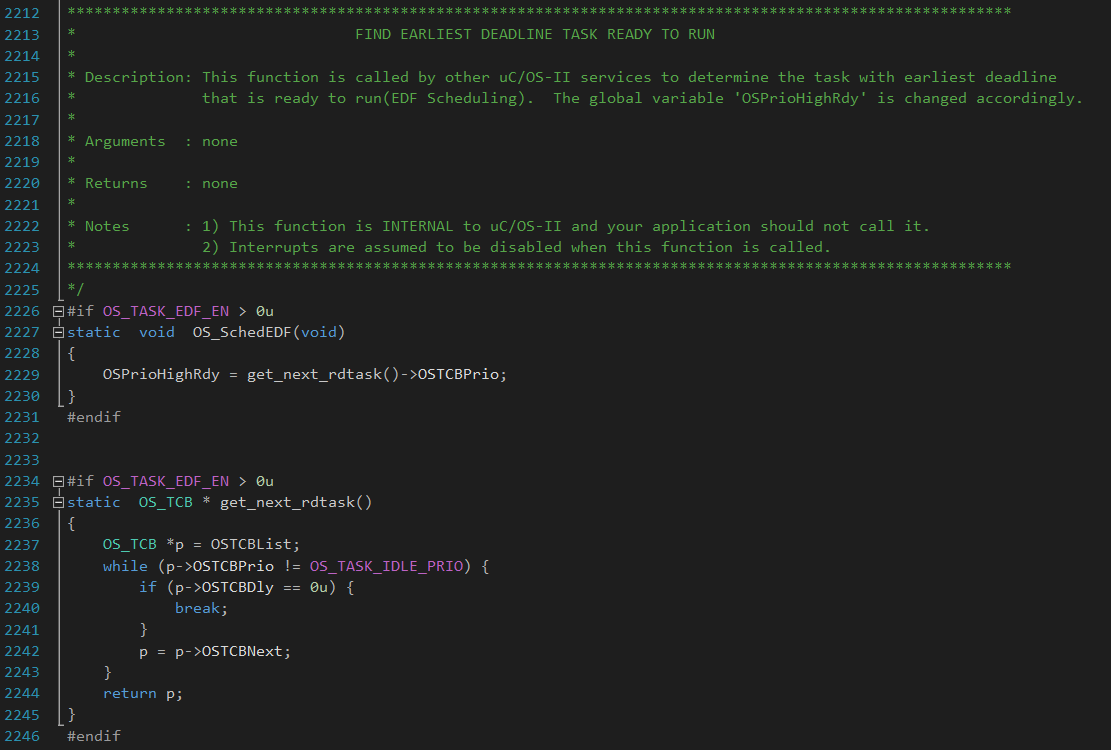
## 3.3 核心算法

### 3.3.1 有序任务链表的维护



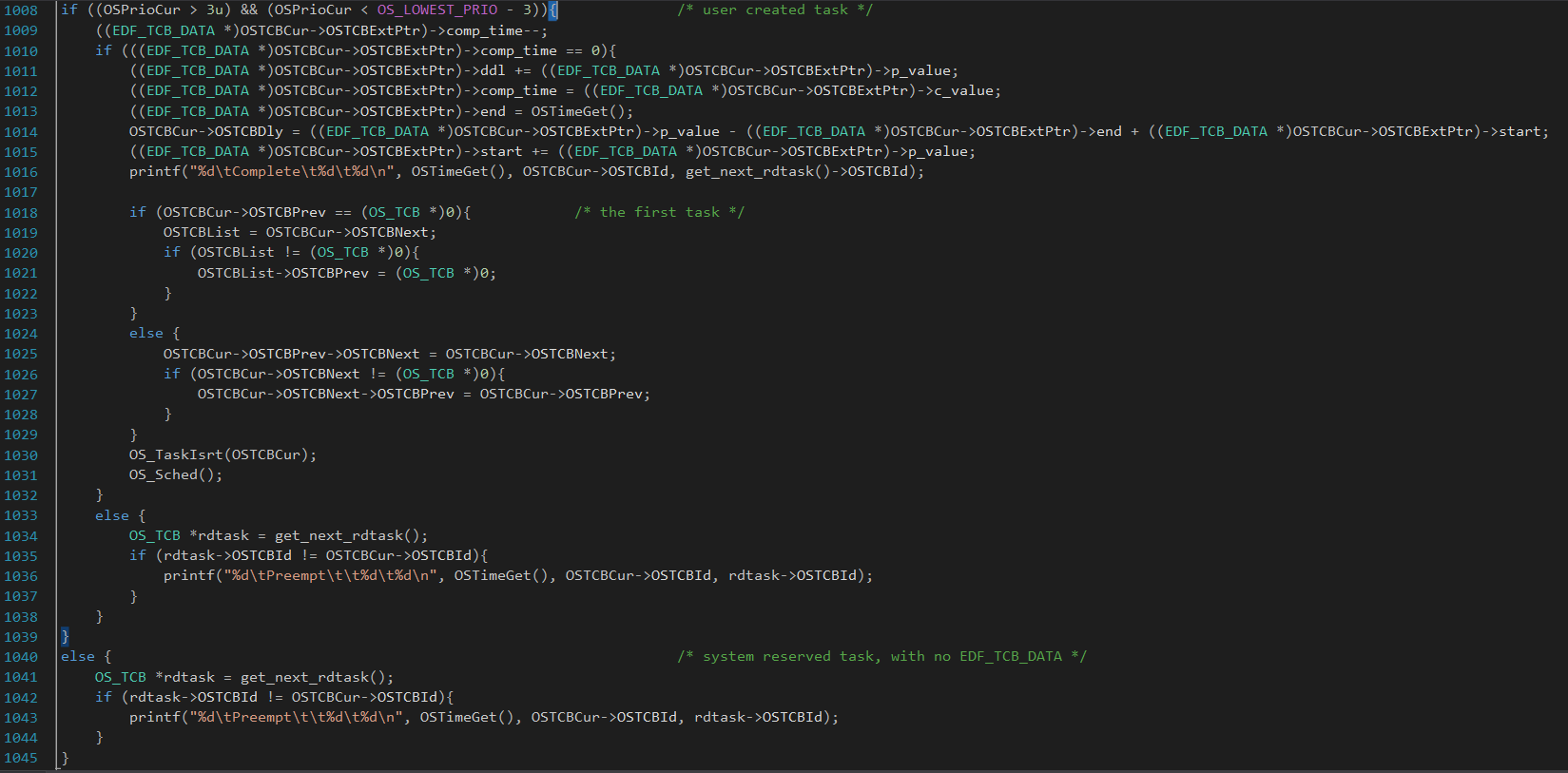
如上图，程序使用OS\_TaskIsrt(OS\_TCB \*ptcb)方法把任务指针插入OSTCBList中，并保持该链表始终有序：截限时间最早的任务放在链表前面，链表最后放idle task。该算法具有O(n)的时间复杂度。

### 3.3.2 调度函数



如上图，调度程序主要逻辑由两个函数组成，其中get\_next\_rdtask()方法从OSTCBList中选择最靠前的OSTCBDly == 0u的任务，即当前周期中事情还没有完成的任务，如果没有则表示所有用户任务当前都处于等待下个周期开始的状态，此时选择idle task执行。选好任务后，只需要将该任务的优先级赋值给OSPrioHighRdy即可，系统的OS\_Sched函数会完成接下来的切换操作。

### 3.3.3 对截止时限的实现



如上图，这段代码插入在os\_core.c中OSTimeTick()方法的后面，在每个时钟周期将等待任务的OSTCBDly减一后，需要额外做的工作如下：

* 将当前任务的comp\_time减一
* 判断当前任务是否是否已经完成，完成则将任务的截限时间向后推延p\_value个tick，并将任务delay响应的tick
* 如果任务没有完成，则调用get\_next\_rdtask()找到截限时间最近的就绪任务，如果不是当前任务则发生切换，否则继续执行。

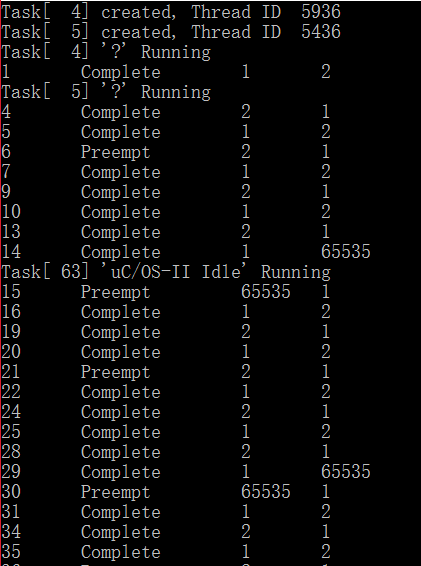
## 3.4测试用例

使用老师ppt上提供的两个任务进行测试：

第一个任务c\_value = 1, p\_value = 3;

第二个任务 c\_value = 3, p\_value = 5.

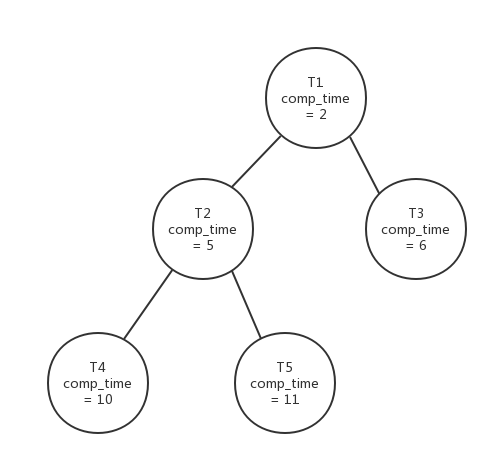
运行截图如下（因为优先级0-3的任务为ucosii保留的任务，所以这里任务1优先级为4，任务2优先级为5，所以截图中出现了”Task[ 4] created”等输出）：



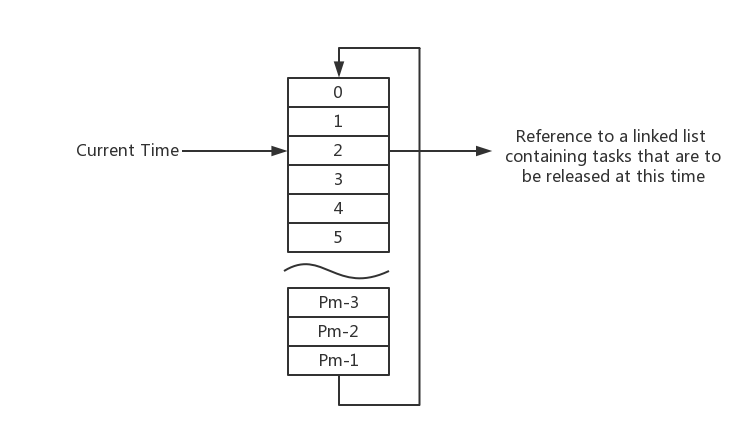
# 4 改进的EDF调度

## 4.1 改进思路

以上的基本实现中由于使用链表维护任务的先后顺序，所以具有O(n)的时间复杂度。改进算法将链表改为一个平衡二叉树构成的优先队列和一个时间轮结构数组。其中优先队列存放当前处于就绪状态的任务，它按照任务的截限时间进行排序并调度执行；时间轮中存储着在不同时间释放的等待任务，其中Pm是所有任务中周期的最大值。



就绪任务优先队列



等待任务时间轮

在每个时钟周期中，令current\_time = (current\_time + 1) % Pm，检查数组当前项结构体中的OS\_TCB指针是否指向一个待释放的任务链表，如果有，则将其移入就绪任务优先队列，如果队首任务发生了改变则进行任务抢占，否则继续执行。当一个任务在本周期执行完成时，将其移入下一个释放时间对应的时间轮的任务链表中。

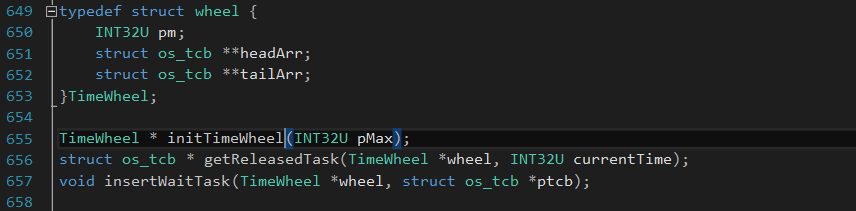
## 4.2 实现方法

首先，新建一个优先队列的minheap.h文件，如下：



该优先队列接受任务的优先级和截限时间，并按照截限时间排序，从而在选择最近的界限时间的任务时可以有O(logn)的时间复杂度。

然后，需要在ucos\_ii.h头文件中增加时间轮的结构体和对应函数声明，如下：



结构体中的两个指针数组分别表示每个时间点对应的待释放任务的头指针数组和结尾指针数组，使用两个指针是为了链表插入式从尾部插入，否则会导致任务释放后插入就绪优先队列时相同释放时间的任务顺序颠倒。任务之间的连接方式使用了在OS\_TCB中增加了一个用于连接时间轮中任务的指针：

struct os\_tcb \*TimeWheelNext; /\* Pointer to next TCB in time wheel \*/

然后，将原来的OS\_SchedEDF()函数修改为使用从优先队列队首peek的方式，获得最近任务的优先级，从而实行调度：

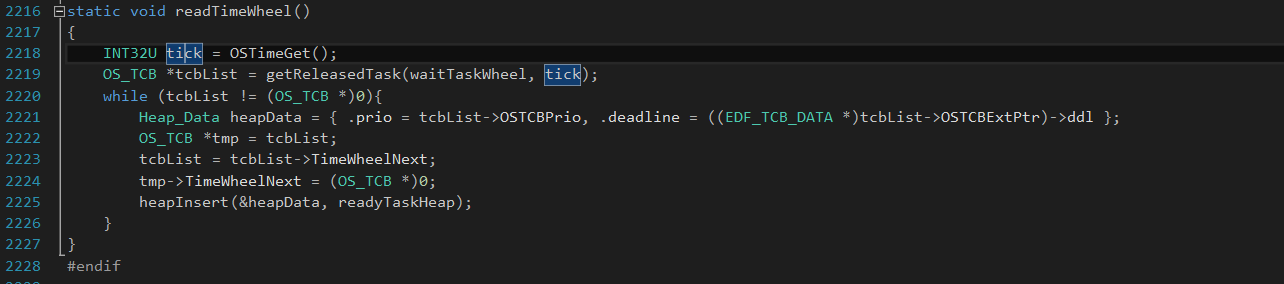
static void OS\_SchedEDF(void)

{

OSPrioHighRdy = heapPeakMinTask(readyTaskHeap);

}

最后，在每个时钟周期的处理函数OSTimeTick()中的最后执行readTimeWheel()，通过时间轮查看当前是否有任务需要释放，如果有则将其拿出来插入优先队列，没有则继续执行，这样就将原来的线性时间复杂度改成了O(logn)的时间复杂度。



## 4.3 测试用例

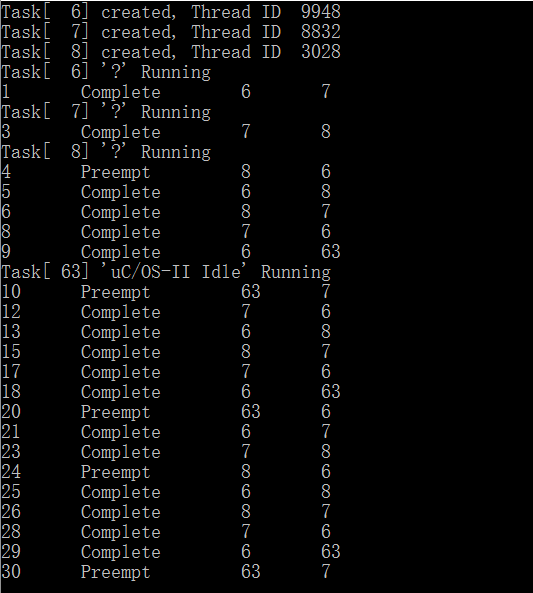
使用老师提供的第二个任务集来测试：

任务1：c\_value = 1, p\_value = 4

任务2：c\_value = 2, p\_value = 5

任务3：c\_value = 2, p\_value = 10

结果如下：



# 5 参考资料

1. Short, Michael. "Improved task management techniques for enforcing EDF scheduling on recurring tasks." 2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium. IEEE, 2010. (<http://ieeexplore.ieee.org/document/5465964/>)