

The Research and Implementation on Extending Algorithm that High Response Ratio Tasks Schedule Preferentially on μ C/OS-II

Fensu Shi

Beifang University of Nationalities
Yinchuan, China
E-mail: shifensu@163.com

Hua Wang

Beifang University of Nationalities
Yinchuan, China
E-mail: lenmohuoyan@126.com

Abstract—To overcome shortcomings that μ C/OS-II, based on static priority scheduling policy, may block the tasks in low priority for a long time, task scheduling policy based on high response ratio scheduling preferentially is added and achieved. This scheme divided the task in the system into significant tasks and secondary tasks. The significant tasks possess the low priority value statically and the secondary tasks possess the high priority value dynamically according to response ratio. It revises TCB and rewrites the function named OSSched and OSCtxSw. It maintains the hard real-time characteristic of μ C/OS-II. Task's scheduling mode is optimized and application field of μ C/OS-II is broadened.

Keywords- μ C/OS-II; scheduling policy; High response ratio scheduling preferentially; OSSched; OSCtxSw

I. INTRODUCTION

μ C/OS-II is a tiny and open-source embedded operating system developed by Jean J. Labross. It is a preemptive and hard real-time kernel based on static priority. It is compact, real-time and multitasking support, so it can be easily transplanted in various singlechips and embedded systems. μ C/OS-II is widely used in industrial control field and it possess high security and instantaneity.

However, μ C/OS-II is a kernel base on static priority. when the high priority tasks require long service time, they can make the low priority tasks starve for a long time, which impacts throughput rate of embedded system seriously. In the field of monitoring or controlling multiple objects at the same time, if the high priority tasks occupy the processor for a long time, it will lead the equipment not to obtain real-time outside information in time and even lose the information needed to be collected in many severe cases. In addition, the kernel can't afford a good mechanism to solve the problem of priority inversion. These defects have greatly restricted the application of the μ C/OS-II.

In some recent years, many researchers and engineers at home and abroad have started to research and realize various scheduling policies on μ C/OS-II. All their work strive to break through the limitation of application which are brought by μ C/OS-II's scheduling policy base on static priority. In literature[3], The author proposes a realized method --time slice scheduling algorithm on μ C/OS-II. This scheduling policy no longer consider task's priority, so the whole tasks use the processor in turn. But, in this case, interrupting and switching task's context frequently is a giant system burden. Strategy that high response ratio tasks

schedule preferentially is a superior task scheduling mechanism. It balances short run-time job and sequence, long run-time job. In this mechanism, every task has its dynamic priority. The scheduler schedules tasks according to their dynamic priority. In this paper, in order to make the task scheduling more efficient and reasonable, we research and realize this scheduling algorithm on μ C/OS-II.

II. PRINCIPLE ANALYSIS ABOUT TASK SCHEDULING STRATEGY ON μ C/OS-II

A. Brief introduction of preemptive scheduling policy based on static priority

μ C/OS-II can manage up to 64 tasks. The author retains the tasks whose priority is 0, 1, 2, 3, OS_LOWEST_PRIO-3, OS_LOWEST_PRIO-2, OS_LOWEST_PRIO-1 or OS_LOWEST_PRIO, so the user can actually create 56 tasks. Every task is given a static priority which is constant in task's life cycle, when created. The lower the priority value is, the higher its actual priority will be. The system always make the task which has been already and has the highest priority run. The work identifying the highest priority task and deciding which task to run is finished by scheduler.

B. The analysis on adding the scheduling strategy that the high response ratio task schedules preferentially

In scheduling strategy that the high response ratio task schedules preferentially, the response ratio needs to be calculated before every task scheduling. At last, the task, possessing the highest response ratio and having been already, obtains the processor.

If the wait time is T_w , required service time is T_s , and response ratio is R_{sp} . There is a following formula.

$$R_{sp} = \frac{T_w + T_s}{T_s} \quad (1)$$

It's easy to know that The higher the response ratio is, the lower the task's priority value will be.

To maintain the μ C/OS-II's hard real-time characteristic, we divided the task into two categories. The first category includes important and real-time tasks. We call them significant tasks and let them own low priority values which are statically typed; The other category includes secondary tasks which do not care too much about the characteristic of real-time and whose priority values are

dynamically typed and relatively large. When created, they acquire referential priority values. Their practical priority values are based on response ratio. Set up a global variable 'BNDRY_Pri'. When we need to create a significant task, we give it a priority value less than BNDRY_Pri and the secondary tasks greater. According to the task's characteristics in the system, the size of BNDRY_Pri is set up by embedded system developers when configuring the kernel.

III. IMPLEMENTATION OF MODIFIED TASK SCHEDULING STRATEGY ON μ C/OS-II

A. Modification on relevant data structure

In the original task control block (TCB), we add following members: priority category, commit time, required service time, response ratio, starting time, referential priority. Typedef double RSP_RATIO_TYPE
typedef struct os_tcb {

```

    . . . . .
    int  OSPriType; // priority category, dynamic or static
    INT 32U  OSTSKSubTim; // commit time
    INT 32U  OSTSKserTim; // required service time
    RSP_RATIO_TYPE  OSTSKSPRAT; // response ratio
    INT 32U  OSTCBBgnrTim; // starting time
    INT 8U   OSTCBORfnclPrio // referential priority
    . . . . .
}

```

After modifying TCB, some program statements are needed to be added to initialize new members.

```

    INT8U OSTCBInit (....., INT32U  ostsksertim)
    {
        . . . . .
        if (prio > BNDRY_Pri) ptcb->OSPriType = DYNMC;
        else if (prio < BNDRY_Pri) Ptcb->OSPriType = STATIC;
        else err = OS_PRIO_INVALID;
        Ptcb->OSTSKSubTim = OSTimeGet();
        Ptcb->OSTSKserTim = ostsksertim;
        Ptcb->OSTSKRSPART = 1;
        Ptcb->OSTCBBgnrTim = 0;
        OSTCBORfnclPrio = prio;
        . . . . .
    }

```

The global variable 'BNDRY_Pri', Set up previously, is the boundary of two kinds of task priority value. The variable is not used when creating tasks, but changing the secondary task's priority value according to response ratio. Before scheduling task, secondary tasks's response ratio are recalculated and the secondary ready task which has highest response ratio is found. At last, this task's priority value is set as BNDRY_Pri.

In this way, the significant tasks occupy low priority values statically, which maintains μ C/OS-II's hard real-time characteristic. At the same time, The priority value of secondary task that possesses the highest response ratio is adjusted to be relatively low in accordance with the thought that high response ratio tasks schedule

preferentially.

B. Modification on relevant function

Redefine the function named OSSched and rename it to OSSched_RSPRT, realize the comprehensive scheduling policy above.

```

    void OSSched_RSPRT (void)
    {
        . . . . .
        OS_ENTER_CRITICAL();
        INT 32U  TimNOW = OSTimeGet ();
        for (TMP = OSTCBLis; TMP != NULL; TMP = TMP->OSTCBNext) {
            if (DYNMC == TCM->OSPriType)
                TMP->OSTSKSPRAT = (TimNOW - TMP->OSTSKSubTim + TMP->OSTSKserTim) / TMP->OSTSKserTim;
        }
        Sort(OSTCBLis); // sort by response ratio
        for (TMP = OSTCBLis; TMP != NULL && DYNMC == TMP->OSPriType; TMP = TMP->OSTCBNext)
            if (TMP->OSTCBStat == OS_STAT_RDY)
            {
                TMP->OSTCBPrio = BNDRY_Pri;
                // Find the secondary ready task which has highest response ratio, set its priority value as BNDRY_Pri, and then make it ready.
                OSRdyGrp |= OSMapTbl[TMP->OSTCBPrio] >> BNDRY_Pri;
            }
            OSRdyTbl [TMP->OSTCBPrio >> BNDRY_Pri] |= OSMapTbl[TMP->OSTCBPrio & 0x07];
            break;
        }
        OSSched();
        OS_EXIT_CRITICAL();
    }

```

Many tasks, which can't be completed in consecutive time due to waiting for resources, being blocked and so on, need to be repeatedly switched. So, before switching tasks, old task's required service time need to be recalculated and the new task's commit time must be reset. After the switchover is completed, the task's starting time should be recorded. Modify task switching function as follows:

```

    Void OSCtxSw(void)
    {
        INT 32U  Tim_TEMP;
        Tim_TEMP = OSTimeGet ();
        OSTCBCur->OSTSKserTim = OSTSKserTim - (Tim_TEMP - OSTCBBgnrTim);
        OSTCBCur->OSTSKSubTim = Tim_TEMP;
        OSTCBCur->OSTCBPrio = OSTCBCur->OSTCBORfnclPrio;
        // The statement in original OSCtxSw function.
        OSTCBCur->OSTCBBgnrTim = OSTimeGet ();
    }

```

IV. TEST AND VERIFICATION ON MODIFIED TASK SCHEDULING STRATEGY

In order to verify the algorithm that high response ratio tasks schedule preferentially on μ C/OS-II and improved and realized in this paper, use BC45 to compile, link and debug the modified μ C/OS-II on X86 under MS-DOS. Create five tasks. they are Task1, Task2, Task3, Task4, Task5.

Set BNDRY_Pri at 5 and clock rates at 100. The basic information of the tasks is in the Table 1.

TABLE 1: THE BASIC INFORMATION OF THE TEST TASKS

Task name	Referential priority	Task descriptions
Task1	4	Collect data every 200ms and each data collection needs 50ms. Print character 'A' on screen before and after this task.
Task2	5	Communicate via serial port every 300ms and each communication needs 50ms. Print character 'B' on screen before and after this task.
Task3	6	Store data and each operation needs 1s. Print character 'C' on screen before and after this task.
Task4	7	Report data to upper computer and each operation needs 500ms. Print character 'D' on screen before and after this task.
Task5	8	Query the amount of sample data which has been collected and each query needs 100ms. Print character 'E' on screen before and after this task.

Run the program, a sequence of characters appear on the screen:

...AAEBBDAACAABBAAE...BAAB...

As Task1's priority value is the highest, Two A are outputted together. Task2's priority value is second-highest. it can be interrupted by Task1, so subsequence "BAAB" appears. The remaining tasks occupy processor in the light of response ratio size when Task1 and Task2 are idle. Two C

or Two D are absolutely impossibly outputted together. Thus, it verifies that the task scheduling strategy realized in this paper is right and feasible.

V. CONCLUSION

μ C/OS-II, a hard real-time kernel, widely used in industrial control field, only supports static priority scheduling policy and can't take lower-priority tasks into account all the time.

Aim to remedy this defect, algorithm that high response ratio tasks schedule preferentially is added and realized in this paper. A small amount of the kernel code has been revamped, which makes μ C/OS-II's kernel schedule tasks more reasonably.

ACKNOWLEDGMENT

It is a project supported by National Science Foundation pre-breeding program (Grant No. 2012QZP01) and School-enterprise cooperation in scientific and technological breeding project (Grant No. 2012XQG2) of Beifang University of Nationalities.

REFERENCES

- [1] Xiaodan Tang. Computer Operating System. Xian: Xian University of Electronic Science and Technology Press, 2007.
- [2] Jean Jlabrosse. MicroC/OS-II, The Real-Time Kernel. Translated by Beibei Shao. Beijing: Beijing University of Aeronautics and Astronautics Press, 2003.
- [3] Siliang Gong, Xiuting Zuo etc. Time slice rotation task scheduling strategy based on dynamic priority. Modern Electronics Technique, 2012.
- [4] Houfa Cheng, Chunjin Yang. UC/OS-II, Improvement on Operating system kernel. Journal of Communication and Computer, 2006.
- [5] Decao Mao, Ximing Hu. Embedded System. Hangzhou: Zhejiang University Press, 2003.