Task Scheduling Method for Embedded System Composed of All Linear Tasks

Xing Chen*, Hongkun Li and Yugang Chen
School of Mechanical Engineering, Dalian University of Technology, Dalian 116023, China
*email: chenxing@dlut.edu.cn

Abstract. There are still a large number of small and medium-sized embedded software applications that do not use operating system due to various conditions, for which a lot of time and effort are spent to build appropriate software structure in doing development. This paper presents a concept of linear task and a scheduling method based on the linear task, which can all use C language to realize the fully distributed dynamic multi-task scheduling of embedded single machine system based on the basic foreground-background system structure. On this basis, the system structure of the embedded software application without operating system can be quickly built and the subsequent application function development can be completed. Through the example implementation and simulation verification, it is proved that this method has good hardware independence, low CPU occupancy, small data and code space, flexible application support, comprehensive function, easy time coordination and simple real-time control. The application of this method to build an embedded system composed of all linear tasks can provide an effective foundation for dealing with the increasingly complex application requirements and limited development cycle.

1. Introduction
Embedded software development should divide tasks and realize functions on the basis of hardware support and requirement analysis. Reasonable scheduling of tasks with specific methods is the basic guarantee of system reliability and operation efficiency [1][2]. The core function of the operating system is task scheduling[3][4], but the existence of copyright constraints, resource occupation, comprehensive cost of system transplantation and application limits the practical application of embedded operating system[5]. The basic foreground-background system structure is still used in a considerable number of embedded system applications[6]. Research on task construction and scheduling method with less resource demand and good hardware independence is of great significance for the implementation of embedded software application without operating system. This paper attempts to design a cooperative task scheduling method for embedded system composed of all linear tasks based on the foreground-background system structure, in order to support the rapid construction of system structure and the complete application function development.

2. Linear task and task linearization
Software tasks in embedded system can be divided into short-term tasks, long-term tasks and infinite time tasks according to the time of respective completely running. Software task design and management are closely related to the task scheduling. Non-preemptive scheduling allows tasks to run continuously until they are finished or blocked, while preemptive scheduling can forcibly suspend the continuous...
running of tasks to prevent a single task from monopolizing the CPU for a long time. The former is suitable for short-term tasks and self-blocking long-term tasks, and its real-time characteristics depend on the cooperation of the tasks, so it is more used for special system scheduling; The latter with large system overhead and good real-time response has no special requirements for the types of tasks to be scheduled, and is more oriented to infinite loop tasks. It is the first choice of general operating system, and is also used in the currently available embedded operating systems. If the embedded system application does not use or is not suitable for the operating system for various reasons, the foreground-background system structure is the most basic starting point of task management[7], that is, the interrupt task performs the most urgent preliminary processing of the event in the foreground, and the main thread completes the necessary scheduling and coordination of subsequent processing tasks in the background. In the task design, the foreground interrupt task must be a short-term task, and the background main thread is equivalent to a large infinite loop body. The processing task involved in the background main thread is not allowed to be an infinite time task, or even a long-term task when the real-time performance of this kind of system is required.

The linear tasks defined in this paper firstly refer to those short-term tasks that can go through a complete execution as simply and quickly as interrupt threads, and also include those modified long-term tasks and infinite time tasks that can be run in segments, in which each running segment must have no explicit or implicit timeout and can simply and quickly complete one shot running of itself just like short-term tasks. The basic implementation of foreground-background system or even non-preemptive operating system is that the main thread cooperates with the interrupt thread to call or schedule related linear tasks in a certain way. The real-time performance of such systems is mainly determined by all linear tasks.

Nonlinear tasks are those long-term and infinite time tasks that have not been modified. The main reason why long-time tasks do not meet the requirements of linear tasks is that there is an event or time waiting in at least one part of its internal program, or the super long time-consuming caused by complex processing and operation. The linearization transformation of long-term task is to segment the program running, set new segments in each place where there is waiting of event or time, and reasonably divide the long-time processing and operation into multiple segments that can be connected, and record the current segment of the program running using static state variables. After the linearization transformation, the task will be further scheduled repeatedly by the scheduling system, and each segment will be run in turn under the control of state variables. Finally, the overall operation of the whole task program can be synthesized. This method of subsection transformation and operation of program can be called program slice method. For the infinite loop task, keep its loop body and carry out similar subsection transformation when necessary. The original outermost infinite loop will be supported by the infinite scheduling of the scheduling system.

3. Full linear task scheduling method
Embedded system composed of all linear tasks can be called embedded system of full linear tasks. The full linear task scheduling method accordingly may first assume that all related tasks are linear tasks, and all work is expanded around a linear task list by the main thread of the foreground-background system. Each list item of the linear task list contains the vector address and task state of the corresponding registered tasks, and each registered task can be in different control states such as stop, ready and delay. The reserved length of task list can be accurately determined according to the maximum number of tasks in embedded system. The full linear task scheduling scheme is shown in Figure 1. The system support function library includes the support for task list operation. The initial part of the main thread completes the necessary initialization of the related hardware, and the subsequent work of the main thread is in the infinite loop of the system start-up support function. The system tick interrupt thread is mainly responsible for timing decrement of task delay until 0. The cycle of tick interrupt determines the resolution of task delay.

The flow of system start-up support function is shown in Figure 2. First, the task list is cleared, and the start-up task is loaded into the list to complete the initial registration of the task (task registration
can be carried out implicitly, as long as the operation task is not in the task list, it will be automatically registered as a new task, and new linear tasks can be registered at any time later); After that, the program goes into the infinite cycle of task scheduling, where the user tasks in the ready state will be sequentially scheduled and executed. The tasks registered in the list have three different list states: Ready, Delay and Stop. Tasks that are not in the list are regarded as offline and will not be scheduled; The current task will be temporarily marked as running state, and the list state of the task can be modified by its own or other tasks using system support call according to the needs of subsequent functions. Except for the two cases that the task in Delay state automatically turns to Ready state at the end of delay and the task in Ready state is scheduled to running state, other task state transitions are all the result of the explicit call of related system support functions by linear tasks.

In the full linear task scheduling method described above, each task can modify the list state of itself or other tasks by calling the system support function. The change of task state is the result of self-operation and inter-operation of all related tasks. Therefore, this scheduling model is fully distributed and cooperative.

4. Sample implementation
The core of a sample implementation of the above full linear task scheduling method is the set of system support functions listed in Table 1.

① Task login and logout: Function 1 is to assign an item in the task list for the corresponding task, and the corresponding item number is the task number used for internal management; Function 2 is to mark the corresponding table item as invalid, unless the number of table items reserved in the task list is insufficient due to the shortage of RAM space, it is usually unnecessary to log off the task.

② Task state setting and query: Function 3 is to set a task state marked with 16 bit signed binary number for the corresponding task, where the value 0 corresponds to ready state, the value greater than zero corresponds to delay state and is used to mark the tick number of remaining delay, the pre-defined negative values are used to correspond to the Stop state and Offline state respectively; Function 4 returns the current status of the specified task in the task list.

③ System start-up: Function 5 is used to start-up the scheduling system. As the input parameter of the system start-up function, the start-up task submitted is responsible for the initial loading of other necessary linear tasks.

④ Timing callback: Function 6 should be called periodically in the hardware timing interrupt service program to provide tick service and count down the task delay.

⑤ Idle proportion: Function 7 returns a value between 0 and 100, indicating the time percent of running idle tasks in the corresponding time length of every 100 ticks.

⑥ Task duration: Function 8 returns a table pointer, which records the time (number of ticks) consumed by the longest linear execution of every registered tasks so far.
Table 1. System support function list

| Nr | Definition Definition                                      | Description Description                        |
|----|----------------------------------------------------------|-----------------------------------------------|
| 1  | TaskNr LT_Sys_TaskLogin(TaskPnt taskname)                | Task login                                    |
| 2  | TaskNr LT_Sys_TaskLogout(TaskPnt taskname)               | Task logout                                   |
| 3  | TaskNr LT_Sys_TaskStatSetting(TaskPnt taskname, TaskSt stnum) | Task state setting                            |
| 4  | TaskSt LT_Sys_TaskStatQuery(TaskPnt taskname)            | Task state querying                           |
| 5  | void LT_Sys_Start(TaskPnt starttask)                     | System start-up                               |
| 6  | void LT_Sys_iCallback(void)                             | Timing callback                               |
| 7  | TaskNr LT_Sys_IdleRate(void)                            | Idle proportion check                         |
| 8  | TaskNr * LT_Sys_TaskTicks(void)                          | Task duration check                           |

All task functions should have a unified function prototype. The simplest form is that there are neither input parameters nor return parameters. In particular, linear tasks may be scheduled at any time and executed repeatedly from the beginning. Task function prototype with input parameters or return parameters is meaningless, and it will increase the difficulty of management with input parameters. This example implements the definition of task serial number type TaskNr, task state type TaskSt and function pointer type TaskPnt as follows:

- typedef signed char TaskNr;
- typedef signed short int TaskSt;
- typedef TaskNr (* TaskPnt)(void);

According to the definition of TaskPnt, the task function must have no input parameters and a return parameter of type TaskNr. The return parameter of type TaskNr is only reserved for system debugging, and can be removed uniformly in practical application. There is yet no support for information exchange between tasks in the set of system support functions. Global variables can be used to share information among tasks. Every scheduling execution of each task is a complete function call, and no consideration of critical area protection between tasks is needed.

5. Application verification

Complete C language programming makes the implementation of full linear task scheduling method on specific hardware very simple. Both 51 series microcontrollers with insufficient internal RAM and 32-bit ARM core microcontrollers can be applied and implemented quickly. One of the most basic works is to reasonably determine the reserved space of task list according to the expected number of linear tasks involved in scheduling. The differences of data types in different hardware C language environments may also have subtle effects, which need to be paid attention to during deployment.

Figure 3 is the program block diagrams of application verification on microcontroller LPC2138 based on KEIL ARM compiler environment. It is assumed that 8 buttons (named K1~K8) are connected through P0.16~P0.23 of port P0. Each port wire is grounded when the button is pressed, and pulled up to high level through resistance when the button is opened; There are 8 LED indicators (named L1~L8) connected through P1.16~P1.23 of port P1; Both serial port 0 and serial port 1 are used as standard I/O interfaces. Seven linear tasks are defined in the software application part, which are start task itaskMenu, L5~L8 cycle control task itaskL5~itaskL8, button response task itaskBtn and information output task itaskInfo. Therefore, at least seven items should be preserved in the task list. Through Proteus simulation, tasks itaskL5~itaskL8 flip their respective LED lights L5~L8 in different cycles, task itaskBtn flips L1~L8 when press of button K1~K8 is confirmed, task itaskInfo responds with the scheduling idle information when query request received, task itaskMenu linearized by means of program slicing is responsible for the initially loading of other tasks and the menu control and response. The whole system runs stably and the real time response meets expectations. The screenshot of Proteus debugging run of this verifying application is shown in Figure 4.

Through the above application verification, it can be seen that compared with most embedded operating system's "centralized" task scheduling implementation, tasks of system using fully linear task scheduling method can be closely related to scheduling and become a part of the "distributed" task scheduling. The code of this software is very small. The size of object program of the system support function set in KEIL ARM7 environment is about 700 bytes, and in KEIL C51 environment is about...
The real-time performance of task response will be simply discussed here. On the one hand, it can be required that the processing capacity of the hardware carrying the application system should be enough to meet the routine operation requirements of all tasks; On the other hand, the design of linear tasks should ensure the reasonable CPU consumption of each task. The return information of function LT_Sys_TaskTicks can be used to guide the division and cooperation of tasks, and test the real-time response of tasks in extreme cases.

6. Conclusions
A large number of small and medium-sized embedded software applications do not use operating system due to various conditions, and a lot of time and effort are spent to build appropriate software structure for them. The cooperative multitask processing method based on linear tasks can realize the dynamic multitask scheduling of embedded single machine system only by using C language. Based on this, the system structure can be built quickly and the subsequent application function development can be completed in the embedded software application without operating system. The results of the example implementation and simulation show that the method has good hardware independence, small data and code size, low CPU occupancy, flexible application support, comprehensive function and simple real-time control. This can provide a strong foundation guarantee for the small and medium-sized embedded system to cope with the increasingly complex application demand and limited development cycle. Further research may be carried out on the method of synchronization of linear tasks and the evaluation of real-time of full linear task embedded system.

Figure 3. Program block diagrams of application verification
Figure 4. Application verification example running in Proteus simulation environment

References

[1] Amit K. Shukla, Rachit Sharma, Pranab K. Muhuri. A Review of the Scopes and Challenges of the Modern Real-Time Operating Systems[J]. International Journal of Embedded and Real-Time Communication Systems (IJERTCS), 2018, 9(1): 66-82.

[2] Muhammad Ali Awan, Stefan M. Petters. Intra-task device scheduling for real-time embedded systems[J]. Journal of Systems Architecture, 2015, 61(8): 321-340.

[3] Sheng- lei Zou. Research on task scheduling algorithm based on embedded system[J]. Electronic Design Engineering, 2019(7): 180-183.

[4] Amina Magdich, Yessine Hadj Kacem, Mickaël Kerboeuf, Adel Mahfoudhi, Mohamed Abid. A design pattern-based approach for automatic choice of semi-partitioned and global scheduling algorithms[J]. Information and Software Technology, 2018, 97(5): 83-98.

[5] Fan Qian, Bo Gao. A Real-time Task Scheduling Method Based on DSP6713[J]. Electronic Test, 2019(08): 12-14.

[6] Xuan Chen, Jinyu Chen. Design of Embedded Front and Back Control System Based on STM32[J]. Modern Information Technology, 2020, 4(01): 39-40+43.

[7] Chunhua Zhou. Event-driven Programming Frame Design for Embedded System[J]. Microcontrollers & Embedded Systems, 2017, 17(4): 5-7.