Software Agent Design with Real Time Scheduling for Embedded Systems

Hu Jin¹, Liang-Yin Chen², Nian-Wei Chen¹, Yang Lei¹
¹Computer Science Department, Chengdu Univ. of Information Technology, Chengdu 610041, Sichuan, China
²College of Computer Science, Sichuan University, Chengdu 610065, Sichuan, China
cn_jh@cuit.edu.cn

Abstract

The Embedded Systems are now in use more commonly in our daily lives. They can accomplish many complicated tasks in different working environments, and concurrency and synchronization are ordinary problems the embedded software usually must be confronted with. This issue presents a novel method for controlling such problems efficiently. To avoid the inefficiency or collision induced by directly coupling amongst multi-tasks that conventional methods mostly did, software agent architecture designing can cope with such work more properly. The main work of this article includes designing agent architecture for typical concurrency embedded system applications, and adapting classic RMA for scheduling tasks which have soft real time requirements. Lastly, a case study has been done on a typical embedded system scenario, where multi-peripheral devices parallel processing is analyzed with this method and the results show that can get a good concurrence performance.

Key words: Software Agent, Architecture Design, Real Time, Embedded System, RMA.

1. Introduction

As the application area of embedded system widely broadened in recent years, its working environments are more complex and constraint. Thus it needs much more dedicate and systematic architecture design for meeting the requirements of the application such as real time, concurrence, efficiency, etc. Processes fulfilling different tasks are not simply added in together for an integrated system [1] [2] [3]. The relationship amongst processes needs to be considered more carefully. For embedded software system, having many peripherals is quite normal. Device drivers are responsible for controlling the behaviors of the peripherals. Since devices are rather independent of center module, and have much more uncertainty time cost because of different tasks such as data collection, status control or data preprocess, the drivers then are converters that connect slower speed devices with higher performance center modules. In conventional architecture, drivers usually directly interact with the center modules [4] [5]. When the number of driver tasks is enormous and much more synchronization constraints existing for those parallel processes, such a directly coupling structure seems not very suitable, which may lead to a disaster for the performance of system.

Since directly coupling architecture is not fit for solution of the previous problem, software agent design seems proper for effectively controlling multi-access and parallel processing [6] [7]. Software agent can be defined as referring to a component of software and/or hardware which is capable of acting exactly in order to accomplish tasks on behalf of its user [9]. In this issue, agent is deputy for the peripheral drivers to concurrency and synchronizing. It rearranges the multi-access sequence for supporting as many parallel processing requests, and with specific scheduling algorithm [9] [10] such as Rate Monotonic Scheduling (RMS) algorithm [11] [12] [13] for satisfying time constraints of tasks. The indirect coupling architecture of agent can well adapt to meeting the requirements of parallel processing for better performance, and it has good extensibility for future multi-agent applications.

This issue is organized as follows. Section 2 introduces software agent architecture design for parallel processing embedded applications. Section 3 applies RMS algorithm for scheduling concurrency tasks by agent. Section 4 does case study with a typical
scenario and verifies the effectiveness of this method. Last section summarizes the work of this paper.

2. Agent Architecture for Embedded System

Embedded system architectures normally comprising of software programmable components and customized hardware co-processors, integrated into a single cost-efficient chip \[1\][2]. Heterogeneous ever has been most often used design architecture for embedded system. Such architecture design is an efficient way for combining hardware or software with general purpose computational capabilities and more dedicated modules \[4\]. Although depending on different application requirements, applications may have variable components and with different combination methods, there are also some similarities in them. Next figure 1 depicts typical heterogeneous embedded system architectures.

![Figure 1 Heterogeneous embedded system architectures](image)

As the above illustration depicts, DSP is the kernel component of embedded system. Also multi-processors architecture will be needed depending on the specific applications requirements. Aside from the kernel component, integrating hardware and software components together and providing communication between them are rather a timing consuming and error prone task. Moreover, if tasks have deadlines or time constraints, such a directly coupling architecture amongst modules would not be competent in such work. It will cost tremendous amount of time for cooperation tasks properly, and thus may lead to violation of time constraints.

Except directly coupling amongst peripheral drivers, the parallel processing can also be worked out with agent. Software agent has evolved from multi-agent systems. It disposes issues such as interaction and communication between processes, the scheduling and allocation of resources, negotiation and cooperation. Although multi-thread programming tool kits can implement such synchronization or concurrency even with priority support, it is still a tough work to fulfill parallel tasks spontaneously but at the same time guaranteeing meeting individual deadline constraints and integral optimal performance.

Software agent is a more prominent solution for such problems. According to the research of Steven \[5\], the directional kernel communication can be redesigned for an intermediate scheduling layer for better performance. With considering the time constraints of parallel processing tasks, the agent contains a task scheduler for deciding and arranging tasks activities sequence. Figure 2 illustrates the agent architecture design, and the agent is responsible for communication with different parallelizable tasks, allocation resources and collaboration between tasks.

![Figure 2 Heterogeneous embedded system architectures with agent](image)

Agent design is in software level, and would not change the hardware structure of embedded system. The agent schedulers and synchronizes components that in charge of the concurrent tasks. Concurrent tasks with synchronization or real time constraints should register in the agent firstly. Then, after agent figuring out reasonable execution sequence, it synchronizes all tasks and schedules them in turn. That can avoid the heavy and complicated synchronization work amongst processes and lowers shared resources access collision. The agent needs not to be persistent for economic resources usage, and it can betriggered by concurrent tasks request dynamically. When receiving new service requests, the agent will re-computes possible response time and decides whether or not they can be merged into current task group for scheduling.

Nowadays, distribution applications connected by networks are in common. It can afford more complex work in scalable network environment. Steven \[14\] represents a programming model and I/O substrate for high concurrency services. Software agent architecture design can also be adapted like that for internets distribute processing with concurrency and co-operations.
3. Soft Real Time Scheduling Supporting

As described above, the software agent should properly group the peripheral requests and enforce specific scheduling algorithm for concurrent tasks. Rate Monotonic Scheduling Algorithm was proposed by Liu [15] is a highly effective and widely used algorithm for real-time application. Then it can be introduced into the agent design for such group scheduling work.

According to Liu, supposing system consists of n tasks \( S = \{t_1, t_2, \ldots, t_n\} \). Also these tasks should be independent of each other. If this group of tasks is schedulable, it can satisfy the next equation,

\[
\sum_{i=1}^{n} \frac{C_i}{T_i} \leq U(n) = n(2^{\frac{1}{n}} - 1)
\]  

where \( C_i \) is the worst time cost and \( T_i \) is the period of task \( i \). Tasks with shorter periods are assigned higher priorities with RMS algorithm.

Usually, the tasks are not completely independent because there is synchronization or cooperation. Then, RMS relaxes the previous constraint and gets the next equation.

\[
\sum_{i=1}^{n} \left( \frac{C_i + B_i}{T_i} \right) \leq U(n) = n(2^{\frac{1}{n}} - 1)
\]  

where \( C_i \) and \( T_i \) are the same as that of (1). \( B_i \) is the time that task \( i \) blocked by lower priority tasks. Case that lower priority processes block higher ones is said to priority inverse. Formula (2) guarantees that synchronization tasks can meet real time requirement.

Sprunt and Lehoczky [16] advanced this research to support non-periodical tasks scheduling. Considering two types of non-periodical tasks \( T_E \) and \( T_R \). \( T_E \) is a temporal emergent task, and its deadline is \( D_E \). \( T_R \) is also non-periodical task but with little real time constraint. \( C_R \) is the worst execution time for non-periodical tasks. In fact, \( T_R \) is equivalence period for non-periodical cycle. Then, average response time consists of the average waiting time \( W_q \) and average execution time \( W_e \).

\[
W = W_q + W_e
\]  

\[
\rho = \frac{D_q}{I}
\]

\( D_q \) is the time interval between continuous tasks quitting from queue, \( \rho \) is the tasks occurrence frequency which is used for evaluating the average CPU occupation. \( I \) denotes average tasks occurrence interval.

\[
W_q = \frac{(\rho^* D_q) / (2(1-\rho))}{\rho^* D_q / (2(1-\rho))}
\]  

Substitute \( D_q \) with \( T_R \), \( W_q \) can be represented as

\[
W = \frac{T_R^2 / I}{2(1-T_R/I)} + C_R
\]

Then, the \( T_R \) is estimated as next formula:

\[
T_R = (C_R - W) + SQRT((W - C_R) (W - C_R + 2I))
\]

The advanced algorithm designed by Lehoczky [16] can meet most of application time constraints.

With RMS scheduling strategy, algorithm can be designed for agent to manage a group of parallel tasks. When multi-access requests happened in designate intervals, the agent can figure out whether or not the current tasks could be merged. If they can meet their time constraints, then compute their priority and sequential scheduling the tasks. Else, tasks should be scheduled in a new group. Figure 3 depicts the agent work process.

From this figure, it shows a soft real-time scheduling function because it is not a preemptive method. It simply delays new task request when current queue can not satisfy constraints. The schedule module is implemented with RMA algorithm described previously. It will lead client task to miss its deadline when new request can not be satisfied with current schedule group. However, it at least can avoid the case that new task contest with scare resources, thus guarantee not to deteriorate the whole system performance.
4. Case Study

Experiment has been done with typical configurable software of an embedded system. The architecture of that system is centralized oriented. Tasks concerned with distributed nodes in charge of data collection from peripherals but with shared bus and memory to connect center node. Peripheral devices work cooperatively for efficient access of shared resources. Each PLC controller directs data collection task periodically and acquires shared resources when request was replied. Two methods can be applied for solve such problem. The one is by dedicating designing every device driver activity for access synchronization and cooperation to meet its time constraints. Evidently it is a very complicate work. The other is by using the agent method. Simply by register time constraint requirements and synchronization needs of every driver task, the agent is responsible for actual accessing scheduling.

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Worst Exec. Time</th>
<th>Task Period</th>
<th>SRT Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>50ms</td>
<td>100ms</td>
<td>High</td>
</tr>
<tr>
<td>T2</td>
<td>100ms</td>
<td>200ms</td>
<td>Normal</td>
</tr>
<tr>
<td>T3</td>
<td>150ms</td>
<td>200ms</td>
<td>Normal</td>
</tr>
<tr>
<td>T4</td>
<td>200ms</td>
<td>400ms</td>
<td>Normal</td>
</tr>
<tr>
<td>T5</td>
<td>300ms</td>
<td>1000ms</td>
<td>Low</td>
</tr>
</tbody>
</table>

Next figure is the scenario for multiple data collection devices with synchronization or parallel cooperation.

The main procedure issues control signals for starting every peripheral task. Then, the driver initializes device status and sets devices parameters such as task period, intervals or communication channel. When device is ready, it registers a timer for controlling periodical behaviors and submits its information to the agent for scheduling almost at the same time. For every new resource mutual task request is coming, the agent should re-compute current group execution information, and decide to reset scheduling priority and synchronize concurrent task group again. According to returned scheduling information, drivers direct devices operation sequentially and periodically.

Table 2 describes average performance comparison with and without agent in a period of 10 minutes burst time for parallel processing.
Table 2. Performance Comparison for two methods

<table>
<thead>
<tr>
<th>Agent Occur.</th>
<th>Collision Aver. Wait Time</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>1000</td>
<td>48</td>
</tr>
<tr>
<td>No</td>
<td>1000</td>
<td>75</td>
</tr>
</tbody>
</table>

In that period, the agent architecture design can accommodate about 13% concurrent tasks activity number more than the counter part. Also cases of missing dead line are less than that of spontaneous method. After scheduling, it sacrifices waiting time for better execution sequence.

From above data, the performance of parallel processing is evidently improved by using agent architecture design. It is mainly because that the agent makes a forward estimation thus avoid access collision amongst concurrent tasks and schedules synchronous tasks in a more efficient way.

5. Conclusion

In summary, this paper has presented software agent architecture managing multi-tasks in typical embedded systems. By applying scheduling algorithm, it can get more reasonable tasks activities sequence than that of by spontaneous co-ordination. The future work will focus on how such software agent architecture can be applied in internet environment for distribution tasks.

Acknowledgement

This work was supported by the National scientific and technological middle or small sized enterprise innovative fund under grant No 06C26225101730.

6. References


