A Study of Optimization of RTOS for Individual Applications

LI JIN(1310206)

School of Information Science
Japan Advanced Institute of Science and Technology

May 19, 2016

キーワード: RTOS, system call, optimization, embedded system, error checking, locking .

- **Introduction**

  Nowadays, RTOS is widely used in the development field of embedded systems. By using RTOS, the efficiency of resource management and real time processing can be easily guaranteed. However, the functions of RTOS are fixed and a lot of RTOS functions are unnecessary for the developing of target applications. Therefore, this may affect the system overhead and binary file size. Since RTOS is assumed to be used in all kinds of embedded systems, various error checking codes are included for executing applications. Error checking for illegal use of ID numbers, non-existing objects, and using of unauthorized parameters are examples. The error checking is performed during runtime, hence execution overhead may happen. In general, most embedded systems are specially designed for specific applications. Therefore, not all of the provided RTOS error checking are required for particular applications. Thus, by removing the extra error check codes irrelevant to that specific application the execution overhead and the size of binary codes can be reduced significantly. In addition, some system calls could be requested at the same time from multiple tasks. In order to guarantee the consistency of data, the locking mechanisms in system calls are used. However, in some applications, it is not necessary to provide such exclusive access to kernel data. In the same manner as described above, it would be effective to remove the locking codes on the place where locking would not be required. In this research, a new method is proposed for automatic removal of unnecessary checking and locking codes, based on the application code analysis results. By using the proposed method, the system developers can optimize the embedded systems automatically. Finally, evaluation of the proposed methods is presented to illustrate the effectiveness of the proposed methods. The evaluation

showed the importance of such RTOS optimization that can greatly help in future development of embedded systems.

- **Proposed Method**
  In this research, the application source code will be analyzed, and the methods to detect the unnecessary error code checking by system calls will be implemented. Then, the necessity of the error check codes will be determined and the required ones will be stored in a header file named “define.h”. Depending on the values written in the header file, the error checking codes within the system calls can be removed. In the next step, a method to detect unnecessary locking done by the system calls is proposed. Based on the analysis results of the application source code, after checking if the locking code is necessary or not, the results will be recorded in another header file named “define2.h”. Similarly, depending on the values written in the header file, the locking codes within the system calls can be removed.

- **Evaluation**
  The proposed system is implemented and evaluated via a simulation. A set of tasks is created to be executed by the system. In order to evaluate its performance, size and execution time of the binary code as well as the number of executed instructions will be recorded. During the experiment, removal of the error checking codes from the target system call has reduced the average binary size by 16.88%. Average runtime was lowered by 10.46%, as well the average number of executed instructions which was reduced by 9.46%. The average size of the binary code after the removal of the lock code was reduced by 6.84%. Runtime was reduced by 28.51% on average whilst the average number of executed instructions was reduced by 35.21%.

- **Conclusion**
  In this research, a method to automatically remove the unnecessary codes from the system calls was proposed, based on the results of the application code analysis. In the evaluation results, it was shown that this method can reduce the execution overhead and binary size. By analyzing the source code of the application task-set, the method detects unnecessary error checking codes in system calls. The necessity of error checking related to system calls was determined by symbolic constant analysis and operands. In addition, the analysis of actual possible conflicts in-between tasks determines the necessity of utilizing locking mechanisms. The proposed methods work inputting the system/application implementation source codes. The system

calls from which unnecessary error checking and locking codes are removed, are the output of the proposed methods. An implemented task-set is used to evaluate the performance of the proposed methods.