Phased Hierarchical Formal Verification of Memory Management System for Spacecraft

(Extended Abstract)

Lei Qiao
Beijing Institute of Control Enginerring
State Key Lab. of Computer Science, Institute of Software, CAS
Beijing, China
fly2moon@aliyun.com

Bo Liu✉
Beijing Institute of Control Enginerring
Beijing, China
liub@bice.org

Hua Yang
Beijing Institute of Control Enginerring
Beijing, China
yangh@bice.org

Yanliang Tan
Beijing Institute of Control Enginerring
Beijing, China
tanyl@bice.org

Mengfei Yang
Chinese Academy of Space Technology
Beijing, China
yangmf@bice.org

Abstract—The dependability of operating system software is crucial, due to its central position in various types of spacecraft computer applications. With the rapidly increasing complexity of software, traditional methods based on testing are no longer capable to ensure the correctness and reliability of spacecraft operating systems. Whereas, formal methods have gradually become the fundamental guarantee for the dependability of spacecraft operating systems. However, the current level of research is still at an exploratory stage in general and faces many challenges. Based on recent development in the theory and tools of software verification on modern operating systems, this paper proposes a phased hierarchical theory with top-down cyclic iterations, which is demonstrated in verifying the memory management algorithm as well as code implementation of SpaceOS, a spacecraft embedded operating system with practical applications.

SpaceOS, currently undergoing space flight, is the first spacecraft embedded real-time operating system developed independently in China. It is positioned as a spacecraft-specific real-time operating system, with low system resource occupancy, strong real-time performance, and reduced functionality. Due to high efficiency and accuracy, SpaceOS has been used in China's lunar exploration projects, the new generation of navigation satellites, space station and other major projects. Although SpaceOS has been validated through numerous manual and automated tests, some obvious errors and irregularities are discovered afterwards, while further testing is required for potential vulnerabilities and errors remaining in the system. In order to verify the correctness of SpaceOS in terms of the highest level of software safety, formal methods are needed. Because the design of SpaceOS is of good modularity, it can be divided into clear modules for formal verification. This paper is intended to verify the memory management module of SpaceOS.

Most of the spacecraft operating systems marked by VxWorks use the first adaptive algorithm to organize the memory allocation, which is based on the single block linked list and the worst-case time complexity is O(n). Spacecraft embedded operating system features a strong real-time property, and thus requires high real-time performance of the memory algorithm. The general dynamic memory allocation algorithm has problems of indefinite execution time and excessive memory fragmentation, and is rarely used in real-time embedded systems. Whereas, the Two Level Segregated Fit (TLSF) dynamic memory algorithm has the advantages of constant memory allocation, reduced time complexity, automatic memory merging, high flexibility, and less memory fragmentation, and has been widely applied in practical systems, such as Amiga OS, Xtratum Hypervisors, Orocos, and SpaceOs as well. This paper presents verification of the TLSF algorithm encoded in the memory management module of SpaceOS to ensure the reliability under efficient memory allocation.

The TLSF algorithm adopts a dynamic memory management method based on a two-level index table combined with separate tables. It reduces the wort-case time complexity from O(n) to O(1), yielding superiority in time average performance and certainty. The memory fragmentation rate is also reduced through the separation of tables, the organization of memory space, the good-fit matching principle based on two-level bitmaps, and the method of immediate retrieving and releasing. This algorithm not only solved the dynamic task creation problem, but also addressed the trade-off between flexibility and real-time reliability. Based on this method, the spacecraft real-time multi-tasking operating system solves the problems of real-time scheduling and real-time switching of application tasks under shared computing resources, while satisfying the multi-task concurrency of integrated software. The scheduling requirements of co-existing tasks provide basic guarantees for the multi-functional parallelism and smooth completion of tasks for complex spacecraft. In the meantime, however, the complexity of the algorithm is increased compared with the aforementioned memory management algorithms. Formal verification of the algorithm will also be more complex concerning the following aspects: (1) formal description of the complex data structure in the memory management module; (2) specification semantics of interface functions and operations of the memory

management module; (3) specification and assertion definitions of the internal functions of the memory management code and loop invariant definitions; (4) real-time verification of the memory management operations.

Concerning the insufficient results currently on formal verification of the spacecraft memory management module, this paper deeply analyzes the characteristics of practical operating system memory management, explores and studies general methods and theories of semantical description and verification of memory management based on a phased hierarchical approach with top-down cyclic iterations, and applies the results to the verification of the memory management algorithm as well as code implementation of SpaceOS with practical applications. Beyond that, our approach can be readily extended to conduct verification of other modules of the operating system. The main contribution of this work lies in performing an essential exploration of the reliability verification tailored for practical embedded operating systems, and the results presented in this paper are expected to be applied directly to the new generation of spacecraft systems in China.

Focusing on the key issues mentioned above, the basic idea follows from a modeling process where the requirement, design and implementation of the memory management module are formalized in a hierarchical top-down manner, together with a verification process performed in the tool Rodin that conducts Event-B mathematical abstraction. The resulted model is then verified in terms of specifications extracted from the requirement documents, thus illustrating the correctness of the memory model in the operating system.

According to the development process of software engineering, our method is composed of three major stages, i.e., demand analysis, design, and coding. Every stage is based on the existing document data to formally describe and establish a model. Internal correctness is ensured within each stage while guaranteeing consistency between consecutive phases. In the first stage, the functions and properties are extracted from the requirement specification documents, and the models of different modules are established. The main concern is that the functional operations proposed in the user requirements must satisfy the nature of the design requirements description, and the correctness of the functions and properties themselves. The second stage is to incrementally refine the model in a hierarchical manner to obtain the design model, which ensures the consistency between the upper and lower layers. The design model continues to be refined, where specific data structures and other implementation methods are used as refinement bases, until the implementation model is obtained. In the third stage, an abstract model is extracted from the C code implementation, followed by a checking of consistency between the implementation model and the C model. Hoare logic is further used to verify the abstract model as a basis for correct implementation. For C code verification, we make improvements on problems found to be inconsistent, detect bugs in the original code, and make corrections.

ACKNOWLEDGMENTS

This work is supported in part by grants from National Natural Science Foundation of China (NSFC) under Grant Nos. 61632005 and 61502031. Lei Qiao's work is also supported by grant from The State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences. Any opinions, findings, and conclusions contained in this document are those of the authors and do not reflect the views of these agencies.