Abstract—Currently, more and more application-specific operating systems (ASOS) are applied in real-time embedded systems. With the development of microkernel technique, the ASOS is usually customized based on the microkernel using the configurable policy, which has various alternatives. In the design of the real-time embedded system (RTES) based on such ASOS, evaluating its timing performance at the early design stage is helpful to guide the designer towards choosing the most appropriate policy. However, the existing works lack a uniform approach to support analyzing the various alternatives of the configured policy. To solve this problem, this paper presents a general-purpose timing analysis approach for the ASOS-based RTES. In the analysis, a timing analysis tree is proposed to characterize the tasks and the ASOS in the RTES. Then, each of the alternative policies in the ASOS is refined by a traversal of the execution rules in the tree. Finally, the task’s response time under the various alternative policies is analyzed by a traversal of the timing analysis tree for the various alternatives. In the case study, we take the scheduling policy as an example to show the use of our approach on a real-life robot controller system.

Index Terms—real-time embedded system, microkernel-based RTOS, application-specific operating system, alternative policy, timing analysis

I. INTRODUCTION

In real-time embedded systems (RTES), the real-time operating system (RTOS) is usually used to manage the tasks in the system, and directly impacts their timing performance [1]. The RTESs in various domains may suffer from the general-purpose operating system (OS) due to their specific characteristics. Currently, many works are aimed at the application-specific operating systems (ASOSs) to enhance the performance of a certain application [2], e.g., microkernel architectures are representative ASOSs. Nowadays, more and more practical RTOSs are designed based on a microkernel, such as QNX, Integrity, and FreeRTOS. A microkernel [3] is a minimalistic kernel that contains the near-minimum amount of functions and features required to implement an OS, it adopts the “separation of mechanism and policy” principle. Such principle makes it convenient to build arbitrary OS services using the configurable policy. When customizing an ASOS, every configurable policy has various alternatives, each of which has a different influence on the response time of the task. Hence, in this work, we are interested in the timing analysis of the design of the RTES, which is implemented on a customized ASOS based on the microkernel with various alternatives for the configurable policy.

In the real-time systems, the timing analysis approaches can be divided into two categories: dynamic and static. The dynamic approaches, which include the simulation and model checking, suffer from the efficiency problem when applied to the case mentioned here. For the simulation, each alternative policy requires generating a policy-dedicated simulation model, which is not feasible for a general-purpose. For the model checking, it also needs to concern the policy-dedicated rules throughout the task model [4]. Thus, we resort to the static analysis approaches, which include three classes [5]: structure-based, path-based, and the technique using implicit path enumeration (IPET). Both the path-based approach [6] and the IPET [7] are inadequate for our case due to they do not consider the OS. As for the structure-based approach, it can only support the specific function of the OS [8], [9], and is inadequate to analyze the various alternative policies here in a general purpose way.

In this paper, we propose a timing analysis approach specific for the RTES based on a microkernel-based ASOS, which is customized by the configurable policy that has various alternatives. In order to perform the timing analysis for the various alternatives uniformly, we first propose a structure of timing analysis tree, which is used to characterize the tasks and the ASOS in the RTES. Then, we define a canonical form of the execution rules to refine the various alternatives in the timing analysis tree. Based on the execution rules, we finally propose a general-purpose analysis technique by a traversal of such timing analysis tree for the various alternatives. In the case study, we take the scheduling policy as an example to show the use of our approach on a real-life robot controller system. Comparing with the state-of-the-art methods, the superiority of our approach is that it simplifies the analysis by fixing the tasks and the ASOS mechanisms, and only replacing the part of the configurable policy.

This paper is organized as follows: Sect. II discusses the related works; Sect. III introduces the background and overview of our approach; Sect. IV proposes the timing analysis...
analysis approach; Sect. V evaluates our approach on a real-life case; and Sect. VI gives some concluding remarks and perspectives.

II. RELATED WORKS

Currently, the timing analysis of the RTES includes two different classes of methods [5], that is the dynamic methods and the static methods.

The dynamic methods rely on the simulation or the model checking. For the simulation-based methods, the works [10], [11] mapped the MARTE model to the SymTA/S model for timing analysis based on formal scheduling analysis techniques and symbolic simulation; the work [12] proposed a simulation-based timing analysis depending on a more detailed system model, which described the execution control flow at the code level. When used in our case, the simulation-based methods need a model transformation (or refinement) for each alternative policy, which is inflexible. For the model checking, the work [13] presented an analysis method for the worst-case execution time (WCET) using UML-MARTE model checker, which was aimed at detecting wrong software designs and refined the correct ones with respect to WCET; the work [14] mapped the activity diagram of UML into the priority time Petri net (PTPN) to enhance the formal schedulability test of given real-time tasks; the work [15] mapped the workload model of real-time systems into a Petri Nets formalism to generate all transactions for the timing analysis. However, as for our case, the model checking method needs to specify the policy-dedicated rules throughout the task model, which is flexible or even impossible.

The static methods include three classes [5]: structure-based, path-based, and techniques using implicit path enumeration (IPET). In the path-based method [6], the execution time is determined by analyzing the paths in the task. In IPET [7], the control flow and the basic-block execution time are combined into the constraints to analyze the execution time of the task. Both the path-based method and the IPET don’t consider the OS’s functions in the execution of the task. In the structure-based method [16], the execution time is analyzed in a bottom-up traversal of the syntax tree of the task. The syntax tree takes the functions or subtasks of a task as the nodes, so the interactions between the nodes can be used to concern the OS’s functions, such as synchronization [17], instruction cache locking [9], etc. However, the structure-based method can’t support analyzing the various realizations of the function in a general purpose way. For example, the work [17] proposed three analysis methods for the three instruction cache locking strategies, i.e. static locking, semi-dynamic locking and dynamic locking.

III. BACKGROUND AND OVERVIEW

A real-time embedded application is usually designed as a set of tasks managed by the RTOS [18], i.e. the ASOS here. The microkernel-based ASOS includes three basic mechanisms that cover the essential functions of the microkernel, i.e. the task scheduling, the inter-process communication (IPC), and the resource access [3]. Each mechanism can be extended using a set of alternative policies. The task consists of a sequence of functional blocks with some system calls [19]. The system call is realized by the basic system calls in the ASOS. The functional block is used to realize an independent function and composes the execution sequence of a task [19].

The overview of our approach is shown in Fig. 1. The RTES design includes the tasks and the ASOS. The response time of each task is analyzed based on its control flow, which is characterized by the execution sequence of the task. Such control flow is influenced by the ASOS behavior, which varies with different configured policy. Here, we propose to refine the ASOS behavior at the analysis stage. Besides, the timing information needs to be specified for the tasks and the ASOS. Specifically, the worst-case execution time (WCET) is pre-defined for each functional block in the tasks and each basic system call in the ASOS. For each alternative policy, the timing analysis of the RTES design is implemented by combining the control flow and the ASOS behavior to analyze the tasks’ response time.

IV. TIMING ANALYSIS APPROACH

In our approach, we define an extensible timing analysis tree (ETAT) to characterize the task and the ASOS, where the alternative policy can be replaced flexibly (i.e. extensible). If a new policy is configured, the only part needs to be modified in the ETAT is the policy node together with its child node.

A. Extensible Timing Analysis Tree (ETAT)

In this section, we first define the semantics for the extensible timing analysis tree (ETAT); then, we propose a canonical form to define the execution semantics for the ETAT, which is used to refine the ASOS behavior to perform the timing analysis. Based on the proposed canonical form, we introduce the execution rules for the three mechanisms in ASOS, i.e. scheduling, IPC and resource access.

1) Definition of ETAT: In the RTES, each task is modeled as an ETAT, which consists of a set of nodes and edges. The node is defined as TreeNode = (time_cost, component_attribute), where the time_cost attribute records the time cost of the represented component, the component_attribute attribute characterizes the attributes of the represented component. There are three types of nodes in the ETAT as follows:

![Fig. 1. Overview of our approach](image-url)
object node specifies the tasks and the functional blocks.

operation node specifies the ASOS behaviors, including mechanisms and configurable policies

parameter node specifies the basic system calls and execution rules for the ASOS behavior.

The various relationships between the nodes are defined as different types of edges in the ETAT. Each type of edge can only exist between a pair of certain type of nodes. The are five types of edges, which are listed as follows:

use: A task or a functional block uses the mechanism or the policy in the ASOS; The execution rules and the basic system calls are used by the mechanism or the policy.

realize: A policy is realized based on the mechanism.

consist: A task consists of a set of functional blocks.

sequence: The successor of a functional block in the execution sequence is its sub-sequence.

operate: The mechanism or the policy operates on the task or the functional block.

2) Definition of the Execution Semantics: The ASOS behavior is refined by the execution rules in the ETAT, which define the operating actions and the timing actions for the behavior. Specifically, the operating action expresses the operation for this behavior; the timing action indicates there is a time cost for the operation. A canonical form for the execution rules is defined as

\[
\text{State } [\text{Condition}] / \text{Action} \rightarrow \text{State'}
\] (1)

where State represents the current state of a task, Condition means the condition affecting the execution of the task, and Action is the operating or the timing action for the task triggered by the satisfaction of conditions.

The execution rules for scheduling mechanism are defined as shown in Fig. 2.

- Four basic states of a task (running, ready, blocked, suspended) are represented by St_Run, St_Ready, St_Block and St_Suspend, respectively.
- The set of conditions consists of Cond_Preempted, Cond_First_Run, Cond_Wait_Event, Cond_Event_Arrive, and Cond_Time_Out.
  - Cond_Preempted represents the condition that makes a task be preempted.
  - Cond_First_Run represents that the task is selected to run first among the tasks in the ready queue.
  - Cond_Wait_Event represents that the task is waiting for an event.
  - Cond_Event_Arrive represent that the waited event arrives.
  - Cond_Time_Out represents that the waiting is timeout.
- The operating actions (i.e. running, readying, blocking and suspending) are represented by Act_Run, Act_Ready, Act_Block and Act_Suspend respectively, and the timing action is defined as Act_Timing.

The execution rules for resource access mechanism are defined as shown in Fig. 3.

- Two basic states of St_Run and St_Block are involved.
- The conditions of Cond_Request_Resource and Cond_Release_Resource are used.
  - Cond_Request_Resource represents that the task requests a resource during its execution.
  - Cond_Release_Resource represents that the requested resource is available right now.
- The operating actions include Act_Run, Act_Block, Act_Check_Resource, Act_Get_Resource and Act_Timing. Among them, Act_Check_Resource is to check whether the resource is available, Act_Get_Resource is to obtain the available resource.

The execution rules for IPC mechanism are defined as shown in Fig. 4.

- Two basic states of St_Run and St_Block are involved.
- The conditions of Cond_Request_Communication and Cond_Release_Communication are used.
  - Cond_Request_Communication represents that the task requests a communication with other task during its execution.
  - Cond_Release_Communication represents that the connection for the requested communication is set up.
- The operating actions include Act_Run, Act_Block, Act_Connect_Setup, Act_Communicate and Act_Timing. Among them, Act_Connect_Setup is to set up the connection, Act_Communicate is to communicate with other task.

B. Timing Analysis for ETAT

The response time of a task consists of the scheduling time, the interaction time (with other functional blocks), and the WCET of the functional blocks in this task. Both the
scheduling time and the interaction time rely on the ASOS behavior, and consist of the WCET of the basic system calls. Such response time is analyzed by a traversal of the ETAT of this task. During the traversal, the time cost of the root node (i.e., the task node) indicates the current execution time of the task, and is updated once the time cost of the scheduling node or that of each functional block node is worked out. The time spent at each object node of functional block is analyzed based on its operation child-node. The time spent at each operation node (include scheduling and interaction) is analyzed based on its parameter child-node. When the ETAT is completely traversed, the time cost of the root node indicates the response time of the task.

function timing_analysis_process(T)
    1: visit the root node rn of T
    2: if (rn.traversalsFinished == true) then
    3:     return
    4: else
    5:     visit the scheduling child-node sc of rn
    6:     OperationNodeAnalysis_forScheduling
    7:     update rn.time_cost
    8:     if (task.state != "executable") then
    9:         return
   10: else
   11:     visit the functional block child-node fb of rn
   12:     set fb as the functional block node to be analyzed fb_ta
   13:     while (fb_ta != null)
   14:         if fb_ta.executionCondition is satisfied then
   15:             ObjectNodeAnalysis_forFunctionalBlock
   16:             update rn.time_cost
   17:         else
   18:             return
   19:         end if
   20:     end while
   21:     set the functional block child-node of fb_ta as fb_ta.
   22:     visit the node fb_ta
   23:     return rn.time_cost as the response time
   24: end if

Fig. 5. Timing analysis process

Given an ETAT T, the timing analysis process is shown in Fig. 5. First, visit the root node of T to check whether T is completely traversed (L. 1). If not, visit the scheduling child-node of the root node to check whether the task is executable (L. 5). Then, analyze the time cost of the scheduling child-node, and update the time cost of the root node (L. 6-7). If the task is executable, we visit the functional block child-node (say fb) of the root node, then visit the functional block child-node (say fb') of fb, then visit the functional block child-node of fb', ... until all functional block nodes are visited (L. 20). For each functional block node, we check its execution condition and analyze its time cost based on the operation node (if exists) and the parameter node (the analysis procedure will be introduced later), then update the time cost of the root node (L. 14-16). When all the functional block nodes are visited, the task is set as completely traversed, the time cost on the root node indicates the response time of the task.

Next, we specifically introduce the scheduling node and the functional block node mentioned above. The time cost of the scheduling node is the time spent at the scheduling operation. The time cost of the functional block node includes the time spent at the object itself and at the interaction operation (if exists). Therefore, we focus on the two types of nodes, i.e. the object node and the operation node. According to the definition of ETAT, the basic structures of object node and operation node are summarized in Fig. 6. For the object node, it has a child node of the object type with a consist (for task) or sequence (for functional block) edge between them. If the object node has a scheduling operation or an interaction operation, an operation node is generated as its another child node with the use edge. For the operation node, it has a child node of parameter with the use edge. If the operation node has an extended operation (for policy), the realize edge is used to link them. If the operation node has an other operand, a child node of the object type is generated for the operand with the operate edge.
analyzed based on the execution rules (the timing actions particularly) and the WCET of basic system calls. If this operation node has a realize child-node, the realize child-node is actually an operation node, whose time cost is analyzed by the same way. If this operation node has other operands except for its father node (as the operation node is used by its father node, the father node is one operand of this operation), the operate child-node is actually an object node, whose time cost is analyzed by the same way as C1.

V. CASE STUDY

A. Experimental setup

In this section, we will illustrate the application of our approach to a real-life robot controller system [20]. The robot controller system (RCS), which consists of three tasks, is used to keep the robot operating normally. Among the tasks, the balance task is to keep the balance of the robot by calculating the input from the gyroscope and the inclinometer; the navigation task is to avoid obstacles during the process of going to the destination; the remote task is to receive a remote command via the infrared. The services of the infrared sensor, the gyroscope and the inclinometer are realized by the interrupt service routines (ISR), which are corresponding to infrared_isr, gyro_isr and inclino_isr respectively. To implement such RCS, we use the μC/OS-II kernel [21] to configure the ASOS. The μC/OS-II kernel implements a static priority scheduling policy, and has an optional policy of round robin scheduling. In this case study, we analyze the timing performance of the three tasks in the RCS to assess these two scheduling policies.

For the ASOS, the WCET of the basic system calls in the μC/OS-II kernel given in [22] is used in this case study. About the two scheduling policies in the ASOS, the time slice of the round robin (RR) scheduling is set as 10 thousands CPU cycles, the priorities (P) for the three tasks in the static priority (SP) scheduling are set as: P(balance) = 4, P(navigation) = 6, P(remote) = 5. For the tasks, their timing requirements are represented by the deadline (D), and set as (in one thousand CPU cycles): D(balance) = 200, D(navigation) = 40, D(remote) = 4000. Within the tasks, the functional blocks (FBs) together with their WCET are set as shown in Table I.

B. Experimental process and results

First, we define the execution rules for the two scheduling policies. As shown in Fig. 7, these execution rules refine the preempted condition in the scheduling mechanism (as shown in Fig. 2). Specifically, for the SP scheduling, an arbitrary task T is preempted when there exists a ready task with a higher priority than T; for the RR scheduling, the task T is preempted when the time slice for T is used up. It should be noted that the CON_PREEMPTED in the execution rules of scheduling mechanism (as shown in Fig. 2) is set by the actions of ACT_SET_PREEMPTED_TRUE or ACT_SET_PREEMPTED_FALSE in the execution rules of the two scheduling policies.

Then, we construct the timing analysis trees for the three tasks as shown in Fig. 8, where the object nodes, the operate nodes and the parameter nodes are represented by the colors of green, orange and blue respectively in each timing analysis tree. As space is limited, the attributes of each node in the trees are not presented.

After the timing analysis, the response time of each task under the two scheduling policies is presented in Table II. As seen, the static priority scheduling can meet the deadline of the tasks, while the round robin scheduling cannot. This case study takes the two scheduling policies as an example to illustrate the feasibility of our approach. Without loss of generality, any other scheduling policies can also be analyzed based on the timing analysis trees in Fig. 8 by defining their execution rules.

VI. CONCLUSION AND PERSPECTIVE

In the domain of the real-time embedded system, more and more application-specific operating systems (ASOS) are customized based on the microkernel using the configurable policy. The existing methods usually need an individual timing analysis for each alternative policy. To simplify the analysis, we propose a general-purpose timing analysis approach for such ASOS-based RTES design. A real-life robot controller system is used as a case study to show the feasibility of our
approach. Currently, our approach only supports the configurable policies of the three basic aspects, i.e., scheduling, interprocess communication, and resource access. With the RTES is becoming more and more complex, the more functions are needed by the ASOS, such as network management, file system, etc. In the near future, we will extend our approach to support more configurations in the ASOS.

REFERENCES


[18] Y. Harada, K. Abe, M. Yoo, and T. Yokoyama, “Aspect-oriented customization of the scheduling algorithms and the resource access protocols of a real-time operating system family,” in Smart City/SocialCom/SustainCom (SmartCity), 2015 IEEE International Conference on. IEEE, 2015, pp. 87–94.


