Real-Time Scheduling for Preventing Information Leakage with Preemption Overheads

Hyeongboo BAEK\textsuperscript{1}, Jinkyu LEE\textsuperscript{1}, Jaewoo LEE\textsuperscript{2}, Pyung KIM\textsuperscript{3}, Brent Byunghoon KANG\textsuperscript{4}

\textsuperscript{1}Department of Computer Science and Engineering, Sungkyunkwan University, Suwon, 16419, Republic of Korea
\textsuperscript{2}Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104, USA
\textsuperscript{3}Department of Industrial and Systems Engineering, Seoul National University of Science and Technology, Seoul, 139743, Republic of Korea
\textsuperscript{4}School of Computing, Korea Institute of Science and Technology, Daejeon, 305338, Republic of Korea

Corresponding author Brent Byunghoon Kang: brentkang@kaist.ac.kr

Abstract—Real-time systems (RTS) are characterized by tasks executing in a timely manner to meet its deadlines as a real-time constraint. Most studies of RTS have focused on these criteria as primary design points. However, recent increases in security threats to various real-time systems have shown that enhanced security support must be included as an important design point, retro-fitting such support to existing systems as necessary. In this paper, we propose a new pre-flush technique referred to as flush task reservation for FP scheduling (FTR-FP) to conditionally sanitize the state of resources shared by real-time tasks by invoking a flush task (FT) in order to mitigate information leakage/corruption of real-time systems. FTR-FP extends existing works exploiting FTs to be applicable more general scheduling algorithms and security model. We also propose modifications to existing real-time scheduling algorithms to implement a pre-flush technique as a security constraint, and analysis technique to verify schedulability of the real-time scheduling. For better analytic capability, our analysis technique provides a count of the precise number of preemptions that a task experiences offline. Our evaluation results demonstrate that our proposed schedulability analysis improves the performance of existing scheduling algorithms in terms of schedulability and preemption cost.

Index Terms—embedded software, real-time systems, scheduling algorithms, security, system analysis and design.

I. INTRODUCTION

Real-time systems (RTS) control multiple physical devices executing multiple real-time tasks in a timely manner. Since meeting deadlines of real-time tasks is the most important requirement of RTS, most conventional studies have focused on real-time aspects of RTS for several decades while security has not been considered as a first class principle of the design of RTS [1-15]. During this period of time, security has not been accorded sufficient attention at a level commensurate with scheduling algorithm design. However, recent attacks on various RTS have resulted in catastrophizes, including loss of life. We believe that, at this point in time, lack of paramount importance that security be added as a design point for inclusion in any redesign of a RTS. There are many documented attacks on a variety of RTS, including telematics units installed in modern automobiles [16-17], industrial control systems [18], and unmanned autonomous vehicles [19-20]. However, applying existing security techniques to RTS cannot be effective because most of these techniques were developed without consideration of the real-time requirements [21-30], and this explains, to a large extent, why RTS are easily compromised.

The lack of support for robust security as an indigenous component of RTS has recently generated a large amount of research. In recent three years, three studies [31-33] have proposed approaches that a pre-flush mechanism to invoke a flush task (FT) is included into real-time scheduling with a design point of real-time systems to eliminate or substantially reduce information leakage that occurs in real-time tasks that share resources and execute with different levels of security.

The initial work [31] first proposed a pre-flush mechanism that will invoke a FT to sanitize the state of all shared resources such as all caches, DRAM and I/O buses. Figure 1 describes an example how a FT is invoked between the executions of two tasks having different security levels; for two tasks A and B, A has a security level higher than B. If B is dispatched immediately subsequent to the dispatch of A, then B is more easily compromised by attackers due to its lower security level and can be used to remove, copy, or corrupt sensitive data that exist on the shared resources in use by A. To prevent this, a FT is invoked at the beginning of the execution of B and executes non-preemptively. The work presented in [31] proposed a new fixed priority (FP) [34-35] real-time scheduling algorithm that includes security constraints regarding conditional invocations of FTs and an associated new response time analysis (RTA) [36-37]. However, this work is limited to non-preemptive FP scheduling and does not provide tight upper-bounds of the worst-case number of FTs thereby illustrating the limited analytic capability for schedulability.

A subsequent work [32] extended the initial work to...
include preemptive FP scheduling and to provide a tighter upper-bound of the worst-case number of FTs for schedulability analysis by transforming the problem of upper-bounding the worst-case number of FTs into a min-cost flow problem [38]. However, this work did not suggest the exact schedulability analysis including the calculation of the exact preemption cost.

Our previous work [33] extended the two works proposed in [31-32] by deriving a new schedulability analysis technique that can compute the exact number of FTs that a task under analysis will experience during the life of a task. We proposed a new preemptive FP scheduling algorithm referred to as lowest security-level first (LSF). This algorithm employs a new pre-flush technique that invokes FTs in a more intelligent manner, referred as flush task reservation (FTR), which induces the exact schedulability analysis (i.e. sufficient and necessary analysis) that can compute the exact number of FTs hindering the execution of a task offline. However, FTR can be used only with LSF scheduling, and its performance is insufficient relative to rate monotonic (RM) [34-35] scheduling, the optimal scheduling algorithm deployed for preemptive FP uniprocessor scheduling.

This paper also addresses the problem of information leakage, the subject of three prior studies, and we propose a new pre-flush mechanism referred to as flush task reservation for FP scheduling (FTR-FP) and a new schedulability analysis technique thereof. FTR-FP is not restricted to a specific FP scheduling algorithm, and will relax the limitation of a FTR that we described above. Unlike the results from three prior conventional [31-33], our new schedulability analysis can estimate not only the exact number of FTs but also the exact number of preemptions that a task will experience. Previous two studies [31, 33] considered introducing a security constraint that a FT is invoked during a context switch from a task with a higher security-level to a task with a lower security-level. Invoking a FT during a context switch in both directions, , and , can alleviate the real-time scheduling performance due to the invocation of more FTs. However, the possibility of information leakage from task to task should not be ignored as it can also compromise the security of the RTS. We show our technique maintain reasonable performance even though we consider both directions of information leakage to better mitigate the possibility of information leakage.

The primary contributions of this paper are the advantages of our proposed techniques compared to the prior techniques in [31-33]:

- By proposing a FT invocation during the context switch in both directions of transition, and , as a security constraint, we enhance the system security relative to the conventional approaches [31], [33] that implement FT invocation at context switch time in a single direction, .
- The new pre-flush technique and schedulability analysis are not limited to a specific FP scheduling, offering an advantage over the previous technique [33] that is restricted to LSF.
- A new schedulability analysis can calculate the precise number of FTs and preemptions, but existing three schedulability analyses [31-33] can upper-bound maximum number of FTs only.

II. ADVERSARY AND SYSTEM MODELS

In this section, we present the adversary and system models. The design of our adversary and system models are based on production RTSs in various fields as described in [31-33], e.g., an avionics system based on the DO-178B model [32]. Our models also include a new security constraint that extends the security constraints pursuant to [31], [33].

A. Adversary Model

We assume that an adversary understands the nature of the task parameters as well as the RTS scheduling policy. The primary objective of the adversary is to execute a successful intruder initiated by an attacker that exceeds the probability of successful intrusion originating from tasks executing at a lower security-level. For example, an adversary can launch a side-channel attack by hijacking a task executing at a low security-level to access sensitive data within the shared cache, concurrently accessed by a task executing at a higher security-level. The attacks against a RTS can be successful in a RTS that supports a multi-tier or hierarchical security level model. An example of such a model is an avionics RTS that implements the DO-178B standard. This model partitions the RTS into sub-systems with defined interfaces. Usually, the development of the sub-systems is contracted to different vendors such that each vendor possesses the necessary security level (clearance) to bid on the sub-system. For example, a sub-system that controls a camera that captures images to be sent to the command center can be designed by a vendor with the required security clearance to use and observe confidential data. On the other hand, the avionics navigation system may be developed by a vendor with less restrictive security credentials as the function of this sub-system is less critical relative to securing shared resources. If the navigation system, developed by a vendor at a low security level, is compromised or corrupts shared sensitive data, then the camera system, created by a vendor with a higher security level, can generate images that can be copied, deleted, or corrupted.

B. System Model

We consider the preemptive FP scheduling of a set of periodic task set on a uni-processor [34-35]. A task is characterized by a 3-tuple , where is the period of a task or minimum inter arrival time, is the worst-case execution time (WCET) and is the relative deadline. We assume implicit deadlines, i.e., is equal to . A task releases its job periodically according to its period . We use the notation if the job indicates the -th released job of task . The WCET of a FT is
denoted by $C_{ji}$. We assume the synchronous release of tasks, so that the first jobs of all tasks are released concurrently, i.e., at the time instant 0. We also assume quantum-based time for real-time scheduling and schedulability analysis.

Previous work [31], [33] considered the security-level of real-time tasks and focused on the prevention of information leakage from the resources shared by a task executing at a higher security-level $\tau_H$ than a task executing at a lower security-level $\tau_L$. As defined by this security model, a FT should be invoked to sanitize the state of the shared resources only if a lower security-level task $\tau_L$ executes immediately after the execution of a higher security-level task $\tau_H$. However, the possibility of information leakage attributed to a context switch from $\tau_L$ to $\tau_H$ should not be ignored because the change of execution state can also compromise the security of the RTS. Thus, we address the problem of information leakage that arises during both context switches, $\tau_H \rightarrow \tau_L$ and $\tau_L \rightarrow \tau_H$, to better reduce the possibility of information leakage. We define a security constraint to be integrated within the real-time scheduling algorithm as follows:

Definition 1: (Security Constraint). Before a job is scheduled, a timing penalty should be spent to cleanse the state of shared resource after another job is executed.

To satisfy the security constraint, a real-time scheduler dispatches a FT when a context switch occurs from job $j_i \rightarrow j_j$, regardless of source and target task security-levels. Thus, our scheduling and schedulability analysis techniques are not sensitive to the security ordering of tasks. We assume that a FT executes concurrently with a job, whenever the job is dispatched by the scheduler in accordance with our pre-flush techniques, i.e., the remaining execution time of $j_i$ is increased by $C_{ji}$ to allow adequate time to serialize the task execution followed by the FT execution. Every job must complete its execution within $C_{ji}$ time units prior to its deadline to be schedulable since the proposed scheduling algorithm requires that a FT executes non-preemptively to completion at the preemption or completion of every task.

III. SCHEDULING INCORPORATING SECURITY CONSTRAINTS

In this section, we propose a preemptive real-time scheduling algorithm that includes a new pre-flush technique to satisfy the security constraint described in the previous section. We first provide a detailed discussion of the existing pre-flush algorithms introduced in [31] and [32] and then we present our algorithm as an improvement.

![Figure 1. Conventional flush task invocation case](image)

In the existing pre-flush scheme, a FT can be invoked at a time instant at which a job $j_i$ starts its execution after release of the job or resume from any interference caused by higher priority jobs, and shared resource needs to be flushed. The FT executes non-preemptively with highest priority, and this will frequently block the scheduled job $j_j$, increasing its time-to-completion.

Figure 1 illustrates the effect of interleaving execution of the FT with other jobs competing for the processor, characteristic of an existing pre-flush scheduling algorithm. Considering job $j_i$ that begins or resumes execution at time instant $t$, let us assume that an earliest higher priority job $j_j$ will be released at time instant $t_t' > t$. As seen in Figure 1, the higher priority job $j_j$ is preempted by the FT (this task has the highest priority) dispatched to prevent information leakage that may have occurred during the context switch, $j_i \rightarrow j_j$. Thus, a higher priority task $\tau_H$ can be subject to the execution of a lower priority task $\tau_L$ if a FT is invoked between two tasks. Due to this property, it is not straightforward to apply conventional schedulability analysis techniques providing strong analytic capability to this scheduling scheme since most of those are based on the following property:

Definition 2 (Priority Isolation): The execution behavior of higher priority tasks is independent of the execution behavior of lower priority tasks.

For example, standard RTA approach for preemptive FP scheduling is an exact schedulability analysis based on priority isolation only considering interference from higher priority tasks, ignoring the execution behavior of lower priority tasks, to test schedulability of a given task.

While the two documented pre-flush techniques [31-32] violate the principle of priority isolation and the associated schedulability analysis, we propose a new pre-flush technique, which we refer to as flush task reservation for FP scheduling (FTR-FP), accompanied by a new exact schedulability analysis that will conserve the principle of priority isolation. Considering a job $j_i$ that begins or resumes its execution at time instant $t$ and an earliest higher priority job $j_j$ released at time $t_t' > t$, let $e$ be the time instant at which $j_j$ will finish its execution. FTR-FP determines an invocation of a FT in every scheduling (release of a job, or resume of a job from any preemption) and invokes FTs in a more intelligent manner creating three cases based on the value of $t_t' - C_{ji}$:

(a) if $e \leq t_t' - C_{ji}$ (the case of Figure 2 (a)), such that $C_{ji}$ is the WCET of a FT, the scheduler reserves a FT to prevent information leakage from $j_i$ to $j_j$ so that a FT will be invoked at the time instant $e$;

(b) in another case for $t \leq t_t' - C_{ji} < e$ (the case of Figure 2 (b)), a FT is reserved and will be invoked at time instant $t_t' - C_{ji}$; and

(c) if $t_t' - C_{ji} < t$ (the case of Figure 2 (c)), the execution of $j_i$ is suspended, and CPU remains idle in the time space between $t$ and $t_t'$ so that a FT cannot interfere $j_j$.

We refer to these cases as FTR case (a), FTR case (b) and FTR case (c) respectively. Note that if the sum of remaining
execution time of \( j_i \) and \( C_{\mu} \) is smaller than the remaining time space until the deadline of \( j_i \), then \( j_i \) cannot be schedulable. Thus, FTR-FP tests schedulability of \( j_i \) in every scheduling, and then it considers three FTR cases for \( j_i \). We assume that the schedulability of \( j_i \) in Figure 2 is verified.

In FTR case (a), the time interval between \( t \) and \( t' \) is sufficiently large for a FT to execute in the time space without interfering not only the job \( j_i \) but also a higher priority job \( j_j \). To satisfy the security constraint that we defined, a FT is invoked at time instant \( t \) as soon as \( j_i \) completes its execution. In FTR case (b), the time space between \( t \) and \( t' \) is larger than \( C_{\mu} \), but it interferes \( j_j \) if it is invoked at time instant \( e \), which violates priority isolation. Thus, FTR-FP reserves a FT so that it will invoke at time instant \( t' \) and not interfere \( j_j \) but \( j_i \). In FTR case (c), the time space between \( t \) and \( t' \) is smaller than \( C_{\mu} \). Thus, FTR-FP suspends \( j_i \) and let CPU idle in the time space between \( t \) and \( t' \) to prevent \( j_i \) from being interfered from a FT; a FT is not invoked in this case to conserve priority isolation. In both FTR cases (b) and (c), \( j_j \) is preempted and suspended respectively, and it resumes its execution at time instant \( t'' \) after it suffers interference from higher priority jobs including \( j_i \), FTs or CPU idle time; higher priority jobs also can cause CPU idle time due to another higher priority job. FTR-FP iteratively considers three cases for \( j_i \) until \( j_i \) is in the FTR case (a) so that it completes its execution. If \( j_i \) has the highest priority, it executes without any interference, and a FT is invoked as soon as \( j_i \) completes its execution. For the job having the highest priority, \( t' \) is considered as \( +\infty \) since the job does not have higher priority jobs, and it is always in FTR case (a).

FTR-FP is applicable to any FP preemptive scheduling since it invokes a FT on every transition \( j_i \rightarrow j_j \), and it does not depend on a specific priority ordering. Figure 2 illustrates that FTR-FP satisfies the security constraint and conserves the principle of priority isolation for two jobs \( j_i \) and \( j_j \) from tasks \( \tau_i \) and \( \tau_j \) respectively. We now show that FTR-FP also satisfies the security constraint and conserves priority isolation for a task set \( r \) containing more than two tasks based on the following reasoning. As \( j_j \) begins its execution at time instant \( t \), \( j_j \) is the highest priority job in the scheduling queue at time instant \( t \). Also, as an earliest higher priority job \( j_j \) releases at \( t' \), no jobs having higher priorities than \( j_j \) execute in the time space between \( t \) and \( t' \). FTR-FP iteratively finds such time spaces at every scheduling of every task. Since a FT is invoked on every transition \( j_i \rightarrow j_j \) (FTR cases (a) and (b)) or is not invoked if \( j_j \) does not execute (FTR case (c)), FTR-FP always satisfies the security constraint. Also, since \( j_j \) and the FT executing with \( j_j \) always execute in the time space \( t \) and \( t' \), priority isolation is conserved; it is guaranteed that \( j_i \) and the FT do not interfere higher priority job \( j_j \).

IV. SCHEDULABILITY ANALYSIS

This section presents a new schedulability analysis for our new FP preemptive scheduling employing FTR-FP. We extend previous exact schedulability analysis proposed in [39] aiming at calculating the exact number of preemptions to be applicable our scheduling algorithm. We first introduce the definitions and notations used for our schedulability analysis based on those described in [32], and then, we illustrate our schedulability analysis procedure. We also show that our new schedulability analysis can calculate the precise number of preemptions.

A. Definitions and Notations

As FTR-FP guarantees priority isolation, each task \( \tau_i \) only needs to consider interference from higher priority tasks including executions, FTs and CPU idle time for its schedulability test but not those from lower priority tasks. As we assume periodic task model, for the schedulability analysis of \( \tau_i \), we only need to investigate what happens in the time space from 0 to the least common multiple of periods of tasks including \( \tau_i \) and higher priority tasks of \( \tau_i \).

Thus, we define hyperperiod at level \( i \), denoted by \( H_i = LCM \{ T_j \}_{\tau_j, (hep(\tau_j))} \), where \( LCM \{ T_j \} \) is the least common multiple of \( T_j \) and \( hep(\tau_j) \) is a set of tasks having a priority higher than or equal to task \( \tau_j \). \( \tau_i \) releases its job \( \sigma_i \) times in hyperperiod at level \( i \), where \( \sigma_i \) is calculated by

\[
\sigma_i = \frac{H_i}{T_i} = \frac{LCM \{ T_j \}_{\tau_j, (hep(\tau_j))}}{T_i} \quad (1)
\]
Also, \( \tau_{i+1} \) releases its job \( \lambda_i \) times in hyperperiod at level \( i \), where \( \lambda_i \) is calculated by

\[
\lambda_i = \frac{H_i}{T_{i-1}} = \frac{\text{LCM}\{T_j\} \cdot \text{shift}(\tau_i)}{T_{i-1}}
\]  

(2)

We then define \( T_i \)-mesoid as an ordered set describing the state of each job \( j \) of a task \( \tau_i \). \( T_i \)-mesoid consists of three types of time units: a time unit already executed or suspended by CPU idle time, called consumption, a time unit for a FT invocation and a time unit still available. We present consumptions by its cardinal inside brackets (c), with \( c \in \mathbb{N}^+ \). We use another brackets \([r] \), with \( r \in \mathbb{N}^+ \) for WCET of FTs. In addition, we enumerate the sequence of available time units according to the natural numbers. Each of these natural numbers is called availability. Note that the natural numbers of consumptions and FTs present how many availabilities it consumes while that of availability presents its sequence. For example, \( \{(3), [2], 1, 2, (4), [2], 3, 4\} \) is the 15-mesoid containing two sets of consumptions of length 3 and 4 respectively, two FTs where \( C_B \) is 2, and four availabilities represented by 1, 2, 3 and 4 respectively. Since each \( T_i \)-mesoid describes the state of each job, there are \( \sigma_i \), \( T_i \)-mesoid in \( H_i \), which are the sequences of \( T_i \)-mesoid. We define \( L_i^a = \{M_i^{b,1}, M_i^{b,2}, \ldots , M_i^{b,\lambda_i}\} \) as the sequence of \( \sigma_i \) \( T_i \)-mesoid before \( \tau_i \) is scheduled. For example, \( L_i^a = \{(5),[1],[1],2,(2),[1]\}, \{(1,2),(3)[1],3,[3],[1]\}\) is a sequence of \( \sigma_i = 2 \) 11-mesoid where \( C_B = 1 \).

We define the cell of \( L_i^a \) as a set of consecutive available time units between a FT and consumption. With the definition of cell of \( T_i \)-mesoid, the above example is rewritten as

\[
L_i^a = \{(5),[1],[1],2,(2),[1]\}, \{(1,2),(3)[1],3,[3],[1]\}\ 
\]

(3)

where \([m]^j\) denotes the \( m \)th cell in \( j \)th \( T_i \)-mesoid of \( L_i^a \).

The number of time units in the cell \([m]^j\) is denoted by \([m]^j\).

As we can see from the example, the WCET of the first job \( j_1 \) and the second job \( j_2 \) should not exceed 1; otherwise task \( \tau_i \) cannot be schedulable. We call \( L_i^a = \{M_i^{a,1}, M_i^{a,2}, \ldots , M_i^{a,\lambda_i}\} \) the sequence of \( \sigma_i \) \( T_i \)-mesoid of task \( \tau_i \) after \( \tau_i \) is scheduled. \( L_i^a \) is resulted from \( L_i^a \) as the available time units will have been consumed up by executions of \( j_i \) and FTs associated to it in the response time during each mesoid \( M_i^{b,k} \), \( k = 1, \ldots , \lambda_i \) of task \( \tau_i \) after \( \tau_i \) is scheduled. The process of building \( L_i^a \) from \( L_i^a \), and \( L_i^{a+1} \) from \( L_i^a \) respectively will be detailed in the next section. To sum up, for every task \( \tau_i \), we have

\[
\tau_i = \begin{cases}
L_i^a = \{M_i^{b,1}, M_i^{b,2}, \ldots , M_i^{b,\lambda_i}\} \\
L_i^{a+1} = \{M_i^{a,1}, M_i^{a,2}, \ldots , M_i^{a,\lambda_i}\}
\end{cases}
\]  

(4)

B. Schedulability Analysis

Our new schedulability analysis approach for FP preemptive scheduling employing FTR-FP tests schedulability of each task in a top-down manner from the highest priority task to the lowest priority task. As FTR-FP conserves priority isolation, our schedulability analysis only needs to consider interference from higher priority tasks. The analysis conducts two phases for each task: inheritance phase and FTR phase.

- In inheritance phase, \( L_i^a \) of \( \tau_i \) inherits \( L_{i+1}^a \) \( \lambda_i \) times so that all availabilities in \( L_i^a \) are filled with the time units of \( L_{i+1}^a \) in a cyclic manner. In detail, \( (x\times L_{i+1}^a + y) \)th time unit in \( L_i^a \) is filled with \( y \)th time unit in \( L_{i+1}^a \), where \( |L_{i+1}^a| \) is the number of time units in \( L_{i+1}^a \), \( 0 \leq x, y < |L_{i+1}^a| \).

- In FTR phase, availabilities in \( L_i^a \) are consumed by executions of jobs on CPU idle time, and FTs reserved according to the policy of FTR-FP investigating which case each job of \( \tau_i \) is among three FTR cases, and then \( L_i^a \) is constructed for \( L_i^a \) of \( \tau_{i+1} \) if any.

Before we generalize our schedulability analysis process to support for \( n \) tasks, we describe the procedure of schedulability analysis for two tasks, \( \tau_1 \) and \( \tau_2 \), assuming that \( \tau_1 \) has higher priority than \( \tau_2 \). We further assume that \( \tau_1 \) and \( \tau_2 \) have task parameters \((8, 1, 8) \) and \((12, 3, 12) \) respectively, and the WCET of a FT \( C_B = 2 \). As \( \tau_1 \) has the highest priority, schedulability analysis does not conduct inheritance phase for \( \tau_1 \) since it does not have higher priority tasks. In FTR phase, \( \tau_1 \) produces \( C_B \) consumptions, a FT and \( T_1 = (C_B + C_B) \) availabilities in each \( T_i \)-mesoid since each job \( j_i \) of \( \tau_1 \) can consume any availability in \( T_i \) and never be preempted. Thus, \( T_i \)-mesoid of \( \tau_1 \) are described as

\[
\tau_1 = \begin{cases}
L_1^a = \{M_1^{b,1}\} = \{(1,2,3,4,5,6,7,8)\} \\
L_1^{a+1} = \{M_1^{a,1}\} = \{(1),(2,1,2,3,4,5)\}
\end{cases}
\]

(5)

Then, the inheritance phase and the FTR phase are conducted for task \( \tau_2 \). As \( T_2 \) and \( T_1 \) are 8 and 12 respectively, \( H_2 = \text{LCM}(8, 12) = 24 \). In inheritance phase of \( \tau_2 \), as \( \sigma_2 = 2 \) and \( \lambda_2 = 3 \), we construct \( L_2^a \) composed of a sequence of two 12-mesoids by inheriting time units of \( L_2^a \) three times as described above. Based on the mechanism, we have

\[
L_2^a = \{M_2^{b,1}, M_2^{b,2}\} \\
= \{(1),(2,1,2,3,4,5),(1),(2,1,2,3,4,5)\}
\]

(6)

With the definition of cell, \( L_2^a \) can also be written as

\[
\{1,2,3,4,5,6,7,8,9\}
\]

(6)
\[ L^n_2 = \{ \{1\}, \{2\}, \{3\}, \{2\}, \{1\}, \{2\} \}, \{6\} \} \]

\[
\{1\}^2 = \{1, 2, 3, 4, 5\}, (1), [2], [2]_2 = \{5, 6, 7, 8, 9\} \}
\]

We then perform FTR phase satisfying the security constraint and priority isolation. In FTR phase, we investigate each cell sequentially and see which case each job starting or resuming its execution is in among three FTR cases. According to the policy of FTR-FP for each case, time units of each cell are consumed by execution of each job and FTs. In detail, FTR phase is conducted by following policies:

- A single job is released in a T jesoid;
- Consumptions of a job begin with the first availability in a cell;
- Availability is consumed by executions of \( f_j \), FTs and CPU idle time by the policy of corresponding case that each job is in among three FTR cases until the job completes its execution.

Subsequent to the FTR phase of our schedulability analysis, \( L^n_2 \) of \( r_2 \) becomes

\[
L^n_2 = \{ \{1\}, \{2\}, \{3\}, \{2\}, \{1\}, \{2\} \}, \{6\} \}
\]

New consumptions of jobs of \( r_2 \) and consumed time units by FTs are underlined. The first job of \( L^n_2 \) consumes three availabilities from the first availability to the third availability in \( M^{2,3}_2 \), and then a FT follows as the job is in the FTR case (a). The second job of \( L^n_2 \) is in the FTR case (b) first resulting in two consumptions and consumed time units by a FT, and then it is preempted. After preemption from a single consumption and a FT, it resumes its execution and it completes its execution producing a single consumption and a FT, which is in the FTR case (a).

After the completion of both phases, the schedulability of the task \( r_2 \) is verified as schedulabilities of all jobs in the hyper period \( H_l \) are tested. In the above example, \( r_2 \) is schedulable since all jobs in \( r_2 \) complete its execution within its deadline \( D_2 \).

Our schedulability analysis also can calculate the exact number of preemptions by counting preemption whenever a job \( j \) does not complete its execution in a cell; it is in the FTR case (b). For example, the second job in \( L^n_2 \) experiences a preemption by a FT following the job at the third time unit in \( M^{2,3}_2 \), which is in the FTR case (b). Assuming that the cost of preemption denoted by \( C_p \) is 1, the resulted \( L^n_2 \) is rewritten as

\[
L^n_2 = \{ \{1\}, \{2\}, \{3\}, \{2\}, \{1\}, \{2\} \}, \{6\} \}
\]

If the cost of preemption exceeds 2, the second job cannot be schedulable.

---

1: for \( i = 2 \) to \( n \) do
  2: \( \sigma_i \leftarrow \text{RedCMS}(\text{job}(i)) \)
  3: for \( x = 1 \) to \( \sigma_i \) do
  4: Build empty \( A^{\text{FL}}_i \) composing \( L^n_i \)
  5: end for
  6: INHERITANCE-PHASE (\( L^n_{i-1}, L^n_i \))
  7: Build all cells in \( L^n_i \)
  8: if FTR-PHASE (\( L^n_i, \sigma_i \)) = fail then
  9: return unschedulable
  10: end if
  11: Build \( L^n_i \) by duplicating \( L^n_0 \)
  12: end for
  13: return schedulable

---

Figure 3. Algorithm for Schedulability Analysis

1: for \( x = 0 \) to \( i-1 \) do
  2: \( y \leftarrow C_i \)
  3: for \( m = 1 \) to the number of cells in \( A^{\text{FL}}_i \) do
  4: if \( y + C_p \geq \text{[m]} \) then
  5: Produce \( [m] \) consumptions and a FT
  6: \( y \leftarrow y \)
  7: break
  8: end if \( C_p \geq \text{[m]} \) \( y + C_p \) then
  9: Produce \( [m] \) consumptions and a FT
  10: \( y \leftarrow y - \text{[m]} \)
  11: \( j \leftarrow j + C_p \)
  12: else
  13: Produce \( [m] \) consumptions
  14: end if
  15: end for
  16: if \( y > 0 \) then
  17: return fail
  18: end if
  19: end for
  20: return success

---

Figure 4. Algorithm for Inheritance phase (\( L^n_{i-1}, L^n_i \))

1: for \( x = 1 \) to \( \sigma_i \) do
  2: \( y \leftarrow C_i \)
  3: for \( m = 1 \) to the number of cells in \( A^{\text{FL}}_i \) do
  4: if \( y + C_p \geq \text{[m]} \) then
  5: Produce \( [m] \) consumptions and a FT
  6: \( y \leftarrow y \)
  7: break
  8: end if \( C_p \geq \text{[m]} \) \( y + C_p \) then
  9: Produce \( [m] \) consumptions and a FT
  10: \( y \leftarrow y - \text{[m]} \)
  11: \( j \leftarrow j + C_p \)
  12: else
  13: Produce \( [m] \) consumptions
  14: end if
  15: end for
  16: if \( y > 0 \) then
  17: return fail
  18: end if
  19: end for
  20: return success

---

Figure 5. Algorithm for FTR phase (\( L^n_i, \sigma_i \))

Figure 3 presents the procedure of our schedulability analysis using functions of inheritance phase (Figure 4) and FTR phase (Figure 5) for \( n \) tasks in detail. Since both our schedulability analysis and existing one proposed in [33] are extended from the top-down based exact schedulability analysis introduced in [39], to consider a FT invocation, two approaches share the similar framework; they investigate what happens in the interval of hyperperiod at level \( i \) and inherits the schedules of higher priority tasks since they target FP scheduling in inheritance phase (Figures 3 and 4). However, our approach has the following two main advanced parts from the existing one: we support different flush task reservation mechanism preventing information leakages in both directions \( r \rightarrow r \) and \( r \rightarrow r \) while the existing one is only applicable to \( r \rightarrow r \) only, and we can calculate the preemption cost exactly by counting the number of preemption happened exactly, which are implemented in FTR phase (Figures 5).

Since the highest priority task \( r_1 \) is never preempted, the loop starts from the index of the second priority task \( r_2 \) as we conduct analysis towards lower priority task. The schedulability analysis for \( r_1 \) has following steps:
The detail procedures of inheritance phase and FTR phase are presented in Figure 4 and 5 respectively. In inheritance phase, the time units in \( L_i^B \) are filled with the time units in \( L_i^M \), in a cyclic manner as we described previously (Lines 1-5 in Figure 4). In FTR phase, one of the policies of three FTR cases is applied as many times as the number of cells in \( M_i^B \), for a job in \( L_i^B \); FTR cases (a), (b) and (c) are corresponding to Lines 4-7, 8-11 and 12-14 in Figure 5 respectively. Note that preemption cost is added only in FTR case (b) where preemption occurs (Line 11 in Figure 5). If any job does not complete its execution even when all cells are consumed by executions, CPU idle time or FTs, then the job is unschedulable (Lines 16-17 in Figure 5) otherwise it is schedulable (Line 20 in Figure 5).

V. EVALUATION

In this section, we evaluate our analysis technique compared to the conventional approach. We first illustrate the simulation environment of our evaluation, and then we discuss the simulation result of considered techniques. For the main metric of performance, we use the schedulability ratio defined as the number of task sets deemed schedulable by an analysis to the total number of generated task sets.

A. Simulation environment

We randomly generate task sets implemented by our own JAVA program, which are based on the task sets generation method used in [31-33]. Then, we investigate how many tasks are deemed schedulable by each considered schedulability analysis technique (also implemented by JAVA program). Note that as our interest is the schedulability ratio of each considered technique, the evaluation results are independent on our computing capability of simulation environment, e.g., CPU or memory; it only depends on the mathematical calculation of each considered schedulability analysis technique. The base utilization groups \([0.02 + 0.1 \cdot i, 0.08 + 0.1 \cdot i]\) for \( i = 0, \ldots, 9 \), are generated, and each group has 200 task sets. For example, the first group of base utilization has a range from 0.02 to 0.08, and the last group has a range from 0.92 to 0.98. Then, each base utilization group contains task sets whose total sum of task utilizations (i.e., sum of \( C_i/T_i \) of tasks in each task set) belonging to its range of base utilization.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of task, ( N )</td>
<td>( N \in {3, 4, \ldots, 10} )</td>
</tr>
<tr>
<td>Task period, ( T_i )</td>
<td>( T_i \in {50, 100, \ldots, 1000} )</td>
</tr>
<tr>
<td>Task execution time, ( C_i )</td>
<td>( C_i \in {5, 6, \ldots, 50} )</td>
</tr>
<tr>
<td>FT overhead, ( C_{\beta} )</td>
<td>( C_{\beta} \in {0, 1, \ldots, 10} )</td>
</tr>
</tbody>
</table>

Table 1 describes the task parameters used for our evaluation. Each task set can contain at least 3 tasks and at most 10 tasks. Each task has a period \( T_i \in \{50, 100, \ldots, 1000\} \) and an execution time \( C_i \in \{5, 6, \ldots, 50\} \). The deadline of each task \( D_i \) is equal to its period. The values are based on actual resources in actual architecture (e.g. the core i7 or Tegra 3 device) as described in [33]. We generate 2,000 task sets for each value of \( C_{\beta} \), and therefore, 20,000 tasks sets are generated in total.

B. Simulation results

For our evaluation, we consider RM for scheduling algorithm and RTA for schedulability analysis as techniques for comparison since those are the de facto standards techniques which are the optimal preemptive FP scheduling and exact schedulability analysis on uni-processor. We also consider LSF schedulability analysis proposed in [33] for comparison to FTR-FP since, likewise our mechanism, it is also extends the schedulability analysis suggested in [39], and it supports LSF, one of the preemptive scheduling algorithms. Following five schedulability analyses designed for FP preemptive scheduling algorithm on uniprocessor are considered:

- RTA approach for preemptive RM scheduling algorithm not considering a security constraint (RM–RTA),
- RTA approach with the obviously bounded number of FT invocations (the number of FT invocation is \( 2^N_{\text{hpr(i)}} + 1 \), where \( N_{\text{hpr(i)}} \) is the number of higher or equal priority jobs for each task \( T_i \) ) for preemptive RM scheduling, proposed in [31-32] (RM–RTA–ob),
- RTA approach for preemptive LSF scheduling algorithm not considering a security constraint (LSF–RTA),
- LSF schedulability analysis exactly counting the maximum number of FT invocations proposed in [33] (LSF–FTR), and
- the schedulability analysis introduced in Section 4 (RM–FTR–FP).

Note that RM–RTA and LSF–RTA do not consider a security constraint and the rest techniques consider the security constraint. RM–RTA–ob upper-bounds the maximum number of FTs, and LSF–FTR and RM–FTR–FP exactly count that. The performances of all techniques except RM–RTA–ob are corresponding to those of scheduling algorithms respectively since the techniques are the exact schedulability analyses providing exact predictability.

We first investigate how performances of three RM...
schedulability analyses change according to the varying value of FT overhead. Figure 6 shows the changing schedulable ratio of RM-FTR-FP compared to those of RM-RTA and RM-RTA-ob with the varying value of overheads of FT ranging from 0 to 10. As shown in Figure 6, RM-RTA makes most task sets schedulable, and it is not affected by the varying value of FT overhead since it does not consider a security constraint. On the other hands, the schedulable ratio of RM-RTA-ob drops sharply due to its pessimistic bound on the maximum number of FT; it is below 50% for an FT overhead of 10. In the case of RM-FTR-FP that is the proposed technique in this paper, it maintains a high schedulable ratio in spite of the increasing value of FT overhead as it can calculates the exact number of FT; the ratio is approximately 79% even with an FT overhead of 10. Thus, Figure 6 demonstrates that our proposed technique can enhance security while reducing performance degradation.

In our second simulation, we compare our technique (i.e. RM-FTR-FP) with two existing LSF schedulability analyses (i.e. LSF-RTA and LSF-FTR). As both LSF-RTA and RM-FTR-FP are based on the schedulability analysis technique proposed in [39], initially designed to calculate the exact number of preemptions, both mechanisms are also capable to calculate the exact number of preemptions. However, due to the performance gap between LSF and RM scheduling algorithms, LSF-FTR shows limited analytic capability as LSF-FTR is for LSF scheduling algorithm whose performance is inferior to the performance of the RM scheduling algorithm. Figure 7 compares the schedulability ratio of three schedulability analyses: RM-RTA, LSF-RTA and LSF-FTR. As show in Figure 7, LSF-RTA shows constant performance in spite of varying values of FT overhead as it does not consider a security constraint. Even though RM-FTR-FP considers a security constraint, it outperforms LSF-RTA and LSF-FTR as it is based on RM scheduling algorithm.

As shown in Figure 6 and 7 there is about 23% gap of schedulable ratio between RM-RTA and LSF-RTA. Although LSF-FTR shows similar performance degradation rate to RM-FTR-FP for increasing value of FT overhead, its schedulable ratio is limited to below that of LSF-RTA (about 76%). Therefore, as we made FTR to be applicable to any FP scheduling algorithm including RM that is the de facto standard FP scheduling algorithm in uni-processor, we significantly improved the performance of FTR compared to LSF-FTR; we improved the schedulability ratio by 23% for \( C_{PR} = 0 \) and 11% for \( C_{PR} = 10 \) respectively.

As one of the major advantage of our schedulability analysis compared to the existing three techniques in [31-33] is a capability to exactly calculate the number of preemption that occurs in the scheduling of preemptive scheduling algorithm. Note that although the schedulability analysis technique proposed in [33] has the similar framework of ours that is advantageous to calculate the number of preemptions, the method to calculate the preemption was not implemented in [33]. In following evaluation, we measure the average actual number of preemptions incurred by each task sets during 100,000 time units. Since existing three techniques are not capable to calculate the exact number of preemptions, we calculate the number of jobs actually released during scheduling as the upper-bound of the number of preemptions since the number of preemption is no larger than the number of jobs actually released in preemptive scheduling. As we considered RM scheduling algorithm for the comparison with techniques in [31-32] and LSF scheduling algorithm for the comparison with the technique in [33], in the previous two evaluation, we measures the number of jobs actually released and preemptions actually incurred during scheduling of RM and LSF. Table 2 present the average actual number of jobs released and preemptions incurred during 100,000 time units of the scheduling of RM and LSF. Note that two scheduling algorithms are based on the same set of tasks having the same take parameters, and therefore the average actual numbers of jobs actually released thereof during 100,000 time units are the same, e.g., 1234.8. As shown in Table 2, two scheduling algorithm incurs similar number of preemptions; each average actual number of preemptions incurred during RM and LSF is 542.8 and 579.3 respectively. The ratio of jobs to preemptions of RM and LSF are 227.4% and 213.1% respectively. This indicates that our schedulability analysis can be performed much better than the existing three techniques when the cost of preemption increases.

<table>
<thead>
<tr>
<th>Jobs</th>
<th>Preemptions</th>
<th>Jobs/Preemptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>RM</td>
<td>1234.8</td>
<td>542.8</td>
</tr>
<tr>
<td>LSF</td>
<td>1234.8</td>
<td>579.3</td>
</tr>
</tbody>
</table>

Thus, Figure 6 demonstrates that our proposed technique can enhance security while reducing performance degradation.
VI. RELATED WORK

Previous studies dealt with the security problem by combining the real-time requirement and security mechanisms. Xie and Lin considered periodic task scheduling requiring service security which has the varying overhead in relation to the level of service [40–41]. They extended the conventional scheduler to satisfy the real-time requirement and proposed new scheduler while maximizing the level of service security. On the other hand, we focused on a more concrete security problem, i.e., information leakage on shared resources and security solution, i.e., FT, and aimed at improving schedulability of FP scheduling while meeting security constraint that we defined in this paper.

Some works have addressed the information leakage problem within real-time database systems with embedded security constraints [42–44]. Son [43] proposed transaction scheduling and a concurrency control algorithm satisfying security and real-time requirements. This work included definitions of metrics that will measure the extent to which the security requirements and the concurrency control algorithm are satisfied, allowing a trade-off between the security and real-time requirements [42]. Son [44] also proposed a concept of partial security to enable a trade-off between the two functions cited above. While these studies focused on the area of real-time database system, we aimed at developing new real-time scheduling algorithm and schedulability analysis thereof.

There have been works that suggested architectural frameworks to solve problems of intrusion detection in real-time or cyber physical systems [45–48]. The key idea is to develop hardware/software mechanisms using characteristics of real-time or cyber physical systems to protect against security vulnerabilities. While these studies aimed at detecting of intrusion in real-time systems, we focused on preventing information leakage on the shared resources.

As a FT is invoked on the transition between executions of two jobs, the number of FT invocations interfering with a job under analysis is closely related to the number of preemptions occurring during the execution of a job. An existing work suggested a new schedulability analysis that can exactly count the number of preemptions for preemptive FP scheduling algorithms [39]. Thus, the previous work provided many hints regarding the computation of the precise number of FTs in schedulability analysis. However, the approach in [39] did not consider security constraint.

The recent three works in [31–33] are most closely related to our study, which aim at preventing information leakage by conditionally cleansing the state of shared resource in real-time systems. The initial work [31] first proposed the pre-flush mechanism to invoke a FT, the following work [32] extended it to be applicable more general scheduling model, the latest one [33] proposed exact schedulability analysis that can calculate the exact number of FTs. However those studies are not capable to calculate the exact number of preemptions, and show limited analytical performance or applicability to a single direction of information leakage.

VII. CONCLUSION

In this paper, we defined a new security constraint to mitigate the possibility of information leakage arisen on real-time tasks sharing resources in real-time systems. We then proposed a new pre-flush mechanism referred to as FTR-FP relaxing the limitation of FTR [33] that its schedulability analysis is limited to LSF. We show that our techniques are applicable to all FP scheduling including RM which is the de facto scheduling algorithm on uni-processor. We also showed that FTR-FP invokes FT during the context switch in both directions of transition, $\tau_H \rightarrow \tau_L$ and $\tau_L \rightarrow \tau_H$, as it meet security constraint that we defined, FR-FP enhances the system security relative to the conventional approaches [31], [33] that implement FT invocation at context switch time in a single direction, $\tau_H \rightarrow \tau_L$.

Moreover, our schedulability analysis can calculate not only the exact number FTs hindering the execution of a job but also the exact number of preemptions while existing three studies [31–33] focused on bounding the maximum number of FTs only. Our experimental results showed that our mechanism maintains reasonable performance in terms of schedulability and preemption cost.

For the future works, we would like to extend our approach into multi-processor platform. We are also planning to implement our scheduling algorithm incorporating FTR-FP and schedulability analysis techniques into real-time operating system and demonstrate how our approach is working well practically.

REFERENCES

Advances in Electrical and Computer Engineering  Volume 17, Number 2, 2017

132


