Resource Access Protocols

Sheikh Muhammad Adib Bin Sh Abu Bakar
University of Applied Science Hamm-Lippstadt
B.Eng. Electronic Engineering
Lippstadt, Germany
sheikh-muhammad-adib.bin-sh-abu-bakar@stud.hshl.de

***Abstract*—In hard real time system, it is important to ensure every single task not only to be successfully executed but to produce the right value on the right time. Thus, scheduling algorithm play a big role in handling various of task. It is common in real time system environment, some tasks share the same resource. To maintain system predictability, every concurrent operating system should use effective synchronization to guarantee mutual exclusion among competing tasks, which is important especially when safety is a crucial part. This synchronization mechanism spawned another problem in real time system which is *priority invention* that bring many implications. This is why one of the most significant topics in developing a real-time system to handle the problem is resource access protocol. In this paper, I will describe two resource access protocols for uniprocessor that were created under fixed priority assignment and are used to handle a specific problem that arises when two or more tasks share a resource.**

***Index Terms*—real time system, resource access protocol**

I. Introduction

In this paper, two resource access protocols that are used in real-time systems will be discussed. But, before we get into the resource access protocol, we must first comprehend why it exists. Hence, it is necessary to describe it, and I will begin with the definition of the real-time system.

According to [3] - "Real Time System is a system whose response time and delay time are deterministic without uncertainty and non-reproducibility and has an internal configuration that makes the worst value predictable or makes it easy to produce an educated guess value".

So, one of the most important characteristics of a real-time system is its predictability, which ensures that it can give the appropriate output at the right moment. If two or more tasks use the same resource with a defined synchronization mechanism, as indicated in the *background* and *problem description* section, this will become a major issue. This is where the resource access protocol comes in to assist us in resolving the issue.

In the following section, I will go through the basics of the concepts and terms used in this paper so that each protocol's explanation is clear. Then we will go through the major issue with real-time systems with tasks sharing a resource. Following that, the two resource access protocols will be discussed.

II. Background

In this section, we will describe the concept and terms that are dominant in this research paper.

A. Task

According to [4]- "a Task is a sequence of instructions that, in the absence of other activities, is continuously executed by the processor until completion."

Task, thread and process are the basic component in task scheduling. At the level of scope of this paper, the term process and thread have the same definition as task.

An example of a task could be writing or reading a variable's value. Some task have its instance called Job as defined in [4]-"Job is an instance of a task executed on a specific input data".

Next I will explain how tasks are executed.

B. Scheduling Policy and Scheduling Algorithm

In real time system, more than one task that are concurrent, are handled even in system with uni-processor. To realize it, the task are executed in order using a defined algorithm called scheduling algorithm. For that reasons many scheduling algorithm have been proposed such as Earliest Deadline First, The Earliest Deadline Late Server and many more. The task is ordered by assigning their priority through set of predefined criterion called scheduling policy. [1]

Considering that the schedule algorithm handles more than one task, now we will extend the definition of a task to correspond to their state as listed in the list below.

- A task that could potentially execute on the CPU can be either in execution (if it has been selected by the scheduling algorithm) or waiting for the CPU (if another task is executing) [1].
- A task that can potentially execute on the processor, independently of its actual availability, is called an active task [1].
- A task waiting for the processor is called a ready task, whereas the task in execution is called a running task [1].
- All ready tasks waiting for the processor are kept in a queue, called ready queue [1].

Scheduling Algorithms are built to solve a certain problem for a certain environment. Means that they can be classified base on their characteristic. The classification of the algorithm will be explained in the next subtopic.

C. Classification of Scheduling Algorithms

In this subtopic, we will only focus on important classification of scheduling algorithm that are related to the scope of this paper, which is resource access protocols that will be discussed in resource access protocol section.

- Preemptive: running task can be interrupted at any time [1].
- Non-preemptive: a task, once started, is executed until completion [1].
- Static: scheduling decisions are based on fixed parameters (off-line) [1].
- Dynamic: scheduling decisions are based on parameters that change during system evolution [1].
- Off-line : Scheduling algorithm is performed on the entire task set before start of system. Calculated schedule is executed by dispatcher [1].
- On-line : scheduling decisions are taken at run-time every time a task enters or leaves the system [1].
- Optimal : the algorithm minimizes some given cost function, alternatively : it may fail to meet a deadline only if no other algorithm of the same class can meet it [1].
- Heuristic : algorithm that tends to find the optimal schedule, but does not guarantee to find it [1].

In real time system not only the characteristic of algorithm are matter but also the characteristics of task that being handle by scheduling algorithm, such as periodic and aperiodic that will be discussed in the next subtopic.

D. Periodic and Aperiodic Task

A task can be periodic or aperiodic, based on the way it is activated. Their definition is listed below:

- Periodic Task: a task in which jobs are activated at regular intervals of time, such that the activation of consecutive jobs is separated by a fixed interval of time, called the task period [4].
- Aperiodic Task: a task in which jobs may be activated at arbitrary time intervals [4].

The resource access protocol, which we shall cover later, only handles periodic tasks. A periodic task is depicted in "Fig. 1". As previously stated, a scheduling algorithm is used to handle many tasks by executing them in the order of their priority. The constraint they have influences their priority. Following that, I will go through some of the most important task constraints.

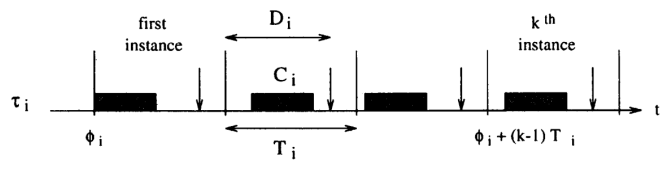

Fig. 1. periodic task [1]

E. Task Constraint

Task constraints are rules that are used by the scheduling algorithm to set priority for each task. There are 3 main type of task constraint, and they are:

- Timing constraint
- Precedence constraint
- Resource constraint

Next we will explain more detail about timing constraint and resource constraint and how they are related to this paper topic. Precedence constraint is not covered in this paper.

1) Timing Constraint: One of the timing constraint that are essential in this paper is deadline. There is two type of deadline:

- Relative Deadline: the longest interval of time within which any job should complete its execution [4].
- Absolute Deadline (of a job): the time at which a specific job should complete its execution [4].

If concurrent tasks are not adequately managed, a task or job's deadline may be missed. As a result, we must make certain that our real-time system is well-predicted. This is why, in order to overcome the problem of system predictability, we have other constraints, such as resource constraint, which will be discussed in the following subsection. We may divide real-time systems into three groups based on their timing constraint, as follows:

- Hard Real Time System - failed to meet the deadline can cause catastrophic event
- Soft Real Time System - failed to meet the deadline only cause the degradation of the system
- Firm Real Time System - failed to meet the deadline only cause the production of useless output

Hard real time is the main focus in this research paper.

2) Resource Constraint: As mentioned before, another important constraint related to this topic is resource constraint. First, we need to understand what resource means in this paper.

According to [1] - "resource is any software structure that can be used by the process to advance its execution. Typically, a resource can be a data structure, a set of variables, a main memory area, a file, a piece of program, or a set of registers of a peripheral device. A resource dedicated to a particular process is said to be private, whereas a resource that can be used by more tasks is called a shared resource. A shared resource protected against concurrent accesses is called an exclusive resource."

Resource constraint that we are focus on is the mechanism that avoid more than one concurrent tasks to access an exclusive resource at the same time. "Fig. 2" shows an example of resource, R that holding the value x and y and two tasks τ_W and τ_D with their critical sections. In general, concurrent tasks are not allowed to access shared resources at the same time to avoid data inconsistency, as illustrated in "Fig. 3". Now, the value of x and y in critical section of τ_D is 4 and 2 instead of 4 and 8.

Various synchronization mechanisms, such as *semaphore*, were devised to prevent this problem. Each critical section

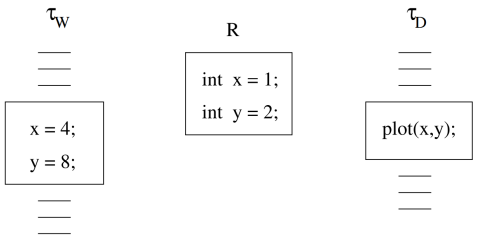

Fig. 2. Two tasks sharing a buffer with two variables. [1]

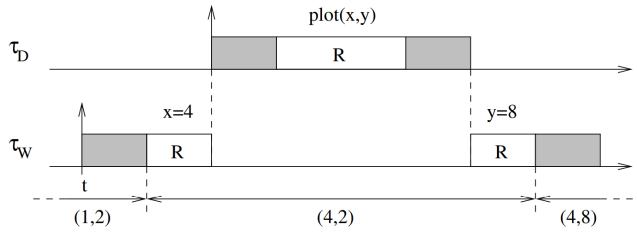

Fig. 3. Example of schedule creating data inconsistency. [1]

in this mechanism must start with wait(s) primitive and end with signal(s) primitive, where s is a binary semaphore, as shown in "Fig. 4", and the state of a task may be extended thanks to this mechanism, as shown in "Fig. 6". If another job is presently using the resource, the task will be unable to access it until the signal(s) is executed. As seen in "Fig. 5", this approach can assure data consistency. For the sake of simplicity, we will refer to exclusive resources as "resources" throughout this paper.

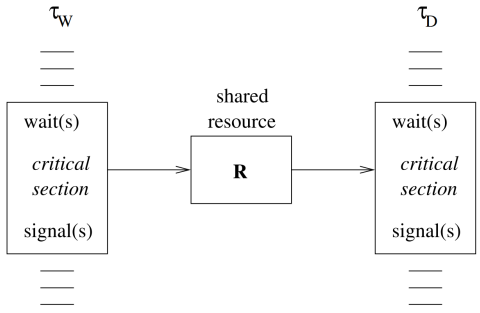

Fig. 4. Structure of two tasks that share a mutually exclusive resource protected by a semaphore. [1]

F. Schedulability

Before we contemplate implementing a resource access protocol, we must ensure that all tasks are schedulable or that a schedule on the set of tasks is feasible. A schedulability test can be used to accomplish this. The schedulability test will not be discussed in this paper because all tasks are

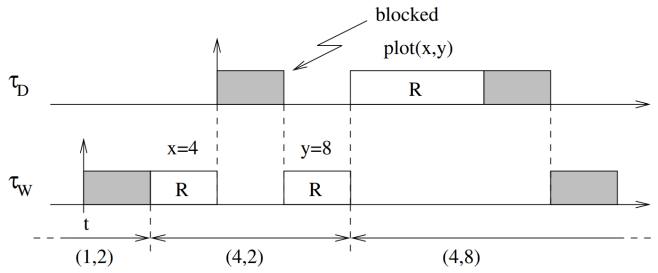

Fig. 5. Example of schedule when the resource is protected by a semaphore.. [1]

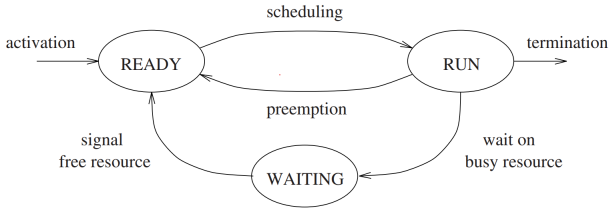

Fig. 6. Waiting state caused by resource constraints. [1]

already considered schedulable. I refer the interested reader to reference [1] for details.

In the following section, we will look at an issue that developed as a result of the suggested synchronization mechanism, which has an impact on the predictability of a real-time system.

III. Problem Definition: Invention Priority

In theory, a set of task that are schedulable will be executed upon on its arriving time and preempted if a task with higher priority arrive. That will not be always true if we apply resource constraint. The problem arrive when a scheduler want to preempt a lower task that is currently accessing a resource, R and want to execute higher priority task that also want to use the resource. Since the lower priority task did not yet produce signal(S) primitive where S is the binary semaphore, the higher priority task will be blocked. This phenomenon called priority invention. The implication from this phenomenon is that it will cause unbounded delay on execution of task with higher priority and reduce the predictability of the system because the highest priority task could miss its deadline. The priority invention is illustrated in "Fig. 11" where τ_1 have the highest priority.

Here is where we need resource access protocol to make some adjustment to the scheduler so that the implication of invention priority could be reduced. We are going to discuss this solution in the next sections.

IV. Resource Access Protocol

Now we will discuss 2 main resource access protocol for scheduling algorithm handling periodic tasks which are:

- Non-Preemptive Protocol
- Highest Locker Priority

The following list are the important notation that I will use in explaining each protocol. The main idea about those protocols is to reduce the priority invention problem, so that the blocking time experienced by the task that have the highest priority by lower priority task will also be discussed for each protocol.

- B_i denotes the maximum blocking time task τ_i can experience [1].
- $z_{i,k}$ denotes a generic critical section of task τ_i guarded by semaphore S_k [1].
- $Z_{i,k}$ denotes the longest critical section of task τ_i guarded by semaphore S_k [1].
- $\delta_{i,k}$ denotes the duration of $Z_{i,k}$ [1].
- $z_{i,h} \subset z_{i,k}$ indicates that $z_{i,h}$ is entirely contained in $z_{i,k}$ [1].

V. Non-Preemptive Protocol

In this section, I will explain about Non-Preemptive Protocol.

A. Definition

This protocol is a simple protocol and named non-preemptive(NP) because it avoids any interruption on running task τ_j that accessing a resource R_k that guarded by, S_k. To reduce total blocking time experienced by the task τ_i that have the highest priority, this protocol just increase the priority of the task τ_j that currently accessing the resource R_k, so that the task will not be interrupted and can be done much faster.Without this protocol, the task that highest priority τ_i will interrupt the task τ_j that currently accessing the resource R_k even though the task cannot access the resource because it already guarded by S_k. The scheduler then switch back to the task before to finish its process, this switch context process could cause longer blocking time experienced by the highest priority task. After the taskτ_j finish accessing the resources, its priority will be back to its nominal priority P_j. These situations can be compared through "Fig. 7" and "Fig. 8". So, the priority of the task $\tau_j$that currently accessing the resource is

$$p_j(R_k) = \max_h\{P_h\} \qquad [1] \quad (1)$$

B. Blocking time computation

The total of critical section of lower priority task τ_j blocking higher priority task τ_i is

$$\gamma_i = \{Z_{j,k} | P_j < P_i, k = 1, ..., m\} \qquad [1] \quad (2)$$

From my understanding, the value of k could be more than one if the task that have the highest priority want to access more than one resource and each resource are currently accessed by the task that have lower priority. Hence, in the total duration, the highest priority task is blocked is

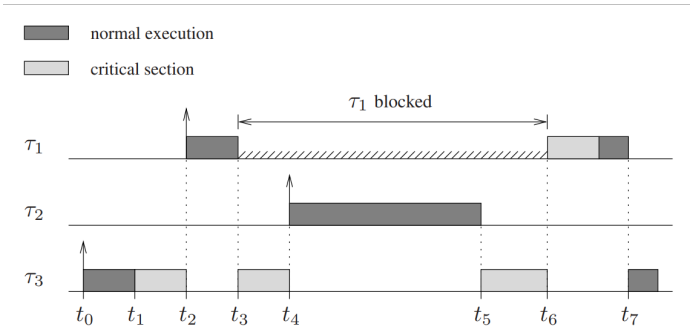

Fig. 7. An example of priority inversion.. [1]

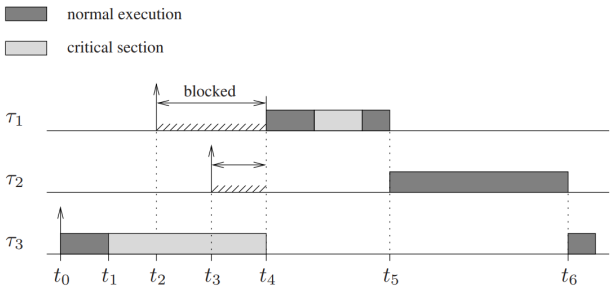

Fig. 8. Example of NPP preventing priority inversion. [1]

$$B_i(R_k) = \max_{j,k}\{\delta_{j,k} - 1 | Z_{j,k} \in \gamma_i\} \qquad [1] \quad (3)$$

The value of $\delta_{j,k}$is reduced one unit time because the task that have lower priority, τ_j could block the task with the highest priority, τ_i from accessing a resource only and only if the task τ_j arrived at least one unit time earlier than τ_i.

C. Implementation Strategies

As state by [2] - "All commercial RTOSs have a means for beginning and ending a critical section. Invoking this Scheduler operation prevents all task switching from occurring during the critical section. If we write our own RTOS, the most common way to do this is to set the Disable Interrupts bit on our processor's flags register. The precise details of this vary, naturally, depending on the specific processor.". Means that, this resource access protocol is nothing more than avoiding interrupt on running task.

Base on the model that I made using UPPAAL, it is true that the result in terms of the longest blocking time experienced by the highest priority task in both case, disabling interrupt without NPP and enabling interrupt with NPP is the same.

Using the same model, I found that it is possible that a higher priority task could miss deadline without applying NNP while interrupt is enabled. The model can be downloaded from https://github.com/Adib6637/UPPAAL to see the simulation.

D. Sample Model

In this section, I will explain the sample model of NPP that I made. For this sample model, I use a clock that can display time and date. Considering the task of displaying date,τ_d have higher priority than task displaying time,τ_t and they share the same resource which is Display unit,R_d, the scheduler will preempt the task τ_t whenever the task τ_d is ready as shown in "Fig. 9".

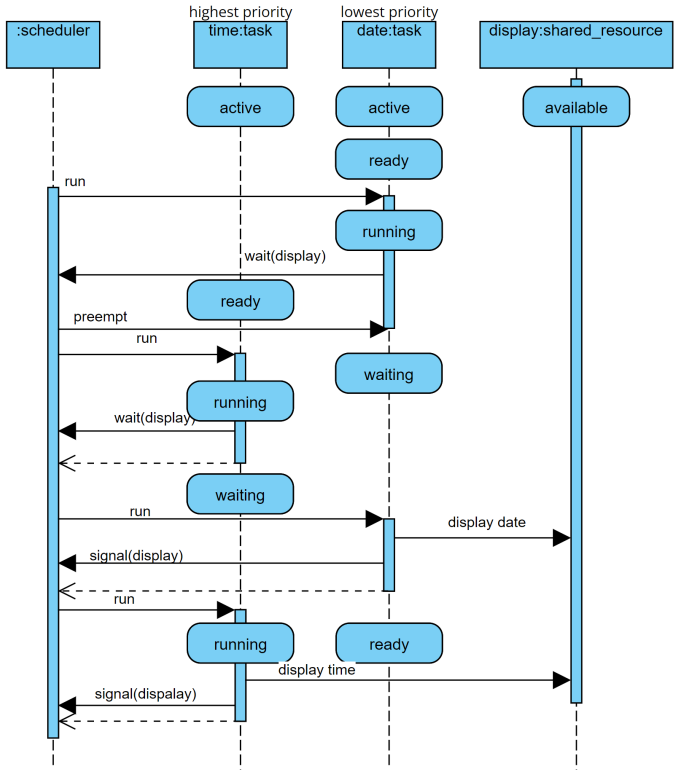

Fig. 9. Sample Model without Non-Preemptive Protocol [2]

By applying NNP as shown in "Fig. 10", the task τ_d cannot be preempted in event the task τ_t is ready because the dynamic priority of the task τ_d is arisen to the highest and back to its nominal value when the task τ_d leave its critical section.

E. Problem Arise

As shown in "Fig. 11", this protocol will block the highest priority task τ_1 even though the task will not access the resource because the priority of the task τ_3 have been increase to the maximum.

Back to our sample model, this situation can be illustrated by adding a new task that didn't use the resource,R_d like alarm task, τ_a as shown in "Fig. 12".

This problem could be solved by the protocol in the next section, which is Highest Locker Priority (HLP) protocol.

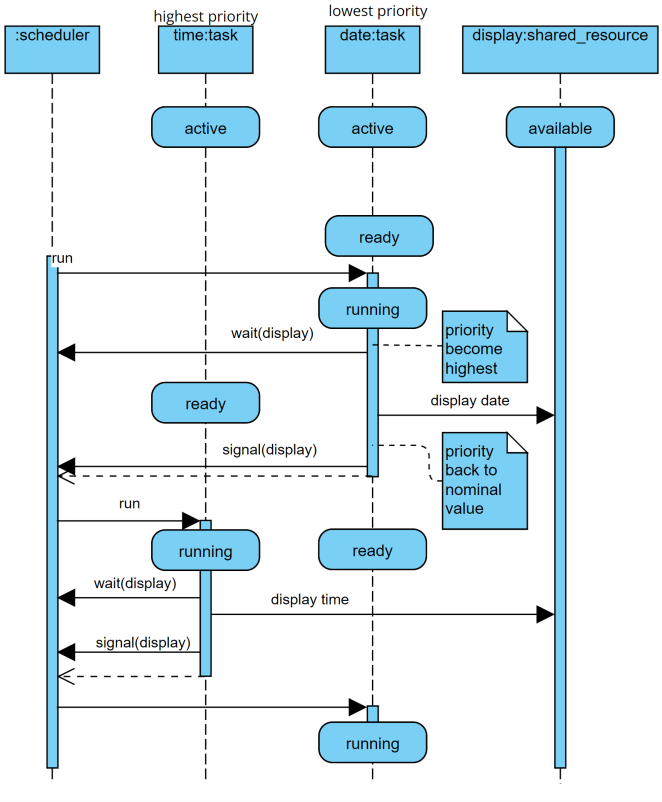

Fig. 10. Sample Model with Non-Preemptive Protocol

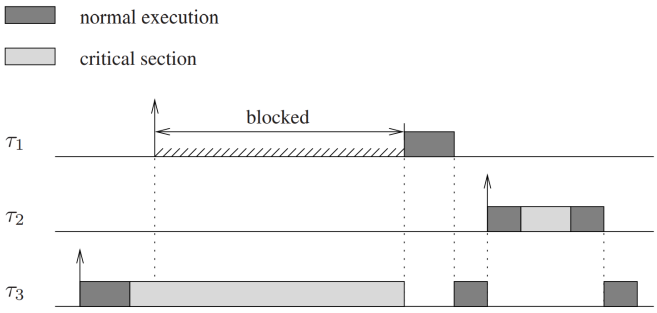

Fig. 11. Example in which NPP causes unnecessary blocking on τ_1 [1]

VI. Highest Locker Priority

In this section, I will explain about Highest Locker Priority protocol.

A. Definition

Highest Locker Priority (HLP) is the improvement of the previous protocol to allow the highest priority task τ_i that doesn't use resource R_k to interrupt the lower priority task τ_j that use the resource, R_k by limiting the raised priority of the task τ_j. So,

$$p_j(R_k) = \max_h\{P_h|\tau_h \text{ uses } R_k\} \qquad [1] \quad (4)$$

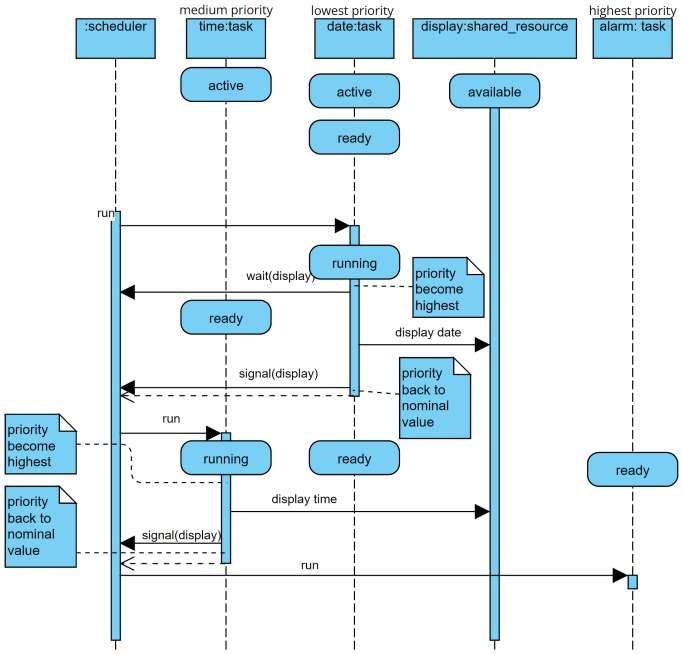

Fig. 12. Example in which NPP causes unnecessary blocking on alarm task, τ_a

.

where the priority of task, τ_j that are currently accessing the resource, R_k is increased to the maximum only among the tasks that want to access the resource. This dynamic priority then set back to its nominal value P_j when the task leave its critical section. The maximum raised priority of a task τ_j is called priority ceiling $C(R_k)$ and computed off-line. The maximum priority $C(R_k)$ of the tasks sharing R_k is the computed online such

$$C(R_k) \stackrel{def}{=} \max_h \{P_h | \tau_h \text{ uses } R_k\} \qquad [1] \quad (5)$$

Since the priority of lower priority task τ_j is raised as soon as the task entering R_k, this protocol also known as Immediate Priority Ceiling. This protocol can be visualized as in "Fig. 13" where task τ_1 have the highest priority and task τ_3 is the first task arrived.

B. Blocking Time Computation

The total of critical section of lower priority task τ_j blocking a higher priority task τ_i is reduced by adding a new parameter as shown below. Means that the highest priority task that did not want to use the resource could not be blocked by the lower priority task that are currently using the resource.

$$\gamma_i = \{Z_{j,k} | P_j < P_i \text{ and } C(R_k) \geq P_i\} \qquad [1] \quad (6)$$

According to [1] - "Under HLP, a task τ_i can be blocked, at most, for the duration of a single critical section belonging to the set γ_i".

As shown in "Fig. 13", τ_i can be blocked at maximum once, means that

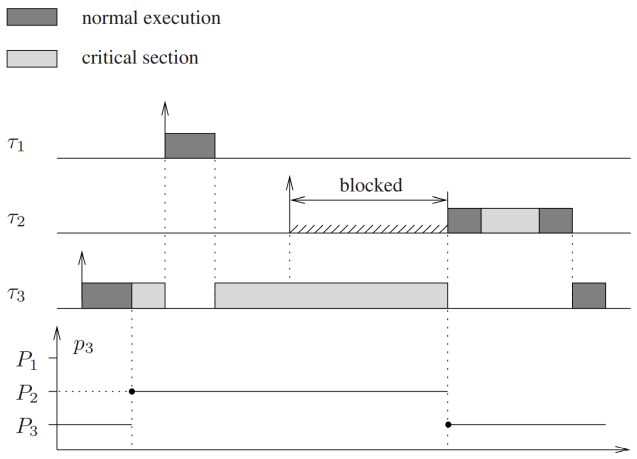

Fig. 13. Example of schedule under HLP, where $p3$ is raised at the level $C(R) = P_2$ as soon as τ_3 starts using resource R [1]

$$B_i(R_k) = \max_{j,k} \{\delta_{j,k} - 1 | Z_{j,k} \in \gamma_i\} \qquad [1] \quad (7)$$

We need to minus one unit of time because the lower priority task τ_j need to access R_k at least one unit of time earlier than τ_i to block it.

C. Implementation Strategies

According to [2] - "Fewer RTOSs support the Highest Locker Pattern more than the basic Priority Inheritance Pattern. The implementation of this pattern is fairly straightforward, with the addition of priority ceiling attributes in the Shared Resource. When the mutex is locked, it must notify the Scheduler to elevate the priority of the locking task to that resource's priority ceiling."

D. Sample Model

In this section, I will explain the sample model of HLP that I made. Our aim now is to solve the problem in "Fig. 12" by applying HLP. As the result, the dynamic priority of task that currently accessing the resource, R_d, which is the display is risen to the highest only among the task that want to access the resource, R_d. This solution is illustrated in "Fig. 14" . So, the task τ_t can be preempted by the highest priority task, the alarm task, τ_a even though the task is currently accessing the resource R_d.

E. Problem Arise

As claimed by [1] -"despite the fact that this algorithm improves the previous algorithm, it still could produce some unnecessary blocking. This algorithm block a task at the time it attempts, before it actually require a resource [1]. If a critical section is contained only in one branch of a conditional statement, then the task could be unnecessarily blocked, since during execution it could take the branch without the resource."

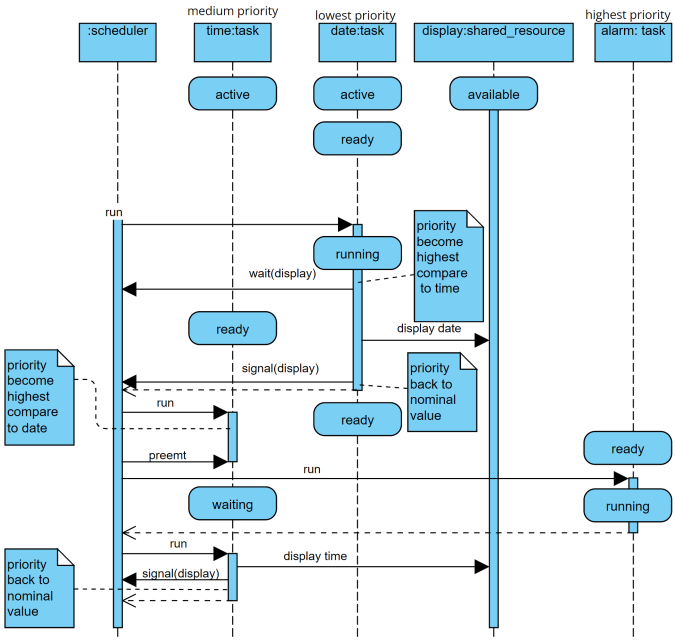

Fig. 14. Sample model with HLP

VII. Conclusion

As the conclusion, Resource access protocol is used in order to maintain the predictability of a real time system by reducing the blocking time experienced by the highest priority task and the most important things is that it help the highest priority task to complete before its deadline to avoid any catastrophic consequence. There are more resources access protocol that can found in [1] that improves the pervious protocol.

Acknowledgment

I would like to express my very great appreciation to Prof. Dr. Stefan Henkler for his valuable and constructive suggestions during the planning and development of this paper. His willingness to give his time so generously has been very much appreciated.

References

[1] Giorgio C. Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications*. Real-time systems series. Springer, 3rd ed edition. OCLC: ocn741541202.

Task state definition, resource definition, scheduling algorithm classification, diagrams, equations and notation and problems due to resource access protocol implementation.

[2] Bruce Powel Douglass. *Real-Time Design Patterns: robust scalable architecture for Real-time systems*. The Addison-Wesley object technology series. Addison-Wesley.

Protocol Implementation strategies.

[3] IEEE. *IEEE Std 2050-2018: IEEE Standard for a Real-Time Operating System (RTOS) for Small-Scale Embedded Systems.* IEEE. OCLC: 1053697231.

Definition of Real time system

[4] IEEE, IEEE TCRTS, and IEEE Computer society. Terminology and notation definitions related to a system.

Task definition, job definition, deadline definition and task characteristics.

[5] Intel Corporation. Real-time systems overview and examples.

The need of real time system and Real-Time System Components

[]