Resource Access Protocols

1st Sheikh Muhammad Adib Bin Sh Abu Bakar
Hochshule 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, either statically or dynamically. It is common in real time system environment, some tasks share the same resource, which is one of the complicated part in concurrent operating system. Any concurrent operating system should utilize proper synchronization to assure mutual exclusion among competing activities, in order to ensure the predictability of the system, which is important in oder to produce reliable system especially when safety is a crucial part. This is why resource access protocol one of the important topic in building a real time system. In this research paper, various of resource access protocol for uni-processor that are developed under fixed priority assignment will be explained. So, it will be clear for us which protocol should be implemented based on the environment that we are working on. To ensure this paper is easy to understand, I also include basic information about real time system before going deep into resource access protocol.**

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

I. Introduction

Before we start, first we need to make sure the reader have basic knowledge about Real Time System. Hence it is essential to explain a basic term and concept that will be used in this research paper. We will start with the definition with the real time system since this protocol are for real time system.

Real Time System - 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 [1].

So, according to Intel [2]-For a real-time system to be capable of real-time computing, it must satisfy two requirements:

- Timeliness: The ability to produce the expected result by a specific deadline.
- Time synchronization: The capability of agents to coordinate independent clocks and operate together in unison.

A real-time system must have the ability to not only process data in a defined, predictable time frame but also ensure that critical tasks, such as safety-related workloads, are completed prior to less critical tasks [2].

So, the main key that distinguish between the real time system and non-real time system is the predictability of the system. We will discuss more about predictability in upcoming subtopics.

Identify applicable funding agency here. If none, delete this.

II. Task Scheduling

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

A. Task, Thread and Process

The following definition are according to [4]

- Task: a sequence of instructions that, in the absence of other activities, is continuously executed by the processor until completion.
- Job: an instance of a task executed on a specific input data.
- Thread: a task sharing a common memory space with other tasks.
- Process: a task with its private memory space, potentially generating different threads sharing the same memory space.

B. Scheduling Policy and Scheduling Algorithm

When a single processor has to execute a set of concurrent tasks – that is, tasks that can overlap in time – the CPU has to be assigned to the various tasks according to a predefined criterion, called a scheduling policy and the set of rules that, at any time, determines the order in which tasks are executed is called a scheduling algorithm [5].

Considering that the schedule algorithm handles more than one task, now we will extend the definition of a task correspond to their state according to [5]

- 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).
- A task that can potentially execute on the processor, independently on its actual availability, is called an active task.
- A task waiting for the processor is called a ready task, whereas the task in execution is called a running task.
- All ready tasks waiting for the processor are kept in a queue, called ready queue.

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 research paper.

- Preemptive: running task can be interrupted at any time.
- Non-preemptive: a task, once started is executed until completion
- Static: scheduling decisions are based on fixed parameters (off-line) .
- Dynamic: scheduling decisions are based on parameters that change during system evolution.
- Off-line : Scheduling algorithm is performed on the entire task set before start of system. Calculated schedule is executed by dispatcher.
- On-line : scheduling decisions are taken at run-time every time a task enters or leaves the system.
- 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.
- Heuristic : algorithm that tends to find the optimal schedule but does not guarantee to find it

D. Periodic and Aperiodic Task

The following definition are according to [4]

- 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.
- Aperiodic Task: a task in which jobs may be activated at arbitrary time intervals.

The topic of this paper is limited to periodic tasks. This can be illustrated in figure 1.

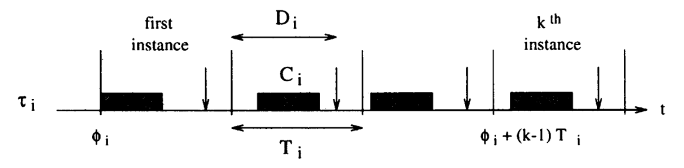

Fig. 1. periodic task [5]

E. Type of task constraint

Task constraints 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. 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].

With time constraint we can characterize real time system into three category :

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

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

2) Resource Constraint: Another important constraint related to this topic is resource constraint.

According to [5] - 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.

Many shared resources do not allow concurrent access by competing processes in order to ensure data consistency, and instead require mutual exclusion. This means that if another task is within R modifying its data structures, a task cannot access R. R is referred to as a mutually exclusive resource in this scenario. A piece of code executed under mutual exclusion constraints is called a critical section. This can be illustrate in figure 2 where R is the mutually exclusive resource, τ_1 and τ_2 are two distinct task. In figure 3 we can see the data inconsistency without mutual exclusion

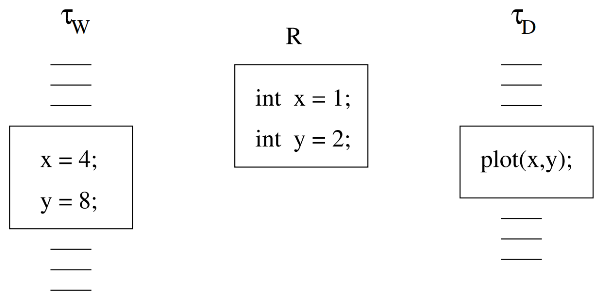

Fig. 2. Two tasks sharing a buffer with two variables. [5]

Semaphore is one of the very well known example of mechanism for synchronization provided by operating system. Each critical section must begin with wait(S) primitive and end with signal(s) primitive where s is a binary semaphore as shown in figure 4. If a task want to access a resource

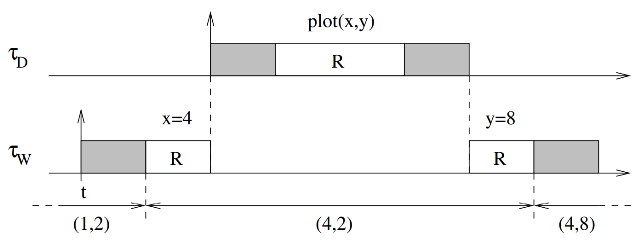

Fig. 3. Example of schedule creating data inconsistency. [5]

that currently accessed by another task, that task must wait until signal(S) is executed by the previous task as illustrated in figure 5 and the states of the task are shown in figure 6

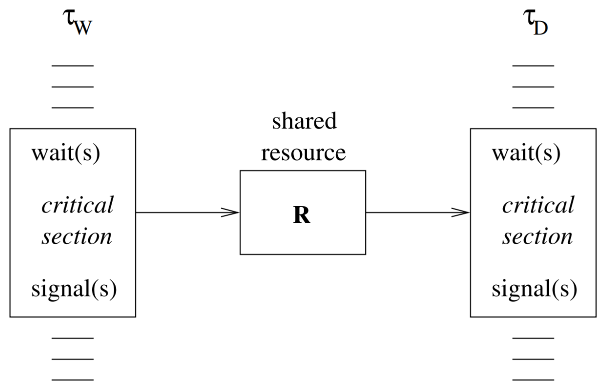

Fig. 4. Structure of two tasks that share a mutually exclusive resource protected by a semaphore. [5]

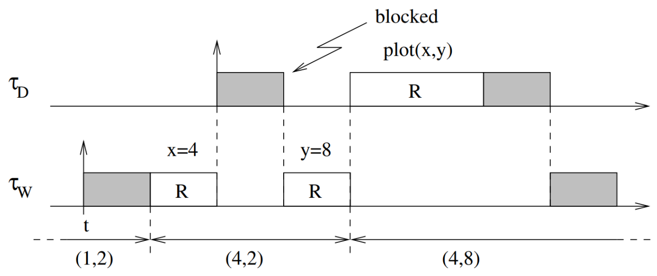

Fig. 5. Example of schedule when the resource is protected by a semaphore.. [5]

F. Schedulability

Before we considering to implement resource access protocol it is important to make sure that all task are schedulable or a schedule is feasible on a set of task. This can be done through scheduability test. This paper will not explain about scheduability test since all task already considered as schedulable. More detail about scheduability test can be found in [5].

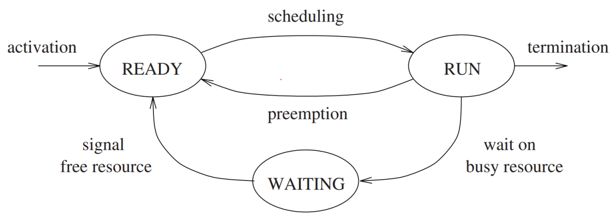

Fig. 6. Waiting state caused by resource constraints. [5]

III. Invention Priority

In theory, a set of task that are schduable 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 currently accessing a mutually exclusive resource R and want to execute higher priority task that also want to use the resource. Since the lower priority task didn't yet produce signal(S) primitive where S is the binary semaphore, the higher priority task will be block. 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. The priority invention is illustrated in figure 9 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 reduce which we going to discuss in incoming sections.

IV. Resource Access Protocol

Now we will discus 4 main resource access protocol for periodic task which are:

- Non-Preemptive Protocol
- Highest Locker Priority
- Priority Inheritance Protocol
- Priority Ceiling Protocol

The first protocol is for non-preemptive-able schedule the the last three are for . The following list are the important notation that I use in explaining each protocol. The main idea about those protocol is to avoid priority invention, so that the blocking time experienced by the task that have highest priority by lower priority task could be reduced. Before I explaining the protocols, here are some notation that need to be understood.

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

- σ_i denotes the set of semaphores used by τ_i.
- $\sigma_{i,j}$ denotes the set of semaphores that can block τ_i, used by the lower-priority task τ_j .
- $\gamma_{i,j}$ denotes the set of the longest critical sections that can block τ_i, accessed by the lower priority task τ_j . That is,
$$\gamma_i = \{Z_{j,k} | P_j < P_i \text{ and } (S_k \geq \sigma_{i,j})\}$$
- γ_i denotes the set of all the longest critical sections that can block τ_i. That is,
$$\sigma_i = \bigcup_{j:P_j > P_i} \gamma_{i,j}$$
-

V. Non-Preemptive Protocol

A. Definition

This protocol is a simple protocol and named non preemptive because it avoid 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 highest priority, this protocol just increase the priority of the task τ_j that currently accessing the resource τ_j, 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 figure 7 and 8 . So, the priority of the task $\tau_i$that currently accessing the resource is

$$p_i(R_k) = \max_h \{P_h\}$$

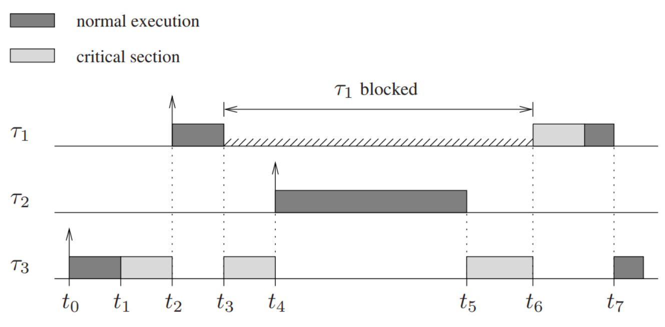

Fig. 7. An example of priority inversion.. [5]

B. Blocking time computation

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

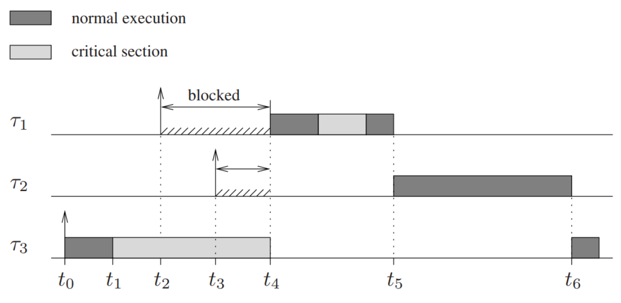

Fig. 8. Example of NPP preventing priority inversion. [5]

$$\gamma_i = \{Z_{j,k} | P_j < P_i, k = 1, ..., m\}$$

Hence the total duration highest priority task is blocked is

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

C. Problem Arise

As shown in figure 9, this protocol will block highest priority task τ_1 even though the task will not access the resource.This problem could be solved in the next protocol which is Highest Locker Priority (HLP) protocol.

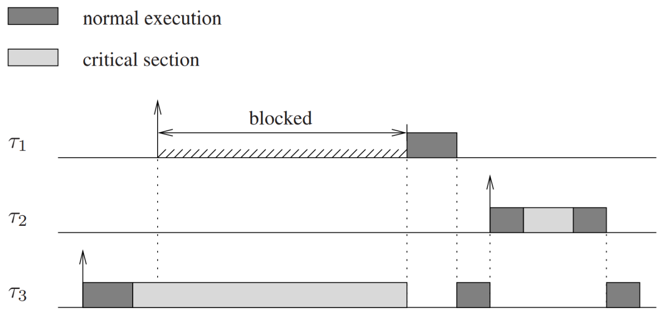

Fig. 9. Example in which NPP causes unnecessary blocking on τ_1 [5]

VI. Highest Locker Priority

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 task τ_j. So,

$$p_i(R_k) = \max_h \{P_h | \tau_h \text{ uses } R_k\}$$

This dynamic priority then set back to its nominal value P_i when the task leave its critical section. The maximum raised priority of 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

$$C(R_k) \overset{def}{=} \max_h \{P_h | \tau_h \text{ uses } R_k\}$$

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 visualize as in **??** where task τ_1 have higest priority and task τ_3 is the first task arrive

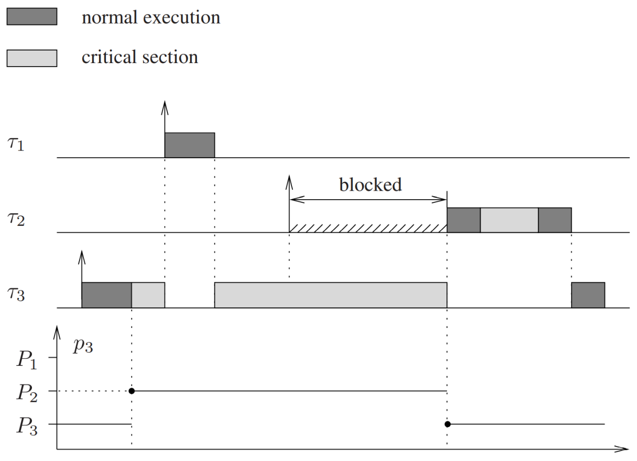

Fig. 10. Example of schedule under HLP, where $p3$ is raised at the level $C(R) = P_2$ as soon as τ_3 starts using resource R [5]

B. Blocking Time Computation

So, total of critical section of lower priority task τ_j blocking higher priority task τ_i is reduced by adding new parameter as shown below.

$$\gamma_i = \{Z_{j,k} | P_j < P_i \text{ and } C(R_k) \geq P_i\}$$

According to [5] - 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 and this theorem is proved by contradiction, assuming that τ_i is blocked by two critical sections, $z_{1,a}$ and $z_{2,b}$. For this to happen, both critical sections must belong to different tasks (τ_1 and τ_2) with priority lower than P_i, and both must have a ceiling higher than or equal to P_i. That is, by assumption, we must have

$$P_1 < P_i \leq C(R_a)$$
$$P_2 < P_i \leq C(R_b)$$

Now, τ_i can be blocked twice only if τ_1 and τ_2 are both inside the resources when τ_i arrives, and this can occur only if one of them (say τ_1) preempted the other inside the critical section. But, if τ_1 preempted τ_2 inside $z_{2,b}$ it means that $P_2 > C(R_b)$, which is a contradiction. Hence, the theorem follows.

As shown in figure 10, τ_i can be block at maximum once, means that

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

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

C. Problem Arise

Despite the fact that this algorithm improve the previous algorithm, it still could produce some unnecessary blocking. This algorithm block a task at the time it attempt, before it actually require a resource [5]. It also says that - 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.

This protocol was improved by Priority Inheritance protocol that will be expalain in the next section.

VII. Priority Inheritance Protocol

A. Definition

$C(R_k) \overset{def}{=} \max_h \{P_h | \tau_h uses R_k\}$
11

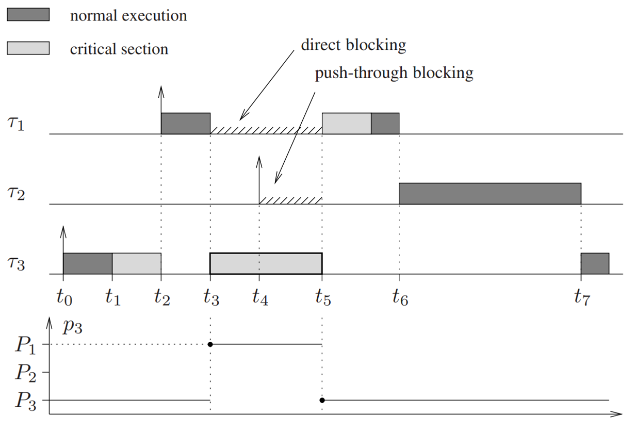

Fig. 11. Example of Priority Inheritance Protocol. [5]

12
13

B. Blocking Time computation

$\sigma\frac{dir}{i,j} = \sigma_i \cap \sigma_j$
$\sigma\frac{pt}{i,j} = \bigcup_{h:P_h > P_i} \sigma_h \cap \sigma_j$
$\sigma_{i,j} = \sigma\frac{dir}{i,j} \cup \sigma\frac{pt}{i,j} = \bigcup_{h:P_h > P_i} \sigma_h \cap \sigma_j$
$\gamma_{i,j} = \{Z_{j,k} | P_j < P_i and R_k \in \sigma_{i,j}\}$
$\sigma_i = \bigcup_{j:P_j > P_i} \gamma_{i,j}$
$B\frac{l}{i} = \sum_{j=i+1}^{n} \max_k \{\sigma_{j,k} - 1 | C(S_k) \geq P_i\}$
$B_i = \max_{j,k} \{\delta_{j,k} - 1 | Z_{j,k} \in \gamma_i\}$
$\gamma_1 = \{\}$

C. Problem Arise

1) Chained Blocking: 14

2) Deadlock: 15

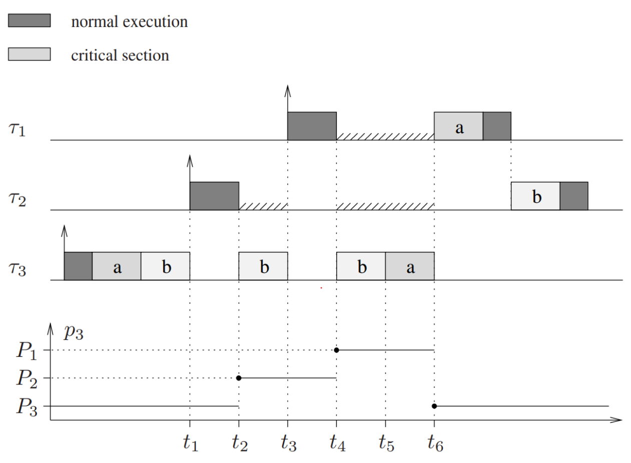

Fig. 12. Priority inheritance with nested critical sections. [5]

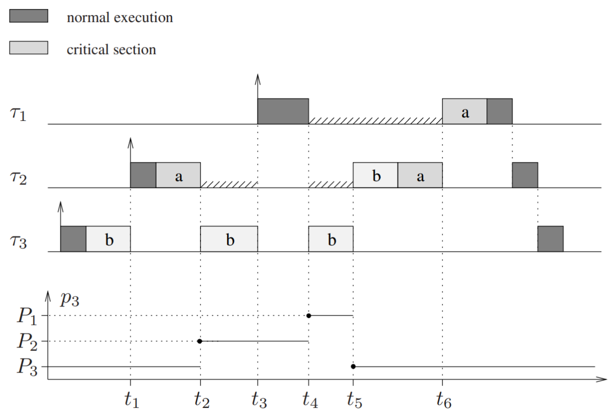

Fig. 13. Example of transitive priority inheritance. [5]

VIII. Priority Ceiling Protocol

$$C(S_k) \overset{def}{=} \max_i\{P_i|S_k in \sigma_i\}$$
$$p_j(R_k) = max\{P_j, \max_h\{P_h|\tau_h isblockedonR_k\}\}$$
$$C(S_A) = P_1$$
$$P_0 \geq P_i > C^*$$
$$\gamma_i = \{Z_{j,k}|(P_j < P_i)andC(R_k) \geq P_i\}$$

IX. Compare

X. Conclusion

Acknowledgment

I, Sheikh Muhammad Adib

References

[1] ”IEEE Standard for a Real-Time Operating System (RTOS) for Small-Scale Embedded Systems,” in IEEE Std 2050-2018 , vol., no., pp.1-333, 24 Aug. 2018, doi: 10.1109/IEEESTD.2018.8445674.
[2] “Real-time systems overview and examples,” Intel. [Online]. Available: https://www.intel.com/content/www/us/en/robotics/real-time-systems.html. [Accessed: 04-May-2022].
[3] “Characteristics of real-time systems,” GeeksforGeeks, 04-May-2020. [Online]. Available: https://www.geeksforgeeks.org/characteristics-of-real-time-systems/. [Accessed: 04-Apr-2022].

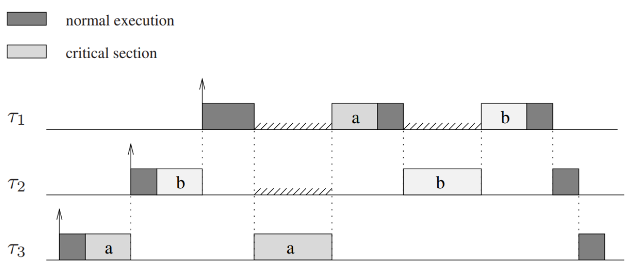

Fig. 14. Example of chained blocking. [5]

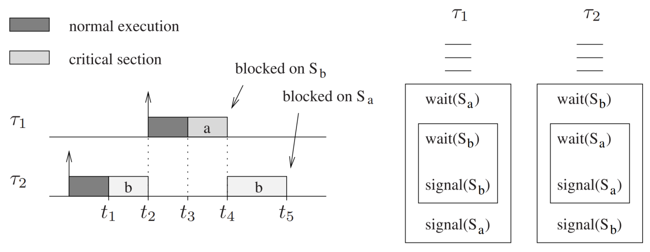

Fig. 15. Example of deadlock. [5]

[4] “Terminology and notation,” Terminology and Notation —. [Online]. Available: https://cmte.ieee.org/tcrts/education/terminology-and-notation/. [Accessed: 04-Apr-2022].
[5] G. C. Buttazzo, ”Hard real-time computing systems: Predictable scheduling algorithms and applications”. New York: Springer, 2011.
[6] L. Sha, R. Rajkumar and J. P. Lehoczky, ”Priority inheritance protocols: an approach to real-time synchronization,” in IEEE Transactions on Computers, vol. 39, no. 9, pp. 1175-1185, Sept. 1990, doi: 10.1109/12.57058.
[7] Rajkumar, Raj and Juvva, Kanaka and Molano, Anastasio and Oikawa, Shuichi ”Resource Kernels: A Resource-centric Approach to Real-Time and Multimedia Systems” 1997 Proceedings of SPIE - The International Society for Optical Engineering. 3310. 10.1117/12.298417.
[8] Nakajima, Tatsuo and Tokuda, Hideyuki, ”Real-Time Synchronization in Real-Time Mach”, 1994