Department of Electrical and Computer Engineering
University of Maryland at College Park

ENEE 642: Software Systems Implementation
Spring 2000, Prof. D. Stewart

Handout #7
Introduction to Real-Time Scheduling Theory

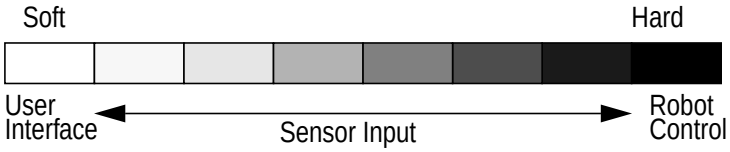
A real-time system is one in which the correctness of the computations not only depends upon the logical correctness of the computations but also upon the time at which the result is produced.

- A LATE answer is a WRONG answer.
- Real-time DOES NOT mean fast!
- Real-time DOES mean predictable, deterministic execution.
- Real-time systems DO react with the real-world, and thus failures can be catastrophic.

A real-time system can be classified as either *hard* or *soft*. The distinction, however, is very fuzzy.

Hard Real-Time: Missing a deadline results in the failure of performance degradation of the system.

Soft Real-Time: The system will properly perform as long as deadlines are met most of the time. Missing a few deadlines will not affect the system.



Most complex real-time systems use preemptive multitasking. To ensure that timing constraints are met, priorities must be assigned according to different rules than typical desktop systems. In this handout, we review some real-time scheduling algorithms, along with the advantages and disadvantages of doing so.

Real-time scheduling algorithms can be

- Fixed priority scheduling; where priority = *K*
  - task priorities specified by user
  - rate monotonic
- Dynamic priority scheduling; where priority = *f(t)*
  - earliest deadline first algorithm
  - minimum laxity first algorithm
- Mixed priority scheduling; where priority = *K+f(t)*
  - maximum urgency first algorithm

Scheduling Theory Definitions

Transient Overload: The state of a real-time system when at least one task will fail because of a lack of CPU time available.

Guaranteed Task: A task is said to be guaranteed if it will always meet its deadlines, even in a transient overload situation.

Task Set: The set of all tasks which may execute on the same processor.

Critical Task Set: The set of guaranteed tasks. It must be a subset of the task set

Task Utilization: percentage of CPU utilization required by a task, in the worst case. Utilization of task i is
$U_i = C_i/T_i$, where T_i = period of task i, and C_i = worst case execution time of task i.

Total utilization: $U = \Sigma U_i$

Schedulable bound: maximum worst-case utilization for which a task set will not miss any deadlines. The value is scheduler dependent; the maximum value is 100%.

1. SCHEDULING ALGORITHMS

In this section, we take a look at various scheduling algorithms, and show whether or not each task meets its deadline.

Assume the following two tasks, P_1 and P_2; how do we schedule them?

- T_1 = 50 msec, C_1 = 25 msec, U_1 = 50%
- T_2 = 100 msec, C_2 = 40 msec, U_2 = 40%
- U = U_1 + U_2 = 90%

The deadline for each task is assumed to be the start of next period. If we use a highest-priority-first algorithm, there are two possibilities

- Case 1: Priority P_1 > Priority P_2
- Case 2: Priority P_1 < Priority P_2

Let us consider each case separately. Case 1 is shown in Figure 1, while Case 2 is shown in Figure 2. In the second case, the task set is not schedulable, even though total utilization is less than 100%.

2. FIXED PRIORITY SCHEDULING: RATE MONOTONIC ALGORITHM

The Rate-Monotonic Algorithm (RMA) is a procedure for assigning fixed priorities to tasks to maximize the possibility that the task set is schedulable. The basic algorithm is to do the following:

- Assigned fixed priorities proportionally to the tasks frequency
- Highest frequency task gets highest priority
- Lowest frequency task gets lowest priority

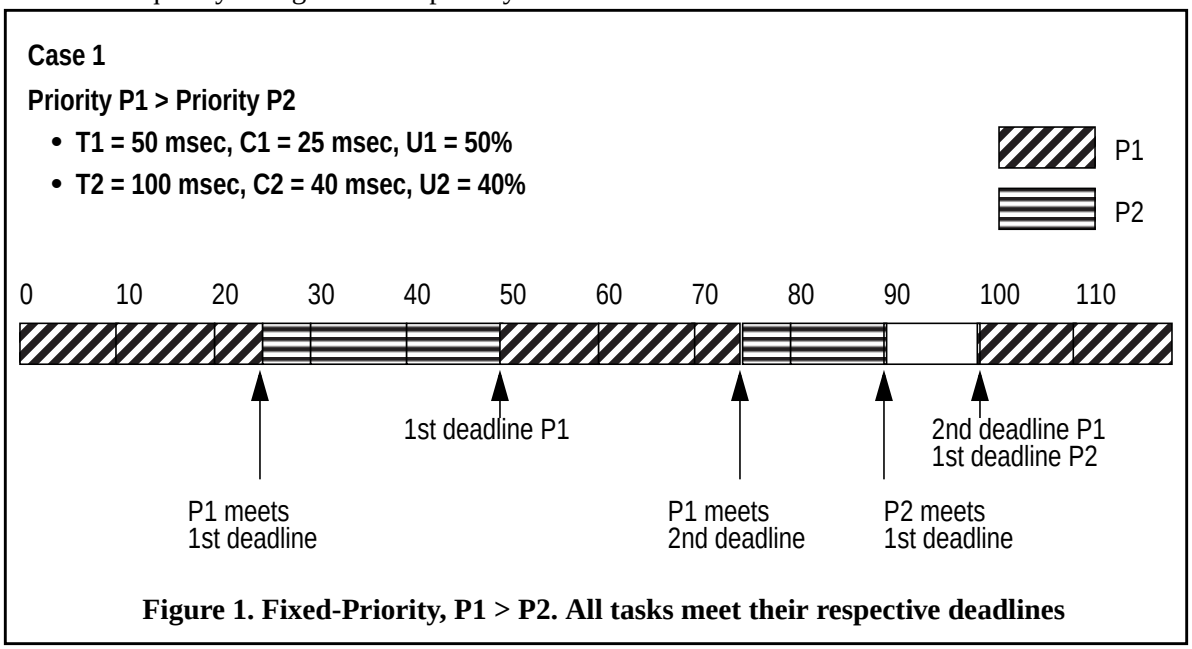


Figure 1. Fixed-Priority, P1 > P2. All tasks meet their respective deadlines

RMA is an optimal *fixed* priority algorithm. This means that if a task set cannot be scheduled using the rate monotonic algorithm, then it cannot be scheduled using *any fixed* priority algorithm.

3. Problems with Rate Monotonic Algorithm

One major problem for the RMA is that the schedulable bound is less than 100%. Specifically, it has been proven that the worst case schedulable bound is as low as 69%, while the average schedulable bound is 88%. This means that even though the total processor utilization is less than 100%, if that utilization is higher than the schedulable bound, then there is no guarantee that the task set will meet all deadlines.

Example, Case 3 in Figure 3 is a modification of Case 1

- T_1 = 50 msec, C_1 = 25 msec, U_1 = 50%
- T_2 = 75 msec, C_2 = 30 msec, U_1 = 40%
- $U = U_1 + U_2$ = 90% (same as Case 1)

Using rate monotonic algorithm, assign Priority P_1 > Priority P_2

Even though utilization = 90% is the same as Case 1, this task set cannot be scheduled using any FIXED priority assignment.

4. Dynamic Scheduling

An alternative to fixed priority scheduling is *dynamic priority scheduling*, where priority is a function of time. There exist the following optimal dynamic priority algorithms:

- Earliest Deadline First (EDF)
  - earliest deadline gets highest priority
- Minimum Laxity First (MLF)
  - laxity = deadline - present - cpu_still_needed; smallest laxity gets highest priority.

If a task set cannot be scheduled with one of these algorithms, then it cannot be scheduled with *any* algorithm. The schedulable bound in each of these cases is 100%. Thus, as long as total utilization is less than 100%, then task set is schedulable. As an example, consider the task set in Case 3, that was not schedulable

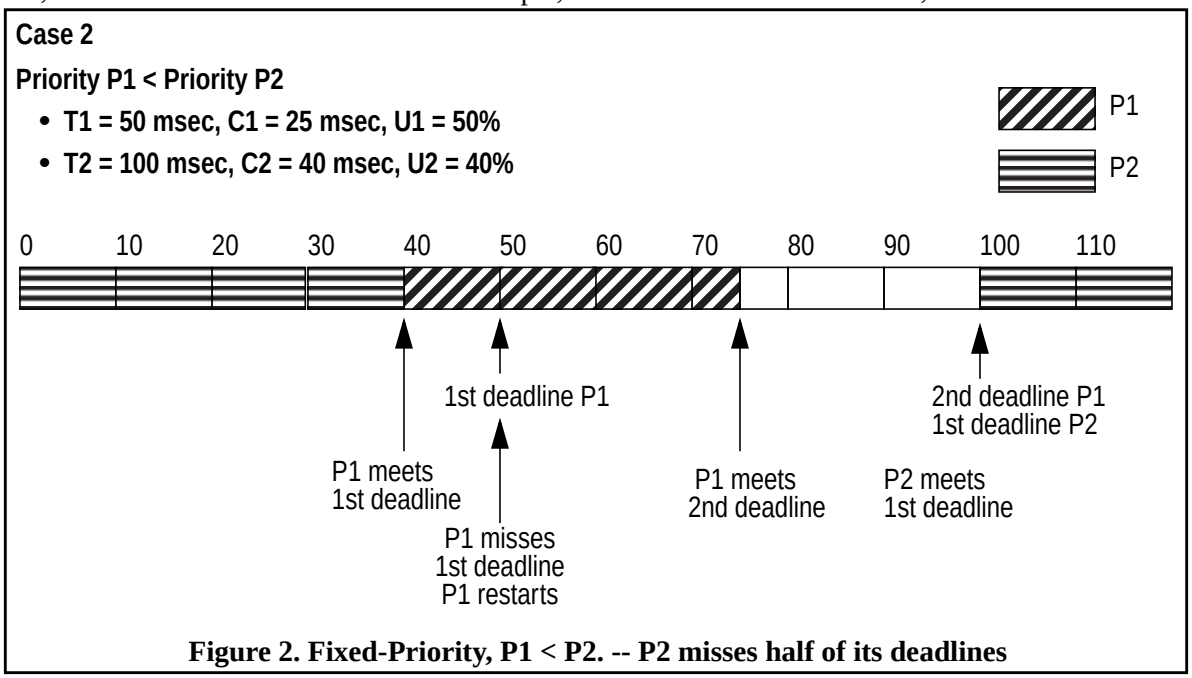


Figure 2. Fixed-Priority, P1 < P2. -- P2 misses half of its deadlines

using a fixed-priority scheduling algorithm. In Case 4, we see that the task set is indeed schedulable when using EDF.

5. Advantages/Disadvantages of Fixed and Dynamic Scheduling

Rate Monotonic Algorithm

Advantage: A critical set can be specified, thus tasks can be selectively chosen during a transient overload

Disadvantage: Worst case schedulable bound is 69% (average case 88%)

Disadvantage: Deadline Failures are difficult to detect

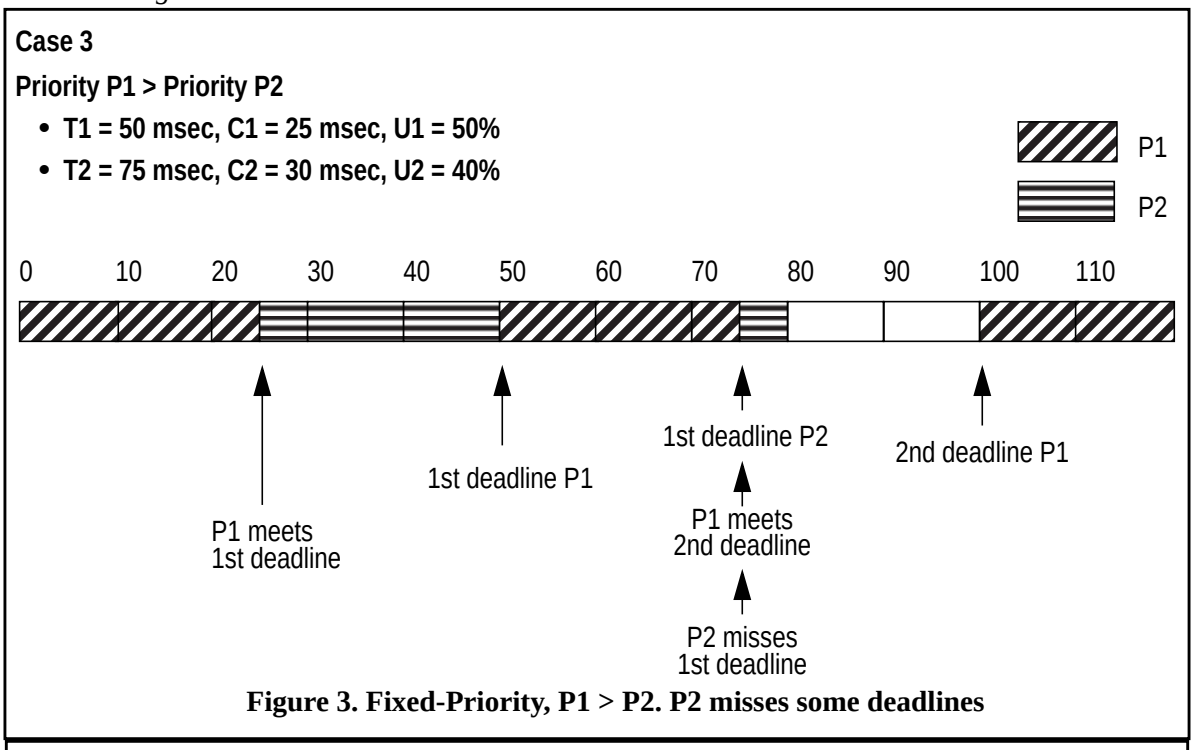


Figure 3. Fixed-Priority, P1 > P2. P2 misses some deadlines

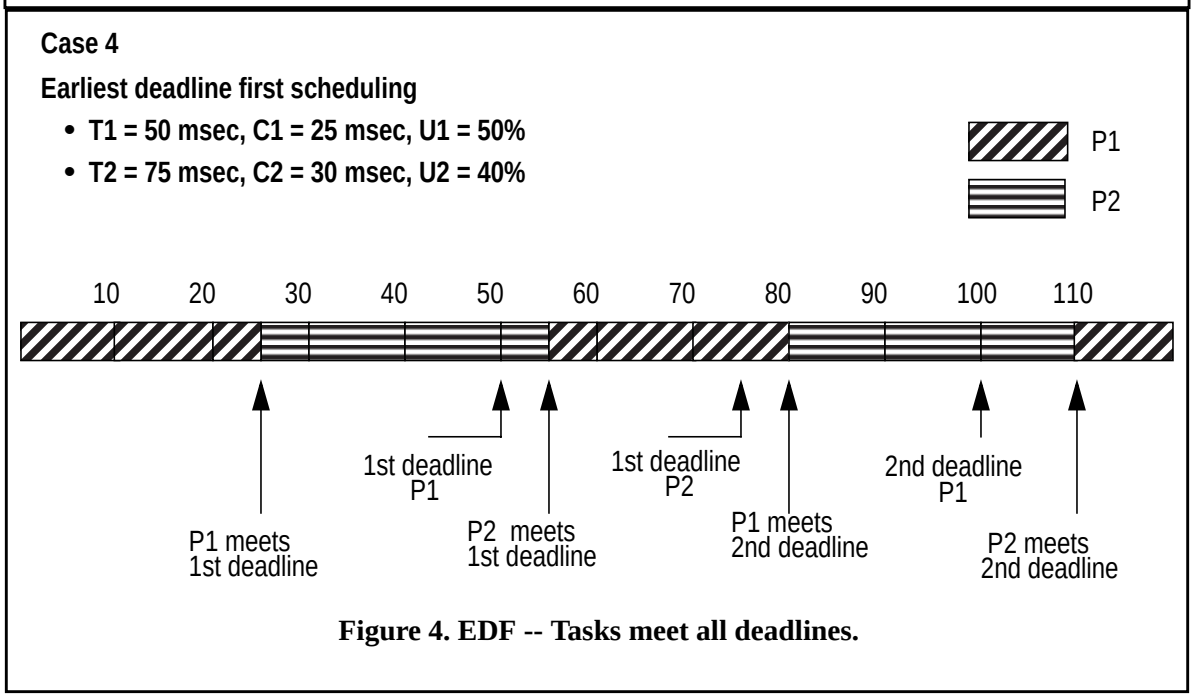


Figure 4. EDF -- Tasks meet all deadlines.

Earliest Deadline First and Minimum-Laxity First:

Advantage: Schedulable bound is 100%

Advantage: Deadline Failures can be detected

Disadvantage: Cannot specify a critical task set, thus no control of which tasks fail during a transient overload

Disadvantage: Require more overhead during a reschedule

There also exist mixed-priority scheduling algorithms, that combine some of the advantages of the fixed and mixed priority algorithms.

6. COMMUNICATION AND SYNCHRONIZATION

As with processes in non-real-time systems, real-time tasks must often share resources, using the same mechanisms, such as shared memory, semaphores, messages, and locks. Sometimes, a real-time task must block until the shared resource is available. What happens if a high-priority process needs the resource immediately in order to not miss a deadline, but a lower-priority process is holding the resource?

We define *Priority Inversion* as the blocking of a high priority task due to a lower priority task locking a shared resource. For example, Figure 5 illustrates the priority version problem. The task set consists of 3 tasks:

- PH: A high priority task
- PM: A medium priority task
- PL: A low priority task

For this example, we are not concerned with their period or execution time.

Assume a resource R1 shared between PH and PL, and PL acquires the resource first.

There have been many solutions proposed for addressing problems such as priority inversion. For example:

- Priority Inheritance Protocol
  - Lower priority tasks inherit higher priority when holding a shared resource
- Priority Ceiling Protocol

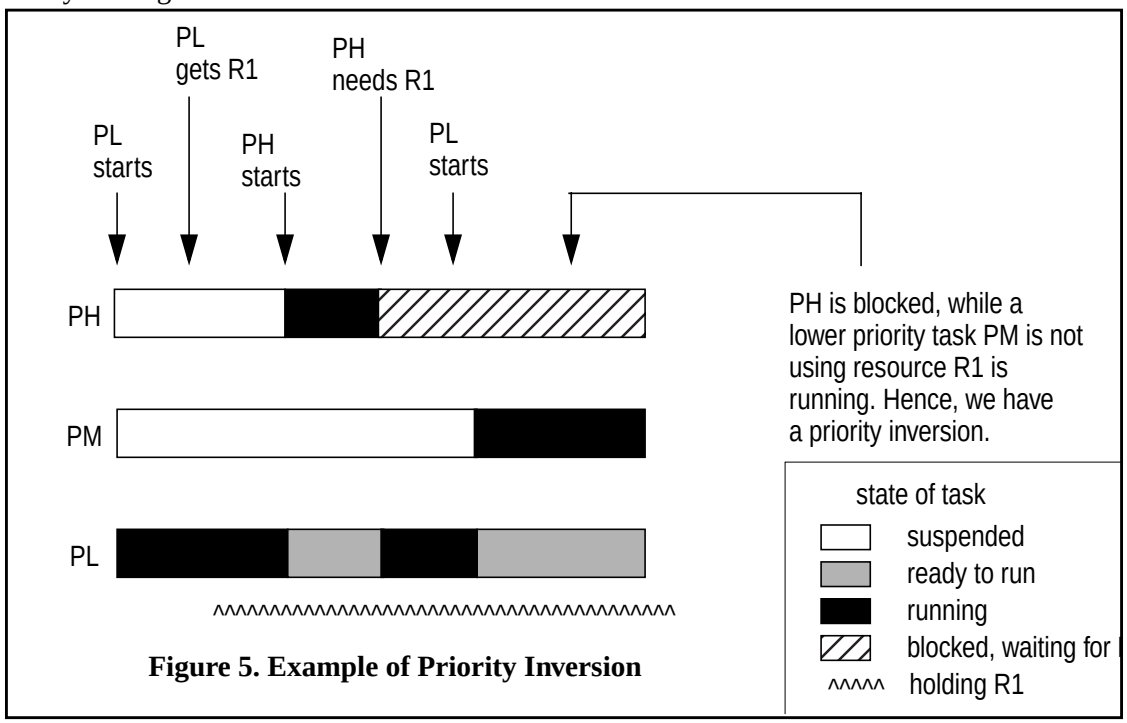


Figure 5. Example of Priority Inversion

  - Maximum blocking time can be predetermined
  - Used mostly with fixed priority scheduling
  - Deadlock free
- Resource scheduling
  - Schedule resources as well as tasks
  - Various algorithms proposed
  - Often used in client/server environments.
- Prioritized Message Passing
  - Higher priority messages are treated first
  - Messages can specify maximum response time

Figure 6 illustrates the priority inheritance solution to the priority inversion problem. Although this solution has some merits, it is usually used because the method is deadlock-prone, and it is difficult to analyze since the number of times a task can be preempted due to resource usage is not bounded.

A better solution is the priority ceiling protocol. Theoretically it provides the desired result. In practice, however, it is expensive to implement, as significant setup time and overhead is required.

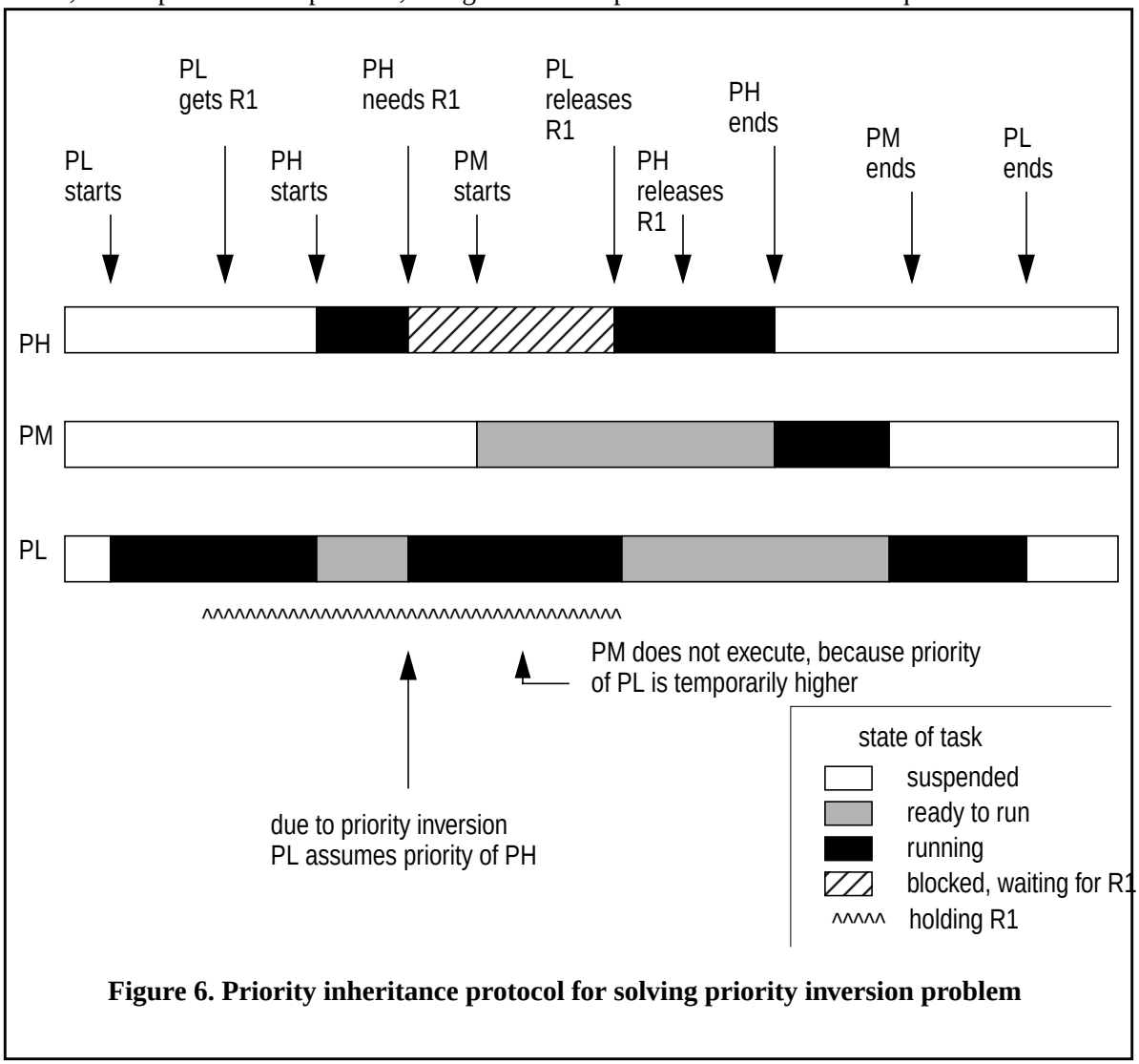


Figure 6. Priority inheritance protocol for solving priority inversion problem