Scheduling Algorithms for Project Management: a Real-Time Operating System Type Approach

Mădălina M. MIERLĂ and Dana M. VÎLCU

***Abstract*—An important task in project management, once the activities to be executed have been clearly defined, is to distribute those activities according to their respective constraints. Changes may occur during the development of the project, relative to the parameters characterizing the activities, or to the execution units (individuals or teams), or to the necessary resources. In those cases, a reassessment of the feasibility of the activities is required, which in turn could induce their redistribution. In large-scale projects, it is desirable that this (re)distribution of activities, their (re)planning, can be done automatically. The present article proposes such a planning automation, by adapting some scheduling algorithms used by real-time operating systems to order their tasks. Our idea is validated through simulations that we describe in the last part of the paper. We also give justifications for the need to use several algorithms, as well as suggestion to deal with some restricted cases.**

***Index Terms*—project management, task model, real-time operating system, multiprocessor scheduling.**

I. Introduction

An important step in project management is the planning of activities. Numerous project management applications have been developed, among which we mention the most used and more well-known ones: Asana [1], Trello [2], Taskworld [3], Monday.com [4]. They allow the rigorous definition of projects and of activities within them, of teams and the assignments of activities to the team members. Ways of communication among the team members are also established, as well as monitoring of the achievement of the tasks.

A crucial problem in project management is the scheduling of the activities whenever this is required. An introduction to project scheduling can be found in [5]. Also, [6] gives a very good presentation of the actual approaches.

The approach we propose in this article is, to our knowledge, new. It states that, for the planning of activities within projects, multiprocessor scheduling algorithms from the theory of operating systems should be used. In order to explain the matching of the concepts from the two apparently distant fields, a presentation of the canonical model of tasks on which the scheduling theory is developed is made in Section II. This section may seem to be too elementary to operating systems specialists; however, our paper is rather addressed to project managers and to developers of project management applications, and this explains the necessity of this section. Section II presents the preliminaries in order to understand Section III, where a parallelism is provided between the involved concepts, and also Section IV, dedicated to the modeling of activity planning based on multiprocessor scheduling algorithms.

Simulations were carried out, by adapting the algorithms to consider the execution of activities by humans and not by processors. Section V is about the adaptation of the algorithms, and Section VI reinforces the justification for the need to implement several algorithms, through the results of a simulation.

The task model used in the simulation and the implemented application did not take into account all cases of task constraints. Therefore, in Section VII, reasons are presented for using the same approach, with the necessary adaptations, for other task models, or for other types of constraints.

We conclude by highlighting the main ideas and results of the proposed approach.

II. The Canonical Task Model in the Scheduling Theory

In the following, we will use the terms referring to the description of tasks as they are presented in [7], in what is called the canonical model of tasks.

The role of using elements of the scheduling theory and specific algorithms is, on the one hand, to obtain information regarding the feasibility of carrying out the proposed activities with human resources and in the desired time frame. On the other hand, an automation of the planning process can be achieved by proposing multiple solutions, allowing to the human expert to decide on the most appropriate one.

Realizing the planning requires defining the characteristics of all activities, the data structures involved and the necessary resources. In this section we present notions and general aspects of the scheduling theory from real-time operating systems. This presentation allows us to highlight the analogy of the notions involved in the two fields. Thus, the results and the algorithms known from the scheduling theory could be adapted to achieve planning activities in the project management.

Recall that an operating system is software that manages hardware and software resources of one or more processors, and provides a set of services to system users [8]. The kernel is the core of an operating system, which, by the other, controls the scheduling of processes to achieve multitasking [9]. The component of the kernel that assures the distribution of the tasks to the system CPU(s) is the scheduler [10]. In our approach we are interested by the

M. M. MIERLĂ was an undergraduate student at Military Technical Academy "Ferdinand I", Bucharest, Romania, and developed the simulation application as part of her BSc thesis. She is now a graduate student at the University of Pitești, Romania, Faculty of Communications, Electronics and Computers (email: maria.mierla_338@student.upit.ro).

D. M. VÎLCU is a scientific researcher of Military Technical Academy "Ferdinand I", Bucharest, Romania (e-mail: dana.vilcu@mta.ro).

Symmetric MultiProcessor operating systems that balance tasks dynamically between CPUs.

Real-time operating systems are characterized by having time as key parameter, and are used, for example in industrial process-control systems. Often there are hard deadlines that tasks active in the system must meet. If the action absolutely must occur at a certain moment (or in a certain range), we have a hard real-time system [10]. The algorithms used by real-time operating system schedulers are deeply studied for their efficiency and feasibility for different models of tasks.

From the point of view of the operating systems, a task represents an execution unit, defined and treated for the achieving of a specific goal. The canonical model of tasks includes the following features (see Fig. 1):

- R: the time moment of the first task execution request;
- C: the maximum execution cost of the task, if it has a processor assigned, and its execution is not interrupted (the execution cost is expressed as the number of processor cycles required to execute the task);
- D: the critical acceptable delay for the execution of the task (the deadline, after which, if the task has not been completed, it is considered to have failed);

P: the period (for the periodic tasks).

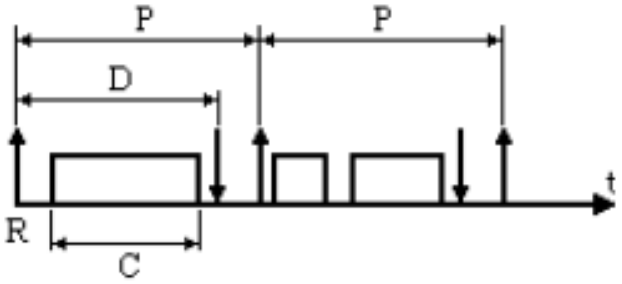

Figure 1. Canonical model of tasks [1]

Other elements involved in defining the characteristics of a task are:

- the execution priority, and
- the preemptibility (a task can be interrupted from its execution for the execution of another task).

III. Project Management Activities and Operating System Tasks

In terms of project management, we will consider the decomposition of the actual activities into smaller activities, to the level where such an activity can be carried out by an execution unit. To preserve the analogy with operation systems but to avoid confusion in the use of the term “activity”, we will use the term “task” only within the framework of operating systems and call “activity” the corresponding term the in project management modeling. The interpretations given by the scheduling theory should be applied to these “split” activities.

It should be noted that the execution of an activity can be interrupted by the need to execute other activities (from other projects). It may also require resource sharing.

We introduce the following terms:

- the execution unit: this can be a person, a structure, a team, etc. with the necessary skills to perform the proposed activity. (By analogy, it corresponds to the processor assigned to the execution of tasks at the level of operating systems);
- the time unit: it is defined according to the specific features of each project, as an hour, a day, a week, etc.

The characteristics of the definition of a task are taken from the operating systems, adapted and implemented accordingly at the database level of our simulation application. Thus, the maximum execution time of a task is given by the number of time units required by an execution unit to complete that task, if it were to deal only with its execution.

As a convention, in our simulation we have done the reasoning as if:

- the time unit is an hour,
- the execution unit (CPU) is a person,
- each person has a maximum number of hours they can work per day. These will be defined as the maximum number of working time units allowed for working on projects in a day.

It must also be determined for each execution unit the attributions, the authority and the constraints it has:

- the maximum number of working time units allowed for working on each project, and
- the number of available time units per time interval.

For each activity there must be specified:

- the necessary skills or competences to solve it,
- for each separate competence, the number of time units (hours) required by an execution unit, with the corresponding competence, to solve the entire activity if it deals only with solving it,
- its priority (which allows the execution unit to process activities in the order of their priority), and
- its preemptibility (allows the execution unit to interrupt the execution of an activity to solve another activity).

In conclusion, once the characteristics mentioned above are established for each activity, accordingly to the suggested decomposition, a model of activities similar to the model of tasks in a (real-time) operating system is obtained.

That is why the planning in advance of their realization can be done based on known scheduling algorithms. The result will either be an a priori planning of the realization of the activities, or the impossibility of planning will be established, which would mean that the proposed activities are not feasible and must be rethought.

It should be noted that the same idea of using scheduling algorithms can be used in the case of task resizing, too. A commonsense condition may come into play, given that planning has already been done, namely: changes to the planning of other activities should be minimal and non-existent for other activities independent of the replanned activity. These aspects are to be analyzed to see if such a condition is really to be imposed in the planning framework.

Note that it makes sense to "split" an activity down to as many activities as needed to be solved each with only one competency and one unit of execution. The reason is simple: to carry out an activity, that competence might only be needed for a short time, and it would not be fair, at least from the point of view of the efficiency, to block an execution unit for the entire duration of the execution of that activity of which it is a part. As such, it is necessary that, in order to implement the planning in advance, the activities

 Digital Object Identifier 10.32754/JMT.2022.2.06

are defined in such a way that the execution cost of an activity (from the viewpoint of the scheduling of activities within the project management) should be given by the number of time units required by an execution unit with the appropriate competence to solve the respective activity.

Notice that, at some moment, it could be necessary that some activities be executed by several execution units simultaneously, which means that an activity can only be considered completed under such conditions. According to the new interpretation, at the level of the database, the actual activity will be split down into as many activities as execution units are needed to solve it. All new activities have similar other characteristics and run in parallel. However, the cost of the initial activity, as execution time, will be given by the execution per single execution unit.

IV. Modeling the Planning of Activities through Scheduling Algorithms

To complete a project [11], it is split down into logical activities. This breakdown into activities is considered to be achieved up to the level of activities that can each be performed by a single employee. To ensure the compatibility of the idea of activity from project management with that of task from operating systems, it is necessary that all the attributes of an activity (that are taken from the canonical model of tasks) be defined.

In order to validate the ideas proposed above we implemented them in a web application, with the description of the tasks and the scheduling algorithms at the level of a database.

We defined the following tables (PK is used to denote the primary key of the table, and FK the foreign key):

Activity (transposing the notion of task from the real-time operating system theory): `Id (PK)`, `Name`, `Description`, `CostTime` (the number of time units required by an execution unit with the appropriate competence to solve the respective activity), `Periodicity` (`0` – no, default; `1` – yes), `Period` (in days), `Activation` (day starting from which one can work on that activity; `= R`, request, the point in time from which the activity must be considered for execution), `Deadline` (the last day on which one can work on this activity to be completed), `Priority`, `Preemptible` (`0` - no; `1` - yes, default), `Planned` (`0` - it was not planned, default value; `1` – it was planned, i.e. all time units necessary for its execution were consumed by the performed planning, `2` - in the process of planning), `CostTimeUnplanned` (the number of time units that still need to be planned to complete the activity), `CompetencyId (FK->Competency.Id)`, `ActivityId (FK->Activitate.Id`; on delete cascade; on update cascade on all activities decomposed from the main one; allows defining the decomposition of an activity into smaller activities and indicates that the current record represents an activity that is part of the activity indicated by `ActivityId`).

Competence: `Id (PK)`, `Name`, `Description`.

ExecutionUnit: `Id (PK)`, `Name`, `ExecutionCapacity` - maximum number of time units that the execution unit can perform, according to the specificity of the project thus modeled (e.g. if the execution unit is a person who works full-time, let's say 8 hours a day, one puts 8 here; or refer to the project type and the time unit used within the project type), `Description`, `ExecutionUnitId` (specific if it is part of a larger unit, i.e. team).

Non-reporting (allows managing the time intervals in which the execution unit cannot perform activities, not because the work schedule is full but for reasons of days off, holidays, etc.): `Id (PK)`, `ExecutionUnitId (FK -> ExecutionUnit.Id)`, `From`, `To`, `Reason`.

CompetenceExecutionUnit: `CompetenceId (PK, FK -> Competence.Id)`, `ExecutionUnitId (PK, FK -> ExecutionUnit.Id)`.

Planning (the table where it is recorded, for each execution unit which activity it executes, in which time intervals and how many time units are allocated to it. We will thus have planning of the execution of the activities, but also the follow-up of their execution): `ActivityId (PK; FK -> Activity.Id)`, `ExecutionUnitId (PK; FK -> ExecutionUnit.Id)`, `NoTimeUnits` (number of time units allocated in that interval to be executed by the respective execution unit in the considered interval), `StartDay`, `EndDay`.

In particular situations of implementing the management of some project(s), according to the philosophy described above, other attributes or adaptations of those already presented might be necessary.

V. Adaptations of the Algorithms to Achieve Planning and Results

A reasonable assumption in this modeling is that, in a unit of time, a unit of execution works on a single activity. (A person, in an hour, works on a single project).

Realizing the plans involves distributing the activities related to a project to the team members, so as not to exceed the number of hours they can allocate to the project and without exceeding the deadlines of the activities. In such a case we say that the planning is feasible and then, from the planning point of view, the entire project is feasible.

The result of the planning will allow one to know:

- for each unit of execution, every day, which activities must be worked on and how many units of time,
- for each activity, on which days, by which execution units is it executed and how many time units each of them allocates to it, every day,
- at each moment or in any time interval, what activity each execution unit executes and how much time it allocates to it.

The realization of planning is based on the following constraint in the allocation of activities to units of execution: not exceeding the load level of each execution unit in the time interval for which the allocation is made.

The scheduling algorithm

- determines the eligible activities to be executed at a given time;
- distributes them to the execution units that agree to the aforementioned constraint, updates the number of remaining time units available to the execution unit for each day, updates the number of time units that are still to be performed for each activity;
- "puts on hold" activities that were currently processed at the execution unit level, if it was necessary to interrupt them to prioritize a new activity, so that each interrupted activity can become eligible again at a new iteration of the algorithm (to be able to continue its execution).

For the implementation of the activity planning, we propose an approach based on the principles of scheduling algorithms known from the theory of real-time operating systems. In our application we adapted and implemented the most well-known ones, precisely the corresponding versions for multiprocessor scheduling: Round Robin, Least Laxity First, Earliest Deadline First, and Priority Scheduling.

These implemented algorithms are the most frequently used in hard real-time systems, i.e., systems where meeting deadlines is paramount. For example, these algorithms are implemented in the kernel of Linux-type operating systems, to allow the real-time operation of systems on which the appropriate configuration is made. The specificity given by activity planning and the mentioned parallelism with operating systems induces the need to use variants of multiprocessor algorithms. A description of them can be found in [7]; see also [12] for properties related to their optimality. Very useful for a good understanding of the taxonomy of multiprocessor platforms and the scheduling concepts and goals is [13].

The invocation of the scheduling algorithm is done for each day, starting with the nearest activation of an unplanned activity. Thus, starting with one of the activities with the closest activation and which corresponds to the criteria of the scheduling algorithm, it is checked if there is an execution unit with the competence corresponding to the activity and which has time units available for that day. The identified execution unit is assigned a number of time units for the execution of the new activity. The tables corresponding to the activity, respectively the planning, are updated accordingly.

The iteration of the scheduling algorithm continues until:

- either any activity becomes "planned", in which case it is reported that "planning has been completed successfully",
- or it is found an activity for which at least one of the constraints cannot be achieved, in which case it is announced that "planning could not be achieved". In this case, information is provided about the reason for the non-achievement, and it is also possible
    - to propose to try a planning based on another algorithm,
    - to provide visual output for the planning, up to the moment of blocking,
    - to offer the possibility of "assisted" planning, respecting the existing constraints.

The adaptation of the scheduling algorithms was done on similar principles, as follows:

Scrolling through the list of activities, it is checked who is the person with the minimum degree of occupation within a team (who completed a task the longest time ago) and if the person selected above has the number of hours available to complete the activity.

If yes, the activity is distributed to her/him and the period required to complete it will be marked as busy, otherwise checks will be made for the next person on the team.

If there is no team member available, the task is distributed to the first person for whom the checks have been made.

The principle of the Round Robin algorithm ([12], [14]) is to schedule tasks with identical priorities in the order in which they appear. This requires in our framework that, in the implementation, the list of project activities be ordered by start date (the moment from which they can be executed).

In the case of the Earliest Deadline First algorithm, the tasks are executed according to the deadline. Thus, the activities are also ordered according to their deadline.

The calculation of the laxity of a task is given by:

$$Laxity = Deadline - (StartDate + Cost) \quad (1)$$

In the Least Laxity First algorithm, the activities are ordered similarly to the other algorithms, but according to their laxity.

The description of the principles of the Priority Scheduling algorithm can be consulted in [15]. Three types of priorities were used in our proposed simulation: high, medium and low. For activities with the same priority, the ordering takes into account the start date of each task.

VI. Justification of Using Several Algorithms

The proposal to use several algorithms is given, on the one hand, by the fact that, for the multiprocessor case, no algorithm guarantees the feasibility of any task model. On the other hand, even in the case of feasibility, each of the algorithms can determine a feasible or not feasible schedule, different from the other algorithms. The suitable variant would then be chosen by the human decision-maker, depending on other criteria, not included in the proposed modeling.

In this direction, we present the following example in which the implementation proposes different schedules resulting from the running of the four algorithms.

Within Project P, 10 independent activities are provided to be distributed to the members of a team of 3 people.

In the following figures, presenting the scheduling provided by the four mentioned algorithms, the activities assigned to the same person are illustrated with the same color.

The activities are named from T01 to T10, and their parameters are presented in Table I. The cost of the activities is approximately equal to the difference between their deadline and their start date.

Table I. Characteristics of The Project Activities

Nr.	Activity Name	Start date	Deadline	Cost (hours)	Priority
1	T01	01.04.2023	07.04.2023	48	low
2	T02	01.04.2023	05.04.2023	30	medium
3	T03	01.04.2023	07.04.2023	24	high
4	T04	02.04.2023	06.04.2023	4	medium
5	T05	04.04.2023	05.04.2023	10	low
6	T06	08.04.2023	09.04.2023	8	low
7	T07	01.04.2023	03.04.2023	16	high
8	T08	07.04.2023	10.04.2023	20	medium
9	T09	07.04.2023	08.04.2023	1	low
10	T10	08.04.2023	09.04.2023	5	medium

 Digital Object Identifier 10.32754/JMT.2022.2.06

Following the application of the algorithms, the scheduling results are presented in Fig. 2 to Fig. 6. In all of them we can observe that:

- different persons have different colors allocated,
- the effective date when the task effectively begins can be different from the start date,
- the end date can be after the deadline of the task,
- for those activities that missed their deadlines, the deadline is marked by a black vertical arrow, down oriented.

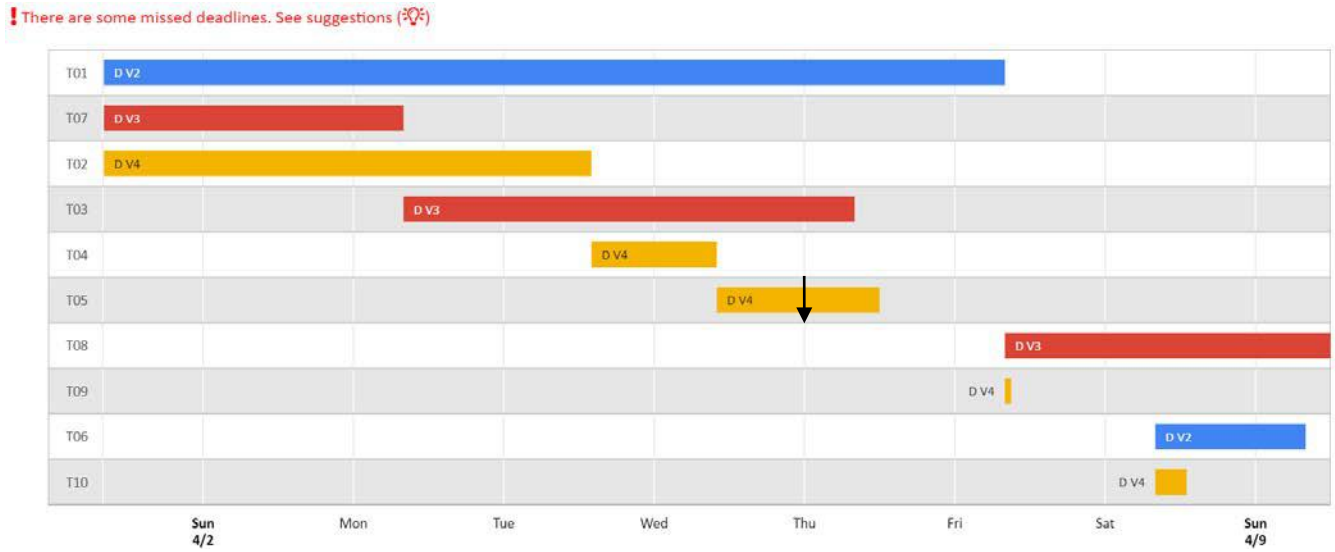

Figure 2. The schedule obtained by the Least Laxity First algorithm: the activity T05 is missing its deadline

As presented in Fig. 2, in the schedule obtained by the Least Laxity First algorithm:

- red and blue persons completes in time all their corresponding activities, and
- yellow person completes in time the activities: T02, T04, T09 and T10, but not the T05 one.

Also, we have obtained some suggestions for feasible scheduling:

- to extend the deadline for T05 by one day, or
- to use a team of 4 persons.

The scheduling of the LLF by involving four persons is depicted in Fig. 3, and we can see no deadline missed.

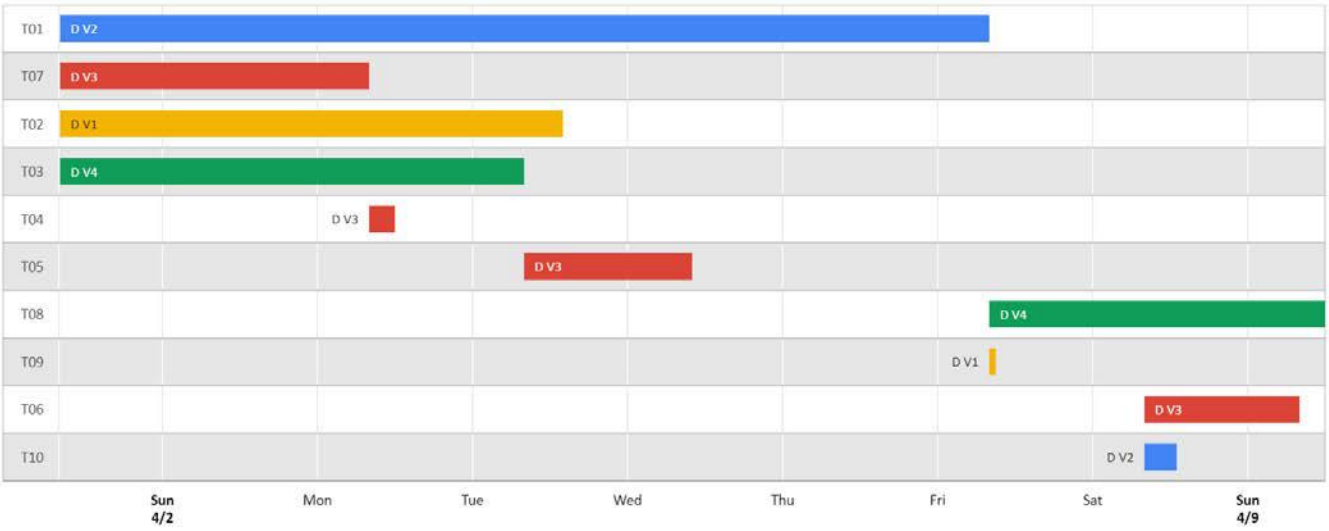

Figure 3. The schedule obtained by the Least Laxity First algorithm, with 4 persons: no missed deadlines

As presented in Fig. 4, the results obtained by the Round Robin algorithm prove that:

- red person completes in time their corresponding activities: T02, T04, T09 and T10, but not the T05 activity,
- yellow person completes in time the activities: T03 and T08, but not T07 activity,
- blue person completes all the activities distributed to him (T01 and T06) before there deadlines.

A suggestion obtained for a feasible scheduling is to extend the deadline for T07 by three days and for T05 by one day.

As presented in Fig. 5, in the schedule obtained by the Earliest Deadline First algorithm:

- red person completes in time four activities (T02, T04, T09 and T10), but not the activity T05,
- blue person completes in time all their activities,
- yellow person completes in time the activities T03 and T08, but T07 miss its deadline.

For the EDF scheduling the suggestion for not missing deadlines is to extend the deadlines of the activity T01 by two days and of T08 by one day.

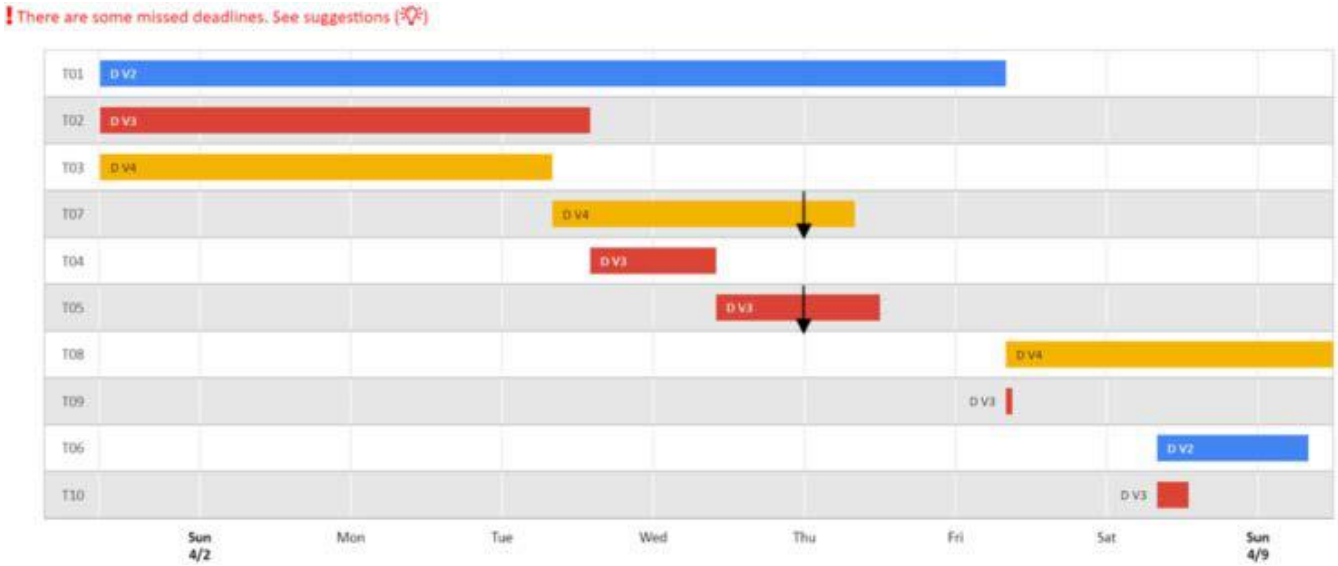

Figure 4. The schedule obtained by the Round Robin algorithm: the activities T05 and T07 are missing their deadlines

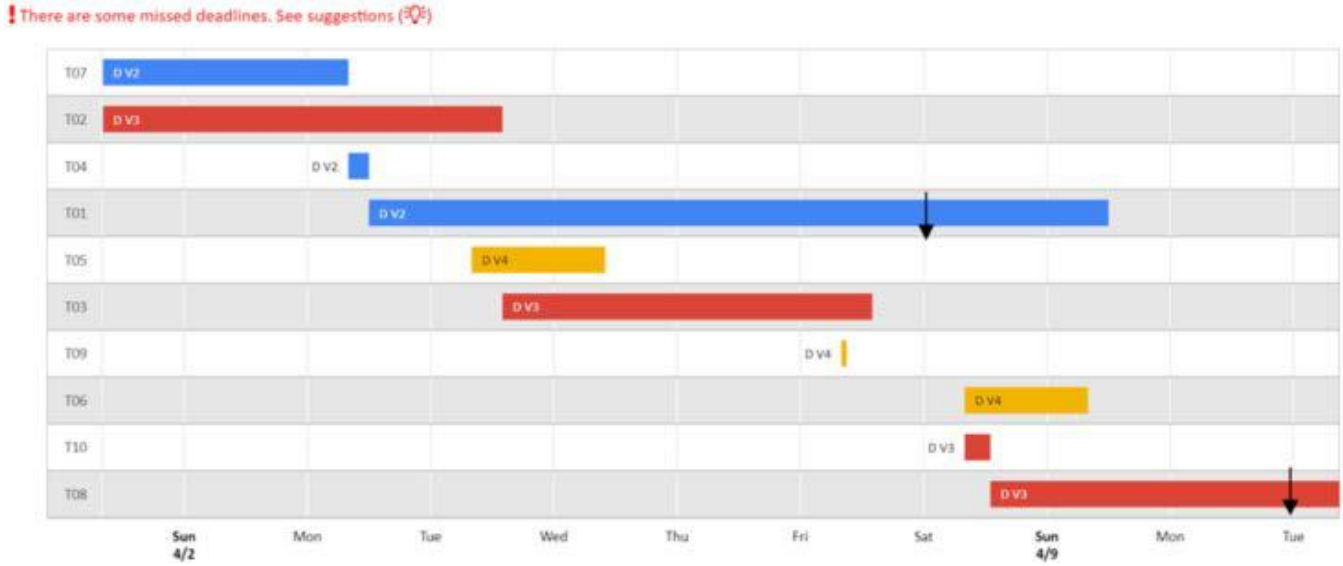

Figure 5. The schedule obtained by the Earliest Deadline First algorithm: the activities T01 and T08 are missing their deadlines

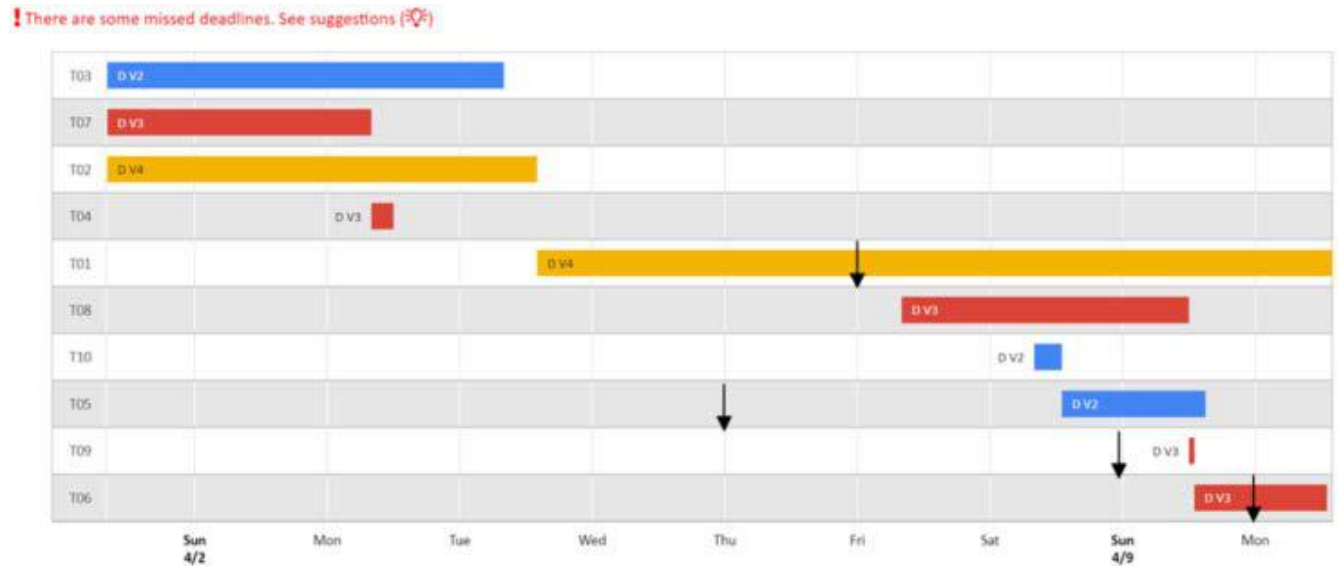

Figure 6. The schedule obtained by the Priority Scheduling algorithm: the activities T01, T05, T06 and T09 are missing their deadlines

The worst scheduling was obtained by Priority Scheduling algorithm. In this case, as seen in Fig. 6, there are four activities missing their deadlines, and not one of the three persons is ready in time with their respective activities.

 Digital Object Identifier 10.32754/JMT.2022.2.06

VII. Considerations Regarding other Task Models and other Restrictions

The simulations carried out considered only the situation of independent tasks and with no resource constraints or resource sharing.

We understand by resource constraints the situations in which two or more tasks must access a resource that is not sharable at the same time. These situations are not the object of the scheduling algorithms themselves, thus they are not included in their operating principles.

As such, the scheduling algorithms are the same for the case where tasks must share resources. Resource sharing modeling techniques is also known from the operating systems theory. Other situations encountered in task scenario modeling are mentioned below.

- dependency between tasks (the execution of a task directly depends on the completion of other tasks),
- mutual exclusion (we say that two tasks are in mutual exclusion if, during the execution of one of them, the other cannot run, and conversely; partial exclusion can also occur, if not the entire duration of the execution is targeted, but only a part of it).

Situations such as those mentioned above present special implementation mechanisms within the operating systems. As mentioned, they are not an intrinsic part of the scheduling algorithms themselves.

To adapt the model used in the simulations, for non-interruptible tasks, so that mutual exclusion situations are taken into account, the database must be completed with the following table:

Exclusion: Id (PK), Id_Task1 (FK -> Tasks.Id), Id_Task2 (FK -> Tasks.Id), Data_I, Ora_I, Data_F, Ora_F),

with the meaning that, during the execution of the task Id_Task1, between Date_I, Time_I and Date_F, Time_F, Id_Task2 cannot be executed.

The adaptation for the more general case of interruptible tasks requires taking into account the actual time / remaining time from the execution of a task and not the start and end times of the tasks.

Exclusion: Id (PK), Id_Task1 (FK -> Tasks.Id), Id_Task2 (FK -> Tasks.Id), remaining_hours1, remaining_hours2),

meaning that, during the execution of the task Id_Task1, as long as the value of Tasks.remaining_hours corresponding to Id_Task1 is between remaining_hours1 and remaining_hours2, it will not be possible to execute Id_Task2.

In the situation where a task planning has already been carried out, but new tasks appear in the system, they must be planned. We call this dynamic (re)planning.

The algorithms implemented in the model proposed above give dynamism to the system for which they are implemented, in the previously mentioned sense. Thus, within the operation of the operating systems, for which they were originally designed, the scheduling algorithms are executed periodically, determining the distribution / redistribution of tasks for execution on the existing processors in the system. We mention this to highlight that they remain unchanged and apply as such to the proposed system whenever needed.

The transposition of these algorithms into a project management system involves running them as follows:

- once for carrying out the initial planning, after having entered the data related to its own activities in the form proposed above, followed by the identification of the best of the proposed feasible solutions.
- whenever there is a change in the characteristics of one of the activities, to follow (or to adjust to) the implications that could be on the already existing orders.
- whenever it is required to include a new activity that has not yet been planned.

We consider necessary to make clear the following specifications:

1. Non-guarantee of feasibility is not a minus of the proposed system. Currently, for the multiprocessor case, no general feasible algorithm valid for any task model is known.
2. In the case of multiprocessor scheduling, certain anomalies are known. Mainly, they say that, if on a system for which there is a feasible scheduling of tasks, changes are made to the system characteristics, they may cause infeasible scheduling, even if they would apparently be beneficial for optimizing the execution of the set of tasks (e.g. increasing the number of processors, decreasing the execution time of a task, etc.).
3. The use of the proposed algorithms does not guarantee obtaining a feasible solution for the entire system, but it proposes solutions that allow the human decision-maker to adopt the right option or consider changes in the activities so as to reach feasible solutions. Thus, activity changes can be entered into the system and new scheduling simulations run.

Within the operating systems, the scheduling algorithms are run periodically, thus taking into account, in a systematic way, the changes that occur within the existing task sets in the system and inducing replanning corresponding to the principles of the algorithms and the properties of the newly introduced tasks between the two time points of the release.

Thus, periodic rescheduling is analogous to the operation of task scheduling in operating systems. It can also be considered that dynamic replanning, in the sense mentioned in the previous section, is a particular case of periodic replanning.

VIII. Conclusion

This article proposes a new way of planning activities in project management, justified by the analogy of notions with those of tasks in multiprocessor scheduling operating systems. This approach is based on taking over the principles of algorithms and techniques from a theoretically well-founded field. It yields the benefits of established results, and encourages efforts to transpose the other notions from scheduling theory. The goal is to cover the entire range of situations that may arise in project management. Tests should be carried on, for large scale real-life projects.

Acknowledgment

Thanks are due to the referees, for their comments, improving a previous version of the paper.

REFERENCES

[1] Asana, https://asana.com/.
[2] Trello, https://trello.com/.
[3] Taskworld, https://taskworld.com/.
[4] Monday.com, https://monday.com/.
[5] J. P. Lewis, *Project Planning, Scheduling & Control*, 5th ed., McGraw Hill, 2011.
[6] *Handbook on Project Management and Scheduling*, C. Schwindt, J. Zimmermann, Eds., Springer, 2015.
[7] F. Cottet, J. Delacroix, C. Kaiser and Z. Mammeri, *Ordonnancement temps reel*: Hermes, 2000.
[8] W. Stallings, *Operating systems: internals and design principles*, 7th ed., Prentice Hall, 2012.
[9] M. Bar, *Linux Internals*, McGraw-Hill, 2000.
[10] A. S. Tanenbaum, H. Bos, Modern Operating Systems, 4th ed., Pearson, 2015.
[11] H. Kerzner, *Project Management: A System Approach to Planning, Scheduling, and Controlling*, Wiley, 2017.
[12] M. L. Dertouzos, A. K. Mok, "Multiprocessor online scheduling of hard-real-time tasks," IEEE Transactions on Software Engineering, vol. 15, pp. 1497-1506, 1989.
[13] S. Baruah, M. Bertogna, G. Buttazzo, *Multiprocessor Scheduling for Real-Time Systems*, Springer, 2015.
[14] L. Williams, "Round Robin Scheduling Algorithm with Example". https://www.guru99.com/round-robin-scheduling-example.html
[15] L. Williams, "Priority Scheduling Algorithm: Preemptive, Non-Preemptive EXAMPLE". https://www.guru99.com/priority-scheduling-program.html.

 Digital Object Identifier 10.32754/JMT.2022.2.06