Scheduling — Introduction

Scheduling is studied primarily in two research communities:

- Operations Research
  - job shop scheduling and shop-flow problems
  - schedule physical devices and processes - machines, workcells, maintenance, orders, batches, projects
  - mostly (not exclusively) static schedules
- Computer Science
  - schedule tasks in a uni- or multiprocessor environment
  - mostly (not exclusively) dynamic schedules

Scheduling — Introduction (cont.)

Problem Formulation

Events

- events occur that require a response in the system — computation, work, communication
- interrupts
- periodic, aperiodic, sporadic, and stochastic events

Time

- execution times for event handling — how much contiguous CPU time will it take to execute a piece of code to handle an event, how much time will it take to change a bearing on the milling machine.

Deadline

- an upper bound on the time required to service an event

Scheduling

- reflects decisions about which event to process at a given time

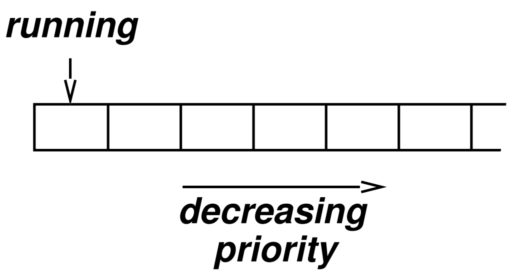
Priorities and Scheduling

How should processes be scheduled in order to maximize the utilization of the system while guaranteeing deadlines? Consider the following task scheduling policy.

Each process is assigned a priority that reflects the severity of the time demands of the process (high priority implies more severe, low priority implies less severe).

$$\begin{array}{ccc} high & \longleftrightarrow & low \\ 0 & \longleftrightarrow & 255 \end{array}$$

The ReadyQueue is sorted according to priority:



A blocked process that becomes ready and has higher priority than the currently running process causes a context shift. This task *preempts* the running task.

This scheduling policy is called:

priority-based preemptive scheduling.

Priorities and Scheduling (cont.)

Priority Inversion

Consider tasks T_1, T_2, and T_3 with corresponding priorities $P_1 > P_2 > P_3$ (that is, task T_1 has the *highest* priority).

If task T_1 depends on task T_3, it may be suspended in order to wait for a result from T_3. However, a priority-based preemptive scheduler will then choose to run task T_2, thus further blocking task T_1 indirectly by way of task T_3 even though $P_1 > P_2$.

Priority Inheritance

The priority of a process is the max of its own default priority and the priorities of all other processes that are, at that time, dependent on it.

therefore:

> task T_3 inherits priority P_1 while T_1 is waiting for a T_3 event $\Rightarrow$ T_3 will run in preference to T_2.

For example, a synchronous communication protocol (rendezvous) requires the sender to wait until contact is established and the receiver is verified before the message is sent. In ADA, the priority of the rendezvous itself is the highest priority of the two participating processes.

Scheduling

...to ensure that the run-time scheduler dispatches the runnable processes in an order that will result in all deadlines being met.

orders the ReadyQueue

$\Downarrow$

allocates priorities

BOTH STATIC AND DYNAMIC PRIORITY ALLOCATIONS ARE POSSIBLE

Scheduling (cont.)

Schedulability

Given a single processor, a set of n-independent, periodic processes with only execution time deadlines (no start/stop constraints) are schedulable if the overall processor utilization is less than:

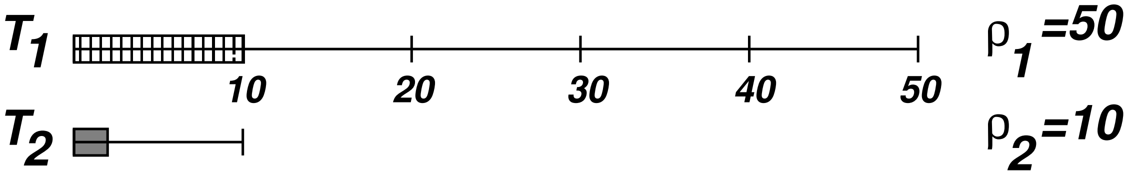
$$n(2^{1/n} - 1)$$

For large n this function approaches a utilization of 0.693. This is a very conservative estimate, based on worst-case relationships between the periods of the processes.

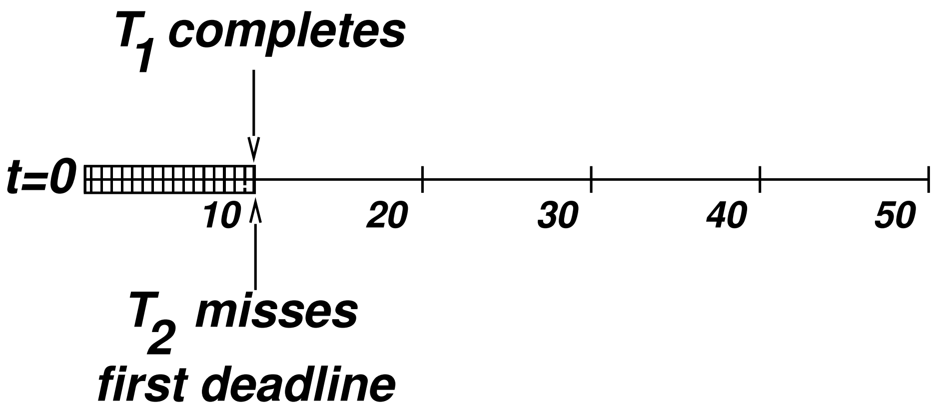
Liu and Layland (1973)

Scheduling (cont.)

Consider two tasks, T_1 and T_2, with $P_1 > P_2$ (T_1 is more "important"). T_1 has a period of 50 units of time and requires 10 units of time to run. Task T_2 has a period of 10 units of time and requires 2 units of time to run.



Under a priority-based preemptive schedule, T_1 will execute first and when it completes, T_2 will have already missed its first deadline!



The problem is related to the use of "preference" priorities.

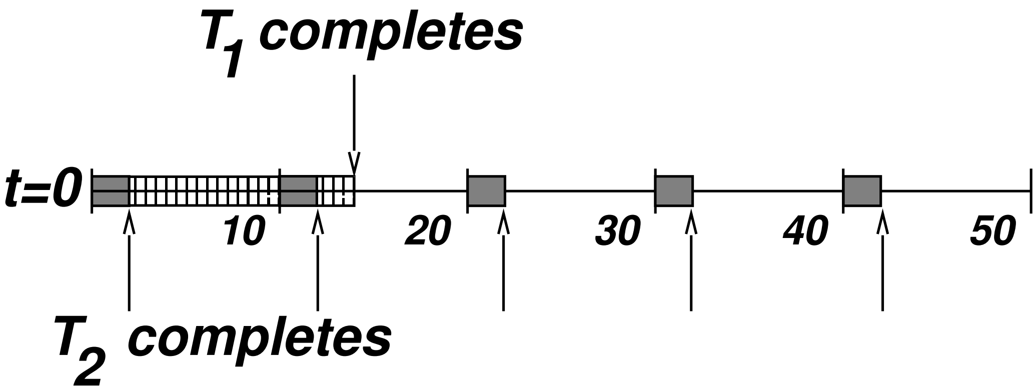
Scheduling (cont.)

Rate Monotonic Scheduling

Assign fixed priorities inversely proportional to the period for the task,

$$P_i \propto \frac{1}{\rho_i},$$

so that the shorter the period, the higher the priority.



This is an optimal static schedule for independent periodic processes — if a process can be scheduled by any static scheduling algorithm, then it can also be scheduled by the rate monotonic algorithm.

Scheduling (cont.)

Extensions to Rate Monotonic Scheduling

Soft RT Systems — If all deadlines cannot be met, the most "important" processes must be run first. If the ReadyQueue is schedulable, then the process with the shortest period should be first.

"Specification Aliases" — Designer intentions and period-based priorities can be addressed simultaneously by transforming specifications for the most important tasks. If T_1 ($\rho = 50$, and execution time $= 10$) were re-specified T_1' ($\rho = 5$, and execution time $= 1$), then its relative preference is consistent with its rate-based priority. The details are non-trivial.

Periodic Servers — A generalization of the previous idea can be used to incorporate both aperiodic and important soft-deadline tasks into the rate monotonic schedule. A periodic server (or several of them) can be incorporated into the run-time schedule and used to poll these tasks. The result, in theory, is a global schedule — even for tasks with random arrival rates! However, to guarantee performance during bursty aperiodic task arrivals, it is necessary to reserve enough time in the periodic server to handle worst-case arrival rates. These types of extensions compromise the system's ability to handle truly hard real-time tasks.

Scheduling (cont.)

Other Scheduling Algorithms

- Earliest Deadline — need to know about deadlines
- Least Slack Time — needs deadlines and the amount of execution time needed to complete...

$$\Downarrow$$

...scheduler allocates resources to the task with the least "free time"

When all deadlines can be met, Earliest Deadline and Least Slack Time are equivalent to Rate Monotonic — every periodic task load schedulable by one is schedulable by all. These alternatives are designed to do the best they can with instantaneous and aperiodic task loads.