

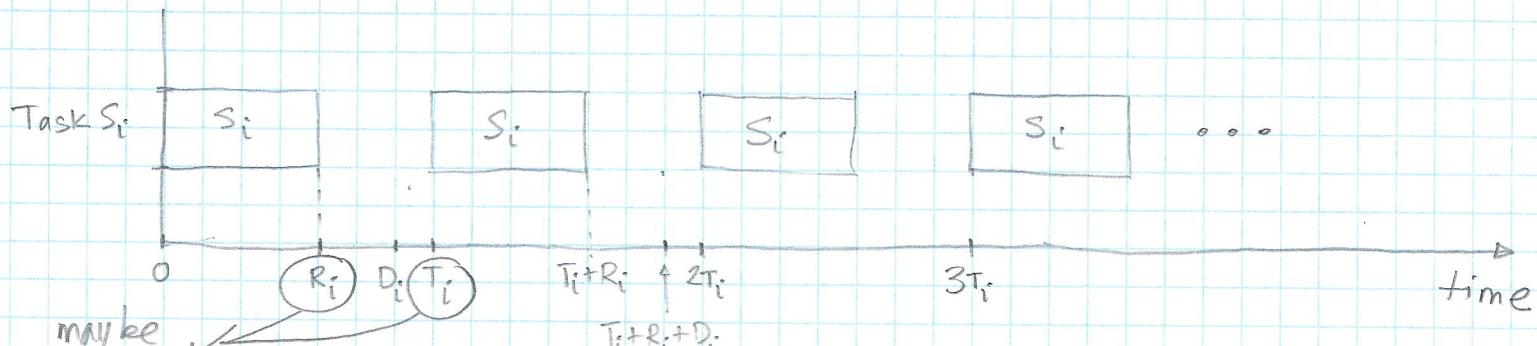
- as we have seen, the conventional Linux kernel is not strict in the sense of task timing; Linux is designed as a "time sharing" system that attempts to provide CPU time for all users, rather than prioritize any given process.
- in a real-time OS, the scheduler is designed to be task-deadline driven, prioritizing certain tasks over others.
- hard real-time systems have time constraints which are absolute; that is, a total system failure may result from a single missed deadline. Examples of hard real-time systems include signal processing devices with fixed sampling periods, high-performance autonomous control systems found in robotics, cars and aircraft + if strict timing is not maintained, the consequences can be severe.
- soft real-time systems are those for which strict timing is desirable but for which an occasional missed deadline has insignificant or mild consequences.
- in truth, it is difficult in many cases to prove that a system will never miss a deadline — reality presents situations that the designer may not have anticipated
- to help us to estimate the "real-time robustness" of an RTOS, we often assume certain conditions that simplify things in order to perform analysis. For example, we might assume deterministic (predictable) conditions, which preclude unexpected events. Tasks may be assumed to have fixed execution times and periods:

## useful terminology follows...

RTS\_W2-2

- Suppose we have a number of tasks, each denoted by  $S_i$ ,  $i=1, \dots, N$  each with a corresponding deadline ( $D_i$ ), release time ( $R_i$ ), and task period ( $T_i$ ).

On a timeline, we could represent the task as:



maybe  
we should  
not  
distinguish  
between  
these  
two

the deadline  $D_i$  is the amount of time the task  $S_i$  has to execute;

the release time  $R_i$  is the time taken by the task, from when it first becomes active, until completion

IT IS  
BEST  
TO  
THINK  
OF  
RELEASE  
PERIOD  
AS  $T_i$

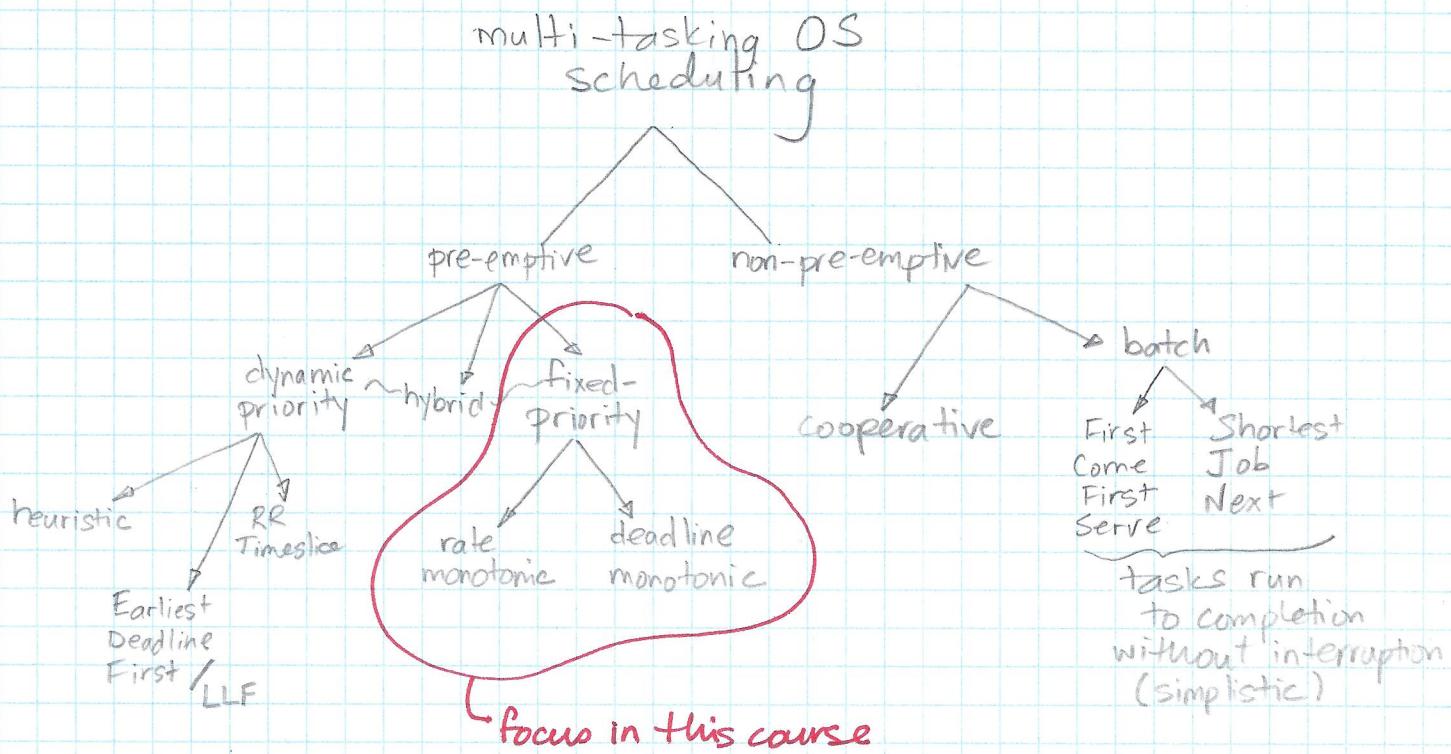
NOTE: in many examples, we often see that  $D_i = R_i = T_i$  is used as a simplifying condition

$T_i$  is the task period — it is sometimes helpful to assume a fixed amount of time between executions of a task, although in many real-world cases some deviation from periodic behaviour should be anticipated.

- Please bear in mind that:

- real-time systems are often analyzed with deterministic assumptions (i.e., events are predictable);
- service/task requests are periodic, and execution times of tasks are fixed (and known)
- designers attempt to prove optimality (see below)

## taxonomy of multi-tasking operating systems:



- fixed-priority design is at the foundation of pre-emptive operating systems
- Rate Monotonic (RM) and Deadline Monotonic (DM) two assignment policies in the fixed-priority category; both are optimal.
- An assignment policy is optimal if any set of services that can be scheduled with no missed deadlines can be scheduled with the assignment policy
  - optimality is a theoretical guarantee that a system is a hard real-time system ←
  - but bear in mind that many assumptions are made (described above) in any proof of optimality.

- in a **rate monotonic** (RM) policy higher priority is given to tasks with the highest request frequency (shortest task period);
- in a **deadline monotonic** (DM) policy, higher priority is given to tasks with the shortest deadline;
- note that if  $D_i = R_i = T_i$ , RM and DM are equivalent.

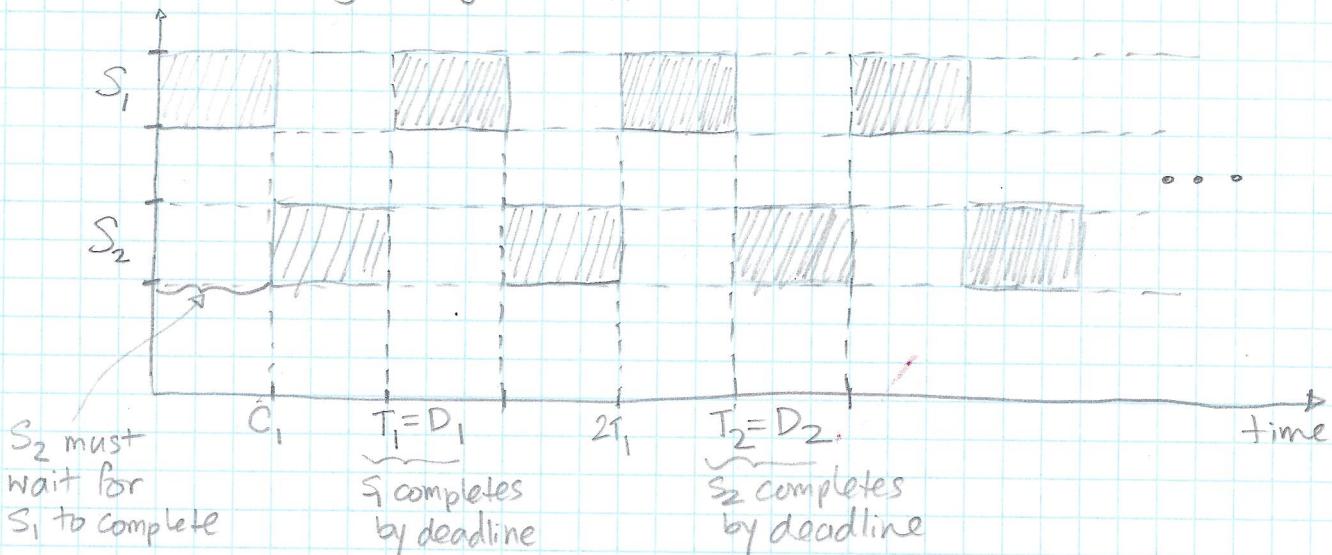
### EXAMPLE :

- given two tasks  $S_1$  and  $S_2$  with request periods  $T_1 = 6$  and  $T_2 = 15$  (in some unit of time), and execution times  $C_1 = 3$  and  $C_2 = 6$  (same time units), respectively, how does RM policy apply?

SOLN :

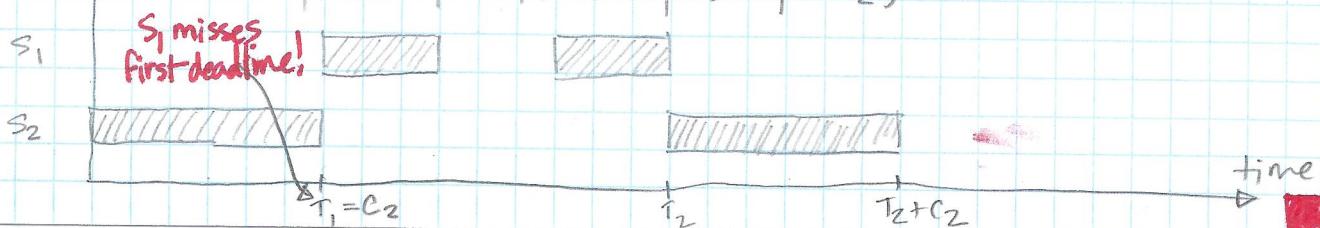
- because RM policy assigns higher priority to shorter task periods, we have that

priority( $S_1$ ) > priority( $S_2$ )  
a timing diagram appears as follows:



- if we adopt the anti-policy, i.e.,

$$\text{priority}(S_1) < \text{priority}(S_2)$$



- in dynamic-priority RTOS, priority assignments are adjusted as tasks are released — think of it as a continuous adjustment
- please note that the terms "release period" and "request period" are used interchangeably in the text and some confusion might be created; we will use the symbol  $T_i$  to denote the release or request period from now on.
- also note that we will use the parameters  $T_i$  and  $C_i$  to characterize a task or service  $S_i$ , where  $C_i$  is the task execution time.
- here is an example comparing RM and EDF policies — recall that EDF is a dynamic method.

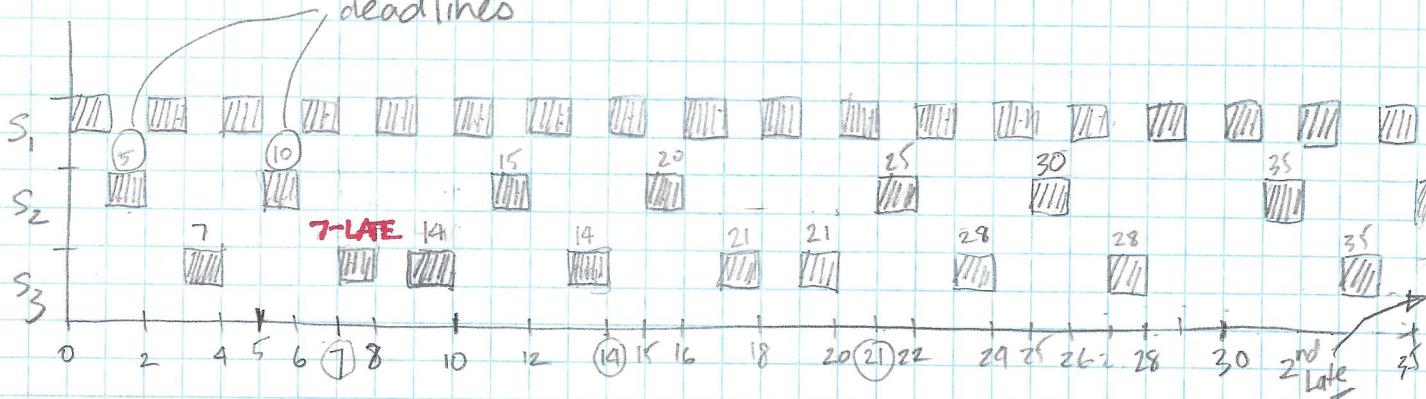
### EXAMPLE

Suppose there are three tasks,  $S_1$ ,  $S_2$  and  $S_3$ , characterized as follows:

task	$T_i$	$C_i$
$S_1$	2	1
$S_2$	5	1
$S_3$	7	2

Produce timing diagrams of the RM policy scheduling and EDF policy scheduling.

i) RM: priority( $S_1$ ) > priority( $S_2$ ) > priority( $S_3$ ) ( $\because T_1 < T_2 < T_3$ )



- as we can see, RM policy produced two late errors for task  $S_3$  in our timing diagram (one error is not shown, just to right of  $t = 37$  units)

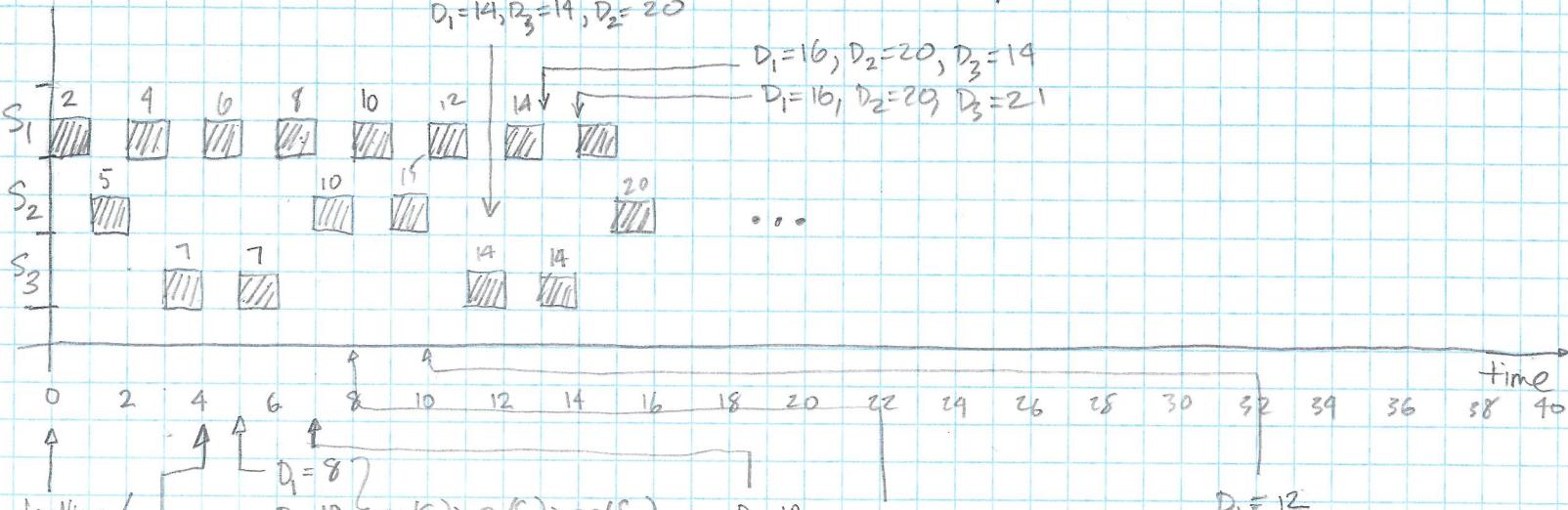
RTS\_W2-6

- ii) EDF policy  $\rightarrow$  priorities re-assigned according to earliest deadline at every dispatch.

$$D_1 = 14, D_2 = 14, D_3 = 20$$

$$D_1 = 16, D_2 = 20, D_3 = 14$$

$$D_1 = 16, D_2 = 20, D_3 = 21$$



deadlines reassessed

$pr(S_1)$  deadlines here are:

$pr(S_2)$   $S_1 = 6 \quad pr(S_1) > pr(S_3) > pr(S_2)$

$pr(S_3)$   $S_2 = 10 \quad pr(S_2) > pr(S_3)$

$S_3 = 7 \quad swapped$

$$D_1 = 10 \\ D_2 = 10 \\ D_3 = 14$$

$$D_1 = 10 \\ D_2 = 15 \\ D_3 = 14$$

$$(pr(S_1) > pr(S_2) > pr(S_3))$$

$$D_1 = 12 \\ D_2 = 20 \\ D_3 = 14$$

$$(pr(S_1) > pr(S_3) > pr(S_2))$$

where RM Fails, EDF Succeeds (at least in this example).

Why don't we always use EDF?

- there is some computational overhead associated with dynamic priority policies
- in an overload situation (missing deadlines becomes imminent), whereas in RM only lower priority tasks are affected, in EDF it is hard to predict what the effects on the system will be (potentially catastrophic!)
- usually dynamic priority policies are used only in soft real-time systems.

Homework

- 1) read first three chapters of Richard Barry's Mastering the FreeRTOS Real-Time Kernel
- 2) order & receive your LPC1769 eval kit
- 3) download & install the MCU Xpresso IDE