Real-time systems
“Real-time scheduling of independent tasks”

Mathieu Delalandre
University of Tours, Tours city, France
mathieu.delalandre@univ-tours.fr

Lecture available at http://mathieu.delalandre.free.fr/teachings/realtime.html

Real-time scheduling of independent tasks

1. About real-time scheduling
2. Process and diagram models
3. Basic on-line algorithms for periodic tasks
   3.1. Basic scheduling algorithms
   3.2. Sufficient conditions
4. Hybrid task sets scheduling
   4.1. Introduction to hybrid task sets scheduling
   4.2. Hybrid scheduling algorithms

About real-time scheduling (1)

There are important properties that real-time systems must have to support for critical applications.

	System	
Features	**no real-time**	**real-time**
Scalability	++	+
Maintainability	+	+
Fault tolerance	+	++
Design for peak load	+	++
Timeliness	no	yes
Predictability	no	yes

Considering the operating system level, real-time OS are based on kernels which are modified versions of time-sharing OS (i.e. no real-time). As a consequence, they have the same basic features and differ in terms of:

	Operating System	
Features	**no real-time**	**real-time**
Scheduling	different	
IPC and synchronization	different	
Resource management	different	
OS type	full OS	micro kernel
Interrupt handling	slow	fast
Context switch (dispatcher)	slow	fast
Process model	basic	extended

About real-time scheduling (2)

(Short-term) scheduler is a system process running an algorithm to decide which of the ready processes are to be executed (allocated a CPU). The short-term scheduler is concerned with:

- ✓Response time: total time between submission of a request and its completion
- ✓Waiting time: amount of time a process has been waiting in the ready queue
- ✓Throughput: number of processes that complete their execution per time unit
- ✓CPU utilization: to keep the CPU as busy as possible
- ✓Fairness: a process should not suffer of starvation i.e. never loaded to CPU
- ✓Etc.

Depending of the considered systems (mainframes, server computers, Personal Computers (PC), Real-Time Systems, embedded systems, etc.) schedulers could be designed in different ways:

Scheduling problems	Parameters		Time-sharing OS	Real-time OS
algorithm's features	on-line	off-line	on-line	on-line
	preemptive	no preemptive	both	both
	optimal	not optimal	both	both
	relative deadline	strict deadline	relative deadline	both
Processus model	independents	dependents	both	both
	static priority	dynamic priority	both	both
	without resource	with resource	both	both
	aperiodic	periodic	aperiodic	both
Type of system	lazy processor	full-time busy processor	lazy processor	lazy processor
	mono-core	multi-core	both	both
	centralized	distributed	centralized	centralized

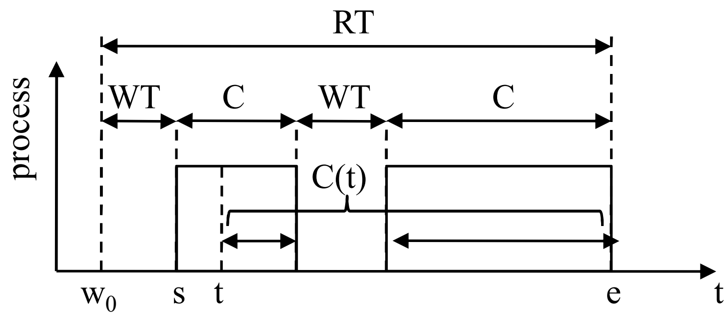
Real-time scheduling of independent tasks

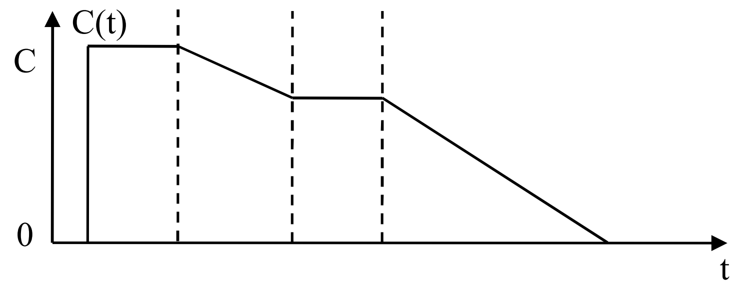
1. About real-time scheduling
2. Process and diagram models
3. Basic on-line algorithms for periodic tasks
   3.1. Basic scheduling algorithms
   3.2. Sufficient conditions
4. Hybrid task sets scheduling
   4.1. Introduction to hybrid task sets scheduling
   4.2. Hybrid scheduling algorithms

Process and diagram models (1)

Process model and context parameters

PID	process number	Process parameters
rank	rank in the ready queue	
w_0	wakeup time	
C	capacity	
P	priority	
s	start time (run as a first time)	context parameters
e	end time (termination)	
$RT = e- w_0$	response time	
$WT = RT-C$	waiting time	
$C(t)$	residual capacity at t $C(w_0) = C$, $C(e)=0$	
$T(t)=C-C(t)$	CPU time consumed at t $T(w_0)=0$, $T(e) = C$,	
$E(t)=t-w_0$	CPU time entitled $E(w_0)=0$, $E(e)=RT$	
$WT(t)=E(t)-T(t)$	waiting time at t $WT(w_0)=0$, $WT(e)=WT$	





Process and diagram models (2)

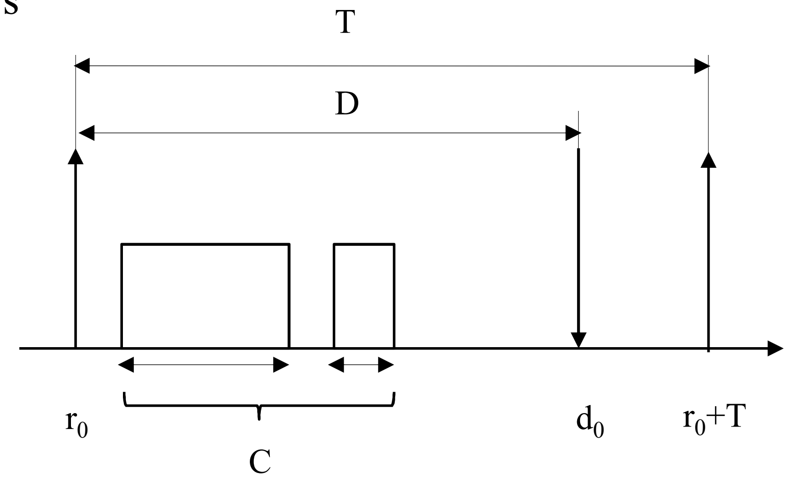
Task (i.e. process) model and context parameters

Process parameters

PID	processus number
Rank	rank in the ready queue
r_0 (i.e. w_0)	release time (in the ready queue)
C	capacity
P	priority

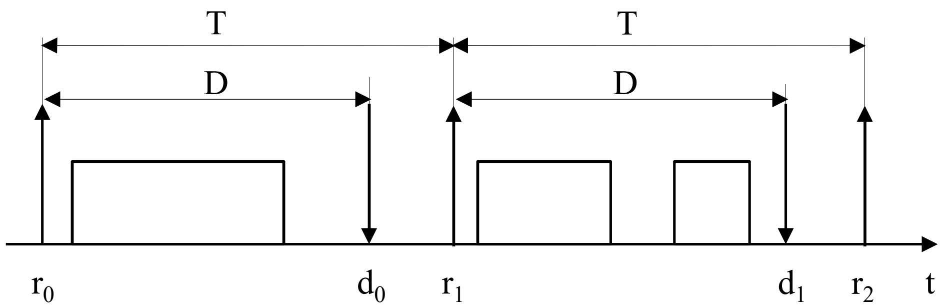
Extended parameters for real-time

D	relative deadline
T	period



context parameters

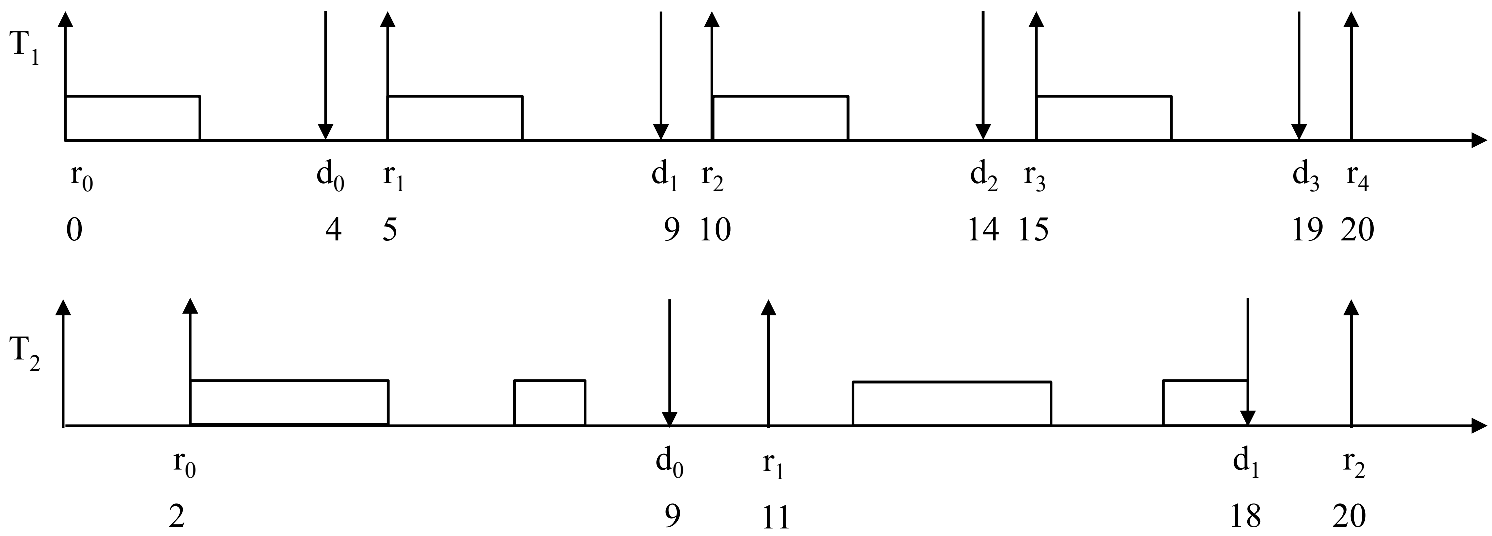
$r_k = r_0 + k\times T$	the k^{th} release
s_k	the k^{th} start time
e_k (or f)	the k^{th} end (finishing) time
$d_k = r_k + D$	the k^{th} absolute deadline
$0 \leq C \leq D \leq T$	well formed task
$L_k = e_k - d_k$	Lateness
$E_k = \max(0, L_k)$	Tardiness or exceeding time



Process and diagram models (3)

e.g. here is a random CPU diagram (i.e. virtual scheduling algorithm) respecting scheduling constraints, absolute deadlines and releases for the following set of tasks:

	r_0	C	D	T
T1	0	2	4	5
T2	2	4	7	9



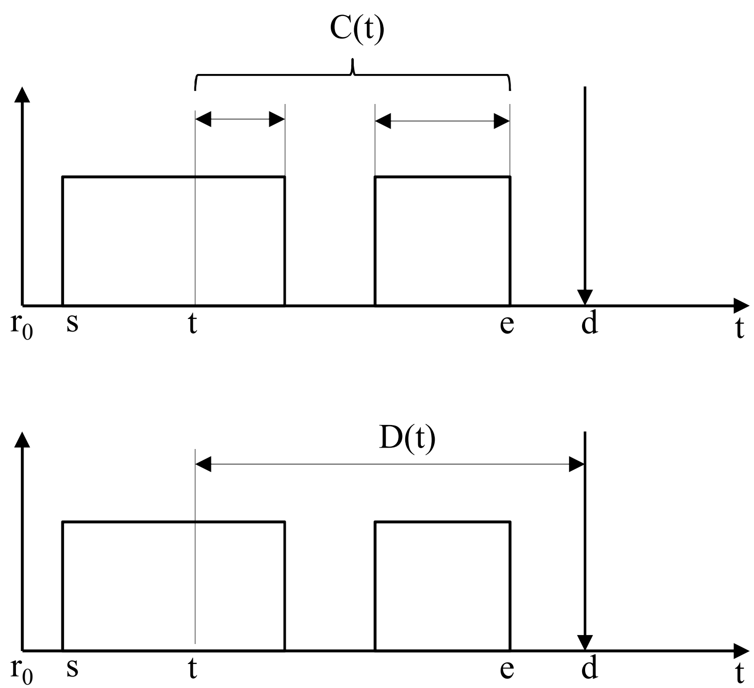
k	s_k	e_k	L_k	E_k
0	0	2	-2	0
1	5	7	-2	0
2	10	12	-2	0
3	15	17	-2	0

k	s_k	e_k	L_k	E_k
0	2	8	-1	0
1	12	18	0	0

Process and diagram models (4)

Task (i.e. process) model and context parameters, next …

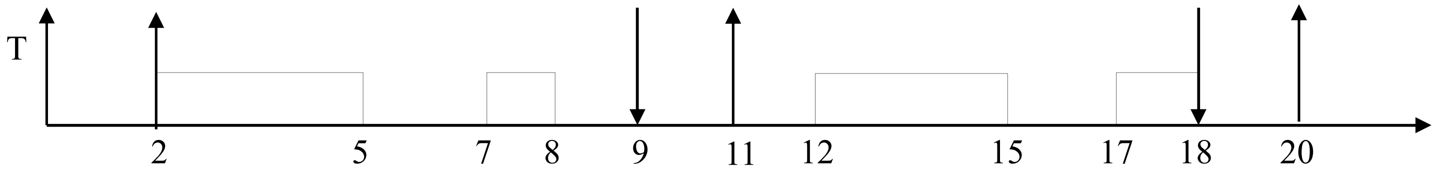
$u = \frac{C}{T}$	processor utilization factor $u > 0$	context parameters
$U = \sum_{i=1}^{n} u_i = \sum_{i=1}^{n} \frac{C_i}{T_i}$	mean processor utilization factor $U > 0$	
$ch = \frac{C}{D}$	processor load factor $ch > 1$	
$CH = \sum_{i=1}^{n} ch_i = \sum_{i=1}^{n} \frac{C_i}{D_i}$	mean processor load factor $CH > 0$	
D(t) = d-t	residual relative (absolute) deadline $0 \le D(t) \le D \quad if\ t \in [r_k, d_k]$ $D(t) < 0 \qquad t > d_k$	
CH(t) = C(t)/D(t)	residual load $CH(t) \in [0,+\infty[\quad t \in [r_k, d_k[$ $C(t) = D(t) \qquad CH(t) = 1$	
L(0) = D-C	nominal laxity	
L(t) = D(t)-C(t)	residual nominal laxity $L(t) \in]-\infty,+\infty[$	



Process and diagram models (5)

e.g. here is a random CPU diagram (i.e. virtual scheduling algorithm) to illustrate CH(t), L(t) vs. C(t), D(t).

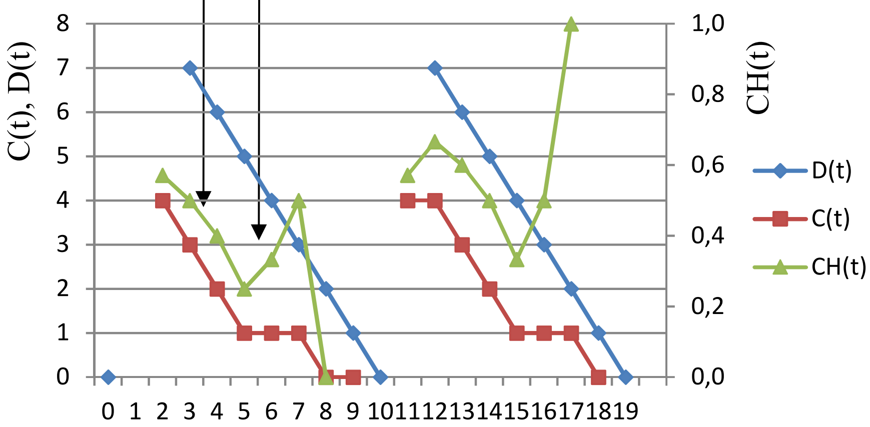
	r0	C	D	T
T	2	4	7	9



t	0-1	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9	9-10	10-11	11-12	12-13	13-14	14-15	15-16	16-17	17-18	18-19	19-20
C(t)			4-3	3-2	2-1	1-1	1-1	1-0				4-4	4-3	3-2	2-1	1-1	1-1	1-0		
D(t)			7-6	6-5	5-4	4-3	3-2	2-1				7-6	6-5	5-4	4-3	3-2	2-1	1-0		
CH(t) 10^{-2}			57-50	50-40	40-25	25-33	33-50	50-0				57-66	66-60	60-50	50-33	33-50	50-100	100-Na		
L(t)			3-3	3-3	3-3	3-2	2-1	1-1				3-2	2-2	2-2	2-2	2-1	1-0	0-0		

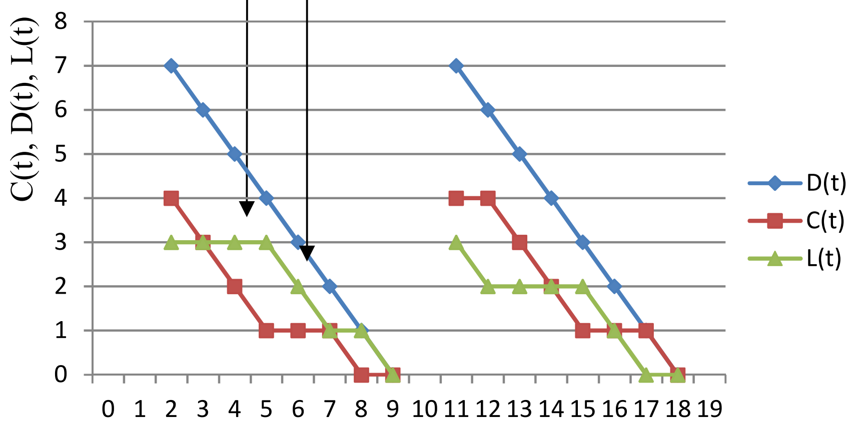
CH(t) decreases when C(t) decreases

CH(t) grows when C(t) is constant



L(t) is constant when the task is running

L(t) decreases when the task is waiting



Real-time scheduling of independent tasks

1. About real-time scheduling
2. Process and diagram models
3. Basic on-line algorithms for periodic tasks
   3.1. Basic scheduling algorithms
   3.2. Sufficient conditions
4. Hybrid task sets scheduling
   4.1. Introduction to hybrid task sets scheduling
   4.2. Hybrid scheduling algorithms

Basic scheduling algorithms

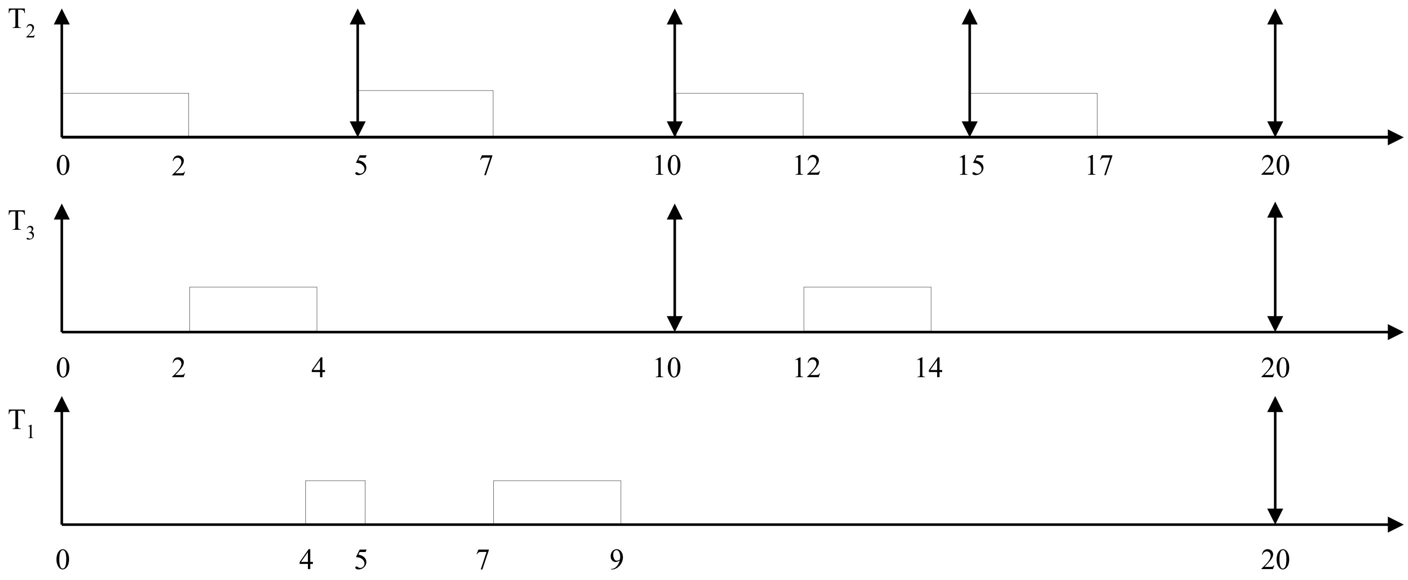
Algorithm	Preemptive	Criterion	Priority	Predictable capacity	Performance criteria and constraints
Rate Monotonic (RM)	yes	T	static	no	easy to implement, cannot use the full processor bandwidth, increase the context switch
Deadline Monotonic (DM)		D			
Earliest Deadline (ED)		D(t)	dynamic	no	hard implementation, can use the full processor bandwidth, limit the context switch, LL supports the best average response time
Least Laxity (LL)		L(t)		yes	

Basic scheduling algorithms
"Rate Monotonic (RM)"

For a set of periodic tasks, assigning the priorities for the Rate Monotonic (RM) algorithm means that tasks with shortest periods T (i.e. the higher request rates) get higher priorities. e.g.

	r_0	C	T
T1	0	3	20
T2	0	2	5
T3	0	2	10

According to the T values and the RM scheduling, priority order is given to T2 (T=5), T3 (T=10) and T1 (T=20)

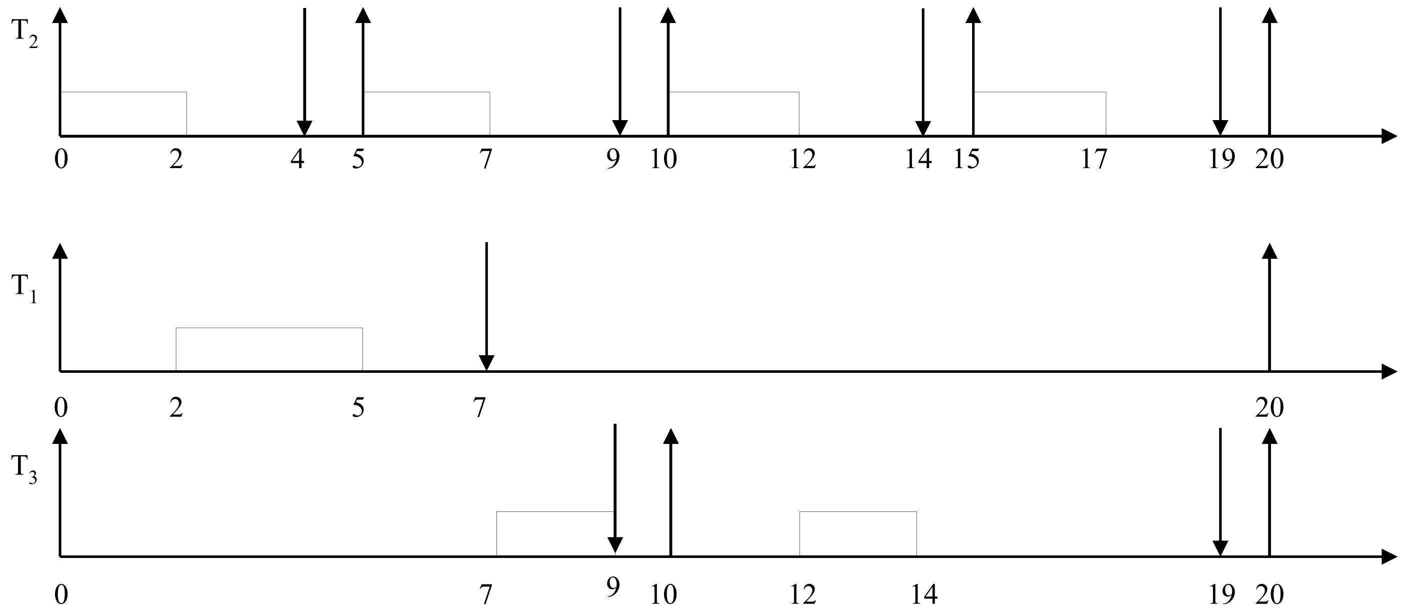


Basic scheduling algorithms
"Deadline Monotonic (DM)"

The Deadline Monotonic (ED), or inverse deadline, algorithm assigns the priorities to tasks according to their relative deadlines D. The task with the shortest relative deadline is assigned to the highest priority. e.g.

	r_0	C	D	T
T1	0	3	7	20
T2	0	2	4	5
T3	0	2	9	10

According to the D values and the DM scheduling, priority order is given to T2 (D=4), T1 (D=7) and T3 (D=9)



Basic scheduling algorithms

Algorithm	Preemptive	Criterion	Priority	Predictable capacity	Performance criteria and constraints
Rate Monotonic (RM)	yes	T	static	no	easy to implement, cannot use the full processor bandwidth, increase the context switch
Deadline Monotonic (DM)		D			
Earliest Deadline (ED)		D(t)	dynamic	no	hard implementation, can use the full processor bandwidth, limit the context switch, LL supports the best average response time
Least Laxity (LL)		L(t)		yes	

Basic scheduling algorithms
"Earliest Deadline (ED)"

The Earliest Deadline (ED), or Earliest Deadline First, algorithm assigns the priorities to tasks according to their residual relative deadline D(t). The task with the earliest absolute deadline will be executed at the highest priority. e.g.

	r_0	C	D	T
T1	0	3	7	20
T2	0	2	4	5
T3	0	1	8	10

According to the D(t) values and the ED scheduling, priority order is given to:

t		0-1	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9	9-10	10-11	11-12	12-13	13-14	14-15	15-16	16-17
T1	**D(t)**	7-6	6-5	5-4	4-3	3-2												
	C(t)	3-3	3-3	3-2	2-1	1-0												
T2	**D(t)**	4-3	3-2				4-3	3-2	2-1			4-3	3-2				4-3	3-2
	C(t)	2-1	1-0				2-2	2-1	1-0			2-1	1-0				2-1	1-0
T3	**D(t)**	8-7	7-6	6-5	5-4	4-3	3-2					8-7	7-6	6-5				
	C(t)	1-1	1-1	1-1	1-1	1-1	1-0					1-1	1-1	1-0				

the task with the lowest D(t) starts first

once C(t) at zero, we shift to the lowest D(t)

T2 restarts at r_0+T

T3 can be executed at the first time

the scheduling will go on ….

Basic scheduling algorithms
“Least Laxity (LL)”

The Least Laxity (LL) algorithm assigns the priorities to tasks according to their nominal residual laxity L(t). The task with the smallest laxity will be executed at the highest priority. e.g.

	r_0	C	D	T
T1	0	3	7	20
T2	0	2	4	5
T3	0	1	8	10

	$L(r_0)$
T1	7-3=4
T2	4-2=2
T3	8-1=7

We compute the values $L(r_0)$ (i.e. the nominal laxity). According to the L(t) values and the LL scheduling, priority order is given to:

t		0-1	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9	9-10	10-11	11-12	12-13	13-14	14-15	15-16	16-17	17-18
T1	**L(t)**	4-3	3-2	2-2	2-2	2-2													
	C(t)	3-3	3-3	3-2	2-1	1-0													
T2	**L(t)**	2-2	2-2				2-2	2-1	1-1			2-2	2-2				2-2	2-2	
	C(t)	2-1	1-0				2-1	1-1	1-0			2-1	1-0				2-1	1-0	
T3	**L(t)**	7-6	6-5	5-4	4-3	3-2	2-1	1-1				7-6	6-5	5-5					
	C(t)	1-1	1-1	1-1	1-1	1-1	1-1	1-0				1-1	1-1	1-0					

T2 of lowest laxity L(t) starts,
L2(t) is constant when T2 running,
L3(t) and L1(t) decrease since T2,T3 are waiting

once C2(t) at zero,
we shift to the lowest Li(t),
T1 is scheduled first

T2 restarts with T=5,
L2(t) and L3(t) are equivalent,
we consider here the task id
T1 > T2 > T3

L3(t) is the lowest,
T3 is scheduled

T3 ends,
T2 restarts

T2 and T3 are starting a new period
L2(t) < L3(t), T2 is scheduled first

only T2 restarts

Real-time scheduling of independent tasks

1. About real-time scheduling
2. Process and diagram models
3. Basic on-line algorithms for periodic tasks
   3.1. Basic scheduling algorithms
   3.2. Sufficient conditions
4. Hybrid task sets scheduling
   4.1. Introduction to hybrid task sets scheduling
   4.2. Hybrid scheduling algorithms

Sufficient conditions
"Introduction"

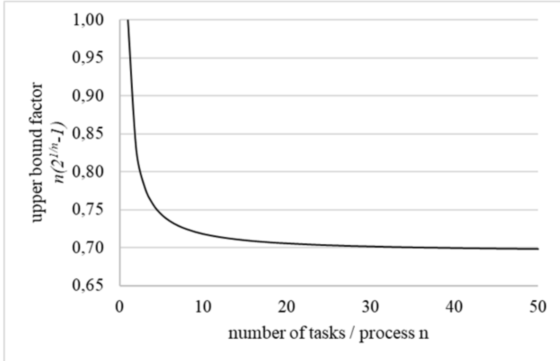
A set of periodic task is schedulable with the RM, DM, ED and LL algorithms if they respect the following sufficient conditions. A sufficient condition is one that, if satisfied, assures the statement's truth. (i.e. a necessary condition of a statement must be satisfied for the statement to be true).

Rate Monotonic $\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$ — i.e. mean utilization processor factor lowest to an upper bound factor $n(2^{1/n}-1)$

Deadline Monotonic $\sum_{i=1}^{n} \frac{C_i}{D_i} \leq n(2^{1/n} - 1)$

Earliest Deadline, Least Laxity $\sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1$

(Deadline Monotonic, Earliest Deadline, Least Laxity) — i.e. mean load factor lowest to an upper bound factor, either $n(2^{1/n}-1)$ either 1



e.g.

	C	D	T
T1	1	5	5
T2	2	4	7
T3	2	7	8

$$n(2^{1/n} - 1) = 3\left(2^{1/3} - 1\right) = 0{,}7798$$

$$\sum_{i=1}^{n} \frac{C_i}{T_i} = \frac{1}{5} + \frac{2}{7} + \frac{2}{8} = 0{,}7357$$

$$\sum_{i=1}^{n} \frac{C_i}{D_i} = \frac{1}{5} + \frac{2}{4} + \frac{2}{7} = 0{,}9857$$

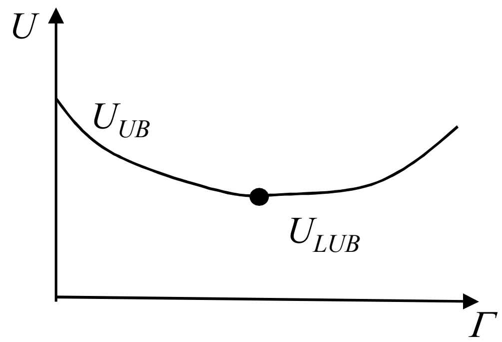
$0{,}7357 \leq 0{,}7798$ can be scheduled with Rate Monotonic

$0{,}9857 \geq 0{,}7798$ can't be scheduled with Deadline Monotonic

$0{,}9857 \leq 1$ can be scheduled with Earliest Deadline and Least Laxity

Sufficient conditions
"Calculation of the Least Upper Bound U_{LUB}" (1)

	Equation	Comments
Utilization factor (U)	$U = \sum_{i=1}^{n} \frac{C_i}{T_i}$	Given a set of *n* periodic tasks, the utilization factor *U* is the fraction of processor time spent in the execution of the task set.
Upper Bound (U_{UB})	$U = U_{UB}(\Gamma, A)$	Let $U_{UB}(\Gamma,A)$ be the upper bound of the processor utilization factor: • for a task set Γ, • under a given algorithm A, when $U = U_{UB}(\Gamma,A)$, the set Γ is said to fully utilize the processor.
Least Upper Bound (U_{LUB})	$U_{LUB}(A) = \min U_{UB}(\Gamma, A)$	For a given algorithm A, the least upper bound U_{LUB} of the processor utilization factor is the minimum of the utilization factors over all task sets Γ that fully utilize the processor.



Sufficient conditions
"Calculation of the Least Upper Bound U_{LUB}" (2)

	Equation	Comments
Utilization factor (U)	$U = \sum_{i=1}^{n} \frac{C_i}{T_i}$	Given a set of *n* periodic tasks, the utilization factor *U* is the fraction of processor time spent in the execution of the task set.
Upper Bound (U_{UB})	$U = U_{UB}(\Gamma, A)$	Let $U_{UB}(\Gamma,A)$ be the upper bound of the processor utilization factor: • for a task set Γ, • under a given algorithm *A*, when $U = U_{UB}(\Gamma,A)$, the set Γ is said to fully utilize the processor.
Least Upper Bound (U_{LUB})	$U_{LUB}(A) = \min U_{UB}(\Gamma, A)$	For a given algorithm *A*, the least upper bound U_{LUB} of the processor utilization factor is the minimum of the utilization factors over all task sets Γ that fully utilize the processor.

U_{LUB} defines an important characteristic of a scheduling algorithm because it allows to easily verify the schedulability of a task set:

- Any task set whose processor utilization factor *U* is below U_{LUB} is schedulable by *A*.
- On the other hand, utilization factor *U* above U_{LUB} can be achieved only if the periods of the tasks are suitably related.

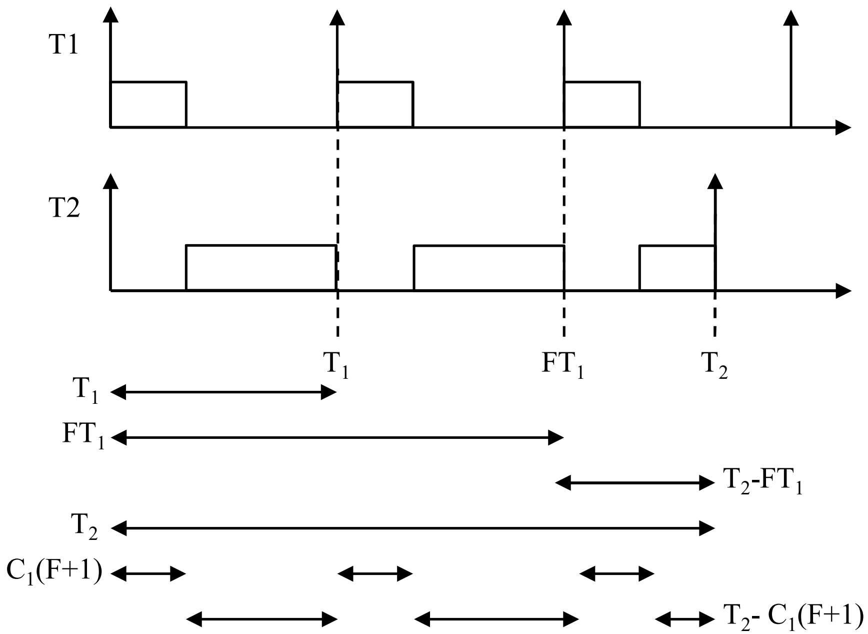
Sufficient conditions
"Calculation of the Least Upper Bound U_{LUB}" (3)

e.g. Consider a set of two periodic tasks T1, T2 with T1 < T2, in order to compare U_{LUB} with the RM algorithm, we have:

- To assign priorities to tasks according to RM, so that T1 is the task with the shortest period.
- To compute the Upper Bound U_{UB} for the set of setting task's computation times to fully utilize the processor.
- To minimize the Upper Bound U_{UB}, to get the U_{LUB}, with respect to all the other task parameters.

To do this, we adjust the computation time of T2 to fully utilize the processor, two cases must be considered.

Case 1: The computation time is short enough that all the requests of T1 within the critical zone of T2 are completed before the second request of T2.



Let T_1, T_2, C_1, C_2 be the periods and capacities of tasks T1, T2 respectively.	
Let F be the number of periods of T1 entirely contained in T2.	$F = \left\lfloor T_2 / T_1 \right\rfloor$
That is,	$C_1 \leq T_2 - FT_1$
In this situation, the largest possible value for C_2 is	$C_2 = T_2 - C_1(F+1)$

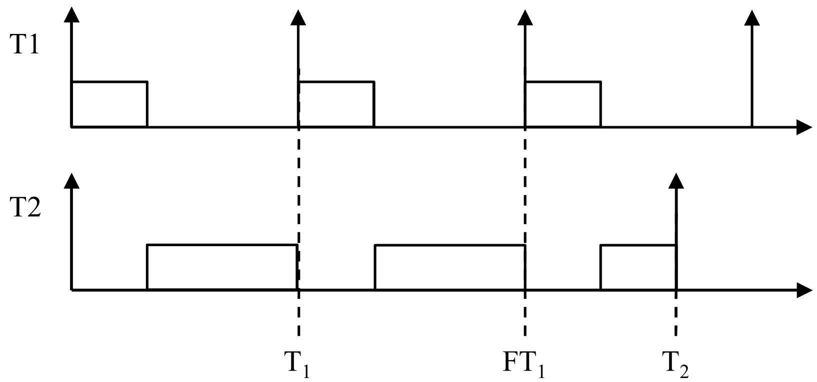
Sufficient conditions
"Calculation of the Least Upper Bound U_{LUB}" (4)

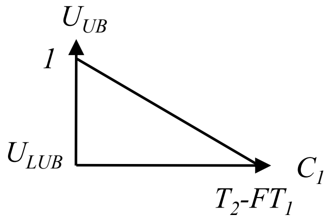
e.g. Consider a set of two periodic tasks T1, T2 with T1 < T2, in order to compare U_{LUB} with the RM algorithm, we have:

- To assign priorities to tasks according to RM, so that T1 is the task with the shortest period.
- To compute the Upper Bound U_{UB} for the set of setting task's computation times to fully utilize the processor.
- To minimize the Upper Bound U_{UB}, to get the U_{LUB}, with respect to all the other task parameters.

To do this, we adjust the computation time of T2 to fully utilize the processor, two cases must be considered.

Case 1: The computation time is short enough that all the requests of T1 within the critical zone of T2 are completed before the second request of T2.





Considering the largest possible value for C_2 , the corresponding Upper Bound U_{UB} is then,	$U_{UB} = \frac{C_1}{T_1} + \frac{C_2}{T_2}$ $U_{UB} = \frac{C_1}{T_1} + \frac{T_2 - C_1(F+1)}{T_2}$ $U_{UB} = 1 + \frac{C_1}{T_2}\left(\frac{T_2}{T_1} - (F+1)\right)$
Since the quantity in brackets is negative, U_{UB} is monotonically decreasing in C_1, and being the minimum of U_{UB} then U_{LUB} occurs for	$\left(\frac{T_2}{T_1} - (F+1)\right)$ $C_1 \le T_2 - FT_1$ $C_1 = T_2 - FT_1$

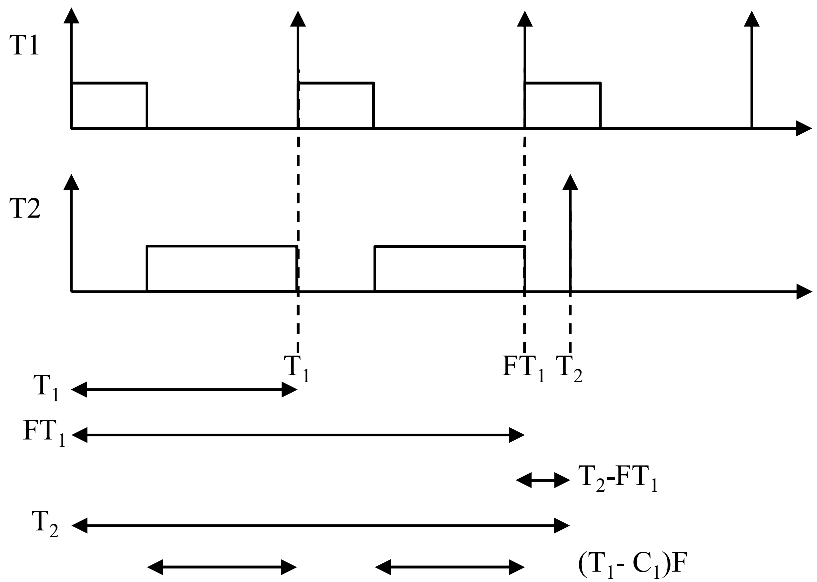
Sufficient conditions
"Calculation of the Least Upper Bound U_{LUB}" (5)

e.g. Consider a set of two periodic tasks T1, T2 with T1 < T2, in order to compare U_{LUB} with the RM algorithm, we have:

- To assign priorities to tasks according to RM, so that T1 is the task with the shortest period.
- To compute the Upper Bound U_{UB} for the set of setting task's computation times to fully utilize the processor.
- To minimize the Upper Bound U_{UB}, to get the U_{LUB}, with respect to all the other task parameters.

To do this, we adjust the computation time of T2 to fully utilize the processor, two cases must be considered.

Case 2: The execution of the last request of T1 in the critical time zone of T2 overlaps the second request of T2.



Let T_1, T_2, C_1, C_2 be the periods and capacities of tasks T1, T2 respectively.	
Let F be the number of periods of T1 entirely contained in T2.	$F = \left\lfloor T_2 / T_1 \right\rfloor$
That is,	$C_1 \geq T_2 - FT_1$
In this situation, the largest possible value for C_2 is	$C_2 = (T_1 - C_1)F$

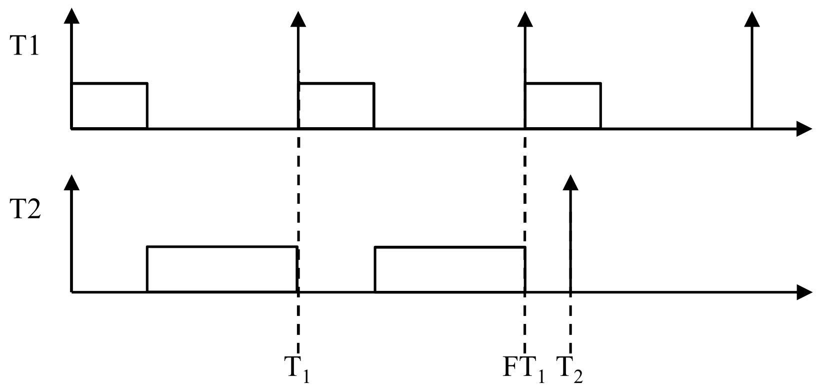
Sufficient conditions
"Calculation of the Least Upper Bound U_{LUB}" (6)

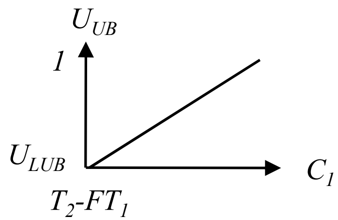
e.g. Consider a set of two periodic tasks T1, T2 with T1 < T2, in order to compare U_{LUB} with the RM algorithm, we have:

- To assign priorities to tasks according to RM, so that T1 is the task with the shortest period.
- To compute the Upper Bound U_{UB} for the set of setting task's computation times to fully utilize the processor.
- To minimize the Upper Bound U_{UB}, to get the U_{LUB}, with respect to all the other task parameters.

To do this, we adjust the computation time of T2 to fully utilize the processor, two cases must be considered.

Case 2: The execution of the last request of T1 in the critical time zone of T2 overlaps the second request of T2.





Considering the largest possible value for C_2, the corresponding Upper Bound U_{UB} is then,	$U_{UB} = \frac{C_1}{T_1} + \frac{C_2}{T_2}$ $U_{UB} = \frac{C_1}{T_1} + \frac{(T_1 - C_1)F}{T_2}$ $U_{UB} = \frac{T_1}{T_2}F + \frac{C_1}{T_2}\left(\frac{T_2}{T_1} - F\right)$
Since the quantity in brackets is positive, U_{UB} is monotonically increasing in C_1, and being the minimum of U_{UB} then U_{LUB} occurs for	$\left(\frac{T_2}{T_1} - F\right)$ $C_1 \geq T_2 - FT_1$ $C_1 = T_2 - FT_1$

Sufficient conditions
"Calculation of the Least Upper Bound U_{LUB}" (7)

e.g. Consider a set of two periodic tasks T1, T2 with T1 < T2, in order to compare U_{LUB} with the RM algorithm, we have:

- To assign priorities to tasks according to RM, so that T1 is the task with the shortest period.
- To compute the Upper Bound U_{UB} for the set of setting task's computation times to fully utilize the processor.
- To minimize the Upper Bound U_{UB}, to get the U_{LUB}, with respect to all the other task parameters.

To do this, we adjust the computation time of T2 to fully utilize the processor, two cases must be considered.

In both cases 1 and 2:

the minimum of U_{UB} then U_{LUB} occurs for	$C_1 = T_2 - FT_1$
Considering the minimum value C_1 within the Upper Bound U_{UB} calculation of case 2 we have	$U_{UB} = \frac{T_1}{T_2}F + \frac{C_1}{T_2}\left(\frac{T_2}{T_1} - F\right) = \frac{T_1}{T_2}F + \frac{T_2 - FT_1}{T_2}\left(\frac{T_2}{T_1} - F\right)$ $U_{UB} = \frac{T_1}{T_2}\left(F + \left(\frac{T_2}{T_1} - F\right)^2\right)$
To simplify the notation, let $G = \left(\frac{T_2}{T_1} - F\right)$	$U_{UB} = \frac{T_1}{T_2}\left(F + G^2\right) = \frac{\left(F + G^2\right)}{T_2/T_1} = \frac{\left(F + G^2\right)}{\left(T_2/T_1 - F\right) + F} = \frac{\left(F + G^2\right)}{F + G}$ $U_{UB} = \frac{(F + G) - \left(G - G^2\right)}{F + G} = 1 - \frac{G(1 - G)}{F + G}$

Sufficient conditions
"Calculation of the Least Upper Bound U_{LUB}" (8)

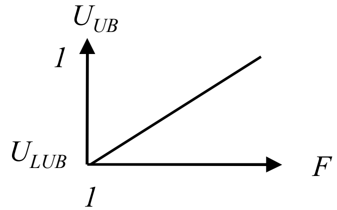
e.g. Consider a set of two periodic tasks T1, T2 with T1 < T2, in order to compare U_{LUB} with the RM algorithm, we have:

- To assign priorities to tasks according to RM, so that T1 is the task with the shortest period.
- To compute the Upper Bound U_{UB} for the set of setting task's computation times to fully utilize the processor.
- To minimize the Upper Bound U_{UB}, to get the U_{LUB}, with respect to all the other task parameters.

To do this, we adjust the computation time of T2 to fully utilize the processor, two cases must be considered.

In both cases 1 and 2:

Since	$0 \leq G < 1$
with	$G = \left(\frac{T_2}{T_1} - F \right)$ $\quad F = \left\lfloor T_2 / T_1 \right\rfloor$
the term	$G(1-G)$
is non negative, hence U_{UB}	$U_{UB} = 1 - \frac{G(1-G)}{F+G}$
is monotonically increasing in F, and being the minimum value of F of U_{UB} then U_{LUB} occurs for	$F = 1$



Sufficient conditions
"Calculation of the Least Upper Bound U_{LUB}" (9)

e.g. Consider a set of two periodic tasks T1, T2 with T1 < T2, in order to compare U_{LUB} with the RM algorithm, we have:

- To assign priorities to tasks according to RM, so that T1 is the task with the shortest period.
- To compute the Upper Bound U_{UB} for the set of setting task's computation times to fully utilize the processor.
- To minimize the Upper Bound U_{UB}, to get the U_{LUB}, with respect to all the other task parameters.

To do this, we adjust the computation time of T2 to fully utilize the processor, two cases must be considered.

In both cases 1 and 2:

Minimizing U over G with	$U_{UB} = \frac{\left(F + G^2\right)}{F + G}$
we have	$U_{UB} = \frac{\left(1 + G^2\right)}{1 + G}$
the first derivative is	$\frac{dU_{UB}}{dG} = \frac{2G(1+G) - (1+G^2)}{(1+G^2)} = \frac{G^2 + 2G - 1}{(1+G^2)}$
we can fix	$\frac{dU_{UB}}{dG} = 0$
for with	$G^2 + 2G - 1 = 0$ $G_1 = -1 - \sqrt{2}$ $G_2 = -1 + \sqrt{2}$
the negative solution is discarded and	$U_{LUB} = U_{UB}(G_2) = \frac{\left(1 + \left(\sqrt{2} - 1\right)^2\right)}{1 + \left(\sqrt{2} - 1\right)} = \frac{4 - 2\sqrt{2}}{\sqrt{2}} = 2\left(\sqrt{2} - 1\right) = 0.83$

Real-time scheduling of independent tasks

1. About real-time scheduling
2. Process and diagram models
3. Basic on-line algorithms for periodic tasks
   3.1. Basic scheduling algorithms
   3.2. Sufficient conditions
4. Hybrid task sets scheduling
   4.1. Introduction to hybrid task sets scheduling
   4.2. Hybrid scheduling algorithms

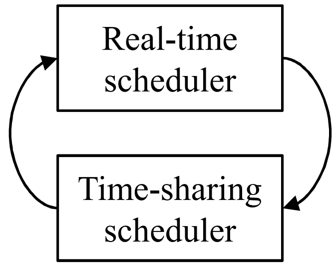
Introduction to hybrid task sets scheduling (1)

Basic on-line algorithms deal with homogeneous set of tasks where all are periodic. However, some real-time applications may require aperiodic tasks.

	Use	Constraint
periodic	regular event in the system	strict deadline
aperiodic	irregular event in the system	could be strict or relative

Hybrid task set scheduling deals with the both type of task. Such a scheduling is based on hybrid scheduler, composed of a real-time scheduler combined with a time-sharing one. Shifting between the two levels is controlled according to some go-up and go-down criteria.

(2) Go up: the time-sharing level shifts to the real-time one regarding a criterion β.



(1) Go down: the real-time level shifts to the time-sharing one regarding a criterion α.

Two main approaches exist to design hybrid schedulers:

(1) the background/joint processing exploits the free idle time of the processor to schedule the aperiodic tasks, or to schedule jointly the aperiodic and the periodic tasks.

(2) the server based processing implements a virtual periodic task (i.e. the server) in charge to schedule the aperiodic tasks.

Introduction to hybrid task sets scheduling (2)

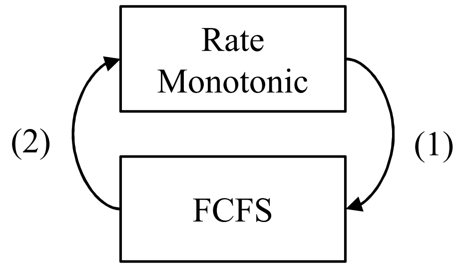
Algorithms	scheduler type	Schedulers		periodic→ aperiodic		aperiodic→ periodic		Predictable capacity	Performance criteria and constraints
		periodic	aperiodic	preemptive	criterion	preemptive	criterion		
Background	background	RM/DM	FCFS	no	idle time≠0	yes	idle time=0	no	worst response times for aperiodic requests, minor issues for implementation
Slack Stealing	background	RM/DM	FCFS	yes	L(t)>0	yes	L(t)=0	yes	optimum response times for aperiodic requests at a high aperiodic load, hard implementation issues
Pooling	Fixed-priority server	RM/DM	FCFS	yes	poling at the start time	yes	limit of capacities	no	little improvement compared to the background processing
Deferrable Server	Fixed-priority server	RM/DM	FCFS	yes	polling at any time	yes	limit of capacities	no	a better average response time for aperiodic requests, mainly with SS
Sporadic Sever	Fixed-priority server	RM/DM	FCFS	yes	polling at any time	yes	limit of capacities	no	a better average response time for aperiodic requests, mainly with SS
Priority Exchange	Fixed-priority server	RM/DM	FCFS	yes	polling at any time	yes	limit of capacities	no	optimum response times for short aperiodic requests

Real-time scheduling of independent tasks

1. About real-time scheduling
2. Process and diagram models
3. Basic on-line algorithms for periodic tasks
   3.1. Basic scheduling algorithms
   3.2. Sufficient conditions
4. Hybrid task sets scheduling
   4.1. Introduction to hybrid task sets scheduling
   4.2. Hybrid scheduling algorithms

<table>
<tr><th rowspan="2">Algorithms</th><th rowspan="2">scheduler type</th><th colspan="2">Schedulers</th><th colspan="2">periodic→ aperiodic</th><th colspan="2">aperiodic→ periodic</th><th rowspan="2">Predictable capacity</th><th rowspan="2">Performance criteria and constraints</th></tr>
<tr><th>periodic</th><th>aperiodic</th><th>preemptive</th><th>criterion</th><th>preemptive</th><th>criterion</th></tr>
<tr><td>Background</td><td rowspan="2">background</td><td rowspan="2">RM/DM</td><td rowspan="2">FCFS</td><td>no</td><td>idle time≠0</td><td rowspan="2">yes</td><td>idle time=0</td><td>no</td><td>worst response times for aperiodic requests, minor issues for implementation</td></tr>
<tr><td>Slack Stealing</td><td>yes</td><td>L(t)>0</td><td>L(t)=0</td><td>yes</td><td>optimum response times for aperiodic requests at a high aperiodic load, hard implementation issues</td></tr>
<tr><td>Pooling</td><td rowspan="4">Fixed-priority server</td><td rowspan="4">RM/DM</td><td rowspan="4">FCFS</td><td rowspan="4">yes</td><td>poling at the start time</td><td rowspan="4">yes</td><td rowspan="4">limit of capacities</td><td rowspan="4">no</td><td>little improvement compared to the background processing</td></tr>
<tr><td>Deferrable Server</td><td rowspan="3">polling at any time</td><td rowspan="2">a better average response time for aperiodic requests, mainly with SS</td></tr>
<tr><td>Sporadic Sever</td></tr>
<tr><td>Priority Exchange</td><td>optimum response times for short aperiodic requests</td></tr>
</table>

Hybrid task set scheduling
"Background scheduling"



Aperiodic tasks are scheduled on the processor idle time once all the periodic tasks end. Periodic and aperiodic tasks are scheduled according to RM and FCFS strategies, respectively. e.g.

(1) If they are no periodic task ready to be executed.
(2) Whenever a periodic task restarts.

	r_0	C	T
Tp_1	0	2	5
Tp_2	0	2	10
Ta_1	4	2	Na
Ta_2	10	1	Na
Ta_3	11	2	Na

t		0-1	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9	9-10	10-11	11-12	12-13	13-14	14-15	15-16	16-17	17-18	18-19
Tp_1	C(t)	2-1	1-0				2-1	1-0				2-1	1-0				2-1	1-0		
Tp_2	C(t)	2-2	2-2	2-1	1-0							2-2	2-2	2-1	1-0					
Idle time																				
Ta_1	C(t)					2-1	1-1	1-1	1-0											
Ta_2	C(t)											1-1	1-1	1-1	1-1	1-0				
Ta_3	C(t)												2-2	2-2	2-2	2-2	2-2	2-2	2-1	1-0

RM scheduling between Tp_1, Tp_2

the first idle time, Ta_1 is ready and can be scheduled

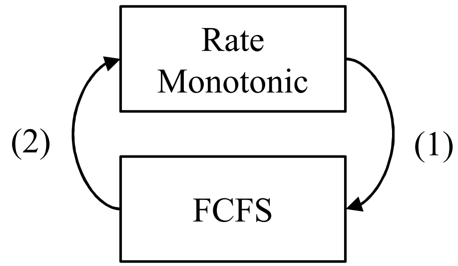
the idle time is over when Tp_1 restarts

when Tp_1 ends, a new idle time slot appears, but too large comparing to the C(t) of Ta_1

Ta_2, Ta_3 blocked while periodic tasks are running

the scheduling will go on …

Hybrid task set scheduling
“Slack stealing”



Each time an aperiodic task enters in the system, time for servicing this aperiodic task is made by “stealing” processing time from the periodic tasks looking for laxity without causing a deadline missing. e.g.

(1) If the residual nominal laxities Li(t) of periodic tasks are up to zero.

(2) Whenever a residual nominal laxity Li(t) of a periodic task goes down to zero.

	r_0	C	D=T
Tp_1	0	2	5
Tp_2	0	2	10
Ta_1	4	2	Na
Ta_2	10	1	Na
Ta_3	11	3	Na

t		0-1	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9	9-10	10-11	11-12	12-13	13-14	14-15	15-16	16-17	17-18
Tp_1	C(t)	2-1	1-0				2-2	2-1	1-0			2-2	2-2	2-2	2-1	1-0	2-2	2-1	1-0
	L(t)	3-3	3-3				3-2	2-2	2-2			3-2	2-1	1-0	0-0	0-0	3-2	2-2	2-2
Tp_2	C(t)	2-2	2-2	2-1	1-0							2-2	2-2	2-2	2-2	2-2	2-2	2-2	2-2
	L(t)	8-7	7-6	6-6	6-6							8-7	7-6	6-5	5-4	4-3	3-2	2-1	1-0
Ta_1	C(t)					2-1	1-0												
Ta_2	C(t)											1-0							
Ta_3	C(t)												3-2	2-1	1-1	1-1	1-0		

RM scheduling between Tp_1, Tp_2

Ta_1 starts at the first idle time

Tp_1 restarts with L1(t)>0, Ta_1 continues

when Ta_1 ends, Tp_1 can be scheduled

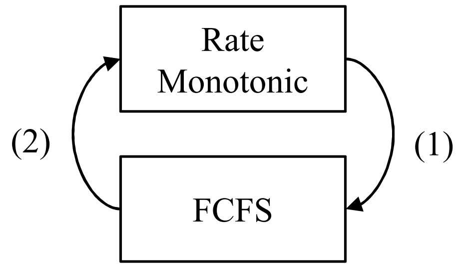
Tp_1, Tp_2 blocked while the aperiodic tasks are running

when L1(t)=0 for Tp_1, Tp_1 preempts the aperiodic tasks

the scheduling will go on …

Algorithms	scheduler type	Schedulers		periodic→ aperiodic		aperiodic→ periodic		Predictable capacity	Performance criteria and constraints
		periodic	aperiodic	preemptive	criterion	preemptive	criterion		
Background	background	RM/DM	FCFS	no	idle time $\neq 0$	yes	idle time=0	no	worst response times for aperiodic requests, minor issues for implementation
Slack Stealing				yes	L(t)>0		L(t)=0	yes	optimum response times for aperiodic requests at a high aperiodic load, hard implementation issues
Pooling	Fixed-priority server	RM/DM	FCFS	yes	poling at the start time	yes	limit of capacities	no	little improvement compared to the background processing
Deferrable Server					polling at any time				a better average response time for aperiodic requests, mainly with SS
Sporadic Sever									
Priority Exchange									optimum response times for short aperiodic requests

Hybrid task set scheduling "Pooling Server (PS)"



The Pooling Server (PS) becomes active at regular intervals equal to its period and serves the aperiodic tasks within its capacity. If none aperiodic task is waiting, the polling server suspends itself until the beginning of its next period, and releases time to periodic tasks. e.g.

(1) Whenever the server starts its period with aperiodic task(s) waiting for him.
(2) If the server ends its capacity, or none aperiodic task is waiting.

	r_0	C	T
Tp_s	0	2	5
Tp_1	0	3	20
Tp_2	0	2	10
Ta_1	4	2	Na
Ta_2	10	1	Na
Ta_3	11	2	Na

t	0-1	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9	9-10	10-11	11-12	12-13	13-14	14-15	15-16	16-17	17-18	18-19
Tp_s	2-0					2-1	1-0				2-1	1-0				2-0			
Tp_2	2-1	1-0									2-2	2-2	2-1	1-0					
Tp_1			3-2	2-1	1-0														
Ta_1					2-2	2-1	1-0												
Ta_2											1-0								
Ta_3												2-1	1-1	1-1	1-1	1-0			

no aperiodic requests are pending, the server Tp_s suspends itself

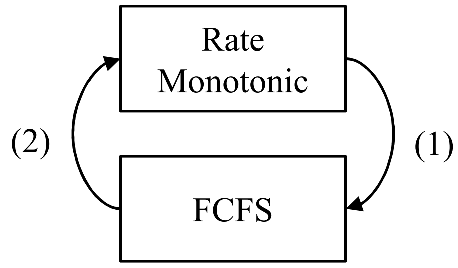
the server capacity is not preserved for aperiodic execution, Ta_1 must wait until the beginning of the next pooling period

Tp_s restarts, Ta_1 is scheduled

Tp_s is active and serves any pending requests within the limit of its capacity

the scheduling will go on …

Hybrid task set scheduling
“Deferrable Server (DS)”



The Deferrable Server (DS) looks like a polling server. However, it preserves its capacity if no request are pending upon the invocation of the server. The capacity is maintained until the end of the period. This improves the average response time of the aperiodic requests. e.g.

(1) Whenever the server can scheduled aperiodic tasks with respect to its priority and remaining capacity.
(2) If the server ends its capacity, or none aperiodic task is waiting.

	r_0	C	T
Tp_s	0	2	5
Tp_1	0	1	4
Tp_2	0	2	6
Ta_1	2	2	Na
Ta_2	9	1	Na
Ta_3	13	2	Na

t	0-1	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9	9-10	10-11	11-12	12-13	13-14	14-15	15-16	16-17	17-18
Tp_1	1-0				1-0				1-0				1-0				1-0	
Tp_s	2-2	2-2	2-1	1-0		2-2	2-2	2-2	2-2	2-1	2-2	2-2	2-2	2-1	1-0	2-2	2-2	2-2
Tp_2	2-2	2-1	1-1	1-1	1-1	1-0	2-1	1-0					2-2	2-2	2-2	2-1		1-0
Ta_1			2-1	1-0														
Ta_2										1-0								
Ta_3														2-1	1-0			

capacity of the server is maintained while none aperiodic request is here

Ta_1 is in the ready queue, Tp_s with a higher priority preempts Tp_2 in respect with the pooling service

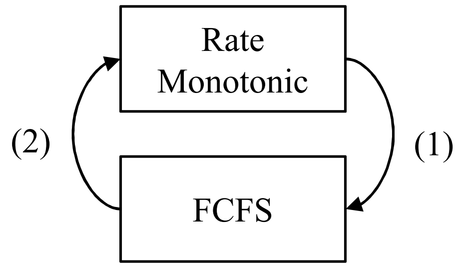
at any new period, the server Tp_s reloads its capacity

Ta_2 is scheduled in respect with the pooling service

as Ta_1, Tp_s with a higher priority preempts Tp_2 to schedule Ta_2 in respect with the pooling service

the scheduling will go on …

Hybrid task set scheduling "Sporadic Server (SS)"



The Sporadic Server (SS) preserves its capacity until an aperiodic task occurs. When it processes a set of task as first time (at t_0) it must wait a time equals to T_s (its period) to replenish its capacity. A count down R(t) can be computed like $R(t) = t_0 + T_s - t$ with $t \geq t_0$

(1) Whenever the server starts its period with aperiodic task(s) waiting for him.

(2) If the server ends its capacity, or none aperiodic task is waiting.

e.g.

	r_0	C	T
Tp_s	0	2	5
Tp_1	0	3	20
Tp_2	0	2	10
Ta_1	4	2	Na
Ta_2	10	1	Na
Ta_3	11	2	Na

t		0-1	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9	9-10	10-11	11-12	12-13	13-14	14-15	15-16	16-17	17-18
Tp_s	C(t)	2-2	2-2	2-2	2-2	2-1	1-0	0-0	0-0	0-0	2-2	2-1	1-0	0-0	0-0	0-0	2-1	1-1	1-1
	R(t)	∞	∞	∞	∞	5-4	4-3	3-2	2-1	1-0	∞	5-4	4-3	3-2	2-1	1-0	5-4	4-3	3-2
Tp_2		2-1	1-0									2-2	2-2	2-1	1-0				
Tp_1				3-2	2-1	1-1	1-1	1-0											
Ta_1						2-1	1-0												
Ta_2												1-0							
Ta_3													2-1	1-1	1-1	1-1	1-0		

capacity of the server is maintained while none aperiodic request is here

Ta_1 is scheduled in respect with the pooling service, the replenishment time is set to $R(t) = T_s$ with $t_0 = t$

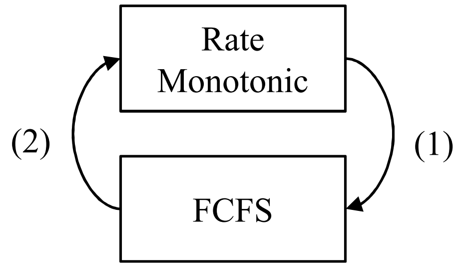
Tp_s is active and serves any pending requests within the limit of its capacity

at $t = t_0 + T_s$ we have $R(t) = 0$, the replenishment amount is set to the capacity consumed within the interval $[t_0, t_0 + T_s]$

the scheduling will go on …

Algorithms	scheduler type	Schedulers		periodic→ aperiodic		aperiodic→ periodic		Predictable capacity	Performance criteria and constraints
		periodic	aperiodic	preemptive	criterion	preemptive	criterion		
Background	background	RM/DM	FCFS	no	idle time≠0	yes	idle time=0	no	worst response times for aperiodic requests, minor issues for implementation
Slack Stealing				yes	L(t)>0		L(t)=0	yes	optimum response times for aperiodic requests at a high aperiodic load, hard implementation issues
Pooling	Fixed-priority server	RM/DM	FCFS	yes	poling at the start time	yes	limit of capacities	no	little improvement compared to the background processing
Deferrable Server					polling at any time				a better average response time for aperiodic requests, mainly with SS
Sporadic Sever									
Priority Exchange									optimum response times for short aperiodic requests

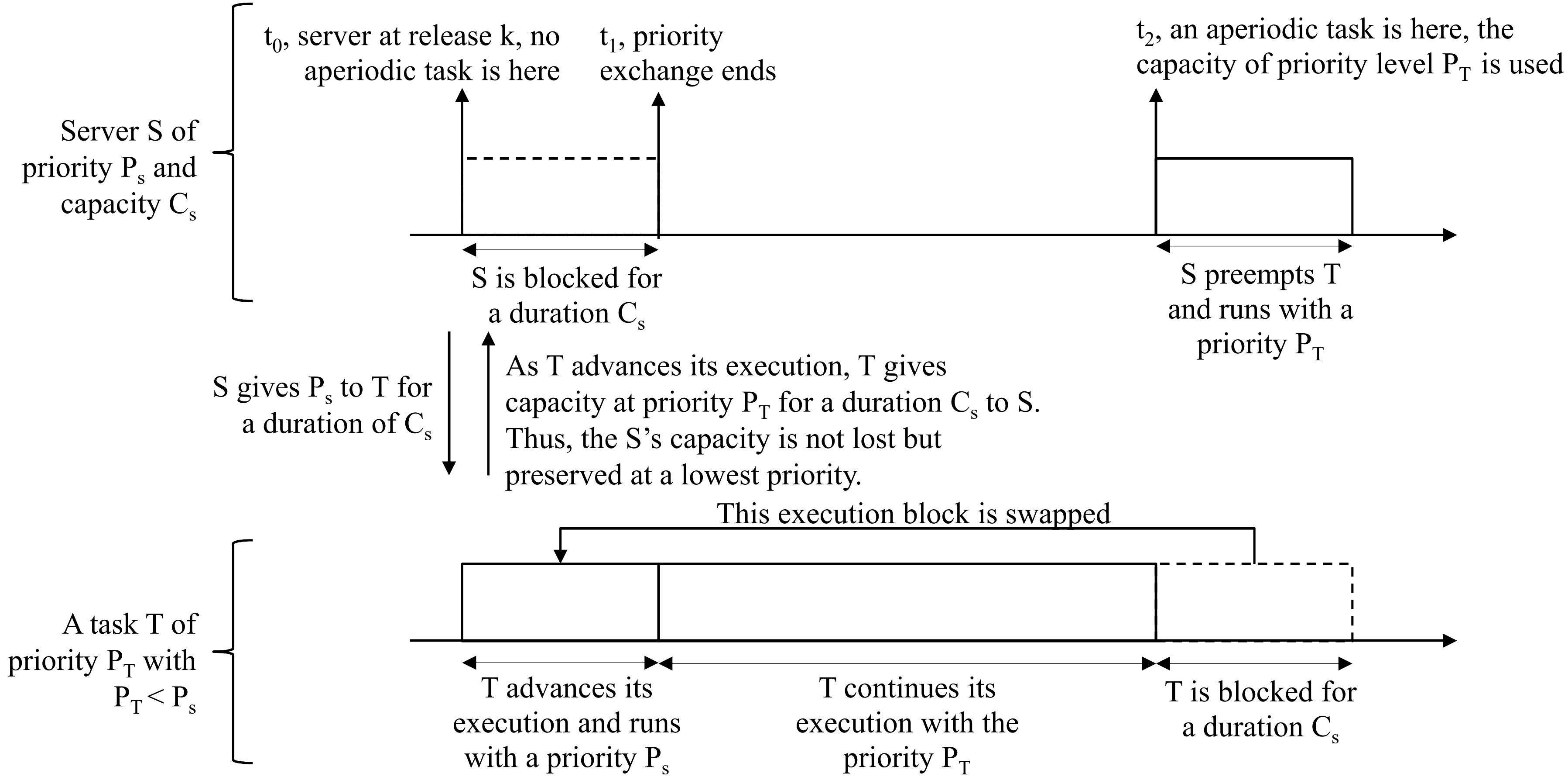
Hybrid task set scheduling "Priority Exchange (PE)" (1)



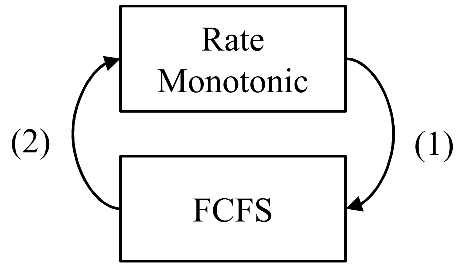
(1) Whenever the server can use some (accumulated or not) capacities.

(2) If no server capacities are available, or if a task with higher priority occurs.

Like the Deferrable server (DS), Priority Exchange (PE) algorithm uses a periodic task for servicing aperiodic requests. However, it differs from DS in the manner in which the capacity is preserved. PE preserves its capacity by exchanging it for the execution time of a lower priority task.



Hybrid task set scheduling
"Priority Exchange (PE)" (2)



(1) Whenever the server can use some (accumulated or not) capacities.
(2) If none server capacity is available, or if a task with higher priority occurs.

The Priority Exchange (PE) can be defined as follows:

- Like the pooling and the deferrable servers, the PE algorithm uses a periodic task (usually at a high priority) for servicing aperiodic requests.

- At the beginning of each server period, the capacity is replenished at its full value.

- Like the deferrable server, if aperiodic requests are pending and the server is the ready task with the highest priority, then the requests are serviced using the available capacity.

- If no aperiodic task exists, the high priority server exchanges its priority with a lower priority periodic task (the next priority) for a duration of C_s, where C_s is the remaining computation time of the server. Thus, the priority task advances its execution, and the server capacity is not lost but preserved at a lowest priority.

- If no periodic and aperiodic requests arrive to use the capacity, priority exchange continues with other periodic tasks until either the capacity is used for aperiodic services or either it is degraded to the priority level of background processing.

- Otherwise, if aperiodic requests are pending the capacity accumulated at lowest priority levels are used to execute the aperiodic requests from highest to lowest priorities. When the server runs at a lowest priority level, it preempts the periodic tasks at the same level of priority.

Hybrid task set scheduling “Priority Exchange (PE)” (3)

e.g. Tp_s accumulates capacities from Tp_1, Tp_2:

- the capacity of Tp_1 is used to process the latest aperiodic release Ta_2.
- the capacity of Tp_2 is degraded to the priority level of background.

	r_0	C	T	P
Tp_s	0	1	5	0
Tp_1	0	4	10	1
Tp_2	0	8	20	2
Ta_1	5	1	Na	Na
Ta_2	12	1	Na	Na

t		0-1	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9	9-10	10-11	11-12	12-13	13-14	14-15	15-16	16-17	17-18	18-19	19-20
Tp_s	**C0(t)**	1-0					1-0					1-0					1-0				
	C1(t)	0-1	1	1	1	1-0						0-1	1	1-0							
	C2(t)					0-1	1	1	1	1	1	1	1	1	1	1	1-2	2	2	2-1	1-0
	P(t)	1	0	0	0	2	0	0	0	0	0	1	0	1	0	0	2	0	0	0	0
Tp_1	**C(t)**	4-3	3-2	2-1	1-0							4-3	3-2	2	2-1	1-0					
	P(t)	0	1	1	1	1	1	1	1	1	1	0	1	1	1	1					
Tp_2	**C(t)**	8	8	8	8	8-7	7	7-6	6-5	5-4	4-3	3	3	3	3	3	3-2	2-1	1-0		
	P(t)	2	2	2	2	0	2	2	2	2	2	2	2	2	2	2	0	2	2		
Ta_1	**C(t)**						1-0														
Ta_2	**C(t)**													1-0							

- a priority exchange occurs between Tp_s and Tp_1
- Tp_s accumulates a capacity of value C_s at the priority level of the Tp_1
- Tp_s exchanges its priority with Tp_1 for a duration of C_s

after C_s, Tp_s recovers its nominal priority

- no aperiodic request arrive, priority exchange shifts to Tp_2
- Tp_s shifts the accumulated capacity from Tp_1 to Tp_2
- Tp_s exchanges its priority with Tp_2 for a duration of C_s

an aperiodic request arrives while the server is restarting, Tp_s uses its capacity C_s to process Ta_1

Tp_s is restarting while none aperiodic request is here, a priority exchange occurs with Tp_1

Ta_2 is in the queue while the capacity C_s is null, Tp_s uses its highest accumulated capacity of Tp_1 to schedule Ta_1 and shifts its priority, at lowest priority level Tp_s preempts Tp_1 of same priority

Tp_s is restarting while no aperiodic request is here, a priority exchange occurs with Tp_2

no periodic and aperiodic request arrives, the accumulated capacity of Tp_2 it is degraded to the priority level of background

Hybrid task set scheduling “Priority Exchange (PE)” (4)

e.g. Tp_s accumulates capacities from Tp_1, Tp_2:

- the both capacities of Tp_1, Tp_2 are used to process the first aperiodic release Ta_1.
- during the schedule of Ta_1, at the lowest priority level Tp_2, Tp_s is preempted by Tp_1.

	r_0	C	T	P
Tp_s	0	1	5	0
Tp_1	0	2	10	1
Tp_2	0	12	20	2
Ta_1	11	2	Na	Na
Ta_2	18	1	Na	Na

t		0-1	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9	9-10	10-11	11-12	12-13	13-14	14-15	15-16	16-17	17-18	18-19	19-20
Tp_s	**C0(t)**	1-0					1-0					1-0					1-0				
	C1(t)	0-1	1	1-0								0-1	1-0								
	C2(t)			0-1	1	1	1-2	2	2	2	2	2	2	2	2-1	1	1-2	2	2	2-1	1-0
	P(t)	1	0	2	0	0	2	0	0	0	0	1	1	2	2	0	2	0	0	2	0
Tp_1	**C(t)**	2-1	1-0									2-1	1	1-0							
	P(t)	0	1	1	1	1	1	1	1	1	1	0	1								
Tp_2	**C(t)**	12	12	12-11	11-10	10-9	9-8	8-7	7-6	6-5	5-4	4	4	4	4	4-3	3-2	2-1	1-0		
	P(t)	2	2	0	2	2	0	2	2	2	2	2	2	2	2	2	0	2	2		
Ta_1	**C(t)**												2-1	1	1-0						
Ta_2	**C(t)**																			1-0	

- (0-1) Tp_s accumulates a capacity from Tp_1
- (1-2) after C_s, Tp_s recovers its nominal priority
- (2-3) the accumulated capacity shifts from the Tp_1 to Tp_2 level
- (5-6) Tp_s accumulates one more time a capacity from Tp_2
- (10-11) Tp_s accumulates a capacity from Tp_1
- (11-12) Ta_1 is in the queue while the capacity of Tp_s is null, Tp_s uses its accumulated capacity of highest priority Tp_1 to schedule Ta_1 and preserves its priority at the Tp_1 level
- (12-13) the accumulated capacity Tp_1 of Tp_s is empty, Tp_s shifts its priority to Tp_2 but it is blocked while Tp_1 is here
- (13-14) Tp_1 resumes, Tp_s can continue at the priority level of Tp_2
- (15-16) the scheduling will go on …