Beágyazott információs rendszerek

2. Ütemezés (folyt.)

2020. szeptember 30.

Válaszidő számítás periodikus és sporadikus taszk-ok esetén:

Emlékeztető

Példa: Egy 4 taszkot és egy megszakítást (i_1) kiszolgáló rendszer adatai a következők:

Taszk	T	C	D
i_1	10	0.5	3
τ_1	3	0.5	3
τ_2	6	0.75	6
τ_3	14	1.25	14
τ_4	50	5	50

(az idők pl. ms-ban értendők)
Határozzuk meg a τ_4 taszk worst-case válaszidejét az iteratív eljárás segítségével!
Az iteratív eljárás táblázatos formában:

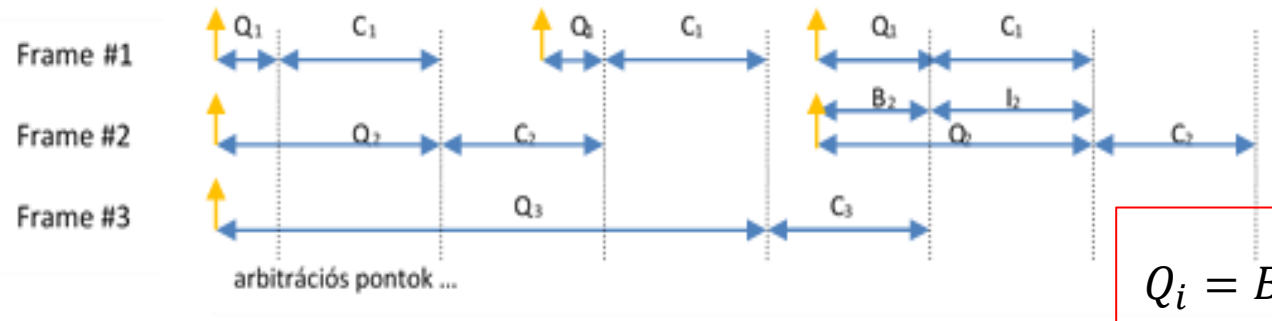
Lépés	R^n	I	R^{n+1}
1	0	0	5
2	5	0.5+1.0+0.75+1.25	8.5
3	8.5	0.5+1.5+1.50+1.25	9.75
4	9.75	0.5+2.0+1.50+1.25	10.25
5	10.25	1.0+2.0+1.50+1.25	10.75
6	10.75	1.0+2.0+1.50+1.25	10.75

10.75<50, tehát a határidő minden esetben teljesül!

Az ismertetett **DMA analízis** technikákat **autógyárak** intenzíven használják **worst-case válaszidő** analízis céljából, hogy a terméket **optimalizálják** a szükséges **órajel frekvenciák/sávszélességek** és az ehhez kapcsolódó **zavarérzékenységek** csökkentésével. (Volvo 1995-től, az S80-asnál.)

Példa: A DMA analízis egy módosított formája használható nem preemptív, azaz az éppen futó task-ot nem megszakító működés esetén is: A prioritásos CAN bus válaszidő analízise.

A **CAN** (Control Area Network, ISO 11898, Bosch) buszon történő kommunikáció jellegzetességei:



A válaszidő számítása:

$$R_i = C_i + Q_i$$

ahol

$$Q_i = B_i + \sum_{\forall k \in hp_i} \left\lceil \frac{Q_i}{T_k} \right\rceil C_k$$

Emlékeztető

Üzenet	T [ms]	C[ms]
1	3	1.35
2	6	1.35
3	10	1.35
4	30	1.35
5	40	1.35
6	40	1.35
7	100	1.35

Az üzenetek **periodikusak** és prioritásuk **felülről csökkenő.**
A küldésükre vonatkozó kérés érkezése **aszinkron**, tehát **tetszőleges kezdőfázissal** érkezhetnek.
A **7. üzenet** fékezéssel kapcsolatos információt hordoz, **100 ms** alatt a rendeltetési helyére kell kerüljön.

Az iteratív eljárás a **várakozási időre** vonatkozóan:

Lépés	Q^n	I							B	Q^{n+1}
		1	2	3	4	5	6	Összeg		
1	0	-	-	-	-	-	-	0	1.35	1.35
2	1.35	1	1	1	1	1	1	8.1	1.35	9.45
3	9.45	4	2	1	1	1	1	13.5	1.35	14.85
4	14.85	5	3	2	1	1	1	17.55	1.35	18.9
5	18.9	7	4	2	1	1	1	21.6	1.35	22.95
6	22.95	8	4	3	1	1	1	24.3	1.35	25.65
7	25.65	9	5	3	1	1	1	27	1.35	28.35
8	28.35	10	5	3	1	1	1	28.35	1.35	29.7
9	29.7	10	5	3	1	1	1	28.35	1.35	29.7

A worst-case várakozási idő $29.7\ ms$.

A worst-case válaszidő $29.7ms + 1.35ms = 31.05\ ms$.

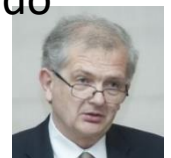
Ütemezési stratégiák:

Rate-monotonic (RM) (1973): Periodikus, független taszkok, $D_i = T_i$ és C_i ismert és konstans.
A legnagyobb prioritást a legkisebb periódusidejű taszk kapja. Az eljárás **preemptív.**
Feltételezzük, hogy a taszkok közötti átkapcsolás ideje elhanyagolható. **Jogos?**
Az RM algoritmusra elégséges teszt adható:

$$\mu = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n\left(2^{\frac{1}{n}} - 1\right) \xrightarrow[n\to\infty]{} ln\,2 \sim 0.7$$

n az ütemezendő taszkok száma.
Véletlenszerűen választott T_i és C_i esetén a szimulációk $\mu = 0.88$ értékig sikerrel jártak.

Ha a periódusidők egymással harmonikus viszonyban vannak, azaz mindegyik periódusidő a rövidebb idejű egészszámú többszöröse, akkor bizonyítható, hogy $\mu = 1$ elérhető!

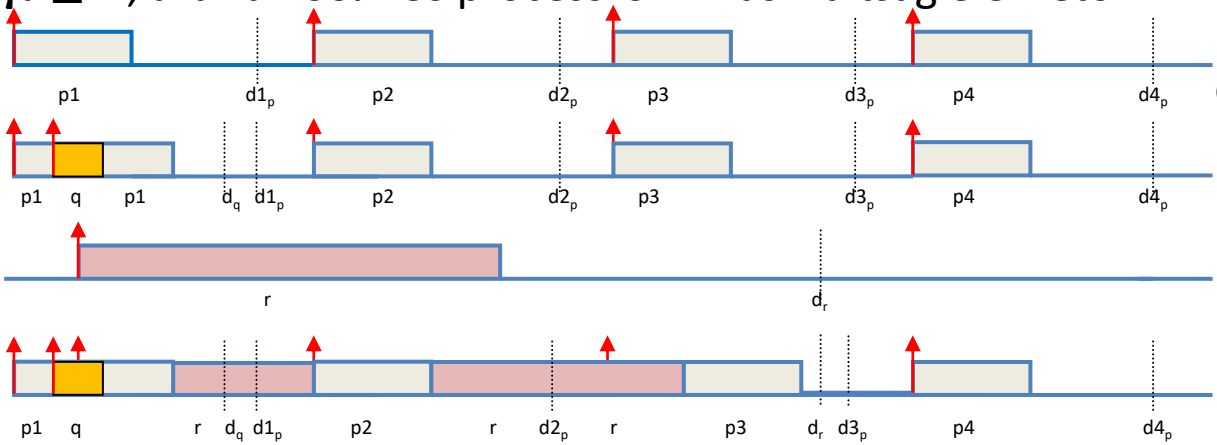


Emlékeztető

Earliest Deadline First (EDF) stratégia:

Periodikus, egymástól független taszkok ,$D_i \leq T_i$ és C_i ismert, továbbá konstans.
Futás közben a processzort (a legnagyobb prioritást) az a taszk kapja, amelyiknek **legközelebbi a határideje**. Az eljárás **preemptív**. A taszkok közötti átkapcsolás idejét elhanyagoljuk. **Jogos?**

Elégséges teszt adható: A megadott feltételeknek eleget tevő taszk-együttes ütemezhető, ha $\boldsymbol{\mu \leq 1}$, azaz a **100%-os** processzor-kihasználtság elérhető.



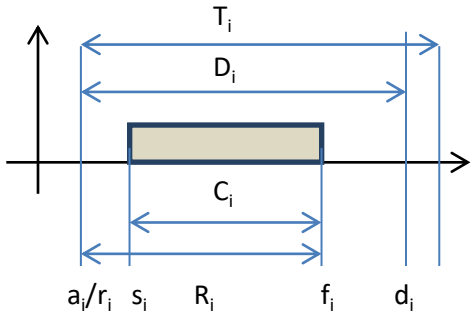
Első sor: a **p** taszk kérései, futásai (p…) és a kapcsolódó határidők (d…$_p$).

Második sor: q taszk kérése a p1 futás alatt.

Harmadik sor: r taszk kérése és határideje.

Negyedik sor: összegzi a három taszk futását.

Least Laxity First (LLF) stratégia: Az EDF-hez hasonló. A processzort (a legnagyobb prioritást) a legkisebb „lazasággal” (laxity-vel) rendelkező task kapja meg. Ez a vizsgálati időpontban a határidő és a még hátralévő számítási idő különbsége.



A megadott feltételeknek eleget tevő taszk-együttes ütemezhető, ha $\boldsymbol{\mu \leq 1}$, azaz a **100%-os** processzor-kihasználtság elérhető.

Megjegyzés: Az EDF és az LLF stratégia aperiodikus task-ok esetén is alkalmazható, de az elégséges feltétel nem!

Ugyanis a processzor-kihasználtsági tényező aperiodikus taszkok esetében csak eltérő módon értelmezhető.

Emlékeztető

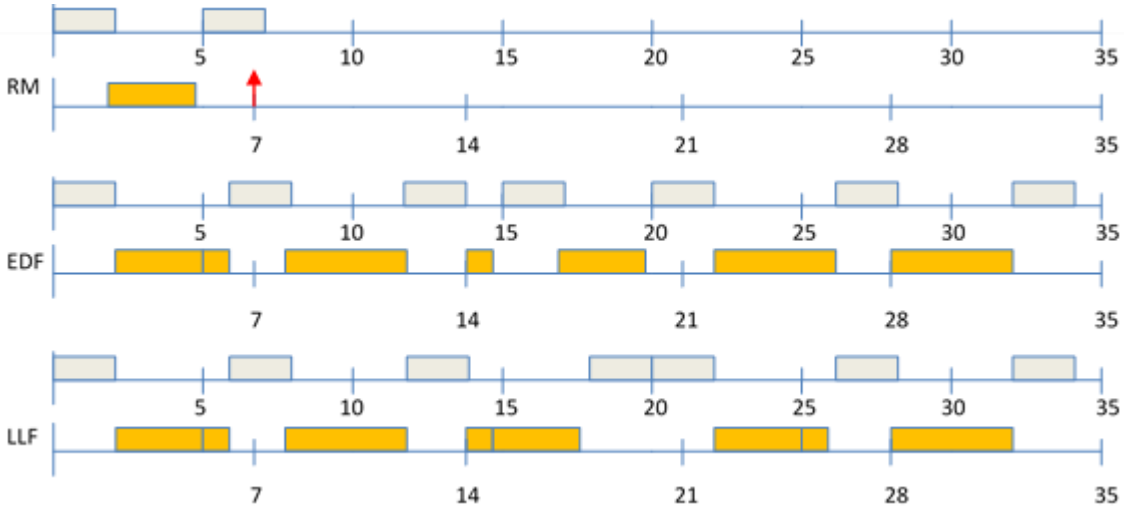
Példa: Az RM és az EDF algoritmusok összehasonlítása. Két taszkunk van.
A periódus idejük és a határidejük megegyezik. $T_1 = 5$ ms, $C_1 = 2$ ms, $T_2 = 7$ ms, $C_2 = 4$ ms.

A processzor-kihasználtsági tényező:

$$\mu = \frac{2}{5} + \frac{4}{7} = 0.4 + 0.57 = 0.97$$

A szükséges feltétel az ütemezhetőséghez teljesül, de az elégségesség csak az EDF esetén.

Induláskor egyidejű kérést feltételezve a RM eljárás, az EDF eljárás és a LLF eljárás:



A RM eljárás esetében a második task 7 ms-nál lekési a határidőt, az EDF és az LLF eljárással pedig ütemezhetőek lesznek a taszkok. Mind az EDF, mind az LLF eljárásnál természetesen adódó szabály, hogy azonos határidő, ill. laxity esetén a kevesebb taszk-váltást eredményező választással élünk.

A taszk váltások processzor-időt igényelnek, mert az éppen futó taszk futtatási környezetét (regiszter-tartalmak) menteni kell a taszkhoz rendelt, és a memóriában található Task Control Bock-ba (TCB), míg váltás keretében a futtatandó taszk futási környezetét pedig a memóriából a processzor regisztereibe kell tölteni.

A regiszterek feltöltésére, ill. tartalmuk kimásolására a processzorok általában rendelkeznek gyors mechanizmusokkal, de értelemszerűen ezeknek is van időigényük.

Az EDF ütemezhetőség bizonyítása:

Emlékeztető

A bizonyítást periodikus taszkok és $D_i = T_i$ esetére mutatjuk be. Az állítás a következő:
Egy periodikus taszk-készlet EDF-fel akkor és csak akkor ütemezhető, ha

$$\mu = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1.$$

A bizonyítás: **1. *csak akkor*** rész: Azt mutatjuk meg, hogy $\mu > 1$ esetében a taszk-készlet nem ütemezhető. Ehhez definiáljuk a $T = T_1 T_2 \dots T_n$ időtartamot, azaz a periódusidők közös többszörösét.

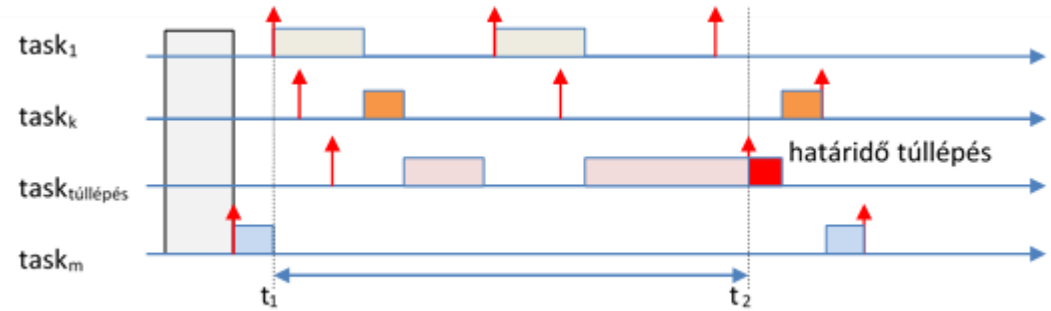
Ez alatt az idő alatt a taszkok által igényelt processzor idő a következőképpen számítható:

$$\sum_{i=1}^{n} \frac{T}{T_i} C_i = \mu T.$$

Ha $\mu > 1$, akkor az igényelt processzoridő meghaladja a hozzáférhető processzor-időt, tehát a taszk-készlethez nem létezik ütemezés.

2. *ha* rész: Az elégségességet ellentmondással bizonyítjuk.

Tegyük fel, hogy $\mu < 1$, de a task-készlet mégsem ütemezhető. A bizonyítás gondolatmenetének megértését az alábbi ábra segíti.



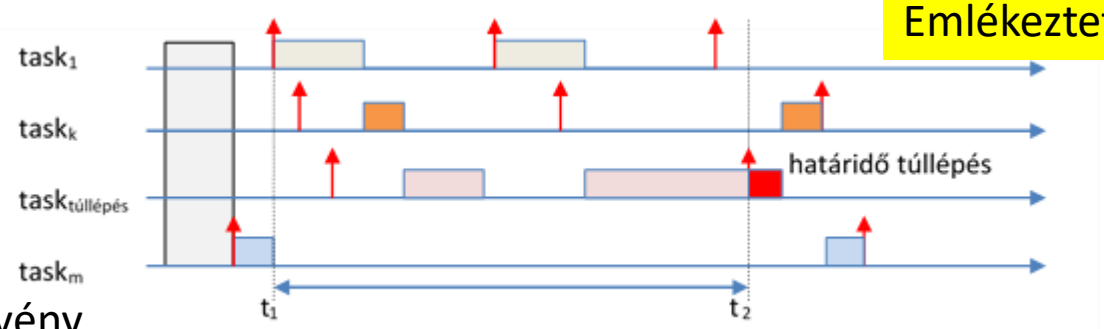
Az ábrán periodikus taszkok ütemezését látjuk EDF stratégia szerint.

Ha feltételezésünk szerint a taszk-készlet nem ütemezhető, akkor kell legyen olyan taszk, amelyik lekési a határidőt.

Legyen t_2 az az időpont, amikor a határidő túllépés bekövetkezik, és $[t_1, t_2]$ pedig a leghosszabb folyamatos processzor-használat a határidő-túllépés előtt úgy, hogy a $[t_1, t_2]$-ben csak t_2 előtti vagy azzal egyező határidejű kérések végrehajtására került sor. t_1 valamelyik periodikus kéréssel egybeeső időpont.

Legyen $C_P(t_1, t_2)$ a periodikus task-ok által a $[t_1, t_2]$-ben kért teljes számítási idő, ami a következő módon számítható:

$$C_P(t_1,t_2) = \sum_{r_k \geq t_1, d_k \leq t_2} C_k = \sum_{i=1}^{n} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor C_i$$



ahol $\lfloor \ldots \rfloor$ az alsó-egészt kijelölő függvény.

(Vegyük észre, hogy a legfelső sorban a harmadik kérés teljesítésére az algoritmus szabályai szerint nem kerül sor, ezért helytálló az alsó-egész hozzárendelés.)

Ha ezt majoráljuk az alábbiak szerint:

$$C_P(t_1,t_2) = \sum_{r_k \geq t_1, d_k \leq t_2} C_k = \sum_{i=1}^{n} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor C_i \leq \sum_{i=1}^{n} \frac{t_2 - t_1}{T_i} C_i = (t_2 - t_1)\mu$$

mivel t_2-ben túlléptük a határidőt, a $C_P(t_1,t_2)$ időnek nagyobbnak kell lennie, mint a rendelkezésre álló processzor-idő, azaz $(t_2 - t_1)$. Ezzel $(t_2 - t_1) < C_P(t_1,t_2) \leq (t_2 - t_1)\mu$

amiből $\mu > 1$ következik, ami pedig ellentmondás, vagyis a megfogalmazott állítás **hamis!**

Periodikus és aperiodikus taszkok együttes kezelése:

Elsősorban HRT(kemény határidejű) rendszerek esetén, de a SRT (puha határidejű) rendszerek ütemezését is kezeljük.

Kemény és puha határidejű taszkok együttes kezelésénél két szabály alkalmazására kerül sor:

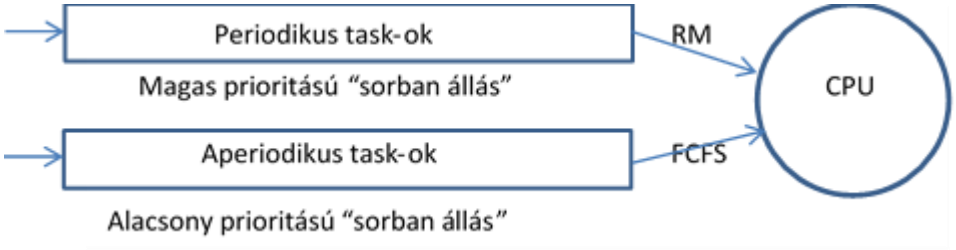
1. szabály: ∀ taszk ütemezhető kell legyen **átlagos** végrehajtási és érkezési idő feltételezésével.

2. szabály: ∀kemény határidejű (HRT) taszk ütemezhető kell legyen valamennyi task legkedvezőtlenebb végrehajtási (**worst-case execution**) és érkezési (**worst-case arrival**) idejének feltételezése mellett.

Az alábbiakban ismertetett módszerek esetében a következő előzetes feltételezésekkel élünk:

1. A periodikus taszkok ütemezése RM algoritmus szerint történik.
2. A periodikus taszkok egyidejűleg (nulla kezdőfázissal) indulnak és $D_i = T_i$.
3. Az aperiodikus kérések érkezési ideje ismeretlen.
4. Sporadikus taszkok esetén $D_i = T_i$.

A háttérbeni ütemezés (Background Scheduling) módszere:



A módszer előnye egyszerűsége, hátránya pedig az, hogy az aperiodikus taszkok válaszideje nagyon nagy lehet. (FCFS=First-Come-First-Served.)

Ha az aperiodikus taszkok esetén a válaszidő kritikus, akkor az ún. **szerver-módszerek** alkalmazása jobb eredményt adhat.
A **szerver-módszer** az aperiodikus taszkok végrehajtásához **szeparáltan** biztosít processzor időt. Ennek eszköze a **szerver-taszk**, amelyet a periodikus taszkokkal együtt ütemezünk.

1. Polling Server (PS): Az aperiodikus kérések teljesítése külön ún. **szerver taszk** (S) segítségével, a szerver kapacitás (T_S, C_S) terhére, **független** ütemezési stratégiával történik. **Példa:**
Ha nincsen aperiodikus kérés, amikor a szerver futására sor kerülhetne, akkor a PS **felfüggeszti** magát, kapacitása **nem örződik** meg!

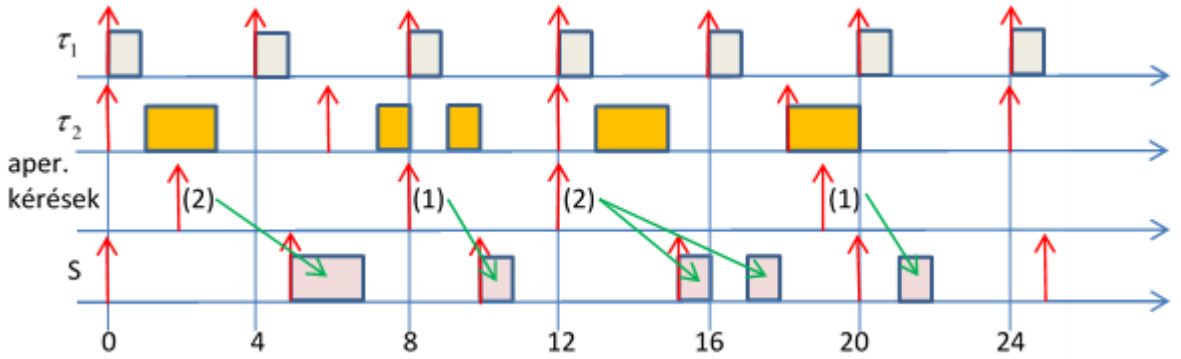
Legyen T_S=5, C_S=2.

	C	T
τ_1	1	4
τ_2	2	6

A szerver taszk (RM szerint) a középső prioritásra kerül.
A taszkok egyidejű indítását, azaz azonos kezdőfázist feltételezve az ütemezés a következőképpen alakul:

Legyen T_S=5, C_S=2.

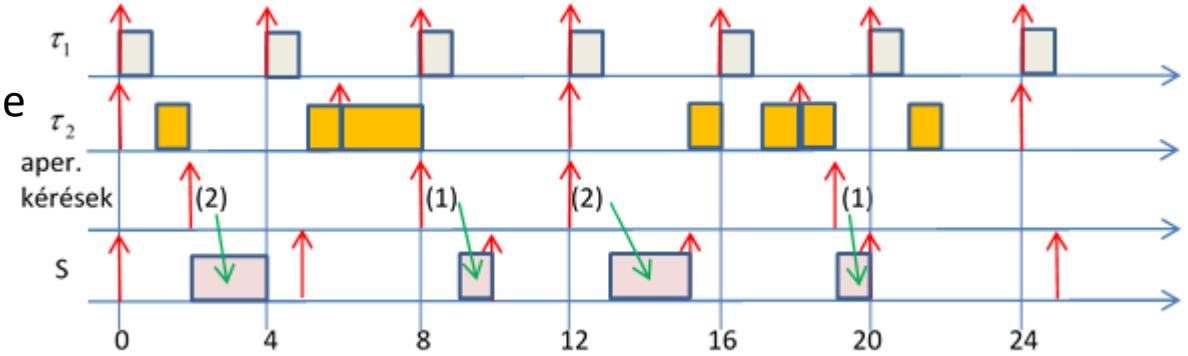
	C	T
τ_1	1	4
τ_2	2	6



Látható, hogy a legkedvezőtlenebb esetben az aperiodikus kérések teljesítésére – a magasabb prioritású taszkok által okozott interferenciát nem számítva – csak egy teljes szerver taszk periódus elteltével kerül sor.

2. Deferrable Server (DS):

Az aperiodikus kérések teljesítése külön ún. **szerver taszk** (S) segítségével, a szerver kapacitás (T_S,C_S) terhére, független ütemezési stratégiával történik.



Ha nincsen aperiodikus kérés, amikor a szerver futására sor kerülhetne, akkor a **DS** taszk futása halasztódik, kapacitását a periódus végéig megőrzi. **Példa:** Az előző példa adataival…

Ezzel a módszerrel az aperiodikus taszkokra **sokkal jobb válaszidők** érhetők el.
(A szerver taszk ütemezése az előzővel azonos módon, RM stratégiával történt.)

3. Priority Exchange Server (PE): Olyan, mint a DS, magas prioritáson futó szervert használ, de másképpen őrzi a kapacitást: alacsonyabb prioritású periodikus taszk kapacitásával cseréli ki.

Példa: Legyen T_S=5, C_S=1. Az ezen kívül ütemezendő task-ok adatai:

	C	T
τ_1	4	10
τ_2	8	20

T_S=5,
C_S=1.

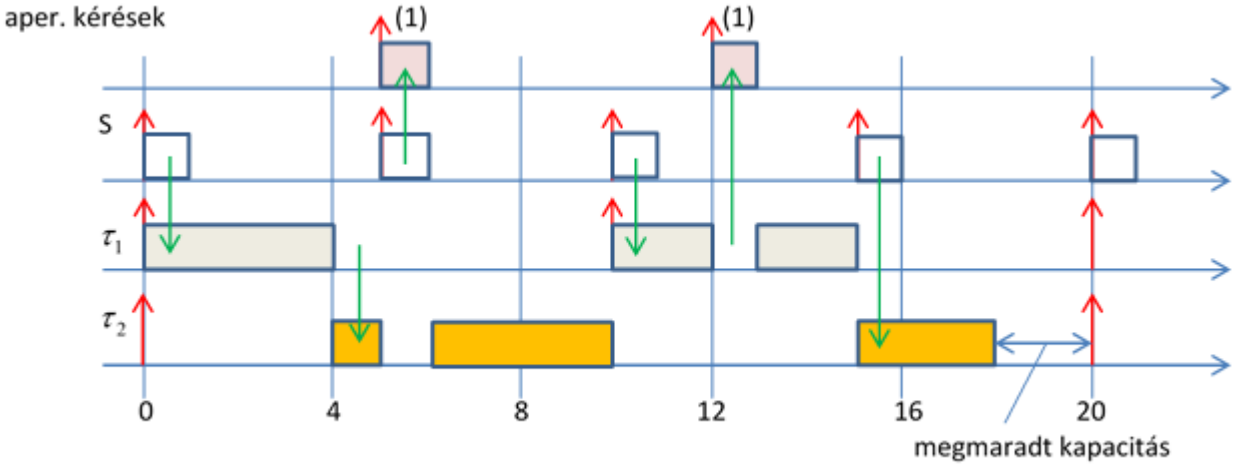
	C	T
τ_1	4	10
τ_2	8	20

A szerver task (RM szerint) a legmagasabb prioritásra kerül.

a processzor kihasználtsági tényező:

$$\mu = \frac{1}{5} + \frac{4}{10} + \frac{8}{20} = 1$$

A taszkok egyszerre indulnak.



Mivel nincsen előzetesen aperiodikus kérés, **az első ütemben** megjelenő szerver kapacitást felhasználja a τ_1 taszk.

Ennek következtében τ_2 taszk korábban indulhat, vagyis a szerver kapacitás ide kerül.

A második ütemben megjelenő szerver kapacitást közvetlenül felhasználjuk.

A **harmadik ütemben** megjelenő szerver kapacitást a τ_1 taszk használja fel, amit a második aperiodikus kérés kiszolgálására visszacserél.

A **negyedik ütemben** érkező szerver kapacitást a τ_2 taszk hasznosítja.

Ezzel együtt kétperiódusnyi szerver kapacitás “halmozódik fel” a végrehajtásánál, ami mozgósítható lenne, ha lenne további **aperiodikus** kérés.

Példa: Legyen T_S=5, C_S=1. A szerver taszk (RM szerint) a legmagasabb prioritásra kerül.

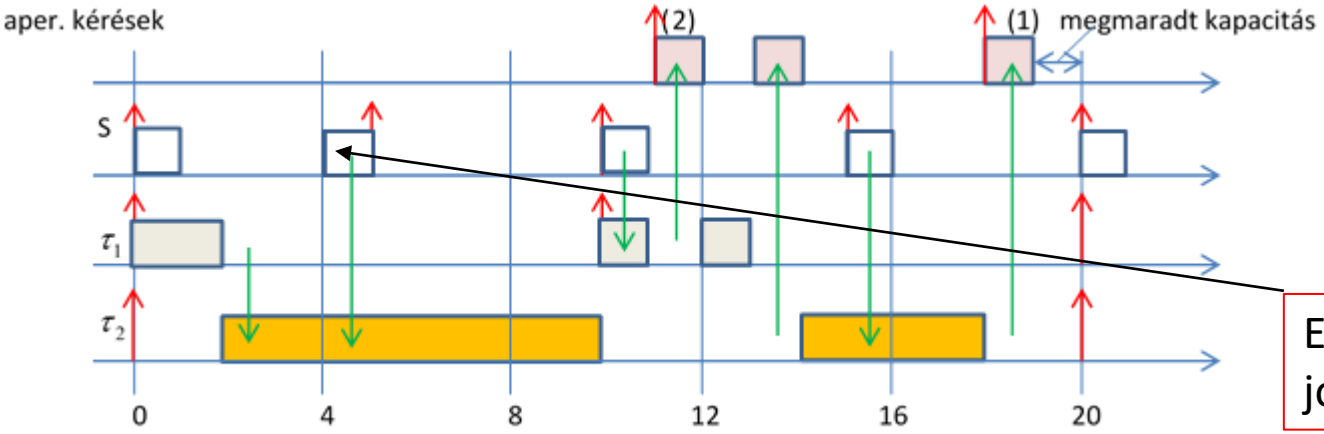
Az ütemezendő taszkok:

A task-ok egyszerre indulnak:

	C	T
τ_1	2	10
τ_2	12	20

A processzor kihasználtsági tényező:

$$\mu = \frac{1}{5} + \frac{2}{10} + \frac{12}{20} = 1.$$



A kapacitás másik taszkhoz történő áthelyezése azzal is jár, hogy az áthelyezett kapacitás a befogadó taszk prioritásán használható fel.

Ez a blokk helyesen eggyel jobbra helyezendő!

Lásd: a **11.** időpillanatban kért **2** időegységnyi kapacitás első fele a τ_1 taszknál lelhető fel, a második fele pedig a τ_2 taszknál. Az első fél futását követően a τ_1 taszk fut tovább, majd csak annak lefutása után áll rendelkezésre a τ_2 taszk prioritásán elérhető második fél. Itt egy periódusnyi szerver kapacitás "halmozódik fel" a τ_2 végrehajtásánál, ami mozgósítható lenne, ha lenne további aperiodikus kérés.

Sporadic Server (SS): Olyan, mint a **DS**, megőrzi kapacitását, de másképpen tölti vissza: Nem a periódus elején, hanem **a felhasználást követően**.

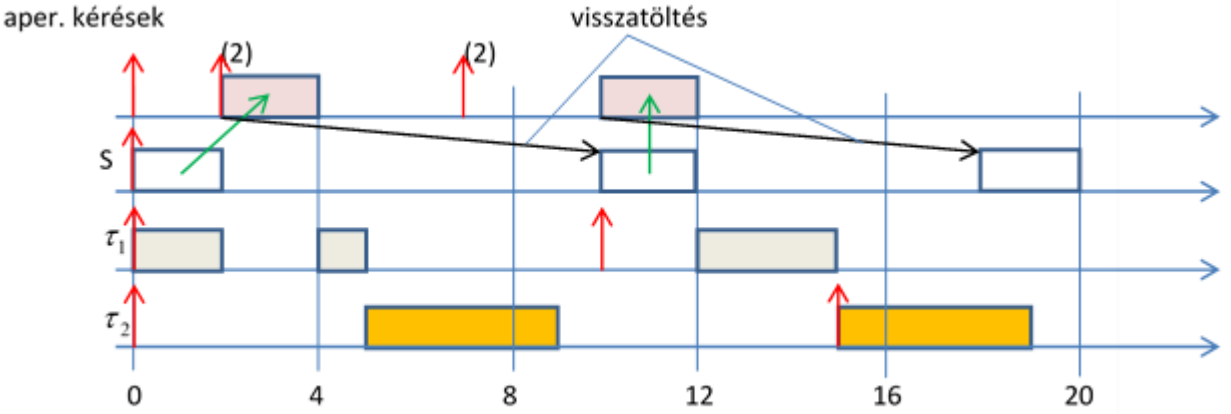
A **felhasználás kezdetétől** egy szerver taszk **periódusnyira** jelenik meg a szerver kapacitás.

Példa: T_S=8, C_S=2. Az ezen kívül ütemezendő task-ok adatait az alábbi táblázat tartalmazza:

	C	T
τ_1	3	10
τ_2	4	15

T_S=8, C_S=2. A szerver taszknak a legmagasabb prioritása. A taszkok egyidejűleg indulnak.

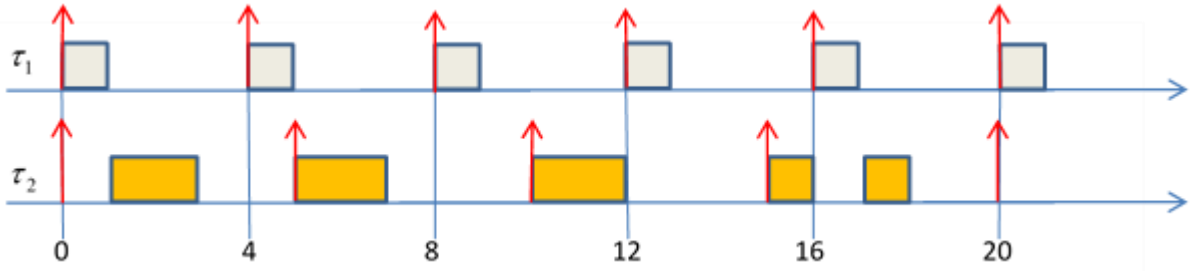
	C	T
τ_1	3	10
τ_2	4	15



Slack stealing: Az egyes task-ok végrehajtása között fellelhető szabadidőt, "lazaságot" használjuk fel. Sokkal jobb válaszidőt ad, mint a **DS**, a **PE** vagy a **SS** eljárás.

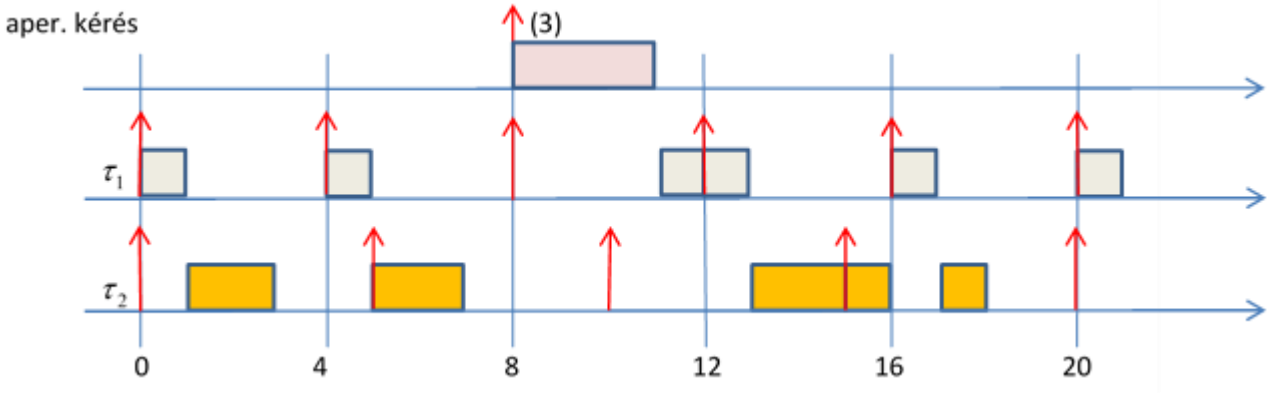
Példa: A normál ütemezés **RM** szerint:

	C	T
τ_1	1	4
τ_2	2	5



Aperiodikus kérés érkezését követően kiszámításra kerül, hogy mennyi tartalék "lazaság" van a rendszerben,

és azt megkapja az aperiodikus task a legnagyobb prioritással az alábbiak szerint:



Dual Priority Scheduling: Három prioritási szint van: **alacsony, közepes** és **magas**.

Kezdetben **a kemény valós idejű** taszkok az **alacsony** prioritáson futnak!

A **puha valós idejű** taszkok és az aperiodikus taszkok a **közepes** prioritási szintre kerülnek.

A **kemény valós idejű** taszkok a határidő előtt $X_i = D_i - R_i$ ún. **promóciós idővel** átkerülnek a magas prioritásra, hogy a határidőt be tudják tartani. ($R_i = B_i + C_i + I_i$)

Az **alacsony, közepes** és **magas** szintek értelemszerűen önmagukon belül további prioritási szintekre bonthatók.

Megjegyzés: A fentiekben bemutatott szerver megoldások rendre a **RM** ütemezési stratégiát követve működnek. Ezek **fix prioritású** szerverek. Az **EDF** ütemezési stratégiára is alapozhatók szerverek. Ezek **dinamikus prioritású** szerverek.