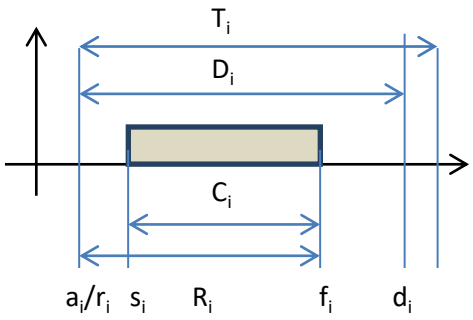
Beágyazott információs rendszerek

2. Ütemezés (folyt.)

2020. szeptember 24.

2. Ütemezés



Probléma: a processzor(ok)nak többféle időzítés mellett többféle feladatot (taszk) kell ellátniuk. Egy *i*-edik feladathoz (taszk-hoz) köthető időviszonyok az alábbiak szerint értelmezhetőek:

$\mathbf{a_i}$ vagy $\mathbf{r_i}$ az érkezési idő (arrival/release/request time),
$\mathbf{s_i}$ a végrehajtás kezdésének ideje (start time),
$\mathbf{f_i}$ a végrehajtás befejezésének ideje (finishing time),
$\mathbf{d_i}$ a végrehajtás határideje (deadline),

$\boldsymbol{T_i}$ a periódusidő (period time), $\boldsymbol{D_i}=\mathbf{d_i}-\mathbf{a_i}$ a kérés időpontjához képesti határidő (deadline),
$\boldsymbol{C_i}$ a számítási idő (computation time), $\boldsymbol{R_i}=f_i-a_i$ a válaszidő (response time).

1. **Ciklikus ütemezés**: A legegyszerűbb, tervezési időben fix időszeletek osztunk ki periodikus kérések kiszolgálására, és ezt ciklikusan ismételjük.

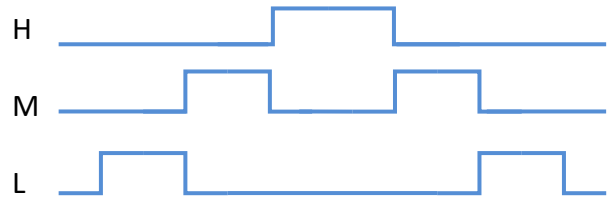
A kiosztást tipikusan óra-vezérelt módon oldjuk meg, ezért **óra vezérelt** vagy **idő-vezérelt** ütemezésnek is nevezzük. Az ütemezéssel kapcsolatos döntések **tervezési időben** történnek.

Emlékeztető

2. **Időosztásos (time-shared)/körforgó (round-robin) ütemezés:**
A futtatható task-ok egy FIFO-ba kerülnek, és a legelöl álló task fog futni **maximum egy időszelet ideig.** Az időszelet általában néhányszor **10 ms**, ami a taszk-októl független paraméter.
Ha az adott taszk nem fut le az időszelet alatt, akkor futása megszakad, és a FIFO végére kerül.

3. **Prioritásos ütemezés:** A futtatható taszk-ok közül az fut, amelyiknek legnagyobb a prioritása.
A prioritás hozzárendelés történhet tervezési és futási időben egyaránt.

Illusztráció:



A három taszk rendre alacsony (**L**=low), közepes (**M**=medium) és magas (**H**=high) prioritású.
Ezeket a prioritásokat tervezési időben osztottuk ki.
Az ábrán mindhárom taszk azonnal futni kezd, amint futtathatóvá válik.

Az ábrán látható esetben a legalacsonyabb prioritású taszk válaszideje $R_L=C_L+C_M+C_H$.

Ha a középső és/vagy a magas prioritású taszk periodikusan kér, akkor az időviszonyok függvényében elképzelhető, hogy az R_L idő alatt többször is lefut.

A válaszidő számítást a legkedvezőtlenebb esetre, az i-edik taszk-ra vonatkoztatva, a következő képlettel tudjuk elvégezni:

$$R_i = C_i + I_i = C_i + \sum_{\forall k \in hp_i} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

ahol I_i az ún. interferencia idő, azaz az az időtartam, amíg a magasabb prioritású taszk-ok futása akadályozza az alacsonyabb prioritású taszk-ok végrehajtását.

A $\forall k \in hp_i$ azokat a taszk-okat jelöli ki, amelyek prioritása nagyobb, mint i (*hp*=higher priority).

A $\lceil \quad \rceil$ zárójel a felső-egész képzés operátora. $\lceil 1.02 \rceil = 2, \lceil 2.0 \rceil = 2.$

A módszer a válaszidő lehető legkedvezőtlenebb értékét adja meg.

(Worst-case response time.)

Emlékeztető

Mivel a képletben a baloldalon szereplő R_i a jobboldalon is szerepel egy erősen nemlineáris függvény argumentumában, ezért iteratív eljárás alkalmazására kényszerülünk:

$$R_i^{n+1} = C_i + I_i = C_i + \sum_{\forall k \in hp_i} \left\lceil \frac{R_i^n}{T_k} \right\rceil C_k$$

Az iterációt addig folytatjuk, amíg: egy n_0 érték mellett $R_i^{n_0+1} = R_i^{n_0}$.

A módszer neve: Deadline Monotonic Analysis (**DMA**).

Feltételezi, hogy a task-okhoz aszerint rendelünk prioritást, hogy mekkora a D_i határidejük.

A módszer alkalmazásánál feltételezzük, hogy $\boldsymbol{D_i \leq T_i}$.

A módszer periodikus task-ok mellett ún. sporadikus task-okra is alkalmazható.

Periodikus taszk: ismert és fix $\boldsymbol{T_i}$ periodusidővel jellemezhető.

Sporadikus taszk: a kérések nem periodikusak, de **ismert** és fix egy olyan $\boldsymbol{T_i}$ időérték, ami minimálisan eltelik két kérés között.

Aperiodikus taszk: a kérések nem periodikusak, és **nincs** egy olyan ismert és fix $\boldsymbol{T_i}$ időérték, ami minimálisan eltelik két kérés között. Egy kérést követően azonnal jöhet egy következő kérés.

Ebben az esetben a **DMA** módszer nem alkalmazható.

Példa: Egy 4 taszkot kiszolgáló rendszer adatai a következők (az idők pl. ms-ban értendők):

Task	T	C	D
1	250	5	10
2	10	2	10
3	330	25	50
4	1000	29	1000

A taszkok sorrendje a prioritási sorrend. Ha a határidők megegyeznek, akkor másodlagos szempontok alapján döntünk a prioritásról.

Határozzuk meg a 3-as taszk worst-case válaszidejét az iteratív eljárás segítségével!

Lépés	R^n	I	R^{n+1}
1	0	0	25
2	25	5+3*2	36
3	36	5+4*2	38
4	38	5+4*2	38

Emlékeztető

Megjegyzések:

38<50, tehát a 3-as task legkedvezőtlenebb esetben is teljesíti az előírt határidőt.

Vegyük észre, hogy a 4-es taszk adatait az eljárás során nem használtuk fel, a számításhoz felesleges volt megadnunk.
Vegyük azt is észre, hogy taszkokat egymástól függetleneknek képzeltük el.

Egymástól nem független, azaz például egymással kommunikáló, egymásnak adatot továbbító taszkok esetében előfordul(hat), hogy magasabb prioritású taszk alacsonyabb által szolgáltatott adatra várni kénytelen.

Ez a várakozási idő értelemszerűen a mindenkori és a worst-case válaszidejét egyaránt módosítani fogja.

Ütemezhetőség, ütemezhetőségi tesztek:

Szükséges teszt: nem ütemezhető, ha a szükséges feltétel nem teljesül.
Elégséges teszt: biztosan ütemezhető, ha az elégséges feltétel teljesül.
Egzakt teszt: szükséges és elégséges, és a teszt az ütemezés létezését is megmutatja.

Az egzakt ütemezhetőségi tesztek komplexitásuk alapján az **NP-teljes** problémák osztályába tartoznak, ezért számítástechnikailag kezelhetetlenek, ezekkel a továbbiakban nem foglalkozunk.

Periodikus taszkok esetén a **szükséges feltételek** között elsőként az ún. **processzor-kihasználtsági tényező** említhető, ami az időegységre vetített processzor-idő igények összege:

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

Egyprocesszoros rendszerben, ha $\boldsymbol{\mu \leq 1}$ nem teljesül, akkor a task-ok nem ütemezhetőek. A $\boldsymbol{\mu \leq 1}$ tehát szükséges feltétel. n a taszkok száma.

Ha $\boldsymbol{N}$ processzorunk van, akkor $\boldsymbol{\mu \leq N}$ a feltétel.

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:

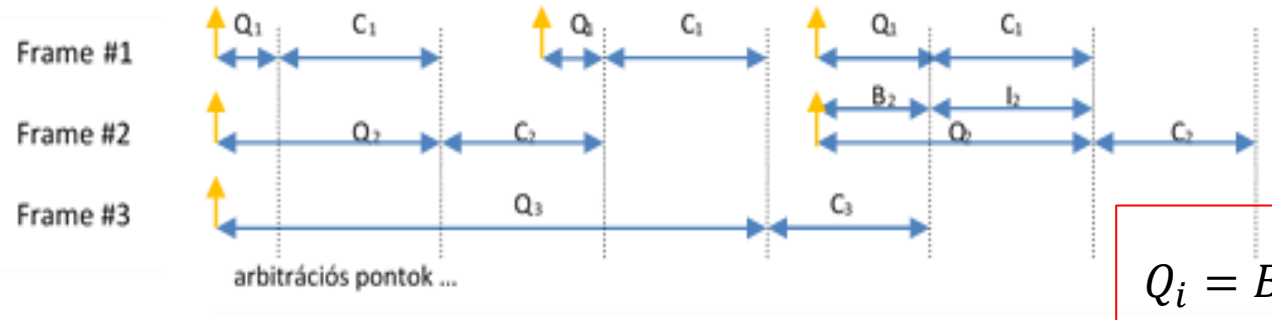
Lépés	R^n	I	R^{n+1}
1	0	0	5
2	5	0.5+1.0+0.75+1.25	8.5
3	8.5	0.5+1.5+1.50+1.25	9.75
4	9.75	0.5+2.0+1.50+1.25	10.25
5	10.25	1.0+2.0+1.50+1.25	10.75
6	10.75	1.0+2.0+1.50+1.25	10.75

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

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

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 $29.7ms + 1.35ms = 31.05\ ms$.

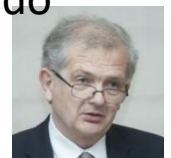
Ütemezési stratégiák:

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!



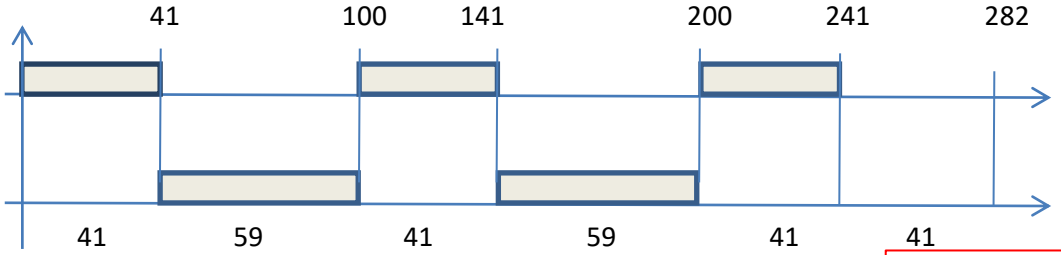
Példa: Milyen periódusidő és számítási idő viszonyok esetén jutunk el az ütemezhetőség határára $n = 2$ esetén? $\frac{T_2}{T_1} = \sqrt{2}$, $C_1 = T_2 - T_1$, $\frac{C_1}{T_1} = \frac{T_2 - T_1}{T_1} = \frac{C_2}{T_2}$ választással:

$\mu = 2(\sqrt{2} - 1)$, illetve tetszőleges i esetén:

Ha $\frac{T_{i+1}}{T_i} = 2^{\frac{1}{n}}$, $C_i = T_{i+1} - T_i$, $\frac{C_i}{T_i} = \frac{T_{i+1} - T_i}{T_i} = \frac{C_{i+1}}{T_{i+1}}$ akkor $\mu = n\left(\frac{T_{i+1}}{T_i} - 1\right) = n\left(2^{\frac{1}{n}} - 1\right)$

Példa: Két taszk esetén $T_1 = 100$, $C_1 = 41$, $T_2 = 141$, $C_2 = 59$, mind *ms* dimenziójú.

$\mu = \frac{41}{100} + \frac{59}{141} = 0.41 + 0.4184 = 0.8284$ $\sim 2(\sqrt{2} - 1)$, egyidejű kezdést feltételezve:



a számítási idők minimális növelése esetén az ütemezés az RM stratégia esetén ellehetetlenül!

241 és 282 között nincsen ütemezhető feladat!

Megjegyzések:

1. Az RM eljárás alkalmazása esetén a legkedvezőtlenebb esetet a taszkok induláskor **egyidejű kezdése** jelenti.

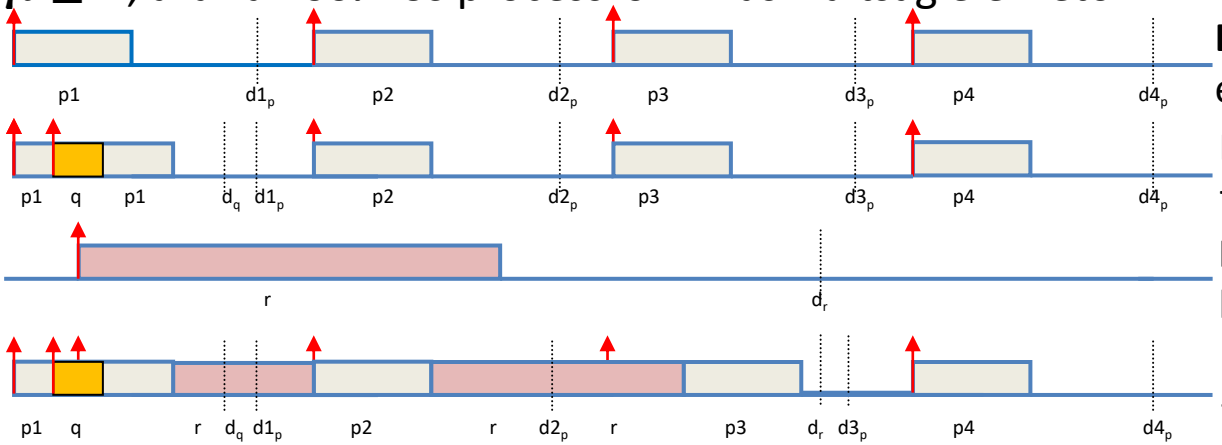
Nullától különböző **kezdőfázis** ütemezhetőségi szempontból kedvező!

2. Az RM eljárás alkalmazása esetén, ha csak a szükséges feltétel teljesül, az elégséges nem, akkor az **ütemezhetőségi vizsgálatot** a periódusidők **legkisebb közös többszörösére** kell elvégezni, ami igen nagy szám lehet!

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.



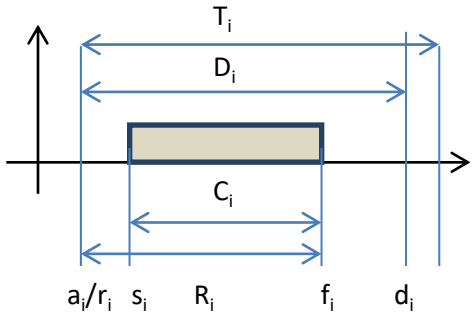
Első sor: a **p** taszk kérései, futásai (p…) és a kapcsolódó határidők (d…$_p$).

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.

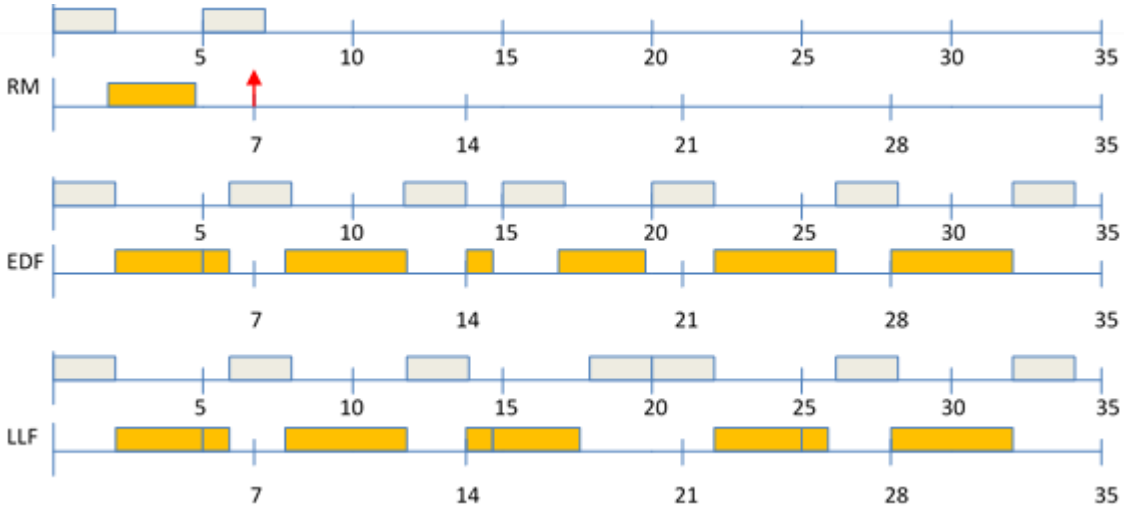
Példa: Az RM és az EDF algoritmusok összehasonlítása. Két taszkunk van.
A periódus idejük és a határidejük megegyezik. $T_1 = 5$ ms, $C_1 = 2$ ms, $T_2 = 7$ ms, $C_2 = 4$ ms.

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:

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.

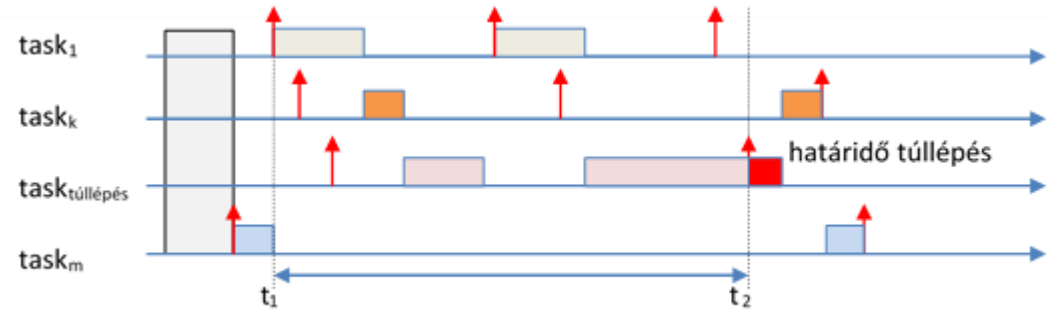
Ez alatt az idő alatt a taszkok által igényelt processzor idő a következőképpen számítható:

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

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.

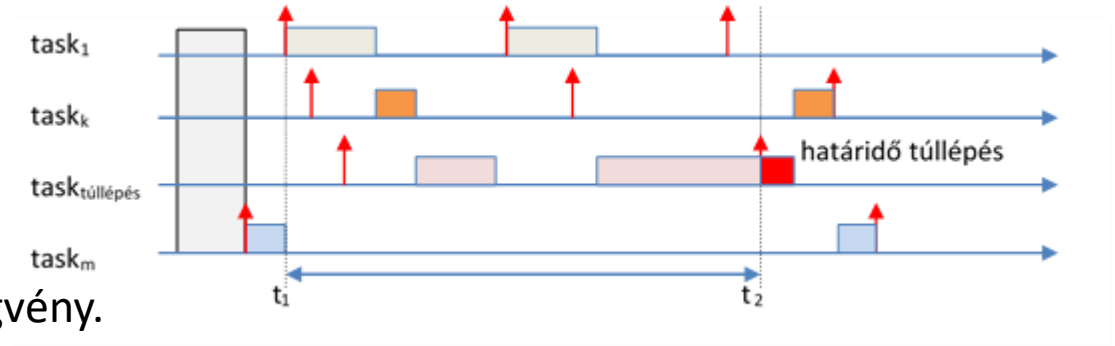


Az ábrán periodikus taszkok ütemezését látjuk EDF stratégia szerint. Ha feltételezésünk szerint a taszk-készlet nem ütemezhető, akkor kell legyen olyan taszk, amelyik lekési a határidőt.

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:

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



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:

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

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 $(t_2 - t_1) < C_P(t_1, t_2) \le (t_2 - t_1)\mu$

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