Progresivní technologie v informatice II

Systémy reálného času
princip a využívání v praxis

prof. Ing. Hana Kubátová, CSc.

Katedra číslicového návrhu
Fakulta informačních technologií
České vysoké učení technické v Praze

Obsah

- Terminologie, klasifikace RT systémů
- Hard x Soft RT systémy
- Systémy bezpečné při poruše x funkční při poruše

  (fail-safe, fail-operational)
- Systémy s garantovanou odezvou x best-effort
- Systémy s dostatečnými x nedostatečnými zdroji
- Systémy řízené událostmi x časem
- Plánování
- Operační systémy reálného času (RTOS)
- Programování pro reálný čas

Zdroje

- **TU Wienna, Institut für Technische Informatik, Real-Time Systems Group,**
- **Přednášky ČVUT-FEL M. Sojky** http://rtime.felk.cvut.cz/psr/
- **Článek** „Programovací jazyky s podporou Real-time“, Ing. Dalibor Zacios, 2003 (viz materiály a http://www.fit.vutbr.cz/study/courses/TJD/public/0203TJD-Zacios.pdf

Knihy:

- **Kopetz, H.: Real-Time Systems.** Springer,
- **Jane W.S. Liu: Real-Time Systems** Prentice Hall, 2000
- **další doporučené studijní materiály:** https://courses.fit.cvut.cz/BI-SRC/

Návaznosti

- **bakalářské předměty BI-OSY a BI-VES**

Úvod do Systémů Reálného Času

Systém reálného času (SRČ) je systém, jehož specifikace zahrnuje zároveň

- **logické i časové požadavky na správnost**.

- Logická správnost: Dostáváme *správné výsledky.*

  Např. 1 + 1 = 2

- Časová správnost: Výsledky dostáváme *ve správném čase.*

Základní koncepty R-T systémů

- Fyzický čas ... celosvětový, dosažitelný kdekoli v R-T systému
- Deadline ... R-T úloha musí produkovat výsledky v daných časových limitech
- Časově omezená platnost R-T dat ... zneplatnění „správných“ dat postupem času
- Distribuovanost a komunikace ... chytré senzory a další uzly R-T systému spolu musí komunikovat v reálném čase

Charakteristiky a vlastnosti SRC

- Jsou řízené událostmi, reaktivní
- Jejich selhání bývá drahé a nebezpečné
- Paralelní/vícevláknové programování
- Nepřetržitý provoz bez zásahu operátora
- Požadavky na spolehlivost a odolnost vůči poruchám (i chybám) ... fault-tolerance
- Predikovatelné (tzn. deterministické) chování)

Poznámka: SRC, SRČ, Real-Time, RT, R-T

(aneb obvyklý problém se zkratkami, terminologií a překladem z angličtiny)

Jak a kdy RT systém reaguje na stimuly

- okamžik, kdy musí být k dispozici výsledek = **deadline**
- pokud je výsledek platný i po uplynutí deadlinu, je tzv. měkký (*soft*), jinak je pevný (*firm*)
- jestliže mohou nastat závažné následky při zmeškání pevného deadlinu, jde o tvrdý (*hard*) deadline
- když má R-T systém alespoň jeden *hard deadline*, říkáme že je:
  - *hard R-T system* nebo
  - *safety-critical R-T system*

Zdroje pro realizaci RT systému

⟶ množství a zařazení (procesor, paměť, síť, . . .)

- Dostatečné … lze použít běžnou metodologii návrhu systému pro naplnění časových požadavků aplikace.
- Nedostatečné … aplikace má požadavky, za hranicemi současných technologií. Žádná metodologie není použitelná pro splnění časových požadavků
- Dostatečné, ale vzácné … je možné naplnit časové požadavky aplikace, ale jen s pečlivým a promyšleným způsobem alokace zdrojů.

Klasifikace R-T aplikací

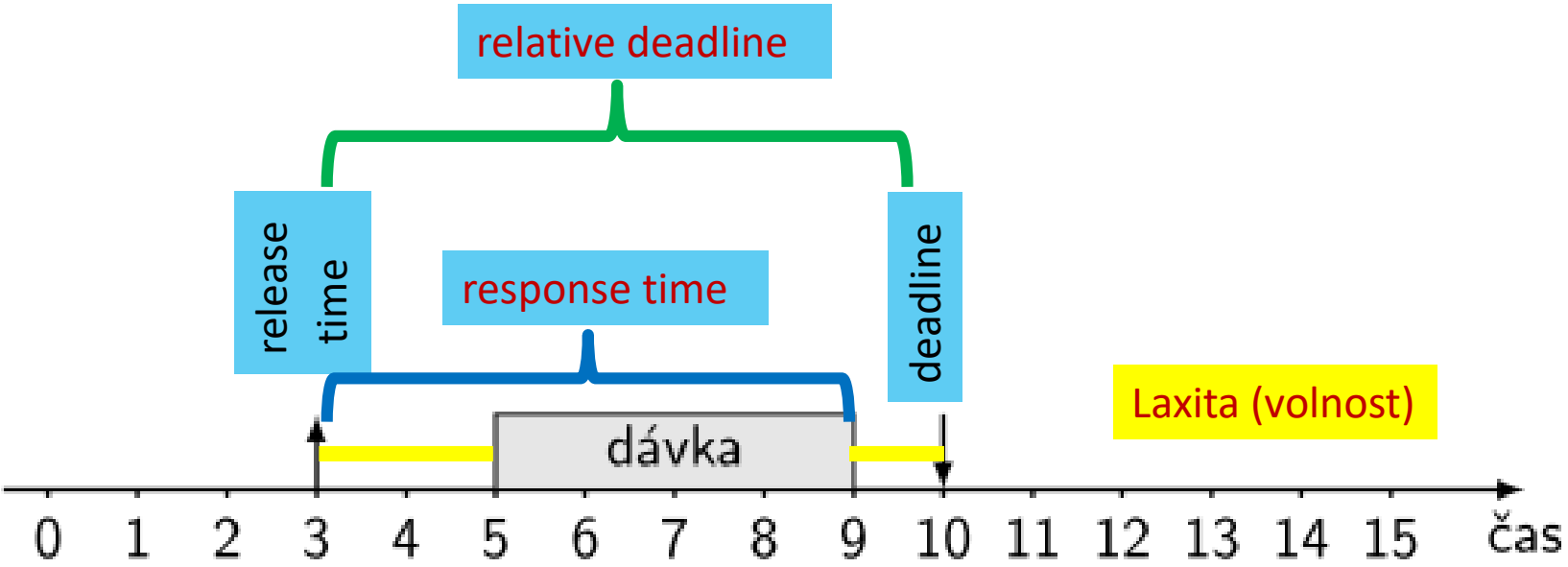
- Čistě cyklické
  - Každá úloha je spouštěna periodicky
  - Téměř neměnné požadavky na systémové prostředky
  - Příklady: digitální regulátor, řízení letu, monitorování v reálném čase
- Převážně cyklické
  - Většina úloh je spouštěna periodicky
  - Systém reaguje na nějaké externí asynchronní události
  - Příklady: Moderní avionické a řídicí systémy
- Asynchronní a predikovatelné
  - Většina úloh není periodická
  - Požadavky na systémové prostředky se mohou během po sobě jdoucích spuštění velmi lišit, ale mají známé meze či statistické rozložení.
  - Příklady: Multimediální komunikace, zpracování signálu z radaru a sledování objektů
- Asynchronní a nepredikovatelné
  - Převážně se reaguje na asynchronní události
  - Úlohy s vysokou složitostí
  - Příklady: inteligentní real-time řízení, real-time simulace a VR

Terminologie

- **Úloha** (*task*): Sekvenční úsek kódu. Množina dávek, které se spouští, aby pomáhaly vykonávat určitou funkci systému.
- **Dávka** (*job*): Instance úlohy.
  - Dávky potřebují k běhu potřebují **zdroje**.

    Příklady zdrojů: CPU, síť, disk, kritická sekce, všechny hardwarové zdroje jsou pro jednoduchost „procesory“.
- **Čas uvolnění dávky** *(release time)*: Časový okamžik, kdy je dávka připravena k běhu.
- **Termín dokončení dávky** *(deadline):* Časový okamžik, do kdy dávka musí skončit.
- **Relativní termín dokončení dávky** *(relative deadline)*: Délka intervalu mezi časem uvolnění a termínem dokončení.
- **Doba odezvy dávky** *(response time)*: Skutečný čas dokončení mínus čas uvolnění.

Příklad



- ↑ = čas uvolnění; dávka je uvolněna v čase 3.
- ↓ = termín dokončení; absolutní termín dokončení je 10.
- Relativní termín dokončení je 7.
- Doba odezvy je 6.

Proč požadovat časové garance?

- většina vestavných systémů jsou ***hard RT:***
  - *reakce na stav senzorů*
  - *bezpečná funkce (např. automatické řízení vlaku, vlak musí zastavit na signál stůj na správném místě, tzn. podle jeho aktuální rychlosti) (safety-critical)*
- kritické informační systémy :
  - musí být neustále k dispozici (pohotovost, *availability - mission-critical*)

pro bezpečné nebo vysoce pohotové systémy musí být zcela vyloučena špatná funkce

Systém bezpečný při poruše x funkční při poruše

- při detekci poruchy systém přejde do „bezpečného stavu“:
  - vlastnost řízeného objektu (ne řídícího počítače)
  - vysoká pravděpodobnost detekce poruch *(téměř 1)*

(např. železnice – při detekci poruchy všude spadnou semafory na “stůj”, nebo vlaky smí jet jen malou rychlostí přes přejezd)

  - watchdog – monitorování funkce počítače a periodické zasílání signálu „life-sign“ do watchdogu (pokud nepřijde, přechod objektu do bezpečného stavu)

x řídící systém musí fungovat na minimální úrovni (např. autopilot)

Spolehlivostní režimy

Systémy několika typů:

- *fail-operational*: pokračuje v činnosti, i když selže jeho řídící systém (výtah, nukleární reaktor, zbraně)
- *fail-safe*: když přestanou pracovat, jsou bezpečné (např. infuzní pumpa v medicíně – čekání na pomoc člověka do bezpečné doby, železniční signalizace, ..)
- *fail-secure*: maximální bezpečnost (security), když nepracují (elektronické dveře se při požáru odemknou)
- *fail-passive*: pracuje i při celkovém selhání (autopilot nechá řídit pilota)
- *fault-tolerant*: pracuje dál, i když nastane chyba/porucha, např. až do opravy (zálohování, TMR)

Systémy s garantovanou odezvou x „best-effort“

- pravděpodobnost selhání je redukovaná na pravděpodobnost, že mezní zátěž ani počet poruch reálně nenastane (= pokrytí předpokladů, *„assumption coverage“)*
  - náročná návrhová fáze (pečlivé plánování a analýza)

x „best-effort“ ... není vyžadována přesná specifikace mezní zátěže a poruch

- správnost návrhu je posouzena při testování a uvádění do provozu
- pro systémy, které nejsou „*safety-critical*“

Garantovaná odezva závisí na dostupných zdrojích, které obslouží mezní zátěž nebo chování při poruše – *neekonomické*

(proto dynamická alokace a sdílení zdrojů podle očekávané zátěže a poruch ... nelze pro safety-critical systémy)

Budoucnost: přesun k metodám s dostatečnými zdroji (řízení auta) ... Mooreův zákon, cena HW

Systémy řízené událostmi x časem

„Event-Triggered“: ET **x** *„Time-Triggered“: TT*

trigger = spouštěč, tzn. událost, která způsobí akci řídícího systému/počítače (provedení úlohy, přenos zprávy, ...)

- **ET** řízení: iniciace akcí událostí ne hodinami (periodicky), ale využitím přerušení (→ dynamické plánování)
- **TT** systém:
  - všechny akce řízené reálným časem,
  - případné přerušení jen od časovače,
  - periodické děje odvozené od CLK,
  - u distribuovaných systémů je nutná synchronizace - globální čas musí být k dispozici ve všech uzlech

Příklad ukázat na rozdílu ET a TT při řízení výtahu

Klasifikace, opakování

- Hard real-time systémy x Soft real-time systémy
- Bezpečné při poruše x funkční při poruše
- Systémy s garantovanou odezvou x „best-effort“
- S dostatečnými zdroji x s nedostatečnými zdroji
- Systémy řízené událostmi *„Event-Triggered“* x systémy řízené časem *„Time-Triggered“*

Plánovací algoritmy pro R-T systémy

Plánovací problém:

Hard R-T systém musí provést množinu souběžných R-T úloh tak, aby všechny kritické (*time-critical*) úlohy splnily své *deadliny* (termíny dokončení dávky).

Každá úloha potřebuje výpočetní prostředky, data a další zdroje.

Problém plánování = alokace zdrojů tak, aby se splnily všechny časové požadavky

Statické x dynamické

- statický plánovač plánuje předem, off-line:
  - musí mít úplnou znalost o všech úlohách (maximální čas provedení WCET, omezení, vzájemné vazby a deadliny) ⟶ *deterministické chování*
- dynamický plánovač, on-line:
  - rozhodnutí za běhu, vybírá jednu z připravených úloh, je adaptivní, uvažuje jen běžící úlohy ⟶ *nedeterministické chování*

Nepreemptivní x preemptivní

- nepreemptivní plánovač:
  - nemůže sám přerušit běžící úlohu, musí počkat až sama uvolní požadované zdroje ⟶ hodí se pro plánování (mnoha) krátkých úloh
- preemptivní plánovač :
  - běžící úloha může být přerušena, když urgentnější požaduje

Preemptivní operační systém => systém může běžícímu procesu odebrat procesor. Nad přidělováním a odebíráním procesoru procesům má preemptivní OS plnou kontrolu.

V nepreemptivních OS se proces musí procesoru sám vzdát.

Centralizované x distribuované plánování

- centrální plánovač:
  - ➢ kritický bod, možnost poruchy
- distribuovaný plánovač :
  - ➢ kritický problém je komunikace (jsou třeba aktuální informace o využití a zátěži všech uzlů)

Testy plánovatelnosti

= test zda množina připravených úloh je plánovatelná tak, aby všechny splnily své deadliny

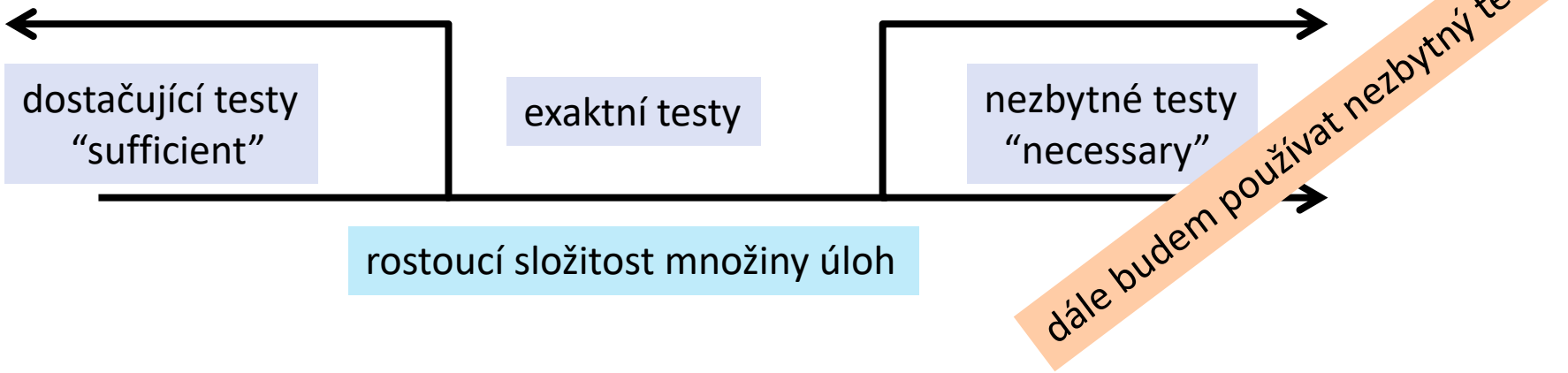
exaktní, nezbytné a dostačující testy:

ale může se to podařit i když je test negativní

ale nemusí se to podařit i když je test pozitivní

jestliže je dostačující test plánovatelnosti pozitivní, tyto úlohy **jsou** plánovatelné

jestliže je nezbytný test plánovatelnosti negativní, tyto úlohy **nejsou** plánovatelné



plánovač je optimální, jestliže najde plán vždy, když existuje

Periodické úlohy

Požadavky na provedení úloh:

u **periodických** úloh jsou po startu známé všechny časy požadavků všech dalších úloh:

T_i ... množina periodických úloh které mají:

- periody p_i,
- intervaly mezi deadliny d_i (*deadline – release time* přes všechny instance) ... též relativní deadliny
- doby výpočtu c_i (WCET)
- $l_i = d_i - c_i$ ... volnost (laxita) úlohy

Stačí zkoumat rozvrhy o délce ***nejmenšího společného násobku period úloh*** = plánovací perioda), tzn. **součet faktorů využití μ *kde μ_i je čas potřebný pro obsluhu úlohy T_i procesorem).***

Nezbytný test plánovatelnosti pro periodické úlohy je:

$$\mu = \sum c_i/p_i \leq n,$$ *kde n je počet procesorů*

Sporadické úlohy

- U **sporadických** úloh nejsou po startu apriori známé časy požadavků pro jejich provedení (release time).

→ aby byly plánovatelné, musí ∃ minimální interval mezi kterýmikoli dvěma požadavky provedení všech úloh (jinak nezbytný test plánovatelnosti selže)

Jestliže neexistuje žádné omezení pro požadavek provedení úlohy ⟶ úloha je **aperiodická**

Periodické vs sporadické úlohy

- Periodické a sporadické úlohy jsou uvolňovány opakovaně.
- Periodická úloha je uvolňována v pravidelných intervalech.
- Sporadická úloha je uvolňována v libovolných časech, ale je definován **minimální časový interval** mezi po sobě jdoucími uvolněními (*interrelease time*).
- Sporadické úlohy se vyskytují při zpracování událostí typu **vstup od uživatele nebo přerušení od různých periferních zařízení**. Události mohou nastávat opakovaně, ale doba mezi jejich výskytem se mění a může být libovolně dlouhá.

Kombinace úloh

- Co se stane, když je třeba plánovat periodické úlohy spolu se sporadickými?
- Je to běžná situace ....
- Potřebujeme dynamický plánovač (*zatím je jedno jaký*)
- Záleží na vlastnostech úloh
- Někdy nelze naplánovat, přestože test plánovatelnosti je splněn (časové poměry jsou dostačující, ale úlohy jsou na sobě závislé, např. s kritickými sekcemi)

Platný plán splňuje následující zásady:

- Každý procesor je přidělen nanejvýš jedné dávce v čase *t*
- Každá dávka je přiřazena nanejvýše jednomu procesoru v čase *t*
- Žádná dávka není plánovaná před svým časem uvolnění
- V závislosti na použitém plánovacím algoritmu, celkové množství procesorového času přiřazeného každé dávce se rovná její maximální nebo aktuální době provedení
- Jsou splněna všechna precedenční omezení a omezení využití zdrojů

Optimalita a proveditelnost

- Plán/rozvrh je **proveditelný** pokud jsou dodržena všechna časová omezení.
- Množina úloh τ je **plánovatelná (rozvrhnutelná)** plánovacím algoritmem A, pokud výstupem A je vždy proveditelný rozvrh pro τ.
- Plánovací algoritmus je **optimální** pokud je jeho výstupem vždy proveditelný rozvrh, pokud nějaký existuje (bez ohledu na výběr a použití typu plánovače).
  - Podobně se definuje optimalita pro třídu plánovačů – např. „optimální plánovač se statickou prioritou“.

Worst-Case Execution Time (WCET)

- Deadliny mohou být garantovány jen při znalosti doby provedení všech úloh a komunikace
- WCET = garantovaná horní mez pro dobu mezi aktivací a ukončením úlohy

- nalezení i horních mezí zpoždění způsobené OS (*worst-case administrative overhead* ..WCAO)
  - neřídí je přímo úloha, i když je způsobuje
  - přepínání kontextu, plánování, zpracování přerušení, změna obsahu cache, načtení chybějící stránky, DMA, ...

Jak určit WCET

Bezpečné a přesné určení WCET má zásadní vliv na kvalitu testu plánovatelnosti:

- žádný skutečný běh programu nepřekročí odhadnutý čas,
- měl by být co nejblíž reálnému času běhu programu

Metody určení WCET

- statická analýza ... statická analýza kódu bez jeho spuštění
- dynamická analýza ... měření několikerým spuštěním programu (různé vstupy), měřeno SW metodami i HW (osciloskopem, log. analyzátorem)
- Realita: u *safety-critical* RT systémů tato měření není vždy možné realizovat.

  Řešení: statická analýza (CFG –control flow graph) + zahrnutí WCAO, HW modely

WCET pro jednoduché úlohy

tzn. sekvenční úlohy běžící na dedikovaném HW, bez preempce a bez požadavků na servis OS

WCET závisí na:

1. zdrojovém kódu úlohy (pozor na nekonečné smyčky, cykly, apod.)
2. vlastnostech object kódu generovaném překladačem
3. vlastnostech HW

Praxe ... jak na to

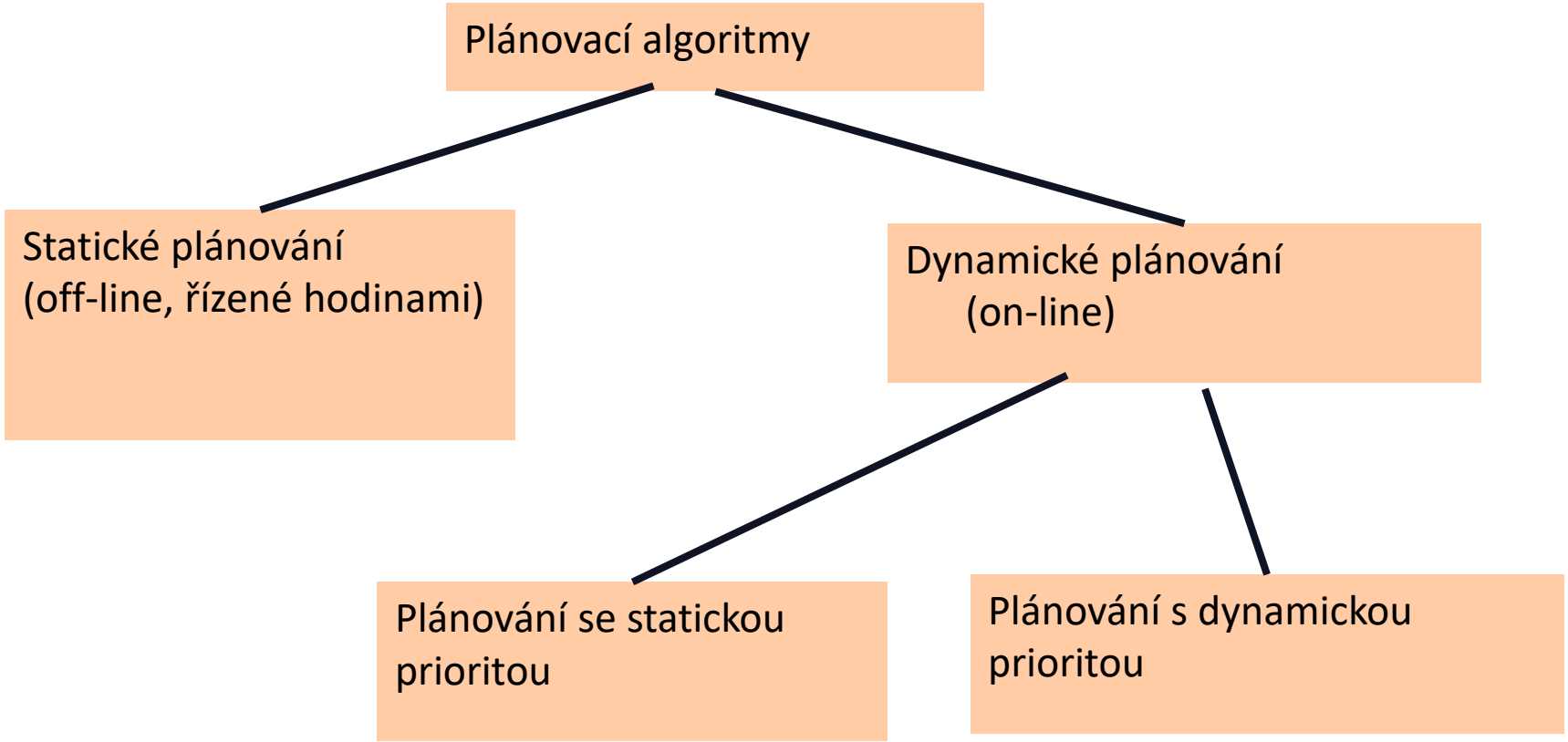
Pro hard R-T systémy je určení WCET nutné pro všechny kritické úlohy (SW i HW podrobnosti):

- redukovaná architektura (minimalizace přepínání kontextu, např. zákaz přerušení) vyžaduje analýzu řídící struktury
- modely pro WCET
- řízené měření sub-systémů pro kalibraci modelů
- implementace „anytime“ algoritmu místo WCET – zajištění meze pro WCET kořenového segmentu
- extensivní testování a měření plné implementace pro validaci požadavků

Problém s garancí výsledků získaných testováním, spíš BestCET než WorstCET ....

..... budoucnost: MPSoC s lokální pamětí

Klasifikace plánovacích algoritmů pro R-T systémy



Příklad (statické plánování)

Mějme množinu nezávislých periodických úloh pro které je určena doba výpočtu *c* a deadline interval je roven periodě (*d* = *p*), *Ti*(*c*, *p*):

{T1(5,8); T2(2,9); T3(4,13)}

(a) Vypočtěte laxitu (volnost) úloh … 3, 7, 9

(b) Určete, zda je tato množina úloh plánovatelná na jednom procesoru. Použijte nezbytný test plánovatelnosti ($\mu = \sum c_i/p_i \leq n$).

$$\mu = \frac{5}{8} + \frac{2}{9} + \frac{4}{13} = \ldots \quad 1,155$$

(c) Naplánujte pro 2-procesorový systém (off-line)

Řešení ... výtah ze slidu 23:

T_i ... množina periodických úloh které mají:

- periody p_i,
- intervaly deadlinů d_i ... též relativní deadline
- doby výpočtu c_i
- $l_i = d_i - c_i$ ... volnost (laxita) úlohy

využití procesoru: μ *(kde* μ_i *je čas potřebný pro obsluhu úlohy* T_i *procesorem).*

Nezbytný test plánovatelnosti pro periodické úlohy je:

$$\mu = \sum c_i/p_i \leq n,$$ *kde n je počet procesorů*

Plánování nezávislých úloh

Rate Monotonic algoritmus … dynamický preemptivní algoritmus založený na statických prioritách úloh

Předpoklady:

1. Požadavky pro úlohy s hard deadliny jsou periodické
2. Úlohy jsou nezávislé (žádné vzájemné vyloučení nebo precedenční omezení)
3. Relativní deadliny D_i jsou stejné jako periody p_i
4. Doba výpočtu c_i (WCET) je *a priori* známá a konstantní
5. Dobu pro přepnutí kontextu (WCAO) je možné zanedbat
6. Když pro ***n*** úloh bude platit $\mu = \sum \frac{c_i}{p_i} \leq n(2^{1/n} - 1)$ budou splněny všechny deadliny

(μ: faktor využití pro ***n*** úloh s periodami ***p***)

Liu & Layland (1973)

Praxe …

$$\lim_{n\to\infty} n(\sqrt[n]{2}-1) = \ln 2 \approx 0.693147\ldots$$

- Jestliže $\mu < 0.693$, plánovatelnost je garantovaná
- Úlohy ale mohou být plánovatelné i pro
  $\mu > 0.693$ (např. pro harmonické periody až $\mu = 1$)
- Nejvyšší prioritu mají úlohy s **nejmenší periodou** → snadná implementace
- Optimální algoritmus pro jednoprocesorový systém
- Preemptivní

Příklad:

Proces/úloha	WCET	Perioda
P1	1	8
P2	2	5
P3	2	10

- Využití: $\frac{1}{8} + \frac{2}{5} + \frac{2}{10} = 0.725$
- Teoretická mez plánovatelnosti podle vzorce:

$$\mu = 3(2^{\frac{1}{3}} - 1) = 0.77976\ldots$$

- Protože $0.725 < 0.77976\ldots$

  je systém plánovatelný RMS!!!

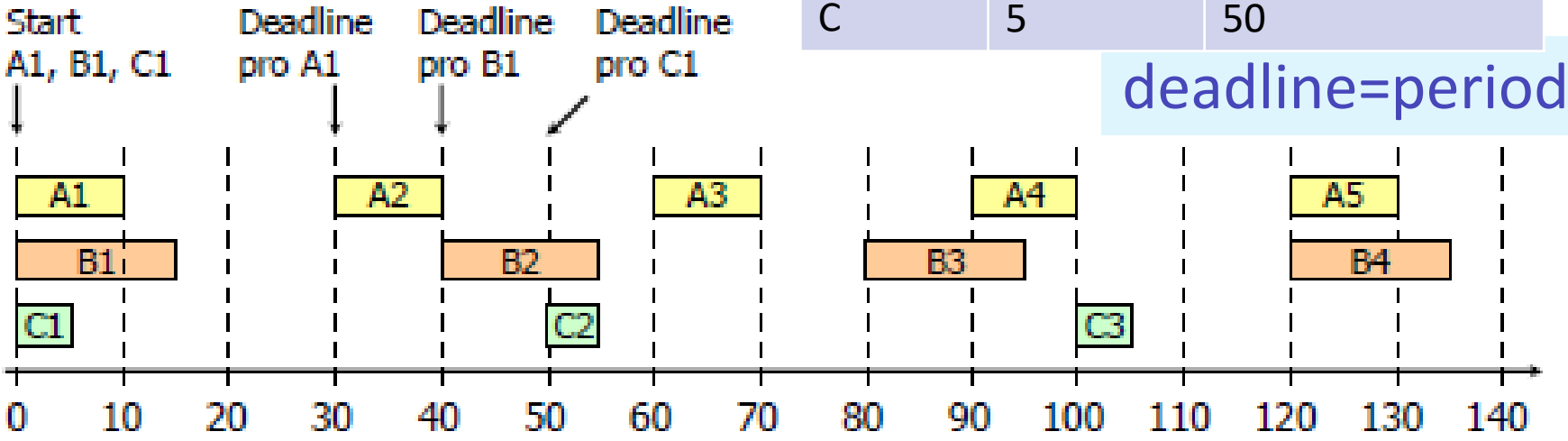
RMS

0,81 využití
Liu-Lyland limit: 3 procesy 0,77976

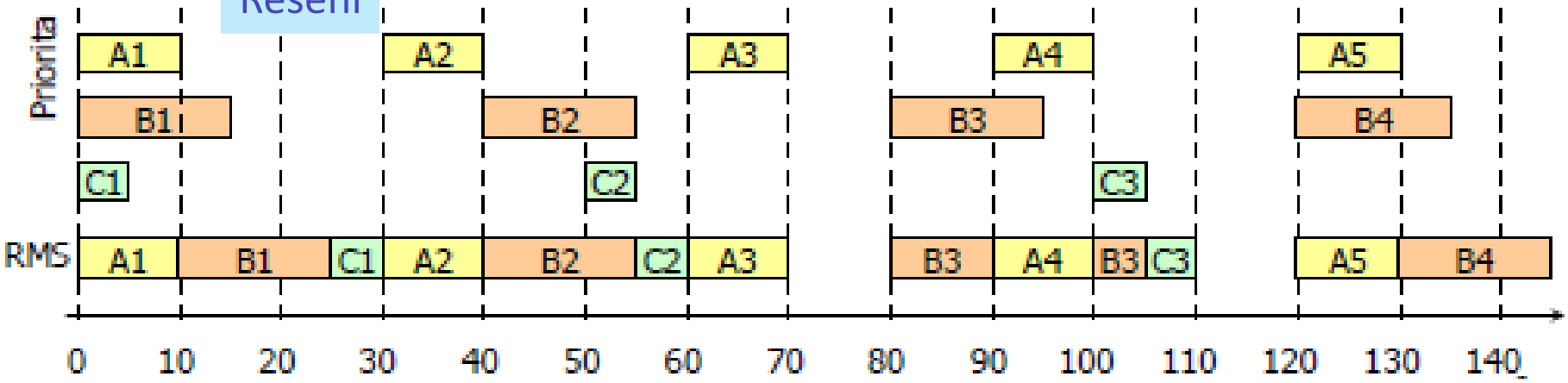
Proces	WCET	Perioda (rel. deadline)
A	10	30
B	15	40
C	5	50

nejvyšší priorita

deadline=perioda

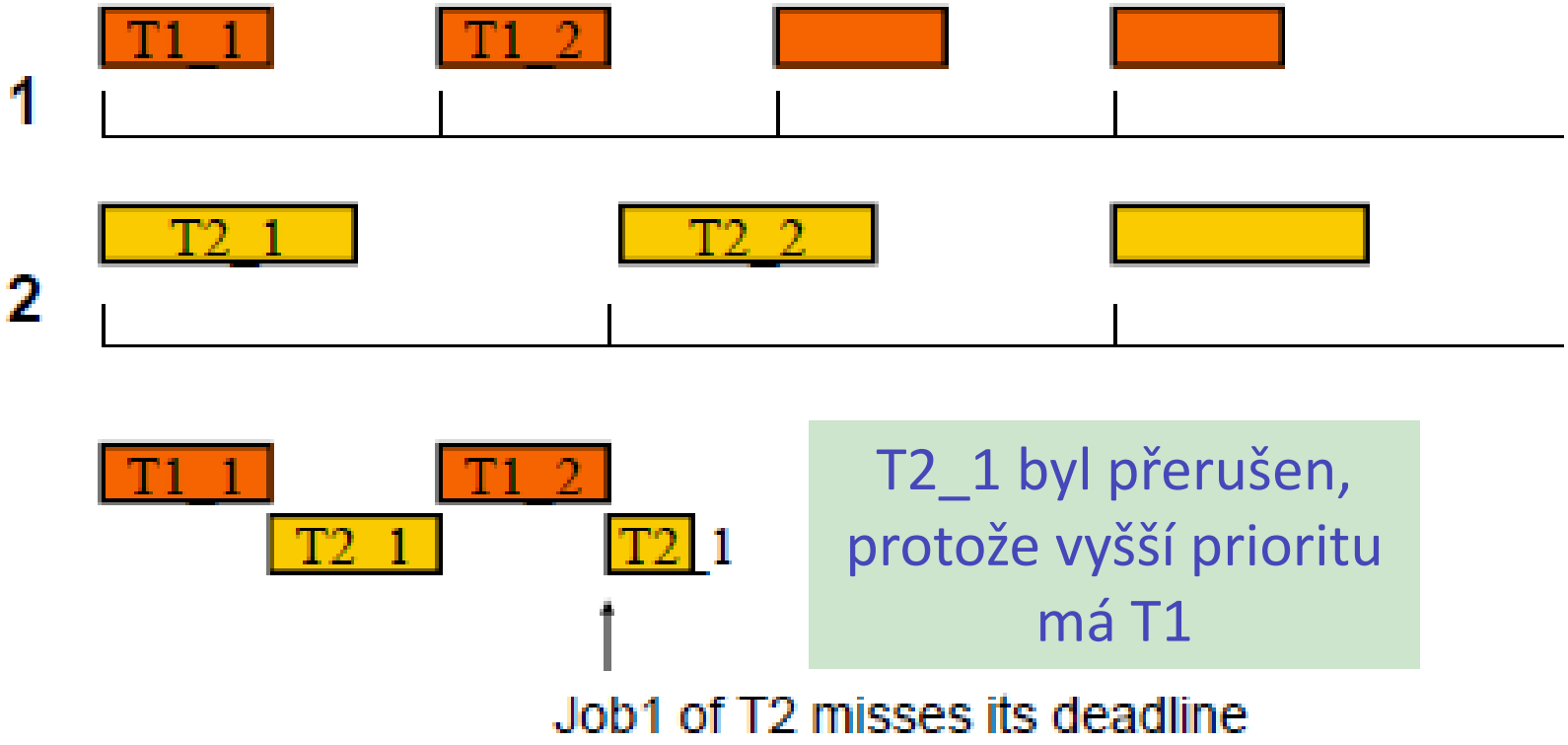


Řešení



RMS nestíhá deadline

- **T1** = (10,20), **T2**=(15,30), μ = 1 > RMS mez



T2_1 byl přerušen, protože vyšší prioritu má T1

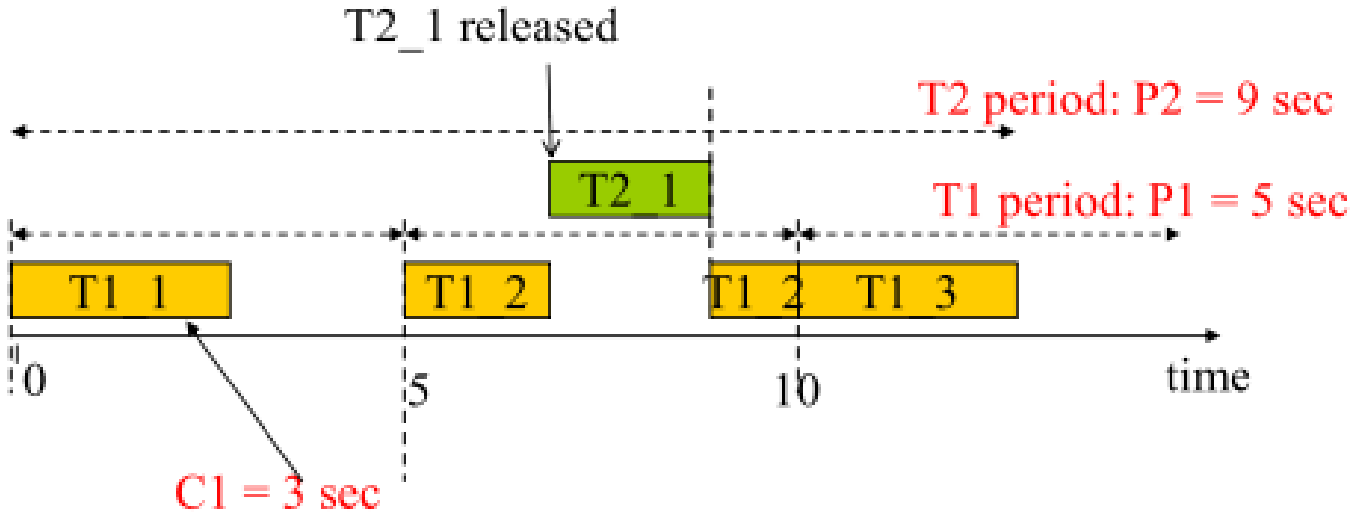
EDF

- **Earliest-Deadline First (EDF)** algoritmus
  - Dynamický preemptivní algoritmus (pro jeden procesor) založený na **dynamických prioritách**
  - Musí platit předpoklady 1 až 5 z RMS:
    1. Požadavky pro úlohy s hard deadliny jsou periodické
    2. Úlohy jsou nezávislé (žádné vzájemné vyloučení nebo precedenční omezení)
    3. Deadline intervaly (relativní deadliny) D_i jsou stejné jako periody p_i
    4. Doba výpočtu c_i je *a priori* známá a konstantní
    5. Dobu pro přepnutí kontextu je možné zanedbat
  - Faktor využití μ roste až k jedné, i když periody úloh nejsou násobky nejkratších
  - Když nastane ***významná událost***, úloha s nejbližším deadlinem dostane nejvyšší prioritu
  - Nejpoužívanější, i pro nepreemptivní

EDF – Dynamický Optimální Algoritmus

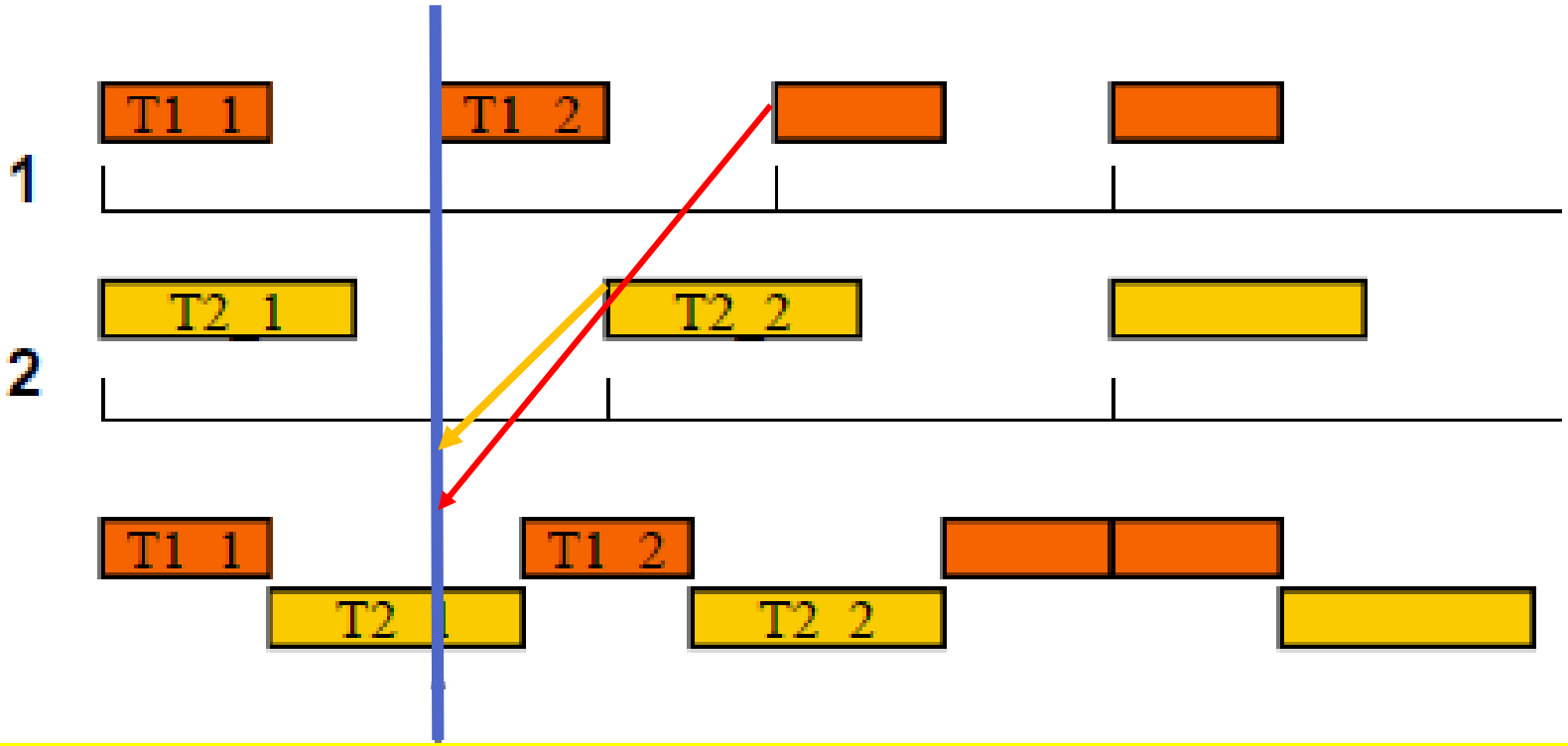
- Earliest Deadline First (EDF)
  - dřívější absolutní deadline .... vyšší priorita
  - optimální preemptivní plánovací algoritmus s dynamickými prioritami

(zvládne příklady ze slajdu 19 i 22)



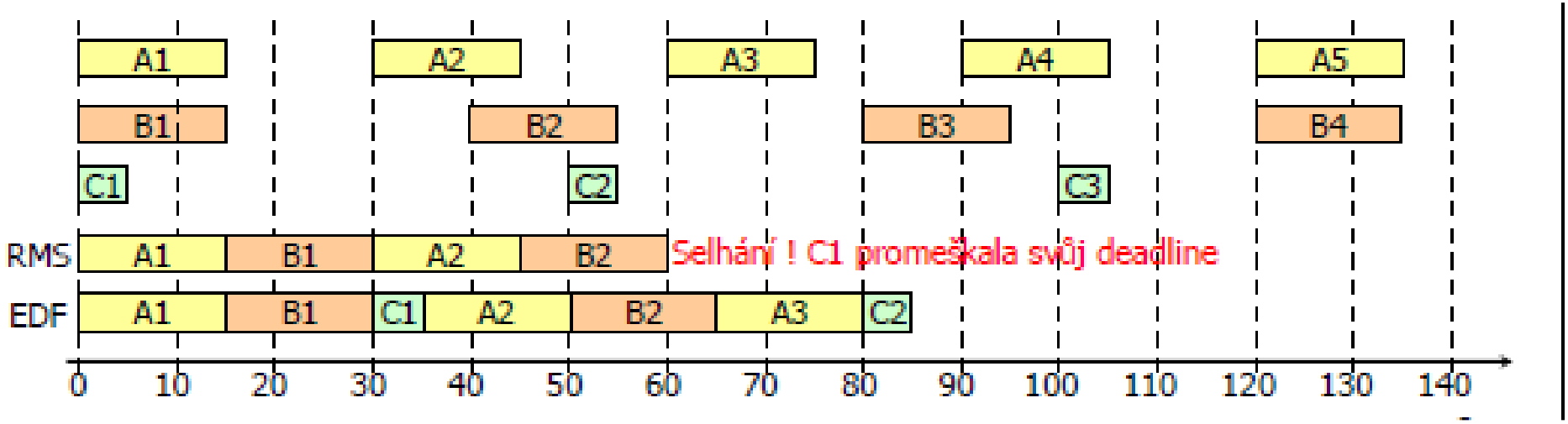
EDF stíhá deadline

- **T1** = (10,20), **T2**=(15,30), μ = 1 > RMS mez



V plánovacím okamžiku (modrá čára) se rozhoduje mezi T2-1 a T1-2
protože T2-1 má bližší deadline než T1-2,
proto T2-1 nebude přerušena (priorita se vypočítává za běhu).

Rozdíl RMS/EDF

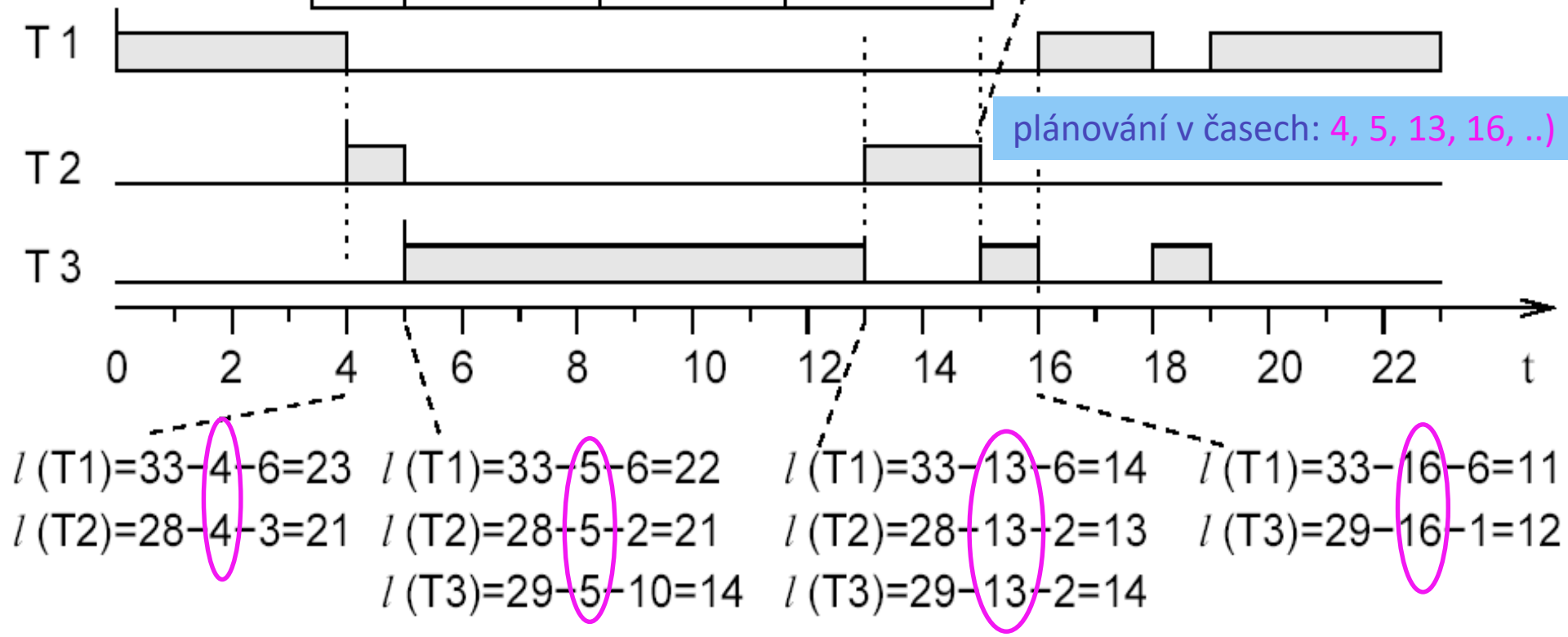


Least-Laxity (LL) algoritmus

- Podobná rozhodnutí jako EDF
- Nejvyšší dynamická priorita je přiřazována úlohám s nejmenší laxitou (volností) … $l = d - c$
- Optimální pro jednoprocesorový systém
- Pro multiprocesorové systémy nejsou EDF ani LL optimální, ale někdy je možné je použít

LLF příklad

	arrival	duration	deadline
T 1	0	10	33
T 2	4	3	28
T 3	5	10	29



Plánování závislých úloh

Prakticky: úlohy pracují souběžně a se společnými daty, zdroji, vzájemnými závislostmi (precedence, vzájemná vyloučení)

Protokol s mezními prioritami („priority ceiling protocol„)

- Plánování periodických úloh s vyloučeným přístupem ke sdíleným zdrojům řízený semafory
- Mezní priorita semaforu = priorita úlohy, která může zamknout semafor a má nejvyšší prioritu

RTOS – proč je používat a kdy

- Souběžné zpracování dat odráží přirozený paralelismus skutečných dějů.
- Synchronizace a výměna dat mezi úlohami reprezentuje interakce mezi reálnými ději.
- Oddělení funkcí (co úloha dělá) a časových charakteristik (kdy to dělá) jednotlivých úloh.
- Prostředník mezi hardwarem, uživatelskými programy a uživateli
- Pomáhá programům snadno a efektivně využívat hardware
- Správce prostředků
- Vzájemná izolace programů

Požadované vlastnosti RT OS

- Víceúlohový či vícevláknový
- Preemptivní
  - RT úloha by neměla být blokována jinými úlohami pokud to není nutné (sdílené zdroje).
- Mechanismus pro zabránění inverzí priorit, ….

  Cíl: zabránění deadlocku
- Řízený prioritami (EDF – earliest deadline first)
- Dostatečný rozsah priorit
- Definovaná maximální doba mezi vznikem přerušení a jeho obsloužením (interrupt latency)
- Dostatečně jemné rozlišení času

Distribuované real-time systémy

- Používají se z mnoha důvodů
  - Zvýšení spolehlivosti (redundance)
  - Distribuce “výkonu” na místa kde je spotřebováván
  - Subsystémy dodány třetí stranou
- Problém: *end-to-end* vlastnosti (deadline)
  - *End-to-end response-time* (doba odezvy) je ovlivňována různými zdroji (CPU, síť, ...)

Jaký programovací jazyk???

- Real-time Java
- C
- Specializované
- ???

Odlišnosti RT od klasického programování

- Snahou real-time programátorů je dosáhnout toho, aby se software vestavných **číslicových** počítačů choval podobně jako analogová zařízení.
- Ideální vestavný systém je skrytě hybridní
- Pracuje podle pravidel stanovených programátorem, ale chová se podle fyzikálních zákonů.
- Kritickým faktorem je odezva takového vestavného systému v reálném case.

Př. softwarový proces, který interaguje s fyzikálním procesem:
v okamžiku předání vstupu a výstupu se soft-time může stát real-timovým (mezitím složitá situace, např. kritická sekce).
Problém: programový čas je diskrétní.

ctd … Odlišnosti RT od klasického programování

- Softwarové procesy nelze neomezeně skládat – důvodem mohou být sdílené prostředky, které mohou vést i k deadlocku
- Real-time programování potřebuje kompozitní (souhrné, kombinované, složené) modely.
- Reálný svět je paralelní, proto i vestavný software musí být paralelní.
- Vestavný hardware je distribuovaný, tedy i vestavný software musí být distribuovaný.
- Ex. tři typy R-T programových modelů – synchronní, plánovaný a časový.

Synchronní model

- Jakákoliv výpočetní akce synchronního programu včetně komunikace trvá nulový čas
- Synchronní program se chová deterministicky, pokud zpracovává nejvýše jednu reakci pro každou událost a řídící stav.
- Pokud počítá alespoň jednu reakci pro každou událost a řídící stav, je reaktivní (tedy výpočet je konečný)

synchronní reaktivní programování:

Esterel (http://www-sop.inria.fr/meije/esterel/esterel-eng.html)

Lustre (http://www-verimag.imag.fr/Synchrone-main.html?lang=en

Plánovaný model

- Plánovaný program se skládá z procesů, vláken nebo úloh. Plánovač rozhoduje, kdy která část programu poběží
- Soft-time v plánovaném modelu je čas, který program spotřebuje pro své výpočetní aktivity
- Soft-time musí být menší nebo roven real-time
- Limity v plánovaném programu – hard a soft.

Př. Ada, Real-Time Java

- https://www.aicas.com/download/rtsj/rtsj-and-iot.pdf

Časový model

- Pevně dané nenulové časové kvantum nezávislé na skutečné době výpočtu
- Soft-time je roven real-time
- Programátor specifikuje čas, který "*timed program*" potřebuje pro výpočet výstupu
- Časová bezpečnost ("*Time safety*") závisí na výkonnosti systému, využití procesoru a plánovacím schématu
- Kompilátor zajišťuje time-safety … je-li k dispozici dostatek strojového času a pokud ne, program odmítne přeložit
- Složité ověřování (testy plánovatelnosti, analýza doby provádění programu, nemusí být v době překladu)

Př. Giotto: A Time-triggered Language for Embedded Programming

- https://link.springer.com/chapter/10.1007/3-540-45449-7_12
- https://ptolemy.berkeley.edu/projects/embedded/giotto/