Protsessoriaja planeerimine

- Üldised põhimõtted
- Plaanimise eesmärgid
- Plaanimisalgoritmid
- Plaanimine reaalajasüsteemides
- Plaanimine mitme protsessoriga süsteemides
- Näited: Solaris, Windows, Linux

Üldised põhimõtted

- Multiprogrammeerimisega saab protsessorit maksimaalselt ära kasutada
- Protsessori ja I/O kasutuse "pursete" (*burst*) tsüklid — protsessi täitmisel vahelduvad protsessorikasutus ja I/O
- Protsessorikasutuse faasi pikkus varieerub
- Protsessoriaja jagamine võib olla kas vabatahtlik (*cooperative, non-preemptive*) või väljatõrjuv (*preemptive*)

Vabatahtlik vs väljatõrjuv protsessoriaja jagamine

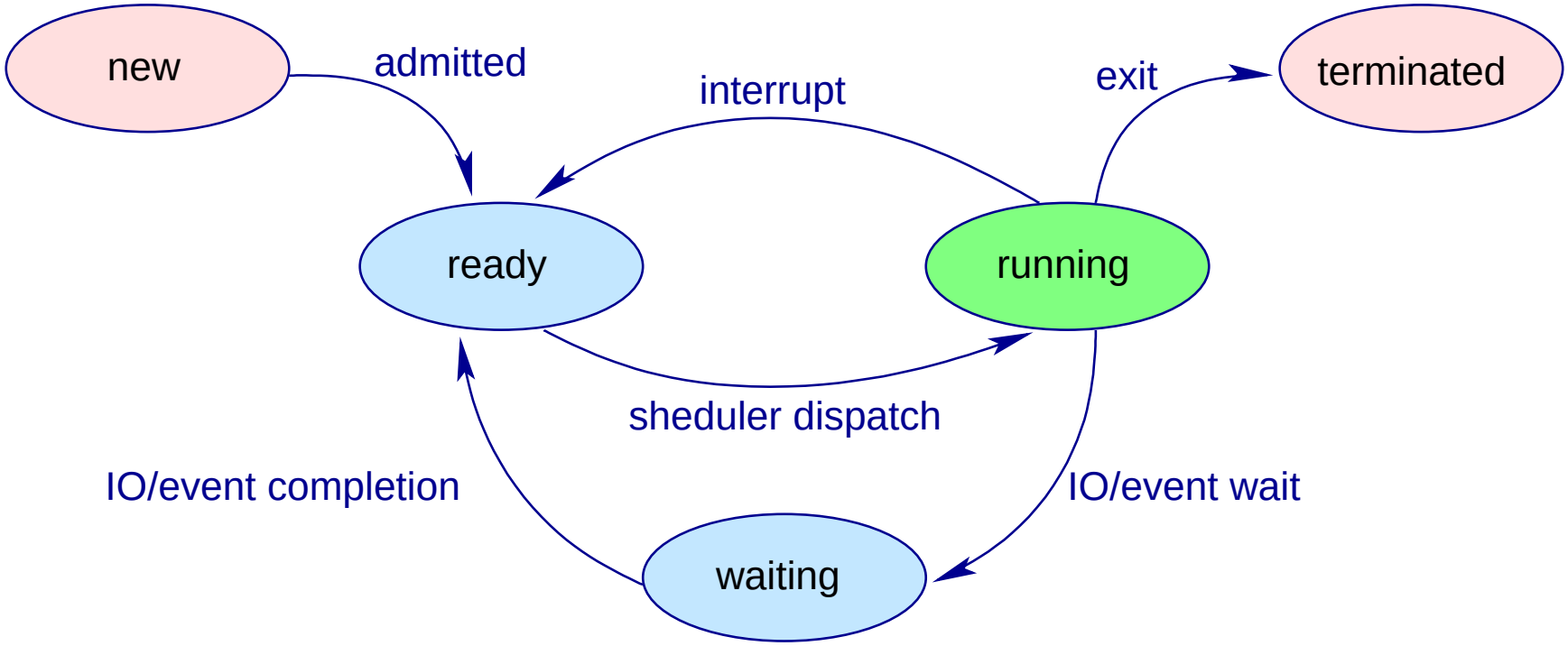
Vabatahtlik protsessoriaja jagamine:

- Iga protsess annab aegajalt juhtimise üle planeerijale
  - Süsteemifunktsioon `yield`
  - Protsess teab ise kõige paremini, millal ta protsessoriaega kõige rohkem vajab ja millal mitte
- Eeldab kõigi protsesside korrektset käitumist; ei tööta, kui mõni protsess ei loovuta kunagi oma protsessoriaega
  - kas siis sihilikult pahatahtlikul eesmärgil või
  - ”õnnetusjuhtumi” tõttu (näiteks läheb lõpmatusse tsüklisse)

Mittevabatahtlik protsessoriaja jagamine:

- Tuum kasutab programmeeritavat intervallkella perioodiliste katkestuste esile kutsumiseks ja planeerija välja kutsumiseks

Protsessi olekuskeem



Planeerija

- Planeerija valib valmis protsesside nimekirjast sobiva ja annab talle protsessoriaja
- Planeerimisotsuseid võidakse teha järgmistes situatsioonides:
  1. Protsess läheb tööolekust ooteolekusse
  2. Protsess läheb tööolekust valmis olekusse
  3. Protsess läheb ooteolekust valmis olekusse
  4. Protsess lõpetab töö
- Juhud 1 ja 4 ei tõrju protsessi sunniviisiliselt välja
- Juhud 2 ja 3 on väljatõrjuvad

Dispetšer

- Dispetšer annab juhtimise planeerija poolt valitud protsessile:
  - Kontekstivahetus
  - Kasutajatasemele lülitumine
  - Hüppamine sellele aadressile, kust programm peaks jätkuma
- Dispetšeri viivitus (*dispatch latency*) — kui kaua võtab dispetšeril aega eelmise programmi peatamine ja uue käivitamine

Plaanimise eesmärgid

- Protsessoriaja maksimaalne ära kasutamine — hoida protsessor võimalikult pidevalt töös
- Läbilaskevõime — protsesside arv, mis antud ajaühikus saab oma töö tehtud
- Valmissaamise aeg (*turnaround time*) — kui kaua aega võtab ühe protsessi valmis saamine
- Ooteaeg — kui kaua aega veedab protsess valmis järjekorras oodates
- Vastama hakkamise aeg — kui kaua võtab aega protsessi käivitamise soovist esimese väljundini jõudmiseni

Optimeerimiskriteeriumid

- Pakksüsteemid
  - Maksimeerida protsessoriaja kasutus
  - Maksimeerida läbilaskevõime
  - Minimeerida valmissaamise aeg
  - Minimeerida ooteaeg
- Interaktiivsed süsteemid
  - Minimeerida vastama hakkamise aeg
  - Kasutaja ootustele vastamine
- Reaalajasüsteemid
  - Garanteerida tähtaegade järgimine
  - Ennustatavuse tagamine

FCFS (*First Come First Served*)

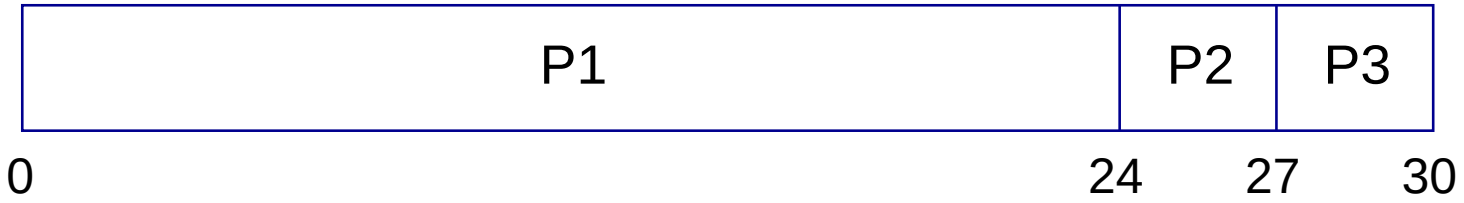
- Planeerija täidab töid saabumise järjekorras kuni valmissaamiseni
  - Modifikatsioon — kui protsess blokeerub, siis võetakse järgmine töö
- Eelised/puudused:
  - \+ Lihtne realiseerida
  - − Keskmine ooteaeg väga varieeruv
  - − Konvoiefekt
  - − CPU- ja IO-seotud protsesside kehv ülekattuvus

FCFS näide

- Näide:

Protsess	P_1	P_2	P_3
Ajahulk	24	3	3

- Protsessid saabuvad järjekorras P_1, P_2, P_3.
- Gantti diagramm:



- Ooteaeg: $P_1 = 0$, $P_2 = 24$, $P_3 = 27$
- Keskmine ooteaeg: $(0 + 24 + 27)/3 = 17$
- Saabumise järjekord P_2, P_3, P_1, siis keskmine ooteaeg 3

SJF (*Shortest Job First*)

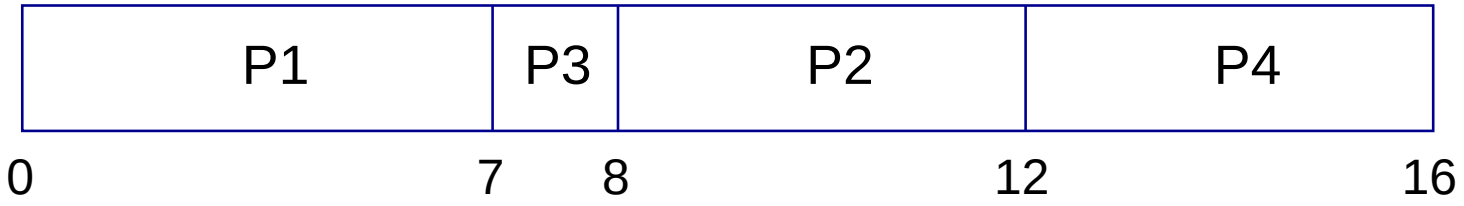
- Seome iga protsessiga järgmise protsessorisoovi ajalise pikkuse, kasutame seda lühima järgmise soovija valimiseks
- Mitteväljatõrjuv variant — protsessoriaega vägisi ära ei võeta
- Väljatõrjuv variant — kui saabub väiksema protsessoriaja sooviga protsess, siis käesolev katkestatakse
  - Seda kutsutakse ka *Shortest-Remaining-Time-First* (SRTF)
  - IO-seotud protsessid on CPU-seotud protsessidest prioriteetsemad
- Eelised/puudused:
  + Tõestatavalt optimaalne — minimeerib keskmise ooteaja
  − Vaja minevat protsessoriaega pole reeglina võimalik täpselt ennustada
  − Näljutusoht (*starvation*) suurtele CPU-seotud protsessidele

Mitteväljatõrjuva SJF näide

- Näide:

Protsess	Saabumisaeg	Ajahulk
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- SJF (mitteväljatõrjuv)



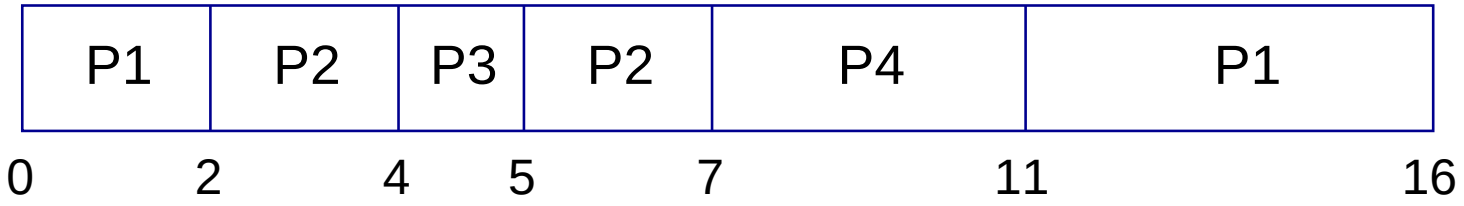
- Keskmine ooteaeg $(0+6+3+7)/4 = 4$

Väljatõrjuva SJF näide

- Näide:

Protsess	Saabumisaeg	Ajahulk
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- SJF (väljatõrjuv)



- Keskmine ooteaeg $(9+1+0+2)/4 = 3$

Järgmise ajakasutuse pikkuse hindamine

- Saame ainult ennustada varasema järgi
- Näiteks eksponentsiaalse keskmise kaudu:
  - t_n — n-le sammule kulunud tegelik aeg
  - τ_{n+1} — $n+1$-e sammu ennustatav aeg
  - α ($0 \leq \alpha \leq 1$)
- Defineerime:

$$\begin{aligned} \tau_{n+1} &= \alpha t_n + (1-\alpha)\tau_n \\ &= \alpha t_n + (1-\alpha)\alpha t_{n-1} + \ldots + (1-\alpha)^j \alpha t_{n-j} + \ldots + (1-\alpha)^{n+1}\tau_0 \end{aligned}$$

- Näide — $\alpha = 1/2$, $\tau_0 = 10$

tegelik aeg (t_i)		6	4	6	4	13	13	13	...
ennustatav aeg (τ_i)	10	8	6	6	5	9	11	12	...

Prioriteedi järgi plaanimine

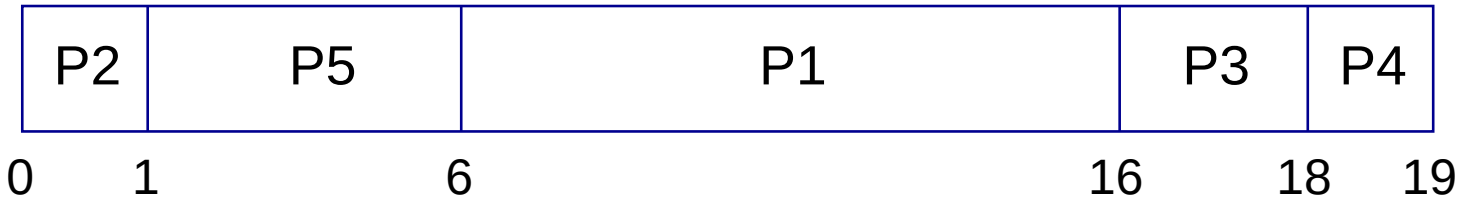
- Iga protsessiga seotakse prioriteet
- Prioriteedid on staatilised või dünaamilised
- Protsessori saab suurima prioriteediga protsess (sama prioriteedi korral FCFS)
- Nii väljatõrjuv kui mitteväljatõrjuv
- SJF on prioriteete kasutav, prioriteedi ennustab järgmine protsessorikasutusaeg
- Näljutamine — lahenduseks protsesside ea arvestamine
- Protsessoriaega mitte saanud protsesside prioriteete tõstetakse dünaamiliselt
  - Puudus — süsteemi ülekoormatusel keskmine ooteaeg kasvab tugevalt

Prioriteedi järgi planeerimise näide

- Näide:

Protsess	Ajahulk	Prioriteet
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Prioriteedi järgi plaanimine



- Keskmine ooteaeg $(6+0+16+18+1)/5 = 8.2$

Ringiratast planeerimine

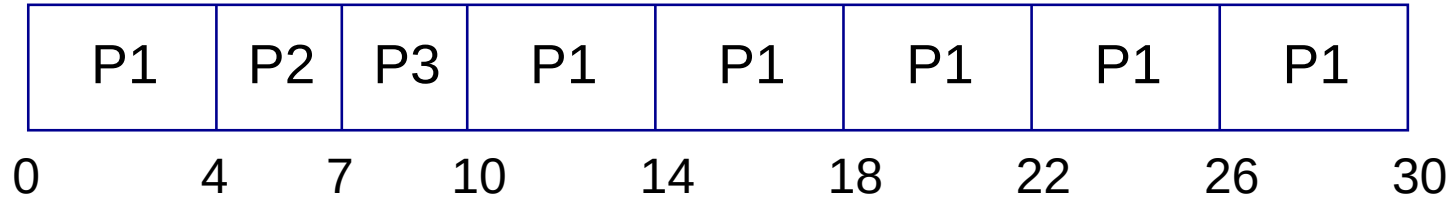
- Ingl. k *Round-Robin* (RR)
- Igale protsessile antakse väike jupp protsessoriaega (ajakvant), harilikult 10 - 100 ms
- Selle aja möödudes tõrjutakse protsess välja ja pannakse valmis-järjekorra lõppu
- Kui järjekorras on n protsessi ja ajakvandi pikkus on q, siis iga protsess saab $1/n$ osa protsessoriajast kuni q ühiku kaupa korraga. Ükski protsess ei oota kauem kui $(n-1)q$ ühikut.
- q on suur $\Longrightarrow$ FIFO
- q peab olema siiski piisavalt suur kontekstivahetusega võrreldes, muidu kulub planeerimisele liiga suur osa ajast
- Veidi pikem keskmine ooteaeg kui SJF, aga parem reageerimiskiirus

Ringiratast planeerimise näide

- Näide:

Protsess	Ajahulk
P_1	24
P_2	3
P_3	3

- Ringiratast plaanimine



- Keskmine ooteaeg $(6+4+7)/3 = 5.66$

Mitmetasemeline järjekord

- Ingl. k *multilevel queue*
- Valmis-järjekord lõhutakse mitmeks eraldi järjekorraks, millest igal võib olla oma planeerimisalgoritm. Näiteks:
  - Esiplaan (interaktiivsed tööd) — Round Robin
  - Taustatööd (pakktöötlus) — FCFS
- Kuidas erinevate järjekordade vahel aega jagada?
  - Fikseeritud prioriteet — kõigepealt kõik esiplaanil olijad, kui neid pole, siis taustatööd väljatõrjutavalt
  - Ajakvantidega — igale järjekorrale mingi protsent kogu protsessoriajast (näiteks 80% esiplaanil olijatele RR jaoks ja 20% taustatöödele FCFS jaoks)

Mitmetasemeline tagasisidega järjekord

- Protsessil lastakse mitme järjekorra vahel liikuda (paindlikkus!)
- Arvutusmahukad kolitakse automaatselt allapoole, I/O mahukad ja vanad protsessid automaatselt ülespoole
- Konkreetse mitmetasemelise tagasisidega järjekorra defineerimiseks on vaja järgmisi parameetreid:
  - Järjekordade arv
  - Iga järjekorra sisemised planeerimisalgoritmid
  - Meetod määramaks, millal protsess kõrgema prioriteedi järjekorda ülendada
  - Meetod määramaks, millal protsess madalama prioriteedi järjekorda tagandada
  - Meetod määramaks, kuhu järjekorda protsessid algul satuvad

Mitmetasemeline tagasisidega järjekord: näide

- Kolm järjekorda:
  - Q_0 — ajakvant 8 millisekundit
  - Q_1 — ajakvant 16 millisekundit
  - Q_2 — FCFS
- Planeerimine
  - Uued protsessid lisatakse järjekorra Q_0 lõppu
  - Q_0 (ja Q_1) sees valitakse täitmiseks järjekorra esimene
  - Q_0 sees antakse aega 8 ms, kui selle jooksul tööd tehtud ei saa, pannakse Q_1 lõppu
  - Kui Q_0 on töödeldud, aetakse aega Q_1 protsessidele. Kui protsessile ei piisa ka 16 ms kvandist, pannakse ta Q_2 lõppu
  - Q_2 sees kasutatakse FCFS (selle protsessoriaja sees, mis Q_0 ja Q_1 poolt üle jääb)

CFS — *Completely Fair Scheduler*

- Üks realisatsioon *Weighted Fair Queuing* (WFQ) planeerijast protsessidele
- Järjekorra asemel puu edasiste protsesside planeerimise infoga
- Nanosekundi täpsusega ajaarvestus protsessidele jagatud aja kohta
- Puus hoitakse ainult neid protsesse, mis muidu vastavas järjekorras oleksid
  - Uuesti ootele tulles algab kasutatud aeg nullist
- Puu on indekseeritud kulutatud aja järgi, alati valitakse kõige vähem aega saanud protsess
- Aja käest andmisel või ajakvandi täis saamisel pannakse puusse uue koha peale vastavalt uuele kulunud ajale

Loterii-planeerimine

- Igale protsessile antakse mingi arv loteriipileteid
- Iga ajakvandi eel loositakse juhuslikult ”võitev” pilet, mille omanik saab selle ajakvandi jagu protsessoriaega
- Keskmiselt saab iga protsess protsessoriaega proportsionaalselt piletite arvuga
- Lühikestele/kõrgeprioriteedilistele protsessidele anname rohkem pileteid ning pikematele/madalaprioriteedilistele vähem
- Näljutuse vältimiseks saab iga protsess vähemalt ühe pileti
- Süsteemi üldkoormus jaotub ühtlaselt protsesside vahel

Reaalajaline planeerimine

- Range reaalaeg — kriitilise protsessi mingi lõik tuleb garanteeritult mingi aja jooksul täita
  - Ressursside reserveerimine
  - Ei salvestusseadmeid ega virtuaalmälu
- Mitterange reaalaeg — kriitilised protsessid peavad olema prioriteetsemad kui mittekriitilised
  - Protsesside ebavõrdsus, isegi näljutamine
  - Ajas mittevähenev prioriteet
  - Dispetšeri viivitus peab minimaalne olema
- Et tuuma koodi täites liiga kaua ei peaks ootama, võetakse kasutusele väljatõrjumispunktid või iseennast väljatõrjuv tuum
- Prioriteedi inversioon ja prioriteedi pärimine

Reaalajaline planeerimine

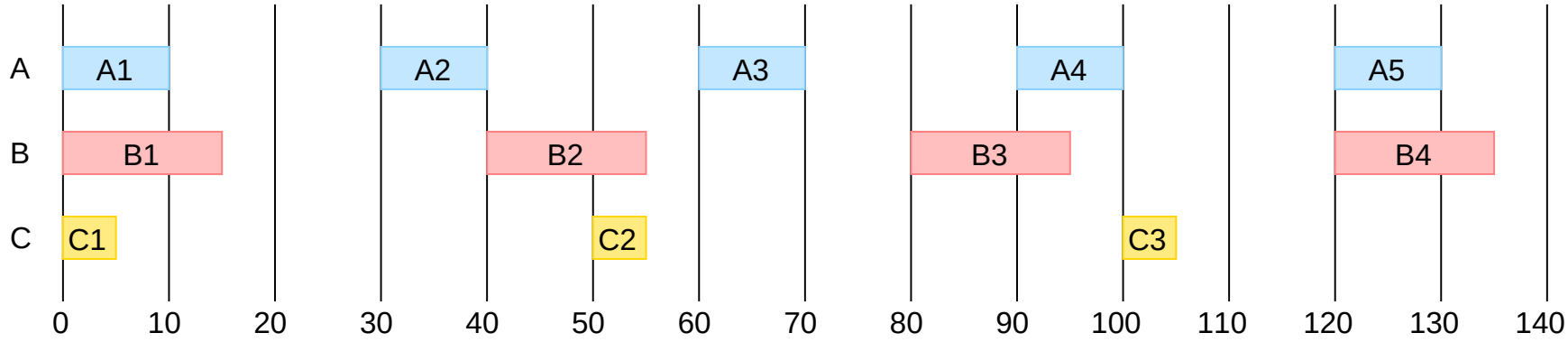
- Olgu i-nda protsessi periood P_i ning tema kestus C_i, siis süsteem on planeeritav, kui

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

- Näide:

Protsess	A	B	C
Periood	30	40	50
Kestus	10	15	5

$$\frac{10}{30} + \frac{15}{40} + \frac{5}{50} = 0.808$$



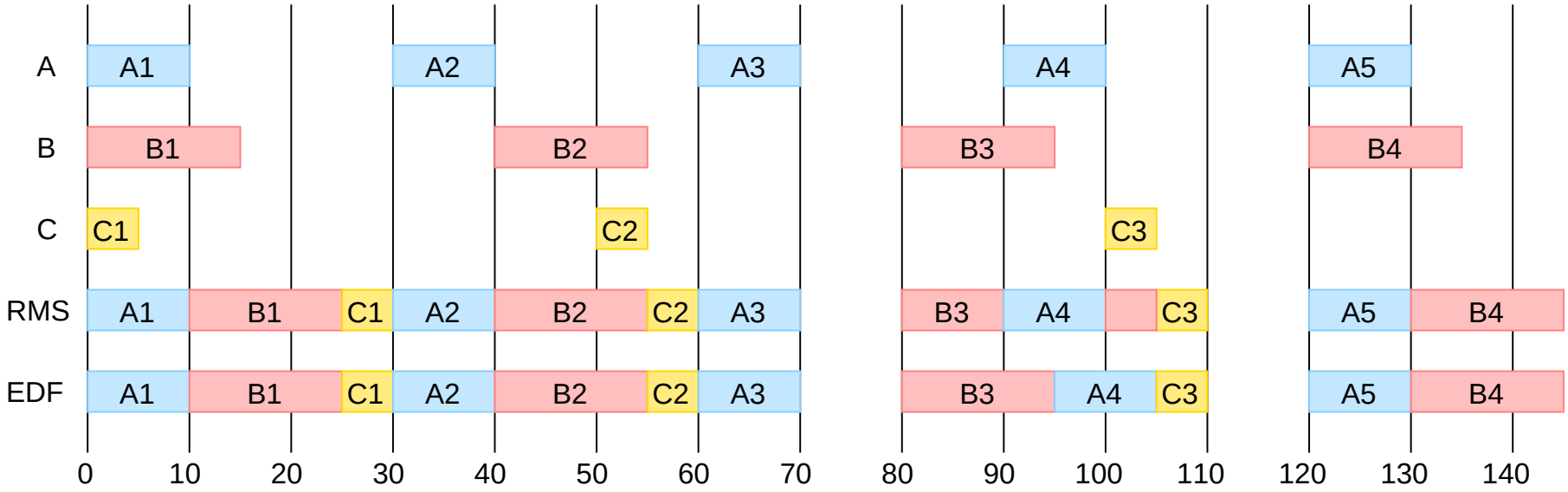
RMS (*Rate Monotonic Scheduling*)

- Staatiline reaalaja planeerimisalgoritm väljatõrjutavatele perioodilistele protsessidele
- Iga protsessiga seotakse fikseeritud prioriteet, mis on pöördvõrdeline tema perioodiga
- Kasutatav, kui on täidetud järgmised tingimused:
  - iga perioodiline protsess lõpetab oma perioodi jooksul
  - ükski protsess ei sõltu teisest
  - iga protsess vajab kõigil täitmistel sama palju aega
  - mitteperioodilistel protsessidel puudub tähtaeg
  - (protsesside väljatõrjumine toimub hetkeliselt)
- On tõestatavalt optimaalne staatiliste planeerimisalgoritmide hulgas

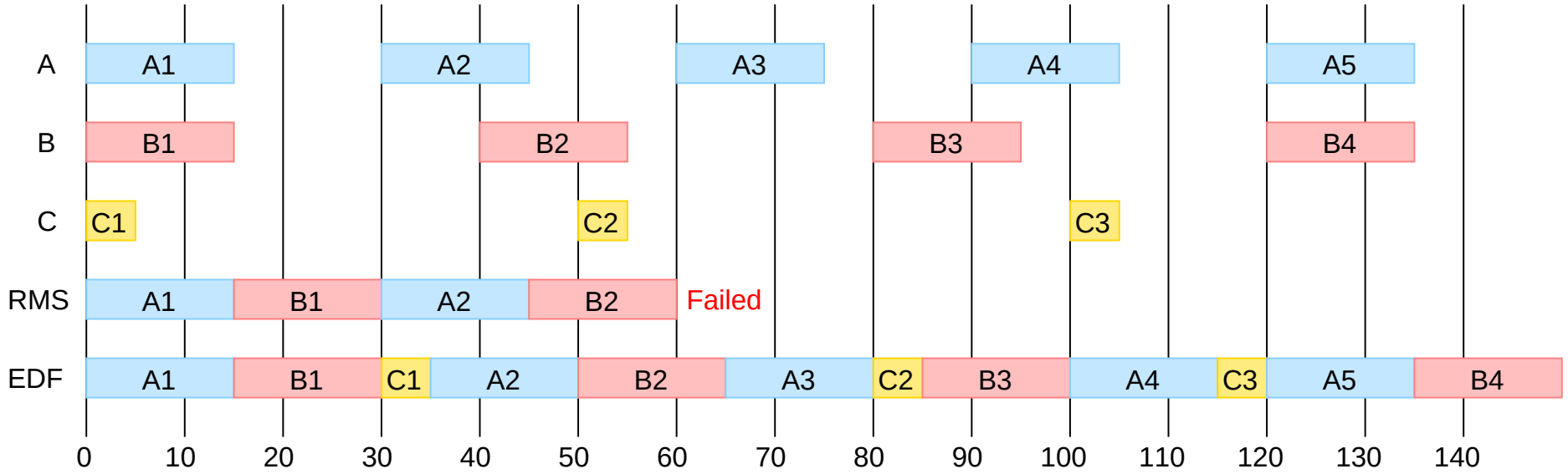
EDF (*Earliest Deadline First*)

- Dünaamiline reaalaja planeerimisalgoritm väljatõrjutavatele protsessidele
- Protsessid ei pea olema perioodilised ja ka täitmisaeg võib erinevatel täitmistel olla erinev
- Enne protsessoriaja küsimist teatab protsess oma kestuse ja tähtaja
- Alati valitakse lähima tähtajaga protsess
- Kui saabub jooksvast protsessist lähema tähtajaga protsess, siis jooksev tõrjutakse välja

RMS vs EDF näide (1)



RMS vs EDF näide (2)



- RMS töötab garanteeritult ainult juhul, kui

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq m(2^{1/m} - 1)$$

- EDF töötab garanteeritult iga planeeritava süsteemi korral

Planeerimine mitme protsessoriga süsteemides

- Planeerimine on keerulisem, kui masinas on mitu protsessorit
- Homogeensed protsessorid (SMP)
- Koormuse jagamine
- Üks või mitu valmis-järjekorda?
- Asümmeetrilised süsteemid — ainult üks protsessor puudutab süsteemseid struktuure, andmete jagamist on vähem vaja
- Protsesside migreerimine ühelt protsessorilt teisele

Näide: Solaris

- Prioriteedi järgi planeerimine
- 4 prioriteediklassi:
  - Reaalaja
  - Süsteemi (tuuma lõimed, väljatõrjuvad + prio)
  - Interaktiivsed
  - Ajajaotus (vaikimisi)
- Igal protsessil algul 1 LWP, LWP-d pärivad prioriteedi protsessilt
- Ajajaotusklassis mitmetasemeline tagasisidega planeerimine dünaamiliste prioriteetidega
- Interaktiivne klass: ajajaotussüsteem, kus aknasüsteemi rakendustel on suurem prioriteet
- Klasside sisestest prioriteetidest valitakse globaalselt kõrgeim

Näide: Windows

- Väljatõrjuv prioriteetidel baseeruv ajakvantidega süsteem
- 32 prioriteeditaset, igale oma järjekord

  0 — mäluhaldus

  1-15 — harilik klass

  16-31 — reaalajaklass

- Win32 API defineerib 6 prioriteediklassi:
  - Reaalaja
  - Kõrge
  - Kõrgem kui normaalne
  - Normaalne
  - Alla normaalse
  - Idle
- Igas klassis omaette 7 suhtelise prioriteedi taset (kokku 6×7 tabel prioriteetidega)
- Esiplaani aknaga protsess saab boonust

Näide: Windows

		Win32 process class priorities					
		Realtime	High	Above normal	Normal	Below normal	Idle
Win32 thread priorities	Time critical	31	15	15	15	15	15
	Highest	26	15	12	10	8	6
	Above normal	25	14	11	9	7	5
	Normal	24	13	10	8	6	4
	Below normal	23	12	9	7	5	3
	Lowest	22	11	8	6	4	2
	Idle	16	1	1	1	1	1

Näide: Linux

- Tööd on jaotatud kolme planeerimisklassi:
  - Ajajaotus RR (`SCHED_OTHER/SCHED_NORAML`) — CFS
  - Ajajaotus BATCH (`SCHED_BATCH`) — CFS ilma *fairness*'ita
  - Ajajaotus IDLE (`SCHED_IDLE`) — eriti madal prioriteet
  - Reaalaja FIFO (`SCHED_FIFO`)
  - Reaalaja RR (`SCHED_RR`)
  - Reaalaja DEADLINE (`SCHED_DEADLINE`) — EDF + CBS
    - CBS (*Constant Bandwidth Server*) — segule rangest ja pehmest reaalajast
- Reaalaja klassi kuuluvatel töödel prioriteet *rt_priority* 1 – 99
  - Kõigil ajajaotusklassi kuuluvatel töödel sama prioriteet 0
- RR klassidesse kuuluvatel töödel on *nice* väärtus -20 – +19 (vaikimisi 0), millele vastab prioriteet $priority = 20 - nice$

Näide: Linux

- Alates versioonist 2.6 nn. $O(1)$ planeerija, tänapäeval CFS
- Igal protsessoril on oma valmis-järjekord
- Väike overhead
- Skaleerub hästi protsesside arvu järgi (protsessi valik $O(1)$, puusse tagasi panek $O(\log N)$)
- Skaleerub hästi protsessorite arvu järgi ($O(\log N)$ planeerimine, $O(N)$ balansseerimine)
- Tugev afiinsus hoiab protsesse samal protsessoril
- Algne afiinsus hoiab omavahel ”põrgatavaid” protsesse samal protsessoril
- Interaktiivsuse automaatne avastamine
- SMT tugi (*hyperthreading* ja sugulased)