

PLÁNOVANIE PROCESOV

Two horizontal bars at the bottom of the slide: an orange bar on the left and a blue bar on the right.

Plánovacie algoritmy

2

- Používané pri
 - ▣ Krátkodobom plánovaní
 - Prepína sa kontext procesov
 - Efektívnejšie sa využíva CPU
 - Potrebné
 - keď proces končí
 - v iných prípadoch

Prepínanie kontextu

3

□ Stav procesu sa mení z:

□ **bežiaci na pripravený**

- Spôsobuje ho prerušenie od časovača, napr. vypršanie časového kvanta

□ **pripravený na bežiaci**

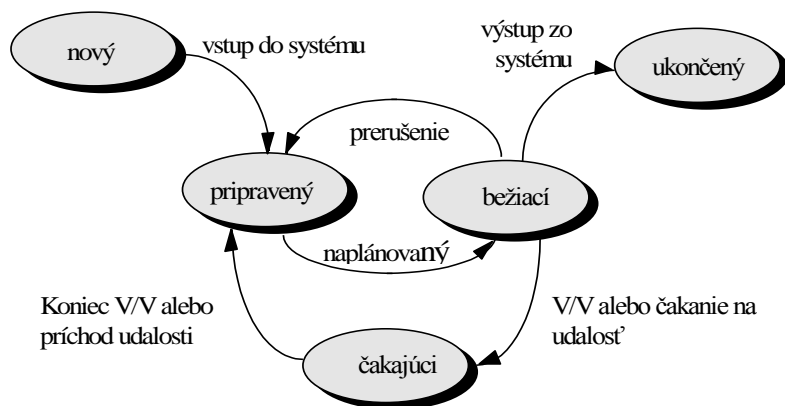
- Spôsobuje ho rozhodnutie plánovača

□ **bežiaci na zablokovaný**

- Spôsobuje ho I/O alebo iné udalosti

□ **zablokovaný na pripravený**

- Pri ukončení I/O operácie
- Pri výskyte udalosti, na ktorú proces čaká



Dispečer

4

- ▣ Dôležitý prvok plánovania
- ▣ Po rozhodnutí ktorý proces sa spustí (podľa použitej politiky) sa stará o
 - **prepnutí kontextu** vrátane doplnenia alebo odstránenia procesov z príslušných frontov
 - **prepnutí do užívateľského režimu z režimu supervizora**, v ktorom sa vykonáva plánovač
 - **skok na príslušný bod do užívateľského adresného priestoru** a vykonanie momentálne „bežiaceho“ procesu

Metriky (kritéria) v plánovaní

5

- **Efektívnosť využitia CPU** : percento z času, ktorý CPU strávi aktívnymi výpočtami v prospech užívateľského procesu
- **Priepustnosť** : počet ukončených procesov pre danú časovú jednotku
- **Čas vykonania**: časový interval od vzniku procesu do jeho ukončenia
- **Čas čakania** : čas, ktorý proces strávi čakaním vo frontu pripravených procesov
- **Čas odozvy** : čas od vystavenia požiadavky do prvej odozvy na túto požiadavku
- Posledné dve sú ovplyvnené priamo plánovačom

Správanie procesu

6

- ▣ CPU verzus I/O zhluikov
- ▣ Chovanie procesu je postupnosť
 - aktivít na CPU (*CPU* zhluiky)
 - beh iných (nie-CPU, obyčajne OS) aktivít alebo OS zhluikov
- ▣ Celé vykonanie procesu pozostáva z prechodov medzi tými zhluikmi aktivít

Preempcia

7

- Hlavná klasifikácia plánovacích algoritmov
- **Preemptívne** verzus **nepreemptívne** plánovanie

Odpovedajúca politika plánovania

- je **nepreemptívna** ak

- proces sa prepína do stavu zablokovaný **len v dôsledku svojho vlastného správania**

- napr. ak volá službu OS alebo keď skončí

- je **preemptívna** ak

- prepínanie stavov má aj **iné príčiny**

Rozhodnutie o pridelení CPU a stavový diagram

8

- Môže sa urobiť vždy pri jednom z nasledovných prechodov:

1. **zo stavu bežiaci do stavu čakajúci**

- (čakanie na dokončenie V/V operácie alebo čakanie na ukončenie potomka),

2. **zo stavu bežiaci do stavu pripravený,**

3. **zo stavu čakajúci do stavu pripravený,**

4. **keď proces končí.**

Pri prechodoch v bodoch 1 a 4 nemáme možnosť výberu.

Nový proces sa musí vybrať pre vykonanie.

- Keď sa plánovanie vykonáva len v prípadoch 1 a 4 - jedná o *nepreemptívne* plánovanie, ináč plánovanie je *preemptívne*.

Algoritmus FCFS (First Come First Served)

9

- Spracovanie v poradí príchodu
- Nepreemptívny
- Algoritmus implementuje:
 - ▣ front procesov
 - ▣ nový proces sa zaraďuje na koniec frontu pripravených procesov
 - ▣ keď proces skončí
 - CPU sa prideli procesu, ktorý je na začiatku frontu

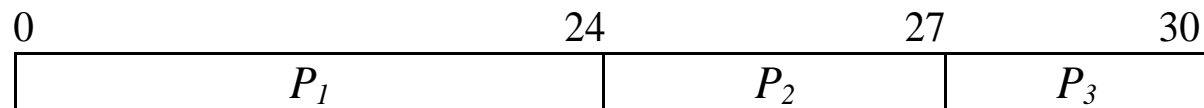
Spracovanie v poradí príchodu (FCFS)

10

- Veľmi jednoduchý kód a údajová štruktúra – front FIFO, čo znamená mala réžia
- Stredná doba čakania pri použití FCFS je často veľmi dlhá
- Príklad

Proces	Požadovaný čas procesora
P_1	24
P_2	3
P_3	3

Gantov diagram:



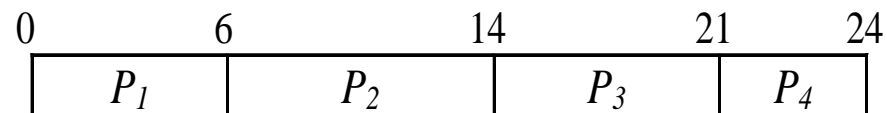
Priemerná doba čakania: $(0 + 24 + 27)/3 = 17 \text{ ms}$

Iný príklad FCFS

11

<i>Proces</i>	Požadovaný čas procesora	Doba čakania
P_1	6	3
P_2	8	16
P_3	7	9
P_4	3	0

Gantov diagram:



Priemerná doba čakania je $(0+6+14+21)/4 = 10.25$ ms

Nedostatky FCFS

12

- Nie je preemptívny
- Ťažko použiteľný v time-sharing-ových systémoch
- Môže neúmerne predĺžiť čas čakania krátkych procesov.
- Špecifický problém:
 - ▣ ak je jeden dlhý proces s veľkými požiadavkami na CPU a skupina iných procesov, ktoré požadujú intenzívne I/O operácie

Varianty použitia FCFS

13

- Algoritmus, podobný predchádzajúcemu ale spúšťaný tiež:
 - ▣ keď sa proces zablokuje
 - ide na koniec frontu a CPU sa prideluje procesu, ktorý je na začiatku frontu
- V kombinácií s inými algoritmami

Najkratší proces najskôr (SJF)

14

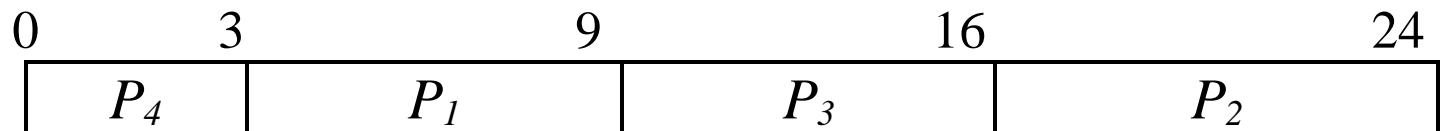
- ❑ Poradie spracovania procesov sa určuje **podľa požadovanej doby obsluhy procesu** - vyberá proces, ktorý potrebuje **najmenšiu** dobu na svoje dokončenie (ak proces medzitým bol čiastočne spracovaný)
- ❑ Podľa FCFS, ak sú rovnako veľké

Príklad SJF

15

<i>Proces</i>	Požadovaný čas procesora	Doba čakania
P_1	6	3
P_2	8	16
P_3	7	9
P_4	3	0

Gantov diagram:



Priemerná doba čakania je $(3+16+9+0)/4 = 7$ ms

SJF - prednosti a nedostatky

16

□ Prednosti

- ▣ Ak časy sú dobre odhadnuté – SJF dáva **minimálnu priemernú dobu čakania**
- ▣ často sa používa pri dlhodobom plánovaní.

□ Nedostatky

- ▣ **potrebné dopredu vedieť dĺžku požadovanej doby obsluhy**
- ▣ riešenie – odhad
 - Pri dávkovom spracovaní - časový limit pre spracovanie dávky, ktorý zadáva užívateľ.
 - Očakávame, že ďalšia požiadavka bude mať dĺžku podobnú predchádzajúcim

Odhad dĺžky d'alšej požiadavky

17

- ▣ **Pre krátkodobé plánovanie** – odhad na základe predchádzajúcich hodnôt
 - Snažíme sa odhadnúť nejakú lokalitu v CPU zhlukoch procesu

$$\tau_{n+1} = \alpha \cdot t_n + (1 - \alpha) \cdot \tau_n$$

kde

τ_{n+1} - predpoveď dĺžky $n+1$ -vej požiadavky

t_n - skutočná dĺžka n -tej požiadavky na čas procesora

τ_n - predpoveď dĺžky n -tej požiadavky

$0 \leq \alpha \leq 1$, obyčajne $\frac{1}{2}$

Tento vzorec definuje tzv. **exponenciálny priemer**

Príklady odhadu parametru α

18

$$\tau_{n+1} = \alpha \cdot t_n + (1-\alpha) \cdot \tau_n$$

□ $\alpha = 0$

$\tau_{n+1} = \tau_n$ Najbližšia história nemá vplyv.

□ $\alpha = 1$

$\tau_{n+1} = t_n$ Vplyv má len aktuálny CPU cyklus.

□ Ak rozvineme vzorec, máme :

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

□ Pretože α a $(1 - \alpha)$ sú ≤ 1 , každý ďalší člen má menšiu váhu ako predchádzajúci

Modifikácie SJF

19

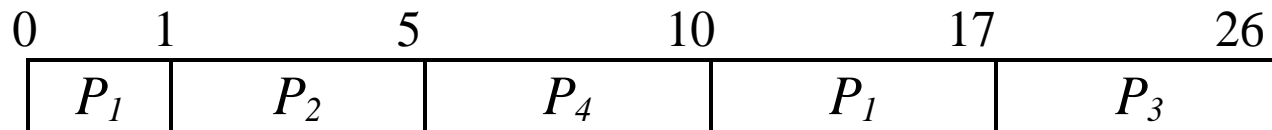
- ▣ Prípád, keď do frontu pripravených procesov príde nový proces a *predchádzajúci sa ešte vykonáva*.
- ▣ Nový proces môže mať menšie požiadavky na čas procesora ako zostávajúce požiadavky práve vykonávaného procesu.
 - Preemptívny algoritmus prepne bežiaci proces,
 - niekedy sa nazýva *plánovanie podľa najkratšej zostávajúcej doby na vykonanie* (***shortest remaining time first***).
 - Nonpreemptívny ho nechá dobehnúť.

Príklad preemptívneho SJF

20

Proces	Čas príchodu	Požadovaný čas procesora
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

Gantov diagram:



Priemerný čas čakania z tohto príkladu je

$$((10 - 1) + (1 - 1) + (17 - 2) + (5 - 3))/4 = \mathbf{6.5 \text{ ms.}}$$

Nepreemptívne plánovanie by dosiahlo čakaciu dobu $\mathbf{7.75 \text{ ms.}}$

Prioritné plánovanie: všeobecne

21

- ▣ Prirad'uje sa **numerická priorita** každému procesu

- ▣ konvencia: menšie číslo znamená vyššiu prioritu

Príklad

priorita = (odhadnutá) hodnota d'alšieho CPU cyklu

- ▣ Procesor sa pridel'uje procesu s najvyššou prioritou
- ▣ Môže byť preemptívny alebo nepreemptívny.
- ▣ Priority sa môžu zakladať na
 - ▣ *externých alebo interných pozorovaniach*

Pridel'ovanie priorít

22

□ Externé

▣ na základe napr.

- dôležitosti skupiny ku ktorej patri užívateľ, spúšťajúci proces
- ekonomické investície do systému

□ Interné

▣ na základe napr.

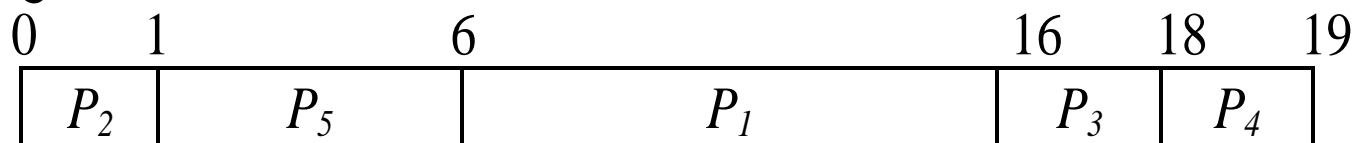
- Potrieb úlohy – pamäť alebo iné
- pomer časov, strávených pri CPU a I/O
- počet otvorených súborov

Prioritné plánovanie

23

Proces	Požadovaný čas procesora	Priorita
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

Gantov diagram:



Priemerná doba čakania je **8.2 ms**.

Nedostatky prioritného plánovania

24

- ▣ Proces môže byť sústavne predbehovaný procesmi s vyššou prioritou, ktoré vznikli neskôr
- ▣ Môže viesť ku starvácií
- ▣ Vedie k lepšiemu výkonu ale
 - nie z hľadiska procesov, ktoré sú plánované
- ▣ Ak sa neprijmú dodatočné opatrenie, predbiehanie sa môže prihodiť aj v skutočnom OS

Bežné riešenie

25

- **Postupne zvyšovanie** priorít procesov, ktoré dlho čakajú.
 - ▣ FCSF sa použije v prípade rovnakého času vzniku
- Proces nebude čakať nekonečne
 - ▣ čakaním vo fronte pripravených procesov sa mu priorita zvýši

Variant: Unix

26

- Priorita sa zvyšuje pri menšom využití CPU
- Unix
 - ▣ akumuluje časy na CPU
 - ▣ každú časovú jednotku (~ 1 sekunda)
 - **prepočítava priority**
 - $\text{priorita} = \text{CPU-využitie} + \text{zákl. priorita}$
 - **prepočítava CPU-využitie :**
 - $\text{CPU-využitie} = (\text{CPU-využitie}) / 2$
 - ▣ základnú prioritu si nastavuje užívateľ
 - Proces môže byť “nice”

Zhrnutie

27

- **Striktne nepreemptívne** algoritmy
- Algoritmy, ktoré môžu byť **aj preemptívne**, podľa voľby

Pokračujeme s algoritmami ktoré sú **striktne preemptívne**



Cyklické plánovanie - Round Robin (RR)

28

- ▣ Všeobecne
 - Pridel'uje CPU v časových *kvantách* (q)
 - Plánovač pridel'uje postupne každému procesu z frontu jedno časové kvantum
 - Po uplynutí kvanta prepína proces
- ▣ Obyčajne FCFS sa používa pre serializovanie - každý nový proces sa pridá na koniec frontu pripravených procesov
 - Keď je proces zablokovaný alebo je prepnutý, presunie sa na koniec frontu
- ▣ **Veľmi často používaný algoritmus pre interaktívne systémy**

Výber kvanta (q)

29

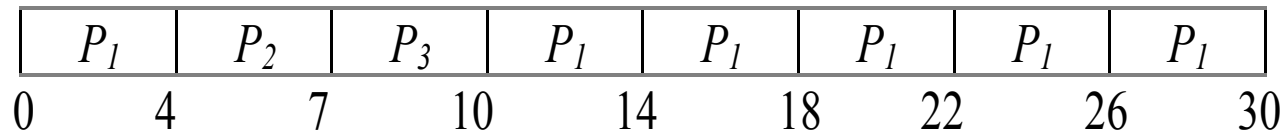
- Hodnota q je veľmi dôležitá
- Ovplyvňuje čakanie a časy prechodu cez systém
 - Nech q je veľkosť kvanta
 - n počet procesov vo frontu pripravených procesov
 - čakanie môže byť max. $(n-1) \cdot Q$ časových jednotiek
 - Ak q vzrastie  FCFS
 - Ak q klesá
 - zvýši sa počet prepnutí  zvýši sa réžia
 - priemerná doba čakania klesá a rýchlosť systému klesá na $1/n$ z rýchlosti pôvodného systému

Príklad plánovania pomocou RR

30

Proces	Požadovaný čas procesora	$q = 4 \text{ ms}$
P_1	24	
P_2	3	
P_3	3	

Gantov diagram:

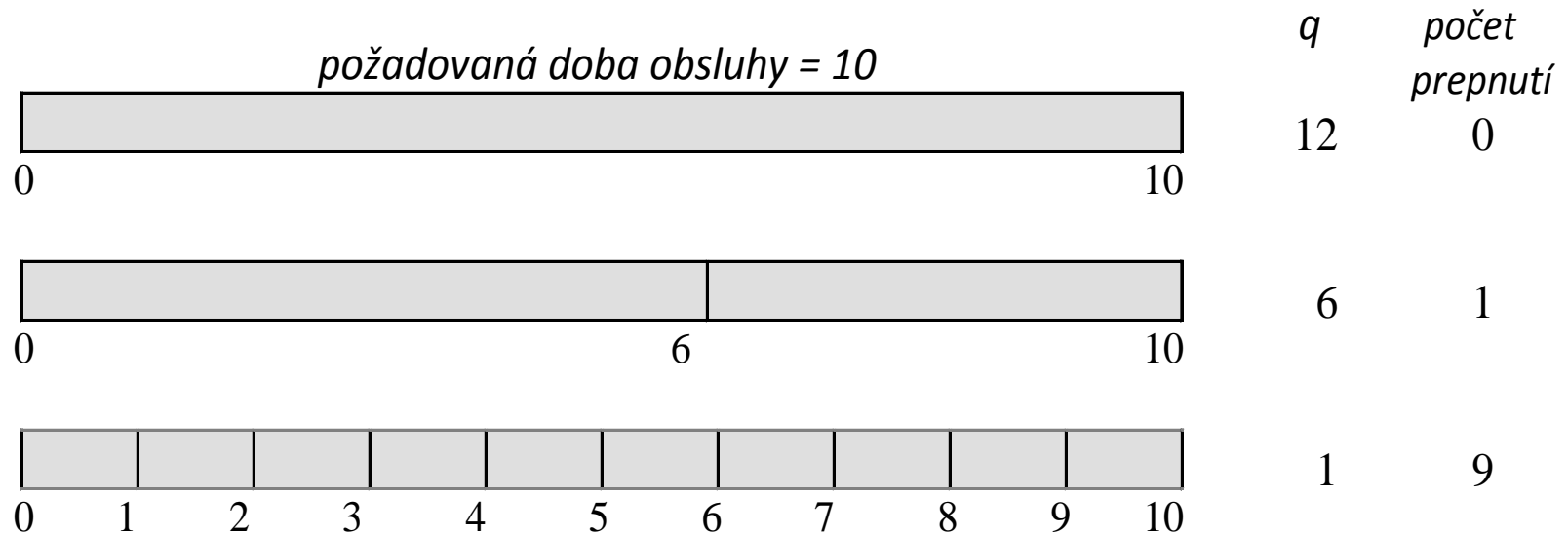


Priemerná doba čakania je $17/3 = 5.66 \text{ ms}$.

RR – Vplyv veľkosti časového kvanta na počet prepnutí

31

Máme len jeden proces s požadovanou dobou obsluhy 10 ms.



Plánovanie s viacerými frontmi

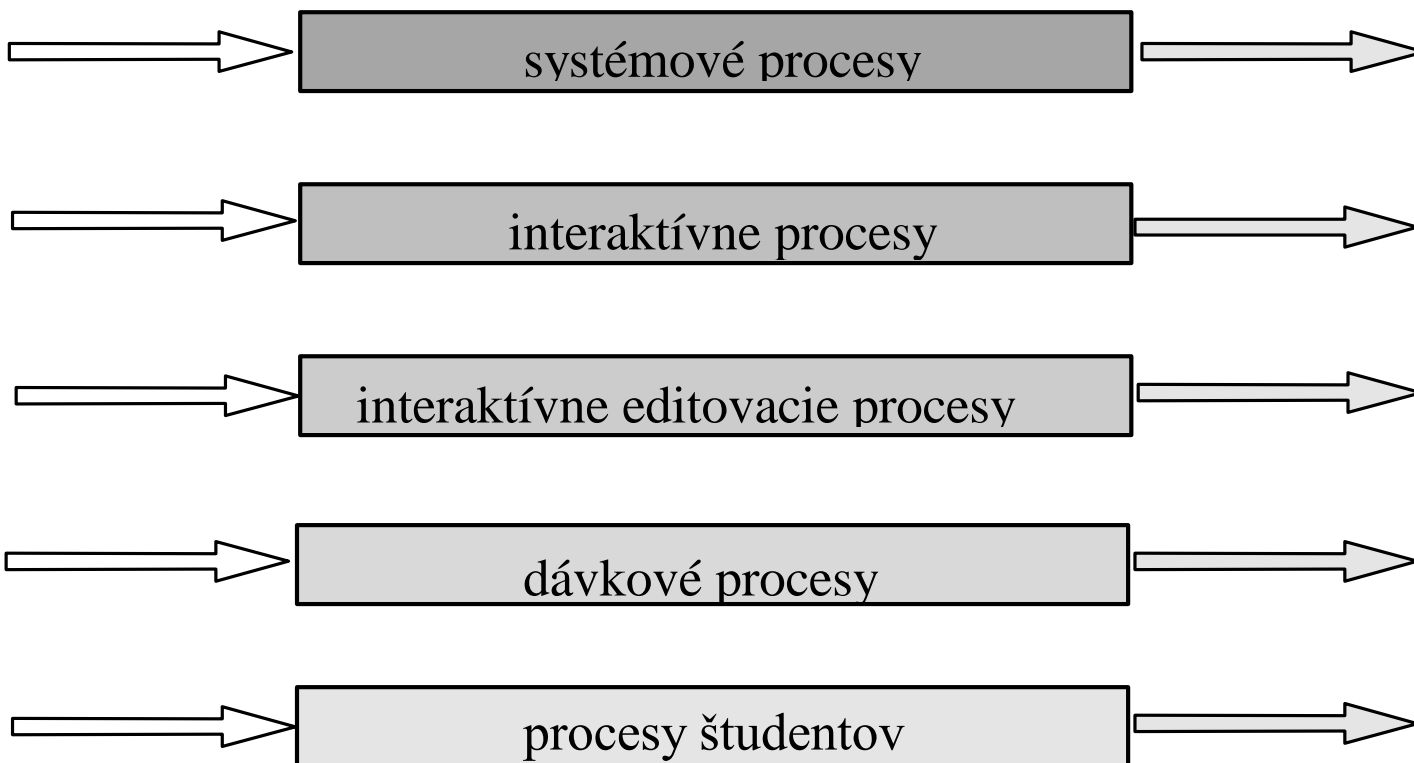
32

- Pre situácie, kedy sa procesy dajú ľahko rozdeliť na rôzne skupiny.
 - Napríklad, veľmi často procesy v systéme sa delia na procesy na pozadí a procesy na popredí.
- Všetky procesy z danej skupiny zdieľajú tu istú stratégiu plánovania a svoju „rodinu“ frontov
- Jednotlivé skupiny môžu mať rôzne plánovacie algoritmy
- CPU sa prideluje jednotlivým skupinám tiež podľa nejakého pevného plánovacieho algoritmu, obyčajne podľa pevných priorít

Plánovanie s viacerými frontmi

33

Najvyššia priorita

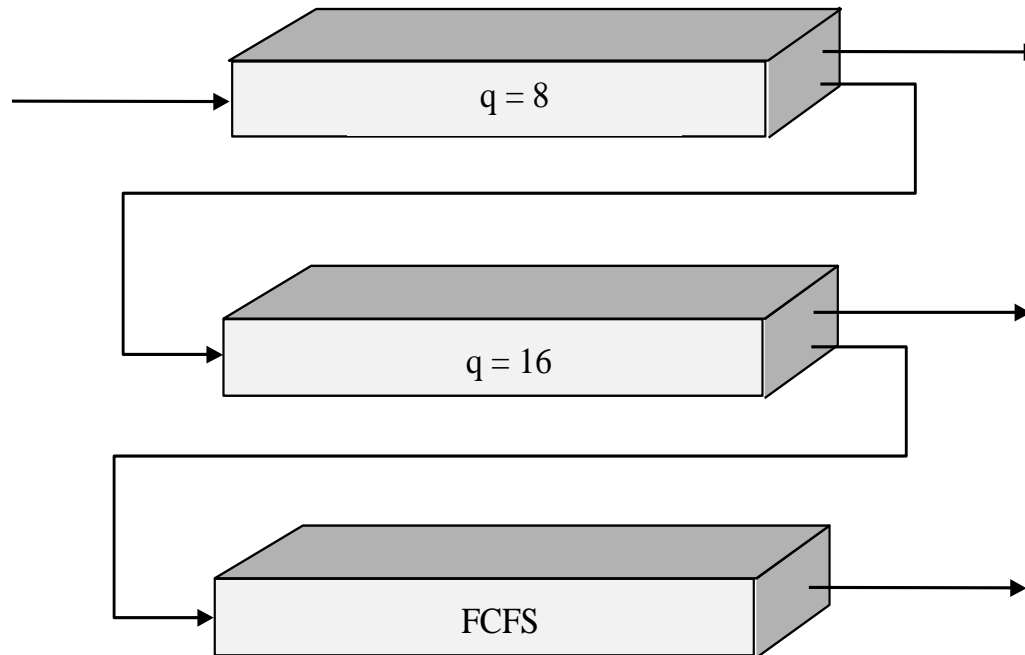


Najnižšia priorita

Plánovanie s viacerými frontmi so spätnou väzbou

34

- Procesy sa delia podľa ich požiadaviek na cyklus procesora



Plánovanie s viacerými frontmi so spätnou väzbou

35

- Plánovač používajúci fronty so spätnou väzbou je definovaný pomocou týchto parametrov:
 - počet frontov,
 - plánovací algoritmus pre každý front,
 - metóda pre určenie momentu, kedy proces má byť presunutý do frontu s vyššou prioritou,
 - metóda pre určenie momentu, kedy proces má byť presunutý do frontu s nižšou prioritou,
 - metóda pre určenie frontu, do ktorého sa zaradí proces, ktorý potrebuje byť obslužený.
- **Najuniverzálnejší plánovací algoritmus**, ale aj **najzložitejší**.
- Môže byť prispôsobený pre rôzne systémy, potrebuje starostlivý výber parametrov, aby sa docielilo optimálne plánovanie.

Linux2.4 (pr. 0-140), FreeBSD(0-255), NetBSD(0-223), Solaris (0-169)

Animácia algoritmov plánovania

36

[http://cs.uttyler.edu/Faculty/Rainwater/COSC3355/
Animations](http://cs.uttyler.edu/Faculty/Rainwater/COSC3355/Animations)

Plánovanie viacprocesorových systémov

37

- Plánovanie viacprocesorových systémov je **NP-t'ažká** optimalizačná úloha.
- Systémy podľa typov procesorov
 - ▣ **homogénne** - funkčne *identické* - procesy môžu byť vykonávané na ľubovoľnom procesore.
 - centralizované pridelovanie, distribuované pridelovanie (synchronizácia prístupu k spoločným dátovým štruktúram), zdieľanie záťaže (*load sharing*),
 - vyrovnanie záťaže (*load balancing*)
 - ▣ **heterogénne** - procesory sú *neidentické*, t.j. proces môže byť vykonaný len na procesore, pre ktorého inštrukčný súbor bol skompilovaný. To je prípad niektorých distribuovaných systémov.

Plánovanie viacprocesorových systémov pokr.

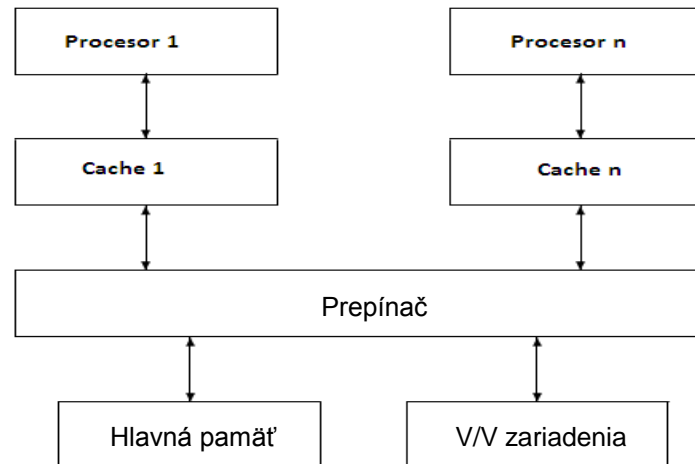
38

□ *asymetrický multiprocessing*

- jeden procesor je tzv. *master server*, vykonáva plánovanie

□ *symetrický multiprocessing*

- procesory sú rovnocenné, každý procesor si vyberá úlohu na vykonanie zo spoločného frontu pripravených procesov



Operačné systémy pre reálny čas

39

□ ***Hard real-time*** systémy

- ▣ časové závislosti sú veľmi tvrdé
- ▣ proces sa dodáva aj s termínom ukončenia
- ▣ plánovač
 - prijme proces a garantuje, že bude ukončený do stanoveného termínu,
 - odmietne ako nesplniteľný - musí vedieť, koľko času vyžaduje určitá systémová funkcia.
- ▣ Nedá sa aplikovať na systémy s virtuálnou pamäťou a sekundárnou pamäťou
 - Chýba im univerzálnosť a plná funkčnosť moderných OS

□ **Soft real-time** systémy

- ▣ Požaduje sa, aby kritické procesy dostali vyššiu prioritu ako ostatné procesy - multimediálne aplikácie alebo vysokorýchlostná interaktívna grafika
 - Môžu dosiahnuť úroveň funkčnosti univerzálneho OS
 - Starostlivý návrh plánovača -
 - systém musí používať prioritné plánovanie
 - procesy reálneho času musia mať najvyššiu prioritu a tá nesmie klesať
 - **Čas reakcie** dispečera **musí byť krátky**
 - Mnoho operačných systémov nemôže zaistiť dostatočne malý čas reakcie dispečera – I/O, syst. volanie
 - Riešenie
 - dovoliť preempciu systémových. volaní
 - navrhnuť celé jadro ako prerušiteľné! Toto je metóda použitá v systéme Solaris 2.

Výber algoritmu plánovania

41

- Metriky pre hodnotenie
 - ▣ Priepustnosť, čas čakania, čas prechodu ...
 - ▣ Začína sa so špecifikáciou kritérií pre výber plánovača
 - ▣ Napr. chceme aby nový systém
 - maximalizoval využitie procesora, pričom je stanovená max. doba odozvy
 - maximalizoval priepustnosť tak, že čas prechodu by bol proporcionálny celkovému času vykonania

Skúmanie výkonu plánovacích algoritmov

42

- Deterministické modelovanie
- Modely s frontmi
- Simulácie
- Implementácie

Deterministické modelovanie

- druh analytického modelovania – určuje výkon algoritmu pre určitú dopredu definovanú záťaž

Napr. mix úloh s požadovanými dobami obsluhy

P1	P2	P3	P4	P5
10	29	3	7	12

dáva priemernú dobu čakania pre:

FCFS: 28 ms;
nepreemptívny SJF: 13 ms ,
RR: 23 ms

- **Prednosti** - dobré základ pre porovnanie algoritmov, vyžaduje presné údaje pre výpočet rôznych metrík
- **Nedostatky** – nie je veľmi užitočný v praxi s výnimkou systémov pre RT

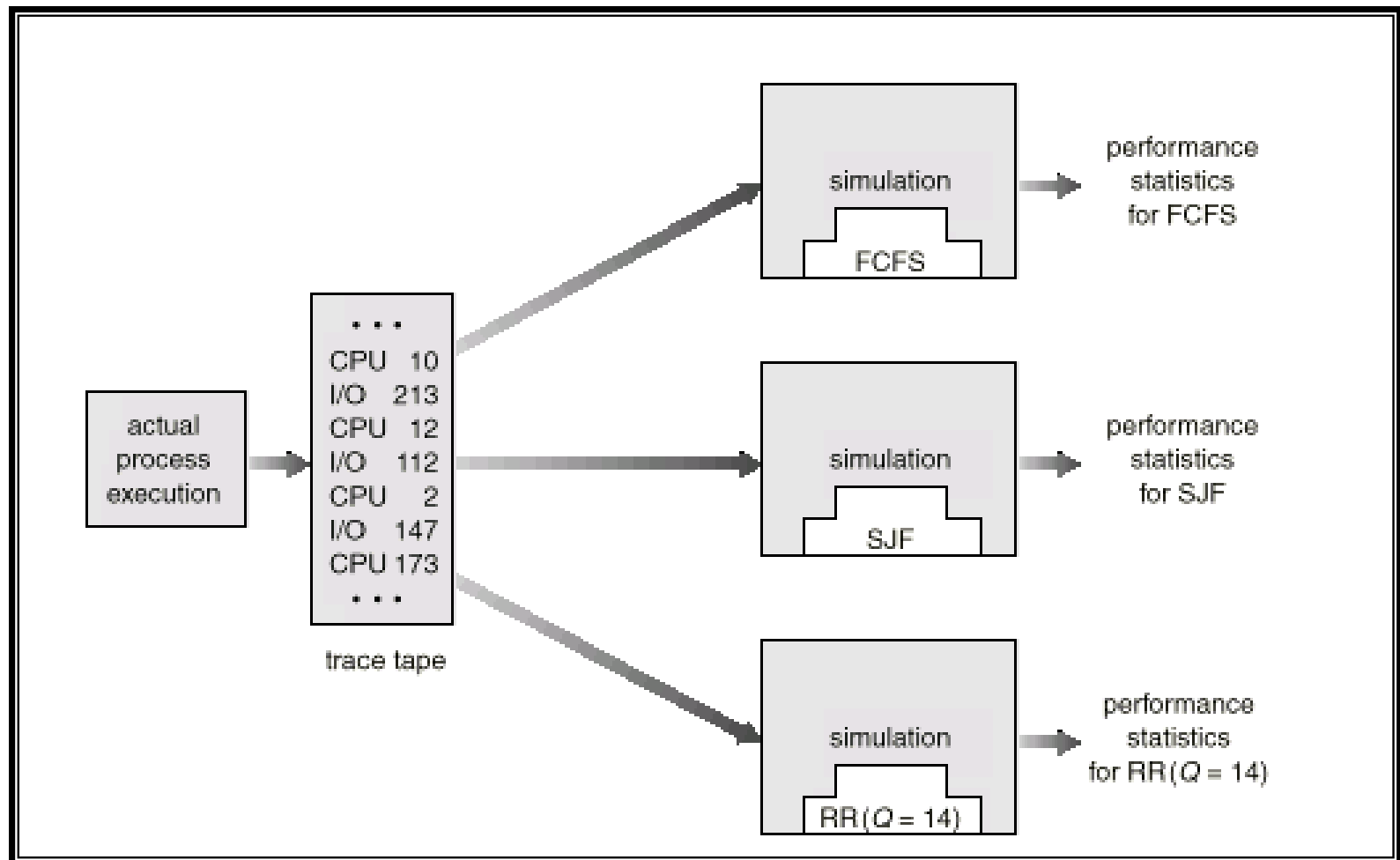
- **Modely s frontmi** - hodnotí sa distribúcia CPU a I/O časov
 - ▣ meranie alebo odhad, výsledok – vzorec udávajúci pravdepodobnosť výskytu CPU alebo I/O „zhľuku“
 - ▣ distribúcia pravdepodobnosti príchodu nového procesu
 - ▣ Charakterizuje systém ako *náhodný proces*
 - náhodné premenné majú *exponenciálnu* distribúciu
 - nás zaujímajú priemerné hodnoty
 - ▣ Teória frontov :
 - Pre front platí Little-ova formula
$$n = R.W$$
kde R je frekvencia príchodu;
 W je priemerná doba čakania;
 n je priemerná dĺžka frontu

Simulácia

- ▣ model počítačového systému
- ▣ generátor dát – náhodný, alebo definovaná distribučná f-cia (normálna, Poissonovska, exponenciálna)
- ▣ drahé, nie veľmi presné
- ▣ vedie k realistickejším predpovediam ako analýza
- ▣ drahšia

Príklad odhadu simuláciou

46



Implementácia

- Pre reálne systémy, môže dodať veľmi presne informácie o systéme
- *Nepohodlná, zdĺhavá*
 - ▣ problém
 - pozorovanie mení systém
 - môže byť potrebný aj HW monitoring