# Procesy, plánovanie a spolupráca


| | |
|---|---|
| Portál: | edu.ukf.sk - Vzdelávací portál - Univerzita Konštantína Filozofa, Nitra |
| Kurz: | Operačné systémy (KI/OS/15) |
| Kniha: | Procesy, plánovanie a spolupráca |
| Vytlačil(a): | Zuzana Pavlendová |
| Dátum: | Streda, 1 december 2021, 17:58 |

# Opis

## 2 b) Procesy

> ### 2 .3 Plánovanie procesov
>
> #### 2.3.1 Plánovanie procesora
>
> 2.3.1.1 Prepínanie programov
> 2.3.1.2 Kooperatívny multitasking
> 2.3.1.3 Nepreemptívne plánovanie
> 2.3.1.4 Preemptívne plánovanie
>
> #### 2.3.2 Stratégia plánovania
>
> 2.3.2.1 Plánovanie FCFS
> 2.3.2.2 Plánovanie Shortest-Job-First (SJF)
> 2.3.2.3 Prioritné plánovanie
> 2.3.2.4 Round Robin (RR)
>
> #### 2.3.3 Zmena kontextu - Context switch
>
> # 2.4 Spolupráca medzi procesmi
>
> #### 2.4.1 Zasielanie správ
>
> #### 2.4.2 Zdieľaná pamäť
>
> 2.4.2.1 Súbeh - race condition
> 2.4.2.2 Problém kritickej sekcie
>
> #### 2.4.3 Prostriedky pre zaistenie výlučného prístupu
>
> 2.4.3.1 Zákaz prerušenia
> 2.4.3.2 Inštrukcie Test and set lock
> 2.4.3.3 Semafory
> 2.4.3.4 Synchronizačné prostriedky odstraňujúce aktívne čakanie

**V tejto kapitole sa dozvieme:**

**· Čo je to plánovanie.**

**· Ako sú plánované procesy v operačnom systéme.**

**Kľúčové slová tejto kapitoly:**

P lánovač, kooperatívny multitasking, preemptívne a nepreemptívne plánovanie procesov, kontext, súbeh .

# Obsah

# Úvod

**V tejto kapitole sa dozvieme:**

**· Čo je to plánovanie.**

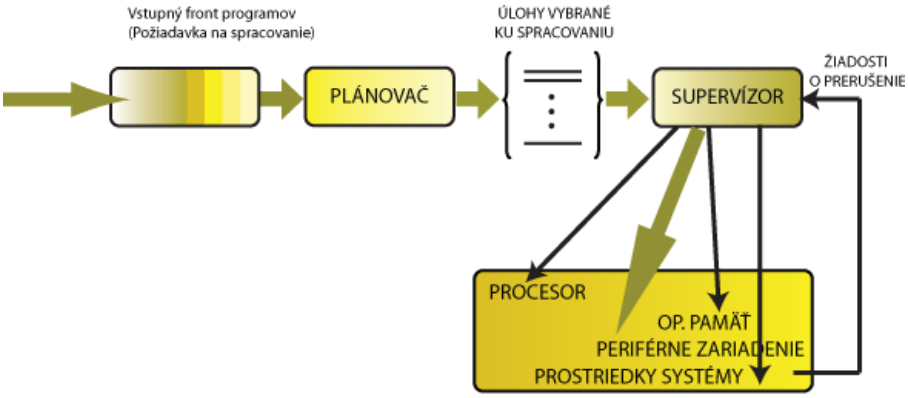
**· Ako sú plánované procesy v operačnom systéme.**

**Kľúčové slová tejto kapitoly:**

P lánovač, kooperatívny multitasking, preemptívne a nepreemptívne plánovanie procesov, kontext, súbeh .

# 2.3 Plánovanie procesov

Operačný systém koordinuje činnosť všetkých prostriedkov počítačového systému pri súčasnej práci na niekoľkých procesoch (programoch). Táto činnosť je naznačená na nasledujúcom obrázku.




Riadenie multiprogramového spracovania

Supervízor udržuje prehľad o stave všetkých systémových prostriedkov (procesora, operačnej pamäte, vnútornej pamäte, vstupných a výstupných zariadení), či sú pripojené alebo odpojené, či majú poruchu, či sú voľné alebo obsadené a riadia ich činnosť. U pamäti vedie tabuľku voľných a obsadených oblastí. Voľné zariadenie a oblasti pamätí prideľujú programom vybraným ku spracovaniu. Po dokončenom spracovaní o ňom dostáva informáciu signálom prerušenia, zaisťuje obsluhu požiadavky na prerušenie, s príp. uvoľnením systémového (-vých) prostriedku (ov) a jeho (ich) pridelení inému programu (úlohe), čakajúcemu na obsluhu. Okrem toho obsluhuje/rieši i mimoriadne stavy v systéme.

Úlohou plánovača je vyberať vstupné požiadavky (programy) k spracovaniu, na základe zistených potrebných prostriedkov pre ne a dávať podnet k zavádzaniu vybraných požiadaviek do operačnej pamäte, čím sa z nich vytvárajú úlohy pripravené k spracovaniu procesorom.

Plánovač vyberá požiadavky podľa zvolenej stratégie a zaznamenáva pre ne potrebné prostriedky podľa zadania k ich vykonaniu: programy, dáta, vyhradené oblasti operačnej i vnútorných pamätí, vstupné a výstupné zariadenia. Informácie o pridelených prostriedkoch zaznamenáva plánovač do tabuliek. Program, ktorý získa všetky potrebné prostriedky, sa stáva úlohou vybranou k spracovaniu v počítači, pod riadením supervízora.

Supervízor opúšťa úlohu tým, že zodpovedajúci program zavedie do operačnej pamäte. Ak sa uvoľní procesor, zaháji sa výpočet úlohy, ktorý prebieha, pokiaľ sa nevyskytne potreba vstupu alebo výstupu informácie. Takúto požiadavku dostane supervízor a obslúži ho tak, že zaháji činnosť potrebného prideleného vnútorného zariadenia. Úloha čaká na dokončenie vstupu/výstupu a supervízor pridelí procesor inej úlohe, ktorá čaká na výpočet. Ukončenie vstupnej/výstupnej operácie sa oznamuje supervízorovi, prevedie úlohu do stavu čakania na výpočet v procesore. Keď skončí spracovanie nejakej úlohy, supervízor žiada plánovač o prípravu ďalšieho požiadavku k spracovaniu. Plánovacie algoritmy sa vyberajú vždy s istým cieľom. V daných podmienkach majú zaisťovať optimálne využitie prostriedkov systému, minimálnu dobu odpovede a pod. Preto je v rade prípadov nutné, okrem pridelenia prostriedkov a výbere ú1ohy k spracovaniu, tiež stanoviť časový interval vymedzený pre spracovanie.

Všimneme si najjednoduchšie plánovacie algoritmy a s nimi súvisiace charakteristické veličiny a vlastnosti procesov, na základe ktorých plánovač rozhoduje, ktorému procesu pridelí procesor. Charakteristické veličiny môžu byť:

$t_{oi}$ - okamih príchodu (vytvorenie) i-tého procesu,

$t_i$ - doba potrebná k prevedeniu i-tého procesu,

$t_{di}$ -doba zostávajúca na dokončenie i-tého procesu,

$P_i$ - statická priorita i-tého procesu (najnižšia hodnota najvyššia prednosť),

$t_{bi}$ - doba doterajšieho behu programu ($t_{bi} <= t_i$),

$\Delta t_i$ - pridelený časový interval.

Plánovacie algoritmy, ktoré využívajú vyššie uvedené charakteristické veličiny, sú zhrnuté v tabuľke:

| Typ | Využíva charakteristickú veličinu | Princíp |
|---|---|---|
| FCFS, tiež FIFO (first come - first served / First in - first out) | $t_{oi}$ | riadny frontový režim |
| SXFS (shortest execution - first served) | $t_i$ | proces s najkratšou dobou vykonávania, je prvý obslúžený |

| | | |
|---|---|---|
| LCFS (least completed - first served) | $t_{bi}$ | prednostne sa obsluhuje proces, ktorý zatiaľ bežal najkratšiu dobu |
| EDFS (earliest - due-time fisrst served) | $t_{di}$ | prednostne sa obsluhuje proces, ktorému zostáva najmenej času na dokončenie, tj. do okamihu, kedy musí byť dokončený |
| HSFS (highest static priority first served) | $P_i$ | prednostne sa obsluhuje proces s najvyššou statickou prioritou |
| RR (round-robin) | $\Delta t_i$ | cyklická obsluha procesov po časových intervaloch |

Rozoznávame tri druhy plánovania procesov (process scheduling):

- krátkodobé (short-term), CPU scheduling (plánovanie procesora): výber, ktorému z pripravených procesov bude pridelený procesor, vo všetkých viacúlohových systémoch,
- strednodobé (medium-term): výber, ktorý proces blokovaný alebo pripravený bude odsunutý z vnútornej pamäti na disk, ak je vnútornej pamäte nedostatok (swap out, roll out),
- dlhodobé (long-term), job scheduling (plánovanie prácou, úloh): výber, ktorá úloha bude spustená (má význam hlavne pri dávkovacom spracovaní). Účelom je namixovať úlohy tak, aby bol počítač čo najviac vyťažený (triedy úloh podľa náročnosti).

V jednotlivých OS sa nemusia nutne používať všetky tri druhy plánovania procesov. V niektorých prípadoch je napríklad plánovanie úloh zjednodušené na pravidlo: pokiaľ je dostatok zdrojov OS, spustí proces.

# 2.3.1 Plánovanie procesora

Predpokladajme, že výpočtový systém je vybavený jedným procesorom. Nie je teda technicky možné, aby na jednom procesore mohlo naraz bežať niekoľko programov. Operačný systém musí súčasný beh programov simulovať a tak plánovať prideľovanie jedného procesora niekoľkým procesom. Plánovanie procesora sa používa vo viacúlohových systémoch.
Môže nastať v nasledujúcich situáciách:

- pokiaľ niektorý proces prejde zo stavu bežiaci do stavu blokovaný (čakanie na I/O operáciu, semafor, čakanie na uplynutie zadaného časového intervalu, čakanie na ukončenie procesu-potomka),
- pokiaľ niektorý proces skončí,
  • pokiaľ je niektorý proces prevedený zo stavu bežiaci do stavu pripravený,
- pokiaľ niektorý proces prejde zo stavu čakajúci do stavu pripravený.

Ak dochádza k plánovaniu procesora iba v prvých dvoch vyššie uvedených prípadoch, hovoríme, že OS používa nepreemptívne plánovanie CPU (procesov). Inak hovoríme, že OS používa preemptívne plánovanie CPU. Preplánovanie procesoru v poslednom uvedenom prípade sa používa zriedka (napríklad v systéme reálneho času).

#### 2.3.1.1 Prepínanie programov

Multitaskový operačný systém umožňuje súčasný beh niekoľkých programov. Prepínanie programov (task switching) je predchodcom kooperatívneho multitaskingu a môže byť realizovaný dvoma spôsobmi:
S obmedzeným prepínaním programov je možné prepínať len medzi jedným „normálnym“ programom - hovorí sa mu hlavný program – a niekoľkými špeciálnymi programami vytvorenými zvláštnym spôsobom, výhradne pre prepínanie.
S neobmedzeným prepínaním, umožňuje spúšťanie niekoľkých „normálnych“ programov a prepínanie medzi nimi.

#### 2.3.1.2 Kooperatívny multitasking

Proces, ktorý práve beží, musí pravidelne volať systémovú službu, ktorou dáva najavo, že môže byť prerušený (táto služba býva kombinovaná s inými službami systému). Pokiaľ si používateľ nevyžiada prepnutie na iný proces, nerobí táto služba nič (alebo iba vykoná systémovú službu, s ktorou je kombinovaná). Ak si vyžiada používateľ prepnutie procesov, zaistí zmienená služba prepnutie kontextu pri najbližšom vyvolaní. Po jeho ukončení teda už beží nový proces a dosiaľ aktívny proces čaká, až bude znovu aktivovaný.

Kooperatívny multitaskový operačný systém využíva prepínanie medzi procesom na popredí (foreground) a procesom na pozadí (background). Výhodou je lepšie využitie procesoru, tzn. dobu, kedy procesor čaká, vyplní spracovaním iného procesu. Nevýhodou je spomalenie procesu na popredí, a preto je nepoužiteľný na realizáciu paralelných úloh (správa siete, komunikácie prostredníctvom sériového rozhrania apod.). Chyba v aktívnom procese vedie totiž k nekonečnému oblúku.

#### 2.3.1.3 Nepreemptívne plánovanie

V prípade nepreemptívneho plánovania sa proces musí procesoru sám vzdať. Pokiaľ má byť doba, po ktorú je proces v stave bežiaci obmedzený, je nutné, aby proces kontroloval časovač a po prekročení stanovenej doby sa dobrovoľne vzdal procesoru vyvolaním služby OS, ktorá je pre tento účel určená. Výhodou je, že proces nemôže byť prerušený, pokiaľ nechce (napr. v kritickej sekcii viď ďalej). Nevýhodou je, že zle chovajúci sa proces, môže zablokovať celý OS. Takto fungujú napríklad MS Windows.

#### 2.3.1.4 Preemptívne plánovanie

V prípade preemptívneho plánovania môže OS odobrať procesu procesor. Spravidla sa tak deje pri uplynutí časového kvanta určeného pre beh procesu a celá akcia je vyvolaná prerušením od časovača. Príkladom OS, ktorý používa preemptívne plánovanie je OS Unix.

# 2.3.2 Stratégia plánovania

Stratégia použitá pre výber, ktorému z pripravených procesov bude pridelený procesor, býva tvorcovi operačného systému vyberaná podľa týchto kritérií:

- spravodlivosť: každý proces dostane spravodlivý diel času procesora,
- efektivita: udržovať maximálne vyťaženie procesoru, príp. inej časti systému,
- čas odozvy: minimalizovať dobu odozvy pre interaktívnych používateľov,
- doba obrátky: minimalizovať dobu spracovania každej dávkovej úlohy,
- priechodnosť: maximalizovať množstvo úloh spracovaných za jednotku času.

Podľa toho, ktoré z týchto vlastností brali tvorcovia systému do úvahy a akú váhu im prikladali, používajú rôzne operačné systémy rôzne stratégie plánovania procesoru.

#### 2.3.2.1 Plánovanie FCFS

FCFS (first come, first served - kto skôr príde, ten je skôr obslúžený: procesy prichádzajúce do stavu pripravený, sú umiesťované na koniec frontu typu FIFO (first in first out). Pri plánovaní procesora sa procesor pridelí tomu procesu, ktorý je vo fronte prvý. Túto stratégiu je možné používať pri preemptívnom i nepreemptívnom plánovaní procesora. Pri preemptívnom plánovaní sa niekedy nazýva round robin scheduling.

**Príklad 1:**

Proces A=24

Proces B=3

Proces C=3

Procesy vznikli v poradí A,B,C

Doby čakania – A=0, B=24, C=27

Priemerná doba čakania (0+24+27)/3=17

**Príklad 2:**

Proces A=24

Proces B=3

Proces C=3

Procesy vznikli v poradí B,C,A

Doby čakania – B=0, C=3, A=6

Priemerná doba čakania (0+3+6)/3=3

**Závery k plánovaniu FCFS**

Príklad 2 vykazuje lepší výsledok ako Príklad 1, i keď sa jedná o rovnaké procesy. Krátke procesy nasledujúce po dlhom procese ovplyvňuje „konvojový efekt".

## 2.3.2.2 Plánovanie Shortest-Job-First (SJF)

S každým procesom sa spojí dĺžka jeho nasledujúcej CPU dávky. Vyberá sa proces s najkratšou dobou dávky CPU pričom docielime dve varianty:

- **Nepreemptívnu, bez predbiehania** – ako náhle sa CPU odovzdá vybranému procesu, tento nemôže byť predbehnutý žiadnym iným procesom, dokiaľ dávku CPU nedokončí.

- **Preemptívnu, s predbiehaním** – ako náhle sa vo fronte pripravených objaví proces s dĺžkou dávky CPU kratší než je doba zostávajúca k dokončeniu dávky práve bežiaceho procesu, práve bežiaci proces je vo využívaní CPU predbehnutý novým procesom (Shortest-Remaining-Time-First – SRTF). SJF je optimálny algoritmus, pre danú množinu procesov dáva minimálnu priemernú dobu čakania.

## 2.3.2.3 Prioritné plánovanie

S každým procesom je spojené prioritné číslo, pričom prioritné číslo vyjadruje preferencie procesu pre výber nabudúce bežiaceho procesu. CPU sa prideľuje procesu s najvyššou prioritou a tejto najvyššej priorite zodpovedá najnižšie prioritné číslo. Opäť môžeme pozorovať dve varianty:

- nepreemptívnu, bez predbiehania,

- preemptívnu, s predbiehaním.

SJR je prioritné plánovanie, kde prioritou je predpovedaná dĺžka nasledujúcej CPU dávky.

Problém: starnutie – procesy s nižšou prioritou sa nemusia nikdy vykonať.

Riešenie: zrenie – priorita sa postupom času zvyšuje.

## 2.3.2.4 Round Robin (RR)

RR - Round Robin Scheduling – je preemptívnym plánovaním.

Každý proces dostáva CPU na malú jednotku času (časové kvantum) desiatky až stovky ms.

Po uplynutí tejto doby je bežiaci proces predbehnutý najstarším procesom vo fronte pripravených procesov a zaraďuje sa na koniec tejto fronty. Či je vo fronte pripravených *n* procesov a časové kvantum je q, potom každý proces získava 1/n-tinu doby CPU, naraz nanajvýš po q časových jednotkách. Žiadny proces nečaká na pridelenie CPU viac než (n-1)q časových jednotiek.

Výkonnostné hodnotenie záleží v prvom rade na veľkosti prideľovaného časového kvanta q:

• ak q je veľké, potom je plánovanie podobné typu FCFS,

• ak q je malé, potom môže byť neefektívne – q musí byť dostatočne veľké s ohľadom na réžiu prepínania kontextu.

Určitou modifikáciou metódy RR je začlenenie viacerých front v spätnej väzbe, ako je naznačené na nasledujúcom obrázku.

# 2.3.3 Zmena kontextu - Context switch

U operačných systémov s preemptívnym plánovaním procesora, môže byť proces prerušený medzi ľubovoľnými dvoma strojovými inštrukciami v programe a riadenie môže byť predané inému procesu. Programy sú však písané tak, že nepredpokladajú, že by došlo ku zmene obsahu registrov procesora, prípadne niektorých ďalších oblastí medzi dvoma inštrukciami. Pri prepnutí na iný proces, musia byť zmenené tiež ďalšie registre (ukazovateľ na tabuľku stránok, kľúč pre ochranu pamäte, register udávajúci, či je procesor v privilegovanom stave apod.).

Preto sa pri prepínaní medzi procesmi vykonáva tzv. uloženie kontextu (context save) pôvodne bežiaceho procesu a obnovenie kontextu (context restore) procesu, ktorému sa prideľuje procesor. Pod pojmom context, je myslený stav procesora (obsah registrov), stav prípadného koprocesora, prípadne i stav ďalších zariadení. Tento context sa ukladá buď na zásobník procesu alebo do dopredu pripravenej oblasti dát v adresnom priestore procesu.

# 2.4 Spolupráca medzi procesmi

Dva používané mechanizmy:

• zasielanie správ,

• zdieľaná pamäť.

Niektoré operačné systémy podporujú obidva mechanizmy.

Zdieľaná pamäť: jednoduchšie programovanie, mocnejší - programátor má viac prostriedkov, spravidla i jednoduchšia implementácia.

Zasielanie správ: flexibilnejší - je možné použiť i pre komunikáciu medzi procesmi bežiacimi na rôznych procesoroch alebo počítačoch.

# 2.4.1 Zasielanie správ

Zaslanie správy môže byť realizované predaním ukazovateľa na správu v zdieľanej pamäti, umiestnením správy do vyrovnávacej pamäte alebo do fronty správ buď v zdieľanej pamäti alebo v pamäti jadra systému, pričom systém potom zaisťuje kopírovanie správ do pamäťového priestoru príjemcu správy. Možná je i komunikácia prostredníctvom siete. Implementácia komunikačného spojenia pre zasielanie správ sa v rôznych systémoch môže líšiť v nasledujúcich detailoch:

• spôsobom vytvárania spojenia,

• počtom procesov využívajúcich spojení: 2 alebo viac,

• počtom spojení medzi dvoma procesmi: 1 alebo viac,

• kapacitou spojení t.j. koľko nespracovaných správ môže spojenie obsahovať:

*nulová kapacita*: buď musí odosielateľ čakať na príjemcu (rendezvous), alebo sa správa, na ktorú nikto nečaká, stratí (aby k tomu nedošlo, je nutné použiť nejaký synchronizačný prostriedok); - obmedzená kapacita: pokiaľ je linka zaplnená, musí odosielateľ čakať,

*neobmedzená kapacita*: odosielateľ nikdy nečaká,

• veľkosťou správ: pevná alebo premenná,

• jednosmernosťou alebo obojsmernosťou spojenia

• priamosťou (správa sa zasiela konkrétnemu procesu, je prijímaná od konkrétneho procesu) alebo nepriamosťou (správa sa zasiela a prijíma prostredníctvom poštovej schránky) komunikácie,

• symetrickosťou alebo nesymetrickosťou komunikácie,

• asynchrónnej alebo synchrónnej komunikácii: odosielateľ musí pri synchrónnej komunikácii čakať na odpoveď (použité napr. pre RPC - remote procedure call),

• automatické alebo explicitné ("ručné") použitie vyrovnávacích pamätí,

• zasielanie kópie alebo referencie.

Ošetrenie chýb pri zasielaní správ musí zahŕňať tieto situácie:

• jeden z partnerských procesov skončil,

• strata správy,

• duplicita správy,

• skomolenie správy.

Programové prostriedky predávania správ:

• SEND zašle správu. Nikdy sa nezablokuje. Pokiaľ na správu čaká príjemca operácií RECEIVE, je mu správa doručená a príjemca odblokovaný.

• RECEIVE správu prijíma. Pokiaľ žiadna správa nie je dostupná, prijímajúci proces je zablokovaný.

Je potrebné vyriešiť problém adresácie, t.j. určenie odosielateľa a príjemcu. Dá sa vyriešiť napr. identifikáciou procesu na danom počítači. Iným riešením je zavedenie schránok (mailboxov) a adresácie pomocou identifikácie schránky. Schránka je vyrovnávacia pamäť typicky pevnej diaľky. Do nej sú správy ukladané operáciou SEND a z nej vyberané operáciou RECEIVE. Schránka modifikuje operáciu SEND: pokiaľ je schránka plná, zablokuje sa i SEND.

Zaujímavým prípadom je situácia, kedy je schránka nulovej veľkosti. Pokiaľ je odosielateľom prevedený SEND a ešte tam nie je žiadny príjemca čakajúci operáciou RECEIVE, musí odosielateľ počkať. Podobne príjemca čaká na odosielateľa, kým niečo zašle. Po predaní správy, tj. je prevedená operácia SEND i RECEIVE na jednu schránku, sa oba procesy opäť rozbehnú. Tento prípad sa nazýva dostaveníčko (randezvous).

# 2.4.2 Zdielaná pamäť

Zdieľaná pamäť je pamäť, do ktorej má prístup viacero procesov. Môže byť použitá pre komunikáciu medzi procesmi.

## 2.4.2.1 Súbeh - race condition

Súbeh (race condition) je situácia, kedy pri prístupe dvoch alebo viacerých procesov k zdieľaným dátam dôjde ku chybe, pretože každý z procesov samostatne, sa chová korektne. Ku chybe dochádza vďaka tomu, že dáta sú modifikované niektorým procesom v dobe, kedy s nimi iný proces vykonáva niekoľko operácií, o ktorých sa predpokladalo, že budú vykonané ako jeden nedeliteľný celok.

Dátam, ktoré sú zdieľané niekoľkými procesmi tak, že pri prístupe k nim by mohlo dôjsť k súbehu, sa hovorí kritická oblasť. Kritická sekcia je najmenšia časť programu, v ktorej sa pracuje s dátami v nejakej kritickej oblasti, a ktorá musí byť vykonaná ako jeden celok. Hlavne u zložitejších dátových štruktúr (obojsmerné spojové zoznamy, zložité dynamické štruktúry uložené v súboroch a pod.) dochádza často k tomu, že v určitom štádiu spracovania sú dáta dočasne nekonzistentné. Pokiaľ v tom okamihu dôjde k prepnutiu kontextu na proces, ktorý tieto dáta tiež používa, môže nastať súbeh.

**Príklady súbehu:**

1) Pri "súčasne" vykonanom vklade a výbere peňazí v banke realizovanom na viacúlohovom počítači môže dôjsť vplyvom súbehu k strate vkladu:

1. proces (výber) 2. proces (vklad)

pom:=konto;

pom:=pom-10000;

-> (context switch) ->

pom:=konto;

pom:=pom+20000;

konto:=pom;

<- (context switch) <-

konto:=pom;

2) Dva procesy sa snažia vytvoriť súbor. Pokiaľ si zvolí rovnaké meno súboru, môže dôjsť k tomu, že prvý proces zistí, že súbor tohto mena neexistuje. Potom je prerušený druhým procesom, ktorý tiež zistí, že súbor neexistuje, vytvorí ho a zapíše doň nejaké dáta. Prvý proces potom prevedie operáciu vytvorenia súboru, čím dáta zapísané do súboru prvým procesom zmaže. Pri použití jednoduchých dátových štruktúr sa súbehu dá zabrániť takým naprogramovaním, že dáta sú po dokončení každej strojovej inštrukcie konzistentné:

1. proces (výber) 2. proces (vklad)

konto:=konto-10000;

-> (context switch) ->

konto:=konto+20000;

<- (context switch) <-

V prípade vytvárania súborov môže operačný systém poskytovať službu, ktorá vytvorí súbor. Pokiaľ však súbor daného mena existuje, pokus o jeho vytvorenie skončí chybou. Ak je prevedenie tejto služby neprerušiteľné, k súbehu nemôže dôjsť.

U zložitejších dátových štruktúr (napríklad obojsmerný spojový zoznam), však nie je možné dodržať podmienku, že každá modifikácia dát sa vykonáva jedinou strojovou inštrukciou alebo jediným neprerušiteľným volaním systému.

## 2.4.2.2 Problém kritickej sekcie

Problémom kritickej sekcie je umožniť prístup ku kritickej oblasti procesom, ktoré sa o to usilujú, pri dodržaní nasledujúcich podmienok:

• výhradný prístup; v každom okamihu môže byť v kritickej sekcii najviac jeden proces,

• vývoj; rozhodovanie o tom, ktorý proces vstúpi do kritickej sekcie, ovplyvňujú iba procesy, ktoré sa o vstup do kritickej sekcie usilujú; toto rozhodnutie pre žiadny proces nemôže byť odkladané do nekonečna; nedodržanie tejto podmienky môže viesť napríklad k tomu, že je umožnená iba striktná alternácia (dva procesy sa pri priechode kritickou sekciou musí pravidelne striedať),

• obmedzené čakanie; pokiaľ sa jeden proces usiluje o vstup do kritickej sekcie, nemôžu ostatné procesy tomuto vstupu zabrániť tým, že sa v kritickej sekcii neustále striedajú - môžu do tejto kritickej sekcie vstúpiť iba obmedzený počet krát (spravidla iba raz).

Pokiaľ o prístup do kritickej sekcie usiluje niektorý proces v dobe kedy je v nej iný proces, prípadne sa o prístup usiluje v jednom okamihu viac procesov, je nutné niektoré z nich pozdržať. Toto pozdržanie je možné realizovať oblúkom. Toto tzv. aktívne čakanie (busy waiting) však zbytočne spotrebúva čas CPU - je možné čakajúci proces zablokovať a obnoviť jeho beh až v okamihu, kedy proces, ktorý je v kritickej sekcii, túto sekciu opustí.

# 2.4.3 Prostriedky pre zaistenie výlučného prístupu

K zaisteniu vzájomného vylúčenia postačí dosadenie toho, že žiadne dva procesy nie sú v rovnakom časovom okamihu v kritickej sekcii. Táto podmienka je postačujúca k dosiahnutiu vzájomného vylúčenia, nehovorí však nič o efektivite a korektnosti. K dosiahnutiu dobrých výsledkov je potrebné dodržať nasledujúce štyri pravidlá:

• Žiadne dva procesy nie sú súčasne v kritickej sekcii.

• Nemôžu byť urobené žiadne predpoklady o rýchlostiach alebo počte CPU.

• Žiadny proces bežiaci mimo svoju kritickú sekciu nesmie blokovať iný proces.

• Žiadny proces nesmie čakať nekonečne dlho v kritickej sekcii.

## 2.4.3.1 Zákaz prerušenia

Zákaz prerušenia znemožní prepnutie kontextu. Nebezpečné - proces môže zakázať prerušenie a zhavarovať alebo sa dostať do nekonečného cyklu -> celý systém je mŕtvy -> u väčšiny systémov môže prerušenie zakázať iba jadro OS, čo je zaistené tak, že zákaz prerušenia je privilegovaná inštrukcia. OS spravidla vnútorne používa zákaz prerušenia, aby zaistil nedeliteľnosť postupností inštrukcií, používaných pri implementácii iných synchronizačných konštrukcií.

## 2.4.3.2 Inštrukcie Test and set lock

Inštrukcie typu "test and set lock" (TSL). Inštrukcie nastavia premennú Lock (zámok) typu boolean na true (uzamknuté) a vrátia pôvodnú hodnotu tejto premennej. Celá akcia musí byť nedeliteľná (neprerušiteľná). Na počítačoch, ktoré inštrukciu TSL nemajú, je možné ju implementovať s použitím zákazu prerušenia:

*function TestAndSet(var Lock: boolean): boolean;*

*begin*

*DisableInterrupts;*

*TestAndSet:=Lock;*

*Lock:=true;*

*EnableInterrupts;*

*end;*

Inštrukcia sa používa pred vstupom do kritickej sekcie. Po výstupe z kritickej sekcie sa premenná Lock bežným spôsobom nastaví na false (odomknuté):

*while TestAndSet(Lock) do { nothing } ;*

... kritická sekcia ...

*Lock:=false;*

Niektoré počítače, ktoré inštrukciu TSL nemajú, môžu namiesto nej použiť inštrukciu, ktorá prehodí obsah registra s obsahom premennej v pamäti (SWAP nebo XCHG):

*function Swap(var a, b: boolean);*

*var*

*Temp: boolean;*

*begin*

*DisableInterrupts;*

*Temp:=a;*

*a:=b;*

*b:=Temp;*

*EnableInterrupts;*

*end;*

Použitie tejto inštrukcie vyzerá takto:

*repeat*

*x:=true;*

*Swap(Lock, x);*

*until x=false;*

... kritická sekcia ...

*Lock:=false;*

Inštrukcia TSL (prípadne SWAP a XCHG) sa môže použiť pre zaistenie výhradného prístupu. Samy o sebe však nespĺňajú podmienku obmedzeného čakania a neodstraňujú aktívne čakanie, môžu byť však základom konštrukcií, ktoré tieto nedostatky nemajú.

## 2.4.3.3 Semafory

Inštrukcie "test and set lock" môžeme zobecniť takým spôsobom, že dvojstavovú premennú typu boolean, nahradíme čítačom (premenná typu integer). Toto zobecnenie popísal E. W. Dijkstra v roku 1965. Operácie boli pôvodne nazvané P a V, podľa dánskych slov proberen (testovať) a verhogen (zväčšiť). Niektoré pramene tieto operácie nazývajú down a up.

Existujú dve sémantiky vzhľadom k hodnotám čítača:

**1. čítač >= 0:**

Operácia DOWN skontroluje hodnotu semaforu. Ak je väčší než 0, zníži hodnotu semaforu o 1 a operácia skončí. Aj je rovná 0, operácia DOWN sa zablokuje a príslušný proces je pridaný do fronty čakajúcich procesov na danom semafore.

Operácia UP zistí, či je fronta čakajúcich procesov na danom semafore plná. Pokiaľ áno, vyberie jeden z čakajúcich procesov (napr. najdlhšie čakajúci) a ten odblokuje (t.j. pokračuje za svojou operáciou DOWN). Pokiaľ je fronta prázdna, zväčší čítač semaforu o 1.

**2. Čítač ľubovolný (záporná hodnota je počet zablokovaných procesov):**

Operácia DOWN zníži hodnotu semaforu o 1. Ak je väčší alebo rovný 0, operácia skončí. Ak je menší než 0, operácia DOWN sa zablokuje a príslušný proces je pridaný do fronty čakajúcich procesov na danom semafore.

Operácia UP zväčší čítač semaforu o 1. Pokiaľ je hodnota menšia alebo rovná 0, skontroluje, či je fronta čakajúcich procesov na danom semafore plná. Pokiaľ áno, vyberie jeden z čakajúcich procesov a ten odblokuje.

Implementácia v jadre systému (s použitím zákazu prerušenia) v prípade použitia aktívneho čakania vyzerá takto:

*procedure Down(var S: semaphore);*

*begin*

*DisableInterrupts;*

*while S<=0 do begin*

*EnableInterrupts;*

*DisableInterrupts;*

*end;*

*S:=S-1;*

*EnableInterrupts;*

*end;*

*procedure Up(var S:semaphore);*

*begin*

*DisableInterrupts;*

*S:=S+1;*

*EnableInterrupts;*

*end;*

Semafory sa používajú podobne ako inštrukcie "test nad set" - pred vstupom do kritickej sekcie sa vyvolá Down, po výstupe Up. Semafory popísané vyššie tiež nespĺňajú podmienku obmedzeného čakania a neodstraňujú aktívne čakanie.

## 2.4.3.4 Synchronizačné prostriedky odstraňujúce aktívne čakanie

Odstrániť aktívne čakanie je možné tým spôsobom, že pokiaľ proces nemôže vstúpiť do kritickej sekcie, je mu odobraný procesor a beh procesu je dočasne blokovaný. Aby pri uvoľnení kritickej sekcie bolo známe, či a ktorý proces odblokovať, priradí sa ku každému semaforu alebo premennej, ktorá funguje ako zámok používaný inštrukciou "test and set lock", front čakajúcich procesov:

*type Semaphore = record*

*value: integer;*

*queue: list of process;*

*end;*

Predpokladajme, že sú k dispozícii operácie Enqueue pre zaradenie objektu do fronty a Dequeue pre vyradenie objektu z fronty. Operácia Sleep zaistí, že aktuálny proces bude zablokovaný (tj. prevedený do stavu blocked); operácia Wakeup zaistí odblokovanie procesu (prevedenie do stavu ready).

*procedure Wait(var S: Semaphore);*

*begin*
*DisableInterrupts;*
*S.value:=S.value-1;*

*if S.value<0 do begin*

*Enqueue(S.queue, CurrentTask);*

*Sleep;*

*end;*

*EnableInterrupts;*

*end;*

*procedure Signal(var S:semaphore);*

*begin DisableInterrupts;*

*S.value:=S.value+1;*

*P:=Dequeue(S.queue);*

*if P<>nil Wakeup(P);*

*EnableInterrupts;*

*end;*