4. Procesy a paralelismus, koordinace běhu procesů, synchronizace procesů a synchronizace procesů pomocí komunikace mezi nimi.

Procesy

Proces

- Proces je instance provádění programu
- Proces je identifikovatelný
- Proces vlastní nějaké prostředky (např. otevřené soubory, prostor v paměti)
- Stav procesu je v každém jeho okamžiku definovatelný pomocí:
    - obsahu registrů
    - zásobníku
    - datové sekce
    - programu, který jej řídí

Z hľadiska jadra je proces považovaný za entitu ktorá práve disponuje pridelenými systémovými prostriedkami. Teda nebude to len vykonávaný kód programu ale bude zahŕňať aj kontext v ktorom sa vykonáva. Do tohto kontextu budú patriť hodnoty čítača inštrukcií a registrov procesora. Tak isto proces zahŕňa aj zásobník, ktorý obsahuje dočasne dáta procesu (ako sú parametre podprogramov, návratové adresy a lokálne premenné) a dátový segment, ktorý obsahuje globálne premenné.

Hlavný rozdiel medzi programom a procesom spočíva v tom, že program je pasívna jednotka- – obsah súboru uložený na disku, zatiaľ čo proces je aktívna jednotka, v ktorej čítač inštrukcií určuje, ktorá inštrukcia sa bude vykonávať.
Nad jedným programom sa môže vykonávať viac procesov, ale každý program má svoju sekvenciu vykonávania. Na druhej strane je možné, že jeden proces počas svojho behu vytvorí viacej procesov.

Význam procesu v OS

- OS maximalizuje využití procesoru prokládáním běhu procesů (minimalizuje tak dobu odpovědi)
- OS spravuje zdroje přidělováním procesům podle zvolené politiky (priorita, vzájemná výlučnost, zabraňuje uváznutí)
- OS podporuje tvorbu procesu a jejich komunikaci
- OS s multiprogramováním multiprogramování = multitasking

Data nutná pro spravování procesu

OS si udržuje tabulku se seznamem procesů, s množstvím různých údajů

- Stavové informace (registry procesoru, čítač instrukcí)

- Informace nutné pro správu a řízení procesu (priorita procesu, stav procesu, inf. o používání zdroj, PID - proces identifier, inf. o používání paměti)

Stavy procesu

- **Nový** – právě vytvořený proces
- **Připravený** – čeká na přidělení procesoru
- **Běžící** – program se interpretuje procesorem
- **Čekající** – např. na I/O operaci
- **Ukončený** – ukončil své provádění

Procesy a vlákna

Proces je jednotka provádění programu, charakterizovaná přiděleným kontextem a paměťovým prostorem.

Vlákno je systémový objekt viditelný uvnitř procesu, typicky se vlákna přidělují procesorům. Vlákno může být buď **běžící** nebo **připravené** a může přistupovat ke zdrojům vlastněným procesem.

Pokud vlákno přistoupí k nějakému zdroji, např. si otevře soubor, vidí jej všechna ostatní vlákna daného procesu.

Vlákna si může spravovat bud aplikace sama nebo se o ně může starat kernel.

Kritéria pro hodnocení plánování

- Využití CPU (max)
- Propustnost (max) – počet procesů, které dokončí svůj běh za jednotku času
- Doba obrátky (min) – doba provedení konkrétního procesu
- Doba čekání (min) – doba, po kterou proces čekal ve frontě „připravený“ (ready)
- Doba odpovědi (min) – doba, která uplyne od zadání požadavku do doby první reakce

Hlavním úkolem plánovací strategie je zamezit stárnutí procesů (*starvation*) a zajistit spravedlivé rozdělení času procesoru

Plánovací algoritmy

plánovanie s preempciou - dovoľuje dočasné pozastavenie spracovania procesu a pridelenia procesora inému procesu.
plánovanie bez preempcie - nedovoľuje odobratie procesora spracovaványému procesu až do jeho dokončenia

FCFS - First Come First Served – použiteľný pri dávkovom spracovaní.

SJF (Shortest-Job-First Scheduling) - s kazdym procesem je asociovan cas jeho nasledujiciho CPU cyklu. Je-li CPU volny, je pridelen procesu s nejkratsim nasledujicim CPU cyklem. (nonpreemptivní, preemtivní)

Prioritní plánování (nonpreemptivní, preemtivní) spracovanie z radu pripravených procesov podľa určitého kritéria . Priorita môže byť statická, dynamická, môže byť stanovená vonkajším zadaním, alebo môže byť vypočítaná vnútorne v závislosti od priebehu spracovania a od zaťaženia vypočetného systému

RR (Round Robin) – podobny algoritmu FCFS, obsahuje vsak navic moznost preemptivniho planovani. V systemu je definovana mala casova jednotka - *casove kvantum* vetsinou 10 - 100 ms. Fronta pripravenych je definovana jako cyklicka fronta a planovac CPU touto frontou prochazi stale dokola a prideluje CPU jednotlivym procesum vzdy na jedno casove kvantum.

Paralelné procesy

Pre paralelné procesy musí obecne platiť podmienka, že prvá operácia jedného procesu musí byť zahájená skôr než je ukončená posledná operácia druhého procesu. Z hľadiska synchronizácie paralelných procesov teda nie je dôležité, či sa skutočne prekrývajú v čase, alebo len striedavo využívajú ten istý procesor.

Paralelné procesy delíme na:
Procesy súperiace - sa snažia získať výlučný prístup k zdieľanému prostriedku, pre tieto procesy sa rieši úloha vzájomného vylučovania.
Príklad: Proces A aj proces B zapisujú do rovnakej oblasti v pamäti. Zdieľajú spoločnú premennú *Zapisany*. Ak by tieto procesy boli prerušiteľné a proces A bol prerušený po vykonaní príkazu Zapis (BlokA, InformaciaA), nezmenila by sa hodnota premennej *Zapisany* a proces B by prepísal údaje zapísané procesom A. Pri takýchto úlohách musí systém zabezpečiť vzájomné vylučovanie t.j. nedovolí prerušiť proces využívajúci zdieľanú oblasť.

procesy spolupracujúce - tiež využívajú spoločné prostriedky na odovzdávanie správ, pre spolupracujúce procesy sa rieši úloha vzájomného vylučovania i úloha synchronizácie pre zabezpečenie správneho poradia vykonania oboch procesov - klasická úloha: producent, konzument.
Príklad: Proces A zapisuje údaje do pamäti a proces B ich číta a tlačí na tlačiareň. V tomto prípade treba zabezpečiť, aby procesy neboli počas vykonávania prerušené a navyše, aby proces B, vždy nasledoval za pracesom A. Tu musí systém okrem vzájomného vylučovania synchronizáciu procesov ustrážiť aj synchronizáciu procesov

Synchronizace procesů

Kritické oblasti sú časti programu, ktoré využívajú zdieľané premenné a sú citlivé na synchronizáciu. Sú to teda len tie časti programu, ktoré pracujú so zdieľanými premennými, a pre ktoré treba zabezpečiť vzájomné vylučovanie. Zostávajúca časť procesov netvorí kritické oblasti, pretože nepoužíva zdieľané premenné a nekladie preto žiadne požiadavky na synchronizáciu. Vzájomné vylučovanie zabezpečuje, aby žiadne dva procesy neboli súčasne v kritickej oblasti v rovnakom čase a tým sa má zabráňuje časovej závislosti. Samotné vzájomné vylučovanie nie je dostatočnou podmienkou pre správnu spoluprácu paralelných procesov a pre efektívne využívanie zdieľaných prostriedkov.

Pro správné řešení problému kritické sekce je třeba zajistit:

- **Podmínku bezpečnosti (vzájemné vyloučení; mutual exclusion) –** když proces provádí kritickou sekci, nemůže žádný jiný proces začít provádět kritickou sekci sdruženou se stejným zdrojem.
- **Podmínku živosti (trvalost postupu) –** jestliže žádný proces neprovádí kritickou sekci sdruženou s nějakým zdrojem, pak výběr procesu, který chce vstoupit do kritické sekce sdružené s tímto zdrojem, nesmí trvat nekonečně dlouho.
- **Podmínka spravedlivosti (konečnost doby čekání) –** musí existovat horní mez počtu, kolikrát je povolen vstup do kritické sekce sdružené s nějakým sdíleným zdrojem jiným procesům, než procesu, který vydal žádost o vstup.

O časovej závislosti procesov (race condition) hovoríme v prípadoch, keď výsledok priebehu dvoch alebo viacerých procesov závisí od relatívnej rýchlosti ich vykonania (na poradí, v akom im bol prideľovaný procesor). Časová závislosť je výsledkom explicitného alebo implicitného zdieľania dátových štruktúr alebo periférnych zariadení dvomi alebo viacerými procesmi. Pre riešenie problému časovej závislosti operačný systém musí poskytnúť mechanizmy pre synchronizáciu vykonania paralelných procesov.

Úlohou synchronizácie je zaistiť vzájomné vylúčenie paralelných procesov, ktoré využívajú zdieľané prostriedky. Prakticky to znamená, že sa rýchlosti procesov musia zosúladiť tak, aby sa časy vykonania ich kritických sekcií neprekrývali. Pri tom sa uplatňujú dva základné princípy:

synchronizácia aktívnym čakaním znamená, že sa odsun kritickej sekcie uskutoční vložením pomocných (obyčajne prázdnych) inštrukcií do kódu procesu.

prostriedky pre synchronizaciu aktívnym čakaním: Zákaz prerušenia (vhodné pre procecy jadra OS), použitie premenných programov (algoritmus pekára), inštrukcie procesora TSL (Test and Set Lock) a SWAP,

Synchronizácia pasívnym čakaním znamená, že sa odsun kritickej sekcie uskutoční dočasným pozastavením procesu, kým sa kritická sekcia neuvoľní

Prostriedky pre synchronizáciu pasívnym čakaním:
Semafory sú synchronizačné prostriedky, ktoré navrhol E.W.Dijkstra. Používajú pasívne čakanie procesov na uvoľnenie zdieľaného prostriedku a dovoľujú chrániť naraz viac prostriedkov určitého druhu.
Proces, který vstoupí do kritické sekce zvedne semafor a ostatní procesy mohou provádět své kritické sekce s jinými semafory. Proces pak po ukončení kritické sekce zas semafor shodí. Takto řeší OS delší kritické sekce. Špecikace smeaforu:

- Datový typ semafor je proměnná typu integer + 2 operace: čekej na událost (na zvednutí semaforu a zvednutý shoď); oznam událost (zvednutí semaforu). Je to synchronizační nástroj nízké úrovně.
- Semafor má celočíselný čítač, a funkce *acquire*() a *release*(). Pokud je čítač nula, pak zavolání *acquire* () čeká, až bude semafor zvednut. Pokud je větší, než nula, pak získá

přístup do kritické sekce a semafor shodí (sníží čítač na nulu). Po provedení kritické sekce operace *release*() následně semafor zase zvedne (zvýší čítač). *Tento popis je asi nepřesný, ale pro jednoduchý popis by to mohlo stačit, jedná se asi o binární semafor.*

Obecný semafor – celočíselná hodnota z neomezeného intervalu.

Binární semafor (mutex) – hodnota 0 nebo 1, jednodušší implementace, lze s jeho pomocí implementovat obecný semafor.

Monitory - pokud zapomeneme v semaforu použít operaci *release*() na konci kritické sekce, potom procesy čekající ve frontě budou čekat navždy (uváznutí). Proto se používá monitor, synchronizační nástroj vyšší úrovně, který zavádí invariant, který musí být splněn v případě, že proces končí kritickou sekci. Dále obsahuje semafor pro výlučný přístup a sadu procedur, kterými lze manipulovat se sdílenou proměnnou. Deklaruje se proměnná typu condition, na které jsou definované dvě operace: *wait*() – proces, který volá tuto operaci je potlačen do doby, než jiný proces provede operaci *signal*(); *signal*() – aktivuje právě jeden proces, který volal *wait*(). Monitory jsou bezpečnější než semafory, neboť jejich použití může řešit kompilátor a ne programátor, který může na správné zamykání zapomenout.

Synchronizační úlohy

Producent – Konzument V tejto úlohe sa jedná o dva spolupracujúce procesy, ktoré komunikujú cez vyrovnávaciu pamäť obmedzenej veľkosti. Prvý proces produkuje informáciu a vkladá ju do vyrovnávacej pamäte, odkiaľ ju druhý proces vyberá

- Je třeba jeden semafor pro vzájemné vyloučení na bufferu (*mutex*) a další 2 semafory pro indikaci prázdných a plných míst (*full* a *empty*).
- Zajistí se tím vlastnosti vzájemného vyloučení, dále vyloučení možnosti plnění již plného bufferu nebo čtení z prázdného bufferu.

Čtenáři a Písaři – souběh čtení a modifikace dat. Dátový objekt (súbor alebo záznam) je zdieľaný medzi niekoľkými procesmi. Niektoré procesy môžu len čítať obsah zdieľaného objektu a sú nazývané čitatelia, iné ho môžu modifikovať (t.j. čítať a zapisovať) a sú nazývané zapisovatelia. Synchronizačný problém spočíva v zaistení výlučného prístupu zapisovateľov. Potom čitatelia môžu pristupovať k objektu paralelne, zatiaľ čo zapisovatelia len po jednom. V opačnom prípade môže vzniknúť chaos! Tiež sa to da riesit pomocou semaforov.

- Operace zápisu musí být exkluzivní. Operace čtení mohou čtený zdroj sdílet. Operace čtení musí být výlučná s operací zápisu.
- S prioritou čtenářů – první čtenář daného zdroje zablokuje všechny písaře, poslední čtenář opět uvolní přístup písařům, písaři můžou stárnout. Řešení: semafor *wrt* pro písaře; semafor *readcountmutex* pro čtenáře, navíc využití integer *readcount* určující počet aktivních čtenářů, pokud se zvýší na 1, zamkne se *wrt*, pokud se sníží na 0, odemkne se *wrt*.
- S prioritou písařů – totéž + první písař zablokuje přístup ke zdroji písařům, čtenáři můžou stárnout. Řešení složitější: 5 semaforů a 2 integer.

Večeřící filozofové – řešení uváznutí. Je päť filozofov, ktorí trávia svoj život jedením a premýšľaním. Filozofi sedia okolo okrúhleho stola, na stole je päť tanierov so špagetami a päť vidličiek. Z času na čas premýšľanie filozofa vyruší hlad a on sa chce najesť. Za týmto účelom potrebuje dve vidličky. V danom okamihu filozof môže zobrať len jednu vidličku zo stola a potom sa pokúsi zobrať tú druhú. Samozrejme nemôže zobrať vidličku, ktorú už má jeho sused. Ak dostane obidve vidličky, môže jesť, inak musí nechať vidličku na stole a pokúsiť sa o získanie vidličiek neskôr. Naopak, ak ich dostane, naje sa, položí vidličky na stôl a pokračuje v premýšľaní. Synchronizačný problém tu spočíva v tom nedovoliť filozofovi držať jednu vidličku a čakať na uvoľnenie druhej, pretože môže nastať uviaznutie.
Možné řešení: zrušení symetrie, filozof smí uchopit vidličky jen když jsou obě dvě volné a musí je uchopit v kritické sekci. Rieši sa to pomocou Monitora.

Synchronizace procesů pomocí komunikace mezi nimi

Prostriedky pre komunikáciu medzi procesmi v rámci jedného systému:

Správy: nad správami sú definované obyčajne aspoň dve základné operácie:
send(správa)
receive(správa)
Správy môžu mať buď pevnú alebo variabilnú dĺžku. Procesy, ktoré komunikujú, musia mať možnosť ako sa na seba odkazovať. Môžu použiť priamu alebo nepriamu komunikáciu.

Pri priamej komunikácií každý proces, ktorý potrebuje komunikovať, musí explicitne pomenovať svojho partnera. V tomto prípade operácie send a receive majú tvar:
send (P, správa) - zasiela správu procesu P,
receive (Q, správa) - prijíma správu od procesu Q.

Pri nepriamej komunikácii sú správy zasielané a prijímané z **mailboxov** (niekedy sú tiež označované ako porty). Na mailbox sa môžeme pozerať abstraktne ako na schránku, do ktorej sa správy môžu posielať a vyzdvihovať. Každý mailbox má svoj unikátny identifikátor. Jeden proces môže komunikovať s ďalšími procesmi cez niekoľko rôznych mailbox-ov. Dva procesy môžu komunikovať cez mailbox, len ak je tento zdieľaný. Operácie send a receive majú v tomto prípade tvar:
send (A, správa) - zasiela správu do mailbox-u A,
receive (A, správa) - prijíma správu od mailbox-u A.
Ukážka komunikácie:
Máme k dispozici schránku o kapacitě *n* zpráv a funkce pro zaslání zprávy (send) a pro příjem zprávy (receive). Pokud je schránka prázdná a proces volá *receive*(), pak se proces uspí. Pokud je schránka přeplněná, a proces volá *send*(), pak se také uspí. Pokud je schránka prázdná a proces volá *send*(), pak probudí jeden z čekajících procesů, který zavolal *receive*() na prázdnou schránku a analogicky naopak.
Příklad:
Vzájemné vyloučení zprávami

- mailbox *mutex* sdílená *n* procesy
- *send*() je asynchronní, končí odesláním zprávy
- *receive*() je synchronní, čeká, až je mailbox *mutex* neprázdná
- inicializace: *send*(*mutex*, ‘go’)
- do kritické sekce vstoupí proces P, který dokončí *receive*() jako první
- ostatní procesy čekají, dokud P zprávu „go“ nevrátí do schránky

Producen-konzument

- producent umisťuje položky do mailboxu *mayconsume*
- konzument může konzumovat položky z *mayconsume*
- do mailbox *mayproduce* se umístí *n* prázdných zpráv, tyto neprázdné zprávy se můžou přepsat produkovanými zprávami (omezení velikosti na *n* zpráv)

Zdieľaná pamäť - poskytuje najrýchlejšiu komunikáciu medzi procesmi. Ten istý pamäťový segment je mapovaný do adresných priestorov dvoch alebo viacerých procesov. Ihneď ako sú dáta zapísané do zdieľanej pamäte, procesy, ktoré majú k nej prístup môžu tieto dáta čítať. Pri súbežnom prístupe k zdieľaným dátam je potrebné zaistiť synchronizáciu prístupu. V tomto prípade zodpovednosť za komunikáciu padá na programátora, operačný systém poskytuje len prostriedky pre jej uskutočnenie.

Rúry (Pipes) a FIFOs (pomenované rúry) používané najčastejšie na presmerovanie výstupu jedného procesu na vstup ďalšieho. Je to najvhodnejší prostriedok na implementáciu interakcie medzi procesmi založenej na metóde producent/konzument. V unixových systémoch je rúra reprezentovaná znakom |. FIFOs sa od rúr líšia len tým, že používajú špeciálne súbory, do ktorých po poradí zapisujú svoje bajty a tak je možné tieto pomenované rúry za behu medzi sebou zdieľať.

Prostriedky pre komunikáciu medzi procesmi v distribuovaných systémoch:

RPC - synchronna komunikacia pomocou sprav (vysielač musí počkať až do okamihu, kedy je k prevzatiu správy pripravený aj prijímač), procedurálna komunikácia, ktorá pripomína bežnú procedúru. RPC mechanizmus okrem vlastnej výmeny žiadostí medzi klientom a serverom, prevedením funkcie serverom a vrátením odpovede klientovi aj mechanizmus, ktorý dáva procedurálnej komunikácii formu bežného volania procedúry. Tento mechanizmus sa nazýva ako stub
RMI - Volanie vzdialenej metody (Remote Method Invocation) pouzivane v JAVE, podobne RPC s tym rozdielom ze RMI dovoluje odovzdavanie objektov ako parameter volania vzdialenej metody.
Sokety - koncove body pre komunikaciu. Procesy komunikuju pomocou dvojice soketov. Sokety vyuzivaju architektury klinet-server, co znamena ze mozu byt pouzite pre lokalne alebo sietove spojenia. Soket pozostava z IP adresy uzla ku ktorej je pripojene cislo portu

Uváznutí

Hovoríme, že množina procesov uviazla, keď každý proces z množiny čaká na udalosť, ktorá môže vyvolať len iný proces z množiny. Pretože všetky procesy z množiny čakajú, žiadny z nich nemôže vyvolať vznik niektorej očakávanej udalosti, ktorá by mohla uvolniť niektorý z čakajúcich procesov a všetky procesy preto čakajú do nekonečna.
Pro vznik uváznutí jsou 4 podmínky (3 nutné, poslední postačující):

- Výlučný přístup (mutual exclusion)
- Postupné přidělování prostředků (hold and wait)
- Nepreemptivní přidělování prostředků – jednou přidělené prostředky nelze odebrat.
- Cyklické čekání

Všeobecne môžeme problém uviaznutia riešiť:
Ignorovaním problému - Ak napríklad môžme odhadnúť strednú dobu medzi dvoma uviaznutiami na jeden rok, zatial čo poruchy technických prostriedkov alebo zlihania vinou chyby v OS sa vyskytnú približne každý mesiac, je jednoduhšie i lacnejšie poruchy uviaznutia ignorovať.
Detekcia uviaznutia a zotavenia systému (prevenciu pred uviaznutím) - nesplniť niektorú z podmienok (spooling, jednorázové pridelovanie, násilné odobratie prostreidku, hierarchické pridelovanie)
Vyvarovanie sa uviaznutiu - prideľovať procesom prostriedky a zároveň kontrolovať, či požadované prideľovanie nevedie k uviaznutiu procesov. Správca prostriedkov si pri tom ponecháva potrebné rezervy prostriedkov, ktoré zaručia, že procesy budú môcť dokončiť svoje spracovávanie a nedôjde k uviaznutiu. Táto metóda sa nazýva *algoritmus bankára.*