PDF1

Operační systém (OS)

- **Jadro + systemove programy**

• most mezi hardwarem a běžícím programem
• most mezi hardwarem a uživatelem
• vytváří prostředí pro běh programů (procesů, vláken)

Koncepční pohled na OS:

1. **Abstrakce -** nutná pro zvládnutí složitosti, skrývá složitost přístupu k hardware na nejnižší úrovni, detaily architektury a implementace. Jádro operačního systému poskytuje základní množinu operací na vyšší úrovni než jsou instrukce – rozšiřuje instrukční repertoár.
    a. Operacie jadra POSIX: vytváření procesů, běh procesů, přidělování paměti, správa paměti, vstup/výstup, soubory
2. **Virtualizace** – sdílení prostředků
    a. Běh procesů v režimu sdílení času (time sharing) procesoru vytváří iluzi vlastního procesoru pro každý proces, podobně virtualizace paměti poskytuje každému procesu iluzi vlastního adresového prostoru (vlastní paměti)
3. **Správa prostředků** – maximalizace využití, bezpečnost. Operační systém musí tyto prostředky přidělovat tak, aby byly maximálně využity
    a. OS = správce sdílených hardwarových prostředků:
        i. Procesor,
        ii. Pamať
        iii. V/V
    b. Operace s prostředky:
        i. přidělování
        ii. evidence přidělení a využití
        iii. ochrana proti nesprávnému použití
        iv. odebírání, uvolňování
        v. řešení výjimek a chybových stavů

Služby operačního systému:

- operace jádra (rozšíření instrukčního repertoáru)
- uživatelské rozhraní – interpret příkazů (shell), GUI
- autentizace, účtování
- správa služeb (spouštění, monitorování, restart)
- systémové služby (tisk, zálohování, správa systému, atd.)
- síťové služby (souborové, DNS, DHCP, Web, DB, atd.)

Typy jadier:

- Vsetko v jadre - MS DOS

- Mikrojadro - minimum v jadre
- V jadre je to co tam ma byt z hladiska efektivity - unix

Jadro OS:
- Složitý paralelní program, který je zaveden do paměti při startu počítače a řídí činnost hardware počítače.
- Uživatelské a systémové programy běžící vně jádra si můžeme představit jako podprogramy volané z jádra.
- kritéria návrhu jádra:
    - efektivita využití hardware
    - spolehlivost
    - Bezpečnost
- komplikace návrhu jádra:
    - asynchronní (nedeterministický) paralelní běh různých částí jádra
    - asynchronní (nedeterministický) vnější zdroj přerušení spouštějící paralelní běh různých částí jádra

Architektura počítačů
- jeden nebo více procesorů
- V/V řadiče a procesory komunikují každý s každým nebo omezeně
- procesory a V/V řadiče pracují nezávisle a paralelně
- **Počet procesorů a organizace paměti:**
    - Jednoprocesorové
    - Víceprocesorové:
        - jedna sdílená paměť
        - rozprostřená sdílená paměť
        - distribuované systémy
    - Medzi procesormi je zajištěna konzistence obsahu všech pamětí na úrovni hardware

Redukce latence v zařízení
- Minimalizace režie - doba zpracování, interpretace příkazů
- Zkracování přístupovej doby
- **Řešení:**
    - Čtení dopředu, zápis v pozadí
    - Paralelně zpracovávat více příkazů současně

Operační systém v době provádění V/V:
- Může aktivně čekat na výsledek
- Může dělat něco užitečného - V/V řadič musí mít možnost signalizovat ukončení operace a přerušit běh procesoru

Zpracování přerušení (interrupt):
1. dokončení právě prováděné instrukce
2. zablokování dalších přerušení (všech nebo menší priority)

3. uložení stavu procesoru (registry, stav, adresový prostor)
4. skok do obslužného podprogramu přerušení v systémovém (privilegovaném) režimu činnosti procesoru
5. obsluha přerušení (může být přerušena přerušením vyšší prio)
6. obnova stavu procesoru (pokračování v přerušené práci)

Klasické jádro - podprogram obsluhy přerušení má speciální postavení z hlediska synchronizace. Pokud běželo v bodě 1 jádro, nemůže podprogram obsluhy přerušení volně manipulovat s datovými strukturami jádra – nutná synchronizace přístupu k datovým strukturám používaným z podprogramu obsluhy přerušení a jádra samotného - nesmí dlouhodobě blokovat běh
Koncepce mikrojádra - obsluha přerušení ve formě procesu – přerušení pouze odblokuje odpovídající proces v jádře, ten dále používá normální synchronizaci uvnitř jádra

Návrat do uživatelského režimu po preruseni – není problém, struktury jádra nejsou používány, lze před návratem dělat cokoli
Návrat do systémového režimu -

- do přerušeného místa → nelze na základě přerušení spustit nějakou aktivitu v jádře nepreemptivní jádro
- jinam = lze pozastavit přerušenou rozpracovanou službu

Typy operačních systémů

1. A) Monoprogramové - MS-DOS:
   • aktivní pouze jeden proces,
   • jednodušší implementace - nenastává souběžnost provádění,
   • využití zdrojů slabé, obvykle jen jeden uživatel.
1. Multiprogramové (multitasking, multiprogramming):
   a. • aktivních více procesů současně,
   b. • efektivnější využití prostředků,
   c. • jednodušší implementace vyšších vrstev (GUI, síť. rozhraní),
   d. • nutnost pro víceuživatelský systém.

   B-1) Jednoprocesorové (uniprocessor, UP)
   B-2) Víceprocesorové, paralelní (multiprocessor, MP)
   B-2-a) Symetrické multiprocesorové systémy (SMP)
   • kód jádra i kód procesů je prováděn na všech procesorech,
   • procesory mají rovnocenný přístup k operační paměti, V/V zařízením a přerušovacímu systému.
   B-2-b) Nesymetrické multiprocesorové systémy

   Režim sdílení času
   - v daný okamžik je prováděno pouze tolik procesů, kolik je procesorů,
   - • každý proces má procesor přidělen pouze omezenou dobu, poté je vystřídán jiným aktivním procesem,

- • během delšího intervalu se rozprostře výpočetní výkon procesorů mezi všechny aktivní procesy a tím se vytváří iluze současného provádění všech běžících procesů,
- operační systém musí umět pozastavit rozpracovaný proces a poté v něm pokračovat tak, aby to nebylo z procesu pozorovatelné.

Ochrana operačního systému:
1. znemožnění modifikace kódu a datových struktur jádra OS,
2. zabránění provádění V/V operací,
3. zabránění přístupu do paměti mimo přidělený prostor,
4. odolnost vůči chybám (odebrání procesoru při zacyklení).

Nutná podpora na úrovni hardware:
1. Dva režimy činnosti procesoru,
2. privilegované instrukce povolené pouze v systémovém režimu (V/V, změna režimu, zpracování přerušení),
3. ochrana paměti – definuje přístupné úseky adresového prostoru pro běžící proces,
4. přerušovací systém, přerušení převede procesor do systémového režimu,
5. generátor pravidelných přerušení (časovač),
6. pro efektivní V/V nutné DMA nebo inteligentní řadič (přenosy velkých objemů dat)

Přechody mezi režimy:
- Přerušení
- Návrat z obsluhy přerušení
- Volání jádra
- Návrat z volání jádra
- volání jádra je realizováno speciální instrukcí (SVC, lcall, int, trap),
- • jediný styk procesu s okolním prostředím,
- • vše je zprostředkováno jádrem operačního systému,
- • proces je zapouzdřen, operační systém pro něj vytváří iluzi virtuálního počítače.

Definice rozhraní jádra
1. **Na úrovni binární** - systémovo závislé
2. **Na úrovni zdrojové -** systémovo závislé

Jádro OS
Sprava processoru
Proces – prováděný program
Virtuální procesor – jádro OS zajišťuje provádění kódu programu se zabráněním vlivu činnosti jiných procesů, program je prováděn tak, jakoby měl procesor pouze pro sebe

Funkce správy procesoru

- vytváření, rušení procesů
- pozastavení, pokračování provádění procesu
- synchronizace procesů (čekání na ukončení spuštěného)
- komunikace mezi procesy (předávání parametrů, dat)

Přepnutí kontextu – pozastavení prováděného procesu a pokračování v provádění jiného pozastaveného procesu = předání procesoru mezi procesy:

Preemptivní – bez spoluúčasti procesů,
Nepreemptivní – kooperativní předání procesoru.

Kdy nastává přepnutí kontextu:

- vyčerpání časového kvanta (plánování v režimu sdílení času)
- • zahájena blokující operace (čtení, zápis)
- • proces je pozastaven (sleep, wait)
- • má běžet proces s vyšší prioritou

Adresový prostor - všechny adresy dostupné programu (logický AP) - iluze virtuálního počítače
Stav procesu – všechny informace, které musí být uloženy při pozastavení procesu = registry, PC, SP

Hlavní paměť

- Adresový prostor procesů je u většiny moderních operačních systémů oddělený
- Transformaci adresy z logického adresového prostoru procesu na fyzickou adresu operační paměti zajišťuje MMU
- Operační systém musí zajistit správné naplnění MMU před přidělením procesoru danému procesu (při přepnutí kontextu) a dále při každé změně obsazení (využití) adresového prostoru

Funkce správy paměti

- virtualizace adresového prostoru
- • nastavení a aktualizace transformace adres
- • udržování procesů v paměti (co nejvíce)
- • přidělování a evidence přidělených úseků paměti
- • zavádění a odkládání procesů
- • sdílení paměti mezi procesy (sdílení kódu, dat)
- • ochrana paměti

C) V/V

- Standardní rozhraní mezi procesy a V/V zařízeními.
- Realizace abstraktních V/V operací (čti řádek, zapiš blok):
  - • zahájení V/V
  - • obsluha přerušení
  - • ukončení V/V

• systémový časovač

Správa V/V - společná vrstva rozhraní V/V a ovladačů

Typy operačních systémů

- Plánování:
  • dávkové (batch),
  • sdílení času (timesharing),
  • systémy reálného času (real-time)
- Použití:
  • univerzální
  • specializované (souborový server, databázový server, RT)
- Počet uživatelů:
  • jednouživatelské,
  • víceuživatelské

Struktura OS

- Vztah HW-jádro-proces

1. **Monolitické jádro (monitor)**
   a. bez vnitřní struktury (big mess)
   b. volání mezi moduly voláním podprogramů
   c. žádné omezení volání a vztahů mezi moduly
   d. uživatelský proces = podprogram jádra
   e. často bez rozdělení na systémový/uživatelský režim
2. **Jádro (kernel)**
   a. jádro běží v systémovém režimu
   b. • procesy běží v uživatelském režimu a volají jádro (jádro je
   c. pasivní), striktní rozhraní mezi procesy a jádrem
   d. • jádro vytváří pro proces abstrakci virtuálního počítače
   e. Služby jádra jsou podprogramy procesu.
3. **Mikrojádro (Mach)**
   a. Služby jádra částečně v systémovém režimu (mikrojádro), částečně v uživatelském režimu (systémové procesy)
   b. Minimální jádro:
      i. • přepínání kontextu
      ii. • přidělování paměti
      iii. • ochrana paměti, nastavení adresového prostoru
   c. Ostatní služby řešeny samostatnými procesy nad mikrojádrem:
      i. • prostředí procesů, spouštění procesů
      ii. • autentizace, autorizace, účtování
      iii. • virtualizace paměti, odkládání, zavádění
      iv. • V/V
      v. • síťové vrstvy
      vi. • systém souborů
   d. Systémové služby jsou realizovány procesy

4. **Exokernel**
    a. jádro vykonává většinu služeb ve prospěch nějakého procesu
    b. Služby jsou poskytovány jako podprogramy uvnitř uživatelského procesu
    c. Jádro v systémovém režimu je voláno pouze pro synchronizaci a přidělování prostředků
5. **Virtuální počítač**
    a. jádro běží zcela v uživatelském režimu
    b. v systémovém režimu běží pouze monitor virtuálního počítače - zachytává a emuluje privilegované instrukce
    c. • plně virtualizuje všechny prostředky
    d. • nad monitorem mohou běžet plné operační systémy

PDF2

Proces je sekvenčně prováděný program ve vlastním adresovém prostoru, proces můžeme uvažovat také paralelně prováděné příkazy a instrukce. Nezajímá nás děj uvnitř procesu, pouze stav na začátku a po ukončení.
Stav procesu je definován stavem proměnných a pozici v programu.

výsledek paralelního systému při paralelním provádění vždy stejný bez ohledu na posloupnost provádění - **deterministicky**. Výsledek je nedeterministický, pokud výsledky jednotlivých procesů závisí na pořadí jejich provádění. **Deterministicky** - Posloupnost hodnot zapisovaných do všech proměnných závisí pouze na počátečním stavu proměnných.

Bernsteinovy podmínky neinterference:
Dva procesy P_i a P_j jsou neinterferující, jestliže platí:

1. $P_i < P_j$ nebo
2. $P_j < P_i$ nebo
3. $R(P_i) \cap W(P_j) = W(P_i) \cap R(P_j) = W(P_i) \cap W(P_j) = \emptyset$

Věta: Paralelní systém skládající se ze vzájemně neinterferujících procesů je deterministický.

Dva paralelní systémy obsahující stejné procesy jsou **ekvivalentní** , pokud jsou deterministické a generují stejné sekvence hodnot pro všechny proměnné.
Paralelní systém T je **maximálně paralelní** , pokud je deterministický a vyjmutí libovolné hrany (P_i , P_j) z grafu pokrytí způsobí, že P_i a P_j budou interferující.

PDF3

Proces = samostatně prováděný program ve vlastním adresovém prostoru.
Program - statický kód, počáteční data (program vi)
Stav procesu – registry procesoru, data, zásobník, systémové prostředky.

- Proces může čekat pouze na své dětské procesy
- Pokud rodič skončí dříve než dítě, stane se dítě sirotkem => jeho rodičem se stane proces s pid=1
- Pokud dětský proces skončí a rodič nečeká na dokončení (nezajímá se o jeho stav), stane se z dětského procesu zombie

Vlákna

- samostatně prováděná část programu v rámci jednoho „procesu“
- Proces“ v systémech s vlákny:
    - • sada souběžně prováděných vláken v jednom adresovém prostoru,
    - • přestává být jednotkou přidělování procesoru,
    - • zůstává obálkou vláken pro přidělování systémových prostředků a správu adresového prostoru.
- Přepínání kontextu mezi vlákny, spouštění a synchronizace vláken by měly mít menší režii než u procesů (jinak by neměla vlákna smysl). Spuštění vláken je typicky o 1 až 2 řády rychlejší než u procesů (nemusí se vytvářet a kopírovat adresový prostor), stejně tak synchronizace.
- **použití vláken**
    - Urychlení běhu - paralelní programování (multiprocessing)
    - Proložení V/V a běhu - zálohování, vypalování CD, multimédia
    - Síťové servery (Web, FTP)
    - Zpřístupnění sdílených dat více klientům (DB, IRC, MUD)
    - Grafické uživatelské rozhraní
    - Systémy reálného času
- Všechna vlákna v rámci jednoho „procesu“ sdílí společný adresový prostor a systémové prostředky
- •všechna vlákna sdílí stejný kód a data (není zde ochrana!),
- každé vlákno má své registry, zásobník, stav provádění.

Termín **proce**s v praxi (POSIX) reprezentuje vlákna běžící v jednom adresovém prostoru, která sdílí:

- • identifikaci procesu (pid = getpid())
- • majitelství (uživatel - uid = getuid(), skupina - gid = getgid())
- • nastavení zpracování signálů (sigaction())
- • deskriptory souborů (fd = open(), pipe(), socket())

Proces - jednotka pro přidělování systémových prostředků
Vlákno - jednotka pro přidělování procesoru (čili proces v terminologii teorie OS)

Přidělování procesoru vláknům:

- • globální - pro každé vlákno v systému nezávisle na procesech
- • lokální - na úrovni procesů, čas procesoru dostává proces

Vlakna nemaju vztahy rodic-potomok

Problémy implementace vláken - statické proměnné standardních funkcí:

- errno musí obsahovat chybový kód specifický pro vlákno, které volalo funkci standardní knihovny
- Mnohé funkce standardní knihovny C si ukládají něco do statických proměnných – nejsou pak reentrantní.
- Stejně tak funkce klasického rozhraní

Implementace vláken

1. **N:N (1:1)** - na úrovni jádra systému, vlákna na úrovni uživatelské jsou reprezentována v jádře
    a. režie přepínání kontextu
    b. jádro musí evidovat všechna vlákna
    c. plné využití více procesorů v jednom programu
    d. volání jádra přímá
    e. podpora vláken na úrovni uživatelské nemá dostatečné informace o akcích na straně jádra, plánování je problém.
2. **N:1** - na úrovni knihoven, vlákna jsou plně implementována v rámci uživatelského procesu, jádro o nich nic neví
    a. nízká režie jádra, plná kontrola nad plánováním
    b. − všechna blokující volání jádra musí být zapouzdřena
    c. − nelze využít více procesorů v jednom programu, jednotkou přidělování času procesoru je proces
3. **N:M (N≥M≥1)** - kombinovaný přístup, důvody
    a. prováděná vlákna musí být reprezentována v jádře pro správu procesorů, nicméně nemá smysl reprezentovat všechna běžící, stačí tolik, kolik je procesorů,
    b. • čekající (pozastavená) a připravená vlákna nemusí být reprezentována datovými strukturami v jádře, jádro o nich vůbec nemusí vědět - menší režie
    c. • přepínání kontextu přes jádro má větší režii než v rámci uživatelského procesu – dokud lze využít přidělený procesor, probíhá běh v režimu sdílení času pro všechna aktivní vlákna v daném procesu.
    d. • lze simulovat N:1 až N:N nastavením max. počtu vláken na úrovní jádra (thr_setconcurrency())
    e. podpora vláken na úrovni uživatelské nemá dostatečné informace o akcích na straně jádra, plánování je problém.

Light Weight Process - vlákno na úrovni jádra systému. Implementace modelu M:N je značně složitá, LWP mají v jádře obdobnou režii jako procesy (bez adresového prostoru). LWP jsou z hlediska jádra jednotkami přidělování procesorů. Klasický proces pak běží jako 1 LWP vlákno.

PDF4

Virtualizace procesoru u multiprogramování → vlastní procesor
Přepínání kontextu → souběžný běh více procesů

Jednoprocesorový systém - pseudosouběžnost
- Preemptivní přepínání kontextu - kdykoli, proces nemůže ovlivnit okamžik přepnutí kontextu
- Nepreemptivní - kooperativní, proces se musí vzdát procesoru

Víceprocesorový systém - fyzická souběžnost
Nezávislé procesy - bez interakce, neinterferující, deterministické
Kooperující procesy - sdílená paměť, nedeterministické
signály - funkce obsluhy signálu běží pseudosouběžně s programem
Časově závislé chyby (souběh, race conditions) - chyby vznikající díky interferenci při různé relativní rychlosti provádění procesů v paralelním systému. Obvykle jsou spojeny se sdílenými proměnnými.
Atomická operace - nedělitelná operace, nemůže být přerušena uprostřed. stav struktury zůstává konzistentní i při paralelním přístupu,
- **na úrovni procesoru**: Přístup do paměti atomický, ale pouze pro slovo zarovnané na hranici přístupu do společné paměti. Čtení a zápis různých proměnných nemusí probíhat na úrovni společné paměti ve stejném pořadí jak jsou v programu:
    - - překladač – může přeuspořádat příkazy (volatile)
    - - procesor – spekulativní provádění, write buffer
    - - paměťová sběrnice – rozprostřená paměť
  - procesory mohou vidět čtení i zápisy různých proměnných v jiném pořadí! Pořadí zápisů stejné proměnné z jednoho procesoru musí být zachováno.

Synchronizace: zajištění kooperace mezi paralelně (souběžně) prováděnými procesy
Vzájemné vyloučení - pouze jeden může provádět dané operace. Vytváříme tím složitější nedělitelnou atomickou operaci.
Kritická sekce – kód, jehož provádění je vzájemně vyloučené = atomicky prováděný kód
Fairness (spravedlnost) přidělování procesoru
- **unconditional fairnes**s (nepodmíněná) - každý aktivní nepodmíněný atomický příkaz bude někdy proveden
- **weak fairnes**s = unconditional fairness + každý aktivní podmíněný atomický příkaz bude proveden za předpokladu, že podmínka nabude hodnoty TRUE a nebude se měnit
- **strong fairness** = unconditional fairness + každý podmíněný atomický příkaz, jehož podmínka se nekonečně častokrát mění, bude nekonečně krát proveden
- **Formální požadavky na hledané řešení -**
    - bezpečnost (safeness), v daném případě zaručuje vyloučení
    - živost (liveness):
        - - nedochází k uváznutí (deadlock)
        - - nedochází k blokování (blocking)
        - - nedochází k hladovění (starving)

Uváznutí (deadlock) - procesy čekají v synchronizaci na stav, který by mohl nastat, kdyby jeden z nich mohl pokračovat. Porušuje podmínku proces se dostane do kritické sekce v konečném čase

- nemusí nastat vždy
- obtížná detekce

Verifikace živosti – hledání cyklů, které trvale neprochází přes označená místa algoritmu (v daném případě přes kritickou sekci):

Blokování - Bezpečný, Nedochází k uváznutí, Pokud je některý proces ve stavu F, trvale blokuje opakovaný vstup do kritické sekce druhému procesu

- proces čeká v synchronizaci na stav, který generuje jiný proces
- Porušuje podmínku každý proces se dostane do kritické sekce v konečném čase.
- blokování znamená vždy potenciální problém (co, když proces
- ve stavu F skončí), obvykle neefektivní

Hladovění - Bezpečný, Nedochází k uváznut, Nedochází k blokování

- proces může čekat v synchronizaci na stav, který nemusí být nikdy pravdiný v okamžiku testování. Není striktně omezena horní mez čekání (jinak jako blokování).
- - v praxi se obvykle toleruje, závisí na plánovacím algoritmu

Živost (liveness) - algoritmus je živý, pokud je bezpečný a nedochází ke uváznutí, blokování a stárnutí (je zaručeno jeho dokončení v konečné době).

Petersonův algoritmus - Bezpečný, Nedochází k uváznutí, blokovani, starnuti

PDF5

Implementace vzájemného vyloučení
úrovně abstrakce

- synchronizační zámky, semafory, monitory, zasílání zpráv –
- nástroje na úrovni programovacích jazyků
- synchronizační čtení/zápis, zakázání přerušení, speciální
- Prostředky instrukce – nutné pro implementaci nástrojů

Jednoprocesorové systémy

- úsek kódu je atomický, pokud nemůže dojít k přepnutí kontextu (multiprogramování) nebo přerušení
- Vzájemné vyloučení mezi
    - 1) procesy/vlákny jádra a obsluhou přerušení
    - 2) procesy/vlákny v jádře
    - 3) uživatelskými procesy

- Ad1 Vzájemné vyloučení proti obsluze přerušení - Jediný způsob je zakázaní přerušení a tím spuštění paralelně vykonávaného kódu.
- Ad2 Vzájemné vyloučení v rámci jádra -
  - **synchronně** (zahájení čekání, spuštění jiného procesu) – není problém, datové struktury jsou v konzistentním stavu,
  - **asynchronně**, na externí událost (přerušení od časovače, apod.) – pokud zabráníme, je kód jádra nepreemptivní (nemůže se spustit jiný proces, dokud se předchozí synchronně nezastaví).
- Ad3 Vzájemné vyloučení mezi uživatelskými procesy v uživatelském režimu nelze zakázat přepnutí kontextu, musí se řešit jako pro víceprocesorové systémy

a) Zakázat přerušení - blokuje přepínání kontextu

b) Zakázat preemptivní přepínání kontextu v jádře - Kontext se může přepnout pouze:

- • synchronně (explicitně, zahájení čekání),
- • při ukončení služby jádra před návratem do uživatelského režimu (explicitně),
- • po přerušení uživatelského režimu před návratem zpět do uživatelského režimu (jádro neběží, není problém).
- Kontext se nesmí se přepnout v obsluze přerušení, ani při návratu z přerušení zpět do systémového režimu!

c) Vzájemné vyloučení zamykáním datových struktur a povolení preemptivního přepínání kontextu uvnitř jádra

Víceprocesorové systémy - Vždy nutná synchronizace

- 1) **spin_lock** - krátkodobé vyloučení s aktivním čekáním
  - Může střežit pouze kritické sekce, které jsou krátké, neblokující a bez preempce
  - Aktivní čekání je v tomto případě přijatelné, protože pak může být kritická sekce obsazena pouze procesem běžícím na jiném procesoru a ten ji brzy uvolní. Pozastavení procesu by bylo náročnější než krátké aktivní čekání
  - a) Implementace pouze čtením/zápisem
    - • elegantní algoritmy pouze pro malý počet procesů, složitost
    - • dostupnost lepších speciálních atomických instrukcí
  - b) Speciální atomické instrukce
    - Nutná atomická instrukce nedělitelného čtení a zápisu (RMW) do paměti
    - bezpečný, nedochází k uváznutí, blokování
    - - stárnutí (starving) – je možné předbíhání

- 2) **mutex, lock** - dlouhodobé s pozastavením procesu
  -

Problémy implementace vzájemného vyloučení

- cyklus test&set (ll/sc) stále čte a zapisuje společnou paměť
  - zatěžuje společnou paměťovou sběrnici
  - - atomická instrukce je většinou zároveň paměťová bariéra
  - - hyperthreading (SMT) – aktivní čekání blokuje 2. Logický procesor na jednom fyzickém jádru CPU

- Starnutie

Lock-free (Wait-free) programming
- Režie zamykání je v tomto případě větší než provedení pár instrukcí.
- Musíme znát paměťové modely a správně synchronizovat obsah paměti - zámek to udělá za nás
- Nemáme k dispozici dostatečně silné atomické instrukce

Vzájemné vyloučení v uživatelském režimu
Implementace službou jádra je pomalá, má příliš velkou režii v případě, kdy je kritická sekce volná (99% případů), vždy se musí volat jádro.
Atomickými instrukcemi lze na úrovni uživatelské bezpečně testovat a nastavit příznak, co ale v případě, kdy je nastaven?
- Nelze aktivně čekat
- Nelze pozastavit

PDF6

Synchronizační nástroje - semafor
Binární semafor
- Init, lock, unlock
- jako zámek strážící kritickou sekci, pouze dva stavy
- nelze číst hodnotu
- čekání v operaci lock(sem) je pasivní
- operace lock(sem) a unlock(sem) jsou atomické
- odemykat může jiný proces než zamknul (předávání zámku)
- Silný/slabý semafor - nepodléhá/podléhá stárnutí
- Mutex – speciální binární semafor určený pouze pro vzájemné vyloučení, při zamčení má identifikovaného vlastníka, pouze vlastník ho může odemknout (nutné pro řešení inverze priority)
- Pouzitie: vzajomne vylucenie, signalizacia udalosti (nevhodne),
- Problematika implementace binárních semaforů:
    - Rekurzivny : printf zamkne stdout, putchar zamkne stdout, atd..
    - implementace na úrovni uživatelského režimu
    - inverze priority - proces s nižší prioritou blokuje provádění procesu s vyšší prioritou, Řešení:
        - dědění priority (priority inheritance) - po dobu provádění kritické sekce je priorita procesu v kritické sekci zvýšena na max. prioritu všech čekajících:
            - + pokud žádný proces nečeká, zůstává priorita procesu v kritické

sekci nezměněná (neovlivňuje chování systému)
    - - musí se dynamicky upravovat při každém blokujícím zamčení
- horní mez priority (priority ceiling) - po dobu provádění kritické sekce je nastavena vždy statická pevná priorita:
    - + pevně deklarovaná priorita je jednoduchá na implementaci
    - - procesu se musí zvyšovat priorita vždy
- Vzdy iba pre binarny semafor, mutex, monitor

Obecný semafor

- Počáteční hodnota určuje „kapacitu“ semaforu – kolik jednotek zdroje chráněného semaforem je k dispozici. Jakmile se operací down() zdroj vyčerpá, jsou další operace down() blokující, dokud se operací up() nějaká jednotka zdroje neuvolní
- Init, down, up

Důvody proč mutex:

1. Inverze priority při zamykání (nelze řešit).
2. Rekurzivní deadlock – co když zamkne zámek ten samý proces, který už ho má zamčený? U binárního zámku lze detekovat (je majitel zámku), u obecného nelze detekovat
3. Efektivnější u relaxovaných paměťových modelů (acq/rel).
4. Deadlock při ukončení procesu – co když proces, který má zámek zamčen, skončí bez uvolnění zámku? U binárního lze detekovat a řešit, u obecného nelze
5. Náhodné uvolnění – co když se párová operace down() zratí? Obecný semafor pak pustí příště dva procesy do kritické sekce, u binárního semaforu se nic nestane.

- Pouzitie: vzajomne vylucenie, signalizacia udalosti, hlidani zdroje s definovanou kapacitou n

Klasické synchronizační úlohy

1. Vzájemné vyloučení (Mutual Exclusion)
2. Producent/konzument (Producer/Consumer, Bounded Buffer)
    - producenti produkují data do sdílené paměti, konzumenti je z ní odebírají
    - • konzumenti musí čekat, pokud nic není vyprodukováno
    - • producenti musí čekat, pokud je paměť plná
    - • operace s pamětí musí být synchronizovány
3. Čtenáři/písaři (Readers/Writers)
    - přístup ke sdíleným datům
    - čtenář pouze čte data
    - písař čte a zapisuje
    - vzájemné vyloučení všech je příliš omezující:
        - • více čtenářů současně
        - • pouze jeden písař
4. Pět filozofů (Dining philosophers)
    - vyhladovění – je třeba zajisti, že když chce jíst, dostane v konečném čase najíst a nebude systematicky předbíhán jinými filozofy

- • uváznutí – všichni přijdou ke stolu a uchopí levou vidličku, nikdo nemůže jíst

PDF7

Monitor - abstraktní datový typ
- 1. Sdílené proměnné dostupné pouze operacemi monitoru.
- 2. Provádění operací jednoho monitoru je vzájemně vyloučené.
- 3. Inicializace monitoru nastaví sdílené proměnné.
- zajistí vzájemné vyloučení operací nad monitorem = všechny operace monitoru jsou atomické.
- Pozastavení - nelze čekat aktivně uvnitř monitoru
    - condition - fronta čekajících procesů
    - Operace:
        - C.wait; pozastavení procesu, vzdání se monitoru
        - c.signal; odblokování prvního čekajícího (pokud je), ten získá opět výlučný přístup k monitoru
        - po c.signal jsou dva procesy v monitoru -> někdo musí být pozastaven
            - 1. Blokující signalizace podmínky
            - 2. Neblokující signalizace podmínky

1. Hoare, Hansen - proces P provádějící c.signal je pozastaven a pokračuje odblokovaný proces Q za c.wait. Až Q opustí monitor nebo začne zase čekat, pokračuje nejprve P a pak teprve ostatní procesy čekající na vstup do monitoru. Signalizace stavu je kooperativní, ten kdo stav změnil, efektivně předá monitor probuzenému čekajícímu procesu. Pokud nikdo nečeká, je signalizace prázdnou operací.

2. Lampson, Redell (Mesa, 1980) - operace c.notify pouze odblokuje pozastavený proces, monitor zůstane dále stejnému procesu, teprve po opuštění monitoru se může odblokovaný proces dostat do monitoru a pokračovat za c.wait, soutěží ovšem s procesy vstupujícími do monitoru normálně - nelze zaručit splnění testované podmínky v odblokovaném procesu! Proto je třeba vždy testovat podmínku čekání po probuzení znovu!

Podmíněné kritické sekce - vymezení kritické sekce a sdílených proměnných
Bariéra - čekání na dosažení stejného místa v N procesech
- Implementace pomocí semaforů:
    - • čítač s počátečním stavem N
    - • proces v bariéře dekrementuje čítač a pokud není nula, čeká na obecný blokující semafor
    - • až přijde poslední, čítač je nula, musí odblokovat všechny čekající (N-1 krát operace up())
- Problém: • blokovaný proces nemusí stihnout provést down(b), další odblokuje bariéru cyklem up() a bude tam o jednu signalizaci více, takže vstup do následující bariéry nebude blokující!

- Řešení - dva čítače a dva blokující semafory, jednou použít jeden pár, pak druhý pár (nebo jeden pro vstup, druhý pro výstup)
- neefektivní řešení, nutná efektivnější synchronizace na nižší úrovni (stromová synchronizace, po párech)

RWLOCK
Nástroj pro synchronizaci typu čtenáři/písaři. Umožňuje paralelní přístup ke sdílenému prostředku bez modifikace a výlučný přístup pro modifikaci.

Signály = standard ISO jazyka C
Typy signálů:

• chybové (dle PDP) = SIGILL, SIGTRAP, SIGIOT, SIGEMT, SIGFPE, SIGBUS a SIGSEGV, implicitně ukončení procesu
• uživatelské = SIGHUP, SIGINT, SIGQUIT, SIGKILL, SIGTERM, SIGUSR1 a SIGUSR2, ukončení procesu (SIGKILL nelze ignorovat).
• systémové = SIGCHLD, SIGSYS, SIGPIPE a SIGALRM, ukončení procesu s výjimkou SIGCHLD.

Zpracování:

• zaslání signálu procesu = jádro systému nebo kill(),
• pokud je proces pozastaven ve službě jádře, pak může zaslaný signál přerušit čekání , havarijně ukončit volání systému
• pokud signál nemůže přerušit čekání pak je příchod signálu pouze zaznamenán a zpracování odloženo až na ukončení čekání nebo návrat z volání jádra systému,
• pokud proces čeká na přidělení procesoru nebo běží, pak je příchod signálu zaznamenán a zpracování odloženo až do chvíle, kdy se vrací ze systémové fáze do uživatelské,
• pokud má proces zaznamenány signály čekající na zpracování, pak při přechodu ze systémové fáze do uživatelské proběhne zpracování čekajících signálů, což může být:

a) implicitní zpracování - u většiny signálů způsobí ukončení procesu.
b) ignorování – signál je zahozen,
c) zpracování ošetřující funkcí v uživatelském programu

Problémy:

• nebezpečí ukončení procesu, přestože si proces ošetřuje signál vlastní funkcí,
• po dobu provádění ošetřující funkce není zablokováno doručování a zpracování signálů stejného typu, takže ošetřující funkce může být vyvolána několikanásobně,
• nereentrantnost standardní knihovny,
• nutnost ošetřit všechna volání standardní knihovny testem na chybový návrat a errno==EINTR a opakovat volání,
• nelze bezpečně čekat na příchod signálu

Bezpečné čekání na příchod signálu - Součástí stavu každého procesu (resp. vlákna) je množina blokovaných signálů. Zpracování signálů z této množiny je blokováno trvale

Zasílání zpráv -
synchronní - odesílatel čeká na přijetí zprávy příjemcem (CSP)
asynchronní - bez čekání, proces může pokračovat

Adresace:

- explicitní (přímá): send(p, msg), receive(q, msg) - p, q jsou procesy
- Implicitní: send(msg), receive(msg) - komukoli, od kohokoli
- Nepřímá: send(m, msg), receive(m, msg) - m je schránka, port

Problémy implementace:

- priorita zpráv a výběr podle priority
- velikost zpráv a efektivní kopie mezi adresovými prostory
- vyrovnávací paměť (0 = rendezvous)

1. Zasílání zpráv

- + jednoduché
- - volný formát zpráv
- - různé implementace
- - zpracování chyb
- - závislost na komunikačním mechanismu

2. Vzdálené volání podprogramů

- Klasické zasílání zpráv - jednosměrné
- Volání podprogramů - zaslání parametrů a převzetí výsledku
- + přijatelná složitost (IDL, XDR)
- + definovaný formát zpráv (IDL)
- + síťová reprezentace dat (XDR)
- + automatické generování pomocného kódu
- + automatické ošetření chyb
- + nezávislé na komunikačním protokolu
- - omezená množina typů
- - statický procedurální charakter

3. Komunikace objektů (CORBA, COM)

- - značná složitost (IDL, XDR, uživatelské typy)
- + definovaný formát zpráv (IDL)
- + síťová reprezentace dat (XDR)
- + automatické ošetření chyb
- + uživatelské typy (jednoduché, složené)
- + dynamický charakter
- + komunikují ne programy, ale objekty

PDF7 pozret!

PDF9

Plánování a přidělování procesoru

- plánování (scheduler) - volba strategie a řazení procesů
    - • sdílený plánovač
    - • samostatný plánovač
- přidělování (dispatcher) - přepínaní kontextu na základě naplánování

Cíle plánování:

- • Minimalizace doby odezvy
- • Efektivní využití prostředků
- • Spravedlivé dělení času procesoru
- • Doba zpracování
- • Průchodnost (počet úloh/čas)

fronta připravených procesů - seznam procesů, které mohou běžet
plánování - organizace fronty připravených procesů
přidělování procesoru - přidělení procesoru prvnímu procesu z fronty připravených = přepínání kontextu podle plánování

Univerzální plánovač

- popis běžných plánovacích algoritmů
- Založen na prioritě procesů. Procesor je v okamžiku rozhodování přidělen procesu s nejlepší prioritou.
- Definován třemi charakteristikami:
    - 1. interval rozhodování
    - 2. prioritní funkce
    - 3. výběrové pravidlo

1. Interval rozhodování

- Definuje časové okamžiky, ve kterých je aktivován plánovač.
- Uspořádání procesů se nemůže během intervalu měnit.
- **Nepreemptivní** - proces běží do ukončení nebo čekání
    - + malá režie přepínání kontextu, jednoduchá implementace

    - − delší odezva, nevhodné pro systémy reálného času
- **Preemptivní** - procesu může být odebrán procesor:
    - • v pravidelných intervalech - časové kvantum
    - • odblokování procesu s vyšší prioritou
    - • příchod nového procesu (přerušení, spuštění proces)
- **Selektivní preempce** - pro každý proces (u p , v p):
    - u p = 1, pokud p může přerušit jiný proces, jinak 0
    - v p = 1, pokud p může být přerušen jiným procesem, jinak 0
    - Příklad:
        - • pro časově kritické procesy (1, 0)
        - • pro ostatní (0, 1)
    - Selektivní preempce podle priority:
        - • procesy s vysokou prioritou jsou nepreemptibilní
        - • ostatní preemptibilní

2. Prioritní funkce

- Funkce určující prioritu procesu na základě parametrů
    - paměťové požadavky
    - • spotřebovaný čas procesoru a
    - • doba čekání na přidělení procesoru w
    - • doba strávená v systému r = a+w
    - • celkový čas procesoru t = a po dokončení
    - • externí priorita (důležitost)
    - • perioda d
    - • lhůta zpracování
    - • zátěž systému

3. Výběrové pravidlo

výběr z více procesů se stejnou prioritou:

- • náhodně
- • cyklicky
- • FIFO

Univerzální plánovač vyhodnocuje v určených okamžicích dle (1) prioritní funkci (2) pro všechny připravené procesy a dle výsledné priority a výběrového pravidla (3) přiděluje procesor procesu s nejlepší prioritou (číselně nejvyšší).
Plánovací algoritmy závislé na časových parametrech: P(a, r, t, d)

1. FIFO (FCFS)

Zpracování procesů v pořadí jejich příchodů
(1) nepreemptivní, v čase příchodu procesu (odblokování)
+ jednoduché, malá režie přepínání kontextu
+ deterministická odezva
− krátké procesy musí čekat na dříve zahájené dlouhé,
delší celková doba zpracování, odezva

2. LIFO
Zpracování vždy posledního přišlého procesu
(1) nepreemptivní, v čase příchodu procesu (odblokování)
− stárnutí

3. SJF (Shortest Job First)
Zpracovat vždy nejkratší proces (nutná znalost t)
(1) nepreemptivní, v čase příchodu procesu
+ kratší celková doba zpracování
+ malý počet čekajících procesů
− odhad t (dávkové systémy)
− stárnutí dlouhých procesů při neustálém příchodu krátkých

4. SRT (Shortest Remaining Time)
Zpracovat vždy nejkratší proces k dokončení (nutná znalost t)
(1) preemptivní, v čase příchodu procesu
+ minimální celková doba zpracování
− odhad t (dávkové systémy)

5. Statická priorita (prioritní plánování)
(1) preemptivní (v pravidelných intervalech) i nepreemptivní
Typický algoritmus pro RT systémy, dvě úrovně priorit = vyšší, bez časového kvanta, nižší s časovým kvantem

6. RR (Round Robin - cyklická obsluha)
Rovnoměrné přidělování procesoru po časových kvantech q.
Procesu, který vyčerpá časové kvantum, je procesor odebrán a proces je zařazen na konec fronty připravených procesů.
(1) preemptivní, v pravidelných intervalech
+ dobrá odezva, spravedlivé dělení času procesoru
− celková doba zpracování
Volba velikosti q (10 - 100 ms):
- příliš malé - velká režie
- příliš velké - velká doba odezvy (průměrně q.n/2)

Nastavení q při pozastavení procesu čekáním a následném odblokování:
- ponecháno původní částečně vyčerpané kvantum
- nastaveno nové plné kvantum

7. MLF (MultiLevel Feedback)
(1) preemptivní, v pravidelných intervalech q
• úrovně priorit i = 0..n, počátečně i = 0 (nejvyšší priorita)
• procesy ve stejné úrovni i obsluhovány cyklicky
• po vyčerpání časového limitu T i , je proces přeřazen do úrovně i+1

• délka časového limitu T i = 2 i .T 0
• při dosažení i=n je proces ukončen nebo vrácen do i=n-1
Priorita dlouhodobě běžících procesů je postupně snižována
Problémy:
- překročení maximální úrovně
- proces po delším výpočtu má navždy špatnou prioritu

8. Rate Monotonic (RM)
Plánovací algoritmus pro reálný čas
Proces se opakuje s dobou periody d
Každý běh procesu musí být dokončen dříve než bude spuštěn znovu. Spouštíme vždy nejdříve ten, který má nejkratší periodu.
(1) preemptivní, v čase příchodu procesu

9. Earliest Deadline First (EDF)
Plánovací algoritmus pro reálný čas
Proces se opakuje s dobou periody d
Každý běh procesu musí být dokončen dříve než bude spuštěn znovu. Spouštíme vždy nejdříve ten, který má nejkratší lhůtu k dokončení (který má nejkratší čas do dalšího začátku)
(1) preemptivní, v čase příchodu procesu

Plánovač FreeBSD 4.4
Priorita běžícího procesu je přepočítávána po 40 ms

Plánovač Unix SVR4
Konfigurovatelný plánovač, standardně 3 třídy:
RT (100-159) - statická priorita, různé časové kvantum podle priority
SYS (60-99) - procesy v systémové fázi, vlákna jádra – statická priorita, bez časového kvanta
TS, IA (0-59) - tabulkově parametrizovaný MLF

Plánovač Windows
Rozsah priorit – 0 až 31 (0 je nejnižší)
„normální“ priority – 1..15 – priorita je dynamická, podle kvanta, kde kvantum je 2 (short) nebo 12 (long) intervalů časovače ,real-time – 16..31 – priorita je statická
Plánovaní procesoru je pro vlákna, vlákna mají počátečně prioritu podle třídy procesu

Plánování pro víceprocesorové systémy
Různé požadavky na univerzální systém:
- víceuživatelský,
- web server,
- databázový server,
- paralelní aplikace.

Je třeba zohlednit architekturu (SMP, NUMA), organizaci a propustnost paměťových sběrnic,

vícejádrové/SMT jádra procesorů, sdílení cache L2/L3, atd.

A) Plánovač společný pro všechny procesory:
- neodpovídá požadavkům, přiděluje procesory náhodně,
- omezující při větším počtu procesorů.

B) Jedna fronta připravených procesů, každý procesor vybírá samostatně procesy z této fronty (varianta předchozího):
- globální zámek fronty omezuje paralelní plánování

C) Fronta připravených procesů pro každý procesor:
- problém vyrovnávání zátěže procesorů a migrace procesů („kradení“ procesů z nejdelší fronty)
- řazení nových/probuzených procesů do správné fronty

Efektivní využití cache:
- vícevláknové programy omezit na jeden procesor/jádra v jednom procesoru – efektivní využití cache, TLB, ale ne výpočetního výkonu

Procesor-affinity – plánovač zohledňuje, na kterém procesoru vlákno běželo, a snaží se nemigrovat vlákno na jiné procesory (pouze v případě, že jiný je volný a nemá co dělat)

Plánovač ULE ve FreeBSD

Problém klasického plánovače – musí procházet všechny procesy a přepočítávat priority pro implementaci MLF algoritmu – O(n) (Linux 2.4, BSD plánovač). Princip O(1) plánovače (Linux<2.6.23) – použít pro každý procesor dvě fronty: první – ze které se bere od začátku, druhou - do které se vkládá podle priority. Po vyprázdnění první se fronty prohodí. Prohozením dojde k tomu, že proces s nízkou prioritou nemůže být trvale odstaven od procesoru, dostane jej přidělen alespoň jednou během 2 zpracování front.

Problémy:
- nelze přímo implementovat „nice“
- vkládání do fronty podle priority je náročná operace

Plánovač ULE ve FreeBSD od verze 7:
- pro každý procesor 3 fronty: Idle, Current, Next
- prováděná vlákna vybírána postupně z Current podle priority
- když je Current prázdná, prohodí se Current a Next
- vlákna z Idle se provádí pouze, když je Current i Next prázdná
- pokud nemá procesor co dělat, „krade“ vlákna z front ostatních procesorů (vybírá první z nejzatíženějšího procesoru)
- může se stát, že Idle vlákno na jednom procesoru nedovolí „kradení“ a tím vybalancování zátěže – periodicky (co 0,5s) se zvolí nejméně a nejvíce zatížený procesor a jeho nadbytečná vlákna se přesunou na nejméně zatížený tak, aby měli stejně
- vlákna obsluhy přerušení a realtime se vkládají vždy do Current (tím předbíhají ostatní v Next)
- „interaktivní“ vlákna se také vkládají do Current, ostatní do Next
- prioritní vkládání je řešeno indexem podle priority a frontou pro každou úroveň priority (přímé vkládání, bit ve slově indikuje, zda je fronta prázdná, lze jednou instrukcí otestovat, zda jsou všechny prázdné a která je první neprázdná) – obdobně O(1) plánovač v Linuxu.

Completely Fair Scheduler (CFS)
Princip: výpočetní kapacita systému ve formě volného strojového času procesorů je rozdělena spravedlivě mezi procesy podle jejich „priority“ podobně jako kapacita komunikační linky. Procesory s normální prioritou dostávají stejný díl, procesy s nižší prioritou se podělí o patřičně menší díl (poloviční). Na rozdíl od předchozích algoritmů není časové kvantum, ale granularita (jak moc může překročit „spravedlivý díl“).
Procesy, které čekají na přidělení procesoru, jsou uloženy v balancovaném binárním stromu (red/black) podle virtuálního času procesoru (vruntime). Proces, který dostal nejméně virtuálního času, je nejlevější ve stromu, je vyjmut ze stromu a dostane přidělen procesor jako první. Běží tak dlouho, dokud jeho virtuální čas nepřekročí čas nyní nejlevějšího ve stromu. Jeho virtuální čas se zvětší o (dobu běhu/váha) a je zařazen zpět do stromu. Nově spuštěný proces dostane počáteční virtuální čas rovný minimálnímu v systému (jinak by mohl dlouhodobě předbíhat všechny právě běžící).

Hlavní problém všech plánovačů –
jak identifikovat interaktivní procesy a dát jim větší prioritu pro lepší odezvu.
Co když se stane z interaktivního procesu proces výpočetně náročný a naopak?
Jak rychle na to plánovač zareaguje?
O(1) plánovač – složitá heuristika aktualizace dynamické priority
CFS – vruntime po probuzení z čekání nastaveno na minimum
ULE plánovač – poměr doby synchronního čekání a běhu, po čase se redukuje a připočítá znovu aktuální (historie + nové chování), vlákna překračující nastavený limit se považují za interaktivní a vkládají se do Current

Hodnocení plánovacích algoritmů
1. matematický model - systémy hromadné obsluhy
2. simulace
3. měření na reálném systému – monitorování

Markovské systémy M/M/1
- počet příchodů je nekonečný
- distribuce příchodů a doby obsluhy má exponenciální rozložení
- režim fronty je FIFO
- zkoumá se ustálený stav

PDF10

Uváznutí - čekání na událost, která nemůže nastat díky čekání
Kdy nastává uváznutí?
1. Přidělený prostředek může používat pouze jeden proces.
2. Proces, který má přidělené prostředky, se při alokaci dalších nevzdá přidělených prostředků, uvolní je až po ukončení.
3. Proces získává prostředky sekvenčně, oddělenými alokacemi.
4. Prostředek nemůže být preemptivně odebrán, proces uvolňuje prostředky explicitně.

Problém uváznutí:
1. Detekce - jak zjistit, které procesy v paralelním systému uvázly? (Stačí cyklus?)
2. Zotavení - jak se nejlépe dostat z uváznutí?
3. Prevence - jak se vyhnout uváznutí

Přechody stavů systému:
1. požadavek (request)
2. přidělení (allocation)
3. uvolnění (release)

Typy prostředků:
SR - opakovaně použitelné - serially reusable
CR - jednorázově použitelné - consumable resources

Stav systému
• Stav systému reprezentuje stav alokace prostředků v systému.
• Stav systému je měněn procesy při požadavku, získání nebo uvolnění prostředku.
• Pokud není proces v daném stavu systému blokovaný (čeká na přidělení), může potenciálně změnit stav systému.
- proces P i může změnit stav systému S x (žádostí, přidělením, uvolněním) na některý stav z množiny stavů
- Def.: Proces P i je blokován v daném stavu S, pokud nemůže žádným způsobem změnit stav systému
- Def.: Proces P i uváznul v daném stavu S, pokud je blokován ve stavu S a bez ohledu na následující změny stavu systému zůstává stále blokovaný
- Def.: Stav S je stavem uváznutí, pokud existuje proces P i uváznutý ve stavu S.

Princip prevence uváznutí: omezení přechodů mezi stavy systémů tak, aby všechny dostupné stavy nebyly stavem uváznutí.

SR prostředky
1. Prostředek se skládá z konstantního počtu stejných jednotek
2. Jednotka prostředku je buď volná nebo přidělená
3. Proces může uvolnit jednotku, pokud ji má přidělenu

O1: Prostředek může být přidělen do max. kapacity
O2: Proces nesmí žádat o více než je kapacita prostředku

1. Požadavek
Pokud proces p i nemá žádné požadavky, pak může žádat k jednotek prostředku R j
2. Přidělení
Pokud má proces p i požadavek na k jednotek prostředku R j a požadavek je uspokojitelný
3. Uvolnění
Pokud nemá proces p i žádný požadavek a má přiděleno k jednotek prostředku R j , může uvolnit l jednotek prostředku

pozret wiki.fituska