INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA STROJNÍ

OPERAČNÍ SYSTÉMY

ZABLOKOVÁNÍ (DEADLOCK)

doc. Dr. Ing. Oldřich Kodym

Ostrava 2013

© doc. Dr. Ing. Oldřich Kodym

© Vysoká škola báňská – Technická univerzita Ostrava

ISBN 978-80-248-3053-7

Tento studijní materiál vznikl za finanční podpory Evropského sociálního fondu (ESF) a rozpočtu České republiky v rámci řešení projektu: CZ.1.07/2.2.00/15.0463, MODERNIZACE VÝUKOVÝCH MATERIÁLŮ A DIDAKTICKÝCH METOD

OBSAH

6. ZABLOKOVÁNÍ (DEADLOCK)

OBSAH KAPITOLY:

Model systému, charakteristiky deadlock.

Metody obsluhy deadlock.

Předcházení a zamezení zablokování.

Detekce zablokování.

Náprava zablokování.

MOTIVACE:

Chod operačního systému využívá mnoha standardních mechanizmů známých z jiných oblastí řízení i běžného života. V multiprogramovém prostředí si mohou různé procesy konkurovat v získání konečného počtu zdrojů. Proces požaduje zdroje; jestliže tyto zdroje nejsou v danou chvíli dostupné, je proces přesunut do stavu čekající. Může se stát, že čekající proces svůj stav již nikdy nezmění, protože zdroje, na které čeká, drží jiné čekající procesy. Snad nejlepší ilustrací zablokování můžeme nalézt v jednom kansaském zákonu ze začátku století: Jestliže se k sobě vzájemně blíží dva vlaky na křížení kolejí, oba musí zcela zastavit a žádný se nesmí opětovně rozjet, dokud druhý neodjede.

CÍL:

Zablokování – model a charakteristika

Způsoby předcházení a zamezení vzniku zablokování

Detekce zablokování a jeho náprava

1. ÚVOD

V multiprogramovém prostředí si mohou různé procesy konkurovat v získání konečného počtu zdrojů. Proces požaduje zdroje; jestliže tyto zdroje nejsou v danou chvíli dostupné, je proces přesunut do stavu *čekající*. Může se stát, že čekající proces svůj stav již nikdy nezmění, protože zdroje, na které čeká, drží jiné čekající procesy. Tato situace je nazývána *deadlock (zablokování)*. Částečně jsme se o této problematice zmínili v předcházející kapitole v souvislosti se semafory.

Snad nejlepší ilustrací zablokování můžeme nalézt v jednom kansaském zákonu ze začátku století: Jestliže se k sobě vzájemně blíží dva vlaky na křížení kolejí, oba musí zcela zastavit a žádný se nesmí opětovně rozjet, dokud druhý neodjede.

V této kapitole si ukážeme metody, které může užít operační systém pro řešení deadlocku. Poznamenejme přesto, že řada současných operačních systémů prevenci deadlocku neprovádí. Tyto mechanismy budou zřejmě doplněny časem, až bude problematika deadlocku aktuálnější. Některé vývojové trendy k této situaci jistě povedou: čím dál větší množství procesů na počítači, čím dál větší množství zdrojů (včetně více CPU) atd.

2. MODEL SYSTÉMU

Systém obsahuje konečný počet zdrojů, které mohou být distribuovány mezi konkurující si procesy. Zdroje jsou rozdělené do tříd dle typu a každá třída obsahuje určitý počet identických instancí. Takovou třídou je např. paměťový prostor, cykly CPU, soubory a I/O zařízení. Jestliže má systém 2 CPU, potom zdroj typu *CPU* má 2 instance. Stejně tak, např. zdroj typu *tiskárna* může mít 5 instancí apod.

Jestliže proces požaduje instanci určitého typu zdroje, jeho žádost uspokojí alokace jakékoliv instance daného typu. Pokud tomu tak není, instance nejsou identické a třída zdroje není definována korektně. Např. nechť systém obsahuje dvě tiskárny. Tyto dvě tiskárny mohou být zahrnuty do téže třídy, jestliže nezáleží na tom, která tiskárna tiskne který tisk. Přesto, jestliže jedna tiskárna je v devátém poschodí a druhá v přízemí, je zřejmé, že tyto dvě tiskárny navzájem ekvivalentní nejsou.

Proces musí o zdroj požádat před jeho užitím a musí ho po užití uvolnit. Proces může požadovat tolik zdrojů, kolik jich vyžaduje uskutečnění zpracovávané úlohy. Počet požadovaných zdrojů většinou nepřekročí celkový počet zdrojů v systému. Proces nemůže požadovat tři tiskárny, jsou-li v systému pouze dvě.

Při běžném vykonávání operace proces může užít zdroj pouze dle následující sekvence:

- **Dotaz**: Jestliže nemůže být dotaz vyplněn okamžitě (např. proto, že daný zdroj právě využívá jiný proces), potom dotazující proces musí čekat, než zdroje může nabýt.
- **Užití**: Proces může pracovat se zdrojem (např. je-li zdrojem tiskárna, proces na ni může tisknout)
- **Uvolnění**: Proces uvolní zdroj k užití procesy jinými.

Dotaz a uvolnění jsou systémová volání. Příkladem mohou být volání **dotaz** a **uvolnění zařízení**, **otevření** a **uzavření souboru**, **alokování** a **uvolnění paměti**. Dotaz a uvolnění jiných zdrojů mohou být provedeny pomocí operací *cekej* a *signal* na semaforu. Pročež OS před každým užitím zdroje kontroluje, zda daný proces vznesl dotaz na tento zdroj a jestli mu byl zdroj přidělen. Jedna systémová tabulka vede evidenci o každém zdroji, zda je volný nebo alokovaný, případně kterým procesem. Jestliže se proces dotazuje na zdroj, který je právě alokovaný, může být zařazen do fronty čekajících na tento zdroj.

Říkáme, že množina procesů je ve stavu zablokována, jestliže každý proces v množině čeká na událost, kterou může vyvolat pouze jiný proces z téže množiny. Událost, se kterou se zde budeme setkávat zejména je obsazení zdroje a jeho uvolnění. Zdrojem může být buď fyzické zařízení (tiskárna, magnetopásková mechanika, paměťový prostor a čas CPU) nebo logické zařízení (např. soubor, semafor a monitor). Ovšem i jiné typy událostí mohou vést k deadlocku (např. IPC facility).

K ilustraci deadlocku předpokládejme systém se třemi magnetopáskovými mechanikami. Nechť v něm existují 3 procesy a každý užívá jednu mechaniku. Jestliže za daného stavu každý proces vznese dotaz na další mechaniku, všechny tři se dostanou do deadlocku. Každý z nich čeká na událost „magnetopásková mechanika je uvolněna“, která může být vyvolána pouze jiným čekajícím procesem. Tento příklad ilustruje deadlock vyvolaný procesy ucházejícími se o týž typ zdroje.

Deadlock může být také vyvolán různými typy zdrojů. Uvažujme např. systém s jednou tiskárnou a jednou magnetopáskovou mechanikou. Nechť proces Pi užívá magnetopáskovou mechaniku a proces Pj tiskárnu. Jestliže Pi požádá o tiskárnu a Pj o páskovou mechaniku nastává deadlock.

3. CHARAKTERISTIKA DEADLOCKU

Je zřejmé, ze deadlock je nežádoucí. V deadlocku procesy nemohou dokončit své spuštění a systémové zdroje jsou obsazené a ani jiné procesy nemohou tyto zdroje alokovat. Než budeme diskutovat různé metody řešení deadlocku, popišme rysy, které ho charakterizují.

3.1 Nutné podmínky

Deadlock může nastat, jestliže jsou v systému současně splněny následující 4 podmínky:

- **Vzájemná jedinečnost**: Minimálně jeden zdroj v systému musí existovat jako nesdílitelný – tj. pouze jeden proces může v danou chvíli užívat tento zdroj. Jestliže se na zdroj tou dobou dotazuje jiný proces, je pozdržen do doby, než se zdroj uvolní.
- **Drží a čeká**: Musí existovat proces, který užívá alespoň jeden zdroj a čeká na přidělení dalšího zdroje, který je užíván jiným procesem.
- **Nepreemptivnost**: Zdroj nesmí byt preemptivně plánován, tzn. zdroj může být uvolněn pouze dobrovolně po ukončení úlohy procesem, který ho na požádání dostal.
- **Kruhové čekání**: Musí existovat množina {P0, P1, ..., Pn-1} čekajících procesů taková, že P0 čeká na zdroj užívaný procesem P1, P1 čeká na zdroj užívaný procesem P2, ..., Pn-1 čeká na zdroj užívaný procesem Pn a Pn čeká na zdroj užívaný procesem P0 (tj. Pi čeká na zdroj užívaný procesem Pi+1 mod n).

Zdůrazněme, že pro vznik deadlocku musí nastat všechny uvedené podmínky současně, přičemž podmínka kruhového čekání implikuje podmínku drž a čekej. Uvedené podmínky tedy nejsou na sobě navzájem nezávislé. Jak však ještě uvidíme, je výhodné definovat podmínky všechny čtyři.

3.2 Graf alokace zdrojů

Deadlock může být precizně popsán pomocí orientovaného grafu zvaného graf alokace zdrojů. Tento graf sestává z množiny vrcholů V a množiny hran H. Množina vrcholů sestává ze dvou podmnožin:

- P = {P1, P2, ..., Pn} – množina všech procesů v systému a
- R = {R1, R2, ..., Rm} – množina všech zdrojů v systému.

Orientovaná hrana od procesu Pi ke zdroji Rj je zaznamenána jako Pi -> Rj a znamená, že proces Pi požaduje instanci zdroje Rj a čeká na její přidělení. Orientovaná hrana od zdroje Rj k procesu Pi je zaznamenána jako Rj -> Pi a znamená, že instance zdroje Rj byla alokována procesu Pi. Orientovaná hrana Pi -> Rj se nazývá *hrana dotazu* a orientovaná hrana Rj -> Pi *hrana přidělení*.

Graficky prezentujme každý proces Pi jako kruh a každý zdroj Rj jako obdélník. Protože každý zdroj Rj může sestávat z více instancí, každou instanci definujme jako tečku uvnitř obdélníku. *Hrana dotazu* tedy dle uvedených předpokladů ukazuje na celý obdélník Rj, zatímco *hrana přidělení* musí ukazovat na konkrétní tečku uvnitř obdélníku.

Pokud se proces Pi dotazuje na instanci zdroje Rj, je do grafu alokace zdrojů přidána hrana dotazu Pi R Rj. Když je dotaz vyplněn, hrana dotazu je *okamžitě* transformována na hranu přidělení Rj R Pi. V okamžiku, kdy proces Pi uvolní zdroj Rj je hrana přidělení smazána.

Na obr. 1 je zobrazena následující situace.

Množiny *P*, *R* a *H*:

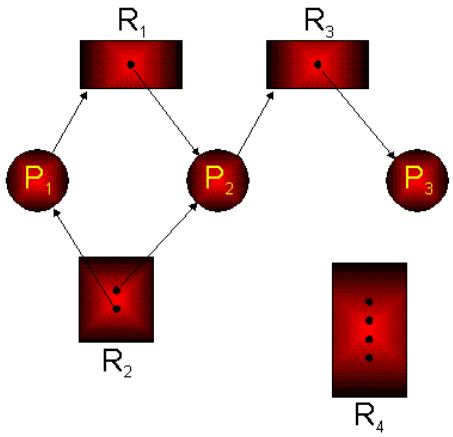
```
P = {P1, P2, P3}
R = {R1, R2, R3, R4}
H = { P1 -> R1, P2 -> R3, R1 -> P2, R2 -> P2, R2 -> P1, R3 -> P3}
```

Instance zdrojů:

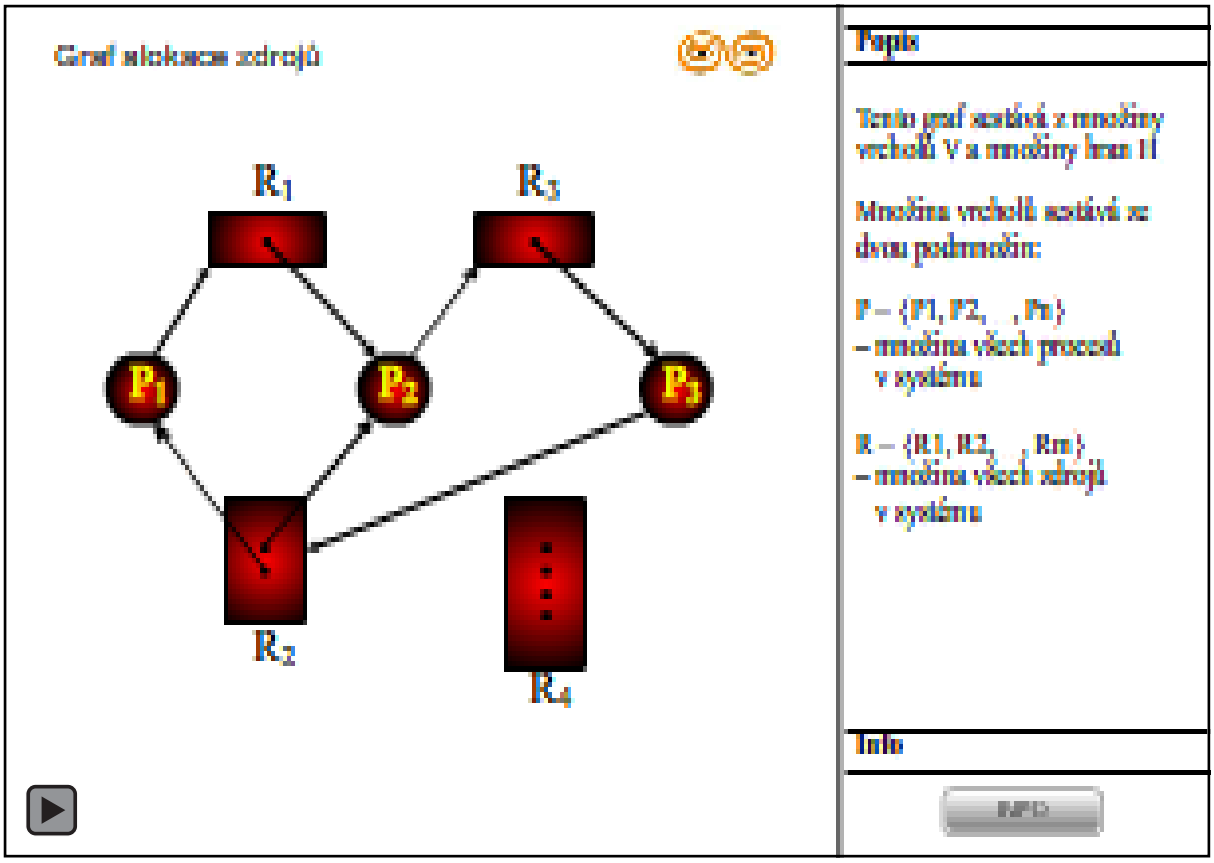
- Jedna instance zdroje R1
- Dvě instance zdroje R2
- Jedna instance zdroje R3
- Tři instance zdroje R4

Stav procesu:

- Proces P1 drží jednu instanci zdroje R2 a čeká na instanci zdroje R1.
- Proces P2 drží jednu instanci zdroje R1 a R2 a čeká na instanci zdroje R3.
- Proces P3 drží jednu instanci zdroje R3.



Obrázek 1 - Graf alokace zdrojů



Animce 6.1 Graf alokace zdrojů

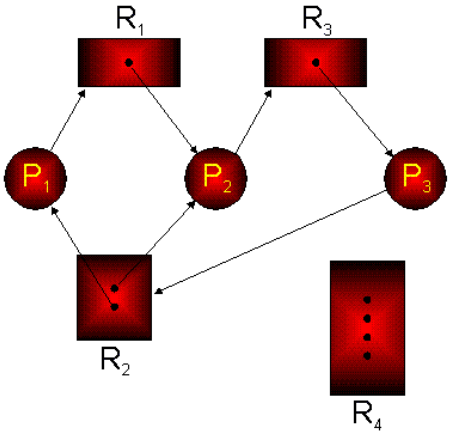
Díky definici grafu alokace zdrojů je velmi snadné ukázat na případný deadlock v systému. Pokud graf neobsahuje žádnou kružnici, není žádný proces zablokován. Na druhé straně, jestliže graf kružnice obsahuje, deadlock může nastat.

Pokud by každý typ zdroje obsahoval právě jednu instanci, kružnice v grafu by automaticky znamenala deadlock. Každý proces v takové kružnici by byl zablokován. V tomto případě je kružnice v grafu nutnou i postačující podmínkou deadlocku.

Jestliže každý zdroj obsahuje několik instancí, potom kružnice v grafu nemusí nutně znamenat deadlock. Zde je kružnice pro vznik deadlocku podmínkou nutnou, ne však postačující.

Pro ilustraci tohoto mechanismu se vraťme k obr. 1. Nechť dále proces P3 požádá o instanci zdroje R2. Protože žádná instance tohoto zdroje není volná, je hrana P3 -> R2 přidána do grafu. V tomto případě existují v grafu dvě kružnice (viz obr. 2)

```
P1 -> R1 -> P2 -> R3 -> P3 -> R2 -> P1
P2 -> R3 -> P3 -> R2 -> P2
```



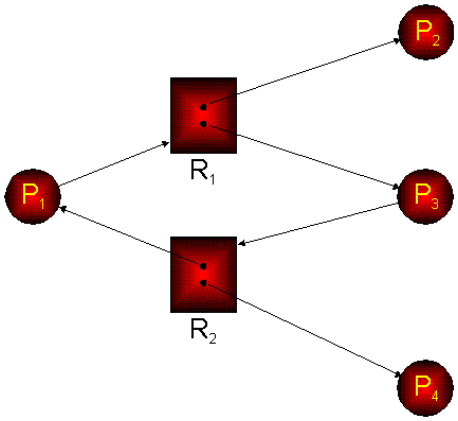
Obrázek 2 - Graf alokace zdrojů s deadlockem

V tomto případě jsou procesy P1, P2 a P3 zablokovány. Proces P2 čeká na zdroj R3, který drží proces P3. Proces P3 čeká na zdroj R2, jehož dvě instance drží procesy P1 a P2 a proces P1 čeká na proces P2, který drží zdroj R1.

Nyní si obraťme pozornost ke grafu alokace zdrojů na obr. 3. V tomto grafu se také vyskytuje kružnice

```
P1 -> R1 -> P3 -> R2 -> P1
```

přesto ovšem k deadlocku nedochází. Proces P4 může totiž uvolnit instanci zdroje R2, ta může být přidělena procesu P3 a kružnice je přerušena.



Obrázek 3 - Graf alokace zdrojů s kružnicí bez deadlocku

Shrňme zjištěné poznatky: Pokud graf alokace zdrojů neobsahuje žádnou kružnici, potom v systému není žádný deadlock. Pokud v grafu kružnice je, v systému deadlock být může i nemusí. Tento závěr bude důležitý v okamžiku, kdy budeme řešit problém deadlocku v systému.

4. METODY OBSLUHY DEADLOCKU

Principiálně existuji 3 metody řešení deadlocku:

- Můžeme v systému užít protokol, který zajistí, že *nikdy* deadlock nenastane.
- Můžeme systému dovolit, aby deadlock nastal a potom ho vyřešit.
- Můžeme celý problém deadlocku ignorovat a předstírat, že k němu nikdy nedojde. Toto řešení zatím užívá většina OSu, včetně Unixu.

Upřesněme krátce každou metodu a v následujících kapitolách ukážeme konkrétní algoritmy.

K zajištění, že deadlock v systému nikdy nenastane, může systém užít buď schéma *deadlock–prevention* nebo schéma *deadlock–avoidance.*

Deadlock–prevention je množina metod, které zajišťují, že minimálně jedna z nutných podmínek (viz kapitola 6.3.1) nenastane. Tato metoda spočívá v omezení možnosti vytváření žádostí o zdroje.

Deadlock–avoidance naopak požaduje, aby OS měl k dispozici další rozšiřující informace o tom, které zdroje bude proces požadovat, a užíval jich za běhu procesu. Díky těmto rozšířeným informacím může systém u každé žádosti o přidělení zdroje rozhodnout, je-li ho možno bezpečně přidělit či nikoli. K bezpečnému vyřešení žádosti o přidělení zdroje potřebuje systém informace o volných instancích zdroje, o instancích alokovaných jiným procesům, o budoucích žádostech o instance tohoto zdroje a o budoucích uvolněních těchto instancí.

Nevyužívá-li systém buď schéma deadlock–prevention nebo schéma deadlock–avoidance obecně může dojít k deadlocku. V takovémto prostředí může systém provádět detekci možného deadlocku nastane-li, pak ho vyřešit.

Jestliže systém nemá žádné zabezpečení proti deadlocku, a tedy neprovádí žádný mechanismus pro detekci a odstranění deadlocku, tehdy může nastat situace, kdy se systém dostane do deadlocku a nemá žádný mechanismus k tomu, aby zjistil, co se stalo. Takovýto nedetekovaný deadlock vede ke snížení výkonu systému, protože zdroje jsou drženy procesy, které nemohou byt ukončeny, a tak se další a další procesy, které nárokují blokované zařízení, dostávají do deadlocku. Nakonec může systém zcela přestat fungovat a vyžaduje manuální restartování.

Ačkoliv se poslední zmiňovaná metoda nemusí zdát právě optimální pro řešení problému deadlocku, je přece jen užita některými operačními systémy. V mnoha systémech deadlock nastává velmi zřídka (řekneme jednou za rok) a tudíž se nevyplatí drahá režie jeho obsluhy. Deadlock avoidance, deadlock prevention či deadlock detection musí být spuštěny neustále. Navíc existují situace, kdy systém zamrzne, aniž by byl v deadlocku. Představme si například proces spuštěný v reálném čase s maximální prioritou (nebo jakýkoliv proces v systému s nepreemptivním plánováním), který nikdy nevrátí řízení operačnímu systému.

5. DEADLOCK PREVENTION

Jak jsme již uvedli v kapitole 6.3.1, má-li nastat deadlock, musí být splněny současně všechny čtyři tam uvedené podmínky. Zajistíme-li, aby alespoň jedna z nich nenastala, provedeme dostatečnou prevenci deadlocku. Co je proto nutné?

5.1 Vzájemná jedinečnost

Podmínka vzájemné jedinečnosti musí být dodržena u nesdělitelných zdrojů. Např. tiskárnu nemohou najednou využívat dva procesy. Sdílené zdroje na druhé straně podmínku vzájemné jedinečnosti nevyžadují, a proto *nemohou* vyvolat deadlock. Jako ukázku takového zdroje můžeme označit soubor, určený pouze ke čtení. Jestliže z tohoto souboru chce číst současně více procesů, OS jim to může s klidem dovolit. Proces nikdy nemusí čekat na sdílitelný zdroj. V globálu ovšem není možné provádět prevenci deadlocku popřením podmínky vzájemné jedinečnosti, protože některé zdroje prostě sdílitelné nejsou.

5.2 Drží a čeká

K zajištění, aby v systému nikdy nenastala podmínka drží a čeká, musíme garantovat, že kdykoli požádá proces o zdroj, nesmí držet žádné jiné zdroje. Řešením problému drží a čeká, může být protokol, který umožňuje procesu alokovat všechny potřebné zdroje před tím, než bude spuštěn. Implementaci tohoto protokolu lze jednoduše provést tak, že systémovým voláním alokace zdrojů procesu dáme přednost před všemi ostatními.

Jiný protokol umožňuje procesu alokovat zdroje pouze tehdy, nemá-li aktuálně žádné v držení. V takovém případě může zdroje požadovat a užít. Před tím, než může požádat o další zdroje, však musí již alokované zdroje uvolnit.

Pro přiblížení rozdílu mezi těmito dvěma protokoly uvažujme proces, který kopíruje data z pásky na disk, soubor na disku uspořádá a vytiskne výsledek na tiskárně. Při užití prvního protokolu bude mít proces alokovanou tiskárnu po celou dobu jeho spuštění, ačkoli ji evidentně potřebuje až na konci. Druhá metoda umožňuje alokovat procesu pouze disk a pásku. Potom musí obě zařízení uvolnit a požádat o opětovné přiděleni disku spolu s tiskárnou. Po zkopírování souboru do tiskárny proces uvolní obě zařízení a ukončí se.

Tyto protokoly mají dvě hlavní nevýhody:

- Může dojít k *nízkému využití zdroje*, protože mnoho zdrojů může být alokováno a nevyužito po dlouhou dobu. V uvedeném příkladu při druhém protokolu můžeme uvolnit pásku a diskový soubor a potom opětovně požádat o soubor a tiskárnu pouze v případě, když máme jistotu, že naše data zůstala na disku v tom stavu, v jakém jsme je zanechali. Jestliže si jisti být nemůžeme, musíme na počátku požádat o všechny zdroje v obou případech.
- Může dojít k *umoření*. Proces, který požaduje nějaký populární zdroj, může čekat neomezeně dlouho, když nejméně jeden ze zdrojů, které požaduje, je neustále alokován jiným procesem.

5.3 Nepreemptivnost

Třetí nutná podmínka deadlocku je nepreemptivní plánování alokovaných zdrojů. K zajištění toho, že tato podmínka nikdy nenastane, můžeme užít následující protokol: Pokud proces, který má alokovaný nějaký zdroj, požaduje ještě další zdroj, nemůže zdroj alokovat ihned (tj. musí čekat), než budou všechny zdroje, které má přidělené preemptivně uvolněny. Tzn. všechny před tím alokované zdroje jsou implicitně uvolněné a přidané do seznamu zdrojů, na které proces čeká. Proces může být restartován pouze v případě, že může znovu nabýt svých starých zdrojů, stejně jako zdroje nového, o který původně žádal.

Jinak, jestliže proces požaduje zdroje, nejdříve zkontrolujeme, jestli jsou volné. Pokud ano, přidělíme mu je. Jestliže volné nejsou, zjistíme, nejsou-li alokovány procesům čekajícím na jiné zdroje. Pokud ano, požadované zdroje preemptivně uvolníme a přidělíme žádajícímu procesu. Jestliže zdroje nejsou volné, ani nejsou v držení čekajícího procesu, žádající proces musí čekat. Zatímco tento proces čeká, může být jím požadované zařízení preemptivně uvolněno, ovšem pouze v případě, že o to požádá jiný proces. Proces může být obnoven pouze v případě, že alokuje nové zdroje, o které žádal a případně znovu nabyde zdrojů, o které přišel, zatímco čekal.

Tento protokol je často používán u zdrojů, jejichž stav může být snadno uložen a potom opětovně obnoven (např. registry CPU nebo paměťový prostor). Nemůže být ovšem aplikován globálně na všechny zdroje (např. stěží na tiskárnu).

5.4 Cyklické čekání

Jedna z cest, jak zajistit, že nikdy nedojde k cyklickému čekání je definovat absolutní uspořádání všech typů zdrojů a vyžadovat, aby se procesy dotazovaly na zdroje ve vzestupné posloupnosti.

Nechť R = {R1, R2, …, Rm} je množina typů zdrojů. Přiraďme každému zdroji jedinečné celé číslo. Tato čísla nechť definují uspořádání na množině R. Definujeme tedy vzájemně jednoznačnou funkci F: R -> N, kde N je podmnožina množiny přirozených čísel. Obsahuje-li například množina typů zdrojů magnetopáskové mechaniky, pevné disky a tiskárny, funkce F může být definována následovně:

```
F(magnetopaskova mechanika) = 1,
F(pevny disk) = 5,
F(tiskarna) = 12.
```

Nyní můžeme definovat následující protokol prevence deadlocku:

Každý proces může požadovat zdroje systému pouze v uspořádané posloupnosti definované enumerace. Tzn. proces může nejdříve požádat o zdroj Ri. O další zdroj Rj může pak požádat pouze v případě, že F(Ri) < F(Rj). Jestliže požaduje více instancí téže třídy zdroje, musí tak učinit *jediným* dotazem. V souladu s uvedeným očíslováním zdrojů, požaduje-li proces současně páskovou mechaniku a tiskárnu, musí nejprve požádat o pásku a potom o tiskárnu.

Stejně tak můžeme jednoduše požadovat, aby kdykoliv se proces dotazuje na instanci zdroje Rj musel před tím uvolnit všechny zdroje Ri, pro které platí F(Ri) >= F(Rj).

Jestliže jsou užity oba dva předešlé protokoly, potom cyklické čekání nemůže nastat. Toto tvrzení lze dokázat sporem. Nechť {P0, P1, ..., Pn-1} je množina cyklicky čekajících procesů, kde Pi čeká na zdroj Ri, který je držen procesem Pi+1 mod n (je užito modulo indexování, takže proces Pn čeká na zdroj Rn, který drží proces P0). Potom, jestliže Pi+1 drží zdroj Ri a zároveň se dožaduje zdroje Ri+1, musí pro všechna i platit, že F(Ri) < F(Ri+1) => F(R0) < F(R1) < ... < F(Rn) < F(R0). Z tranzitivity potom plyne, že F(R0) < F(R0), což je SPOR.

Poznamenejme, že funkce F by měla být definována podle pořadí důležitosti zdrojů v systému. Např. magnetická páska bývá většinou užita před užitím tiskárny, takže by mělo platit F(magnetopásková mechanika) < F(tiskárna).

6. DEADLOCK AVOIDANCE

Algoritmus deadlock–prevention, diskutovaný v předcházející kapitole, předchází deadlocku omezením možnosti toho, jak mohou procesy žádat o zdroje. Tato omezení zajišťují, že nenastane minimálně jedna z nutných podmínek deadlocku. Možný negativní dopad těchto mechanismů na OS je ve snížení jeho výkonu a propustnosti.

Jiná metoda prevence deadlocku je požadovat více informací o tom, jak budou procesy vyžadovat zdroje. Např. v systému s jednou páskovou mechanikou a jednou tiskárnou můžeme říci, že proces P požádá nejprve o pásku a potom o tiskárnu nežli oba zdroje uvolní. Proces Q nechť naopak nejprve požaduje tiskárnu a potom pásku. S těmito znalostmi úplného pořadí žádostí a uvolněni zdrojů každým procesem můžeme pro každý dotaz rozhodnout, má-li proces čekat či nikoli.

Pro rozhodnutí, zda aktuální dotaz má být vyplněn nebo čekat bez nebezpečí následného deadlocku, jsou třeba informace o volných zdrojích systému, i o zdrojích aktuálně alokovaných různým procesům, které tyto procesy posléze uvolní.

Různé algoritmy se liší v množství a typech požadovaných informací. Nejjednodušší a velmi užitečný model požaduje, aby každý proces deklaroval *maximální počet* zdrojů každé třídy,

které může potřebovat. Jsou-li tyto informace dostupné pro každý proces, lze sestavit algoritmus, který zajistí, že se systém nikdy neocitne v deadlocku. To je cesta prevence deadlocku metodou deadlock–avoidance.

Algoritmus deadlock–avoidance dynamicky ohodnocuje stav alokace zdrojů, aby se ujistil, že nemůže nastat podmínka cyklického čekání. *Stav* alokace zdrojů je definován jako počet volných a alokovaných zdrojů a maximální počet požadavků procesu.

6.1 Bezpečný stav

Stav je bezpečný, jestliže systém může alokovat zdroje každému procesu (v mezích maximálního počtu) v nějakém pořadí a zároveň nenastane deadlock. Formálně, systém je v bezpečném stavu, jestliže existuje *bezpečná sekvence*. Sekvence procesu <P1, P2, ..., Pn> je bezpečná pro aktuální stav alokace, jestliže každému procesu Pi mohou být přiděleny požadované zdroje ze zdrojů volných + zdrojů držených procesy Pj, kde j < i. Za této situace, pokud zdroje požadované procesem Pi nejsou momentálně volné, potom Pi čeká, dokud nejsou ukončeny všechny procesy Pj. V momentě, kdy dokončeny jsou, Pi může dostat všechny požadované zdroje, splnit svou úlohu, zdroje uvolnit a skončit. Je-li proces Pi ukončen, může dostat všechny požadované zdroje proces Pi+1 atd. Pokud v systému neexistuje bezpečná sekvence, říkáme o něm, že není bezpečný.

Deadlock není bezpečný stav, nicméně ne všechny stavy, které nejsou bezpečné, musí být deadlockem. Stav, který není bezpečný, může však k deadlocku vést. Tak dlouho jak je systém v bezpečném stavu, může se vyvarovat nebezpečí včetně deadlocku. V nebezpečném stavu nemůže systém zabránit procesům, aby nevyžadovaly zdroje tak, aby k deadlocku došlo. Nebezpečný stav může vyvolat chování procesu.

Pro ilustraci si představme systém s 12 magnetickými páskami a třemi procesy P0, P1 a P2. Nechť proces P0 požaduje 10 pásek, P1 nejvýše 4 a proces P2 9 pásek. Dále nechť v čase t0 má proces P0 alokováno 5 pásek a procesy P1 a P2 po dvou páskách (tzn. v systému jsou ještě 3 volné pásky).

	Maximum potřebných pásek	Počet přidělených pásek
P0	10	5
P1	4	2
P2	9	2

V čase t0 je systém v bezpečném stavu. Sekvence procesů <P1, P0, P2> dodrží všechny podmínky bezpečnosti. Proces P1 alokuje pro sebe všechny potřebné pásky, vykoná svou úlohu, bude ukončen a vrátí pásky systému. V ten moment je v systému 5 volných pásek. Všechny je alokuje proces P0 a může dokončit svou úlohu. Když vrátí pásky systému, může být dokončen i proces P2.

I tento systém však může jednoduše přejít do nebezpečného stavu. Nechť v čase t1 proces P2 alokuje další pásku. V ten moment se systém dostává do nebezpečného stavu. Za této situace může získat všechny potřebné pásky pouze proces P1. Když P1 skončí a vrátí všechny pásky, jsou v systému dostupné pouze 4. Neboť P0 má alokovaných 5 pásek a potřebuje 10, snaží se získat dalších 5. Ty ale nejsou v systému dostupné, takže P0 musí čekat. Stejně tak proces P2 potřebuje ke svému ukončení dokonce dalších 6 pásek a nastává deadlock.

Chybou bylo povolení alokace jedné pásky procesem P2. Pokud bychom proces P2 zadrželi ve stavu *čekající*, dokud nebude některý z procesů P0 nebo P1 ukončen, k deadlocku by nedošlo.

Za přispění mechanismu bezpečného stavu můžeme definovat algoritmus prevence deadlocku. Základní myšlenka tohoto algoritmu je jednoduchá: zajistit, aby systém byl neustále v bezpečném stavu.

Na počátku systém v bezpečném stavu pochopitelně je. V okamžiku, kdy proces žádá zdroj, který je aktuálně volný, musí systém rozhodnout, zda procesu zdroj přidělí, nebo ho nechá čekat. Zdroj může být procesu přidělen pouze v případě, že systém i po přidělení zůstane v bezpečném stavu.

Je zřejmé, že tento mechanismus, kdy proces mnohdy musí čekat na zdroj, který je aktuálně volný může vést ke snížení výkonu systému.

7. DETEKCE DEADLOCKU

Jestliže systém nevyužívá žádného mechanismu pro prevenci deadlocku, může deadlock v systému nastat. V takovém prostředí musí systém provádět:

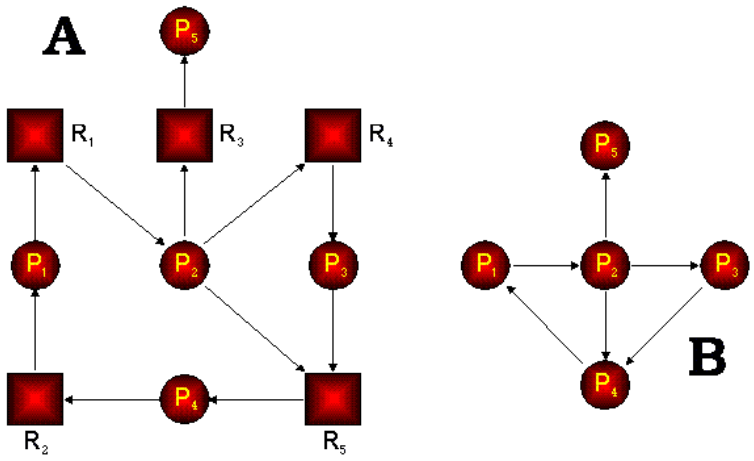
- algoritmus ohodnocující stav systému, který je schopen odhalit nastalý deadlock a
- algoritmus, který nastalý deadlock vyřeší.

V následující diskusi rozebereme tyto dva požadavky v systému s jednou instancí každého zdroje i v systému s více instancemi. Podotkněme, že odhalení a vyřešení deadlocku vyžaduje režii, která neobsahuje jen čas systému nutný k obsluze datových struktur a spuštění algoritmu detekce, ale také potenciální ztráty vzniklé při nápravě deadlocku.

7.1 Jedna instance v každé třídě zdroje

Jestliže všechny třídy zdrojů mají pouze jednu instanci, je možno k detekci deadlocku užít speciální tvar grafu alokace zdrojů, zvaný *graf čekání*. Tento graf můžeme získat z grafu alokace zdrojů odebráním všech uzlů zdrojů a kolabováním příslušných hran.

Precizněji, hrana *Pi* -> *Pj* v grafu čekání znamená, že proces *Pi* čeká na to, až proces *Pj* uvolní zdroje, které *Pi* potřebuje. Hrana *Pi* -> *Pj* v grafu čekání existuje pouze v případě, jestliže v analogickém grafu alokace zdrojů existuji hrany *Pi* -> *Rq* a *Rq* -> *Pj* pro zdroj *Rq*. Na obr. 4 jsou vidět jak graf alokace zdrojů i graf čekání popisující stejnou situaci.



Obrázek 4 - Stav systému: (A) Graf alokace zdrojů (B) Graf čekání

Stejně jako u grafu alokace zdrojů, deadlock je v systému tehdy a jen tehdy, obsahuje-li adekvátní graf čekání kružnici. Pro detekci deadlocku musí systém *udržovat* graf čekání a periodicky *spouštět algoritmus*, který hledá v grafu kružnici.

Algoritmus detekující kružnici v takovémto grafu vyžaduje sekvenci n^2 operací, kde n je počet vrcholů grafu.

7.2 Více instancí v každé třídě

Schéma užívající graf čekání není aplikovatelné pro odhalení deadlocku v systému s více instancemi v jednotlivých třídách. Popíšeme si algoritmus použitelný za této situace. Algoritmus užívá analogických datových struktury jako Bankéřův algoritmus:

- *Volny*: vektor délky *m* indikující počet volných instancí každého typu zdroje
- *Alokace*: matice typu [*n,m*] definující počet zdrojů každé třídy aktuálně alokovaných každému procesu.
- *Zadost*: matice typu [*n,m*] definující aktuální žádost každého procesu. Jestliže *Zadost[i,j] = k*, potom proces *Pi* požaduje dalších *k* instancí zdroje třídy *Rj*.

Relace menší než (<) nechť je definována stejně jako v kapitole 6.5.3. Pro zjednodušení zápisu opět definujme i-ty řádek matic *Alokace* a *Zadost* jako *Alokacei* a *Zadosti*. Algoritmus prostě prohledává všechny možné sekvence alokace zdrojů procesy, které mají být dokončeny.

1. Nechť *Prace* je vektor délky *m* a *Konec* je vektor délky *n*. Inicializujme *Prace = Volny* a pro *i* = 1, 2, …, *n* přiraď *Konec[i] := false* jestliže *Alokacei* <> 0, jinak *Konec[i] := true*.
2. Vyhledej *i* pro které platí obě nesledující podmínky:

   Konec[i] = false

   Zadosti L Prace

   Jestliže takové *i* neexistuje, přejdi na bod 4.
3. Prace := Prace + Alokacei

   Koneci := true

   Přejdi na bod 2.
4. Jestliže *Konec[i] = false* pro nějaké *i*, 1 <= *i* <=; *n*, potom systém je v deadlocku. Navíc, jestliže *Konec[i] = false*, potom je proces *Pi* zablokován.

Tento algoritmus požaduje sekvenci $m * n^2$ operací pro detekci deadlocku v systému.

Může se zdát podivné, že přidělíme požadované zdroje procesu *Pi* (v kroku 3) okamžitě poté, co zjistíme, že *Zadosti* <= *Prace* (v kroku 2b). Víme, že *Pi* není aktuálně v deadlocku (protože *Zadosti* <= *Prace*). Takže postavíme optimistický předpoklad, že *Pi* nebude vyžadovat další zdroje k dokončení svého úkolu, a že tedy brzy vrátí všechny přidělené zdroje zpět systému. Jestliže je tento náš předpoklad chybný, může později nastat deadlock. Ten bude následně detekován algoritmem detekce deadlocku.

Pro ilustraci tohoto algoritmu uvažujme systém s pěti procesy *P0* až *P4* a třemi zdroji typu *A*, *B*, *C*. Zdroj typu *A* má 7 instancí, zdroj typu *B* 2 a zdroj typu *C* 6 instancí. Nechť v čase *t0* je systém v následujícím stavu:

	Alokace			Max			Volny		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	0	0	0	0
P1	2	0	0	2	0	2			
P2	3	0	3	0	0	0			
P3	2	1	1	1	0	0			
P4	0	0	2	0	0	2			

V této situaci není systém v deadlocku. Skutečně, pokud spustíme výše uvedený algoritmus, potom sekvence < *P0, P2, P3, P1, P4*> vrací výsledek *Konec[i] = true* pro všechna *i*.

Uvažujme, že proces *P2* požádá o jednu další instanci zdroje třídy *C*. Matice *Zadost* potom vypadá následovně:

	Zadost		
P0	0	0	0
P1	2	0	2
P2	0	0	1
P3	1	0	0
P4	0	0	2

V takové situaci je systém v deadlocku. Můžeme uvolnit zdroje držené procesem *P0*. Počet volných zdrojů však i přesto nedostačuje k dokončení žádného jiného procesu. Deadlock nastal a obsahuje procesy *P1*, *P2*, *P3* a *P4*.

7.3 Užití algoritmu detekce deadlocku

Kdy má být spuštěn algoritmus detekce deadlocku? Odpověď na tuto otázku je podmíněna dvěma faktory:

- Jak *často* k deadlocku pravděpodobně dochází?
- *Kolik* procesů bude deadlockem postiženo v případě, že nastane?

Pokud k deadlocku dochází často, potom je třeba algoritmus detekce spouštět často. Zdroje alokované procesům v deadlocku budou zablokované, dokud nebude deadlock odstraněn. Počet procesů v deadlocku se může časem zvětšovat.

Deadlock může nastat pouze v případě, že některý proces vytvoří žádost, která nemůže být okamžitě vyřízena. Je možné, že tato žádost vyvolá řetězec čekajících procesů. Jedna extrémní možnost je spouštět algoritmus detekce vždy, když nemůže být žádost libovolného procesu okamžitě vyřízena. V takovém případě můžeme odhalit nejen celou množinu procesů v deadlocku, ale i jeho původce. (Ve skutečnosti každý deadlockovaný proces vyvolá jedno spojení v kružnici grafu alokace zdrojů, takže procesy vyvolají deadlock společně.)

Je-li v systému mnoho různých tříd zdrojů, může jedna žádost vyvolat více kružnic v grafu alokace zdrojů. Případná kružnice v grafu je uzavřena poslední žádostí a tedy „vyvolána“ jedním identifikovatelným procesem.

Samozřejmě, spouštěni algoritmu detekce deadlocku po každé žádosti může vyvolat značně nároky na výpočetní čas. Méně náročná alternativa je vyvolávat algoritmus v malých pravidelných intervalech (např. jednou za hodinu) nebo když využití CPU klesne např. pod 40 %. (Deadlock může ochromit systém do té míry, že výrazně poklesne využití CPU.)

Je-li algoritmus detekce spouštěn v libovolném čase, může v grafu alokace zdrojů existovat velké množství kružnic. Obecně pak nejsme schopni určit, který z množství zablokovaných procesů deadlock vyvolal.

8. NÁPRAVA DEADLOCKU

V okamžiku, kdy algoritmus detekce zjistí přítomnost deadlocku v systému, jsou různé alternativy další cennosti systému. Jedna možnost je informovat správce systému, že nastal deadlock a nechat ho odstranit deadlock manuálně. Druhou možností je ponechat automaticky nápravu na systému. Tato automatická náprava může být provedena dvojím způsobem. Nejjednodušší možností je ukončit jeden nebo více cyklicky čekajících procesů. Druhá možnost je preemptivně ukončit alokaci nějakého zdroje deadlockovaným procesem.

8.1 Ukončení procesu

K eliminaci deadlocku ukončením některého zablokovaného procesu je možno užít jednu ze dvou metod. V obou případech systém uvolní a získá všechny zdroje alokované ukončovaným procesům.

- **Ukončení všech deadlockovaných procesů**: Ukončit všechny procesy v kruhu cyklického čekání je sice jednoduché, ale mnohdy drahé. Vypočet nejednoho procesu mohl trvat i značně dlouho než se proces dostal do deadlocku, a celá tato výpočetní doba je najednou ztracena a výpočet procesu musí začít znovu.
- **Ukončení jednoho nebo více procesů, dokud není deadlock eliminován**: Tato metoda vyžaduje značnou režii, neboť po každém ukončeni každého procesu musí být spuštěn algoritmus detekce deadlocku ke zjištění, nachází-li se systém stále v deadlocku.

Poznamenejme, že ukončení procesu nemusí být vždy jednoduchá věc. Jestliže proces pravé provádí změnu nějakého souboru, jeho ukončení by mohlo zanechat soubor v nekonzistentním stavu. Podobná situace nastává, jestliže proces právě provádí tisk. V takovém případě musí systém provést reset tiskárny a nastavit ji do bezpečného stavu před tím, než povolí tisk další.

Jestliže je užita metoda ukončení pouze některých procesů, potom je třeba z množiny cyklicky zablokovaných procesů vybrat ten (nebo ty), které mají být ukončeny, aby se deadlock ze systému odstranil. To je politické rozhodnutí podobného charakteru jako plánovaní CPU. Celý problém se točí kolem toho, jak vybrat proces(y), jejichž ukončení bude pro systém nejméně náročné. Na konkretizaci této obecné formulace má vliv mnoho faktorů, např.:

- Jakou má proces prioritu
- Jak dlouho je již proces zpracováván systémem a jaká doba je ještě třeba k jeho dokončení
- Kolik zdrojů a jakého typu má proces alokováno (nedají-li se tyto zdroje preemptivně uvolnit)
- Kolik dalších zdrojů bude proces ještě potřebovat ke svému dokončení
- Kolik procesů je třeba ukončit
- Je proces interaktivní nebo dávkový

8.2 Preemptivní uvolnění zdroje

Při eliminaci deadlocku preemptivním uvolněním zdroje postupně uvolňujeme zdroje alokované zablokovaným procesům, přidělujeme je jiným, dokud není deadlock odstraněn.

Jestliže chceme tuto metodu užít, je třeba vyřešit tři problémy:

- **Vybrat oběť**: Který zdroj a který proces mají být preemptivně rozděleni? Stejně jako u ukončení procesu musíme určit nejméně náročné pořadí preemptivních uvolňování. Faktory náročnosti zde mohou být počet zdrojů, které drží zablokovaný proces a množství systémového času, které již spuštěný proces spotřeboval.
- **Zabezpečení**: Jestliže uvolníme preemptivně zdroj, musíme proces, který o něj přišel, uvést do bezpečného stavu. Tento proces přišel o zdroj nutný k jeho výpočtu. Tento nedostatek je třeba ošetřit a umožnit procesu další bezpečný výpočet. Protože obecně je velmi těžké stanovit, co to je bezpečný stav konkrétního procesu, je nejjednodušší provést absolutní návrat procesu, tj. ukončit proces a restartovat ho.
- **Umořeni**: Jak zajistit, aby nedošlo k umoření? To jest, jak zaručit, aby zdroje nebyly stále preemptivně uvolňovány a opětovně přidělovány témuž procesu? V systému, kde je výběr oběti veden pouze požadavky maximální efektivity se může stát, že bude obětován stále

jeden a tentýž proces. Tento proces ovšem nikdy nedokončí svou úlohu a nastává umoření procesu. Je tedy třeba zajistit, aby byl proces obětován pouze konečně krát. Výhodnou cestou je zařadit počet navrácení procesu do faktorů, podle kterých je vybírán obětovaný proces.

7. KOMBINOVANÝ PŘÍSTUP K ŘEŠENÍ DEADLOCKU

Systémoví programátoři namítají, že žádná ze základních metod řešeni deadlocku (prevention, avoidance, detection) není sama vhodná pro řešení všech problémů alokace zdrojů v operačním systému. Jedna z možností je kombinovat tyto tři základní metody a pro každou třídu zdrojů užít metodu optimální.

Tento přístup je založen na myšlence, že zdroje mohou být rozděleny do hierarchicky uspořádaných tříd. Technika uspořádání zdrojů je užita na třídy. Potom může být pro každou třídu zdrojů užita nejpřijatelnější technika řešení deadlocku.

Je jednoduché ukázat, že pokud je v systému aplikována tato strategie, není vystaven problému deadlocku. Deadlock nemůže postihnout více než jednu třídu zdrojů, protože je užita technika uspořádání zdrojů a každá třida má asociován algoritmus obsluhy deadlocku.

K ilustraci popisovaného přístupu užijeme systém obsahující nesledující třídy zdrojů:

- **Interní zdroje**: zdroje užívané systémem (např. process control block)
- **Centrální paměť**: paměť vyhrazená pro uživatelské úlohy
- **Zdroje procesu**: aplikovatelné zdroje (např. pásky, tiskárny apod.) a soubory
- **Swapovací prostor**: úložný prostor pro uživatelské procesy

Kombinované řešeni deadlocku pro výše uvedené pořadí tříd zdrojů užívá následujících postupů pro jednotlivé třídy:

- **Interní zdroje**: Lze užít *deadlock prevention* popření podmínky cyklického čekání užitím uspořádání tříd zdrojů, nedochází k rozhodování mezi více žádostmi v jednom okamžiku
- **Centrální paměť**: Lze užít *deadlock prevention* užitím preemptivní prevence, protože každá úloha může být vyswapována z paměti a tato paměť může být preemptivně přidělena jiné úloze.
- **Zdroje procesu**: Lze užít *deadlock avoidance*, neboť potřebné rozšiřující informace je možno získat.
- **Swapovací prostor**: Lze užít *předběžné přidělení (preallocation)*, protože maximální požadavky jsou předem známy.

Tento příklad ilustruje, jak mohou být základní metody obsluhy deadlocku kombinovány v uspořádané struktuře zdrojů, aby bylo dosaženo efektivního řešení deadlocku.