Architektury (= struktura) OS

Zde si popíšeme nejčastější možnosti, může být OS strukturován.

* monolitický OS - většina starších systémů (UNIX, VMS, OS/360, MS DOS)
  jeden spustitelný soubor, který běží na holém počítači. Uvnitř OS
  existují moduly pro jednotlivé funkce (fs, proc...), řízení se
  předává voláním podprogramů.

Většina CPU 2 režimy: privilegovaný (režim jádra - všechny instrukce
dovoleny), uživatelský (problémový - I/O a některé další instrukce
zakázány, CPU je neprovede).

Vyvolání služby systému:

 - parametry se uloží na určené místo (registry, zásobník)
 - vyvolá se speciální instrukce, která vyvolá obslužnou proceduru
   v jádře a zároveň se přepne do privilegovaného režimu
 - OS zjistí která služba je vovolána, převezme parametry, provede
   službu
 - vrátí se zpátky do programu, zároveň přepne CPU zpět do
   uživatelského režimu

Problém: monolitický OS je ”jedna velká hromada”, proto pokusy různě
strukturovat.

* vrstvený
  - hierarchie procesů - každá vyší úroveň poskytuje abstraknější
    virtuální stroj; např. THE na úrovni 0 virtualizace CPU (procesy
    nad úrovní 0 se nemusely zabývat počtem CPU), 1 virtualizace
    paměti... (logicky THE, s podporou HW OS MULTICS)
  - funkční hierarchie - např. správa procesů vs. správa paměti; Pilot

S HW podporou - nižší vtrsva volána z vyšší podobným způsobem jako
systémové volání, nelze obcházet.

* model klient-server (systémy založené na mikrojádře) - moderní OS (OSF/1,
  MkLinux, QNX, Hurd)
  - vlastní jádro má na starosti pouze nejdůležitější nízkoúrovňové
    funkce (předávání zpráv, často obsluhu přerušení, nízkoúrovňovou
    správu procesů, někdy vstupy/výstupy)
  - pouze mikrojádro v privilegivaném režimu
  - zbytek (= většina) OS beží jako samostatné procesy mimo jádro
    (systém souborů, ...), běží v uživatelském režimu

Nejčastěji v moderních distribuovaných systémech - klienti a servery
mohou komunikovat po síti.

* objektově orientovaná struktura - iAPX 432, částečně Windows NT
  - systém je množina objektů
  - capability = odkaz na objekt + množina práv definujících operace
  - jádro si vynucuje tento abstraktní pohled

Existuje ještě celá řada možností jak strukturovat OS, ale většina OS
monolitická nebo založených na mikrojádře.

Procesy

* nejdůležitější koncepce v OS: abstrakce běžícího programu
* vše ostatní v OS s procesy souvisí => je zapotřebí se s tím seznámit
  co nejdříve
* předpokládáme systémy kde může běžet ”několik procesů souběžně”
* dvě možnosti:
 - skutečný paralelismus - pro každý proces je k dispozici CPU
 - pseudoparalelismus - CPU rychle přepíná mezi procesy a vytváří tím

iluzi, že procesy jsou vykonávány paralelně (v systémech se sdílením času)

Proces jako abstrakce

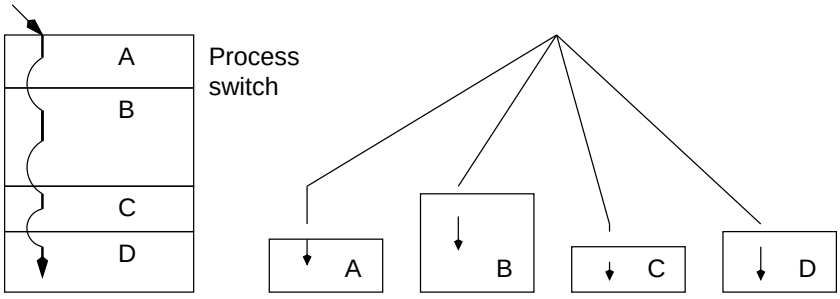
* všechen SW běžící na počítači (včetně většiny OS) organizován jako _množina_sekvenčních_procesů_, krátce jen _procesů_
* proces = běžící program včetně obsahu čítače instrukcí, registrů, proměnných; běží ve vlastní paměti
* koncepčně má každý proces vlastní _virtuální_CPU_

Ve skutečnosti reálný procesor přepíná mezi procesy, ale pro porozumění systému je snazší představa množiny procesů běžících (pseudo)paralelně.

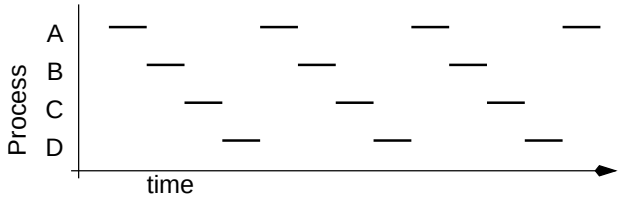
Příklad:

Čtyři procesy, každý z nich má vlastní ”bod běhu” (tj. vlastní čítač instrukcí), každý běží nezávisle na ostatních.

Na obr. (a) pseudoparalelní běh, (b) paralelní běh čtyř procesů, používáme jako koncepční model pro nezávislé sekvenční procesy.



Při pseudoparalelním běhu je v jednu chvíli aktivní pouze jeden proces. Po nějakém čase se OS rozhodne pozastavit běh procesu a spustit běh dalšího (např. proto, že proces běžel ”dlouho”). Po dostatečně dlouhém čase všechny procesy vykonají část své činnosti.



Přirovnání - kuchař peče koláče, má recept (kód programu) a potřebné ingredience (vstupní data). Přijde šéf a chce udělat minutku; kuchař si založí kde skončil s koláčem (uloží stav aktuálního procesu) a začne připravovat minutku podle příslušného receptu.

Pozor, rychlost běhu procesu není konstantní! Většinou není ani reprodukovatelná.

Procesy _nesmějí_ mít vestavěné předpoklady o časování; např. čtení pásky v jednoprogramovém systému: spuštění, čekací smyčka (rozjezd), čtení prvního záznamu. V multiprogramovém systému - pokud CPU zatím přepnut na jiný proces, bude záznam již za čtecí hlavou.

Procesy také _neběží stejně rychle_; příchod události s vysokou prioritou (příchod šéfa).

Rozdíl mezi programem a procesem:

* program: v analogii recept, algoritmus vyjádřený ve vhodné podobě
* proces: aktivita, v analogii vaření; má program, vstup, výstup a
  stav (analogie: recept, ingredience, koláč, stav).

Stavy procesu

Příklad:

```
$ ls -l | more
```

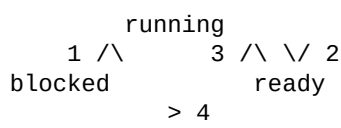
Příkaz ls vypíše adresář, příkaz more zobrazí jednu obrazovku a čeká na vstup uživatele. Na začátku bude ”more” připraven běžet, ale nemá žádný vstup => musí se {zablokovat} dokud vstup nedostane.

* blokování procesu - proces nemůže pokračovat, protože čeká na zdroj
  (vstup, zařízení, paměť) které zatím není dostupné

Proces může také být koncepčně připraven pokračovat, ale CPU vykonává jiný proces - odlišná situace: v prvním případě nemůže logicky pokračovat (nemůže zpracovat vstup který ještě neexistuje).

=> proces může být ve třech stavech:

1. běžící (running) - skutečně využívá CPU
2. připraven (ready, runnable) - dočasně pozastaven, aby jiný proces
   mohl běžet
3. blokován (blocked; někteří čekající = waiting) - neschopný běhu,
   dokud nenastane externí událost



 1. Proces se zablokuje, protože čeká na zdroj (zablokování provede
    systém např. pokud proces chce číst z klávesnice a není žádný vstup)
 2. Proces pozastaven, plánovač (= část OS která vybírí procesy) vybere jiný
    proces
 3. Plánovač vybere tento proces
 4. Zdroj je dostupný.

=> můžeme mít představu o systému - jádro OS obsahuje plánovač, nad OS řada procesů, střádají se o CPU.

Implementace procesu

Zatím zjednodušeně - jak se udělá, že každý proces má vlastní paměť apod. se dozvíme postupně.

OS udržuje tabulku nazývanou {tabulka procesů} - každý proces položku obsahující informace o stavu procesu. Konkrétní obsah se liší mezi systémy, ale většina obsahuje pole týkající se:

* správy procesů
 - identifikátor procesu (řetězec nebo číslo)
 - stavový vektor
  . obsah registrů CPU (někdy jen koncepčně)
  . ukazatel na další intrukci (obsah čítače instrukcí)
  . stav CPU (Program Status Word, PSW)
  . ukazatel zásobníku
 - stav procesu (běžící, připraven, blokován)
 - priorita
 - odkazy na rodiče a potomky
 - účtovací informace
  . čas spuštění procesu
  . čas CPU spotřebovaný procesem
* správy paměti

```
 - (popis paměti = ukazatel, velikost, přístupová práva)
   . úsek paměti s kódem (instrukcemi) programu
   . data (hromada - Pascal - new/release, C - malloc/free)
   . zásobník (návratové adresy, parametry fcí a procedur, lokální proměnné)
* systému souborů
 - prostředí
   . pracovní adresář, ...
 - otevřené soubory
   . způsob otevření - pro čtení / zápis
   . pozice v otevřeném souboru
```

Položce popisující proces se říká PCB (Process Control Block).

Jak probíhá přepnutí procesu - viz stará cvičení:

* systém nastaví časovač, provádí pravidelně přerušení (přerušení může
  přijít také od I/O zařízení, např. disku po dokončení operace)
* na předem definovaném místě je adresa obslužného podprogramu
  přerušení
* po příchodu přerušení CPU:
 - uloží PC do zásobníku
 - načte do PC adresu obslužného podprogramu přerušení
* vyvolaná obslužná procedura přerušení:
 - uloží obsah registrů do zásobníku
 - nastaví nový ”prozatimní” zásobník
* plánovač nastaví proces jako ready, vybere nový (= ”nejdůležitější”)
  proces
* ”přepnutí kontextu”
 - nastaví mapu paměti nového procesu
 - nastaví zásobník
 - načte obsah registrů
 - provede ”návrat” - RET = do PC adresu ze zásobníku

Poznámka: Obsah registrů může být v PCB i jen koncepčně.

Jak se procesy plánují si povíme téměř až na konci povídání o
procesech.

Programové konstrukce pro vytváření procesů

OS podporující koncepci procesu musí poskytovat způsob, jak potřebné
procesy vytvořit.

* nejjednodušší systémy: všechny potřebné procesy jsou spuštěny při
  startu systému (zapouzdřené systémy)
* statické: proces obsahuje deklaraci pevné množiny podprocesů, všechny
  podprocesy jsou spuštěny při spuštění procesu
  (implementace např. tabulkou - v procesu tabulka podprocesů)
* dynamické: procesy mohou vytvářet potomky dynamicky
  (implementace - volání služby systému ”vytvoř proces” apod.)

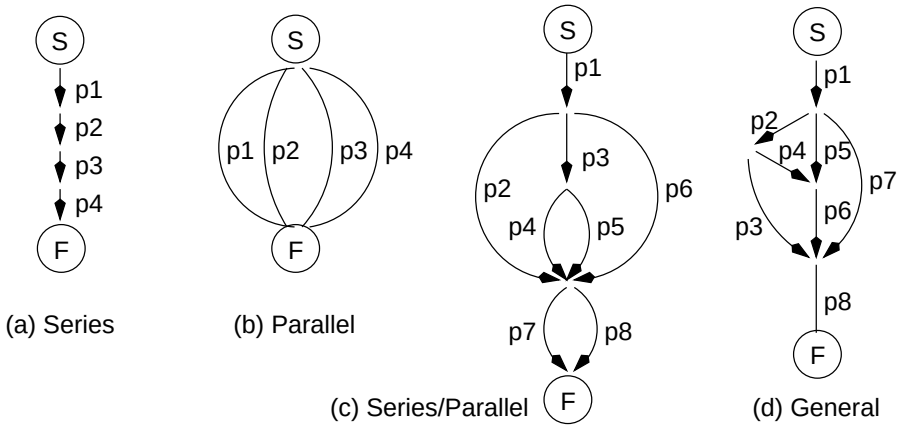
Například v UNIXu existuje služba fork(), která vytvoří identickou
kopii procesu; po vyvvolání fork() pokračuje starý proces (= rodič)
paralelně s novým procesem (= potomkem).

Precedenční grafy procesů

* grafy se hodí pro popis - konstrukce pro vytváření/rušení procesu
  musí být schopna vyjádřit různé {precedenční relace mezi procesy}

Za předpokladu, že systém má společný start a konec procesů možno
znázornit pomocí {precedenčního grafu procesů} (process flow graph):

* acyklický orientovaný graf
* běh procesu p_i je vyjádřen orientovanou hranou grafu
* vztahy mezi procesy (sériové nebo paralelní spojení)
  znázorněny spojením hran.



Správně vnořené precedenční grafy
.....

V případě (a) a (b) jsou všechny komponenty správně vnořené.

Nechť:

* S(a, b) označuje sériové spojení procesů
  (za procesem a následuje proces b),
* P(a, b) označuje paralelní spojení procesů a a b.

Precedenční graf je {správně vnořený} pokud může být popsán kompozicí
fcí S a P.

Vlastnost je podobná ”správnému vnoření” v blokově strukturovaných
jazycích (Pascal).

Příklad:

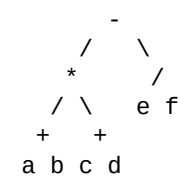
První tři grafy mohou výt zapsány jako:

(a) S(p1, S(p2, S(p3, p4)))
(b) P(p1, P(p2, P(p3, p4)))
(c) S(p1, S(P(p2, P(S(p3, P(p4, p5)), p6)), P(p7, p8)))

Čtvrtý graf není správně vnořený, nemůžeme popsat jako kompozici fcí S
a P: sériově spojené procesy (např. p2, p3) mají další proces (p4)
který začíná v uzlu mezi nimi, ale nelze jej popsat jako S(pi, pj) nebo
P(pi, pj).

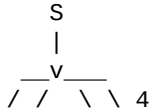
Příklady paralelismu, při kterém vznikají správně vnořené procesy:

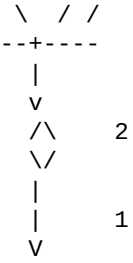
* vyhodnocení aritmetického výrazu:
  např. (a+b) * (c+d) - (e/f)



[[obr. precedenční graf bic-40]]

* třídění, např. dvoucestným slučováním:
  dvojice setříděných souborů velikosti 2^{i-1} jsou sloučeny do
  seznamu velikosti 2^i





[[obr. [zos-6]]]

* maticové násobení:
  při maticovém násobení A = B x C mohou být všechny prvky A počítány
  paralelně.