Procesy

PB 152 ◇ Operační systémy

Jan Staudek
http://www.fi.muni.cz/usr/staudek/vyuka/

Verze: jaro 2017

Úvodem k procesům

- Počítačová platforma se skládá z kolekce hardwarových prostředků
  - ✓ procesor, hlavní paměť, IO moduly, časovač, disky, ...
- Počítačové aplikace jsou vyvinuty k tomu, aby provedli nějaký úkol
  - ✓ získají vstupy z okolí, něco řeší, poskytnou výstupy do okolí
- Je neefektivní psát počítačové aplikace přímo pro jistou hardwarovou platformu
  - ✓ mnoho funkcí lze sdílet v mnoha různorodých aplikacích
  - ✓ procesor podporuje multiprogramování minimálně
  - ✓ zdroje používané souběžně řešenými aplikacemi je nutné chránit před neoprávněným zpřístupňování

Úvodem k procesům

- Byly vyvinuty OS s cílem poskytnout počítačovým aplikacím pohodlné, mnohoúčelové, bezpečné a konzistentní rozhraní na hardwarovou platformu
  - ✓ OS je vrstva mezi aplikacemi a hardware
- OS lze považovat za poskytovatele jednotných abstraktních reprezentací zdrojů, které aplikace mohou požadovat a zpřístupňovat si
  - ✓ hlavní paměť, síťová rozhraní, soubory dat, ...
  - ✓ OS zprostředkovává sdílení zdrojů a zajišťuje jejich ochranu

Program a proces

- Program
  - ✓ soubor přesně definovaného formátu obsahující
    – posloupnost instrukcí, případně sdílená více procesy
    – data potřebná k provedení stanoveného úkolu
- Proces (*Process*, *Task*)
  - ✓ akt provádění programu, instance výpočtu podle programu
  - ✓ proces je systémový objekt – jednotka aktivity charakterizovatelná prováděním posloupnosti instrukcí, okamžitým stavem a relevantní množinou systémových zdrojů
  - ✓ je charakterizovaný svým kontextem:
    OS přiděluje prostor ve FAP procesům, ne programům,
    proces vlastní obraz virtuálního (logického) adresového prostoru uchovávaný na vnější paměti,
    proces může vlastnit soubory, I/O zařízení, okna na obrazovce, komunikační kanály k jiným procesům (*sockets*), ...
  - ✓ OS přiděluje procesu čas procesoru, proces drží procesor = běží

Klíčová role procesů v OS

- OS maximalizuje využití procesoru prokládáním běhů procesů
- OS minimalizuje dobu odpovědi procesu prokládáním běhů procesů
- OS spravuje zdroje jejich přidělováním procesům podle vhodné politiky
  - ✓ priorita, vzájemná výlučnost, ...
  - ✓ a musí přitom zabránit „uváznutí" procesů
- OS podporuje optimální strukturování aplikačních systémů
- OS podporuje tvorbu procesů a meziprocesovou komunikaci
- Všechny OS s multiprogramováním jsou založeny konceptu proces
  - ✓ multiprogramování = multitasking

Vlákno a prostředí procesu

- Vlákno, *thred*
  - ✓ Abstrakce jedné z možných sekvenčních činnosti procesu
  - ✓ Proces zahrnuje mimo prostředí běhu alespoň jedno vlákno nebo se může realizovat pomocí více souběžně běžícími vlákny
- Prostředí běhu procesu, kontext procesu
  - ✓ Jedinečný identifikátor procesu v prostředí řízeném OS
  - ✓ Čítač instrukcí
  - ✓ Data obsažená v registrech procesoru
  - ✓ Logický a fyzický adresový prostor, ukazatele a data je vymezující
  - ✓ Nástroje pro synchronizaci a komunikaci vláken a procesů (semafory, ...), priorita
  - ✓ Komunikační rozhraní pro síťovou komunikaci (sockets, IO zařízení a související stavové informace
  - ✓ Zdroje vyšší úrovně (soubory, okna, ...)
  - ✓ Účtovací informace (doba běhu, doba existence)
  - ✓ Všechna vlákna jednoho procesu plně sdílejí prostředí svého procesu

Data nutná pro správu a řízení procesů

- Stavové informace
  - ✓ registry procesoru
    - obecné (střadače, ...), speciální
    - stav procesoru, ukazatelé zásobníku, ukazatelé haldy, čítač instrukcí
    - registr s návratovou adresou, ...
  - ✓ běží, čeká na událost, je připravený běžet, ...
- Informace nutné pro správu a řízení procesů
  - ✓ priorita procesu, stav procesu
  - ✓ informace o používání zdrojů systému procesem
    - spotřebovaný čas procesoru, doba běhu
  - ✓ PID (*process identifier*), účet vlastníka procesu, ...
  - ✓ seznam IO zařízení vlastněných procesem, seznam otevřených souborů, ukazatelé vyrovnávacích pamětí používaných pro otevřené soubory, ...
  - ✓ informace o používaných prostorech v operační paměti
    - báze, délka oblasti / stránkovací tabulka při virtualizaci paněti, ...
  - ✓ ...

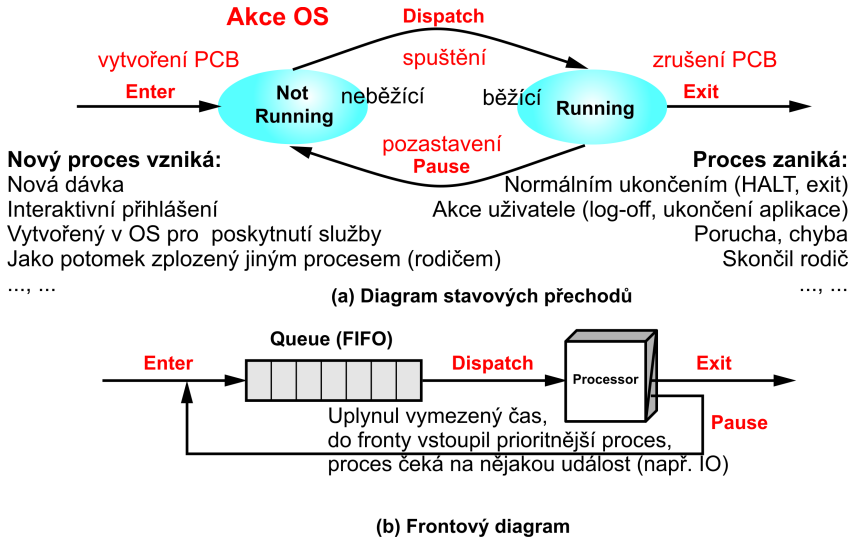
Process (Task) Control Block (PCB, TCB), Process Descriptor

- Tabulka OS informací potřebných pro definici a správu procesu, vytvářená, udržovaná a používaná operačním systémem, příp. některými mikroprogramy
- Prostor pro uchování definice prostředí procesu, který umožňuje běžící proces přerušit a později v něm pokračovat jakoby k přerušení nedošlo
- Dalším cílem je podpora koexistence více procesů pod jedním OS

Generický obsah PCB

- ✓ id procesu, jméno procesu
- ✓ vlastník procesu
- ✓ priorita procesu
- ✓ stav procesu
- ✓ stav čítače instrukcí a dalších registrů procesoru po přerušení
- ✓ informace potřebné pro plánování procesoru
- ✓ informace potřebné pro správu paměti, definice adresového prostoru
- ✓ účtovací informace
- ✓ informace potřebné pro správu I/O
- ✓ seznam otevřených souborů
- ✓ přidělená práva procesu
- ✓ seznam vláken procesu
- ✓ seznam potomků procesu
- ✓ vazební pole pro řazení PCB do front, seznamů, ...

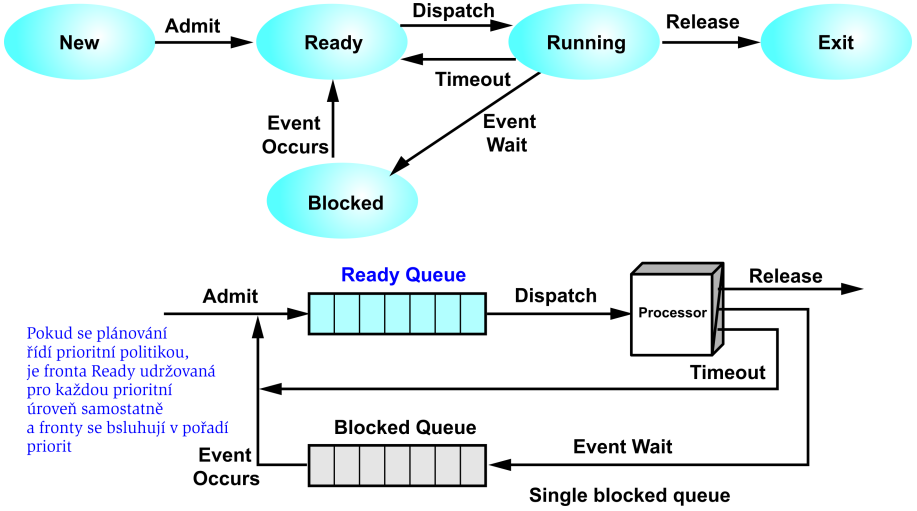
Stavy procesu, 2-stavová abstrakce



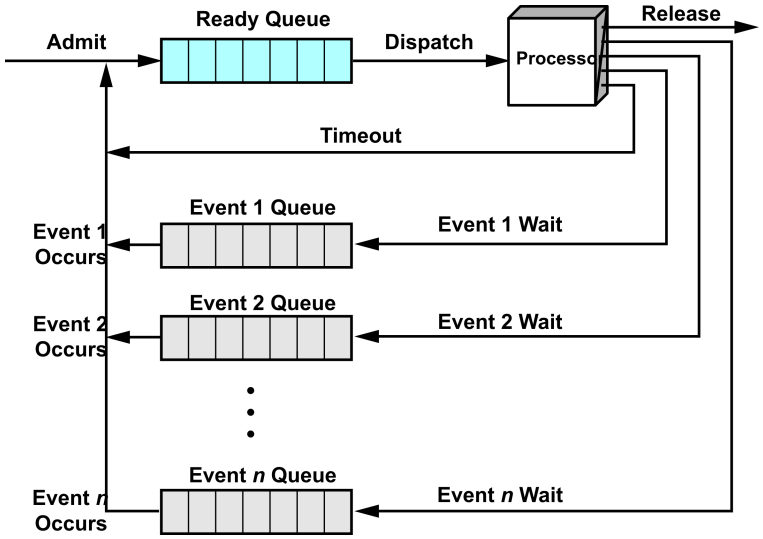
Stavy procesu, 5-stavová abstrakce

- ☐ nový, *new* –
  iniciální stav, právě vytvořený proces, identifikovatelný, dosud neplánovaný
- ☐ běžící, *running* –
  jeho program se právě interpretuje některou CPU
- ☐ čekající, waiting –
  čeká až nastane jistá událost, jiná než přidělení procesoru
- ☐ připravený, *ready* –
  čeká na přidělení procesoru, zvláštní stav čekání na jediný typ události, na přidělení procesoru
- ☐ ukončený, *terminated* – ukončil své provádění, stále ještě identifikovatelný

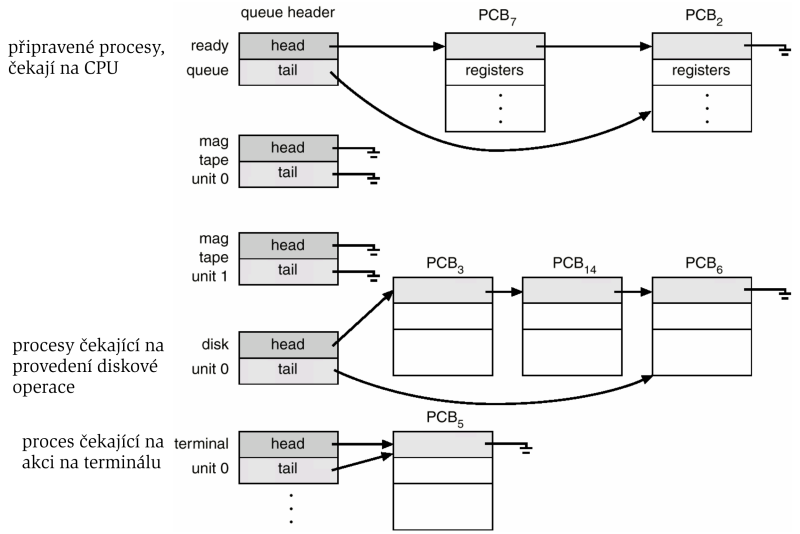
Stavy procesu, reálnější 5-stavová abstrakce



Stavy procesu, 5-stavová abstrakce, více front čekajících



Fronty / seznamy procesů, příklady



Fronty / seznamy procesů hrající roli při plánování procesů

- fronta připravených procesů
  - ✓ množina procesů připravených k běhu čekajících pouze na přidělení procesoru
- fronta na přidělení zařízení
- seznam odložených procesů
  - ✓ množina procesů čekajících na přidělení místa v hlavní paměti, FAP
- fronta na semafor
  - ✓ množina procesů čekajících synchronizační událost
- . . .
- Procesy mezi různými frontami migrují

Odkládání, swapping

- Běžící proces musí mít přidělený prostor ve FAP pro (alespoň aktuální část) LAP
- I když se používá princip virtuální paměti, příliš mnoho procesů ve FAP snižuje výkonnost systému
  - ✓ jednotlivé procesy obdrží malý prostor ve FAP a často se jim po částech vyměňují aktuální části LAP ve FAP
- OS musí při přetížení FAP provádění některých procesů odložit
  - ✓ odložení – ***swap-out***, ,,plácnutí''na disk
  - ✓ obnova – ***swap-in***, ,,plácnutí''do FAP
- přibývají tak další dva stavy procesů – 7stavový model
  - ✓ odložený připravený
  - ✓ odložený čekající

Další důvody pro odkládání

- OS může potlačit kterýkoliv záložní nebo pomocný proces nebo proces podezřelý, že by tento mohl způsobit problém
- Uživatel může potlačit svůj proces např. při ladění nebo v souvislosti s možností používat nějaký zdroj
- Časování může periodicky obnovovat a odkládat např. účtovací nebo monitorovací procesy
- Rodičovský proces může odložit provádění potomka z důvodu koordinace činností potomků
- ...

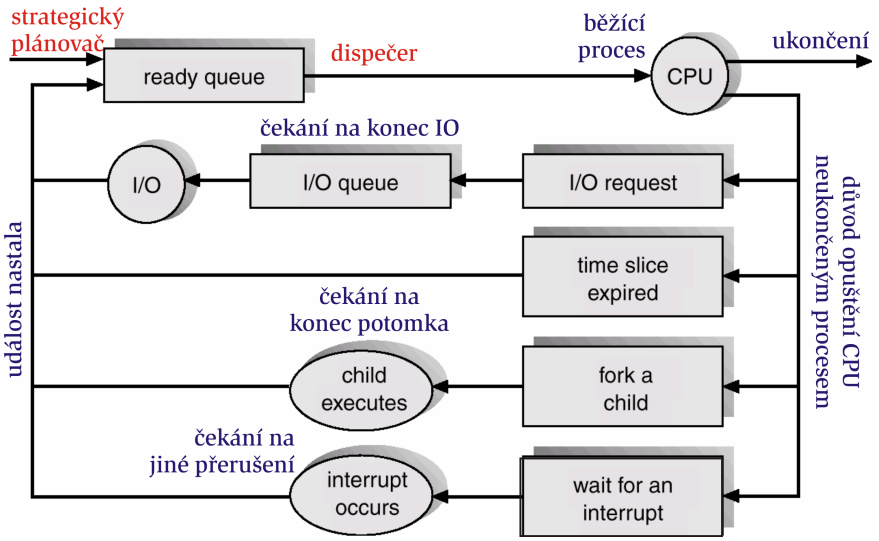
Přechody mezi stavy (odložených) procesů

- čekající → odložený čekající
  - ✓ uvolnění oblasti FAP obsazenou ladem ležící částí LAP
- odložený čekající → odložený připravený
  - ✓ stala se očekávaná událost (stavovou info má OS přístupnou)
  - ✓ proces zůstává odložený
- odložený připravený → připravený
  - ✓ uvolnil se prostor ve FAP
- připravený, resp. běžící → odložený připravený
  - ✓ uvolnění části FAP ve prospěch prioritnějších procesů

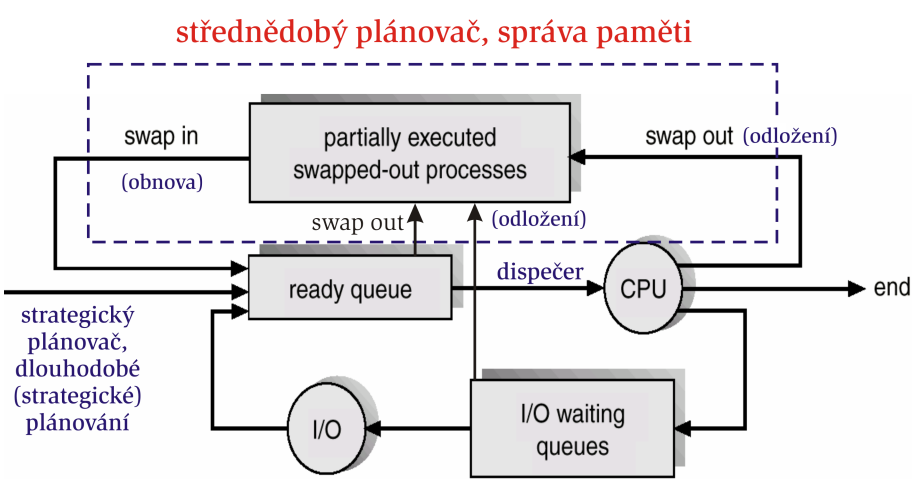
Plánovače v OS

- dlouhodobý plánovač (strategický plánovač, *job scheduler*)
  - ✓ vybírá který požadavek na výpočet lze zařadit mezi procesy
  - ✓ definuje stupeň multiprogramování
  - ✓ je vyvoláván řídce, nemusí být rychlý
- krátkodobý plánovač (operační plánovač, dispečer, *dispatcher*)
  - ✓ reprezentace správy procesoru(-ů)
  - ✓ vybírá proces, který poběží na uvolněném procesoru přiděluje procesu procesor (CPU)
  - ✓ vyvoláván velmi často, desítky – stovky milisekund, musí být rychlý
- Střednědobý plánovač (taktický plánovač)
  - ✓ Logicky náleží částečně i do správy hlavní paměti (FAP)
  - ✓ taktika využívání omezené kapacity FAP při multitaskingu
  - ✓ vybírá který proces je nutné zařadit mezi odsunuté procesy (odebírá mu prostor ve FAP)
  - ✓ vybírá kterému odsunutému proces lze opět přidělit prostor ve FAP

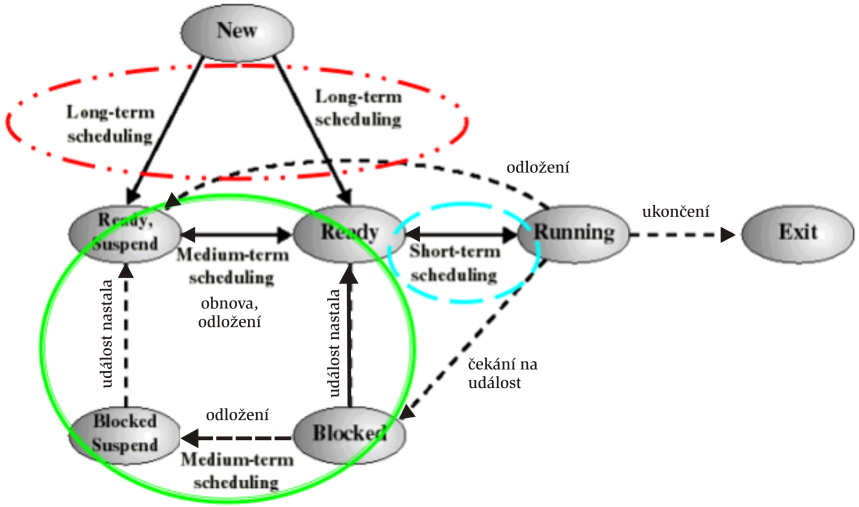
Dlouhodobý plánovač, dispečer, získání/opuštění CPU



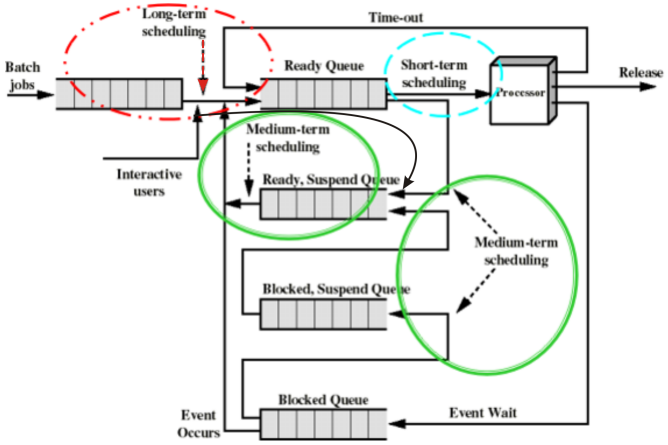
Střednědobý plánovač, odkládání (suspending) procesů



Plánování procesoru a 7-stavový model života procesu



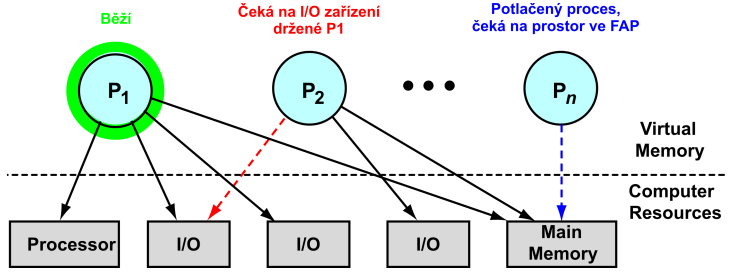
Frontový 7-stavový model plánování procesoru



✓ Plánovač CPU vybírá z procesů, které sídlí v hlavní paměti, ty procesy, které jsou připravené k běhu – *ready*

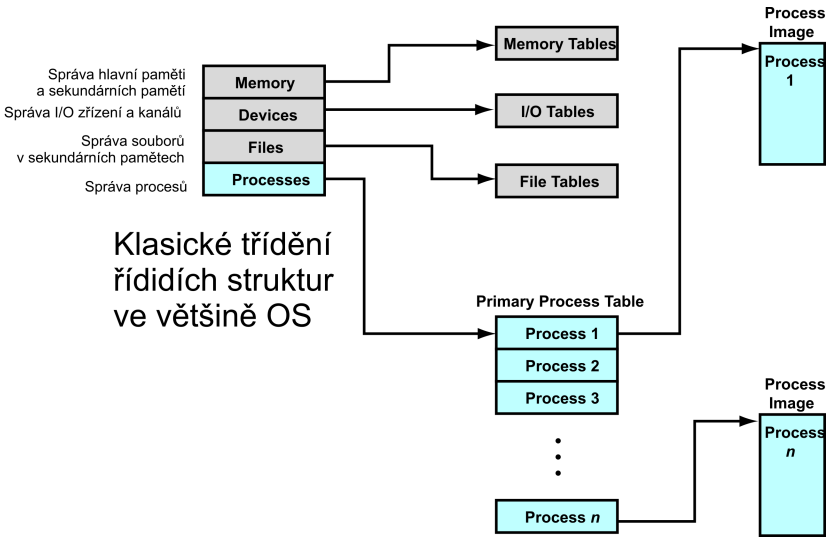
Multitasking, procesy, zdroje, organizování v OS

- Ve virtuální paměti koexistuje n procesů



- Má-li OS řídit běh procesů a využívání zdrojů, musí mít informaci o jejich stavu – typicky v tabulkách

Řídicí tabulky v OS



Řídicí tabulky v OS

- Tabulky správy paměti definují
  - ✓ procesům přidělené prostory ve FAP
  - ✓ procesům přidělené prostory v sekundární paměti
  - ✓ bezpečnostní vlastnosti přístupů do sdílených oblastí FAP
  - ✓ data potřebná pro virtualizaci paměti
- Tabulky I/O systému definují
  - ✓ zda jsou I/O zařízení přidělená, a pokud ano, kterým procesům
  - ✓ stavy I/O operací
  - ✓ umístěni v hlavní paměti s vyrovnávacími pamětmi I/O operací
- Tabulky systému souborů definují
  - ✓ externí identifikaci souborů
  - ✓ umístění souborů v sekundární paměti
  - ✓ vlastnosti souborů (bezpečnost, organizace, . . .)

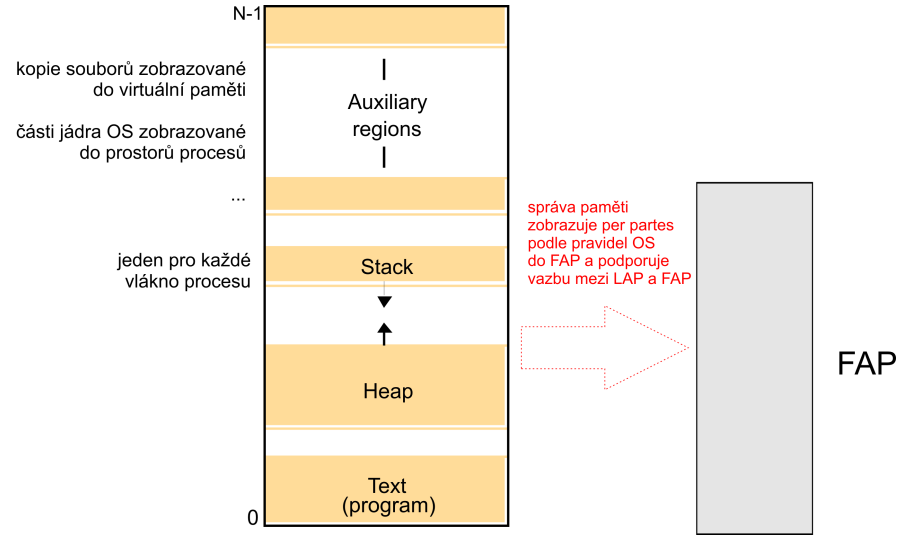
Řídicí tabulky v OS

- Tabulky správy procesů
  - ✓ id procesů, zdroje přidělené procesům (paměť, IO, . . .)
    stavy procesů, vazby mezi procesy – PCB (*Process control block*)
  - ✓ provázaní s přísluš. tabulkami IO systému, systému souborů, . . .
  - ✓ jsou předmětem správy paměti OS
- PCB (*Process control block*)
  - ✓ identifikace procesu (unikátní id, slouží co reference na proces)
  - ✓ informace o stavu procesu – obsah uživatelských, řídicích a stavových registrů, ukazatelé zásobníků, . . .
  - ✓ informace nutné pro řízení procesů
  - ✓ *Program status word* (PSW) – stavové informace
    např. v Pentiu registr EFLAGS

Adresový prostor procesu

- Adresový prostor procesu (Logický adresový prostor procesu)
  - ✓ správní jednotka virtuální paměti pro proces odpovídající části LAP využívané procesem
  - ✓ nepřekrývající se oblasti dostupné vláknům procesu
  - ✓ každá oblast
    - má svůj prostor (*extent*) vymezený typicky bázovou adresou a délkou
    - má r/w/x přístupová práva pro vlákna procesu
    - je typicky stránkováním zobrazovaná do FAP
    - může se v době běhu procesu zvětšovat / zmenšovat
- Adresový prostor v systémech s OS Unix
  - ✓ pevná nemodifikovatelná oblast text region s programem
  - ✓ halda, heap, oblast rozšiřitelná do vyšších adres virtuálního adresového prostoru procesu
  - ✓ zásobník, stack, oblast rozšiřitelná do nižších adres virtuálního adresového prostoru procesu
  - ✓ libovolný počet dalších pomocných oblastí (auxiliary region)

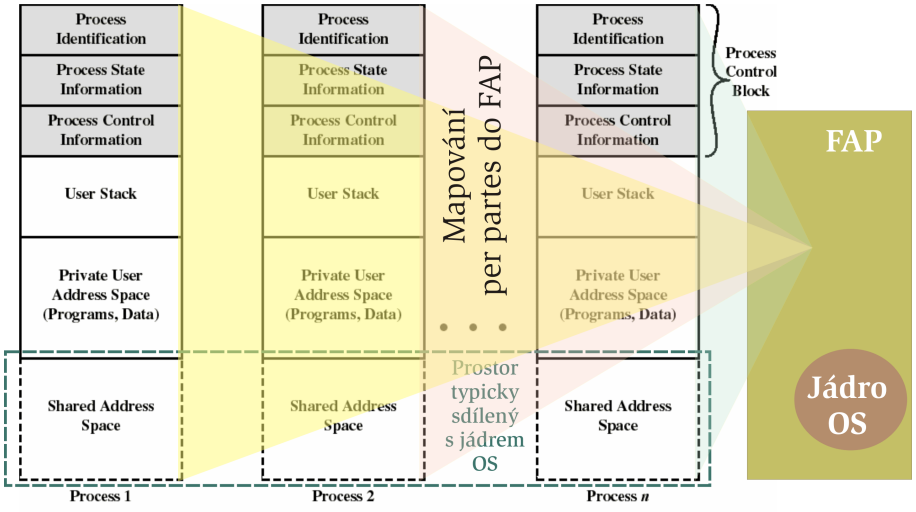
Adresový prostor procesu (př. Unix)



Adresový prostor procesu – vybrané komponenty

- Halda
  - ✓ inicializovaná z části hodnotami ze souboru s binární verzí programu
  - ✓ dynamicky rozšiřovatelná, dynamicky definovaný obsah
- Zásobník
  - ✓ obecně vždy jeden pro každé vlákno v procesu
  - ✓ obsluha typu LIFO (last-in, first out)
  - ✓ pro ukládání návratové adresy při volání funkce
  - ✓ pro ukládání lokálních proměnných funkce, předávání parametrů, . . .
- příklad pomocné oblasti – oblasti souborů (File regions)
  - ✓ Podpora zobrazování souborů z vnější paměti do virtuální paměti
- příklad pomocné oblasti – sdílené oblasti (Shared Memory Regions)
  - ✓ Pro komunikaci sdílením paměti mezi procesy, mezi procesem a jádrem OS, pro umístění knihovních podprogramů, . . .

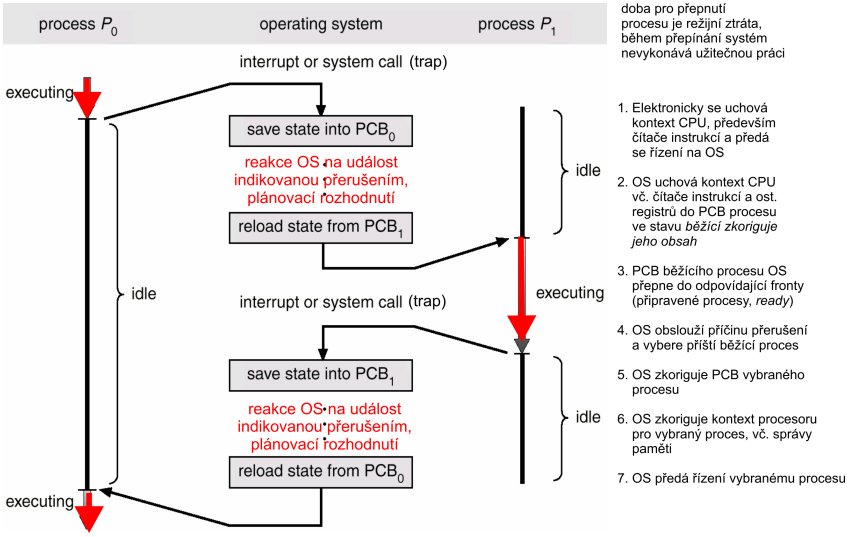
Obrazy procesů ve virtuální paměti



Vytvoření procesu, přepnutí proces → OS

- OS rozhodl, že vytvoří nový proces
  - ✓ přidělí novému procesu jedinečný id
  - ✓ vyhradí ve fyzickém adresovém prostoru místo pro každou potřebnou komponentu obrazu procesu
  - ✓ inicializuje PCB
  - ✓ proces zařadí do vhodné fronty (mezi připravené procesy)
  - ✓ vytvoří nebo doplní potřebné datové struktury (účtovací soubor apod.)
- Příčiny přepnutí proces → OS
  - ✓ přerušením (*interrupt*) – je nutná reakce OS na asynchronní událost, externí vůči běžícímu procesu (I/O, časovač, výpadek, . . .)
  - ✓ synchronním přerušením (*trap*) – je nutná reakce OS na událost související s prováděnou instrukcí (chyba, výjimka, volání služby OS)
  - ✓ OS obslouží příčinu přerušení a rozhodne o dalším postupu

Context Switching, přepínání CPU mezi procesy



Context Switching, přepínání CPU mezi procesy

- přepínání procesů může kriticky ovlivňovat výkon systému
- přepínání procesů je klíčový koncept multiprogramování (multitasking)
- asociované problémy
  - ✓ proces čekající na signál od IO zařízení nesmí signál ztratit
  - ✓ opakovaně použitelné zdroje musí být procesy zpřístupňované exkluzivně – problém synchronizace
  - ✓ determinismus – výsledky procesů nesmí být závislé na frekvenci a okamžiky přepínání mezi procesy
  - ✓ . . . – viz později synchronizační úlohy, uváznutí, . . .
- požadavky na OS z hlediska multiprogramování
  - ✓ existuje politika (pravidla) určující běžící proces(y)
  - ✓ v jádru existuje kód (program) řídící přepínání – dispečer, plánovač
  - ✓ v jádru jsou implementované mechanismy ochrany před škodlivými akcemi mezi (nezávislými) procesy

Plánovač CPU, dispečer

- Plánovač CPU je aktivovaný a plánovací rozhodnutí vydává v okamžiku, kdy:
  - ✓ běžící proces přechází do stavu čekající (na jistou událost)
  - ✓ běžící proces končí
  - ✓ běžící proces přechází do stavu připravený (uplynul čas po který směl běžet)
  - ✓ čekající proces přechází do stavu připravený (nastala událost, na kterou proces čekal)
- První dva případy se řeší voláním příslušné služby OS (synchronně, nepreemptivní plánování)
- Poslední dva případy se iniciují na základě výskytu relevantního přerušení (asynchronně, preemptivní plánování)

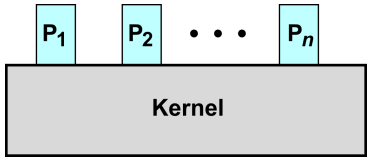
Plánovač CPU, dispečer

- předání procesoru z dispečera na proces představuje:
  - ✓ nastavení kontextu procesu
  - ✓ přepnutí režimu procesoru z privilegovaného režimu na uživatelský režim
  - ✓ předání řízení na místo v uživatelském programu určené pro restart (start) procesu
- zpoždění způsobené dispečerem
  - ✓ doba, kterou potřebuje dispečer pro pozastavení běhu jednoho procesu a start běhu jiného procesu

OS se neřeší jako proces ?

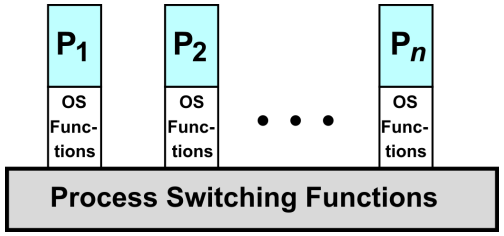
- OS je kolekce programů, která řídí procesor
- Tradiční řešení – jádro OS se řeší mimo rovinu procesů
  - ✓ OS jako samostatná entita
    běží v privilegovaném režimu mimo kontext procesů
  - ✓ je aktivovaná přerušením, provede službu a službu končí
    přepnutím procesoru na vybraný příští uživatelský proces
- Na malých počítačích bývá celý OS řešený v kontextu uživatelského procesu,
  - ✓ OS se chápe jako kolekce (pod)programů,
    které uživatelský proces volá k provedení různých funkcí
- Valnou část OS lze implementovat jako kolekci systémových procesů běžících nad samostatným (funkčně i rozsahem) minimálním jádru (μ-jádru)

Neprocesové jádro



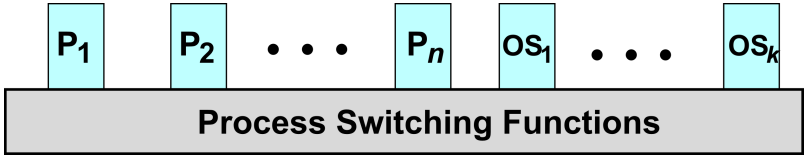
- koncept procesu se aplikuje pouze na aplikační programy
- OS přebere řízení přerušením, běží v privilegovaném režimu, uklidí kontext běžícího procesu do jeho PCB
- OS provede odpovídající službu a vybere příště běžící proces
- OS obnoví kontext vybraného procesu z jeho PCB
  vč. přepnutí do neprivilegovaného režimu

Běh OS v režimu uživatelského procesu



- Součástí obrazu procesu jsou podstatné části jádra
- OS přebere řízení přerušením,
  CPU se přepne do privilegovaného režimu
- NEUKLÍZÍ se stav běžícího procesu do jeho PCB
- OS provede přísl. službu a rozhodne o příštím běžícím procesu
- Pokud vybere přerušený proces, neobnovuje kontext z PCB

Procesově řešený OS



- Modulární návrh, jasně definovaná rozhraní
- Jádro – kritické funkce:
  dispečer, správa paměti, předávání zpráv mezi procesy
- Vhodné pro multiprocesorové systémy

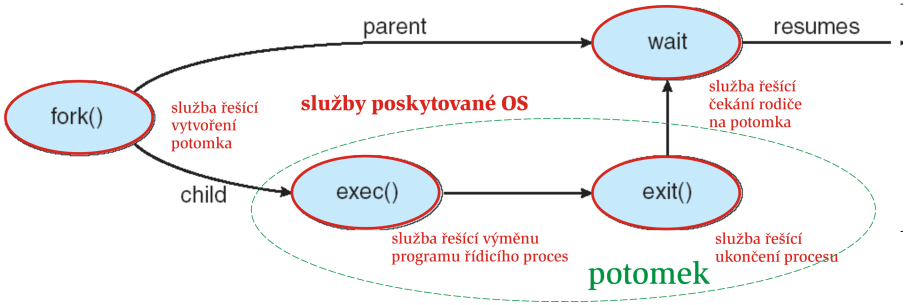
Vytvoření procesu

- každý OS musí mít mechanismy pro vytváření nových procesů
- 2 způsoby vytvoření nového procesu
- ,,na zelené louce'':
  - ✓ do přidělené paměti (FAP) se zavede text programu a data
  - ✓ vytvoří se (prázdná) halda
  - ✓ vytvoří se a inicializuje se PCB
  - ✓ proces se zpřístupní dispečerovi
- klon existujícího procesu
  - ✓ běžící proces se pozastaví (vrátí mezi připravené, ...)
  - ✓ okopíruje se text, data, dynamické oblasti paměti, ... a PCB
  - ✓ nový proces se zpřístupní dispečerovi
  - ✓ generující i generovaný proces je informovaný o své roli (rodič, potomek)

Vytvoření procesu klonem

- rodičovský proces vytváří procesy potomky
  - ✓ pomocí služby OS
- Potomci mohou vystupovat v roli rodičů a vytvářet svoje potomky, ...
  - ✓ formuje se tak strom genealogický strom procesů
- Sdílení zdrojů mezi rodiči a potomky – varianty
  - ✓ rodič a potomek mohou sdílet zdroje původně vlastněné rodičem
  - ✓ potomek může sdílet s rodičem rodičem vyčleněnou podmnožinu zdrojů
  - ✓ potomek a rodič jsou plně samostatné procesy, nesdílí žádný zdroj

Rodič a potomek



- Souběh mezi rodiči a potomky – varianty
  - ✓ rodič a potomek může běžet souběžně
  - ✓ rodič čeká na ukončení potomka

Příklad vytváření procesu – MS-DOS

- MS-DOS byl monoprogramový systém
- potomka vytvářela služba OS LOAD_AND_EXEC
- rodič zůstával potlačený, čekal na ukončení potomka, procesy neběžely souběžně

Příklad vytváření procesu – Unix

- rodič vytváří nový proces – potomka voláním služby fork
- vznikne identická kopie rodičovského procesu,
  oba procesy zůstávají připravené
  - ✓ potomek je úplným duplikátem rodiče a
  - ✓ každý z obou procesů se při vytváření procesu dozvídá zda je rodič nebo potomek
  - ✓ do potomkova adresového prostoru se při vytváření procesu zavádí program shodný s rodičem
- potomek příp. použije volání služby exec pro náhradu programu rodiče svým novým programem
- rodič se po vytvoření dozvídá, že je rodič,
  potomek se po svém vytvoření dozvídá, že je potomek
  (*fork()* vrací potomku 0, rodiči PID potomka)

Unix, postup při vytvoření procesu voláním *fork*

1. Řízení převezme OS a v režimu jádra
2. v tabulce procesů OS přidělí místo pro nový PCB
3. potomkovi přidělí jedinečné PID
4. až na sdílenou oblast okopíruje obraz procesu rodiče
5. zvýší o 1 čítače použití všem otevřeným souborům rodičem
6. potomka zařadí mezi připravené procesy
7. rodiči vrací jako funkční hodnotu PID potomka, potomkovi 0.
8. Dispečer vybere příště běžící proces,
   řízení se může vrátit rodiči, předat potomkovi nebo i jinému procesu
   - ✓ potomek je spouštěný za místem, kde v rodiči je *fork*

Co se děje po vytvoření procesu

- sdílení zdrojů
  - ✓ rodič a potomci mohou sdílet zdroje rodiče
  - ✓ potomek může sdílet podmnožinu zdrojů rodiče
  - ✓ rodič a potomek nemusí sdílet žádné zdroje
- běh procesů po vytvoření procesu
  - ✓ rodič a potomek mohou běžet souběžně
  - ✓ rodič může čekat na ukončení potomka (v Unixu *wait*)
  - ✓ rodič se může bezprostředně ukončit
    - pokud rodič skončí dříve než potomek,
      potomek po ukončení se stane *zombie*

Vytvoření procesu klonem, Unix

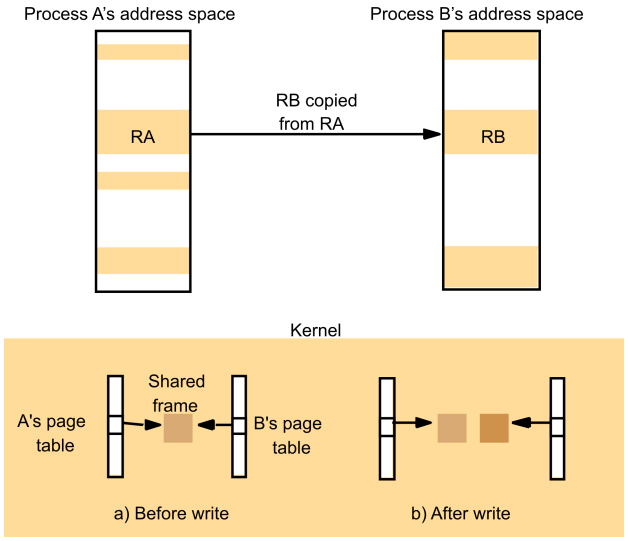
- voláním služby fork()
  - ✓ *fork()* vytvoří identickou kopii volajícího procesu
  - ✓ po provedení *fork()* zůstává rodič připravený proces a soupeří s potomkem o získání CPU
  - ✓ po provedení *fork()* lze použít volání služby exec,
    která nahradí paměťový prostor procesu novým programem

```
pid = fork();
if (pid == 0) {
// proces potomka – zde se nastaví prostředí pro potomka
...
} else {
// rodičovský proces – např. čeká na ukončení potomka
wait(pid);}
```

Vytvoření procesu stylem Copy on Write

- Nově vytvářený proces požaduje vytvoření nového prostředí běhu
- Tradiční forma vytvoření procesu unixového typu
  - ✓ služba OS *fork* vytvoří nové prostředí běhu kopií prostředí žádajícího procesu + sdělení novému procesu, že je potomkem vytvářejícího procesu
  - ✓ služba *exec* umožní volajícímu procesu definovat nový program řídicí proces kopií z udaného souboru
- Vytvoření procesu způsobem *Copy on Write*
  - ✓ iniciálně nový proces sdílí stránky s původním procesem
  - ✓ při zápisu do stránky novým procesem se vytvoří pro nový proces samostatná kopie modifikované stránky

Vytvoření procesu způsobem *Copy on Write*



Ukončení procesu

- Proces provede poslední příkaz programu a žádá OS o (své) ukončení (voláním služby exit)
  - ✓ výstupní data procesu-potomka se mohou předat procesu-rodiči, který čeká v provádění služby wait
  - ✓ Zdroje končícího procesu se uvolňují
- O ukončení procesu-potomka může požádat jeho rodič (abort), protože (např.)
  - ✓ potomek překročil stanovenou mez čerpání zdrojů
  - ✓ úkol přidělený potomkovi rodič dále nepotřebuje
  - ✓ Rodič končí svoji existenci a při vytváření potomka nebylo povoleno, aby potomek přežil svého rodiče – může docházet ke kaskádnímu ukončování
- proces může končit abnormálně,
  - ✓ chybou v programu, uplynutím povolené doby k běhu, . . .

Formy kooperace procesů, viz samostatná přednáška

- Nezávislé procesy
  - ✓ nemohou se vzájemně ovlivňovat
- Kooperující procesy
  - ✓ mohou ovlivňovat běh jiných sdružených procesů nebo jiné sdružené procesy mohou ovlivňovat jejich běh
- Přínosy kooperace procesů
  - ✓ sdílení informací
  - ✓ urychlení výpočtů – souběžnost řešení
  - ✓ modularita – snadnější implementace „rozděl a panuj"
  - ✓ pohodlí – každý z procesů je snadněji přizpůsobitelný okolí
- Klasická paradigma kooperace
  - ✓ producent – konzument, klient – server, čtenáři – písaři, . . .