



Počítačové sítě a operační systémy

Procesy a vlákna

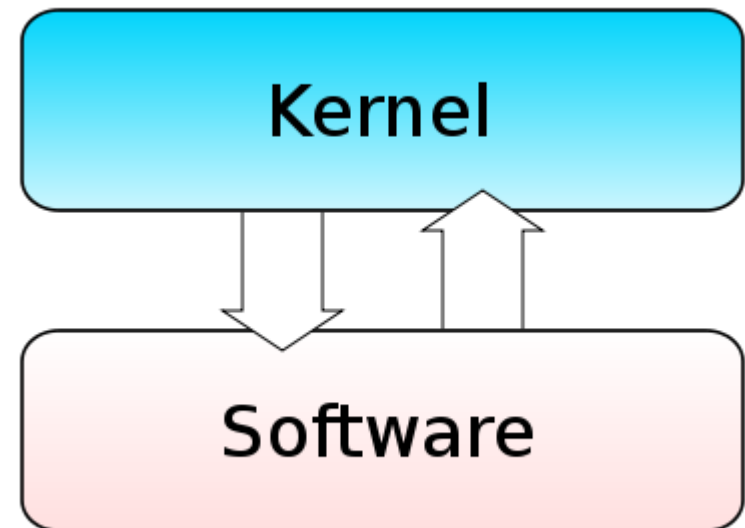
Jaromír Plhák
xplhak@fi.muni.cz

Vnitřní struktura OS

- Existuje řada přístupů a implementací
 - Jedno velké monolitické jádro
 - Modulární, hierarchický přístup
 - Malé jádro a samostatné procesy
- Struktura mnoha OS je poznamenána historií OS a původními záměry, které se mohou od současného stavu radikálně lišit

Monolitické jádro

- OS běží kompletně v jaderném paměťovém prostoru (kernel space)
- Činnosti jednotlivých subsystémů jsou odděleny, ale velice silně provázány
 - Navíc sdílejí stejný paměťový prostor
- Například MS-DOS

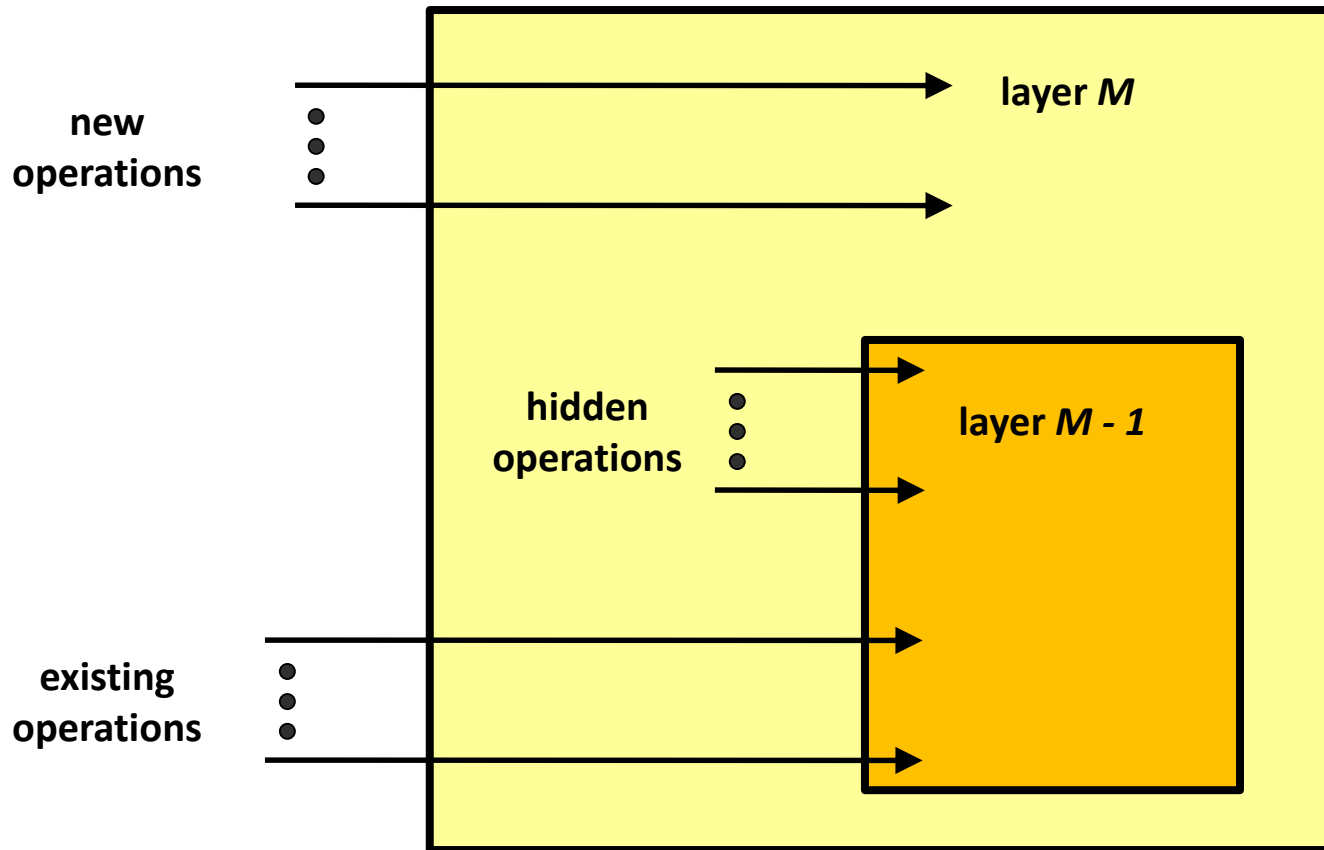




Hierarchická vrstvová architektura (1)

- OS se dělí do jistého počtu vrstev (úrovní)
- Každá vrstva je budována na funkcionalitě nižších vrstev
- Nejnižší vrstva (0) je hardware
- Nejvyšší vrstva je uživatelské rozhraní
- Pomocí principu modulů jsou vrstvy vybírány tak, aby každá používala funkcí (operací) a služeb pouze vrstvy $n - 1$

Hierarchická vrstvová architektura (2)



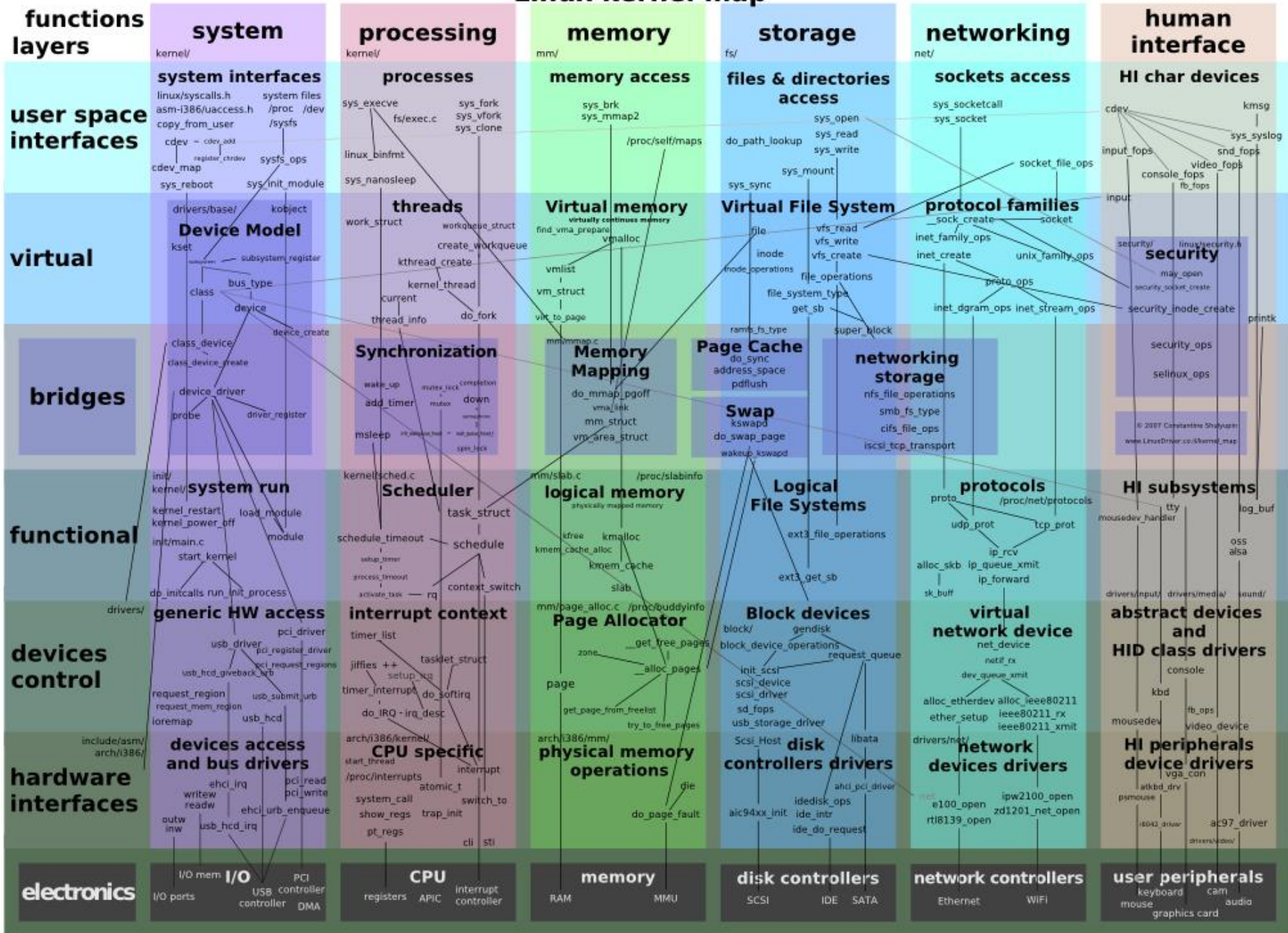
Hierarchická vrstvová architektura (3)

- Řeší problém přílišné složitosti velkého systému
 - Provádí se dekompozice velkého problému na několik menších zvládnutelných problémů
- Každá úroveň řeší konzistentní podmnožinu funkcí
- Nižší vrstva nabízí vyšší vrstvě „primitivní“ funkce (služby)
- Nižší vrstva nemůže požadovat provedení služeb vyšší vrstvy
- Používají se přesně definovaná rozhraní
 - Jednu vrstvu lze uvnitř modifikovat, aniž to ovlivní ostatní vrstvy – princip modularity

Hierarchická vrstvová architektura (4)

- Výhodou je modularita OS
- Nevýhodou je především vyšší režie a tím pomalejší vykonávání systémových volání
- Protože efektivita hraje v jádře OS významnou roli je třeba volit kompromis
 - Pouze omezený počet úrovní pokrývající vyšší funkcionalitu
 - Příkladem je první verze Windows NT
 - Měly hierarchickou strukturu s řadou vrstev, avšak pro zvýšení výkonu OS bylo ve verzi NT 4.0 rozhodnuto přesunout více funkcionality do jádra a sloučit některé vrstvy

Linux kernel map



Struktura s mikrojádrem (1)

- Microkernel System Structure
- Malé jádro OS plní pouze několik málo nezbytných funkcí
 - Primitivní správa paměti (adresový prostor)
 - Komunikace mezi procesy – Interprocess communication (IPC)
- Většina funkcí z jádra se přesouvá do „uživatelské“ oblasti
 - Ovladače HW zařízení, služby systému souborů, virtualizace paměti ...
 - Mezi uživatelskými procesy se komunikuje předáváním zpráv

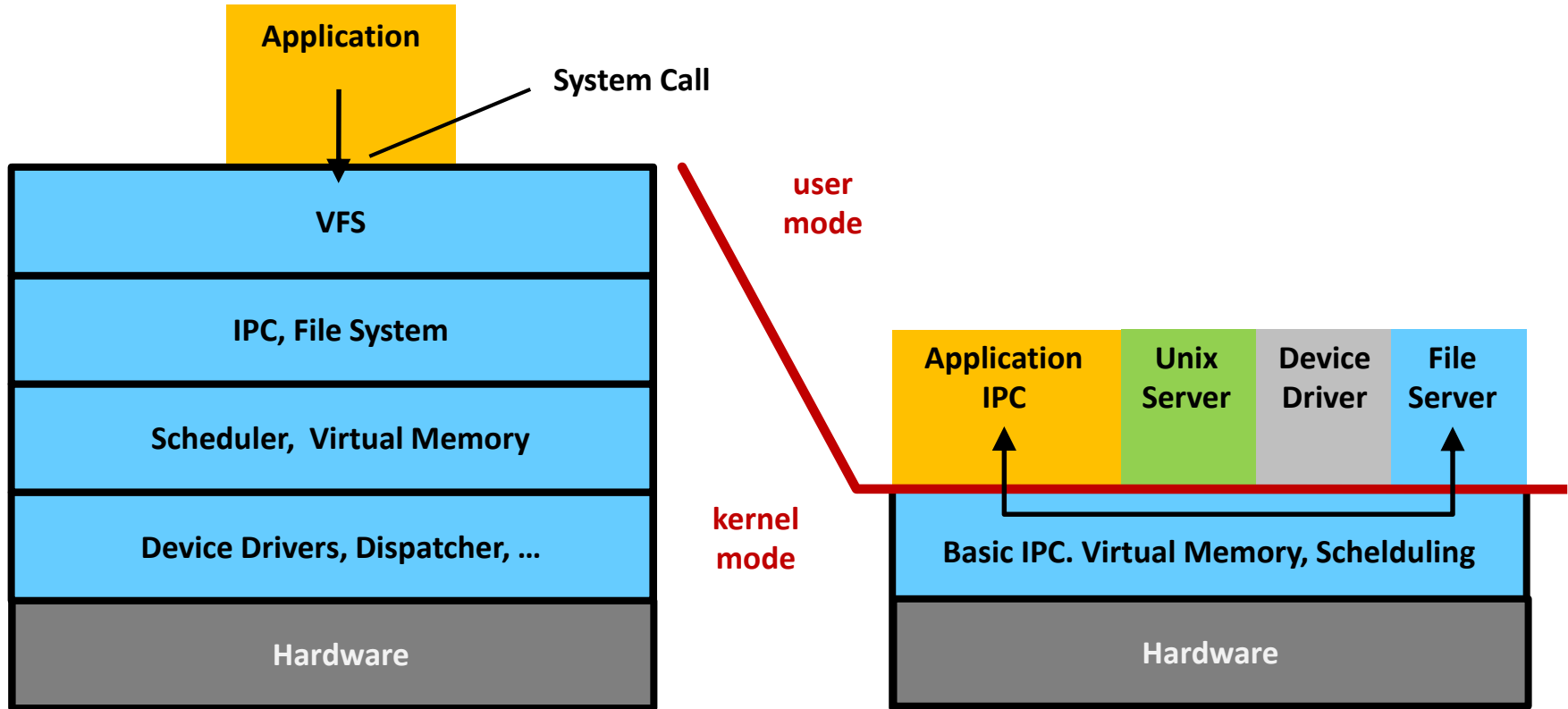
Struktura s mikrojádroem (2)

- Výhody mikrojádra
 - Snadná přenositelnost OS, jádro je malé
 - Vyšší spolehlivost (moduly mají jasné API a jsou snadněji testovatelné)
 - Vyšší bezpečnost (méně kódu OS běží v režimu jádra)
 - Flexibilita (jednodušší modifikace, přidání, odebrání modulů)
 - Všechny služby jsou poskytovány jednotně (výměnou zpráv)
- Nevýhoda mikrojádra
 - Zvýšená reže
 - Volání služeb je nahrazeno výměnou zpráv mezi procesy

Monolitické jádro a mikrojádru

Monolithic Kernel based Operating System

Microkernel based Operating System



Program a proces (1)

- Program
 - Soubor přesně definovaného formátu obsahující
 - Posloupnost instrukcí, případně sdílená více procesy
 - Data potřebná k provedení stanoveného úkolu
- Proces (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

Program a proces (2)

- Proces (pokr.)
 - 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**

Co je to proces?

- Pro spuštěný program máme řadu pojmenování
 - Dávkové systémy – úlohy, dávky, jobs
 - Multitaskingové systémy – procesy, vlákna
- Společné pojmenování pro spuštěný program je **proces**
- Dále zavádíme pojem **vlákno** pro „dílčí“ proces v rámci procesu
 - Abstrakce jedné z možných sekvenčních činností procesu
 - Proces zahrnuje mimo prostředí běhu alespoň jedno vlákno
 - Může se realizovat pomocí několika vláken



Data nutná pro správu a řízení procesů (1)

- 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, ...



Data nutná pro správu a řízení procesů (2)

- 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 I/O 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 paměti
- ...

Informace OS o procesu

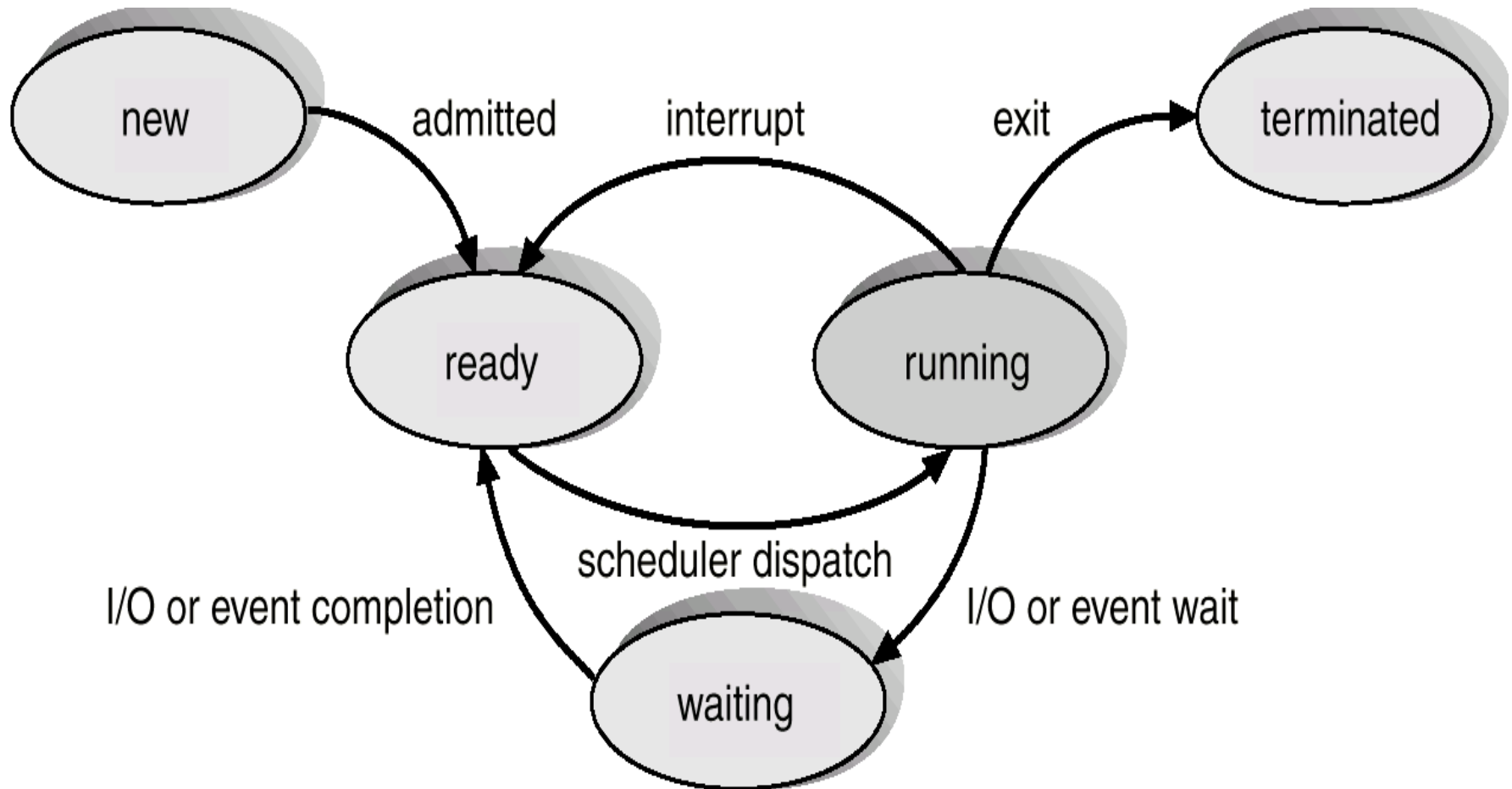
- Process Control Block – tabulka obsahující informace potřebné pro definici a správu procesu (udržovaná OS)
 - Id procesu a jeho priorita
 - Stav procesu (běžící, připravený, ...)
 - Čítač instrukcí
 - Registry procesoru
 - Informace potřebné pro správu paměti
 - Informace potřebné pro správu I/O
 - Účtovací informace
 - ...

pointer	process state
process number	
program counter	
registers	
memory limits	
list of open files	
⋮	

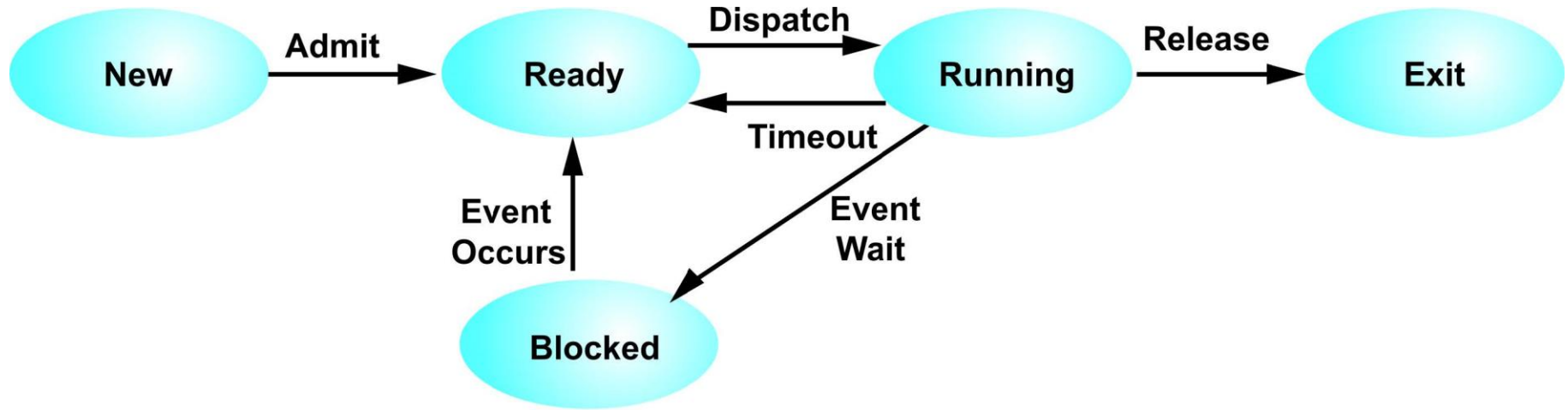
Stavy procesu (1)

- Proces se může nacházet v jednom ze stavů
 - Nový (new) – právě vytvořený proces
 - Běžící (running) – některý procesor právě vykonává instrukce procesu
 - Čekající (waiting) – čeká na určitou událost
 - Připravený (ready) – čeká na přidělení času procesoru
 - Ukončený (terminated) – ukončil své provádění

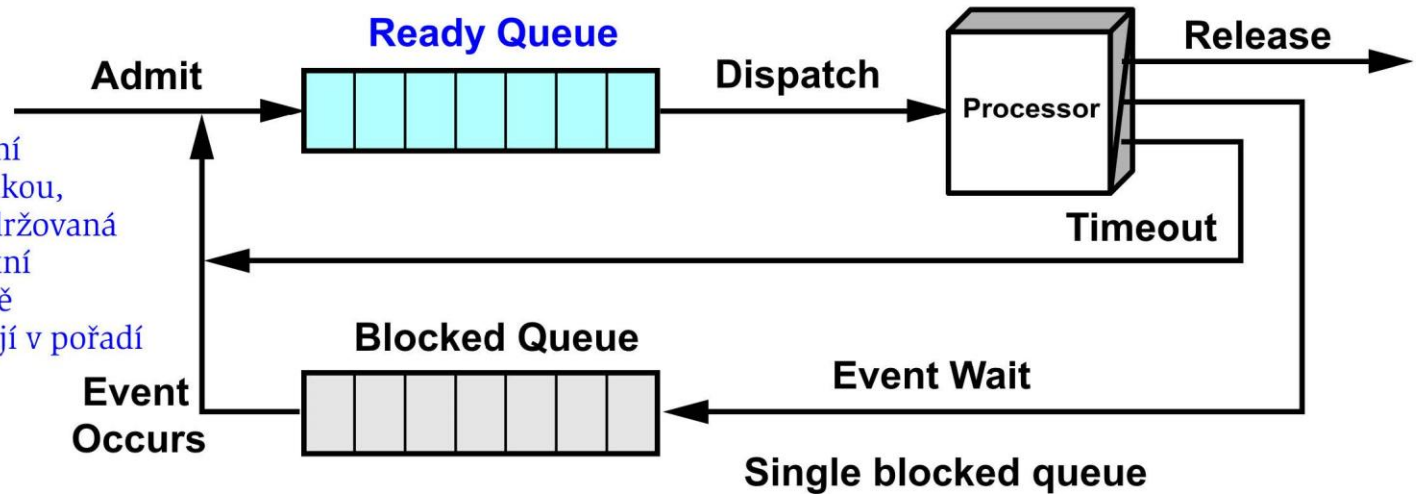
Stavy procesu (2)



Stavy procesu (3)



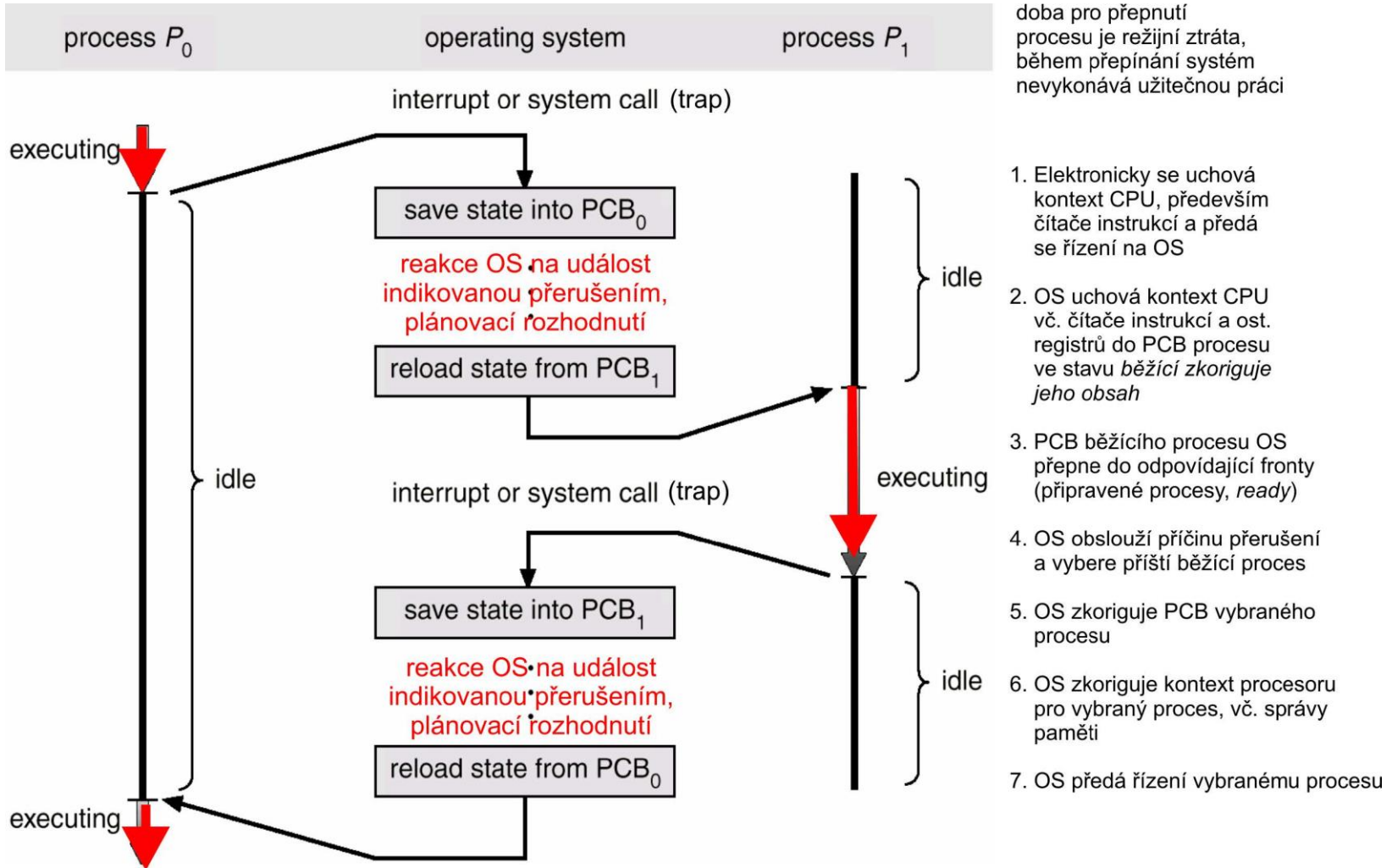
Pokud se plánování řídí prioritní politikou, je fronta Ready udržovaná pro každou prioritní úroveň samostatně a fronty se bsluhují v pořadí priorit



Vytvoření procesu, přepnutí z procesu na 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 (účetovací soubor apod.)
- Příčiny přepnutí z procesu na OS
 - Přerušením – je nutná reakce OS na asynchronní událost, externí vůči běžícímu procesu (I/O, časovač, výpadek, ...)
 - Synchronním přerušením – je nutná reakce OS na událost související s prováděnou instrukcí (chyba, výjimka, volání služby OS)
 - OS obslouží příčinu přerušení a rozhodne o dalším postupu

Přepnutí procesu



Přepnutí kontextu (1)

- Vyžádá se služba, akceptuje se některé asynchronní přerušení, obslouží se a nově se vybere jiný běžící proces
- Když OS přepojuje CPU z procesu X na proces Y, musí
 - Uchovat (uložit v PCB procesu X) stav původně běžícího procesu
 - Zavést stav nově běžícího procesu (z PCB procesu Y)
- Přepnutí kontextu představuje režijní ztrátu (zátěž)
 - Během přepínání systém nedělá nic efektivního

Přepnutí kontextu (2)

- Doba přepnutí závisí na konkrétní HW platformě
 - Počet registrů procesoru, speciální instrukce pro uložení/načtení všech registrů procesoru apod.
- Při přerušení musí procesor
 - Uchovat čítač instrukcí
 - Zavést do čítače instrukcí hodnotou adresy vstupního bodu ovladače přerušení z vektoru přerušení

Vytvoření procesu

- Rodič vytváří potomky (další procesy)
- Potomci mohou vytvářet další potomky ...
- Vzniká strom procesů
- Sdílení zdrojů – varianty při vytváření potomků
 - Rodič a potomek sdílejí zdroje původně vlastněné rodičem
 - Potomek sdílí rodičem vyčleněnou podmnožinu zdrojů s rodičem
 - Potomek a rodič jsou plně samostatné procesy, nesdílí žádný zdroj
- Běh
 - Rodič a potomek mohou běžet souběžně
 - Rodič čeká na ukončení potomka

Ukončení procesu

- Proces provede poslední příkaz a sám požádá OS o ukončení
 - Výstupní data procesu se předají rodiči (pokud o to má zájem – např. čeká na ukončení potomka voláním wait)
 - Zdroje končícího procesu se uvolňují operačním systémem
- O ukončení procesu žádá jeho rodič (nebo jiný proces s dostatečnými právy), protože např.
 - Potomek překročil stanovenou kvótu přidělených zdrojů
 - Úkol přidělený potomkovi rodič již dále nepotřebuje
 - Rodič končí svoji existenci a nebylo povoleno, aby potomek přežil svého rodiče
 - Může docházet ke kaskádnímu ukončování (ukončí se celá větev stromu procesů)

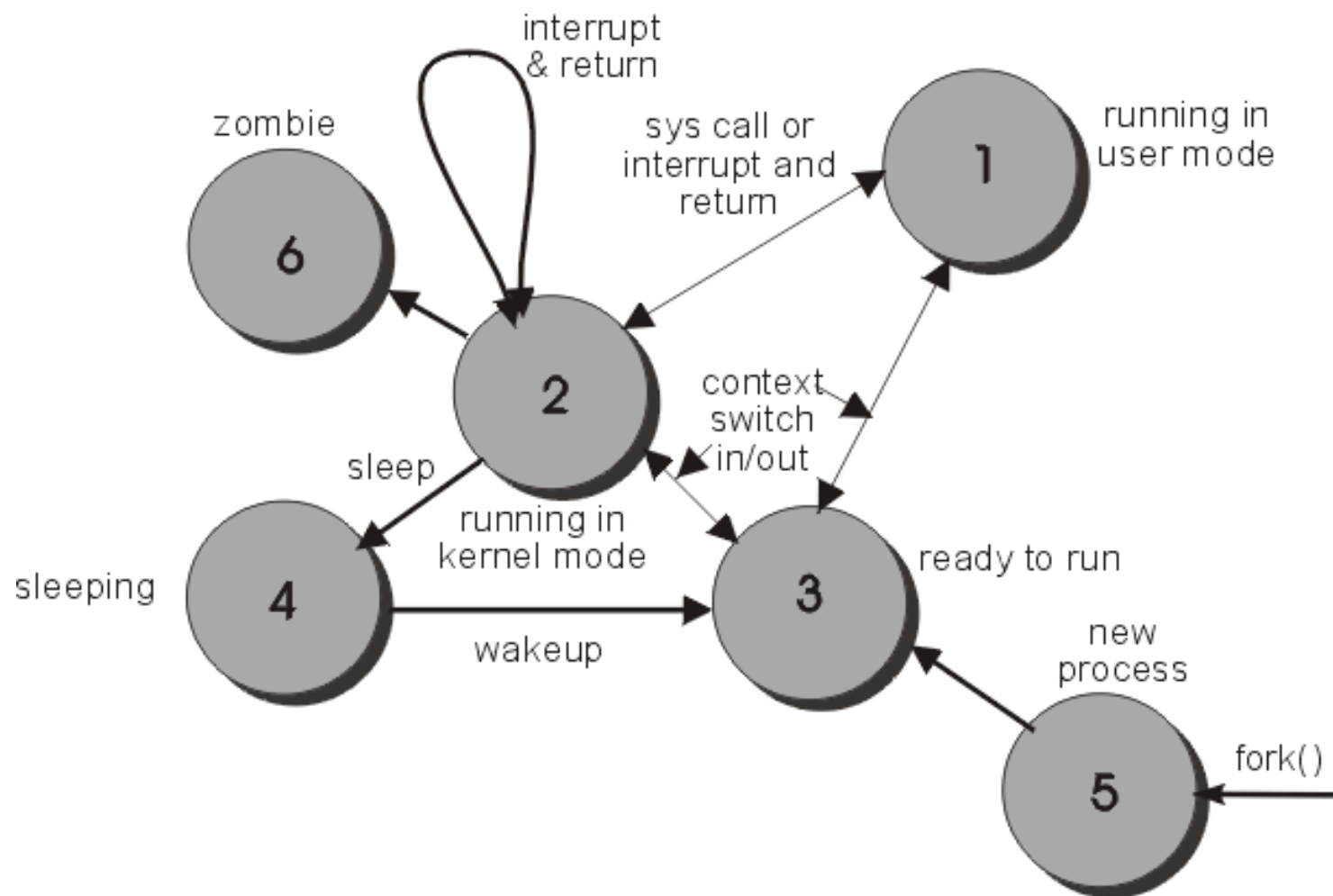
Příklad – Linux (1)

- volání `fork()` implementováno jako copy-on-write (tj. dokud paměť není měněna je sdílena a až při pokusu o modifikaci je vytvořena kopie)
- `vfork` – upravené `fork`, které nekopíruje stránky paměti rodičovského procesu
 - Rychlejší
 - Vhodné pro okamžité spuštění execve

Příklad – Linux (2)

- clone – upravené fork, které umožňuje sdílet některé zdroje (například paměť, deskriptory souborů, ovladače signálů) mezi rodičovským a nově vytvořeným procesem.
- Informace o procesu jsou uloženy ve struktuře `task_struct` (viz `usr/include/sched.h`)

Příklad – Linux (3)



Procesy a vlákna (1)

- Program
 - Soubor definovaného formátu obsahující instrukce, data a další informace potřebné k provedení daného úkolu
- Proces
 - Systémový objekt charakterizovaný svým paměťovým prostorem a kontextem (paměť i některé další zdroje jsou přidělovány procesům)
- Vlákno, také „sled“
 - Objekt, který vzniká v rámci procesu, je viditelný pouze uvnitř procesu a je charakterizován svým stavem (CPU se přidělují vláknům)

Procesy a vlákna (2)

- Model – jen procesy (ne vlákna)
 - Proces – jednotka plánování činnosti i jednotka vlastními prostředky
- Model – procesy a vlákna
 - Proces – jednotka vlastními zdroje
 - Vlákno – jednotka plánování činnosti

Procesy a vlákna (3)

- Každé vlákno si udržuje svůj vlastní
 - Zásobník
 - PC (Program Counter)
 - Registry
 - TCB (Thread Control Block)
- Vlákno může přistupovat k paměti a ostatním zdrojům svého procesu
 - Zdroje procesu sdílí všechna vlákna jednoho procesu
 - Jakmile jedno vlákno změní obsah (nelokální – mimo zásobník) buňky, všechna ostatní vlákna (téhož procesu) to vidí
 - Soubor otevřený jedním vláknem mají k dispozici všechna ostatní vlákna (téhož procesu)

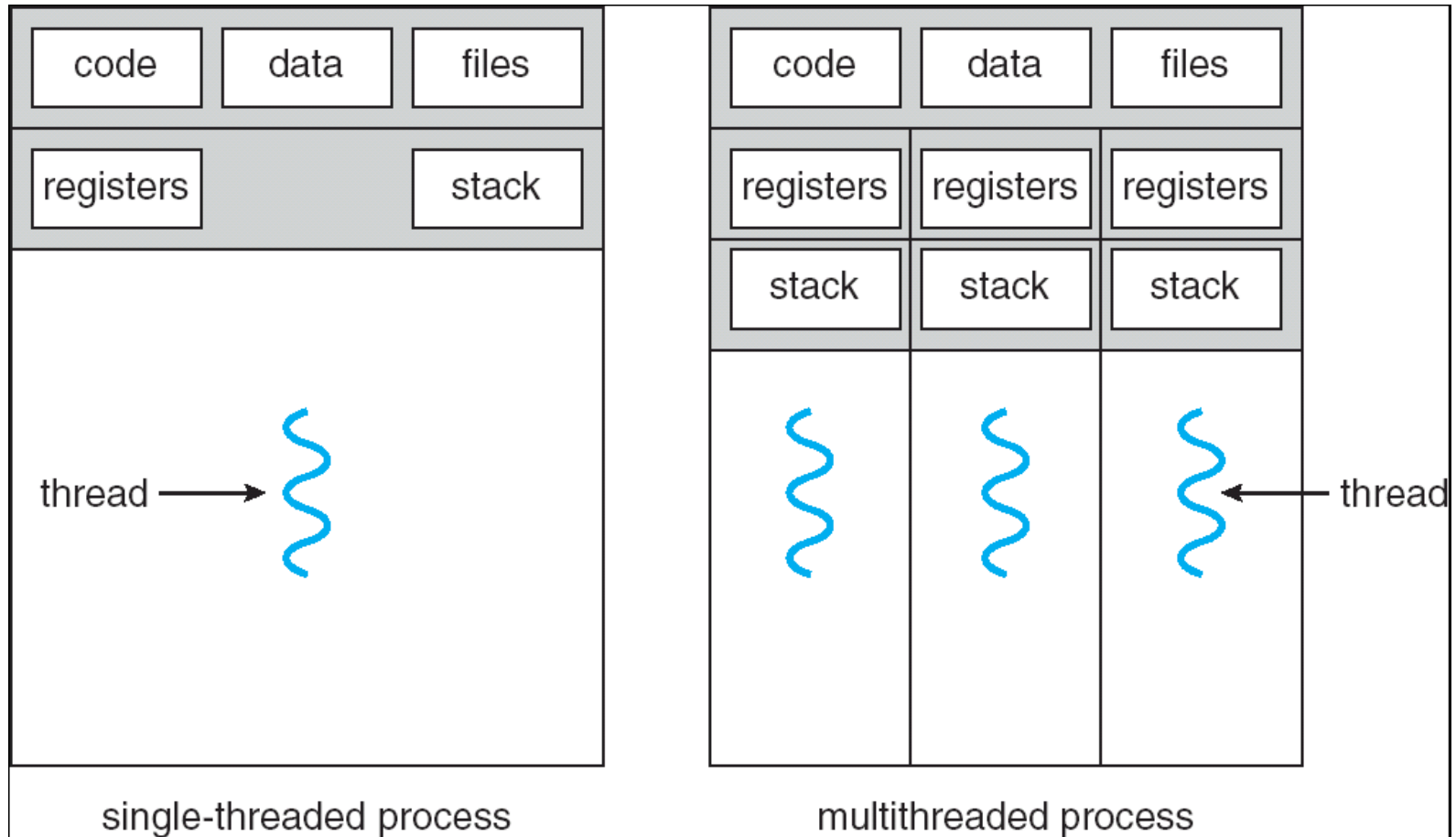
Procesy a vlákna (4)

- Proč využít vlákna
 - Využití multiprocesorových strojů (vlákna jednoho procesu mohou běžet na různých CPU)
 - Typický příklad
 - Jedno vlákno provádí uživatelem požadovaný úkol a druhé vlákno překresluje obrazovku

Procesy a vlákna (5)

- 1:1
 - UNIX Systém V, (MS-DOS)
 - Pojem vlákno neznámý, každé „vlákno“ je procesem s vlastním adresovým prostorem a s vlastními prostředky
- 1:M
 - OS/2, Windows XP, Mach, ...
 - V rámci jednoho procesu lze vytvořit více vláken
 - Proces je vlastníkem zdrojů (vlákna sdílejí zdroje procesu)

Procesy a vlákna (6)



Výhody využití vláken

- Vlákno se vytvoří rychleji než proces
- Vlákno se ukončí rychleji než proces
- Mezi vlákny se rychleji přepíná než mezi procesy
- Jednodušší programování (jednodušší struktura programu)
- U multiprocessorových systémů může na různých procesorech běžet více vláken jednoho procesu současně

Příklady využití vláken

- Síťový souborový (nebo i jiný) server
 - Musí vyřizovat řadu požadavků klientů
 - Pro vyřízení každého požadavku vytváří samostatné vlákno (efektivnější než samostatný proces)
- Jedno vlákno zobrazuje menu a čte vstup od uživatele a současně druhé vlákno provádí příkazy uživatele
- Překreslování obrazovky souběžně se zpracováním dat

Problém konzistence

- Program se skládá z několika vláken které běží paralelně
- Výhody
 - Když vlákno čeká na ukončení I/O operace, může běžet jiné vlákno téhož procesu, aniž by se přepínalo mezi procesy (což je časově náročné)
 - Vlákna jednoho procesu sdílí paměť a deskriptory otevřených souborů a mohou mezi sebou komunikovat, aniž by k tomu potřebovala služby jádra (což by bylo pomalejší)
- Konzistence
 - Vlákna jedné aplikace se proto musí mezi sebou synchronizovat, aby se zachovala konzistentnost dat (musíme zabránit současné modifikace stejných dat dvěma vlákny apod.)

Problém konzistence – příklad

- Situace
 - 3 proměnné – A, B, C
 - 2 vlákna – T1, T2
 - Vlákno T1 počítá $C = A+B$
 - Vlákno T2 přesouvá hodnotu X z A do B (jakoby z účtu na účet)
- Představa o chování
 - T2 dělá $A = A-X$ a $B = B+X$
 - T1 počítá konstantní C, tj. $A + B$ se nezmění
- Ale jestliže
 - T1 spočítá $A+B$ poté co T2 udělá $A = A-X$
 - Ale dříve než co T2 udělá $B = B+X$
 - Pak T1 nezíská správný výsledek $C = A+B$

Stavy vláken

- Tři klíčové stavy vláken
 - Běžící
 - Připravené
 - Čekající
- Vlákna se (samostatně) neodkládají
 - Všechny vlákna jednoho procesu sdílejí stejný adresový prostor
- Ukončení procesu ukončuje všechny vlákna existující v rámci tohoto procesu



Příklad – Win32 (1)

- Implementuje vlákna na úrovni jádra OS
 - Implementace je zdařilá, umožňuje mimo jiné paralelní běh vláken jednoho procesu na různých procesorech
- Služby OS
 - CreateThread
 - ExitThread
 - GetExitCodeThread
 - CreateRemoteThread (vytváří vlákno jiného procesu)
 - SuspendThread

Příklad – Win32 (2)

- Služby OS pokračování
 - ResumeThread
 - GetProcessAffinityMask (běh vlákna na procesorech)
 - SetProcessAffinityMask
 - SetThreadIdealProcessor
 - SwitchToThread (spustí jiný thread – je-li připraven)
 - TlsAlloc, TlsFree, TlsSetValue, TlsGetValue (thread local storage)