Kapitola 21

Operační systémy (státnice)

Zdroje:

- The Hebrew University in Jerusalem Lectures Notes: The Hebrew University in Jerusalem

see also Operační systémy (předmět)

21.1 Struktura operačního systému

- monolit (vše v kernel mode)
- mikrokernel (client-server model)
- hybridní kernel (wen:Hybrid kernel ... některé služby kompatibility u NT jádra běží v user mode)
- layered system
  - používáno v Multicsu, dnes už ne
  - 6 vrstev: alokace CPU, paměť, přístup ke konzoli, IO, user démony, operátor
  - každá vrstva požívala výhod abstrakce toho co bylo pod ní (paměťová správa se nemusela starat o CPU a tak...)
- virtuální stroje: uvnitř "virtual machine monitor", který dalším vrstvám poskytuje několik čistých kopií hardware. na tom pak může běžet víc různých OS

architektura mikrojádra, abstrakce poskytované mikrojádry

- v kernel mode jen address space management, přepínání vláken a ipc
- zbytek (filesystémy, síťové protokoly, ovladače zařízení) v userspace serverech, jen mají přístup k paměťovým rozsahům svých specifických zařízení
- když nějaký server zhučí dá se restartovat

`http://www.tldp.org/HOWTO/html_single/Implement-Sys-Call-Linux-2.6-i386/#AEN22` syscalls `http://www.cs.purdue.edu/homes/cs354/LectureNotes/CS354-part3.pdf`

21.2 Virtuální stroje

Původně pro IBM/360 – řešili multitasking pomocí virtuálního stroje. Dnes simulují buď celý systém (VMWare), nebo jen prostředí pro jednu aplikace (Java Virtual Machine, .NET Common Language Runtime). Použití pro abstrakci hardware, izolaci (např. při hostingu), provozování více OS na jednom počítači.

wen:Popek and Goldberg virtualization requirements: virtualizace je možná, jen pokud jsou všechny instrukce které závisí na stavu procesoru (např registrech ochrany paměti) nebo ho mění privilegované – tj. všechny instrukce které by se mohly tlouci předají nejdřív kontrolu hypervisoru, který je může simulovat

Podmínky splňuje například IBM/360. x86 tyto podmínky nesplňuje, hodně neprivilegovaných instrukcí je citlivých – odhaluje stavy procesoru, nejde například simulovat privilegovaný mód v user mode. Tohle poprvé vyřešil VMWare dynamickým překladem, Intel i AMD představily rozšíření pro IA-32/64 i AMD64, které tohle řeší.

Softwaru, který se stará o to, aby aplikace, které běží v guest OS, žily v iluzi, že běží na skutečném stroji a ne na tom virtuálním, se říká hypervisor nebo také VMM (Virtual Machine Monitor).

Hypervisor může být typu I nebo typu II.

- Typ I běží přímo na skutečném HW. Při jeho implementaci je tedy třeba starat se o hodně low-level věci a sahat přímo na železo (není k dispozici žádná abstrahující vrstva). Na druhou stranu je to nejrychlejší (ve smyslu efektivity) způsob, jak dělat virtualizaci.
- Typ II je od skutečného HW oddělen operačním systémem a běží v něm jako (víceméně obyčejná) aplikace. Takže je o něco snazší ho napsat (máme k dispozici vrstvu, která abstrahuje úplně low-level věci), ale každá citlivá instrukce, která vyvolá trap, se musí propagovat přes víc vrstev, a je to tedy pomalejší než typ I.

Docela pěkně je to popsané tady: OK Labs virtualization
Asi by bolo dobré vedieť povedať aj niečo napr. o JVM — ako tam vyzerajú inštrukcie (srandy ako výroba objetkov? :-)), JIT atď...

21.3 Správa procesů a vláken, plánování

- vlákno má kontext procesoru: stavy registrů, stack
- proces má kontext paměti (adresový prostor, data v něm), a prostředí: terminál, otevřené soubory, env, ...

proces může mít víc vláken, plánují se vlákna.
plánovač určuje, které vlákno pustit dál. typické metriky:

- doba odezvy (u interaktivních procesů)
- propustnost (co nejvíc dokončených úloh za jednotku času)
- využití procesoru (neflákat se)
- spravedlnost

off-line plánování: mám pevný počet procesů, vím jak dlouho poběží, nepřerušuji je

- first come first served
- shortest job first: minimalizuje průměrnou dobu odezvy

on-line plánování

- procesy se objevují neočekávaně, doba běhu neznámá
- plánuje se podle priority, vázanosti na IO, interaktivity, chování v minulosti (výpadky stránek)
- preemptivní: potřebuje podporu HW (časovač), možnost měnit plán na základě nových informací
- context switch – přepnutí na jiný proces/vlákno

metody:

- first come first served: nepreemptivní
- round robin: střídám vlákna v časových kvantech, pak přepínám – spravedlivé
- prioritní s více frontami a zpětnou vazbou: některé úrovně FIFO, jiné RR, přesouvám podle toho jak dlouho už běžely

pro více CPU:

- každý procesor má vlastní frontu
- mezi nimi vyvažování zátěže, zohlednění vztahu procesů

real-time systémy:

- běh omezen reálným časem dokončení – deadline

metriky (doba odozvy / spravodlivost / vyuzite cpu / priepustnost) inverzia priorit
multiCPU

- (ne-)distribuovane planovanie
- zviazanost procesov s cpu

realtime planovanie (soft/hard)

21.4 Komunikace a synchronizace procesů, kritické sekce, synchronizační problémy a primitiva

Komunikace – posílání zpráv, přístup ke sdíleným datům
kritické sekce

- část programu která potřebuje exkluzivní přístup ke sdílenému prostředku (typicky paměti)
- (v kernelu část která nemůže být přeplánovaná nebo přerušena)

Synchronizační primitiva:

- semafor: celočíselná proměnná + fronta čekajících procesů... může reprezentovat počet volných/přidělených prostředků
- fronty zpráv: operace send + receive, receive blokuje když není zpráva
- monitor: konstrukce programovacího jazyka, ošetřující kritické operace; v monitoru může být jen jeden proces najednou
- ^ vzájemně ekvivalentní

synchronizační problémy:

- obědvající filozofové
- producent-konzument
- holič

memory model (kompilatory + optimalizacie multithread apliacii)
pipe / signaly (len jeden thread procesu — pthread_sigmask())
posielanie sprav, zdielanie pamati
kriticke sekcie (blokovanie ineho mimo CS, nekonecne cakanie, 1 proces v CS, bez predpokladov o cpu)
metody bez primitiv (dosledne striedanie, petersnovo riesenie)
aktivne cakanie (TSL, zakazanie prerusenia) | pasivne

- mutex, semafor, RWL, monitor, condition variable (wait(+mutex)|signal|bcast)
- fronty sprav (receive zablokuje bez spravy)

convoys, priority inversion, starvation & deadlock
nonblocking synchronizacia, okrem ferovosti aj

- wait free (vsetky skoncia v konecnom case)
- lock free (niektore)
- obstruction free (kazdy po konecnom pocte krokov — ak ostatne nic nerobia)

http://en.wikipedia.org/wiki/Lock_convoy

uváznutí a jeho řešení

Coffmanovy podmínky

1. Výlučný přístup: prostředek je přidělen výhradně jednomu procesu
2. Neodnímatelnost: prostředek nejde odejmout
3. Drž a čekej: proces může zároveň držet prostředek a čekat na další
4. Kruhová závislost: procesy čekají na prostředky v kruhu

Prevence deadlocků: napadení jedné z Coffmanových podmínek

- výlučný přístup: spooling (např. u tiskáren)
- neodnímatelnost
  - jen tam kde jde bez následků (procesor, paměť)
  - většinou nejde bez selhání procesu

- drž a čekej: nutno žádat o všechny najednou, nebo před další žádostí dosavadní prostředky uvolnit
- kruhová závislost
  - očíslování prostředků, pak jde žádat jen o prostředky s vyšším číslem

Předcházení: zabránit realizaci všech podmínek

- bankéřův algoritmus: vím, co bude proces max. chtít k dokončení, bezpečný stav: jsem schopen plně uspokojit alespoň 1 proces, on mi pak svoje prostředky vrátí
- složité rozhodování
- informace typicky nejsou dostupné

Detection & recovery: řešení deadlocků, až nastanou

- graf závislostí + hledání cyklu
- pak odebrání prostředku pod dohledem operátora
- nebo přerušení cyklu zabitím procesu
- recovery:
  - os ukládá stav procesů, při deadlocku se vrátí k savepointu a pustí je s jiným naplánováním
  - transakční zpracování: abort + pustit znovu (typické pro databáze)

pštrosí algoritmus:

- předstíráme, že problém neexistuje
- když už nastane, vyřeší ho za nás uživatel

21.5 Podpora multiprocesorových systémů

- SMP: víc procesorů, stejná paměť (multi-core procesory jsou taktéž SMP, ne vždy však ekvivalentní více procesorům — například u intel core sdílí jádra L2 cache, u některých architektur dokonce některé koprocesory)
- NUMA: víc procesorů, paměť je ale pro každý procesor jinak dostupná (někde pomalejší)
- hyperthreading: jedno jádro, zdvojené části co berou instrukce, takže se jeví jako dva

cache coherency: zajištění, že procesor namá ve své cache zastaralý obsah (zneplatněný jiným procesorem)

1. co procesor zapsal, může i přečíst, pokud tam někdo jiný mezitím nezapsal (chceme vždycky)
2. pokud mezitím do paměti zapsal procesor A, můžou to hned přečíst i ostatní CPU
3. zápisy musí být v daném pořadí: pokud do jednoho místa zapíšu hodnotu A a potom hodnotu B, čtení z kteréhokoli CPU mezi tím musí nejdřív ukázat A a pak B, nikdy naopak

postupy:

- snooping (bus sniffing): řadič cache kouká na sběrnici, co kdo píše do paměti, podle toho si invaliduje svoje záznamy (cache buď musí být write-through, nebo se použije nějaký model níže)
- snarfing: jako předchozí, ale rovnou si podle zápisů upravuje svoje cachovaná data
- directory-based: centrální seznam cachovaných bloků (pro velké systémy nad 64 CPU, kde není centrální sběrnice)

`http://www.cmpe.boun.edu.tr/courses/cmpe511/fall2004/Mehmet%20Senvar%20-%20Cache%20Coherence%20Protocol`
`ppt`

modely:

- MSI – stavy Modified/Shared/Invalid
  - čtení: M/S – dodá data, pokud je v I, tak se ověří jestli jinde není M (ta pak musí zapsat a přejít do S nebo I)
  - zápis: M – modifikuje; S – musí říct ostatním S, ať si to vyndají; I – ostatní S musí vyndat, M zapsat
- MOSI

  - přidává stav Owned, kdy cache vlastní daný blok a dodává data ostatním cache

- MESI – Modified (jen v této cache), Exclusive (jen v této cache, asi odpovídá hlavní paměti), Shared (může být i jinde), Invalid
  - jde psát jen ve stavech M/E
  - cache v M/E stavu musí poslouchat na všechna čtení a dodávat svůj obsah (M), nebo přejít do S (E)
  - Read For Ownership (RFO) – broadcast signál indikující přechod cache ze stavu S do E – ostatní si musí blok přehodit do I

- MOESI
  - Modified: má aktuální data, ty v hlavní paměti jsou stará, jiné cache nemají nic
  - Owned: má aktuální data, ty v hlavní paměti jsou stará, ostatní cache mohou mít data v Shared stavu
  - Exclusive: má aktuální data, totožná s hlavní pamětí, jiné cache nemají nic
  - Shared: má aktuální data, můžou je mít i ostatní cache; aktuální data jsou i v hlavní paměti, leda že by je měla nějaká cache v Owned stavu
  - Invalid: nemá nic, aktuální jsou buď v hlavní paměti nebo v jiné cache

NUMA

Bez cache coherence problematické použití, proto se používá cache-coherent NUMA (ccNUMA), kdy radiče cache mezi sebou zajišťují koherenci. To zpomaluje zejména v situacích kdy na stejné místo sahá víc procesorů rychle za sebou, takže operační systémy co to podporují alokují paměť a procesory tak se takový provoz minimalizoval.

21.6 Mechanismus přerušení v OS

see also Architektura počítačů a sítí#Přerušení

Přerušení předchází nutnosti pollovat zařízení. Místo toho dá zařízení signál procesoru, ten předá řízení OS, který uloží stav aktuálního procesu, spustí obsluhu přerušení (zjištění podle přiřazeného IRQ kanálu, pak se pouštějí rutiny pro ovladače co tak jsou), a nakonec se vrátí k původní činnosti.

Přerušení se dají maskovat, aby nerušily (např. při obsluze jiných přerušení).

Přerušení mohou být:

- asynchronní: vyvolané vnějším zařízením
- synchronní: vyvolané přímo procesorem
  - výpadek stránky, systémové volání (trap)
  - chybná instrukce, dělení nulou (exception)

Aby ošetření přerušení nezdržovalo a zbytečně se neblokovaly další události, je obsluha rozdělena na samotnou obslužnou funkci, která udělá to nejnutnější, a další zpracování (bottom half, softirqs, tasklets v Linuxu, deferred procedure calls ve Windows), která se jen naplánuje na později a třeba dál zpracovává příchozí data pro aplikační programy.

DMA

see also Architektura počítačů a sítí#DMA (Direct Memory Access), wen:Direct memory access

Direct memory access: Přenos dat mezi zařízením a pamětí se odehrává bez účasti procesoru, ten k němu akorát dá pokyn a na konci dostane přerušením info že už je hotovo. Procesor se tak nejen nemusí otravovat s kopírováním, ale ani nemusí na zařízení čekat. Kopírování pak provádí buď DMA řadič (ISA sběrnice), nebo zařízení samo (bus mastering u PCI). (Při DMA je třeba dát pozor na cache procesoru.)

Dříve DMA řadič čekal na signál procesoru že je volná sběrnice, eventuálně číhal kdy ji nevyužije nebo procesor úplně uspával. Dnes už jsou sběrnice pro procesor/hlavní paměť (northbridge) a periferie (southbridge) oddělené`https://dsrg.mff.cuni.cz/pipermail/osy/2005-February/000148.html`.

21.7 Správa periferií, ovladače zařízení

OS kontroluje periferie a zařízení, kód který je řídí se označuje jako ovladače zařízení. Umožňuje přistupovat k zařízení více aplikacím, a zároveň poskytuje abstrakci přístupu k nim.

Driver se skládá ze synchronní části, volané když aplikace po zařízení něco chce, a části asynchronní, volané když přijde přerušení od zařízení. Tyto část mezi sebou komunikují pomocí front a bufferů – aby se v přístupu netloukly, dovoluje operační systém obsluhu přerušení odložit na později (bottom halves etc `http://dsrg.mff.cuni.cz/~ceres/sch/osy/text/ch04s01s01.php`)

API: blokující funkce, asynchronní signalizace, signalizace chyb.

V Unixu zařízení rozdělená na bloková zařízení (náhodný přístup k adresám, cache, ...) a znaková (jen čtení/psaní, bez cache). V Linuxu výpis zařízení podle sběrnic, namatchování k driverům.

Ovladače se dají připojovat z běhu systému (.ko u Linuxu, .sys u Windows).

Scatter/Gather (vectored) I/O `http://en.wikipedia.org/wiki/Vectored_I/O`

21.8 Správa paměti, hierarchie pamětí, segmentace, stránkování, strategie alokace, odkládání

see Architektura počítačů a sítí#Paměťová hierarchie, vyrovnávací paměti

see Architektura počítačů a sítí#Stránkování

see Architektura počítačů a sítí#Segmentace

TLB – asociativní paměť, cachuje položky stránkovacích tabulek; při přepínání adresových prostorů nutno splachovat `http://www.informit.com/articles/article.aspx?p=29961&seqNum=4`

PAE — `http://www.tldp.org/HOWTO/html_single/Implement-Sys-Call-Linux-2.6-i386/#AEN221` (aj s ukazkou strankovacich tabuliek pre rozne kombinacie noPAE/PAE/4kB/4MB pages)

21.9 Sdílení paměti mezi adresovými prostory, paměťově mapované soubory

V Linuxu obsahuje blok sdílené paměti (shmem) seznam všech procesů, které jej mají připojený. Když se proces pokusí paměť použít, dojde k výpadku stránky, při jehož řešení buď alokuje novou stránku, nebo použije už alokovanou, existuje-li `http://www.linux-tutorial.info/modules.php?name=MContent&pageid=293`. Když se stránky vyswapuje, tak je potřeba opravit tabulky stránek pro každý proces který ji používá `http://www.tldp.org/HOWTO/html_single/Implement-Sys-Call-Linux-2.6-i386/#AEN220`

Při forku je paměť procesů jen označená jako sdílená, pak se používá copy-on-write.

mmap: soubor se namapuje do paměťového prostoru aplikace, a označí se jako vyswapovaný. Pak při přístupu do paměťi dojde k výpadku stránky, kus souboru se nakopíruje do rámce a v paměti a používá se.

pomenovanie zdielanych oblasti

mmap / swap

- problemy (velkost suboru (nezaokruhlena na napr. 4kB) vs. velkost stranok)

21.10 Souborové systémy, souborové a adresářové služby, síťové souborové systémy

Souborové systémy dovolují přistupovat k v podstatě lineárnímu prostoru na zařízení jako ke stromu adresářů a souborů, řeší přístupová práva, rozmístění souborů na disku a tak. Soubory mají různé atributy, moderní filesystémy mají žurnálování, aby se předešlo nutnosti všechno kontrolovat při výpadku. Příklady: FAT, ext2, NTFS.

Souborové a adresářové služby: ?

Síťové souborové systémy: problémy se zamykáním a stavovostí

see also Distribuované systémy#NFS, AFS, CODA, ...

- NFS – funguje přes RPC, je bezestavový, problémy většinou ignoruje; NLM – zamykání; od verze 4 NFS stavový
- SMB
- AFS – hodně serverů v jedné struktuře, lokální cache (kde se provádí všechny změny, po zavření se to nakopíruje zpátky); server o cache ví, a když se v souboru něco změní tak klienta upozorní
- Coda – dá se operovat i off-line, replikace, odolnost proti výpadkům, občas nutno řešit konflikty

LDAP

- DN (CN + RN), atributy (+schema), X.500, TLS...

open => syscall

- copyfn z userspace
- najst volny fd
- filp_open
  - open_namei => dentry
    - zacne v root/pwd (fsystem+dentry)
    - prehladava dentry_cache, potom realdentry (disk?), pokial moze, po komponentoch
    - (prechod medzi fs?)
  - dentry_open(dentry, mnt)
- ulozit fd do files

21.11 Informační bezpečnost a základy šifrování

ruzne crypto-loopy, ssl, pam, acl, capabilities a tak
hash
sifrovanie

- asymetricke (RSA, diffie-hellman)
- symetricke

podpisovanie
autentikacia / autorizacia

21.12 Síťové služby OS

sockety, RPC.
Při zpracování paketů je žádoucí zabránit zbytečnému kopírování – nechávat místa okolo pro hlavičky.

- packet filtering – iptables a kamarádi
- packet scheduling – různé fronty na výstupu (SFQ, priority, HTB)

Aplikace: síťové filesystémy, load balancing, clustery
sockety

- socket+bind / connect
  - PF_UNIX, PF_INET, PF_NETLINK; SOCK_DGRAM, SOCK_SEQPACKET, SOCK_RAW
- listen(backlog) + accept
- send(to,msg)/recv(from,msg)
- select(rd,wr,xcept,timeout), poll(fds,timeout)

RPC

- XDR => skeletony+stuby (rpcgen)
- portmapper (cislo sluzby+verzia => port)

kopirovanie paketov (=znizeny vykon)

- data dispatching / smart hardware

iptables (dorucovanie, forwardovanie, zahadzovanie paketov, maskarada)
scheduling

- Stochastic Fair Queue (per process queue+round robin; hashovanie do queues+obcas zmena h)
- token bucket

  - ked treba ohlidat bandwidth; flowu priradeny bucket
  - odoberanie tokenov ked odosiela data, pravidelne pridavanie
  - velkost bucketu = fluctuation limit
- class based ???
- random early detection
  - pre TCP a spol. — ked sa zaplna, pakety su zahadzovane s vacsou pst. => lepsi flow control (early warning)

sietove filesystemy / load balancing / clustery
NFS

- architektura (protokoly, hlavne operacie)
- obsah file handle v NFS
  - fs identifier, node #, generation # (nedostupne z userspace) => export iba celeho FS — nie subtree
- close-to-open cache consistency
  - uplne sync narocna => nerobi sa
  - pri zatvarani suboru sync, close() moze vratit chyby servera
    * potom sa vezme getattr() — ak rovnake ako pri dalsom open, cache sa vyuzije (inak zahodi)
  - cache atributov a/alebo dat sa da zakazat
- problemy bezstavovosti (testovanie pristupovych prav, mazanie otvorenych suborov, zamykanie suborov) + riesenia
  - read pomocou gen#
  - append — malo casty, nedolezity