

1) Základní architektura počítače, reprezentace čísel, dat a programů

- binární reprezentace

- různé datové typy → fixní # bitů

př. int = 32-bit

- znaky:

ASCII → 8 bitů (původ), podmnožina Unicode, anglické znaky

Unicode → všechny znaky (>140 tis.)

UTF-8 → 1B ASCII, 2-4B znak, UTF-16 → 4B / # znak

endianita

→ pořadí ukládání bytů ve vícebytových datech (např. int)

→ little end / big end 0 jako vejíčko

little end = least significant byte na nejnižší

(big end = -na nejvyšší- byte na nejnižší adrese)

většina procesorů

př. 0x1A2B3C4D

LE: 4D 3C 2B 1A

BE: 1A 2B 3C 4D

Adresa

→ každý byte v paměti má svou jedinečnou adresu

→ většinou v hexadecimálním formátu

Adresový prostor

→ rozsah adres, ke kterým může procesor přistoupit

→ určen šířkou adresní sběrnice (př. 32-bit, 64-bit)

32-bitová adresová sběrnice → 4GB adresový prostor (2^{32})

Adresace dat v paměti

- direct memory access (DMA)
 - bez intervence CPU
 - většinou pro velký přenos dat (pr. I/O na disk)
- hierarchie paměti:
 - různé typy paměti s různou rychlostí přístupu a různou velikostí
 - nejrychlejší: CPU registry → cache → RAM → disk storage
- zarovnávání dat v paměti, aby odpovídaly velikosti dat
 - optimalizuje access speed (4B int na adresu = násobek 4)
- virtuální paměť:
 - abstrakce, díky které mohu používat velký souvislý úsek paměti
 - virtuální adresy jsou namapovány na fyzické adresy paměti

Jednoduché datové typy:

- int = fixní # bitů (běžně 32 nebo 64)
- float

normalizovaný format: sign. 1,01101 · $2^{10001001}$

1 1000 1001 01101
↑ exponent mantisa
 $0=+, 1=-$ ← sign

mantisa = nese hodnotu čísla

(1 před čárkou je vždy → můžu ignorovat)

$$\text{pr. } 3,5 = 11,1 = 1.11 \cdot 2^3$$

$$\rightarrow 011111$$

- boolean → 1 bit 0/1

→ často 1B kvůli zarovnání

Složené datové typy

- pole

- souvislý blok paměti s daty stejného typu

- struktura

- množina proměnných pod stejným názvem

- v paměti uloženy sekvenčně

- k prvkům může obsahovat padding kvůli zarovnání

- třída

- podobná strukturám, ale obsahuje i metody

- navíc pointery na virtuální tabulku metod

- pointer

- adresa v paměti pro jinou proměnnou

- velikost podle architektury (př. 32-bit, 64-bit)

alokace paměti

- statická

- velikost a umístění paměti pevně dané při komilaci

- dynamická

- mění se za běhu

Aritmetické operace

sčítání, odčítání, násobení, dělení, modulo

Logické operace

AND, OR, NOT, XOR

Bitové operace

bitové AND, OR, NOT, XOR, shift dleva / doprava

! přetěžení např. při sčítání → výsledek > kapacita paměti

! přesnost operací s floaty 0.1 + 0.2 nemusí být přesně 0.3

2) Instrukční sada.

vazba na prvky vyšších programovacích jazyků

instrukční sada

= mužina instrukcí, které procesor podporuje

Programové konstrukce

• přiřazení

→ přesunutí dat z adresy na jinou

→ MOV destination, source

prí. int A = 10

int B = A

↑ → registr A

MOV A, 10

MOV B, A

• podmínka

→ if / else

→ porovnávací instrukce např. CMP

podmíněný skok : JE → jump if equal

JNE → jump if not equal

prí. if (A == B) { C = 1; }

else { C = 0; }

↑

CMP A, B

JE EQUAL

MOV C, 1 ← if not equal

JMP END

EQUAL : MOV C, 1

END :

◦ cyklus

→ for loop (specificky # opakování)

- initialize counter

CMP pro porovnání

JL jump if less

inkrementace counteru

př. for (int COUNTER = 5; COUNTER > 0; COUNTER -)

{ //loop body }

↑

MOV COUNTER, 5

LOOP-START: CMP COUNTER, 0

JE LOOP-END

/* Loop-body */

DEC COUNTER

JMP LOOP-START

LOOP-END:

→ while loop (dokud platí podmínka)

- CMP pro condition check

- JNE jump if not equal

◦ volání funkce

CALL → skok na funkci

RET → návrat z fce

př. function()

void function(){}

// do sth

}

↔

CALL: FUNCTION

FUNCTION: // do sth

RET:

3) Podpora pro běh operačního systému

privilegovaný režim procesoru

- = operační mód, ve kterém má procesor plný přístup ke všem systémovým prostředkům
- také systémový nebo kernel režim
- běží v něm OS nebo jeho část

neprivilegovaný režim procesoru

- uživatelský režim
- limitovaný přístup k registrům a instrukcím

přepínání mezi režimy

- při interruptu nebo výjimce procesor přepne do systémového režimu, aby to vyřešil
- uživatelské aplikace používají systémovou volání, aby požádaly služby OS, který se přepne do systémového režimu, aby požadavek splnil

jadro operačního systému

- kernel
- spravuje systémové zdroje a hardware
- most mezi hardwarem a softwarovými aplikacemi
- architektury jádra:
 - monolitická
 - 1 velká hromada, žádna struktura
 - celý v privilegovaném režimu (pr. Linux)
 - ④ rychle, efektivně využít zdrojů a kódů
 - ⑤ nedají se schovat data

- vrstevnatá

- hierarchie vrstev

- n-ta vrstva používá (pouze) služby podporované (n-1). vrstvou

- jednodušší na rozšiřování a správu

- obtížné na nauhnutí

- microkernel

- v chráněném režimu běží minimální část

- rychlá komunikace mezi aplikacemi

- kernel předává zprávy (klient/server)

- ⊕ snadno rozšířitelné, bezpečné

- ⊖ těžké zabit systém (rychle se restartuje)

- ⊕ pomalejší, posílání zpráv nahrazuje fc (Windows)

- scheduling procesů

- správa paměti

- Správa zařízení

- Správa souborů / souborových systémů

- networking

4) Rozhraní periferických zařízení a jejich obsluha

PIO = programmed input / output

zařízení = věc pro speciální použití, slouží ke konkrétní činnosti

řadič zařízení

- elektricky připojuje zařízení k počítači
- řadič zařízení do topologie
- HW

ovládač zařízení

- SW, část OS
- prostředník mezi zařízením a procesorem
- spravuje přenos dat

programem řízená obsluha zařízení (PIO)

- myš

- I/O : PS/2 nebo USB rozhraní

- většinou : IO port 0x60 pro data

- 0x64 pro status / protokol

- čtu polohy myši a detekuju stisk tlačítka
z datového portu

- příklad: READ_MOUSE : IN 0x60, EAX

- → přečtení dat z myši do EAX registru

- disk

- I/O : často porty 0x1F0 - 0x1F7 pro data a
kontrolní signály

- posílám příkazy na ovladač pro čtení/zápis dat

- správa převodu do/z sektorů disku

pr.

READ-DISK: OUT 0x1F2, SECTOR-COUNT

→ zápis # sektorů k přečtení do disku

OUT 0x1F3, SECTOR-NUMBER

→ zápis čísla sektoru, kde začínám číst

OUT 0x1F7, READ-COMMAND

→ poslání čtecího příkazu disku

interrupt

→ signalizace pro procesor, že zařízení vyžaduje pozornost
nebo že se dokončil přenos dat

→ redukuje čas, kdy procesor čeká (idle stav)
→ dovoluje multitasking



reakce hw:

- CPU acknowledge interrupt request a určí jeho zdroj
- CPU uloží svůj aktuální stav (aby ho pak obnovil)
- CPU přeruší aktuální proces a provede relevantní interrupt service routine
- CPU obnoví uložený stav a pokračuje v přerušeném procesu

reakce sw:

- zpracovává interrupt requesty přes své interrupt handling mechanismy
- prioritizuje tasky na základě IR
- zařizuje, aby byly time-critical operace zvládnutý rychle
- komunikuje s jednotkou, která zpracovává IR specifika

př.

disk pošle interrupt při dokončení read/write operace

CPU:

- uloží svůj stav a přeruší proces
- provede ISR, která typicky obsahuje rutiny OS pro zpracovávání I/O disku
- obnoví stav a pokračuje v procesu

OS:

- řídící disk vyřeší přenos dat z/na diskové buffery
- může dať čekajícímu procesu signál, že I/O operace byla dokončena

5) Základní abstrakce, rozhraní a mechanismy OS pro běh programů, sdílení prostředků a in/out

neprivilegované (uživatelské) procesy

- operují s omezenými systémovými permissions
- nemůžou přímo přistupovat k HW nebo zdrojům jádra
- přistupuje ke zdrojům pomocí systémových volání, které pak zprostředkovává jádro

Sdílení procesoru

proces

- = nezávislé exekuční entity v OS
- "užívající program" (ozivený nějakým systémovým voláním)
- OS eviduje prostředky nazívané na proces
 - paměť, otevřené soubory, zařízení
- př. webový prohlížeč běžící jako oddělený proces

vláknko

- menší jednotky provedení procesu
- sdílí paměť a zdroje procesu
- objekt uvnitř OS
- při spuštění 1 vlákna
 - to může vytvořit další vlákna
 - třívláknový zásobník

př. H karta prohlížeče je jiné vlákno v procesu

přepínání kontextu:

- = proces uložení stavu běžícího procesu nebo vlákna, aby mohl běžet jiný proces/vláknko
- store/restore kontextu

Kontext

= stav jednotky plánování

proces: všechny informace, které OS potřebuje

k managementu procesu

→ program counter, registry, stav procesu,
pointery do paměti

uložení když proces není actively executing
(při přepínání kontextu)

vlákno: podmnožina kontextu procesu

zásobník, registry, stav vláknha

rychlejší přepínání kontextu než procesy

Multitasking

• kooperativní

→ OS nedělá samy context switch

→ task se sám musí vzdát CPU (kooperovat)

→ context switch když task odevzdá CPU

⊕ jednoduchá implementace

tasky mají kontrolu nad svou exekucí

⊖ 1 nekooperující proces může zablokovat celý systém

• preemptivní

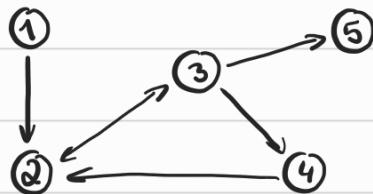
→ V plánovací jednotce dostane čas, který má právo běžet (time slice)

→ po uběhnutí času je task přerušen

→ OS provede context switch

Stavy vlákna / procesu:

1. Created → čeká na spuštění
2. Ready → čeká na naplánování a přiřazení procesoru
3. Running → executing
4. Blocked → čeká na zdroje nebo I/O
5. Terminated → skončený



Plánování

- A proces má release time a deadline

release time = kdy musí proces začít poté, co nastala nějaká událost

deadline = do kdy musí dokončit

hard → po něm nemá smysl pokračovat

soft → může zkoušet dopočítat i když ho nestihne

- priorita

↳ statická = nastavena na začátku

↳ dynamická = průběžně se mění (časem se zvyšuje)

↳ součet těch dvou

virtuální paměť

6) Parallelismus, vlákna a rozhraní pro jejich správu, synchronizace vláken

race-conditions (časově závislé chyby)

- 2 nebo více vláken přistoupí ke stejné paměti
a snaží se ji měnit ve stejnou chvíli
- způsobena špatnou synchronizací přístupů k paměti
- výsledek záleží na postupnosti nebo načasování
execuce daných vláken

- kritická sekce = část kódu, kde vlákna přistupují
ke sdíleným zdrojům
 - potřeba synchronizace pro zabránění vzniku
konkurenčního přístupu (může vést k RC)

vzájemné vyloučení

- = koncept zajišťující, že v kritické sekci vždy
probíhal pouze 1 vlákno
- mutexy, zámky, semafory

zámky

- = synchronizační primitivum, které kontroluje přístup
ke sdíleným zdrojům více vláken
- zajišťuje, že v určité čas může provádět kritickou
sekci kód pouze 1 vlákno

- vlákno vstoupí do sekce → zamkne ji
ostatní vlákna čekají na odemčení
- operace se zámkem: acquire a release

čekání

- aktivní (busy-waiting / spinning)
 - vlákno se neustále dívá, zda je podmínka splněna' nebo zdroj dostupný
 - pro případy s krátkým očekávaným wait-time
 - náročnější na CPU ale okamžitá odezva
- pasivní
 - uspání vláknů
 - uvolnění CPU pro jiná vlákna
 - vlákno se vzbudí, když je splněna' podmínka nebo zdroj dostupný
 - efektivnější využití CPU