Curs 05: Gestiunea memoriei

Sumar curs anterior

mai multe procese concurează pe procesoarele sistemului; planificatorul (parte a sistemului de operare) gestionează accesul proceselor la procesoare (trecerea în starea RUNNING); planificarea unui proces, adică înlocuirea unui proces cu altul se numește schimbare de context
schimbarea de context se poate produce:
* voluntar: un proces decide sau cauzează schimbarea de context
** un proces cedează de bună voie procesorul: yielding
** un proces execută o operație blocantă (de exemplu citire de la un dispozitiv care nu are încă date)
** un proces își încheie execuția
* nevoluntar: planificatorul sistemului de operare decide forțarea unui proces de pe procesor
** un proces a stat prea mult timp pe procesor
** apare în sistem un proces mai important
planificatoarele pot fi cooperative și preemptive; cele cooperative nu pot preveni apariția situației de starvation
planificatorul urmărește două obiective conflictuale: productivitate și echitate
planificatoarele preemptive folosesc cuantă de timp și prioritate pentru fiecare proces
comunicarea intreprocese presupune transfer de mesaje, memorie partajată, notificare sau sincronizare

Procese și memorie

într-un sistem de calcul modern, acțiunile sunt realizate în cadrul unui proces care abstractizează procesor, memorie, I/O
executabile încărcate în memorie: date statice, cod; apoi date dinamice în memorie
memoria proceselor trebuie să fie izolată
ideal procesele au câtă memorie își doresc

Funcționarea unui sistem de calcul. Rolul memoriei

la nivel scăzut acțiunile sunt realizate de procesor prin interacțiunea cu memoria și I/O-ul, conform modelului von Neumann
+ diagramă cu modelul von Neumann
procesorul preia date din memorie sau din I/O, le prelucrează, le stochează înapoi în memorie sau I/O
procesorul execută instrucțiuni stocate tot în memorie

ciclul de execuţie al procesorului: instruction fetch, instruction decode, data fetch, execution, data writeback
+ actualizare diagramă von Neumann cu ciclul de execuţie al procesorului
instrucţiunile sau codul (ce e de făcut) şi datele (cu ce faci) sunt reţinute în memorie
memoria este legată de procesor/procesoare prin magistrală (bus, uzual FSB: Front Side Bus): magistrala de date şi magistrala de adrese
memoria poate fi privită ca un vector de octeţi adresat prin magistrala de adrese
citirea din memorie: TODO
scrierea în memorie TODO
mai multe procesoare pot concura la accesarea memorie şi genera probleme de concurenţă (race conditions): mai multe la cursul 9 şi la ASC

Internele memoriei

Ulrich Drepper - What every programmer should know about memory
tipuri de memorie
static vs dynamic
RAM
ROM (PROM, EPROM, EEPROM)
rânduri, coloane
detalii la PM
refresh
latenţă, bandwidth, frecvenţă
diferenţa de viteză memorie procesor
instruction prefetch
branch prediction
speculative execution
nevoia de memorie cache
+ demo cu informaţii despre memoria sistemului (meminfo, /proc/cpuinfo, getconf)

Memoria cache

reminder de la CN
viteză memorie cache
cache hit, cache miss
There are only two hard things in Computer Science: cache invalidation and naming things.
-- Phil Karlton
There are two hard things in computer science: cache invalidation, naming things, and off-by-one errors.
There are only two hard problems in computer science:
0) Cache invalidation
1) Naming things
5) Asynchronous callbacks
2) Off-by-one errors
3) Scope creep

6) Bounds checking

Memoria în sistemele multitasking

nevoia de izolare
suficient de multă memorie pentru un proces
fiecărui proces i se asociază un spaţiu virtual de adrese
în spate, sistemul de operare asociază adresele virtuale cu adrese efective
memoria virtuală e o convenţie, memoria fizică reţine datele efective
avantaje memorie virtuală: separaţie între procese, impresia că fiecare proces are tot spaţiul disponibil pentru sine, nu e nevoie să ştii ce adrese fizice vei folosi
dezavantaje memorie virtuală: overhead de calcul a adresei fizice din adresa virtuală la fiecare acces

Spaţiul virtual de adrese al unui proces

unic fiecărui proces
asigură separaţie între procese
sistemul de operare face asocierea între memoria virtuală (a fiecărui proces) şi memoria fizică: există o tabelă de asociere pentru fiecare procese
+ diagramă cu adresă virtuală -> tabelă de asociere -> adresă fizică
spaţiul virtual de adrese ocupa 2^32 octeţi (4GB) pe un sistem pe 32 de biţi
cuprinde zone: text (cod), rodata, data (date iniţializate), bss (date neiniţializate), heap, biblioteci (cu subzone: text, rodata, data, bss), stivă
în mod uzual partea superioadă din spaţiul de adrese al fiecărui proces este rezervată sistemului de operare
+ diagramă cu spaţiul virtual de adrese
memoria sistemului de operare este mapată în partea finală a spaţiului virtual de adrese al tuturor proceselor
pe Linux spaţiul (3GB, 4GB) este rezervat sistemului de operare (0xc0000000-0xffffffff): este inaccesibil din user mode (spunem că avem split 3/1); pe Windows 32bit avem split 2/2
+ demo cu urmărirea spaţiului de adrese al unui proces (pmap)
Ce se întâmplă dacă avem mai mult spaţiu virtual pentru un proces decât spaţiu fizic? Nu vom putea folosi spaţiul virtual complet, sau vom folosi spaţiul de swap, sau vom partaja memorie fizică.
Ce se întâmplă dacă avem mai mult spaţiu fizic decât spaţiu virtual (de exemplu rularea de procese legacy 32 de biţi pe un sistem pe 64 de biţi)? Mai multe procese pot ocupa întreg spaţiul fizic.
x86: 32 biţi pentru adrese virtuale, 32 de biţi pentru adrese fizice (maxim RAM accesibil 2^32 = 4GB)
x86_64 (curent, 2019): 48 de biţi folosiţi pentru adrese virtuale (plan pentru 57 de biţi), 46 sau 52 de biţi pentru adrese fizice (maxim RAM accesibil: 2^46 = 64 TB)
https://simonis.github.io/Memory/,
https://en.wikipedia.org/wiki/X86-64#Virtual_address_space_details

Compartimentarea spaţiului virtual de adrese. Segmentare

pentru asocierea spaţiului virtual cu spaţiul fizic, prima opţiune este să asociem tot spaţiul virtual (4GB) la o parte fizică, alt spaţiu virtual (4GB) la altă parte fizică
probleme: avem nevoie de 8GB RAM pentru doar două procese; multe zone din spaţiul virtual nu sunt folosite
alternativa: segmentare: compartimentarea spaţiului virtual de adrese în zone (precum text, heap, stivă) şi asocierea fiecărei zone la spaţiul fizic
+ diagramă cu segmentare
implementarea segmentării: tabela de segmentare, selector de segment
adresa virtuală este compusă din selectorul de segment şi offset-ul în cadrul segmentului (în notaţia CS:EIP, CS e selectorul de segment, code segment; EIP e adresa/offset-ul în cadrul segmentului); selectorul extrage adresa fizică asociată segmentului şi adună offsetul, rezultă adresa fizică efectivă
+ diagramă pentru calculul adresei fizice
segmentarea este un concept mai vechi care este acum depăşit de paginare; are sens sa vorbim de segmentare pentru a vedea evoluţia compartimentării memorie şi pentru cazurile în care, pe arhitectura x86, veţi întâlni precizări legate de segmentare
segmentation fault (SIGSEGV - Signal Segment Violation): acces dincolo de limita unui segment al procesului
avantaj: asociem doar ce este nevoie
dezavantaj: fragmentare externă: este dificil să găsim loc (potrivit) pentru un nou segment fizic
segmentarea e o soluţie legacy, mai are unele relicve pe sistemele x86
soluţie: folosirea paginării

Paginare

soluţia folosită curent în sisteme moderne ce folosesc memorie virtuală
compartimentarea spaţiului virtual al fiecărui proces şi al spaţiului fizic al sistemului în componente de dimensiune fixă, numite pagini: pagini virtuale (pages) şi pagini fizice (frames)
în mod uzual, paginile au 4KB
pe un sistem pe 32 de biţi, un proces are spaţiu virtual de 4GB şi are deci, 4GB/4KB = 2^20 pagini virtuale
o tabelă de asociere, tabela de pagini (page table), face asocierea între pagini virtuale (pages) şi pagini fizice (frames)
asocierea se mai numeşte mapare/mapping
+ diagramă de sus cu proces, spaţiu virtual de adrese, pagini virtuale, tabelă de pagini, pagini fizice
translatarea de adrese este folosirea tabelei de pagini pentru a calcula o adresă fizică dintr-o adresă virtuală

avantaj folosire paginare: nu mai avem fragmentare externă; găsim un loc pentru o pagină şi o alocăm acolo; paginile fizice corespunzătoare unui proces vor fi împrăştiate în memoria fizică
dezavantaj: poate apărea fragmentare internă: alocăm o pagină (4KB) şi folosim doar câţiva octeţi; spaţiu ocupat de tabela de pagini
+ demo cu granularitatea alocării (multiplu de pagină prin apel de sistem)

Tabela de pagini

conţine 2^20 intrări pe un sistem pe 32 de biţi
este indexată cu adresa paginii virtuale şi conţine adresa paginii fizice
o adresă virtuală este cuprinsă din: adresă de pagină virtuală (20 de biţi) + offset în pagină (12 biţi): 2^12 = 4KB, cât cuprinde o pagină
o adresă virtuală este cuprinsă din: adresă de pagină fizică (20 de biţi) + offset în pagină (12 biţi)
+ diagramă cu translatarea de adrese din adresa virtuală în adresa fizică folosind tabela de pagini
responsabilitatea translatării aparţine unei componente hardware de pe procesor: MMU (Memory Management Unit)
fiecare intrare în tabela de pagini se numeşte page table entry (PTE)
bitul valid/invalid din PTE precizează dacă o pagină este validă; dacă nu, rezultă în excepţie de acces la memorie (page fault)
în PTE apar precizări despre tipul de operaţii permise (read-only, read-write)
există o tabelă de pagini per proces, reţinută în memoria fizică
un registru dedicat reţine adresa tabelei de pagini în memorie pentru procesul curent: PTBR (Page Table Base Register), cr3 pe x86, TTBR pe ARM
neajunsuri ale tabelei de pagini:
* este stocată în memorie, ocupă spaţiu: soluţie: tabelă de pagini ierarhică
* este nevoie de accesarea tabelei de pagini pentru fiecare operaţie cu memoria: soluţie: TLB

Tabelă de pagini ierarhică

sau paginare multi-nivel
în forma implicită a adresei virtuale: 20 de biţi pentru adresa paginii (PTE index) şi 12 biţi pentru offset: 2^20 intrări
tabelă de pagini ierarhică: 10 biţi (page directory index), 10 biţi (PTE index), 12 biţi offset
o intrare în page directory referă o structură page table (cu 2^10 intrări), o intrare în tpage table are 2^10 intrări şi conţine adrese de pagini fizice
mai multe niveluri (4-5) pentru sistemele pe 64 de biţi
+ diagramă cu tabela de pagini ierarhică
dacă o zonă lipseşte, intrarea în page directory este nevalidă şi nu referă page table
avantaj: spaţiu redus
dezavantaj: mai mult overhead de translatare

TLB

Translation Lookaside Buffers
o operaţie cu memoria înseamnă un acces la tabela de pagini (în memoria fizică) pentru extragerea mapării şi apoi un acces la memoria efectivă pentru extragerea informaţiei (2 accese)
pentru a reduce overhead-ul, TLB reţine cele mai recent accesate intrări în tabela de pagini; este un cache
are 128-256 intrări cu cele mai recente mapări
cu un "hit rate" bun se reduce timpul de translatare
+ diagramă cu folosirea TLB pentru translatarea de adrese
+ demo cu informaţii despre TLB
când are loc TLB flush? la address space switch (context switch)
după address space switch TLB este repopulat pe măsură ce sunt accesate noi pagini
de ce se mapează sistemul de operare în spaţiul virtual de adrese al fiecărui proces şi nu i se asociază un spaţiu virtual de adrese separat? la apel de sistem şi întoarcerea din apel de sistem nu este nevoie de schimbarea spaţiului de adrese şi deci, de TLB flush
TLB flush e soluţia cea mai simplă
procesoarele moderne oferă precizarea unui identificator de spaţiul de adresă (Address Space ID, ASID, pe ARM sau Process-Context ID, PCID, pe x86) pentru flush selectiv la nivelul TLB-ului când are loc un address space switch

continuăm la cursul următor cu detalii despre memoria virtuală: demand paging, swapping, shared memory, file mapping, page replacement, thrashing

Sumar

memoria este folosită pentru a reţine cod/instrucţiuni şi date folosite de procesor
memoria fizică (RAM) este mai înceată decât procesorului: folosim memorie cache
pentru izolare şi utilizare facilă folosim memorie virtuală
fiecare proces are asociat un spaţiu virtual de adrese compus din zone
sistemul de operare asociază/mapează spaţiul virtual de adrese al fiecărui proces la spaţiul fizic (memorie RAM)
o formă veche de translatare era segmentarea
forma curentă este paginarea: împărţirea spaţiului virtual şi a spaţiului fizic în pagini
tabela de pagini face asocierea între pagini virtuale (pages) şi pagini fizice (frames)
translatarea este realizată de o componentă hardware (de pe procesor) numită MMU (Memory Management Unit)
tabela de pagini este reţinută în memorie şi este: mare (ocupă spaţiu) şi înceată (accesarea tabelei de pagini înseamnă acces la memorie)
PTBR (Page Table Base Register) reţine adresa în memorie a tabelei de pagini pentru procesul curent
tabela de pagini ierarhică combate dezavantajul spaţiului ocupat în memorie de tabela de pagini clasică (neierarhică)

TLB (Translation Lookaside Buffers) combate dezavantajul overhead-ului de translatare (nevoie de acces la memorie)
este nevoie de TLB flush la address space switch, când se schimbă tabelele de pagini