Operativsystem

Minneshantering II
(kap 4 i boken)

Sidutbytesalgoritmer

- Då en sidfel (page fault) uppstår, måste operativsystemet byta ut innehåller i en sidram mot det innehåll som den som förorsakade sidfelet behöver
- Då innehållet i en sidram byts ut
  - om förändrat (skrivet i), måste förändringarna sparas någonstans (= på hårdskiva)
  - om ej förändrat, kan man helt enkelt ”radera” det gamla innehållet (eftersom innehållet redan finns sparat någon annanstans)

Sidutbytesalgoritmer II

- Det nya innehållet kan tas från flere ställen
  - om programkod: direkt från filen som innehåller det körbara programmet
  - om data: från swap-området på hårdskivan
  - om ej tidigare använt, en ny ”tom” sida
- Problemställning !
  - vilken sidram skall tas i användning då ett sidfel uppstår ???
    - Val av sidutbytesalgoritm

Utbytesalgoritmer generellt

- Problemet med vilken sidram innehållet skall bytas i är generellt
  - Cache-minnen: Vilken cache-rad skall bytas
  - Disk cache: Vilket / vilka block skall bytas?
  - Web server och minnes-cache: Vilken fil skall bytas?

Sidutbytesalgoritmer

- Den optimala
- Icke nyligen använd sida (not recently used, NRU)
- Först in, först ut (First In, First out, FIFO)
- Andra möjligheten (Second Chance)
- ”Klock”-algoritmen
- Minst nyligen använda sidan (Least Recently Used, LRU)
- Inte ofta använd (Not Frequently Used)
- Working set

Den optimala algoritmen

- Då innehållet i en sidram skall bytas, numrera sidramarna enligt hur länge det tar innan sidramen nästa gång accesseras
- Byt innehållet i den ram som längst i framtiden kommer att accesseras!!
- Problemet: Denna algoritm är omöjlig att implementera

Not recently used

- Varje gång man läser från en sida, sätts en bit R (read), varje gång man skriver sätts en bit M (modified)
- Med jämna mellanrum nollställer operativsystemet R-biten (t.ex. vid klock-avbrott)
- Vid ett sidfel, kan klassificera sidramarna enligt
  - Klass 0: ej refererad, ej modifierad
  - Klass 1: ej refererad, modifierad
  - Klass 2: refererad, ej modifierad
  - Klass 3: refererad, modifierad
- Byt ut innehållet i en slumpmässigt vald sidram av lägsta möjliga klass

First-In, First-Out

- Algoritm: Sidram väljs i löpande ordning
- Används sällan, ger dåliga resultat, eftersom sidor som används mycket byts ut lika ofta som sådana som använts t.ex. bara en gång

Second chance

- Som FIFO, men använder sig också av R-biten, dvs. sida som refererats nyligen byts ej ut, utan man ser på nästa i listan
  - dvs. sidorna får en ”andra möjlighet”
- Klock-algoritmen är samma som FIFO/Second chance, det är bara fråga om hur algoritmen är implementerad

Least recently used (LRU)

- Introducera en räknare C som inkrementeras för varje instruktion som CPU:n utför
- Då en sida refereras, skriv värdet på räknaren C till ett fält i sidtabellen
- Vid sidfel, välj den sidram som har det lägsta räknarvärdet sparat för att byta ut
- Problem: Hårdvara implementerar sällan denna funktionalitet

Mjukvaru-LRU

- Vid varje klock-avbrott, inkrementera en räknare C
- Vid samma klock-avbrott, kopiera räknarens värde till sidtabellen för sådana sidor som har R-biten satt
- Vid sidfel, byt ut innehållet i den sidram med lägsta motsvarande räknare

Not Frequently Used (NFU)

- Vid klockavbrott, om R-biten satt, inkrementera ett räknare-fält i sidtabellen
- Vid sidfel, ersätt sida med lägsta värdet på räknaren
- Problem: Detta system ”glömmer” aldrig, en sida med högt räknarvärde förblir i minne, även om den inte använts på länge

NFU med åldrande

- Innan R-biten läggs till räknaren, högerskiftas räknaren
- Dessutom, R-biten adderas till den mest signifikanta biten i räknaren
- Motsvarar nu ganska långt LRU

Working set algoritmer

- Tidigare presenterades ”Demand paging” sidor laddas efter behov
  - Problem: Då en sida laddas, tar det tid p.g.a. I/O -> proc. blockeras. Då sidan laddats, fortsätter processen och behöver med stor sannolikhet en ny sida i närheten av den förra -> ett nytt sidfel -> ”trashing” (kontinuerliga sidfel)
  - Borde ha laddat flere sida i samband med samma I/O!
- Working set = den mängd av sidor i minnet som processen för tillfället använder sig av
- Man strävar efter att processens ”working set” skall finnas i sidramar då processen skeduleras (”prepaging”)

Working set

- Working set kan definieras som
  w(k,t)
  vid en tidpunkt *t* den mängd av sidor som använts vid de senaste *k* minnesreferenserna
- En sidbytesalgoritm är härmed: Använd en sidram vars sida inte finns i working set
- Det är dock svårt att använda sig av *k* (de senaste referenserna), istället använder man sig ofta av approximationen de sidor som refererats under den senaste tiden *T*

Working set algoritm

- Upprätthåll en räknare *C* för nuvarande tid
- Vid klockavbrott, kopiera värdet på *C* till ett fält (*current_time*) i sidtabellen
- Vid sidfel, beräkna ”åldern” på sidorna (*C*-*current_time*), byt ut en sida med ålder större än *T*

Sidstorlek ?

- Små sidor -> stora sidtabeller (ryms t.ex. ej i TLB)
- Stora sidor -> intern fragmentering (eftersom i medeltal 50 % av den sista sidan är oanvänd)
- Overhead kan beräknas
  overhead = *se/p* + *p/2*
  där s är medelprocesstorleken, *e* är radstorleken i sidtabellen, *p* är sidstorleken

Design av paging systems

- Globala eller lokala sidbytesalgoritmer
  - Globala: Sidramarna gemensamt för alla processer
    - Dynamiskt avgöra hur många sidramar till en process
  - Lokala: Varje process blir tilldelad en fixerad mängd sidramar
    - Minnesbehovet ökar: trashing
    - Minnesbehovet minskar: slöseri med minne

Cleaning policy

- Vi vill ha sidor lediga
- Paging daemon: undersöker periodiskt vilka sidor som kan skrivas till minnet

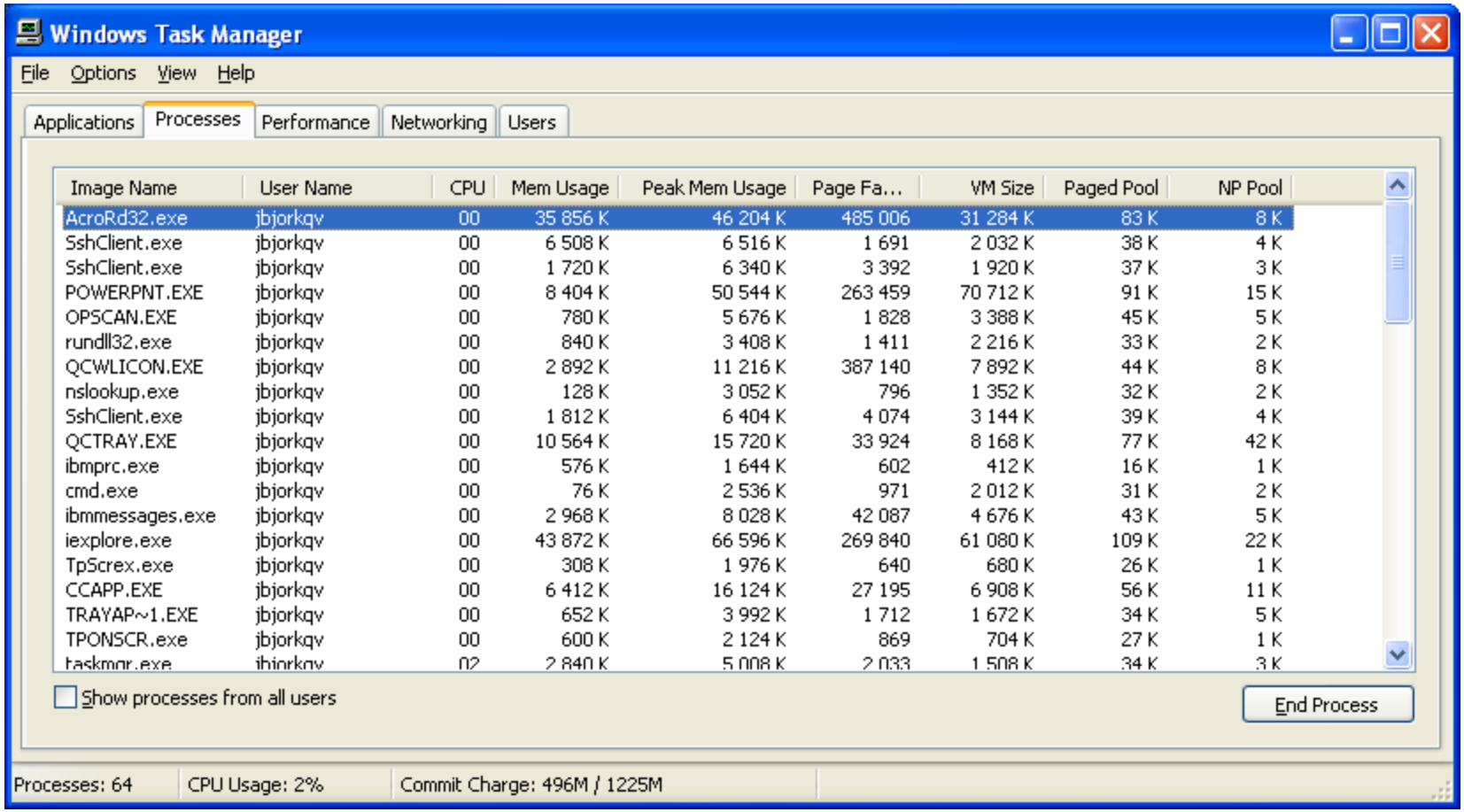
Committed Memory

- Committed memory = totalt antal sidor som måste ha en representation i antingen RAM eller swap-utrymme på skiva
  - Vad händer om vi skriver till för mycket minne?

NT (4.0) Minneshantering

- i386: 2 GB user space / 2 GB kernel
  - genom boot switch /3GB även 3GB/1GB
- Varje process har en minimum samt maximim ”working set”, samt en nuvarande storlek på working set
- Sidhämtning: ”Clustered demand paging”, hämtar ett kluster av sidor kring den sida som begärs (0-7 sidor)
- Sidbyte: Klock-algoritmen
  - NT 5.0 – om R-bit satt, nollställ räknare, annars inkrementera räknare, byt ut sidram med största värde på räknare
  - Multiprocessor-maskiner: FIFO

Processlista NT



NT Sidramar; status

Valid (or Active)

The page is part of at least one process' working set.

Transition

The page is not on any page list, and the Memory Manager is in the process of reading the page in from or writing the page out to a file (e.g., a paging file).

Standby

The Memory Manager has recently removed the page from a working set, and any copy of the page on disk is consistent with the in-memory copy.

Modified

The Memory Manager has recently removed the page from a working set, but the process modified the page, and a copy of the file on disk (e.g., in a paging file) does not have the new content.

Modified No-write

The Memory Manager recently removed the page from a working set, but something, typically a file system driver, marked the page so that it didn't automatically write back to a file, even if a process modified the page. A file system indicated that the Memory Manager need not write the data to disk yet.

Free

The page is not part of any working set, and the process that referenced it removed any other references to it (e.g., it was deallocated). The zero-page thread has not cleared the page (i.e., filled it with zeroes).

Zeroed

The Memory Manager freed the page, and the zero-page thread cleared it.

Bad

The MMU detected the page as faulty, and the page is now off-limits.

NT Working set

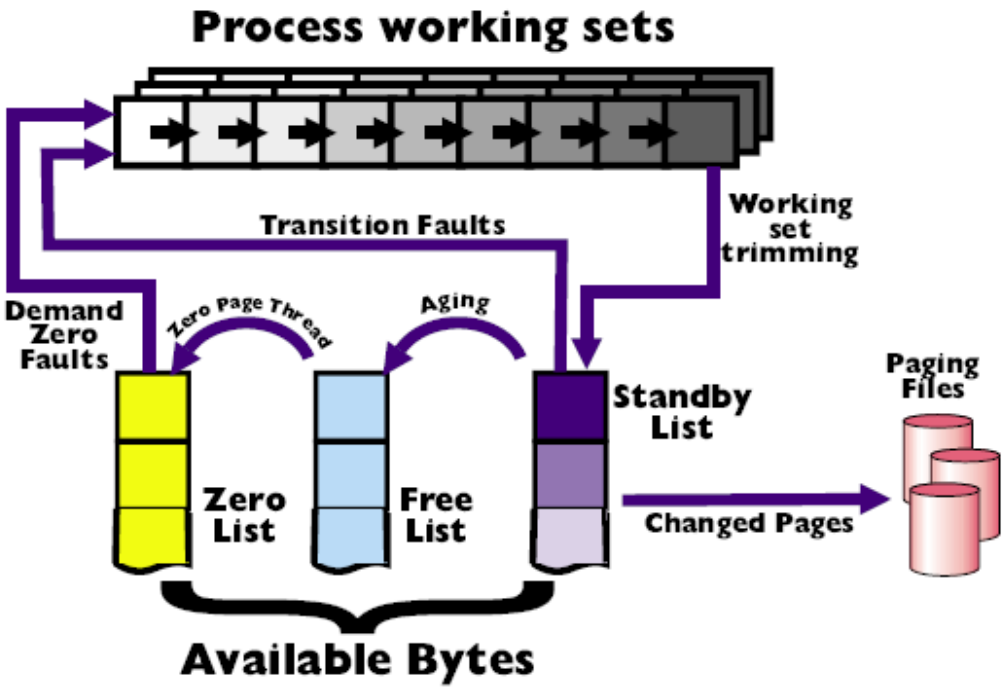
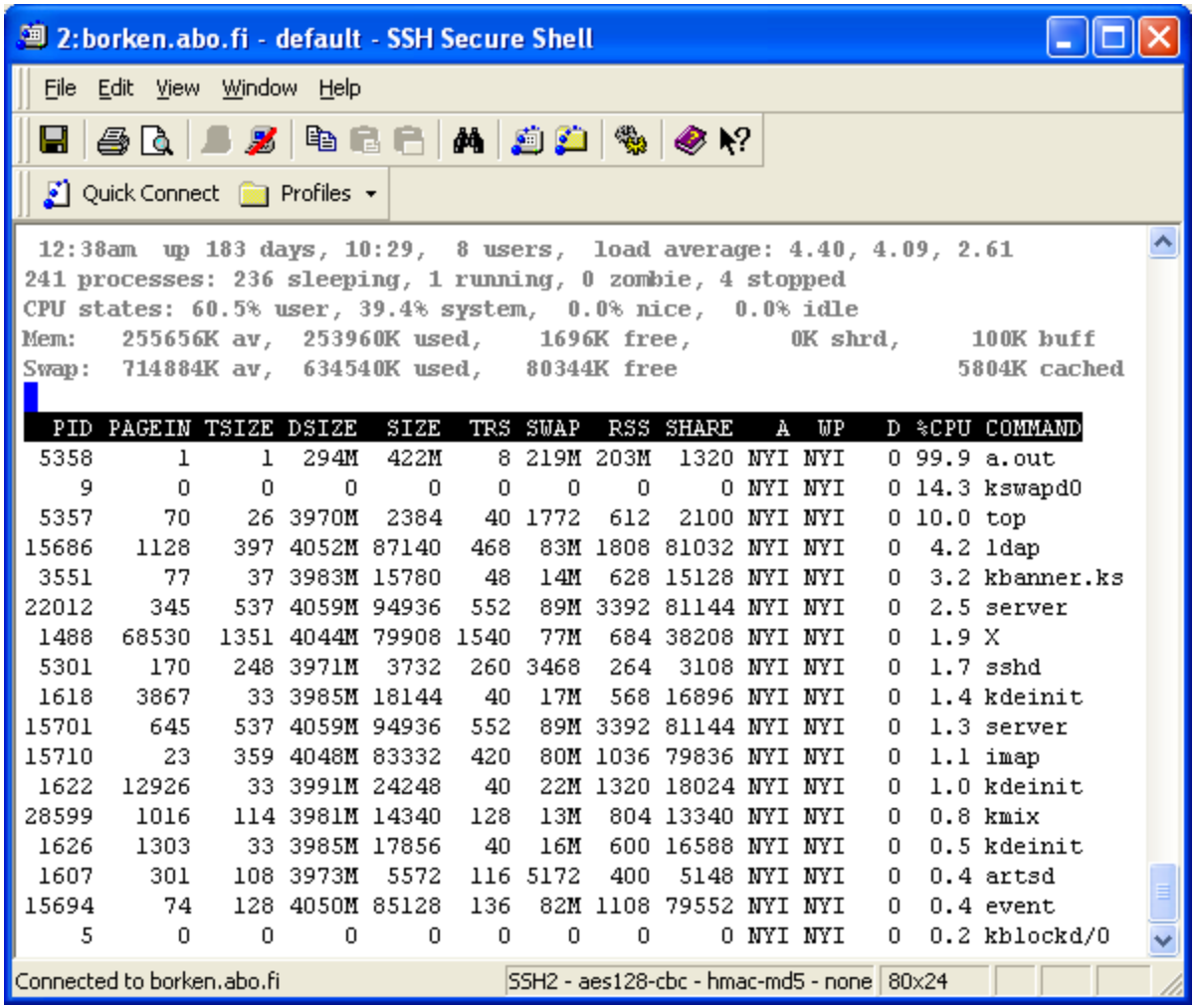


FIGURE 8. PROCESS WORKING SET TRIMMING IN WINDOWS NT. PERIODICALLY, ALL PAGES ABOVE THE PROCESS MINIMUM WORKING SET ARE STOLEN AND PLACED ON THE STANDBY LIST. TRIMMED PAGES AGE AND MOVE TO THE FREE LIST AND, ULTIMATELY, TO THE ZERO LIST. MEANWHILE, TRIMMED PAGES WHICH ARE STILL ACTIVE ARE "SOFT-FAULTED" BACK INTO THE PROCESS WORKING SET WHEN REFERENCED. MODIFIED PAGES MUST BE COPIED TO A PAGING FILE BEFORE MOVING TO THE FREE LIST. A MODIFIED PAGE WRITER THREAD OF THE KERNEL WRITES PAGES TO DISK IN BULK WHENEVER A DESIGNATED THRESHOLD OF CHANGED PAGES ACCUMULATES.

Linux minneshantering

- 3 GB User space / 1 GB kernel space
- Upprätthåller resident set size (RSS), mängden av sidor en process har allokerade i sidramar
- Sidhämtning: demand paging (ingen prepaging)
- Sidbyte: Soft-LRU, räknare som uppdateras vid accesser, högerskiftas då ej accesserad

Linux minneshantering



Linux sidramar

- Kärnan håller reda på tillståndet för varje sidram genom en räcka av deskriptorer av typen *struct page*

– Räckan med sidramsdeskriptorer kallas `mem_map`

– Håller reda på statusen för varje sidram *(== 0 is free, > 0 is used)*

– Flaggor för *dirty*, *locked*, *referenced*, *etc.*

- Kärnan allokerar och frigör sidramar genom `__get_free_pages(gfp_mask, order)` och `free_pages(addr, order)`

Linux sidramar

```
struct page {
        unsigned long flags;            /* atomic flags, some possibly
                                           updated asynchronously */
        atomic_t count;                 /* Usage count, see below. */
        struct list_head list;          /* ->mapping has some page lists. */
        struct address_space *mapping;  /* The inode (or ...) we belong to. */
        unsigned long index;            /* Our offset within mapping. */
        struct list_head lru;           /* Pageout list, eg. active_list;
                                           protected by zone->lru_lock !! */
        union {
                struct pte_chain *chain;/* Reverse pte mapping pointer.
                                         * protected by PG_chainlock */
                pte_addr_t direct;
        } pte;
        unsigned long private;          /* mapping-private opaque data */
#if defined(WANT_PAGE_VIRTUAL)
        void *virtual;                  /* Kernel virtual address (NULL if
                                           not kmapped, ie. highmem) */
#endif /* WANT_PAGE_VIRTUAL */
};
```

Linux sidramar, statusflaggor

PG_locked – I/O på gång
PG_error – I/O error när sidan flyttades
PG_referenced – Nyligen accesserad för I/O
PG_uptodate – Satt efter I/O
PG_dirty – Sidan har modifierats
PG_lru – Finns i aktiv / inaktiv lista
PG_active – i aktiva listan
PG_slab – intern minnesallokering
PG_reserved – kernelen använder sidan

Linux minneshantering

```
#include <malloc.h>
main() {
  long i,j;
  void *p;
 for (i=0; i<4000; i++) {
    p = malloc(1<<20);
    if (p) ((int *)p)[0] = 12;
    printf("allokerat %lu bytes, p: %p\n", i*(1<<20), p);
    if (p) {
      for (j=0; j<1<<20; j++) {
        ((char *)(p))[j] = 3;
      }
    }
  }
 while (1);
}
```

Linux sidramar

• In theory, paging eliminates the need for contiguous memory allocation, but…

– Some operations like DMA ignores paging circuitry and accesses the address bus directly while transferring data

• As an aside, some DMA can only write into certain addresses

– Contiguous page frame allocation leaves kernel paging tables unchanged, preserving TLB and reducing effective access time

• As a result, Linux implements a mechanism for allocating contiguous page frames

Allokering av löpande sidramar

Buddy system algorithm

– All page frames are grouped into 10 lists of blocks that contain groups of *1*, *2*, *4*, *8*, *16*, *32*, *64*, *128*, *256*, and *512* contiguous page frames, respectively

• The address of the first page frame of a block is a multiple of the group size, for example, a 16 frame block is a multiple of 16 × 212

– The algorithm for allocating, for example, a block of 128 contiguous page frames

• First checks for a free block in the *128* list

• If no free block, it then looks in the *256* list for a free block

• If it finds a block, the kernel allocates 128 of the 256 page frames and puts the remaining 128 into the *128* list

• If no block it looks at the next larger list, allocating it and dividing the block similarly

• If no block can be allocated an error is reported

Repetition

- Minneshantering
  - Se till att processer har tillgång till minne; Virtuellt minne:
    - Varje process får en virtuell adressrymd (=möjlighet att läsa / spara data)
    - OS ser till att det tillgängliga fysiska minnet används ändamålsenligt (bl.a. delas mellan processer)
      - Upprätthåller per process en minneskarta över tillgängligt minne
      - Upprätthåller datastrukturer över hur det fysiska minnet används
      - Ser till att data vid behov laddas / flyttas till och från den fysiska minnet