Operating systems

ELTE IK.

Dr. Illés Zoltán

zoltan.illes@elte.hu

Financed from the financial support ELTE won from the Higher Education Restructuring Fund of the Hungarian Government

We have talked earlier…

- Evolution of operating systems
- Notions of Operating system, structures
- Files,directories, filesystems
- Processes
  - process communications
  - critical sections, , semaphores
- Classical IPC problems
- Scheduling
- I/O, deadlock problem

What comes today…

- Memory management
  - Base memory handling
  - Swapping
  - Virtual memory
  - Page swapping algorithms
  - Designing paging systems
  - Segmentation

Memory handler

- Memory types
- Part of operating systems
  - Often in kernel
- Tasks:
  - To register memory which are free, occupied
  - To allocate memory for processes.
  - To free memory.
  - To control the swap between the RAM and the hard disk

Basic memory handling

- Two different group of algorithms:
  - There is a need to translate, change processes between the memory and the disk. (swap)
  - There is no need of it.
- If there is enough memory there is no need of swapping!
- Is there enough memory?
  - HT1080Z – 16 KB, C64 – 64 KB, IBM PC-640 KB
  - Nowadays: PC 2-4-8 GB, a small server 4-16 GB

Monoprogramming

- One program in the same time.
  - There is no need of an „algorithm”. Type the command, execute it and wait for the next one.
  - Typical situations
    - Op.system is on the lower addresses, program is above it (it is rarely used today, formerly it was used at mainframes or at minicomputers.
    - On the lower addresses the program, the higher area the operating system in ROM (hand-held computer, embedded system)
    - Op.system is on the lower addresses, above the user programs and above the device drivers in ROM. (MS-DOS, ROM-BIOS)

Multiprogramming (Multitask)

- „Parallel” several programs are running. Memory must be divided among the processes.
  1. To implement multiprogramming with fixed memory slices.
  2. Multiprogramming with memory swapping.
  3. Multiprogramming with the usage of virtual memory.
  4. Multiprogramming with segmentation.

1. Multiprogramming with fixed memory slices

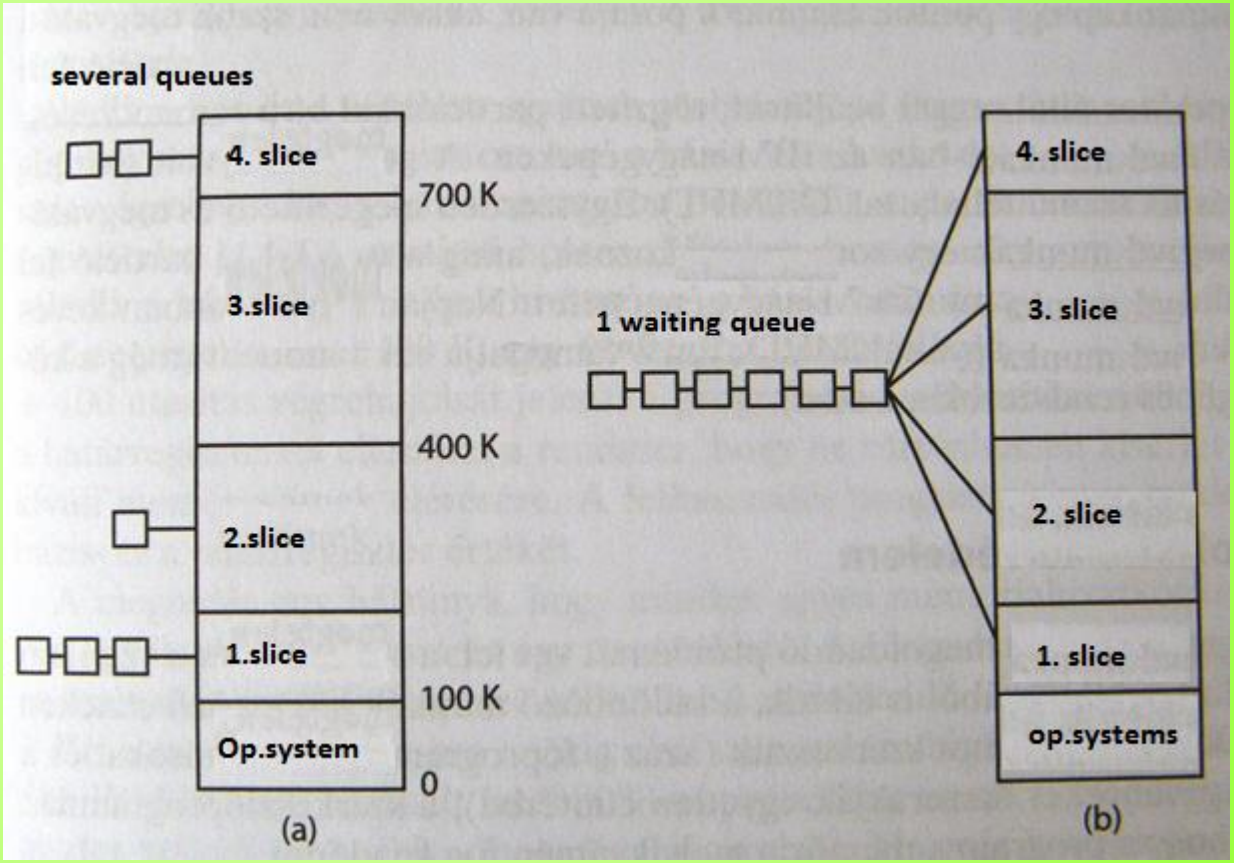
- Monoprogramming is present today mainly in embedded systems (PIC)
- Nowadays typically there are several „running” processes in the memory in preemptive time sharing systems.
- Divide the memory into n slices (not equal) (Fix slices)
  - E.g. at system start it may be done
  - One common waiting queue.
  - Each slice has an own waiting queue.
  - Typical solution for batch systems.

Memory division

Several waiting queues:
• Not used partitions

One queue:
•If a partition becomes empty, the first from the queue get into it.

IBM used in OS/360 system: OS/MFT (Multiprogram Fix Task)



Relocation – protection

- We do not know where the process will be, so the memory addresses can not be compiled to fix ones.
  - OS/MFT program at loading fresh the relocated addresses.
- It is not desirable if a program can access the memory of another program! (Protect)
  - OS/MFT PSW (program status word, 4 bits long safety key)
- Other solution is: Base+limit register usage
  - Programs are not able to modify these.
  - Must be checked at each address reference: it is slow.

2. Multiprogramming with memory swapping

- Fixed memory slices are a typical solution for formerly used batch systems. (IBM OS/MFT)
- It is not proper for time sharing, graphical systems.
- To swap the whole process between the memory and the disk.
- There is no fixed memory partition, each of them changes dynamically due to how the operating system pack them to-and-fro.
- It will be dynamical, better memory usage, but the great number of swapping result holes!
  - Memory compression must do! (In several cases (during watching a football match) this time loss is not permitted.

Dynamic memory allocation

- Generally it is not known forward how many dynamic data, stack area is needed for a program.
- The program code gets a fix slice, while the data and the stack gets changable ones. They can grow and ease down.
  - If the memory runs out, the process stops and waits for carrying on or it is swapped to the disk to give the other running processes more memory.
    - If there is any waiting process in the memory it may be swapped too.
- How can we register the dynamic memory?

Registering the dynamic memory

- To define an allocation unit.
  - The size of it is questionable. If it is small there will not be so many holes but the resource (memory) demand for registering is big.
  - If it is big, then the loss we get from the remainder memory (not used) in a unit is too much.
- The implementation of registering:
  - Using a bitmap
  - Using chained list
    - If a process ends the memory holes being beside each other should be concatenated.
    - The holes and the processes have separate lists.

Bitmap, implementation of chained list

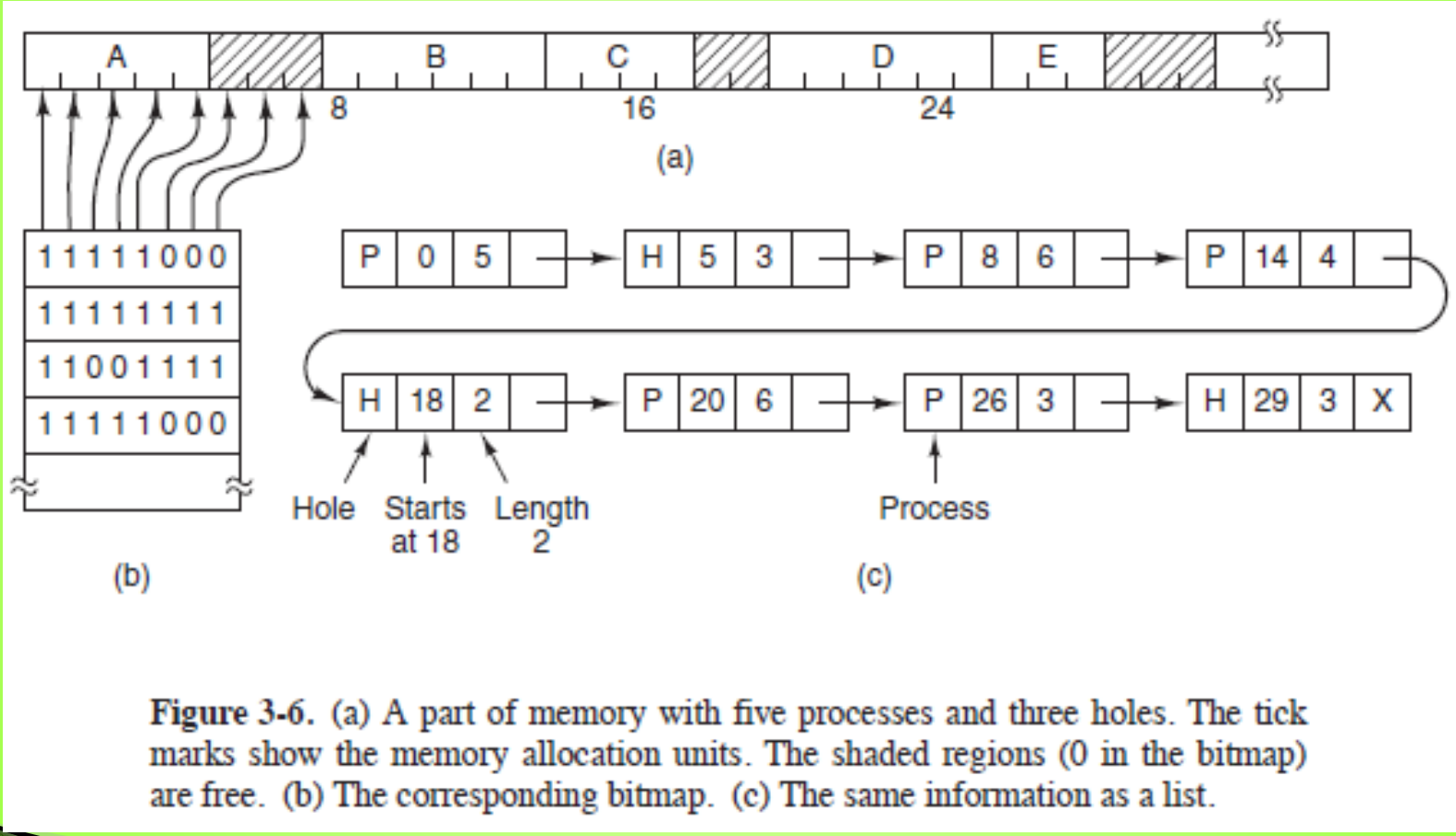


Figure 3-6. (a) A part of memory with five processes and three holes. The tick marks show the memory allocation units. The shaded regions (0 in the bitmap) are free. (b) The corresponding bitmap. (c) The same information as a list.

Memory allocation strategies

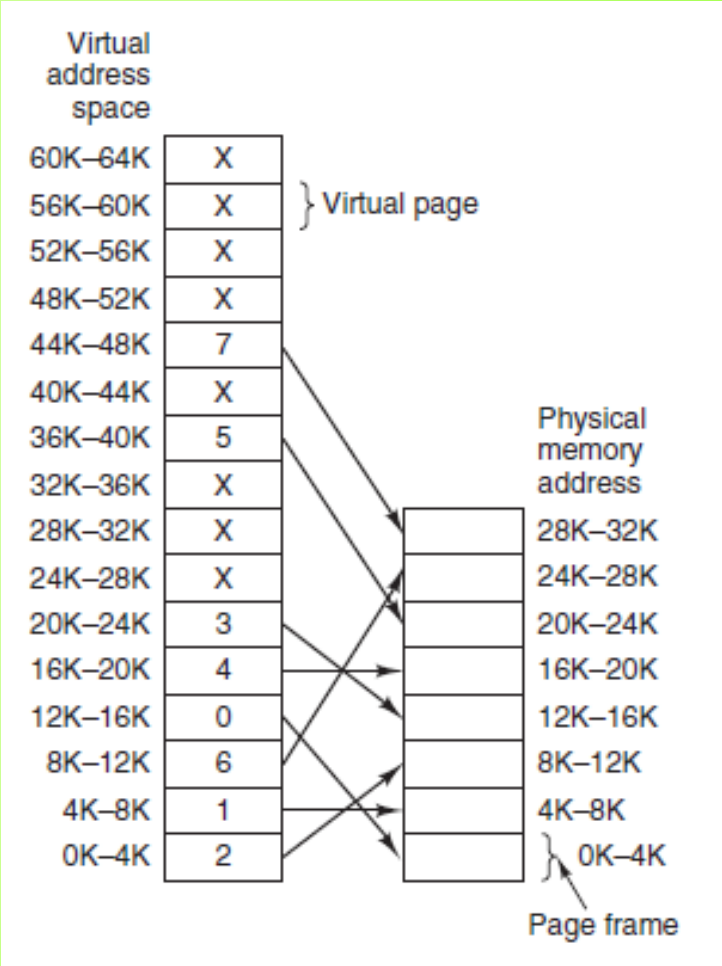
- For deciding a memory location for a new (or a previously running one, which is now in the swap partition) there are several memory allocation algorithms. (similar to the ones at disks) :
  - First Fit (to the first place where it can fit, quickest, simplest method.
  - Next Fit (the searching starts not from the starting point but from the last search finishing point, not so efficient as first fit)
  - Best Fit (it is slow, it produce a lot of small holes)
  - Worst Fit (will not be a lot of small holes, but it is not efficient)
  - Quick Fit (a list of holes ordered by their size, the concatenation of holes are expensive.)

3. Multiprogramming with virtual memory usage

- A program may use more memory than the available amount of physical memory.
  - The operating system keeps only the needed part in the physical memory.
  - A program is in the „virtual memory space”.
  - John Fotheringham (1961, ACM)
  - The ide may be used in the case of a monoprogramming environment too.
    - In 1961 and later there were „single task” system like (small) computers.

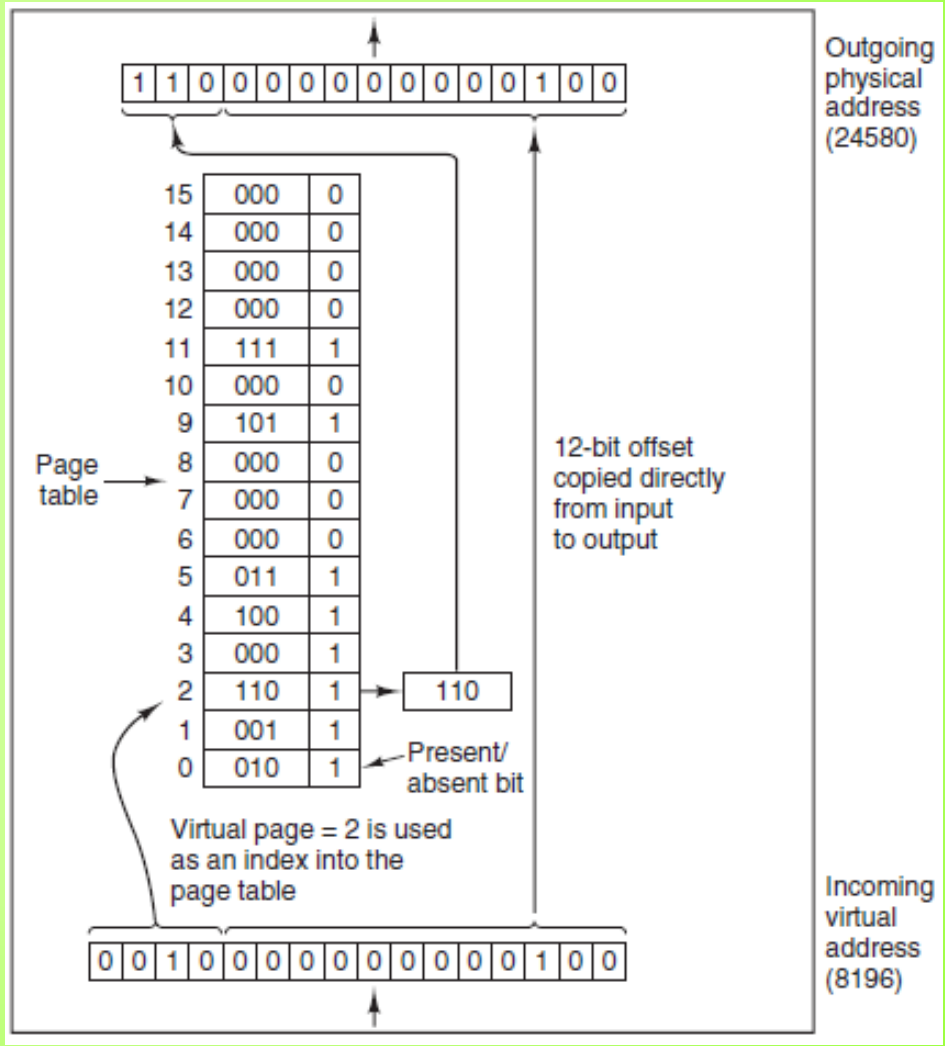
Memory Management Unit (MMU)

- The virtual address space is devided into pages. (it is called page-table)
- present, not present bit
- Binding of virtual-physical pages
- If MMU sees that a page is not in the memory, causes a page fault, creates a page frame and load the needed page.



MMU working
Example: 16 piece – 4kb page

- Size of pages are two on the power
- The first 4 bit from the 16 bits long virtual address is the page number, the remaining ones are the offset.
- 1 bit for signing the present or absence.
- 15 bites are going out to the physical address bus.



Page table problems

- Previos example (16 bit virtual address space, 4 bit page table, 12 bit,4 KB, offset) simple, quick.
- A modern processor is 32 or 64 bit.
  - 32 bit virtual address space: at 12 bit (4kb) page size 20 bit is the size of page table. (1MB, kb. 1million element)
    - Each process has an own virtual address space, and an own page table! Such size page table is imaginable!
  - 64 bit virtual address space such page table size is not realizable!

Multi-level page tables

- Headache how to use a simple page table in the case of 32 bit virtual address space.
- Use a multi-level one:
  - Page table1= 10 bit (1024 elements)- higher level page table
  - Page table2= 10 bit – second level page table
  - Within a page Offset= 12 bit
  - Advantage: while it is needed for a process to keep the page table in the memory too, here we have to get only 4 tables with 1024 elements (LT1 plus the program, data and the stack) and not 1 million!
- We choose classical values for bit numbers for table and offset – we can modify them but will not be essential changings.
- We can change the two level to more levels! (More complicated!)

The structure of a table registration

- Contains the page frame number (physical memory and the offset size decide how many bits)
- present/absence bit
- Protect bit, if the value is 0 it is readable, in the case of 1 it is only writeable.
  - 3 protection bit: read, write, execution permission for each page.
- Dirty bit (modifying). If it is 1 then the page frame was modified, so in the case of writing out to the disk you have to write it really!
- Reference bit, if the value is 1 if the page is referenced (it is in use). If a page is used it is not able to be written out to the disk!
- Cache disabled bit. It is important in the case when a given are of the physical memory is at the same time the data area of any I/O device.

Instead of (besides) paging TLB (Translation Lookaside Buffer)

- The classical MMU possibility is slow, we need at least one table reference for each memory reference, so the memory operation time may decrease to the half of it minimum
- Insert a small HW unit (TLB- associative memory) to the MMU with a small amount of registry (not more than 64 at the beginning)
  - Today at Nehalem there is a two-level TLB, the TLB has 512 elements.
- Software handling of TLB
  - 64 elements are rather big to get only few TLB faults, so HW solution may be saved.
  - Such systems are e.g.. Sparc Mips, Alpha, HP PA, PowerPC

Inverted page tables

- Start from the size of the physical memory.
- The page table contains so many elements which is given by the physical memory.
  - It spares a lot of space but it is more complicated to get the physical memory address from the virtual one. (No possibility of using an automatic index!)
  - E.g.: 4 kb page size, 1GB RAM, 2^{18} number of reference.
- Out of the woods: Use the TLB, if it faults search the inverted page table and write the result into TLB.

Inverted table example

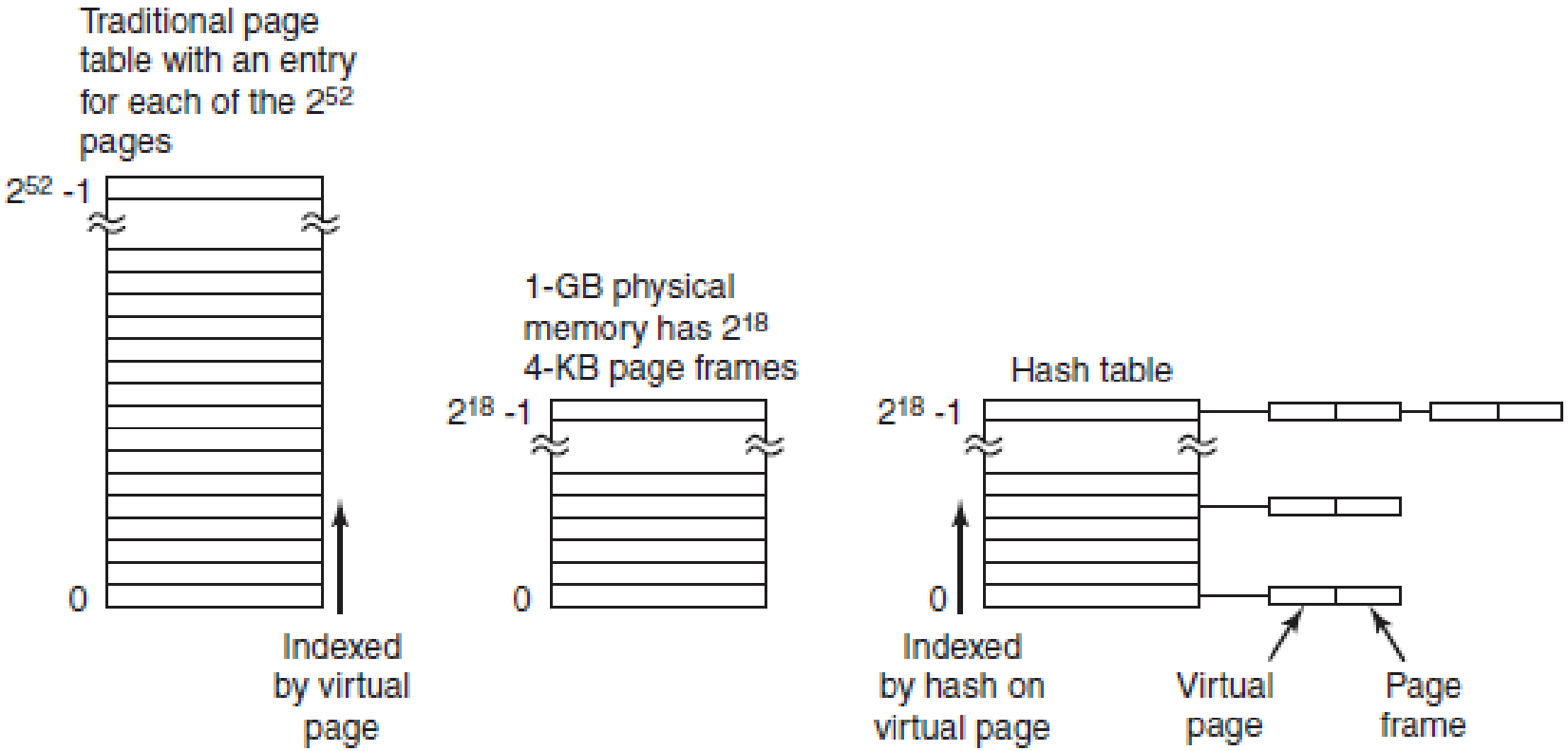


Figure 3-14. Comparison of a traditional page table with an inverted page table.

Page swapping algorithms

- If the virtual addressed page is not in the memory then a page must be throw away and translate this new page.
  - The question is how?
  - Randomly? It is better to move out thet page which was not used „recently”.
- The case is similar at the usage of the processor cache memory or at the local cache of a browser.

Optimal page swapping

- Sign each of the pages with a number which indicates how many CPU instructions was executed before this reference!
- Throw away that page in which this number is the least!
- There is a problem, it is impossible to implement!
- This method can be used at testing running twice the program!

NRU (Not Recently Used) algorithm

- Use the modify and the reference bit of the page table.
  - Set the reference bit to 0 time to time (at each clock interrupt at about 0.02 sec) with this signing whether it was used recently or not.
  - 0.class: not referenced, not modified
  - 1.class: not referenced, modified
    - Page can get here if the clock interrupt sets the reference bit to 0.
  - 2.class: referenced, not modified
  - 3.class: referenced, modified
- Choose randomly a page from the least not empty class.
  - Simple, not really efficient its implementation, it gives a proper result.

FIFO page swapping second chance algorithm

- Simple FIFO, it is known from other areas too, if we need a new page we throw away the oldest one!
  - The pages are in a list ordered by arriving time, a page arrives to the beginning of the list and leaves from the end.
- The correction of this is the second chance page swapping algorithm.
  - Almost the same as FIFO. The difference is if the page at the end of the list has got a reference bit with value 1 then it gets a second chance. It is moved to the beginning of the list and the reference bit is set to 0.

Clock page swapping

- Clock algorithm: like the second chance method but do not move the pages inside a list instead of it use a circled list and go around with a pointer.
- The pointer shows the oldest page.
- At page fault if the pointed page reference bit is 1 change it to 0 and move to the next page!
- If the examined page reference bit is 0 then we throw away!

LRU (Least Recently Used) algorithm

- Throwing away the least recently used page.
- How can we implement it?
  - HW or SW
  - HW1: Take a counter which increases with 1 at each reference. We can store this counter at each page table. At each page reference we write this counter into the page. At page fault we search for the page with the smallest counter!
  - HW2: LRU bitmatrix usage, n page, n x n bitmatrix. At a k-th page frame reference set the k-th row of the matrix to 1, whilst the k-th column to 0.
  - At page fault the smallest row value indicates the oldest page!

NFU (Not Frequently Used) algorithm

- Give a counter to each page. At each clock interrupt give this the value of the page reference R bit.
- At page fault throw away the page with the smallest counter. (It is the least used)
- It is a problem, the NFU does not forget, the pages used frequently at the beginning of the program save their high values.
- Modify it: At each clock interrupt shift to the right the counter with one bit and put from the left the refernce bit (shr). (aging algorithm)
  - It approximates well the LRU algorithm.
  - This page counter is implemented with n bits so it is not able to diferentiate events before n time unit.

Viewpoints of designing page swapping I.

- Working set model
  - Load the demanded pages. At starting paging before. To keep in the physical memory only those pages of a process which are used. This may alter with time.
  - We have to register the pages. If a page was not referenced in the last N time unit we throw it at a page fault.
  - Correction of clock algorithm: Examine if a page is a member of the working set or not? (WSClock algorithm)
- Local, global place allocation
  - If we examine all of the processes in the case of a page fault then it is global. If we examine only the pages of the same process then it is called local.
  - In the case of a global algorithm we can give a proper page due to its size which we can change dynamically.
  - Page Fault Frequency (PFF) algorithm, rate of page fault /sec, if there is a lot of page fault increase the page numbers of the process in the memory. If there is a lot of process we may write out the whole process to the disk. (load balance)

Viewpoints of designing page swapping II.

- To decide a proper page size.
  - Small page size
    - The page loss is small, but we need a big page table for registration.
  - Big page size
    - Inversely, „page loss” is big, small page table.
  - Typically: n x 512 byte the page size, XP, Linux-s 4KB page size. 8KB is used too (servers)
- Common memory
  - We can allocate a memory area which can be used by several processes.
  - Distributed common memory
    - To share memory between processes in the network.

Segmentation

- Virtual memory: one dimesional address space, from 0 till maximum address (4,8,16 GB, …)
- Several programs have dynamical area which may increase with uncertain size.
- Create address spaces independent from each other these are called segments.
  - In this world an address consists of 2 parts: segment number and an address within it. (shift)
  - Segmentation makes possible to implement „simply” distributed directories.
  - A program can be split logically to program, data etc. segment…
  - To give a protection level to each segment.
  - The page sizes are fix sized, the segments are not.
    - Fragmentation of segments appears. This fragmentation can correct with concatenation.

Pentium processor – virtual address handling I.

- There can be a lot of segments, the address space of a segment: 2^{32} byte (4GB)
  - XP, Linux-s use simple page swapping model. One process is one segment – one address space.
  - The number of segments: Pentium 16000, from TSS segment value, processor feature.
- LDT- Local Descriptor Table
  - Each process has got an own.
- GDT- Global Descriptor Table
  - There is only one from this, everybody uses it.
  - The system segments (operating system) is in it.
- Physical address: Selector + offset operation execution

Pentium processor – virtual address handling II.

- Achieving the segment: 16 bit segment selector
  - The low 0-1. bit of it gives the permission of the segment.
  - 2. bit=0=in GDT, 1=in LDT is the segment description.
  - High 13 bit is the index of the table. (Both of the two tables consist of 8192 elements)
- The LDT, GDT table elements are 32 bit long, from this we can get the base address to which we add the shift. (The result also 32 bit long!) This gives the linear address.
- If the paging is forbidden this is the real physical address. If it is not forbidden then two level page table usage: page size 4 KB, 1024 (10 bit) elements in page table.
- To get a page frame quicker TLB used too.

The protection levels of Pentium processor

- 0- Kernel level
- 1- System calls
- 2- Shared libraries
- 3- User applications
- To access data from lower levels is permitted.
- To access lower level (0-1 level) data from a higher level is forbidden.
- To call subroutines is permitted but only under controlling (call selector)
- User program can access data of shared libraries, but they do not modify them!

Thanks for your attention!

zoltan.illes@elte.hu