Free and Occupied Zones (1)

Problem

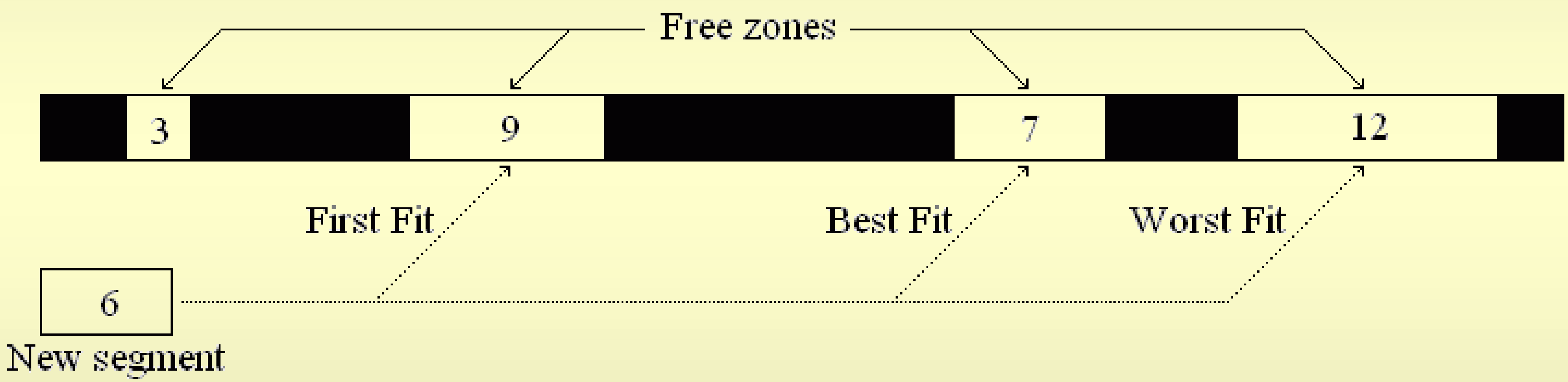
- a new segment is created → placed into memory
- a free, contiguous, large enough zone is required
- there may be many such zones - which one will be chosen?

Free and Occupied Zones (2)

Algorithms for placing segments into memory

- *First Fit* - first free, large enough zone that is found
- *Best Fit* - smallest free zone that is large enough
- *Worst Fit* - largest free zone (if it is large enough)

Free and Occupied Zones (3)



Memory Fragmentation (1)

External memory fragmentation

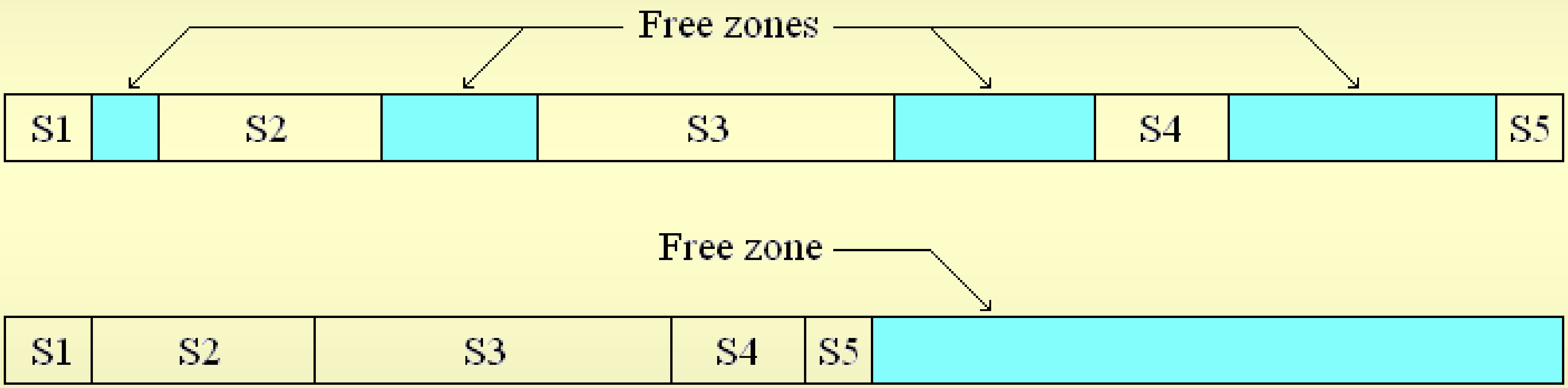
- many free zones, too small to be used
- this situation arises after a large number of segment allocations and releases
- no matter what algorithm was used
- placing a new segment may fail, even though total free space would be enough

Memory Fragmentation (2)

Eliminating external fragmentation

- memory defragmentation
  - move the segments such that there remain no free zones between them
  - a single free zone, of maximum size, is created
  - performed by a specialized program, part of the operating system

Defragmentation (1)



Defragmentation (2)

Running the program for memory defragmentation

- time consuming
  - moving the segments through memory
  - updating the segment descriptors
- cannot be run very often
- only when necessary

Defragmentation (3)

Situation when the defragmentation program could be run

- when placing a segment into memory fails because of insufficient space
- on a periodical basis
- when memory fragmentation exceeds a certain level

Segmentation - Conclusions

Problems of the segmentation mechanism

- complex management
  - segment overlapping - hard to detect
- external fragmentation - usually strong
  - a lot of free space that cannot be used
- defragmentation takes time

V.5.2. Memory Pagination

Basic Principle

- virtual address space - split into pages
  - zones of fixed size
- physical address space - split into page frames
  - same size as pages
- size - usually 4 KB

Page Tables (1)

- the correspondence between pages and page frames - managed by the operating system
- basic structure - the page table
- one for each running process
- allows detecting wrong (illegal) memory accesses

Page Tables (2)

- upon different runs of the program, pages are placed into different frames
- the effect on location addresses
  - none
  - must only modify the page table
  - only once (when the page is loaded into memory)
  - performed by the operating system

Memory Access

- the program requests the virtual address
- determine the page containing the address
- look for the page in the page table
  - if the page is not found - generate a trap
- determine the corresponding page frame
- compute the physical address
- access the location at the computed address

Illustration (1)

Page table (simplified)

Pages	0	1	2	8	9	11	14	15
Page frames	5	7	4	3	9	2	14	21

Illustration (2)

Example 1:

- page size: 1000
- virtual address: 8039
  - page: $[8039/1000]=8 \rightarrow$ page frame 3
  - offset: $8039\%1000=39$ (within the page)
- physical address: $3 \cdot 1000+39=3039$

Illustration (3)

Example 2:

- page size : 1000
- virtual address: 5276
  - page: [5276/1000]=5
  - not present in the page table
  - error $\rightarrow$ a trap is generated

Intel Case

31 12	11 9	8	7	6	5	4	3	2	1	0
PAGE BASE ADDRESS	AVL	G	0	D	A	PCD	PWT	U/S	R/W	P

Restrictions

Building the page tables

- performed by the operating system
- must avoid applications from overlapping
- restrictions
  - a virtual page may appear in at most one position in a page table
  - a physical page frame may appear at most once throughout all page tables of the currently existing processes

Memory Fragmentation (1)

Internal memory fragmentation

- no space left between the pages $\rightarrow$ no external fragmentation
- internal fragmentation
  - free, unused space inside a page
  - cannot be used by another process
  - defragmentation is not possible
- less severe than external fragmentation

Memory Fragmentation (2)

Choosing the page size

- power of 2
- once it was chosen, it cannot be changed
- set up as a compromise
  - too large - strong internal fragmentation
  - too small - a lot of space spent for page tables
  - usually - 4 KB

Really That Simple?

32-bit processor

- address space: 4 GB (=2^{32})
- page size: 4 KB (=2^{12})

→ page table would have 2^{20} entries
  - too large
  - less memory available for applications

- 64-bit processors - even worse

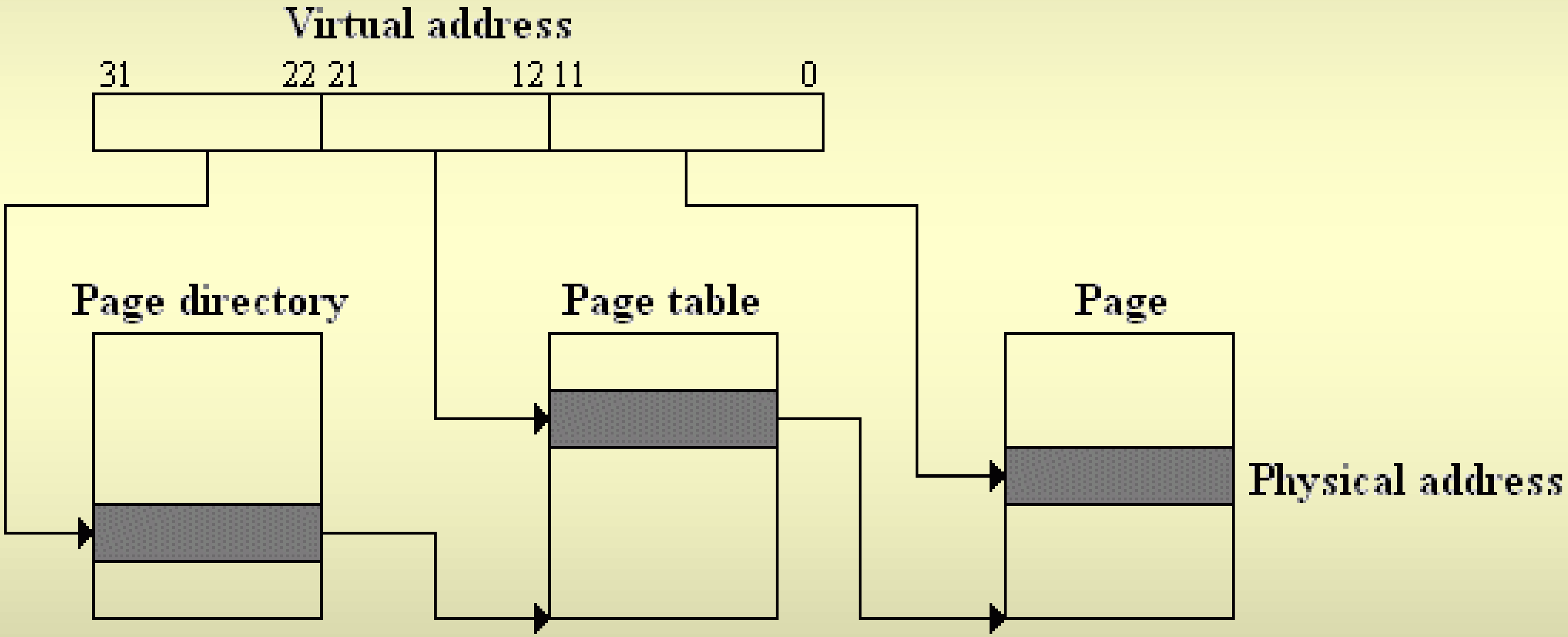
Solution 1

- reversed page tables
- do not contain all 2^{20} entries
  - only those which are used
- insert/search the table - *hash* function
- hard to implement in hardware
  - speed
  - avoid collisions (specific to hash functions)

Solution 2

- multi-level tables
- Intel case - 2 levels
  - Page Directory
  - Page Table
- directory entries - addresses of the page tables
- only page tables that are used are allocated

Intel Structure



Performance (1)

- page directories and page tables are kept into memory
  - too large to be kept into the processor
  - too many - specific to each process
- result - poor performance
  - for each memory access requested by the program - 2 additional accesses
- solution - dedicated cache

Performance (2)

- TLB (*Translation Lookaside Buffer*)
  - inside the processor
  - keeps pairs of corresponding virtual pages and physical page frames
  - the last ones that have been accessed
  - must be invalidated when a process switch occurs (the running process is deactivated, another one is activated)

V.5.3. Virtual Memory

Basic Idea

The problem

- applications - high memory consumption
- available memory - not enough

How can we solve it?

- the hard disk - very large capacity
- not all occupied memory zones are necessary at a certain moment

Virtual Memory

The solution - virtual memory (*swap*)

- some memory zones - evacuated to the disk
- when needed again, they are brought back to memory

Who manages the virtual memory?

- global information is required
- the operating system

Page File

- contains the memory zones evacuated to the disk
- information necessary for retrieving of a zone it stores
  - memory addresses
  - the program it belongs to
  - the size
  - etc.

Replacement Policy (1)

- the problem - same as for cache memory
- bringing a memory zone back fro the page file $\rightarrow$ evacuating another one
  - which one?
- purpose - reducing accesses to disk
- ineffective policy $\rightarrow$ large number of disk accesses $\rightarrow$ lower speed

Replacement Policy (2)

- working set - the memory zones that are necessary to the program at a certain moment
- usually much smaller than the total amount of zones used by the program
- if it fits into memory - disk accesses are rare

Replacement Policy (3)

- select for evacuation that zone which will not be necessary in the near future
- cannot know for sure - estimate
  – based on the near past behavior
- demand paging - evacuate to disk only if absolutely necessary

Implementation

- through the memory management mechanisms discussed before
  - if a program tries to access a memory location that has been temporarily saved on the disk, the same kind of detection is necessary
  - virtual memory can be used together with both segmentation and pagination
- the role of the interrupt system - enhanced

Memory Access (1)

Pagination case

1. the program requests the virtual address
2. determine the page containing the address
3. look for the page in the page table
4. if the page if found - jump to step 9
5. generate a trap
6. the service routine looks for the page in the page file

Memory Access (2)

Pagination case (continued)

7. if the page is not in the page file - the program is terminated
8. bring the page into the physical memory
9. determine the corresponding page frame
10. compute the physical address
11. access the location at the computed address

Reducing Accesses to Disk (1)

- leads to improved performance
- a page is saved to disk and brought back to memory many times
- when a page is brought back to memory, its copy on the disk is not erased
- the page and its copy on disk are identical until the page in memory is modified

Reducing Accesses to Disk (2)

- when a page is evacuated from memory
  - if no modification has occurred since it was last brought into memory - no need to save it
  - especially useful for code pages
- hardware support is required for detecting this situation
  - it is enough to detect write operations

Reducing Accesses to Disk (3)

- page table - extended structure
  - for each page there is a supplementary bit (*dirty bit*)
  - indicates whether the page has been modified since it was last brought into memory
  - reset when the page is brought into memory
- for each instruction that writes into memory
  - the processor sets the bit corresponding to the page that contains the modified location

V.5.4. Inter-process Communication

Communication (1)

- to cooperate, the processes must be able to exchange data
  - sometimes in large volumes
- physical implementation
  - common memory zones
    - shared variables
  - memory zones under the kernel's control
    - more complex data structures, with specific access methods

Communication (2)

- in the first case, two or more processes access the same memory zone
  - the same segment is included simultaneously in the descriptor tables of multiple processes
  - the same page frame is included simultaneously in the page tables of multiple processes
- either way, the operating system controls the common memory zones
  - the processes are aware of their shared nature

Mutual Exclusion (1)

- accessing a common resource takes time
  - and may consist in many operations
- risk of interference
- example
  - a process starts accessing a shared variable
  - before it finishes, another process starts accessing the same variable
  - the variable may be changes incorrectly

Mutual Exclusion (2)

- access to a common resource - only under certain condition
- mutual exclusion
  - at a certain moment, a single process may access a certain resource
- control mechanisms
  - semaphore - the simplest
  - shared structures whose access methods enforce mutual exclusion (e.g., monitor)

Implementation

- access to a resource can be controlled (and blocked) only by the operating system
- so any kind of acces to a shared resource involves a system call
- when working at low level (shared variables + semaphores), it is the programmer's task to make sure the call is executed correctly

Threads - Communication

- for the threads of the same process, global variables are automatically shared
  - higher speed
  - increased risk of programming errors
- the necessity mutual exclusion is present here too