English Edition

Memory Management (2)

Allocation of free memory

A specific amount of requested memory can be allocated as follows:

- First Fit Method
  - First sufficiently large free memory area is allocated.
- Best Fit Method
  - Smallest sufficiently large free memory area is allocated.
  - More complex, since whole list or bitmap must be searched.
  - Strong fragmentation of free memory into many small areas.

Allocation of free memory

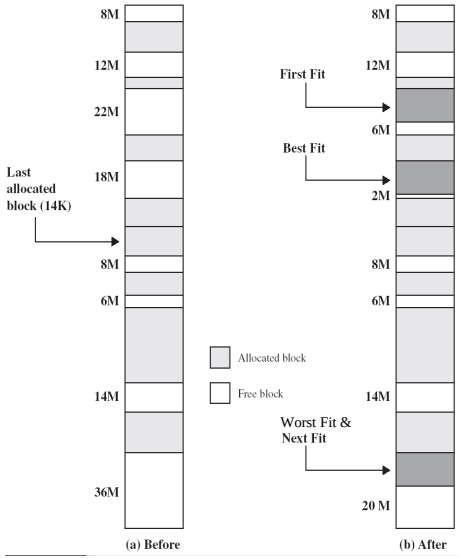
- Worst Fit Method
  - Largest free memory area is allocated.
  - Rather large free areas remain.
- Quick Fit Method
  - Keeping several lists of free memory areas for specific standard sizes (e.g. 4 KB, 8 KB, 12 KB etc.) leads to quick allocation.
  - After freeing an area it must be joined with its free neighbors (if there are any). This is more work when there are several lists.
  - Special variant: Buddy System

Allocation of free memory: Example

Example for memory usage

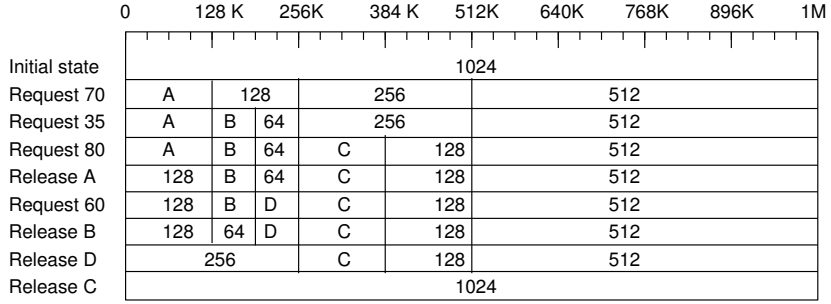
- Before and
- after

allocation of a 16 Mbyte block



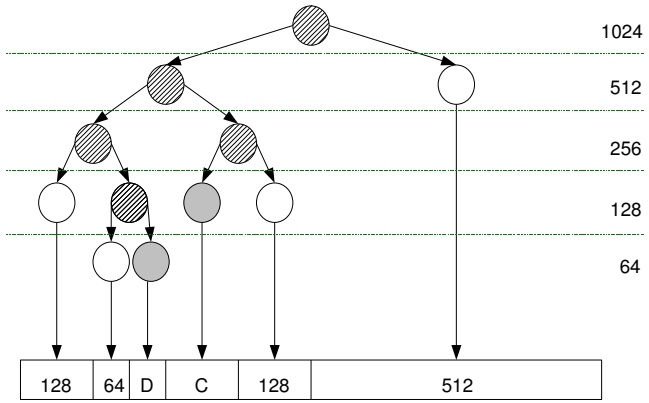
Memory Management: Buddy System

- Separate lists of free areas with sizes 1, 2, 4, 8, 16 etc. KByte (up to memory size).
- When freeing a memory area, only search one list to find out whether area can be joined with another free area.
- Block size always a power of 2 – not very efficient memory usage.



Buddy System: Tree Representation

Situation before release of process D



Segmentation (1)

- Partition Programs into several segments, e.g. code segment, data segment(s), stack segment(s) etc.
- Each Segment corresponds to a lineare address space from 0 to a maximum value.
- Addresses are composed of:
  - a segment address
  - a relative address within the segment

  (two-dimensional address space)

Segmentation (2)

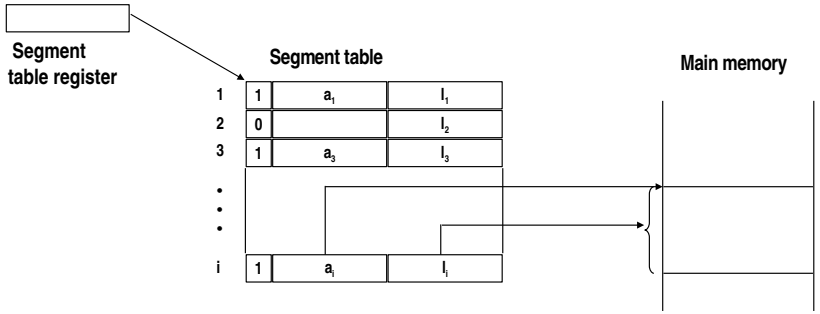
- Advantages of segmentation:
  - Easier to find free space for the smaller segments.
  - Segments can grow independently.
  - Not all segments have to be inside the main memory (Swapping segments).
  - Segments can have separate levels of protection (read, write, ...).
  - Differentiate between process-private segments and segments that are shared between several processes.

Segmentation (3)

- Hardware must allow for segmentation.
- For each process there is a segment table in the main memory, holding at least three values for each segment:
  - Is the segment loaded in main memory?
  - Segment's start address in main memory
  - Length of segment

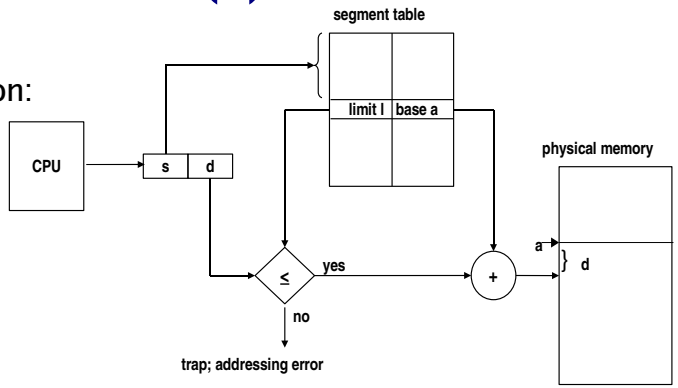
Segmentation (4)

- Address of the segment table saved in the segment table register.



Segmentation (5)

- Address translation:



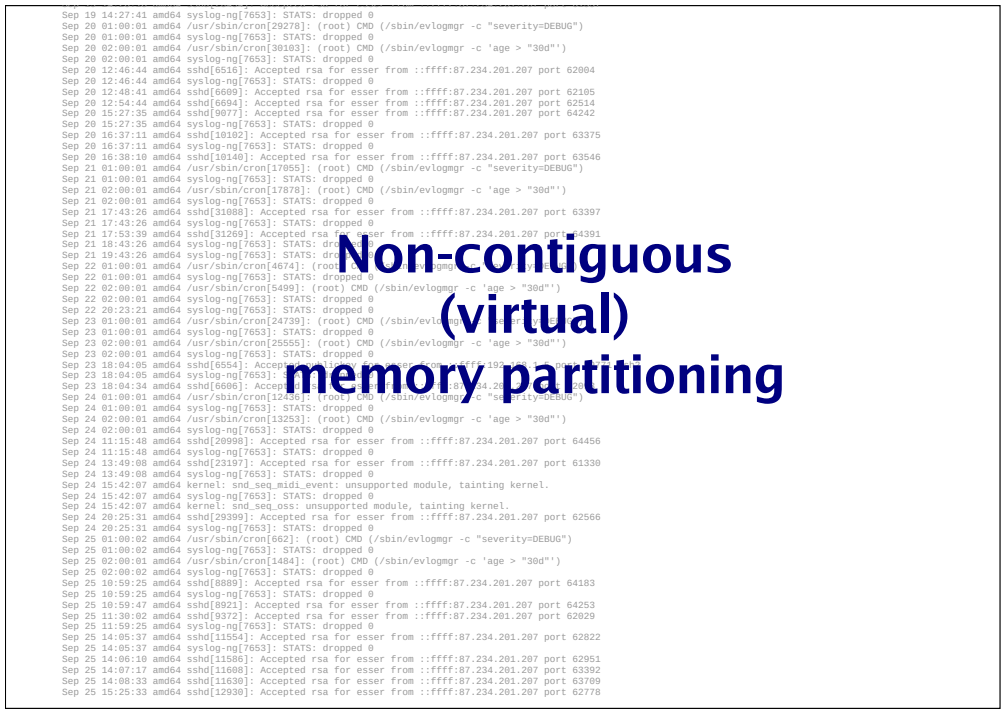
- Each memory access requires an additional memory access (segment table). Special hardware registers must speed-up this process.

Segmentation (6)

- Accessing a segment that is not inside the main memory causes a Segment Fault: Operating system must fetch segment into memory.
- Protection of a segment:
  Entries in segment table contain additional protection code (e.g. for read, write, execute accessses).
  Code generated by programmer (or compiler/ linker).

Segmentation (7)

- Sharing of a segment in main memory is simple:
  Let entries in segment tables of several processes point to the same main memory address.
- Fragmentation:
  Since every segment is contiguous inside the main memory, there is external fragmentation.



Starting position

- Not enough memory for large programs
  -> Overlay programming
- Load parts of the program dynamically (from disk) when they are needed
- Programmer must care about creation of overlays

Overlay Programming

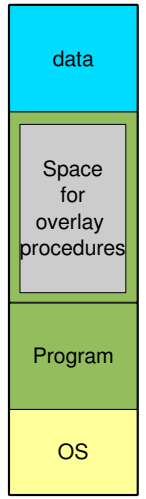
Turbo Pascal, about 1985-90:

```
program bigproject;

overlay procedure customerdata;
...

overlay procedure storagedata;
...

{ main program }
begin
  while input <> "exit" do begin
    case input of
      "customer": customerdata;
      "storage":  storagedate;
    end;
  end;
end.
```



Solution of the problem

- Virtual memory that is big enough for the whole program
- Program sees memory area that was allocated for it – how much "real" memory the system has, does not matter (for the program)

Virtual Memory Management (Paging)

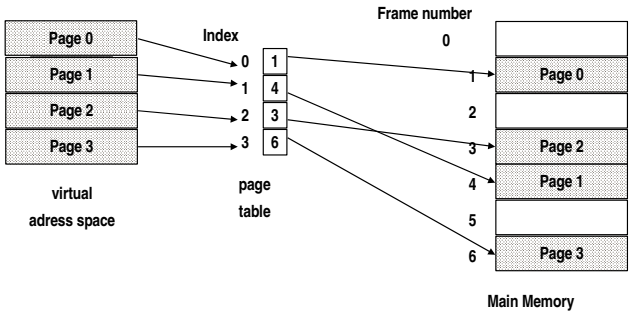
- Partition the address space into pages of fixed size and the main memory into page frames of equal size.
  - Typical page sizes: 512 Byte – 8192 Byte (always a power of 2).
- The linear contiguous address space of a process ("virtual" address space) is mapped onto non-contiguous page-frames.
- Operating system manages one single list of free page frames.

Virtual Memory Management (Paging)

- Calculation of physical memory address from the virtual address given by the program
  - happens at the program's runtime,
  - is transparent for the program,
  - must be hardware-assisted.
- Advantages of virtual memory management:
  - Allocation of main memory is simple.
  - No external fragmentation, small internal fragmentation.
  - Nothing to do for the programmer.

Virtual Address Space (1)

- With Paging, the relationship between program addresses and physical (main memory) addresses is only established at runtime by using the page tables.



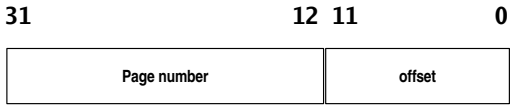
Virtual Address Space (2)

- The addresses used by the program are therefore called virtual addresses.
- The virtual address space of a program is the linear, contiguous address space that can be used by the program.

Address translation with Paging (1)

- Program address is split into two parts:
  - a page number
  - a relative address (offset) into the page

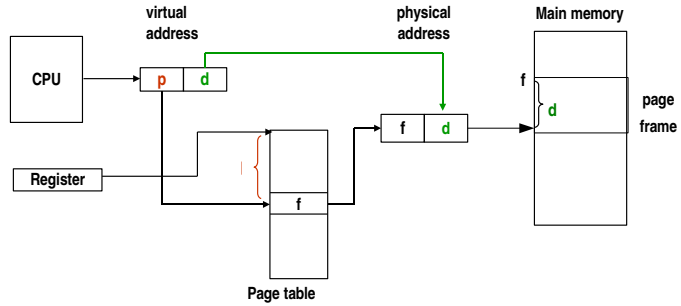
Example: 32 bit address, page size of 4096 (=2^12) bytes:



Address translation with Paging (2)

- For each process there is a page table. It contains, for each process page:
  - the information whether the page is in main memory,
  - the number of the page frame in main memory, which holds the page.
- A special Register contains the start address of the page table for den current (running) process.
- Page number is used as index into the page table.

Address translation with Paging (3)



- Each memory access requires an additional memory access (page table). Special caches (hardware) must speed-up this process!
- Page not in main memory -> special exception, so called page fault.

Virtual memory in general (1)

- Several processes can be kept in memory effectively
  --> better system utilization
- A process can request much more memory than physically available

Virtual memory in general (2)

- General procedure:
  - Only parts of the process reside in the physical (main) memory
  - When accessing an address that is not in memory:
    - OS sets process state to *blocked*
    - OS executes Disk-I/O read command
    - After loading the missing part (page or segment) an I/O interrupt occurs
    - OS sets process state back to *ready*

Virtual memory in general (3)

- „*thrashing*“ (see later): Processor spends most of its time swapping process parts in and out instead of running process code
- Locality principle:
  - Data and program code accesses often locally grouped;
    --> it is ok to assume, that only few process parts must be kept in memory simultaneously during a short period of time