Lecture 6

Principles of memory management

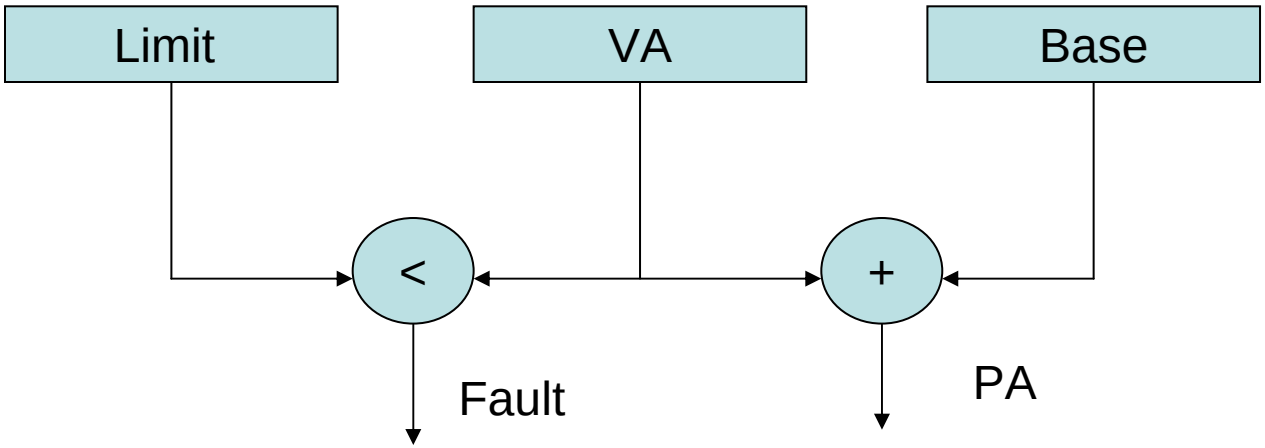
- Is there any difference between the physical memory space and the memory space of a program ?
- Why is the memory organised hierarchically ?
- How is segmentation working ?
- What are the benefits and limitations of paging.
- What other service regarding memory management can be provided by the OS ?

Address translation

- The memory addresses are *physical (real) addresses*. They uniquely identify memory locations and create the physical address space.
- On the other hand, the process layout is specified in terms of *virtual addresses*. Their set represents the virtual address space.
- Therefore, a mechanism is needed for address translation.
- In addition to address translation, *the memory management unit (MMU)* takes care of protecting the memory space among processes.
- Methods of address translation:
  - base registers;
  - segmentation;
  - paging.

Base registers

- The virtual address is added to the content of a base register. The result is the physical address.



- The limit can be either the size of the allocated memory space or the last physical memory address allowed to this process.

Segmentation

- In this case, different memory segments store different parts of the program: code, data, stack. Each segment will have separate base and limit registers.

- One possibility is to use CPU dedicated registers (i.e. Intel x86), or the higher-order bits of the VA point within tables storing the base and limit values.

Paging

- The virtual address space is divided into pages of 2^k bytes each. If the virtual address is *n* bits, then the virtual memory space consists of 2^{n-k} pages.
- The upper *n-k* bits form a page number and the lower k bits are an offset into page.
- In the physical memory space, one 2^k byte space where a page can be mapped is called a *page frame*.
- Pages are managed with the help of tables, called *page tables*.
- A page table is indexed by the page number in a virtual address; the page table entry (PTE) defines the translation.
- The complete set of fields in PTE includes the following:

PTE fields

- *Page frame number* – this determines the page frame to which the page is mapped; this field is concatenated with the offset to give the physical address.
- *Protection bits* – e.g., shared pages are marked as “read-only”, or data pages are separated from code pages.
- *Present bit (P)* – called also *the valid bit* is set when there is a translation of this page number into a page frame number. Otherwise, when an attempt to access a page for which there is no valid translation, an interrupt will be generated (“page fault”). There can be two causes of this error: either that page was not allocated to the process, or the page is not present in the memory.
- *Dirty bit (D)* – called also as *the modified bit* signals when a write operation occurred in that page.
- *Accessed bit (A)* – is set when the page is being accessed.

Management of page tables

- Let's assume a 32-bit virtual address and that the page size is 2^{12} = 4096 byte. There are 2^{20} pages and therefore the same number of PTE. If each PTE is 4 bytes, the page table will be 4 MB !
- A better solution is to organize the virtual space hierarchically – two-level page table. The 20 bit are split in two groups of 10. the most significant 10 bits index a page table, called page directory. The page frame number stored in the selected PTE identifies the page holding the page table, which is then indexed by the other 10 bits of the page number.
- Another solution is to use cache memories to improve the speed of the paging system. The cache (Translation Lookaside Buffer) is addressed by a lookup on the page number. If hit, the PTE is in TLB….
- With inverted page tables, used by 64-bit systems, page frame numbers are mapped to page numbers. TLB works the same way.

Other memory services

- The OS provides two primary memory services to processes: allocation/de-allocation.
- Explicit allocation: the process specifies exactly which memory addresses are required;
- Implicit allocation: the process needs a particular block of memory but it doesn't mention where it should be in the memory space.
- Either way, applications often further subdivide the allocated space with their own memory management.
- De-allocation can be explicit (the process invokes a system call to free some memory areas), or implicit when areas not addressed by code are freed – *mark and sweep algorithm*.
- An additional service can be the control of memory sharing among processes.

Memory layout

- The main memory stores the interrupt vectors (generally at the beginning of the memory space), OS, user processes, and I/O mapped addresses.
- Generally, the process will have allocated segments, code, data, stack. How are they positioned ?
  - If the system has a large virtual memory space, the segments will be positioned at fixed virtual address boundaries (e.g., code segment at address 0, for 1 GB, then the data segment, and then the stack segment.
  - The second possibility is to allocate the code segment only as needed. At its end, the data segment starts, growing upwards. The stack segment starts at the top of the address space and grows downwards.
- Not the entire addressable memory space can be functional.
- Some memory areas can be reserved.