CHAPTER 9: MEMORY ORGANIZATION

The memory hierarchy of most computer systems includes not only physical memory, but also cache and virtual memory. First off, understand that by “cache”, we mean a collection of data duplicating original values stored elsewhere or computed earlier, where the original data is expensive to fetch (owing to longer access time) or to compute, compared to the cost of reading the cache. To put it another way, a cache operates as a temporary storage area, where frequently accessed data can be stored for rapid access. Cache memory is a high-speed memory situated between the CPU and physical memory; one level of cache is often located within the CPU chip. Cache memory may be unified or split into separate instruction and data caches. This division of caches is known as the Harvard Architecture, named for the Harvard Mark I relay-based computer, which stored instructions on punched tape (24 bits wide) and data in electro-mechanical counters. It’s interesting to note that the Harvard Mark I was also the first modern computer to have a program written for it, by Grace Hopper back in the 1940s. The performance of cache memory is largely determined by its hit ratio and average memory access time.

Cache memory may employ one of three mapping strategies. These mapping strategies are how a CPU decides where in the cache a copy of a particular entry of main memory will go, also known as its associativity. Associative mapping is the most flexible, but it requires a relatively expensive associative memory rather than standard static RAM. Associative mappings are free to choose any entry in the cache to hold the copy of the entry from main memory. Direct mapping is less flexible, but also less expensive. In direct mapping, each entry in main memory can go in only one place in the cache. Set-associative mapping offers the advantages of direct mapping as well as some of the flexibility of associative mapping. In set-associative mapping, there is a compromise between associative mapping and direct mapping, where each entry in main memory can go to any one of N places in the cache. This type of mapping is also known as N-way set associative mapping. Caches with set-associative memory employ FIFO, LRU, or Random replacement policies.

Virtual memory gives an application program the impression that is has contiguous working memory, or an address space, while in fact it may be physically fragmented or even overflow on to disk storage space. It is much less costly than adding physical memory to a system and does not significantly degrade system performance. The two primary functions of virtual memory are that each process has its own address space, and each process sees one contiguous block of free memory upon launch. Any fragmentation is hidden. A memory management unit, or MMU, is usually built into the CPU to help facilitate virtual memory. Systems that use virtual memory make programming of large applications easier and use real physical memory more efficiently than those without virtual memory. Virtual memory isn’t just using disk space to extend physical memory size, though. Virtual memory redefines the address space with a contiguous virtual memory address to “trick” programs into thinking they are using large blocks of contiguous addresses.

A virtual memory system may use paging or segmentation. A paging system moves frames of fixed size between physical memory and virtual memory. Page tables are used to translate the virtual addresses seen by an application program into physical addresses, or real addresses, used by the hardware to process instructions. Each entry in the page table contains a mapping for a virtual page to either the real memory address at which the page is stored, or an indicator that the page is currently held in a disk file. A system can have one page table for the whole system, or a separate page table for each application. While executing an instruction, a CPU will fetch an instruction located at a particular virtual address, or stores data to a particular virtual address, the virtual address must be translated to the corresponding physical address. The CPU’s MMU looks up the real address from the page table corresponding to a virtual address and passes the real address to the parts of the CPU which execute instructions. If the page tables indicate that the virtual memory page is not currently in real memory, the hardware raises a page fault exception (a special interrupt signal) which invokes the paging supervisor component of the operating system. Certain memory areas in virtual memory are “pinned down”, or in other words, cannot be swapped out to secondary storage. Interrupt mechanisms, the page tables themselves, data buffers that are accessed outside of the CPU, and timing-dependent kernels/applications are examples of elements that are pinned down. Paged virtual memory is by far the most common type of virtual memory in use today.

Some systems, such as the Burroughs large systems, use segmentation instead of page tables. In segmentation, an application’s virtual address space is divided into variable-length segments. A virtual address consists of a segment number and an offset within the segment. Memory is still physically addressed with a single number (called absolute or linear address). To obtain it, the processor looks up the segment number in a segment table to find a segment descriptor.  The segment descriptor contains a flag indicating whether the segment is present in main memory and, if it is, the address in main memory of the beginning of the segment (segment's base address) and the length of the segment. It checks whether the offset within the segment is less than the length of the segment and, if it isn't, an interrupt is generated. If a segment is not present in main memory, a hardware interrupt is raised to the operating system, which may try to read the segment into main memory, or to swapin. The operating system might have to remove other segments (swap out) from main memory in order to make room in main memory for the segment to be read in. This reduces internal fragmentation but can introduce external fragmentation.

It is possible to combine these two methods in a system that uses variable-sized segments constructed using a variable number of fixed-size pages. In some systems that use this mixture, virtual memory is usually implemented with paging, with segmentation used to provide memory protection. A problem that can commonly arise when working with swapping memory between physical and virtual memory is “thrashing”. Thrashing occurs when the computer spends too much time shuffling blocks of virtual memory between real memory and disks, and therefore appears to work slower. Better design of application programs can help, but ultimately the only cure is to install more real memory, once thrashing begins to become a problem.