6.5 Virtual Memory

- Cache memory enhances performance by providing faster memory access speed.
- Virtual memory enhances performance by providing greater memory capacity, without the expense of adding main memory.
- Instead, a portion of a disk drive serves as an extension of main memory.
- If a system uses paging, virtual memory partitions main memory into individually managed page frames, that are written (or paged) to disk when they are not immediately needed.

6.5 Virtual Memory

- A physical address is the actual memory address of physical memory.
- Programs create virtual addresses that are mapped to physical addresses by the memory manager.
- Page faults occur when a logical address requires that a page be brought in from disk.
- Memory fragmentation occurs when the paging process results in the creation of small, unusable clusters of memory addresses.

6.5 Virtual Memory

- Main memory and virtual memory are divided into equal sized pages.
- The entire address space required by a process need not be in memory at once. Some parts can be on disk, while others are in main memory.
- Further, the pages allocated to a process do not need to be stored contiguously—either on disk or in memory.
- In this way, only the needed pages are in memory at any time, the unnecessary pages are in slower disk storage.

6.5 Virtual Memory

- Information concerning the location of each page, whether on disk or in memory, is maintained in a data structure called a page table (shown below).
- There is one page table for each active process.
6.5 Virtual Memory

- When a process generates a virtual address, the operating system translates it into a physical memory address.
- To accomplish this, the virtual address is divided into two fields: A page field, and an offset field.
- The page field determines the page location of the address, and the offset indicates the location of the address within the page.
- The logical page number is translated into a physical page frame through a lookup in the page table.

The page field determines the page location of the address, and the offset indicates the location of the address within the page.

The logical page number is translated into a physical page frame through a lookup in the page table.

If the valid bit is zero in the page table entry for the logical address, this means that the page is not in memory and must be fetched from disk.
- This is a page fault.
- If necessary, a page is evicted from memory and is replaced by the page retrieved from disk, and the valid bit is set to 1.
- If the valid bit is 1, the virtual page number is replaced by the physical frame number.
- The data is then accessed by adding the offset to the physical frame number.

As an example, suppose a system has a virtual address space of 8K and a physical address space of 4K, and the system uses byte addressing.
- We have \(2^{13}/2^{10} = 2^3\) virtual pages.
- A virtual address has 13 bits (8K = \(2^{13}\)) with 3 bits for the page field and 10 for the offset, because the page size is 1024.
- A physical memory address requires 12 bits, the first two bits for the page frame and the trailing 10 bits the offset.

As an example, suppose a system has a virtual address space of 8K and a physical address space of 4K, and the system uses byte addressing.
- We have \(2^{13}/2^{10} = 2^3\) virtual pages.
- A virtual address has 13 bits (8K = \(2^{13}\)) with 3 bits for the page field and 10 for the offset, because the page size is 1024.
- A physical memory address requires 12 bits, the first two bits for the page frame and the trailing 10 bits the offset.

The address 10101010100112 is converted to physical address 010101010011 because the page field 101 is replaced by frame number 01 through a lookup in the page table.

The address 10101010100112 is converted to physical address 010101010011 because the page field 101 is replaced by frame number 01 through a lookup in the page table.
6.5 Virtual Memory

- We said earlier that effective access time (EAT) takes all levels of memory into consideration.
- Thus, virtual memory is also a factor in the calculation, and we also have to consider page table access time.
- Suppose a main memory access takes 200 ns, the page fault rate is 1%, and it takes 10 ms to load a page from disk. We have:
  \[ \text{EAT} = 0.99(200 \text{ ns} + 200 \text{ ns}) + 0.01(10 \text{ ms}) = 100,396 \text{ ns}. \]

Even if we had no page faults, the EAT would be 400 ns because memory is always read twice: First to access the page table, and second to load the page from memory.

Because page tables are read constantly, it makes sense to keep them in a special cache called a translation look-aside buffer (TLB).

TLBs are a special associative cache that stores the mapping of virtual pages to physical pages. The next slide shows how all the pieces fit together.

Another approach to virtual memory is the use of segmentation.

Instead of dividing memory into equal-sized pages, virtual address space is divided into variable-length segments, often under the control of the programmer.

A segment is located through its entry in a segment table, which contains the segment’s memory location and a bounds limit that indicates its size.

After a page fault, the operating system searches for a location in memory large enough to hold the segment that is retrieved from disk.

Both paging and segmentation can cause fragmentation.
- Paging is subject to internal fragmentation because a process may not need the entire range of addresses contained within the page. Thus, there may be many pages containing unused fragments of memory.
- Segmentation is subject to external fragmentation, which occurs when contiguous chunks of memory become broken up as segments are allocated and deallocated over time.

Large page tables are cumbersome and slow, but with its uniform memory mapping, page operations are fast. Segmentation allows fast access to the segment table, but segment loading is labor-intensive.

Paging and segmentation can be combined to take advantage of the best features of both by assigning fixed-size pages within variable-sized segments.

Each segment has a page table. This means that a memory address will have three fields, one for the segment, another for the page, and a third for the offset.
6.6 A Real-World Example

- The Pentium architecture supports both paging and segmentation, and they can be used in various combinations including unpaged unsegmented, segmented unpaged, and unsegmented paged.
- The processor supports two levels of cache (L1 and L2), both having a block size of 32 bytes.
- The L1 cache is next to the processor, and the L2 cache sits between the processor and memory.
- The L1 cache is in two parts: an instruction cache (I-cache) and a data cache (D-cache).

The next slide shows this organization schematically.