Memory Management: there are various memory management algorithms discussed in this chapter, including swapping, contiguous allocation, paging, and segmentation. They differ in many aspects:

- Memory is central to the operation of a modern computer system and consists of a large array of bytes, each with its own address (***byte-addressable***). One simple way to allocate an address space to each process is through the use of **base** and **limit registers**. The base register holds the smallest legal physical memory address, and the limit specifies the size of the address range.
- The binding of instructions and data to physical memory addresses can be done at different steps, 1) **compile time**, in which **absolute codes** are generated. If the location changes later, the codes must be recompiled; 2) **load time**, in which compiler must generate **relocatable codes**. If the starting address changes, we need to reload the codes into the memory; 3) **execution time**, in which the process can move during the execution from one memory segment to another. Special hardware must be used. Modern general-purpose operating systems all use the execution binding approach.
- **Logical** vs. **physical addresses:** logical address relevant to a process, is generated by CPU, whereas the physical address is the actual physical memory location seen by the memory management unit or MMU. The logical address and physical address are different only if the *execution time address binding* is used, which is the case in all modern operating systems. The logical (or virtual) address to physical address translation is usually done with **hardware support**, which can be either simple *base register* and *limit register* or use complicated page/segmentation tables with TLBs.
- **Performance**: logical address to physical address translation can be time-consuming since the translation tables (page or segmentation table) are usually stored inside memory, in which each address translation can require multiple memory accesses. Fast registers or/and caches in hardware (such as TLB) help to improve the access to translation tables.
- **Fragmentation**: fragmentation occurs when memory is broken up into small chunks, and some portions are too small to be used by memory requests. Fragmentation results in memory space waste. Fixed-size allocations such as paging suffer from **internal fragmentation** while variable-sized allocations such as contiguous allocation and segmentation suffer from **external fragmentation**.
- **Relocation**: *Compaction* is a solution to external fragmentation problem by moving programs to new memory locations, transparent to processes. This requires logical address to be relocated dynamically at the execution time. If address binding occurs at compile time or load time, compaction is not feasible.
- **Swapping*:*** Swapping can be added to any memory allocation algorithms. Processes are copied from main memory to a backing store and later copied back to main memory. This scheme allows more processes to fit into memory at one time to be running, thus increases the degree of multiprogramming.

- **Sharing**: This allows certain pages or segments to be shared by different processes. Some *protection mechanism* must be in place to ensure proper access, e.g., read-only, execute-only, read-write. Both swapping and sharing can improve the degree of multiprogramming.

Contiguous Allocation

- Each process occupies a single contiguous section of memory.
- **Base register** contains the smallest physical address. **Limit register** contains range of logical addresses – each logical address must be less than the limit register.
- This is a classical dynamic storage-allocation problem, which deals with the selection from a list of free holes in memory to satisfy a request of size n (variable size). There are three commonly used methods including **first-fit**, **best-fit** and **worst-fit**.
- This may result in serious external fragmentation problem under contiguous allocation scheme, as the total memory required by a process is usually very large, leaving many free holes (potentially large) to be unused. Thus, the contiguous allocation was only be used when the OS runs a single user-process (like in MS-DOS) or with only very few processes. In multi-programming OS (such as in all modern OS systems), such a scheme cannot be used.

Segmentation

- The memory allocation to each process is a collection of segments, which are variable-sized blocks reflecting the users' logical views of the memory, e.g., main program, procedures, functions, methods, and variables.
- The logical address explicitly specifies both a **segment number** and an **offset** within the segment. This is in contrast to a paging scheme in that user has one single address and the operating system automatically partitions the address into a page number and an offset (invisible to users).
- A segmentation table is associated with each process, which is stored inside memory pointed by a *segmentation table base register* (STBR); a *segmentation-table length register* (STLR) specifies the number of segments of the process, i.e., number of entries in a segmentation table. A segment offset must be smaller than its corresponding segment length.
- Each entry in a segmentation table specifies a **base** (the starting address where the segment resides in memory) and a **limit** specifying the length of the segment.
- The segmentation scheme is also one type of ***dynamic storage-allocation***, which suffers from external fragmentation. However, each segment (of a variable size) is much smaller, and there usually exists certain segments from some processes that can be particularly small, therefore most of the small free holes if not all can still be utilized, resulting in much less severe external fragmentation problem.
- Sharing a segment among different processes makes more sense than sharing a page since each segment is a logical entity, for example, a subroutine.
- Each segment can be further handled by a paging scheme. In this case, suppose

the logical address is s1/s2/d, s1 is used to locate one entry in the segmentation table, which points to corresponding page table. It then uses s2 to locate the entry to find out the corresponding frame number (f), so the physical address is f+d.

Paging Scheme

- The main memory is divided into fixed-size blocks called **frames**; logical address space is divided into the ***same*** fixed-size blocks called **pages**. The operating system keeps track of all free frames in the main memory – a ***free frame list.***
- The address translation is done through a **page table**, which maps a **page number** to a corresponding **frame number**. The starting physical address of a frame can be easily (automatically) obtained, that is ***frame number x frame size***.
- The page table associated with each process is also stored inside main memory, pointed by a register called *page-table base register* (PTBR), which is contained usually within the PCB of a process. Thus, each memory access thus requires two memory accesses (one for page table and one for the actual data or code). The performance can be improved by storing part of the page table entries in a special fast-lookup hardware called *translation look-aside buffers* (**TLB**). The improvement depends on the *hit ratio* in the TLB (like in any caching scheme).
- Using the TLB in address translation under a paging system involves obtaining the page number from a logical address and checking if the frame for the page is in the TLB. If it is (TLB hit), the frame number is obtained from the TLB. If the frame number is not present in the TLB (TLB miss), it must be retrieved from the page table in memory.
- Paging scheme suffers from internal fragmentation, in which on average half of the last page (frame) allocated to a process may be wasted.

Hierarchical Paging

- A large logical addresses space can result in an excessively large page table, in which case a page table needs to be stored in multiple frames. Hierarchical paging scheme such as two-level paging can deal with this problem, in which it usually tries to keep each (smaller) page table to fit within one page (frame). However, this would result in more than two memory access for each address translation due to the multiple level address translation required. Different TLBs can be used in each level for address translation (see ARM architecture).
- In practice, a system usually adopts multiple page sizes to satisfy different needs, with 4K and 4M page sizes for example in Intel IA-32 architecture.

Hashed Page Table

- One approach for handling address spaces larger than 32 bits is to use a hashed page table, with the hash value being the virtual page number. Each entry in the hash table contains a linked list of elements that hash to the same location (to handle collisions). Each element consists of three fields: (1) the virtual page number, (2) the value of the mapped page frame, and (3) a pointer to the next element in the linked list.