Chapter 8 summary

- Program must be brought (from disk) into memory and placed within a process for it to be run.
- Main memory can take many cycles, causing a **stall**.
- **Cache** sits between main memory and CPU registers.
- A pair of **base** (relocation) and **limit** registers define the logical address space.
- CPU must check, every memory access generated in user mode, to be sure it is between base and limit for that user.
- **Address Binding:**
  - Programs on disk, ready to be brought into memory to execute form an input queue.
  - Inconvenient to have first user process physical address always at 0000.
  - Address binding of instructions and data to memory addresses can happen at three different stages:
    - **Compile time:** If memory location known a priori, absolute code can be generated.
    - **Load time:** Must generate relocatable code if memory location is not known at compile time.
    - **Execution time:** Binding delayed until run time if the process can be moved during its execution from one memory segment to another.
- **Logical address** – generated by the CPU; also referred to as virtual address.
- **Physical address** – address seen by memory unit (happens at address binding).
- **Logical address space** is the set of all logical addresses generated by a program.
- **Physical address space** is the set of all physical addresses generated by a program.
- **Memory-Management Unit (MMU):**
  - Hardware device that at run time maps logical (virtual) to physical address.
  - User program deals with logical addresses; it never sees the real physical addresses.
    - Execution-time binding occurs when reference is made to location in memory.
- **Dynamic Relocation Using a Relocation Register:**
  - Useful when large amounts of code are needed to handle infrequently occurring cases.
  - No special support from the operating system is required.
    - OS can help by providing libraries to implement dynamic loading.
- **Static linking** – system libraries combined by the loader into the binary program image (if needed or not).
- **Dynamic linking** – linking of system libraries postponed until execution time.

    - Small piece of code, **stub**, used to locate the appropriate memory-resident library routine.
    - Dynamic linking is particularly useful for libraries.
    - This system is also known as **shared libraries**.
- **Swapping:**
    - A process can be **swapped** temporarily out of memory to a backing store, and then brought back into memory for continued execution.
    - **Backing store** – fast disk large enough to accommodate copies of all memory images for all users.
    - **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed.
    - Major part of swap time is **transfer time**.
        - Total transfer time is directly proportional to the amount of memory swapped.
    - System maintains a **ready queue** consisting of all ready-to-run processes whose memory images are on backing store or in memory.
    - Does the swapped-out process need to swap back into same physical addresses?
        - If it was at compile/load time: yes!
        - If it was at execution time: no!
- If next processes to be put on CPU is not in memory, need to swap out a process and swap in target process.
    - Can reduce time through decreasing size of memory swapped.
- Pending I/O – cannot swap out as I/O would occur to wrong process or transfer I/O to kernel space, then to I/O device
    - Known as double buffering but adds overhead.
- Standard swapping not used in modern Oss and mobile systems.
- **Contiguous Allocation:**
    - Main memory must accommodate both OS and various user processes.
    - Main memory usually divided into **two** partitions:
        - Resident operating system usually held in low memory with interrupt vector.
        - User processes then held in high memory.
        - Each process contained in a single contiguous section of memory.
    - **Relocation registers** used to protect user processes from each other, and from changing operating-system code and data.
        - Base register contains value of the smallest physical address.
        - Limit register contains range of logical addresses.

        - Each <u>logical address</u> must be **less** than the <u>limit register</u>.
    - **MMU** maps logical address dynamically by adding the value in the base register.
    - As part of context switch, dispatcher (PCB) loads <u>base</u> and <u>limit</u> registers values of a process under execution.
  - This scheme provides an effective way to allow operating system's size to change dynamically.
    - Suitable if an OS service is not commonly used.
    - Such services code is called **transient code**.
  - Memory is divided into several partitions – **multiple-partition method**:
    - <u>Degree of multiprogramming</u> limited by number of <u>partitions</u>.
    - <u>Variable-partition</u> sizes for efficiency (sized to a given process' needs).
    - <u>Hole</u> – block of available memory; holes of various size are scattered throughout memory.
- **Dynamic Storage-Allocation Problem:**
  - <u>First fit:</u> Allocate the **first** hole that is **big** enough.
  - <u>Best fit:</u> Allocate the **smallest** hole that is big enough; must search entire list, unless ordered by size.
    - Produces the **smallest** leftover hole.
  - <u>Worst fit</u>: Allocate the **largest** hole; must also search entire list.
    - Produces the **largest** leftover hole.

  ***First fit** and **best fit** better than **worst fit** in terms of <u>speed</u> and <u>storage utilization</u>.
- **Fragmentation:**
  - <u>External Fragmentation</u> – total memory space is enough to satisfy a request, but it is not contiguous.
    - Reduce external fragmentation by **compaction**.
      - Shuffle memory contents to place all free memory together in <u>one large block</u>.
      - Compaction is possible only if relocation is dynamic and is done at <u>execution time</u>.
      - I/O problem.
  - <u>Internal Fragmentation</u> – allocated memory may be slightly **larger** than requested memory; this size difference is memory internal to a partition, but not being used.
- **Segmentation:**
  - Memory-management scheme that mapped the programmer's view of memory (not equal in size).
  - Logical address consists of a two tuple: <segment-number, offset>

- A **segment table** is used to map the two-dimensional logical addresses of each process into one-dimensional physical addresses.
  - Each entry in the segment table has:
    - Base – contains the starting physical address where the segment resides in memory.
    - Limit – specifies the length of the segment.
- Hardware support:
  - Segment-table base register (STBR) points to the segment table's **location** in memory
  - Segment-table length register (STLR) indicates **number of segments** used by a process.
- Protection bits associated with **segments**; code sharing occurs at segment level.
- Since segments vary in length, memory allocation is a **dynamic storage-allocation** problem.

- **Paging:**
  - Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available.
    - Avoids external fragmentation.
    - Avoids problem of varying sized memory chunks.
  - Divide physical memory into fixed-sized blocks called frames.
  - Divide logical memory into blocks of same size called pages.
  - To execute a program of size N pages, need to find N free frames and load program.
  - Set up a page table to translate logical addresses to physical addresses.
  - Backing store likewise split into pages.
  - Still have internal fragmentation.
    - Last frame allocated may not be completely full.
  - Address generated by CPU is divided into:
    - Page number (p) – used as an index into a page table which contains base address of each page in physical memory.
    - Page offset (d) – combined with base address to define the physical memory address that is sent to the memory unit.
  - Process view and physical memory now very different.
  - By implementation, process can only access its own memory.
  - Frame size ↓ Page table ↑ Memory ↑ Search ↑
- **Implementation of Page Table:**
  - Most computers allow the page table to be very large; therefore, the page table is kept in main memory.

        - Page-table base register (PTBR) points to the page table.
        - Page-table length register (PTLR) indicates size of the page table.
    - In the last scheme every data/instruction access requires two memory accesses
        - One for the page table and one for the data/instruction.
    - The two-memory access problem can be solved using a special fast lookup memory cache called **translation look-aside buffer (TLB).**
        - Each entry in TLB consists of two parts: a page no. as a key and a corresponding frame value.
        - If a page number is not in the TLB (a TLB miss), its frame is added for faster access next time.
- **Effective access time:**
    - Associative Lookup = ε time unit.
    - Can be < 10% of memory access time.
    - Hit ratio = α
        - Hit ratio – percentage of times that a page number is found in the associative registers; ratio related to number of associative registers.
    - EAT = $(1 + \varepsilon)\,\alpha + (2 + \varepsilon)(1 - \alpha) = 2 + \varepsilon - \alpha$
- **Memory Protection:**
    - Memory protection implemented by associating protection bit with each frame to indicate if read-only or read-write access is allowed.
    - Valid-invalid bit attached to each entry in the page table:
        - "valid" indicates that the associated page is in the process' logical address space, and it is a legal page.
        - "invalid" indicates that the page is not in the process' logical address space.
        - Or use page-table length register (PTLR).
    - Any violations result in a trap to the kernel.
- **Shared pages:**
    - One copy of read-only (reentrant) code shared among processes.
- **Private code and data:**
    - Each process keeps a separate copy of the code and data.
- **Structure of the Page Table:**
    - Memory structures for paging can get huge using straight-forward methods.
    - Hierarchical Page Tables: **Important example in slide 54 and 56**
        - A simple technique is a two-level page table .
    - Hashed Page Tables:
        - The virtual page number is hashed into a page table.

            - This page table contains a chain of elements hashing to the same location.
        - Virtual page numbers are compared in this chain searching for a match.
            - When a match is found, the corresponding physical frame is extracted.
    - Inverted Page Table:
        - Rather than each process having a page table and keeping track of all possible logical pages, **track all physical pages**.
        - Decreases **memory** needed to store each page table, but increases time needed to **search** the table when a page reference occurs.